Université de Montréal

Advances in uncertainty modelling : from epistemic uncertainty estimation to generalized generative flow networks

par

Salem Lahlou

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

18 Août 2023

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Advances in uncertainty modelling : from epistemic uncertainty estimation to generalized generative flow networks

présentée par

Salem Lahlou

a été évaluée par un jury composé des personnes suivantes :

Simon Lacoste-Julien
(président-rapporteur)

Yoshua Bengio
(directeur de recherche)

Ioannis Mitliagkas
(membre du jury)

Christian Andersson Naesseth
(examinateur externe)

Lucas Benigni
(représentant du doyen de la FESP)

Résumé

Les problèmes de prise de décision se produisent souvent dans des situations d'incertitude, englobant à la fois l'incertitude aléatoire due à la présence de processus inhérents aléatoires et l'incertitude épistémique liée aux connaissances limitées. Cette thèse explore le concept d'incertitude, un aspect crucial de l'apprentissage automatique et un facteur clé pour que les agents rationnels puissent déterminer où allouer leurs ressources afin d'obtenir les meilleurs résultats.

Traditionnellement, l'incertitude est encodée à travers une probabilité postérieure, obtenue par des techniques d'inférence *Bayésienne* approximatives. Le premier ensemble de contributions de cette thèse tourne autour des propriétés mathématiques des réseaux de flot génératifs, qui sont des modèles probabilistes de séquences discrètes et des échantillonneurs amortis de distributions de probabilités non normalisées. Les réseaux de flot génératifs trouvent des applications dans l'inférence *Bayésienne* et peuvent être utilisés pour l'estimation de l'incertitude. De plus, ils sont utiles pour les problèmes de recherche dans de vastes espaces compositionnels. Au-delà du renforcement du cadre mathématique sous-jacent, une étude comparative avec les méthodes variationnelles hiérarchiques est fournie, mettant en lumière les importants avantages des réseaux de flot génératifs, tant d'un point de vue théorique que par le biais d'expériences diverses. Ces contributions incluent une théorie étendant les réseaux de flot génératifs à des espaces continus ou plus généraux, ce qui permet de modéliser la probabilité postérieure et l'incertitude dans de nombreux contextes intéressants. La théorie est validée expérimentalement dans divers domaines.

Le deuxième axe de travail de cette thèse concerne les mesures alternatives de l'incertitude épistémique au-delà de la modélisation de la probabilité postérieure. La méthode présentée, appelée Estimation Directe de l'Incertitude Épistémique (DEUP), surmonte une faiblesse majeure des techniques d'inférence *Bayésienne* approximatives due à la mauvaise spécification du modèle. DEUP repose sur le maintien d'un prédicteur secondaire des erreurs du prédicteur principal, à partir duquel des mesures d'incertitude épistémique peuvent être déduites.

Mots-clés : apprentissage automatique, incertitude épistémique, probabilité postérieure, échantillonnage, réseaux de flot génératifs, inférence variationnelle.

Abstract

Decision-making problems often occur under uncertainty, encompassing both aleatoric uncertainty arising from inherent randomness in processes and epistemic uncertainty due to limited knowledge. This thesis explores the concept of uncertainty, a crucial aspect of machine learning and a key factor for rational agents to determine where to allocate their resources for achieving the best possible results.

Traditionally, uncertainty is encoded in a posterior distribution, obtained by approximate *Bayesian* inference techniques. This thesis's first set of contributions revolves around the mathematical properties of generative flow networks, which are probabilistic models over discrete sequences and amortized samplers of unnormalized probability distributions. Generative flow networks find applications in *Bayesian* inference and can be used for uncertainty estimation. Additionally, they are helpful for search problems in large compositional spaces. Beyond deepening the mathematical framework underlying them, a comparative study with hierarchical variational methods is provided, shedding light on the significant advantages of generative flow networks, both from a theoretical point of view and via diverse experiments. These contributions include a theory extending generative flow networks to continuous or more general spaces, which allows modelling the *Bayesian* posterior and uncertainty in many interesting settings. The theory is experimentally validated in various domains.

This thesis's second line of work is about alternative measures of epistemic uncertainty beyond posterior modelling. The presented method, called Direct Epistemic Uncertainty Estimation (DEUP), overcomes a major shortcoming of approximate *Bayesian* inference techniques caused by model misspecification. DEUP relies on maintaining a secondary predictor of the errors of the main predictor, from which measures of epistemic uncertainty can be deduced.

Keywords: machine learning, epistemic uncertainty, posterior, sampling, generative flow networks, variational inference

Contents

List of Tables

List of Figures

List of Symbols

$X, Y, Z, \ldots$	Random variables
$p_X(x), p(x), P(x)$	Probability, or density, of the random variable X taking the value x
$\mathbb{E}$	Expectation of a distribution
$\mathbb{H}$	Entropy of a distribution
$\mathbb{R}$	The real line
$\mathbb{R}^+$	The set of non-negative numbers, also denoted $[0, \infty)$
$[\![a, b]\!]$	The discrete set $\{a, a+1, \ldots, b\}$, used when $b > a$ are two integers
$\boldsymbol{u} \in \mathbb{R}^d$	A real-valued vector $\boldsymbol{u}$ of d dimensions, seen as a column matrix in $\mathbb{R}^{d \times 1}$
$\boldsymbol{A}^\top \in \mathbb{R}^{m \times n}$	The transpose of the real-valued matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$
$\arg\max_{x \in \mathcal{X}} f(x)$	The subset of $\mathcal{X}$ that maximizes the function $f : \mathcal{X} \to \mathbb{R}$
$\arg\min_{x \in \mathcal{X}} f(x)$	The subset of $\mathcal{X}$ that minimizes the function $f : \mathcal{X} \to \mathbb{R}$
$\mathcal{P}(\mathcal{U})$	The set of probability distributions over $\mathcal{U}$
$\|\boldsymbol{u}\|_1$	The L^1 norm of the vector $\boldsymbol{u} \in \mathbb{R}^d$: $\sum_{i=1}^d \lvert\boldsymbol{u}_i\rvert$
$\|\boldsymbol{u}\|_2$	The L^2 norm of the vector $\boldsymbol{u} \in \mathbb{R}^d$: $\sum_{i=1}^d \lvert\boldsymbol{u}_i\rvert^2$. It is also equal to $\sqrt{\boldsymbol{u}^\top \boldsymbol{u}}$
$\mathbf{1}(\textit{condition})$	One when *condition* is satisfied, zero otherwise
$\mathbf{1}_A(x)$	One when $x \in A$, zero otherwise, similar to $\mathbf{1}(x \in A)$
$\det(\boldsymbol{A})$	The determinant of the square matrix $\boldsymbol{A}$
$\boldsymbol{I}_n$	The identity matrix of $\mathbb{R}^{n \times n}$

إلى روح أبي الطيبة الطاهرة، الأستاذ الدكتور فؤاد لحلو

Remerciements

Je voudrais d'abord remercier mon directeur de thèse, Yoshua Bengio, qui m'a offert l'opportunité de poursuivre mon doctorat à Mila. En plus de m'avoir communiqué sa passion pour l'intelligence humaine et artificielle, de m'avoir donné la liberté d'explorer, Yoshua m'a inspiré et motivé à maintes reprises. Je suis reconnaissant de l'avoir eu comme directeur de thèse.

La recherche scientifique peut être frustrante à certains moments. Elle est naturellement plus ludique à plusieurs. Je suis reconnaissant envers mes collaborateurs, avec qui j'ai énormément appris, et pris du plaisir à réfléchir à des problèmes non résolus et à écrire des articles.

Je voudrais remercier les membres de ma famille, mon père Fouad, ma mère Raja, mes soeurs Yasmine et Khadija, pour leur amour et soutien inconditionnels. Je remercie aussi Salim, mon beau-frère. Une pensée particulière à mes mes nièces et mon neveu: Aya, Miya, et Rhali, qui, par leur belle innocence, me rappelaient à quoi servait tout cela.

Je suis content d'avoir développé de belles amitiés et grandi avec certaines personnes depuis mon arrivée à Montréal. Je remercie Adam, Adrien, Ahmed, Alex, Anne-marie, António, Arnaud, Christos, David, Gabriel, Gabrielle, Gauthier, Ghait, Greta, Hugo, Jad, Joseph, Julia, Karam, Laura, Léna, Mandana, Mariane, Marwa, Morgane, Nikolay, Nizar, Oussama, Padideh, Rémi, Rim, Simo, Simon, Taha, Tigran, Tom, Tristan, Valentin, Victor, Zhor, pour les magnifiques moments passés et discussions eues ensemble. Je suis aussi heureux d'avoir croisé sur mon chemin des personnes inspirantes, par leur façon de voir le monde, comme Léonard, Lucas, Moksh, Stefano, Théo, Xu, et Younesse. Je suis aussi reconnaissant envers mes amis de longue date, Hachem, Omar, Nada et André, pour leur soutien et pour les beaux moments passés ensemble avant et pendant la période de mon doctorat.

Je remercie les membres du jury de ma thèse, Simon Lacoste-Julien, Ioannis Mitliagkas, et Christian Andersson Naesseth, pour leur temps et leurs suggestions utiles pour mon manuscrit.

Pour finir, je remercie l'Académie Hassan II des sciences et techniques du Maroc, pour leur soutien pendant mes études, y compris mes premières années au doctorat.

Je dédie cette thèse à mon père, décédé pendant mon doctorat. Par ses valeurs et sa soif de connaissances, il a su m'inspirer et m'encourager à suivre la voie de la recherche scientifique. Je suis reconnaissant de l'avoir eu comme père.

Introduction

Perhaps the questions that most eluded humans throughout history, which will probably remain evasive in the future, are those about *intelligence* and *consciousness*. Philosophy of mind distinguishes the dualist view, in which the *mind* is seen as a separate entity from the *body* (Robinson, 2023)[1], from the monist view, in which there is no fundamental division between mind and body.

Functionalism (Levin, 2023) is a view that is popular among monists[2], which states that every mental process is solely defined by the way it functions or the role it plays in the larger system of which it is part. Early antecedents of functionalism include Aristotle's (350 BCE) theory of the soul, which he identifies as "whichever powers and capacities enable a natural, organized human body to fulfill its defining function", and Hobbes' argument in the Leviathan that reasoning is "nothing but reckoning, that is adding and subtracting, of the consequences of general names agreed upon for the marking and signifying of our thoughts." (Hobbes, 1651)

More recently, Turing proposed to replace the question "can machines think?" (Turing, 1950) with one where both *machines* and *think* are thoroughly defined. His seminal paper introduced the now popular concepts of the *Turing machine* and the *Turing test* and provided a fruitful functionalist theory: *Machine state functionalism*. The theory identifies mental states with machine states of a probabilistic automaton, which are similar to Turing machines except that transitions between computational states are stochastic (Rescorla, 2020). The computational theory of mind sparked a cognitive revolution, with the emergence of the **cognitive science** field (Miller, 2003), with the aim of understanding and formulating the principles of *intelligence*, using experimental techniques from psychology to construct testable theories of the human mind. It also prompted many scientists to dream about building computers capable of thought, or more specifically, computing machines able to solve decision-making problems, thus essentially **replicating intelligence**. The field of **artificial intelligence** (AI) aims not only at understanding intelligence but also at building

1. The dualist view dates back at least to 650 BCE with the Sāṃkhya-Yoga school of Hindu philosophy (Tuske, 2021), or more recently, in the 17th century, with René Descartes's mind-body problem.
2. given that the underlying functions result from physical properties of the body.

intelligent entities. A complete history of the field is provided in Russell et al. (2010) and Bringsjord and Govindarajulu (2022).

The past couple of decades have witnessed a bloom of artificial intelligence applications. Success stories include game solving (Schrittwieser et al., 2020), self-driving cars (Liang et al., 2018), medical applications (Yu et al., 2018) and the complex problem of protein folding (Jumper et al., 2021). More recently, generative models for text (OpenAI, 2023) and images (Ramesh et al., 2022) have gained much popularity amongst the large public, given the ease of interaction and the appealing results. The years 2022 and 2023 have even seen numerous experts trying to raise awareness about the substantial societal implications such generative models, and probably future AI applications, will have in the short term (Bengio, 2023). The fast pace at which the field is advancing is hugely due to **machine learning**, a sub-field of AI that aims at building systems, or agents, that improve their performance on a task with repeated experience on the given task. A popular, more formal definition of machine learning is presented in Mitchell (1997): "A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.". Machine learning as a field has blossomed in the 90s thanks to works that use statistics to formally ground ML (Boser et al., 1992; Cortes and Vapnik, 1995).

The classical computational theory of mind suggests that the mind is a computational system similar in essential respects to a Turing machine, where the inputs and outputs are symbols inscribed in memory locations. On the other hand, *connectionism*, which is a movement in cognitive science that gained traction in the 1980s, hopes to explain intellectual abilities using artificial neural networks, drawing inspiration from neurophysiology rather than logic and computer science (Buckner and Garson, 2019). The increasing popularity of connectionism is mainly due to the construction of connectionist models of object recognition, speech recognition, sentence comprehension, and other phenomena (Rumelhart et al., 1987; Mcclelland et al., 1987) and to the development of the *backpropagation* algorithm, a practical and efficient solution to the central problem in connectionism: finding the correct set of weights to accomplish a given task (Linnainmaa, 1970, 1976). This led to the emergence of a sub-field of machine learning: **deep learning** (LeCun et al., 2015; Schmidhuber, 2015). Deep neural networks are, in fact, central to all the examples cited above.

Even though deep learning has been labelled as the new wave of connectionism (Buckner and Garson, 2019), to this day, computer scientists still debate whether the recent successes of deep learning in artificial systems mark a clear superiority of connectionist models of computationalism over the classical symbolic theory. The main critiques are that connectionists do not model the variety of neurons in the brain and fail to provide a biologically plausible yet efficient alternative to backpropagation for learning. Some unanswered criticisms remain regarding high-level reasoning and language acquisition with connectionist models (Pinker

and Prince, 1988). This thesis does not attempt to directly address the ongoing debate nor favour one position over the other but instead tackles a central concept common to both views: **uncertainty**.

Uncertainty is a fundamental pillar of artificial intelligence, as most decision-making problems are inherently characterized by uncertainty. Whether exploring the dynamics of a game or unravelling the laws of physics, agents must measure the aspects of the model they are most uncertain about. This process is essential for obtaining critical information about a particular phenomenon. It allows agents to make informed and effective decisions regarding where to allocate more modelling resources. Such a process is central to AI and reminiscent of humans, especially babies building their world models, whose innate curiosity propels them to explore their environment and acquire knowledge about the world.

Uncertainty is usually codified with probabilities. Thus, *probability theory* is paramount when formalizing machine learning tasks. *Bayesian inference*, on the other hand, provides an elegant formalism for updating prior subjective beliefs about hidden quantities. In most practical settings, however, Bayesian inference is intractable, and approximations are required.

This thesis tackles the question of uncertainty in machine learning through two different but complementary approaches. Following an introduction of machine learning, which we provide in Chapter 2, with an emphasis on Bayesian inference and uncertainty, we study in Chapter 3 the mathematical properties of generative flow networks, which are probabilistic models useful not only for approximate Bayesian inference, but also for search problems in large compositional spaces[3], and showcase through a series of experiments their superiority to existing methods. In Chapter 4, we develop a theory that extends generative flow networks to more general settings, bypassing the need for discrete structures, and we illustrate in different simulated experiments that they retain their already proven advantages. In Chapter 5, we investigate a different approach to Bayesian inference for determining measures of *epistemic uncertainty*[4]. Through various experiments, we demonstrate that this alternative method can produce more accurate uncertainty estimates valuable in downstream tasks. In Appendix B, we present `torchgfn`, a recently developed open-source software that facilitates research on generative flow networks.

Before delving into the core of the thesis, we give in Chapter 1 an overview of the publications upon which it is based and mention the contributions of the author of the thesis in them.

3. In this thesis, we say that a set is compositional if the meaning of an object within this set is entirely determined by its structure and the meanings of its constituents.

4. a definition is available in Section 2.1.7

Chapter 1

Overview of the underlying publications

This manuscript is based on the following papers[1]:

— Bengio et al. (2023): "GFlowNet Foundations" - Yoshua Bengio*, Salem Lahlou*, Tristan Deleu*, Edward J. Hu, Mo Tiwari, Emmanuel Bengio, published in 2023 in the Journal of Machine Learning Research (JMLR).

— Malkin et al. (2023): "GFlowNets and variational inference" - Nikolay Malkin*, Salem Lahlou*, Tristan Deleu*, Xu Ji, Edward J Hu, Katie E Everett, Dinghuai Zhang, Yoshua Bengio, published in 2023 in the proceedings of the International Conference on Learning Representations (ICLR).

— Lahlou et al. (2023a): "A theory of continuous generative flow networks" - Salem Lahlou, Tristan Deleu, Pablo Lemos, Dinghuai Zhang, Alexandra Volokhova, Alex Hernández-García, Léna Néhale Ezzine, Yoshua Bengio, Nikolay Malkin, published in 2023 in the proceedings of the International Conference on Machine Learning (ICML).

— Lahlou et al. (2021): "DEUP: Direct Epistemic Uncertainty Prediction" - Salem Lahlou*, Moksh Jain*, Hadi Nekoei, Victor I Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov, Yoshua Bengio, published in 2023 in Transactions on Machine Learning Research (TMLR).

Additionally, Appendix B refers to the `torchgfn` code library: the author of this thesis played a leading role in its development. The contents of the appendix are based on the following:

— Lahlou et al. (2023b): "`torchgfn`: A PyTorch GFlowNet library" - Salem Lahlou, Joseph Viviano, Victor Schmidt, Yoshua Bengio, available as a preprint.

1. A star next to an author's name refers to an equal contribution among the authors with a star.

1.1. Contributions of the author

The main contributions of the author of this thesis in the four papers above are the following:

(1) **GFlowNet Foundations**:

— formally defining GFlowNets and conditional GFlowNets as mathematical objects,

— writing and proving the lemmas and propositions that show the different properties of flow networks and GFlowNets,

— coming up with the equivalence relationship between flows that ascertains the importance of Markovian flows,

— writing sections 2 and 3 of the paper and part of section 4, and making some of the figures.

(2) **GFlowNets and variational inference**:

— coming up with the graded directed acyclic graph idea that bridges the gap between GFlowNets and hierarchical variational models,

— extending the nested variational inference objective from transitions to sub-trajectories,

— extending the equivalence between the trajectory balance objective and hierarchical variational inference to the sub-trajectory balance objective and the extended nested variational inference objective,

— interpreting the log partition function in the trajectory balance loss as a learned quasi-optimal control variate in the score function estimator of the variational objective gradient,

— performing the experiments in the hypergrid domain,

— contrasting generative flow networks to the Reverse KL, Forward KL, Wake-sleep, Reverse Wake-sleep algorithms,

— writing parts of sections 2 and 4, the proofs, and the appendix.

(3) **A theory of continuous generative flow networks**:

— developing the theory and adjusting the assumptions as needed, with help from Nikolay and Tristan,

— writing most of the paper, excluding the experimental section, and including lemmas, propositions, theorems, and proofs,

— performing the experiments in the continuous square domain,

— leading the project, with help from Nikolay and Yoshua.

(4) **DEUP: Direct Epistemic Uncertainty Prediction**:

— performing the experiments showcasing the shortcomings of misspecified Gaussian processes,

— performing the experiments related to sequential model optimization,

— iterating on the writing of the paper and adjusting to different reviewers' comments,

— coming up with section 2 as a formal motivation for the proposed method,

— writing most of sections 1, 2 and 3,

— leading the project, with help from Moksh and Yoshua.

1.2. Excluded research

To keep the thesis consistent and concise, the author has decided to exclude the following publications or preprints produced during his Ph.D.:

(1) Chevalier-Boisvert et al. (2018): "Babyai: A platform to study the sample efficiency of grounded language learning" - Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, Yoshua Bengio, published in 2018 in the proceedings of the International Conference On Learning Representations (ICLR):

— The author of this thesis wrote the code for the bot, an essential component for the imitation learning experiments, and performed initial reinforcement learning, imitation learning, and transfer learning experiments.

(2) Willems et al. (2020): "Mastering rate based curriculum learning" - Lucas Willems*, Salem Lahlou*, Yoshua Bengio, available as a preprint:

— The author of this thesis performed the experiments in the supervised setting (addition of integers with a recurrent neural network) and wrote most of the paper.

(3) Liu et al. (2023): "Gflowout: Dropout with generative flow networks" - Dianbo Liu, Moksh Jain, Bonaventure Dossou, Qianli Shen, Salem Lahlou, Anirudh Goyal, Nikolay Malkin, Chris Emezue, Dinghuai Zhang, Nadhir Hassen, Xu Ji, Kenji Kawaguchi, Yoshua Bengio, published in 2023 in the proceedings of the International Conference on Machine Learning (ICML):

— The author contributed to the mathematical formalism of the proposed approach and performed experiments in sequential model optimization that were not included in the final version of the paper.

(4) Malik et al. (2023): "BatchGFN: Generative Flow Networks for Batch Active Learning" - Shreshth A. Malik, Salem Lahlou, Andrew Jesson, Moksh Jain, Nikolay Malkin, Tristan Deleu, Yoshua Bengio, Yarin Gal, presented in 2023 in the workshop on Structured Probabilistic Inference & Generative Modeling (SPIGM - ICML):

— The author wrote the GFlowNet part of the code, participated in the choice of experiments and figures to showcase the paper and helped write the paper.

(5) Chevalier-Boisvert et al. (2023): "Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks" - Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, Jordan Terry, submitted to the Datasets and Benchmarks Track of the 2023 Conference on Neural Information Processing Systems (NeurIPS), and available as a preprint:

— The author significantly contributed to the open-source `minigrid` software [2], by simplifying the code and adapting to changes in the `Gymnasium` API (Towers et al., 2023).

2. available in https://github.com/Farama-Foundation/Minigrid.

Chapter 2

Background

In this chapter, we will lay down the foundations upon which the contributions of the author are based. First, we will provide a brief overview of machine learning, highlighting a probabilistic perspective, and focusing on the supervised learning problem. We will see that the Bayesian inference problem is central to machine learning. Next, we will discuss neural networks and algorithms to train them. We will then delve into the importance of modelling uncertainty in machine learning. Finally, we will discuss the common mathematical tools and algorithms used for approximate Bayesian inference. This chapter is inspired by Bishop (2006b); Murphy (2022, 2023).

2.1. Machine Learning

Driven by the ultimate goal of replicating intelligence, by creating artificial agents capable of efficiently solving decision-making problems, the field of **machine learning** emerged as a natural candidate, as it aims to build systems that can not only *predict* observable patterns in data but also represent the regularities and the *latent structure* behind the data. Indeed, for a computationally limited artificial system tasked with solving a problem by repeatedly interacting with an environment, the entire history of actions and sensory observations is the essence of what can be used to *understand* the environment. "Data is holy" (Schmidhuber, 2009), and compressing it, by learning its underlying regular patterns, becomes necessary.

Furthermore, as the quest for achieving a general artificial intelligence continues, machine learning plays a crucial role in building a better understanding of the structure behind observed data in other sciences, such as healthcare (Beam and Kohane, 2018; Ghassemi et al., 2020) and physics (VanderPlas et al., 2012; Tanaka et al., 2021; He et al., 2023).

The rest of this section is divided as follows:

— First, we will briefly motivate and introduce Bayesian inference, a core concept in the probabilistic perspective on machine learning.

— We will then lay down the basics of Bayesian decision theory, a framework allowing to turn inferred beliefs into actions and decisions.

— Next, we will discuss the three main learning paradigms, into which most machine learning tasks could be categorized. The part on supervised learning will be expanded upon the most, given that some of the core underlying concepts are referred to extensively in the remainder of the thesis.

— Moving forward, we will delve into neural networks, a powerful class of models used in most modern applications of machine learning.

— We will conclude the section by touching upon the importance of uncertainty estimation in machine learning.

2.1.1. Probability and inference

We will adopt a probabilistic perspective on machine learning in this chapter and in the remainder of the thesis, meaning that the unknown quantities, whether they represent the predictions an agent is supposed to make or the parameters of the data-generating process a system is supposed to infer, are treated as random variables endowed with probability distributions. The probabilistic approach is adequate, not only because machine learning ultimately deals with decision-making under uncertainty, but also because other sciences heavily rely on **probabilistic modelling**, notably whenever there is an inverse problem at hand (Tarantola and Valette, 1981; Murphy, 2022, 2023). We adopt a *subjective interpretation* of probability (Hájek, 2019), where probabilities represent degrees of confidence or credences of a given *rational* agent [1]. This interpretation is central to the Bayesian approach to *inference*. It contrasts with the *frequentist interpretation*, which states [2] that " the probability of an attribute A in a finite reference class B is the relative frequency of actual occurrences of A within B" (Hájek, 1997).

2.1.1.1. Probability notations

Throughout the thesis, we will be playing with probabilities. We will clarify here some of the notational conventions used, which are common in the fields of machine learning and statistics, similar to Gelman et al. (2013):

— A **random variable**, usually denoted by a capital letter such as X, is used to represent any unknown quantity. Its **sample space**, denoted by $\mathcal{X}$ e.g., is the set of the possible values the quantity can take. An **event** is a set of outcomes (e.g., $X \in \mathcal{X}_1$, where $\mathcal{X}_1$ is a subset of $\mathcal{X}$). It can be either one of:

  — Discrete: when $\mathcal{X}$ is finite or countably finite. The beliefs of a rational agent are encoded in a **probability mass function (pmf)** $x \in \mathcal{X} \mapsto p_X(x) \in [0, 1]$

1. i.e., one that follows the three Kolmogorov axioms of probability (Kolmogorov, 1950).
2. This is in fact called *finite frequentism*. For more details, refer to Hájek (2019).

that satisfies $\sum_{x \in \mathcal{X}} p_X(x) = 1$. p_X will also be referred to as the **distribution** of X.

— Continuous: when $\mathcal{X}$ is a measure space (e.g. the real n-space $\mathbb{R}^n$ endowed with the Lebesgue measure, which we will consider hereafter, except in Chapter 4). The beliefs of a rational agent are usually[3] encoded in a **probability density function (pdf)** $x \in \mathcal{X} \mapsto p_X(x) \in \mathbb{R}^+$ that satisfies $\int_{\mathcal{X}} p(x)dx = 1$. p_X will also be referred to as the distribution of X.

— We will use the terms **distribution** and **density** interchangeably to refer to both the pmf of a discrete random variable and the pdf of a continuous one[4].

— We will not be sub-scripting the densities, and will solely use the symbol p whenever there is only one agent at play. The random variable the density is referring to can be inferred from the symbol used in its argument, e.g. $p(x)$ would refer to a random variable X, and $p(y)$ would refer to a random variable Y.

— We will use $p(. \mid .)$ to denote a conditional probability distribution.

— When referring to parameterized distributions, we will use a semicolon inside p to refer to the parameters. For example, $p(y; \boldsymbol{\theta})$ denotes the distribution parameterized with $\boldsymbol{\theta}$, and $p(y \mid x; \boldsymbol{\theta})$ denotes the conditional distribution $p(y \mid x)$ parameterized with $\boldsymbol{\theta}$. However, if the parameters $\boldsymbol{\theta}$ are treated as random variables, then usual conditioning notations will be used (e.g. $p(y \mid \boldsymbol{\theta})$ and $p(y \mid x, \boldsymbol{\theta})$).

— When referring to usual distributions, such as a univariate Gaussian with mean μ and variance σ^2, we will write $x \sim \mathcal{N}(\mu, \sigma^2)$ or $p(x) = \mathcal{N}(x; \mu, \sigma^2)$. The common distributions used in this thesis are expanded upon in Appendix A.1.1.

— The subscript of the expectation symbol $\mathbb{E}$ will be either of the form "$x \sim p(x)$" or "$p(x)$", and both notations will be used interchangeably. We will also use the notation $\mathbb{E}[. \mid x \sim p(x)]$.

2.1.1.2. Bayesian inference

The field of statistics deals with **inference** problems, where we essentially want to *infer* unknown quantities $\boldsymbol{h}$ (e.g., parameters of a model $\boldsymbol{\theta}$, or predictions y) given some observables $\boldsymbol{o}$ (e.g., an input $\boldsymbol{x}$ or a dataset of observations $\mathcal{D}$), while still accounting for our uncertainty about the unknowns. **Frequentist**, or classical statistics, which uses probability to refer to limiting relative frequencies, use *confidence intervals* to represent uncertainty about a predicted unknown. For instance, a 95 percent confidence interval should trap the value of the parameter with limiting frequency at least 95 percent (Wasserman, 2004). The **Bayesian**

3. sometimes, no probability density function exists, in which case it is common to deal with the **cumulative distribution function (cdf)** of the random variable.

4. In fact the pmf of a discrete random variable is its density with respect to the counting measure.

approach, on the other hand, treats $\boldsymbol{h}$ as a random variable and uses probability distributions to encode all *subjective* knowledge an agent has about $\boldsymbol{h}$: starting from a *prior distribution* $p(\boldsymbol{h})$ encoding initial beliefs, upon observing some data $\boldsymbol{o}$, with *likelihood* $p(\boldsymbol{o} \mid \boldsymbol{h})$, the beliefs can be updated into a *posterior distribution* $p(\boldsymbol{h} \mid \boldsymbol{o})$, using **Bayes' rule**:

$$p(\boldsymbol{h} \mid \boldsymbol{o}) = \frac{p(\boldsymbol{o} \mid \boldsymbol{h})p(\boldsymbol{h})}{p(\boldsymbol{o})}. \tag{2.1}$$

As argued in Jaynes (2003), while there has been a long controversy over frequentist versus Bayesian methods of inference, where both sides argued on the level of philosophy (Gelman and Shalizi, 2013), there is now a plethora of numerical experiments proving the superiority of the Bayesian approach to statistics when representing uncertainty:

— The Dutch book theorem (Skyrms, 1984; Hájek, 2008; Vineberg, 2022) ensures that any non-Bayesian gambler is guaranteed to make a decision leading to money loss.

— De Finetti's theorem and its extensions (De Finetti, 1929; Aldous, 1985; Kerns and Székely, 2006; Kirsch, 2018) state that if a sequence of random variables (representing the observed data) is *exchangeable*, then there is a variable $\boldsymbol{\theta}$, a prior $p(\boldsymbol{\theta})$, and a likelihood function $\boldsymbol{o} \mapsto p(\boldsymbol{o} \mid \boldsymbol{\theta})$, such that the observed data is independent and identically distributed conditional on $\boldsymbol{\theta}$.

Another common argument is that if machine learning aims to replicate human intelligence, and humans and other animals have been shown to be Bayesian to some extent (Valone, 2006; Griffiths, 2020), then it makes sense to mimic this approach to knowledge representation as well [5].

2.1.2. Bayesian Decision Theory

Ultimately, the goal of a machine learning agent is to take decisions, or actions to perform in their environment. In supervised learning (Section 2.1.3) for example, an action corresponds to predicting the value or the class associated with some observation, whereas in reinforcement learning (Section 2.1.5), an action leads to a transition in the agent's state. While Bayesian inference provides a principled way to update prior beliefs into a posterior distribution, the framework of Bayesian decision theory (DeGroot, 2005; Kochenderfer et al., 2022) turns such updated beliefs into actions, taking into account the uncertainty and the unpredictability of the underlying environment.

Following the notations of Murphy (2023), an agent is endowed with an action set $\mathcal{A}$, a set $\mathcal{H}$ representing states of nature, and a loss function $l : \mathcal{A} \times \mathcal{H} \to \mathbb{R}^+$. The loss $l(a, h)$ represents the *cost* of taking action a when the state of nature is h. More details about $\mathcal{A}$ and $\mathcal{H}$ are provided in Section 2.1.3.

5. There are various psychological studies that show on the other hand that people commonly violate the usual probability laws due to various cognitive biases (Kahneman et al., 1982).

Given a state or input x belonging to a set $\mathcal{X}$, associated to a subset of $\mathcal{H}$ via a probability distribution $p(h \mid x)$, each action $a \in \mathcal{A}$ induces a *risk*, or *expected loss*:

$$R(a \mid x) = \mathbb{E}_{p(h|x)}[l(a, h)]. \tag{2.2}$$

We say that an agent is *rational* if their goal is to always pick the risk minimizing action. Such an agent is perpetually looking for an *optimal policy*, or a **Bayes estimator**, which is a function $\pi^* : \mathcal{X} \to \mathcal{A}$ defined by:

$$\pi^*(x) \in \arg\min_{a \in \mathcal{A}} R(a \mid x). \tag{2.3}$$

2.1.3. Supervised learning

Typically, in supervised learning, an agent (also called the *learner*) is provided a dataset

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i \in [\![1,N]\!]} \tag{2.4}$$

of input-output pairs from an unknown distribution $p(\boldsymbol{x}, y)$ over a set $\mathcal{X} \times \mathcal{Y}$, called the training set, and is tasked of predicting an unseen value y corresponding to a new input $\boldsymbol{x}$.

To this end, the learner can choose to model a posterior distribution over the unknown y: $p(y \mid \boldsymbol{x})$ [6], and then use the decision theory framework (Section 2.1.2) to turn the posterior into actions or unique predictions. The actions $\mathcal{A}$ and the states of nature $\mathcal{H}$ correspond to the output space:

$$\mathcal{A} = \mathcal{H} = \mathcal{Y}. \tag{2.5}$$

In classification problems, the output space $\mathcal{Y}$ represents a set of *class labels*: $\mathcal{Y} = [\![1, C]\!] = \{1, 2, \dots, C\}$, and the loss function is usually the **zero-one loss** defined by:

$$l_{0,1}(\hat{y}, y) = \mathbf{1}(\hat{y} \neq y), \tag{2.6}$$

meaning that the learner incurs a cost of 1 when misclassifying a given input $\boldsymbol{x}$, and a cost of 0 otherwise. The risk (2.2) becomes:

$$R(\hat{y} \mid \boldsymbol{x}) = 1 - p(\hat{y} \mid \boldsymbol{x}), \tag{2.7}$$

and Bayes estimators (2.3) satisfy:

$$\pi^*(\boldsymbol{x}) \in \arg\max_{y \in \mathcal{Y}} p(y \mid \boldsymbol{x}), \tag{2.8}$$

meaning that the learner should pick the mode of the posterior distribution if tasked with making a decision at $\boldsymbol{x}$. An example of a classification task is provided in Figure 2.1.

6. It is also sometimes called the *likelihood*. See e.g. Section 2.1.3.2. Note that this is different from the posterior distribution $p(y \mid x, \mathcal{D})$, and modelling the likelihood is agnostic to whether the approach is Bayesian or not.

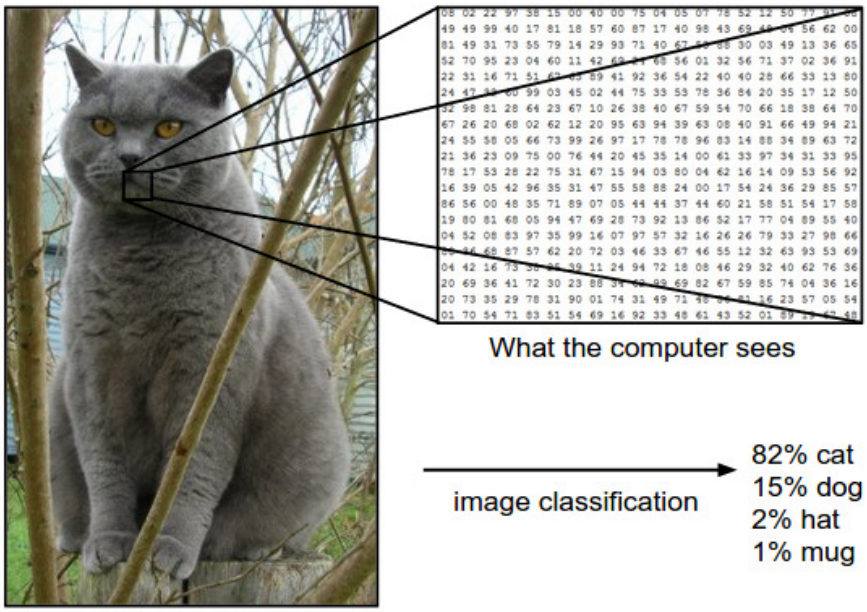


Figure 2.1 – From (Karpathy, 2023). An image classification model takes a single image and assigns probabilities to 4 labels, (cat, dog, hat, mug). As shown in the image, to an artificial agent, an image is represented as one large 3-dimensional array of numbers. In this example, the cat image is 248 pixels wide, 400 pixels tall, and has three color channels Red, Green, Blue (or RGB for short). Therefore, the image consists of 248 x 400 x 3 numbers, or a total of 297,600 numbers. Each number is an integer that ranges from 0 (black) to 255 (white). The agent's task is to turn this quarter of a million numbers into a single label, such as "cat".

In regression settings, the output space is the real line: $\mathcal{Y} = \mathbb{R}$ or a subset thereof. The most commonly used loss function is the l_2 loss, or the **squared error**:

$$l_2(\hat{y}, y) = (y - \hat{y})^2, \tag{2.9}$$

meaning that the learner incurs a cost that grows quadratically with the distance to the ground truth value. The risk becomes:

$$R(\hat{y} \mid \boldsymbol{x}) = \mathbb{E}_{p(y|\boldsymbol{x})}[y^2] - 2\hat{y}\mathbb{E}_{p(y|\boldsymbol{x})}[y] + \hat{y}^2. \tag{2.10}$$

Because the risk is a quadratic function of $\hat{y}$, then it has a unique minimizer, and there is thus a unique Bayes estimator:

$$\pi^*(\boldsymbol{x}) = \mathbb{E}_{p(y|\boldsymbol{x})}[y]. \tag{2.11}$$

The remaining central question in supervised learning, is then how to obtain the posterior distribution $p(y \mid \boldsymbol{x})$. The inference problem can be tackled with two different approaches:

— The **discriminative approach** (also called the **conditional approach** (Jebara, 2004; Lacoste-Julien, 2016)), models the posterior $p(y \mid \boldsymbol{x})$ directly.

— The **generative approach** models the joint distribution $p(\boldsymbol{x}, y)$, either directly, or by separately infering the conditional densities $p(\boldsymbol{x} \mid y)$ and the prior $p(y)$, then use Bayes' theorem (2.1) to obtain $p(y \mid \boldsymbol{x})$. Such approaches are called generative because sampling from the learned distribution $p(\boldsymbol{x}, y)$ allows to *generate* synthetic data points.

While the generative approach is naturally more computationally expensive, an inherent advantage is their ability to provide approximations of the marginal density of the data $p(\boldsymbol{x})$, thus allowing the detection of new inputs with low density under the model, for which the predictions may be of low accuracy. This problem is called **outlier detection** or **out-of-distribution detection**. This approach is more relevant to unsupervised learning (Section 2.1.4), and will be expanded upon later.

2.1.3.1. Modelling the posterior

Models of the posterior distribution $p(y \mid \boldsymbol{x})$ fall into two categories:

— **Parametric models** have a fixed number of parameters, independent of the training set size. They assume a fixed probability distribution for the data to model.

— **Non-parametric models** have a potentially infinite number of parameters. In practice, an infinite dimensional parameter is represented with finite parameters whose dimensionality grows with the amount of training data. Put differently, such models are distribution-free, as they make no strong distributional assumption on the data.

Many non-parametric models are based on comparing an input $\boldsymbol{x}$ to the inputs of the training set $\mathcal{D}$, using some distance or kernel-based similarity. Examples include models based on classification and regression trees (Breiman, 1984), Gaussian processes (Appendix A.2), and K nearest neighbours models (Kramer, 2013).

Most models discussed in this thesis are parametric and thus deserve a deeper dive. In regression, it is common to assume $p(y \mid \boldsymbol{x})$ to be a Gaussian distribution (Appendix A.1.1), whereas in classification, the output is supposed to follow a conditional Categorical distribution (Appendix A.1.1). The parameters of those distributions (e.g., the mean or the logits [7]) are functions of the input $\boldsymbol{x}$. Essentially, assuming the chosen distribution family for $p(y \mid \boldsymbol{x})$ can be parametrized with elements of a set $\mathcal{U} \subseteq \mathbb{R}^d$, parametric models require the specification of a function $f : \mathcal{X} \to \mathcal{U}$. The function f itself usually has its own parameters $\boldsymbol{\theta} \in \Theta$. Here are two examples:

— $\boldsymbol{\theta}$ can be a vector or a matrix, and f is a linear function of $\phi(\boldsymbol{x})$, a fixed feature transformation of $\boldsymbol{x}$: $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\boldsymbol{x})$. Such models are called **generalized linear models**.

7. the *logit* associated to a probability p is the logarithm of the corresponding odds, i.e., $\log p - \log(1-p)$.

— $\boldsymbol{\theta}$ can represent the parameters of a fixed-size neural network. More details are provided in Section 2.1.6.

Finding the parameters $\boldsymbol{\theta}$ of the function f is usually referred to as *fitting* or *training* the model. Once the family of distributions and the family of functions f are chosen, it is common to unambiguously denote the posterior with $p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$.

2.1.3.2. Training parametric models

For simplicity, we restrict the explanation in this section to the discriminative approach. Several principles could be used to fit a model, the most popular of which is **maximum likelihood** estimation. It aims at answering the question "which parameters $\boldsymbol{\theta}$ were the most likely to generate the observed data $\mathcal{D}$?". The likelihood of the dataset $\mathcal{D}$ (2.4), denoted by $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta})$ is:

$$\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) = \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) \tag{2.12}$$

$$= \prod_{i=1}^{N} p(\boldsymbol{x}_i) p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}). \tag{2.13}$$

Because the logarithm function is monotonically increasing, maximizing the likelihood (2.13) amounts to maximizing the **log likelihood**:

$$\log \mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i) + \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}). \tag{2.14}$$

As $\boldsymbol{\theta}$ does not intervene in the first sum of (2.12), the maximum likelihood principle boils down to finding the **MLE** estimate[8]:

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta} \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}). \tag{2.15}$$

On the other hand, **maximum a posteriori** estimation treats $\boldsymbol{\theta}$ as a random variable and aims at answering the question "Which parameters $\boldsymbol{\theta}$ are the most probable given the data $\mathcal{D}$?". It requires, however, the specification of some prior knowledge on $\boldsymbol{\theta}$ via a *prior* distribution $p(\boldsymbol{\theta})$. The logarithm of the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ can be obtained from Bayes' rule (2.1), using the likelihood $p(\mathcal{D} \mid \boldsymbol{\theta}) = \mathcal{L}(\mathcal{D}; \boldsymbol{\theta})$ (2.12):

$$\log p(\boldsymbol{\theta} \mid \mathcal{D}) = \log p(\boldsymbol{\theta}) + \sum_{i=1}^{N} \log p(\boldsymbol{x}_i) + \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}) - \log p(\mathcal{D}). \tag{2.16}$$

8. arg max usually denotes the set of maximizers, and there could be many MLE estimates. However, for simplicity, we use the notation $\boldsymbol{\theta}_{\text{MLE}} = \arg\max$ instead of $\boldsymbol{\theta}_{\text{MLE}} \in \arg\max$. This notation shortcut will be common hereafter.

Although the **evidence**[9] $p(\mathcal{D})$ can be intractable, the maximization of (2.16) boils down to finding the **MAP** estimate:

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{2.17}$$

Note that, for example, if Θ is a bounded space on which a uniform distribution can be defined, and the prior $p(\boldsymbol{\theta})$ is this uniform distribution, then the MAP estimate (2.17) coincides with the MLE estimate.

It is noteworthy that neither the uniqueness of $\boldsymbol{\theta}_{\text{MLE}}$ nor that of $\boldsymbol{\theta}_{\text{MAP}}$ can be guaranteed, in which case the arg max in (2.15) and (2.17) should be treated as sets.

2.1.3.3. The fully discriminative approach to supervised learning

A **fully discriminative approach** to supervised learning does not model a posterior $p(y \mid \boldsymbol{x})$, but aims at finding a function $f : \mathcal{X} \to \mathcal{Y}$, called a discriminant function, that minimizes a given overall risk. This approach combines both the inference and the decision stages into a single learning problem. Given a loss function $l : \mathcal{Y}^2 \to \mathbb{R}^+$, the **overall risk** associated with the function f is defined as:

$$R(f) = \mathbb{E}_{p(\boldsymbol{x},y)}[l(y, f(\boldsymbol{x}))]. \tag{2.18}$$

However, the distribution $p(\boldsymbol{x}, y)$ is unknown to the learner, and can only be approximated using the training set $\mathcal{D}$, given that each $(\boldsymbol{x}_i, y_i)$ is supposed to be a sample from $p(\boldsymbol{x}, y)$. The learner is then tasked with finding f that minimizes the **empirical risk**, and the whole approach is often referred to as **empirical risk minimization**:

$$\hat{R}(f) = \frac{1}{N} \sum_{i=1}^{N} l(y_i, f(\boldsymbol{x}_i)). \tag{2.19}$$

Oftentimes, the function f is parametrized with a vector of parameters $\boldsymbol{\theta} \in \Theta$, and the goal is to find the **ERM** estimate:

$$\boldsymbol{\theta}_{\text{ERM}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \hat{R}(f_{\boldsymbol{\theta}}). \tag{2.20}$$

The squared error (2.9) is commonly used in regression tasks. In classification tasks, however, although the zero-one loss (2.6) should, in principle, be used, given that it is robust to outliers (Nguyen and Sanner, 2013), minimizing the corresponding empirical risk is known to be an NP-hard problem (Ben-David et al., 2003), and continuous convex surrogate losses to the zero-one loss are used instead, such as the **cross-entropy loss**, also called the **log loss**, for which the action set $\mathcal{A}$ corresponds to $\mathcal{P}(\mathcal{Y})$, the set of probability distributions over $\mathcal{Y}$, usually represented with the C-simplex:

$$l_{\text{CE}}(\boldsymbol{a}, y) = -\log \boldsymbol{a}(y) \quad \text{(remember that } \boldsymbol{a} \in \mathcal{A} \text{ is a probability distribution).} \tag{2.21}$$

9. term used in MacKay (1992) for example. It is also called the **marginal likelihood**.

It is worth observing that in regression, the ERM estimate associated with the squared error (2.9) corresponds to the MLE estimate under the assumption of a Gaussian posterior $p(y \mid \boldsymbol{x})$ with fixed variance (Lemma A.4.1). Similarly, in classification, the ERM estimate associated with the log loss (2.21) corresponds to the MLE estimate under the assumption of a Categorical posterior (Lemma A.4.2). The quantity being minimized in (2.15), (2.17) and (2.20) is also commonly called the **training loss**.

2.1.3.4. The generalization problem

Recalling that what we care about in supervised learning is making predictions on unseen data, that is not part of the training set, it is natural to wonder whether the objectives (2.15) and (2.20), which rely on the training set $\mathcal{D}$ only are adequate. The ability to accurately make predictions on new examples is known as **generalization**. In fact, if the family of distributions $\{p(. \mid .; \boldsymbol{\theta}),\ \boldsymbol{\theta} \in \Theta\}$ is expressive enough, MLE and ERM will try to pick a value of $\boldsymbol{\theta}$ that make the training loss zero, essentially *interpolating* the training data. This problem is called **overfitting**.

Overfitting occurs when the trained model is too complex in the sense that it attempts to capture the noise in the training dataset $\mathcal{D}$, and thus generalize poorly. Explicitly shrinking the parameter space Θ to exclusively represent simpler models might, on the other hand, harm the training procedure and lead to **underfitting**: the trained model cannot capture the structure of the training set. A common solution to overfitting that does not explicitly shrinks Θ is to impose soft constraints on the parameters via an additive penalty term $C(\boldsymbol{\theta})$ to the negative log-likelihood or the empirical risk. This approach is called **regularization**. The term $C(\boldsymbol{\theta})$ should penalize values of $\boldsymbol{\theta}$ that lead to complex models. For example if $\boldsymbol{\theta}$ is a vector, then a common penalty is its L^2 norm: $C(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2$. A coefficient λ balances the importance given to each term in the resulting loss. For example, the MLE objective becomes:

$$\max_{\boldsymbol{\theta} \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}) + \lambda C(\boldsymbol{\theta}). \tag{2.22}$$

In MAP estimation, such soft constraints can be imposed through the prior $p(\boldsymbol{\theta})$. While a uniform prior leads to a MAP estimate that coincides with the MLE one, other choices of the prior can encode different constraints. For example, if $\boldsymbol{\theta}$ is a vector, then a common prior that penalizes large component values is an isotropic Gaussian centred at $\mathbf{0}$, with a standard deviation controlling the penalty importance, $\mathcal{N}(\mathbf{0}; \sigma^2 \boldsymbol{I})$. In fact, such a prior is equivalent to MLE estimation with L^2 regularization.

Neither the coefficient λ in (2.22), nor the parameters of the prior $p(\boldsymbol{\theta})$, are parameters of the resulting model, but rather a **hyperparameter** of the training procedure. Ideally, their chosen value should lead to a model $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}^*)$ (or $f_{\boldsymbol{\theta}^*}$ for the fully discriminative approach)

with the lowest overall risk (2.18), seen as a function of $\boldsymbol{\theta}$ [10]. However, as we do not have access to the actual distribution $p(\boldsymbol{x}, y)$, we can only approximate it using a **validation set** $\mathcal{D}_{\text{val}}$, which is a subset of $\mathcal{D}$ set apart, and not used for training, and it is common practice to pick the value of λ, or any other hyperparameter of the model or the learning procedure, or even the family of models itself (this is called *model selection*), by minimizing the resulting validation risk [11] for example:

$$\hat{R}_{\text{val}} = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_{\text{val}}} l(y, f_{\boldsymbol{\theta}^*}(\boldsymbol{x})). \tag{2.23}$$

The final performance of the chosen model can be evaluated with a yet separate subset of $\mathcal{D}$, called the **test set**, $\mathcal{D}_{\text{test}}$, in a similar fashion.

Using a validation set for model selection and hyperparameter optimization might not be sensible in limited data settings. A common alternative is $k-$fold **cross validation**, where $\mathcal{D}$ is split into k subsets, called the *folds*, and k models are trained, one on each of the k unions of $k-1$ folds, and separately evaluated on the fold unused for training. The chosen model and hyperparameters are those that maximize the average validation risk of the k models.

2.1.3.5. Bayesian models

Going back to MAP estimation (2.16), that treats $\boldsymbol{\theta}$ as a random variable, a Bayesian treatment of $\boldsymbol{\theta}$ involves not only finding the mode of the posterior but actually modelling the whole posterior:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}, \tag{2.24}$$

where the evidence is defined by:

$$p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{2.25}$$

One advantage of modelling the posterior is their resulting probabilistic predictions. Given a new input $\boldsymbol{x}$, the **posterior predictive** distribution is available to the agent, by *marginalizing* over all parameter settings, rather than maximizing:

$$p(y \mid \boldsymbol{x}, \mathcal{D}) = \int p(y \mid \boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta} \mid \mathcal{D})d\boldsymbol{\theta}. \tag{2.26}$$

The posterior predictive can be used as a substitute $p(y \mid \boldsymbol{x})$ in (2.8) and (2.11) for decision making. However, there is a benefit in considering the whole predictive distribution rather than its mode or its mean: it quantifies **uncertainty** about the unknown y.

At first glance, one might argue that even with parametric models of the posterior, the distribution $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}^*)$, where $\boldsymbol{\theta}^*$ is the MLE or the MAP estimate, also encodes uncertainty

10. Similar to ERM, we can define an overall risk with the MLE and MAP approaches, by replacing the empirical distribution with a distribution over the unknown $p(\boldsymbol{x}, y)$.
11. or the log-likelihood of the validation set in the case of MLE or MAP.

about y. In fact, $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}^*)$ can be seen as an approximation of the posterior predictive, using the Dirac distribution at $\boldsymbol{\theta}^*$ (Appendix A.1.1) as a replacement for the actual posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ in (2.26):

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}^*) = \int p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) \delta_{\boldsymbol{\theta}^*}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{2.27}$$

The MAP estimate is thus called a **plugin approximation** of the posterior predictive. However, its encoded uncertainty is merely a result of the distributional choice for $p(y \mid \boldsymbol{x})$ (e.g., Gaussian for regression, and Categorical for classification), and only accounts for **aleatoric uncertainty**, while completely disregarding the **epistemic uncertainty** of the learner. More on this on Section 2.1.7.

By modelling uncertainty in the parameter space, rather than taking the risk of selecting a unique value for model parameters which leads to a complex model, Bayesian models are less prone to overfitting than ERM, MLE, or MAP estimation.

Note that, unlike MAP estimation, which disregards the evidence in the optimization, evaluating the posterior (2.24) and (2.26) requires the evaluation of $p(\mathcal{D})$.

There are scenarios in which exact posterior inference is tractable; this is when the prior $p(\boldsymbol{\theta})$ is **conjugate** to the likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$, meaning that the family of distributions $\mathcal{F}$ the prior belongs to is *closed* under Bayesian updating (i.e., the posterior would also be a member of $\mathcal{F}$). This is the case, for example, when the prior and likelihood are of the following form [12]:

$$p(\boldsymbol{\theta}) \propto e^{\boldsymbol{\lambda}_0^\top \mathcal{T}(\boldsymbol{\theta})}, \tag{2.28}$$

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) \propto e^{\boldsymbol{\lambda}_i(\boldsymbol{x}, y)^\top \mathcal{T}(\boldsymbol{\theta})}, \tag{2.29}$$

for some *sufficient statistics* $\mathcal{T} : \Theta \to \mathbb{R}^s$, and *canonical parameters* $\boldsymbol{\lambda} \in \mathbb{R}^s$.

In most interesting cases, however, we cannot compute posteriors exactly, and approximate inference techniques become necessary (Section 2.2).

Finally, it is worth noting that the Bayesian approach can also be applied to non-parametric models. For example, this is commonplace for Gaussian processes (Appendix A.2).

2.1.4. Unsupervised learning

Unlike supervised learning, in which a learner is provided with a labelled dataset, unsupervised learning revolves around learning the regular structures of an unlabelled dataset (2.4)

$$\mathcal{D} = \{\boldsymbol{x}_i\}_{i \in [\![1, N]\!]}. \tag{2.30}$$

12. We say that the prior and the likelihood belong to an exponential family.

The probabilistic perspective on unsupervised learning hinges on fitting an unconditional model $p(\boldsymbol{x})$. Examples of tasks that can be solved using unsupervised learning algorithms include:

— **Clustering**: the goal is to partition the input space into regions containing similar points. **K-means** (MacQueen, 1967; Lloyd, 1982) is a popular clustering algorithm and relies on alternatively finding the cluster centers and assigning each point $\boldsymbol{x} \in \mathcal{D}$ to its closest center. K-means is illustrated in Figure 2.2. Other popular methods are surveyed in Saxena et al. (2017); Ezugwu et al. (2022).

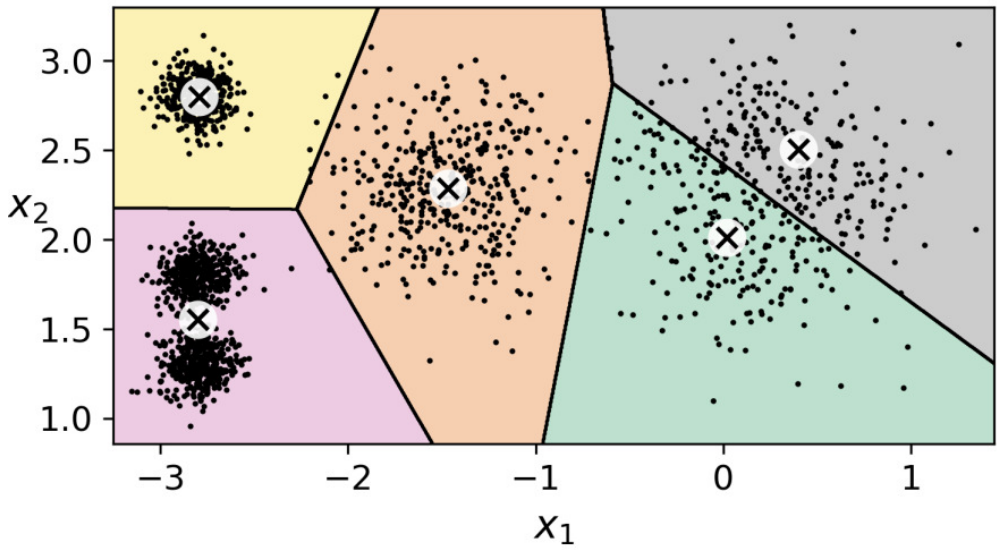


Figure 2.2 – From Géron (2022). An illustration of K-means clustering in two dimensions. The crosses are cluster centers, and different colours refer to different clusters.

— **Data compression and dimensionality reduction**: the goal is to map each input $\boldsymbol{x} \in \mathcal{X}$ to a lower-dimensional vector $\boldsymbol{z} \in \mathcal{Z}$. $\mathcal{Z}$ is called the latent space, and $\boldsymbol{z}$ is called the embedding of $\boldsymbol{x}$. **Principal components analysis (PCA)**(Wold et al., 1987) is the most popular form of dimensionality reduction when $\mathcal{X} \subseteq \mathbb{R}^D$, and relies on linearly projecting the data from $\mathbb{R}^D$ to a lower dimensional space $\mathbb{R}^L$, defined by the *principal components* of the data. An illustration of PCA is provided in Figure 2.3. **Autoencoders** are an extension of this method that relies on using neural networks (Section 2.1.6) to embed the data into a lower dimensional space, trained by minimizing a reconstruction loss. It has been shown in Japkowicz et al. (2000) that the learned representations are more useful than those obtained with PCA, for example. **Variational autoencoders (VAEs)** (Kingma and Welling, 2014a; Rezende et al., 2014b) are a probabilistic version of the autoencoders, which fits a distribution over latents $p(\boldsymbol{z} \mid \boldsymbol{x})$ rather than learning a deterministic mapping

from $\mathcal{X}$ to $\mathcal{Z}$, and have the advantage of being a **generative model**, as they can be used to create new samples from $p(\boldsymbol{x})$.

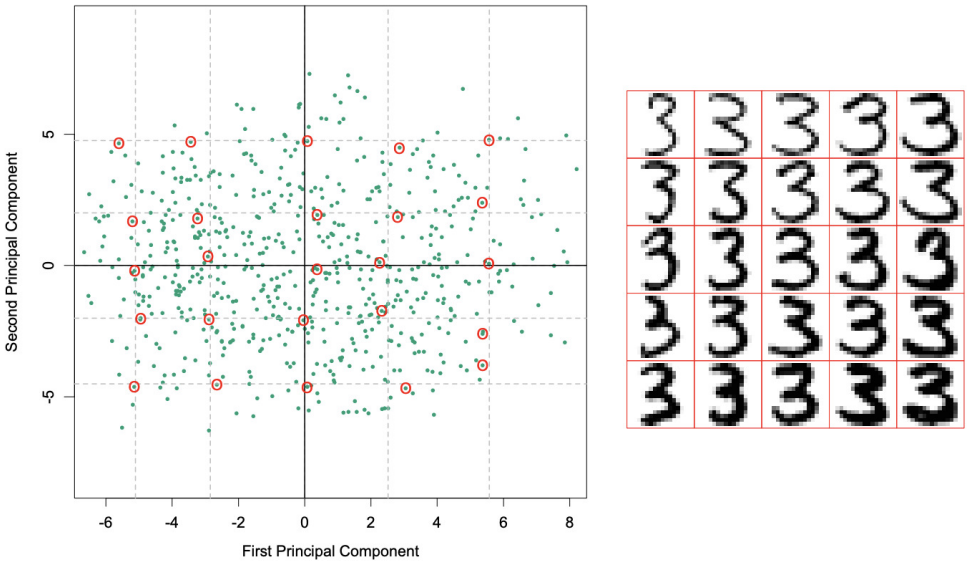


Figure 2.3 – From Hastie et al. (2009). An illustration of PCA applied to MNIST digits (Deng, 2012) from the class 3. **Left:** the first two principal components of the handwritten threes. The circled points are the closest projected images to the vertices of a grid, defined by the marginal quantiles of the principal components. **Right:** The images corresponding to the circled points. These show the nature of the first two principal components (thickness and curliness).

— **Density estimation**: the goal is to evaluate the probability mass or density $p(\boldsymbol{x})$ of a point $x \in \mathcal{X}$, which can be useful for out-of-distribution detection for example. The simplest approach to this problem in vector spaces is **kernel density estimation (KDE)** (Węglarczyk, 2018), which is a non-parametric method that relies on a density kernel $\kappa : \mathcal{X} \to \mathbb{R}^+$ to define $p(\boldsymbol{x})$. More details are provided in Appendix A.3. Aggarwal et al. (2001) showed that it suffers from the **curse of dimensionality**, meaning that as the dimensionality of the data grows, the number of examples required to generalize accurately grows exponentially. Parametric models are thus preferred in higher dimensions. For example, in **normalizing flows**, the goal is to learn an invertible transformation $f : \mathbb{R}^D \to \mathbb{R}^D$ that transforms samples $\boldsymbol{u}$ from a simple base distribution $p(\boldsymbol{u})$ to samples $\boldsymbol{x}$ from the target distribution $p(\boldsymbol{x})$. The transformation f is made by composing different parametric building blocks, such as *coupling layers* (Dinh et al., 2014). Two comprehensive reviews of normalizing flows are provided in Papamakarios et al. (2021); Kobyzev et al. (2020). In **energy-based**

models, the target distribution uses a parametrized **energy function** $\mathcal{E}_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}^+$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{e^{-\mathcal{E}_{\boldsymbol{\theta}}(\boldsymbol{x})}}{Z_{\boldsymbol{\theta}}}, \tag{2.31}$$

where $Z_{\boldsymbol{\theta}} = \int e^{-\mathcal{E}_{\boldsymbol{\theta}}(\boldsymbol{x})} d\boldsymbol{x}$ is called the **partition function**. Because the energy function does not need to integrate to one, it is common to use a variety of neural networks (Section 2.1.6) to parametrize it. Maximum likelihood estimation is the standard way of training energy-based models and hinges on maximizing, using gradient-based optimization techniques Section 2.1.6.2, the expected log-likelihood over the data distribution $p_{\text{data}}(\boldsymbol{x})$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]. \tag{2.32}$$

The gradient of (2.32) with respect to $\boldsymbol{\theta}$ involves the term $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[-\nabla_{\boldsymbol{\theta}} \mathcal{E}_{\boldsymbol{\theta}}(\boldsymbol{x})]$, which can only be approximated, as long as we can draw random samples from the current model $p_{\boldsymbol{\theta}}$ (see Section 2.2.2 for a discussion on sampling). Alternatives to MLE training of energy-based models include the **score matching** objective (Hyvärinen, 2005), where the goal is to equalize the score of the EBM $\nabla_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ and the score of the data distribution $\nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})$.

— **Generation**: the goal is to create new data samples that would come from the same distribution as that of a given dataset $\mathcal{D}$ (2.30). Examples include:

(1) **Auto-regressive** models, i.e., those that factorize as:

$$p(\boldsymbol{x}_{1:T}) = p(\boldsymbol{x}_1) \prod_{t=2}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1}), \tag{2.33}$$

where we substituted $\boldsymbol{x}$ with $\boldsymbol{x}_{1:T}$ to emphasize that the inputs have T components. Auto-regressive models are also generative, as it would suffice to sequentially sample from $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1})$ to obtain a sample from the target $p(\boldsymbol{x}_{1:T})$. Usually, a restricted function form is imposed on the conditionals $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1})$. Neural networks can be used, for example, leading to **neural language models** (Bengio et al., 2000), neural auto-regressive density estimators (**NADE**) in Larochelle and Murray (2011), and the **pixelCNN** variants (van den Oord et al., 2016). More recently, transformer decoders (Section 2.1.6) are becoming increasingly popular in parameterizing autoregressive models and are the basis of many popular generative models for sequences, such as the Generative pre-training transformer (**GPT**) in Radford et al. (2018, 2019); Brown et al. (2020); OpenAI (2023) for text generation, or **DALLE-E** (Ramesh et al., 2021, 2022) for *text-to-image* generation, for which an example is provided in Figure 2.4.

(2) **Diffusion models** (Sohl-Dickstein et al., 2015a; Ho et al., 2020a; Kingma et al., 2021; Song et al., 2020) are a recent and popular class of models that use a

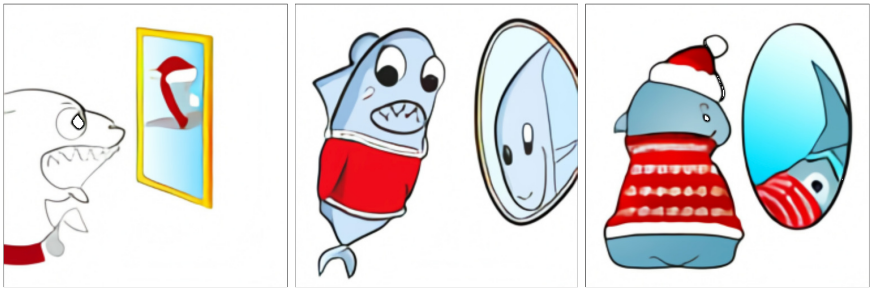


Figure 2.4 – Some images generated by the DALL-E model from https://openai.com/research/dall-e in response to the text prompt: "an illustration of a baby shark in a Christmas sweater staring at its reflection in a mirror".

diffusion process to gradually convert observed data, denoted by $\boldsymbol{x}_0$ into a noisy version $\boldsymbol{x}_T$, by passing it through T steps of a stochastic encoder $q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})$, and such that $p(\boldsymbol{x}_T) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The core idea is to learn a *reverse process* to undo the noising (forward) process, by passing the noise $\boldsymbol{x}_T$ through T steps of a stochastic decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ until we generate $\boldsymbol{x}_0$. The process is illustrated in Figure 2.5.

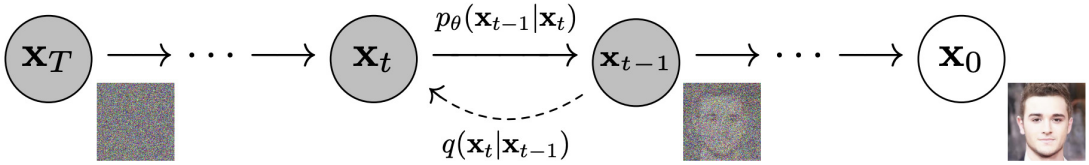


Figure 2.5 – From Ho et al. (2020a). Illustration of the noising (forward) and the denoising (reverse) processes in a diffusion model.

2.1.4.1. Probabilistic graphical models

Another form of unsupervised learning is **Bayes net structure learning**, where the goal is to estimate the structure of the *directed graphical model* that generated the data (2.30). In fact, any distribution over T variables can be compactly represented with a **directed acyclic graph (DAG)**, encoding the conditional independences underlying the data, as illustrated in Figure 2.6. Said differently, going back to (2.33), each conditional $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1})$ can be rewritten as $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{pa(t)})$, where $pa(t) \subseteq [\![1, t-1]\!]$ is the largest set satisfying the conditional independence property (Definition A.1.1):

$$\boldsymbol{x}_t \perp \boldsymbol{x}_{[\![1,t-1]\!] \setminus pa(t)} \mid pa(t). \tag{2.34}$$

$pa(t)$ is called the parent set of the node t, where each of the indices $1, 2, \ldots, T$ is represented with a node. The set of parents $\{pa(t),\ t \in [\![1, T]\!]\}$, with the convention $pa(1) = \emptyset$, essentially defines a DAG.

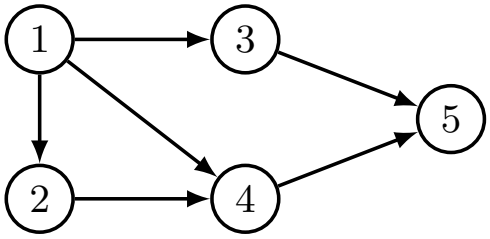


Figure 2.6 – Illustration of a directed acyclic graph whose nodes represent 5 variables and whose edges encode conditional independences. The network structure specifies a factorization of the joint distribution:
$p(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5) = p(\boldsymbol{x}_1)p(\boldsymbol{x}_2 \mid \boldsymbol{x}_1)p(\boldsymbol{x}_3 \mid \boldsymbol{x}_1)p(\boldsymbol{x}_4 \mid \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)p(\boldsymbol{x}_5 \mid \boldsymbol{x}_3, \boldsymbol{x}_4)$.

A simple example is that of **Markov chains**, where the **Markov assumption** is made:

$$\forall t > 1,\ p(\boldsymbol{x}_t \mid \boldsymbol{x}_{1:t-1}) = p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}). \tag{2.35}$$

Such probabilistic graphical models (**PGMs**) can also be used to encode *causal relationships* between different variables. Additionally, one can be interested in learning both the DAG structure and the parameters of the conditionals $p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{pa(t)})$ (assuming they are parametric). From a probabilistic perspective, this task amounts to learning a posterior over graphs G and parameters $\boldsymbol{\theta}$ from the *observational* data $\mathcal{D}$:

$$p(G, \boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(G)p(\boldsymbol{\theta} \mid G)p(\mathcal{D} \mid G, \boldsymbol{\theta})}{p(\mathcal{D})}. \tag{2.36}$$

Friedman and Koller (2000) provide an overview of the problem and Friedman et al. (2013) provide a bootstrap approach to the problem.

There is, however, a fundamental problem to causal learning, i.e., learning the DAG structure where each edge $i \to j$ indicates that the variable $\boldsymbol{x}_i$ casually influences the corresponding value of $\boldsymbol{x}_j$, given that even with infinite data, it is only possible to know the DAG up to its **Markov equivalence class** (Definition A.1.2). Identifying the right ground-truth graph can be improved by performing **interventions**, i.e., experiments affecting the value of the variables corresponding to a specific set of graph nodes, and observing how they change the value of the other variables (Tong and Koller, 2001; Murphy, 2001). An example is provided in Figure 2.7.

2.1.5. Reinforcement Learning

Reinforcement Learning (RL) deals with sequential decision-making in order to maximize a specific utility function. Contrary to supervised Learning, RL agents aren't trained by observing a knowledgeable teacher's actions. Instead, they must learn which actions lead to the highest rewards by trying them and observing their effects on the environment. In the usual framework, at each time step, an agent observes the environment state and then

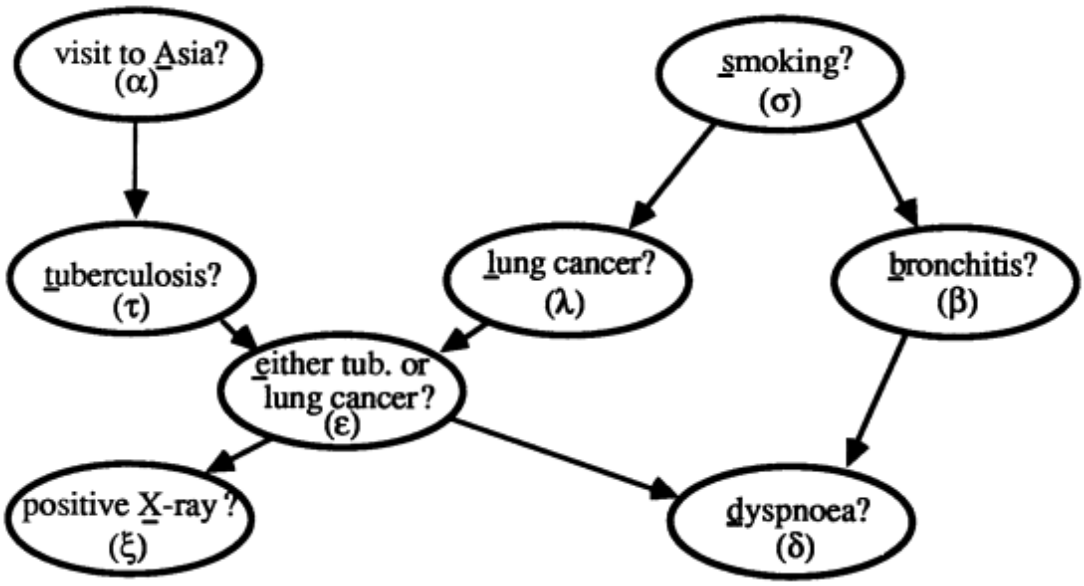


Figure 2.7 – From Lauritzen and Spiegelhalter (1988). The *Asia network*, an example of a causal network depicting causal relationships between 8 variables. The corresponding dataset is a testbed for causal learning algorithms that select which interventions to perform to discern the graph from its Markov equivalence class.

chooses an action that results in a reward and a transition to another state. Using trial-and-error, the agent should learn how to optimize some cumulative reward in various scenarios: the environment can be stochastic, the rewards can be delayed, or the agent may only observe some partial information about the visited states, etc...

Recently, RL algorithms were successfully used to train agents to reach super-human performances in different types of games: Atari games (Bellemare et al., 2013; Mnih et al., 2015), Go (Silver et al., 2016, 2017b), Chess (Silver et al., 2017a), Poker (Moravčík et al., 2017), etc... RL has also been shown to apply to real-world problems such as self-driving cars (Pan et al., 2017), smart grids (François-Lavet et al., 2016), and robotics (Gu et al., 2017; Nagabandi et al., 2019). More recently, RL techniques have been incorporated in GPT text generators, using human feedback as a basis of a reward function (Christiano et al., 2017; Ouyang et al., 2022).

RL agents face the *exploration-exploitation* dilemma: should they act in order to improve their knowledge about the environment (explore) or in order to maximize the returns using their current knowledge (exploit)?

Markov Decision Processes (MDPs) (Bellman, 1957) are a natural way to represent the environment when formalizing the agent-environment interface. An MDP is a tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, P is a state probability distribution that models the dynamics of the environment, $R : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{R})$ is the reward function, where $\mathcal{R} \subseteq \mathbb{R}$ is the set of possible reward values and $\mathcal{P}(\mathcal{R})$ is the set of probability

distributions over reward values. ($R(s,a)$ is a random variable that represents the reward obtained when taking action a at state s), and $\gamma \in [0,1)$ is the discount rate determining future rewards' present value. P has to satisfy the Markov property that for any state s_t and state-action history $(s_0, a_0, \ldots, s_{t-1}, a_{t-1})$:

$$P(s_{t+1} \mid s_0, a_0, \ldots, s_t, a_t) = P(s_{t+1} \mid s_t, a_t), \tag{2.37}$$

which means that the future is independent of the past given the present, i.e., the agent has no interest in looking at the full history to make decisions. Because of the Markov property, it is sometimes useful to model the dynamics of the environment using a transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ ($\mathcal{T}(s,a,s')$ represents the probability of transitioning to state s' when taking action a from state s).

The behaviour of the agent can be described with a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$. For simplicity, we will overload the notation and use both $\pi(a \mid s)$ and $\pi(s,a)$ to refer to the probability assigned by the policy of taking action a at state s. The goal of the agent is to find a policy function that maximizes the expected return, defined as the sum of discounted future rewards when starting from its initial state s and following π:

$$V^\pi(s) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\middle|\; s_0 = s, a_t \sim \pi(. \mid s_t), s_{t+1} \sim P(. \mid s_t, a_t)\right]. \tag{2.38}$$

Similar to the *state value function* V^π, we can define the *state-action value function* Q^π, that represents the expected return when starting from a state s, taking action a, and following the policy π:

$$Q^\pi(s,a) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\middle|\; (s_0, a_0) = (s,a), a_t \sim \pi(. \mid s_t), s_{t+1} \sim P(. \mid s_t, a_t)\right]. \tag{2.39}$$

Because of the Markov property, the value function (2.39) satisfies a recursive equation, called the *Bellman equation*:

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(.|s,a), s' \sim \pi(.|s')}\left[Q^\pi(s', a')\right], \tag{2.40}$$

The goal of an RL agent is to find the optimal policy π^* that maximizes the expected return $V^\pi(s)$ at every state s. The value functions of π^* are denoted by V^* and Q^*. These optimal value functions satisfy the following equations, called the *Bellman optimality equations*:

$$V^*(s) = \max_{a \in \mathcal{A}} \left(\mathbb{E}\left[R(s,a)\right] + \gamma \mathbb{E}_{s' \sim P(.|s,a)}[V^*(s')]\right), \tag{2.41}$$

$$Q^*(s,a) = \mathbb{E}\left[R(s,a)\right] + \gamma \mathbb{E}_{s' \sim P(.|s,a)}[\max_{a' \in \mathcal{A}} Q^*(s', a')]. \tag{2.42}$$

In the general case, there might exist more than one optimal policy. However, knowing Q^* is sufficient to define a deterministic optimal policy:

$$\pi^*(a \mid s) = \mathbf{1}(a = \arg\max_{a' \in \mathcal{A}} Q^*(s, a')). \tag{2.43}$$

Knowing the optimal state value function V^* is also sufficient to define an optimal policy given that:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a). \tag{2.44}$$

Value functions are crucial in RL since the Bellman equations are used to design algorithms that solve the return maximization problem. **Q-learning** (Watkins and Dayan, 1992), for example, is one of the earliest breakthroughs in RL and aims at learning the values $Q(s, a)$, stored in a lookup table. It relies on bootstrapping: at time step t, the value of the pair (s_t, a_t) is updated according to the following rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)). \tag{2.45}$$

Watkins and Dayan (1992) prove that under the condition that each state-action pair is visited infinitely often, these updates create a sequence of Q functions $\{Q_t\}_{t \geq 0}$ that converges almost surely to the optimal state-action value function Q^*.

Q-learning becomes impractical when dealing with large state spaces and action spaces. In this context, we may learn the state-action values using a parameterized value function $(s, a) \mapsto Q(s, a; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of parameters. Riedmiller (2005) propose to minimize an error of the form $(Q(s_t, a_t; \boldsymbol{\theta}) - (R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_{t+1}; \boldsymbol{\theta})))^2$ with respect to $\boldsymbol{\theta}$ instead, using gradient descent techniques (Section 2.1.6.2), and using multilayer perceptrons (Section 2.1.6.1) as function approximators. However, as shown in Boyan and Moore (1995) and Sutton and Barto (2018), combining Q-learning and function approximators with extrapolation capabilities leads to instabilities during learning. This problem is known as the deadly triad of reinforcement learning.

The **Deep Q-Network (DQN)** algorithm introduced in Mnih et al. (2015) tries to limit the instabilities introduced by the usage of neural networks as function approximators by relying on two heuristics: using a replay memory (Lin, 1992) from which the transitions will be sampled uniformly at random to make updates to the Q-Network, and periodically updating the target network that is used to define $(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}))$, called the temporal difference (TD) target. These modifications made it possible to train agents that reach super-human performances in dozens of Atari games. This work paved the way for more research in *Deep Reinforcement Learning* (DRL): the combination of reinforcement learning and deep learning techniques.

A popular alternative to DQN-based algorithms in DRL is the family of algorithms based on **policy gradients**, which model the optimal policy π directly, as they can be used to learn

stochastic policies, which is necessary, for example, in partially observable environments. They rely on the policy gradient theorem (Sutton and Barto, 2018), which gives an analytic formula for the gradient of the expected return at the initial state as a function of the parameters $\boldsymbol{\theta}$ of the policy π being learned. The **REINFORCE** (Williams, 1992) algorithm was the precursor to many recent improvements that include actor-critic algorithms (Konda and Tsitsiklis, 2000), the asynchronous advantage actor-critic (A3C) algorithm (Schulman et al., 2015), the trust region policy optimization (TRPO) algorithm (Mnih et al., 2016), and the proximal policy optimization (PPO) algorithm (Schulman et al., 2017).

2.1.6. Deep Learning

2.1.6.1. Neural networks

As previously mentioned, most recent successes in machine learning rely in one way or another on using neural networks as discriminative functions to model posteriors $p(y \mid \boldsymbol{x})$. Neural networks, which have their origins in attempts to formalize information processing in brains (Rosenblatt et al., 1962), bypass the need of manually specifying feature transformations, as in generalized linear models $\boldsymbol{x} \mapsto f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\boldsymbol{x})$. The basic neural network model uses a series of transformations relying on D_1 transformations of the D input variables $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_D$ (we assume here that $\mathcal{X} \subseteq \mathbb{R}^D$) of the form:

$$\boldsymbol{a}_j^{(1)} = \sum_{i=1}^{D} \boldsymbol{\theta}_{j,i}^{(1)} \boldsymbol{x}_i + \boldsymbol{\theta}_{j,0}^{(1)}, \quad \forall j \in [\![1, D_1]\!], \tag{2.46}$$

where the superscript (1) indicates that the parameters $\boldsymbol{\theta}^{(1)} \in \mathbb{R}^{D_1(D+1)}$ are in the first **layer** of the network. The quantities $\boldsymbol{a}_j^{(1)}$are called **pre-activations**, and are transformed using a differentiable non-linear function h, called the **activation function**, leading to the **activations** $\boldsymbol{z}^{(1)} \in \mathbb{R}^{D_1}$:

$$\boldsymbol{z}_j^{(1)} = h(\boldsymbol{a}_j^{(1)}), \quad \forall j \in [\![1, D_1]\!]. \tag{2.47}$$

Multiple layers can be composed. For example, a second layer of D_2 **units**, with parameters $\boldsymbol{\theta}^{(2)} \in \mathbb{R}^{D_2(D_1+1)}$, can be used similarly, leading to activations $\boldsymbol{z}^{(2)}$ in $\mathbb{R}^{D_2}$. After the L-th layer, leading to activations $\boldsymbol{z}^{(L)}$ in $\mathbb{R}^{D_L}$, an **output layer** transforms $\boldsymbol{z}^{(L)}$ to outputs in $\boldsymbol{z}^{(o)} \in \mathbb{R}^{D_o}$ using an output function that depends on the task at hand. For example, in regression problems, it is common to use the identity function:

$$\boldsymbol{z}^{(o)} = \boldsymbol{z}^{(L)}. \tag{2.48}$$

In multi-class classification problems, with C classes, D_L is chosen to be equal to C, and the $softmax$ function is used in the output layer:

$$\boldsymbol{z}^{(o)} = softmax(\boldsymbol{z}^{(L)}), \tag{2.49}$$

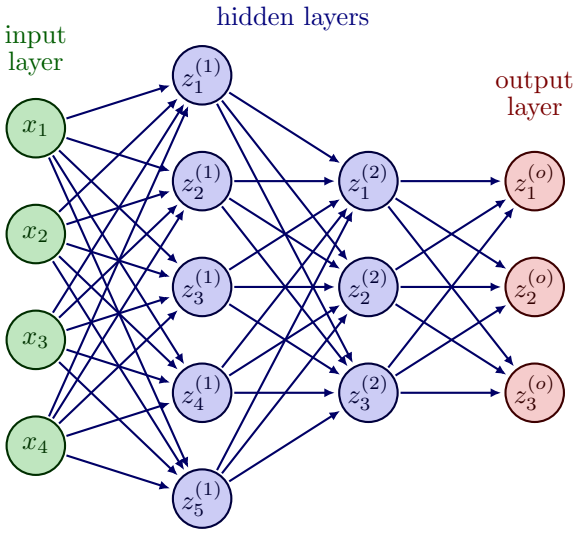


Figure 2.8 – Illustration of a feedforward neural network with 2 hidden layers of size 5 and 3 respectively.

which means that for every $i \in [\![1, C]\!]$:

$$\boldsymbol{z}_i^{(o)} = \frac{e^{\boldsymbol{z}_i^{(L)}}}{\sum_{j=1}^{C} e^{\boldsymbol{z}_j^{(L)}}}, \tag{2.50}$$

thus efficiently representing a probability distribution over $\mathcal{Y} = [\![1, C]\!]$.

Usual activation functions are the sigmoid function, the hyperbolic tangent function, or the rectifier linear unit:

$$h_1(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.51}$$

$$h_2(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.52}$$

$$h_3(x) = \mathrm{ReLU}(x) = \max(0, x). \tag{2.53}$$

The neural network described thus far is called a **feedforward neural network**, or a **multilayer perceptron (MLP)**. Its parameters $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(L)}$ are often compactly represented as a large vector $\boldsymbol{\theta}$. A visual illustration is provided in Figure 2.8.

Feedforward neural networks have been shown to be *universal function approximators* (Funahashi, 1989; Cybenko, 1989; Hornik et al., 1989), in the sense that any continuous function on a compact space can be uniformly approximated with a linear two-layer network (i.e., one **hidden** layer, and a linear output layer) with some given parameters $\boldsymbol{\theta}$, up to any desired level. However, multiple works (Håstad, 1986; Bengio et al., 2006; Montúfar et al., 2014; Raghu et al., 2016) have shown that *deep networks*, i.e., those with a higher number of layers, work better than shallow ones, given that they can leverage the features learned in earlier layers.

Besides the linear layer (2.46), different building blocks can be used to construct a neural network. For example, with image data, applying the same weight matrix to each patch of the image is common, thus essentially reducing the number of parameters. This can be done with a **convolutional layer**, parametrized with a **kernel** matrix $\boldsymbol{\theta} \in \mathbb{R}^{h \times w}$, and used for input images $\boldsymbol{x} \in \mathbb{R}^{H \times W}$ leading to pre-activations:

$$\boldsymbol{a}_{i,j} = \sum_{u=0}^{h-1} \sum_{v=0}^{w-1} \boldsymbol{x}_{i+u,j+v} \boldsymbol{\theta}_{u,v}. \tag{2.54}$$

Convolutional layers, along with **pooling layers**, which are parameter-free layers that compute the maximum or average value in each local patch of an input matrix, are the basis of convolutional neural networks (CNNs), which are at the heart of all modern image classifiers (Yalniz et al., 2019).

Attention layers (Bahdanau et al., 2015; Vaswani et al., 2017), on the other hand, are central to the **transformer** architectures, that are popular for sequence processing. Denoting by $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ a sequence of T vectors of size D, and representing $\boldsymbol{X}$ as a matrix of size $D \times T$, three matrices, called the **query, key,** and **value** matrices, are evaluated:

$$\boldsymbol{Q} = \boldsymbol{W}_q \boldsymbol{X} + \boldsymbol{b}_q \boldsymbol{1}^\top, \tag{2.55}$$

$$\boldsymbol{K} = \boldsymbol{W}_k \boldsymbol{X} + \boldsymbol{b}_k \boldsymbol{1}^\top, \tag{2.56}$$

$$\boldsymbol{V} = \boldsymbol{W}_v \boldsymbol{X} + \boldsymbol{b}_v \boldsymbol{1}^\top, \tag{2.57}$$

where $\boldsymbol{W}_q, \boldsymbol{W}_k \in \mathbb{R}^{d_{attn} \times D}$ and $\boldsymbol{W}_v \in \mathbb{R}^{d_{out} \times D}$, leading to the score matrix $\boldsymbol{S} \in \mathbb{R}^{T \times T}$ and the output matrix $\tilde{\boldsymbol{V}} \in \mathbb{R}^{d_{out} \times T}$:

$$\boldsymbol{S} = \boldsymbol{K}^\top \boldsymbol{Q}, \tag{2.58}$$

$$\tilde{\boldsymbol{V}}_{.,j} = \sum_{i=1}^{T} softmax\left(\frac{1}{\sqrt{d_{attn}}} \boldsymbol{S}_{.,j}\right)_i \boldsymbol{V}_{.,i}, \quad \forall j \in [\![1, T]\!]. \tag{2.59}$$

Details about efficiently implementing attention are provided in Phuong and Hutter (2022).

Other layers are commonly used, such as the residual or skip connections (He et al., 2016) that allow the signal to skip one more layer, thus preventing a potential waste of information in deep networks, normalization layers (Ioffe and Szegedy, 2015; Ba et al., 2016; Ulyanov et al., 2016; Wu and He, 2018) that ensure that the magnitude of the weights across layers and dimensions are comparable, dropout layers (Srivastava et al., 2014) that introduce a specific stochasticity in the training procedure that helps prevent overfitting, and recurrent layers (Chung et al., 2015) that augment the input $\boldsymbol{x}$ with an updatable state $\boldsymbol{s}$, in order to account for temporality in the input (e.g., if $\boldsymbol{x}$ is a sequence $\boldsymbol{x}_{1:T}$).

In the next section, we discuss how the parameters $\boldsymbol{\theta}$ or $\boldsymbol{W}$ of neural network layers can be fit from data.

2.1.6.2. Optimization and backpropagation

Optimizing a loss function $\boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\theta})$ such as (2.15), (2.17) and (2.20), when it is a differentiable objective of its parameters $\boldsymbol{\theta}$ can be done using gradient-based methods.

Such methods require the specification of a starting point $\boldsymbol{\theta}_0$, and then perform iterative updates of the form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \boldsymbol{d}_t, \tag{2.60}$$

where $\boldsymbol{d}_t$ is a *descent direction*, and η_t is called the **learning rate**, which is a hyperparameter of the training procedure.

The **gradient-descent** algorithm uses the gradient as a descent direction:

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_t}, \tag{2.61}$$

and the updates are done until reaching a *stationary point*, at which the gradient is zero.

In many practical applications however, computing the gradient of the loss function is too computationally expensive, as that would require the evaluation of a loss or log-likelihood at each point of a potentially large training set. When the loss is a finite sum objective of the form:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(\boldsymbol{\theta}), \tag{2.62}$$

it is common to use an *unbiased estimate of the gradient*, evaluated on a **minibatch** $\mathcal{B}_t \subseteq [\![1, N]\!]$ of size $B = |\mathcal{B}_t|$:

$$\hat{\boldsymbol{d}}_t = \frac{1}{B} \sum_{i \in \mathcal{B}_t} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i(\boldsymbol{\theta}_t). \tag{2.63}$$

This method is called **stochastic gradient descent (SGD)**, is central to all optimization methods used in modern machine learning. SGD can be improved by incorporating **momentum** (Bertsekas, 1997) to move faster along known good directions, or **Nesterov momentum** (Nesterov, 2003) to perform a one-step look ahead when deciding the descent direction, or **RMSProp** (Tieleman et al., 2012) to automatically scale the effective learning rate, or a combination of some of these elements, called the **Adam** method (Kingma and Ba, 2014).

Backpropagation (Bryson, 1975; Werbos, 1974; Rumelhart et al., 1985) is the algorithm used to evaluate the gradient of a loss function $\boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\theta})$, or a part of it ,$\boldsymbol{\theta} \mapsto \mathcal{L}_i(\boldsymbol{\theta})$, with respect to the parameters of each layer of the neural network. At its core, it relies on repeated applications of the chain rule of calculus, which states that for functions $f : \mathbb{R}^m \to \mathbb{R}^k$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ and a point $\boldsymbol{x} \in \mathbb{R}^n$, the Jacobian matrix of $f \circ g$ at $\boldsymbol{x}$, $\boldsymbol{J}_{f \circ g}(\boldsymbol{x})$ satisfies:

$$\underbrace{\boldsymbol{J}_{f \circ g}(\boldsymbol{x})}_{\in \mathbb{R}^{k \times n}} = \underbrace{\boldsymbol{J}_f(g(\boldsymbol{x}))}_{\in \mathbb{R}^{k \times m}} \underbrace{\boldsymbol{J}_g(\boldsymbol{x})}_{\in \mathbb{R}^{m \times n}}. \tag{2.64}$$

The deep learning revolution hinges on open-source software libraries such as Tensorflow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019), which support **automatic differentiation**, i.e., the evaluation of gradients of a differentiable loss with respect to the parameters of a neural network automatically using the backpropagation algorithm.

2.1.7. Uncertainty Estimation

This section uses content from Lahlou et al. (2021), which is at the heart of Chapter 5.

A remaining challenge in machine learning research is purposeful knowledge-seeking by learning agents, which can benefit from the estimation of **epistemic uncertainty**, a measure of lack of knowledge that an active learner should minimize. Unlike **aleatoric uncertainty**, which refers to an intrinsic notion of randomness and is irreducible by nature, epistemic uncertainty can potentially be reduced with additional information (Kiureghian and Ditlevsen, 2009; Hüllermeier and Waegeman, 2019).

In machine learning, epistemic uncertainty estimation is already a key ingredient in interactive decision-making settings such as:

— **Active learning** (Settles, 1994; Aggarwal et al., 2014; Gal et al., 2017), where the goal is to select which training examples the agent should train on in order to reach the best performances the fastest,

— **Sequential model optimization** (Frazier, 2018; Garnett, 2022), where the goal is to optimize an objective function $f : \mathcal{X} \to \mathbb{R}$ that is expensive to evaluate,

— **Causal learning** (He and Geng, 2008; Agrawal et al., 2019), which can be seen as a form of active learning, where the learner needs to select which variables to intervene on in order to discern the ground truth DAG from other DAGs in its Markov equivalence class (Definition A.1.2),

— **Exploration in RL** (Kocsis and Szepesvári, 2006; Tang et al., 2017), which is of paramount importance in *sparse* reward environments,

given that epistemic uncertainty estimators can inform the learner about the potential information gain from collecting data around a particular area of state-space or input data space.

In the subjective interpretation of probability (De Morgan, 1847; Ramsey, 1926), probabilities are representations of an agent's *subjective* degree of confidence (Hájek, 2019). It is thus unsurprising that probabilities have been widely used to represent uncertainty. In supervised learning, Bayesian methods (Section 2.1.3.5), which aim at predicting the posterior predictive (2.26), are natural candidates for providing uncertainty estimates that account for both the aleatoric and epistemic uncertainties. As previously mentioned, however, Bayesian

inference is usually intractable. We will discuss in Section 2.2 and Chapters 3 and 4 approximate inference techniques, and in Chapter 5, an alternative way of evaluating epistemic uncertainty that does not rely on the Bayesian posterior.

2.2. Approximate Bayesian inference

In Bayesian inference, we are interested in modelling the posterior over the parameters $\boldsymbol{\theta} \in \Theta$ of a probabilistic model (e.g., $p(y \mid \boldsymbol{x}, \boldsymbol{\theta})$ in supervised learning), upon observing a dataset $\mathcal{D}$:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}. \tag{2.65}$$

For most probabilistic models, only *approximate inference* is possible, given the intractability of $p(\mathcal{D})$ in (2.65). We have seen in Section 2.1.3.5 that a Dirac distribution at the MAP estimate is itself an approximation to the posterior.

Another simple approximation can be obtained by discretizing Θ into a finite set of regions $\Theta_1, \ldots, \Theta_K$, each of which contains a point $\boldsymbol{\theta}_k$. Instead of evaluating the integral in (2.25), it suffices to evaluate $p(\mathcal{D} \mid \boldsymbol{\theta}_k)$ for $k \in [\![1, K]\!]$, leading to the **grid approximation** of the posterior:

$$\tilde{p}(\boldsymbol{\theta} \in \Theta_k) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k)}{\sum_{k'=1}^{K} p(\mathcal{D} \mid \boldsymbol{\theta}_{k'})p(\boldsymbol{\theta}_{k'})}. \tag{2.66}$$

This approach, however, suffers from the curse of dimensionality and is usually only applicable when $\boldsymbol{\theta}$ is of dimension 1 or 2.

In the remainder of this section, we will discuss alternative ways of approximating the Bayesian posterior that scale to higher dimensions.

2.2.1. Variational inference

In **variational inference (VI)** (Jordan et al., 1999; Jaakkola and Jordan, 2000; Jaakkola, 2000; Wainwright et al., 2008; Blei et al., 2017; Zhang et al., 2018), the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ is approximated with a tractable distribution $q(\boldsymbol{\theta})$ by minimizing a discrepancy D (usually the Kullback-Leibler divergence D_{KL}) between the two distributions, over a tractable family of distributions:

$$q^* = \arg\min_{q \in \mathcal{Q}} D_{\mathrm{KL}}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta} \mid \mathcal{D})). \tag{2.67}$$

When applied to the problem of modelling the posterior, variational inference is also referred to as **Variational Bayes** (Tran et al., 2021).

It is common to parameterize the distributions in $\mathcal{Q}$ using **variational parameters** $\boldsymbol{\psi} \in \Psi$, in which case the problem is called **fixed-form** variational inference (Salimans and

Knowles, 2013), and the inference problem reduces to the following optimization problem:

$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}\in\Psi} D_{\mathrm{KL}}(q_{\boldsymbol{\psi}}(\boldsymbol{\theta})\|p(\boldsymbol{\theta} \mid \mathcal{D})) \tag{2.68}$$

$$= \arg\min_{\boldsymbol{\psi}\in\Psi} \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\left[\log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}) - \log\left(\frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}\right)\right] \tag{2.69}$$

$$= \arg\min_{\boldsymbol{\psi}\in\Psi} -L(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) + \log p(\mathcal{D}), \tag{2.70}$$

where

$$L(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\left[\underbrace{\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}_{l_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\right], \tag{2.71}$$

is the **evidence lower bound (ELBO)**, given that it is a lower bound on the log-evidence $\log p(\mathcal{D})$:

$$L(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) \leq \log p(\mathcal{D}), \tag{2.72}$$

given that the KL divergence is non-negative.

The **variational free energy** is defined as $\mathcal{L}(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) = -L(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D})$. The name is due to:

$$\mathcal{L}(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}[\mathcal{E}(\boldsymbol{\theta})] - \mathbb{H}(q_{\boldsymbol{\psi}}(\boldsymbol{\theta})), \tag{2.73}$$

where $\mathcal{E}(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}, \mathcal{D})$ is the **energy** and $\mathbb{H}(q_{\boldsymbol{\psi}}(\boldsymbol{\theta}))$ is the **entropy** of q. There are two interpretations of the ELBO:

$$L(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\left[\log p(\boldsymbol{\theta}, \mathcal{D})\right]}_{\text{expected log joint}} + \underbrace{\mathbb{H}(q_{\boldsymbol{\psi}}(\boldsymbol{\theta}))}_{\text{entropy of posterior}} \tag{2.74}$$

$$L(\boldsymbol{\psi} \mid \boldsymbol{\theta}, \mathcal{D}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\left[\log p(\mathcal{D} \mid \boldsymbol{\theta})\right]}_{\text{expected log likelihood}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\psi}}(\boldsymbol{\theta})\|p(\boldsymbol{\theta}))}_{\text{KL from posterior to prior}} \tag{2.75}$$

Black-box variational inference (BBVI) is a fixed form VI algorithm that assumes we can evaluate $l_{\boldsymbol{\psi}}(\boldsymbol{\theta})$ (2.71) point-wise, but that we cannot obtain the gradients of $l_{\boldsymbol{\psi}}$ (the model is thus called **black-box**). The gradient of the ELBO uses the score function estimator, also called the **REINFORCE** estimator (Williams, 1992):

$$\nabla_{\boldsymbol{\psi}} L(\boldsymbol{\psi} \mid \mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}[l_{\boldsymbol{\psi}}(\boldsymbol{\theta})\nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta})], \tag{2.76}$$

which can be estimated using Monte-Carlo samples Section 2.2.2.

If the evaluation of $l_{\boldsymbol{\psi}}$ is intractable because the dataset size N is too large, an unbiased minibatch approximation $\hat{l}_{\boldsymbol{\psi}}$ can be used instead. This is called **stochastic VI**.

An alternative to BBVI is the **Reparametrized VI**, where we assume $l_{\boldsymbol{\psi}}(\boldsymbol{\theta})$ is a differentiable function of $\boldsymbol{\theta}$, and sampling $\boldsymbol{\theta} \sim q_{\boldsymbol{\psi}}(\boldsymbol{\theta})$ is equivalent to sampling $\epsilon \sim q_0(\epsilon)$ and

applying a deterministic mapping $\boldsymbol{z} = r(\boldsymbol{\psi}, \epsilon)$. In such settings, the gradient of the ELBO can be evaluated by pushing the gradient operator inside the expectation (over ϵ). This is common in training VAEs (Kingma and Welling, 2014a; Rezende et al., 2014b), introduced in Section 2.1.4.

Automatic Differentiation VI, or **ADVI**, takes away the burden of specifying the variational family $q_{\boldsymbol{\psi}}$, by considering inherently expressive variational families, such as normalizing flows posteriors Section 2.1.4, or **hierarchical posteriors**, and directly estimating the gradients with respect to the underlying parameters. Hierarchical posteriors use auxiliary latent variables $\boldsymbol{a}$ such that $q_{\boldsymbol{\psi}}(\boldsymbol{\theta} \mid \mathcal{D}, \boldsymbol{a})$ is conditionally factorized, but when we marginalize out $\boldsymbol{a}$, we induce dependencies between the elements of $\boldsymbol{\theta}$. This is called **hierarchical variational inference**, and will be expanded upon in Chapter 3.

2.2.2. Sampling as a stochastic approximation

Although variational inference techniques do not suffer from the curse of dimensionality, a specific form of the variational family $\mathcal{Q}$ might lead to a biased posterior approximation. An alternative is to resort to a non-parametric approximation of the form:

$$q(\boldsymbol{\theta}) = \frac{1}{K} \sum_{k=1}^{K} \delta_{\boldsymbol{\theta}_k}, \tag{2.77}$$

where $\boldsymbol{\theta}_k \sim p(\boldsymbol{\theta} \mid \mathcal{D})$ for $k \in [\![1, K]\!]$. This is called a **Monte Carlo approximation**.

With such an approximation, it is possible to approximate the posterior predictive (2.26) as:

$$p(y \mid \boldsymbol{x}, \mathcal{D}) \approx \frac{1}{K} \sum_{k=1}^{K} p(y \mid \boldsymbol{x}, \boldsymbol{\theta}_k), \tag{2.78}$$

and still obtain uncertainty estimates.

The key issue is how to create samples from the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ without having to evaluate the evidence $p(\mathcal{D})$ [13]. The remainder of this section will present some of the standard techniques.

There are other reasons, beyond Bayesian posterior approximation, one might care about sampling from a probability distribution

$$p(\boldsymbol{\theta}) = \frac{R(\boldsymbol{\theta})}{Z} \tag{2.79}$$

with unknown partition $Z = \int R(\boldsymbol{\theta}) d\boldsymbol{\theta}$. For instance, we might want to sample objects $\boldsymbol{\theta}$ from a large search space Θ proportionally to a given positive score or reward $R(\boldsymbol{\theta})$. This is in contrast to the RL problem, which cares about finding a reward function's mode without

13. If $\boldsymbol{\theta}$ is a high-dimensional object, then even the knowledge of $p(\mathcal{D})$ does not entail straightforward sampling schemes.

incentivizing the generated modes' diversity. This prompted research on generative flow networks, which we will delve into in Chapter 3.

Rejection sampling (Neumann, 1951; Good, 1960) is a simple method for sampling from unnormalized distributions (2.79). It requires a proposal distribution $q(\boldsymbol{\theta})$ that satisfies $\forall \boldsymbol{\theta} \in \Theta,\ Cq(\boldsymbol{\theta}) \geq R(\boldsymbol{\theta})$ for some constant C, from which it is easy to sample and evaluate the probability mass or density $q(\boldsymbol{\theta})$. The difficulty lies in finding such a distribution q, with a small enough constant C. Adaptive rejection sampling (Gilks and Wild, 1992) adaptively builds such a distribution. A fundamental weakness of such methods, however, is the curse of dimensionality, making them useless for high-dimensional target distributions, such as learning the posterior over the parameters of a neural network.

Annealed importance sampling (Neal, 2001a) is an alternative sampling scheme for high dimensional distributions. It requires building a sequence of distributions $p_j(\boldsymbol{\theta})$ for $j \in [\![0, n]\!]$, such that hence $p_0 = p$, and $p_n = \frac{R_n}{Z_n}$ is easy to sample from, and such that, for every $j \in [\![0, n]\!]$, $p_j \propto R_j$ where:

$$R_j(\boldsymbol{\theta}) = R(\boldsymbol{\theta})^{\beta_j} R_n(\boldsymbol{\theta})^{1-\beta_j}, \tag{2.80}$$

where $0 = \beta_n < \cdots < \beta_0 = 1$. The technique relies on sampling iteratively from $p_n, p_{n-1}, \ldots$ until we eventually sample from $p_0 = p$. Sampling from each f_j requires the specification of a Markov chain kernel $p_j(\boldsymbol{x}' \mid \boldsymbol{x})$ that leaves p_0 invariant, i.e.,

$$\int p_j(\boldsymbol{x}' \mid \boldsymbol{x}) p_0(\boldsymbol{x}) d\boldsymbol{x} = p_0(\boldsymbol{x}'). \tag{2.81}$$

Markov Chain Monte Carlo (MCMC, Metropolis et al., 1953; Hastings, 1970; Geman and Geman, 1984; Gelfand and Smith, 1990) is a popular method for sampling from high-dimensional distributions using Markov chains. It hinges on building a Markov chain on the sample space Θ whose stationary distribution is the target distribution $p(\boldsymbol{\theta})$. The simplest MCMC algorithm is the **Metropolis Hastings (MH)** algorithm. It relies on a kernel $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$ and builds a chain by moving from the current state $\boldsymbol{\theta}$ to a new state $\boldsymbol{\theta}'$ with probability $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$. The new state $\boldsymbol{\theta}'$ is accepted with probability

$$A = \min\left(1, \frac{p(\boldsymbol{\theta}')q(\boldsymbol{\theta} \mid \boldsymbol{\theta}')}{p(\boldsymbol{\theta})q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})}\right), \tag{2.82}$$

and if rejected, $\boldsymbol{\theta}'$ is set to $\boldsymbol{\theta}$. Note that the ratio in (2.82) only requires the knowledge of R. As the chain grows, the samples $\boldsymbol{\theta}$ approach those that would come from the target distribution p. The initial samples, however, should be discarded. The number of time steps required to reach the target distribution is called the **mixing time** or the **burn-in time**, and ideally, one would choose the kernel q that leads to the lowest possible mixing time. MCMC-based methods have already found some amount of success with learned deep neural networks used to drive sampling (Grathwohl et al., 2021; Dai et al., 2020; Xie et al., 2021;

Nash and Durkan, 2019; Seff et al., 2019). More details about MCMC, and some extensions are provided in Barbu et al. (2020); Lao et al. (2020); Andrieu and Thoms (2008); Neal (2003); Neal et al. (2011); Betancourt (2017); Welling and Teh (2011); Zhang et al. (2020); Vadera et al. (2020a). Some drawbacks of MCMC methods are discussed in Section 3.5.

2.2.3. Dropout and Deep Ensembles

Deep neural networks Section 2.1.6 can represent many functions that fit the training data well but generalize differently. This phenomenon is usually called the **underspecification** of neural networks (D'Amour et al., 2022). Bayesian deep learning, which revolves around learning the posterior over the parameters $\boldsymbol{\theta}$ of a neural network, tackles the underspecification problem by considering all possible models together. However, Bayesian deep learning is naturally intractable, and besides the approximate Bayesian inference techniques presented thus far, there are approximate inference schemes specific to neural networks. We will discuss two of them in the remainder of this section.

Monte Carlo dropout (MC-Dropout) (Gal and Ghahramani, 2016a; Kendall and Gal, 2017) is a simple method for approximating the posterior predictive (2.26). It hinges on adding dropout layers to both the training procedure, but also at inference time (i.e., when doing forward passes in the network): each hidden unit in the neural network is "turned off" by sampling from a Bernoulli distribution (Appendix A.1.1) with probability r. By repeating the procedure K times, we obtain K models with weights $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K$, in which the proportion of zeros is close to r. The posterior predictive is thus approximated with:

$$\tilde{p}(y \mid \boldsymbol{x}, \mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} p(y \mid \boldsymbol{x}, \boldsymbol{\theta}_k). \tag{2.83}$$

Another simple approximation is to only be Bayesian about the weights in the final layer and use MAP estimates for other layers. This technique is called the **neural-linear** approximation (Riquelme et al., 2018) and has been shown to provide reasonable uncertainty estimates in Kristiadi et al. (2020).

Deep ensembles is an approximation scheme that trains multiple models, differing in terms of their random seed used for initialization (Lakshminarayanan et al., 2017a), or hyperparameters (Wenzel et al., 2020), or architecture (Zaidi et al., 2021), or by using different subsets of $\mathcal{D}$ to train each member of the ensemble, leading to MLE or MAP parameters $\boldsymbol{\theta}_k$ for $k \in [\![1, K]\!]$. The posterior is thus approximated with the following:

$$\tilde{p}(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} \delta_{\boldsymbol{\theta}_k}. \tag{2.84}$$

Chapter 3

On generative flow networks

This chapter is based on the two following papers:

— Bengio et al. (2023): "GFlowNet Foundations" - Yoshua Bengio*, Salem Lahlou*, Tristan Deleu*, Edward J. Hu, Mo Tiwari, Emmanuel Bengio, published in 2023 in the Journal of Machine Learning Research (JMLR).

— Malkin et al. (2023): "GFlowNets and variational inference" - Nikolay Malkin*, Salem Lahlou*, Tristan Deleu*, Xu Ji, Edward J Hu, Katie E Everett, Dinghuai Zhang, Yoshua Bengio, published in 2023 in the proceedings of the International Conference on Learning Representations (ICLR).

Sections 3.1 to 3.4 are shortened versions of Sections 1 to 4 of Bengio et al. (2023), with a focus on the contributions of the author of this thesis to the paper, to avoid repetitions of the content discussed in Chapter 2 and to improve the reading *flow*. Section 3.2.7, however is novel and unpublished. Section 3.4.4, on the other hand, is loosely based on Appendix B of the preprint version of Bengio et al. (2023). Section 3.5, which discusses related work, merges content from Sections 3.1, 4.6 and 7 of Bengio et al. (2023). Section 3.6 is based on Sections 1 and 3 of Malkin et al. (2023). Section 3.7 merges content from Section 3 and Appendices A and C of Malkin et al. (2023). Finally, Section 3.8 is a mix of Section 4 and Appendix D of Malkin et al. (2023).

3.1. Introduction

Building upon the introduction of Generative Flow Networks (GFlowNets) by Bengio et al. (2021), we provide here an in-depth formal foundation and expansion of the set of theoretical results in ways that may be of interest for the active learning scenario of Bengio et al. (2021) but also much more broadly. We also relate generative flow networks to hierarchical variational inference and highlight some critical differences that make GFlowNets more appealing in many scenarios.

GFlowNets tackle the problem of sampling from a space $\mathcal{X}$, given an energy function $\mathcal{E}$, or an unnormalized probability mass function, also called the reward function R, such that the objects $\mathcal{X}$ can be constructed sequentially by combining multiple primitive blocks, that share some structure. Examples of such objects are graphs, whose primitive blocks are the nodes and edges. A sequential graph creation process step is depicted in Figure 3.1. Other examples include molecules, DNA sequences, and arbitrary sets of objects.

The key property of GFlowNets is that their sampling policy is trained to make the probability $P_\top(x)$ of sampling an object $x \in \mathcal{X}$ approximately proportional to the value $R(x)$ of the given reward function applied to that object.

This conversion of an energy function or unnormalized probability function to a sampler is similar to what MCMC methods (Section 2.2.2) achieve. Still, once trained, GFlowNets will generate a sample in one shot instead of generating a long sequence of samples whose distribution would gradually approach the desired one. GFlowNets thus avoid the lengthy stochastic search in the space of such objects and the associated mode-mixing intractability challenge of MCMC methods (Jasra et al., 2005; Bengio et al., 2013; Pompe et al., 2020). Multiple independent and identically distributed samples can be obtained from the GFlowNet by calling the sampler several times. GFlowNets exchange that intractability of sampling with MCMC for the challenge of *amortized training* of the generative policy, which will be expanded upon in Section 3.5. The latter problem would be equally intractable if the modes of the reward function did not have an inherent (but not necessarily known) structure over which the learner could generalize, i.e., the learner had almost no chance to correctly guess where to find new modes based on (i.e., training on) those it had already visited.

The energy function or reward function (exponential of minus energy) is evaluated only at the end of the sequential construction process for objects x, in what we call a terminating state. Every such constructive sequence starts in the single initial state s_0 and ends in a terminating state. As illustrated in Figure 3.2, we can visualize the set of all trajectories starting from s_0 and ending in a terminal state s. The term "flow" in "generative flow network" refers to unnormalized probabilities that GFlowNet learning procedures can learn. The flow in an intermediate state s is a weighted sum of the non-negative rewards of the terminating states reachable from s. Those weights are such as to avoid double-counting: if we were to inject a fixed flow of liquid in s_0 and dispatch that liquid in each child of any state s proportionally to the GFlowNet policy for choosing a child of s, we would obtain the flow at each state, and the flow at terminating states would match the reward function at those states. As shown in greater detail here and for the first time in the first GFlowNet paper (Bengio et al., 2021), this can be achieved with a flow constraint at each state: the sum of incoming flows must match the sum of outgoing flows. This constraint will be explained and justified later on in this chapter.

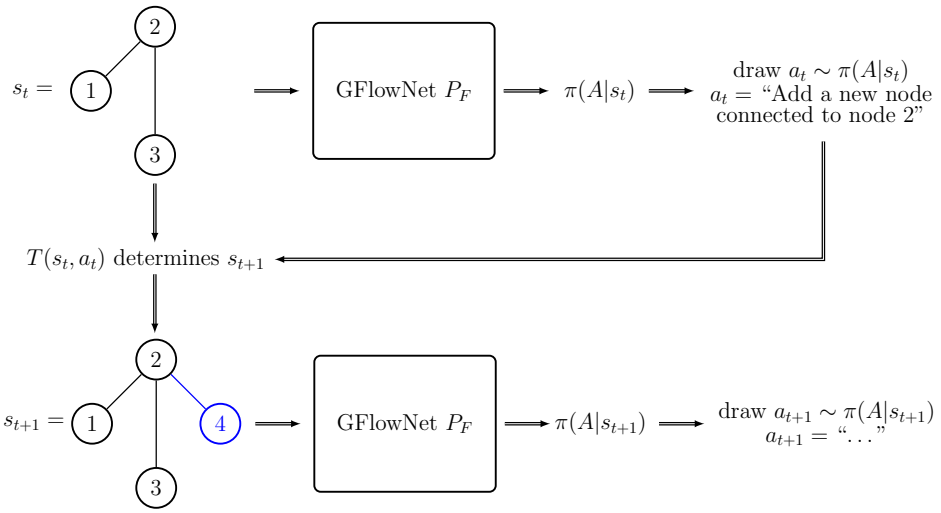


Figure 3.1 – A diagram of how a GFlowNet iteratively constructs an object. We adopt a notation that is common in the reinforcement learning literature: s_t represents the state of the partially constructed object (in this case, a graph) at time t, a_t represents the action taken by the GFlowNet at time t to transition to state $s_{t+1} = T(s_t, a_t)$. In this diagram, the GFlowNet takes a 3-node graph as input and determines an action. The action, combined with the environment transition function $T(s_t, a_t)$, determines s_{t+1}: a four-node graph. This process repeats until an exit action is sampled and the sample is complete.

Amortized probabilistic inference is an interesting application of GFlowNets: if the reward function is set to be a prior over some random variable $p(\boldsymbol{\theta})$, multiplied by a likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$ that tells how well the dataset $\mathcal{D}$ is fit given $\boldsymbol{\theta}$, then the GFlowNet policy learns to sample from the corresponding Bayesian posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ [1].

The remainder of this chapter is structured into the following sections:

— In Section 3.2, we start with some essential elements of graph theory, from which we define the notions of *flows* and *flow networks*. A particular class of flows called *Markovian flows* is highlighted. We derive the conditions a network needs to satisfy to become a flow network, from which a sampler can be defined. These conditions correspond to the flow-matching conditions, initially described in Bengio et al. (2021), but also to the novel set of constraints called the detailed balance conditions. We provide a dynamic programming-based algorithm to find the edge flows given a reward function.

— In Section 3.3, we formalize GFlowNets as mathematical objects and relate them to flow networks. The constraints presented in Section 3.2 are turned into training losses that make the GFlowNets correct samplers if minimized.

— In Section 3.4, we extend flow networks and GFlowNets to conditional ones, which enable the estimation of intractable sums corresponding to marginalization over many

1. This is a direct consequence of Bayes' rule (2.1).

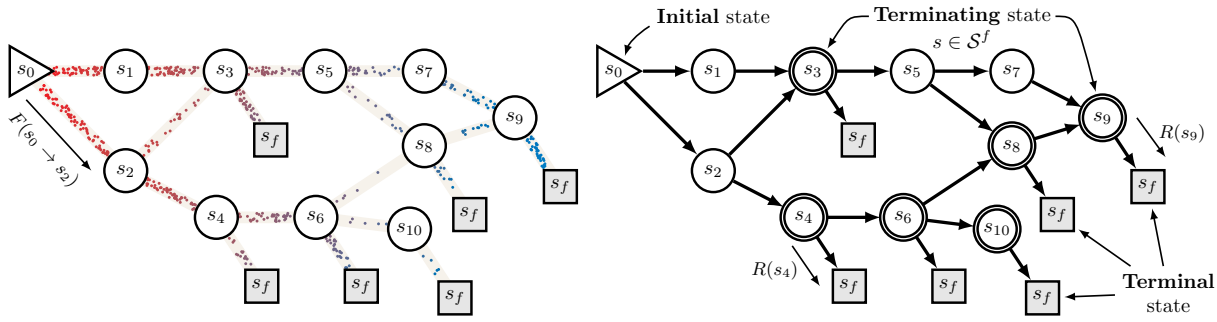


Figure 3.2 – Illustration of the structure of a Generative Flow Network (GFlowNet), as a pointed DAG (defined more formally in Section 3.2.1) over states s, with particles flowing along edges to represent the flow function. Any object sampled by the GFlowNet policy can be obtained by starting from the initial state s_0 and then, at each step, choosing a child with probability proportional to the GFlowNet policy's transition probability. This process stops when a terminating action is chosen from a terminating state x (yielding the terminal state s_f, also called the sink state, and also denoted $\perp$), at which point a reward $R(x)$ is obtained. The figure shows a tiny GFlowNet and the possible trajectories from s_0 to any terminal state. It illustrates that, in general, a state can be reached through several trajectories. GFlowNet algorithms learn a policy such that the probability of sampling terminating state x is proportional to $R(x)$. It can e.g., learn a flow function $s \mapsto F(s)$ and $(s \to s') \mapsto F(s \to s')$ over all states (including intermediate states) s and transitions $s \to s'$, with $F(x \to s_f) = R(x)$ at terminating states, and $F(s_0)$ being the sum of rewards over all terminal states. A sufficient property to achieve this is that the sum of incoming flows at each state equals the sum of outgoing flows (Section 3.2.5).

steps of object construction. Conditional GFlowNets can also be used to compute free energies over different types of joint distributions.

— We discuss related work in Section 3.5, including MCMC, generative models, and interactive learning.

— In Section 3.6, we discuss how hierarchical variational methods can efficiently define samplers in compositional spaces and contrast them to GFlowNets.

— In Section 3.7, we delve into the theoretical connection between hierarchical variational models and GFlowNets, highlighting some key differences and advantages of GFlowNets.

— In Section 3.8, we show some experiments that empirically demonstrate that in many interesting settings, GFlowNets lead to more accurate approximate samplers than their variational counterpart.

3.2. Flow Networks and Markovian Flows

All lemmas, propositions and corollaries of this section are proved in Appendix D.2.

3.2.1. Some elements of graph theory

In this section, we recall some basic definitions and properties of graphs, which are the basis of flow networks and GFlowNets.

Definition 3.2.1 (Directed acyclic graphs and Trajectories)**.** A directed graph is a tuple $G = (\mathcal{S}, \mathbb{A})$, where $\mathcal{S}$ is a finite set of states, and $\mathbb{A}$ is a subset of $\mathcal{S}\times\mathcal{S}$ representing directed edges. Elements of $\mathbb{A}$ are denoted $s\to s'$ and called **edges** or **transitions**.
A **trajectory** in such a graph is a sequence $\tau = (s_1, \dots, s_n)$ of elements of $\mathcal{S}$ such that every transition $s_t\to s_{t+1} \in \mathbb{A}$ and $n > 1$.
A **directed acyclic graph** (DAG) is a directed graph in which there is no trajectory $\tau = (s_1, \dots, s_n)$ satisfying $s_n = s_1$.

For a trajectory τ, we denote $s \in \tau$ to mean that s is in the trajectory τ, i.e., $\exists t \in \{1, \dots, n\}\ s_t = s$, and similarly $s\to s' \in \tau$ to mean that $\exists t \in \{1, \dots, n-1\}\ s_t = s, s_{t+1} = s'$. We also use the notation $\tau = s_1 \to \cdots \to s_n$ for convenience. The **length** of a trajectory is the number of edges in it (the length of $\tau = (s_1, \dots, s_n)$ is thus $n - 1$).

Given a DAG $G = (\mathcal{S}, \mathbb{A})$, and two states $s, s' \in \mathcal{S}$, if there exists a trajectory in G starting in s and ending in s', then we write $s < s'$. The binary relationship "$<$" defines a **strict partial order** (i.e., it is irreflexive, asymmetric and transitive). We write $s \leq s'$ if $s < s'$ or $s = s'$. The binary relation "$\leq$" is a (non-strict) **partial order** (i.e., it is reflexive, antisymmetric and transitive).

Definition 3.2.2 (Parent and child sets)**.** Given a DAG $G = (\mathcal{S}, \mathbb{A})$, the **parent set** of a state $s \in \mathcal{S}$, which we denote $Par(s)$, contains all of the direct parents of s in G, i.e., $Par(s) = \{s' \in \mathcal{S}\ :\ s'\to s \in \mathbb{A}\}$; similarly, the **child set** $Child(s)$ contains all of the direct children of s in G, i.e., $Child(s) = \{s' \in \mathcal{S}\ :\ s\to s' \in \mathbb{A}\}$.

Definition 3.2.3 (Pointed DAGs)**.** Given a DAG $G = (\mathcal{S}, \mathbb{A})$. G is called a **pointed DAG** if there exist two states $s_0, s_f \in \mathcal{S}$ that satisfy:

$$\forall s \in \mathcal{S} \setminus \{s_0\}\ \ s_0 < s \text{ and } \forall s \in \mathcal{S} \setminus \{s_f\}\ \ s < s_f. \tag{3.1}$$

s_0 is called the **source state** or **initial state**. s_f is called the **sink state** or **final state**. These two states are unique because "$<$" is a strict partial order.
A **complete trajectory** in such a DAG is any trajectory starting in s_0 and ending in s_f. We denote such a trajectory as $\tau = (s_0, s_1, \dots, s_n, s_{n+1} = s_f)$ or $\tau = s_0 \to s_1 \dots \to s_n \to s_{n+1} = s_f$.

We denote by $\mathcal{T}$ the set of all complete trajectories in G, and by $\mathcal{T}^{partial}$ the set of (possibly incomplete) trajectories in G. For any state $s \in \mathcal{S} \setminus \{s_f\}$, we denote by $\mathcal{T}_{s,f} \subseteq \mathcal{T}^{partial}$ the set of trajectories in G starting in s and ending in s_f; and for any state $s \in \mathcal{S} \setminus \{s_0\}$, we denote by $\mathcal{T}_{0,s} \subseteq \mathcal{T}^{partial}$ the set of trajectories in G starting in s_0 and ending in s.

A state $s \in \mathcal{S}$ is called a **terminating state** if it is a parent of the sink state, i.e., $s \to s_f \in \mathbb{A}$. The transition $s \to s_f$ is called a **terminating edge**. We denote by:

— $\mathbb{A}^{-f} = \{s \to s' \in \mathbb{A},\ s' \neq s_f\}$, the set of non-terminating edges in G,

— $\mathbb{A}^{f} = \{s \to s' \in \mathbb{A},\ s' = s_f\} = \mathbb{A} \setminus \mathbb{A}^{-f}$, the set of terminating edges in G,

— $\mathcal{S}^f = \{s \in \mathcal{S},\ s \to s_f \in \mathbb{A}^f\} = Par(s_f)$, the set of terminating states in G.

In Figure 3.3, we visualize the concepts introduced in the previous definitions.

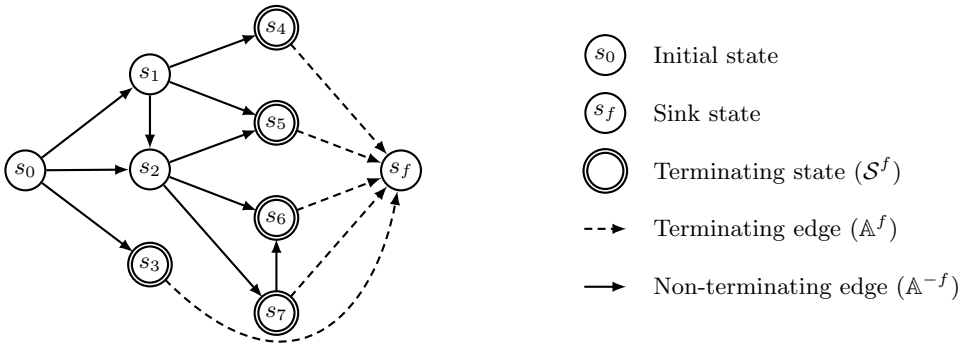


Figure 3.3 – Example of a pointed DAG G illustrating the notions of initial state (s_0), final or sink state (s_f), terminating states in $\mathcal{S}^f$, with a transition to s_f called a terminating edge, in $\mathbb{A}^f$. A terminating state may have other children different from the sink state (e.g., the terminating state s_7).

Note that the constraint of a single source state and single sink state is only a mathematical convenience since a bijection exists between general DAGs and those with this constraint (by adding a unique source/sink state connected to all the other source/sink states).

Definition 3.2.4 (Probability functions consistent with a DAG)**.** Let G be a pointed DAG with source state s_0 and sink state s_f. A **forward** (resp. **backward**) **probability function consistent with** G is any non-negative function $\hat{P}_F$ (resp. $\hat{P}_B$) defined on $\mathbb{A}$ that satisfies $\forall s \in \mathcal{S} \setminus \{s_f\},\ \sum_{s' \in Child(s)} \hat{P}_F(s' \mid s) = 1$ (resp. $\forall s \in \mathcal{S} \setminus \{s_0\}, \sum_{s' \in Par(s)} \hat{P}_B(s' \mid s) = 1$).

With pointed DAGs, consistent forward and backward probability functions, which are probabilities over states, can be used to define probabilities over trajectories, i.e., probability measures on some subsets of $\mathcal{T}^{partial}$. The following lemma shows how to construct such factorized probability measures:

Lemma 3.2.5 (Probabilities over partial trajectories)**.** *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG, and consider a forward probability function $\hat{P}_F$, and a backward probability function $\hat{P}_B$ both consistent with G.*
Consider the extensions of $\hat{P}_F$ and $\hat{P}_B$ on $\mathcal{T}^{partial}$ defined by:

$$\forall \tau(s_1, \ldots, s_n) \in \mathcal{T}^{partial} \quad \hat{P}_F(\tau) := \prod_{t=1}^{n-1} \hat{P}_F(s_{t+1} \mid s_t) \tag{3.2}$$

$$\forall \tau = (s_1, \ldots, s_n) \in \mathcal{T}^{partial} \quad \hat{P}_B(\tau) := \prod_{t=1}^{n-1} \hat{P}_B(s_t \mid s_{t+1}) \tag{3.3}$$

We have the following:

$$\forall s \in \mathcal{S} \setminus \{s_f\} \quad \sum_{\tau \in \mathcal{T}_{s,f}} \hat{P}_F(\tau) = 1 \tag{3.4}$$

$$\forall s' \in \mathcal{S} \setminus \{s_0\} \quad \sum_{\tau \in \mathcal{T}_{0,s'}} \hat{P}_B(\tau) = 1 \tag{3.5}$$

3.2.2. Trajectories and Flows

We augment pointed DAGs with a function F called a *flow*. An analogy which helps to picture flows is a stream of particles flowing through a network where each particle starts at s_0 and flows through some trajectory terminating in s_f. The flow $F(\tau)$ associated with each complete trajectory τ contains the number of particles sharing the same path τ.

Definition 3.2.6 (Flows and flow networks)**.** Given a pointed DAG, a **trajectory flow** (or "**flow**") is any non-negative function $F : \mathcal{T} \mapsto \mathbb{R}^+$ defined on the set of *complete trajectories* $\mathcal{T}$. F induces a *measure* over the σ-algebra $\Sigma = 2^{\mathcal{T}}$, the power set on the set of complete trajectories $\mathcal{T}$. In particular, for every subset $A \subseteq \mathcal{T}$, we have

$$F(A) = \sum_{\tau \in A} F(\tau). \tag{3.6}$$

The pair (G, F) is called a **flow network**.

This definition ensures that $(\mathcal{T}, 2^{\mathcal{T}}, F)$ is a measure space. We abuse the notation here, using F to denote both a function of complete trajectories, and its corresponding measure over $(\mathcal{T}, 2^{\mathcal{T}})$. A special case is when the event A is the singleton trajectory $\{\tau\}$, where we just write its measure as $F(\tau)$. We also abuse the notation to define the flow through either a particular state s or through a particular edge $s \to s'$ in the following way.

Definition 3.2.7 (State flows and edge flows)**.** The **flow through a state** (or state flow) $F : \mathcal{S} \mapsto \mathbb{R}^+$ corresponds to the measure of the set of complete trajectories going through a particular state:

$$F(s) := F(\{\tau \in \mathcal{T} \,:\, s \in \tau\}) = \sum_{\tau \in \mathcal{T} \,:\, s \in \tau} F(\tau). \tag{3.7}$$

Similarly, the **flow through an edge** (or edge flow) $F : \mathbb{A} \mapsto \mathbb{R}^+$ corresponds to the measure of the set of complete trajectories going through a particular edge:

$$F(s \to s') := F(\{\tau \in \mathcal{T} \,:\, s \to s' \in \tau\}) = \sum_{\tau \in \mathcal{T} \,:\, s \to s' \in \tau} F(\tau). \tag{3.8}$$

Note that with this definition, we have $F(s \to s') = 0$ if $s \to s' \notin \mathbb{A}$ is not an edge in the pointed DAG (since $F(\emptyset) = 0$). We call the flow of a terminating transition $F(s \to s_f)$ a *terminating flow*. The following proposition relates the state flows and the edge flows:

Proposition 3.2.8. *Given a flow network (G, F). The state flows and edge flows satisfy:*

$$\forall s \in \mathcal{S} \setminus \{s_f\} \quad F(s) = \sum_{s' \in Child(s)} F(s \to s') \tag{3.9}$$

$$\forall s' \in \mathcal{S} \setminus \{s_0\} \quad F(s') = \sum_{s \in Par(s')} F(s \to s') \tag{3.10}$$

3.2.3. Flow Induced Probability Measures

Definition 3.2.9 (Total flow)**.** Given a flow network (G, F), the **total flow** Z is the measure of the whole set $\mathcal{T}$, corresponding to the sum of the flows of all the complete trajectories:

$$Z := F(\mathcal{T}) = \sum_{\tau \in \mathcal{T}} F(\tau). \tag{3.11}$$

Proposition 3.2.10. *The flow through the initial state equals the flow through the final state equals the total flow Z.*

Intuitively, Proposition 3.2.10 justifies the use of the term "flow", introduced by Bengio et al. (2021), by analogy with a stream of particles flowing from the initial state to the final states.

We use the letter Z in Definition 3.2.9, often used to denote the partition function in probabilistic models and statistical mechanics, because it is a normalizing constant which can turn the measure space $(\mathcal{T}, 2^{\mathcal{T}}, F)$ defined above into the probability space $(\mathcal{T}, 2^{\mathcal{T}}, P)$:

Definition 3.2.11 (Flow probability)**.** Given a flow network (G, F), the **flow probability** is the probability measure P over the measurable space $(\mathcal{T}, 2^{\mathcal{T}})$ associated with F:

$$\forall A \subseteq \mathcal{T} \quad P(A) := \frac{F(A)}{F(\mathcal{T})} = \frac{F(A)}{Z}. \tag{3.12}$$

For two events $A, B \subseteq \mathcal{T}$, the conditional probability $P(A \mid B)$ thus satisfies:

$$P(A \mid B) := \frac{F(A \cap B)}{F(B)}. \tag{3.13}$$

Similar to the flow F, we abuse the notation P to define the probability of going through a state:

$$\forall s \in \mathcal{S} \qquad P(s) := \frac{F(s)}{Z}, \tag{3.14}$$

and similarly for the probability of going through an edge. Note that $P(s)$ *does not* correspond to a distribution over states, in the sense that $\sum_{s \in \mathcal{S}} P(s) \neq 1$; in particular, it is easy to see that $P(s_0) = 1$ (in other words, the probability of a trajectory passing through the initial state s_0 is 1). Additionally, for a trajectory $\tau \in \mathcal{T}$, we also use the abuse of notation $P(\tau)$ instead of $P(\{\tau\})$ to denote the probability of going through a specific trajectory τ.

Definition 3.2.12 (Transition probabilities)**.** Given a flow network (G, F), the **forward transition probability** operator P_F is a function on $\mathcal{S} \times \mathcal{S}$, that is a special case of the conditional probabilities induced by F ((3.13)):

$$\forall s \to s' \in \mathbb{A} \quad P_F(s' \mid s) := P(s \to s' \mid s) = \frac{F(s \to s')}{F(s)}. \tag{3.15}$$

Similarly, the **backwards transition probability** is the operator defined by:

$$\forall s \to s' \in \mathbb{A} \quad P_B(s \mid s') := P(s \to s' \mid s') = \frac{F(s \to s')}{F(s')}. \tag{3.16}$$

Note how P_F and P_B are consistent with G (in the sense of Definition 3.2.4), as a consequence of Proposition 3.2.8.

Because flows define probabilities over states and edges, they can be used to define probability distributions over the terminating states of a graph (denoted by $\mathcal{S}^f = Par(s_f)$)[2] as follows:

2. The set $\mathcal{S}^f$ is also denoted by $\mathcal{X}$ throughout this chapter.

Definition 3.2.13 (Terminating state probability)**.** Given a flow network (G, F), the **terminating state probability** $P_\top$ is the probability over terminating states $\mathcal{S}^f$ under the flow probability P:

$$\forall s \in \mathcal{S}^f \quad P_\top(s) := P(s \to s_f) = \frac{F(s \to s_f)}{Z} \tag{3.17}$$

Contrary to the probability $P(s)$ of going through a state s, the terminating state probability $P_\top$ is a well-defined distribution over the terminating states $s \in \mathcal{S}^f$, in the following sense:

Proposition 3.2.14. *The terminating state probability $P_\top$ is a well-defined distribution over the terminating states $s \in \mathcal{S}^f$, in that $P_\top(s) \geq 0$ for all $s \in \mathcal{S}^f$, and*

$$\sum_{s \in \mathcal{S}^f} P_\top(s) = 1. \tag{3.18}$$

The terminating state probability is particularly important in the context of estimating flow networks (see Section 3.3), as it shows that a flow network (G, F) induces a probability distribution over terminating states which is proportional to the terminating flows $F(s \to s_f)$, the normalization constant Z being given by initial flow $F(s_0)$.

3.2.4. Markovian Flows

Defining a flow requires the specification of $|\mathcal{T}|$ non-negative values (one for every trajectory $\tau \in \mathcal{T}$), which is generally exponential in the number of graph edges. Markovian flows, however have the remarkable property that they can be defined with much fewer "numbers", given that trajectory flows factorize according to G.

Definition 3.2.15 (Markovian flows)**.** Let (G, F) be a flow network, with flow probability measure P. F is called a **Markovian flow** (or equivalently (G, F) a **Markovian flow network**) if, for any state $s \neq s_0$, outgoing edge $s \to s'$, and for any trajectory $\tau = (s_0, s_1, \ldots, s_n = s) \in \mathcal{T}^{partial}$ starting in s_0 and ending in s:

$$P(s \to s' \mid \tau) = P(s \to s' \mid s) = P_F(s' \mid s). \tag{3.19}$$

Note that the Markovian property does not hold for all of the flows as defined in the previous sections (e.g., Figure 3.5). Intuitively, a flow can be considered non-Markovian if a particle in the "flow stream" can remember its past history; if not, its future behaviour can only depend on its current state and the flow must be Markovian. In this work, we will primarily be concerned with Markovian flows, though later we will re-introduce a form of

memory via state-conditional flows that allow each flow "particle" to remember parts of its history. The following proposition shows that Markovian flows have the property that the flows at (or the probabilities of) complete trajectories factorize according to the graph, and that it is a sufficient condition for defining Markovian flows.

Proposition 3.2.16. *Let (G, F) be a flow network, and P is the corresponding flow probability. The following three statements are equivalent:*

(1) F is a Markovian *flow*

(2) There exists a unique probability function $\hat{P}_F$ consistent with G such that for all complete trajectories $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}). \tag{3.20}$$

Moreover, the probability function $\hat{P}_F$ is exactly the forward transition probability associated with the flow probability P: $\hat{P}_F = P_F$.

(3) There exists a unique probability function $\hat{P}_B$ consistent with G such that for all complete trajectories $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = \prod_{t=1}^{n+1} \hat{P}_B(s_{t-1} \mid s_t). \tag{3.21}$$

Moreover, the probability function $\hat{P}_B$ is exactly the backwards transition probability associated with the flow probability P: $\hat{P}_B = P_B$.

The decomposition of (3.20) shows how Markovian flows can be used to draw terminating states from the terminating state probability $P_\top$ ((3.17)). Namely, we have the following result:

Corollary 3.2.17. *Let (G, F) be a Markovian flow network, and P_F the corresponding forward transition probability. Consider the procedure starting from $s = s_0$, and iteratively drawing one sample from $P_F(. \mid s)$ until reaching s_f. Then the probability of the procedure terminating in a state s is $P_\top(s)$.*

The following proposition shows that, as a consequence of the Proposition 3.2.16, we obtain three different parameterizations of Markovian flows.

Proposition 3.2.18. *Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, a Markovian flow on G is* completely *and* uniquely *specified by one of the following:*

(1) the combination of the total flow $\hat{Z}$ and the forward transition probabilities $\hat{P}_F(s' \mid s)$ for all edges $s \to s' \in \mathbb{A}$,

(2) the combination of the total flow $\hat{Z}$ and the backward transition probabilities $\hat{P}_B(s \mid s')$ for all edges $s \to s' \in \mathbb{A}$.

(3) the combination of the terminating flows $\hat{F}(s \to s_f)$ for all terminating edges $s \to s_f \in \mathbb{A}^f$ and the backwards transition probabilities $\hat{P}_B(s \mid s')$ for all non-terminating edges $s \to s' \in \mathbb{A}^{-f}$,

3.2.5. Flow-matching Conditions

In Proposition 3.2.18, we saw how forward and backward probability functions can be used to define a Markovian flow uniquely. We will show in the next proposition how non-negative functions of states and edges can be used to define a Markovian flow. Such functions cannot be unconstrained (as $\hat{P}_F$ and $\hat{Z}$ in Proposition 3.2.18 e.g.), as we have seen in Proposition 3.2.8.

Proposition 3.2.19. *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG. Consider a non-negative function $\hat{F}$ taking as input either a state $s \in \mathcal{S}$ or a transition $s \to s' \in \mathbb{A}$. Then $\hat{F}$ corresponds to a flow if and only if the* ***flow-matching conditions****:*

$$\begin{aligned} \forall s' > s_0, \quad \hat{F}(s') &= \sum_{s \in Par(s')} \hat{F}(s \to s') \\ \forall s' < s_f, \quad \hat{F}(s') &= \sum_{s'' \in Child(s')} \hat{F}(s' \to s'') \end{aligned} \tag{3.22}$$

are satisfied. More specifically, $\hat{F}$ uniquely defines a Markovian flow F matching $\hat{F}$ on states and transitions:

$$\forall \tau = (s_0, \dots, s_{n+1} = s_f) \in \mathcal{T} \quad F(\tau) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}. \tag{3.23}$$

Note how (3.22) can be used to recursively define the flow in all the states if Z is given and either the forward or the backwards transition probabilities are given. Either way, we would start from the flow at one of the extreme states s_0 or s_f and then distribute it recursively through the directed acyclic graph of the flow network, either going forward or going backward. A setting of particular interest that will be central in Section 3.3, is when we

are given all the terminal flows $F(s \to s_f)$, and we would like to deduce a state flow function $F(s)$ and a forward transition probability function $P_F(s' \mid s)$ for the rest of the flow network.

Next, we will see how to parameterize Markovian flows using forward and backward probability functions consistent with the DAG. Interestingly, the resulting condition is analogous to the *detailed balance* condition of Monte-Carlo Markov chains.

Definition 3.2.20 (Compatible transition probability functions)**.** Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, a forward transition probability function $\hat{P}_F$ and a backward transition probability function $\hat{P}_B$ consistent with G, $\hat{P}_F$ and $\hat{P}_B$ are **compatible** if there exists an edge flow function $\hat{F} : \mathbb{A} \to \mathbb{R}^+$ such that

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\sum_{s' \in Child(s)} \hat{F}(s \to s')}, \quad \hat{P}_B(s \mid s') \quad = \frac{\hat{F}(s \to s')}{\sum_{s'' \in Par(s')} \hat{F}(s'' \to s')} \tag{3.24}$$

Proposition 3.2.21. *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG. Consider a non-negative function $\hat{F}$ over states, a forward transition probability function $\hat{P}_F$ and a backwards transition probability function $\hat{P}_B$ consistent with G. Then, $\hat{F}, \hat{P}_B$, and $\hat{P}_F$ jointly correspond to a flow if and only if the* ***detailed balance conditions*** *holds:*

$$\forall s \to s' \in \mathbb{A} \quad \hat{F}(s)\hat{P}_F(s' \mid s) = \hat{F}(s')\hat{P}_B(s \mid s'). \tag{3.25}$$

More specifically, $\hat{F}, \hat{P}_F$, and $\hat{P}_B$ uniquely define a Markovian flow F matching $\hat{F}$ on states, and with transition probabilities matching $\hat{P}_F$ and $\hat{P}_B$. Furthermore, when this condition is satisfied, the forward and backward transition probability functions $\hat{P}_F$ and $\hat{P}_B$ are compatible.

At first glance, it may seem that when $\hat{P}_B$ is unconstrained, the detailed balance condition can trivially be achieved by setting

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_B(s \mid s') = \frac{\hat{P}_F(s' \mid s)\hat{F}(s)}{\hat{F}(s')} \tag{3.26}$$

However, because we also have the constraint $\sum_{s \in Par(s')} \hat{P}_B(s \mid s') = 1$, then (3.26) can only be satisfied if the flows are consistent with the forward transition:

$$\sum_{s \in Par(s')} \hat{P}_F(s' \mid s)\hat{F}(s) = \hat{F}(s').$$

3.2.6. Backwards Transitions can be Chosen Freely

Consider the setting in which we are given terminating flows to be matched, i.e., where the goal is to find a flow function with the right terminating flows. This is the setting

introduced in Bengio et al. (2021), and that will be studied in Section 3.2.7 and Section 3.3. In this case, Proposition 3.2.18 tells us that in order to fully determine the forward transition probabilities and the state or state-action flows, it is not sufficient in general to specify only the terminating flows; it is also necessary to specify the backwards transition probabilities on the edges other than the terminal ones (the latter being given by the terminating flows).

What this means is that the terminating flows do not specify the flow completely, e.g., because many different paths can land in the same terminating state. The preference over such different ways to achieve the same final outcome is specified by the backwards transition probability P_B (except for $P_B(s \mid s_f)$ which is a function of the terminating flows and Z). For example, we may want to give equal weight to all parents of a node s, or we may prefer shorter paths, which can be achieved if we keep track in the state s of the length of the shortest path to the node s, or we may let a learner discover a P_B that makes learning P_F or F easier.

3.2.7. Solving for the flows

Going back to the flow-matching conditions of Proposition 3.2.19, consider a scenario where an agent is provided a reward function $R : \mathcal{S}^f \to \mathbb{R}^+$, and is tasked of finding a function $\hat{F} : \mathcal{S} \cup \mathbb{A} \to \mathbb{R}^+$ that satisfies (3.22). This is equivalent to finding a function $\hat{F} : \mathbb{A} \to \mathbb{R}^+$ satisfying the following constraints:

$$\begin{cases} \forall s' \in \mathcal{S} \setminus \{s_0, s_f\},\ \sum_{s \in Par(s')} \hat{F}(s \to s') = \sum_{s'' \in Child(s')} \hat{F}(s' \to s''), \\ \forall s \to s' \in \mathbb{A},\ \hat{F}(s \to s') \geq 0, \\ \forall s' \in \mathcal{S}^f,\ \hat{F}(s' \to s_f) = R(s'), \end{cases} \tag{3.27}$$

then extending $\hat{F}$ to $\mathcal{S}$ as in (3.22). If such a function $\hat{F}$ is found, then according to Propositions 3.2.16 and 3.2.19 and Corollary 3.2.17, a sampler from the terminating state probability $P_\top \propto R$ (3.17), using the *forward policy* $\hat{P}_F$, defined as in (3.24). The sampling procedure is summarized in Algorithm 1. (3.27) shows that finding the edge flow function corresponds to solving a **linear system** of equations with **non-negativity constraints**.

An illustration of pointed DAG with the solution space to the corresponding system of constraints (3.27) is provided in Figure 3.4. Incidentally, this example shows that there are many solutions for the edge flow function, as argued for in Section 3.2.6. This can also be seen by analyzing (3.27): there are $|\mathcal{S}| + |\mathcal{S}^f| - 2$ equality constraints for $|\mathbb{A}|$ unknowns. Each solution $\hat{F}$ corresponds to a particular choice of the backward probability function P_B.

Interestingly, the structure of the pointed DAG implies that the system of constraints (3.27) can be solved efficiently, using dynamic programming. The process is detailed in Algorithm 2, with theoretical guarantees summarized in Proposition 3.2.22.

Algorithm 1 Sampling terminating states given edge flows

Input: A function $\hat{F} : \mathbb{A} \to \mathbb{R}^+$ satisfying (3.27) in a pointed DAG $G = (\mathcal{S}, \mathbb{A})$ for a given reward function $R : \mathcal{S}^f \to \mathbb{R}^+$.

Output: An object $s \in \mathcal{S}^f$, sampled with probability $P_\top(s) = \frac{R(s)}{\sum_{s' \in \mathcal{S}^f} R(s')}$.

$s \leftarrow s_0$
$s' \leftarrow \emptyset$
while $s' \neq s_f$ **do**
  Sample $s' \sim \hat{P}_F(. \mid s)$, where $\hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\sum_{s'' \in Child(s)} \hat{F}(s \to s'')}$
  if $s' \neq s_f$ **then**
    $s \to s'$
  end if
end while
Return: s

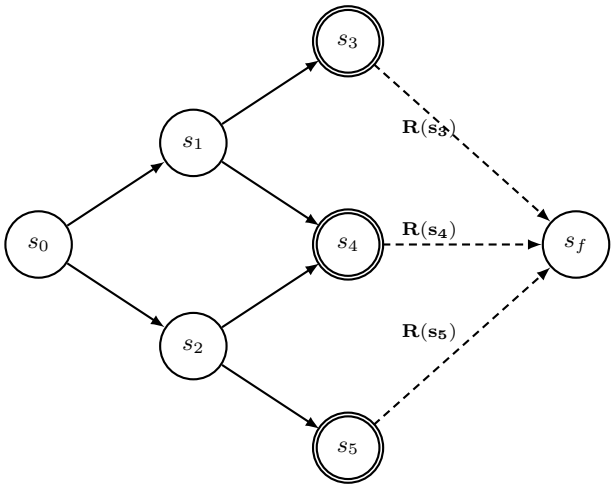


Figure 3.4 – An example of a pointed DAG with 3 terminating states, with a given reward function R. There is an infinite number of functions $\hat{F} : \mathbb{A} \to \mathbb{R}^+$ satisfying (3.27) for this pointed DAG. Besides the imposed values on $\hat{F}(s \to s_f)$ for $s \in \{s_3, s_4, s_5\}$, each $x \in [0, R(s_4)]$ defines a solution function as: $\hat{F}(s_0 \to s_1) = R(s_3) + x$, $\hat{F}(s_0 \to s_2) = R(s_5) + R(s_4) - x$, $\hat{F}(s_1 \to s_4) = x$, $\hat{F}(s_2 \to s_4) = R(s_4) - x$, $\hat{F}(s_1 \to s_3) = R(s_3)$, $\hat{F}(s_2 \to s_5) = R(s_5)$.

Proposition 3.2.22. *Algorithm 2 runs in $\mathcal{O}(|\mathbb{A}|)$ time, and produces an edge flow function $\hat{F}$ that satisfies (3.27).*

While Proposition 3.2.22 shows that Algorithm 2 provides an exact flow function for each given P_B in linear time (rather than cubic, for solutions to generic linear systems of equations), the computational cost $\mathcal{O}(|\mathbb{A}|)$ might still be restrictive for very large pointed DAGs. In fact, it is for such large DAGs that finding an edge flow function, or an approximation thereof is interesting for the sampling problem. Therefore, in practice, we resort

Algorithm 2 Solving for the edge flow function using dynamic programming

Input: A pointed DAG $G = (\mathcal{S}, \mathbb{A})$, a given reward function $R : \mathcal{S}^f \to \mathbb{R}^+$, and a given backward probability function $s \to s' \in \mathbb{A} \mapsto P_B(s \mid s')$.

Output: A function $\hat{F} : \mathbb{A} \to \mathbb{R}^+$ satisfying (3.27).

$\mathcal{Y} \leftarrow \emptyset$, a set representing states $s \in \mathcal{S}$ that do not need revisiting.
$U \leftarrow \emptyset$, a *queue* of tuples $(s, t) \in \mathcal{S} \times \mathbb{R}$, representing states with all outgoing edge flows known, and their corresponding state flows.
$\hat{F} \leftarrow \{s \to s' : \emptyset,\ s \to s' \in \mathbb{A}\}$, a dictionary representing the final edge flow values.
$V \leftarrow \{x : R(x),\ x \in \mathcal{S}^f\}$, a dictionary representing intermediate values of the state flows.
for $x \in \mathcal{S}^f$ **do**
  if $Child(x) = \{s_f\}$ **then**
    (Enqueue) Add the tuple $(x, R(x))$ to the queue U.
  end if
  $\hat{F}(s \to s_f) \leftarrow R(s)$
end for
while $U \neq \emptyset$ **do**
  (Dequeue) Pick and remove the first tuple (s', t) from the queue U.
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{s'\}$.
  for $s \in Par(s')$ **do**
    $\hat{F}(s \to s') \leftarrow tP_B(s \mid s')$.
    if $s \notin V$ **then**
      $V(s) \leftarrow \hat{F}(s \to s')$.
    else
      $V(s) \leftarrow V(s) + \hat{F}(s \to s')$.
    end if
    if $Child(s) \subseteq \mathcal{Y}$ **then**
      (Enqueue) Add the tuple $(s, V(s))$ to the queue U.
    end if
  end for
end while
Return: $\hat{F}$.

to approximating the edge flow function only, via function approximators such as neural networks, trained to minimize a non-negative loss that is zero only when the flow-matching conditions are satisfied. This is the core idea behind GFlowNets and will be expanded upon in Section 3.3, and more specifically in Section 3.3.2.

3.2.8. Equivalence Between Flows

In the previous sections, we have seen that Markovian flows have the property that trajectory flows or probabilities factorize according to the DAG, and we have seen different ways of characterizing Markovian flows. In Section 3.3, we show how to approximate Markovian flows in order to define probability measures over terminating states. In this section, through

an equivalence relation between trajectory flows, we justify the focus on Markovian flows. Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, we denote by:

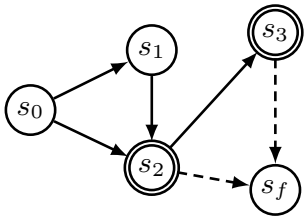
— $\mathcal{F}(G)$: the set of flows on G, i.e., the set of functions from $\mathcal{T}$, the set of complete trajectories in G, to $\mathbb{R}^+$,

— $\mathcal{F}_{Markov}(G)$: the set of flows in $\mathcal{F}(G)$ that are Markovian.

Definition 3.2.23 (Equivalent flows)**.** Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG, and $F_1, F_2 \in \mathcal{F}(G)$ two trajectory flow functions. We say that F_1 and F_2 are equivalent if they coincide on edge-flows, i.e.,

$$\forall s \to s' \in \mathbb{A} \quad F_1(s \to s') = F_2(s \to s'). \tag{3.28}$$

Figure 3.5 shows four flow functions in a simple pointed DAG that are pairwise equivalent.



τ	$F_1(\tau)$	$F_2(\tau)$	$F_3(\tau)$	$F_4(\tau)$
s_0, s_2, s_f	1	4/5	1	6/5
s_0, s_1, s_2, s_f	1	6/5	1	4/5
s_0, s_2, s_3, s_f	1	6/5	2	9/5
s_0, s_1, s_2, s_3, s_f	2	9/5	1	6/5

Figure 3.5 – Equivalent flows and Markovian flows. Flows F_1 and F_2 are equivalent. F_3 and F_4 are equivalent, but not equivalent to F_1 and F_2. F_2 and F_4 are Markovian. F_1 and F_3 are not Markovian. F_1, F_2, F_3 and F_4 coincide on the terminating flows, i.e., at $s_2 \to s_f$ and $s_3 \to s_f$.

This defines an equivalence relation (i.e., a relation that is reflexive, symmetric, and transitive). Hence, each flow F belongs to an equivalence class, and the set of flows $\mathcal{F}(G)$ can be partitioned into equivalence classes. Note that if two flows are equivalent, then the corresponding state flow functions also coincide (as a direct consequence of Proposition 3.2.8).

Proposition 3.2.24. *Given a pointed DAG G. If two flow function $F_1, F_2 \in \mathcal{F}_{Markov}(G)$ are equivalent, then they are equal. Additionally, for any flow function $F' \in \mathcal{F}(G)$, there exists a unique Markovian flow function $F \in \mathcal{F}_{Markov}(G)$ such that F and F' are equivalent.*

The previous proposition shows that in each equivalence class stands out a particular flow function, that has a property the other flows in the same equivalence class don't have: it is Markovian.

A consequence of this is that, if we care essentially about state and edge flows, instead of dealing with the full set of flows $\mathcal{F}(G)$, it suffices to restrict any flow learning problem to the

set of Markovian flows $\mathcal{F}_{Markov}(G)$. The advantage of this restriction is that defining a flow requires the specification of $F(\tau)$ for all trajectories $\tau \in \mathcal{T}$, whereas defining a Markovian flow requires the specification of $F(s \to s')$ for all edges $s \to s' \in \mathbb{A}$, which is generally exponentially smaller than $\mathcal{T}$ (note that the edge flows still need to satisfy the flow-matching conditions in Proposition 3.2.19). Thus, in order to approximate or learn a flow function that satisfies some conditions on its edge or state values, it suffices to approximate or learn a Markovian flow, by learning the edge flow function, which is a much smaller object than the actual flow function.

3.3. GFlowNets: Learning a Flow

All propositions and corollaries of this section are proved in Appendix D.2.

With the theoretical preliminaries established in Section 3.2, we now consider the general class of problems introduced by Bengio et al. (2021) where some constraints or preferences over flows are given. Our goal is to find functions such as the state flow function $F(s)$ or the transition probability function $P(s \to s' \mid s)$ that best match these desiderata using corresponding estimators $\hat{F}(s)$ and $\hat{P}(s \to s' \mid s)$ which may not correspond to a proper flow. Such learning machines are called Generative Flow Networks (or GFlowNets for short). We focus on scenarios where we are given a target *reward function* $R : \mathcal{S}^f \to \mathbb{R}^+$, and aim at estimating flows F that satisfy:

$$\forall s \in \mathcal{S}^f \quad F(s \to s_f) = R(s), \tag{3.29}$$

such that the corresponding sampler, given by Algorithm 1, is with probability masses proportional to the reward R.

We assume that the pointed DAG is too large for Algorithm 2 to be practical, and we resort to the humbler goal of only approximating the edge flows (or the transition probability functions).

Because of the equivalences that exist in the set of flows, then without loss of generality, we choose GFlowNets to approximate Markovian flows only. We are thus interested in the following set of flows:

$$\mathcal{F}_{Markov}(G, R) = \{F \in \mathcal{F}_{Markov}(G),\ \forall s \in \mathcal{S}^f \quad F(s \to s_f) = R(s)\} \tag{3.30}$$

For now, we informally define a **GFlowNet** as an **estimator of a Markovian flow** function $F \in \mathcal{F}_{Markov}(G, R)$. We provide a more formal definition later-on.

With an estimator $\hat{F}$ of such a Markovian flow F, we can define an approximate forward transition probability function $\hat{P}_F$, as in Proposition 3.2.16, in order to draw trajectories $\tau \in \mathcal{T}$ (the set of complete trajectories in G) by iteratively sampling each state given the

previous one, starting at s_0 and then with $s_{t+1} \sim \hat{P}_F(. \mid s_t)$ until we reach the sink state $s_{n+1} = s_f$ for some n, as explained in Algorithm 1.

Next, we will clarify how such an estimator can be obtained.

3.3.1. GFlowNets and flow-matching losses

We have seen in Section 3.2.4 and Section 3.2.5 different ways of parameterizing a flow. For example, with a partition function and forward transition probabilities, or with edge flows that satisfy the flow-matching conditions. Because there are many ways to parameterize GFlowNets, we start with an abstract formulation for them, where $o \in \mathcal{O}$ represents a parameter configuration (e.g., resulting from or while training of a GFlowNet), $\Pi(o)$ gives the corresponding probability measure over trajectories $\tau \in \mathcal{T}$, and $\mathcal{H}$ maps a Markovian flow F to its parameterization o. In the following definition, we show what conditions should be satisfied in order for such a parameterization to be valid.

Definition 3.3.1 (Flow parameterization)**.** Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, with an initial and sink states s_0 and s_f respectively, and a target reward function $R : \mathcal{S}^f \to \mathbb{R}^+$, we say that the triplet $(\mathcal{O}, \Pi, \mathcal{H})$ is a **flow parameterization** of (G, R) if:

(1) $\mathcal{O}$ is a non-empty set,

(2) Π is a function mapping each object $o \in \mathcal{O}$ to an element $\Pi(o) \in \mathcal{P}(\mathcal{T})$, the set of probability distributions on $\mathcal{T}$,

(3) $\mathcal{H}$ is an injective functional from $\mathcal{F}_{Markov}(G, R)$ to $\mathcal{O}$,

(4) For any $F \in \mathcal{F}_{Markov}(G, R)$, $\Pi(\mathcal{H}(F))$ is the probability measure associated with the flow F (Definition 3.2.11).

To each object $o \in \mathcal{O}$, the distribution $\Pi(o)$ implicitly defines a **terminating state probability** measure:

$$\forall s \in \mathcal{S}^f \quad P_\top(s) := \sum_{\tau \in \mathcal{T} : s \to s_f \in \tau} \Pi(o)(\tau), \tag{3.31}$$

where the dependence on o in $P_\top$ is omitted for clarity.

The intuition behind the introduction of $(\mathcal{O}, \Pi, \mathcal{H})$ is that we can define a probability measure over $\mathcal{T}$ for each object $o \in \mathcal{O}$, but only some of these objects correspond to a Markovian flow with the right terminating flows. For such objects o (i.e., those that can be written as $o = \mathcal{H}(F)$ for some flow $F \in \mathcal{F}_{Markov}(G, R)$), the probability measure $P_\top$ corresponds to the distribution of interest, according to Definition 3.2.13, i.e.,

$$\forall s \in \mathcal{S}^f \quad P_\top(s) \propto R(s). \tag{3.32}$$

GFlowNets thus provide a solution to the generally intractable problem of sampling from a target reward function R, or its associated **energy function**:

$$\forall s \in \mathcal{S}^f \quad \mathcal{E}(s) := -\log R(s) \tag{3.33}$$

Directly approximating flows $F \in \mathcal{F}_{Markov}(G, R)$ is a hard problem, whereas with some sets $\mathcal{O}$, searching for an object $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R)) \subseteq \mathcal{O}$ is a simpler problem that can be tackled with function approximation techniques.

Note that the set $\mathcal{O}$ cannot be arbitrary, as there needs to be a way to define an injective function from $\mathcal{F}_{Markov}(G, R)$ to $\mathcal{O}$. Below, for a given DAG G, we show three examples clarifying the abstract concept of parameterization:

Example 3.3.2 (Edge-flow parameterization)**.** *Consider $\mathcal{O}_{edge} = \mathcal{F}(\mathbb{A}^{-f}, \mathbb{R}^+)$, the set of functions from $\mathbb{A}^{-f}$ to $\mathbb{R}^+$, and the functionals $\mathcal{H}_{edge} : \mathcal{F}_{Markov}(G, R) \to \mathcal{O}_{edge}$ and $\Pi_{edge} : \mathcal{O}_{edge} \to \mathcal{P}(\mathcal{T})$ defined by:*

$$\mathcal{H}_{edge}(F) : (s \to s') \in \mathbb{A}^{-f} \mapsto F(s \to s'), \tag{3.34}$$

$$\forall \tau = (s_0, \dots, s_n = s_f) \in \mathcal{T} \quad \Pi_{edge}(\hat{F})(\tau) \propto \prod_{t=1}^{n} P_{\hat{F}}(s_t \mid s_{t-1}), \tag{3.35}$$

where $P_{\hat{F}}$ is defined for every $s \to s' \in \mathbb{A}$ by:

$$P_{\hat{F}}(s' \mid s) = \begin{cases} \frac{\hat{F}(s \to s')}{\sum_{s'' \in Child(s) \setminus \{s_f\}} \hat{F}(s \to s'') + \mathbf{1}(s_f \in Child(s)) R(s)} & \text{if } s' \neq s_f \\ \frac{\mathbf{1}(s_f \in Child(s)) R(s)}{\sum_{s'' \in Child(s) \setminus \{s_f\}} \hat{F}(s \to s'') + \mathbf{1}(s_f \in Child(s)) R(s)} & \text{if } s' = s_f \end{cases} \tag{3.36}$$

The injectivity of $\mathcal{H}_{edge}$ follows directly from Proposition 3.2.24 (two Markovian flows that coincide on both their terminating and non-terminating edge flow values are equal). And for any Markovian flow $F \in \mathcal{F}_{Markov}(G, R)$, $\Pi_{edge}(\mathcal{H}_{edge}(F))$ equals the probability measure associated with F, as is shown in Proposition 3.2.16.

$(\mathcal{O}_{edge}, \Pi_{edge}, \mathcal{H}_{edge})$ is thus a valid flow parameterization of (G, R).

Example 3.3.3 (Forward transition probability parameterization)**.** *Consider the set $\mathcal{O}_{PF} = \mathcal{O}_1 \times \mathcal{O}_2$, where $\mathcal{O}_1 = \mathcal{F}(\mathcal{S} \setminus \{s_f\}, \mathbb{R}^+)$ is the set of function from $\mathcal{S} \setminus \{s_f\}$ to $\mathbb{R}^+$ and $\mathcal{O}_2$ is the set of forward probability functions $\hat{P}_F$ consistent with G , and the functionals $\mathcal{H}_{PF} : \mathcal{F}_{Markov}(G, R) \to \mathcal{O}_{PF}$ and $\Pi_{PF} : \mathcal{O}_{PF} \to \mathcal{P}(\mathcal{T})$ defined by:*

$$\mathcal{H}_{PF}(F) = \left(s \in \mathcal{S} \setminus \{s_f\} \mapsto F(s), (s \to s') \in \mathbb{A} \mapsto P_F(s' \mid s)\right), \tag{3.37}$$

$$\forall \tau = (s_0, \dots, s_n = s_f) \in \mathcal{T} \quad \Pi_{PF}(\hat{F}, \hat{P}_F)(\tau) \propto \prod_{t=1}^{n} \hat{P}_F(s_t \mid s_{t-1}), \tag{3.38}$$

where P_F is the forward transition probability function associated with F ((3.15)). To verify that $\mathcal{H}_{PF}$ is injective, consider $F_1, F_2 \in \mathcal{F}_{Markov}(G, R)$ such that $\mathcal{H}_{PF}(F_1) = \mathcal{H}_{PF}(F_2)$. It

means that $\forall s \in \mathcal{S}^f$, $F_1(s) = F_2(s)$, *and* $\forall s \to s' \in \mathbb{A}$, $\frac{F_1(s\to s')}{F_1(s)} = \frac{F_2(s\to s')}{F_2(s)}$. *It follows that* $\forall s \to s' \in \mathbb{A}$, $F_1(s \to s') = F_2(s \to s')$. *Which, according to Proposition 3.2.24, means that* $F_1 = F_2$. *And for any Markovian flow* $F \in \mathcal{F}_{Markov}(G, R)$, $\Pi_{PF}(\mathcal{H}_{PF}(F))$ *equals the probability measure associated with* F, *as is shown in Proposition 3.2.16.*

$(\mathcal{O}_{PF}, \Pi_{PF}, \mathcal{H}_{PF})$ *is thus a valid flow parameterization of* (G, R).

Example 3.3.4 (Transition probabilities parameterization)**.** *Similar to Example 3.3.3, we can parameterize a Markovian flow using the state-flow function and both its forward and backward transition probabilities, i.e., with* $\mathcal{O}_{PFB} = \mathcal{O}_{PF} \times \mathcal{O}_3$, $\mathcal{H}_{PFB}$, *and* Π_{PFB} *defined as:*

$$\mathcal{H}_{PFB}(F) = \Big(\mathcal{H}_{PF}(F), (s \to s') \in \mathbb{A}^{-f} \mapsto P_B(s \mid s'), \Big), \tag{3.39}$$

$$\forall \tau = (s_0, \ldots, s_n = s_f) \in \mathcal{T} \quad \Pi_{PFB}(\hat{F}, \hat{P}_F, \hat{P}_B)(\tau) \propto \prod_{t=1}^{n} \hat{P}_F(s_t \mid s_{t-1}), \tag{3.40}$$

where P_B *is the function defined by (3.16). and* $\mathcal{O}_3$ *is the set of backward probability functions* $\hat{P}_B$ *consistent with* G. *The injectivity of* $\mathcal{H}_{PFB}$ *is a direct consequence of that of* $\mathcal{H}_{PF}$. *And for any Markovian flow* F, $\Pi_{PFB}(\mathcal{H}_{PFB}(F))$ *equals the probability measure associated with* F, *as is shown in Prop.3.*

$(\mathcal{O}_{PFB}, \Pi_{PFB}, \mathcal{H}_{PFB})$ *is thus a valid flow parameterization of* (G, R).

We now have all the ingredients to formally define a GFlowNet:

Definition 3.3.5 (GFlowNets)**.** A **GFlowNet** is a tuple $(G, R, \mathcal{O}, \Pi, \mathcal{H})$, where:

- $G = (\mathcal{S}, \mathbb{A})$ is a pointed DAG with initial state s_0 and sink state s_f,
- $R : \mathcal{S}^f \to \mathbb{R}^+$ a target reward function,
- $(\mathcal{O}, \Pi, \mathcal{H})$ a flow parameterization of (G, R).

Each object $o \in \mathcal{O}$ is called a GFlowNet configuration. When it is clear from context, we will use the term GFlowNet to refer to both $(G, R, \mathcal{O}, \Pi, \mathcal{H})$ and a particular configuration o; similar to how the term "Neural Network" refers to both the class of functions that can be represented with a particular architecture, and to a particular element of that class or weight configuration.

If $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R))$, then the corresponding terminating state probability measure ((3.31)) is proportional to the target reward R.

Once we have a GFlowNet $(G, R, \mathcal{O}, \Pi, \mathcal{H})$, we still need a way to find objects $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R)) \subseteq \mathcal{O}$. To this end, it suffices to design a **loss function** $\mathcal{L}$ on $\mathcal{O}$ that equals zero on objects $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R))$ and only on those objects. If our loss function $\mathcal{L}$ is chosen to be non-negative, then an approximation of the target distribution (on $\mathcal{S}^f$) is obtained by approximating the minimum of the function $\mathcal{L}$. This provides a recipe for casting

the search problem of interest to a minimization problem, as we typically do in machine learning. Such loss functions can be easily designed for the natural parameterizations we considered in Example 3.3.2, Example 3.3.3, and Example 3.3.4, as we will illustrate below.

Definition 3.3.6 (Flow-matching losses)**.** Let $(G, R, \mathcal{O}, \Pi, \mathcal{H})$ be a GFlowNet. A **flow-matching loss** is any function $\mathcal{L} : \mathcal{O} \to \mathbb{R}^+$ such that:

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = 0 \;\Leftrightarrow\; \exists F \in \mathcal{F}_{Markov}(G, R) \;\; o = \mathcal{H}(F) \tag{3.41}$$

We say that $\mathcal{L}$ is **edge-decomposable**, if there exists a function $L : \mathcal{O} \times \mathbb{A} \to \mathbb{R}^+$ such that:

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = \sum_{s \to s' \in \mathbb{A}} L(o, s \to s'), \tag{3.42}$$

We say that $\mathcal{L}$ is **state-decomposable**, f there exists a function $L : \mathcal{O} \times \mathcal{S} \to \mathbb{R}^+$ such that:

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = \sum_{s \in \mathcal{S}} L(o, s), \tag{3.43}$$

We say that $\mathcal{L}$ is **trajectory-decomposable** if there exists a function $L : \mathcal{O} \times \mathcal{T} \to \mathbb{R}^+$ such that:

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = \sum_{\tau \in \mathcal{T}} L(o, \tau) \tag{3.44}$$

As mentioned above, with a such a loss function, our search problems can be written as minimization problems of the form

$$\min_{o \in \mathcal{O}} \mathcal{L}(o), \tag{3.45}$$

which can be tackled with gradient-based learning if the function $\mathcal{L}$ is differentiable. Note that with an edge-decomposable flow-matching loss, the minimization problem in (3.45) is equivalent to:

$$\min_{o \in \mathcal{O}} \mathbb{E}_{(s \to s') \sim \pi_T}[L(o, s \to s')], \tag{3.46}$$

where π_T is any full support probability distribution on $\mathbb{A}$, i.e., a probability distribution such that $\forall s \to s' \in \mathbb{A} \;\; \pi_T(s \to s') > 0$. A similar statement can be made for state-decomposable or trajectory-decomposable flow-matching losses.

Example 3.3.7 (Flow-matching loss)**.** *Consider the edge-flow parameterization* $(\mathcal{O}_{edge}, \Pi_{edge}, \mathcal{H}_{edge})$*, and the function* $L_{FM} : \mathcal{O}_{edge} \times \mathcal{S} \to \mathbb{R}^+$ *defined for each* $\hat{F} \in \mathcal{O}_{edge}$ *and* $s' \in \mathcal{S}$ *as*

$$L_{FM}(\hat{F}, s') = \begin{cases} \left(\log \left(\frac{\delta + \sum_{s \in Par(s')} \hat{F}(s \to s')}{\delta + R(s') + \sum_{s'' \in Child(s') \setminus \{s_f\}} \hat{F}(s' \to s'')} \right) \right)^2 & \text{if } s' \neq s_f, \\ 0 \quad \text{otherwise} \end{cases} \tag{3.47}$$

where $\delta \geq 0$ is a hyper-parameter. The function $\mathcal{L}_{FM}$ mapping each $\hat{F} \in \mathcal{O}_{edge}$ to

$$\mathcal{L}_{FM}(\hat{F}) = \sum_{s \in \mathcal{S}} L_{FM}(\hat{F}, s) \tag{3.48}$$

is a flow-matching loss that is (by definition) state-decomposable.

To see this, let $\hat{F} \in \mathcal{O}_{edge}$ such that $\mathcal{L}_{FM}(\hat{F}) = 0$, and extend it to terminating edge:

$$\forall s \in \mathcal{S}^f \quad \hat{F}(s \to s_f) := R(s) \tag{3.49}$$

Now that $\hat{F}$ is defined for all edges in G, we can write that

$$\forall s' \in \mathcal{S} \quad \sum_{s \in Par(s')} \hat{F}(s \to s') = \sum_{s'' \in Child(s)} \hat{F}(s' \to s''). \tag{3.50}$$

Which, according to Proposition 3.2.19, means that there exists a Markovian flow $F \in \mathcal{F}_{Markov}(G, R)$ such that $\mathcal{H}_{edge}(F) = \hat{F}$. The converse

$$\forall F \in \mathcal{F}_{Markov}(G, R) \quad \mathcal{L}_{FM}(\mathcal{H}_{edge}(F)) = 0 \tag{3.51}$$

is a trivial consequence of Proposition 3.2.19.

This is the loss function proposed in Bengio et al. (2021). δ allows to reduce the importance given to small flows (those smaller than δ), and the usage of the square of the log-ratio is justified as a way to ensure that states with large flows do not contribute to the gradients of $\mathcal{L}_{FM}$ much more than states with small flows.

Example 3.3.8 (Detailed-balance (DB) loss)**.** *Consider the transition probabilities parameterization $(\mathcal{O}_{PFB}, \Pi_{PFB}, \mathcal{H}_{PFB})$, and the function $L_{DB} : \mathcal{O}_{PFB} \times \mathbb{A} \to \mathbb{R}^+$ defined for each $(\hat{F}, \hat{P}_F, \hat{P}_B) \in \mathcal{O}_{PFB}$ and $s \to s' \in \mathbb{A}$ as*

$$L_{DB}(\hat{F}, \hat{P}_F, \hat{P}_B, s \to s') = \begin{cases} \left(\log\left(\frac{\delta + \hat{F}(s)\hat{P}_F(s'|s)}{\delta + \hat{F}(s')\hat{P}_B(s|s')}\right)\right)^2 & \text{if } s' \neq s_f, \\ \left(\log\left(\frac{\delta + \hat{F}(s)\hat{P}_F(s'|s)}{\delta + R(s)}\right)\right)^2 & \text{otherwise,} \end{cases} \tag{3.52}$$

where $\delta \geq 0$ is a hyper-parameter. The function $\mathcal{L}_{DB}$ mapping each $(\hat{F}, \hat{P}_F, \hat{P}_B) \in \mathcal{O}_{PFB}$ to

$$\mathcal{L}_{DB}(\hat{F}, \hat{P}, \hat{P}_B)) = \sum_{s \to s' \in \mathbb{A}} L_{DB}(\hat{F}, \hat{P}, \hat{P}_B, s \to s') \tag{3.53}$$

is a flow-matching loss that is (by definition) edge-decomposable. The proof of this statement is similar to the one of the example above, using Proposition 3.2.21.

According to Section 3.2.6, the reward function does not completely specify the flow. Thus, the detailed-balance loss of Example 3.3.8 can be used with the $(\mathcal{O}_{PF}, \Pi_{PF}, \mathcal{H}_{PF})$ parameterization, using any function $\hat{P}_B \in \mathcal{O}_3$ as input to the detailed-balance loss.

Example 3.3.9 (Trajectory-balance (TB) loss)**.** *This loss has been introduced in Malkin et al. (2022) for the parameterization $(\mathcal{O}_{TB}, \Pi_{TB}, \mathcal{H}_{TB})$, where $\mathcal{O}_{TB} = \mathcal{O}_1 \times \mathcal{O}_2 \times \mathcal{O}_3$, with $\mathcal{O}_1 = \mathbb{R}^+$ parameterizes the partition function $\hat{Z}$, and $\mathcal{O}_2$ and $\mathcal{O}_3$ introduced in Example 3.3.3*

and Example 3.3.4 (the set of forward and backward probabilities consistent with G). $\mathcal{H}_{TB}$ maps a Markovian flow in $\mathcal{F}_{Markov}(G, R)$ to the corresponding triplet (Z, P_F, P_B), and Π_{TB} maps a parameterization $(\hat{Z}, \hat{P}_F, \hat{P}_B)$ to a probability over trajectories defined by $\hat{P}_F$ as in Example 3.3.3. Proposition 3.2.18 justifies the validity of this parameterization. The loss $\mathcal{L}_{TB}$ maps each $(\hat{Z}, \hat{P}_F, \hat{P}_B) \in \mathcal{O}_{TB}$ to:

$$\mathcal{L}_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B) = \sum_{\tau \in \mathcal{T}} L_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B, \tau), \tag{3.54}$$

where

$$\forall \tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T} \quad L_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B, \tau) = \left(\log \frac{\hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1})}{R(s_n) \prod_{t=1}^{n} \hat{P}_B(s_{t-1} \mid s_t)} \right)^2. \tag{3.55}$$

Malkin et al. (2022) prove that $\mathcal{L}_{TB}$ is a flow-matching loss and call it trajectory balance. *It is trajectory-decomposable by definition.*

3.3.2. Training by stochastic gradient descent:

In the examples of the previous section, given a GFlowNet $(G, R, \mathcal{O}, \Pi, \mathcal{H})$ and a flow-matching loss $\mathcal{L}$, objects $o \in \mathcal{O}$ are themselves functions or combinations of functions, and we can thus parameterize $\mathcal{O}$ with function approximators such as neural networks. However, most of the times, the evaluation (let alone the minimization) of $\mathcal{L}(o)$ is intractable, given that even with a full support distribution, only a subset of edges (or states or trajectories) can be visited in finite time. In practice, with an edge-decomposable loss e.g., we resort to a stochastic gradient, such as

$$\nabla_o L(o, s \to s'), \quad s \to s' \sim \pi_o \tag{3.56}$$

for edge-decomposable losses, or

$$\nabla_o L(o, \tau), \quad \tau \sim \pi_o \tag{3.57}$$

for trajectory-decomposable losses, where π_o, called the **training distribution**, is a distribution over edges or trajectories that can be associated with $\Pi(o)$, corresponding to the online setting in RL, or defined in other ways, corresponding to the behavior policy in offline RL, see Section 3.3.3.3 below.

3.3.3. Extensions

In this section, we discuss possible relaxations to the GFlowNet training paradigm introduced thus far.

3.3.3.1. Introducing Time Stamps to Allow Cycles

Note that the state space of a GFlowNet can easily be modified to accommodate an underlying state space for which the transitions do not form a DAG, e.g., to allow cycles. Let $\mathcal{S}$ be such an underlying state-space. Define the augmented state space $\mathcal{S}' = \mathcal{S} \times \{0, \ldots, T\}$, and $s'_t = (s_t, t)$ is the augmented state, where t is the position of the state s_t in the trajectory. With this augmented state space, we automatically avoid cycles. Furthermore, we may design or train the backwards transition probabilities $P_B(s'_t \mid s'_{t+1} = (s_{t+1}, t+1))$ to create a preference for shorter paths towards s_{t+1}, as discussed in Section 3.2.6. Note that we can further generalize this setup by replacing $\{0, \ldots, T\}$ with any finite totally ordered indexing set; the augmented state space will still have an associated DAG. The ordering "$<$" in the original state-space is lifted to the augmented state-space: $(s_t, t) < (s'_{t'}, t')$ if and only if $t < t'$ and $s_t < s'_t$.

3.3.3.2. Stochastic Rewards

We also consider the setting in which the given reward is stochastic rather than being a deterministic function of the state, yielding training procedures based on stochastic gradient descent. For example, with the trajectory balance loss of (3.55), if $R(s)$ is stochastic (even when given s), we can think of what is being really optimized is the squared loss with $\log R(s)$ replaced by its expectation (given s) [3]. This is a straightforward consequence of minimizing the expected value of a squared error loss, as for example in neural networks trained with a squared error loss and a stochastic target output, where the neural network effectively tries to estimate the expected value of that target.

3.3.3.3. GFlowNets can be trained offline

As discussed in Section 3.3.2, we do not need to train a GFlowNet using samples from its own trajectory distribution $\hat{P} = \Pi(o)$. Those training trajectories can be drawn from any training distribution π_T with full support, as already shown by Bengio et al. (2021). It means that a GFlowNet can be trained offline, as in offline reinforcement learning (Ernst et al., 2005; Riedmiller, 2005; Lange et al., 2012).

It should also be noted that with a proper adaptive choice of π_T, and assuming that computing R is cheaper or comparable in cost to running the GFlowNet on a trajectory, it should be more efficient to continuously draw new training samples from π_T than to rehearse the same trajectories multiple times. An exception would be rehearsing the trajectories leading to high rewards if these are rare.

How should one choose the training distribution π_T? It needs to cover the support of R, but if it were uniform it would be very wasteful, and if it were equal to the current GFlowNet policy π it might not have sufficient effective support and thus miss modes of R. Hence the

3. In this case, the resulting terminating state distribution would be proportional to $\exp(\mathbb{E}[\log R(s)])$.

training distribution should be sampled from an exploratory policy that visits places that have not been visited yet and may have a high reward. High epistemic uncertainty around the current policy would make sense and the literature on acquisition functions for Bayesian optimization (Srinivas et al., 2010) may be a good guide. More generally, this means the training distribution should be adaptive. For example, π_T could be the policy of a second GFlowNet trained mostly to match a different reward function that is high when the losses observed by the main GFlowNet are large. It would also be good to regularly visit those trajectories corresponding to known large R, i.e., according to samples from π, to make sure those are not forgotten, even temporarily.

3.3.3.4. Exploiting Data as Known Terminating States

In some applications we may have access to a dataset of $(s, R(s))$ pairs and we would like to use them in a purely offline way to train a GFlowNet, or we may want to combine such data with queries of the reward function R to train the GFlowNet. For example, the dataset may contain examples of some of the high-reward terminating states s which would be difficult to obtain by sampling from a randomly initialized GFlowNet. How can we compute a gradient update for the GFlowNet parameters using such $(s, R(s))$ pairs?

If we choose to parameterize the backwards transition probabilities P_B (which is necessary for implementing the detailed balance loss), then we can just sample a trajectory τ leading to s using P_B and use these trajectories to update the flows and forward transition probabilities along the traversed transitions. However, this alone is not guaranteed to produce the correct GFlowNet sampling distribution because the empirical distribution over training trajectories τ defined as above does not have full support. Suppose for example that the dataset only contains high-reward terminating states with $R(s) = 1$. The GFlowNet could then just sample trajectories uniformly (which would be wrong, we would like the probability of most states not in the training set to be very small). On the other hand, if we combine the distribution of trajectories leading to terminal transitions in the dataset with a training distribution whose support covers all possible trajectories, then the offline property of GFlowNet guarantees that we can recover a flow-matching model.

3.4. Conditional Flows and Free energies

A remarkable property of flow networks is that we can recover the normalizing constant Z from the initial state flow $F(s_0)$ (Proposition 3.2.10). Z also gives us the partition function associated with a given terminal reward function R specifying the terminating flows.

What about internal states s with $s_0 < s < s_f$? If we had something like a normalizing constant for only the terminating flows achievable from s, we would be able to obtain a form of marginalization given state s, i.e., a conditional probability for terminating states $s' \geq s$, given s. Naturally, one could ask: does the flow $F(s)$ through a state s give us that kind of

marginalization over only the downstream terminating flows? Unfortunately in general, the answer to this question is *no*, as illustrated in Figure 3.6: in this example $F(s_2) = 4$, whereas the sum of terminating flows achievable from s_2 is 6 (the terminating states reachable from s_2 are $\{s_5, s_6, s_7\}$). The discrepancy is caused by the flow through (s_0, s_1, s_5) that contributes to the terminating flow $F(s_5 \to s_f)$, but not to $F(s_2)$ since there is no order relation between s_1 and s_2.

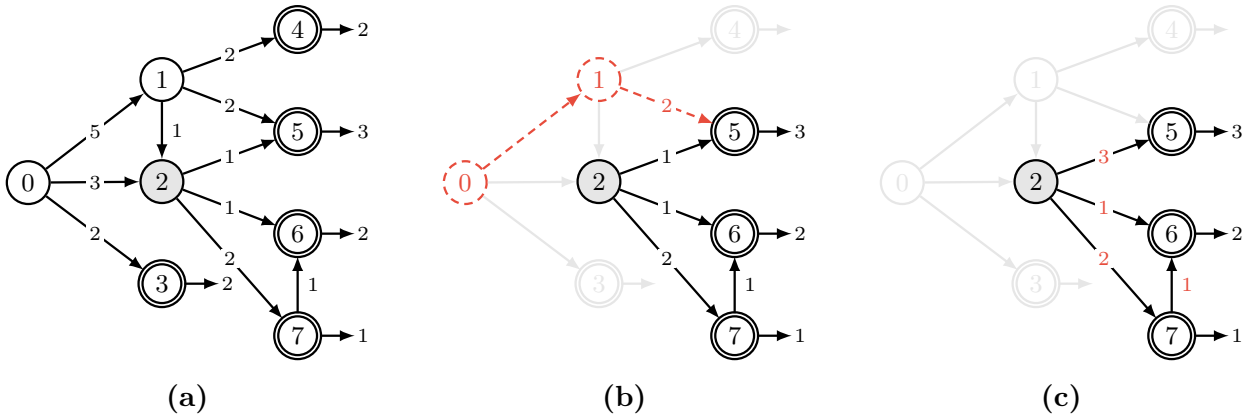


Figure 3.6 – Example of a state-conditional flow network. (a) The original (Markovian) flow network. (b) The subgraph of states reachable from s_2; there is a flow through (s_0, s_1, s_5) that contributed to $F(s_5 \to s_f)$, but not to $F(s_2)$, showing that $F(s_2)$ does not marginalize the rewards of its descendant. (c) State-conditional flow network F_{s_2}, which differs from the original flow F on the subgraph, but satisfies the desired marginalization property.

This motivates the following definition:

Definition 3.4.1 (Free energies)**.** Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, the corresponding partial order denoted by $\geq$, and a function $\mathcal{E} : \mathcal{S} \to \mathbb{R}$, called the **energy function**, we define the **free energy** $\mathcal{F}(s)$ of a state s as:

$$e^{-\mathcal{F}(s)} := \sum_{s' : s' \geq s} e^{-\mathcal{E}(s')}. \tag{3.58}$$

Free energies are generic formulations for the marginalization operation (i.e., summing over a large number of terms) associated with energy functions, and their estimation opens the door to interesting applications where expensive MCMC methods would typically be the main approach otherwise.

3.4.1. Conditional flow networks

In Section 3.2.2, we defined a flow network as a DAG, augmented with some function F over the set of complete trajectories $\mathcal{T}$. We can extend this notion of flow networks by conditioning each component on some information x. In general, this conditioning variable can

represent any conditioning information, either external to the flow network (but influencing the terminating flows), or internal (e.g., x can be a property of complete trajectories over another flow network, like passing through a particular state).

Definition 3.4.2 (Conditional flow networks)**.** Let $\mathbb{X}$ be a set of conditioning variables. We consider a family of DAGs $G_x = (\mathcal{S}_x, \mathcal{A}_x)$ indexed by $x \in \mathbb{X}$, along with a family of initial and terminal states denoted by $(s_0 \mid x) \in \mathcal{S}_x$ and $(s_f \mid x) \in \mathcal{S}_x$ respectively. For each DAG G_x, we denote by $\mathcal{T}_x$ the set of complete trajectories in G_x, and we denote by $\mathcal{T}$ their union:

$$\mathcal{T} = \bigcup_{x \in \mathbb{X}} \mathcal{T}_x. \tag{3.59}$$

A **conditional flow network** is the specification of $\mathbb{X}$, the family $\{G_x,\ x \in \mathbb{X}\}$, along with a conditional flow function F, i.e., a function $F : \mathbb{X} \times \mathcal{T} \to \mathbb{R}^+$ such that $F(x, \tau) = 0$ if $\tau \notin \mathcal{T}_x$. For clarity, we will denote, for each $x \in \mathbb{X}$, by F_x the function mapping each $\tau \in \mathcal{T}_x$ to $F(x, \tau)$. Similar to Section 3.2.2, F_x induces a measure of the σ-algebra $2^{\mathcal{T}_x}$ for each x.

Conditional flow networks effectively represent a family of flow networks, indexed by the value of x. Since conditional flow networks are defined using the same components as an unconditional flow network, they inherit from all the properties of flow networks for all DAGs G_x and flow functions F_x. In particular, we can directly extend the notion of probability distribution over flows, state and edge flows, forward and backward transition probabilities (Section 3.2.3), of Markovian flows (Section 3.2.4), and any flow-matching condition (Section 3.2.5) to conditional flows; the only difference is that now every term explicitly depends of the conditioning variable x.

In Section 3.4.2 and Section 3.4.3, we will elaborate two important examples of conditional flow networks: flow networks conditioned on external information that changes the reward $R(s \mid x)$, and state-conditional flow networks that depend on internal information, i.e., previously visited states.

3.4.2. Reward-conditional flow networks

Definition 3.4.3 (Reward-conditional flow networks)**.** Let $\mathbb{X}$ be a set of conditioning variables. Consider a flow network given by a pointed DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow function F. Consider a family $\mathcal{R}$ of non-negative functions of $\mathcal{S}$: $\{R_x : \mathcal{S}^f \to \mathbb{R}^+,\ x \in \mathbb{X}\}$. A **reward-conditional flow network** compatible with the family $\mathcal{R}$ is a conditional flow network (Definition 3.4.2), with $G_x = G$ for every $x \in \mathbb{X}$, such that the edge-flow functions induced by the conditional flow function F satisfy:

$$\forall x \in \mathbb{X} \quad \forall s \in \mathcal{S}^f \quad F_x(s \to s_f) = R_x(s). \tag{3.60}$$

We will use the notations $R_x(s)$ and $R(s \mid x)$ interchangeably.

Note that the definition above implies that all the DAGs of a reward-conditional flow network are identical, and only the terminating flows differ amongst the members of the family.

Example 3.4.4. *We will see in Section 3.4.4 that we can estimate a conditional flow network using a GFlowNet (Section 3.3), given a reward function $R(s \mid x)$. In an Energy-Based Model, the model $P_\theta(s)$ is associated with a given energy function $\mathcal{E}_\theta(s)$, parameterized by θ, with*

$$P_\theta(s) = \frac{\exp(-\mathcal{E}_\theta(s))}{Z(\theta)}. \tag{3.61}$$

This model can be parameterized using a reward-conditional flow network, conditioned on θ with the reward function $R(s \mid \theta) = \exp(-\mathcal{E}_\theta(s))$.

3.4.3. State-conditional flow networks

Definition 3.4.5 (State-conditional flow networks)**.** Consider a flow network given by a DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow function F. For each state $s \in \mathcal{S}$, let G_s be the subgraph of G containing all the states s' such that $s' \geq s$. A **state-conditional flow network** is given by the family $\{G_s,\ s \in \mathcal{S}\}$, along with a conditional flow function $F : \mathcal{S} \times \mathcal{T} \to \mathbb{R}^+$, where $\mathcal{T} = \bigcup_{s \in \mathcal{S}} \mathcal{T}_s$, and $\mathcal{T}_s$ the set of complete trajectories in $\mathcal{G}_s$, that satisfies:

$$F_s(s' \to s_f) = F(s' \to s_f). \tag{3.62}$$

Note that in the definition above, we abused the notation F to refer to both flow functions and edge flow functions, but also used F_s to refer to the conditional flow function (or the corresponding edge flow function) $\tau \mapsto F(s, \tau)$. Unlike the reward-conditional flow networks defined in Section 3.4.2, the structure of the DAG in a state-conditional flow network depends

on the anchor state s. In particular, this means that the initial state $(s_0 \mid s) = s$ changes, but the final state $(s_f \mid s) = s_f$ remains unchanged, for any state s.

Since the definition of a state-conditional flow network depends on an original flow network, we must ensure that this definition is indeed correct, i.e., that such a state-conditional flow network that satisfies the conditions in (3.62) exists.

Proposition 3.4.6. *For any flow network given by a DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow F, we can define a state-conditional flow network as per Definition 3.4.5.*

While we saw in Proposition 3.2.10 that the initial flow $F(s_0)$ was equal to the partition function, the initial state-conditional flow also benefit from a marginalizing property, and is now related to the free energy at s.

Proposition 3.4.7. *Given a state-conditional flow network (G, F) as in Definition 3.4.5, for any state s, the initial flow of the state-conditional flow network corresponds to marginalizing the terminating flows $F(s' \to s_f)$ for $s' \geq s$:*

$$F_s(s_0 \mid s) = F_s(s) = \sum_{s' \,:\, s' \geq s} F(s' \to s_f) = \exp(-\mathcal{F}(s)), \tag{3.63}$$

where $\mathcal{F}(s)$ is the free energy associated to the energy function $\mathcal{E}(s') = -\log F(s' \to s_f)$.

Note that the definition of state-conditional flow networks is consistent with our original definition of (unconditional) flow networks in Section 2.1, in the sense that the original flow network is a valid state-conditional flow networks anchored at the initial state.

Another quantity of interest that state-conditional flow networks allow us to evaluate, is the probability of terminating a trajectory in a state s' if all terminating edge flows were diverted towards an earlier state $s < s'$:

Corollary 3.4.8. *Consider a flow network given by a DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow F, from which we define any state-conditional flow network, as per Definition 3.4.5. Given a state s, the flow function F_s induces a probability distribution over $\{s'' \in \mathcal{S}^f : s'' \geq s\} \subseteq \mathcal{S}^f$, that we denote by $P_\top(. \mid s)$.*
Under this measure, the probability of terminating a trajectory in G_s in a state s' (i.e., the last edge of the trajectory is $s' \to s_f$) is:

$$P_\top(s' \mid s) = \mathbf{1}(s' \geq s) e^{-\mathcal{E}(s') + \mathcal{F}(s)}, \tag{3.64}$$

where $\mathcal{E}$ is the energy function mapping each state s' that is parent of s_f to $-\log F(s' \to s)$, and $\mathcal{F}$ is the corresponding free energy function.

3.4.4. Conditional GFlowNets

Similar to the way we used a GFlowNet to estimate the flow of a flow network, we can also use a (conditional) GFlowNet in order to estimate a conditional flow network, with given target reward functions. A conditional GFlowNet follows the construction presented in Section 3, with the exception that all quantities to be learned now depend on the conditioning variable $x \in \mathbb{X}$ (e.g., x is an additional input of the neural network).

Definition 3.4.9. Consider a set of conditioning information $\mathbb{X}$, a family of DAGs $\mathcal{G} = \{G_x = (\mathcal{S}_x, \mathbb{A}_x),\ x \in \mathbb{X}\}$, a family of target reward functions $\mathcal{R} = \{R_x : \mathcal{S}_x^f \to \mathbb{R}^+,\ x \in \mathbb{X}\}$, and a flow parameterization $(\mathcal{O}_x, \Pi_x, \mathcal{H}_x)$ of (G_x, R_x) for every $x \in \mathbb{X}$. The three functions $\mathcal{O} : x \in \mathbb{X} \mapsto \mathcal{O}_x$, $\Pi : x \in \mathbb{X} \mapsto \Pi_x$, and $\mathcal{H} : x \in \mathbb{X} \mapsto \mathcal{H}_x$ parameterize the family of DAGs and target reward functions, and we say that $(\mathcal{O}, \Pi, \mathcal{H})$ form a **conditional flow parameterization** of $(\mathbb{X}, \mathcal{G}, \mathcal{R})$. The tuple $(\mathbb{X}, \mathcal{G}, \mathcal{R}, \mathcal{O}, \Pi, \mathcal{H})$ is called a **conditional GFlowNet**.
For clarity, for an object $o \in \mathcal{O}$, we will use o_x and $o(x)$ interchangeably.

All parameterizations and losses presented in Section 3.3.1 could, in principle be used to train a conditional GFlowNet, regardless of the conditioning set. Below we discuss yet another loss, first presented in Deleu et al. (2022), that could be used to train both GFlowNets and conditional GFlowNets.

Example 3.4.10. *Given a family of DAGs G_x and reward functions R_x indexed by $x \in \mathbb{X}$, where each state $s \in G_x$ is terminating (i.e., is a parent of s_f), and following Examples 3.3.3 and 3.3.4, we consider a parameterization given by the forward and backward transition probabilities $\mathcal{O}_P^x = \mathcal{O}_2^x \times \mathcal{O}_3^x$, where $\mathcal{O}_2^x$ (resp. $\mathcal{O}_3^x$) is the set of forward (resp. backward) probability functions $\hat{P}_F$ (resp. $\hat{P}_B$) consistent with G_x for every $x \in \mathbb{X}$, and $(\Pi_P^x, \mathcal{H}_P^x)$ defined as in Example 3.3.4. Each $(\mathcal{O}_P^x, \Pi_P^x, \mathcal{H}_P^x)$ is a flow parameterization of (G_x, R_x), which can be trained with an edge-decomposable flow-matching loss, as proved in Deleu et al. (2022), and defined for every $s \to s' \in \mathbb{A}^{-f}$:*

$$\mathcal{L}_{DB}(\hat{P}_F, \hat{P}_B, s \to s', x) = \left(\log \frac{R_x(s')P_B(s \mid s', x)P_F(s_f \mid s, x)}{R_x(s)P_F(s' \mid s, x)P_F(s_f \mid s', x)}\right)^2 \tag{3.65}$$

In Appendix D.1, we show how conditional GFlowNets can be used to estimate entropies, conditional entropies, and the mutual information between two random variables.

3.5. GFlowNets are more than amortized samplers

3.5.1. GFlowNets as amortized samplers

The main established methods to approximately sample from the distribution associated with an energy function $\mathcal{E}$, or an unnormalized probability mass function or reward R, are MCMC methods, already discussed in Section 2.2.2.

Instead, the GFlowNet approach amortizes upfront computation to train a generator that yields efficient computation for each new sample without requiring any chain.

In contrast, GFlowNets belong to the family of **amortized** sampling methods (which includes VAEs, Kingma and Welling, 2013), where we train a machine learning system to produce samples: we have exchanged the complexity of sampling through long chains for the complexity of training the sampler.

For example, Bengio et al. (2021) build a GFlowNet that constructs a molecule via a small sequence of actions, each adding an atom or a molecular substructure to an existing molecule represented by a graph, starting from an empty graph. Only one such configuration needs to be considered, in contrast with MCMC methods, which require potentially very long chains of such configurations, and suffer from the challenge of mode-mixing (Jasra et al., 2005; Bengio et al., 2013; Pompe et al., 2020): the chances of going from one mode to a neighbouring one may become exponentially small (and thus require exponentially long chains) if a long sequence of low-probability configurations separates the modes. This can be alleviated by burning more computation (sampling longer chains) but becomes exponentially unsustainable with increased mode separation. The issue can also be reduced by introducing random sampling (e.g., drawing multiple chains) and simulated annealing (Andrieu et al., 2003) to facilitate jumping between modes. However, this becomes less effective in high dimensions and when the modes occupy a tiny volume, which can become an exponentially small fraction of the total space as its dimension increases, since random sampling is unlikely to land in the neighbourhood of a mode.

The potential advantage of such amortized samplers is when the distribution of interest has a generalizable structure: when it is possible to guess reasonably well where high-probability samples can be found based on the knowledge of a set of known high-probability samples (the training set). Consider, for instance, having already constructed some configurations x and obtained their unnormalized probability or reward $R(x)$. With these pairs $(x, R(x))$, a machine learning system could potentially generalize about the value of R elsewhere, and if it is a generative model, sample new x's in places of large $R(x)$. Hence, if there is an underlying statistical structure in how the modes of R are related to each other, a generative learner that generalizes could guess the presence of modes it has not visited yet,

taking advantage of the patterns it has already uncovered from the $(x, R(x))$ pairs it has seen.

Amortization is what makes deep generative models (Sections 2.1.4 and 3.5.2) work in the first place and thus suggests that in such high-dimensional settings where modes occupy tiny volumes (as per the manifold hypothesis, Cayton, 2005; Narayanan and Mitter, 2010; Rifai et al., 2011), one can capitalize on the already observed $(x, R(x))$ pairs (where x is an already visited configuration and $R(x)$ its reward) to "jump" from known modes to yet unvisited ones, even if these are far from the ones already visited.

In the case of no structure (and thus no possibility to generalize when learning about the distribution), there is no reason to expect that amortized ML methods will fare better than MCMC.

Another factor to consider (independent of the mode mixing issue) is the amortization of the computational costs: GFlowNets pay a hefty price upfront to train the network and then a tiny price (sampling once from $P_\top$) to generate each new sample. Instead, MCMC has no upfront cost but pays a lot for each independent sample. Hence, if we want to only sample once, MCMC may be more efficient, whereas if we want to generate a lot of samples, amortized methods may be advantageous. One can imagine settings where GFlowNets and MCMC could be combined to achieve some of the advantages of both approaches.

3.5.2. GFlowNets as generative models

The main difference between GFlowNets and established deep generative models is that whereas the latter are trained by being provided with a finite set of examples sampled from the distribution of interest, a GFlowNet is usually trained by being provided with an energy function or a reward function.

This reward function tells us not just about the samples that are likely under the distribution of interest but also about those that are unlikely and those in-between, whose reward is not large but is not zero. Let's think of the maximum likelihood training objective in those terms. It is like a reward function that gives a high reward to every training example and a zero reward everywhere else. However, other reward functions are possible, as seen in the application of GFlowNets to the discovery of new molecules (Bengio et al., 2021), where the reward is not binary and increases monotonically as a function of the value of a desirable property of the candidate molecule.

Note, however, that the difference with other generative modelling approaches blurs when we include the learning of the energy function along with the learning of the GFlowNet sampler, as is done in Zhang et al. (2022b), who simultaneously train an energy-based model and a GFlowNet: the energy function is trained with samples from a GFlowNet, which, in turn, uses the energy function to form its reward. Their method results in a generative model

for binary vectors in high dimensions, e.g., binarized digits. More specifically, consider the model $P_\theta(s)$ associated with a given parameterized energy function $\mathcal{E}_\theta(s)$ with parameters θ: $P_\theta(s) = \frac{e^{-\mathcal{E}_\theta(s)}}{Z}$. Sampling from $P_\theta(s)$ could be approximated by sampling from the terminating probability distribution $P_\top(s)$ of a GFlowNet trained with target terminal reward $R(s) = e^{-\mathcal{E}_\theta(s)}$ (see (3.31)). In practice, $\hat{P}_T$ would be an estimator for the true P_θ because the GFlowNet training objective is not zeroed (insufficient capacity or finite training time). The GFlowNet samples drawn according to $\hat{P}_T$ could then be used to obtain a stochastic gradient estimator for the negative log-likelihood of observed data x with respect to parameters θ of an energy function $\mathcal{E}_\theta$:

$$\frac{\partial - \log P_\theta(x)}{\partial \theta} = \frac{\partial \mathcal{E}_\theta(x)}{\partial \theta} - \sum_s P_\theta(s) \frac{\partial \mathcal{E}_\theta(s)}{\partial \theta}. \tag{3.66}$$

An approximate stochastic estimator of the second term could thus be obtained by sampling one or more terminating states $s \sim \hat{P}_T(s)$, i.e., from the trained GFlowNet's sampler. Furthermore, if the GFlowNet's loss is 0, i.e., $\hat{P}_T = P_\theta$, the gradient estimator would be unbiased.

Zhang et al. (2022b) jointly train an energy function $\mathcal{E}_\theta$ and a corresponding GFlowNet by alternating updates of θ using the above equation (with sampling from P_θ replaced by sampling from $\hat{P}_T$) and updates of the GFlowNet using the updated energy function for the target terminal reward.

Note, however, that GFlowNets have been designed for generating discrete variable-size compositional structures (like sets or graphs), for both latent and observed variables, whereas many generative models start from the point of view of modelling real-valued fixed-size vectors using real-valued fixed-size latent variables. The extension of GFlowNets to continuous domains (Chapter 4) was only introduced later on.

An interesting difference between GFlowNets and most generative model training frameworks is in the very nature of the training objective for GFlowNets, which came about in the context of active learning scenarios. Whereas the GFlowNet training pairs $(s, R(s))$ can come from any distribution over s (any full-support training policy π_T), which does not have to be stationary (and indeed will generally not be, in an active learning setting), the maximum likelihood framework is very sensitive to changes in the distribution of the data it sees. This is connected to the "offline learning" property of the flow-matching objective (Sections 3.3.3.3 and 3.3.3.4)).

3.5.3. GFlowNets for interactive learning

An interesting variant on Section 3.5.2's scheme is one where the GFlowNet sampler is used not just to produce negative examples for the energy function but also to actively explore the environment. Jain et al. (2022a) use an active learning scheme where the GFlowNet is

used to sample candidates x for which we expect the reward $R(x)$ to be generally large (since the GFlowNet approximately samples proportionally to $R(x)$). The challenge is that evaluating the true reward R for any x is computationally expensive and can potentially be noisy (for example, a biological assay to measure the binding energy of a drug to a given target protein). Thus, instead of using the true reward directly, the authors introduce a proxy $\hat{f}$ (which approximates the true reward function f), which is used to train the GFlowNet. This would lead to a setup similar to Section 3.5.2, with an inner loop where a GFlowNet is trained to match the proxy $\hat{f}$ and an outer loop where the proxy $\hat{f}$ is learned in a supervised fashion using (x, y) pairs, where x is proposed by the GFlowNet, and y is the corresponding *true* reward from the environment (for example, outcome of a biological of chemical assay). It is important to note here that the GFlowNet and the proxy are intricately linked since the coverage of proxy $\hat{f}$ over the domain of x relies on diverse candidates from the GFlowNet. And similarly, since the GFlowNet matches a reward distribution defined by the proxy reward function $\hat{f}$, it also depends on the quality of the true reward function f.

This setup can be further extended by incorporating information about how *novel* a given candidate is, or how much epistemic uncertainty, $u(x, f)$, there is in the prediction of $\hat{f}$ (Section 2.1.7). We can use the acquisition function heuristics (like Upper Confidence Bound (UCB) or Expected Improvement (EI)) from Bayesian optimization (Močkus, 1975; Srinivas et al., 2010) to combine the predicted usefulness $\hat{f}(x)$ of configuration x with an estimate of the epistemic uncertainty around that prediction. Using this as the reward can allow the GFlowNet to explore areas where the predicted usefulness is high ($\hat{f}(x)$ is large) and, at the same time, explore areas where there is more information to be gathered about useful configurations of x. The uncertainty over the predictions of $\hat{f}$ with the appropriate acquisition function can provide more control over the exploratory behaviour of GFlowNets.

As discussed by Bengio et al. (2021) when comparing GFlowNets with return-maximizing reinforcement learning methods, an interesting property of sufficiently trained GFlowNets is that they will sample from all the modes of the reward function, which is particularly desirable in a setting where exploration is required, as in active learning. The experiments in the paper also demonstrate this advantage experimentally in terms of the diversity of the solutions sampled by the GFlowNet compared with PPO, an RL method that had previously been used for generating molecular graphs and that tends to focus on a single mode of the reward function.

3.5.4. GFlowNets as an alternative to Reinforcement Learning

The flow-matching loss of GFlowNets (Bengio et al., 2021) arose from the inspiration of the temporal-difference training (Sutton and Barto, 2018) objectives associated with the Bellman equation. The flow-matching equations are analogous to the Bellman equation in the

sense that the training objective is local (in time and states), credit assignment propagates through a bootstrap process and tries to fix the parameterization so that these equations are satisfied, knowing that if they were (everywhere), we would obtain the desired properties. However, these desired properties are different, as elaborated in the next paragraph. The context in which GFlowNets were developed is also different from the typical way of thinking about agents learning in some environment: we can think of the deterministic environments of GFlowNets as involving internal actions typically needed by a cognitive agent that needs to perform some kind of inference through a sequence of steps (predict or sample some things given other things), i.e., through actions internal to the agent and controlling its computation. This is in contrast with the origins of RL, focused on the actions of an agent in an external and unknown stochastic environment.

An alternative perspective on RL than that provided in Section 2.1.5, that emerged out of both the probabilistic inference literature (Toussaint and Storkey, 2006) and the bandits literature (Auer et al., 2002), is concerned with finding policies of the form $\pi(a \mid s) \propto f(s, a)$. It turns out that maximizing both return and *entropy* (Ziebart et al., 2008) of policies in a control setting yields policies such that the probability of a trajectory $\tau = (s_0 a_0, s_1, a_1, \ldots, s_T)$ is

$$p(\tau) = \left[p(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t) \right] \exp\left(\eta \sum_{t=0}^{T-1} R(s_t, a_t) \right), \tag{3.67}$$

where η can be seen as a temperature parameter. This result can also be found under the control-as-inference framework (Haarnoja et al., 2017; Levine, 2018). In deterministic MDPs with terminal rewards and no discounting of future rewards, this simplifies to $p(\tau) \propto \exp(\eta\rho(\tau))$, where ρ is the return.

In recent literature, this entropy maximization (MaxEnt) is often interpreted as a regularization scheme (Nachum et al., 2017), entropy being used either as an intrinsic reward signal or as an explicit regularization objective to be maximized. Another way to understand this scheme is to imagine ourselves in an adversarial bandit setting (Auer et al., 2002) where each arm corresponds to a unique trajectory, drawn with probability $\propto \exp(\rho(\tau))$.

An important distinction to make between MaxEnt RL and GFlowNets is that, in the general case, they do not find the same result. A GFlowNet learns a policy such that $P_\top(s) \propto R(s)$, whereas MaxEnt RL (with appropriately chosen temperature and R) learns a policy such that $P_\top(s) \propto n(s)R(s)$, where $n(s)$ is the number of paths in the DAG of all trajectories that lead to s (a proof is provided in Bengio et al., 2021). An equivalence only exists if the DAG minus s_f is a tree rooted at s_0, which has been found to be useful (Buesing et al., 2019). What this overweighting by a factor $n(s)$ means practically is that states corresponding to longer sequences (which typically will have exponentially more paths to them) will tend to be sampled much more often (typically exponentially more often) than states corresponding

to shorter sequences. Clearly, this breaks the objective of sampling terminating states in proportion to their reward and provides a strong motivation for considering GFlowNets instead.

Another perspective on maximizing entropy in RL is that one can also maximize entropy on the states' *stationary distribution* d^π (Ziebart et al., 2008), rather than the policy. In fact, one can show that the objective of training a policy such that $P_\top(s) \propto R(s)$ is equivalent to training a policy that maximizes $r(s,a) = \log R(s,a) - \log d^\pi(s,a)$. Unfortunately, computing stationary distributions, although possible (Nachum et al., 2019; Wen et al., 2020a), is not always tractable nor precise enough for purposes of reward regularization.

3.6. GFlowNets and Variational Inference

Many probabilistic generative models produce a sample through a sequence of stochastic choices. Non-neural latent variable models (e.g., Blei et al., 2003), autoregressive models, hierarchical variational autoencoders (Sønderby et al., 2016), and diffusion models (Ho et al., 2020b) can be said to rely upon a shared principle: richer distributions can be modelled by chaining together a sequence of simple actions, whose conditional distributions are easy to describe, than by performing generation in a single sampling step. When many intermediate sampled variables could generate the same object, making exact likelihood computation intractable, hierarchical models are trained with variational objectives that involve the posterior over the sampling sequence (Ranganath et al., 2016b).

In this section and in the remainder of this chapter, we connect variational inference (VI) methods for hierarchical models (i.e., sampling through a sequence of choices conditioned on the previous ones) with GFlowNets. Although GFlowNets appear to have different foundations (Sections 3.1 to 3.5) and applications than hierarchical VI algorithms, we will show here that the two are closely connected.

Expanding on Section 2.2.1, VI (Zhang et al., 2019a) techniques originate from graphical models (Saul et al., 1996; Jordan et al., 1999), which typically include an inference machine and a generative machine to model the relationship between latent variables and observed data. The line of work on black-box VI (Ranganath et al., 2014) focuses on learning the inference machine given a data-generating process, i.e., inferring the posterior over latent variables. Hierarchical modelling exhibits appealing properties under such settings as discussed in Ranganath et al. (2016b); Yin and Zhou (2018); Sobolev and Vetrov (2019). On the other hand,VAEs (Kingma and Welling, 2014b; Rezende et al., 2014a) focus on generative modelling, where the inference machine – the estimated variational posterior – is a tool to assist the optimization of the generative machine or decoder. Hierarchical construction of multiple latent variables has also been shown to be beneficial (Sønderby et al., 2016; Maaløe et al., 2019; Child, 2021).

While earlier works simplify the variational family with mean-field approximations (Bishop, 2006a), modern inference methods rely on amortized stochastic optimization (Hoffman et al., 2013). One of the oldest and most commonly used ideas is REINFORCE (Williams, 1992; Paisley et al., 2012) which gives unbiased gradient estimation. Follow-up work (Titsias and Lázaro-Gredilla, 2014; Gregor et al., 2014; Mnih and Gregor, 2014; Mnih and Rezende, 2016) proposes advanced estimators to reduce the high variance of REINFORCE. The log-variance loss proposed by Richter et al. (2020) is equivalent, in terms of the expected gradient of P_F, to the on-policy TB loss for a GFlowNet with a batch-optimal value of $\log Z$. On the other hand, path-wise gradient estimators (Kingma and Welling, 2014b) have much lower variance but have limited applicability. Later works combine these two approaches for particular distribution families (Tucker et al., 2017; Grathwohl et al., 2018).

Beyond the evidence lower bound (ELBO) objective used in most variational inference methods, more complex objectives have been studied. Tighter evidence bounds have proved beneficial to the learning of generative machines (Burda et al., 2016; Domke and Sheldon, 2018; Rainforth et al., 2018; Masrani et al., 2019). As KL divergence optimization suffers from issues such as mean-seeking behaviour and posterior variance underestimation (Minka et al., 2005), other divergences are adopted as in expectation propagation (Minka, 2001; Li et al., 2015), more general f-divergences (Dieng et al., 2017; Wang et al., 2018; Wan et al., 2020), their special case α-divergences (Hernández-Lobato et al., 2016), and Stein discrepancy (Liu and Wang, 2016; Ranganath et al., 2016a). GFlowNets could be seen as providing a novel pseudo-divergence criterion, namely the trajectory-balance (TB) loss, as discussed below.

Another branch of work, called **wake-sleep**, starting with Hinton et al. (1995), proposes to avoid issues from stochastic optimization (such as REINFORCE) by alternatively optimizing the generative and inference (posterior) models. Modern versions extending this framework include reweighted wake-sleep Bornschein and Bengio (2015); Le et al. (2019) and memoized wake-sleep (Hewitt et al., 2020; Le et al., 2022). It was shown in Le et al. (2019) that wake-sleep algorithms behave well for tasks involving stochastic branching.

As our main theoretical contributions of the remainder of this chapter, we show in Section 3.7 that special cases of variational algorithms and GFlowNets coincide in their expected gradients. In particular, hierarchical VI (Ranganath et al., 2016b) and nested VI (Zimmermann et al., 2021) are related to the trajectory balance and detailed balance objectives for GFlowNets (Malkin et al., 2022; Bengio et al., 2023). We also point out the differences between VI and GFlowNets: notably, that GFlowNets automatically perform gradient variance reduction by estimating a marginal quantity (the partition function) that acts as a baseline and allow off-policy learning without the need for reweighted importance sampling.

Our theoretical results are accompanied by experiments, in Section 3.8, that examine what similarities and differences emerge when one applies hierarchical VI algorithms to discrete problems where GFlowNets have been used before. These experiments serve two purposes. First, they supply a missing hierarchical VI baseline for problems where GFlowNets have been used in past work. The relative performance of this baseline illustrates the aforementioned similarities and differences between VI and GFlowNets. Second, the experiments demonstrate the ability of GFlowNets, not shared by hierarchical VI, to learn from off-policy distributions without introducing high gradient variance. We show that this ability to learn with exploratory off-policy sampling is beneficial in discrete probabilistic modelling tasks, especially in cases where the target distribution has many modes.

Note that a connection of the theoretical foundations of GFlowNets (Bengio et al., 2021, 2023) with variational methods was first mentioned by Malkin et al. (2022) and expanded in Zhang et al. (2022a).

A closely related paper (Zimmermann et al., 2022), published a few days after Malkin et al. (2023) (upon which this section and the remainder of this chapter is based), theoretically and experimentally explores interpolations between forward and reverse KL objectives.

3.7. Theoretical analysis of the relation between GFlowNets and Hierarchical Variational Inference

All lemmas and propositions of this section are proved in Appendix E.3.

3.7.1. GFlowNets: Notation and background

We consider the setting studied thus far, where we are given a pointed DAG $\mathcal{G} = (\mathcal{S}, \mathbb{A})$, a subset $\mathcal{X}$ of $\mathcal{S}$ called the set of terminating states, an unnormalized density or reward function, $R : \mathcal{X} \to \mathbb{R}^+$, and we are tasked with learning a functional approximation of the target distribution $P_\top \propto R$. While there exist different parameterizations and loss functions for GFlowNets (Section 3.3.1), they all define a *forward transition probability function*, or a *forward policy*, $P_F(- \mid s)$, which is a distribution over the children of every state $s \in \mathcal{S}$. The forward policy is typically parameterized by a neural network that takes a representation of s as input and produces the logits of a distribution over its children. Any forward policy P_F induces a distribution over complete trajectories $\tau \in \mathcal{T}$ (denoted by P_F as well), which in

turn defines a marginal distribution over terminating states $x \in \mathcal{X}$ (denoted by $P_\top$):

$$P_F(\tau = (s_0 \to \ldots \to s_n \to s_{n+1} = s_f)) = \prod_{i=0}^{n} P_F(s_{i+1} \mid s_i) \qquad \forall \tau \in \mathcal{T}, \tag{3.68}$$

$$P_\top(x) = \sum_{\tau \in \mathcal{T} : x_\tau = x} P_F(\tau) \qquad \forall x \in \mathcal{X}, \tag{3.69}$$

where we use $x_\tau \in \mathcal{X}$ to denote the terminating state of a trajectory τ.

Given a forward policy P_F, terminating states $x \in \mathcal{X}$ can be sampled from $P_\top$ by sampling trajectories τ from $P_F(\tau)$ and taking their final states x_τ, as in Algorithm 1.

GFlowNets aim to find a forward policy P_F for which $P_\top(x) \propto R(x)$. Because the sum in (3.69) is typically intractable to compute exactly, training objectives for GFlowNets introduce auxiliary objects into the optimization. For example, the trajectory balance objective (TB; Malkin et al., 2022) introduces an auxiliary *backward policy* P_B, which is a learned distribution $P_B(- \mid s)$ over the *parents* of every state $s \in \mathcal{S}$, and an estimated partition function Z, typically parameterized as $\exp(\log Z)$ where $\log Z$ is the learned parameter. The TB objective for a complete trajectory τ is defined as

$$\mathcal{L}_{\text{TB}}(\tau; P_F, P_B, Z) = \left(\log \frac{Z \cdot P_F(\tau)}{R(x_\tau) P_B(\tau \mid x_\tau)}\right)^2, \tag{3.70}$$

where

$$P_B(\tau \mid x_\tau) = \prod_{(s \to s') \in \tau, s' \neq s_f} P_B(s \mid s'). \tag{3.71}$$

If $\mathcal{L}_{\text{TB}}$ is made equal to 0 for every complete trajectory τ, then $P_\top(x) \propto R(x)$ for all $x \in \mathcal{X}$ and Z is the inverse constant of proportionality: $Z = \sum_{x \in \mathcal{X}} R(x)$.

The objective (3.70) is minimized by sampling trajectories τ from some distribution and making gradient steps on (3.70) with respect to the parameters of P_F, P_B, and $\log Z$. The distribution from which τ is sampled amounts to a choice of scalarization weights for the multi-objective problem of minimizing (3.70) over all $\tau \in \mathcal{T}$. If τ is sampled from $P_F(\tau)$ – note that this is a nonstationary scalarization – we say the algorithm runs *on-policy*. If τ is sampled from another distribution, the algorithm runs *off-policy*; typical choices are to sample τ from a tempered version of P_F to encourage exploration (Bengio et al., 2021; Deleu et al., 2022) or to sample τ from the backward policy $P_B(\tau \mid x)$ starting from given terminating states x (Zhang et al., 2022b). By analogy with the RL nomenclature, we call the *behaviour policy* the one that samples τ for the purpose of obtaining a stochastic gradient, e.g, the gradient of the objective $\mathcal{L}_{\text{TB}}$ in (3.70) for the sampled τ.

We also consider here the detailed balance (DB) loss that parameterizes a GFlowNet using its forward and backward policies P_F and P_B, respectively, along with a state flow function F, which is a positive function of the states, that matches the target reward function on the

terminating states. It decomposes as a sum of transition-dependent losses:

$$\forall s \to s' \in \mathbb{A}^{-f} \quad \mathcal{L}_{\text{DB}}(s \to s'; P_F, P_B, F) = \left(\log \frac{F(s)P_F(s' \mid s)}{F(s')P_B(s \mid s')}\right)^2. \tag{3.72}$$

Both the DB and TB objectives can be seen as special instances of the subtrajectory balance (SubTB; Malkin et al., 2023; Madan et al., 2022b). Malkin et al. (2022) suggested instead of defining the state flow function F for every state s, a state flow function could be defined on a subset of the state space $\mathcal{S}$, called the *hub states*. The loss can be decomposed into a sum of subtrajectory-dependent losses:

$$\forall \tau = (s_1, \ldots, s_n) \in \mathcal{T}^{\text{partial}} \quad \mathcal{L}_{\text{SubTB}}(\tau; P_F, P_B, F) = \left(\log \frac{F(s_1)P_F(\tau)}{F(s_n)P_B(\tau \mid s_t)}\right)^2, \tag{3.73}$$

where $P_F(\tau)$ is defined for partial trajectories similarly to complete trajectories (3.69), $P_B(\tau \mid s) = \prod_{(s \to s') \in \tau} P_B(s \mid s')$, and we again fix $F(x) = R(x)$ for terminating states $x \in \mathcal{X}$). The SubTB objective reduces to the DB objective for subtrajectories of length 1 and to the TB objective for complete trajectories, in which case we use Z to denote $F(s_0)$.

In the next sections, we will show how the TB objective relates to hierarchical variational objectives and, more generally, how the SubTB loss relates to a special form of hierarchical variational losses.

3.7.2. Hierarchical variational models and GFlowNets

Variational methods provide a way of sampling from distributions by means of learning an approximate probability density. Hierarchical variational models (HVMs; Ranganath et al., 2016b; Sobolev and Vetrov, 2019; Vahdat and Kautz, 2020; Zimmermann et al., 2021)) typically assume that the sample space is a set of sequences $(z_1, \ldots, z_n)$ **of fixed length**, with an assumption of conditional independence between z_{i-1} and z_{i+1} conditioned on z_i, i.e., the likelihood has a factorization $q(z_1, \ldots, z_n) = q(z_1)q(z_2 \mid z_1) \ldots q(z_n \mid z_{n-1})$. The marginal likelihood of z_n in a hierarchical model involves a possibly intractable sum,

$$q(z_n) = \sum_{z_1, \ldots, z_{n-1}} q(z_1)q(z_2 \mid z_1) \ldots q(z_n \mid z_{n-1}). \tag{3.74}$$

The goal of VI algorithms is to find the conditional distributions q that minimize some divergence between the marginal $q(z_n)$ and a target distribution. The target is often given as a distribution with intractable normalization constant: a typical setting is a Bayesian posterior (used in VAEs, variational EM, and other applications), for which we desire $q(z_n) \propto p_{\text{likelihood}}(x \mid z_n)p_{\text{prior}}(z_n)$.

Before delving deeper into the connections, we need to define **graded DAGs**.

Definition 3.7.1 (Graded DAG)**.** A pointed graded DAG is a pointed DAG in which all complete trajectories have the same length. Pointed graded DAGs $\mathcal{G}$ are also characterized by the following equivalent property: the state space $\mathcal{S}$ can be partitioned into disjoint sets $\mathcal{S} = \bigsqcup_{l=0}^{L+1} \mathcal{S}_l$, with $\mathcal{S}_0 = \{s_0\}$, $\mathcal{S}_L = \mathcal{X}$ and $\mathcal{S}_{L+1} = \{s_f\}$, called *layers*, such that all edges $s \to s'$ are between states of adjacent layers ($s \in \mathcal{S}_i$,$s' \in \mathcal{S}_{i+1}$ for some i).

The GFlowNet corresponding to a HVM:. Sampling sequences $(z_1, \ldots, z_n)$ from a hierarchical model is equivalent to sampling complete trajectories in a certain pointed graded DAG $\mathcal{G}$. The states of $\mathcal{G}$ at a distance of i from the initial state are in bijection with possible values of the variable z_i, and the action distribution is given by q. Sampling from the HVM is equivalent to sampling trajectories from the policy $P_F(z_{i+1} \mid z_i) = q(z_{i+1} \mid z_i)$ (and $P_F(z_1 \mid s_0) = q(z_1)$, $P_F(s_f \mid z_n) = 1$), and the marginal distribution $q(z_n)$ is the terminating distribution $P_\top$. Only the states at a distance n are connected to the sink node s_f, and only have s_f as a child.

The HVM corresponding to a GFlowNet: Conversely, suppose $\mathcal{G} = (\mathcal{S}, \mathbb{A})$ is a graded pointed DAG with $L + 2$ layers and that a forward policy P_F on $\mathcal{G}$ is given. Sampling trajectories $\tau = (s_0 \to s_1 \to \ldots \to s_L \to s_{L+1} = s_f)$ in $\mathcal{G}$ is equivalent to sampling from an HVM in which the random variable z_i is the identity of the $(i+1)$-th state s_i in τ and the conditional distributions $q(z_{i+1} \mid z_i)$ are given by the forward policy $P_F(s_{i+1} \mid s_i)$. Specifying an approximation of the target distribution in a hierarchical model with L layers is thus equivalent to specifying a forward policy P_F in a graded DAG.

The correspondence can be extended to non-graded DAGs. Every pointed DAG $\mathcal{G} = (\mathcal{S}, \mathbb{A})$ can be canonically transformed into a graded pointed DAG by the insertion of dummy states that have one child and one parent. To be precise, every edge $s \to s' \in \mathbb{A}$ is replaced with a sequence of $\ell' - \ell(s)$ edges, where $\ell(s)$ is the length of the *longest* trajectory from s_0 to s, $\ell' = \ell(s')$ if $s' \notin \mathcal{X}$, and $\ell' = \max_{x \in \mathcal{X}} \ell(s'')$ otherwise. This process is illustrated in Figure 3.7. We thus restrict our analysis in this section, without loss of generality, to graded DAGs.

The meaning of the backward policy: Typically, the target distribution is over the objects $\mathcal{X}$ of the last layer of a graded DAG, rather than over complete sequences or trajectories. Any backward policy P_B on the DAG turns an unnormalized target distribution R over $\mathcal{X}$ into an unnormalized distribution over complete trajectories $\mathcal{T}$:

$$\forall \tau \in \mathcal{T} \quad P_B(\tau) \propto R(x_\tau) P_B(\tau \mid x_\tau), \quad \text{with unknown partition function } \hat{Z} = \sum_{x \in \mathcal{X}} R(x). \tag{3.75}$$

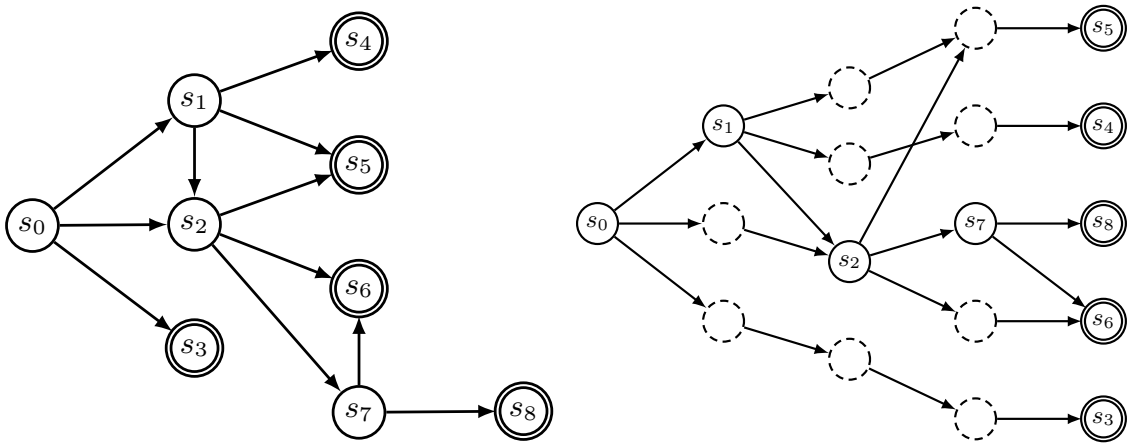


Figure 3.7 – Illustration of the process by which a DAG (left) can turn into a graded DAG (right). For simplicity, the sink node s_f is omitted. It is the only child of the terminating states (those with a double border). Nodes with a dashed border represent dummy states added to make the DAG graded. Note that the operation of converting a DAG into a graded one is idempotent: applying it to a graded DAG yields the same graded DAG.

The marginal distribution of P_B over terminating states is equal to $R(x)/\hat{Z}$ by construction. Therefore, if P_F is a forward policy that equals P_B as a distribution over trajectories, then $P_\top(x) = R(x)/\hat{Z} \propto R(x)$.

VI training objectives: In its most general form, the hierarchical variational objective ('HVI objective' in the remainder of the paper) minimizes a statistical divergence D_f between the learned and the target distributions over trajectories:

$$\mathcal{L}_{\text{HVI},f}(P_F, P_B) = D_f(P_B \| P_F) = \mathbb{E}_{\tau \sim P_F}\left[f\left(\frac{P_B(\tau)}{P_F(\tau)}\right)\right]. \tag{3.76}$$

Two common objectives are the forward and reverse Kullback-Leibler (KL) divergences (Mnih and Gregor, 2014), corresponding to $f : t \mapsto t \log t$ for $D_{\text{KL}}(P_B \| P_F)$ and $f : t \mapsto -\log t$ for $D_{\text{KL}}(P_F \| P_B)$, respectively. Other f-divergences have been used, as discussed in Zhang et al. (2019b); Wan et al. (2020). Note that, similar to GFlowNets, (3.76) can be minimized with respect to both the forward and backward policies or can be minimized using a fixed backward policy.

Divergences between two distributions over trajectories and divergences between their two marginal distributions over terminating states distributions are linked via the data processing inequality, assuming f is convex (see, e.g., Zhang et al. (2019b)), making the former a sensible surrogate objective for the latter:

$$D_f(R/\hat{Z} \| P_\top) \leq D_f(P_B \| P_F) \tag{3.77}$$

When both P_B and P_F are learned, the divergences with respect to which they are optimized need not be the same, as long as both objectives are 0 if and only if $P_F = P_B$. For

	Surrogate loss	
Algorithm	P_F (sampler)	P_B (posterior)
Reverse KL	$D_{\mathrm{KL}}(P_F \Vert P_B)$	$D_{\mathrm{KL}}(P_F \Vert P_B)$
Forward KL	$D_{\mathrm{KL}}(P_B \Vert P_F)$	$D_{\mathrm{KL}}(P_B \Vert P_F)$
Wake-sleep (WS)	$D_{\mathrm{KL}}(P_B \Vert P_F)$	$D_{\mathrm{KL}}(P_F \Vert P_B)$
Reverse wake-sleep	$D_{\mathrm{KL}}(P_F \Vert P_B)$	$D_{\mathrm{KL}}(P_B \Vert P_F)$
On-policy TB	$D_{\mathrm{KL}}(P_F \Vert P_B)$	see Section 3.7.5

Table 3.1 – A comparison of algorithms for approximating a target distribution in a hierarchical variational model or a GFlowNet. The gradients used to update the parameters of the sampling distribution and of the auxiliary backward policy approximate the gradients of various divergences between distributions over trajectories.

example, wake-sleep algorithms (Hinton et al., 1995) optimize the generative model P_F using $D_{\mathrm{KL}}(P_B \Vert P_F)$ and the posterior P_B using $D_{\mathrm{KL}}(P_F \Vert P_B)$. A summary of common combinations is shown in Table 3.1.

We remark that tractable unbiased gradient estimators for objectives such as (3.76) may not always exist, as we cannot exactly sample from or compute the density of $P_B(\tau)$ when its normalization constant $\hat{Z}$ is unknown. For example, while the REINFORCE estimator gives unbiased estimates of the gradient with respect to P_F when the objective is Reverse KL (see Section 3.7.5), other objectives, such as Forward KL, require importance-weighted estimators. Such estimators approximate sampling from P_B by sampling a batch of trajectories $\{\tau_i\}$ from another distribution π (which may equal P_F) and weighting a loss computed for each τ_i by a scalar proportional to $\frac{P_B(\tau_i)}{\pi(\tau_i)}$. Such *reweighted importance sampling* is helpful in various variational algorithms, despite its bias when the number of samples is finite (e.g., Bornschein and Bengio, 2015; Burda et al., 2016), but it may also introduce variance that increases with the discrepancy between P_B and π.

3.7.3. Nested variational inference

From now on, we work with a graded DAG $\mathcal{G} = (\mathcal{S}, \mathbb{A})$, in which the state space $\mathcal{S}$ is decomposed into *layers*: $\mathcal{S} = \bigsqcup_{l=0}^{L+1} \mathcal{S}_l$, with $\mathcal{S}_0 = \{s_0\}$ and $\mathcal{S}_L = \mathcal{X}$ [4].

HVI provides a class of algorithms to learn forward and backward policies on $\mathcal{G}$. Rather than learning these policies (P_F and P_B) using a variational objective requiring distributions over complete trajectories, nested variational inference (NVI; Zimmermann et al., 2021)), which combines nested importance sampling and variational inference, defines an objective dealing with distributions over transitions, or edges. To this end, it makes use of positive functions F_k of the states $s_k \in \mathcal{S}_k$, for $k = 0, \ldots, L-1$, to define two sets of distributions $\check{p}_k$

4. $\mathcal{S}_{L+1}$ always corresponds to the singleton $\{s_f\}$, and can be omitted for clarity and conciseness.

and $\hat{p}_k$ over edges from $\mathcal{S}_k$ to $\mathcal{S}_{k+1}$:

$$\hat{p}_k(s_k \to s_{k+1}) \propto F_k(s_k) P_F(s_{k+1} \mid s_k) \quad \check{p}_k(s_k \to s_{k+1}) \propto \begin{cases} R(s_L) P_B(s_k \mid s_L) & k = L-1 \\ F_{k+1}(s_{k+1}) P_B(s_k \mid s_{k+1}) & \text{otherwise} \end{cases}. \tag{3.78}$$

Learning the policies P_F, P_B and the functions F_k is done by minimizing losses of the form:

$$\mathcal{L}_{\text{NVI}}(P_F, P_B, F) = \sum_{k=0}^{L-1} D_f(\check{p}_k \| \hat{p}_k) \tag{3.79}$$

The positive function F_k plays the same role as the state flow function in GFlowNets (in the DB objective in particular). Before drawing the links between DB and NVI, we first propose a natural extension of NVI to subtrajectories.

3.7.4. A variational objective for subtrajectories

Consider a graded DAG $\mathcal{G} = (\mathcal{S}, \mathbb{A})$ where $\mathcal{S} = \bigsqcup_{l=0}^{L+1} \mathcal{S}_l$, $\mathcal{S}_0 = \{s_0\}$, $\mathcal{S}_L = \mathcal{X}$. Amongst the $L+1$ layers $l = 0, \dots, L$, we consider $K+1 \leq L+1$ special layers, that we call *junction layers*, of which the states are called *hub states*. We denote by $m_0, \dots, m_K$ the indices of these layers, and we constrain $m_0 = 0$ to represent the layer comprised of the source state only, and $m_K = L$ representing the terminating states $\mathcal{X}$. On each non-terminating junction layer $m_k \neq L$, we define a state flow function $F_k : \mathcal{S}_{m_k} \to \mathbb{R}_+^*$. Given any forward and backward policies P_F and P_B respectively, consistent with the DAG $\mathcal{G}$, the state flow functions define two sets of distributions $\check{p}_k$ and $\hat{p}_k$ over partial trajectories starting from a state $s_{m_k} \in \mathcal{S}_{m_k}$ and ending in a state $s_{m_{k+1}} \in \mathcal{S}_{m_{k+1}}$ (we denote by $\mathcal{T}_k$ the set comprised of these partial trajectories, for $k = 0 \dots K-1$):

$$\forall \tau_k = (s_{m_k} \to \dots \to s_{m_{k+1}}) \in \mathcal{T}_k \quad \hat{p}_k(\tau_k) \propto F_k(s_{m_k}) P_F(\tau_k), \tag{3.80}$$

$$\forall \tau_k = (s_{m_k} \to \dots \to s_{m_{k+1}}) \in \mathcal{T}_k \quad \check{p}_k(\tau_k) \propto F_{k+1}(s_{m_{k+1}}) P_B(\tau_k \mid s_{m_{k+1}}), \tag{3.81}$$

where F_K is fixed to the target reward function R.

Lemma 3.7.2 (Nested variational objective for subtrajectories)**.** *If $\hat{p}_k = \check{p}_k$ for all $k = 0 \dots K-1$, then the forward policy P_F induces a terminating state distribution $P_\top$ that matches the target unnormalized distribution (or reward function) R.*

Similar to NVI, we can use Lemma 3.7.2 to define objective functions for P_F, P_B, F_k, of the form:

$$\mathcal{L}_{\text{SubNVI},f}(P_F, P_B, F_{0:K-1}) = \sum_{k=1}^{K-1} D_f(\check{p}_k \| \hat{p}_k) \tag{3.82}$$

Note that the SubNVI objective of (3.82) matches the NVI objective (Zimmermann et al., 2021) when all layers are junction layers (i.e., $K = L$, and $m_k = k$ for all $k \leq L$), and matches

the HVI objective of (3.76) when only the first and last layers are junction layers (i.e., $K = 1$, $m_0 = 0$, and $m_1 = L$).

3.7.5. Analysis of gradients

The following proposition summarizes our main theoretical claim, relating the GFN objective of (3.70) and the variational objective of (3.76).

Proposition 3.7.3. *Given a graded DAG $\mathcal{G}$, and denoting by θ, ϕ the parameters of the forward and backward policies P_F, P_B respectively, the gradients of the TB objective (3.70) satisfy:*

$$\nabla_\phi D_{\mathrm{KL}}(P_B \| P_F) = \frac{1}{2}\mathbb{E}_{\tau\sim P_B}[\nabla_\phi \mathcal{L}_{\mathrm{TB}}(\tau)], \tag{3.83}$$

$$\nabla_\theta D_{\mathrm{KL}}(P_F \| P_B) = \frac{1}{2}\mathbb{E}_{\tau\sim P_F}[\nabla_\theta \mathcal{L}_{\mathrm{TB}}(\tau)]. \tag{3.84}$$

This result can be extended to the subtrajectory balance objective and the SubNVI objective of Section 3.7.4. A special case of this equivalence is between the Detailed Balance objective and the nested VI objective (Zimmermann et al., 2021).

Proposition 3.7.4. *Given a graded DAG $\mathcal{G}$ as in Section 3.7.1, with junction layers $m_0 = 0, m_1, \ldots, m_K = L$ as in Section 3.7.4. For any forward and backward policies, and for any positive function F_k defined for the hubs, consider $\hat{p}_k$ and $\check{p}_k$ defined in (3.80) and (3.81). The subtrajectory variational objectives of (3.82) are equivalent to the subtrajectory balance objective (3.73) for specific choices of the f-divergences. Namely, denoting by θ, ϕ the parameters of P_F, P_B respectively:*

$$\mathbb{E}_{\tau_k\sim\check{p}_k}[\nabla_\phi \mathcal{L}_{\mathrm{SubTB}}(\tau_k; P_F, P_B, F)] = 2\nabla_\phi D_{f_1}(\check{p}_k \| \hat{p}_k) \tag{3.85}$$

$$\mathbb{E}_{\tau_k\sim\hat{p}_k}[\nabla_\theta \mathcal{L}_{\mathrm{SubTB}}(\tau_k; P_F, P_B, F)] = 2\nabla_\theta D_{f_2}(\check{p}_k \| \hat{p}_k) \tag{3.86}$$

where $F = F_{0:K-1}$, and $f_1 : t \mapsto t\log t$ and $f_2 : t \mapsto -\log t$.

As a special case of Proposition 3.7.4, when the state flow function is defined for s_0 only (and for the terminating states, at which it equals the target reward function), i.e., when $K = 1$, the distribution $\hat{p}_0(\tau)$ and $P_F(\tau)$ are equal, and so are the distributions $\check{p}_0(\tau)$ and $P_B(\tau)$. We thus obtain the first two equations of Proposition 3.7.3 as a consequence of Proposition 3.7.4.

In the remainder of this chapter, we will focus on the equivalence with TB, provided by Proposition 3.7.3.

While (3.84) is the on-policy TB gradient with respect to the parameters of P_F, (3.83) is *not* the on-policy TB gradient with respect to the parameters of P_B, as the expectation is taken over P_B, not P_F. The on-policy TB gradient can, however, be expressed through a surrogate loss:

Lemma 3.7.5 (On-policy TB gradient with respect to P_B)**.** *Denoting by $D_{\log^2}$ the pseudo-f-divergence defined by $f(x) = \log(x)^2$, which is not convex for large x.*

$$\mathbb{E}_{\tau\sim P_F}[\nabla_\phi \mathcal{L}_{\mathrm{TB}}(\tau)] = \nabla_\phi \left[D_{\log^2}(P_B\|P_F) + 2(\log Z - \log \hat{Z}) D_{\mathrm{KL}}(P_F\|P_B)\right], \tag{3.87}$$

where $\hat{Z} = \sum_{x\in\mathcal{X}} R(x)$, the unknown true partition function.

The loss in (3.83) is not possible to optimize directly unless using importance weighting (cf. the end of Section 3.7.2), but optimization of P_B using (3.83) and P_F using (3.84) would yield the gradients of Reverse wake-sleep in expectation.

Score function estimator and variance reduction: Optimizing the reverse KL loss $D_{\mathrm{KL}}(P_F\|P_B)$ with respect to θ, the parameters of P_F, requires a likelihood ratio (also known as REINFORCE) estimator of the gradient (Williams, 1992), using a trajectory τ (or a batch of trajectories), which takes the form:

$$\Delta(\tau) = \nabla_\theta \log P_F(\tau;\theta) c(\tau), \quad \text{where } c(\tau) = \log \frac{P_F(\tau)}{R(x_\tau) P_B(\tau \mid x_\tau)} \tag{3.88}$$

(Note that the term $\nabla_\theta c(\tau)$ that is typically present in the REINFORCE estimator is 0 in expectation, since $\mathbb{E}_{\tau\sim P_F}[\nabla_\theta \log P_F(\tau)] = \sum_\tau \frac{P_F(\tau)}{P_F(\tau)} \nabla_\theta P_F(\tau) = 0$.) The estimator of (3.88) is known to exhibit a high variance norm, thus slowing down learning. A common workaround is to subtract a baseline b from $c(\tau)$, which does not bias the estimator. The value of the baseline b (also called control variate) that most reduces the trace of the covariance matrix of the gradient estimator is

$$b^* = \frac{\mathbb{E}_{\tau\sim P_F}[c(\tau)\|\nabla_\theta \log P_F(\tau;\theta)\|^2]}{\mathbb{E}_{\tau\sim P_F}[\|\nabla_\theta \log P_F(\tau;\theta)\|^2]}, \tag{3.89}$$

commonly approximated with $\mathbb{E}_{\tau\sim P_F}[c(\tau)]$ (see, e.g., Weaver and Tao (2001); Wu et al. (2018)). This approximation is itself often approximated with a batch-dependent **local** baseline, from a batch of trajectories $\{\tau_i\}_{i=1}^B$:

$$b^{\mathrm{local}} = \frac{1}{B}\sum_{i=1}^{B} c(\tau_i) \tag{3.90}$$

A better approximation of the expectation $\mathbb{E}_{\tau\sim P_F}[c(\tau)]$ can be obtained by maintaining a running average of the values $c(\tau)$, leading to a **global** baseline. After observing each batch

of trajectories, the running average is updated with step size η:

$$b^{\text{global}} \leftarrow (1-\eta)b^{\text{global}} + \eta b^{\text{local}}. \tag{3.91}$$

This coincides with the update rule of $\log Z$ in the minimization of $\mathcal{L}_{\text{TB}}(P_F, P_B, Z)$ with a learning rate $\frac{\eta}{2}$ for the parameter $\log Z$ (with respect to which the TB objective is quadratic). Consequently, (3.84) of Proposition 3.7.3 shows that the update rule for the parameters of P_F, when optimized using the Reverse KL objective, with (3.91) as a control variate for the score function estimator of its gradient, is the same as the update rule obtained by optimizing the TB objective using on-policy trajectories.

While learning a backward policy P_B can speed up convergence (Malkin et al., 2022), the TB objective can also be used with a *fixed* backward policy, in which case the Reverse KL objective and the TB objective differ only in how they reduce the variance of the estimated gradients, if the trajectories are sampled on-policy. In Section 3.8, we experimentally explore the differences between the two learning paradigms that arise when P_B is learned or when the algorithms run off-policy.

3.8. Experiments

The goal of the experiments is to empirically investigate two main observations consistent with the theoretical analysis provided in Section 3.7:

Observation 1. On-policy VI and TB (GFlowNet) objectives can behave similarly in some cases, when both can be stably optimized, while in others, on-policy TB strikes a better compromise than either the (mode-seeking) Reverse KL or (mean-seeking) Forward KL VI objectives. This claim is supported by the experiments on all three domains below.

However, in all cases, notable differences emerge. In particular, HVI training becomes more stable near convergence and is sensitive to learning rates, which is consistent with the hypotheses about gradient variance in Section 3.7.5.

Observation 2. When exploration matters, off-policy TB outperforms both on-policy TB and VI objectives, avoiding the possible high variance induced by importance sampling in off-policy VI. GFlowNets are capable of stable off-policy training without importance sampling. This claim is supported by experiments on all domains but is especially well illustrated on the realistic domains in Section 3.8.3 and Section 3.8.4. This capability provides advantages for capturing a more diverse set of modes.

Observation 1 and **Observation 2** provide evidence that off-policy TB is the best method among those tested in terms of both accurately fitting the target distribution and effectively finding modes, where the latter is particularly important for the challenging molecule graph generation and causal graph discovery problems studied below.

Code for these experiments can be found at https://github.com/GFNOrg/GFN_vs_HVI/.

3.8.1. Practical details

Evaluation of the terminating state distribution $P_\top$: When the state space is small enough, we can propagate the flows in order to compute the terminating state distribution $P_\top$ from the forward policy P_F. This is done using a flow function F defined recursively:

$$F(s') = \begin{cases} 1 & \text{if } s' = s_0 \\ \sum_{s \in Par(s')} F(s) P_F(s' \mid s) & \text{otherwise} \end{cases} \tag{3.92}$$

$P_\top$ is then given by:

$$P_\top(s) \propto F(s) P_F(s_f \mid s), \tag{3.93}$$

The recursion can be carried out by dynamic programming by enumerating the states in any topological ordering consistent with the graded DAG $\mathcal{G}$. In particular, computation of the flow at a given terminating state s is linear in the number of states and actions that lie on trajectories leading to s, and computation of the full distribution $P_\top$ is linear in $|\mathcal{S}| + |\mathbb{A}|$.

Evaluation of the Jensen-Shannon divergence (JSD). Similarly, when the state space is small enough, the target distribution $P^\top = R/Z^*$ can be evaluated exactly, given that the marginalization is over $\mathcal{X}$ only. The JSD is a symmetric divergence, thus motivating our choice. The JSD can directly be evaluated as:

$$JSD(P^\top \| P_\top) = \frac{1}{2}\left(D_{\mathrm{KL}}(P^\top \| M) + D_{\mathrm{KL}}(P_\top \| M)\right) \quad \text{where } M = (P^\top + P_\top)/2 \tag{3.94}$$

$$= \frac{1}{2} \sum_{s \in \mathcal{S}^o} \left(P^\top(s) \log \frac{2P^\top(s)}{P^\top(s) + P_\top(s)} + P_\top(s) \log \frac{2P_\top(s)}{P^\top(s) + P_\top(s)} \right) \tag{3.95}$$

3.8.2. Hypergrid: Exploration of learning objectives

In this section, we comparatively study the ability of the variational objectives and the GFlowNet objectives to learn a multimodal distribution given by its unnormalized density, or reward function, R. We use the synthetic hypergrid environment introduced by Bengio et al. (2021) and further explored by Malkin et al. (2022).

Details about the environment. In a D-dimension hypergrid of side length H, the state space satisfies $\mathcal{S} = \mathcal{X} \cup \{s_f\}$, where $\mathcal{X} = \{0, \ldots, H-1\}^D$. All states, besides the sink state, are thus terminating. The initial state is $\mathbf{0}_{\mathbb{R}^D} = (0, \ldots, 0) \in \mathcal{X}$, and in addition to the transitions from a non-sink state to another (by incrementing one coordinate of the state), an "exit" action is available for all $s \in \mathcal{X}$, that leads to the sink state s_f. The reward at a

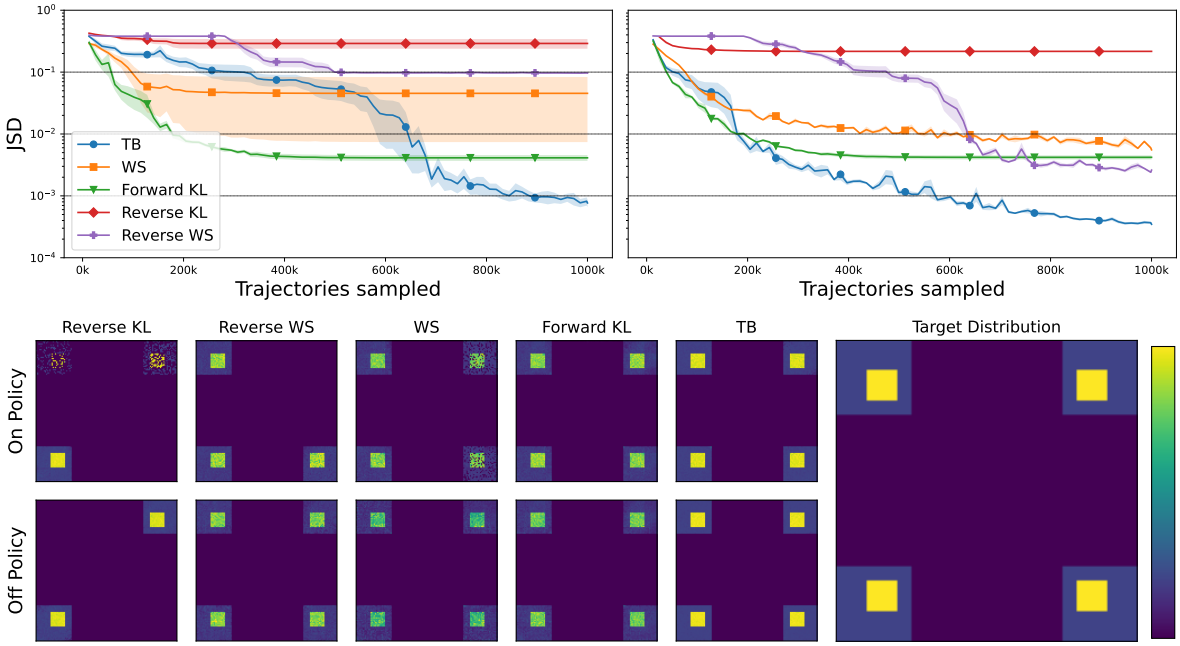


Figure 3.8 – Top: The evolution of the JSD between the learned sampler $P_\top$ and the target distribution on the 128×128 grid, as a function of the number of trajectories sampled. Shaded areas represent the standard error evaluated across 5 different runs (on-policy **left**, off-policy **right**). **Bottom:** The average (across 5 runs) final learned distribution $P_\top$ for the different algorithms, along with the target distribution. To amplify variation, the plot intensity at each grid position is resampled from the Gaussian approximating the distribution over the 5 runs. Although WS, Forward KL, and Reverse WS (off-policy) find the 4 target modes, they do not model them with high precision and produce a textured pattern at the modes, where it should be flat.

terminating state $s = (s^1, \ldots, s^D)^\top$ is:

$$R(s) = R_0 + 0.5 \prod_{d=1}^{D} \mathbf{1}\left(\left|\frac{s^d}{H-1} - 0.5\right| \in (0.25, 0.5]\right) + 2 \prod_{d=1}^{D} \mathbf{1}\left(\left|\frac{s^d}{H-1} - 0.5\right| \in (0.3, 0.4)\right), \tag{3.96}$$

where R_0 is an exploration parameter (lower values indicate harder exploration). The states form a D-dimensional hypergrid with side length H, and the reward function has 2^D flat modes near the corners of the hypergrid. The states form a pointed DAG, where the source state is the origin $s_0 = \mathbf{0}$, and each edge corresponds to the action of incrementing one coordinate in a state by 1 (without exiting the grid).

We focus on the case where P_B is learned, which has been shown to accelerate convergence (Malkin et al., 2022).

In Figure 3.8, we compare how fast each learning objective discovers the 4 modes of a 128× 128 grid, with an exploration parameter $R_0 = 0.001$ in the reward function. The gap between the learned distribution $P_\top$ and the target distribution is measured by the Jensen-Shannon

divergence (JSD) between the two distributions to avoid giving a preference to one KL or the other. Additionally, we show graphical representations of the learned 2D terminating states distribution, along with the target distribution. We provide in Section 3.8.1 details on how $P_\top$ and the JSD are evaluated and how hyperparameters were optimized separately for each learning algorithm.

Exploration poses a challenge in this environment, given the distance that separates the different modes. We thus include in our analysis an off-policy version of each objective, where the behaviour policy is different from, but related to, the trained sampler $P_F(\tau)$. The GFlowNet behaviour policy used here encourages exploration by reducing the probability of terminating a trajectory at any state of the grid. This biases the learner towards sampling longer trajectories and helps with faster discovery of farther modes. When off-policy, the HVI gradients are corrected using importance sampling weights.

For the algorithms that use a score function estimator of the gradient (Forward KL, Reverse WS, and Reverse KL), we found that using a global baseline, as explained in Section 3.7.2, was better than using the more common local baseline in most cases. In Figure 3.9, we illustrate the differences between the two types of baselines considered (**global** and **local**) for the 3 algorithms that use a score function estimator of the gradient, both on-policy and off-policy.

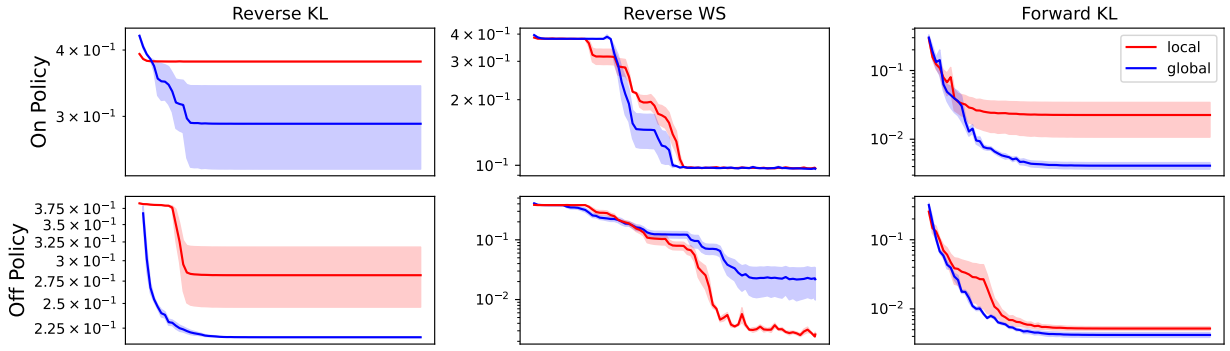


Figure 3.9 – A comparison of the type of baseline used (local or global) for the three HVI algorithms that use a score function estimator of the gradient.

By using global baselines, we bring the VI methods closer to GFlowNets and thus factor out this issue from the comparison with the GFlowNet objectives.

We see from Figure 3.8 that while Forward KL and WS – the two algorithms that use $D_{\mathrm{KL}}(P_B \| P_F)$ as the objective for P_F – discover the four modes of the distribution faster, they converge to a local minimum and do not model all the modes with high precision. This is due to the mean-seeking behaviour of the forward KL objective, requiring that $P_\top$ puts non-zero mass on terminating states x where $R(x) > 0$. Objectives that use the reverse KL to train the forward policy (Reverse KL and Reverse WS) are mode-seeking and can thus have a low loss without finding all the modes. The TB GFlowNet objective offers the

best of both worlds, as it converges to a lower value of the JSD, discovers the four modes, and models them with high precision. This supports **Observation 1**. Additionally, in support of **Observation 2**, while both the TB objective and the HVI objectives benefit from off-policy sampling, TB benefits more, as convergence is greatly accelerated.

Smaller environments: The environment studied thus far (128×128, with $R_0 = 10^{-3}$) already illustrates some key differences between the Forward and Reverse KL objectives. As a sanity check for the HVI methods that failed to converge in this challenging environment, we consider two alternative grids: 64×64 and $8 \times 8 \times 8 \times 8$, both with an easier exploration parameter ($R_0 = 0.1$), and compare the 5 algorithms on-policy on these two extra domains. Additionally, for the two-dimensional domain (64×64), we illustrate in Figure 3.10 a visual representation of the average distribution obtained after sampling 10^6 trajectories for each method separately. Interestingly, unlike the hard exploration domain, the two algorithms with the mode-seeking KL (Reverse KL and Reverse WS) converge to a lower JSD than the mean-seeking KL algorithms (Forward KL and WS), and are on par with TB.

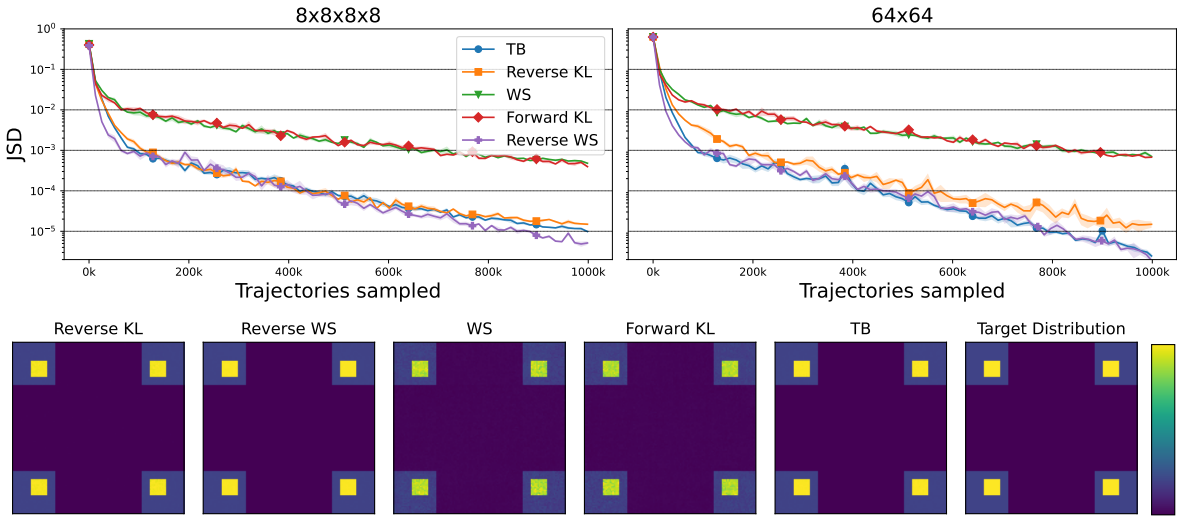


Figure 3.10 – Top: The evolution of the JSD between the learned sampler $P_\top$ and the target distribution on the $8 \times 8 \times 8 \times 8$ grid **left** and the 64×64 grid **right**. Trajectories are sampled on-policy. Shaded areas represent the standard error evaluated across 5 different runs **Bottom:** The average (across 5 runs) final learned distribution $P_\top$ for the different algorithms, along with the target distribution. To amplify variation, the plot intensity at each grid position is resampled from the Gaussian approximating the distribution over the 5 runs.

Implementation details for the Hypergrid experiments are provided in Appendix D.3.1.

3.8.3. Molecule synthesis

We study the molecule synthesis task from Bengio et al. (2021), in which molecular graphs are generated by sequential addition of subgraphs from a library of blocks (Jin et al., 2020;

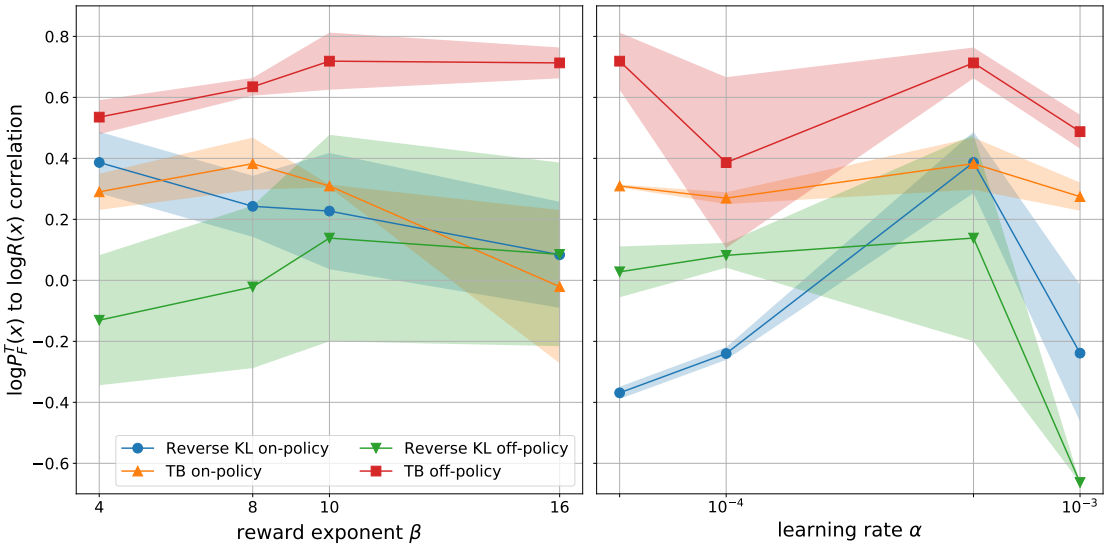


Figure 3.11 – Correlation between marginal sampling log-likelihood and log-reward on the molecule generation task for different learning algorithms, showing the advantage of off-policy TB (red) against on-policy TB (orange) and both on-policy (blue) and off-policy HVI (green). For each hyperparameter setting on the x-axis (α or β), we take the optimal choice of the other hyperparameter (β or α, respectively) and plot the mean and standard error region over three random seeds.

Kumar et al., 2012). The reward function is expressed in terms of a fixed, pretrained graph neural network f that estimates the strength of binding to the soluble epoxide hydrolase protein (Trott and Olson, 2010). To be precise, $R(x) = f(x)^\beta$, where $f(x)$ is the output of the binding model on molecule x, and β is a parameter that can be varied to control the entropy of the sampling model.

Because the number of terminating states is too large to exactly compute the target distribution, we use a performance metric from past work on this task (Bengio et al., 2021) to evaluate sampling agents. Namely, for each molecule x in a held-out set, we compute $\log P_\top(x)$, the likelihood of x under the trained model (tractably computable by dynamic programming, see Section 3.8.1), and evaluate the Pearson correlation of $\log P_\top(x)$ and $\log R(x)$. This value should equal 1 for a perfect sampler, as $\log P_\top(x)$ and $\log R(x)$ would differ by a constant, the log-partition function $\log \hat{Z}$.

In Malkin et al. (2022), GFlowNet samplers using the DB and TB objectives, with the backward policy P_B fixed to a uniform distribution over the parents of each state, were trained off-policy. Specifically, the trajectories used for DB and TB gradient updates were sampled from a mixture of the (online) forward policy P_F and a uniform distribution at each sampling step, with a special weight depending on the trajectory length used for the termination action.

We wrote an extension of the published code of Malkin et al. (2022) with an implementation of the HVI (Reverse KL) objective, using a reweighted importance sampling

correction. We compare the off-policy TB from past work with the off-policy Reverse KL, as well as on-policy TB and Reverse KL objectives. (Note that on-policy TB and Reverse KL are equivalent in expectation in this setting since the backward policy is fixed.) Each of the four algorithms was evaluated with four values of the inverse temperature parameter β and of the learning rate α, for a total of $4 \times 4 \times 4 = 64$ settings. (We also experimented with the off-policy Forward KL / WS objective for optimizing P_F, but none of the hyperparameter settings resulted in an average correlation greater than 0.1.)

The results are shown in Figure 3.11, in which, for each hyperparameter (α or β), we plot the performance for the optimal value of the other hyperparameter. We make three observations:

- In support of **Observation 2**, off-policy Reverse KL performs poorly compared to its on-policy counterpart, especially for smoother distributions (smaller values of β) where more diversity is present in the target distribution. Because the two algorithms agree in the expected gradient, this suggests that importance sampling introduces unacceptable variance into HVI gradients.
- In support of **Observation 1**, the difference between on-policy Reverse KL and on-policy TB is quite small, consistent with their gradients coinciding in the limit of descent along the full-batch gradient field. However, Reverse KL algorithms are more sensitive to the learning rate.
- In support of **Observation 2**, off-policy TB gives the best and lowest-variance fit to the target distribution, showing the importance of an exploratory training policy, especially for sparser reward landscapes (higher β).

3.8.4. Generation of DAGs in Bayesian structure learning

Finally, we consider the problem of learning the (posterior) distribution over the structure of Bayesian networks, as studied in Deleu et al. (2022). The goal of Bayesian structure learning is to approximate the posterior distribution $p(G \mid \mathcal{D})$ over DAGs G, given a dataset of observations $\mathcal{D}$. Following Deleu et al. (2022), we treat the generation of a DAG as a sequential decision problem, where directed edges are added one at a time, starting from the completely disconnected graph. Since our goal is to approximate the posterior distribution $p(G \mid \mathcal{D})$, we use the joint probability $R(G) = p(G, \mathcal{D})$ as the reward function, which is proportional to the former up to a normalizing constant. Details about how this reward is computed, as well as the parameterization of the forward policy P_F, are available in Appendix D.3.3. Note that similarly to Section 3.8.3, and following Deleu et al. (2022), we leave the backward policy P_B fixed to uniform.

We only consider settings where the true posterior distribution $p(G \mid \mathcal{D})$ can be computed exactly by enumerating all the possible DAGs G over d nodes (for $d \leq 5$). This allows us to

Objective	Number of nodes		
	3	4	5
(Modified) Detailed Balance	$5.32 \pm 4.15 \times 10^{-6}$	$2.05 \pm 0.70 \times 10^{-5}$	$\mathbf{4.65 \pm 1.08 \times 10^{-4}}$
Off-Policy Trajectory Balance	$\mathbf{3.70 \pm 2.51 \times 10^{-7}}$	$\mathbf{9.35 \pm 2.99 \times 10^{-6}}$	$5.44 \pm 2.47 \times 10^{-4}$
On-Policy Trajectory Balance	0.022 ± 0.007	0.123 ± 0.028	0.277 ± 0.040
On-Policy Reverse KL (HVI)	0.022 ± 0.007	0.125 ± 0.027	0.306 ± 0.042
Off-Policy Reverse KL (HVI)	0.014 ± 0.008	0.605 ± 0.019	0.656 ± 0.009

Table 3.2 – Comparison of the Jensen-Shannon divergence for Bayesian structure learning, showing the advantage of off-policy TB over on-policy TB and on-policy or off-policy HVI. The JSD is measured between the true posterior distribution $p(G \mid \mathcal{D})$ and the learned approximation $P_\top(G)$.

exactly compare the posterior approximations, found either with the GFlowNet objectives or HVI, with the target posterior distribution. The state space grows rapidly with the number of nodes (e.g., there are 29k DAGs over $d = 5$ nodes). For each experiment, we sampled a dataset $\mathcal{D}$ of 100 observations from a randomly generated ground-truth graph $G^\star$; the size of $\mathcal{D}$ was chosen to obtain highly multimodal posteriors. In addition to the (Modified) DB objective introduced by Deleu et al. (2022), we also study the TB (GFlowNet) and the Reverse KL (HVI) objectives, both on-policy and off-policy.

In Table 3.2, we compare the posterior approximations found using these different objectives in terms of their Jensen-Shannon divergence (JSD) to the target posterior distribution $P(G \mid \mathcal{D})$. We observe that on the easiest setting (graphs over $d = 3$ nodes), all methods accurately approximate the posterior distribution. But as we increase the complexity of the problem (with larger graphs), we observe that the accuracy of the approximation found with Off-Policy Reverse KL degrades significantly, while the ones found with the off-policy GFlowNet objectives ((Modified) DB & TB) remain very accurate. We also note that the performance of On-Policy TB and On-Policy Reverse KL degrades too, but not as significantly; furthermore, both of these methods achieve similar performance across all experimental settings, confirming our **Observation 1**, and the connection highlighted in Section 3.7.2. The consistent behaviour of the off-policy GFlowNet objectives compared to the on-policy objectives (TB & Reverse KL) as the problem increases in complexity (i.e., as the number of nodes d increases, requiring better exploration) also supports our **Observation 2**. These observations are further confirmed when comparing the edge marginals $P(X_i \to X_j \mid \mathcal{D})$ in Figure D.1 (Appendix D.3.3), computed either with the target posterior distribution or with the posterior approximations.

Chapter 4

A theory of continuous generative flow networks

This chapter is based upon the following paper:

— Lahlou et al. (2023a): "A theory of continuous generative flow networks" - Salem Lahlou, Tristan Deleu, Pablo Lemos, Dinghuai Zhang, Alexandra Volokhova, Alex Hernández-García, Léna Néhale Ezzine, Yoshua Bengio, Nikolay Malkin, published in 2023 in the proceedings of the International Conference on Machine Learning (ICML).

Sections 4.1 and 4.2 are shortened versions of Sections 1 and 2 of Lahlou et al. (2023a), to avoid repetition of content discussed in Chapter 3. The rest of the chapter is identical to Lahlou et al. (2023a), except for some content that was moved from the appendix to the main body of text to improve the reading flow and to highlight the contributions of the author of the thesis.

4.1. Introduction

We have introduced in the previous chapter generative flow networks (GFlowNets), an increasingly popular class of methods that amortize sampling from intractable distributions over spaces with a compositional structure by learning a sequential sampling policy. Their applications include the design of biological structures such as molecules (Bengio et al., 2021; Jain et al., 2022b), Bayesian structure learning (Deleu et al., 2022; Nishikawa-Toomey et al., 2022), and robust combinatorial optimization (Zhang et al., 2023a). Naturally, their development and theoretical foundations, delved into in Chapter 3, have been geared towards environments with discrete structures.

As many probabilistic inference and modelling problems involve continuous variables, it is natural to ask whether the advantages of GFlowNets, which include stable off-policy

learning and the ability to capture many modes of the target distribution, extend to general spaces. For example, molecule design implies specifying relative spatial positions of atoms and benefits from modelling continuous variables, such as torsion angles (Jing et al., 2022), and Bayesian structure learning requires the discovery of not only the structure of the graphical model but also its parameters.

As an attempt at using GFlowNet losses to train an amortized sampler of an unnormalized continuous density, Li et al. (2023) presented an extension of the flow-matching conditions (Bengio et al., 2021) to continuous domains; however, this extension relies upon several invalid assumptions, as we expand on in Section 4.3.1.

This chapter presents a theory extending all known GFlowNet training objectives to arbitrary spaces. It relies on measurable pointed graphs, a generalization of directed acyclic graphs (DAGs) to measurable spaces based on Markov kernels. Our main **theoretical contributions** are an extension of the flow-matching (FM), detailed balance (FB), and trajectory balance (TB) conditions, and a theorem proving that the learned *forward kernel* samples from the target distribution when any of these conditions is satisfied. These conditions lead to training losses involving density functions and allowing gradient-based learning. Existing losses for discrete GFlowNets are special cases of the ones we state.

Additionally, we provide **experimental results** in multiple domains with different structures, some of which include action spaces with both discrete and continuous components. These experiments serve both to validate the theoretical claims and to inform practitioners of caveats that are specific to continuous domains. Our comparative experiments confirm that the already-proven advantages of discrete GFlowNets transfer to more general state spaces.

The remainder of the chapter is structured as follows:

Section 4.2 reviews work on stochastic sampling;

Section 4.3 presents the theoretical results and a practical summary;

Section 4.4 is devoted to empirically validating the theory and comparing generalized GFlowNets with baselines.

4.2. Stochastic sampling in continuous spaces

Sequential sampling in continuous spaces has a long history. Specialized MCMC methods (Section 2.2.2) exist for sampling from continuous or differentiable densities, such as Langevin and Hamiltonian MCMC Coffey and Kalmykov (2012); Neal (2012); Hoffman and Gelman (2011).

Another line of work considers stochastic sampling in a finite number of steps. The family of sequential Monte Carlo methods (Doucet et al., 2001) and the closely related annealed importance sampling Neal (2001b), already introduced in Section 2.2.2, specify a sequence

of intermediate target densities with respect to which samplers aim to approximately satisfy detailed balance, but the transition kernel is typically not a learned neural network policy. More recently, learnable-kernel sampling methods, formulated as score-based or stochastic differential equation modelling, have been used for maximum-likelihood generative modelling (e.g., Sohl-Dickstein et al., 2015b; Song and Ermon, 2019; Ho et al., 2020b; Dockhorn et al., 2022), as well as for learning to sample from an intractable target density (Zhang and Chen, 2022). As we show in our experiments, such algorithms can be seen as special cases of GFlowNets where the state space is of a particular form (Section 4.4.3, Section 4.4.6).

Another related direction is stochastic normalizing flows (Wu et al., 2020), which have been interpreted with a Markov chain perspective (Hagemann et al., 2022), relying on Markov kernels and Radon-Nikodym derivatives just as our theory of generalized GFlowNets.

4.3. A theory for generalized GFlowNets

All lemmas, propositions and theorems of this section are proved in Appendix E.3.

4.3.1. Practical summary

Discrete GFlowNet	Generalized GFlowNet	Reference
The state space is a finite set with distinguished source and sink states	The state space is a topological space with distinguished source and sink states and may consist of both continuous and discrete components	Definition 4.3.3
Children and parents of a state s	Supports of the measures $\kappa(s,-), \kappa^b(s,-)$	Definition 4.3.3
All states are reachable from s_0	All open sets are reachable from s_0 with nonzero likelihood	(4.5)
The state $\perp$ has no outgoing edges	The state $\perp$ is absorbing	(4.6)
The state graph is acyclic ($\Rightarrow$ trajectory lengths are bounded)	The measurable pointed graph is finitely absorbing	(4.11)
State flow F, forward policy P_F, backward policy P_B	Flow measure μ, forward kernel P_F, backward kernel P_B	Definitions 4.3.12 and 4.3.17
Transition likelihoods $P_F(- \mid s)$ only positive along edges	Transition kernels $P_F(s,-)$ absolutely continuous with respect to κ	Definition 4.3.12
Terminating distribution $P_\top$	Terminating state measure $P_\top$	Definition 4.3.6 and Theorem 4.3.14
Flow-matching implies sampler matches reward function R	Flow-matching implies sampler matches reward measure R	Definition 4.3.12 and Theorem 4.3.14
Detailed balance: $F(s)P_F(s' \mid s) = F(s')P_B(s \mid s')$	Detailed balance: $\mu(ds)P_F(s,ds') = \mu(ds')P_B(s',ds)$	Definition 4.3.17
Trajectory balance: $ZP_F(\tau) = R(x_\tau)P_B(\tau \mid x_\tau)$	Trajectory balance: $ZP_F(s_0,ds_1)\dots P_F(s_n,\{\perp\}) = R(ds_n)P_B(s_n,ds_{n-1})\dots P_B(s_1,\{s_0\})$	Definition 4.3.19

Table 4.1 – Dictionary between discrete and generalized GFlowNets

A summary of the key differences and analogies between discrete and generalized GFlowNets is provided in Table 4.1, and the precise way in which discrete GFlowNets are

special cases of generalized GFlowNets is stated in Example 4.3.4. Most importantly, the soundness of the theory relies upon the following assumptions, which need to be carefully verified when training a GFlowNet in an infinite space:

(1) The structure of the state space must allow all states to be reachable from the source state s_0 (4.5);

(2) The structure must ensure that the number of steps required to reach any state from s_0 is bounded (4.11);

(3) The learned probability measures need to be expressed through densities over states, rather than over actions.

On previous attempts to train continuous GFlowNets. While Li et al. (2023) proposed to train continuous GFlowNets by writing the flow matching conditions as integrals rather than sums, assumptions (1) and (3) are violated in a critical way. First, the environments considered in Li et al. (2023)'s experiments violate assumption (1), without which the main GFlowNet training theorems do not hold. Second, regarding (3), Li et al. (2023) implicitly assumes that for a state s and flow function $F(s \to s)$,

$$\int_{s':s\to s'} F(s \to s')\, ds' = \int_a F(s \to T(s,a))\, da,$$

where the second integral is taken over actions and $T(s,a)$ is the state reached by taking action a from s. This change of variables is invalid in general: *the integrand on the right side is missing the Jacobian term* $\frac{dT(s,a)}{da}$, which need not equal 1. In particular, it does not equal 1 in the environments studied by Li et al. (2023) (although it may hold in special cases, such as sampling in Euclidean spaces where $T(s,a) = s + a$). These issues are concerning for the scope of that method's applicability.

4.3.2. Structured state space

Note. To help the reader form a mental picture, we list the concepts introduced and their discrete analogues in Table 4.1 and formally state the connection in Example 4.3.4. Paragraphs marked (⋆) explain the meaning of the technical results.

(⋆) How could one describe a structure in general spaces, similar to DAGs on finite sets? In finite sets, it would suffice to enumerate the child sets and parent sets of all states, with the constraint that s' is a child of s if and only if s is a parent of s'. In general state spaces, however, enumeration is replaced by measure. One could thus define, for each state, a measure on the state space describing what states can be accessed in one step.

The structured state spaces we consider will be called *measurable pointed graphs* and rely on *transition kernels* (Nummelin, 2004; Cappé et al., 2009; Petritis, 2012). Before delving into measurable pointed graphs in Section 4.3.2.2, we recall some definitions and important notations in Section 4.3.2.1.

4.3.2.1. Background on measure theory and transition kernels

Notation

Given a measurable space $(\bar{\mathcal{S}}, \Sigma)$, we denote by $\Sigma_{|\mathcal{U}}$ the restriction of Σ to any subset $\mathcal{U}$ of $\bar{\mathcal{S}}$.

Definition 4.3.1 (Transition kernel)**.** Let $(\bar{\mathcal{S}}, \Sigma)$ be a measurable (state) space. A function $\kappa : \bar{\mathcal{S}} \times \Sigma \to [0, +\infty)$ is called a positive σ-finite *transition kernel* if

(1) For any $B \in \Sigma$, the mapping $s \mapsto \kappa(s, B)$ is measurable, where the space $[0, +\infty)$ is associated with the Borel σ-algebra $\mathcal{B}([0, +\infty))$;

(2) For any $s \in \bar{\mathcal{S}}$, the mapping $B \mapsto \kappa(s, B)$ is a positive σ-finite measure on $(\bar{\mathcal{S}}, \Sigma)$.

A transition kernel such that the mappings $B \mapsto \kappa(s, B)$ are probability measures is called a *Markov kernel*.

Products of kernels. Given a measurable space $(\bar{\mathcal{S}}, \Sigma)$, a positive measure ν on $(\bar{\mathcal{S}}, \Sigma)$, and a transition kernel κ on $(\bar{\mathcal{S}}, \Sigma)$, we denote by $\nu\kappa$ (resp. $\nu \otimes \kappa$) the measure on $(\bar{\mathcal{S}}, \Sigma)$ (resp. $(\bar{\mathcal{S}} \times \bar{\mathcal{S}}, \Sigma \otimes \Sigma)$) defined for $B \in \Sigma$ (resp. $B \in \Sigma \otimes \Sigma$) as:

$$\nu\kappa(B) = \int_{\bar{\mathcal{S}}} \nu(ds)\kappa(s, B). \tag{4.1}$$

$$\nu \otimes \kappa(B) = \iint_{\bar{\mathcal{S}}^2} \mathbf{1}_B(s, s')\nu(ds)\kappa(s, ds') \tag{4.2}$$

In particular, for any state $s \in \bar{\mathcal{S}}$, the *n-step measure* $\kappa^n(s, -)$ is recursively defined by $\kappa^0(s, -) = \delta_s$, the Dirac at s, and:

$$\kappa^{n+1}(s, -) = \kappa^n(s, -)\kappa. \tag{4.3}$$

Notation

Given two measures μ, ν on $(\bar{\mathcal{S}}, \Sigma)$, we say that μ is absolutely continuous with respect to ν, and write $\mu \ll \nu$, if for every $B \in \Sigma$ satisfying $\nu(B) = 0$, we have $\mu(B) = 0$.

The following lemma ensures that absolute continuity between transition kernels transfers to n-step measures

Lemma 4.3.2. *Let κ, P_F be two transition kernels on $(\bar{\mathcal{S}}, \Sigma)$ such that $P_F(s, -) \ll \kappa(s, -)$ for every $s \in \bar{\mathcal{S}}$. Then for every $n \geq 1$, $P_F^n(s, -)$ and $s \in \mathcal{S}$ is absolutely continuous with respect to $\kappa^n(s, -)$.*

Equality between measures. Given two measures p and q on $(\bar{\mathcal{S}}, \Sigma)$, and a function $g : \bar{\mathcal{S}} \to \mathbb{R}$, we use the notation:

$$p(ds) = g(s)q(ds)$$

to say that for any measurable bounded function $f : \bar{\mathcal{S}} \to \mathbb{R}$:

$$\int_{\bar{\mathcal{S}}} f(s)p(ds) = \int_{\bar{\mathcal{S}}} f(s)g(s)q(ds). \tag{4.4}$$

Throughout the paper, we use the two notations interchangeably when the context allows it. Our proofs rely mostly on writing the equality with measurable bounded functions.

4.3.2.2. Measurable pointed graphs

Definition 4.3.3 (Measurable pointed graph)**.** A *measurable pointed graph* $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, \perp, \kappa, \kappa^b, \nu)$ consists of:

— A topological space $(\bar{\mathcal{S}}, \mathcal{T})$, where $\mathcal{T}$ is the set of open subsets of $\bar{\mathcal{S}}$ and Σ is the Borel σ-algebra associated to the topology on $\bar{\mathcal{S}}$;

— A pair of distinct distinguished states $s_0 \in \bar{\mathcal{S}}$ and $\perp \in \bar{\mathcal{S}}$, called the *source state* and *sink state*, such that $\{s_0\}$ and $\{\perp\}$ are both open and closed sets. We define $\mathcal{S} = \bar{\mathcal{S}} \setminus \{\perp\}$ and $\mathcal{S}^\circ = \mathcal{S} \setminus \{s_0\}$, so the topology on $\bar{\mathcal{S}}$ is the disjoint union topology on $\{s_0\}$, $\{\perp\}$, and $\mathcal{S}^\circ$.

— A σ-finite transition kernel κ on $(\bar{\mathcal{S}}, \Sigma)$, called the *reference kernel*,

— A σ-finite transition kernel κ^b on $(\bar{\mathcal{S}}, \Sigma)$, called the *backward reference kernel*,

— A *strictly positive* σ-finite measure ν on $(\bar{\mathcal{S}}, \Sigma)$, called the *reference measure*,

such that the following conditions hold:

$$\forall B \in \mathcal{T} \setminus \{\emptyset\} \quad \exists n \geq 0 : \kappa^n(s_0, B) > 0, \tag{4.5}$$

$$\kappa(\perp, -) = \delta_\perp, \tag{4.6}$$

$$\forall B \in \Sigma,\ s \mapsto \kappa(s, B) \text{ is continuous}, \tag{4.7}$$

$$\forall f : \bar{\mathcal{S}} \times \bar{\mathcal{S}} \to \mathbb{R} \text{ measurable bounded, satisfying } f(s_0, s_0) = f(\perp, \perp) = 0 :$$
$$\iint_{\bar{\mathcal{S}} \times \bar{\mathcal{S}}} f(s, s')\nu(ds)\kappa(s, ds') = \iint_{\bar{\mathcal{S}} \times \bar{\mathcal{S}}} f(s, s')\nu(ds')\kappa^b(s', ds), \tag{4.8}$$

$$\forall B \in \Sigma,\ \kappa^b(s_0, B) = 0, \tag{4.9}$$

$$\forall s \in \mathcal{S},\ \kappa(s, \{\perp\}) > 0 \Rightarrow \kappa(s, \{\perp\}) = 1. \tag{4.10}$$

The measurable pointed graph is called *finitely absorbing* if

$$\exists N > 0 : \operatorname{supp}(\kappa^N(s_0, -)) = \{\perp\}, \tag{4.11}$$

where supp denotes the support of a measure, in which case the minimal such N is called the maximal trajectory length.

(⋆) The reference transition kernel κ provides a notion of "structure" of the state space. The support of $\kappa(s, -)$ (resp. $\kappa^b(s, -)$) can be thought of as the child set (resp. parent set) of the state s. For example, in a discrete graph, $\kappa(s, -)$ could be uniform over the children of s. The reference kernel is not a policy to be sampled, but an object needed to define probability densities of policies. The measure ν, the reference with respect to which flows and rewards are defined, is typically a simple measure, such as the counting measure on a discrete set or the standard Lebesgue measure on a Euclidean space.

In practice, if the structure is only defined by the reference kernel κ, then ν and κ^b satisfying the conditions of Definition 4.3.3 can be defined from κ under some mild assumptions, as we discuss in Proposition E.1.2 in Appendix E.1.

The following example shows that pointed directed acyclic graphs (Bengio et al., 2023) are a special case of finitely absorbing measurable pointed graphs.

Example 4.3.4. *Finite state spaces are special cases of measurable pointed graphs. Let $G = (V, E, s_0, \perp)$ be a pointed directed acyclic graph, where V is the finite set of vertices, $E \subset V \times V$ is the set of directed edges, $s_0 \in V$ is the initial state, and $\perp \in V$ is the sink state.*

The set of vertices V with the discrete topology corresponds to the state space. We can define a transition kernel κ such that for any vertex $s \in V$, and any $B \in \mathcal{P}(V)$, with $\mathcal{P}(V)$ the power set of V, containing all subsets of V:

$$\kappa(s, B) = \sum_{s' \in B} \mathbf{1}_E(s \to s') + \mathbf{1}(s = \perp)\mathbf{1}_B(\perp)$$

Using this transition kernel, the measure $B \mapsto \kappa^n(s, B)$ over $(V, \mathcal{P}(V))$ counts the number of trajectories of length n starting at s that ends at a vertex in B in the pointed graph G.

The reverse kernel can be defined for any vertex $s' \in V$ and any $B \in \mathcal{P}(V)$ as:

$$\kappa^b(s', B) = \sum_{s \in B} \mathbf{1}_E(s \to s'),$$

and the reference measure ν can be defined as the counting measure (that counts the number of elements in any $B \in \mathcal{P}(V)$).

Since $(V, \mathcal{P}(V))$ is a discrete space, the condition of accessibility in (4.5) can be verified for only singletons $B = \{s\}$. This condition then corresponds to having a positive number of trajectories of any length $n > 0$ starting at s_0 and ending in s, which is exactly the notion of accessibility in G. The continuity condition is trivially satisfied because the topology is discrete, and (4.8) is trivially satisfied. Finally, (4.11) is satisfied given the acyclicity of G.

From now on, we fix a finitely absorbing measurable pointed graph $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, \perp, \kappa, \kappa^b, \nu)$ with maximal trajectory length N.

Definition 4.3.5 (Terminating states)**.** The set of *terminating states* $\mathcal{X}$ is defined by:

$$\mathcal{X} = \{s \in \mathcal{S} \ : \ \kappa(s, \{\bot\}) > 0\}. \tag{4.12}$$

(⋆) Terminating states are ones from which one can transition to $\bot$ with positive probability. Any transition kernel can be sampled for n steps, yielding a measure over n-step trajectories and a marginal measure over states reached after n steps, as described in Section 4.3.2.3, this can be used to define the marginal terminating measure $P_\top$ of a transition kernel P_F, used in Sections 4.3.3 and 4.3.4.

4.3.2.3. Trajectory and terminating state measures

Definition 4.3.6. Let P_F be a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$. For any $n \geq 0$ and $s \in \bar{\mathcal{S}}$, P_F induces a measure $P_F^{\otimes n}(s, -)$ over the product space $(\bar{\mathcal{S}}^{n+1}, \Sigma^{\otimes(n+1)})$. $P_F^{\otimes n}(s, -)$, called the *n-step trajectory measure at s* recursively defined by

$$P_F^{\otimes 0}(s, -) = \delta_s, \tag{4.13}$$

$$P_F^{\otimes n+1}(s, -) = P_F^{\otimes n}(s, -) \otimes P_F, \tag{4.14}$$

where, as a generalization of (4.2), (4.14) means that for all $B \in \Sigma^{\otimes(n+2)}$:

$$P_F^{\otimes n+1}(s, B) = \int_{\bar{\mathcal{S}}^{n+2}} \mathbf{1}_B(s_1, \ldots, s_{n+2}) P_F^{\otimes n}(s, ds_1 \ldots ds_{n+1}) P_F(s_{n+1}, ds_{n+2}).$$

Notation

We use $\overrightarrow{s_{1:n}}$ to denote $(s_1, \ldots, s_n)$ and $\overrightarrow{ds_{1:n}}$ to denote $ds_1 \ldots ds_n$. We can write for example: $P_F^{\otimes 1}(s, ds'\, ds_1) = \delta_s(ds') P_F(s', ds_1)$ and $P_F^{\otimes 2}(s, ds'\, ds_1\, ds_2) = \delta_s(ds') P_F(s', ds_1) P_F(s_1, ds_2)$, and more generally:

$$P_F^{\otimes n}(s, ds'\, \overrightarrow{ds_{1:n}}) = P_F^{\otimes n-1}(s, ds'\, \overrightarrow{ds_{1:n-1}}) P_F(s_{n-1}, ds_n)$$

Terminating state measure. Given a measurable pointed DAG $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, \bot, \kappa, \kappa^b, \nu)$, any transition kernel P_F on $(\bar{\mathcal{S}}, \Sigma)$ induces a terminating state measure $P_\top$, which is the sum of the n-step terminating state measures defined as follows:

Definition 4.3.7. Let P_F be a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$. For any $n \geq 0$ we define the *n-step terminating state measure* $P_\top^n$ over $(\mathcal{X}, \Sigma_{|\mathcal{X}})$, for any $B \in \Sigma_{|\mathcal{X}}$ as:

$$P_\top^n(B) = \int_{\bar{\mathcal{S}}^{n+1}} P_F^{\otimes n}(s_0, ds_1 \dots ds_{n+1}) \mathbf{1}_B(s_n) \mathbf{1}(s_{n+1} = \bot). \tag{4.15}$$

The terminating state measure is defined as:

$$P_\top : B \in \Sigma_{|\mathcal{X}} \mapsto \sum_{n=1}^{\infty} P_\top^n(B) \tag{4.16}$$

The following lemma, relates the n-step terminating measures to the n-step measures $P_F^n(s_0, -)$:

Lemma 4.3.8. *Let P_F be a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$. For every $n \geq 1$, we have:*

$$P_\top^n(dx) = P_F(x, \{\bot\}) P_F^{n-1}(s_0, dx) \tag{4.17}$$

Its proof relies on the following intermediary result, which relates the n-step trajectory measures $P_F^{\otimes n}(s, -)$ to the n-step measures $P_F^n(s, -)$ defined by (4.3).

Lemma 4.3.9. *Let P_F be a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$. For every $s \in \bar{\mathcal{S}}$, $n \geq 0$, and for any bounded measurable function $f : \bar{\mathcal{S}} \to \mathbb{R}$, we have:*

$$\int_{\bar{\mathcal{S}}^{n+1}} f(s') P_F^{\otimes n}(s, ds_1 \dots ds_n ds') = \int_{\bar{\mathcal{S}}} f(s') P_F^n(s, ds') \tag{4.18}$$

4.3.2.4. Properties of measurable pointed graphs

The following lemma ensures that in a measurable pointed graph, $\{s_0\}$ is not accessible.

Lemma 4.3.10. $\forall s \in \mathcal{S},\ \kappa(s, \{s_0\}) = 0$

The following lemma ensures a compatibility between the definition of the terminating states $\mathcal{X}$ and $\kappa^b(\bot, -)$:

Lemma 4.3.11. *The support of $\kappa^b(\bot, -)$ is the closure of $\mathcal{X}$.*

4.3.3. Flows

Definition 4.3.12 (Flows and flow-matching conditions)**.** Given a σ-finite measure μ over $(\bar{\mathcal{S}}, \Sigma)$ that is absolutely continuous with respect to ν (we write $\mu \ll \nu$), and a σ-finite Markov kernel P_F on $(\bar{\mathcal{S}}, \Sigma)$ (i.e., a transition kernel such that each $P_F(s, -)$ is a probability measure) satisfying:
(1) $P_F(s, -) \ll \kappa(s, -)$ for every $s \in \bar{\mathcal{S}}$,
(2) $s \mapsto P_F(s, B)$ is continuous for every $B \in \Sigma$,
P_F is said to be a **forward kernel** over G. We say that the tuple $F = (\mu, P_F)$ satisfies the **flow-matching (FM) conditions** if for any bounded measurable function $f : \bar{\mathcal{S}} \to \mathbb{R}$ satisfying $f(s_0) = 0$, we have

$$\int_{\bar{\mathcal{S}}} f(s')\mu(ds') = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s')\mu(ds)P_F(s, ds'). \tag{4.19}$$

In which case, we say that F is a **flow** over G.

($\star$) The condition of absolute continuity with respect to the reference kernel κ indicates that the flow F must follow the "structure" of the measurable pointed graph, by assigning positive measure only to parts of the space where the measure induced by κ is also positive. The kernel P_F can be represented through a density function with respect to κ, which represents a probability *mass* (if the action space is discrete) or a probability *density* (if it is continuous). This allows to write conditions such as (4.19) using densities (Radon-Nikodym derivatives), thus providing practical loss functions to train GFlowNets. We expand on this point in Section 4.3.5.

Definition 4.3.13 (Reward-matching conditions)**.** Let $F = (\mu, P_F)$ be a flow over G. Given a positive and finite measure R over $\mathcal{X}$, called the reward measure, satisfying $R \ll \nu$, the flow F is said to satisfy the *reward-matching condition* with respect to R if we have:

$$R(dx) = \mu(dx)P_F(x, \{\perp\}). \tag{4.20}$$

The following theorem ascertains that, similar to discrete GFlowNets, when the flow and reward matching conditions are satisfied, then recursively sampling from the Markov kernel P_F starting from s_0 (until reaching $\perp$) yields samples from the normalized reward.

Theorem 4.3.14. *If $F = (\mu, P_F)$ is a flow over G, that satisfies the reward matching conditions (4.20) with respect to a measure R, then the corresponding terminating state measure $P_\top$ (Definition 4.3.7) is a probability measure and satisfies for all $B \in \Sigma_{|\mathcal{X}}$:*

$$P_\top(B) = \frac{1}{R(\mathcal{X})} R(B). \tag{4.21}$$

($\star$) $R(\mathcal{X})$, the reward measure taken over the set of all terminating states $\mathcal{X}$, corresponds to the total reward or partition function Z of GFlowNets. Certain conditions (Def. 4.3.12 and 4.3.13) on μ, which represents a state flow, and P_F, which represents a policy, imply that the marginal terminating distribution of the policy is proportional to the reward. These conditions correspond to the "flow in = flow out" condition at vertices of a discrete DAG.

The proof of the theorem relies on the following interesting result:

Proposition 4.3.15. *Let $F = (\mu, P_F)$ be a flow over G (i.e., F satisfies the flow-matching conditions (4.19)), then the measure u defined by:*

$$u : B \in \Sigma_{|\mathcal{S}} \mapsto \sum_{n=0}^{\infty} P_F^n(s_0, B), \tag{4.22}$$

is finite, and satisfies for all $B \in \Sigma_{|\mathcal{S}}$

$$\mu(\{s_0\})u(B) = \mu(B) \tag{4.23}$$

The following lemma, relates the flow measures at the source and sink states to the reward measure and shows that the source and sink flows correspond to the "total reward" $R(\mathcal{X})$ (called the partition function with discrete GFlowNets):

Lemma 4.3.16. *Let $F = (\mu, P_F)$ be a flow over G satisfying reward-matching conditions in (4.20) with respect to a measure R, then*

$$\mu(\{s_0\}) = \mu(\{\bot\}) = R(\mathcal{X}). \tag{4.24}$$

4.3.4. Detailed balance and trajectory balance

In finite GFlowNets, the detailed balance conditions (Bengio et al., 2023) and the trajectory balance conditions (Malkin et al., 2022) were converted into training objectives in order

to sample from a target unnormalized distribution. In this section, we present analogous conditions for general measurable pointed graphs.

Definition 4.3.17. Let μ be a σ-finite measure over $(\bar{\mathcal{S}}, \Sigma)$ such that $\mu \ll \nu$, P_F a forward kernel over G, and P_B a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$ such that:
(1) $P_B(s, -) \ll \kappa^b(s, -)$ for every $s \in \bar{\mathcal{S}}$,
(2) $s \mapsto P_B(s, B)$ is continuous for every $B \in \Sigma$,
(3) $P_B(s, -)$ is a probability measure for every $s \neq s_0$,
P_B is then said to be a **backward kernel** over G. We say that (μ, P_F, P_B) satisfy the **detailed balance (DB) conditions** if for any bounded measurable function $f : \mathcal{S} \times \bar{\mathcal{S}} \to \mathbb{R}$ satisfying $f(s, s_0) = 0$ for every $s \in \mathcal{S}$, we have

$$\begin{aligned} &\iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s, s')\mu(ds)P_F(s, ds') \\ &\qquad = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s, s')\mu(ds')P_B(s', ds). \end{aligned} \tag{4.25}$$

The following proposition shows an equivalence between the DB and FM conditions.

Proposition 4.3.18. *If (μ, P_F, P_B) satisfy the detailed balance conditions in Definition 4.3.17, then $F = (\mu, P_F)$ satisfies the flow-matching conditions in Definition 4.3.12 and is thus a flow.*

Definition 4.3.19. Let P_F be a forward kernel over G, P_B a backward kernel over G, and $Z \in \mathbb{R}_+$. Let R be a positive finite measure on $\mathcal{X}$. The triple (Z, P_F, P_B) satisfies the **trajectory balance (TB) conditions** with respect to R if for any $n \geq 0$ and any bounded measurable function $f : \bar{\mathcal{S}}^{n+2} \to \mathbb{R}$:

$$\begin{aligned} &\int_{\bar{\mathcal{S}}^{n+2}} Z f(s, \overrightarrow{s_{1:n+1}})\mathbf{1}(s_n \neq \perp, s_{n+1} = \perp)P_F^{\otimes n+1}(s_0, ds\, \overrightarrow{ds_{1:n+1}}) \\ &\quad = \int_{\bar{\mathcal{S}}^{n+1}} \mathbf{1}(s = s_0)f(s, \overrightarrow{s_{1:n}}, \perp)R(ds_n)P_B^{\otimes n}(s_n, ds'\, \overrightarrow{ds_{n-1:1}}\, ds), \end{aligned} \tag{4.26}$$

where $\overrightarrow{s_{1:n}}$ denotes $(s_1, \ldots, s_n)$ and $\overrightarrow{ds_{1:n}}$ denotes $ds_1 \ldots ds_n$.

The following proposition shows an equivalence between the TB and both the FM and reward matching conditions.

Proposition 4.3.20. *If (Z, P_F, P_B) satisfy the TB conditions (4.26) with respect to a measure R, then $F = (\mu, P_B)$, where μ is defined by:*
(1) $\mu(\{\perp\}) = \mu(\{s_0\}) = Z$
(2) $\forall B \in \Sigma_{|\mathcal{S}}$: $\mu(B) = \mu(\{s_0\}) \sum_{n=0}^{\infty} P_F^n(s_0, B)$
satisfies both the flow-matching conditions (4.19) and the reward matching conditions (4.20) with respect to R.

Its proof relies on the following lemma and proposition, which is an extension of Lemma 3.2.5 to measurable pointed graphs:

Lemma 4.3.21. *If (P_F, P_B, Z) satisfy the trajectory balance conditions with respect to R, then for any $n \in \{0, ..., N\}$, and for any measurable bounded function $f : \mathcal{S} \to \mathbb{R}$:*

$$Z \int_{\mathcal{S}} f(s) P_F^n(s_0, ds) P_F(s, \{\perp\}) = \int_{\mathcal{S}} f(s) P_B^n(s, \{s_0\}) R(ds) \tag{4.27}$$

Proposition 4.3.22. *Let P_B be a backward kernel over G. Let $P_{B,T}$ be the measure defined by:*

$$P_{B,T}(s) = \sum_{n=0}^{\infty} P_B^n(s, \{s_0\}) \tag{4.28}$$

We have $\forall s \in \mathcal{S}$:

$$P_{B,T}(s) = 1 \tag{4.29}$$

(⋆) Analogues of the DB and TB conditions for discrete GFlowNets were stated and shown to imply the FM conditions. In the next section, they will be used to construct training objectives for parametric policies.

4.3.5. Training losses for GFlowNets

Above, we have presented three conditions under which a sampler based on a Markov kernel P_F samples from the normalized version of a given reward measure. In practice, similar to discrete GFlowNets, the objects of interest (μ, P_F, P_B, Z) are parameterized by a vector θ, and the goal is to learn θ using gradient-based learning. In this section, we derive losses corresponding to the previous objectives.

We recall the Radon-Nikodym theorem that states that for any two given σ-finite measures p and q on a measurable space $(U, \mathcal{U})$ satisfying $p \ll q$, there exists a measurable function $f : U \to \mathbb{R}_+$, which is unique up to a set of measure zero under q, called the density

or the Radon-Nikodym derivative of p with respect to q, such that:

$$\forall A \in \mathcal{U},\ p(A) = \int_A f(u)q(du). \tag{4.30}$$

This theorem is convenient as it allows to bypass the need to define the measures μ, $P_F(s,-)$, $P_B(s,-)$ on every measurable set, and only requires parameterizing the corresponding densities (with respect to ν, $\kappa(s,-)$, and $\kappa^b(s,-)$ respectively).

Definition 4.3.23 (Losses)**.** Let $u : \mathcal{S} \to \mathbb{R}_+$, $p_F : \mathcal{S} \times \bar{\mathcal{S}} \to \mathbb{R}_+$, and $p_B : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_+$ be three functions, and $Z \in \mathbb{R}_+$ a scalar, all parameterized by a vector θ, and satisfying for every θ:

$$\forall s \in \bar{\mathcal{S}},\ \int_{\bar{\mathcal{S}}} p_F(s, s'; \theta)\kappa(s, ds') = 1 \tag{4.31}$$

$$\forall s' \in \bar{\mathcal{S}},\ \int_{\bar{\mathcal{S}}} p_B(s', s; \theta)\kappa^b(s', ds) = 1 \tag{4.32}$$

The **flow-matching** (FM) loss is defined for every $s' \in \mathcal{S}$ as:

$$L_{FM}(s'; \theta) = \left(\log \frac{\int_{\mathcal{S}} u(s; \theta)p_F(s, s'; \theta)\kappa^b(s', ds)}{u(s'; \theta)}\right)^2$$

The **detailed balance** (DB) loss is defined for every $(s, s') \in \mathcal{S} \times \mathcal{S}$ as:

$$L_{DB}(s, s'; \theta) = \left(\log \frac{u(s; \theta)p_F(s, s'; \theta)}{u(s'; \theta)p_B(s', s; \theta)}\right)^2$$

Denoting by r the density of the reward measure R with respect to the reference measure ν, the **reward-matching** (RM) loss is defined for any $x \in \mathcal{X}$ as:

$$L_{RM}(x; \theta) = \left(\log \frac{u(x; \theta)p_F(x, \perp; \theta)}{r(x)}\right)^2$$

Finally, the **trajectory balance** (TB) loss is defined for every complete trajectory $\tau = (s_0, s_1, \ldots, s_n, s_{n+1}) \in \{s_0\} \times \mathcal{S}^n \times \{\perp\}$ (also denoted $\overrightarrow{s_{0:n+1}}$) where $s_n \in \mathcal{X}$ and $s_{n+1} = \perp$ as:

$$L^n_{TB}(\tau; \theta) = \left(\log \frac{Z(\theta) \prod_{t=0}^{n} p_F(s_t, s_{t+1}; \theta)}{r(s_n) \prod_{t=0}^{n-1} p_B(s_{t+1}, s_t; \theta)}\right)^2.$$

Note that one could derive in a similar fashion a subtrajectory balance loss, similar to the one used in discrete GFlowNets (Madan et al., 2022a).

(⋆) The above losses resemble discrete GFlowNet losses. When the action space is discrete, and the reference measures are the counting measures over vertices of a DAG, $p_F(s, s')$ is a transition probability $P_F(s' \mid s)$. When it is continuous, it represents a conditional probability density over s', given s.

Conversely, from functions $u(-;\theta), p_F(-;\theta), p_B(-;\theta)$, we can define a measure $\mu(-;\theta)$ on $(\bar{\mathcal{S}}, \Sigma)$ whose density with respect to ν is u and forward and backward kernels $P_F(-;\theta), P_B(-;\theta)$ such that $p_F(s,-;\theta)$ and $p_B(s',-;\theta)$ are their densities of with respect to $\kappa(s,-)$ and $\kappa^b(s,-)$, respectively [1].

(⋆) The following theorem ensures that similar to the discrete case, minimizing the losses above leads to samplers of the right probability measure.

Theorem 4.3.24. *(1) If $L_{FM}(-;\theta) = 0$ ν-almost surely, then $F = (\mu, P_F)$ is a flow (i.e., satisfies the flow-matching conditions in Definition 4.3.12).*
(2) If $L_{DB}(-;\theta) = 0$ $\nu \otimes \kappa$-almost surely, then (μ, P_F, P_B) satisfy the detailed balance conditions in Definition 4.3.17.
(3) If $L_{RM}(-;\theta) = 0$ $\nu_{|\mathcal{X}}$-almost surely, then (μ, P_F) satisfies the reward matching conditions in (4.20).
(4) If $L^n_{TB}(-;\theta) = 0$ $((\nu \otimes \kappa^{\otimes n+1})_{|\{s_0\}\times\mathcal{S}^n\times\{\perp\}})$-almost surely for every $n \geq 0$, then $(Z\nu(\{s_0\}), P_F, P_B)$ satisfy the trajectory balance condition in Definition 4.3.19.

An important consequence of Theorem 4.3.24 is that if we can find density functions that achieve zero loss using any of the above objectives almost surely, in addition to the reward-matching loss, then we obtain a way to sample terminating states (i.e., elements of $\mathcal{X}$) proportionally to the reward measure R, according to Theorem 4.3.14.

Training generalized GFlowNets. The FM, DB, and TB losses can be minimized using states (resp. pairs of subsequent states, trajectories) obtained from trajectories sampled from a training policy π, which can be P_F itself (*on-policy*), or a modification of it to encourage exploration (*off-policy*). Thm. 4.3.24 suggests that the parameters θ could be updated with stochastic gradients $\mathbb{E}_{\tau=\overrightarrow{s_{0:n+1}}\sim\pi}[\nabla_\theta \mathcal{L}]$, where $\mathcal{L}$ is $\sum_{t=1}^n L_{FM}(s_t;\theta) + \alpha L_{RM}(s_n;\theta)$, or $\sum_{t=0}^n L_{DB}(s_t, s_{t+1};\theta) + \alpha L_{RM}(s_n;\theta)$ or $L_{TB}(\tau;\theta)$.

4.4. Experiments

4.4.1. Approximating the Jensen-Shannon Divergence

Given an unnormalized target reward measure r with respect to the Lebesgue measure on a bounded space $\mathcal{X}$, and a GFlowNet sampler $P_\top$ of terminating states, we approximate the JSD between the learned sampler and the normalized distribution $R(dx) = \frac{1}{\int_{\mathcal{X}} r(x')dx'} r(x)dx$ as follows:

(1) We sample N points from the target distribution using rejection sampling, with a uniform distribution as a proposal,

1. The measures at $\perp$ are irrelevant.

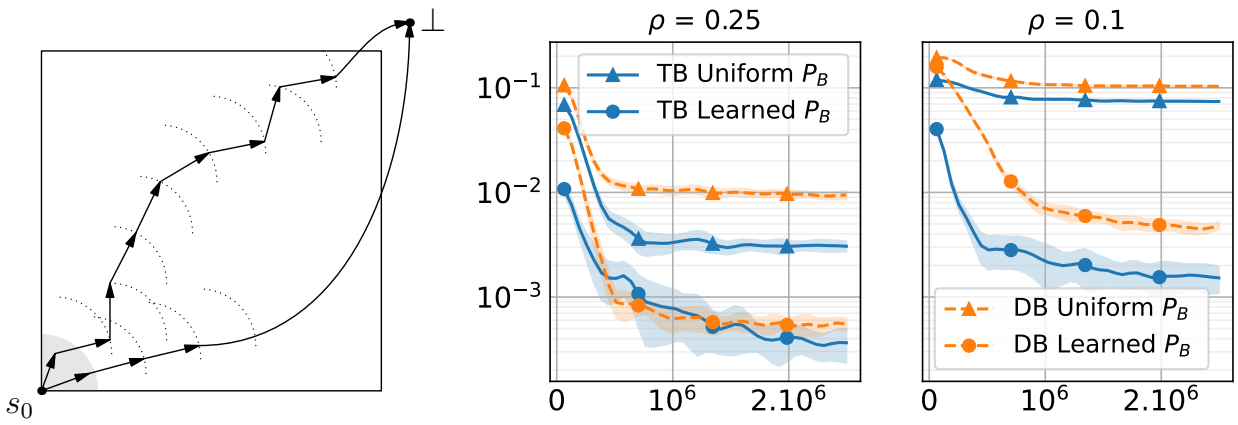


Figure 4.1 – **(a)** Measurable pointed graph structure of the environment in Section 4.4.2: starting at s_0, the first action makes a step within the grey quarter-disc, and subsequent actions make steps of a fixed size or terminate. **(b)** Evolution of the JSD during training of TB and DB, with both a uniform P_B and a learned P_B, for $\rho = 0.25$; **(c)** $\rho = 0.1$. The x-axis is the number of sampled trajectories. Shaded areas represent standard deviations across 6 runs.

(2) We fit a kernel density estimator (KDE, Section 2.1.4) on the above samples,
(3) We fit a second KDE on N samples from $P_\top$,
(4) We use both KDEs to score a fixed set of points defining a discretization of the sample space $\mathcal{X}$,
(5) We normalize both sets of scores in order to obtain valid probability mass functions on the grid,
(6) We evaluate the JSD between the two probability mass functions.

4.4.2. A synthetic continuous environment

Code for these experiments can be found at `https://github.com/saleml/continuous-gfn`.

In this section, we study a synthetic environment inspired by the hypergrid environment (Bengio et al., 2021; Malkin et al., 2022, 2023), with varying trajectory lengths and a pointed graph structure imposing a mixed discrete and continuous probability measure for the policy P_F.

Structure of the state space. The measurable pointed graph is specified by $\mathcal{S} = [0, 1]^2$, and $s_0 = (0, 0)$. A hyperparameter ρ, called the step size, controls the maximal trajectory length. $\kappa(s_0, -)$ is the Lebesgue measure on D_0, the northeastern quarter disk of radius ρ centered at s_0. When $s \neq s_0$, and $\|s\| < 1 - \rho$, $\kappa(s, -)$ is the sum of the one-dimensional Lebesgue arclength measure on C_s^+ (the intersection of the northeastern quarter circle of radius ρ centered at s and $\mathcal{S}$) and the Dirac measure $\delta_\perp$. Finally, when $\|s\| > 1 - \rho$,

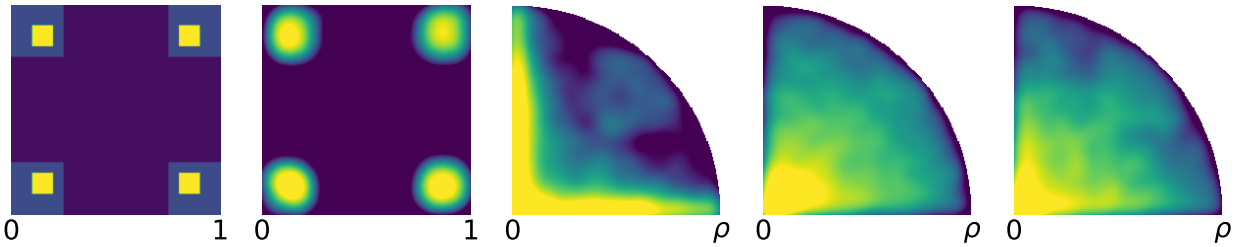


Figure 4.2 – **(a)** Reward density in $[0,1]^2$. **(b)** KDE fit on terminating states of the models trained with TB, $\rho = 0.25$. **(c)** KDE fit on samples from the reward, brought back to D_0 using a **uniform** P_B, corresponding to what $P_F(s_0, -)$ needs to be in order to satisfy DB or TB. A richer search space for the densities $p_F(s, -)$ is required to fit this distribution. **(d)** $P_F(s_0, -)$ for a trained model with learnable P_B. **(e)** The measure induced by a trained P_B on D_0, which matches the learned $P_F(s_0, -)$ in (d).

$\kappa(s, -) = \delta_\perp$. The forward structure is depicted in Figure 4.1(a). The backward reference kernel κ^b is defined similarly.

The reference measure ν is the sum of the Lebesgue measure on $\mathcal{S}$, δ_{s_0}, and $\delta_\perp$. All states besides s_0 are terminating.

The reward measure R on $\mathcal{X}$ is specified by a density function r depicted in Figure 4.2(a). The densities p_F and p_B are parameterized with mixtures of Beta distributions for the continuous components.

The forward and backward kernels P_F, P_B are defined by their densities p_F and p_B with respect to the reference kernels κ, κ^b.

κ, ν, κ^b satisfy the requirements of a finitely absorbing measurable pointed graph. More notably, all states can be reached from s_0 within $1 + \left\lceil \frac{\sqrt{2}}{\rho} \right\rceil$ steps.

The topology $\mathcal{T}$ on $\bar{\mathcal{S}} = \mathcal{S} \cup \{\perp\}$ is the disjoint union topology on $\{s_0, \perp\}$ and $\mathcal{S}$.

We parameterized $p_F(s_0, -)$ using a mixture of four Beta distributions for both the radius $r \in (0, \rho)$ and the angle $\theta \in (0, \frac{\pi}{2})$. We used a mixture of two Beta distributions for the angle $\theta \in (\theta_{min}(s), \theta_{max}(s))$ when modelling $p_F(s, -)$ and $p_B(s, -)$. The forward policy neural network has an extra output head corresponding to the probability of terminating the trajectory, i.e., $p_F(s_0, \perp)$. The learned probabilities were effectively multiplied by the right Jacobians to account for the support of the Beta distributions ($[0, 1]$) being different from that of θ or r.

Reward density. The reward measure R was specified using a density r with respect to the Lebesgue measure λ on $\mathcal{X} = (0, 1)^2$. Following Bengio et al. (2021) and Malkin et al. (2022), the (unnormalized) density is defined for every $x = (x_1, x_2) \in \mathcal{X}$, similar to (3.96), as:

$$r(x) = 0.1 + 0.5 \prod_{i=1}^{2} \mathbf{1}(|x_i - 0.5| \in (0.25, 5]) + 2 \prod_{i=1}^{2} \mathbf{1}(|x_i - 0.5| \in (0.3, 0.4)). \tag{4.33}$$

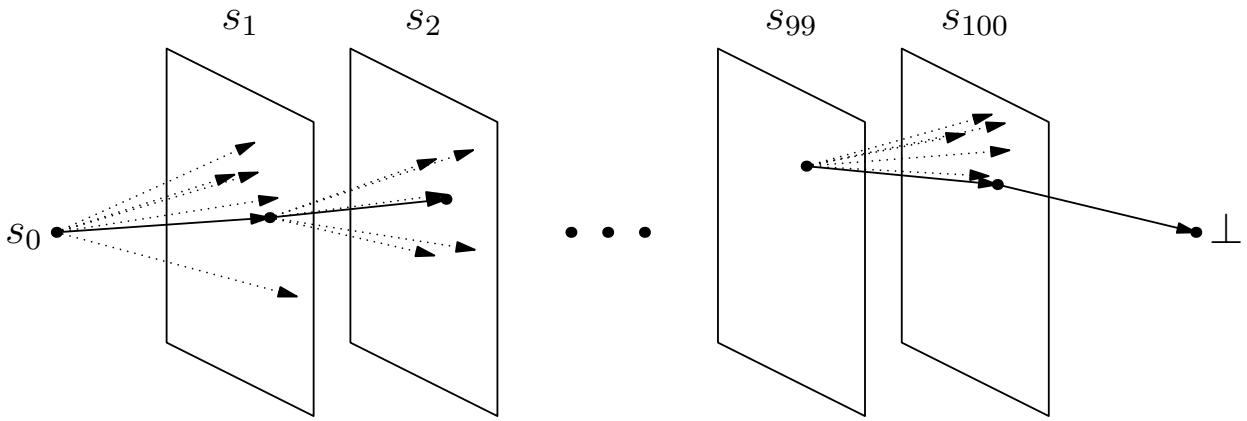


Figure 4.3 – The GFlowNet state space for stochastic control tasks. The solid arrows show a possible sampling trajectory and the dashed arrows show other possible actions, i.e., point to other states in the support of the reference kernel κ.

In Figure 4.1(b,c), we compare DB and TB on two versions of the environment ($\rho \in \{0.1, 0.25\}$), with both a uniform and a learned P_B, using the Jensen-Shannon divergence (JSD, Section 4.4.1) between the learned terminating state distribution and the target distribution as an evaluation metric. The results confirm the findings of Malkin et al. (2022) on the discrete grid domain: the TB loss is more efficient in terms of credit assignment, as it learns to model the target distribution faster and more precisely than DB, and the environment with longer trajectory lengths is harder to model. Additionally, learning a backward policy significantly improves the learning curves of both methods. A justification of the importance of learning in a backward policy is provided in Figure 4.2(c,d,e). Figure 4.2(b) shows a KDE plot fit on terminating states sampled from the model trained with TB on the $\rho = 0.25$ domain. We provide more details in Appendix E.2.1.

4.4.3. Low-dimensional stochastic control

In this section, we show how generalized GFlowNets with a state space of a particular form can be used to learn (discretizations of) stochastic differential equations so as to sample from a black-box target density. We bridge two recent works: Zhang and Chen (2022), from which we borrow the datasets and many parts of the experimental setup, and Malkin et al. (2023), where various algorithms for training stochastic samplers in discrete spaces were considered and whose claims we validate in the continuous case.

We restate the problem considered by Zhang and Chen (2022) in GFlowNet terms. A reward density is given on a Euclidean space $\mathbb{R}^n$ (e.g., the plane in Figure 4.3). The state space is $\mathcal{S} = \{s_0\} \cup (\mathbb{R}^n \times \{1, 2, \ldots, T\})$, where T is the number of moves an agent will make before terminating (here, $T = 100$). Thus the noninitial states are pairs $(\mathbf{x}_t, t)$ where $\mathbf{x}_t \in \mathbb{R}^n$ and $1 \leq t \leq T$; we identify s_0 with $(\mathbf{0}, 0)$. Trajectories begin at s_0 and make successive steps

through the copies of $\mathbb{R}^n$ until reaching the sink state.[2] Learning a forward policy amounts to learning a conditional probability density $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, t)$ over $\mathbb{R}^n$. In particular, if this density is Gaussian, then the policy is the T-step Euler-Maruyama discretization of an Itô stochastic differential equation (SDE).

Zhang and Chen (2022) studied this problem in the case where the backward policy is fixed to be the discretization of a Brownian motion with fixed variance $\frac{\sigma}{T}$ pinned at $(\mathbf{0}, 0)$, and the forward policy is constrained to be Gaussian with the same variance $\frac{\sigma}{T}$ but with learned mean. (The theory of SDEs implies that in the $T \to \infty$ limit, the forward policy P_F that minimizes the GFlowNet loss is indeed Gaussian with the same variance as the fixed P_B.) We thus aim to learn a function $\mu(\mathbf{x}_t, t)$, the mean of the forward policy, that makes the policy sample from the target reward density.

The path integral sampler (PIS) training objective proposed by Zhang and Chen (2022) minimizes the reverse KL divergence between two measures over trajectories: that defined by P_F and that defined by R and P_B. By Theorem 1 of Malkin et al. (2023) (the proof of which trivially generalizes to the continuous case), the gradient of this objective with respect to the parameters of P_F is proportional, in expectation, to that of the TB gradient when trained on-policy. A critical difference between PIS and the on-policy TB objective is that the latter does not require access to the gradient of the reward distribution but treats it as a black box.

Datasets, algorithms, and baselines. We evaluate GFlowNets and baselines on two synthetic densities: a 2-dimensional mixture of 9 Gaussians and the 10-dimensional funnel from MCMC literature (Hoffman and Gelman, 2011).

In addition to GFlowNet TB, we evaluate the two algorithms for minimizing divergences between trajectory measures studied by Malkin et al. (2023): the reverse KL optimized via policy gradient – equivalent in expectation to TB – and the forward KL, for which gradient estimation requires importance weighting. We also evaluate the algorithms in an off-policy setting, where the training trajectories are sampled with additional variance injected into the policy to encourage exploration (see Appendix E.2 for details). We include baselines from Zhang and Chen (2022) as well.

All algorithms use the same model architecture as the PIS baseline for $\mu(\mathbf{x}_t, t)$ and are evaluated using two metrics as defined in Zhang and Chen (2022): the log-partition function estimation bias using simple and importance-weighted variational bounds, as defined in Appendix E.2.2.

Results and discussion. From the results in Table 4.2, and the extended results in Table E.1, we conclude that the two main observations of Malkin et al. (2023) continue to

2. To be precise, the reference measure ν is the Lebesgue measure on each copy of $\mathbb{R}^n$ and the counting measure on $\{s_0\}$. If s_i is a state in the i-th copy of $\mathbb{R}^n$ (if $i > 0$) or the initial state s_0 (if $i = 0$), the reference kernel $\kappa(s_i, -)$ is Lebesgue on the $(i+1)$-st copy of $\mathbb{R}^n$ if $i < T$ and $\delta_\perp$ if $i = T$; κ^b is defined similarly.

Black box?			Gaussian mixture	Funnel
✓	Off-policy	GFlowNet TB	$\mathbf{-0.003} \pm 0.011$	$\mathbf{-0.026} \pm 0.020$
✓	Off-policy	Reverse KL*	-1.609 ± 0.546	–
✓	Off-policy	Forward KL*	$\mathbf{-0.001} \pm 0.013$	-0.087 ± 0.081
✓	On-policy	GFlowNet TB	-1.301 ± 0.434	$\mathbf{-0.012} \pm 0.108$
✓	On-policy	Reverse KL	-1.237 ± 0.413	-0.040 ± 0.023
✓	On-policy	Forward KL*	-0.007 ± 0.023	-0.034 ± 0.143
✓	Non-SDE	SMC	-0.362 ± 0.293	-0.216 ± 0.157
×	On-policy	PIS-NN	-1.192 ± 0.482	-0.018 ± 0.020
×	Non-SDE	HMC	-1.876 ± 0.527	-0.835 ± 0.257

Table 4.2 – Log-partition function estimation bias using importance-weighted bound B_{RW} (mean and standard deviation over 10 runs). The **bold** value in each column shows the best result and all those statistically equivalent to it ($p > 0.1$ under a Welch's t-test). Algorithms assuming access to the gradient of the reward (non-black-box) are shown for comparison. Rows marked with * require importance weighting for gradient estimation. Cells with – were unstable to optimize. Last three rows from Zhang and Chen (2022).

hold in this continuous setting. First, as expected, on-policy TB and reverse KL perform similarly when both can be stably optimized. Second, in settings where off-policy exploration is important, TB is more stable and achieves a better fit to the target than the other objectives, which require importance weighting for gradient estimation. Figure E.1 shows that exploration is necessary to discover modes. Finally, we note that TB is competitive with the PIS objective despite not having access to gradients of the reward density.

4.4.4. Stochastic control on a torus

We consider a variant of the samplers discussed in Section 4.4.3 to model reward densities on the surface of a 2D torus. Distributions over tori are helpful to model torsion angles in molecular conformations, as we illustrate in Appendix E.2.3 and Figure E.2 with the alanine dipeptide molecule.

To model the surface of a torus, the measurable pointed graph is defined by $\mathcal{S} = \{s_0\} \cup [0, 2\pi)^2 \times \{t \in \mathbb{N}, 1 \leq t \leq T\}$, where t denotes the step number and T the trajectory length, and $s_0 = ((0, 0), 0)$. Note that here $[0, 2\pi)$ has the topology of the circle, not that induced from the real line.

We consider two reward densities: a synthetic multimodal density and a density based on the energy $\mathcal{E}$ of the alanine dipeptide molecule as a function of two of the angles defining

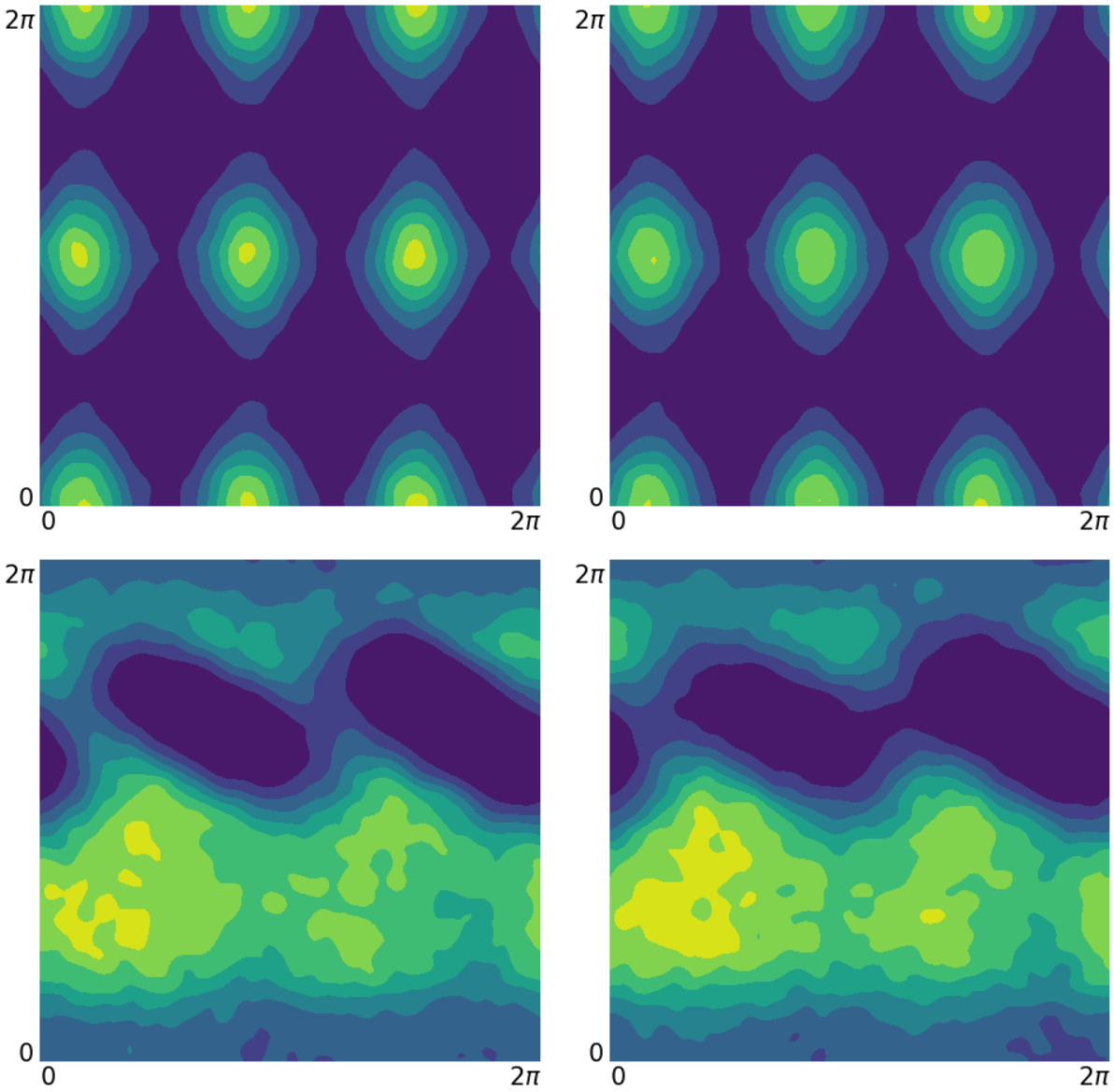


Figure 4.4 – KDEs fit on samples from the reward functions (**a**: synthetic multimodal reward function, **c**: Boltzmann distribution of principal torsion angles of the alanine dipeptide molecule – details in Appendix E.2.3) and on samples from the corresponding trained GFlowNets (**b, d**). The topology of the torus imposes periodic boundary conditions on $[0, 2\pi)^2$.

the molecule's conformation. More details are provided in Appendix E.2.3. We visually represent learned and reward distributions in Figure 4.4.

		Number of variables (d)		
		3	4	5
Graphs	BCD Nets	–	2.13×10^{-1}	2.61×10^{-1}
	DiBS	3.28×10^{-1}	2.95×10^{-1}	3.15×10^{-1}
	GFlowNet	$\mathbf{1.50 \times 10^{-2}}$	$\mathbf{1.61 \times 10^{-2}}$	$\mathbf{1.80 \times 10^{-2}}$
Params.	BCD Nets	–	2.17×10^{2}	2.63×10^{2}
	DiBS	5.87×10^{2}	1.12×10^{3}	2.12×10^{3}
	GFlowNet	$\mathbf{-1.75 \times 10^{0}}$	$\mathbf{-3.06 \times 10^{0}}$	$\mathbf{-5.17 \times 10^{0}}$

Table 4.3 – Comparison between GFlowNets and other methods based on variational inference on the Bayesian structure learning task. (Graphs) RMSE between the estimated edge marginals and the exact edge marginals. (Params.) Average negative log probability of the parameter samples under the exact posterior $P(\theta \mid G, \mathcal{D})$.

4.4.5. Posterior over continuous parameters in Bayesian structure learning

To show the capacity of GFlowNets to model a distribution over a mixed space of discrete and continuous quantities, we study the problem of learning the structure of a Bayesian network and its parameters from a Bayesian perspective. Extending the work of Deleu et al. (2022), our goal here is to approximate the (joint) posterior distribution $P(G, \theta \mid \mathcal{D})$ over the directed acyclic graph (DAG) structure G of the Bayesian Network (discrete component) and the parameters θ of its conditional probability distributions (continuous component), given a dataset of observations $\mathcal{D}$.

We use a GFlowNet structured as follows: starting from the empty graph, the DAG G is first generated one edge at a time, following the structure of DAG-GFlowNet (Deleu et al., 2022). Once the graph G has been entirely generated, we sample the parameters θ associated with it to reach a valid terminating state (G, θ). Details about the state space and the forward transition probability are given in Appendix E.2.4. We use the subtrajectory balance loss (Madan et al., 2022a) to train the GFlowNet with $R(G, \theta) = P(\mathcal{D} \mid \theta, G)P(\theta, G)$ as a reward function.

To evaluate our approximation against the target distribution, we consider problems where the true posterior $P(G, \theta \mid \mathcal{D})$ may be computed in closed form. More precisely, we assume the Bayesian network follows a linear-Gaussian model and the number of random

Method	FID↓	NLL↓
Baseline	17.65	4.57
MLE-GFN	16.36	4.47

Table 4.4 – ImageNet-32 results.

variables $d \leq 5$. Additional details about the experimental settings and metrics are available in Appendix E.2.4. In Section 4.4.5, we compare the performance of the GFlowNet with two baseline methods based on variational inference: DiBS (Lorch et al., 2021) and BCD Nets (Cundy et al., 2021). In Section 4.4.5 (top), we report the root-mean-square error (RMSE) between the edge marginals computed with the approximation and the exact posterior $P(G \mid \mathcal{D})$; we observe that the model learned by the GFlowNet is significantly more accurate on the discrete component, supporting the observation made in Malkin et al. (2023). Moreover, in Section 4.4.5 (bottom), we observe that the sampled θ from the GFlowNet are significantly more likely under the exact posterior $P(\theta \mid G, \mathcal{D})$, suggesting that the GFlowNet's approximation of the continuous component is also more accurate.

4.4.6. Connections with diffusion models

We show how the generalized GFlowNet framework can be applied *beyond* the setting of fitting a sampler to a target reward function. As established in Zhang et al. (2023b), GFlowNets can also be trained to *maximize likelihood* of a given set of terminating states with an algorithm called MLE-GFN. Here we apply MLE-GFN to generalized GFlowNets to improve denoising diffusion probabilistic models (DDPMs; Ho et al., 2020b).

The generative process in a DDPM can be seen as a particular case of the sampling process in a generalized GFlowNet of the same form as in Section 4.4.3 and Figure 4.3. A fixed number of steps T is made through a sequence of copies of a high-dimensional space $\mathbb{R}^n$ (with the i-th state in the trajectory representing, e.g., an image at noise level $T - i$). The policy at the first step, from s_0 to $(\mathbf{x}_1, 1)$, is constrained to be unit Gaussian, while subsequent steps are conditional Gaussians with a known variance.

While DDPMs typically fix the noising process – corresponding to the backward policy P_B in the GFlowNet – and learn only the denoiser (forward process), MLE-GFN allows learning both P_F and P_B as Gaussian policies. The description and proof of the soundness of MLE-GFN, as well as details of the parameterization of means and variances, can be found in Zhang et al. (2023b).

We train a GFlowNet as described above on the ImageNet-32 dataset (treated as a set of terminating states) with $T = 100$ steps. Table 4.4 demonstrates the efficacy of our method compared to the DDPM baseline in terms of both the sample quality (FID) and density modelling (NLL). We defer other details and example images to Appendix E.2.5.

Chapter 5

Direct Epistemic Uncertainty Prediction

This chapter is based upon the following paper:

— Lahlou et al. (2021): "DEUP: Direct Epistemic Uncertainty Prediction" - Salem Lahlou*, Moksh Jain*, Hadi Nekoei, Victor Ion Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov, Yoshua Bengio, published in 2023 in Transactions on Machine Learning Research (TMLR).

Sections 5.1 and 5.4 are shortened versions of Sections 1 and 4 of Lahlou et al. (2021), to avoid repetitions of content discussed in Chapter 2. The rest of the chapter is identical to Lahlou et al. (2021), except for some content that was moved from the appendix to the main body of text, to improve the reading flow, and to highlight the contributions of the author the thesis.

5.1. Introduction

Expanding on Section 2.1.7, we study in this chapter alternative ways of quantifying uncertainty. While probabilities have been widely used to represent uncertainty, there is not a generally agreed upon definition of uncertainty, let alone a way of quantifying it as a number, as confirmed by a meta-analysis performed by Zidek and Van Eeden (2003). In supervised learning, Bayesian methods, which aim at predicting a conditional probability distribution on the variable to predict, called the *posterior predictive*, are natural candidates for providing uncertainty estimates. Bayesian learning is however more computationally expensive than its frequentist counterpart, and generally relies on approximations such as MCMC (Brooks et al., 2011) and Variational Inference (Blei et al., 2017) methods, both introduced in Section 2.2. Prominent uncertainty estimation methods, such as MC-Dropout (Gal and Ghahramani, 2016b) and Deep Ensembles (Lakshminarayanan et al., 2017b), introduced in Section 2.2.3, both of which are approximate Bayesian methods (Hoffmann and Elster, 2021), use the posterior predictive variance as a measure of uncertainty: if multiple neural nets that are

all compatible with the data make different predictions at x, the discrepancy between these predictions is a strong indicator of epistemic uncertainty at x.

Pitfalls of using the Bayesian posterior to estimate uncertainty: Epistemic uncertainty in a predictive model, seen as a measure of lack of knowledge, consists of *approximation uncertainty* - due to the finite size of the training dataset, and *model uncertainty* - due to model misspecification (Hüllermeier and Waegeman, 2019), also called bias. The latter is not accounted for when using variance or entropy as a measure of EU, but has been shown to be important for generalization (Masegosa, 2020) as well as within interactive learning settings like optimal design (Zhou et al., 2003). Approximate Bayesian methods that are widely used in machine learning often suffer from model misspecification, for instance due to the implicit bias induced by SGD (Kale et al., 2021) and the finite computational time. Figure 5.1 illustrates with a Gaussian Process (GP) the suboptimal behavior of the GP to estimate EU under model misspecification. The confidence intervals provided by the GP are underestimated when the model is misspecified, which can lead to confidently wrong predictions and poor decisions (see Figure 5.1, bottom left).

As a **first contribution**, we systematically analyze the sources of uncertainty and misspecification, and analyze the pitfalls of using discrepancy-based measures of EU (such as variance of the Bayesian posterior predictive), given that they miss out on model uncertainty, which we define as a component of the excess risk, i.e., the gap between the risk (or out-of-sample loss) of the predictor at a point x and that of the Bayes predictor at x (the one with the lowest expected loss, that no amount of additional data could reduce).

As a **second contribution**, we take a step back by considering the fundamental notion of epistemic uncertainty as lack of knowledge, and based on this, we propose to estimate the excess risk as a measure of EU. This leads us to our proposed framework **DEUP**, for **Direct Epistemic Uncertainty Prediction**, where we train a secondary model, called the error predictor, with an appropriate objective and appropriate data, to estimate the pointwise generalization error (the risk), and then subtract an estimate of aleatoric uncertainty if available, or provide an upper bound on EU otherwise. Figure 5.1 illustrates in a toy task in which held-out data is available, that the epistemic uncertainty estimated by DEUP is more useful to an interactive learner than those provided by a misspecified GP. It is important to note that, in these settings of interest, DEUP does not use any additional held-out data.

Accounting for bias when measuring EU is particularly useful to an interactive learner whose effective capacity depends on the training data, for instance a neural network, as it can reduced with additional training data, especially in regions of the input space we care about, i.e., where EU is large. DEUP is agnostic to the type of predictors used, and in interactive settings, it is agnostic to the particular search method still needed to select points with high EU in order to propose new candidates for active learning (Aggarwal et al., 2014; Nguyen

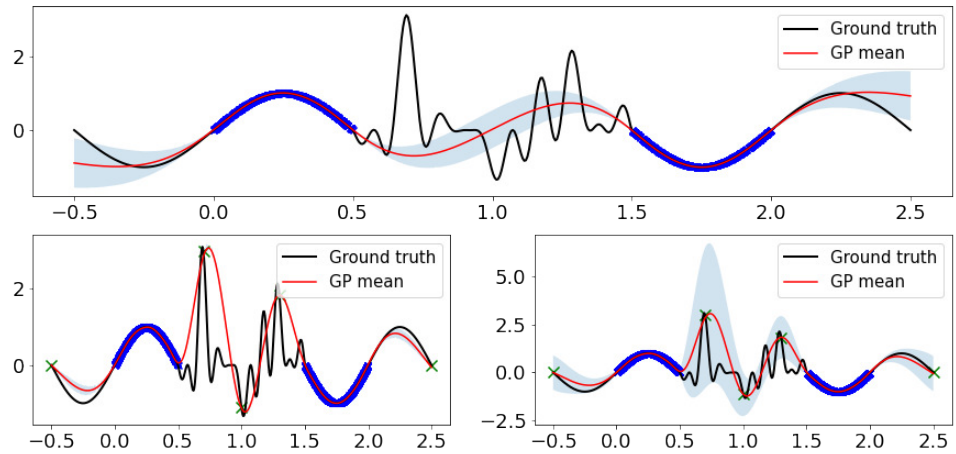


Figure 5.1 – *Top.* A GP is fit on (dark blue) points in $[0, 0.5] \cup [1.5, 2]$. The shaded area represents the GP standard deviation, often used as a measure of epistemic uncertainty: note how it fails badly to enclose the ground function in $[0.5, 1.5]$. *Bottom left.* A second GP fit on the same points, with 5 extra points (green crosses). This second GP predicts almost 0 standard deviation everywhere, even though the first GP significantly underestimated the uncertainty in $[0.5, 1.5]$, because no signal is given to the learner that the region $[0.5, 1.5]$ should be explored more. *Bottom right.* DEUP learns to map the underestimated variances of the first GP to the L2 errors made by that GP, and yields more reasonable uncertainty estimates that should inform the learner of what area to explore.

et al., 2019; Bengio et al., 2021), SMO (Kushner, 1964; Jones et al., 1998; Snoek et al., 2012) or exploration in RL (Kocsis and Szepesvári, 2006; Osband et al., 2016; Janz et al., 2019).

A unique advantage of DEUP, compared with discrepancy-based measures of EU, is that it can be explicitly trained to care about, and calibrate for estimating the uncertainty for examples which may come from a distribution slightly different from the distribution of most of the training examples. Such distribution shifts (referred to as *feedback covariate shift* in Fannjiang et al. (2022)) arise naturally in interactive contexts such as RL, because the learner explores new areas of the input space. In these non-stationary settings, we typically want to retrain the main predictor as we acquire new training data, not just because more training data is generally better but also to better track the changing distribution and generalize to yet unseen but upcoming out-of-distribution (OOD) inputs. This setting makes it challenging to train the main predictor but it also entails a second non-stationarity for *the training data seen by the error predictor*: a large error initially made at a point x before x is incorporated in the training set (along with an outcome y) will typically be greatly reduced after updating the main predictor with (x, y). To cope with this non-stationarity in the targets of the error predictor, we propose, as a **third contribution**, to use additional features as input to the error predictor, that are informative of both the input point *and the dataset* used to obtain

the current predictor. We mainly use density estimates and model variance estimates, that sometimes come at no additional cost.

The remainder of this chapter is divided as follows:

— In Section 5.2, we describe a theoretical framework for analysing sources of uncertainty and describe a pitfall of discrepancy-based measures of epistemic uncertainty.

— In Section 5.3, we present DEUP, a principled model-agnostic framework for estimating epistemic uncertainty, and describe how it can be applied in a fixed dataset as well as interactive learning settings, and propose means to address the non-stationarities arising in the latter.

— In Section 5.4, we discuss related work on uncertainty estimation as well as error predictors in machine learning.

— In Section 5.5, we experimentally validate that EU estimates from DEUP can improve upon existing SMO methods, drive exploration in RL, and evaluate the quality of these uncertainty estimates in probabilistic image classification and in a regression task predicting synergies of drug combinations.

5.2. Excess Risk, Epistemic Uncertainty, and Model Misspecification

In this section, we analyze the sources of lack of knowledge through the lens of the excess risk of a predictor, define model misspecification, and discuss the pitfall of using discrepancy-based measures of uncertainty when the model is misspecified. The section is divided as follows:

— In Section 5.2.1, we define the mathematical framework and notations used in the analysis.

— In Section 5.2.2, we characterize the sources of lack of knowledge, and argue that estimating the excess risk is a sound measure of epistemic uncertainty. This is the main rationale behind our framework, DEUP, presented in Section 5.3.

— In Section 5.2.3, we briefly analyze the sub-optimal behavior of the Bayesian posterior when the model is misspecified, and explain why in practice models are generally biased, even non-parametric ones.

5.2.1. Notations and Background

Predictive models tackle the problem of learning to predict outputs $y \in \mathcal{Y}$ given inputs $x \in \mathcal{X} \subseteq \mathbb{R}^d$ using a function $f : \mathcal{X} \to \mathcal{A}$ estimated from a training dataset $z^N := (z_1, \ldots, z_N) \in \mathcal{Z}^N$, where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $z_i = (x_i, y_i)$, and the unknown ground-truth generative model is defined by $P(X, Y) = P(X)P(Y \mid X)$. In decision theory, the set $\mathcal{A}$ is called the action space, and is typically either equal to $\mathcal{Y}$ for regression problems, or to

$\Delta(\mathcal{Y})$, the set of probability distributions on $\mathcal{Y}$, for classification problems. Given a loss function $l : \mathcal{Y} \times \mathcal{A} \to \mathbb{R}^+$ (e.g., $l(y, a) = \|y - a\|^2$ for regression, and $l(y, a) = -\log a(y)$ for classification), the **point-wise risk** (or **expected loss**) of a predictor f at $x \in \mathcal{X}$ is defined as:

$$R(f, x) = \mathbb{E}_{P(Y|X=x)}[l(Y, f(x))]. \tag{5.1}$$

We define the **risk** (or total expected loss) of a predictor as the marginalization of (5.1) over x:

$$R(f) = \mathbb{E}_{P(X,Y)}[l(Y, f(X))]. \tag{5.2}$$

Given a hypothesis space $\mathcal{H}$, a subset of $\mathcal{F}(\mathcal{X}, \mathcal{A})$, the set of functions $f : \mathcal{X} \to \mathcal{A}$, the goal of any learning algorithm (or learner) $\mathcal{L}$ is to find a predictive function $h^* \in \mathcal{H}$ with the lowest possible risk:

$$h^* = \arg\min_{h \in \mathcal{H}} R(h). \tag{5.3}$$

Naturally, the search for h^* is elusive given that $P(X, Y)$ is usually unknown to the learner. Instead, the learner $\mathcal{L}$ maps a finite training dataset z^N to a predictive function $\mathcal{L}(z^N) = h_{z^N} \in \mathcal{H}$ minimizing an approximation of (5.2), called the **empirical risk**:

$$R_{z^N}(h) = \frac{1}{N} \sum_{i=1}^{N} l(y_i, h(x_i)) \quad \text{where } z_i = (x_i, y_i) \tag{5.4}$$

$$h_{z^N} = \arg\min_{h \in \mathcal{H}} R_{z^N}(h). \tag{5.5}$$

The dataset z^N need not be used solely to define the empirical risk as in (5.4). The learner can for example use a subset of z^N as a validation set to tune its hyperparameters and thus its *effective capacity* (Arpit et al., 2017).

5.2.2. Sources of lack of knowledge

Clearly, a high value of $R(h_{z^N}, x)$ indicates lack of knowledge around x. Similar to Hüllermeier and Waegeman (2019), we characterize the three sources of this lack of knowledge, illustrated in Figure 5.2, as follows:

— (5.1) has a fundamental limiting lower bound, usually reached at a function f^* called the **Bayes predictor** [1]:

$$f^*(x) = \arg\min_{a \in \mathcal{A}} \mathbb{E}_{P(Y|X=x)}[l(Y, a)]. \tag{5.6}$$

$R(f^*, x) > 0$ indicates an irreducible risk due to the inherent randomness of $P(Y \mid X = x)$. $R(f^*, x)$ is thus a measure of **aleatoric uncertainty** at x. Note that there might be

1. An equivalent definition is $f^* = \arg\min_{f \in \mathcal{F}(\mathcal{X}, \mathcal{A})} R(f)$.

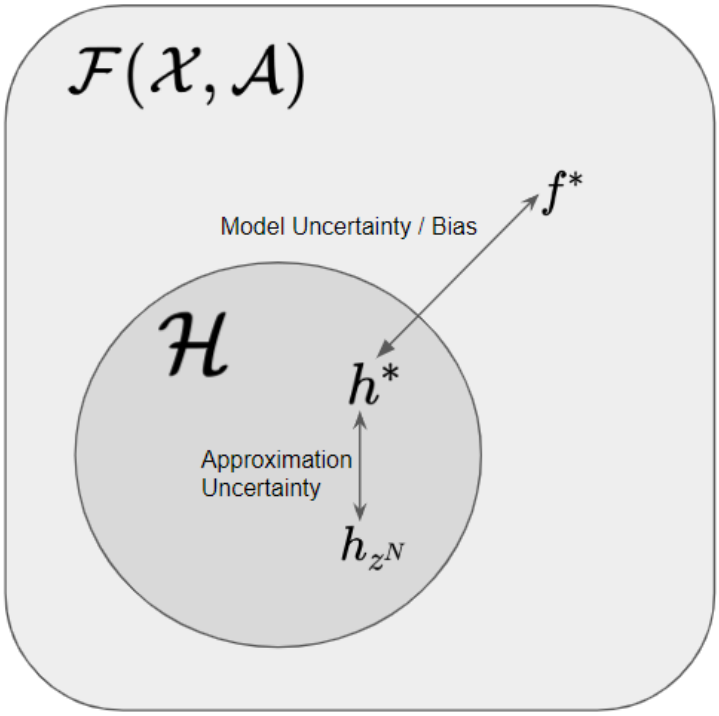


Figure 5.2 – Graphical representations of the two components of epistemic uncertainty. Inspired by Fig. 4 of Hüllermeier and Waegeman (2019).

more than one Bayes predictor, but by definition, they all have the same point-wise risk, which we denote as A:

$$A(x) = R(f^*, x) \tag{5.7}$$

— Minimizing the risk over $\mathcal{H}$ rather than $\mathcal{F}(\mathcal{X}, \mathcal{A})$ induces a discrepancy between the predictor that is optimal in $\mathcal{H}$, h^*, and the Bayes predictor f^*, usually referred to as **model uncertainty** (our hypothesis space could be improved). This can be seen as a form of **bias**, as the optimization is limited to functions in $\mathcal{H}$.

— Minimizing the empirical risk instead of the risk induces a discrepancy between h_{z^N} and h^*, called the **approximation uncertainty**, due both to the finite training set and finite computational resources for training.

Borrowing terminology from Futami et al. (2022), we use the Bayes predictor to define the excess risk as follows:

Definition 5.2.1. The **excess risk** of a predictor $f : \mathcal{X} \to \mathcal{A}$ at x is the gap between the point-wise risk and its fundamental limit:

$$\mathrm{ER}(f, x) = R(f, x) - A(x). \tag{5.8}$$

Both model uncertainty and approximation uncertainty are indicative of the epistemic state of the learner, and can be reduced with better choices (of $\mathcal{H}$) and more data respectively. This implies that an estimator of the excess risk of f at x (Definition 5.2.1) can be used as a measure of **epistemic uncertainty**. ER is intimately linked to the minimum excess risk (Xu and Raginsky, 2020), a measure of the gap between the minimum expected loss attainable by learning from data and $A(x)$, which was shown to decrease and converge towards 0 as the training dataset size grows to infinity, which is a desirable property for EU measures. Using an estimator of ER as a measure of EU is the main idea behind DEUP, and will be expanded upon in Section 5.3.

The following examples illustrate the concepts introduced thus far:

Example 5.2.2. *Consider a univariate regression problem with Gaussian ground truth, i.e., $\mathcal{Y} = \mathbb{R}$ and there exist functions μ and σ such that $P(Y \mid X = x) = \mathcal{N}(Y; \mu(x), \sigma^2(x))$, with the squared loss $l(y, a) = (y - a)^2$. Then, the Bayes predictor is $f^* = \mu$, which has a point-wise risk $x \mapsto A(x) = \sigma^2(x)$, and for any predictor $f : \mathcal{X} \to \mathcal{R}$, and for every $x \in \mathcal{X}$:*

$$R(f, x) = \sigma^2(x) + (f(x) - \mu(x))^2$$
$$\mathrm{ER}(f, x) = (f(x) - \mu(x))^2$$

Example 5.2.3. *Consider a classification problem with K classes, i.e., $\mathcal{Y} = \{1, \ldots, K\}$ and there exists a function $\mu : \mathcal{X} \to \Delta^K$, where Δ^K is probability simplex of dimension $K - 1$, such that $P(Y \mid X = x)$ is the categorical distribution with probability mass function $\mu(x)$, with the log loss $l(y, a) = -\log a(y)$. Then, the Bayes predictor is $f^* = \mu$, which has a point-wise risk $x \mapsto A(x) = \mathbb{H}_{P(Y|X=x)}[Y \mid x]$, the entropy of the ground-truth label distribution, and for any predictor $f : \mathcal{X} \to \Delta^K$, and for every $x \in \mathcal{X}$:*

$$R(f, x) = \mathbb{H}(\mu(x), f(x))$$
$$\mathrm{ER}(f, x) = D_{\mathrm{KL}}(\mu(x), f(x)),$$

where $\mathbb{H}(.\ ,\ .)$ and $D_{\mathrm{KL}}(.\ ,\ .)$ denote respectively the cross entropy and the Kullback-Leibler (KL) divergence between two distributions.

It is clear from the definitions above, and from Figure 5.2 that the choice of the subset $\mathcal{H}$ can contribute to higher epistemic uncertainty, as it introduces **misspecification** (Masegosa, 2020; Hong and Martin, 2020; Cervera et al., 2021):

Definition 5.2.4. A learner with hypothesis space $\mathcal{H} \subseteq \mathcal{F}(\mathcal{X}, \mathcal{A})$ is said to be learning under model misspecification if $f^* \notin \mathcal{H}$, where f^* is the Bayes predictor.

While there is no agreed upon measure of misspecification, some authors (Masegosa, 2020; Hong and Martin, 2020) focus on Bayesian or approximate Bayesian learners, which maintain a distribution over predictors, and use a discrepancy measure between the best

reachable posterior predictive $p(Y \mid X)$ defined by functions $h \in \mathcal{H}$, and the ground-truth likelihood $P(Y \mid X)$. Alternatively, and assuming the function space $\mathcal{F}(\mathcal{X}, \mathcal{A})$ is endowed with a metric, misspecification (or **bias**) can be defined as the distance between h^* and f^*.

Additionally, and as argued by many authors (see e.g., Cervera et al. (2021); Knoblauch et al. (2022)), a common implicit assumption in (approximate) Bayesian methods is correct model specification (i.e., there is no bias). In the next subsection, we will briefly review why this assumption is rarely satisfied in practice, and what it entails in terms of uncertainty modelling.

5.2.3. Bayesian uncertainty under model misspecification

Consider a parametric model $p(Y \mid X; \theta)$ and a learner maintaining a distribution over parameters $\theta \in \Theta$, each corresponding to a predictor h in a parametric set of functions $\mathcal{H}$, possibly starting from a prior $p(\theta)$ that would lead to a posterior distribution $p(\theta \mid z^N)$ upon observing data z^N. Clearly, the fact that multiple θ's and corresponding values of h are compatible with the data and the prior indicates lack of knowledge. Because the lack of knowledge indicates where an interactive learner should acquire more information, this justifies the usage of dispersion measures, such as the variance or the entropy of the posterior predictive, as measures of EU.

While such dispersion measures are indicative of approximation uncertainty, they do not account for model misspecification, or bias. The sub-optimal behavior of Bayesian methods with misspecified models has already been studied in terms of the quality of predictions (Grünwald, 2012; Walker, 2013; Grünwald and Van Ommen, 2017) as well as their generalization properties (Masegosa, 2020). These works show that even though the Bayesian posterior predictive concentrates around the best model $p(Y \mid X; \tilde{\theta})$ with minimum KL divergence to the ground truth $P(Y \mid X)$ within $\mathcal{H}$, it does not fit the data distribution perfectly. Additionally, Kleijn and van der Vaart (2012) provide a theoretical analysis of the sub-optimality of the uncertainty estimates under model misspecification. More specifically, the authors derive a misspecified version of the Bernstein-Von Mises theorem, which implies that the Bayesian credible sets are valid confidence sets when the model is well specified, but prove that under misspecification, the credible sets may over-cover or under-cover, and are thus not valid confidence sets. More recently, Cervera et al. (2021) showed that a Bayesian treatment of a misspecified model leads to unreliable epistemic uncertainty estimates in regression settings, while D'Angelo and Henning (2021) showed that uncertainty estimates in misspecified models lead to undesirable behaviors when used to detect OOD examples.

Sources of model misspecification: In the Bayesian framework, the hypothesis class $\mathcal{H}$ and the corresponding set of posterior predictive distributions $p(Y \mid X)$ are only implicitly defined, through the choices of the prior $p(\theta)$, the likelihood functions $p(Y \mid X; \theta)$, and the

computation budget. Even if *in theory*, the model is well specified, in the sense that that there exists $\theta^* \in \Theta$ such that $p(Y \mid X; \theta^*) = P(Y \mid X)$ [2], *in practice*, the combination of the prior and the finite computational budget limits the set of reachable θ, resulting in a systematic bias (Knoblauch et al., 2022). This behavior is exacerbated in low data regimes, even with modern machine learning models such as neural networks, because the learner may not use all of its capacity due to the implicit (and not fully understood) regularization properties of stochastic gradient descent (SGD, Kale et al. (2021)), explicit regularization, early stopping, or a combination of these. "All models are wrong, but some are useful" - Box (1976).

The importance of bias in interactive settings: Naturally, bad uncertainty estimates also have adverse effects on the performances of interactive learners, that use said estimates to guide the acquisition and evaluation of more inputs (Zhou et al., 2003). In fact, it has long been known (Box and Draper, 1959) that model uncertainty (also called bias in Box and Draper (1959)) is just as important as, if not more than, approximation uncertainty (interestingly referred to as variance in Box and Draper (1959)), for the problem of optimal design. Therefore, a learner using approximation uncertainty alone, measured by the variance of the posterior predictive, can be confidently wrong. When the acquisition of more inputs is guided by these predictions, this can slow down the interactive learning process. In Deep Ensembles (Lakshminarayanan et al., 2017b) for example, if all the networks in the ensemble tend to fail in a systematic (i.e., potentially reducible) way, this aspect of prediction failure will not be captured by variance, or approximation uncertainty. With flexible models like neural networks however, the hypothesis space $\mathcal{H}$ is defined not only through the chosen architecture of the network, but also through the pre-defined training procedure (e.g., the hyperparameters of the optimizer, the stopping criterion based on the performance on a validation set chosen from the training data...). This means that even though the learning strategy is data-independent, the hypothesis space $\mathcal{H}$ depends on the training data. Arpit et al. (2017) formalized this subtle distinction using the notion of "effective capacity". As a consequence, the learner can incorporate new training examples in areas of the input space where the bias is large, in order to change the hypothesis space $\mathcal{H}$ to $\mathcal{H}'$, which can be closer to the Bayes predictor, thus reducing the bias, and de facto the excess risk ER.

As a consequence of the ubiquity of misspecified models, it is important to question the status quo of relying on discrepancy-based measures of epistemic uncertainty in machine learning, and explore alternative ways of measuring EU, such as DEUP, which we introduce in the next section.

2. Note how we use upper case P for the ground truth posterior and lower case p for the likelihood and the Bayesian posterior predictive estimator.

5.3. Direct Epistemic Uncertainty Prediction

As argued in the previous section, the excess risk (5.8) of a predictor f at a point x can be used as a measure of EU that is robust against model misspecification. Estimating the excess risk $\mathrm{ER}(f, x)$ requires an estimate of the expected loss $R(f, x)$, and an estimate of the aleatoric uncertainty $A(x)$ (5.7).

DEUP (Direct Epistemic Uncertainty Prediction) **uses observed out-of-sample errors in order to train an error predictor** $x \mapsto e(f, x)$ estimating $x \mapsto R(f, x)$. These may be in-distribution or out-of-distribution errors, depending on what we care about and the kind of data that is available. Given a predictor $x \mapsto a(x)$ of aleatoric uncertainty, $u(f, x)$ defined by:

$$u(f, x) = e(f, x) - a(x), \tag{5.9}$$

becomes an estimator of $\mathrm{ER}(f, x)$ and as a consequence, an estimator of the epistemic uncertainty of f at x. Before delving into the details of DEUP, we briefly discuss how one can obtain an estimator a of aleatoric uncertainty.

How to estimate aleatoric uncertainty? We consider three possible scenarios for the aleatoric uncertainty:

(1) If we know that $A(x) = 0$, i.e., that the data-generating process is noiseless, then $u(f, x)$ is an estimate of $R(f, x)$, and $a(x)$ can safely be set to 0. This is the case in the experiments we present in Section 5.5.1.

(2) In regression settings with the squared loss, where $A(x)$ equals the variance of $P(Y \mid X = x)$ (Example 5.2.2), if we have access to an oracle that samples Y given a query x from the environment $P(Y \mid x)$ (e.g., in active learning or SMO), then $A(x)$ can be estimated using the empirical variance of different outcomes of the oracle at the same input x. It is common practice for example to perform replicate experiments in biological assays and use variation across replicates to estimate aleatoric uncertainty (Lee et al., 2000; Schurch et al., 2016).

More formally, if we have multiple independent outcomes $y_1, \ldots, y_K \sim P(Y \mid x)$ for each input point x, then training a predictor a with the squared loss on (input, target) examples $\left(x, \frac{K}{K-1}\overline{Var}(y_1, \ldots, y_K)\right)$, where $\overline{Var}$ denotes the empirical variance, yields an estimator of the aleatoric uncertainty.

Naturally, this estimator is asymptotically unbiased if the learning algorithm ensures asymptotic convergence to a Bayes predictor. This is due to the known fact that

$$\mathbb{E}_{y_1,\ldots,y_K \sim P(Y|x)}\left[\frac{K}{K-1}\overline{Var}(y_1, \ldots, y_K)\right] = Var_{P(Y|x)}[Y \mid x]. \tag{5.10}$$

We illustrate how such an estimator could be used to obtain EU estimates from the error predictor in Section 5.5.4.

(3) In cases where it is not possible to estimate the aleatoric uncertainty, we can use the expected loss estimate $e(f, x)$ as a pessimistic (i.e., conservative) proxy for $u(f, x)$, i.e., set $a(x)$ to 0 in (5.9). This is particularly relevant for settings when uncertainty estimates are used only to rank different data-points, and in which there is no reason to suspect that there is a significant variability in aleatoric uncertainty across the input space. This is actually the implicit assumption made whenever the variance or entropy of the posterior predictive (which, in principle, accounts for aleatoric uncertainty) is used to measure epistemic uncertainty. This is the case in the experiments we present in Section 5.5.3.

In real life settings, an estimator $x \mapsto a(x)$ of aleatoric uncertainty can be available (e.g., margins of errors of instruments used in a wet lab experiment), and can thus readily be plugged into (5.9). In the following subsections, we thus assume that $x \mapsto a(x)$ is available, whether it is identically equal to 0 or not, and focus on estimating the point-wise risk or expected loss $R(f, x)$. In Section 5.3.1, we present DEUP in a fixed training set setting, when the learner has access to a held-out validation set. In Section 5.3.2, we demonstrate how the challenges brought about by interactive settings actually bypass the need for the validation set required to train the error predictor.

5.3.1. Fixed Training Set

Using the notations introduced in Section 5.2, we first consider the scenario where we are interested in estimating EU for one predictor h_{z^N}, trained on a given training set z^N of a certain size N, and a held-out validation set z'^K is available. Recall that the goal is to obtain an estimator $x \mapsto e(h_{z^N}, x)$ of the point-wise risk $x \mapsto R(h_{z^N}, x) = \mathbb{E}_{P(Y|x)}[l(Y, h_{z^N}(x))]$.

Each $z'_i = (x'_i, y'_i)$ in the validation dataset can be used to compute an out-of-sample error $e'_i = l(y'_i, h_{z^N}(x'_i))$. Training a secondary predictor e on input-target pairs (x'_i, e'_i), for $i \in \{1, \dots, K\}$ yields the desired estimator $x \mapsto e(h_{z^N}, x)$. The procedure is summarized in Algorithm 3. It is used for the example provided in Figure 5.1

Algorithm 3 DEUP with a fixed training set

Input: $h = h_{z^N}$, a trained predictor; a, an estimator of aleatoric uncertainty; $z'^K = \{(x'_i, y'_i)\}_{i \in \{1,\dots,K\}}$; a validation set.
Output: $u : \mathcal{X} \to \mathbb{R}$, an estimator of the EU of h.
Create input-error dataset $\mathcal{D}_e = \{(x'_i, l(y'_i, h(x'_i))\}_{i \in \{1,\dots,K\}}$
Fit an estimator $x \mapsto e(x)$ on $\mathcal{D}_e$, using the squared loss $l(e, e') = (e - e')^2$
Return: The predictor $x \mapsto u(x) = e(x) - a(x)$.

5.3.2. Interactive Settings

Interactive settings, in which EU estimates are used to guide the acquisition of new examples, provide a more interesting use case for DEUP. However, they bring their own

challenges, as the main predictor is retrained multiple times with the newly acquired points. We discuss these challenges below, along with simple ways to mitigate them.

First, as the growing training set $z^N = \{(x_i, y_i)\}_{i \in \{1,\dots,N\}}$ for the main predictor h_{z^N} changes at each acquisition step (as z^N becomes z^{N+1} by incorporating the pair (x_{N+1}, y_{N+1}) for example), it is necessary to view the risk estimate $e(h_{z^N}, x)$ as a function of the pair (x, z^N), rather than a function of x only, unlike the fixed training set setting. The error predictor e, used to guide acquisition, needs to provide accurate uncertainty estimates as soon as the main predictor changes from h_{z^N} to $h_{z^{N+1}}$ (for clarity, we assume that only one point is acquired at each step, but this is not a requirement for DEUP). This means that e needs to generalize not only over input data x but also over training datasets z^N. e should thus be trained using a dataset $\mathcal{D}_e$ of input-target pairs $((x, z^N), l(y, h_{z^N}(x)))$, where (x, y) is not part of z^N. This would make the error predictor robust to feedback covariate shift (Fannjiang et al., 2022), i.e., the distribution shift resulting from the acquired points being a function of the training data. Conditioning on the dataset explicitly also addresses concerns from Bengs et al. (2022), which shows that directly estimating uncertainty using a second-order predictor trained simply with input data using L2 error yields poor uncertainty estimates. A learning algorithm with good generalization properties (such as modern neural networks) would in principle be able to extrapolate from $\mathcal{D}_e$ and estimate the errors made by a predictor h on points $x \in \mathcal{X}$ not seen so far, i.e., belonging to what we call the *frontier of knowledge*.

Second, the learner usually does not have the luxury of having access to a held-out validation set, given that the goal of such an interactive learner is to learn the Bayes predictor f^* using as little data as possible. This makes Algorithm 3 inapplicable to this setting. While it might be tempting to replace the held-out set z'^K in Algorithm 3 with the training set z^N, this would lead to an error predictor trained with in-sample errors rather than out-of-sample errors, thus severely limiting its ability to generalize its resulting uncertainty estimates out-of-sample and generally underestimating EU.

Denoting by $N_0 \geq 0$ the number of initially available training points before any acquisition, and observing that for $i > N_0$, the pair (x_i, y_i) is not used to train the predictors $h_{z^{N_0}}, h_{z^{N_0+1}}, \dots, h_{z^{i-1}}$, we propose to use the future acquired points as out-of-sample examples for the past predictors, in order to build the training dataset $\mathcal{D}_e$ for the error estimator. At step $M > N_0$, i.e., after acquiring $(M - N_0)$ additional input-output pairs (x, y) and obtaining the predictor h_{z^M}, $\mathcal{D}_e$ is equal to:

$$\mathcal{D}_e = \bigcup_{i=N_0+1}^{M} \bigcup_{N=N_0}^{i-1} \{((x_i, z^N), l(y_i, h_{z^N}(x_i)))\}. \tag{5.11}$$

Using $\mathcal{D}_e$ (5.11) requires storing in memory all versions of the main predictor h (i.e., $h_{z^{N_0}}, \dots, h_{z^M}$), which is impractical. Additionally, it requires using predictors that take as

input a dataset of arbitrary size, which might lead to overfitting issues as the dataset size grows. Instead, we propose the following two approximations of $\mathcal{D}_e$:

(1) We embed each input pair (x_i, z^N) in a feature space Φ, and replace each such pair in $\mathcal{D}_e$ with the feature vector $\phi_{z^N}(x)$, hereafter referred to as the **stationarizing features** of the dataset z^N at x.

(2) To alleviate the need of storing multiple versions of h, we make each pair (x_i, y_i) contribute to $\mathcal{D}_e$ once rather than $i - N_0$ times, by replacing the inner union of (5.11) with the singleton $\{((x_i, z^{i-1}), l(y_i, h_{z^{i-1}}(x_i)))\}$. Said differently, for each predictor h_{z^N}, only the next acquired point (x_{N+1}, y_{N+1}) is used to populate $\mathcal{D}_e$.

These approximations result in the following training dataset of the error estimator at step M:

$$\mathcal{D}_e = \{(\phi_{z^{i-1}}(x_i), l(y_i, h_{z^{i-1}}(x_i)))\}_{i \in \{N_0+1,\dots,M\}}. \tag{5.12}$$

In this paper, we explored $\phi_{z^N}(x) = \left(x, s, \hat{q}(x \mid z^N), \hat{V}(\tilde{\mathcal{L}}, z^N, x)\right)$ or a subset of these four features, where $\hat{q}(x \mid z^N)$ is a density estimate from data z^N at x, $s = 1$ if x is part of z^N and otherwise 0, $\tilde{\mathcal{L}}$ a learner that produces a distribution over predictors, e.g., a GP or an ensemble of neural networks (Lakshminarayanan et al., 2017b)), and $\hat{V}(\tilde{\mathcal{L}}, z^N, x)$ is an estimate of the model variance of $\tilde{\mathcal{L}}(z^N)$ at x. Note that $\tilde{\mathcal{L}}$ can be chosen to be the same as $\mathcal{L}$ (Section 5.2). For numerical reasons, we found it preferable to use $\log \hat{q}$ and $\log \hat{V}$ instead of $\hat{q}$ or $\hat{V}$ as input features. $\hat{q}$ can be obtained by training a density estimator (such as a Kernel Density Estimator or a flow-based deep network (Rezende and Mohamed, 2015)). Like the other predictors, the density estimator must also be fine-tuned when new data is added to the training set. While these features are not required per se to train DEUP, they provide clues to help train the uncertainty estimator, and one can play with the trade-off of computational cost versus usefulness of each clue. They also provide DEUP the flexibility to incorporate useful properties. For instance, the density can help in incorporating *distance awareness* which Liu et al. (2020) show is critical for uncertainty estimation. They sometimes come at no greater cost, if our main predictor is the mean prediction of the learner's output distribution, and if we use the corresponding variance as the only extra feature, as is the case in the experiments of Section 5.5.1 with GPs. As (5.12) is merely an approximation of (5.11), not all subsets of features are expected to be useful in all settings. In the ablations presented in Appendices F.1 and F.3, we experimentally confirm that x alone is not sufficient, and the stationarizing features are critical for reliable uncertainty estimates. Additionally, we observe that having x as part of ϕ_{z^N} can help in some tasks and hurt in others. We note that the choice of features here is a design choice, and the usage of other features could be investigated in future work.

Pre-training the error predictor: If the learner cannot afford to wait for a few rounds of acquisition in order to build a dataset $\mathcal{D}_e$ large enough to train the error predictor (e.g., when the prediction target y is the output of a costly oracle), it is possible to pre-fill $\mathcal{D}_e$ using the N_0 initially available training points z^{N_0} only, following a strategy inspired by $K-$fold cross validation. We present one such strategy in Algorithm 4. The procedure stops when the training dataset for the secondary learner, $\mathcal{D}_u$, contains at least $N_{pretrain}$ elements. In our experiments, we choose $N_{pretrain}$ to be a small multiple of the number of initial training points.

Algorithm 4 Pre-filling the uncertainty estimator training dataset $\mathcal{D}_e$

Input: z^{N_0}, the initial training set; $N_{pretrain}$, the required size of the training dataset $\mathcal{D}_e$
Output: $\mathcal{D}_e$, a training dataset for the uncertainty estimator.

$\mathcal{D}_e \leftarrow \emptyset$
while $|\mathcal{D}_e| < N_{pretrain}$ **do**
  Split $\mathcal{D}_{init} = z^{N_0}$ into K random subsets $\mathcal{D}_1, \dots, \mathcal{D}_K$ of equal size. Define $\tilde{\mathcal{D}} = \bigcup_{k=1}^{K-1} \mathcal{D}_k$
  Fit a new predictor $h_{\tilde{\mathcal{D}}}$ on $\tilde{\mathcal{D}}$, and fit the features $\phi_{\tilde{\mathcal{D}}}$ on $\tilde{\mathcal{D}}$
  $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup \bigcup_{(x,y)\in\mathcal{D}_K} \{(\phi_{\tilde{\mathcal{D}}}(x), l(y, h_{\tilde{\mathcal{D}}}(x)))\}$
end while
Return: $\mathcal{D}_e$.

Putting all these things together yields the pseudo-code for DEUP in interactive learning settings provided in Algorithm 5. In practice, in addition to out-of-sample errors, the secondary predictor can be trained with in-sample errors, which inform the error predictor that the uncertainty at the training points should be low. The stopping criterion is defined by the interactive learning task. It can be a fixed number of iterations for example, a criterion defined by a metric evaluated on a validation set or a criterion defined by the optimal value reached (maximum or minimum) in SMO. The algorithm is agnostic to the acquisition function or policy $\pi(. \mid h, u)$, the acquisition machinery that proposes new input points from $\mathcal{X}$, using the current predictor h and its corresponding EU estimator u. Examples of acquisition functions in the context of SMO include Expected Improvement (Močkus, 1975) and Upper Confidence Bound (UCB, Srinivas et al. (2010)).

5.4. Related work on uncertainty estimation

Bayesian Learning. We have seen in Section 2.1.3.5 and Section 2.2 how Bayesian approaches provide a natural way of representing epistemic uncertainty in the form of the Bayesian posterior distributions.

In the deep learning context, we have already discussed in Section 2.2.3 MC Dropout (Gal and Ghahramani, 2016b), that interprets Dropout (Hinton et al., 2012) as a variational inference technique and in Bayesian Neural Networks (BNNs) and Deep Ensembles (Lakshminarayanan et al., 2017b), who share some similarities with ensemble-based methods, that

Algorithm 5 Training procedure for DEUP in an Interactive Learning setting

Input: $\mathcal{D}_{init} = z^{N_0} = \{(x_i, y_i)\}_{i \in \{1,\dots,N_0\}}$; a, an estimator of aleatoric uncertainty; Φ, the embedding space (i.e., the chosen stationarizing features); π, the acquisition machinery.
Output: $\mathcal{D}$, a dataset populated with the acquired examples.

$\mathcal{D}_e \leftarrow \emptyset$, training dataset for the error predictor e
$\mathcal{D} \leftarrow \mathcal{D}_{init}$, dataset of training points for the main predictor
$x_{acq} \leftarrow \emptyset, \quad y_{acq} \leftarrow \emptyset$
Optional: Pre-fill $\mathcal{D}_e$ using Algorithm 4
while stopping criterion not reached **do**
  Fit predictor $h_{\mathcal{D}}$ and features $\phi_{\mathcal{D}}$ on $\mathcal{D}$
  Fit a predictor $\phi \mapsto e(\phi)$ on $\mathcal{D}_e$
  $x_{acq} \sim \pi(. \mid h_{\mathcal{D}}, x \mapsto e(\phi_{\mathcal{D}}(x)) - a(x))$ ▷ can be a single point or a batch of points
  Sample outcomes from the ground truth distribution: $y_{acq} \sim P(. \mid x_{acq})$
  $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup \{(\phi_{\mathcal{D}}(x_{acq}), l(y_{acq}, h_{\mathcal{D}}(x_{acq})))\}$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_{acq}, y_{acq})\}$
end while
Return: $\mathcal{D}$.

include bagging (Breiman, 1996) and boosting (Efron and Tibshirani, 1994), where multiple predictors are trained, and their outputs are averaged to make a prediction. Deep Ensembles are closer to the Bayesian approach, using an ensemble of neural networks that differ because of randomness in initialization and training. Wen et al. (2020b) present a memory-efficient way of implementing deep ensembles by using one shared matrix and a rank-1 matrix for the parameters per member, while Vadera et al. (2020b); Malinin et al. (2020) improve the efficiency of ensembles by distilling the distribution of predictions rather than the average, thus preserving the information about the uncertainty captured by the ensemble.

Another line of work for approximate Bayesian learning includes SWAG (Maddox et al., 2019), which fits a Gaussian distribution on the first moments of SGD iterates, building upon SWA (Izmailov et al., 2018) to define the posterior over the neural network weights. This distribution is then used as a posterior over the neural network weights. Dusenberry et al. (2020) parametrize the BNN with a distribution on a rank-1 subspace for each weight matrix.

Classical work on *Query by committee* (Seung et al., 1992; Freund et al., 1992; Gilad-Bachrach et al., 2005; Burbidge et al., 2007) also studied the idea of using discrepancy as a measure for information gain for the design of experiments. Tagasovska and Lopez-Paz (2019) propose using orthonormal certificates, which capture the distance between a test sample and the dataset. Liu et al. (2020) further establish the importance of this notion of *distance awareness* for uncertainty estimation, and along with proceeding methods DUE (van Amersfoort et al., 2021), and DDU (Mukhoti et al., 2021) combine feature representations learned by deep neural networks with exact Bayesian inference methods like GPs and Gaussian Discriminant Analysis. This line of work falls under the umbrella of Deep Kernel

Learning (DKL, Wilson et al., 2016). DUN (Antoran et al., 2020) uses the disagreement between the outputs from intermediate layers as a measure of uncertainty.

Distribution-Free Uncertainty Estimation. Conformal prediction (Vovk et al., 2005; Shafer and Vovk, 2008; Angelopoulos and Bates, 2021) is an alternative to Bayesian methods for uncertainty estimation. Conformal prediction involves building statistically rigorous uncertainty sets and intervals for model predictions, which are *guaranteed* to contain the ground truth with a specified probability. It is also closely linked to the statistical paradigm of hypothesis testing (Angelopoulos and Bates, 2021). Conformal prediction is an appealing alternative to Bayesian approaches as it can be applied on top of existing models and does not require any particular training procedure. Recent work has demonstrated the efficacy of using conformal prediction with neural network models for time series (Lin et al., 2022; Zaffran et al., 2022) and image data (Angelopoulos et al., 2020). Fannjiang et al. (2022) study conformal prediction within an active learning setting. While DEUP can broadly be categorized as a distribution-free uncertainty estimation method, it differs from conformal predictions as it does not require a pre-defined degree of confidence before outputting a prediction set.

Loss Prediction. Yoo and Kweon (2019) propose a loss prediction module for learning to predict the value of the loss function. Hu et al. (2020) also propose using a separate network that learns to predict the variance of an ensemble. These methods, however, are trained only to capture the in-sample error and do not capture the out-of-sample error, which is more relevant for scenarios like active learning where we want to pick x where the reducible *generalization error* is large. EpiOut (Umlauft et al., 2020; Hafner et al., 2019) propose learning a binary output that distinguishes between low or high EU.

5.5. Experiments

Code for these experiments can be found at `https://github.com/MJ10/DEUP`.

Through the experiments described below, we aim to provide evidence for the following key claims: (**C1**) Epistemic uncertainty measured by DEUP leads to significant improvements in downstream decision-making tasks compared to established baselines, and (**C2**) The error predictor learned by DEUP can generalize to unseen samples. We emphasize that in order to make fair comparisons, **DEUP does not have access to any additional OOD data during training** in all experiments presented in this section. Instead, when required, we use Algorithm 4 to generate the OOD data used for training the error predictor. Finally, note that in terms of computational cost, training DEUP with density and model variance as stationarizing features is on par with training an ensemble of 5 networks.

5.5.1. Sequential Model Optimization

Sequential model optimization, also called Bayesian optimization, is a form of interactive learning where the learner chooses query examples to label at each stage, looking for samples with a high value of the unknown oracle function. Such examples are selected to have a high predicted value (to maximize the unknown oracle function) and a considerable predicted uncertainty (offering the opportunity to discover yet higher values). Acquisition functions, such as Expected Improvement (EI, Močkus (1975)), trade-off exploration and exploitation, and one can select the next candidate by looking for x's maximizing the acquisition function. We combine EI with DEUP (and call the method DEUP-EI, by analogy to GP-EI, a common SMO baseline), treating the primary predictor and DEUP EU predictions at x respectively as mean and standard deviation of a Gaussian distribution for the learner's guess of the value of the oracle at x. Similarly, when using MC-Dropout or Deep Ensembles as predictors, we refer to the resulting methods as MCDropout-EI and Ensembles-EI, respectively.

5.5.1.1. General remarks about the SMO experiments

For all our Sequential Optimization algorithms, we use Algorithm 5 to train DEUP uncertainty estimators. We found that the optional step of pre-filling the uncertainty estimator dataset $\mathcal{D}_e$ was necessary given the low number of available training points. We used half the initial training set (randomly chosen) as in-sample examples (used to train the primary predictor and an extra-feature generator) and the other half as out-of-sample examples to provide instances of high epistemic uncertainty to train an uncertainty predictor; we repeated the procedure by alternating the roles of the two halves of the dataset. We repeated the whole process twice using a new random split of the dataset, thus ending up with 4 training points in $\mathcal{D}_e$ for every initial training point in $\mathcal{D}_{init}$.

The error predictor is trained with the log targets (i.e., log MSE between predicted and observed error). This helps since the scale of the errors varies over multiple orders of magnitude.

Computationally, the training time of DEUP-EI depends on various choices (e.g., the features used to train the epistemic uncertainty predictor, the dimension of the input, the learning algorithms, etc..). Additionally, the training time for the uncertainty predictor varies at each step of the optimization. In total, the sequential optimization experiments took about 1 CPU day.

We use BoTorch [3] (Balandat et al., 2020) as the base framework for our experiments.

5.5.1.2. One-dimensional objective

We first consider in Figure 5.3 (Left) a synthetic one-dimensional function with multiple local maxima, and starting from an initial dataset of 6 pairs (x, y), we compare the maximum

3. https://botorch.org/

values reached by each of DEUP-EI, GP-EI, MCDropout-EI, and Ensembles-EI to a random acquisition strategy.

For random acquisition, we sampled for different seeds 56 points and used the (average across the seeds of the) maximum of the first 6 values as the first value in the plots (Figures 5.3 and 5.6). Because the function is specifically designed to have multiple local maxima, GP-EI also required more optimization steps and performed worse than random acquisition.

Because MCDropout and Ensembles are trained on in-sample data only, they are unable to generalize their uncertainty estimates, which makes them bad candidates for Sequential Model Optimization, because they are easily stuck in local minima and require many iterations before the acquisition function gives more weight to the predicted uncertainties than the current maximum.

For DEUP-EI, the main predictor is a neural network, and the error predictor is a GP regressor. The stationarizing features $\phi_z(x)$ used are the input x itself, and the variance of a GP fit on the available data at every step. In Section 5.5.1.3, we provide an ablation study of different subsets of the features $\phi_z(x)$, confirming the necessity of extra features and the usefulness of the input x.

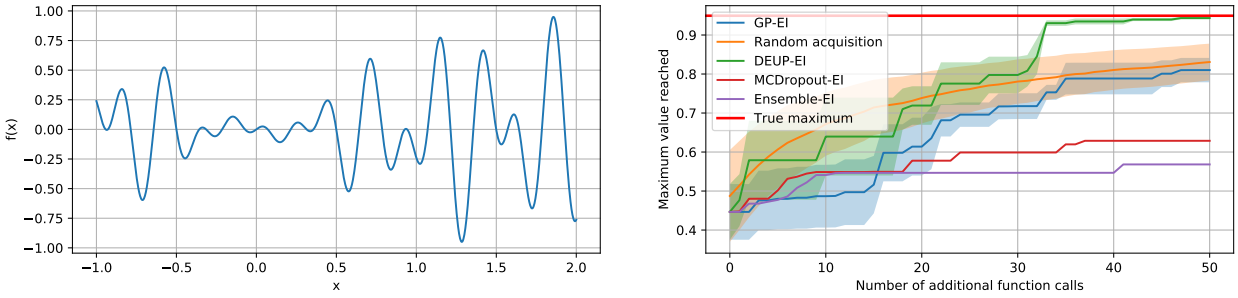


Figure 5.3 – *Left.* Synthetic function to optimize. *Right.* Maximum value reached by the different methods on the synthetic function. The shaded areas represent the standard error across five different runs, with different initial sets of 6 pairs. For clarity, the shaded areas are omitted for the two worst-performing methods. In each run, all the methods start with the same initial set of 6 points. GP-EI tends to get stuck in local optima and requires more than 50 steps, on average, to reach the global maximum.

5.5.1.3. Ablation study for the stationarizing features

To study the usefulness of each subset of the "xdvb" features (respectively representing the input x, a density estimate, a variance estimate, and a bit indicating whether x was part of the training set), we compare in Figure 5.4 the different subsets of the features $\phi_z(x)$ used as input to the error predictor. Most notably, this confirms the importance of the stationarizing features besides the input x and shows that incorporating x to the features can help improve the performances of an interactive learner.

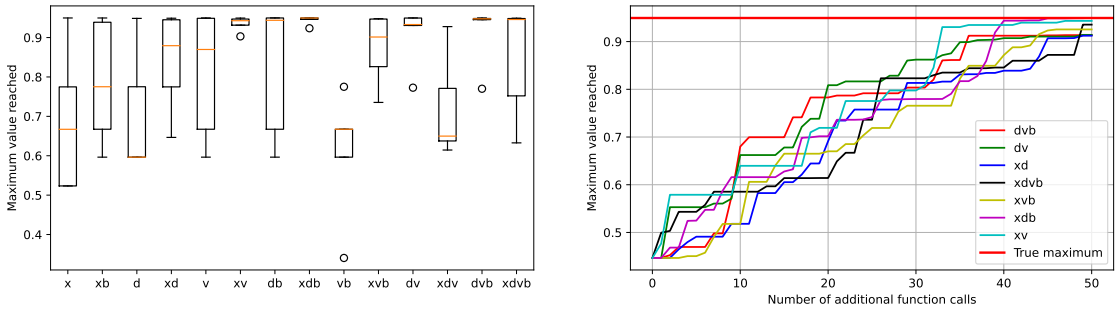


Figure 5.4 – We train DEUP-EI to optimize the synthetic function of Figure 5.3, with different subsets of the stationarizing features as inputs to the error predictor, for 50 iterations. All the algorithm instances start with an initial set of 6 (x, y) pairs, (Left) Maximum value reached at the 40th iteration for each subset of the stationarizing features. The box plots represent the resulting distributions across five different runs. (Right) Evolution of the mean (across the five runs) of the maximum value reached at each iteration for the subsets of features that surpassed the 0.9 threshold.

We found that the binary (in-sample/out-of-sample) feature and density estimates were redundant with the variance feature and didn't improve the performance as captured by the number of additional function calls.

5.5.1.4. Two-dimensional objective

To showcase DEUP's usefulness for Sequential Model Optimization with dimensions greater than 1, we consider optimizing the Levi N.13 function, a known benchmark for optimization. The function f takes a point (x, y) in 2D space and returns:

$$f(x,y) = -\left(\sin^2(3\pi x) + (x-1)^2(1+\sin^2(3\pi y)) + (y-1)^2(1+\sin^2(2\pi y))\right)$$

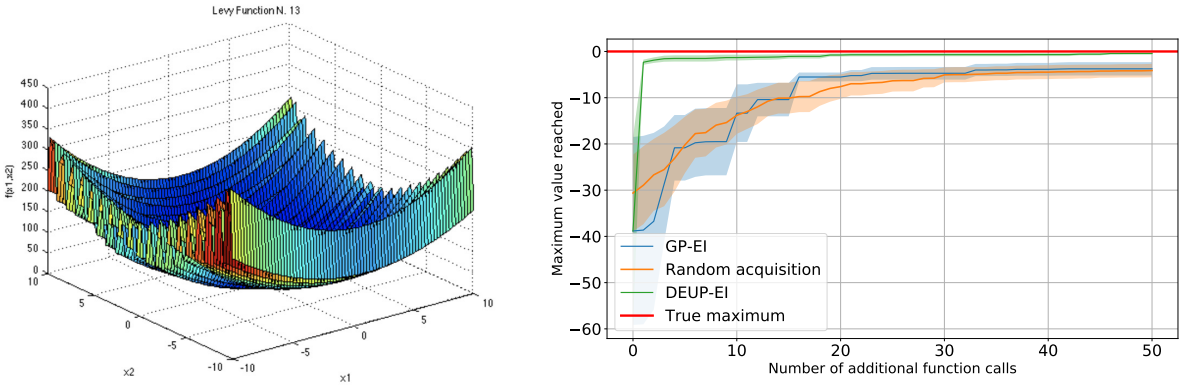
We use the box $[-10, 10]^2$ as the optimization domain. In this domain, the function's maximum is 0, reached at $(1, 1)$. The function has multiple local maxima, as shown in Section 5.5.1.4 [4].

Similar to the previous one-dimensional function, MCDropout and Ensemble provided bad performances and are omitted from the plot in Section 5.5.1.4. We used the same setting and hyperparameters for DEUP as the previous function. DEUP-EI is again the only method that reaches the global maximum consistently in under 56 function evaluations.

5.5.1.5. Multi-dimensional objective

Next, we showcase how using DEUP to calibrate GP variances (the only input for the error predictor) allows for better performance in higher-dimensional optimization tasks. Specifically, we compare DEUP-EI to TuRBO-EI (Eriksson et al., 2019), a state-of-the-art method for sequential optimization that fits a collection of local GP models instead of a global one, to perform efficient high-dimensional optimization, on the Ackley function (Ackley, 2012), a

4. Plot of the function copied from https://www.sfu.ca/ ssurjano/levy13.html



(a) Visualization of $(x, y) \mapsto -f(x, y)$ **(b)** Comparisons with GP-EI and Random acquisition

Figure 5.5 – Sequential Model Optimization on the Levi N.13 function

standard benchmark for optimization algorithms. This function can be defined for arbitrary dimensions and has many local minima.

The Ackley function of dimension d is defined as:

$$Ackley_d : \mathcal{B} \to \mathbb{R}$$

$$x \mapsto A \exp\left(-B\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) + \exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(cx_i)\right) - A - \exp(1)$$

where $\mathcal{B}$ is a hyperrectangle of $\mathbb{R}^d$. $(0, \ldots, 0)$ is the only global optimizer of $Ackley_d$, at which the function is equal to 0. We use BoTorch's default values for A, B, c, which are $20, 0.2, 2\pi$, respectively.

In our experiments, we used $\mathcal{B} = [-10, 15]^d$ for all dimensions d.

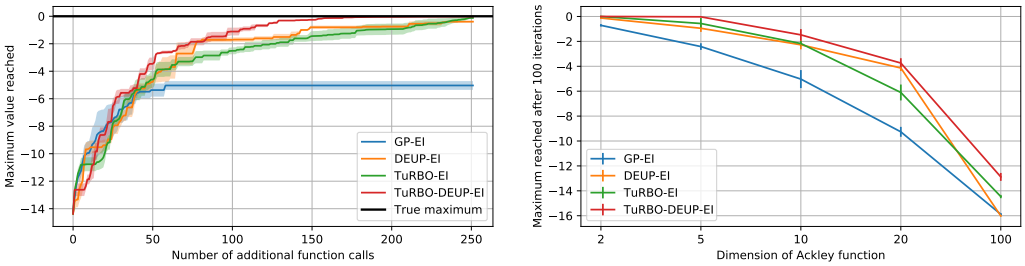


Figure 5.6 – *Left.* Max. value reached by the different optimization methods for the ten-dimensional Ackley function. In each run, all the methods start with the same initial 20 points. Shaded areas represent the standard error across three runs. *Right.* Max. value reached in the budget-constrained setting on the Ackley functions of different dimensions. Error bars represent the standard error across three different runs, with different initial sets of 20 pairs. The budget is 120 function calls in total. Higher is better, and TuRBO-DEUP-EI is less hurt by dimensionality.

In Figure 5.6 (Left), we compare the different methods on the 10-D Ackley function and observe that while GP-EI gets stuck in local optima, DEUP-EI can reach the global maximum consistently. In Figure 5.6 (Right), we show that for budget-constrained optimization problems, adapting DEUP to TuRBO (called TuRBO-DEUP-EI) consistently outperforms regular TuRBO-EI, especially in higher dimensions. More details about this experiment are provided Appendix F.1.

For fair comparisons, for DEUP, we use a Gaussian Process as the primary model and its variance as the only input of the epistemic uncertainty predictor. This means we calibrate the GP variance to match the out-of-sample squared error, using another GP to perform the regression. TurBO-DEUP is a combination of both, in which we perform the variance calibration task for the local GP models of TurBO. The uncertainty predictor, i.e., the GP regressor, is trained with log targets, as in Section 5.5.1.2, but also with log variances as inputs.

Only the stationarizing feature is used as input for the uncertainty predictor. When we used the input x as well, we found that the GP error predictor overfits the x part of the input (x, v), which was detrimental to the final performances. For all experiments, we used 20 initial points.

The experiments presented in this section thus validate our experimental claim **C1**.

5.5.2. Reinforcement Learning

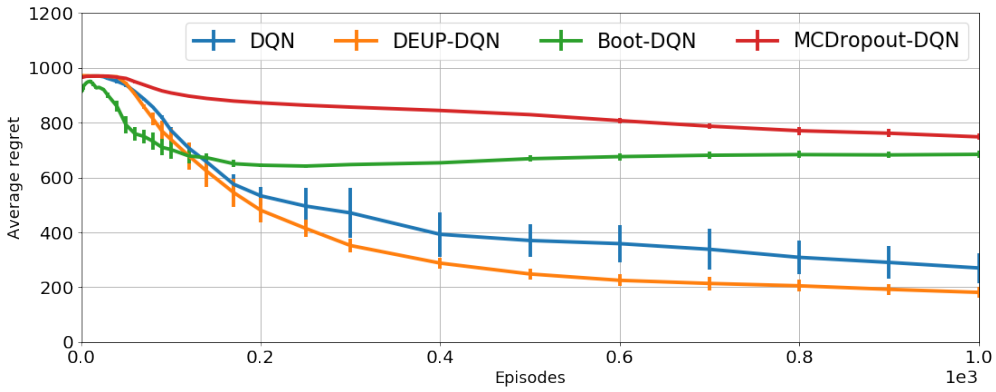


Figure 5.7 – Average regret on CartPole task. Error bars represent standard error across five runs.

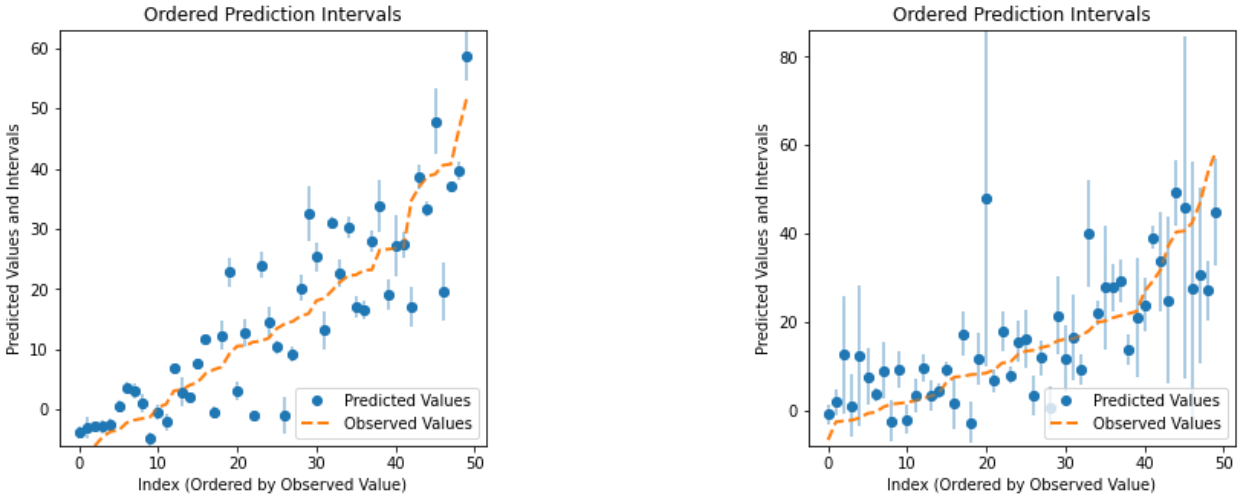
Similar to SMO, a key challenge in RL is efficient exploration of the input state space. To investigate the effectiveness of DEUP's uncertainty estimates in the context of RL, we present a proof-of-concept for incorporating epistemic uncertainties predicted by DEUP to DQN (Mnih et al., 2013), which we refer to as DEUP-DQN. Specifically, we train the uncertainty predictor to predict the TD error, using log-density estimates as a stationarizing feature. The

predicted uncertainties are then used as an exploration bonus in the Q-values. As explained in Section 5.3.2, it is the acquired points, before they are used to retrain the main predictor, that act as the *out-of-sample* examples to train DEUP. In RL, because the targets (e.g., of Q-Learning) are themselves estimates and moving, data seen at any particular point is typically out-of-sample and can inform the uncertainty estimator when the inputs are used with the stationarizing features. Details of the experimental setup are in Appendix F.2. We evaluate DEUP-DQN on CartPole, a classic RL task from *bsuite* (Osband et al., 2020), against DQN + ϵ-greedy, DQN + MC-Dropout (Gal and Ghahramani, 2016b) and Bootstrapped DQN (Osband et al., 2016). Figure 5.7 shows that DEUP achieves lower regret faster than all the baselines, demonstrating the advantage of DEUP's uncertainty estimates for efficient exploration, confirming our claim **C1**. Future work should investigate ways to scale this method to more complex environments.

5.5.3. Uncertainty Estimation

5.5.3.1. Epistemic Uncertainty Estimation for Drug Combinations

We validate DEUP's ability to generalize its uncertainty estimate (claim **C2**) in a real-world regression task predicting the synergy of drug combinations. While much effort in drug discovery is spent on finding novel small molecules, a potentially cheaper method is identifying combinations of pre-existing drugs which are synergistic (i.e., work well together). However, every possible combination cannot be tested due to the high monetary cost and time required to run experiments. Therefore, developing good estimates of the EU can help practitioners select informative and promising experiments. we used the DrugComb and LINCS L1000 datasets (Zagidullin et al., 2019; Subramanian et al., 2017). DrugComb is a dataset of pairwise combinations of anti-cancer compounds tested on various cancer cell lines. The dataset provides access to several synergy scores for each combination, each indicating whether the two drugs have a synergistic or antagonistic effect on cancerous cell death. LINCS L1000 contains differential gene expression profiles for various cell lines and drugs. Differential gene expressions measure the difference in the amount of mRNA related to a set of influential genes before and after the application of a drug. Because of this, gene expressions are a powerful indicator of the effect of a single drug at the cellular level. As shown in Section 5.5.3.1, the out-of-sample error predicted by DEUP correlates better with residuals of the model on the test set compared to several other uncertainty estimation methods. Moreover, DEUP better captured the order of magnitude of the residuals as shown in Figure 5.8, confirming the claim **C2**. Details on experiments and metrics are in Appendix F.4.



(a) Ensemble **(b)** DEUP

Figure 5.8 – Drug Combinations. Predicted mean and uncertainty (error bars) on 50 test examples ordered by increasing value of true synergy score (orange). Model predictions and uncertainties in blue. Ensemble **(a)** (and MC-dropout, not shown) consistently underestimate uncertainty while DEUP **(b)** captures the right order of magnitude.

Model	Corr. w. res.	U. Bound	Ratio	Log Likelihood
MC-Dropout	0.14 ± 0.07	0.56 ± 0.05	0.25 ± 0.12	-20.1 ± 6.8
Deep Ensemble	0.30 ± 0.09	0.59 ± 0.04	0.50 ± 0.13	-14.3 ± 4.7
DUE	0.12 ± 0.12	0.15 ± 0.03	$\mathbf{0.80 \pm 0.79}$	-13.0 ± 0.52
DEUP	$\mathbf{0.47 \pm 0.03}$	0.63 ± 0.05	$\mathbf{0.75 \pm 0.07}$	$\mathbf{-3.5 \pm 0.25}$

Table 5.1 – Drug Combinations. *Corr. w. res.* shows correlation between model residuals and predicted uncertainties $\hat{\sigma}$. A best-case *Upper Bound* on *Corr. w. res.* is obtained from the correlation between $\hat{\sigma}$ and true samples from $\mathcal{N}(0, \hat{\sigma})$. *Ratio* is the ratio between col. 1 and 2 (larger is better). *Log-likelihood*: average over 3 seeds of per sample predictive log-likelihood.

5.5.3.2. Epistemic Uncertainty Predictions for Rejecting Difficult Examples

Epistemic uncertainty estimates can be used to reject difficult examples where the predictor might fail, such as OOD inputs[5]. We thus consider a standard OOD Detection task (Liu et al., 2020; van Amersfoort et al., 2021; Nado et al., 2021), where we train a ResNet (He et al., 2016) model for CIFAR-10 classification (Krizhevsky, 2009) and reject OOD examples using the estimated uncertainty in the prediction. To facilitate rejection of classes other than those in the training set, we use a Bernoulli Cross-Entropy Loss for each class following van Amersfoort et al. (2020): $l(\hat{f}(x), y) = -\sum_i y_i \log \hat{f}_i(x) + (1 - y_i) \log(1 - \hat{f}_i(x))$, where y is a one-hot vector ($y_i = 1$ if i is the correct class, and 0 otherwise), and $\hat{f}_i(x) =$ predicted probability for class i. The target for out-of-distribution data (from *other* classes)

5. e.g., rare but challenging inputs can be directed to a human, avoiding a costly mistake

Model	SRCC	AUROC
MC-Dropout	0.287 ± 0.002	0.894 ± 0.008
Deep Ensemble	0.381 ± 0.004	$\mathbf{0.933 \pm 0.008}$
DUQ	0.376 ± 0.003	0.927 ± 0.013
DUE	0.378 ± 0.004	0.929 ± 0.005
DEUP (D+V)	$\mathbf{0.426 \pm 0.009}$	$\mathbf{0.933 \pm 0.010}$

Table 5.2 – Spearman Rank Correlation Coefficient (SRCC) between predicted uncertainty and OOD generalization error (SVHN); Area under ROC Curve (AUROC) for OOD Detection (SVHN) with CIFAR-10 ResNet-18 models (3 seeds). DEUP significantly outperforms baselines in terms of SRCC and is equivalent to Deep Ensembles but scoring better than the other methods in terms of the coarser AUROC metric.

can be represented as $y = \{0, \ldots, 0\}$. We use Algorithm 3 with stationarizing features from Section 5.3.2 for training DEUP on this task. At inference time, we can reject an example based on the estimated epistemic uncertainty. To ascertain how well an epistemic error estimate sorts unseen examples by the above NLL loss, we consider the rank correlation between the predicted uncertainty and the observed generalization error. Note that we can compute the generalization error for OOD examples with $y = \{0, \ldots, 0\}$ directly since we are using a Binary Cross-Entropy loss. We use examples from SVHN (Netzer et al., 2011) as the OOD examples. This metric focuses on the quality of the uncertainty estimates rather than just their ability to simply classify in- vs out-of-distribution examples. This metric is also an indicator of how accurate the uncertainty estimates are for out-of-distribution examples. We also report the standard AUROC for the OOD detection task. We use MC-Dropout (Gal and Ghahramani, 2016b), Deep Ensembles (Lakshminarayanan et al., 2017b), DUE (van Amersfoort et al., 2021) and DUQ (van Amersfoort et al., 2020) as the baselines. As the primary focus of our work is estimating epistemic uncertainty, we do not consider specialized methods for OOD detection (Morningstar et al., 2021; Haroush et al., 2021). Additionally, to study the effect of model capacity we consider ResNet-18 and ResNet-50 as the main predictors for all the methods. Additional training details along with results for ResNet-50 are presented in Appendix F.3.

Table 5.2 shows, supporting experimental claim **C2**, that with the variance from DUE (van Amersfoort et al., 2021) and the density from MAF (Papamakarios et al., 2017) as stationarizing features, DEUP provides uncertainty estimates that have high rank correlation with the underlying generalization error on OOD data. We also achieve competitive AUROC with the strong baselines, demonstrating that uncertainty estimates from DEUP result in better performance on the downstream task of OOD detection. Additionally, since the error predictor is trained separately from the main predictor, there is no explicit trade-off between the accuracy of the main predictor and the quality of uncertainty estimates. We achieve competitive accuracy of 93.89% for the main predictor. We ignore the effect of aleatoric

uncertainty (due to inconsistent human labelling), which would require a human study to ascertain. We note that we choose the DUE baseline as it is representative of related methods such as SNGP (Liu et al., 2020) and DDU (Mukhoti et al., 2021), and performs best in our experiments. We present additional results in a distribution shift setting in Appendix F.3. Note that in the pretraining phase of the uncertainty estimator (Algorithm 4), we obtain the subsets by splitting the data based on classes, with each split containing $\lfloor n/K \rfloor$ classes. So when we train on $K-1$ subsets, the $\lfloor n/K \rfloor$ classes from the remaining subset become *out-of-distribution*.

5.5.4. DEUP in the presence of aleatoric uncertainty

We consider a scenario similar to that of Figure 5.1, but in which Gaussian noise is added to the ground truth oracle before providing training examples. Because of the noisy training dataset, GP conflates epistemic and aleatoric uncertainty, which makes the gap between the predicted epistemic uncertainty (as measured by the GP variance) and the true epistemic uncertainty (as measured by the mean squared error between the GP mean and the noiseless ground truth function) higher than in the deterministic setting of Figure 5.1. The goal of this experiment is to illustrate that using the noisy training set allows learning an estimator of aleatoric uncertainty (AU), which could be subtracted from DEUP's error predictor e to obtain an estimator of epistemic uncertainty. The estimator of AU is obtained using (5.10). A simple linear regressor is used to estimate AU, to ovoid overfitting issues.

A key distinction of this setting, is that DEUP's training data (the errors of the main predictor) are themselves noisy, which makes it important to use more out-of-sample data to obtain reasonable total uncertainty estimates (from which we subtract the estimates of the aleatoric uncertainty).

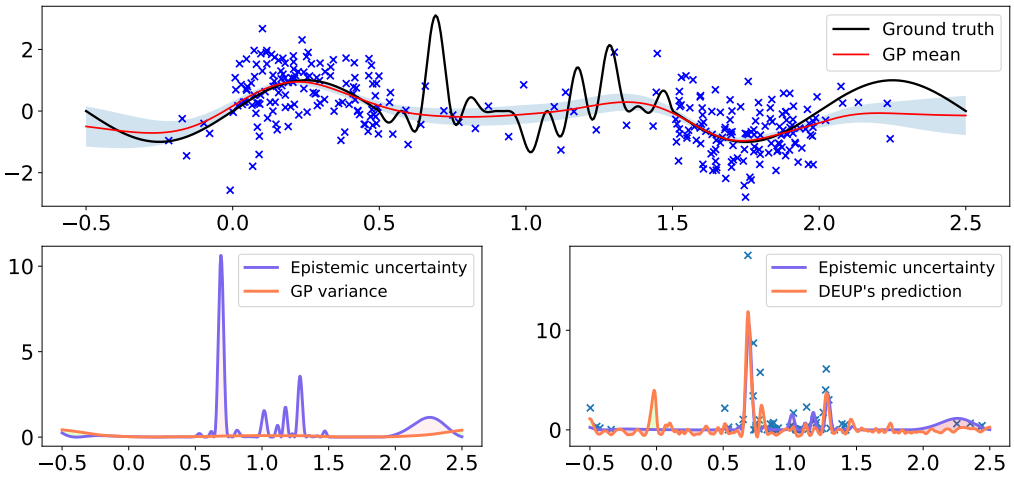


Figure 5.9 – *Top.* A GP is trained to regress a function using noisy samples. GP uncertainty (model standard deviation) is shaded in blue. *Bottom left.* Using GP variance as a proxy for epistemic uncertainty misses out on more regions of the input space, when compared to Figure 5.1 . *Bottom right.* Using additional out-of-sample data in low density regions, a second GP is trained to predict the generalization error of the first GP (total uncertainty). Using second samples from the oracle for each of the training points, a linear regressor fits the training pairs $(x, \frac{1}{2}(y_1 - y_2)^2)$ to estimate the point-wise aleatoric uncertainty. Note that no constraint is imposed on DEUP's outputs, which explains the predicted negative values for uncertainties. In practice, if these predicted uncertainties were to be used, (soft) clipping should be used.

Chapter 6

Conclusion and perspectives

Over the past decade, the field of machine learning has experienced significant growth, marked by substantial investments from various organizations aiming to expand the capabilities of artificial intelligence and address complex challenges. While one can only speculate about the real motivations [1] behind the technological revolution we are witnessing, it is becoming harder by the day to dismiss thoughts about the potentially substantial societal implications of the democratization of AI and its widespread usage. Perhaps an existential revolution is underway, where the nature of intelligence and the pedestalization of human creativity are being massively questioned. At best, the escalating entropy caused by generative models (De Angelis et al., 2023) is met with synchronized efforts to steer human *progress* (Meek Lange, 2023) in well-defined directions over the upcoming years. The multitude of global challenges confronting humanity (UnitedNations, 2022) can be further exacerbated by the widespread proliferation of AI tools, or conversely, these challenges can be partially addressed through the thoughtful and conscientious utilization of AI.

Admittedly, this thesis does not directly tackle the pressing questions mentioned earlier. Nevertheless, the author of this thesis hopes that the work presented in this manuscript at least sheds light on some of the shortcomings of machine learning algorithms deployed in various environments and, at best, contributes to the process of scientific discovery in fields that could have a positive impact in the face of today's challenges.

Summary

In Chapter 3, we deepened generative flow networks' mathematical framework and mathematical properties. More specifically, we formally defined GFlowNets and contrasted them to flow networks. We studied Markovian flows and the different sets of constraints that must

1. if any (Smith and Marx, 1994; Wyatt, 2008), and if there is actually such a thing as *motivation* (Jacob, 2023; Kalis, 2019).

be satisfied to define one, and we extended the framework to conditional generative flow networks. .

Beyond search problems, for which GFlowNets have already shown significant improvements compared to more traditional techniques, we emphasized their applicability to Bayesian inference of discrete structures. Estimating the Bayesian posterior, or the posterior predictive, is paramount for uncertainty-aware predictions, as we argued for in Chapter 2.

Additionally, the theory and experiments presented in the second half of the chapter place GFlowNets in the family of variational methods. They suggest that off-policy GFlowNet objectives may be an advantageous replacement to previous variational objectives, especially when the target distribution is highly multimodal, striking an interesting balance between the mode-seeking (Reverse KL) and mean-seeking (Forward KL) variational variants.

In Chapter 4, we developed a theory for generalized GFlowNets and illustrated it through various experiments. The theory opened the door to Bayesian inference of mixed continuous and discrete variables, which we showcased in an embryonic set of experiments on Bayesian structure learning, and to search problems in continuous spaces, similar to the presented molecular conformer generation experiments. A natural application of the theory is simulation-based inference for inverse problems in the natural sciences, which could benefit from the already proven advantages of GFlowNets, in terms of sample efficiency and quality of obtained posteriors.

We developed `torchgfn`, an open-source software for generative flow networks, both in the discrete and the general setting. It is presented in Appendix B.

Chapter 5 delved into exploring alternative measures of epistemic uncertainty beyond Bayesian posteriors. We argued that the bias introduced by the ineluctable model misspecification should be accounted for to obtain epistemic uncertainty estimates that are reliable enough for decision-making in interactive learning settings. Our proposed method, DEUP, relies on maintaining a secondary predictor, perpetually trained on the errors of the main one, and on deducing from the secondary predictions measures of epistemic uncertainty that could be used for decision-making in downstream tasks. We addressed the non-stationarity challenges raised by interactive learning by using stationarizing features as extra inputs to the secondary predictor.

Future research directions

The work presented in this manuscript prompts many important open questions and paves the way for several applications.

Scaling GFlowNets

In the proven successes of GFlowNets, the trajectories used for learning the sampler have limited lengths. In many interesting applications, the generation process relies on combining many primitive blocks (e.g. large causal graphs, parameters of a large neural network, and large molecules), leading to much longer trajectories. Recent efforts (Madan et al., 2022a; Pan et al., 2017) have attempted to address this limitation of GFlowNets by learning from partial trajectories. However, a promising future research avenue for GFlowNets lies in exploring alternative training objectives that accurately attribute credit to essential components within longer trajectories. Additionally, while the theory presented in Chapter 4 has been experimentally validated, scaling up the experiments to more complex and high-dimensional spaces will bring new challenges, and more research effort into stabilizing the training of GFlowNets is called for.

Off-policy GFlowNet training

Experiments in Chapter 3 and other studies (Deleu et al., 2022; Zhang et al., 2023a; Jain et al., 2022b; Nishikawa-Toomey et al., 2022; Shen et al., 2023) motivate further research into optimizing the choice of behaviour policy during GFlowNet training to enhance sample efficiency. Our work provides both theoretical justifications and experimental validation, establishing that GFlowNets are more suitable for off-policy training than widely-used alternative inference methods. A natural question is how to obtain trajectories or training samples that are most informative. Notably, this aligns with the exploration problem encountered in reinforcement learning (Section 2.1.5). Inspired by random network distillation (Burda et al., 2018), a well-known exploration technique in RL, Pan et al. (2022) adapted it to enhance exploration in GFlowNets. Leveraging ideas from the field of *Bayesian optimal experiment design*, for which a summary of ideas is provided in Appendix C, to guide the selection of transitions or trajectories for efficient learning, as was done in RL in Mehta et al. (2021), is a direction worth exploring.

Scientific discovery

The fourth paradigm of scientific discovery (Hey et al., 2009) revolves around the idea that scientific breakthroughs can be powered by advanced computing capabilities that help researchers manipulate, explore, and exploit large datasets. This contrasts with the traditional slow cycle of experimental science. Machine learning has already contributed to accelerating the pace of scientific discovery in various domains (MacLeod et al., 2020; Stärk et al., 2022; Angermueller et al., 2019; Kim et al., 2021). In fact GFlowNets (Chapters 3 and 4) were initially motivated by the need of designing drugs in an active learning loop. Bengio et al. (2021); Jain et al. (2022b) successfully used them for biological sequence design, albeit in

controlled simulated environments. Closely related is the exploration of material discovery, encompassing insulating materials (Verichev et al., 2021) to reduce energy-intensive air conditioning, carbon-capture materials (Lin et al., 2012), catalysts and chemical processes for converting earth-abundant molecules into fuels (Tabor et al., 2018), and energy storage solutions (Agarwal et al., 2021).

Bayesian optimization, also known as sequential model optimization (Section 5.5.1), plays a central role in machine learning-based scientific discovery, especially when optimizing costly-to-evaluate physical properties. Inquiries like "Which synthetizable molecule, safe for animals and humans, binds most effectively to a specific target protein?" or "How can abundant molecules be combined to create an insulating material?", while involving at least one non-negative reward function, require exploration of large-sized search spaces[2]. Accelerated scientific discovery could thus benefit from:

— better search procedures in large spaces,

— efficient epistemic uncertainty estimation methods that are central to acquisition functions in Bayesian optimization,

While GFlowNets are natural candidates for the former, as explained in detail in Jain et al. (2023), DEUP (Chapter 5) has the potential of addressing the latter. We have shown how the provided epistemic uncertainty estimates are helpful in toy tasks, but more research is needed in order to scale the method to higher-dimensional spaces, especially those in which Bayesian posterior learning is inapplicable. A challenging task is finding informative stationarizing features that do not require potentially large models on the side.

Bayesian optimal experiment design.

In fact, our research on GFlowNets and epistemic uncertainty estimation could be advantageous for *Bayesian optimal experiment design* (BOED, Appendix C), which is at the core of scientific discovery, as it addresses questions of the type "what experiment should I perform in order to learn the most about the scientific process at hand?". Additionally, BOED finds applications in causal structure discovery (Greenewald et al., 2019; Scherrer et al., 2021; Tigas et al., 2022; Toth et al., 2022), a vital aspect of scientific exploration that seeks to comprehend the causes and effects of diverse phenomena based on observational and interventional data. The central question is that of finding what variables to intervene on and how (i.e., what experiment to perform) in order to obtain interventional data that is the most informative about the sought-for causal structure.

Jain et al. (2023) show how conditional GFlowNets can be used to obtain amortized predictors, taking datasets as inputs, which, as explained in Appendix C, is essential to further improve the performances of sequential experimental design algorithms. The future

2. the pharmacological chemical space is estimated to be of size around 10^{60} (Bohacek et al., 1996).

research directions on scaling and improving GFlowNets discussed above could therefore be of great benefit to BOED.

An interesting open question is whether mutual information-based objectives, such as the popular EIG, can be replaced by other utility functions (Ryan et al., 2016), drawing inspiration from the ideas developed in DEUP (Chapter 5) especially when Bayesian inference is intractable.

Better representations for better generalization

To build more intelligent systems, we need to build systems that are able to generalize. There is a wide gap between the amount of resources (data and computation) that animals need in order to adapt to a new environment and generalize their knowledge for better decision-making and that of machine learning agents in out-of-distribution settings.

The ability to generalize is intimately linked with the ability to compress (Schmidhuber, 2009). In the coming years, increased research focus is expected on developing data representations that leverage the compositionality of the physical world, enabling efficient compression of any given scene.

GFlowNets hold substantial promise for this pursuit, as showcased in the work of Hu et al. (2023), where they were utilized to learn posteriors over latent variables in diverse contexts. Drawing inspiration from the adage "If you want to master something, teach it.", we can posit that agents capable of efficiently generating data from the distribution of a given dataset have likely acquired efficient data compression techniques, revealing essential building blocks and reusable elements. An agent trained to identify these building blocks and evaluate their quality through a generative process could potentially acquire abstract representations that enhance out-of-distribution generalization.

Better representations would pave the way for longer *reasoning* sequences, a missing component in today's large language models and RL algorithms, and would bring us one step closer to training agents that are capable of high-level reasoning. Nonetheless, the question of whether this aligns with our societal goals remains open for further consideration.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

David Ackley. *A connectionist machine for genetic hillclimbing*, volume 28. Springer Science; Business Media, 2012.

Garvit Agarwal, Hieu A Doan, Lily A Robertson, Lu Zhang, and Rajeev S Assary. Discovery of energy storage molecular materials using quantum chemistry-guided multiobjective bayesian optimization. *Chemistry of Materials*, 33(20):8133–8144, 2021.

Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, Lecture Notes in Computer Science, pages 420–434, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-44503-6. doi: 10.1007/3-540-44503-X_27.

Charu C Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and S Yu Philip. Active learning: A survey. In *Data Classification: Algorithms and Applications*, pages 571–605. CRC Press, 2014.

Raj Agrawal, Chandler Squires, Karren Yang, Karthikeyan Shanmugam, and Caroline Uhler. Abcd-strategy: Budgeted experimental design for targeted causal structure discovery. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3400–3409. PMLR, 2019.

David J. Aldous. Exchangeability and related topics. In David J. Aldous, Illdar A. Ibragimov, Jean Jacod, and P. L. Hennequin, editors, *École d'Été de Probabilités de Saint-Flour XIII — 1983*, Lecture Notes in Mathematics, pages 1–198, Berlin, Heidelberg, 1985. Springer.

ISBN 978-3-540-39316-0. doi: 10.1007/BFb0099421.

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive mcmc. *Statistics and computing*, 18:343–373, 2008.

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

Anastasios Nikolas Angelopoulos, Stephen Bates, Michael Jordan, and Jitendra Malik. Uncertainty sets for image classifiers using conformal prediction. In *International Conference on Learning Representations*, 2020.

Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. Model-based reinforcement learning for biological sequence design. In *International conference on learning representations*, 2019.

Javier Antoran, James Allingham, and José Miguel Hernández-Lobato. Depth uncertainty in neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 233–242. PMLR, 06–11 Aug 2017.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv: 1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 2015 International Conference on Learning Representations*, 2015.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

Pedro J Ballester and John BO Mitchell. A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking. *Bioinformatics*, 26(9): 1169–1175, 2010.

S Banerjee and AE Gelfand. On smoothness properties of spatial processes. *Journal of Multivariate Analysis*, 84(1):85–100, 2003.

Adrian Barbu, Song-Chun Zhu, et al. *Monte Carlo Methods*, volume 35. Springer, 2020.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint 1806.01261*, 2018.

Andrew L. Beam and Isaac S. Kohane. Big Data and Machine Learning in Health Care. *JAMA*, 319(13):1317–1318, April 2018. ISSN 0098-7484. doi: 10.1001/jama.2017.18391.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Shai Ben-David, Nadav Eiron, and Philip M. Long. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496–514, May 2003. ISSN 00220000. doi: 10.1016/S0022-0000(03)00038-2.

Bengio. How rogue ais may arise, 2023. https://yoshuabengio.org/2023/05/22/how-rogue-ais-may-arise/,.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *NeurIPS'2021, arXiv:2106.04399*, 2021.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A Neural Probabilistic Language Model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/file/5da713a690c067105aeb2fae32403405-Paper.pdf.

Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *International conference on machine learning*, pages 552–560. PMLR, 2013.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 2023.

Viktor Bengs, Eyke Hüllermeier, and Willem Waegeman. On the difficulty of epistemic uncertainty quantification in machine learning: The case of direct uncertainty estimation through loss minimisation. *arXiv preprint arXiv:2203.06102*, 2022.

Herman Bergwerf. Molview, 2014.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv: Arxiv-1701.02434*, 2017.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006a.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006b.

David M. Blei, Michael I. Jordan, Thomas L. Griffiths, and Joshua B. Tenenbaum. Hierarchical topic models and the nested Chinese restaurant process. *Neural Information Processing Systems (NIPS)*, 2003.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Regine S Bohacek, Colin McMartin, and Wayne C Guida. The art and practice of structure-based drug design: a molecular modeling perspective. *Medicinal research reviews*, 16(1): 3–50, 1996.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. *International Conference on Learning Representations (ICLR)*, 2015.

Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71 (356):791–799, 1976.

George EP Box and Norman R Draper. A basis for the selection of a response surface design. *Journal of the American Statistical Association*, 54(287):622–654, 1959.

Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

Leo Breiman. *Classification and Regression Trees*. Routledge, New York, 1984. ISBN 978-1-315-13947-0. doi: 10.1201/9781315139470.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Selmer Bringsjord and Naveen Sundar Govindarajulu. Artificial Intelligence. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2022 edition, 2022.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are

Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Arthur Earl Bryson. *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

Cameron Buckner and James Garson. Connectionism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2019 edition, 2019.

Lars Buesing, Nicolas Heess, and Theophane Weber. Approximate inference in discrete distributions with monte carlo tree search and value functions, 2019.

Robert Burbidge, Jem J Rowland, and Ross D King. Active learning for regression based on query by committee. In *International conference on intelligent data engineering and automated learning*, pages 209–218. Springer, 2007.

Yuri Burda, Roger Baker Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *International Conference on Learning Representations (ICLR)*, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Olivier Cappé, Eric Moulines, and Tobias Rydén. Inference in hidden markov models. In *Proceedings of EUSFLAT conference*, 2009.

Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

Maria R. Cervera, Rafael Dätwyler, Francesco D'Angelo, Hamza Keurti, Benjamin F. Grewe, and Christian Henning. Uncertainty estimation under model misspecification in neural network regression, 2021.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *arXiv preprint arXiv:1810.08272*, 2018.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *arXiv preprint arXiv: 2306.13831*, 2023.

Rewon Child. Very deep VAEs generalize autoregressive models and can outperform them on images. *International Conference on Learning Representations (ICLR)*, 2021.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28, 2015.

Youngseog Chung, Willie Neiswanger, Ian Char, and Jeff Schneider. Beyond pinball loss: Quantile methods for calibrated uncertainty quantification. *arXiv preprint arXiv:2011.09588*, 2020.

Tomas Cihlar and Marshall Fordyce. Current status and prospects of hiv treatment. *Current opinion in virology*, 18:50–56, 2016.

William Coffey and Yu P Kalmykov. *The Langevin equation: with applications to stochastic problems in physics, chemistry and electrical engineering*, volume 27. World Scientific, 2012.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

Chris Cundy, Aditya Grover, and Stefano Ermon. BCD Nets: Scalable variational approaches for Bayesian causal discovery. *Neural Information Processing Systems (NeurIPS)*, 2021.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989. ISSN 1435-568X. doi: 10.1007/BF02551274.

Hanjun Dai, Rishabh Singh, Bo Dai, Charles Sutton, and Dale Schuurmans. Learning discrete energy-based models via auxiliary-variable local exploration. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *The Journal of Machine Learning Research*, 23(1):10237–10297, 2022.

Francesco D'Angelo and Christian Henning. On out-of-distribution detection with bayesian neural networks. *arXiv preprint arXiv: Arxiv-2110.06020*, 2021.

Luigi De Angelis, Francesco Baglivo, Guglielmo Arzilli, Gaetano Pierpaolo Privitera, Paolo Ferragina, Alberto Eugenio Tozzi, and Caterina Rizzo. Chatgpt and the rise of large language models: the new ai-driven infodemic threat in public health. *Frontiers in Public Health*, 11:1166120, 2023.

Bruno De Finetti. Funzione caratteristica di un fenomeno aleatorio. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*, pages 179–190, 1929.

Augustus De Morgan. *Formal logic: or, the calculus of inference, necessary and probable*. Taylor and Walton, 1847.

Morris H DeGroot. *Optimal statistical decisions*. John Wiley & Sons, 2005.

Tristan Deleu, António Góis, Chris Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. In *Uncertainty in Artificial Intelligence*, pages 518–528. PMLR, 2022.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

Adji Bousso Dieng, Dustin Tran, Rajesh Ranganath, John William Paisley, and David M. Blei. Variational inference via χ upper bound minimization. *Neural Information Processing Systems (NIPS)*, 2017.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *International Conference On Learning Representations*, 2014.

Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-based generative modeling with critically-damped langevin diffusion. *International Conference on Learning Representations (ICLR)*, 2022.

Justin Domke and Daniel Sheldon. Importance weighting and variational inference. *Neural Information Processing Systems (NeurIPS)*, 2018.

Arnaud Doucet, Nando Freitas, and Neil Gordon. *An Introduction to Sequential Monte Carlo Methods*, pages 3–14. Springer New York, 2001.

Michael W. Dusenberry, Ghassen Jerfel, Yeming Wen, Y. Ma, Jasper Snoek, K. Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *International Conference on Machine Learning (ICML)*, 2020.

Morris L Eaton. Multivariate statistics: a vector space approach. *(No Title)*, 1983.

Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

David Eriksson, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *arXiv preprint arXiv:1910.01739*, 2019.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Absalom E. Ezugwu, Abiodun M. Ikotun, Olaide O. Oyelade, Laith Abualigah, Jeffery O. Agushaka, Christopher I. Eke, and Andronicus A. Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743, April 2022. ISSN 0952-1976. doi: 10.1016/j.engappai.2022.104743.

Clara Fannjiang, Stephen Bates, Anastasios Angelopoulos, Jennifer Listgarten, and Michael I Jordan. Conformal prediction for the design problem. *arXiv preprint arXiv:2202.03613*, 2022.

Valerii Vadimovich Fedorov. *Theory of optimal experiments*. Academic Press, New York, 1972.

Adam Foster, Martin Jankowiak, Elias Bingham, Paul Horsfall, Yee Whye Teh, Thomas Rainforth, and Noah Goodman. Variational bayesian optimal experimental design. *Advances in Neural Information Processing Systems*, 32, 2019.

Adam Foster, Martin Jankowiak, Matthew O'Meara, Yee Whye Teh, and Tom Rainforth. A unified stochastic gradient approach to designing bayesian-optimal experiments. In *International Conference on Artificial Intelligence and Statistics*, pages 2959–2969. PMLR, 2020.

Adam Foster, Desi R Ivanova, Ilyas Malik, and Tom Rainforth. Deep adaptive design: Amortizing sequential bayesian experimental design. In *International Conference on Machine Learning*, pages 3384–3395. PMLR, 2021.

Vincent François-Lavet, David Taralla, Damien Ernst, and Raphaël Fonteneau. Deep reinforcement learning solutions for energy microgrids management. In *European Workshop on Reinforcement Learning (EWRL 2016)*, 2016.

P. Frazier. A tutorial on bayesian optimization. *ArXiv*, abs/1807.02811, 2018.

Yoav Freund, H Sebastian Seung, Eli Shamir, and Naftali Tishby. Information, prediction, and query by committee. *Advances in neural information processing systems*, 5, 1992.

Nir Friedman and Daphne Koller. Being bayesian about network structure. In Craig Boutilier and Moisés Goldszmidt, editors, *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000*, pages 201–210. Morgan Kaufmann, 2000.

Nir Friedman, Moises Goldszmidt, and Abraham Wyner. Data analysis with bayesian networks: A bootstrap approach. *arXiv preprint arXiv: 1301.6695*, 2013.

Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, January 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90003-8.

Futoshi Futami, Tomoharu Iwata, Naonori Ueda, Issei Sato, and Masashi Sugiyama. Excess risk analysis for epistemic uncertainty with application to variational inference. *arXiv preprint arXiv: Arxiv-2206.01606*, 2022.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1050–1059. JMLR.org, 2016a.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1050–1059. PMLR, 2016b.

Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR, 2017.

Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. in preparation.

Dan Geiger and David Heckerman. Learning Gaussian networks. In *Uncertainty Proceedings 1994*, pages 235–243. Elsevier, 1994.

Alan E Gelfand and Adrian FM Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409, 1990.

Andrew Gelman and Cosma Rohilla Shalizi. Philosophy and the practice of Bayesian statistics. *The British journal of mathematical and statistical psychology*, 66(1):8–38, February 2013. ISSN 0007-1102. doi: 10.1111/j.2044-8317.2011.02037.x.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, pages 721–741, 1984.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

Marzyeh Ghassemi, Tristan Naumann, Peter Schulam, Andrew L. Beam, Irene Y. Chen, and Rajesh Ranganath. A Review of Challenges and Opportunities in Machine Learning for Health. *AMIA Summits on Translational Science Proceedings*, 2020:191–200, May 2020. ISSN 2153-4063.

Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Query by committee made real. *Advances in neural information processing systems*, 18, 2005.

Walter R Gilks and Pascal Wild. Adaptive rejection sampling for gibbs sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(2):337–348, 1992.

Irving J Good. Monte carlo method. *ErIcyc/Oneok Qt SCI'er2ce ond Teohrlo/Ogy, 17th edn. vol. Vn (MCGraw-Hill, 1992)*, pages 414–416, 1960.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Kristjanson Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *International Conference on Learning Representations (ICLR)*, 2018.

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris J. Maddison. Oops i took a gradient: Scalable sampling for discrete distributions, 2021.

Kristjan Greenewald, Dmitriy Katz, Karthikeyan Shanmugam, Sara Magliacane, Murat Kocaoglu, Enric Boix Adsera, and Guy Bresler. Sample efficient active learning of causal trees. *Advances in Neural Information Processing Systems*, 32, 2019.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep AutoRegressive networks. *International Conference on Machine Learning (ICML)*, 2014.

Thomas L. Griffiths. Understanding Human Intelligence through Human Limitations, September 2020. arXiv:2009.14050 [cs].

Peter Grünwald. The safe bayesian. In *International Conference on Algorithmic Learning Theory*, pages 169–183. Springer, 2012.

Peter Grünwald and Thijs Van Ommen. Inconsistency of bayesian inference for misspecified linear models, and a proposal for repairing it. *Bayesian Analysis*, 12(4):1069–1103, 2017.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

Danijar Hafner, Dustin Tran, Timothy Lillicrap, Alex Irpan, and James Davidson. Noise contrastive priors for functional uncertainty. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

Paul Hagemann, Johannes Hertrich, and Gabriele Steidl. Stochastic normalizing flows for inverse problems: A Markov chains viewpoint. *SIAM/ASA Journal on Uncertainty Quantification*, 10(3):1162–1190, 2022.

Matan Haroush, Tzviel Frostig, Ruth Heller, and Daniel Soudry. A statistical framework for efficient out of distribution detection in deep neural networks. In *International Conference on Learning Representations*, 2021.

Johan Håstad. *Computational limitations for small depth circuits*. PhD thesis, Massachusetts Institute of Technology, 1986.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

WK Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, pages 97–109, 1970.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yang-Bo He and Zhi Geng. Active learning of causal networks with intervention experiments and optimal designs. *Journal of Machine Learning Research*, 9(Nov):2523–2547, 2008.

Yang-Hui He, Elli Heyes, and Edward Hirst. Machine Learning in Physics and Geometry, March 2023. arXiv:2303.12626 [hep-th, physics:math-ph].

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Thang D. Bui, Daniel Hernández-Lobato, and Richard E. Turner. Black-box alpha divergence minimization. *International Conference on Machine Learning (ICML)*, 2016.

Luke B. Hewitt, Tuan Anh Le, and Joshua B. Tenenbaum. Learning to learn generative programs with memoised wake-sleep. *Uncertainty in Artificial Intelligence (UAI)*, 2020.

Tony Hey, Stewart Tansley, Kristin Tolle, and Jim Gray. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, October 2009. ISBN 978-0-9825442-0-4.

Charles Robert Hicks. *Fundamental Concepts in the Design of Experiments*. New York,: Holt, Rinehart and Winston, 1964.

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and R M Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268 5214:1158–61, 1995.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020a.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Neural Information Processing Systems (NeurIPS)*, 2020b.

Thomas Hobbes. *Leviathan*. Routledge, 1651. ISBN 978-1-315-50760-6. Google-Books-ID: Twk3DAAAQBAJ.

Matthew D. Hoffman and Andrew Gelman. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research (JMLR)*, 15: 1593–1623, 2011.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research (JMLR)*, 14:1303–1347, 2013.

Lara Hoffmann and Clemens Elster. Deep ensembles from a bayesian perspective. *arXiv preprint arXiv:2105.13283*, 2021.

Liang Hong and Ryan Martin. Model misspecification, bayesian versus credibility estimation, and gibbs posteriors. *Scandinavian Actuarial Journal*, 2020(7):634–649, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8.

Edward J Hu, Nikolay Malkin, Moksh Jain, Katie E Everett, Alexandros Graikos, and Yoshua Bengio. Gflownet-em for learning compositional latent variable models. In *International Conference on Machine Learning*, pages 13528–13549. PMLR, 2023.

S. Hu, Nicola Pezzotti, and M. Welling. A new perspective on uncertainty quantification of deep ensembles. *arXiv*, 2020.

Aapo Hyvärinen. Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. ISSN 1533-7928.

Alan Hájek. "Mises Redux" — Redux: Fifteen Arguments Against Finite Frequentism. In Domenico Costantini and Maria Carla Galavotti, editors, *Probability, Dynamics and Causality: Essays in Honour of Richard C. Jeffrey*, pages 69–87. Springer Netherlands, Dordrecht, 1997. ISBN 978-94-011-5712-4. doi: 10.1007/978-94-011-5712-4_5.

Alan Hájek. Dutch Book Arguments. In Paul Anand, Prasanta Pattanaik, and Clemens Puppe, editors, *The Oxford Handbook of Rational and Social Choice*. Oxford University Press, 2008.

Alan Hájek. Interpretations of Probability. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2019 edition, 2019.

E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Mach. Learn.*, 2019. doi: 10.1007/s10994-021-05946-3.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference On Machine Learning*, 2015.

Desi R Ivanova, Adam Foster, Steven Kleinegesse, Michael U Gutmann, and Thomas Rainforth. Implicit deep adaptive design: policy-based experimental design without likelihoods. *Advances in Neural Information Processing Systems*, 34:25785–25798, 2021.

Pavel Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Tommi S Jaakkola. Tutorial on variational approximation methods. *Advanced mean field methods: theory and practice*, pages 129–159, 2000.

Tommi S Jaakkola and Michael I Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10:25–37, 2000.

Pierre Jacob. Intentionality. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2023 edition, 2023.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure F P Dossou, Chanakya Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, Lena Simine, Payel Das, and Yoshua Bengio. Biological sequence design with gflownets. *International Conference on Machine Learning (ICML)*, 2022a.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure F.P. Dossou, Chanakya Ekbote, Jie Fu, Tianyu Zhang, Micheal Kilgour, Dinghuai Zhang, Lena Simine, Payel Das, and Yoshua Bengio. Biological sequence design with GFlowNets. *International Conference on Machine Learning (ICML)*, 2022b.

Moksh Jain, Tristan Deleu, Jason Hartford, Cheng-Hao Liu, Alex Hernandez-Garcia, and Yoshua Bengio. Gflownets for ai-driven scientific discovery. *Digital Discovery*, 2(3):557–577, 2023.

David Janz, Jiri Hron, Przemysław Mazur, Katja Hofmann, José Miguel Hernández-Lobato, and Sebastian Tschiatschek. Successor uncertainties: Exploration and uncertainty in temporal difference learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

N. Japkowicz, null Jos, and M. A. Gluck. Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, 12(3):531–545, March 2000. ISSN 0899-7667. doi: 10.1162/089976600300015691.

Ajay Jasra, Chris C Holmes, and David A Stephens. Markov chain monte carlo methods and the label switching problem in bayesian mixture modeling. *Statistical Science*, pages 50–67, 2005.

E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, Cambridge, UK ; New York, NY, annotated edition edition, April 2003. ISBN 978-0-521-59271-0.

Tony Jebara. Generative Versus Discriminative Learning. In Tony Jebara, editor, *Machine Learning: Discriminative and Generative*, The International Series in Engineering and Computer Science, pages 17–60. Springer US, Boston, MA, 2004. ISBN 978-1-4419-9011-2. doi: 10.1007/978-1-4419-9011-2_2.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Chapter 11. junction tree variational autoencoder for molecular graph generation. *Drug Discovery*, page 228–249, 2020. ISSN 2041-3211.

Bowen Jing, Gabriele Corso, Regina Barzilay Jeffrey Chang, and Tommi Jaakkola. Torsional diffusion for molecular conformer generation. *Neural Information Processing Systems (NeurIPS)*, 2022.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37:183–233, 1999.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Daniel Kahneman, Paul Slovic, and Amos Tversky. *Judgment under uncertainty: Heuristics and biases*. Cambridge university press, 1982.

Satyen Kale, Ayush Sekhari, and Karthik Sridharan. Sgd: The role of implicit regularization, batch-size and multiple-epochs. In *NeurIPS*, 2021.

Annemarie Kalis. No Intentions in the Brain: A Wittgensteinian Perspective on the Science of Intention. *Frontiers in Psychology*, 10, 2019. ISSN 1664-1078.

Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition, 2023. URL https://cs231n.github.io/classification/.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

G. Jay. Kerns and Gábor J. Székely. Definetti's Theorem for Abstract Finite Exchangeable Sequences. *Journal of Theoretical Probability*, 19(3):589–608, December 2006. ISSN 1572-9230. doi: 10.1007/s10959-006-0028-z.

Samuel Kim, Peter Y Lu, Charlotte Loh, Jamie Smith, Jasper Snoek, and Marin Soljacic. Deep learning for bayesian optimization of scientific problems with high-dimensional structure. *arXiv preprint arXiv:2104.11667*, 2021.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference On Learning Representations*, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014a.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *International Conference on Learning Representations (ICLR)*, 2014b.

Werner Kirsch. An elementary proof of de Finetti's Theorem, September 2018. arXiv:1809.00882 [math].

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009. ISSN 0167-4730. doi: https://doi.org/10.1016/j.strusafe.2008.06.020. Risk Acceptance and Risk Communication.

Bas JK Kleijn and Aad W van der Vaart. The bernstein-von-mises theorem under misspecification. *Electronic Journal of Statistics*, 6:354–381, 2012.

Steven Kleinegesse and Michael U Gutmann. Efficient bayesian experimental design for implicit models. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 476–485. PMLR, 2019.

Steven Kleinegesse and Michael U Gutmann. Gradient-based bayesian experimental design for implicit models using mutual information lower bounds. *arXiv preprint arXiv:2105.04379*, 2021.

NP Klepikov and SN Sokolov. Analysis and design of experiments by the maximum likelihood method, 1964.

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. An optimization-centric view on bayes' rule: Reviewing and generalizing variational inference. *Journal of Machine Learning Research*, 23(132):1–109, 2022.

Ivan Kobyzev, Simon JD Prince, and Marcus A Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.

Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for decision making*. MIT press, 2022.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

AN Kolmogorov. *Foundations of the theory of probability*. Chelsea Publishing Co., 1950.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

Oliver Kramer. K-Nearest Neighbors. In Oliver Kramer, editor, *Dimensionality Reduction with Unsupervised Nearest Neighbors*, Intelligent Systems Reference Library, pages 13–23. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-38652-7. doi: 10.1007/978-3-642-38652-7_2.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks. *International Conference On Machine Learning*, 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Jack Kuipers, Giusi Moffa, and David Heckerman. Addendum on the scoring of Gaussian directed acyclic graphical models. *The Annals of Statistics*, 42(4):1689–1691, 2014.

Ashutosh Kumar, Arnout Voet, and Kam Y.J. Zhang. Fragment based drug design: from experimental to computational approaches. *Current medicinal chemistry*, 19(30):5128–5147, 2012.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

Simon Lacoste-Julien. IFT 6269 : Probabilistic Graphical Models - Fall 2022, 2016. URL https://www.iro.umontreal.ca/~slacoste/teaching/ift6269/A16/.

Salem Lahlou, Moksh Jain, Hadi Nekoei, V. Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov, and Yoshua Bengio. Deup: Direct epistemic uncertainty prediction. *Trans. Mach. Learn. Res.*, 2021.

Salem Lahlou, T. Deleu, Pablo Lemos, Dinghuai Zhang, Alexandra Volokhova, Alex Hernández-García, L'ena N'ehale Ezzine, Y. Bengio, and Nikolay Malkin. A theory of

continuous generative flow networks. *International Conference on Machine Learning (ICML)*, 2023a.

Salem Lahlou, Joseph D. Viviano, and Victor Schmidt. torchgfn: A pytorch gflownet library. *arXiv preprint arXiv: 2305.14594*, 2023b.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017a.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems (NeurIPS)*, 2017b.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

Junpeng Lao, Christopher Suter, Ian Langmore, Cyril Chimisov, Ashish Saxena, Pavel Sountsov, Dave Moore, Rif A Saurous, Matthew D Hoffman, and Joshua V Dillon. tfp. mcmc: Modern markov chain monte carlo tools built for modern hardware. *arXiv preprint arXiv:2002.01184*, 2020.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011.

Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

Tuan Anh Le, Adam R. Kosiorek, N. Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep for models with stochastic control flow. *Uncertainty in Artificial Intelligence (UAI)*, 2019.

Tuan Anh Le, Katherine M. Collins, Luke B. Hewitt, Kevin Ellis, N. Siddharth, Samuel J. Gershman, and Joshua B. Tenenbaum. Hybrid memoised wake-sleep: Approximate inference at the discrete-continuous interface. *International Conference on Learning Representations (ICLR)*, 2022.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539. Number: 7553 Publisher: Nature Publishing Group.

Mei-Ling Ting Lee, Frank C Kuo, GA Whitmore, and Jeffrey Sklar. Importance of replication in microarray gene expression studies: statistical methods and evidence from repetitive cdna hybridizations. *Proceedings of the National Academy of Sciences*, 97(18):9834–9839, 2000.

Janet Levin. Functionalism. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2023

edition, 2023.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Yinchuan Li, Shuang Luo, Haozhi Wang, and Jianye Hao. CFlowNets: Continuous control with generative flow networks. *International Conference on Learning Representations (ICLR)*, 2023.

Yingzhen Li, José Miguel Hernández-Lobato, and Richard E. Turner. Stochastic expectation propagation. *Neural Information Processing Systems (NIPS)*, 2015.

Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European conference on computer vision (ECCV)*, pages 584–599, 2018.

Vincent Lim, Ellen Novoseller, Jeffrey Ichnowski, Huang Huang, and Ken Goldberg. Policy-based bayesian experimental design for non-differentiable implicit models. *arXiv preprint arXiv:2203.04272*, 2022.

Li-Chiang Lin, Adam H Berger, Richard L Martin, Jihan Kim, Joseph A Swisher, Kuldeep Jariwala, Chris H Rycroft, Abhoyjit S Bhown, Michael W Deem, Maciej Haranczyk, et al. In silico screening of carbon-capture materials. *Nature materials*, 11(7):633–641, 2012.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. Conformal prediction with temporal quantile adjustments. *ArXiv*, abs/2205.09940, 2022.

Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970.

Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, June 1976. ISSN 1572-9125. doi: 10.1007/BF01931367.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint 2210.02747*, 2022.

Dianbo Liu, Moksh Jain, Bonaventure F. P. Dossou, Qianli Shen, Salem Lahlou, Anirudh Goyal, Nikolay Malkin, Chris Chinenye Emezue, Dinghuai Zhang, Nadhir Hassen, Xu Ji, Kenji Kawaguchi, and Yoshua Bengio. GFlowOut: Dropout with generative flow networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 21715–21729. PMLR, 23–29 Jul 2023.

Jeremiah Zhe Liu, Zian Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax-Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. In *Neural Information Processing Systems (NeurIPS)*,

2020.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose Bayesian inference algorithm. *Neural Information Processing Systems (NIPS)*, 2016.

S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982. ISSN 1557-9654. doi: 10.1109/TIT.1982.1056489. Conference Name: IEEE Transactions on Information Theory.

Lars Lorch, Jonas Rothfuss, Bernhard Schölkopf, and Andreas Krause. Dibs: Differentiable bayesian structure learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 24111–24123, 2021.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. *International Conference on Learning Representations (ICLR)*, 2017.

Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. BIVA: A very deep hierarchy of latent variables for generative modeling. *Neural Information Processing Systems (NeurIPS)*, 2019.

David JC MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

Benjamin P MacLeod, Fraser GL Parlane, Thomas D Morrissey, Florian Häse, Loïc M Roch, Kevan E Dettelbach, Raphaell Moreira, Lars PE Yunker, Michael B Rooney, Joseph R Deeth, et al. Self-driving laboratory for accelerated discovery of thin-film materials. *Science Advances*, 6(20):eaaz8867, 2020.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, volume 5.1, pages 281–298. University of California Press, January 1967.

Kanika Madan, Jarrid Rector-Brooks, Maksym Korablyov, Emmanuel Bengio, Moksh Jain, Andrei Nica, Tom Bosc, Yoshua Bengio, and Nikolay Malkin. Learning GFlowNets from partial episodes for improved convergence and stability. *arXiv preprint 2209.12782*, 2022a.

Kanika Madan, Jarrid Rector-Brooks, Maksym Korablyov, Emmanuel Bengio, Moksh Jain, Andrei Nica, Tom Bosc, Yoshua Bengio, and Nikolay Malkin. Learning GFlowNets from partial episodes for improved convergence and stability. *arXiv preprint 2209.12782*, 2022b.

Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Shreshth A. Malik, Salem Lahlou, Andrew Jesson, Moksh Jain, Nikolay Malkin, Tristan Deleu, Yoshua Bengio, and Yarin Gal. Batchgfn: Generative flow networks for batch active learning. *arXiv preprint arXiv: 2306.15058*, 2023.

Andrey Malinin, Bruno Mlodozeniec, and Mark Gales. Ensemble distribution distillation. In *International Conference on Learning Representations*, 2020.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *arXiv preprint arXiv:2201.13259*, 2022.

Nikolay Malkin, Salem Lahlou, Tristan Deleu, Xu Ji, Edward Hu, Katie Everett, Dinghuai Zhang, and Yoshua Bengio. GFlowNets and variational inference. *International Conference on Learning Representations (ICLR)*, 2023.

Alina Malyutina, Muntasir Mamun Majumder, Wenyu Wang, Alberto Pessia, Caroline A Heckman, and Jing Tang. Drug combination sensitivity scoring facilitates the discovery of synergistic and efficacious drug combinations in cancer. *PLoS computational biology*, 15 (5):e1006752, 2019.

Andres Masegosa. Learning under model misspecification: Applications to variational and ensemble methods. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, 2020.

Vaden Masrani, Tuan Anh Le, and Frank D. Wood. The thermodynamic variational objective. *Neural Information Processing Systems (NeurIPS)*, 2019.

James L. Mcclelland, David E. Rumelhart, and PDP Research Group. *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*. MIT Press, July 1987. ISBN 978-0-262-63110-5. Google-Books-ID: davmLgzusB8C.

Margaret Meek Lange. Progress. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2023 edition, 2023.

Viraj Mehta, Biswajit Paria, Jeff Schneider, Stefano Ermon, and Willie Neiswanger. An experimental design perspective on model-based reinforcement learning. *arXiv preprint arXiv:2112.05244*, 2021.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

George A. Miller. The cognitive revolution: a historical perspective. *Trends in Cognitive Sciences*, 7(3):141–144, March 2003. ISSN 1364-6613, 1879-307X. doi: 10.1016/S1364-6613(03)00029-9. Publisher: Elsevier.

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. *arXiv preprint 1301.2294*, 2001.

Tom Minka et al. Divergence measures and message passing. Technical report, Citeseer, 2005.

Vladimir Mironov, Yuri Alexeev, Vikram Khipple Mulligan, and Dmitri G. Fedorov. A systematic study of minima in alanine dipeptide. *Journal of Computational Chemistry*,

40(2):297–309, 2018.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *International Conference on Machine Learning (ICML)*, 2014.

Andriy Mnih and Danilo Jimenez Rezende. Variational inference for Monte Carlo objectives. *International Conference on Machine Learning (ICML)*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

Reza Bayat Mokhtari, Tina S Homayouni, Narges Baluch, Evgeniya Morgatskaya, Sushil Kumar, Bikul Das, and Herman Yeger. Combination therapy in combating cancer. *Oncotarget*, 8(23):38022, 2017.

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *NIPS*, 2014.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Harry L Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5 (2):107–113, 1965.

Warren Morningstar, Cusuh Ham, Andrew Gallagher, Balaji Lakshminarayanan, Alex Alemi, and Joshua Dillon. Density of states estimation for out of distribution detection. In *International Conference on Artificial Intelligence and Statistics*, pages 3232–3240. PMLR, 2021.

Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip H. S. Torr, and Yarin Gal. Deep deterministic uncertainty: A simple baseline, 2021.

Peter Müller. Simulation based optimal design. *Handbook of Statistics*, 25:509–518, 2005.

Kevin P Murphy. Active learning of causal bayes net structure. Technical report, technical report, UC Berkeley, 2001.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892*, 2017.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *arXiv preprint arXiv:1906.04733*, 2019.

Zachary Nado, Neil Band, Mark Collier, Josip Djolonga, Michael Dusenberry, Sebastian Farquhar, Angelos Filos, Marton Havasi, Rodolphe Jenatton, Ghassen Jerfel, Jeremiah Liu, Zelda Mariet, Jeremy Nixon, Shreyas Padhy, Jie Ren, Tim Rudner, Yeming Wen, Florian Wenzel, Kevin Murphy, D. Sculley, Balaji Lakshminarayanan, Jasper Snoek, Yarin Gal, and Dustin Tran. Uncertainty Baselines: Benchmarks for uncertainty & robustness in deep learning. *arXiv preprint arXiv:2106.04015*, 2021.

Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. *arXiv preprint arXiv:1909.11652*, 2019.

Vasili Vasilevich Nalimov and Nataliia Andreevana Chernova. Statistical methods for design of extremal experiments. Technical report, FOREIGN TECHNOLOGY DIV WRIGHT-PATTERSON AFB OHIO, 1968.

Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *NIPS'2010*, pages 1786–1794, 2010.

Charlie Nash and Conor Durkan. Autoregressive energy machines. In *International Conference on Machine Learning*, pages 1735–1744. PMLR, 2019.

Radford M Neal. Annealed importance sampling. *Statistics and computing*, 11:125–139, 2001a.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001b.

Radford M Neal. Slice sampling. *The annals of statistics*, 31(3):705–767, 2003.

Radford M. Neal. MCMC using Hamiltonian dynamics. *arXiv preprint 1206.1901*, 2012.

Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Von Neumann. Various techniques used in connection with random digits. *Notes by GE Forsythe*, pages 36–38, 1951.

Tan T. Nguyen and Scott Sanner. Algorithms for direct 0-1 loss optimization in binary classification. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1085–III–1093, Atlanta, GA, USA, 2013. JMLR.org.

Vu-Linh Nguyen, Sébastien Destercke, and Eyke Hüllermeier. Epistemic uncertainty sampling. In *International Conference on Discovery Science*, pages 72–86. Springer, 2019.

Mizu Nishikawa-Toomey, Tristan Deleu, Jithendaraa Subramanian, Yoshua Bengio, and Laurent Charlin. Bayesian learning of causal structure and mechanisms with GFlowNets and variational bayes. *arXiv preprint 2211.02763*, 2022.

Esa Nummelin. *General irreducible Markov chains and non-negative operators*. Cambridge University Press, 2004.

OpenAI. Gpt-4 technical report. *PREPRINT*, 2023.

World Health Organization and Stop TB Initiative. *Treatment of tuberculosis: guidelines*. World Health Organization, 2010.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, December 2022.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, 2019.

John William Paisley, David M. Blei, and Michael I. Jordan. Variational Bayesian inference with stochastic search. *International Conference on Machine Learning (ICML)*, 2012.

Ling Pan, Dinghuai Zhang, Aaron Courville, Longbo Huang, and Yoshua Bengio. Generative augmented flow networks. *International Conference on Learning Representations (ICLR)*, 2022.

Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Neural Information Processing Systems (NeurIPS)*, 2017.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Dimitri Petritis. Markov chains on measurable spaces. *Université de Rennes, UFR Mathématiques. perso. univ-rennes1. fr/dimitri. petritis/.../markov/markov. pdf*, 2012.

Mary Phuong and Marcus Hutter. Formal algorithms for transformers. *ARXIV.ORG*, 2022. doi: 10.48550/arXiv.2207.09238.

Steven Pinker and Alan Prince. On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition*, 28:73–193, 1988. ISSN 1873-7838. doi: 10.1016/0010-0277(88)90032-7. Place: Netherlands Publisher: Elsevier Science.

Emilia Pompe, Chris Holmes, and Krzysztof Łatuszyński. A framework for adaptive mcmc targeting multimodal distributions. *The Annals of Statistics*, 48(5):2930–2952, 2020.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

M. Raghu, Ben Poole, J. Kleinberg, S. Ganguli, and Jascha Narain Sohl-Dickstein. On the expressive power of deep neural networks. *International Conference On Machine Learning*, 2016.

Tom Rainforth, Adam R. Kosiorek, Tuan Anh Le, Chris J. Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. Tighter variational bounds are not necessarily better. *International Conference on Machine Learning (ICML)*, 2018.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8821–8831. PMLR, July 2021. ISSN: 2640-3498.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv: 2204.06125*, 2022.

FP Ramsey. Truth and probability. reprinted in. *Studies in subjective probability*, pages 61–92, 1926.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. *Artificial Intelligence and Statistics (AISTATS)*, 2014.

Rajesh Ranganath, Dustin Tran, Jaan Altosaar, and David M. Blei. Operator variational inference. *Neural Information Processing Systems (NIPS)*, 2016a.

Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. *International Conference on Machine Learning (ICML)*, 2016b.

Michael Rescorla. The Computational Theory of Mind. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2020 edition, 2020.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, pages 1530–1538. PMLR, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning (ICML)*, 2014a.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286. PMLR, June 2014b. ISSN: 1938-7228.

Lorenz Richter, Ayman Boustati, Nikolas Nüsken, Francisco J. R. Ruiz, and Ömer Deniz Akyildiz. VarGrad: A low-variance gradient estimator for variational inference. *Neural Information Processing Systems (NeurIPS)*, 2020.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

Salah Rifai, Yann N Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. *Advances in neural information processing systems*, 24:2294–2302, 2011.

C. Riquelme, G. Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *International Conference On Learning Representations*, 2018.

Howard Robinson. Dualism. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2023 edition, 2023.

Frank Rosenblatt et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, volume 55. Spartan books Washington, DC, 1962.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.

David E. Rumelhart, James L. Mcclelland, and PDP Research Group. *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, July 1987. ISBN 978-0-262-68053-0. Google-Books-ID: lOSMEAAAQBAJ.

Stuart J. Russell, Peter Norvig, and Ernest Davis. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, 3rd ed edition, 2010. ISBN 978-0-13-604259-4.

Elizabeth G Ryan, Christopher C Drovandi, James M McGree, and Anthony N Pettitt. A review of modern computational algorithms for bayesian optimal design. *International Statistical Review*, 84(1):128–154, 2016.

Tim Salimans and David A. Knowles. Fixed-Form Variational Posterior Approximation through Stochastic Linear Regression. *Bayesian Analysis*, 8(4):837–882, December 2013. ISSN 1936-0975, 1931-6690. doi: 10.1214/13-BA858. Publisher: International Society for Bayesian Analysis.

Lawrence K. Saul, T. Jaakkola, and Michael I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, December 2017. ISSN 0925-2312. doi: 10.1016/j.neucom.2017.06.053.

Nino Scherrer, Olexa Bilaniuk, Yashas Annadani, Anirudh Goyal, Patrick Schwab, Bernhard Schölkopf, Michael C Mozer, Yoshua Bengio, Stefan Bauer, and Nan Rosemary Ke. Learning neural causal models with active interventions. *arXiv preprint arXiv:2109.02429*, 2021.

Bernhard Schiilkopf. The kernel trick for distances. *Advances in neural information processing systems*, 13:301–307, 2001.

Juergen Schmidhuber. Driven by Compression Progress: A Simple Principle Explains Essential Aspects of Subjective Beauty, Novelty, Surprise, Interestingness, Attention, Curiosity, Creativity, Art, Science, Music, Jokes, April 2009. arXiv:0812.4360 [cs].

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61: 85–117, January 2015. ISSN 0893-6080. doi: 10.1016/j.neunet.2014.09.003.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Advances in Neural Information Processing Systems 30*, June 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Nicholas J Schurch, Pietá Schofield, Marek Gierliński, Christian Cole, Alexander Sherstnev, Vijender Singh, Nicola Wrobel, Karim Gharbi, Gordon G Simpson, Tom Owen-Hughes, et al. How many biological replicates are needed in an rna-seq experiment and which differential expression tool should you use? *Rna*, 22(6):839–851, 2016.

Ari Seff, Wenda Zhou, Farhan Damani, Abigail Doyle, and Ryan P Adams. Discrete object generation with reversible inductive construction. *arXiv preprint arXiv:1907.08268*, 2019.

Burr Settles. Active learning literature survey. *Machine Learning*, 15(2):201–221, 1994.

H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

Max W. Shen, Emmanuel Bengio, Ehsan Hajiramezanali, Andreas Loukas, Kyunghyun Cho, and Tommaso Biancalani. Towards understanding and improving gflownet training. *arXiv preprint arXiv: 2305.07170*, 2023.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017b.

B. Skyrms. *Pragmatics and Empiricism*. Yale University Press, 1984. ISBN 978-0-300-03174-4.

Merritt Roe Smith and Leo Marx. *Does technology drive history?: The dilemma of technological determinism*. Mit Press, 1994.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems (NeurIPS)*, 2012.

Artem Sobolev and Dmitry Vetrov. Importance weighted hierarchical variational inference. *Neural Information Processing Systems (NeurIPS)*, 2019.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015a.

Jascha Narain Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *International Conference on Machine Learning (ICML)*, 2015b.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. *Neural Information Processing Systems (NIPS)*, 2016.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Neural Information Processing Systems (NeurIPS)*, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Hannes Stärk, Dominique Beaini, Gabriele Corso, Prudencio Tossou, Christian Dallago, Stephan Günnemann, and Pietro Liò. 3d infomax improves gnns for molecular property prediction. In *International Conference on Machine Learning*, pages 20479–20502. PMLR, 2022.

Aravind Subramanian, Rajiv Narayan, Steven M Corsello, David D Peck, Ted E Natoli, Xiaodong Lu, Joshua Gould, John F Davis, Andrew A Tubelli, Jacob K Asiedu, et al. A next generation connectivity map: L1000 platform and the first 1,000,000 profiles. *Cell*, 171(6):1437–1452, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Daniel P Tabor, Loic M Roch, Semion K Saikin, Christoph Kreisbeck, Dennis Sheberla, Joseph H Montoya, Shyam Dwaraknath, Muratahan Aykol, Carlos Ortiz, Hermann Tribukait, et al. Accelerating the discovery of materials for clean energy in the era of smart automation. *Nature reviews materials*, 3(5):5–20, 2018.

Natasa Tagasovska and David Lopez-Paz. Single-model uncertainties for deep learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Akinori Tanaka, Akio Tomiya, and Koji Hashimoto. *Deep Learning and Physics*. Mathematical Physics Studies. Springer, Singapore, 2021. ISBN 978-981-336-107-2 978-981-336-108-9. doi: 10.1007/978-981-33-6108-9.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2017.

A. Tarantola and B. Valette. Inverse problems = Quest for information. *Journal of Geophysics*, 50(1):159–170, October 1981. ISSN 2643-9271.

Luca Thiede, Santiago Miret, Krzysztof Sadowski, Haoping Xu, Mariano Phielipp, and Alan Aspuru-Guzik. Conformer search using SE3-transformers and imitation learning. *NeurIPS 2022 AI for Accelerated Materials Design Workshop*, 2022.

Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Panagiotis Tigas, Yashas Annadani, Andrew Jesson, Bernhard Schölkopf, Yarin Gal, and Stefan Bauer. Interventions, where and how? experimental design for causal models at scale. *arXiv preprint arXiv:2203.02016*, 2022.

Michalis K. Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. *International Conference on Machine Learning (ICML)*, 2014.

Simon Tong and Daphne Koller. Active learning for structure in bayesian networks. In *International joint conference on artificial intelligence*, volume 17, pages 863–869. Citeseer, 2001.

Christian Toth, Lars Lorch, Christian Knoll, Andreas Krause, Franz Pernkopf, Robert Peharz, and Julius von Kügelgen. Active bayesian causal inference. *arXiv preprint arXiv: Arxiv-2206.02063*, 2022.

Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 945–952, 2006.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

Minh-Ngoc Tran, Trong-Nghia Nguyen, and Viet-Hung Dao. A practical tutorial on Variational Bayes, March 2021. arXiv:2103.01327 [stat].

Oleg Trott and Arthur J Olson. AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2):455–461, 2010.

George Tucker, Andriy Mnih, Chris J. Maddison, John Lawson, and Jascha Narain Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. *Neural Information Processing Systems (NIPS)*, 2017.

A. M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, 1950. ISSN 0026-4423. Publisher: [Oxford University Press, Mind Association].

Joerg Tuske. The Concept of Emotion in Classical Indian Philosophy. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2021 edition, 2021.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv: 1607.08022*, 2016.

Jonas Umlauft, Armin Lederer, T. Beckers, and S. Hirche. Real-time uncertainty decomposition for online learning control. *ArXiv*, abs/2010.02613, 2020.

UnitedNations. Global leaders agree on the challenges facing humanity – why can't we agree on action? | United Nations Secretary-General, 2022.

Meet P. Vadera, Adam D. Cobb, Borhan Jalaeian, and Benjamin M Marlin. Ursabench: Comprehensive benchmarking of approximate bayesian inference methods for deep neural networks. *ArXiv*, abs/2007.04466, 2020a.

Meet P. Vadera, Borhan Jalaeian, and Benjamin M Marlin. Generalized bayesian posterior expectation distillation for deep neural networks. In *UAI*, 2020b.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Neural Information Processing Systems (NeurIPS)*, 2020.

Thomas J. Valone. Are Animals Capable of Bayesian Updating? An Empirical Review. *Oikos*, 112(2):252–259, 2006. ISSN 0030-1299. Publisher: Nordic Society Oikos, Wiley.

Joost van Amersfoort, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal. Improving deterministic uncertainty estimation in deep learning for classification and regression. *CoRR*, abs/2102.11409, 2021.

Joost R. van Amersfoort, L. Smith, Y. Teh, and Yarin Gal. Simple and scalable epistemic uncertainty estimation using a single deep deterministic neural network. In *International Conference on Machine Learning (ICML)*, 2020.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, koray kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional Image Generation with PixelCNN Decoders. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Jacob T. VanderPlas, Andrew J. Connolly, Zeljko Ivezic, and Alex Gray. Introduction to astroML: Machine Learning for Astrophysics. In *2012 Conference on Intelligent Data Understanding*, pages 47–54, October 2012. doi: 10.1109/CIDU.2012.6382200. arXiv:1411.5039 [astro-ph].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural*

Information Processing Systems, volume 30. Curran Associates, Inc., 2017.

Konstantin Verichev, Montserrat Zamorano, Armin Fuentes-Sepulveda, Nadia Cardenas, and Manuel Carpio. Adaptation and mitigation to climate change of envelope wall thermal insulation of residential buildings in a temperate oceanic climate. *Energy and Buildings*, 235:110719, 2021.

Susan Vineberg. Dutch Book Arguments. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2022 edition, 2022.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Stephen G Walker. Bayesian inference with misspecified models. *Journal of Statistical Planning and Inference*, 143(10):1621–1633, 2013.

Neng Wan, Dapeng Li, and Naira Hovakimyan. f-divergence variational inference. *Neural Information Processing Systems (NeurIPS)*, 2020.

Dilin Wang, Hao Liu, and Qiang Liu. Variational inference with tail-adaptive f-divergence. *Neural Information Processing Systems (NeurIPS)*, 2018.

Larry Wasserman. *All of statistics: a concise course in statistical inference*, volume 26. Springer, 2004.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. *Uncertainty in Artificial Intelligence (UAI)*, 2001.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011.

Junfeng Wen, Bo Dai, Lihong Li, and Dale Schuurmans. Batch stationary distribution estimation. *arXiv preprint arXiv:2003.00722*, 2020a.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: An alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020b.

Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. *Advances in Neural Information Processing Systems*, 33:6514–6527, 2020.

Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

Lucas Willems, Salem Lahlou, and Yoshua Bengio. Mastering rate based curriculum learning. *arXiv preprint arXiv: 2008.06456*, 2020.

Christopher Williams and Carl Rasmussen. Gaussian Processes for Regression. In *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37–52, August 1987. ISSN 0169-7439. doi: 10.1016/0169-7439(87)80084-9.

Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham M. Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. *International Conference on Learning Representations (ICLR)*, 2018.

Hao Wu, Jonas Köhler, and Frank Noé. Stochastic normalizing flows. *Neural Information Processing Systems (NeurIPS)*, 2020.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Sally Wyatt. Technological determinism is dead; long live technological determinism. *The handbook of science and technology studies*, 3:165–180, 2008.

Stanisław Węglarczyk. Kernel density estimation and its application. *ITM Web of Conferences*, 23:00037, 2018. ISSN 2271-2097. doi: 10.1051/itmconf/20182300037.

Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. {MARS}: Markov molecular sampling for multi-objective drug discovery. In *International Conference on Learning Representations*, 2021.

Aolin Xu and M. Raginsky. Minimum excess risk in bayesian learning. *ieee transactions on information theory*, 2020. doi: 10.1109/tit.2022.3176056.

I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

Mingzhang Yin and Mingyuan Zhou. Semi-implicit variational inference. *International Conference on Machine Learning (ICML)*, 2018.

Donggeun Yoo and I. Kweon. Learning loss for active learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 93–102, 2019.

Kun-Hsing Yu, Andrew L Beam, and Isaac S Kohane. Artificial intelligence in healthcare. *Nature biomedical engineering*, 2(10):719–731, 2018.

Margaux Zaffran, Aymeric Dieuleveut, Olivier F'eron, Yannig Goude, and Julie Josse. Adaptive conformal predictions for time series. In *ICML*, 2022.

Bulat Zagidullin, Jehad Aldahdooh, Shuyu Zheng, Wenyu Wang, Yinyin Wang, Joseph Saad, Alina Malyutina, Mohieddin Jafari, Ziaurrehman Tanoli, Alberto Pessia, et al. Drugcomb: an integrative cancer drug combination data portal. *Nucleic acids research*, 47(W1):W43–W51, 2019.

Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris C Holmes, Frank Hutter, and Yee Teh. Neural ensemble search for uncertainty estimation and dataset shift. *Advances in Neural Information Processing Systems*, 34:7898–7911, 2021.

Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8): 2008–2026, 2018.

Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41: 2008–2026, 2019a.

David Zhang, Corrado Rainone, Markus Peschl, and Roberto Bondesan. Robust scheduling with GFlowNets. *International Conference on Learning Representations (ICLR)*, 2023a.

Dinghuai Zhang, Ricky T. Q. Chen, Nikolay Malkin, and Yoshua Bengio. Unifying generative models with GFlowNets. *arXiv preprint 2209.02606v1*, 2022a.

Dinghuai Zhang, Nikolay Malkin, Zhen Liu, Alexandra Volokhova, Aaron Courville, and Yoshua Bengio. Generative flow networks for discrete probabilistic modeling. *International Conference on Machine Learning (ICML)*, 2022b.

Dinghuai Zhang, Ricky T. Q. Chen, Nikolay Malkin, and Yoshua Bengio. Unifying generative models with GFlowNets and beyond. *arXiv preprint 2209.02606*, 2023b.

Mingtian Zhang, Thomas Bird, Raza Habib, Tianlin Xu, and David Barber. Variational f-divergence minimization. *arXiv preprint 1907.11891*, 2019b.

Qinsheng Zhang and Yongxin Chen. Path integral sampler: a stochastic control approach for sampling. *International Conference on Learning Representations (ICLR)*, 2022.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient mcmc for bayesian deep learning. In *ICLR*, 2020.

Shuxing Zhang, Alexander Golbraikh, Scott Oloff, Harold Kohn, and Alexander Tropsha. A novel automated lazy learning qsar (all-qsar) approach: method development, applications, and virtual screening of chemical databases using validated all-qsar models. *Journal of chemical information and modeling*, 46(5):1984–1995, 2006.

Xiaojie Zhou, Lawrence Joseph, David B Wolfson, and Patrick Bélisle. A bayesian a-optimal and model robust design criterion. *Biometrics*, 59(4):1082–1088, 2003.

James V Zidek and Constance Van Eeden. Uncertainty, entropy, variance and the effect of partial information. *Lecture Notes-Monograph Series*, pages 155–167, 2003.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA,

2008.

Heiko Zimmermann, Hao Wu, Babak Esmaeili, Sam Stites, and Jan-Willem van de Meent. Nested variational inference. *Neural Information Processing Systems (NeurIPS)*, 2021.

Heiko Zimmermann, Fredrik Lindsten, Jan-Willem van de Meent, and Christian A. Naesseth. A variational perspective on generative flow networks. *arXiv preprint 2210.07992*, 2022.

Appendix A

Some mathematical concepts

A.1. Reminders about probability

Definition A.1.1 (Independence of random variables)**.** Two random variables X and Y on $\mathcal{X}$ and $\mathcal{Y}$ respectively are said to be **independent**, which is denoted by $X \perp Y$, if:

$$\forall (x, y) \in \mathcal{X} \times \mathcal{Y}, \quad p(x, y) = p(x)p(y) \tag{A.1}$$

They are said to be **conditionally independent** given a third random variable Z on $\mathcal{Z}$ if:

$$\forall (x, y, z) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z}, \quad p(x, y \mid z) = p(x \mid z)p(y \mid z) \tag{A.2}$$

Definition A.1.2 (Markov equivalence)**.** (From Murphy (2001)) Two DAGs are said to be equivalent if they entail the same set of conditional independences. For example, the graphs $X \to Y \to Z$, $X \leftarrow Y \to Z$, and $X \leftarrow Y \leftarrow Z$ are Markov equivalent since they all represent $X \perp Z \mid Y$.

A.1.1. Standard probability distributions

We describe in this section some common probability distributions used throughout the thesis:

The **Dirac** distribution, or the Dirac delta function, of parameter $\boldsymbol{\theta} \in \Theta$, is a distribution on Θ, defined for every measurable subset A of Θ by:

$$\delta_{\boldsymbol{\theta}}(A) = \mathbf{1}_A(x) = \begin{cases} 0 & \text{if } x \notin A, \\ 1 & \text{if } x \in A \end{cases} \tag{A.3}$$

The **Bernoulli** distribution, of parameter $r \in [0, 1]$ models random variables on $\{0, 1\}$. Its pmf is given by:

$$Ber(x; r) = r^x(1 - r)^{1-x} = \begin{cases} 1 - r & \text{if } x = 0 \\ r & \text{if } x = 1. \end{cases} \tag{A.4}$$

The **Categorical** distribution models random variables on a finite set of *labels* $c \in [\![1, C]\!]$. It is a generalization of the Bernoulli distribution to $C > 2$ values. It is parameterized by a vector $\boldsymbol{\theta} \in [0, 1]^C$ that is constrained to satisfy $\|\boldsymbol{\theta}\|_1 = \sum_{c=1}^{C} \boldsymbol{\theta}_c = 1$. Its pmf is given by:

$$Cat(c; \boldsymbol{\theta}) = \boldsymbol{\theta}_c \tag{A.5}$$

The univariate **Gaussian** distribution, also called **normal** distribution, models real-valued random variables. It is parameterized with its mean μ and variance σ^2 (or standard deviation σ). Its pdf is given by:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{A.6}$$

The **Beta** distribution models random variables with values in $[0, 1]$. It is determined by two parameters α, and β called the *concentration parameters*. Its pdf is given by:

$$\mathcal{B}(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{A.7}$$

where B is the Beta function defined by:

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \tag{A.8}$$

where Γ is the Gamma function defined by:

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx \tag{A.9}$$

The **multivariate Gaussian** distribution models vectors $\boldsymbol{x} \in \mathbb{R}^n$. It is parameterized by its mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and its covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ that is required to be symmetric. It admits a pdf when $\boldsymbol{\Sigma}$ is positive definite (in which case we say the distribution is non-degenerate), which is given by:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\mu)}. \tag{A.10}$$

It is common to use Gaussians with diagonal covariance matrices. An **isotropic** Gaussian is one with a spherical covariance matrix, i.e.there is some positive number σ^2 such that $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix of $\mathbb{R}^{n \times n}$.

To create more complex probability distributions and to model more interesting phenomena (e.g., those with multiple modes), it is common to use a **mixture model**. Such a model requires K base distributions $p_1, \ldots, p_K$, along with mixture weights represented in a vector $\boldsymbol{\pi} \in [0, 1]^K$ satisfying $\|\boldsymbol{\pi}\|_1 = 1$. The resulting density is given by:

$$p(x; \boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K) = \sum_{k=1}^{K} \boldsymbol{\pi}_k p_k(x; \boldsymbol{\theta}_k). \tag{A.11}$$

A.2. Gaussian processes

Definition A.2.1. (Gaussian processes) A **Gaussian process (GP)** is a stochastic process, i.e.a collection of random variables, indexed by a continuous domain $\mathcal{X}$, any finite number of which have a joint Gaussian distribution. It is parameterized by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a positive definite kernel $\kappa : \mathcal{X}^2 \to \mathbb{R}$, i.e.one that satisfies:

$$\forall n \geq 1,\ \forall x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathcal{X}, \forall c_1, \ldots, c_n \in \mathbb{R},\ \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \kappa(x_i, x_j) \geq 0. \tag{A.12}$$

Samples from a Gaussian process $GP(m, \kappa)$ are essentially functions from $\mathcal{X}$ to $\mathbb{R}$, and for any finite set of points $X = (x_1, \ldots, x_n)$, we have:

$$p(f(x_1), \ldots, f(x_n) \mid X) = \mathcal{N}(f(x_1), \ldots, f(x_n); \mu_X, \kappa^{X,X}), \tag{A.13}$$

where $\mu_X = (m(x_1), \ldots, m(x_n))$, and $\kappa_{i,j}^{X,X} = \kappa(x_i, x_j)$.

It should be noted that a sum or a product of positive definite kernels is positive definite, and other transformations or combinations are possible (Schiilkopf, 2001). The most commonly used kernels are transformations and combinations of these basic kernels:

— Constant Kernel: $k(x, x') = C$, where C is a constant value. It is often used as part of a product kernel, where it scales the magnitude of the other kernel, or as a part of a sum kernel, where it modifies the mean of the GP.

— RBF kernel (squared exponential): $k(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\theta^2}\right)$. It is parameterized by its length-scale θ that defines the extent to which the value of the function at a particular point influences other points. Some variants consider an anisotropic variant of the kernel, where the length-scale θ is a vector of the same dimension as x: $k(x, x') = \exp\left(-\frac{1}{2}\sum_{i=1}^{d} \frac{|x_i - x_i'|^2}{\theta_i^2}\right)$.

— Matérn kernel: it is a generalization of the RBF kernel, but with an additional parameter that controls the smoothness of the resulting function. Gaussian processes with such kernels are popular choices for learning functions that are not infinitely differentiable, as is the case for the RBF kernel. The two most famous ones are Matern-3/2 and Matern-5/2, which model functions differentiable at least once and twice, respectively (Banerjee and Gelfand, 2003).

— Matern-3/2: $k(x, x') = \left(1 + \sqrt{3}\frac{\|x_1 - x_2\|}{\theta}\right) \exp\left(-\sqrt{3}\frac{\|x_1 - x_2\|^2}{2\theta^2}\right)$

— Matern-5/2: $k(x, x') = \left(1 + \sqrt{5}\frac{\|x_1 - x_2\|}{\theta} + \frac{5}{3}\frac{\|x_1 - x_2\|^2}{\theta^2}\right) \exp\left(-\sqrt{5}\frac{\|x_1 - x_2\|}{\theta}\right)$

Gaussian processes are a popular tool for regression when the likelihood is Gaussian, given that all computations can be performed in closed form (Williams and Rasmussen, 1995). The kernel's parameters, or the kernel's choice, are hyperparameters that can be optimized using a gradient-based optimizer on the marginal likelihood of the training set.

Regression with Gaussian likelihood in the noise-free setting. Assume we have access to a labelled dataset $\mathcal{D}$ as in (2.4), where $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \mathcal{R}$, and denote by $\boldsymbol{X}$ the $N \times d$ matrix representing the training inputs, and denote by $\boldsymbol{y}$ the vector in $\mathcal{R}^N$ describing the training set labels.

Given a test set $\boldsymbol{X}'$, a matrix of size $N' \times d$, we want to infer $\boldsymbol{y}' \in \mathcal{R}^{N'}$.

The joint distribution of the joint vector $\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{y}' \end{pmatrix} \in \mathbb{R}^{N+N'}$ conditioned on $\boldsymbol{X}, \boldsymbol{X}'$ is:

$$\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{y}' \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_{\boldsymbol{X}} \\ \mu_{\boldsymbol{X}'} \end{pmatrix}, \begin{pmatrix} \kappa^{\boldsymbol{X},\boldsymbol{X}} & \kappa^{\boldsymbol{X},\boldsymbol{X}'} \\ \kappa^{\boldsymbol{X}',\boldsymbol{X}} & \kappa^{\boldsymbol{X}',\boldsymbol{X}'} \end{pmatrix} \right). \tag{A.14}$$

Obtaining the posterior $p(\boldsymbol{y}' \mid \boldsymbol{X}', \mathcal{D})$ amounts to performing Gaussian conditioning, a proof of which is given in Eaton (1983), in (A.14):

$$p(\boldsymbol{y}' \mid \boldsymbol{X}', \mathcal{D}) = p(\boldsymbol{y}' \mid \boldsymbol{X}', \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{y}'; \mu_{\boldsymbol{X}'|\boldsymbol{X}}, \Sigma_{\boldsymbol{X}'|\boldsymbol{X}}), \tag{A.15}$$

where

$$\mu_{\boldsymbol{X}'|\boldsymbol{X}} = \mu_{\boldsymbol{X}'} + \kappa^{\boldsymbol{X}',\boldsymbol{X}} \left(\kappa^{\boldsymbol{X},\boldsymbol{X}}\right)^{-1} (\boldsymbol{y} - \mu_{\boldsymbol{X}}) \tag{A.16}$$

$$\Sigma_{\boldsymbol{X}'|\boldsymbol{X}} = \kappa^{\boldsymbol{X}',\boldsymbol{X}'} - \kappa^{\boldsymbol{X}',\boldsymbol{X}} \left(\kappa^{\boldsymbol{X},\boldsymbol{X}}\right)^{-1} \kappa^{\boldsymbol{X},\boldsymbol{X}'} \tag{A.17}$$

A.3. Kernel density estimation

A **density kernel** is a function $\kappa : \mathbb{R} \to \mathbb{R}^+$ such that $\int \kappa(x)dx = 1$ and $\forall x \in \mathbb{R},\ \kappa(-x) = \kappa(x)$. An example is the Gaussian kernel:

$$\kappa_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}. \tag{A.18}$$

From a kernel κ, we can define the kernel with bandwidth $h > 0$:

$$\kappa_h(x) := \frac{1}{h} \kappa\left(\frac{x}{h}\right). \tag{A.19}$$

Given a dataset $\mathcal{D} = \{x_i\}_{i \in [\![1,N]\!]}$, we can define a non-parametric density estimate of the form:

$$p(x \mid \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \kappa_h(x - x_i). \tag{A.20}$$

A.4. On MLE, ERM, and MAP

Lemma A.4.1. *The ERM estimate associated with the squared error (2.9) corresponds to the MLE estimate, assuming a Gaussian posterior $p(y \mid \boldsymbol{x})$, with fixed variance.*

PROOF.

$$\boldsymbol{\theta}_{\text{ERM}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \hat{R}(f_{\boldsymbol{\theta}}) \tag{A.21}$$

$$= \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2. \tag{A.22}$$

Under the assumption of a Gaussian $p(y \mid \boldsymbol{x})$ with fixed variance σ^2, i.e.$p(y \mid \boldsymbol{x}) = \mathcal{N}(y; f_{\boldsymbol{\theta}}(\boldsymbol{x}), \sigma^2)$ we have:

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) \propto e^{-\frac{1}{2\sigma^2}(y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))^2}. \tag{A.23}$$

Hence,

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}) \tag{A.24}$$

$$= \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} -\frac{1}{2\sigma^2}(y - f_{\boldsymbol{\theta}}(\boldsymbol{x}))^2 \tag{A.25}$$

$$= \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \tag{A.26}$$

$$= \boldsymbol{\theta}_{\text{ERM}} \tag{A.27}$$

□

Lemma A.4.2. *The ERM estimate associated with the log loss (2.21) corresponds to the MLE estimate, assuming a Categorical posterior $p(y \mid \boldsymbol{x})$.*

PROOF. In the discriminative approach to classification, $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a probability distribution over $\mathcal{Y} = [\![1, C]\!]$, which can also be seen as a vector of $[0, 1]^C$

$$\boldsymbol{\theta}_{\text{ERM}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \hat{R}(f_{\boldsymbol{\theta}})$$

$$= \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} -\log f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)_y.$$

Under the assumption of a Categorical $p(y \mid \boldsymbol{x})$, i.e.$p(y \mid \boldsymbol{x}) = Cat(y; f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ we have:

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\boldsymbol{x})_y.$$

Hence,

$$\begin{aligned}
\boldsymbol{\theta}_{\mathrm{MLE}} &= \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} \log f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)_y \\
&= \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{N} -\log f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)_y \\
&= \boldsymbol{\theta}_{\mathrm{ERM}}
\end{aligned}$$

□

Appendix B

---

torchgfn: A PyTorch GFlowNet library

This chapter is based on the following paper:

— Lahlou et al. (2023b): "`torchgfn`: A PyTorch GFlowNet library" - Salem Lahlou, Joseph D. Viviano, Victor Schmidt, Yoshua Bengio, available as a preprint, and submitted to the Journal of Machine Learning Research (JMLR).

The increasing popularity of generative flow networks (GFlowNets or GFNs) is accompanied by a proliferation of code sources. This hinders the implementation of new features, such as training losses, that can readily be compared to existing ones on a set of common environments. In addition to slowing down research in the field of GFlowNets, different code bases use different conventions that might confuse newcomers. `torchgfn` is a library built on top of PyTorch that aims at addressing both problems. It provides users with a simple API for environments and useful abstractions for samplers and losses. Multiple examples are provided, replicating published results. The code is available in https://github.com/saleml/torchgfn.

The library aims to accompany researchers and engineers in learning about GFlowNets, and in developing new algorithms.

Currently, the library is shipped with three environments: two discrete environments (Discrete Energy Based Model and Hyper Grid) and a continuous box environment. The library is designed to allow users to define their own environments.

B.1. Installing the package

The codebase requires Python 3.10 or higher. To install the latest stable version:

```
pip install torchgfn
```

Optionally, to be able to run the attached scripts:

```
pip install torchgfn[scripts]
```

To install the cutting edge version (from the `main` branch of the code repository):

```
git clone https://github.com/saleml/torchgfn.git
conda create -n gfn python=3.11
conda activate gfn
cd torchgfn
pip install.
```

B.2. Standalone example

This example, which shows how to use the library for a simple discrete environment, requires the `tqdm`[1] package to run. The users need to install it via `pip install tqdm` or install all extra requirements with `pip install .[scripts]` or `pip install torchgfn[scripts]`.

1. https://github.com/tqdm/tqdm.

```
import torch
from torch.optim import Adam
from tqdm import tqdm

from gfn.gflownet import TBGFlowNet  # We use the Trajectory Balance (TB) loss
from gfn.gym import HyperGrid  # We use the hyper grid environment
from gfn.modules import DiscretePolicyEstimator
from gfn.samplers import Sampler
from gfn.utils import NeuralNet  # NeuralNet is a simple MLP

env = HyperGrid(ndim=4, height=8, R0=0.01) # Grid of size 8x8x8x8

# The environment has a preprocessor attribute,
# which is used to preprocess the state before feeding it to the policy estimator
module_PF = NeuralNet(
    input_dim=env.preprocessor.output_dim,
    output_dim=env.n_actions
)  # Neural network for the forward policy, with n_actions outputs
module_PB = NeuralNet(
    input_dim=env.preprocessor.output_dim,
    output_dim=env.n_actions - 1,
    torso=module_PF.torso  # We share all the parameters of P_F and P_B, except for the last layer
)

pf_estimator = DiscretePolicyEstimator(module_PF, env.n_actions, is_backward=False,
                                       preprocessor=env.preprocessor)
pb_estimator = DiscretePolicyEstimator(module_PB, env.n_actions, is_backward=True,
                                       preprocessor=env.preprocessor)

gfn = TBGFlowNet(init_logZ=0., pf=pf_estimator, pb=pb_estimator)

sampler = Sampler(estimator=pf_estimator)

 # Policy parameters have their own LR.
non_logz_params = [v for k, v in dict(gfn.named_parameters()).items() if k != "logZ"]
optimizer = torch.optim.Adam(non_logz_params, lr=1e-3)

# LogZ gets a dedicated learning rate (typically higher).
logz_params = [dict(gfn.named_parameters())["logZ"]]
optimizer.add_param_group({"params": logz_params, "lr": 1e-1})

for i in (pbar := tqdm(range(1000))):
    trajectories = sampler.sample_trajectories(env=env, n_trajectories=16)
    optimizer.zero_grad()
    loss = gfn.loss(env, trajectories)
    loss.backward()
    optimizer.step()
    if i % 25 == 0:
        pbar.set_postfix({"loss": loss.item()})
```

B.3. Details about the code base

B.3.1. Defining an environment

To define an environment, the user needs to define the tensor `s0` representing the initial state s_0, from which the `state_shape` attribute is inferred, and optionally a tensor representing the sink state s_f, which is only used for padding incomplete trajectories. If not specified, `sf` is set to a tensor of the same shape as `s0` filled with $-\infty$.

If the environment is discrete, in which case it is an instance of `DiscreteEnv`, the total number of actions should be specified as an attribute.

Suppose the states (as represented in the `States` class) need to be transformed to another format before being processed (by neural networks, for example). In that case, the environment should define a `preprocessor` attribute, which should be an instance of the base preprocessor class. If no preprocessor is specified, the states are used as is (actually transformed using the `IdentityPreprocessor`, which converts the state tensors to `FloatTensor`s). Implementing a specific preprocessor requires defining the `preprocess` function and the `output_shape` attribute, which is a tuple representing the shape of one preprocessed state.

The user needs to implement the following two abstract functions:

— The method `make_States_class`, that creates the corresponding subclass of `States`. For discrete environments, the resulting class should be a subclass of `DiscreteStates` that implements the `update_masks` method specifying which actions are available at each state.

— The method `make_Actions_class`, that creates a subclass of `Actions` simply by specifying the required class variables (the shape of an action tensor, the dummy action, and the exit action). This method is implemented by default for all `DiscreteEnv`s.

The logic of the environment is handled by the methods `maskless_step` and `maskless_backward_step`, which need to be implemented, and specify how an action changes a state (going forward and backward). These functions do not need to handle masking for discrete environments, checking whether actions are allowed, checking whether a state is the sink state, etc... These checks are handled in `Env.step` and `Env.backward_step` functions that do not need to be implemented. Non-discrete environments need to implement the `is_action_valid` function taking a batch of states and actions and returning `True` only if all actions can be taken at the given states.

The environment needs to implement the `log_reward` function, that assigns the logarithm of a non-negative reward to every terminating state (i.e., a state with only s_f as a child in the DAG). If `log_reward` is not implemented, `reward` needs to be.

For `DiscreteEnv`s, the user can define a `get_states_indices` method that assigns a unique integer number to each state, and a `n_states` property that returns an integer representing the number of states (excluding s_f) in the environment. The function `get_terminating_states_indices` can also be implemented and serves the purpose of uniquely identifying terminating states of the environment, which is helpful for tabular `GFNModules`. Other properties and functions can also be implemented, such as the `log_partition` or the `true_dist_pmf` properties.

For reference, it might be useful to look at one of the following provided environments:

— `Hypergrid` is an example of a discrete environment where all states are terminating states.

— `DiscreteEBM` is an example of a discrete environment where all trajectories are of the same length, but only some states are terminating.

— `Box` is an example of a continuous environment with a specific `is_action_valid` function.

B.3.2. States

States are the primitive building blocks for GFlowNet objects, such as transitions and trajectories, on which losses operate.

An abstract `States` class is provided. But a `States` subclass is needed for each environment. A `States` object is a collection of multiple states (nodes of the DAG). A tensor representation of the states is required for batching. If a state is represented with a tensor of shape `(*state_shape)`, a batch of states is represented with a `States` object, with the attribute `tensor` of shape `(*batch_shape, *state_shape)`. Other representations are possible (e.g., a state as a string, a `numpy` array, a graph, etc...). Still, these representations cannot be batched unless the user specifies a function that transforms these raw states into tensors.

The `batch_shape` attribute is required to keep track of the batch dimension. A trajectory can be represented by a States object with `batch_shape = (n_states,)`. Multiple trajectories can be represented by a States object with `batch_shape = (n_states, n_trajectories)`.

Because multiple trajectories can have different lengths, batching requires appending a dummy tensor to trajectories shorter than the longest trajectory. The dummy state is the `sf` attribute of the environment (e.g., $[-1, \ldots, -1]$, or $[-\infty, \ldots, -\infty]$, etc...). Which is never processed and is used to pad the batch of states only.

For discrete environments, the action set is represented with the set $\{0, \ldots, n_{actions} - 1\}$, where the $(n_{actions})$-th action always corresponds to the exit or terminate action, i.e., that results in a transition of the type $s \to s_f$, but not all actions are possible at all

states. For discrete environments, each `States` object is endowed with two extra attributes: `forward_masks` and `backward_masks`, representing which actions are allowed at each state and which actions could have led to each state, respectively. Such states are instances of the `DiscreteStates` abstract subclass of `States`. The `forward_masks` tensor is of shape `(*batch_shape, n_actions)`, and `backward_masks` is of shape `(*batch_shape, n_actions - 1)`. Each subclass of `DiscreteStates` needs to implement the `update_masks` function that uses the environment's logic to define the two tensors.

B.3.3. Actions

Actions should be thought of as internal actions of an agent building a compositional object. They correspond to transitions $s \to s'$. An abstract `Actions` class is provided. It is automatically subclassed for discrete environments but needs to be manually subclassed otherwise.

Similar to `States` objects, each action is a tensor of shape `(*batch_shape, *action_shape)`. For discrete environments for instances, `action_shape = (1,)`, representing an integer between 0 and $n_{actions} - 1$.

Additionally, each subclass needs to define two more class variable tensors:

— `dummy_action`: A tensor padded to sequences of actions in the shorter trajectories of a batch of trajectories. It is $[-1]$ for discrete environments.

— `exit_action`: A tensor corresponding to the termination action. It is $[n_{actions} - 1]$ of discrete environments.

B.3.4. Containers

Containers are collections of `States`, along with other information, such as reward values or densities $p(s' \mid s)$. Two containers are available:

— Transitions, representing a batch of transitions $s \to s'$.

— Trajectories, representing a batch of complete trajectories $\tau = s_0 \to s_1 \to \cdots \to s_n \to s_f$.

These containers can either be instantiated using a `States` object or can be initialized as empty containers that can be populated on the fly, allowing the usage of the `ReplayBuffer` class.

They inherit from the base `Container` class, indicating some helpful methods.

In most cases, one needs to sample complete trajectories. From a batch of trajectories, a batch of states and a batch of transitions can be defined using `Trajectories.to_transitions()` and `Trajectories.to_states()`, in order to train GFlowNets with losses that are edge-decomposable or state-decomposable. These exclude

meaningless transitions and dummy states that were added to the batch of trajectories to allow for efficient batching.

B.3.5. Modules

Training GFlowNets requires one or multiple estimators, called `GFNModule`s. `GFNModule` is an abstract subclass of `torch.nn.Module`. In addition to the usual `forward` function, `GFNModule`s need to implement a `required_output_dim` attribute to ensure that the outputs have the required dimension for the task at hand; and some of them need to implement a `to_probability_distribution` function.

— `DiscretePolicyEstimator` is a `GFNModule` that defines the policies $P_F(. \mid s)$ and $P_B(. \mid s)$ for discrete environments. When `is_backward=False`, the required output dimension is `n = env.n_actions`, and when `is_backward=True`, it is `n = env.n_actions - 1`. These n numbers represent the logits of a Categorical distribution. The corresponding `to_probability_distribution` function transforms the logits by masking illegal actions (according to the forward or backward masks), then return a Categorical distribution. The masking is done by setting the corresponding logit to $-\infty$. The function also includes exploration parameters, in order to define a tempered version of P_F, or a mixture of P_F with a uniform distribution. `DiscretePolicyEstimator` with `is_backward=False` can be used to represent log-edge-flow estimators $\log F(s \to s')$.

— `ScalarModule` is a simple module with required output dimension 1. It is useful to define log-state flows $\log F(s)$.

For non-discrete environments, the user needs to specify their own policies P_F and P_B. The module, taking as input a batch of states (as a `States`) object, should return the batched parameters of a `torch.Distribution`. The distribution depends on the environment. The `to_probability_distribution` function handles the conversion of the parameter outputs to an actual batched `Distribution` object that implements at least the `sample` and `log_prob` functions. An example is provided for the box environment in which the forward policy has support either on a quarter disk or an arc-circle, such that the angle and the radius (for the quarter disk part) are scaled samples from a mixture of Beta distributions. The provided example shows an intricate scenario, and user-defined environments are not expected to need this much detail.

In all `GFNModule`s, note that the input of the `forward` function is a `States` object, which means that they first need to be transformed into tensors. However, `states.tensor` does not necessarily include the structure a neural network can use to generalize. It is common in these scenarios to have a function that transforms these raw tensor states to ones where the structure is more apparent via a `preprocessor` object that is part of the environment

Appendix B.3.1. The default preprocessor of an environment is the identity preprocessor. The `forward` pass thus first calls the `preprocessor` attribute of the environment on `States` before performing any transformation. The `preprocessor` is thus an attribute of the module. If it is not explicitly defined, it is set to the identity preprocessor.

For discrete environments, a `Tabular` module is provided, where a lookup table is used instead of a neural network. A `UniformPB` module is also provided, implementing a uniform backward policy.

B.3.6. Samplers

`Sampler` objects define how actions are sampled at each state. They require a `GFNModule` that implements the `to_probability_distribution` function.

They include a `sample_trajectories` method that samples a batch of trajectories starting from a given set of initial states or s_0.

For off-policy sampling, the parameters of `to_probability_distribution` can be directly passed when initializing the `Sampler`.

B.3.7. Losses

GFlowNets can be trained with different losses, each requiring a different parametrization, which we call in this library a `GFlowNet`. A `GFlowNet` is a `GFNModule` that includes one or multiple `GFNModules`, at least one of which implements a `to_probability_distribution` function. They need to implement a `loss` function that takes either states, transitions, or trajectories as input, depending on the loss.

Currently, the implemented losses are:

— Flow Matching, introduced in Bengio et al. (2021).

— Detailed Balance, introduced in Chapter 3, and its modified variant, introduced in Deleu et al. (2022).

— Trajectory Balance, introduced in Malkin et al. (2022).

— Sub-Trajectory Balance. By default, each sub-trajectory is weighted geometrically (within the trajectory) depending on its length. This corresponds to the strategy defined in Madan et al. (2022a). Other strategies exist and are implemented in the corresponding class.

— Log Partition Variance loss, introduced in Zhang et al. (2023a).

B.4. Provided scripts

Example scripts and notebooks for the three environments shipped with the library are provided. For the `Hypergrid` and the `Box` environments, the provided scripts are supposed to reproduce published results in Malkin et al. (2022), Chapter 3, and Chapter 4.

Appendix C

On Bayesian Optimal Experiment Design

The Bayesian Optimal Experiment Design (BOED) field has historical roots dating back to at least 1964 with pioneering contributions by Hicks (1964); Klepikov and Sokolov (1964); Nalimov and Chernova (1968). Fedorov (1972) provided a defining perspective, describing experimental design as a "broad class of methods which would give not only the means of reduction of experimental data but also would permit the organization of the experiment in an optimal manner". For Ryan et al. (2016), "statistical experimental design provides rules for the allocation of resources in an information gathering exercise in which there is variability that is not under control of the experimenter".

Following the setting of Müller (2005), Ryan et al. (2016) defines the design criterion as a function $U(\boldsymbol{d}, \boldsymbol{\theta}, \boldsymbol{y})$ that describes the worth of choosing the design $\boldsymbol{d}$ from the design space D yielding observable $\boldsymbol{y}$, with model parameter values $\boldsymbol{\theta}$. This function requires a probabilistic model $p(\boldsymbol{\theta}, \boldsymbol{y} \mid \boldsymbol{d})$, consisting of a likelihood $p(\boldsymbol{y} \mid \boldsymbol{d}, \boldsymbol{\theta})$ and a prior distribution $p(\boldsymbol{\theta})$. The experimenter's goal in Bayesian experimental design is to find the optimal design $\boldsymbol{d}^*$ that maximizes an expected utility. Several utility functions U have been proposed in the literature (Ryan et al., 2016), and different utility functions lead to different optimal designs. A principled choice for this expected utility function is the mutual information between parameters $\boldsymbol{\theta}$ and the observables $\boldsymbol{y}$ at design $\boldsymbol{d}$, more commonly referred to as the Expected Information Gain (EIG):

$$EIG(\boldsymbol{d}) := \mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{d})}[\mathbb{H}(p(\boldsymbol{\theta})) - \mathbb{H}(p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{d}))] \tag{C.1}$$

This objective is suitable for sequential experimental design, where the prior $p(\boldsymbol{\theta})$ is updated at each step and is replaced in (C.1) with $p(\boldsymbol{\theta} \mid \mathcal{D}_{k-1})$, where $\mathcal{D}_{k-1} := \{\boldsymbol{d}^*_{1:k-1}, \boldsymbol{y}^*_{1:k-1}\}$ is the historical dataset at step $k-1$.

While maximizing the EIG is intractable, most modern approaches rely on approximate optimization. Foster et al. (2019, 2020); Kleinegesse and Gutmann (2019, 2021) propose

using different variational lower bounds on the EIG. Unfortunately, the sequential experimental design process requires significant computations at each step to update the posterior and optimize an estimator of a lower bound on the EIG. The sequential approach is tackled differently in Foster et al. (2021), where the concept of *design function* or *policy* π is introduced, mapping the set of all previous design-observation pairs $\mathcal{D}$ to the next optimal design. The authors propose to *amortize* the cost of sequential experiment design, "performing upfront training before the start of the experiment to allow very fast design decisions at deployment when time is at a premium", which is useful when deploying the same adaptive sequential experimental design framework numerous times. The standard myopic BOED approaches correspond to a specific choice of the (implicit) policy π that is myopically optimal. The proposed approach, called Deep Adaptive Design (DAD), eliminates the need for estimating posterior distributions or intermediate EIGs, while also allowing to learn non-myopic policies. It considers the total expected information gain of a policy π over a sequence of T experiments:

$$EIG_T(\pi) := \mathbb{E}_{p(\boldsymbol{\theta})p(\mathcal{D}_T|\boldsymbol{\theta},\pi)}\left[\sum_{t=1}^{T} EIG_{\mathcal{D}_{t-1}}(\boldsymbol{d}_t)\right], \tag{C.2}$$

which is shown to be equal to

$$EIG_T(\pi) = \mathbb{E}_{p(\boldsymbol{\theta})p(\mathcal{D}_T|\boldsymbol{\theta},\pi)}\left[\log p(\mathcal{D}_T \mid \boldsymbol{\theta}, \pi) - \log p(\mathcal{D}_T \mid \pi)\right], \tag{C.3}$$

where, as before, $\mathcal{D}_t$ denotes the historical dataset of design-observation pairs up to step t, $EIG_{\mathcal{D}}$ is the EIG of Eq. C.1 with the prior $p(\boldsymbol{\theta})$ replaced with the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$, $p(\mathcal{D}_T \mid \boldsymbol{\theta}, \pi) := \prod_{t=1}^{T} p(\boldsymbol{y}_t \mid \boldsymbol{d}_t, \boldsymbol{\theta})$, and $p(\mathcal{D}_T \mid \pi) := \mathbb{E}_{p(\boldsymbol{\theta})}[p(\mathcal{D}_T \mid \boldsymbol{\theta}, \boldsymbol{\pi})]$. Ivanova et al. (2021); Lim et al. (2022) extend the DAD approach and bypass its need for explicit models.

Appendix D

Appendix for Chapter 3

D.1. Conditional GFlowNets for entropy and mutual information estimation

Definition D.1.1 (Entropic reward function)**.** Given a reward function R with $0 \leq R(s) < 1 \ \forall s$, we define the **entropic reward function** R' associated with R as:

$$R'(s) = -R(s) \log R(s). \tag{D.1}$$

In this section, we show that we can estimate entropies by training two GFlowNets: one that estimates flows as usual for a target terminal reward function $R(s)$, and one that estimates flows for the corresponding entropic reward function. We show below that we obtain an estimator of entropy by looking up the flow in the initial state, and if we do this exercise with conditional flows, we get conditional entropy. Once we have the conditional entropy, we can also estimate the mutual information.

Proposition D.1.2. *Consider a flow network (G, F) such that the terminating flows match a given reward function R, i.e., $\forall s \in \mathcal{S}^f$, $F(s \to s_f) = R(s)$, with $R(s) < 1$ for all s, and a second flow network (G, F') with the same pointed DAG, but with a flow function for which the terminating flows match the entropic reward function R' ((D.1)), then the entropy $H[S]$ associated with the terminating state random variable $S \in \mathcal{S}^f$ with distribution $P_\top(S = s) = \frac{R(s)}{Z}$ ((3.17)) is*

$$H[S] := -\sum_s P_\top(s) \log P_\top(s) = \frac{F'(s_0)}{F(s_0)} + \log F(s_0). \tag{D.2}$$

As a consequence of Proposition D.1.2, if we are given a set $\mathcal{X}$ of conditioning variables, consider a conditional flow network defined by a conditional flow function F, for which the terminating flows match a target reward R family (conditioned on $x \in \mathcal{X}$) that satisfies

$R_x(s) < 1$ for all s, and a second conditional flow network defined by a conditional flow function F', for which the terminating flows match the entropic reward functions R'_x ((D.1)), then the conditional entropy $H[S \mid x]$ of random terminating states $S \in \mathcal{S}^f$ consistent with condition x is given by

$$H[S \mid x] = \frac{F'(s_0 \mid x)}{F(s_0 \mid x)} + \log F(s_0 \mid x). \tag{D.3}$$

In particular, for a state-conditional GFlowNet ($\mathcal{X} = \mathcal{S}$ is the state space of the DAG), we obtain

$$H[S \mid s] = \frac{F'(s \mid s)}{F(s \mid s)} + \log F(s \mid s). \tag{D.4}$$

More generally, the mutual information $\mathrm{MI}(S; X)$ between the random draw of a terminating state $S = s$ according to $P_\top(s \mid x)$ and the conditioning random variable X is

$$\mathrm{MI}(S; X) = H[S] - E_X[H[S \mid X]] = \frac{F'(s_0)}{F(s_0)} + \log F(s_0) - E_X\left[\frac{F'(s_0 \mid X)}{F(s_0 \mid X)} + \log F(s_0 \mid X)\right] \tag{D.5}$$

where $F(s)$ and $F'(s)$ indicate the unconditional flows (trained with no condition x given) while $F(s \mid x)$ and $F'(s \mid x)$ are their conditioned counterparts.

If we have a sampling mechanism for $P(X)$, we can thus approximate the expectation in (D.5) by a Monte-Carlo average with draws from $P(X)$.

D.2. Proofs

Lemma 3.2.5.

Proof. For convenience, we will use $\mathcal{T}_{s \to s', s_f}$ to denote the set of trajectories starting with $s \to s'$ and ending in s_f, and $\mathcal{T}_{0, s \to s'}$ to denote the set of trajectories starting in s_0 and ending with $s \to s'$. This allows us to write:

$$\forall s \neq s_f \quad \mathcal{T}_{s,f} = \bigcup_{s' \in Child(s)} \mathcal{T}_{s \to s', s_f}, \quad \{\mathcal{T}_{s \to s', s_f},\ s' \in Child(s)\} \text{ pairwise disjoint}, \tag{D.6}$$

$$\forall s' \neq s_0 \quad \mathcal{T}_{0,s'} = \bigcup_{s \in Par(s')} \mathcal{T}_{0, s \to s'}, \quad \{\mathcal{T}_{0, s \to s'},\ s \in Par(s')\} \text{ pairwise disjoint}. \tag{D.7}$$

Additionally, for any $s \neq s_f$, we denote by $d_{s,f}$ the maximum trajectory length in $\mathcal{T}_{s,f}$; and for any $s' \neq s_0$, we denote by $d_{0,s'}$ the maximum trajectory length in $\mathcal{T}_{0,s}$.

We will prove (3.4) by strong induction on $d_{s,f}$ and (3.5) by strong induction on $d_{0,s'}$.

Base cases: If $d_{s,f} = 1$ and $d_{0,s'} = 1$, then $\mathcal{T}_{s,f} = \{(s \to s_f)\}$ and $\mathcal{T}_{0,s'} = \{(s_0 \to s')\}$. Hence, $\sum_{\tau \in \mathcal{T}_{s,f}} \hat{P}_F(\tau) = \hat{P}_F(s \to s_f) = \hat{P}_F(s_f \mid s) = 1$ given that s_f is the only child of s (otherwise $d_{s,f}$ cannot be 1), and $\sum_{\tau \in \mathcal{T}_{0,s'}} \hat{P}_B(\tau) = \hat{P}_B(s_0 \mid s') = 1$ given that s_0 is the only parent of s' (otherwise $d_{0,s'}$ cannot be 1).

Induction steps: Consider $s \neq s_f$ such that $d_{s,f} > 1$ and $s' \neq s_0$ such that $d_{0,s'} > 1$. Because of the disjoint unions written above, we have:

$$\sum_{\tau \in \mathcal{T}_{s,f}} \hat{P}_F(\tau) = \sum_{\tilde{s} \in Child(s)} \sum_{\tau \in \mathcal{T}_{s \to \tilde{s},f}} \hat{P}_F(\tau) = \sum_{\tilde{s} \in Child(s)} \hat{P}_F(\tilde{s} \mid s) \sum_{\tau \in \mathcal{T}_{\tilde{s},f}} \hat{P}_F(\tau) = 1, \tag{D.8}$$

$$\sum_{\tau \in \mathcal{T}_{0,s'}} \hat{P}_B(\tau) = \sum_{\tilde{s}' \in Par(s')} \sum_{\tau \in \mathcal{T}_{0,\tilde{s}' \to s}} \hat{P}_B(\tau) = \sum_{\tilde{s}' \in Par(s')} \hat{P}_B(\tilde{s}' \mid s') \sum_{\tau \in \mathcal{T}_{0,\tilde{s}'}} \hat{P}_B(\tau) = 1, \tag{D.9}$$

where we used the induction hypotheses in the third equality of each line. □

Proposition 3.2.8.

Proof. Given $s \neq s_f$, the set of complete trajectories going through s is the (disjoint) union of the sets of trajectories going through $s \to s'$, for all $s' \in Child(s)$:

$$\{\tau \in \mathcal{T} : s \in \tau\} = \bigcup_{s' \in Child(s)} \{\tau \in \mathcal{T} : s \to s' \in \tau\}. \tag{D.10}$$

Therefore, it follows that:

$$F(s) = \sum_{\tau \,:\, s \in \tau} F(\tau) = \sum_{s' \in Child(s)} \sum_{\tau \,:\, s \to s' \in \tau} F(\tau) = \sum_{s' \in Child(s)} F(s \to s') \tag{D.11}$$

Similarly, (3.10) follows by writing the set of complete trajectories going through $s' \neq s_0$ as the (disjoint) union of the sets of trajectories going through $s \to s'$ for all $s \in Par(s')$. □

Proposition 3.2.10.

Proof. Since $\forall \tau \in \mathcal{T},\ s_0, s_f \in \tau$, applying (3.7) to s_0 and s_f yields

$$F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z, \tag{D.12}$$

$$F(s_f) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z. \tag{D.13}$$

□

Proposition 3.2.14.

Proof. Since the flow $F(s \to s_f)$ is non-negative, it is easy to see that $P_T(s) \geq 0$. Moreover, using the definition of $\mathcal{S}^f = Par(s_f)$, Proposition 3.2.8 (relating the edge flows and the state flows), and Proposition 3.2.10 ($F(s_f) = Z$), we have

$$\sum_{s \in \mathcal{S}^f} P_T(s) = \frac{1}{Z} \sum_{s \in \mathcal{S}^f} F(s \to s_f) = \frac{1}{Z} \sum_{s \in Par(s_f)} F(s \to s_f) = \frac{F(s_f)}{Z} = 1. \tag{D.14}$$

□

Proposition 3.2.16.

PROOF. Recall from Lemma 3.2.5 the notations $\mathcal{T}_{0,s}$ to denote the set of partial trajectories from s_0 to s, and $\mathcal{T}_{s',f}$ to denote the set of partial trajectories from s' to s_f. We will prove the equivalences $1 \Leftrightarrow 2$ and $1 \Leftrightarrow 3$.

— $1 \Rightarrow 2$: Suppose that F is a Markovian flow. Then using the laws of probability, the Markov property in (3.19), and $P(s_0) = 1$, for some complete trajectory $\tau = (s_0, \dots, s_{n+1} = s_f)$:

$$P(\tau) = P(s_0 \to s_1 \to \dots \to s_{n+1}) = P(s_0 \to s_1) \prod_{t=1}^{n} P(s_t \to s_{t+1} \mid s_0 \to \dots \to s_t) \tag{D.15}$$

$$= P(s_0 \to s_1) \prod_{t=1}^{n} P(s_t \to s_{t+1} \mid s_t) \tag{D.16}$$

$$= P(s_0) P_F(s_1 \mid s_0) \prod_{t=1}^{n} P_F(s_{t+1} \mid s_t) \tag{D.17}$$

$$= \prod_{t=1}^{n+1} P_F(s_t \mid s_{t-1}), \tag{D.18}$$

where the second line uses to Markov property, and the third line uses the definition of the forward transition probability P_F. P_F thus satisfies (3.20) for all complete trajectories.

To show the uniqueness of P_F, assume (3.20) is satisfied by some $\hat{P}_F$ for all complete trajectories. By definition of the forward transition probability:

$$P_F(s' \mid s) := P(s \to s' \mid s) = \frac{P(s \to s')}{P(s)}. \tag{D.19}$$

Any complete trajectory τ going through a state s can be (uniquely) decomposed into a partial trajectory $\tau' \in \mathcal{T}_{0,s}$ from s_0 to s, and a partial trajectory $\tau'' \in \mathcal{T}_{s,f}$ from s to s_f. Using the definition of $P(s)$, we have:

$$P(s) = \sum_{\tau \,:\, s \in \tau} P(\tau) = \sum_{\tau \,:\, s \in \tau} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_F(s_{t+1} \mid s_t) \tag{D.20}$$

$$= \left[\sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_F(s_{t+1} \mid s_t) \right] \underbrace{\left[\sum_{\tau'' \in \mathcal{T}_{s,f}} \prod_{(s_t \to s_{t+1}) \in \tau''} \hat{P}_F(s_{t+1} \mid s_t) \right]}_{= 1 \quad \text{(Lemma 3.2.5)}} \tag{D.21}$$

$$= \sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_F(s_{t+1} \mid s_t). \tag{D.22}$$

Similarly, any complete trajectory going through $s \to s'$ can be (uniquely) decomposed into a partial trajectory $\tau' \in \mathcal{T}_{0,s}$ from s_0 to s, and a partial trajectory $\tau'' \in \mathcal{T}_{s',f}$

from s' to s_f. Again, using the definition of $P(s \to s')$:

$$P(s \to s') = \sum_{\tau\,:\,(s\to s')\in\tau} P(\tau) = \sum_{\tau\,:\,(s\to s')\in\tau} \prod_{(s_t\to s_{t+1})\in\tau} \hat{P}_F(s_{t+1} \mid s_t) \tag{D.23}$$

$$= \underbrace{\left[\sum_{\tau'\in\mathcal{T}_{0,s}} \prod_{(s_t\to s_{t+1})\in\tau'} \hat{P}_F(s_{t+1} \mid s_t)\right]}_{=P(s)} \hat{P}_F(s' \mid s) \underbrace{\left[\sum_{\tau''\in\mathcal{T}_{s',f}} \prod_{(s_t\to s_{t+1})\in\tau''} \hat{P}_F(s_{t+1} \mid s_t)\right]}_{=1 \quad \text{(Lemma 3.2.5)}} \tag{D.24}$$

$$= P(s)\hat{P}_F(s' \mid s). \tag{D.25}$$

Combining the two results above, we get:

$$P_F(s' \mid s) = \frac{P(s \to s')}{P(s)} = \hat{P}_F(s' \mid s). \tag{D.26}$$

— $2 \Rightarrow 1$: Suppose that there exists a probability function $\hat{P}_F$ consistent with G such that for some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$

$$P(\tau) = \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}). \tag{D.27}$$

For the same reasons as those used to justify the uniqueness in the $1 \Rightarrow 2$ proof, $\hat{P}_F$ is necessarily equal to the forward transition probability P_F, associated with P.

We now want to show that the flow F associated with P is Markovian, by showing the Markov property from (3.19). Let $\tau' \in \mathcal{T}_{0,s}$ be any partial trajectory from s_0 to s; using the definition of conditional probability:

$$P(s \to s' \mid \tau') = \frac{P(s_0 \to \ldots \to s \to s')}{P(s_0 \to \ldots \to s)}. \tag{D.28}$$

Following the same idea as above, we will now rewrite $P(s_0 \to \ldots \to s)$ as a sum over complete trajectories that share the same prefix trajectory τ'. Any such complete trajectory τ can be (uniquely) decomposed into this common prefix τ', and a partial trajectory $\tau'' \in \mathcal{T}_{s,f}$ from s to s_f.

$$P(s_0 \to \ldots \to s) = \sum_{\tau\,:\,\tau'\subseteq\tau} P(\tau) = \sum_{\tau\,:\,\tau'\subseteq\tau} \prod_{(s_t\to s_{t+1})\in\tau} P_F(s_{t+1} \mid s_t) \tag{D.29}$$

$$= \left[\prod_{s_{t-1}\to s_t\in\tau'} P_F(s_t \mid s_{t-1})\right] \underbrace{\left[\sum_{\tau''\in\mathcal{T}_{s,f}} \prod_{(s_t\to s_{t+1})\in\tau''} P_F(s_{t+1} \mid s_t)\right]}_{=1 \quad \text{(Lemma 3.2.5)}} \tag{D.30}$$

$$= \prod_{s_{t-1}\to s_t\in\tau'} P_F(s_t \mid s_{t-1}). \tag{D.31}$$

Similarly, any complete trajectory τ that share the same prefix trajectory $(s_0, \ldots, s, s')$ can be (uniquely) decomposed into this common prefix, and a partial

trajectory $\tau'' \in \mathcal{T}_{s',f}$ from s' to s_f, leading to:

$$P(s_0 \to \ldots \to s \to s') = P(s_0 \to \ldots \to s) P_F(s' \mid s) \tag{D.32}$$

Combining the two results above, we can conclude that P satisfies the Markov property and therefore that the flow F is Markovian:

$$P(s' \to s \mid \tau') = \frac{P(s_0 \to \ldots \to s \to s')}{P(s_0 \to \ldots \to s)} = P_F(s' \mid s) = P(s' \to s \mid s) \tag{D.33}$$

— $\{1, 2\} \Rightarrow 3$: Suppose that F is a Markovian flow. We have shown above that this is equivalent to P being decomposed into a product of forward transition probabilities P_F. For some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = \prod_{t=1}^{n+1} P_F(s_t \mid s_{t-1}) = \prod_{t=1}^{n+1} \frac{P(s_{t-1} \to s_t)}{P(s_{t-1})} = \prod_{t=1}^{n+1} \frac{P(s_{t-1} \to s_t)}{P(s_t)} = \prod_{t=1}^{n+1} P_B(s_{t-1} \mid s_t), \tag{D.34}$$

where the third equality uses the fact that $P(s_0) = P(s_f) = 1$, and using the definition of the backwards transition probability P_B. The proof of uniqueness of P_B is similar to that of P_F in $1 \Rightarrow 2$, and uses:

$$P(s \to s') = \sum_{\tau\,:\,(s \to s') \in \tau} P(\tau) = \sum_{\tau\,:\,(s \to s') \in \tau} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_B(s_t \mid s_{t+1}) \tag{D.35}$$

$$= \underbrace{\left[\sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_B(s_t \mid s_{t+1}) \right]}_{=1 \quad \text{(Lemma 3.2.5)}} \hat{P}_B(s \mid s') \underbrace{\left[\sum_{\tau'' \in \mathcal{T}_{s',f}} \prod_{(s_t \to s_{t+1}) \in \tau''} \hat{P}_B(s_t \mid s_{t+1}) \right]}_{= P(s')} \tag{D.36}$$

$$= P(s') \hat{P}_B(s \mid s'), \tag{D.37}$$

— $3 \Rightarrow 1$: Similar to the proof of $2 \Rightarrow 1$, $\hat{P}_B$ is necessarily equal to the backwards transition probability P_B associated with P. Additionally, P_B is related to the forward transition probability P_F:

$$P(s \to s') = P_B(s \mid s') P(s') = P_F(s' \mid s) P(s). \tag{D.38}$$

We can therefore write the decomposition of P in terms of P_F instead of P_B. For some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = \prod_{t=1}^{n+1} P_B(s_{t-1} \mid s_t) = \prod_{t=1}^{n+1} \frac{P(s_{t-1})}{P(s_t)} P_F(s_{t+1} \mid s_t) = \frac{P(s_0)}{P(s_f)} \prod_{t=1}^{n+1} P_F(s_{t+1} \mid s_t) \tag{D.39}$$

$$= \prod_{t=1}^{n+1} P_F(s_{t+1} \mid s_t), \tag{D.40}$$

where we used the fact that $P(s_0) = P(s_f) = 1$. Using "$2 \Rightarrow 1$", we can conclude that F is a Markovian flow.

□

Proposition 3.2.19.

PROOF. Necessity is a direct consequence of Proposition 3.2.8. Let's show sufficiency. Let $\hat{P}_F$ be the forward probability function defined by:

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_F(s' \mid s) := \frac{\hat{F}(s \to s')}{\hat{F}(s)}. \tag{D.41}$$

$\hat{P}_F$ is consistent with G given that $\hat{F}$ satisfies the flow matching conditions ((3.22)). Let $\hat{Z} = \hat{F}(s_0)$. According to Proposition 3.2.18, there exists a unique Markovian flow F with forward transition probability function $P_F = \hat{P}_F$ and partition function $Z = \hat{Z}$, and such that for a trajectory $\tau = (s_0, \dots, s_{n+1} = s_f) \in \mathcal{T}$:

$$\forall \tau = (s_0, \dots, s_{n+1} = s_f) \in \mathcal{T} \quad F(\tau) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}. \tag{D.42}$$

Additionally, similar to the proof of Proposition 3.2.16, we can write for any state $s' \neq s_0$:

$$F(s') = \hat{Z} \sum_{\tau \in \mathcal{T}_{0,s'}} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_F(s_{t+1} \mid s_t) \tag{D.43}$$

$$= \hat{Z} \frac{\hat{F}(s')}{\hat{F}(s_0)} \underbrace{\sum_{\tau \in \mathcal{T}_{0,s'}} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_B(s_t \mid s_{t+1})}_{=1 \text{ , according to Lemma 3.2.5}} \tag{D.44}$$

$$= \hat{F}(s'), \tag{D.45}$$

where $\hat{P}_B(s' \mid s) := \frac{\hat{F}(s \to s')}{\hat{F}(s')}$ defines a backward probability function consistent with G. And because $\forall s \to s' \in \mathbb{A} \quad P_F(s' \mid s) = \hat{P}_F(s' \mid s)$, it follows that $\forall s \to s' \in \mathbb{A} \quad F(s \to s') = \hat{F}(s \to s')$.

To show uniqueness, let's consider a Markovian flow F' that matches $\hat{F}$ on states and edges. Following Proposition 3.2.16, for any trajectory $\tau = (s_0, \dots, s_{n+1} = s_f) \in \mathcal{T}$

$$F'(\tau) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)} = F(\tau). \tag{D.46}$$

□

Corollary 3.2.17.

PROOF. First, note that the procedure terminates with probability 1, given that G is acyclic.

For the procedure to terminate in a state s, it means that the trajectory $\tau \in \mathcal{T}$ implicitly constructed during the procedure contains the edge $s \to s_f$. The probability of the procedure

terminating in s is thus:

$$\sum_{\tau\in\mathcal{T}:s\to s_f\in\tau} \underbrace{\prod_{s'\to s''\in\tau} P_F(s'' \mid s')}_{P(\tau),\text{ according to (3.20)}} = P(s \to s_f) = P_\top(s) \tag{D.47}$$

□

Proposition 3.2.18.

Proof. In the first two settings, we define a flow function $F : \mathcal{T} \to \mathbb{R}^+$, at a trajectory $\tau = (s_0, s_1, \ldots, s_n, s_{n+1} = s_f)$ as:

(1) $F(\tau) := \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1})$,

(2) $F(\tau) := \hat{Z} \prod_{t=1}^{n+1} \hat{P}_B(s_{t-1} \mid s_t)$

We need to prove that it is the only Markovian flow that can be defined for both settings. The proof for the third setting will follow from that of the second setting.

First setting:

First, we need to show that the total flow Z associated with the flow function F ((3.11)) matches $\hat{Z}$. This is a consequence of Lemma 3.2.5:

$$Z = \sum_{\tau\in\mathcal{T}} F(\tau) = \hat{Z} \underbrace{\sum_{\tau=(s_0,s_1,\ldots,s_{n+1}=s_f)\in\mathcal{T}} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1})}_{=1 \text{ , according to Lemma 3.2.5}} = \hat{Z} \tag{D.48}$$

Then, we need to show that the forward transition probability function P_F associated with F ((3.15)) matches $\hat{P}_F$ and that the flow F is Markovian. To this end, note that the corresponding flow probability P satisfies (3.20). Thus, as a consequence of Proposition 3.2.16, F is a Markovian flow, and its forward transition probability function is $\hat{P}_F$.

As a last requirement, we need to show that if a Markovian flow F' has a partition function $Z' = \hat{Z}$ and a forward transition probability function $P'_F = \hat{P}_F$, then it is necessarily equal to F. This is a direct consequence of Proposition 3.2.16, given that for any $\tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T}$:

$$F'(\tau) = Z' \prod_{t=1}^{n+1} P'_F(s_t \mid s_{t-1}) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}) = F(\tau) \tag{D.49}$$

Second setting:

First, we show that as a consequence of Lemma 3.2.5, the total flow Z associated with F matches $\hat{Z}$:

$$Z = \sum_{\tau\in\mathcal{T}} F(\tau) = \hat{Z} \underbrace{\sum_{\tau=(s_0,s_1,\ldots,s_{n+1}=s_f)\in\mathcal{T}} \prod_{t=1}^{n+1} \hat{P}_B(s_{-1} \mid s_t)}_{=1 \text{ , according to Lemma 3.2.5}} = \hat{Z} \tag{D.50}$$

Second, we note that the flow probability P associated with F satisfies (3.21). Thus, as a consequence of Proposition 3.2.16, F is a Markovian flow, and its backward transition probability function is $\hat{P}_B$.

Finally, if a Markovian flow F' has a partition function $Z' = \hat{Z}$ and a backward transition probability function $P'_B = \hat{P}_B$, then following Proposition 3.2.16, $\forall \tau \in \mathcal{T},\ F'(\tau) = F(\tau)$.

Third setting:

From the terminating flows $\hat{F}(s \to s_f)$ and the backwards transition probabilities $\hat{P}_B(s \mid s')$ for non-terminating edges, we can uniquely define a total flow $\hat{Z}$, and extend $\hat{P}_B$ to all edges as follows:

$$\hat{Z} := \sum_{s \in Par(s_f)} \hat{F}(s \to s_f) \tag{D.51}$$

$$\hat{P}_B(s \mid s') := \begin{cases} \hat{P}_B(s \mid s') & \text{if } s' \neq s_f \\ \frac{\hat{F}(s \to s_f)}{\hat{Z}} & \text{otherwise.} \end{cases} \tag{D.52}$$

This takes us back to the second setting, for which we have already proven that with $\hat{Z}$ and $\hat{P}_B$ defined for all edges, a Markovian flow is uniquely defined.

□

Proposition 3.2.21.

Proof. For necessity, consider a flow F, with state flow function denoted F, and forward and backward transitions P_F and P_B. It is clear from the definition of P_F and P_B (Definition 3.2.12) that (3.25) holds. We prove the sufficiency of the condition by first defining the edge flow

$$\forall s \to s' \in \mathbb{A} \quad \hat{F}(s \to s') := \hat{F}(s) \hat{P}_F(s' \mid s). \tag{D.53}$$

We then sum both sides of (3.25) over s, yielding

$$\forall s' > s0 \quad \sum_{s \in Par(s')} \hat{F}(s) \hat{P}_F(s' \mid s) = \hat{F}(s') \sum_{s \in Par(s')} \hat{P}_B(s \mid s') = \hat{F}(s') \tag{D.54}$$

where we used the fact that $\hat{P}_B$ is a normalized probability distribution. Combining this with (D.53), we get

$$\forall s' > s_0 \quad \hat{F}(s') = \sum_{s \in Par(s')} \hat{F}(s \to s') \tag{D.55}$$

which is the first equality of the flow-matching condition ((3.22)) of Proposition 3.2.19. We can obtain the second equality by first using the normalization of $\hat{P}$ and then using our

definition of the edge flow ((D.53)):

$$\begin{aligned}\forall s' > s_0 \quad \hat{F}(s') &= \hat{F}(s') \sum_{s'' \in Child(s')} \hat{P}_F(s'' \mid s') \\ &= \sum_{s'' \in Child(s')} \hat{F}(s')\hat{P}_F(s'' \mid s') \\ &= \sum_{s'' \in Child(s')} \hat{F}(s' \to s''). \end{aligned} \tag{D.56}$$

Following Proposition 3.2.19, there exists a unique Markovian flow F with state and edge flows given by $\hat{F}$. Using (D.53) and (3.25), it follows that F has transition probabilities $\hat{P}_F$ and $\hat{P}_B$ as required. The uniqueness is also a consequence of (D.53). This proves sufficiency.

To show that $\hat{P}_F$ and $\hat{P}_B$ are compatible (Definition 3.2.20), we first combine (D.53) and (D.56) (with relabeling of variables) to obtain

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\sum_{s' \in Child(s)} \hat{F}(s \to s')},$$

we then isolate $\hat{P}_B$ in (3.25), yielding

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_B(s \mid s') = \frac{\hat{F}(s)}{\hat{F}(s')}\hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\hat{F}(s')} = \frac{\hat{F}(s \to s')}{\sum_{s'' \in Par(s')} \hat{F}(s'' \to s')},$$

We thus get (3.24) of Definition 3.2.20, as desired. □

Proposition 3.2.22.

Proof. First, note that before the main loop (the while loop), the queue U contains at least one element. Indeed, there has to be at least one $x \in \mathcal{S}^f$ such that s_f is its only child, otherwise, the acyclicity constraint would be broken.

Let $d(s)$ be the maximal distance from s_f to a node s. For example, if there are two (incomplete) trajectories from s to s_f: $s \to s' \to s_f$, and $s \to s'' \to s''' \to s_f$, then $d(s) = 3$.

Initially, before the while loop, U is filled with all states $s \in \mathcal{S}$ such that $d(s) = 1$. They will thus be processed before any state s with $d(s) > 1$.

After all states s' satisfying $d(s') = 1$ are dequeued from U, $\mathcal{Y}$ contains all such states. Additionally, all edges $s \to s'$ with $d(s') = 1$ would have been assigned an edge flow $\hat{F}(s \to s')$, and for each corresponding parent state s, $V(s)$ would be equal to $\sum_{s' \in Child(s)} \hat{F}(s \to s')$. Similarly, because for each corresponding s', $\sum_{s \in Par(s')} P_B(s \mid s') = 1$, we get that $\sum_{s \in Par(s')} \hat{F}(s \to s') = R(s') = V(s')$. The obtained edge flows thus satisfy the constraints (3.27). At this point, U is filled with states s satisfying $d(s) = 2$.

Such states would be processed (dequeued from U) iteratively, after which U would be filled with states s satisfying $d(s) = 3$, and the obtained edge flows satisfy the constraints (3.27). This will go on until U contains s_0, at which point the algorithm terminates.

Every edge $s \to s'$ is visited once, and only once, leadign to an algorithm complexity of $\mathcal{O}(|\mathbb{A}|)$. □

Proposition 3.2.24.

PROOF. Because F_1 and F_2 are Markovian, then for any trajectory $\tau = (s_0, \dots, s_{n+1} = s_f)$:

$$F_1(\tau) = \frac{\prod_{t=1}^{n+1} F_1(s_{t-1} \to s_t)}{\prod_{t=1}^{n} F_1(s_t)} \tag{D.57}$$

$$= \frac{\prod_{t=1}^{n+1} F_2(s_{t-1} \to s_t)}{\prod_{t=1}^{n} F_2(s_t)} \tag{D.58}$$

$$= F_2(\tau), \tag{D.59}$$

where we combined the definition of equivalent flows and Proposition 3.2.16.

Given a flow function F', because its state and edge flow functions satisfy the flow matching conditions (as a consequence of Proposition 3.2.8), then according to Proposition 3.2.19, the flow F defined by:

$$\forall \tau := (s_0, \dots, s_{n+1} = s_f) \in \mathcal{T} \quad F(\tau) = \frac{\prod_{t=1}^{n+1} F'(s_{t-1} \to s_t)}{\prod_{t=1}^{n} F'(s_t)} \tag{D.60}$$

is Markovian, and coincides with F' on state and edge flows. Combining this with the statement above, we conclude that F is the unique Markovian flow that is equivalent to F'. □

Proposition 3.4.6.

PROOF. Let $s \in \mathcal{S}$ be a state. Since the structure of the DAG G_s is clearly well-defined, we just need to show that there exists a flow function $F_s : \mathcal{T}_s \to \mathbb{R}^+$ that satisfies (3.62). If such a function exists for every $s \in \mathcal{S}$, then it would suffice to define the conditional flow function $F : \mathcal{S} \times \mathcal{T} \to \mathbb{R}^+$ as:

$$F(s, \tau) = \begin{cases} F_s(\tau) & \text{if } \tau \in \mathcal{T}_s \\ 0 & \text{otherwise.} \end{cases} \tag{D.61}$$

Let $A_{s'|s}$ be the set of complete trajectories in $\mathcal{T}_s$ terminating in $s' \geq s$; the condition in (3.62) then reads:

$$F_s(s' \to s_f) = F_s(A_{s'|s}) = \sum_{\tau \in A_{s'|s}} F_s(\tau) = F(s' \to s_f). \tag{D.62}$$

Note that in (D.62), $F(s' \to s_f)$ is a given quantity because the flow F is known. Since the sets of trajectories $\{A_{s'|s}, \ , s' \geq s\}$ form a partition of all the complete trajectories $\mathcal{T}_s$, (D.62) is a system of linear equations, whose unknowns are $F_s(\tau)$ for all $\tau \in \mathcal{T}_s$, where each equation involves separate sets of unknowns. Therefore there exists at least a solution $F_s(\tau)$ of this system.

We can construct such a solution in the following way. For some $\tau \in \mathcal{T}_s$, we can first start by selecting the complete trajectories $\bar{\tau} \in \mathcal{T}$ that contain τ:

$$C_\tau = \{\bar{\tau} \in \mathcal{T} \,:\, \tau \subseteq \bar{\tau}\}. \tag{D.63}$$

The key difference between the DAG G and the subgraph G_s though is that G may contain trajectories that terminate in some $s' \geq s$ but do not pass through s, and those are therefore not covered by the trajectories of G_s. Let $U_{s'|s}$ be the set of complete trajectories of G defined as

$$U_{s'|s} = \{\bar{\tau} \in \mathcal{T} \,:\, \exists s'' > s, s'' \in \bar{\tau}, s' \in \bar{\tau}, s \notin \bar{\tau}\}. \tag{D.64}$$

For all $\tau \in \mathcal{T}_s$ such that τ terminates in some $s' \geq s$, we can therefore construct the flow $F_s(\tau)$ as

$$F_s(\tau) := F(C_\tau) + \frac{1}{n} F(U_{s'|s}), \tag{D.65}$$

where $n = |A_{s'|s}|$ is the number of trajectories $\tau' \in \mathcal{T}_s$ that terminate in s'. It is easy to verify that $F_s(\tau)$ is a solution of (D.62). □

Proposition 3.4.7.

Proof. This is a direct consequence of Proposition 3.2.10, applied to the state-conditional flow function $\mathcal{F}_s$, along with Definition 3.4.1.

$$F_s(s_0 \mid s) = \sum_{\tau \in \mathcal{T}_s} F_s(\tau) = \sum_{s' \,:\, s' \geq s} F_s(s' \to s_f) \tag{D.66}$$

$$= \sum_{s' \,:\, s' \geq s} F(s' \to s_f) = \sum_{s' \,:\, s' \geq s} e^{-\mathcal{E}(s')} \tag{D.67}$$

□

Corollary 3.4.8.

Proof. Because F_s is a flow function, Definition 3.2.11 and Proposition 3.2.10 tell us that:

$$P_\top(s' \mid s) = \begin{cases} \dfrac{F_s(s' \to s_f)}{F_s(s)} & \text{if } s' \geq s \\ 0 & \text{otherwise} \end{cases} \tag{D.68}$$

Combining this with Proposition 3.4.7, and (3.62), we obtain for $s' \geq s$:

$$P_\top(s' \mid s) = \mathbf{1}_{s' \geq s} \frac{F(s' \to s_f)}{e^{-\mathcal{F}(s)}} \tag{D.69}$$

$$= \mathbf{1}_{s' \geq s} \frac{e^{-\mathcal{E}(s')}}{e^{-\mathcal{F}(s)}} \tag{D.70}$$

□

Proposition D.1.2.

PROOF. First apply the definition of $P_\top(s)$, then (D.12) on both flows:

$$-\sum_s P_\top(s)\log P_\top(s) = -\sum_s \frac{R(s)}{F(s_0)}(\log R(s) - \log F(s_0)) \tag{D.71}$$

$$= \frac{\Big(-\sum_s R(s)\log R(s)\Big) + \Big(\log F(s_0)\ \sum_s R(s)\Big)}{F(s_0)} \tag{D.72}$$

$$= \frac{F'(s_0)}{F(s_0)} + \log F(s_0). \tag{D.73}$$

Note that we need $R(s) < 1$ to make sure that the rewards $R'(s)$ (and thus the flows) are positive. □

Lemma 3.7.2.

PROOF. Consider a complete trajectory $\tau = (s_{m_0} \to \ldots \to s_{m_1} \to \ldots \to \ldots s_{m_2} \to \ldots \to \ldots \to s_{m_K})$. And let $\tau_k = (s_{m_k} \to \ldots \to s_{m_{k+1}})$, for every $k < K$.

Denote by $\hat{Z}_k$ and $\check{Z}_k$ the partition functions (constant of proportionality in (3.78)) of $\hat{p}_k$ and $\check{p}_k$ respectively, for every $k < K$. It is straightforward to see that for every $0 < k < K$:

$$\hat{Z}_{k+1} = \check{Z}_k = \sum_{s_{m_{k+1}} \in \mathcal{S}_{m_{k+1}}} F_{k+1}(s_{m_{k+1}}) \tag{D.74}$$

$$\prod_{k=0}^{K-1} \hat{p}_k(\tau_k) = \frac{\prod_{k=0}^{K-1} F_k(s_{m_k})}{\prod_{k=0}^{K-1} \hat{Z}_k} P_F(\tau), \tag{D.75}$$

$$\prod_{k=0}^{K-1} \check{p}_k(\tau_k) = \frac{\prod_{k=0}^{K-1} F_{k+1}(s_{m_{k+1}})}{\prod_{k=0}^{K-1} \check{Z}_k} P_B(\tau \mid s_{m_K}). \tag{D.76}$$

Because $\hat{p}_k = \check{p}_k$ for all $k = 0 \ldots K-1$, then both right-hand sides of (D.75) and (D.76) are equal. Combining this with (D.74), we obtain:

$$\forall \tau \in \mathcal{T} \quad \underbrace{\frac{F_0(s_0)}{\hat{Z}_0}}_{=1} P_F(\tau) = \frac{R(x_\tau)}{\sum_{x \in \mathcal{X}} R(x)} P_B(\tau \mid x), \tag{D.77}$$

which implies the TB constraint is satisfied for all $\tau \in \mathcal{T}$. Malkin et al. (2022) shows that this is a sufficient condition for the terminating state distribution induced by P_F to match the target reward function R, which completes the proof. □

Proposition 3.7.3.

PROOF. For a complete trajectory $\tau \in \mathcal{T}$, denote by $c(\tau) = \log \frac{P_F(\tau)}{R(x_\tau)P_B(\tau|x_\tau)}$. We have the following:

$$\nabla_\theta c(\tau) = \nabla_\theta \log P_F(\tau) \tag{D.78}$$

$$\nabla_\phi c(\tau) = -\nabla_\phi \log P_B(\tau \mid x_\tau) = -\nabla_\phi \log P_B(\tau) \tag{D.79}$$

Denoting by $f_1 : t \mapsto t \log t$ and $f_2 : t \mapsto -\log t$, which correspond to the forward and reverse KL divergences, respectively, and starting from

$$\mathcal{L}_{\text{HVI},f_2}(P_F, P_B) = D_{KL}(P_F \| P_B) = \mathbb{E}_{\tau\sim P_F}\left[\log \frac{P_F(\tau)}{P_B(\tau)}\right] = \mathbb{E}_{\tau\sim P_F}\left[c(\tau)\right] + \log \hat{Z},$$

$$\mathcal{L}_{\text{HVI},f_1}(P_F, P_B) = D_{KL}(P_B \| P_F) = \mathbb{E}_{\tau\sim P_B}\left[\log \frac{P_B(\tau)}{P_F(\tau)}\right] = -\left(\mathbb{E}_{\tau\sim P_B}\left[c(\tau)\right] + \log \hat{Z}\right),$$

we obtain:

$$\nabla_\theta \mathcal{L}_{\text{HVI},f_2}(P_F, P_B) = \nabla_\theta \mathbb{E}_{\tau\sim P_F}\left[c(\tau)\right] = \mathbb{E}_{\tau\sim P_F}[\nabla_\theta \log P_F(\tau) c(\tau) + \nabla_\theta c(\tau)],$$

$$\nabla_\phi \mathcal{L}_{\text{HVI},f_1}(P_F, P_B) = -\nabla_\phi \mathbb{E}_{\tau\sim P_B}\left[c(\tau)\right] = -\mathbb{E}_{\tau\sim P_B}[\nabla_\phi \log P_B(\tau) c(\tau) + \nabla_\phi c(\tau)].$$

From (D.78) and (D.79), we obtain:

$$\mathbb{E}_{\tau\sim P_F}[\nabla_\theta c(\tau)] = \mathbb{E}_{\tau\sim P_F}[\nabla_\theta \log P_F(\tau)] = \sum_{\tau\in\mathcal{T}} P_F(\tau)\nabla_\theta \log P_F(\tau) = \sum_{\tau\in\mathcal{T}} \nabla_\theta P_F(\tau) = \nabla_\theta 1 = 0$$

Hence, for any scalar $Z > 0$, we can write:

$$\mathbb{E}_{\tau\sim P_F}[\nabla_\theta c(\tau)] = 0 = \mathbb{E}_{\tau\sim P_F}[\nabla_\theta \log P_F(\tau) \log Z]$$

and similarly

$$\mathbb{E}_{\phi\sim P_F}[\nabla_\phi c(\tau)] = 0 = \mathbb{E}_{\tau\sim P_B}[\nabla_\phi \log P_B(\tau) \log Z].$$

Plugging these two equalities back in the HVI gradients above, we obtain:

$$\nabla_\theta \mathcal{L}_{\text{HVI},f_2}(P_F, P_B) = \mathbb{E}_{\tau\sim P_F}[\nabla_\theta \log P_F(\tau) \log \frac{Z P_F(\tau)}{R(x_\tau)P_B(\tau \mid x_\tau)}]$$

$$\nabla_\phi \mathcal{L}_{\text{HVI},f_1}(P_F, P_B) = -\mathbb{E}_{\tau\sim P_B}[\nabla_\theta \log P_B(\tau) \log \frac{Z P_F(\tau)}{R(x_\tau)P_B(\tau \mid x_\tau)}]$$

The last two equalities hold for any scalar Z (that does not depend on the parameters of P_F, P_B, and does not depend on any trajectory). In particular, the equations hold for the

parameter Z of the Trajectory Balance objective. It thus follows that:

$$\nabla_\theta \mathcal{L}_{\mathrm{HVI},f_2}(P_F, P_B) = \frac{1}{2}\mathbb{E}_{\tau\sim P_F}\left[\nabla_\theta\left(\log\frac{ZP_F(\tau)}{R(x_\tau)P_B(\tau \mid x_\tau)}\right)^2\right] = \frac{1}{2}\mathbb{E}_{\tau\sim P_B}[\nabla_\theta \mathcal{L}_{\mathrm{TB}}(\tau; P_F, P_B, Z)]$$

$$\nabla_\phi \mathcal{L}_{\mathrm{HVI},f_1}(P_F, P_B) = \frac{1}{2}\mathbb{E}_{\tau\sim P_B}\left[\nabla_\theta\left(\log\frac{ZP_F(\tau)}{R(x_\tau)P_B(\tau \mid x_\tau)}\right)^2\right] = \frac{1}{2}\mathbb{E}_{\tau\sim P_B}[\nabla_\phi \mathcal{L}_{\mathrm{TB}}(\tau; P_F, P_B, Z)]$$

As an immediate corollary, we obtain that the expected on-policy TB gradient does not depend on the estimated partition function Z. □

Lemma 3.7.5.

Proof. The RHS of (3.87) equals

$$\begin{aligned}
&\nabla_\phi\left[\mathbb{E}_{\tau\sim P_F}\left[\left(\log\frac{P_B(\tau \mid x_\tau)R(x_\tau)}{\hat{Z}P_F(\tau)}\right)^2 + 2(\log Z - \log\hat{Z})\log\frac{P_F(\tau)\hat{Z}}{P_B(\tau \mid x_\tau)R(x_\tau)}\right]\right]\\
=&\mathbb{E}_{\tau\sim P_F}\left[\nabla_\phi\left(\left(\log\frac{P_B(\tau \mid x_\tau)R(x_\tau)}{\hat{Z}P_F(\tau)}\right)^2 + 2(\log Z - \log\hat{Z})\log\frac{P_F(\tau)\hat{Z}}{P_B(\tau \mid x_\tau)R(x_\tau)}\right)\right]\\
=&\mathbb{E}_{\tau\sim P_F}\left[2\nabla_\phi \log P_B(\tau \mid x_\tau)\log\frac{P_B(\tau \mid x_\tau)R(x_\tau)}{\hat{Z}P_F(\tau)} - 2(\log Z - \log\hat{Z})\nabla_\phi \log P_B(\tau \mid x_\tau)\right]\\
=&2\mathbb{E}_{\tau\sim P_F}\left[\nabla_\phi \log P_B(\tau \mid x_\tau)\log\frac{P_B(\tau \mid x_\tau)R(x_\tau)}{ZP_F(\tau)}\right]\\
=&\mathbb{E}_{\tau\sim P_F}[\nabla_\phi \mathcal{L}_{\mathrm{TB}}(\tau)]
\end{aligned}$$

□

Proposition 3.7.4.

Proof. For a subtrajectory $\tau_k = (s_{m_k} \to \ldots \to s_{m_{k+1}}) \in \mathcal{T}_k$, let $c(\tau_k) = \log\frac{F_k(s_{m_k})P_F(\tau_k)}{F_{k+1}(s_{m_{k+1}})P_B(\tau_k|s_{m_{k+1}})}$.

First, note that because $\hat{Z}_k$ and $\check{Z}_k$ are not functions of ϕ, θ ((D.75)):

$$\nabla_\phi c(\tau_k) = -\nabla_\phi \log\frac{F_{k+1}(s_{m_{k+1}})P_B(\tau_k \mid s_{m_{k+1}})}{\check{Z}_k} = -\nabla_\phi \log\check{p}_k(\tau_k) \tag{D.80}$$

$$\nabla_\theta c(\tau_k) = \nabla_\theta \log\frac{F_k(s_{m_k})P_F(\tau_k)}{\hat{Z}_k} = \nabla_\phi \log\hat{p}_k(\tau_k) \tag{D.81}$$

We will prove (3.85). The proof of (3.86) follows the same reasoning and is omitted for conciseness.

$$
\begin{aligned}
D_{f_1}(\check{p}_k \| \hat{p}_k) &= D_{KL}(\check{p}_k \| \hat{p}_k) \\
\nabla_\phi D_{f_1}(\check{p}_k \| \hat{p}_k) &= \nabla_\phi \sum_{\tau_k \in \mathcal{T}_k} \check{p}_k(\tau_k) \log \frac{\check{p}_k(\tau_k)}{\hat{p}_k(\tau_k)} \\
&= -\nabla_\phi \sum_{\tau_k \in \mathcal{T}_k} \check{p}_k(\tau_k) c(\tau_k) + \underbrace{\nabla_\phi \log \frac{\hat{Z}_k}{\check{Z}_k}}_{=0, \text{ according to (D.75)}} \\
&= -\sum_{\tau_k \in \mathcal{T}_k} (\nabla_\phi \check{p}_k(\tau_k) c(\tau_k) + \check{p}_k(\tau_k) \nabla_\phi c(\tau_k)) \\
&= -\sum_{\tau_k \in \mathcal{T}_k} (\check{p}_k(\tau_k) \nabla_\phi \log \check{p}_k(\tau_k) c(\tau_k) + \check{p}_k(\tau_k) \nabla_\phi c(\tau_k)) \\
&= -\mathbb{E}_{\tau_k \sim \check{p}_k}[\nabla_\phi \log \check{p}_k(\tau_k) c(\tau_k)] + \sum_{\tau_k \in \mathcal{T}_k} \check{p}_k(\tau_k) \nabla_\phi \log \check{p}_k(\tau_k) \quad \text{following (D.80)} \\
&= -\mathbb{E}_{\tau_k \sim \check{p}_k}[\nabla_\phi \log P_B(\tau_k \mid s_{m_{k+1}}) c(\tau_k)] + \underbrace{\nabla_\phi \sum_{\tau_k \in \mathcal{T}_k} \check{p}_k(\tau_k)}_{=0} \\
&= \mathbb{E}_{\tau_k \sim \check{p}_k}\left[\nabla_\phi \log P_B(\tau_k \mid s_{m_{k+1}}) \log \frac{F_{k+1}(s_{m_{k+1}}) P_B(\tau_k \mid s_{m_{k+1}})}{F_k(s_{m_k}) P_F(\tau_k)}\right] \\
&= \frac{1}{2}\mathbb{E}_{\tau_k \sim \check{p}_k}\left[\nabla_\phi \left(\log \frac{F_k(s_{m_k}) P_F(\tau_k)}{F_{k+1}(s_{m_{k+1}}) P_B(\tau_k \mid s_{m_{k+1}})}\right)^2\right] \\
&= \frac{1}{2}\mathbb{E}_{\tau_k \sim \check{p}_k}[\nabla_\phi \mathcal{L}_{\text{SubTB}}(\tau_k; P_F, P_B, F)]
\end{aligned}
$$

□

D.3. Additional experimental details

D.3.1. Hypergrid experiments

Architectural details. The forward and backward policies are parameterized as neural networks with two hidden layers of 256 units each. The neural networks take as input a one-hot representation of a state (also called K-hot or multi-hot representations), which is a $H \times D$ vector including exactly D ones and $(H-1)D$ zeros, and output the logits of P_F and P_B respectively. Forbidden actions (e.g., when a coordinate is already maxed out at $H-1$) are masked out by setting the corresponding logits to $-\infty$ after the forward pass. Unlike Malkin et al. (2022), we do not tie the parameters of P_F and P_B.

Behavior policy. The behaviour policy is obtained from the forward policy P_F by subtracting a scalar ϵ from the logits output by the forward policy neural network. The value of ϵ is decayed from ϵ_{init} to 0 following a cosine annealing schedule (Loshchilov and Hutter, 2017),

and the value $\epsilon = 0$ is reached at an iteration T_{max}. The values of ϵ_{init} and T_{max} were treated as hyperparameters.

Hyperparameter optimization. Our experiments have shown that HVI objectives were brittle to the choice of hyperparameters (mainly learning rates) and that the ones used for Trajectory Balance in Malkin et al. (2022) do not perform as well in the larger 128×128 grid we considered. To obtain a fair comparison between GFlowNets and HVI methods, particular care was given to optimizing hyperparameters in this domain. The optimization was performed in two stages:

(1) We use a batch size of 64 for all learning objectives, whether on-policy or off-policy, and the Adam optimizer with secondary parameters set to their default values, for the parameters of P_F, the parameters of P_B, and $\log Z$ (which is initialized at 0). The learning rates of $P_F, P_B, \log Z$, along with a schedule factor $\gamma < 1$ by which they are multiplied when the JSD plateaus for more than 500 iterations (i.e., 500×64 trajectories sampled), were sought after separately for each combination of learning objective and sampling method (on-policy or off-policy), using a Bayesian search with the JSD evaluated at $200K$ trajectories as an optimization target. The choice of the baseline for HVI methods (except WS, which does not have a score function estimator of the gradient) was also treated as a hyperparameter.

(2) All objectives were then trained for 10^6 trajectories using all the combinations of hyperparameters found in the first stage for 5 seeds each. The final set of hyperparameters for each objective and sampling mode was chosen as the one that leads to the lowest area under the JSD curve (approximated with the trapezoids method).

For off-policy runs, T_{max} was defined as a fraction $1/n$ of the total number of iterations (equal to $10^6/64$). The value of n and ϵ_{init} was optimized like the learning rate and the schedule, as described above.

D.3.2. Molecule experiments

Most experiment settings were identical to those of Malkin et al. (2022), in particular, the reward model f the held-out set of molecules used to compute the performance metric, the GFlowNet model architecture (a graph neural network introduced by Bengio et al. (2021)), and the off-policy exploration rate. All models were trained with the Adam optimizer and batch size 4 for a maximum of 50000 batches. The metric was computed after every 5000 batches, and the last calculated value was reported, which was sometimes not the value after 50000 batches when the training runs terminated early because of numerical errors.

D.3.3. Bayesian structure learning experiments

Bayesian Networks. A Bayesian Network is a probabilistic model where the joint distribution over d random variables $\{X_1, \dots, X_d\}$ factorizes according to a directed acyclic graph (DAG) G:

$$p(X_1, \dots, X_d) = \prod_{i=1}^{d} p(X_i \mid \mathrm{Pa}_G(X_i)),$$

where $\mathrm{Pa}_G(X_i)$ is the set of parent variables of X_i in the graph G. Each conditional distribution in the factorization above is also associated with a set of parameters $\theta \in \Theta$. The structure G of the Bayesian Network is often assumed to be known. However, when the structure is unknown, we can learn it based on a dataset of observations $\mathcal{D}$: this is called *structure learning*.

Structure of the state space. We use the same structure of graded DAG $\mathcal{G}$ as the one described in (Deleu et al., 2022), where each state of $\mathcal{G}$ is itself a DAG G, and where actions correspond to adding one edge to the current graph G to transition to a new graph G'. Only the actions maintaining the acyclicity of G' are considered valid; this ensures that all the states are well-defined DAGs, meaning that all the states are terminating here (we define a distribution over DAGs). Similar to the hypergrid environment, the action space also contains an extra action "stop" to complete the generation process and return the current graph as a sample of our distribution; this "stop" action is denoted $G \rightarrow s_f$.

Reward function. Our objective in Bayesian structure learning is to approximate the posterior distribution over DAGs $p(G \mid \mathcal{D})$, given a dataset of observations $\mathcal{D}$. Since our goal is to find a forward policy P_F for which $P_\top(G) \propto R(G)$ (see Section 3.7.1), we can define the reward function as the joint distribution $R(G) = p(G, \mathcal{D}) = p(\mathcal{D} \mid G)p(G)$, where $p(G)$ is a prior over graphs (assumed to be uniform throughout the paper), and $p(\mathcal{D} \mid G)$ is the marginal likelihood. Since the marginal likelihood involves marginalizing over the parameters of the Bayesian Network

$$p(\mathcal{D} \mid G) = \int_\Theta p(\mathcal{D} \mid \theta, G)p(\theta \mid G)\, d\Theta,$$

it is, in general intractable. We consider here a special class of models, called *linear-Gaussian models*, where the marginal likelihood can be computed in closed form; for this class of models, the log-marginal likelihood is also called the BGe score (Geiger and Heckerman, 1994; Kuipers et al., 2014) in the structure learning literature.

For each experiment, we sampled a dataset $\mathcal{D}$ of 100 samples from a randomly generated Bayesian network. The (ground truth) structure of the Bayesian Network was generated following an Erdős-Rényi model, with about d edges on average (to encourage sparsity on such small graphs with $d \leq 5$). Once the structure is known, the parameters of the linear-Gaussian model were sampled randomly from a standard Normal distribution $\mathcal{N}(0, 1)$. See (Deleu et al., 2022) for more details about the data generation process. For each setting

(different values of d) and each objective, we repeated the experiment over 20 different seeds.

Forward policy. Deleu et al. (2022) parameterized the forward policy P_F using a linear transformer, taking all the d^2 possible edges in the graph G as input and returning a probability distribution over those edges, where the invalid actions were masked out. We parameterized P_F using a simpler neural network architecture based on a graph neural network (Battaglia et al., 2018). The GNN takes the graph G as an input, where each node of the graph is associated with a (learned) embedding, and it returns for each node X_i a pair of embeddings $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$. The probability of adding an edge $X_i \rightarrow X_j$ to transition from G to G' (given that we do not terminate in G) is then given by

$$P_F(G' \mid G, \neg s_f) \propto \exp(\boldsymbol{u}_i^\top \boldsymbol{v}_j),$$

assuming that $X_i \rightarrow X_j$ is a valid action (i.e., it doesn't introduce a cycle in G) and where the normalization depends only on all the valid actions. We then use a hierarchical model to obtain the forward policy $P_F(G' \mid G)$, following (Deleu et al., 2022):

$$P_F(G' \mid G) = (1 - P_F(s_f \mid G))P_F(G' \mid G, \neg s_f).$$

Recall that the backward policy P_B is fixed here, as the uniform distribution over the parents of G (i.e., all the graphs were exactly one edge has been removed from G).

(Modified) Detailed Balance objective. For completeness, we recall here the modified Detailed Balance (DB) objective (Deleu et al., 2022) as a particular case of the DB objective (Bengio et al., 2023; see also (3.72)) when all the states of $\mathcal{G}$ are terminating (which is the case in our Bayesian structure learning experiments):

$$\mathcal{L}_{(M)DB}(G \rightarrow G'; P_F, P_B) = \left(\log \frac{R(G')P_B(G \mid G')P_F(s_f \mid G)}{R(G)P_F(G' \mid G)P_F(s_f \mid G')}\right)^2.$$

Optimization. Following (Deleu et al., 2022), we used a replay buffer for all our off-policy objectives ((Modified) DB, TB, and Reverse KL). All the objectives were optimized using a batch size of 256 graphs sampled either on-policy from P_F or from the replay buffer. We used the Adam optimizer, with the best learning rate found among $\{10^{-6}, 3 \times 10^{-6}, 10^{-5}, 3 \times 10^{-5}, 10^{-4}\}$. For the TB objective, we learned $\log Z$ using SGD with a learning rate of 0.1 and momentum 0.8.

Edge marginals. In addition to the Jensen-Shannon divergence (JSD) between the true posterior distribution $p(G \mid \mathcal{D})$ and the posterior approximation $P_\top(G)$ (see Section 3.8.1 for details about how this divergence is computed), we also compare the edge marginals computed with both distributions. That is, for any edge $X_i \rightarrow X_j$ in the graph, we compare

$$p(X_i \rightarrow X_j \mid \mathcal{D}) = \sum_{G \mid X_i \in \mathrm{Pa}_G(X_j)} p(G \mid \mathcal{D}) \quad \text{and} \quad P_\top(X_i \rightarrow X_j) = \sum_{G \mid X_i \in \mathrm{Pa}_G(X_j)} P_\top(G).$$

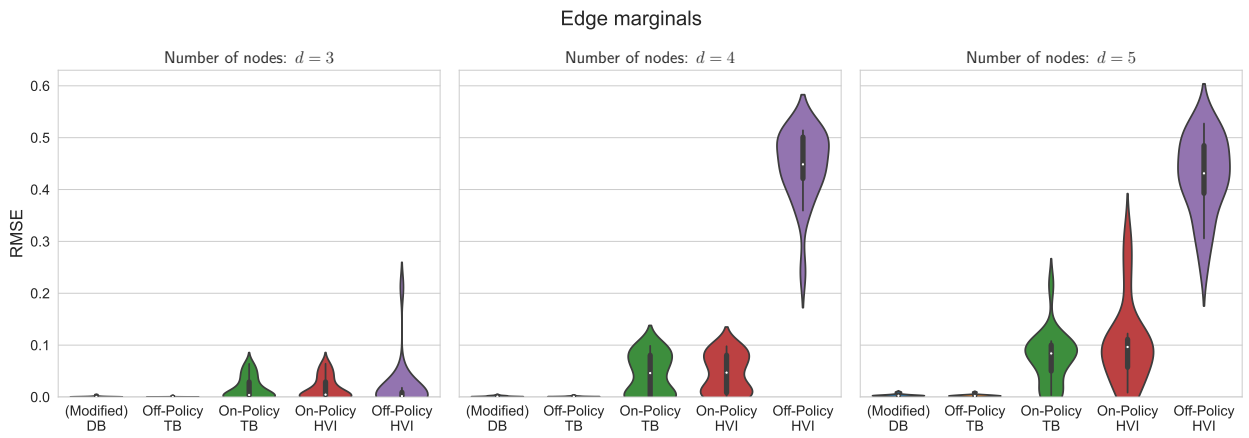


Figure D.1 – Comparison of edge marginals computed using the target posterior distribution and using the posterior approximations found either with the GFlowNet objectives or Reverse KL. Performance is reported as the Root Mean Square Error (RMSE) between the marginals (lower is better).

The edge marginal quantifies how likely an edge $X_i \to X_j$ is to be present in the structure of the Bayesian Network and is of particular interest in the (Bayesian) structure learning literature. To measure how accurate the posterior approximation $P_\top$ is for the different objectives considered here, we use the Root Mean Square Error (RMSE) between $p(X_i \to X_j \mid \mathcal{D})$ and $P_\top(X_i \to X_j)$, for all possible pairs of nodes (X_i, X_j) in the graph.

Figure D.1 shows the RMSE of the edge marginals for different GFlowNet objectives and Reverse KL (denoted as HVI here for brevity). The results on the edge marginals largely confirm the observations made in Section 3.8.4: the off-policy GFlowNet objectives ((Modified) DB & TB) consistently perform well across all experimental settings; On-Policy TB & On-Policy Reverse KL perform similarly and degrade as the complexity of the experiment increases (as d increases); and Off-Policy Reverse KL has a performance that degrades the most as the complexity increases, where the edge marginals given by $P_\top(X_i \to X_j)$ do not match the true edge marginals $p(X_i \to X_j \mid \mathcal{D})$ accurately.

Appendix E

Appendix for Chapter 4

E.1. How to define a backward reference kernel

Given a reference kernel κ, designing a backward reference kernel κ^b can be done using reverse kernels (Cappé et al., 2009), of which we recall a definition below:

Definition E.1.1 (Reverse kernel)**.** Let $(\bar{\mathcal{S}}, \Sigma)$ be a measurable space, κ be a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$, and ν be a positive measure on $(\bar{\mathcal{S}}, \Sigma)$. A *reverse kernel* κ_ν^r associated to ν and κ is a transition kernel over $(\bar{\mathcal{S}}, \Sigma)$ such that:

$$\nu \otimes \kappa = (\nu\kappa) \otimes \kappa_\nu^r \tag{E.1}$$

Note how (E.1) is different from the condition of the backward reference kernel (4.8). Conveniently, there is a reference measure ν for which the two conditions are equivalent, meaning that the backward reference kernel can be defined as the reverse kernel associated to ν and κ. While there is no guarantee that the reverse kernel exists or is unique in general, existence is guaranteed if $(\bar{\mathcal{S}}, \mathcal{T})$ is a Polish space (e.g., a discrete space, the Euclidian space $\mathbb{R}^n$, hyperrectangles or balls in $\mathbb{R}^n$, or products or disjoint unions of countable families thereof) (Cappé et al., 2009). The following proposition shows that.

Proposition E.1.2. *Given a Polish space $(\bar{\mathcal{S}}, \mathcal{T})$, with source and sink states $s_0, \bot \in \bar{\mathcal{S}}$ such that $\{s_0\}$ and $\{\bot\}$ are both open and closed sets, and a transition kernel κ on $(\bar{\mathcal{S}}, \Sigma)$ satisfying (4.5), (4.6) and (4.11). Let ν be the measure defined by:*

$$\nu = \sum_{n=0}^{N} \kappa^n(s_0, -), \tag{E.2}$$

and let κ^r_ν be any reverse kernel associated to κ and ν that satisfies the following two conditions:

$$\kappa^r_\nu(s_0, -) = 0 \quad \textit{i.e., it's the trivial measure} \tag{E.3}$$

$$\forall s \in \mathcal{S},\ \kappa^r_\nu(s, \{\bot\}) = 0. \tag{E.4}$$

Let κ^b be a transition kernel on $(\bar{\mathcal{S}}, \Sigma)$ defined by:

$$\begin{aligned}
&\forall s \neq \bot,\ \kappa^b(s, -) = \kappa^r_\nu(s, -),\\
&\forall B \in \Sigma_{|\mathcal{S}},\ \kappa^b(\bot, B) = \frac{1}{\nu(\{\bot\})} \nu \otimes \kappa(B \times \{\bot\}),\\
&\kappa^b(\bot, \{\bot\}) = 0.
\end{aligned}$$

ν is strictly positive and κ^b satisfies (4.8). Note that the existence of a reverse kernel satisfying (E.3) and (E.4) is guaranteed by Lemma E.1.3 below.

Lemma E.1.3. *Given a Polish space $(\bar{\mathcal{S}}, \mathcal{T})$, with source and sink states $s_0, \bot \in \bar{\mathcal{S}}$ such that $\{s_0\}$ and $\{\bot\}$ are both open and closed sets, and a transition kernel κ on $(\bar{\mathcal{S}}, \Sigma)$ satisfying (4.5), (4.6) and (4.11). Then the measure ν defined by:*

$$\nu = \sum_{n=0}^{N} \kappa^n(s_0, -), \tag{E.5}$$

If κ^b is a reverse kernel associated to ν and κ, then the kernel κ' defined by:

$$\kappa'(s_0, -) = 0, \tag{E.6}$$

$$\kappa'(\bot, -) = \kappa^b(\bot, -), \tag{E.7}$$

$$\forall s' \in \mathcal{S} \setminus \{s_0\},\ \kappa'(s', \{\bot\}) = 0 \tag{E.8}$$

$$\forall s' \in \mathcal{S} \setminus \{s_0\},\ \forall B \in \Sigma,\ \bot \notin B \Rightarrow \kappa'(s', B) = \kappa^b(s, B), \tag{E.9}$$

is also a reverse kernel associated to ν and κ.

E.2. Experimental details

E.2.1. A synthetic continuous environment

Hyperparameters. We learned the concentration parameters of the Beta distributions, which were restrained to the interval $[0.1, 5.1]$, using a three-layered neural network with 128 units per layer, and leaky ReLU activation for $s \neq s_0$. The parameters corresponding to $p_F(s_0, -)$ were learned separately.

Each iteration consisted of sampling 128 trajectories from the forward policy, and evaluating the TB or the DB loss (with $\alpha = 1$), before taking a gradient step on the learned parameters. We trained the models for 20,000 iterations.

For both the DB and TB losses, we used a learning rate of 10^{-3} for the parameters of $p_F, p_B, p_F(s_0, -)$ (and $\log Z$ for TB). The learning rate used for u is 10^{-2}. The learning rate was annealed using a discount factor of 0.5 every 2500 iterations.

In experiments with learned p_B, both p_F and p_B shared parameters except in the output layer. In DB, p_F and u did not share parameters except in the output layer.

Evaluation metric. We approximated the JSD between the learned the terminating state distribution and the target distribution following the scheme described in Section 4.4.1.

E.2.2. Low-dimensional stochastic control

The neural network computing $\mu(\mathbf{x}_t, t)$ had the same architecture as in Zhang and Chen (2022): a pair of 2-layer MLP processing $\mathbf{x}_t$ and a 128-dimensional Fourier feature representation of t, followed by a 3-layer MLP on the concatenation of the features derived from $\mathbf{x}_t$ and from t. We set $\sigma = 5$ for the Gaussians density and $\sigma = 1$ for funnel density. Exploration algorithms added a constant $\frac{\epsilon^2}{T}$ to the sampling policy variance at each step; we used a value of $\epsilon = 0.1$ linearly annealed to 0 over the course of training. All models are trained for 1500 batches of 300 samples with a learning rate of 10^{-2} for the policy and 10^{-1} for $\log Z$ (in the case of GFlowNet algorithms); we found that higher learning rates made optimization unstable. We also observed that the off-policy forward KL and TB algorithms continue to improve with longer training, unlike the on-policy algorithms, which experience mode collapse and cease to discover new areas of the density landscape. Figure E.1 shows samples from models trained with various algorithms and highlights the importance of exploration.

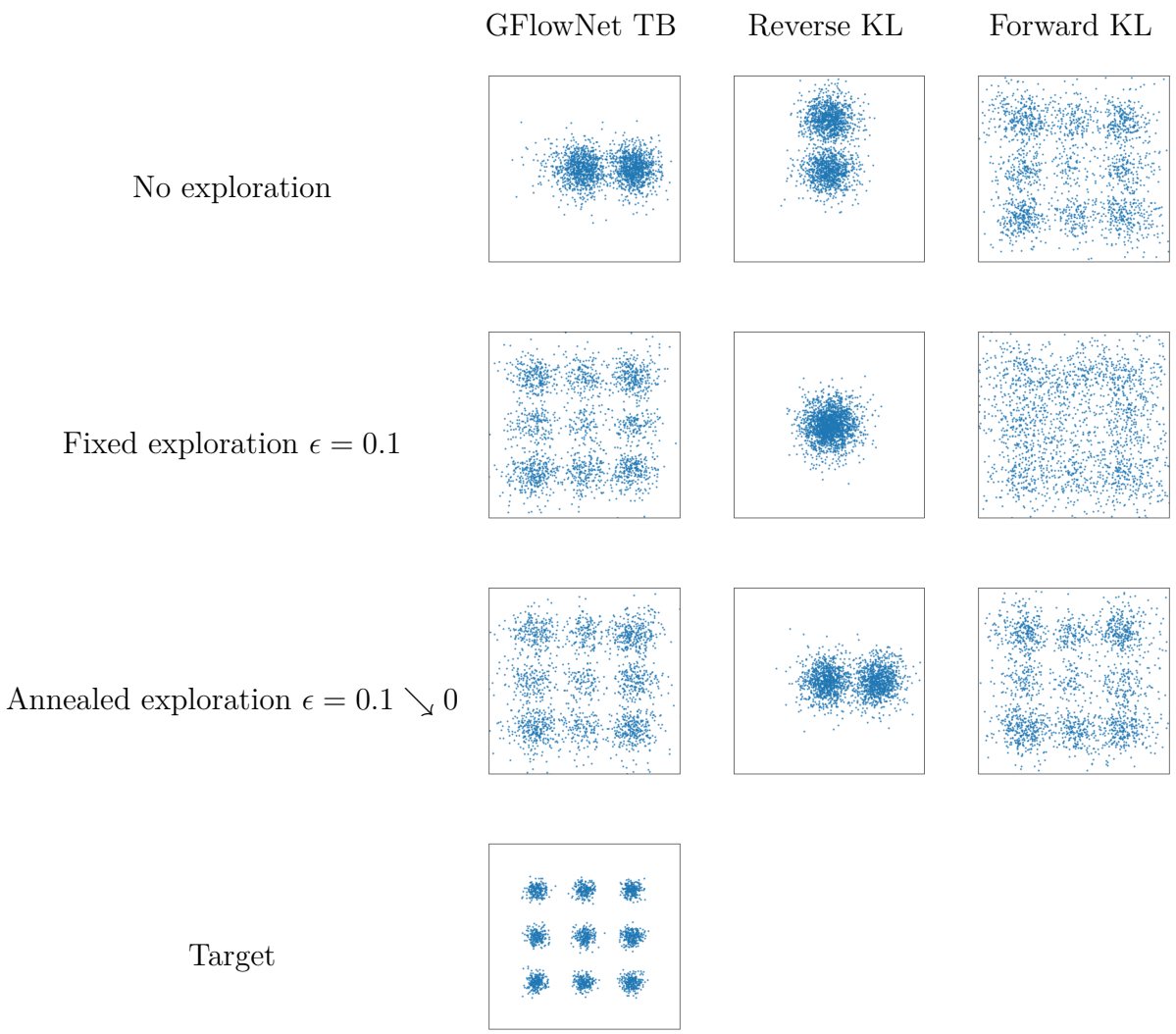


Figure E.1 – The target density for 9 Gaussians and samples from models trained with various algorithms trained for 1500 batches. (When trained longer, GFlowNet TB policies with exploration learn to model the modes with higher precision.)

The simple and importance-weighted estimates of the log-partition function from Zhang and Chen (2022) are defined, in GFlowNet terms, as

$$B = \frac{1}{K}\sum_{i=1}^{K} \log \frac{R(x_T^{(i)}) p_B^{\otimes T}(\tau^{(i)}|x_T^{(i)})}{p_F^{\otimes T}(\tau^{(i)})},$$

$$B_{\text{RW}} = \log \frac{1}{K}\sum_{i=1}^{K} \frac{R(x_T^{(i)}) p_B^{\otimes T}(\tau^{(i)}|x_T^{(i)})}{p_F^{\otimes T}(\tau^{(i)})},$$

where the $\tau^{(i)}$ are K trajectories sampled from P_F, the $x^{(i)}$ are their terminating states, and $p_F^{\otimes T}(\tau^{(i)}), p_B^{\otimes T}(\tau^{(i)}|x_T^{(i)})$ are the products of forward (resp. backward) Gaussian policy densities along the trajectories. Note that both estimates would equal the true integral of the reward

Black box?			Gaussian mixture ($d = 2$)		Funnel ($d = 10$)	
			B	B_{RW}	B	B_{RW}
✓	Off-policy	GFlowNet TB	$\mathbf{-0.150} \pm 0.019$	$\mathbf{-0.003} \pm 0.011$	$\mathbf{-0.219} \pm 0.020$	$\mathbf{-0.026} \pm 0.020$
✓	Off-policy	Reverse KL*	-1.706 ± 0.537	-1.609 ± 0.546	–	–
✓	Off-policy	Forward KL*	-0.306 ± 0.036	$\mathbf{-0.001} \pm 0.013$	-2.822 ± 0.576	-0.087 ± 0.081
✓	On-policy	GFlowNet TB	-1.409 ± 0.427	-1.301 ± 0.434	-0.265 ± 0.026	$\mathbf{-0.012} \pm 0.108$
✓	On-policy	Reverse KL	-1.348 ± 0.397	-1.237 ± 0.413	-0.259 ± 0.018	-0.040 ± 0.023
✓	On-policy	Forward KL*	-0.254 ± 0.032	-0.007 ± 0.023	-1.384 ± 0.284	-0.034 ± 0.143
✓	Non-SDE	SMC	-0.362 ± 0.293		-0.216 ± 0.157	
×	On-policy	PIS-NN	-1.691 ± 0.370	-1.192 ± 0.482	-0.098 ± 0.005	-0.018 ± 0.020
×	Non-SDE	HMC	-1.876 ± 0.527		-0.835 ± 0.257	

Table E.1 – Estimation bias of the log-partition function using simple (B) and importance-weighted (B_{RW}) bounds (mean and standard deviation over 10 runs). The **bold** value in each column shows the best result and all those statistically equivalent to it ($p > 0.1$ under a Welch's t-test). Algorithms assuming access to the gradient of the reward (non-black-box) are shown for comparison. Rows marked with * require importance weighting for gradient estimation. Cells with – were unstable to optimize. Last three rows from Zhang and Chen (2022).

density for a perfect sampler. Identically to Zhang and Chen (2022), we use $K = 2000$ for the 2-dimensional Gaussian mixture dataset and $K = 6000$ for the 10-dimensional funnel dataset.

We show extended results, including both simple and importance-weighted variational bounds, in Table E.1.

E.2.3. Stochastic control on a torus environment

The transition kernel $\kappa(s, -)$ for any $s = (\tilde{s}, t)$ when $t < T$ is the product of the Lebesgue measure on $[0, 2\pi)^2$ with the Dirac measure at $t + 1$, and $\kappa((\tilde{s}, T), -) = \delta_\perp$ for every $\tilde{s} \in [0, 2\pi)^2$. Similar to Section 4.4.3, the reference measure is $\nu = \delta_{s_0} + \sum_{t=1}^{T} \lambda \otimes \delta_t$, where λ is the Lebesgue measure on each copy of the torus $[0, 2\pi)^2$.

We parameterized the densities p_F and p_B with mixtures of independent von Mises distributions defined by a measure of location μ and a measure of concentration κ. We considered two tasks in the torus environment defined by different reward functions.

Synthetic multimodal task. For this task, we designed a reward density with six modes on the torus surface:

$$R_6(\psi, \varphi) = (\sin(3\psi) + \cos(2\varphi) + 2)^3.$$

Molecule conformation task. In this task, we define the reward function using the energy $\mathcal{E}$ of an alanine dipeptide molecule, which depends on the conformation of the molecule $\mathcal{C}$ (spatial arrangement of its atoms). This conformation can be efficiently parameterized using internal coordinates: bond lengths, bond angles, and torsion angles (Jing et al., 2022; Thiede

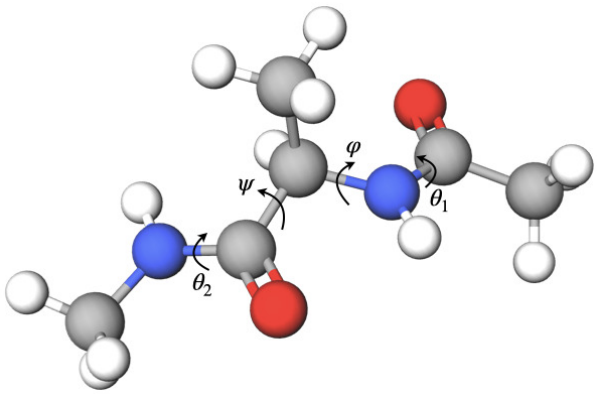


Figure E.2 – Alanine dipeptide 3D structure. Torsion angles ψ, φ, θ_1, θ_2 have the biggest impact on the energy of the molecule. A pair of torsion angles φ and ψ can take any values $\in [0, 2\pi]$, while θ_1 and θ_2 can be either close to 0 or π due to energy barriers (Mironov et al., 2018). The image is rendered using MolView Bergwerf (2014)

et al., 2022). For alanine dipeptide, there are four torsion angles largely influencing the energy (see Figure E.2). In our experiments, the GFlowNet generates values for the angles ψ and φ while keeping all other coordinates fixed. In this way, the support of the reward function remains a torus, and its values are proportional to the Boltzmann distribution with energy $\mathcal{E}$:

$$R_{AD}(\psi, \varphi) = \exp(-\mathcal{E}(\mathcal{C}(\psi, \varphi))),$$

The plots in Figure 4.4 show the results of training a GFlowNet on a toroidal space with a continuous synthetic multimodal reward function R_6 (see text) and a reward function defined by the Boltzmann distribution of the alanine dipeptide molecule R_{AD}. The images represent the density over a discretization of the space $[0, 2\pi)^2$, obtained after fitting KDE with 100,000 samples. The samples to fit the reward densities (Figure 4.4(**a-c**)) were obtained via rejection sampling, and the GFlowNet densities (Figure 4.4**b-d**) use GFlowNet samples from the learned distribution over terminating states $P_\top$.

Results. To evaluate the performance of the GFlowNet trained with the TB loss, we calculated the Jensen-Shannon divergence (see Section 4.4.1 for details about its estimation) between the learned terminating state distribution $P_\top$ and the normalized reward distribution. We provide a visual representation of learned and reward distributions in Figure 4.4. Quantitatively, the GFlowNet achieved a JSD of 0.063 for the synthetic multimodal task and 0.009 for the molecule conformation task. These results show that the generalized GFlowNet can model probability densities over non-Euclidean spaces.

Hyperparameters. We modeled both p_F and p_B with 5-layer perceptrons with 512 hidden units per layer, training the full set of parameters of each model separately. These models output, for each angle ψ and φ, the location μ_i and concentration κ_i of 5 independent von Mises distributions, mixed with learned weights w_i. To take into account the topology of the torus, we encoded input angles with trigonometric transformations ($\sin(k\psi)$, $\cos(k\psi)$, $k = 1, \ldots, 5$, for both angles ψ and φ). We used a learning rate of 10^{-5} for the model parameters and 10^{-2} for $\log Z$, updating the parameters with batches of 100 trajectories of

length $T = 10$. With the synthetic reward, the model converged in about 5,000 iterations; in the molecular conformation task, we trained for 40k iterations.

E.2.4. Posterior over continuous parameters in Bayesian structure learning

Bayesian Networks. Recall that a Bayesian Network is a probabilistic model, where the joint distribution over d random variables $\{X_1, \dots, X_d\}$ factorizes according to a directed acyclic graph (DAG) G as:

$$P(X_1, \dots, X_d; \theta, G) = \prod_{i=1}^{d} P(X_i \mid \mathrm{Pa}_G(X_i); \theta_i),$$

where $\mathrm{Pa}_G(X_i)$ is the set of parents of X_i in G, and θ_i is the set of parameters for the conditional probability distribution of X_i. We denote by $\theta = \{\theta_1, \dots, \theta_d\}$ the set of all the parameters of this model.

We assume that $\theta \in \Theta_G$, where Θ_G is the space of all parameters for the Bayesian Network. Note that this space of parameters depends on the structure G of the Bayesian Network. For example, the Bayesian Network where all the random variables are mutually independent (corresponding to G being empty) has fewer parameters than another Bayesian Network that encodes dependencies between those random variables. We will also denote by $\mathcal{G}$ the space of DAG over d nodes; the number of elements in this space grows super-exponentially with d.

Linear Gaussian model. In order to compute the exact posterior distribution $P(G, \theta \mid \mathcal{D})$ in closed-form, we consider here a linear-Gaussian model for the parameterization of the conditional probability distributions appearing in the Bayesian Network. More precisely, the conditional probability distribution is given by

$$P(X_i \mid \mathrm{Pa}_G(X_i); \theta_i) = \mathcal{N}(X_i \mid \mu_i, \sigma^2) \qquad \text{where} \qquad \mu_i = \sum_{X_j \in \mathrm{Pa}_G(X_i)} \theta_{ij} X_j.$$

In other words, X_i follows a Normal distribution, whose mean μ_i is given by a linear combination of its parents, and with fixed variance σ^2. For this class of models,

$$\Theta_G \simeq \bigtimes_{i=1}^{d} \mathbb{R}^{|\mathrm{Pa}_G(X_i)|}$$

GFlowNet over a mixed state space. We are using the GFlowNet in order to approximate the joint posterior distribution $P(G, \theta \mid \mathcal{D})$, and therefore its terminating states have the form (G, θ), where $G \in \mathcal{G}$ is a DAG, and $\theta \in \Theta_G$ are the associated parameters. Unlike Nishikawa-Toomey et al. (2022), which uses Variational Bayes to update the distribution over parameters θ, we model the distribution over both the graphs and parameters using a single GFlowNet.

The generation of a terminating state follows 2 phases: during the first phase, the DAG G is constructed by adding one edge at a time, starting from the empty graph, following the structure of DAG-GFlowNet (Deleu et al., 2022). To reach a graph G with k edges, we therefore are taking k steps in the GFlowNet. The states traversed during this first phase have no parameters associated to them; we denote by $(G, \sharp) \in \mathcal{S}$ such an (intermediate) state, where $\sharp \notin \Theta_{G'}$ for any $G' \in \mathcal{G}$ is a symbol indicating that G has no corresponding parameters.

Then once we have finished adding edges (in practice, this decision is made by selecting a special "stop" action), we sample the parameters $\theta \in \Theta_G$ associated to G by taking a final step in the GFlowNet to reach the terminating state $(G, \theta) \in \mathcal{X}$. Since the space of parameters depends on the graph G, we define the state space of the GFlowNet as

$$\mathcal{S} = \bigcup_{G \in \mathcal{G}} \{G\} \times \bar{\Theta}_G \qquad \text{and} \qquad \mathcal{X} = \bigcup_{G \in \mathcal{G}} \{G\} \times \Theta_G,$$

where $\bar{\Theta}_G = \Theta_G \cup \{\sharp\}$ indicates the space of parameters, augmented with the special symbol $\sharp$. All the states in this state space are guaranteed to be accessible from the initial state $(G_0, \sharp)$, where G_0 is the empty graph.

Reference kernel. Given a DAG G, the measure $\kappa((G, \sharp), -)$ is the sum of a discrete measure (to transition to another intermediate state $(G', \sharp)$) and a continuous measure (to transition to a terminating state (G, θ)). We can write this measure as

$$\kappa((G, \sharp), -) = \sum_{G' \in \mathrm{Ch}(G)} \delta_{(G', \sharp)} + (\delta_G \otimes \lambda_{\Theta_G}),$$

where $\mathrm{Ch}(G)$ represents the children of G in DAG-GFlowNet (Deleu et al., 2022), i.e., the graphs G' obtained by adding an edge to G, and λ_{Θ_G} is the Lebesgue measure over Θ_G. Moreover, we also have $\kappa((G, \theta), -) = \delta_\perp$ for all terminating state $(G, \theta) \in \mathcal{X}$; in other words, there is no transition from a terminating state other than to the sink state.

The backward reference kernel κ^b on the other hand is simpler: it is always a discrete transition kernel, regardless of the state. We have

$$\kappa^b((G, \theta), -) = \delta_{(G, \sharp)} \qquad \text{and} \qquad \kappa^b((G', \sharp), -) = \sum_{G \in \mathrm{Pa}(G')} \delta_{(G, \sharp)},$$

where $\mathrm{Pa}(G')$ are the parents of G' in DAG-GFlowNet, i.e., they are the graphs obtained by removing a single edge from G'.

Forward transition probability. In order to define the P_F, we consider 2 cases: either we have a distribution of the form $P_F(G' \mid G)$, where G' is the result of adding an edge to G, or a distribution of the form $P_F(\theta \mid G)$. Note that here we are using a slight abuse of notation, where $P_F(G' \mid G)$ (resp. $P_F(\theta \mid G)$) represents $P_F((G', \sharp) \mid (G, \sharp))$ (resp. $P_F((G, \theta) \mid (G, \sharp))$). The distribution $P_F(\theta \mid G)$ is parameterized by a Normal distribution, whose mean and

(diagonal) covariance are returned by a neural network. Similar to (Deleu et al., 2022), P_B is fixed to the uniform distribution.

Data generation. We sampled a dataset $\mathcal{D}$ as follows: (1) we first generated a DAG G^* from an Erdös-Renyi model, then (2) we sampled the parameters θ^* of the conditional probability distributions from a Normal distribution, each edge having a weight $\theta^*_{ij} \sim \mathcal{N}(0, 1)$, and finally (3) we sampled $N = 100$ datapoints from the Bayesian Network described above with (G^*, θ^*), using ancestral sampling. Note that the ground-truths G^* and θ^* are unknown to the GFlowNet, and it only uses the observations from $\mathcal{D}$.

Evaluations. To evaluate the quality of the approximation learned by the GFlowNet, and to compare it against the baseline methods based on variational inference (Lorch et al., 2021; Cundy et al., 2021), we study the distribution over graphs (discrete component) and the distribution over graphs (continuous component) separately. Recall that since we assume that our model is linear-Gaussian over small graphs ($d \leq 5$), we can compute the exact posterior distribution $P(G, \theta \mid \mathcal{D})$ in closed form.

In Section 4.4.5, we compared the edge marginals estimated using the posterior approximations to the exact edge marginals. For any pair of random variables (X_i, X_j), this means evaluating the following marginals

$$P(X_i \to X_j \mid \mathcal{D}) = \sum_{G \mid X_i \in \mathrm{Pa}_G(X_j)} P(G \mid \mathcal{D}).$$

To estimate this marginal using samples $\{(G_k, \theta_k)\}_{k=1}^K$ from the GFlowNet (or from the variational inference methods), we can simply use the sample graphs G_k in order to get an empirical approximation of the maginal posterior $P(G \mid \mathcal{D})$. In other words,

$$\hat{P}(X_i \to X_j) = \frac{1}{K} \sum_{k=1}^{K} \mathbf{1}(X_i \to X_j \in G_k)$$

We then report the root mean-square error (RMSE) between the edge marginals estimated using the posterior approximations, and those computed using the exact posterior:

$$\mathrm{RMSE}(\hat{P}, P) = \left(\frac{1}{d(d-1)} \sum_{i \neq j} \left(\hat{P}(X_i \to X_j) - P(X_i \to X_j \mid \mathcal{D}) \right)^2 \right)^{1/2}.$$

In Appendix E.2.4, we also report the RMSE for other marginals: the marginal of having a directed path between two nodes $P(X_i \rightsquigarrow X_j \mid \mathcal{D})$, as well as the marginal of node X_i being in the Markov blanket of X_j $P(X_i \in \mathrm{MarkovBlanket}(X_j) \mid \mathcal{D})$. In addition to the RMSE between those marginals, we also report the Pearson correlation coefficient, as in (Deleu et al., 2022). Note that no metric is reported on graphs over $d = 3$ nodes for BCD Nets (Cundy et al., 2021) due to technical reasons (the method is not applicable for graphs smaller than 4 nodes).

		RMSE			Pearson's r		
Number of variables (d)		3	4	5	3	4	5
Edges	BCD Nets	–	2.13×10^{-1}	2.61×10^{-1}	–	0.8578	0.7886
	DiBS	3.28×10^{-1}	2.95×10^{-1}	3.15×10^{-1}	0.6903	0.7085	0.7170
	GFlowNet	$\mathbf{1.50 \times 10^{-2}}$	$\mathbf{1.61 \times 10^{-2}}$	$\mathbf{1.80 \times 10^{-2}}$	**0.9993**	**0.9990**	**0.9990**
Paths	BCD Nets	–	2.59×10^{-1}	3.08×10^{-1}	–	0.8378	0.7500
	DiBS	3.50×10^{-1}	3.35×10^{-1}	3.48×10^{-1}	0.6951	0.7080	0.7020
	GFlowNet	$\mathbf{3.39 \times 10^{-3}}$	$\mathbf{1.07 \times 10^{-2}}$	$\mathbf{1.99 \times 10^{-2}}$	**1.0000**	**0.9996**	**0.9989**
Markov blanket	BCD Nets	–	3.02×10^{-1}	3.49×10^{-1}	–	0.8831	0.7864
	DiBS	3.88×10^{-1}	3.80×10^{-1}	4.45×10^{-1}	0.7840	0.7892	0.6888
	GFlowNet	$\mathbf{2.14 \times 10^{-2}}$	$\mathbf{2.38 \times 10^{-2}}$	$\mathbf{2.83 \times 10^{-2}}$	**0.9986**	**0.9982**	**0.9980**

Table E.2 – Comparison between GFlowNets and other methods based on variational inference on the Bayesian structure learning task, for different marginals of interest of the distribution over graphs $P(G \mid \mathcal{D})$.

To evaluate the accuracy of the approximation on the continuous part of the distribution, we report in Section 4.4.5 the (average) negative log-probability of the sampled parameters θ_k from the different approximations (GFlowNet, DiBS (Lorch et al., 2021), and BCD Nets (Cundy et al., 2021)) against the exact posterior distribution $P(\theta \mid G, \mathcal{D})$. More precisely, we compute

$$\text{Measure}_\theta(\hat{P}, P) = -\frac{1}{K|\mathcal{D}|} \sum_{k=1}^{K} \log P(\theta_k \mid G_k, \mathcal{D})$$

In other words, the lower this metric is, the more likely the samples θ_k from those approximations are under the exact posterior distribution over parameters.

E.2.5. Connections with diffusion models

We train a diffusion model-specified GFlowNet with $T = 100$ for $200,000$ steps. This is much shorter than other state-of-the-art work (such as Lipman et al. (2022)) and takes less than 3 days on a single V100 GPU. All NLL results are computed in bits per dimension (BPD). We use $50,000$ generated samples to compute the FID score for evaluating the sample quality. The Adam learning rate is 2×10^{-4} for the forward policy and 2×10^{-5} for the backward policy. The parameter of backward policy is $\{\phi_i\}_{i=1}^{T}$, where the variance coefficient β_i satisfies $\beta_i = \bar{\beta}_i \cdot \exp(\phi_i)$ and $\bar{\beta}_i$ is the original variance coefficient used in Ho et al. (2020b). For details about the MLE-GFN algorithm, we refer to Zhang et al. (2023b). Figure E.3 shows examples of images generated by the algorithm.

E.3. Proofs

Lemma 4.3.2.

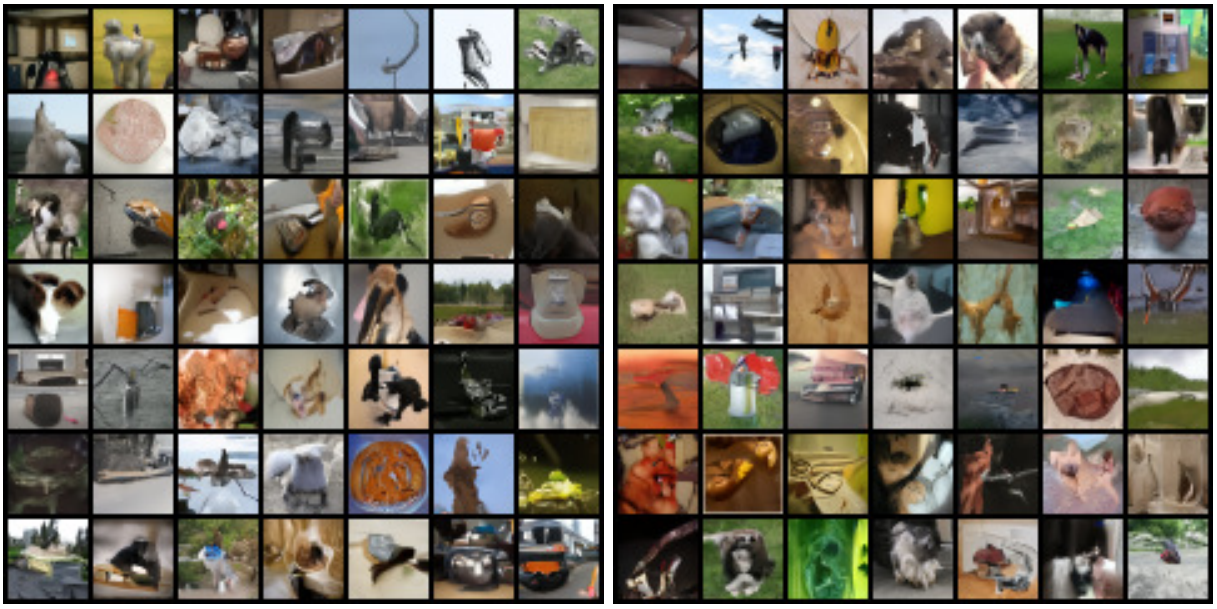


Figure E.3 – Generated samples from MLE-GFN on ImageNet-32 dataset.

PROOF. We prove the lemma by induction on n. The base case ($n = 1$) is trivially satisfied. Assuming the property holds for some $n \geq 0$, let $s \in \bar{\mathcal{S}}$ and $B \in \Sigma$ such that $\kappa^{n+1}(s, B) = 0$.

$$P_F^{n+1}(s, B) = \int_{\bar{\mathcal{S}}} P_F^n(s, ds') P_F(s', B).$$

If $P_F^{n+1}(s, B) > 0$, that would mean there exists an open set $B' \in \mathcal{T}$ such that $P_F^n(s, B') > 0$ and $P_F(s', B) > 0$ for all $s' \in B'$. From the induction hypothesis, it would follow that $\kappa^n(s, B') > 0$ and $\kappa(s', B) > 0$ for all $s' \in B'$, meaning that:

$$\kappa^{n+1}(s, B) = \int_{\bar{\mathcal{S}}} \kappa^n(s, ds')\kappa(s', B) \geq \int_{B'} \kappa^n(s, ds')\kappa(s', B) > 0.$$

A contradiction! Hence, $P_F^{n+1}(s, B) = 0$ □

Lemma 4.3.8.

PROOF. Starting from the definition of $P_\top^n$:

$$\int_{\mathcal{X}} f(x) P_\top^n(dx) = \int_{\bar{\mathcal{S}}} f(s_n) \mathbf{1}_{\mathcal{X}}(s_n) P_\top^n(dx) = \int_{\bar{\mathcal{S}}^{n+1}} \mathbf{1}_{\mathcal{X}}(s_n) f(s_n) P_F^{\otimes n}(s_0, ds_1 \dots ds_{n+1}) \mathbf{1}(s_{n+1} = \perp)$$

Hence, using the recursive definition of $P_F^{\otimes n}$ in (4.14):

$$\begin{aligned}\int_{\mathcal{X}} f(x)P_\top^n(dx) &= \int_{\bar{\mathcal{S}}^{n+1}} \mathbf{1}_{\mathcal{X}}(s_n)f(s_n)P_F^{\otimes n-1}(s_0, ds_1 \dots ds_n)P_F(s_n, ds_{n+1})\mathbf{1}(s_{n+1} = \perp)\\ &= \int_{\bar{\mathcal{S}}^n} \underbrace{\mathbf{1}_{\mathcal{X}}(s_n)f(s_n)P_F(s_n, \{\perp\})}_{g(s_n)} P_F^{\otimes n-1}(s_0, ds_1 \dots ds_n)\\ &= \int_{\bar{\mathcal{S}}^n} g(s_n)P_F^{\otimes n-1}(s_0, ds_1 \dots ds_n)\\ &= \int_{\bar{\mathcal{S}}} g(s_n)P_F^{n-1}(s_0, ds_n) = \int_{\mathcal{X}} f(x)P_F(x, \{\perp\}P_F^{n-1}(s_0, ds_n),\end{aligned}$$

where we applied Lemma 4.3.9 to the bounded and measurable function g. □

Lemma 4.3.9.

Proof. We prove the lemma by induction on n. First, for $n = 0$, using (4.13)

$$\int_{\bar{\mathcal{S}}} f(s')P_F^{\otimes 0}(s, ds') = f(s) = \int_{\bar{\mathcal{S}}} f(s')P_F^0(s, ds').$$

Then, assuming that (4.18) is satisfied for some $n \geq 0$, we get:

$$\begin{aligned}\int_{\bar{\mathcal{S}}^{n+2}} f(s')P_F^{\otimes n+1}(s, ds_1 \dots ds_{n+1}ds') &= \int_{\bar{\mathcal{S}}^{n+2}} f(s')P_F^{\otimes n}(s, ds_1 \dots ds_{n+1})P_F(s_{n+1}, ds')\\ &= \int_{\bar{\mathcal{S}}^{n+1}} \underbrace{\int_{\bar{\mathcal{S}}} f(s')P_F(s_{n+1}, ds')}_{\triangleq g(s_{n+1})} P_F^{\otimes n}(s, ds_1 \dots ds_{n+1})\\ &= \int_{\bar{\mathcal{S}}} g(s_{n+1})P_F^n(s, ds_{n+1}) = \iint_{\bar{\mathcal{S}}\times\bar{\mathcal{S}}} f(s')P_F(s_{n+1}, ds')P_F^n(s, ds_{n+1})\\ &= \int_{\bar{\mathcal{S}}} f(s')P_F^{n+1}(s, ds_{n+1}),\end{aligned}$$

where we applied the inductive hypothesis to a new bounded and measurable[1] function g, and applied the recursive definition of P_F^{n+1}. □

Lemma 4.3.10.

Proof. We will present a proof by contradiction.

Let $\mathcal{N} = \{s \in \bar{\mathcal{S}},\ \kappa(s, \{s_0\}) > 0\}$, and assume that $\mathcal{N} \neq \emptyset$. $(0, \infty)$ being an open set, and $s \mapsto \kappa(s, \{s_0\})$ continuous, this means that $\mathcal{N} \in \bar{\mathcal{T}}$ (i.e., it is open). From (4.5), it follows that there is some $n \geq 0$ such that $\kappa^n(s_0, \mathcal{N}) > 0$.

Applying (4.3), we obtain:

$$\kappa^{n+1}(s_0, \{s_0\}) = \int_{\bar{\mathcal{S}}} \kappa^n(s_0, ds')\kappa(s', \{s_0\}) \geq \int_{\mathcal{N}} \kappa^n(s_0, ds')\kappa(s', \{s_0\}) > 0.$$

1. This can be seen by writing $f = f^+ - f^-$, where f^+, f^- are non-negative, writing each of f^+, f^- as a limit of step functions, and using the monotone convergence theorem.

Writing, for all $m > 1$:

$$\kappa^{m(n+1)}(s_0, \{s_0\}) = \int_{\bar{S}} \kappa^{(m-1)(n+1)}(s_0, ds')\kappa^{n+1}(s', \{s_0\}) \geq \int_{\{s_0\}} \kappa^{(m-1)(n+1)}(s_0, ds')\kappa^{n+1}(s', \{s_0\})$$
$$= \kappa^{(m-1)(n+1)}(s_0, \{s_0\})\kappa^{n+1}(s_0, \{s_0\}),$$

it follows from a simple induction that $\forall m \geq 1$, $\kappa^{m(n+1)}(s_0, \{s_0\}) > 0$, which contradicts (4.11).

$\mathcal{N}$ is thus necessarily empty. □

Lemma 4.3.11.

Proof. Let s be an element in the support of $\kappa^b(\bot, -)$. By definition of the support, it means that for any $B \in \mathcal{T}$ containing s, there is some $B' \subseteq B$ such that $B' \subseteq \mathcal{X}$. In particular, $B \cap \mathcal{X} \neq \emptyset$. This means that s is a point of closure of $\mathcal{X}$.

Conversely, let s be a point of closure of $\mathcal{X}$. Given any open set $B \in \mathcal{T}$ containing s, s being a closure point means that $B \cap \mathcal{X}$ (which is measurable) is non-empty. Following (4.8), we get:

$$\nu(\{\bot\})\kappa^b(\bot, B \cap \mathcal{X}) = \int_{\bar{S}} \mathbf{1}_{B\cap\mathcal{X}}(s')\nu(ds')\kappa(s', \{\bot\})$$

The RHS of the previous equality is positive because ν is a strictly positive measure and $\kappa(s', \{\bot\}) > 0$ for every $s' \in \mathcal{X}$, following Definition 4.3.5. Hence $\kappa^b(\bot, B \cap \mathcal{X}) > 0$. It follows that $\kappa^b(\bot, B) \geq \kappa^b(\bot, B \cap \mathcal{X}) > 0$. Meaning that s is indeed within the support of $\kappa^b(\bot, -)$. □

Theorem 4.3.14.

Proof. Using Lemma 4.3.8, the terminating state measure P_T satisfies for any bounded measurable function $f : \mathcal{X} \to \mathbb{R}$:

$$\int_{\mathcal{X}} f(x)P_T(dx) = \int_{\mathcal{X}} f(x)P_F(x, \bot) \sum_{n=0}^{\infty} P_F^n(s_0, dx).$$

It follows from Proposition 4.3.15 that

$$\mu(\{s_0\}) \int_{\mathcal{X}} f(x)P_T(dx) = \int_{\mathcal{X}} f(x)P_F(x, \bot)\mu(dx).$$

Following Lemma 4.3.16, and the positivity assumption on R that:

$$\int_{\mathcal{X}} f(x)P_T(dx) = \frac{1}{R(\mathcal{X})} \int_{\mathcal{X}} f(x)P_F(x, \bot)\mu(dx).$$

Finally, using (4.20), we obtain:

$$\int_{\mathcal{X}} f(x)P_T(dx) = \frac{1}{R(\mathcal{X})} \int_{\mathcal{X}} f(x)R(dx).$$

P_T being a probability measure follows by applying the last equality to the function $f : x \mapsto 1$. □

Proposition 4.3.15.

Proof. First, using a simple recursion, we show that $\forall n \geq N,\ P_F^n(s_0, -) = \delta_\perp$. The base case ($n = N$) is satisfied as a consequence of Lemma 4.3.2, and the fact that the measurable pointed graph is finitely absorbing. Assuming it holds for some $n \geq N$, going back to the definition of the n-step measure, we have for every $B \in \Sigma$:

$$P_F^{n+1}(s_0, B) = \int_{\bar{\mathcal{S}}} P_F^n(s_0, ds) P_F(s, B) = \int_{\bar{\mathcal{S}}} \delta_\perp(ds) P_F(s, B) = P_F(\perp, B) = \delta_\perp(B),$$

where the last equality stems from the absolute continuity of $P_F(\perp, -)$ with respect to $\kappa(\perp, -)$ and (4.11).

This shows that for every $B \in \Sigma_{|\mathcal{S}}$:

$$u(B) = \sum_{n=0}^{N-1} P_F^n(s_0, B).$$

Which shows the measure u is finite.

Next, we partition $\mathcal{S}$ into N disjoint sets $\mathcal{S}_0, \ldots, \mathcal{S}_{N-1}$, where:

$$s \in \mathcal{S}_n \Leftrightarrow n = \max\{m \in \mathbb{N}_0 :\ \forall B \in \mathcal{T}, s \in B \Rightarrow P_F^m(s_0, B) > 0\}$$

$\mathcal{S}_n \in \Sigma$ given that $\mathcal{S}_n = \mathcal{S}'_n \setminus \bigcup_{k=1}^{\infty} \mathcal{S}'_{n+k}$, where $\mathcal{S}'_n$ is the support of $P_F^n(s_0, -)$, which is known to be a closed set, and hence measurable.

Writing any $B \in \Sigma_{|\mathcal{S}}$ as:

$$B = \bigcup_{n=0}^{N-1} B \cap \mathcal{S}_n,$$

and using the additivity property of the measures u and μ, then proving (4.23) for all $B \in \Sigma_{|\mathcal{S}}$, amounts to proving it for all $B \in \Sigma_{|\mathcal{S}_n}$ for all $n \in \{0, \ldots, N-1\}$, given that the sets $\mathcal{S}_i$ are themselves measurable. We prove this by strong induction on n.

Base case: For $n = 0$, $\mathcal{S}_0 = \{s_0\}$, and $\Sigma_{|\mathcal{S}_0} = \{\{s_0\}\}$.

$$u(\{s_0\}) = P_F^0(s_0, \{s_0\}) = \delta_{s_0}(\{s_0\}) = 1,$$

Hence (4.23) is satisfied for $B = \{s_0\}$.

Induction step: Assume that for some $n \geq 0$, (4.23) is satisfied for all $B \in \Sigma_{|\mathcal{S}_m}$ for all $m \leq n$, and let $B \in \Sigma_{|\mathcal{S}_{n+1}}$.

Define $B' = \{s' \in \mathcal{S} \ :\ P_F(s', B) > 0\}$. We first show that $B' \subseteq \bigcup_{m=0}^{n} \mathcal{S}_m$. If there is $s' \in B'$ and $n' > n$ such that $s' \in \mathcal{S}_{n'}$, then for any open set $\tilde{B}$ (i.e., $\tilde{B} \in \mathcal{T}$) containing s', $P_F^{n'}(s_0, \tilde{B}) > 0$. $\tilde{s} \mapsto P_F(\tilde{s}, B)$ being a continuous function, it follows that B' itself is open.

Hence $P_F^{n'}(s_0, B') > 0$. And noting that:

$$P_F^{n'+1}(s_0, B) = \int_{\bar{\mathcal{S}}} P_F^{n'}(s_0, ds') P_F(s', B) \geq \int_{B'} P_F^{n'}(s_0, ds') P_F(s', B) > 0,$$

which would contradict the fact that $B \subseteq \mathcal{S}_{n+1}$, given that $n' + 1 > n + 1$. This shows that $B' \subseteq \bigcup_{m=0}^{n} \mathcal{S}_m$.

Hence:

$$\begin{aligned}
\mu(\{s_0\})u(B) &= \mu(\{s_0\}) \sum_{m=0}^{n+1} P_F^m(s_0, B) = \mu(\{s_0\}) \underbrace{\delta_{s_0}(B)}_{=0} + \mu(\{s_0\}) \sum_{m=0}^{n} P_F^{m+1}(s_0, B) \\
&= \mu(\{s_0\}) \sum_{m=0}^{n} \int_{B'} P_F^m(s_0, ds') P_F(s', B) = \int_{B'} \mu(\{s_0\}) u(ds') P_F(s', B) \\
&= \int_{B'} \mu(ds') P_F(s', B) = \mu(B),
\end{aligned}$$

where the last two equalities stem from the induction hypothesis and the flow matching conditions respectively. □

Lemma 4.3.16.

Proof. Applying (4.20) to the function $f : x \in \mathcal{X} \mapsto 1$, we get:

$$\int_{\mathcal{X}} R(dx) = \int_{\mathcal{X}} \mu(dx) P_F(x, \{\perp\}). \tag{E.10}$$

Additionally, from (4.12), we get $\forall s \in \mathcal{S} \setminus \mathcal{X},\ \kappa(s, \{\perp\}) = 0$. It follows from the absolute continuity requirements of P_F that $\forall s \in \mathcal{S} \setminus \mathcal{X},\ P_F(s, \{\perp\}) = 0$. Hence:

$$\int_{\mathcal{X}} R(dx) = \int_{\mathcal{S}} \mu(ds) P_F(s, \{\perp\}) = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} \mathbf{1}(s' = \perp) \mu(ds) P_F(s, ds') = \int_{\bar{\mathcal{S}}} \mathbf{1}(s' = \perp) \mu(ds') = \mu(\{\perp\}),$$

where the last line follows from (4.19). This shows that $\mu(\{\perp\}) = R(\mathcal{X})$.

Note that as a consequence of Lemma 4.3.10 and the absolute continuity requirement, a P_F satisfies:

$$\forall s \in \bar{\mathcal{S}}, P_F(s, \{s_0\}) = 0 \tag{E.11}$$

Next, following (4.19) and (E.11), we have:

$$\iint_{\mathcal{S}\times\bar{\mathcal{S}}} \mu(ds) P_F(s, ds') = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} \mathbf{1}(s' \neq s_0) \mu(ds) P_F(s, ds') = \int_{\bar{\mathcal{S}}} \mathbf{1}(s' \neq s_0) \mu(ds') = \mu(\bar{\mathcal{S}}) - \mu(\{s_0\}). \tag{E.12}$$

On the other hand, because each $P_F(s, -)$ is a probability measure on $\bar{\mathcal{S}}$:

$$\iint_{\mathcal{S}\times\bar{\mathcal{S}}} \mu(ds) P_F(s, ds') = \int_{\mathcal{S}} \left(\int_{\bar{\mathcal{S}}} P_F(s, ds') \right) \mu(ds) = \int_{\mathcal{S}} \mu(ds) = \mu(\mathcal{S}) \tag{E.13}$$

Subtracting (E.12) from (E.13), we get:

$$\mu(\{s_0\}) = \mu(\bar{\mathcal{S}}) - \mu(\mathcal{S}) = \mu(\{\perp\}) = R(\mathcal{X})$$

□

Proposition 4.3.18.

PROOF. For any bounded measurable function $f : \bar{\mathcal{S}} \to \mathbb{R}$ satisfying $f(s_0) = 0$, we can define a function $g : \mathcal{S} \times \bar{\mathcal{S}} \to \mathbb{R}$ such that for all $(s, s') \in \mathcal{S} \times \bar{\mathcal{S}}$, $g(s, s') = f(s')$. Note that g satisfies $g(s, s_0) = 0$ for every $s \in \mathcal{S}$. Applying the detailed balance conditions to the function g, we have

$$\iint_{\mathcal{S}\times\bar{\mathcal{S}}} g(s, s')\mu(ds)P_F(s, ds') = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s')\mu(ds)P_F(s, ds')$$

On the other hand, using the RHS of (4.25) in the detailed balance conditions, we get

$$\iint_{\mathcal{S}\times\bar{\mathcal{S}}} g(s, s')\mu(ds')P_B(s', ds) = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s')\mu(ds')P_B(s', ds) = \int_{\bar{\mathcal{S}}} f(s')\mu(ds') \int_{\mathcal{S}} P_B(s', ds),$$

Following (4.6) and the the absolute continuity conditions of P_B with respect to κ^b, we have:

$$\forall s' \in \mathcal{S},\ P_B(s', \{\perp\}) = 0,$$

from which it follows that:

$$\iint_{\mathcal{S}\times\bar{\mathcal{S}}} g(s, s')\mu(ds')P_B(s', ds) = \int_{\bar{\mathcal{S}}} f(s')\mu(ds') \underbrace{\int_{\bar{\mathcal{S}}} P_B(s', ds)}_{=1}$$

This shows that (μ, P_F) satisfy the flow matching conditions. □

Proposition 4.3.20.

PROOF. First we show that μ satisfies the flow-matching condition. for any bounded measurable function $f : \bar{\mathcal{S}} \to \mathbb{R}$ satisfying $f(s_0) = 0$, we have :

$$\begin{aligned}
\iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s')\mu(ds)P_F(s, ds') &= \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s')\mu(s_0) \sum_{n=0}^{\infty} P_F^n(s_0, ds)P_F(s, ds') \\
&= \int_{\bar{\mathcal{S}}} f(s')\mu(s_0) \sum_{n=0}^{\infty} P_F^{n+1}(s_0, ds') \\
&= \int_{\bar{\mathcal{S}}} f(s')\mu(s_0) \sum_{n=0}^{\infty} P_F^{n}(s_0, ds') \qquad \text{(because } f(s_0) = 0\text{)} \\
&= \int_{\bar{\mathcal{S}}} f(s')\mu(ds')
\end{aligned}$$

Now, we will show the reward matching condition. For any bounded measurable function $f : \mathcal{X} \to \mathbb{R}$

$$\begin{aligned}\int_{\mathcal{X}} f(s)\mu(ds)P_F(s,\perp) &= \int_{\mathcal{S}} \mathbf{1}_{\mathcal{X}}(s)f(s)\underbrace{\mu(\{s_0\})}_{=Z}\sum_{n=0}^{\infty} P_F^n(s_0,ds)P_F(s,\perp)\\ &= \sum_{n=0}^{\infty}\int_{\mathcal{S}} \mathbf{1}_{\mathcal{X}}(s)f(s)R(ds)P_B^n(s,\{s_0\})\\ &= \int_{\mathcal{S}} \mathbf{1}_{\mathcal{X}}(s)f(s)R(ds)\sum_{n=0}^{\infty} P_B^n(s,\{s_0\}) = \int_{\mathcal{X}} f(s)R(ds)\end{aligned}$$

where we used Proposition 4.3.22 and Lemma 4.3.21.

□

Lemma 4.3.21.

Proof. Using Lemma 4.3.9, we have:

$$\begin{aligned}Z\int_{\mathcal{S}} f(s)P_F^n(s_0,ds)P_F(s,\{\perp\}) &= Z\int_{\bar{\mathcal{S}}^{n+1}} \mathbf{1}(s\neq\perp)f(s)P_F(s,\{\perp\})P_F^{\otimes n}(s_0,ds'ds_1\dots ds_{n-1}ds)\\ &= Z\int_{\bar{\mathcal{S}}^{n+2}} \mathbf{1}(s\neq\perp,s_{n+1}=\perp)f(s)P_F^{\otimes n+1}(s_0,ds'ds_1\dots ds_{n-1}dsds_{n+1})\\ &= \int_{\bar{\mathcal{S}}^{n+2}} \mathbf{1}(s\neq\perp)\mathbf{1}(s''=s_0)f(s)R(ds)P_B^{\otimes n}(s,ds'ds_{n-1}\dots ds_1,ds'')\\ &= \int_{\bar{\mathcal{S}}} f(s)\mathbf{1}(s\neq\perp)R(ds)\int_{\bar{\mathcal{S}}^{n+1}} \mathbf{1}(s''=s_0)P_B^{\otimes n}(s,ds'ds_{n-1}\dots ds_1,ds'')\\ &= \int_{\bar{\mathcal{S}}} f(s)\mathbf{1}(s\neq\perp)R(ds)\int_{\bar{\mathcal{S}}} \mathbf{1}(s''=s_0)P_B^{n}(s,ds'')\\ &= \int_{\mathcal{S}} f(s)R(ds)P_B^n(s,\{s_0\})\end{aligned}$$

□

Proposition 4.3.22.

Proof. First, using a simple recursion, we show that $\forall n \geq N,\ P_B^n(s,\{s_0\}) = 0$. The base case ($n = N$) is trivially satisfied because the measurable pointed graph has maximal trajectory length N and $P_B(s_0,-)$ is the trivial measure (i.e., it assigns zero to every measurable set), given that it is absolutely continuous with respect to $\kappa^b(s_0,-)$. Assuming it holds for some $n \geq N$, we have:

$$P_B^{n+1}(s,s_0) = \int_{\bar{\mathcal{S}}} P_B(s,ds')P_B^n(s',\{s_0\}) = \int_{\bar{\mathcal{S}}} P_B(s,ds').0 = 0$$

This shows that for every $s \in \mathcal{S}$:

$$P_{B,T}(s) = \sum_{n=0}^{N-1} P_B^n(s,ds_0).$$

Which shows the measure $P_{B,T}$ is finite.

Next, we partition $\mathcal{S}$ into N disjoint sets $\mathcal{S}_0, \dots, \mathcal{S}_{N-1}$, where:

$$s \in \mathcal{S}_n \Leftrightarrow n = \max\{m \in \mathbb{N}_0 : P_B^m(s, \{s_0\}) > 0\}$$

$\mathcal{S}_n \in \Sigma$ given that $\mathcal{S}_n = \mathcal{S}'_n \setminus \bigcup_{k=0}^{\infty} \mathcal{S}'_{n+k}$, where $\mathcal{S}'_n$ is the support of $P_B^n(-, \{s_0\})$, which is known to be a closed set, and hence measurable.

Writing :

$$\mathcal{S} = \bigcup_{n=0}^{N-1} \mathcal{S}_n,$$

Then proving (4.29) for all $s \in \mathcal{S}$, amounts to proving it for all $s \in \mathcal{S}_n$ for all $n \in \{0, \dots, N-1\}$. We prove this by strong induction on n.

Base case: For $n = 0$, $\mathcal{S}_0 = \{s_0\}$, and

$$P_{B,T}(s_0) = P_B^0(s_0, \{s_0\}) = \delta_{s_0}(\{s_0\}) = 1,$$

Hence it is satisfied for n = 0.

Induction step: Assume that for some $n \geq 0$, (4.29) is satisfied for all $s \in \mathcal{S}_m$ for all $m \leq n$, and let $s \in \mathcal{S}_{n+1}$.

Define $B_s = \{s' \in \mathcal{S} : \forall B \in \mathcal{T}, s' \in B \Rightarrow P_B(s, B) > 0\}$. We first show by contradiction that $B_s \subseteq \bigcup_{m=0}^{n} \mathcal{S}_m$. If there is $s' \in B_s$ and $n' > n$ such that $s' \in \mathcal{S}_{n'}$, then $P_B^{n'}(s', \{s_0\}) > 0$, and by continuity of $\tilde{s} \mapsto P_B^{n'}(\tilde{s}, \{s_0\})$, there exists an open set $\tilde{B}$ (i.e., $\tilde{B} \in \mathcal{T}$) containing s' such that $P_B^{n'}(\tilde{s}, \{s_0\}) > 0$ for all $\tilde{s} \in \tilde{B}$. Hence:

$$P_B^{n'+1}(s, \{s_0\}) = \int_{\bar{\mathcal{S}}} P_B(s, ds') P_B^{n'}(s', \{s_0\}) \geq \int_{\tilde{B}} P_B(s, ds') P_B^{n'}(s', \{s_0\}) > 0,$$

which would contradict the fact that $s \in \mathcal{S}_{n+1}$, given that $n' + 1 > n + 1$.

Hence:

$$P_{B,T}(s) = \sum_{m=0}^{n+1} P_B^m(s, \{s_0\}) = \sum_{m=1}^{n+1} P_B^m(s, \{s_0\}) \quad \text{given that } s \neq s_0$$

$$= \sum_{m=1}^{n+1} \int_{s' \in B_s} P_B(s, ds') P_B^{m-1}(s', \{s_0\}) = \int_{s' \in B_s} P_B(s, ds') \underbrace{\sum_{m=0}^{n} P_B^m(s', \{s_0\})}_{=1} = \int_{s' \in B_s} P_B(s, ds') = 1$$

□

Theorem 4.3.24.

Proof. We will first show the result for the flow-matching loss. Let the function $v : \bar{\mathcal{S}} \to \mathbb{R}_+$ be the function, depending on the parameter θ, defined by $\forall s' \in \bar{\mathcal{S}}$:

$$v(s'; \theta) := \int_{\mathcal{S}} u(s; \theta) p_F(s, s'; \theta) \kappa^b(s', ds).$$

If we assume that $L_{FM}(-; \theta) = 0$ ν-almost surely, then we have equivalently $u(-; \theta) = v(-; \theta)$ ν-almost surely. Let $f : \bar{\mathcal{S}} \to \mathbb{R}$ be a bounded measurable function such that $f(s_0) = 0$, we

then have

$$\begin{aligned}\int_{\bar{\mathcal{S}}} f(s')u(s';\theta)\nu(ds') &= \int_{\bar{\mathcal{S}}} f(s')v(s';\theta)\nu(ds') \\ &= \int_{\bar{\mathcal{S}}} f(s') \int_{\mathcal{S}} u(s;\theta)p_F(s,s';\theta)\kappa^b(s',ds)\nu(ds') \\ &= \iint_{\bar{\mathcal{S}}\times\mathcal{S}} f(s')u(s;\theta)p_F(s,s';\theta)\kappa(s,ds')\nu(ds),\end{aligned}$$

where we used the fact that $\nu \otimes \kappa = \nu \otimes \kappa^b$ (from (4.8) in Definition 4.3.3) in the last equality. Replacing the densities (and reference measures) in the equality above with their corresponding measures μ and P_F, we get

$$\int_{\bar{\mathcal{S}}} f(s')\mu(ds') = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s')\mu(ds)P_F(s,ds').$$

Since this equality is valid for any bounded measurable function f satisfying $f(s_0) = 0$, this is the definition of $F = (\mu, P_F)$ satisfying the flow-matching conditions (Definition 4.3.12).

The proof for the detailed balance loss is similar. Let the functions $g : \bar{\mathcal{S}} \times \bar{\mathcal{S}} \to \mathbb{R}_+$ and $h : \bar{\mathcal{S}} \times \bar{\mathcal{S}} \to \mathbb{R}_+$ defined as

$$\begin{aligned}g(s,s';\theta) &:= u(s;\theta)p_F(s,s';\theta) \\ h(s,s';\theta) &:= u(s';\theta)p_B(s',s;\theta).\end{aligned}$$

If $L_{DB}(-;\theta) = 0$ $\nu \otimes \kappa$-almost surely, then we have equivalently $g(-;\theta) = h(-,\theta)$. Let $f : \mathcal{S} \times \bar{\mathcal{S}} \to \mathbb{R}$ be a bounded measurable function such that $f(s, s_0) = 0$ for all $s \in \mathcal{S}$. We have

$$\begin{aligned}&\iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')g(s,s';\theta)(\nu\otimes\kappa)(ds,ds') \\ &\quad= \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')u(s;\theta)p_F(s,s';\theta)(\nu\otimes\kappa)(ds\,ds') \\ &\quad= \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')h(s,s';\theta)(\nu\otimes\kappa)(ds\,ds') \\ &\quad= \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')h(s,s';\theta)(\nu\otimes\kappa^b)(ds'\,ds) \\ &\quad= \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')u(s';\theta)p_B(s',s;\theta)(\nu\otimes\kappa^b)(ds'\,ds),\end{aligned}$$

where we used $\nu \otimes \kappa = \nu \otimes \kappa^b$ in the 3rd inequality. Note that while the equalities between functions are valid $\nu\otimes\kappa$-almost surely over the whole space $\bar{\mathcal{S}}\times\bar{\mathcal{S}}$, we only used the equality restricted to $\mathcal{S} \times \bar{\mathcal{S}}$. Moreover, since u, p_F, and p_B are the densities of the respective measures μ, P_F, and P_B (with respect to the appropriate reference measures), we know that

for $B \in \bar{\Sigma} \otimes \bar{\Sigma}$

$$(\mu \otimes P_F)(B) = \iint_B u(s;\theta) p_F(s, ds')(\nu \otimes \kappa)(ds\, ds')$$
$$(\mu \otimes P_B)(B) = \iint_B u(s';\theta) p_B(s', ds)(\nu \otimes \kappa^b)(ds'\, ds).$$

Replacing these measures in the equality above, we obtain

$$\iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')\mu(ds)P_F(s,ds') = \iint_{\mathcal{S}\times\bar{\mathcal{S}}} f(s,s')\mu(ds')P_B(s',ds).$$

Since this equality is valid for any bounded measurable function f such that $f(s, s_0) = 0$ for all $s \in \mathcal{S}$, this corresponds to (μ, P_F, P_B) satisfying the detailed balance conditions (Definition 4.3.17).

Now, for the trajectory balance loss: for a trajectory $(s, s_1, .., s_{n+1}) \in \mathcal{S}^{n+2}$, we define:

$$p_F^{\otimes n+1}(s, \overrightarrow{s_{1:n+1}}) = p_F(s, s_1, \theta) \prod_{t=1}^{n} p_F(s_t, s_{t+1}, \theta)$$
$$p_B^{\otimes n}(\overrightarrow{s_{n:1}}, s) = p_B(s_1, s, \theta) \prod_{t=1}^{n-1} p_B(s_{t+1}, s_t, \theta)$$

For any bounded measurable function $f : \bar{\mathcal{S}}^{n+2} \to \mathbb{R}$, assuming $L^n_{TB} = 0$ almost surely for every $n \geq 0$:

$$\begin{aligned}
&\int_{\mathcal{S}^{n+2}} Z(\theta) f(s, \overrightarrow{s_{1:n+1}}) \mathbf{1}(s = s_0, s_n \neq \perp, s_{n+1} = \perp) p_F^{\otimes n+1}(s, \overrightarrow{s_{1:n+1}}) \nu \otimes \kappa^{\otimes n+1}(ds\, \overrightarrow{ds_{1:n+1}}) \\
&= \int_{\{s_0\}\times\mathcal{S}^{n-1}\times\mathcal{X}\times\{\perp\}} Z(\theta) f(s, \overrightarrow{s_{1:n+1}}) p_F^{\otimes n+1}(s, \overrightarrow{s_{1:n+1}}) \nu \otimes \kappa^{\otimes n+1}(ds\, \overrightarrow{ds_{1:n+1}}) \\
&= \int_{\{s_0\}\times\mathcal{S}^{n}\times\{\perp\}} f(s, \overrightarrow{s_{1:n+1}}) r(s_n) p_B^{\otimes n}(\overrightarrow{s_{n:1}}, s) \nu \otimes \kappa^{\otimes n+1}(ds\, \overrightarrow{ds_{1:n+1}}) \\
&= \int_{\{s_0\}\times\mathcal{S}^{n}} f(s, \overrightarrow{s_{1:n}}, \perp) r(s_n) \underbrace{\kappa(s_n, \{\perp\})}_{=1 \text{ (see (4.10))}} p_B^{\otimes n}(\overrightarrow{s_{n:1}}, s) \nu \otimes \kappa^{\otimes n}(ds\, \overrightarrow{ds_{1:n}}) \\
&= \int_{\{s_0\}\times\mathcal{S}^{n}} f(s, \overrightarrow{s_{1:n}}, \perp) r(s_n) p_B^{\otimes n}(\overrightarrow{s_{n:1}}, s) \nu \otimes \kappa^{b,\otimes n}(\overrightarrow{ds_{n:1}}\, ds) \\
&= \int_{\mathcal{S}^{n+1}} f(s, \overrightarrow{s_{1:n}}, \perp) \mathbf{1}(s = s_0) r(s_n) \nu(ds_n) p_B^{\otimes n}(\overrightarrow{s_{n:1}}, s) \kappa^{b,\otimes n}(s_n, \overrightarrow{ds_{n-1:1}}\, ds)
\end{aligned}$$

Replacing the measures in the last equality obtained, we recover the TB condition in Definition 4.3.19 with $(Z\nu(\{s_0\}), P_F, P_B)$.

Here, we used $\nu \otimes \kappa^{\otimes n} = \nu \otimes \kappa^{b,\otimes n}$, $\forall n \in \{0, ..N\}$. We can show this by simple induction : for $n = 0$, it is trivially satisfied . Now suppose it is true for a given $n \leq N - 1$, using (4.8). We have :

$$\begin{aligned}
\nu \otimes \kappa^{\otimes n+1}(ds\, \overrightarrow{ds_{1:n+1}}) &= \nu \otimes \kappa^{\otimes n}(ds\, \overrightarrow{ds_{1:n}})\kappa(s_n, ds_{n+1}) \\
&= \nu \otimes \kappa^{b,\otimes n}(\overrightarrow{ds_{n:1}}\, ds)\kappa(s_n, ds_{n+1}) \\
&= \nu(ds_n)\kappa(s_n, ds_{n+1})\kappa^{b,\otimes n}(s_n, \overrightarrow{ds_{n:1}}\, ds) \\
&= \nu(ds_{n+1})\kappa^{b}(s_{n+1}, ds_n)\kappa^{b,\otimes n}(s_n, \overrightarrow{ds_{n:1}}\, ds) \\
&= \nu(ds_{n+1})\kappa^{b,\otimes n+1}(s_{n+1}, \overrightarrow{ds_{n:1}}\, ds) \\
&= \nu \otimes \kappa^{b,\otimes n+1}(\overrightarrow{ds_{n+1:1}}\, ds)
\end{aligned}$$

Which proves the claim above.

□

Proposition E.1.2.

Proof. First, note that (4.6) and (4.11) imply that $\forall n \geq N,\ \kappa^n(s_0, -) = \delta_\perp$. This can be shown by a simple induction on n, writing for any $B \in \Sigma$:

$$\kappa^{n+1}(s_0, B) = \int_{\bar{\mathcal{S}}} \kappa^n(s_0, ds)\kappa(ds, B).$$

This entails that (4.5) could be rewritten as:

$$\forall B \in \mathcal{T} \setminus \{\emptyset\},\ \exists n \in \{0, \dots, N\}\ :\ \kappa^n(s_0, B) > 0.$$

Hence,

$$\forall B \in \mathcal{T} \setminus \{\emptyset\},\ \nu(B) > 0.$$

The measure ν is thus strictly positive.

Note that for any B in $\Sigma_{|\mathcal{S}}$ such that $s_0 \notin B$.

$$\begin{aligned}
\nu\kappa(B) &= \int_{\bar{\mathcal{S}}} \nu(ds)\kappa(s, B) \\
&= \int_{\bar{\mathcal{S}}} \sum_{n=0}^{N} \kappa^n(s_0, ds)\kappa(s, B) = \sum_{n=0}^{N} \int_{\bar{\mathcal{S}}} \kappa^n(s_0, ds)\kappa(s, B) \\
&= \sum_{n=0}^{N} \kappa^{n+1}(s_0, B) = \sum_{n=1}^{N+1} \kappa^n(s_0, B).
\end{aligned}$$

Because $B \subseteq \mathcal{S}$, then $\kappa^{N+1}(s_0, B) = \delta_\perp(B) = 0$. And because $s_0 \notin B$, $\kappa^0(s_0, B) = \delta_{s_0}(B) = 0$. From this it follows that:

$$\nu\kappa(B) = \nu(B)$$

Then, let $B \in \Sigma_{|\mathcal{S}} \otimes \Sigma_{|\mathcal{S}}$ such that $(s_0, s_0) \notin B$. Using the definition of the reverse kernel, we obtain:

$$\begin{aligned}
\int_{\bar{\mathcal{S}}\times\bar{\mathcal{S}}} \mathbf{1}_B(s,s')\nu(ds)\kappa(ds,ds') &= \int_{\bar{\mathcal{S}}\times\bar{\mathcal{S}}} \mathbf{1}_B(s,s')\nu\kappa(ds')\kappa^r_\nu(ds',ds) \\
&= \int_{\mathcal{S}\times(\mathcal{S}\setminus\{s_0\})} \mathbf{1}_B(s,s') \underbrace{\nu\kappa(ds')}_{=\nu(ds)} \kappa^r_\nu(ds',ds) + \int_{\mathcal{S}} \mathbf{1}_B(s,s_0) \underbrace{\nu\kappa(\{s_0\})}_{=0} \kappa^r_\nu(s_0,ds) \\
&= \int_{\mathcal{S}\times(\mathcal{S}\setminus\{s_0\})} \mathbf{1}_B(s,s')\nu(ds')\kappa^r_\nu(ds',ds) \\
&= \int_{\mathcal{S}\times(\mathcal{S}\setminus\{s_0\})} \mathbf{1}_B(s,s')\nu(ds')\kappa^r_\nu(ds',ds) + \int_{\mathcal{S}\setminus\{s_0\}} \mathbf{1}_B(s,s_0)\kappa(s_0) \underbrace{\kappa^r_\nu(s_0,ds)}_{=0} \\
&= \int_{\bar{\mathcal{S}}\times\bar{\mathcal{S}}} \mathbf{1}_B(s,s')\nu(ds')\kappa^r_\nu(ds',ds)
\end{aligned}$$

Finally, if $B \in \Sigma_{|\mathcal{S}}$:

$$\begin{aligned}
\int_{\mathcal{S}} \mathbf{1}_B(ds)\nu(ds)\kappa(s,\{\perp\}) &= \int_{\mathcal{S}} \mathbf{1}_B(ds)\nu(\{\perp\})\kappa^b(\perp,ds) \\
\nu(\{\perp\}) \underbrace{\kappa(\perp,B)}_{=0} &= \int_{\mathcal{S}} \mathbf{1}_B(s')\nu(ds') \underbrace{\kappa^b(s',\{\perp\})}_{=0}
\end{aligned}$$

□

Lemma E.1.3.

PROOF. Let $f : \bar{\mathcal{S}}^2 \to \mathbb{R}$ be a bounded measurable function.

$$\begin{aligned}
\int_{\bar{\mathcal{S}}^2} f(s,s')\nu\kappa(ds')\kappa'(s',ds) &= \int_{\bar{\mathcal{S}}} f(s,s_0) \underbrace{\nu\kappa(\{s_0\})}_{=0} \kappa'(s_0,ds) + \int_{\mathcal{S}} \int_{\bar{\mathcal{S}}\setminus\{s_0\}} f(s,s')\nu\kappa(ds')\kappa'(s',ds) \\
&\quad + \int_{\mathcal{S}\setminus\{s_0\}} f(\perp,s')\nu\kappa(ds') \underbrace{\kappa'(s',\{\perp\})}_{=0} + f(\perp,\perp)\nu\kappa(\{\perp\})\kappa'(\perp,\{\perp\}) \\
&= \int_{\mathcal{S}} \int_{\bar{\mathcal{S}}\setminus\{s_0\}} f(s,s')\nu\kappa(ds')\kappa^b(s',ds) + \int_{\bar{\mathcal{S}}} f(s,s_0) \underbrace{\nu\kappa(\{s_0\})}_{=0} \kappa^b(s_0,ds) \\
&\quad + f(\perp,\perp)\nu\kappa(\{\perp\})\kappa^b(\perp,\{\perp\}) \qquad \text{(E.14)}
\end{aligned}$$

On the other hand, let B be the largest open set within $\mathcal{S}$such that $\forall s' \in B,\ \kappa^b(s',\{\perp\}) > 0$. Applying the definition of the reverse kernel (E.1) to the function $f : (s,s') \mapsto \mathbf{1}(s = \perp)\mathbf{1}_B(s')$, we get:

$$\int_{\bar{\mathcal{S}}} \mathbf{1}_B(s')\nu(\{\perp\})\kappa(\perp,s') = \int_{\bar{\mathcal{S}}} \mathbf{1}_B(s')\nu\kappa(s')\kappa^b(s',\{\perp\})$$

The LHS of the previous equality is 0, following (4.6). It follows from the assumption that $\forall s' \in B,\ \kappa^b(s', \{\perp\}) > 0$ that $\nu\kappa(B) = 0$. Hence:

$$\begin{aligned}
&\int_{\mathcal{S}\setminus\{s_0\}} f(\perp, s')\nu\kappa(ds')\kappa^b(s', \{\perp\}) \\
&\qquad = \int_{\bar{\mathcal{S}}\setminus\{s_0\}} \mathbf{1}_{\mathcal{S}\setminus B}(s')f(\perp, s')\nu\kappa(ds')\kappa^b(s', \{\perp\}) \\
&\qquad\quad + \int_{\bar{\mathcal{S}}\setminus\{s_0\}} \mathbf{1}_{B}(s')f(\perp, s')\nu\kappa(ds')\kappa^b(s', \{\perp\})
\end{aligned}$$

The first summand of the RHS of the last equality is zero by the definition of B. The second summand is zero because $\nu\kappa(B) = 0$. Going back to (E.14), we obtain:

$$\int_{\bar{\mathcal{S}}^2} f(s, s')\nu\kappa(ds')\kappa'(s', ds) = \int_{\bar{\mathcal{S}}^2} f(s, s')\nu\kappa(ds')\kappa^b(s', ds) = \int_{\bar{\mathcal{S}}^2} f(s, s')\kappa(ds)\kappa(s, ds')$$

□

Appendix F

Appendix for Chapter 5

F.1. Sequential Model Optimization Experiments

We used a GP for the DEUP uncertainty estimator. Using a neural net provided similar results but was computationally more expensive in this 1-D case with few data points. We used a 3-hidden layer neural network, with 128 neurons per layer and a ReLU activation function, with Adam (Kingma and Ba, 2015) and a learning rate of 10^{-3} (and default values for the other hyperparameters) to train the main predictor for DEUP-EI (in order to fit the available data). The Dropout and Ensemble baselines used the same network architecture and learning rate. We used three networks for the Ensemble baseline and a dropout probability of 0.3 for the Dropout baseline, with 100 test-time forward passes to compute uncertainty estimates.

For the TurBO baseline, we use BoTorch's default implementation, with Expected Improvement as an acquisition function and a batch size of 1 (i.e., acquiring one point per step).

F.2. Reinforcement Learning Experiments

For RL experiments, we used *bsuite* (Osband et al., 2020), a collection of carefully designed RL environments. *bsuite* also comes with a list of metrics that evaluate RL agents from different aspects. We compare the agents based on the *basic* metric and average regret as they capture both sample complexity and final performance. The default DQN agent is used as the base of our experiments with a three-layer fully connected (FC) neural network as its Q-network. For the Bootstrapped DQN baseline, we used the default implementation provided by *bsuite*. To implement DQN + MC-Dropout, following the implementation from Gal and Ghahramani (2016b), two dropout layers with a dropout probability of 0.1 are used before the second and the third FC layers. In order to take an action, the agent performs a

single stochastic forward pass through the Q-network, which is equivalent to taking a sample from the posterior over the Q-values, as done in Thompson sampling, an alternative to $\epsilon-$greedy exploration. The pseudo-code for DEUP-DQN is provided in Algorithm 6.

As a density estimator, we used a Kernel Density Estimator (KDE) with a Gaussian kernel and bandwidth of 1 to map states to densities. This KDE is fit after every 10000 steps (actions) with a batch of samples from the replay buffer (which is of size 10000). The uncertainty estimator network (E-network) has the same number of layers as the Q-network, with an additional Softplus layer at the end. All other hyperparameters are the same as the default implementation by Osband et al. (2020). One complete training run for the DEUP-DQN with five seeds experiments takes about 0.04-0.05 GPU days on a V100 GPU. In total RL experiments took about 0.15 GPU days on a Nvidia V100 GPU.

F.3. Rejecting Difficult Examples

We adapt the standard OOD rejection task (van Amersfoort et al., 2020; Liu et al., 2020) and measure the Spearman Rank Correlation of the predicted uncertainty with the actual generalization error, in addition to the OOD Detection AUROC. MC-Dropout and Deep Ensemble baselines are based on https://github.com/google/uncertainty-baselines, DUQ based on https://github.com/y0ast/deterministic-uncertainty-quantification and DUE based on https://github.com/y0ast/DUE. Note that for the ResNet50 DEUP model, we continue using the ResNet-18-based DUE as a variance source.

Model	**ResNet-50**
MC-Dropout	0.312 ± 0.003
Deep Ensemble	0.401 ± 0.004
DUQ	0.399 ± 0.003
DEUP (D+V)	$\mathbf{0.465 \pm 0.002}$

Table F.1 – Spearman Rank Correlation between predicted uncertainty and the true generalization error on OOD data (SVHN) with ResNet-50 models (3 seeds) trained on CIFAR-10.

Training. The baselines were trained with the CIFAR-10 training set with 10% set aside as a validation set for hyperparameter tuning. The hyperparameters are presented in Table F.2 and Table F.3. The hyperparameters not specified are set to the default values. For DEUP, we consider the log-density, model-variance estimate and the seen-unseen bit as the features for the error predictor. The density estimator we use is Masked-Autoregressive Flows (Papamakarios et al., 2017), and the variance estimator used is DUE (van Amersfoort et al., 2021). Note that, as indicated earlier, x, the input image, is not used as a feature for the error predictor. We present those ablations in the next sub-section. For training DEUP, the CIFAR-10 training set is divided into five folds, each containing eight unique classes. For each fold, we train an instance of the main predictor, density estimator and

Algorithm 6 DEUP-DQN algorithm

Input: Environment; ϵ, the probability of taking a random action; K, KDE fitting frequency; W, Number of warm-up episodes; gN, replay buffer capacity.
Output: Q-network Q.

Initialize replay buffer $\mathcal{D}$ with capacity $\mathcal{N}$
$Q_\theta(s, a)$: state-action value predictor
$E_\phi(\log d)$: uncertainty estimator network, which takes the log density of the states as the input
$d(s)$: Kernel density estimator (KDE)
for episode=1 to M **do**
  set s_0 as the initial state
  for t=1 to *max-steps-per-episode* **do**
    Sample r uniformly between 0 and 1
    if $r < \epsilon$ **then**
      Set a_t to a random action
    else if *episode* $\leq$ W **then**
      $a_t = max_a Q_\theta(s_t, a)$
    else
      $a_t = max_a \Big[Q_\theta(s_t, a) + \kappa \times E_\phi(\log d(s_t))(a) \Big]$
    end if
    Observe r_t and s_{t+1} and store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch B of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    if s_j is a final state **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma max_a Q(s_t, a)$
    end if
    Update Q-network:
    $\theta \leftarrow \theta + \alpha_Q . \nabla_\theta \, \mathbb{E}_{(s,a)\sim B} \left[\Big(y_j - Q_\theta(s, a) \Big)^2 \right]$
    Update E-network:
    $\phi \leftarrow \phi + \alpha_E . \nabla_\phi \, \mathbb{E}_{(s,a)\sim B} \left[\left[\Big(y_j - Q_\theta(s, a) \Big)^2 - E_\phi(\log d(s_t))(a) \right]^2 \right]$
    if mod(*total-steps*, K) $= 0$ **then**
      fit the KDE d on the states of $\mathcal{D}$
    end if
    $s_t \leftarrow s_{t+1}$
  end for
end for
Return: Q_θ

model variance estimator on only the corresponding eight classes. The remaining two classes act as the out-of-distribution examples for training the error predictor. Using these folds, we construct a dataset for training the error predictor, a simple feed-forward network. The error predictor is trained with the log targets (i.e., log MSE between predicted and observed

error). This helps since the scale of the errors varies over multiple orders of magnitude. We then train the main predictor, density estimator and the variance estimator on the entire CIFAR-10 dataset, for evaluation. The hyperparameters are presented in Table F.3. For all models, we train the main predictor for 75 and 125 epochs for ResNet-18 and ResNet-50, respectively. We use SGD with Momentum (set to 0.9), with a multi-step learning schedule with a decay of 0.2 at epochs $[25, 50]$ and $[45, 90]$ for ResNet-18 and ResNet-50, respectively. One complete training run for DEUP takes about 1.5-2 GPU days on a V100 GPU. In total, this set of experiments took about 31 GPU days on a Nvidia V100 GPU.

Parameters	**Model**	
	ResNet-18	**ResNet-50**
Number of members	5	5
Learning Rate	0.05	0.01

Parameters	**Model**	
	ResNet-18	**ResNet-50**
Number of samples	50	50
Dropout Rate	0.15	0.1
L2 Regularization Coefficient	6e-5	8e-4
Learning Rate	0.05	0.01

Table F.2 – **Left**: Hyperparameters for training Deep Ensemble (Lakshminarayanan et al., 2017b). **Right**: Hyperparameters for training MC-Dropout (Gal and Ghahramani, 2016b).

Parameters	**Model**	
	ResNet-18	**ResNet-50**
Gradient Penalty	0.5	0.65
Centroid Size	512	512
Length scale	0.1	0.2
Learning Rate	0.05	0.025

Parameters	**Model**
	ResNet-18
Inducing Points	50
Kernel	RBF
Lipschitz Coefficient	2
BatchNorm Momentum	0.99
Learning Rate	0.05
Weight Decay	0.0005

Table F.3 – **Left**: Hyperparameters for training DUQ (van Amersfoort et al., 2020). **Right**: Hyperparameters for training DUE (van Amersfoort et al., 2021).

Ablations. We also perform some ablation experiments to study the effect of each feature on the error predictor. The Spearman rank correlation coefficient between the generalization error and the variance feature, V, from DUE (van Amersfoort et al., 2021) alone is 37.84 ± 0.04, and the log-density, D, from MAF (Papamakarios et al., 2017) alone is 30.52 ± 0.03. With only the image (x) the SRCC is 36.58 ± 0.16

Appendix F.3 presents the results for these experiments. We observe that combining all the features performs the best. Also note that using the log density and variance as features

Parameters	**Model**	
	ResNet-18	**ResNet-50**
Uncertainty Predictor Architecture	[1024] x 5	[1024] x 5
Uncertainty Predictor Epochs	100	100
Uncertainty Predictor LR	0.01	0.01
Main Predictor Learning Rate	0.05	0.01

Table F.4 – Hyperparameters for training DEUP.

of the error predictor, we observe better performance than using them directly, indicating that the error predictor perhaps captures a better target for the epistemic uncertainty. The boolean feature (B) showing seen examples, discussed in Section 5.3.2, also leads to noticeable improvements.

Features	Model	
	ResNet-18	ResNet-50
D+V+B	**0.426 ± 0.009**	**0.465 ± 0.002**
D+V	0.419 ± 0.003	0.447 ± 0.003
V+B	0.401 ± 0.004	0.419 ± 0.004
D+B	0.403 ± 0.003	0.421 ± 0.002
x	0.352 ± 0.004	0.376 ± 0.001
x + D + V	0.382 ± 0.006	0.397 ± 0.002

Table F.5 – Spearman Rank Correlation between predicted uncertainty and the true generalization error on OOD data (SVHN) with variants of DEUP with different features as input for the uncertainty predictor. D indicates the log-density from MAF (Papamakarios et al., 2017), V indicates variance from DUQ (van Amersfoort et al., 2020) and B indicates a bit indicating if the data is seen.

F.3.1. Predicting Uncertainty under Distribution Shift

We also consider the task of uncertainty estimation in the setting of shifted distributions (Ovadia et al., 2019; Hendrycks and Dietterich, 2019). We evaluate the uncertainty predictions of models trained with CIFAR-10, on CIFAR-10-C (Hendrycks and Dietterich, 2019), which consists of images from CIFAR-10 distorted using 16 corruptions like Gaussian blur and impulse noise, among others. Figure F.1 shows that even in the shifted distribution setting, the uncertainty estimates of DEUP correlate much better with the error made by the predictor than the baselines.

F.4. Drug Combination Experiments

To validate DEUP's uncertainty estimates in a real-world setting, we measured its performance on a regression task predicting the synergy of drug combinations. While much effort in drug discovery is spent on finding novel small molecules, a potentially cheaper method is identifying combinations of pre-existing drugs which are synergistic (i.e., work well together). Indeed, drug combinations are the current standard of care for several diseases, including HIV, tuberculosis, and some cancers (Cihlar and Fordyce, 2016; Organization and Initiative, 2010; Mokhtari et al., 2017).

However, due to the combinatorial nature of drug combinations, identifying pairs exhibiting synergism is challenging. Compounding this problem is the high monetary cost of

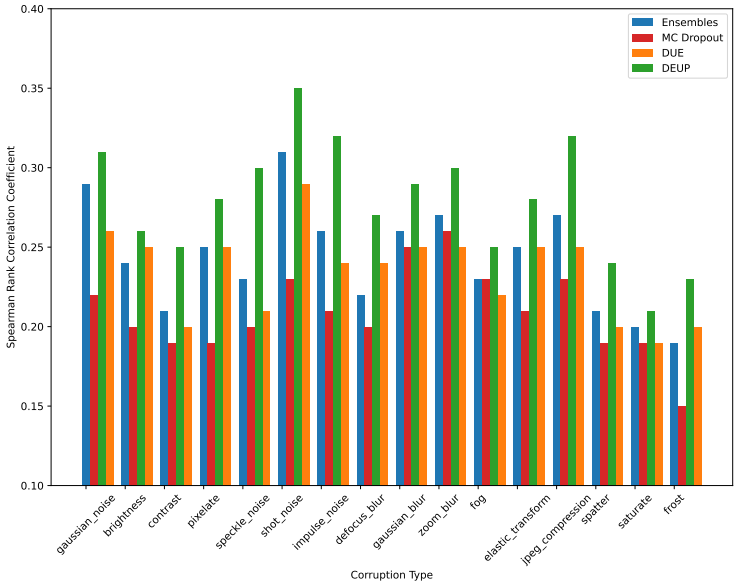


Figure F.1 – Spearman Rank Correlation Coefficient between the predicted uncertainty and true error for models trained with CIFAR-10 and evaluated on CIFAR-10-C. DEUP outperforms the baselines on all types of corruptions.

running experiments on promising drug combinations and the length of time the experiments take to complete. Practitioners could use uncertainty models to help accelerate drug combination treatment discoveries and reduce involved development costs.

To test DEUP's performance on this task, we used the DrugComb and LINCS L1000 datasets (Zagidullin et al., 2019; Subramanian et al., 2017). DrugComb is a dataset of pairwise combinations of anti-cancer compounds tested on various cancer cell lines. The dataset provides several synergy scores for each combination, indicating whether the two drugs have a synergistic or antagonistic effect on cancerous cell death. LINCS L1000 contains differential gene expression profiles for various cell lines and drugs. Differential gene expressions measure the difference in the amount of mRNA related to a set of influential genes before and after the application of a drug. Because of this, gene expressions are a powerful indicator of the effect of a single drug at the cellular level.

In our experiments, each drug is represented by its Morgan fingerprint (Morgan, 1965) [1] (with 1,024 bits and a radius of 3) as well as two differential gene expression profiles (each of dimension 978) from two cell lines (PC-3 and MCF-7). To use gene expression features for every drug, we only used drug pairs in DrugComb, where both drugs had differential gene expression data for cell lines PC-3 and MCF-7.

1. The Morgan fingerprint represents a molecule by associating a boolean vector specifying its chemical structure with it. Morgan fingerprints have been used as a signal of various molecular characteristics to great success (Ballester and Mitchell, 2010; Zhang et al., 2006).

We first compared the quality of DEUP's uncertainty estimations to other uncertainty estimation methods on the task of predicting the combination sensitivity score (Malyutina et al., 2019) for drug pairs tested on the cell line PC-3 (1,385 examples). We evaluated the uncertainty methods using a train, validation, test split of 40%, 30%, and 30%, respectively. The underlying model used by each uncertainty estimation method consisted of a *single drug* fully connected neural network (2 layers with 2048 hidden units and output of dimension 1024) and a *combined drug* fully connected neural network (2 layers, with 128 hidden units). The embeddings of an input drug pair's drugs produced by the *single drug* network are summed and passed to the *combined drug* network, which then predicts final synergy. By adding the embeddings produced by the *single drug* network, we ensure that the model is invariant to permutations in order of the two drugs in the pair. The models were trained with Adam (Kingma and Ba, 2015), using a learning rate of 10^{-4} and weight decay of 10^{-5}. For MC-Dropout, we used a dropout probability of 0.1 on the two layers of the *combined drug* network and 3 test-time forward passes to compute uncertainty estimates. The ensemble used three constituent models for its uncertainty estimates. Both Ensemble and MC-Dropout models were trained with the *MSE* loss.

We also compared against DUE (van Amersfoort et al., 2021), which combines a neural network feature extractor with an approximate Gaussian process. Spectral normalization was added to all the layers of the *combined drug* network and of the *single drug* network. Let d_{emb} denote the dimension of the output of the *combined drug* network, which is also the input dimension of the approximate Gaussian process. We conducted a grid search over different values of d_{emb} (from 2 to 100), the number of *inducing points* (from 3 to 200), the learning rate, and the kernel used by the Gaussian process. The highest correlation of uncertainty estimates with residuals was attained with $d_{\text{emb}} = 10$, 100 *inducing points*, a learning rate of 0.01, and the *Matern12* kernel.

The DEUP model we used outputs two heads $\begin{bmatrix} \hat{\mu} \\ \hat{\sigma} \end{bmatrix}$ and is trained with the *NLL* $\frac{\log(\hat{\sigma}^2)}{2} + \frac{(\hat{\mu}-y)^2}{2\hat{\sigma}^2}$ in a similar fashion as in (Lakshminarayanan et al., 2017b). To obtain a predictor of the out-of-sample error, we altered our optimization procedure so that the μ and σ heads were not backpropagated through at all times. Specifically, we first split the training set into two halves, terming the former the in-sample set $\mathcal{D}_{in}$ and the latter the out-of-sample set $\mathcal{D}_{out}$. We denote as f_σ^{in} the in-sample error predictor and f_σ^{out} the out-of-sample error predictor. f_σ^{out} is used to estimate total uncertainty. Note that in this setting, f_σ^{out} predicts the square root of the epistemic uncertainty ($\hat{\sigma}_{out}$) rather than the epistemic uncertainty itself ($\hat{\sigma}^2_{out}$).

In our experiments, an extra bit is added as input to the model in order to indicate whether a given batch is from $\mathcal{D}_{in}$ or $\mathcal{D}_{out}$. Through this, the same model is used to estimate f_σ^{in} and f_σ^{out} with the model estimating f_σ^{in} when the bit indicates an example is drawn from $\mathcal{D}_{in}$ and f_σ^{out} otherwise. When the batch is drawn from $\mathcal{D}_{in}$, both heads are trained using

Algorithm 7 DEUP for Drug Combinations

Input: $\mathcal{D}$ dataset of pairwise drug combinations, along with synergy scores $((d_1, d_2), y)$
Output: f_μ synergy score predictor; f_σ^{in} in-sample error predictor, f_σ^{out} out-of-sample error predictor

Split training set into two halves, *in-sample* $\mathcal{D}_{in}$ and *out-of-sample* $\mathcal{D}_{out}$
$f_\mu(d_1, d_2)$: $\hat{\mu}$ predictor which takes a pair of drugs as input
$f_\sigma^{in}(d_1, d_2)$: In-sample $\hat{\sigma}_{in}$ error predictor
$f_\sigma^{out}(d_1, d_2)$: Out-of-sample $\hat{\sigma}_{out}$ error predictor
while training not finished **do**
  In-sample update
  Get an *in-sample* batch $(d_{1,in}, d_{2,in}, y_{in}) \sim \mathcal{D}_{in}$
  Predict $\hat{\mu} = f_\mu(d_{1,in}, d_{2,in})$ and *in-sample* error $\hat{\sigma}_{in} = f_\sigma^{in}(d_{1,in}, d_{2,in})$
  Compute *NLL*: $\frac{\log(\hat{\sigma}_{in}^2)}{2} + \frac{(\hat{\mu} - y_{in})^2}{2\hat{\sigma}_{in}^2}$
  Backpropagate through f_μ and f_σ^{in} and update.
  Out-of-sample update
  Get an *out-of-sample* batch $(d_{1,out}, d_{2,out}, y_{out}) \sim \mathcal{D}_{out}$
  Estimate $\hat{\mu} = f_\mu(d_{1,out}, d_{2,out})$ and *out-of-sample* error $\hat{\sigma}_{out} = f_\sigma^{out}(d_{1,out}, d_{2,out})$
  Compute *NLL*: $\frac{\log(\hat{\sigma}_{out}^2)}{2} + \frac{(\hat{\mu} - y_{out})^2}{2\hat{\sigma}_{out}^2}$
  Backpropagate through f_σ^{out} and update.
end while
Return: $f_\mu, f_\sigma^{in}, f_\sigma^{out}$

NLL using a single forward pass. However, when the data is drawn from $\mathcal{D}_{out}$ only the $\hat{\sigma}$ head is trained. To do this, we must still predict $\hat{\mu}$ in order to compute the NLL. But the $\hat{\mu}$ predictor f_μ must be agnostic to the difference between $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$. To solve this, we perform two separate forward passes. The first pass computes $\hat{\mu}$ and sets the indicator bit to 0 so f_μ has no notion of $\mathcal{D}_{out}$, while the second pass computes $\hat{\sigma}$, setting the bit to 1 to indicate the true source of the batch. Finally, we backpropagate through the $\hat{\sigma}$ head only. The training procedure is described in Algorithm 7

We report several measures for the quality of uncertainty predictions on a separate test set in Table F.6.

Model	Corr. w. res.	U. Bound	Ratio	Log Likelihood	Coverage Probability	CI width
MC-Dropout	0.14 ± 0.07	0.56 ± 0.05	0.25 ± 0.12	-20.1 ± 6.8	11.4 ± 0.2	3.1 ± 0.1
Deep Ensemble	0.30 ± 0.09	0.59 ± 0.04	0.50 ± 0.13	-14.3 ± 4.7	10.8 ± 1.4	3.4 ± 0.6
DUE	0.12 ± 0.12	0.15 ± 0.03	$\mathbf{0.80 \pm 0.79}$	-13.0 ± 0.52	15.2 ± 1.0	3.5 ± 0.1
DEUP	$\mathbf{0.47 \pm 0.03}$	0.63 ± 0.05	$\mathbf{0.75 \pm 0.07}$	$\mathbf{-3.5 \pm 0.25}$	$\mathbf{36.1 \pm 2.5}$	$\mathbf{13.1 \pm 0.9}$

Table F.6 – Drug combinations: quality of uncertainty estimates from different methods. *Corr. w. res.* shows correlation between model residuals and predicted uncertainties $\hat{\sigma}$. A best-case *Upper Bound* on *Corr. w. res.* is obtained from the correlation between $\hat{\sigma}$ and true samples from $\mathcal{N}(0, \hat{\sigma})$. *Ratio* is the ratio between col. 1 and 2 (larger is better). *Log-likelihood*: average over 3 seeds of per sample predictive log-likelihood. *Coverage Probability*: Percentage of test samples which are covered by the 68% confidence interval. *CI width*: width of the 86% confidence interval.

For each model, we report the per sample predictive log-likelihood, coverage probability and confidence interval width, averaged over 3 seeds.

We also computed the correlation between the residuals of the model $|\hat{\mu}(x_i) - y_i|$ and the predicted uncertainties $\hat{\sigma}(x_i)$. We noted that the different uncertainty estimation methods lead to different distributions $p(\hat{\sigma}(x))$. For example, predicted uncertainties obtained with DUE always have a similar magnitude. By contrast, DEUP yields a wide range of different predicted uncertainties.

These differences between the distributions $p(\hat{\sigma}(x))$ obtained with the different methods may have an impact on the correlation metric, possibly biasing the comparison of the different methods. In order to account for differences in the distribution $p(\hat{\sigma}(x))$ across methods, we report another metric which is the ratio between the observed correlation $Corr(|\hat{\mu}(x) - y|, \hat{\sigma}(x))$ and the maximum achievable correlation given a specific distribution $p(\hat{\sigma}(x))$.

This maximum achievable correlation (referred to as the *upper bound*) is not *per se* a comparison metric, and is estimated (given a specific $p(\hat{\sigma}(x))$) as follows: we assume that, for each example (x_i, y_i), the predictive distribution of the model $\mathcal{N}(\hat{\mu}(x_i), \hat{\sigma}(x_i))$ corresponds exactly to the distribution of the target, i.e., $y_i \sim \mathcal{N}(\hat{\mu}(x_i), \hat{\sigma}(x_i))$. Under this assumption, the residual of the mean predictor follows a distribution $\mathcal{N}(0, \hat{\sigma}(x_i))$. We can then estimate the upper bound by computing the correlation between the predicted uncertainties $\hat{\sigma}(x_i)$ and samples from the corresponding Gaussians $\mathcal{N}(0, \hat{\sigma}(x_i))$. 5 samples were drawn from each Gaussian for our evaluation. This upper bound is reported in the Table.

Finally, we reported our comparison metric: the ratio between the correlation $Corr(|\hat{\mu}(x) - y|, \hat{\sigma}(x))$ and the upper bound. The higher the ratio is, the closer the observed correlation is to the estimated upper bound and the better the method is doing.

It is interesting to note that the upper bound is much lower for DUE compared to other methods, as its predicted uncertainties lie within a short range of values.

Predicted $\hat{\mu}$ and uncertainty estimates can be visualized in Figure F.2 for different models. MC-dropout, Ensemble and DUE consistently underestimate uncertainty, while the out-of-sample uncertainties predicted by DEUP are much more consistent with the order of magnitude of the residuals. Moreover, we observed that DUE predicted very similar uncertainties for all samples, resulting in a lower upper-bound for the correlation between residuals and predicted uncertainties compared to other methods. We observed a similar pattern when experimenting with the other kernels available in the DUE package, including the standard Gaussian kernel.

Finally, we note that in the context of drug combination experiments, aleatoric uncertainty could be estimated by having access to replicates of a given experiment (*c.f.* Section 5.3), allowing us to subtract the aleatoric part from the out-of-sample uncertainty, leaving us with the epistemic uncertainty only.

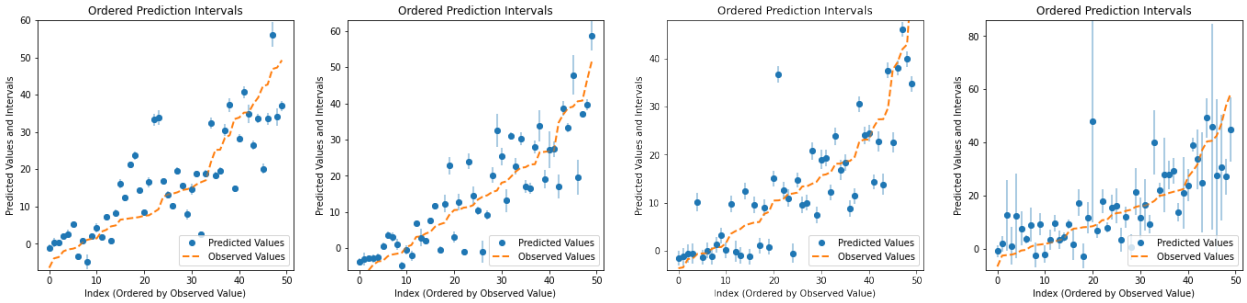


Figure F.2 – Predicted mean and uncertainty for different models on a separate test set. 50 examples from the test set are ordered by increasing value of true synergy score (orange). Model predictions and uncertainties are visualized in blue. MC-Dropout, Ensemble and DUE consistently underestimate the uncertainty while DEUP seems to capture the right order of magnitude. Figures made using The Uncertainty Toolbox (Chung et al., 2020).

One complete training run for the drug combination experiments takes about 0.01 GPU days on a V100 GPU. In total these set of experiments took about 0.2 GPU days on a Nvidia V100 GPU.