Université de Montréal

Reinforcement learning applied to the real world: uncertainty, sample efficiency, and multi-agent coordination

par

Vincent Mai

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

30 décembre 2022

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Reinforcement learning applied to the real world: uncertainty, sample efficiency, and multi-agent coordination

présentée par

Vincent Mai

a été évaluée par un jury composé des personnes suivantes :

Glen Berseth
(président-rapporteur)

Liam Paull
(directeur de recherche)

Marc Bellemare
(membre du jury)

Emmanuel Rachelson
(examinateur externe)

(représentant du doyen de la FESP)

Résumé

L'immense potentiel des approches d'apprentissage par renforcement profond (ARP) pour la conception d'agents autonomes a été démontré à plusieurs reprises au cours de la dernière décennie. Son application à des agents physiques, tels que des robots ou des réseaux électriques automatisés, est cependant confrontée à plusieurs défis. Parmi eux, l'inefficacité de leur échantillonnage, combinée au coût et au risque d'acquérir de l'expérience dans le monde réel, peut décourager tout projet d'entraînement d'agents incarnés. Dans cette thèse, je me concentre sur l'application de l'ARP sur des agents physiques. Je propose d'abord un cadre probabiliste pour améliorer l'efficacité de l'échantillonnage dans l'ARP. Dans un premier article, je présente la pondération BIV (batch inverse-variance), une fonction de perte tenant compte de la variance du bruit des étiquettes dans la régression bruitée hétéroscédastique. La pondération BIV est un élément clé du deuxième article, où elle est combinée avec des méthodes de pointe de prédiction de l'incertitude pour les réseaux neuronaux profonds dans un pipeline bayésien pour les algorithmes d'ARP avec différences temporelles. Cette approche, nommée apprentissage par renforcement à variance inverse (IV-RL), conduit à un entraînement nettement plus rapide ainsi qu'à de meilleures performances dans les tâches de contrôle. Dans le troisième article, l'apprentissage par renforcement multi-agent (MARL) est appliqué au problème de la réponse rapide à la demande, une approche prometteuse pour gérer l'introduction de sources d'énergie renouvelables intermittentes dans les réseaux électriques. En contrôlant la coordination de plusieurs climatiseurs, les agents MARL obtiennent des performances nettement supérieures à celles des approches basées sur des règles. Ces résultats soulignent le rôle potentiel que les agents physiques entraînés par MARL pourraient jouer dans la transition énergétique et la lutte contre le réchauffement climatique.

Mots-clés : Regression hétéroscéedastique, apprentissage par renforcement profond, estimation d'incertitude, apprentissage par renforcement multi-agents, réseau électrique, régulation de fréquence.

Abstract

The immense potential of deep reinforcement learning (DRL) approaches to build autonomous agents has been proven repeatedly in the last decade. Its application to embodied agents, such as robots or automated power systems, is however facing several challenges. Among them, their sample inefficiency, combined to the cost and the risk of gathering experience in the real world, can deter any idea of training embodied agents. In this thesis, I focus on the application of DRL on embodied agents. I first propose a probabilistic framework to improve sample efficiency in DRL. In the first article, I present batch inverse-variance (BIV) weighting, a loss function accounting for label noise variance in heteroscedastic noisy regression. BIV is a key element of the second article, where it is combined with state-of-the-art uncertainty prediction methods for deep neural networks in a Bayesian pipeline for temporal differences DRL algorithms. This approach, named inverse-variance reinforcement learning (IV-RL), leads to significantly faster training as well as better performance in control tasks. In the third article, multi-agent reinforcement learning (MARL) is applied to the problem of fast-timescale demand response, a promising approach to the manage the introduction of intermittent renewable energy sources in power-grids. As MARL agents control the coordination of multiple air conditioners, they achieve significantly better performance than rule-based approaches. These results underline to the potential role that DRL trained embodied agents could take in the energetic transition and the fight against global warming.

Keywords: Heteroscedastic regression, deep reinforcement learning, uncertainty estimation, multi-agent reinforcement learning, power grid, demand response.

Contents

List of tables

List of figures

List of acronyms and abbreviations

RL	reinforcement learning
DRL	deep reinforcement learning
MARL	multi-agent reinforcement learning
PPO	proximal policy optimization
MA-PPO	multi-agent PPO
DQN	deep Q-networks
MA-DQN	multi-agent DQN
SAC	soft actor-critic
DDPG	deep deterministic policy gradient
MA-DDPG	multi-agent DDPG
SGD	stochastic gradient descent
C51	Categorical 51
RPF	randomized prior functions
MSE	mean squared error
RMSE	root mean squared error
MC dropout	Monte Carlo dropout
DEUP	direct epistemic uncertainty prediction
MDP	Markov decision process
Dec-POMDP	decentralized, partially observable MDP
TD	temporal differences
UCB	upper confidence bounds
DTDE	distributed training, decentralized execution
CTDE	centralized training, decentralized execution
ReLU	rectified linear unit
BIV	batch inverse-variance
IV-RL	inverse-variance reinforcement learning

Acknowledgments

A PhD is commonly known to be a lonely endeavour. I was lucky: it was never the case for me. This thesis was a common effort by many amazing people, whom I would like to thank.

At a time when I had small research experience, Liam Paull trusted me with an opportunity and a responsibility - to be one of his first PhD students. I started slowly, but he gave me the time and space to grow and to learn. His reliable and caring support was a solid base upon which I could start building. His wise advice guided me through the winding path of learning to be a researcher. When we first met, Liam told me that he saw his lab as a team, where all worked together towards everyone's success. I am immensely grateful to have been part of his team. If I was to start again, I would not want any other supervisor than Liam.

The different works I present here are the fruits of collaborations with brilliant researchers. Waleed Khamies brought enthusiasm and courage at a time I most needed it. His hard work was instrumental, but it is his ability to cope with difficult situations which most inspired me. Kaustubh Mani had an enormous impact on this thesis. He joined a project which was at its embryonic state, and he was convinced enough that he agreed to work on it for a year. His trust gave me the confidence I needed to push this project forward. Kaustubh's insatiable curiosity, methodical rigour, and wide knowledge supported every next step in my thesis. Philippe Maisonneuve showed me the value of multidisciplinarity when working on an applied project. His intelligence, perseverance, and vision ensured the success of the final part of this thesis. The three of them will continue their path as PhD students. I have no doubt they will be successful – I only wish that they receive the same support they gave me.

Antoine Lesage Landry provided me with the opportunity – so precious for a PhD student – to apply my knowledge on an important challenge. I could count on his unwavering support and domain expertise for a project on renewable energies. This experience has probably shaped my future career. I had amazing collaborators, Tianyu Zhang, Hadi Nokoei and Jorge Montalvo, whose competence and motivation were a constant source of energy.

I was lucky to share my time at the Robotics and Embodied AI Lab with Krishna Murthy Jatavallabhula, who started his PhD at the same time than me, but mentored me in the academic world, Ali Harakeh with whom I shared many certainties about uncertainty, and Manfred Diaz and his always constructive feedback. Sai, Dhaivat, Dishank, Gunshi,

Mahtab, Mostafa, Simon and Miguel became dear friends. Florian, Charlie, Roger, Adriana, Ali, Kirsty, Breandan, Rey, Anthony, Steven, Nithin, Zihan, and Adam made this lab feel like home. A special mention goes to Glen Berseth, whose arrival brought yet another layer of kindness, experience, and expertise to the lab.

Although it is not shown in this thesis, my PhD work was not only technical research. I was immensely privileged to be able to explore the societal impacts of the field I was working on. From the Montreal to UNESCO, Marc-Antoine Dilhac took me in his teams and allowed me to work on high impact AI ethics projects. He was another mentor for me. A special mention to Martin Gibert, the sherpa opening new paths in AI ethics, to Christophe Abrassart, whose prospective scenarios made me think further, and to Anne Marie Savoie, who gave me my first role in this important initiative. I also want to thank Pauline Noiseau, whose strength never faltered, Fatima, Lucia and Camylle, who opened my eyes to completely new, but oh-so important, perspectives, and the courageous Gabrielle. My work in AI ethics and sustainability would not have been this exciting without my friend and partner-in-crime Carl-Maria Mörch. Together, we explored AI ethics in dentistry, we organized a workshop at IROS, and we took part in an international research project on sustainable robotics. I want to thank Bram Vanderboght and Tamas Haidegger for their support, Bhavani Rao R for her expertise and her immense kindness, and An Jacobs, AJung Moon, Dominik Boesl and Alaa Khamis for the precious experience they offered me.

I cannot not mention the people who made Mila a welcoming place. The Mila staff, who build everyday an extraordinary research environment: Émélie, Rose, Jeanne, Jean-François, Linda, Olexa, Benjamin, and I forget many. The professors, always ready to give advice and support: Simon Lacoste-Julien, Yoshua Bengio, Guillaume Dumas, Guillaume Rabusseau, Irina Rish, to name a few with whom I had the privilege to exchange. And the students, Victor, Tegan, David, Ezekiel, Salem, Muawiz, Michael, Nikolaus and so many more, with whom I share precious memories and who influenced my research and vision for the future.

Finally, I want to thank my parents, who always believed in this endeavor, provided me with everything I needed, and helped me to take a step back from time to time. My friends, with whom I could vent the horrors and beauties of graduate research. And my wife, Gabriela, who carried this PhD with me. She shared the burden, finding the right words at each setback, taking upon her during the days, weeks, months before a deadline, and enduring the nights I woke up at 3 AM with an idea I needed to note, now, now, or I would forget. She knew all my presentations by heart, after having been the supportive public of many practice rounds. She celebrated every little success, and took care to show me at every step the advances I had accomplished. I want her to know that we achieved this together.

Introduction

To build an autonomous agent which takes decisions in an intelligent manner, there are two paradigms. In the first, the developers must design instructions that the agent will follow. These rules are based on the programmers' knowledge of the task the agent must achieve, and on algorithms developed particularly for this task. Each module of the agent must be carefully designed and engineered. Depending on the complexity of the task, it can be straightforward or extremely difficult. This limits this classical approach: it can require a prohibitive amount of work for engineers to design an algorithm which will act correctly in every possible situation. For some tasks, a satisfying algorithm is even yet to be found. Engineered algorithms also scale poorly in quantity: it would be very expensive to build many different autonomous agents. Finally, these algorithms must be adapted if the environment changes, requiring further engineering work.

The other paradigm is *machine learning*. In machine learning, the agents learn the rules by themselves, based on data and experience. The branch of machine learning adapted to sequential decision making is reinforcement learning (RL): an agent learns by trial and error in an unknown environment, receiving a reward as a feedback signal. This paradigm has several advantages: it does not require that the programmers have a detailed knowledge of the environment, nor that they design an algorithm to solve the task. An agent is also able to learn continuously, and thus to adapt to environmental changes. While RL is a promising framework, its development is challenging. For decades, researchers struggled to make it work on complex environments with high dimensional or infinite state or action spaces. However, since the recent (2014) arrival of powerful neural network models, RL has taken a giant leap and showcased its immense potential. Trained agents could suddenly beat expert players at the games of Go and Dota 2. More recently, RL was used to fine-tuning a large language model for human preferences.

RL however has not yet been widely applied on agents which enact decisions in the real world. One difficulty is that neural network decisions are extremely difficult to explain. It is thus currently impossible to formally guarantee performance or safety from a trained agent, which is problematic in safety-critical applications of robotics. Another major challenge is that training an agent demands an enormous amount of experience. For virtual agents in

video games, this is not an issue: agents can safely train in accelerated time in parallel. The limiting factor is the amount of computing power available. For real world agents however, gathering experience cannot be accelerated or done in parallel. In addition, the real world is not safe for trial and error. A promising idea is to train an agent in a simulator and deploy it in the real world. However, this brings its own challenges, because simulators never exactly represent reality: there is often a significant sim-to-real gap in the environment dynamics as well as in the modelling of the sensor measurements observed by the agent. These obstacles currently impede the deployment of RL in the real world, and they are the key elements I tackle in this thesis.

In the first part, I focus on the issue of *sample efficiency of deep RL algorithms*. Reducing the amount of samples needed to reach a satisfying performance would reduce the costs and risks associated with training agents in the real world. I propose to improve this efficiency using the tool of probabilistic representations, i.e. modelling the variables' probability distributions. The RL process is inherently stochastic, but most algorithms overlook this aspect. However, as is the case in probabilistic robotics, using a Bayesian framework to represent random variables can lead to significant performance improvements for algorithms dealing with randomness. Indeed, probabilistic representations are richer and allow better informed decisions. The idea to use probabilities distributions in RL is not new, but it is complex to implement because of the lack of prior knowledge about the environment does not allow to quantify the information carried by every sample/experience. In the case of deep RL, there is an additional challenge: while neural networks are powerful as a deterministic tool, using them to handle probability distributions is intricate. Yet, methods to quantify the uncertainty of neural networks predictions have recently been developed, paving the way to the work I propose in this thesis.

RL algorithms typically use temporal difference learning. A neural network learns to predict a *value* in a regression setting where the labels, named targets, are predicted by another neural network. These labels are thus *noisy*, and the noise is *heteroscedastic*: its distribution is different for every label. Noisy labels reduce the amount of information in the dataset, and therefore slow the learning process. Leveraging state-of-the-art uncertainty prediction methods for neural networks, it is possible to quantify the noise variance for each predicted labels. Taking this quantity into account can mitigate the impact of the noise: this is the object of the first article of this thesis, **Batch inverse-variance weighting for deep heteroscedastic regression**. In this paper, a new loss function called batch inverse-variance (BIV) accounts for the variance of each noisy label in the training process of the neural network, reducing the impact of the noise. Because there are many neural regression settings where noisy labels are provided with a known variance, BIV was framed as a general

method and demonstrated on simple noisy regression tasks. It served as a proof of concept for a key element of my research project in RL.

The second article of this thesis, **Sample efficient deep reinforcement learning via uncertainty estimation**, tackles the problem directly. Through a detailed analysis of the sources of uncertainty in deep RL, we selected the relevant uncertainty estimation techniques for the targets. Combining these with BIV, we crafted a new approach called inverse-variance reinforcement learning (IV-RL), which can be applied on common deep RL algorithms such as deep Q-networks or soft actor-critic. IV-RL results in significantly faster training and better overall performance, especially in the case of control tasks.

The next objective of my thesis was to show that RL can be applied to a real world agent problem which *engineered* approaches have not solved. I chose the field of power grids, for two main reasons. First, because power grids are a very interesting domain for RL. It showcases many difficult optimization problems where current hand-engineered algorithms are not optimal. It also benefits precise simulations of dynamics and low dimensional observations, making the sim-to-real gap simpler to handle than for most real world agents, such as mobile robots. Second, I believe the role of research is to develop solutions to society's problems. One of the biggest challenges today is the transition from fossil fuels to renewable energy sources to fight global warming. Renewable sources are hard to control, and bring new algorithmic challenges to power grid control.

In the third article of this thesis, **Multi-Agent Reinforcement Learning for Fast-Timescale Demand Response of Residential Loads**, I worked in collaboration with field experts. We propose a solution to the high frequency intermittency of renewable energy sources such as solar panels or windmills. Through the paradigm of demand response, we suggest to balance power supply and demand by coordinating multiple flexible loads such as air conditioners. Air conditioners are present in many households and consume a significant amount of power. They also are able to satisfy the user's thermal preferences while being flexible on the exact moment they are ON or OFF. They are however discretely controlled, and subject to hardware dynamic constraints which impedes scalable engineered approaches. This problem also combines several challenges for state-of-the-art MARL algorithms: coordination, communication and scaling. We developed several decentralized multi-agent reinforcement learning (MARL) algorithms. The policies trained with few agents reached good performance even when deployed on arbitrarily high number of agents, demonstrating that the decentralized MARL approach is promising for the large scale control of agents in the power grid context.

The three articles presented in this thesis are aimed as steps towards the deployment of deep RL methods on real world agents. They do not solve every obstacle towards unlocking

the potential of deep RL. I hope they can still be some of the proverbial small bricks in humanity's cathedral of knowledge, and that they may support a world were real world autonomous agents assist humans with the significant challenges ahead.

This thesis starts with the background content necessary to understand the contributions. Each article is then presented, accompanied by a prologue detailing the context of its conception and publication, and of my particular contributions. The thesis ends with a discussion chapter, detailing the impact and the limitations of these works as well as further directions of research. After the bibliography, the reader will finally find in appendix the supplementary material for each article.

Chapter 1

Background

In this chapter, I will provide the background necessary to understand the contributions I present in this thesis. We will first focus on uncertainty in regression with deep neural networks. In particular, we will define predictive uncertainty, explain its sources, and have an overview of the existing methods to estimate it. Then, we will go through an overview of RL. We will start with the tabular case, with finite and low-dimensional state and action spaces, to better understand the base concepts of RL. We will then move towards function approximation using deep neural networks. Finally, we will discuss applying reinforcement learning to multiple agents, the particular challenges it brings, and the existing approaches to tackle them.

1.1. Uncertainty in regression with deep neural networks

In supervised learning, regression is the task of predicting a continuous value – as opposed to classification, where the predictions are of discrete nature. As regression using deep neural networks becomes an increasingly common part of complex pipelines, such as robotics systems or deep reinforcement learning (DRL), quantifying the uncertainty of the network's prediction can be instrumental to improve the pipeline's efficiency and robustness.

1.1.1. Objective of regression

The objective of regression is to find the parameters θ of a parameterized function $\hat{f}_\theta(x)$ such that $\hat{f}_\theta(x)$ approximates the maximum likelihood $f(x)$ of an unknown probability distribution $p_Y(y|x)$.

Often, a Gaussian distribution $\mathcal{N}(\mu_y, \sigma_y)$ is assumed for $p_Y(y|x)$. In this case, $\hat{f}_\theta(x)$ approximates the maximum likelihood $f(x)$ of the distribution if it predicts its mean μ_y. It can be proven that this is equivalent to minimizing the expected value of the L2 loss over

the joint distribution of inputs x and labels y:

$$\theta^* = \arg\min_\theta \mathbb{E}_{x,y}\left[\left(y - \hat{f}_\theta(x)\right)^2\right] \tag{1.1.1}$$

In regression, we assume that we have access to a dataset of input-label pairs $\mathcal{D} = \{(x_k, y_k)\}$ where $y_k \sim p_Y(y|x_k)$, $\forall k$. The objective is to use this dataset $\mathcal{D}$ to find the parameters θ^*. The expected error minimized in equation (1.1.1) is called the risk $J(\theta)$.

1.1.2. Neural networks optimization for regression

Neural networks, or perceptrons, are a powerful parameterized model for regression, as they have useful properties in terms of flexibility as well as ease of training. We superficially cover their optimization in this section. The interested reader can refer to [**56**] for more details and insights.

1.1.2.1. Defining a neural network. Neural networks consist of a sequence of operations $h^{(i)}(\cdot)$ where i is the layer's number. We have:

$$\hat{f}_\theta(x) = h^{(n)}\left(h^{(n-1)}\dots\left(h^{(1)}(x)\right)\right) \tag{1.1.2}$$

In the simplest case of multi-layer perceptrons, operations $h^{(i)}$ consist of a linear matrix operation with weights w_i and bias b_i, followed by a non-linear activation function $g(\cdot)$. The last layer $h^{(n)}$ is sometimes activated by a different function, $g_n(\cdot)$. Thus,

$$\hat{f}_\theta(x) = g_n\left(w_{(n-1)}^T\left(g\left(w_{(n-2)}^T\cdots\left(g\left(w_{(1)}^T x + b_1\right)\cdots + b_{(n-2)}\right) + b_{(n-1)}\right)\right)\right) \tag{1.1.3}$$

There are several options for the activation function – often, the rectified linear unit (ReLU) function is used successfully. ReLU is defined by:

$$g_{\text{ReLU}}(x) = \begin{cases} x & \text{if } x \leq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{1.1.4}$$

The last layer's function $g_n(\cdot)$ depends on the range of acceptable y's. For example, it can be the identity function if $y \in \mathbb{R}$, a ReLU function if $y \in \mathbb{R}^+$, or a sigmoid if $y \in [0, 1]$.

The parameters θ of the model are its set of weights and biases, $\{w_{(i)}, b_i\}$.

1.1.2.2. Training a neural network. To optimize the parameters towards θ^*, the neural network is trained over dataset $\mathcal{D} = \{(x_k, y_k)\}$ of size N. Using stochastic gradient descent (SGD), or a SGD-based optimizer such as Adam [**78**], we seek to minimize the empirical risk $\hat{J}(\theta)$. The empirical risk in regression is the average L2 loss, or mean squared error (MSE), over the whole dataset. It approximates the true risk $J(\theta)$ described in eq. (1.1.1). We therefore seek parameters $\hat{\theta}^*$ such that:

$$\hat{\theta}^* = \arg\min_{\theta} \hat{J}(\theta) \tag{1.1.5}$$

$$= \arg\min_{\theta} 1/N \sum_{(x,y)\in\mathcal{D}} \left(y - \hat{f}_\theta(x)\right)^2 \tag{1.1.6}$$

In single batch gradient descent, the parameters are updated through gradient steps:

$$\hat{\theta} \leftarrow \hat{\theta} - \alpha \nabla_\theta \hat{J}(\theta) \tag{1.1.7}$$

where α is the step size, also called learning rate.

To compute the empirical risk's gradient $\nabla_\theta \hat{J}(\theta)$ with respect to the parameters, we use chain derivation. Starting from the end node of the neural network, the derivative is back-propagated through the network's computational graph until it is evaluated for every parameter.

In SGD, the dataset $\mathcal{D}$ is divided into mini-batches of size m_{mini}. We then update the parameters using a sequence of gradients of the empirical risk $\hat{J}_{\text{mini}}(\theta)$ computed on these mini-batches. SGD is more flexible than single batch gradient descent as it is not necessary to optimize the parameters on the whole dataset at once. While the estimated gradient is less precise because it is based on less data, using SGD also helps avoid local minima in non-convex optimization landscapes.

Some more advanced optimizers, such as RMSProp [**160**] or Adam [**78**], build over SGD and propose adaptive learning rates α based on the gradient's moments. They usually lead to more stable and robust training processes.

1.1.3. Predictive uncertainty

Predictive uncertainty is the uncertainty of the output of the model in supervised regression. It quantifies the confidence we can have in the output $\hat{f}_\theta(x)$ in relation to the actual value y we aim to predict.

In regression with L2 loss, we usually assume that $\hat{f}_\theta(x) \sim \mathcal{N}(y, \sigma_p^2)$, i.e. the prediction is unbiased around the true label, but follows a Gaussian with standard deviation σ_p. By estimating this standard deviation, the predictive uncertainty quantifies the expected loss [**72**].

When measuring the performance of a model on a validation dataset, we measure its predictive uncertainty averaged over the whole dataset. However, in some cases, it can be useful to quantify the uncertainty of a given prediction. There may be some inputs inside the dataset for which the expected loss is higher than others. It can also happen that inputs given to a deployed model are out-of-distribution with respect to the training and validation datasets, in which case the performance measurements on the validation set is not trustworthy.

1.1.4. Sources of predictive uncertainty in supervised regression

There are two main sources of error, and thus uncertainty, between $\hat{f}_\theta(x)$ and y [**70**]. (1) The model may not be optimal to predict the maximum likelihood function $f(x)$, in which case, there is epistemic uncertainty. Epistemic uncertainty can be reduced with a better model. (2) Predicting $f(x)$ outputs the maximum likelihood of the original distribution $p_Y(y|x)$. The uncertainty due to the average error between y and $f(x)$ is irreducible, and is called aleatoric uncertainty.

1.1.4.1. Aleatoric uncertainty. In regression, aleatoric uncertainty is often expressed as the variance of the distribution of labels given an input, $p_Y(y|x)$. Following a Bayesian perspective, aleatoric uncertainty is due to the lack of information in the input x to predict the label y precisely, even with the best possible model [**70, 175**]. Quantifying aleatoric uncertainty is therefore related to quantifying the mutual information between the input and the label.

For example, a task of predicting the sum $a + b$ given $x = (a, b)$ has no aleatoric uncertainty: there is only one answer and $p_Y(y|x)$ is a delta Dirac at $a + b$. In a task such as predicting the age y of a person given a picture x of their face, there is some aleatoric uncertainty: while the image contains some information about the person's age, time does not affect the appearance of everyone's face the same way, and there are other elements than y that may have had impacts x too. As a result, no model has enough information from someone's facial picture x to predict their age y with an arbitrary accuracy. The aleatoric uncertainty would be even higher if the x was a picture of someone's face hidden behind a pillar. If in that picture we do not see any piece of the person's body, there is no information in x with respect to y. At maximum, aleatoric uncertainty can reach the uncertainty of the prior over the labels distribution $p(y)$; in our last example, it would be the variance of the general distribution of people's age.

For a given task, aleatoric uncertainty can vary depending on the input x; this is called heteroscedastic aleatoric uncertainty. For example, predicting the age of a new-born given a picture of their face allows for a higher accuracy - to the scale of days or weeks - than predicting the age of an elderly person - which can be in the dozen of years.

Note that aleatoric uncertainty is *agnostic* to the model we use, as it is a characteristic of the input and the task. This is why it is irreducible, because without additional information in the input, no model can reduce this uncertainty.

1.1.4.2. Epistemic uncertainty. Epistemic uncertainty, on the other hand, characterize the distribution of the error due to the accuracy of the model.

For example, an inaccurate model giving a constant value of 40 years for the task of predicting the age of someone based on a picture of their face will often be very wrong,

and this error will lead to a high epistemic uncertainty. A better performing model will by definition reduce the epistemic uncertainty.

In the case of a model trained through a machine learning algorithm, there are several reasons the model can be inaccurate. First, it can be parameterized in a way which does not have enough representation power for the task. For example, a linear model may not have the representation power to accurately to predict someone's age from their picture. Second, the training dataset may not contain enough information for the model to accurately infer the label of a given input. For example, the dataset may be too small, leading to an overfitting of the model. A region in the input space can also lack support in the dataset. In this case, the model may not be able to correctly predict the label of an input from this region. Finally, the optimization algorithm may not update the parameters optimally from the dataset. In the extreme case, a random optimizer would not lead to any learning. In the common case where a recognized optimizer, such as Adam [**78**] or RMSProp [**160**], is correctly implemented, its performance will depend on its hyperparameters. Even with the optimal hyperparameters, it may converge at a local optimum, missing the global optimum, as the optimization landscape may not be convex.

These three factors - representation power, amount of information in the dataset, and optimization - can lead to errors in the prediction due to the model. Quantifying epistemic uncertainty means estimating the expected error of a prediction due to these factors. Similarly to aleatoric uncertainty, epistemic uncertainty can be heteroscedastic. Some areas of the input space may be less well represented by the parameterization, have lower support from the dataset, or be disadvantaged by the optimization process. For example, an imbalanced dataset with significantly more pictures of women than men could lead to higher epistemic uncertainty on men's age predictions.

1.1.4.3. Noisy labels. Learning with noisy labels is a settings in which, in the dataset, the labels are subject to noise, i.e. they are not the true label. One example in regression would be that a 0-mean Gaussian noise δ_i with variance σ_i^2 is added to every label y_i. The noisy labels $\tilde{y} = y_i + \delta_i$ create a noisy dataset $\tilde{\mathcal{D}} = \{x, \tilde{y}\}$.

It is important to differentiate noisy labels and aleatoric uncertainty, as they are sometimes confused in the literature [**77, 116**]. As an example, the task of summing two numbers does not have aleatoric uncertainty, as the sum of two numbers is deterministic. A dataset could still have noisy labels, for example with the following inputs and labels: $\{(1{,}1; 3), (1{,}2; 2), (1{,}1; 1), (1{,}2; 4)\}$. On the other hand, a task with aleatoric uncertainty without noisy labels could be predicting someone's age from a picture of their face, in which the labels in the dataset are the persons' exact ages.

In some particular cases, aleatoric uncertainty can cause noisy labels. This can happen if during the labeling process the label generator only has access to the input x. Consider

the example where a human being is asked to label people's faces with their age to create the dataset on which a model is trained. If given only images of faces, the labeler will make errors which, at minimum, will be due to aleatoric uncertainty. However, they could also have access to the people's birth date and the date the picture was taken, in which case they can compute the exact age.

Because it reduces the amount of information in a dataset, noise on the labels can lead to a less well trained model, and thus induce epistemic uncertainty. Noisy labels can affect the learning process and lead to overfitting. With enough data and unbiased noise, the model can however learn to predict the mean of the noisy labels, i.e. the true label [**137**]: this uncertainty is not irreducible. In the above example of a noisy dataset for the sum of two numbers, a model trained with the two first samples would have a high epistemic uncertainty on inputs (1,1) and (1,2). Training with all samples would however balance the errors out and the model would predict the right label.

When the noise applied on the labels is biased, it also induces epistemic uncertainty, as the model is not trained correctly for the task. Better data is then needed to reduce the epistemic uncertainty.

1.1.5. Measuring uncertainty in deep supervised regression

To measure the uncertainty about a model's prediction is particularly challenging. Some machine learning techniques allow to do so in a theoretically grounded manner: Gaussian processes, for example, are able to estimate the epistemic uncertainty of a prediction based on the support of the dataset in the input space region [**22**]. However, this requires strong assumptions: a known task structure, low dimensional inputs, and explainable models.

Unfortunately, in most cases where deep neural networks are used, none of these assumptions is respected. Rigorously measuring the mutual information between an input and its true label, given an input and its prediction, is particularly challenging. Similarly, estimating the expected amount of error that is due to the model itself is difficult, for example because neural networks parameters do not encode the level of support a prediction has from the training dataset.

Although these issues are deeply rooted in the problem of quantifying aleatoric and epistemic uncertainties, several estimation methods have been proposed. We will review some here, explain how they work, as well as the assumptions they are based upon - and thus, their potential failure cases.

1.1.5.1. Estimating aleatoric uncertainty with neural networks. Most methods estimating aleatoric uncertainty in regression with neural networks are based on the same idea: instead of predicting the value $f(x)$ at maximum likelihood of $p_Y(y|x)$, they aim at learning the whole probability distribution.

In regression, a common assumption is that $p_Y(y|x)$ is a Gaussian, in which case it can be characterized by its mean $\mu_y(x)$ and its variance $\sigma_y^2(x)$. Both can be estimated by a neural network whose parameters θ are optimized to maximize the likelihood of $\mu_\theta(x)$ and $\sigma_\theta^2(x)$:

$$\theta^* = \arg\min_\theta \mathbb{E}_{x,y}\left[\frac{(y-\mu_\theta(x))^2}{\sigma_\theta^2(x)} + \ln \sigma_\theta^2(x)\right] \tag{1.1.8}$$

Equation (1.1.8) is obtained by expressing the maximization of the likelihood as the minimization of the negative log-likelihood. This approach is often referred to as loss attenuation [**77**].

In practice, the dynamics of this loss function can be intuitively understood as follows. The neural network predicts $\mu_\theta(x)$ and compares it with the label to compute the error. Then, it adjusts $\sigma_\theta^2(x)$ to scale down this error in the loss function, until it is balanced by the $\ln \sigma_\theta^2(x)$ term. At the end of the training, $\sigma_\theta^2(x)$ should predict the expected squared error between $\mu_\theta(x)$ and the label.

This correctly evaluates $\sigma_y^2(x)$, and thus aleatoric uncertainty, under three major assumptions:

(1) the training dataset is rich enough to represent the full probabilistic distribution $p_Y(y|x)$ in every relevant region of the input space. Otherwise, the neural network cannot characterize the right distribution;

(2) there is no epistemic uncertainty, i.e. $\mu_\theta(x) \approx \mu_y(x)$. Failing this, the expected error will be overestimated. This means, among other things, that the neural network must have trained sufficiently long for $\sigma_\theta^2(x)$ to be trustworthy;

(3) the labels do not have noise, as this method cannot differentiate noisy labels and aleatoric uncertainty. For example, with the $\{(1{,}1;3),(1{,}2;2),(1{,}1;1),(1{,}2;4)\}$ noisy dataset to learn to sum two numbers, $\sigma_\theta^2(1{,}1)$ would be predicted as 1 whereas, as previously mentioned, this task has a 0 aleatoric uncertainty and $\mu_\theta^2(1{,}1)$ would give the right answer, 2.

There exist other methods to estimate aleatoric uncertainty in regression. Again assuming a Gaussian distribution, calibration loss aims at predicting σ^2 as the average squared error $(\mu_\theta(x_k) - y(x_k))^2$ [**46**]. Other approaches avoid the Gaussian distribution assumption for $p_Y(y|x)$. With quantile regression [**28**], the neural networks outputs different values trained with different pinball losses so that they are biased towards a given quantile of the label distribution. Conformal quantile regression adjusts the predicted quantiles by empirically sorting the labels in an additional dataset in the vicinity of a given x. SampleNet [**63**] predicts different labels approximating a distribution by minimizing the energy score. To avoid overfitting, these are regularized by minimizing an optimal transport cost between these samples and a prior distribution.

All these methods are based on similar assumptions than loss attenuation, i.e. a rich training dataset, no epistemic uncertainty, and no noise in the labels. As such, they suffer from the same failure modes.

1.1.5.2. Epistemic uncertainty. To estimate epistemic uncertainty with neural networks, most existing approaches assume that the main source of epistemic uncertainty is the dataset. This is a reasonable assumption: wide and deep neural networks tend towards universal approximators, reducing the probability that the model underfits the data due to a poor representation power. Existing optimizers are also robust and perform well, and with good practices it can be assumed that the network's parameters are trained to reach near-optimal performance given the dataset.

To evaluate epistemic uncertainty in a particular region of the input space, one could imitate Gaussian processes and compute a density estimation of training data in the input space. The model's epistemic uncertainty should be lower where the density of data is higher. This strategy however suffers from the necessity of finding a distance measurement, or kernel, which correctly characterizes the similarity or difference between high dimensional data relatively to a given task. Manually engineering this kernel is challenging for unknown tasks or high dimensional inputs, whereas learning such distance measurement between two inputs, for example by measuring the distance between their latent representations in the network's hidden layers, requires support in the region, and thus only works in regions with low epistemic uncertainty.

The most successful methods to estimate epistemic uncertainty instead aim at producing a likelihood distribution for the model given the dataset. The Bayesian neural network approaches [**57, 23, 103, 169**] consider the weights of the network as random variables following a Gaussian distribution characterized by its mean and variance. These parameters are trained using variational inference. The epistemic uncertainty on a given prediction can be obtained in closed form or by computing the sample variance of several models randomly generated from the weights distributions.

Because Bayesian neural networks are difficult to train on large datasets or complex model architectures [**128**], other approaches approximate the distribution with the bootstrapping principle. Considering that the learning process is stochastic, several models are trained differently on the same task with the same dataset. At deployment, the different models' predict different outputs for the same input x. Epistemic uncertainty is then measured as the sample variance of the different predictions. These different models can be partially or completely separated, such as with ensembles [**90**], or techniques such as Monte Carlo dropout (MC dropout) [**52**]. Intuitively, diversity between the networks is encouraged such that the networks would predict different values in regions of the input space where they

have not been trained. On the other hand, in the regions they are trained, all the networks should converge to the same prediction.

In practice, these approaches have shown good results but they are not very reliable. Indeed, they assume a diversity in the model predictions, which is not always respected. We will take for example the case of ensembles, but it also applies to MC dropout. Diversity can be encouraged by initializing the neural networks differently, training on masked datasets, or applying adversarial training [**90, 92**]. However, neural networks' parameters are usually initialized so that the optimization is stable and robust, methods such as He [**66**] or Xavier [**53**] initialization. These methods usually lead to neural networks initially predicting random values around 0 with small scale variations depending on the input - the amplitude of which depends on the network's architecture. As such, even before any training, all the networks of the ensemble seem to agree, and bootstrapping leads to a very low epistemic uncertainty prediction. This apparent paradox is only possible because the assumption that encouraging diversity leads to purely random training processes is not satisfied. It prevents estimating the epistemic uncertainty of a model during the training process - which is problematic, as identifying the areas where training is necessary is an important use case of epistemic uncertainty prediction. It also raises the question of the process leading to diversity once trained: networks give different values at an unseen input regions only because they extrapolate differently from their training on other regions. There is no reason, or guarantee, that the sample standard deviation between the networks estimates the actual expected error of the networks in an unseen region of the state space. As an example, several networks trained to predict a constant value of 1 in a single input region A could extrapolate and predict values around 1 in unseen region B, leading to a low variance, even if labels in region B could be very far from 1.

1.1.5.3. Estimating predictive uncertainty. We already underlined the difficulties to separately estimate aleatoric and epistemic uncertainties. Estimating their combination, predictive uncertainty, is not straightforward. Not only most aleatoric estimation methods assume the absence of epistemic uncertainty - it is also not clear how these two quantities interact without strong assumptions.

Often in the literature, one or the other type of uncertainty is prioritized and considered as the only relevant one to estimate. Nevertheless, some approaches combine bootstrapping and negative log likelihood through mixture of Gaussians to output a predictive uncertainty estimation. [**90**] propose variance ensembles combining ensembles for epistemic uncertainty and loss attenuation for aleatoric uncertainty. Each network in the ensemble is trained using the negative log-likelihood loss of equation (1.1.8). The predicted distributions are then bundled as a mixture of Gaussians for which the mean and the variance are computed in

closed form, leading to state-of-the-art uncertainty predictions [**128**]. [**77**] follow the same logic, but they use MC dropout instead of ensembles.

Because of their strong assumptions and multiple failure modes, the different methods we described to estimate aleatoric, epistemic, or predictive uncertainty are usually not considered reliable enough for critical applications, where they would be very needed. They can however bring precious information in cases where the accuracy or the calibration of the uncertainty prediction are not critical. We will give an example in the next section, where some of these methods are used successfully in Bayesian approaches to reinforcement learning, as well as in the second paper of this thesis.

1.2. Reinforcement Learning

RL is a machine learning paradigm aiming at learning a policy performing in a previously unknown environment. The agent is able to train in the environment, gathering experience from which it can optimize its policy. As they do not assume a model of the environment, RL methods can be deployed in very large set of sequential decision-making problems. However, the implementation of RL on real world tasks is challenging, among other things because its training is expensive, risky, and sample inefficient.

In this section, we will first formulate the objectives of RL. We will then describe the setting of online RL and the policy iteration approach used to solve it. The first two sections are mainly based on the excellent book by Sutton and Barto [**154**]. We will then give a broad overview of the main deep reinforcement learning methods which combine RL with deep neural networks, leading to powerful algorithms for larger state and action spaces. We will finish by exploring different probabilistic DRL algorithms, which use probabilistic formulations to tackle some of the main challenges in DRL.

1.2.1. Problem formulation

The sequential decision making problems where RL is deployed can usually be modelled by Markov decision processes (MDPs) [**17, 69**] or one of their variants. A MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, $\mathcal{R}$ the reward space, $\mathcal{P}$ the dynamics of the MDP, and $\gamma \in [0, 1]$ its discount factor .

At time step t, the agent observes the environment's state $s_t \in \mathcal{S}$. It then takes action $a_t \in \mathcal{A}$ according to its current policy π: $a_t \sim \pi(a_t|s_t)$. The environment is then updated based on its dynamics $\mathcal{P}$: a new state s_{t+1} and a reward r_{t+1} are sampled from $p(s_{t+1}, r_{t+1}|s_t, a_t)$.

In RL, we try to find for each state s_t the action a_t which maximizes the expected sum of rewards after time t, which is named the expected return G_t. The return is more precisely

defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.2.1}$$

We differentiate MDPs with finite episodes, where the time stops at a given horizon H after which $r_{t>H} = 0$, and MDPs with infinite episodes, where t can go to infinity. In the latter case, the discount factor γ must be strictly smaller than 1 for the return to be finite. Importantly, the return follows a recursive relationship which can be proven by developing equation 1.2.1:

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{1.2.2}$$

The return from state s_t depends on the agent's trajectory in the future steps, and therefore on policy π. We define the value $V^\pi(s_t)$ as the expected return when following policy π from state s_t:

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t] \tag{1.2.3}$$

$$= \int_{a_t} \pi(a_t|s_t) \int_{s_{t+1}} \int_{r_{t+1}} p(s_{t+1}, r_{t+1}|s_t, a_t)[r_{t+1} + \gamma V^\pi(s_{t+1})] dr_{t+1} ds_{t+1} da_t \tag{1.2.4}$$

Equation 1.2.4 is called the Bellman's expectation equation and links the value at time step t to the value of future states. It is can be obtained from the recursive property of the return. In case the action, reward and/or the state spaces are discrete, the integrals can be replaced by sums. We can also define the Q-value $Q^\pi(s_t, a_t)$, as the expected return when following policy π after having taken action a_t at state s_t.

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t] \tag{1.2.5}$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1})|s_t, a_t] \tag{1.2.6}$$

For a given MDP, there exist an optimal value V^* at each state s. There also exists at least one optimal policy π^* for which value V^{π^*} is equal to V^* at every state s.

Finding the optimal policy π^* may demand very high computational costs and an enormous amount of experience. Instead, we usually aim at finding a policy which approximates π^*, i.e. for which, on most states s_t, value $V^\pi(s_t)$ is close to $V^*(s_t)$.

1.2.2. Online reinforcement learning

In the *online reinforcement learning* context, the agent has no previous knowledge of the MDP, and it must learn the policy by evolving in the environment and gathering experience. This is in opposition to offline RL, where the agent must learn from previously gathered experience upon which it has no control. In this thesis, we focus on *model-free* online RL: the agent does not seek to model the MDP. Instead, it learns the best policy based on the value at each state, through the process of policy iteration.

This section will cover the basics of online, model-free reinforcement learning, under the assumption that the state and action spaces are discrete and small. Large or continuous spaces are discussed in section 1.2.3.

1.2.2.1. Generalized Policy iteration. Behind most model-free RL algorithms is the idea of generalized policy iteration. It consists of a two step cyclic process: evaluate the policy, improve the policy, evaluate the new policy, improve it, etc. It is can be proven that, under certain conditions, generalized policy iteration always leads to a better policy, and ultimately converges to the optimal policy [**69**].

1.2.2.2. Policy evaluation. Evaluating a policy π is the process of learning the value $V^\pi(s)$ by an estimator $\hat{V}^\pi(s)$. There are two main strategies to do so: Monte-Carlo roll-outs and Temporal-Difference learning.

In both cases, we evaluate the value function from target $T(s)$, which is an approximation of the return. In discrete state space environments, we compute the average of the targets:

$$\hat{V}^\pi(s) = 1/N \sum_i^N T_i(s) \tag{1.2.7}$$

where $T_i(s)$ is evaluated every time step i the agent encounters state s. A state can be encountered several times in the same trajectory, or in different trajectories, also called episodes, experienced by the agent in the training process.

Monte-Carlo roll-outs. An approach to compute target $T_i(s)$ at a given state is to sample the return of a trajectory after visiting state s. This is called the Monte-Carlo roll-out return [**154**], and it is computed using equation 1.2.1 by "rolling out" the policy for the next time steps until the horizon – or, in the case of infinite environments, enough time for the discount factor to be close enough to 0. While this gives an unbiased sample of the return, it demands a lot of experience as the policy must be rolled over all the trajectory for the target to be computed.

$$T_t^{\mathrm{MC}}(s_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.2.8}$$

Temporal Differences. An alternative strategy, which allows more flexible updates of the policy's evaluation, uses a bootstrapping approach called temporal differences (TD) learning [**153**]. The recursive nature of the value function allows to use Bellman's equation (1.2.4). It can be proven that, given a MDP, Bellman's equation is satisfied for a given policy π if and only if the evaluation $\hat{V}^\pi$ is equal to the true value V^π of the policy. As such, the target is generated by enforcing the recursive relationship between the values at the current and next states:

$$T_t^{\mathrm{TD}}(s_t) = r_{t+1} + \gamma \hat{V}^{\pi}(s_{t+1}) \tag{1.2.9}$$

TD learning has the advantage of allowing to update the value estimation of a state one step only after visiting that state. However, because it is based on current value estimations, which may be wrong, the target is less informative than Monte Carlo roll-out.

Exploration. In both cases, the agent needs to visit state s to gather the information necessary to update its evaluation. To do so, it must choose actions to reach s, following exploration policy π_{explor}. A state that has not been visited enough may therefore be badly evaluated.

Because it is usually intractable to try all possible trajectories, π_{explor} must trade-off between gathering more experience through exploration, and building upon known successful seen trajectories through exploitation. Too much exploration can lead to inefficient training. On the contrary, too much exploitation can instead converge to sub-optimal policies because unexplored states could have better values than the current evaluation.

To explore, the behaviour during training must be stochastic, adding some noise on the optimal, exploitable action. This can be done uniformly: for example, ϵ-greedy exploration is done by choosing the current best action with a probability $1 - \epsilon$, or with probability ϵ to uniformly sample an action among the action space, at each time step t. It can also be guided by encouraging the agent towards states where the value estimation is uncertain, such as in Thompson sampling or upper confidence bounds (UCB). To do so, a belief distribution must be estimated for the value at each state. In UCB [**12, 11**], which follows the optimism under uncertainty paradigm, π_{explor} selects greedily based on the sum of the value belief's mean and standard deviation, giving an advantage to states with uncertain values. In Thompson sampling [**158, 139**], actions are randomly selected based on their probability of being the best, according to the belief: states with more uncertain values have a broader distribution and thus a higher probability to be selected. However, a major issue when applying these methods is the difficulty to update the belief by correctly quantifying the information received when visiting a state and accounting for the non-stationarity of the policy.

On-policy and off-policy algorithms. When computing the target, the next steps taken by the agent count: the Monte Carlo target needs samples from the whole subsequent trajectory, while the TD target needs at least s_{t+1} and r_{t+1} for TD. An agent following an exploration strategy with a stochastic policy will not be representative of a deployed agent, which will instead be greedy and always choose the best action based on its evaluation.

This leads to a contradiction: during training, the agent evaluates policy π_{explor}. Once the training is over and the agent is deployed, it will use another policy, π_{greedy}, which is not exactly the one that was evaluated. In *on-policy* learning, this contradiction is accepted

and assumed. The agent evaluates and improves π_{explor}, as this should lead to a better π_{greedy}. In *off-policy* learning, we decouple both policies. π_{explor} is abstracted and the agent directly evaluates π_{greedy}, using strategies such as importance sampling to deal with the difference between them when calculating the target. Off-policy allows more flexibility in the exploration strategy, and it gives the possibility to keep and re-use previous trajectories in memory. On the other hand, data gathered from π_{explor} is less informative relatively to π_{greedy}.

Q-learning. A very popular off-policy evaluation algorithm is Q-learning, a TD approach where the value is computed over a greedy policy at the next time step [**167**]. To learn the Q-value $\hat{Q}(s_t, a_t)$, the target $T_t^{\text{QL}}(s_t, a_t)$ is computed using the action with the maximal Q-value at this state, instead of the action chosen by π_{explor}.

$$T_t^{\text{QL}}(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) \tag{1.2.10}$$

1.2.2.3. Policy improvement. Once the value of a state is estimated, the policy can be improved by being greedy on the new value estimations: the agent must choose the action that leads to the best value. With discrete state and action spaces, this can be done through a simple table lookup.

Once the policy is improved, it has changed and the value estimation must therefore be repeated. It is worth noting that policy evaluation does not have to be complete before improving the policy. Policy improvement can be done after any number of steps of policy evaluation. In TD learning, it is therefore possible to update the policy at every single time step, after updating the value estimation. The inverse also works: policy improvement does not need to be complete for a new policy evaluation step to be ran.

1.2.3. Deep Reinforcement Learning

While the above cited approaches are suited to small and discrete action and state spaces, they need to be adapted when these spaces are continuous and/or larger. This is because, in these cases, it is not possible to explore every state and every action. The key to adapt to such environments is function approximation: instead of learning the value for each state and action, the agent learns a function that can generalize on states it has never seen before.

There are many function approximation methods, from discretization to memory-based to linear parameterization. However, the most recent successes of reinforcement learning came with function approximation through deep neural networks, used to parameterize the value estimator $\hat{V}(s)$ or $\hat{Q}(s,a)$, and/or the policy $\pi(s)$.

These algorithms, grouped into the field of deep reinforcement learning (DRL), can be divided in two families. Deep Q-learning algorithms approaches, such as DQN [**115**], put the emphasis on learning Q-values estimators, from which the policy is derived. Policy-gradient

methods, such as proximal policy optimization (PPO) [**142**], instead focus on learning a policy by approximating their gradient. Some algorithms are in between both approaches, such as soft actor-critic (SAC) [**61**]. An overview of these algorithms will be given in this section, as they are all used in this thesis.

Many of the latest advanced in DRL have been reached by incorporating probabilistic representations in the algorithms. This can help to guide exploration, to weigh updates, or to better represent the distributions of values. These will be reviewed in the last part of this section.

1.2.3.1. Deep Q-Learning. Deep Q-learning algorithms are based on the Q-learning approach described in section 1.2.2.2. The most popular deep Q-learning method is deep Q-networks (DQN) [**115**]. DQN works for continuous state but discrete action spaces. It learns the Q-value of a given state-action pair by optimizing the parameters θ of its value predictor $\hat{Q}_\theta(s,a)$ to predict the Q-learning target computed according to equation (1.2.10). $\hat{Q}_\theta(s,a)$ is a neural network which only takes state s as an input. It then outputs a Q-value for each discrete action a. The training is done through the same regression process as presented in section 1.1.2. The greedy policy is then simply chosen by selecting the action with maximal predicted value. Exploration is usually managed through ϵ-greedy.

To cope with known instabilities with offline TD learning with function approximation, DQN introduces several features. A target network, which follows the Q-network but with a delay, produces the Q-value estimation when computing the target. Memory is kept in a replay buffer, which is shuffled when training the network's parameters to reduce the correlation between steps from the same trajectory.

DQN is a simple yet powerful DRL algorithm. Several improvements exist, such as Double-Q learning to prevent the algorithm to overestimate the value of certain actions [**64**], or prioritized experience replay [**141**] which improves the selection of samples in the replay buffer. RainbowDQN [**68**] applies a combination of several improvements on DQN. DQN is however limited to discrete action spaces. A different framework, deep deterministic policy gradient (DDPG), can be considered as an extension of DQN to continuous action spaces.

1.2.3.2. Policy-gradient methods. These methods instead parameterize the policy $\pi_\phi(s)$, and directly optimize parameters ϕ through gradient descent. They are usually on-policy, as the gradient is computed from the current policy's experience. Because it is on-policy, the policy $\pi_\phi(a_t|s_t)$ must be stochastic to handle exploration. For discrete action spaces, $\pi_\phi(a_t|s_t)$ can be represented by a discrete probability distribution - for example, by adding a soft-max at the last layer of the neural network for each of the action space dimensions. They can also handle continuous action spaces, where $\pi_\phi(a_t|s_t)$ is often represented by a Gaussian, and the neural network outputs a mean and a variance for each dimension of the action space.

The parameters ϕ are updated based on the estimated gradient of a performance measure $J(\phi)$:

$$\phi \leftarrow \phi + \alpha \hat{\nabla}_\phi J(\phi) \tag{1.2.11}$$

This measure $J(\phi)$ can be the value, or expected return, of the policy $V_\phi^\pi(s)$ at any state s. In this case, it is possible to prove that:

$$\nabla_\phi J(\phi) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t) G_t\right] \tag{1.2.12}$$

Because we do not have access to the expected value, we estimate it with a mean computed on the set $\mathcal{D}$ of collected trajectories:

$$\hat{\nabla}_\phi J(\phi) = \sum_{\mathcal{D}} \sum_{t=0}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t) G_t \tag{1.2.13}$$

The sampled return G_t can have a lot of variance, making the learning difficult. It can be shown however that any value that only depends on the state s_t can be removed from G_t in equation (1.2.13). This is called a baseline, and can be used to stabilize the return. A popular choice of baseline is the value estimation $\hat{V}^{\pi_\phi}(s_t)$. This allows to optimize the advantage $J(\phi) = A^{\pi_\phi}(s_t, a_t) = Q^{\pi_\phi}(s_t, a_t) - V^{\pi_\phi}(s_t)$, where $Q^{\pi_\phi}(s_t, a_t)$ is approximated by Monte-Carlo using the return G_t experienced by the agent after taking action a_t at s_t. To compute $V^{\pi_\phi}(s_t)$, it is however necessary to learn a parameterized model for policy value. The type of architecture where both policy and value are learned models is called Actor-Critic. The actor is the policy model, trained through policy gradient, while the critic is the value model, trained through supervised regression.

Proximal policy optimization. A commonly used policy-gradient method is proximal policy optimization (PPO) [**142**]. It tackles one of the main problems of using equation (1.2.13): if the gradient is too high, the policy changes too much, which can destabilize the learning process.

PPO therefore limits the divergence between the old and the new policies. There are several versions of PPO, but the simplest - and most used - one uses a clipping function over the gradient, based on the ratio between the old and new $\pi_\phi(a_t|s_t)$. Because PPO is on-policy, its policy updates rely only on experience gathered since the last policy update.

1.2.3.3. Soft-actor critic. Soft actor-critic (SAC) [**61**] is an off-policy actor-critic algorithm which uses a part of Q-learning, as its critic learns the Q-value of state-action pairs based on Q-targets. To cope with continuous action spaces, it however takes in input both the state and the action, and outputs a single value as prediction. Similarly to DDPG, it then trains a parameterized policy to optimize the Q-value estimation.

An important feature of SAC is that it is entropy-regularized, i.e. that it gives an additional reward term at each time step to promote higher entropy on the policy. This guides the trade-off between exploration and exploitation, and is a key factor in the algorithm's good performance. The policy's stochasticity is parameterized as the hyperbolic tangent of a Gaussian with mean $\mu_\phi(s)$ and standard deviation $\sigma_\phi(s)$. It is possible to compute the policy's entropy $H(\pi_\phi(\cdot|s)) = -\mathbb{E}_a \log(\pi(a|s)$.

The value $V^{\pi_\phi}(s)$ is modified to include the entropy bonus:

$$V^{\pi_\phi}(s_t) = \mathbb{E}_{\pi_\phi}\left[\sum_{k=0}^{\infty} \gamma^k \left(r_{t+k+1} + \alpha H(\pi(\cdot|s_{t+k}))\right)|s_t\right] \tag{1.2.14}$$

Another important difference with Q-learning is that the action selection for SAC's target is not the one maximizing the Q-value, as in DQN or DDPG, but is instead sampled from the current policy. The critic therefore learns by supervised regression over the entropy-regularized target, which is sampled as:

$$T^{\pi_\phi}(s_t,a_t) = r_{t+1} + \gamma\left(\hat{Q}^{\pi_\phi}(s_{t+1},a_{t+1}) - \alpha \log \pi_\phi(a_{t+1}|s_{t+1})\right), \qquad a_{t+1} \sim \pi_\phi(\cdot|s_{t+1}) \tag{1.2.15}$$

As the critic and the entropy are differentiable with respect to actions, we use gradient ascent with respect to the policy parameters ϕ to optimize:

$$\phi = \max_\phi \mathbb{E}_{s_t \in \mathcal{D}}[V^{\pi_\phi}(s_t)] \tag{1.2.16}$$

$$= \max_\phi \mathbb{E}_{s_t \in \mathcal{D}, a_t \in \mathcal{A}}[\hat{Q}_\theta(s_t, a_t) - \alpha \log \pi_\phi(a_t|s_t)] \tag{1.2.17}$$

where $\mathcal{D}$ is the dataset of trajectories, $\mathcal{A}$ is the action space, and $a_t \sim \pi_\phi(\cdot|s_t)$.

1.2.4. Bayesian Deep Reinforcement Learning

In the algorithms presented until now, stochastic exploration policies were the only elements with a probabilistic representation. However, because the DRL algorithms are inherently stochastic, several works have demonstrated the advantage of learning the distributions of the random variables sampled in these algorithms. Probabilistic representations are richer than sampled values, and they allow for better informed actions or updates.

In this section, we will focus on the use of probabilistic representations to improve the performance of DRL algorithms on common metrics such as expected return after training and sample efficiency. There also exists a body of works exploring the use of probabilistic representations for safe or robust policies – we will not discuss these questions here.

1.2.4.1. Distributional DQN. The first distributional RL algorithm, Categorical DQN [**16**] aims at learning a distribution of returns instead of the value function. It defines the value distribution $Z^\pi_\theta(s_t,a_t)$, a random variable mapping state-action pairs to distributions over returns.

$Z^\pi_\theta(s_t,a_t)$ is discretely supported by N atoms z_i separated by constant intervals, and ranging from preset minimum and maximum. Each parameter θ_i is mapped to atom z_i such that the probability distribution is:

$$Z^\pi_\theta(s_t,a_t) = z_i \text{ with probability } p_i(s_t,a_t) = \frac{\exp \theta_i(s_t,a_t)}{\sum_j \exp \theta_j(s_t,a_t)} \tag{1.2.18}$$

Parameters θ are updated by projecting the targets computed by Q-learning from $Z^\pi_\theta(s_{t+1},a_{t+1})$ to the support of $Z^\pi_\theta(s_t,a_t)$. To select actions greedily, the agent maximizes the expected value $Q^\pi_\theta(s_t,a_t) = \mathbb{E}[Z^\pi_\theta(s_t,a_t)] = \sum_i z_i p_i(s_t,a_t)$.

Learning such distributions allows to stabilize the optimization process, to better handles handling multimodal return distributions, and overall lead to significant improvements in performance. Empirically, $N = 51$ atoms has led to good results, and the resulting algorithm is named Categorical 51 (C51). Applying quantile regression to distributional learning has shown to perform even better, among other reasons because the distribution support automatically covers the range of all values [**37**]. While C51 was built over DQN, it has been adapted to other Q-learning algorithms such as DDPG [**15**] or SAC [**106**], also leading to better performance or allowing for risk-sensitive policies.

1.2.4.2. Epistemic uncertainty estimation for exploration. Exploration is a key challenge in RL: a bad exploration strategy can significantly reduce the sample efficiency of the algorithm or converge to sub-optimal policies. On-policies algorithms such as PPO inherently manage exploration through policy stochasticity, whereas off-policy algorithms such as DQN need explicit exploration strategies such as ϵ-greedy.

However, these approaches are not guided, in the sense that they do not seek to explore regions of the state-action space which are unknown and could bring valuable experience, compared to well visited state space regions. To guide exploration, for example through UCB or Thompson sampling as explained in section 1.2.2.2, it is necessary to maintain a belief over the value estimation, i.e. represent it as a probability distribution. This is very different from Categorical RL, which aims at estimating the stochasticity of the return. Here, the belief instead represents the agent's current epistemic uncertainty of the value, which is not a random variable as it is an expectation.

There are several tools to estimate epistemic uncertainty in regression, as presented in section 1.1.5.2. Several algorithms use ensembles to do so. In [**32, 92**], an ensemble of Q-networks is maintained and used to estimate the mean and variance of the value estimation. The algorithm then uses UCB to guide exploration. In Bootstrapped DQN [**125**], an ensemble of Q-networks is also used; however, exploration is done by choosing a different, single Q-network at the beginning of each episode. Under the assumption that the Q-networks will give arbitrary values in region spaces where they have not been trained, this can be linked to Thompson sampling. Because this assumption is not respected at initialization as explained

in section 1.1.5.2, it can be beneficial to add a prior. This is proposed by randomized prior functions (RPF) [**127**], where the output of a fixed, differently initialized neural network is added to each Q-network's output in the ensemble.

While there are many other works on probability distribution for exploration in DRL, they mostly propose the same approach. Such strategies lead to better performance, especially in the case of environments with sparse rewards where exploration is key.

1.2.4.3. Out of distribution detection. In off-policy DRL algorithms such as DQN and SAC, the TD target is computed by bootstrapping from a current value estimation. However, the current value estimation is not always trustworthy, especially when the state-action pair at the next time step is out of distribution for the model, i.e. in an input region where the model has not been trained. To prevent these values from misleading the learning process, epistemic uncertainty estimation can be used as a threshold under which some updates are rejected. This is done in SUNRISE [**92**] in on-policy learning, where the uncertainty is estimated with ensembles, as well as in UWAC [**171**] in off-policy learning, estimating epistemic uncertainty with MC dropout.

1.3. Multi-agent reinforcement learning for decentralized collaboration

It is possible to using RL for autonomous sequential decision making in contexts where there are more than one agent. When the other agents have stationary policies, they can be considered by the learning agent as part of the environment's dynamics $\mathcal{P}$. However, it is often the case that other agents learn too, and that the task is to train the set of agents together. This adds different layers of complexity to the RL problem, such as non-stationary environments, coordination and communication, or scalability.

In this section, we define the multi-agent reinforcement learning (MARL) problem, identify the main challenges to MARL, and discuss the existing approaches to overcome them. We will focus on MARL for decentralized collaboration. A significant body of works exist on competitive MARL but is not particularly relevant for this thesis. On the other hand, centralized MARL where a single entity controls every agent to maximize a common reward can be simply formulated as a single agent RL problem with a multi-dimensional action space. A major source for this section is the survey on MARL by Gronauer and Diepold [**58**]. We also consulted the selective overview by Zhang et al [**182**]. We advise the interested reader to consult these two excellent sources for details and further approaches and methods.

1.3.1. Problem definition

The decentralized MARL problem differs from the single agent RL settings. When several agents learn together in the environment, its dynamics $\mathcal{P}$ become non-stationary environment from the perspective of a single agent. Every agent can seek to optimize a different return, which can consist of individual and shared elements, leading to collaborative, competitive, or mixed dynamics. The agents can also have access from different communication structures depending on the use case and if they are being trained or deployed.

1.3.1.1. Problem formulation. In the general case, the decentralized MARL problem can be formulated as a Markov game, an MDP defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ as in section 1.2.1, where the action space $\mathcal{A}$ is a joint action space, divisible between several agents i: $\mathcal{A} = \Pi_{i=1}^{N} \mathcal{A}^i$. The reward space is also separated between every agent: $\mathcal{R} = \Pi_{i=1}^{N} \mathcal{R}^i$. The environment dynamics $\mathcal{P}$ include the reward r^i distributions for each agent, conditioned on the state s and the joint actions $\{a^i\}$ [**58**]. This formulation assumes that every agent observes the full environment, for example in chess games where each agent can take an action once every second time step.

However, it is often the case that agents do not have complete access to the state s_t of the environment. They can observe a part of this state, through their own observations o_t^i. This is the case, for example, of autonomous cars, where each car has its own sensors and selects its actions according to their measurements. This setting is modelled by a decentralized, partially observable MDP (Dec-POMDP) [**18, 122**], characterized by the tuple $\langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$. $\mathcal{S}$ is the global state of the environment. $\mathcal{O} = \Pi_{i=1}^{N} \mathcal{O}^i$ is the joint observation space, where $\mathcal{O}^i$ is a partial observation of $\mathcal{S}$. Similarly to the Markov game formulation, $\mathcal{A} = \Pi_{i=1}^{N} \mathcal{A}^i$ a joint action space, and $\mathcal{R} = \Pi_{i=1}^{N} \mathcal{R}^i$ the joint reward space. The environment dynamics $\mathcal{P}$ include the observation o^i and reward r^i distributions for each agent, conditioned on the state s and the joint actions $\{a^i\}$.

1.3.1.2. Environment characteristics. When selecting or designing a MARL method to learn to apply to a Dec-POMDP, it is important to look at different other aspects of the problem. Identifying some aspects of the environment, such as training communication paradigm, agents' homogeneity, and reward structure, can help understand its challenges and opportunities.

Training communication paradigm. Even if the execution happens in a decentralized way, the agents can be trained in a centralized or a distributed manner.

In a *distributed training, decentralized execution (DTDE)* paradigm, the agents have to learn independently based only on their own experience. This happens when agents have to learn in an environment where they do not have additional means of communication

compared to the deployment environment. DTDE is a difficult paradigm [**111**], which scales poorly in number of agents [**59**].

In the *centralized training, decentralized execution (CTDE)* paradigm, we instead assume that the agents can share additional information during training [**81**]. This can happen when training in a simulator, or in an environment with additional communication infrastructures compared to the deployment environment. It allows, for example, to share experience, to account for other agents' observations and action intentions, or even to share policy parameters. Such strategies, which we will detail in subsequent sections, allow for higher sample efficiency and more stable training, and lead to state of the art performance.

Agent homogeneity. Homogeneous agents are agents whose observation and action spaces, as well as overall objectives, are similar. This allows for additional simplifications, compared to heterogeneous agents.

MARL reward structures. As every agent receives a different reward, they can have different objectives. We distinguish cooperative, competitive, and mixed settings.

In *fully cooperative* settings, all agents receive in common the exact same common reward, and they must therefore collaborate to increase the reward together. For example, a team of mobile robots handling products in a warehouse can have for objective the maximization of efficiency and productivity for the whole warehouse. *Cooperative* rewards is a relaxed version where agents may not receive the same reward but benefit from collaborating. For example, autonomous cars can aim at carrying their individual passenger from their current location to their objective as fast as legally possible, but they also need to collaborate with other cars to prevent accidents.

In *fully competitive* settings, the agents play a zero-sum game where they can only win as much as the other agents lose. They must therefore compete against the others. For example, a game of chess is fully competitive as only one player wins. The more relaxed *competitive* setting relates to environments which are not zero-sum but where agents can still benefit from reducing the others' reward. For example, an environment where agents must manage companies will see agents competing against other for a given market, but also acting together to expand this market.

Because many environments do not perfectly fit in these categories, the more general *mixed* settings encompasses the other possibilities. For example, in a team game such as soccer, agents have fully collaborative goals with agents on their team, but fully competitive goals with respect to the other team.

1.3.2. Challenges particular to MARL

As previously mentioned, MARL leads to several challenges. We will detail some of them here, focusing on collaborative settings.

1.3.2.1. Non-stationarity. Single agent RL approaches are based on the assumption that the environment dynamics, characterized by $\mathcal{P}$ in the MDP tuple, are stationary. This means that they do not evolve in time, making it possible for an agent to evaluate and improve a policy based on previous experience. Other agents can fit in this framework if they have stationary policies: they are then considered as part of the environment's dynamics.

In MARL, the other agents are learning, and therefore their policies are not stationary. From a single agent's perspective, the environment dynamics changes every time the policies are updated. This can lead to high instability in the learning process, as values and policies learned on previous environment dynamics may be completely unfit to the new dynamics.

1.3.2.2. Partial observation. In a Dec-POMDP, the agents are not able to individually fully observe the state of the environment. This breaks the Markov assumption over the environment. On the policy side, it adds aleatoric uncertainty as the input has less information to choose the best action. This is also true on the critic side, where it is harder to learn the state value based on a partial state observation.

1.3.2.3. Credit assignment. In collaborative settings, several agents take actions at a given time step, and this leads to a high, or low, common reward at current and later time steps. This leads to a difficult problem on the agent side: how to differentiate the impact of its own action from the others' actions? This issue is called the credit assignment problem and can lead to confusing feedback for the agent.

1.3.2.4. Coordination. In collaborative environments, the agents need to learn to coordinate. However, exploration for coordination may be tricky. As an example with two agents: if one agent follows an optimal coordination policy but the other one does not, it may lead to a low reward and send negative feedback to the first agent, who will reduce the probability of doing this action. Coordination therefore suffers from the combination of non-stationary dynamics and credit assignment issues.

1.3.2.5. Communication. It is sometimes necessary to share information between agents to better coordinate. It is possible to engineer the messages from the emitter's observations [**55**]. This strategy means that the receiver only must learn to interpret the messages. It is equivalent to increasing the observability of the state by the agents. However, it is limited in flexibility and does not allow agents to indicate their intentions.

Instead, it is also possible to learn to communicate. Agents must develop a communication protocole alongside their policy [**152**]. This allows more flexibility, but it leads to a

similar issue than learning to coordinate. The emitter needs to learn to communicate the right information, and the receiver to understand it. If one of these two agents acts the right way but the other one does not, both will be penalized, and the first agent will receive negative feedback for communicating correctly.

1.3.2.6. Scaling in number of agents. Most challenges previously mentioned become harder when scaling in number of agents. The environment becomes less stationary, and a single agent observes less of the relevant state information. Credit assignment naturally becomes a harder problem, making coordination and communication more difficult to learn too. On the top of these issues, training with more agents can be expensive, as it multiplies the computational power needed in simulation, or the efforts and risks in the case of real world agents.

1.3.3. MARL approaches

Many approaches have been developed to tackle the specific challenges of MARL. We will describe here the most relevant ones for this thesis.

1.3.3.1. Parameter sharing. In a CTDE setting, it is possible for the agents' actors and critics to share some model parameters [**59, 34**]. This reduces the amount of parameters to train, allowing for a better scalability of the learning process. The more similar are the agents, the more they can share parameters. Homogeneous agents can even share their whole learning model, making the learning process arbitrarily scalable in terms of number of agents.

1.3.3.2. Shared experience. In CTDE settings, homogeneous agents can also share their experience, as it can be valuable to the others. This can increase the scalability of the learning process, as the number of gathered samples is multiplied by the number of agents. It is naturally implemented in the case of full parameter sharing.

1.3.3.3. Centralized critics. In some actor-critic algorithms such as PPO, the critic is not deployed and only serves to stabilize the training, as seen in section 1.2.3.2. As such, in a CTDE setting, it is possible to have centralized critics which are given more information than the actor [**81**]. Centralized critics are aware of the observations and actions of the other agents. They predict the value $\hat{V}^{\pi}(o_t^1, ..., o_t^N, a_t^1, ..., a_t^N)$ where the action of the current agent a_t^i is not given as an input – otherwise it would be the Q-value. Including the other agents' observations reduces the impact of partial observations, while conditioning the prediction on the other agents' actions marginalizes their policies and reduces the impact of non-stationarity [**49, 24**].

1.3.3.4. Mean field RL. In environments with numerous agents, an agent can consider the mean of all other agents' impacts on itself as a single, global agent with which it has a bilateral interaction. It is then possible to model the interactions between a single agent and

the others as a mean-field environment with two agents. This approach, called mean-field RL, allows to reduce the complexity when scaling the number of agents [**177, 151, 30**].

1.3.3.5. Reduced communications. In some environments, deployment constraints impose a structure, such as locally constrained communication where agents only talk to neighbors [**183**]. This forced local coordination can still be sufficient depending on the task, and may even allow better scalability.

Even without constraints, reducing the number of other agents each agent communicates with can lower the complexity of learning communications. Neighborhood weighted communications can be relevant if there exists a relevant measure of distance between agents to characterize the impact of each other on their rewards [**33**]. For example, it is more relevant for autonomous cars to communicate with other cars in their vicinity. Attention mechanisms can also be used, to decide when to communicate [**74**], or with whom [**39**]. A recent survey about communications in MARL is available in [**186**].

1.3.3.6. Common MARL algorithms. The relevance of the approaches presented in the previous section strongly depends on the environment characteristics. They are usually implemented on the top of single agent DRL algorithms.

Multi-agent PPO (MA-PPO), in particular, is often regarded as a state-of-the-art baseline for MARL with Dec-POMDPs [**179, 123**]. In its most common implementation, a centralized critic is added to single agent PPO. Because it is on-policy, its experience always represents the current environment dynamics, making it less vulnerable to non-stationarity than off-policy alternatives. However, it needs significant amounts of experience, which can be expensive to scale in MARL settings.

Off-policy algorithms, such as multi-agent DQN (MA-DQN), MA-Rainbow DQN [**14**] and their continuous action counterpart multi-agent DDPG (MA-DDPG) [**104**], have been proposed when computational resources are limited. Off-policy algorithms allow to reuse experience but they suffer more than on-policy algorithms from the non-stationarity of environments, as the experience they re-use may not be representative of the latest environment dynamics after other agents update their policies. In addition, Q-learning algorithms such as MA-DQN use their critic in deployment, which prevents centralizing the critic in CTDE settings.

Chapter 2

Prologue to Article 1

This article was accepted and presented at the Thirty-eighth International Conference on Machine Learning (ICML) 2021, at the Workshop on Uncertainty & Robustness in Deep Learning. It was a joint work with Waleed Khamies and Liam Paull. Waleed Khamies was an intern at the Robotics and Embodied AI Lab.

In this article, we propose a new loss function to mitigate the effect of noisy labels in regression with deep neural networks, when the noise is heteroscedastic and its variance known for each label. While this paper could be relevant to many different use cases, it is a key element towards the second paper in this thesis.

Contribution statement: The problem tackled by this paper was identified and defined by myself, and I designed the method we propose. The experiments were conceived, implemented, ran and analyzed together with Waleed Khamies, under the guidance of Liam Paull. I prepared the manuscript, which was reviewed by both Waleed and Liam. After acceptance, Waleed ensured the code was available open source, tested extensively, and well documented.

Chapter 3

Article 1: Batch Inverse-Variance Weighting: Deep Heteroscedastic Regression

Résumé

Dans la tâche d'apprentissage supervisé de régression hétéroscédastique, chaque étiquette est sujette à un bruit provenant d'une distribution distincte. Le générateur d'étiquettes peut estimer la variance de la distribution du bruit pour chaque étiquette, ce qui constitue une information utile pour atténuer son impact. Nous adaptons une erreur quadratique moyenne pondérée par la variance inverse, basée sur le théorème de Gauss-Markov, pour la descente de gradient sur les réseaux neuronaux. Nous introduisons la fonction de perte BIV (Batch Inverse-Variance), qui est robuste aux échantillons sans bruit et permet de contrôler le taux d'apprentissage effectif. Nos résultats expérimentaux montrent que BIV améliore significativement les performances des réseaux sur deux ensembles de données bruyantes, par rapport à la perte L2, à la pondération de la variance inverse et à une méthode de référence basée sur le filtrage.

Mots clés: régression hétéroscédastique, étiquettes bruitées

Abstract

In the supervised learning task of heteroscedastic regression, each label is subject to noise from a different distribution. The label generator may estimate the variance of the noise distribution for each label, which is useful information to mitigate its impact. We adapt an inverse-variance weighted mean square error, based on the Gauss-Markov theorem, for gradient descent on neural networks. We introduce Batch Inverse-Variance, a loss function which is robust to near-ground truth samples, and allows to control the effective learning rate. Our experimental results show that BIV improves significantly the performance of the networks on two noisy datasets, compared to L2 loss, inverse-variance weighting, and a

filtering-based baseline.

Key words: heteroscedastic regression, noisy labels

3.1. Introduction

In supervised learning, a central assumption is that the samples in the training dataset, used to train the model, and the samples in the testing set, used to evaluate the model, are sampled from identical distributions. Formally, for input $\mathbf{x}$ and label y, this assumption implies that $p_{\text{train}}(\mathbf{x},y) = p_{\text{test}}(\mathbf{x},y)$. This can be decomposed as the product $p_{\text{train}}(\mathbf{x}) \cdot p_{\text{train}}(y|\mathbf{x}) = p_{\text{test}}(\mathbf{x}) \cdot p_{\text{test}}(y|\mathbf{x})$, which is true if :

(1) The training dataset is representative and the features in both datasets are sampled from the same distribution: $p_{\text{train}}(\mathbf{x}) = p_{\text{test}}(\mathbf{x})$.

(2) The labels in both datasets are sampled from the same conditional distribution: $p_{\text{train}}(y|\mathbf{x}) = p_{\text{test}}(y|\mathbf{x})$. If this condition is violated, the training labels are *noisy*.

One case of noisy labels is when the labelling process induces uncertainty about the labels. In this case, $p_{\text{train}}(y|\mathbf{x})$ encapsulates the uncertainty of the labelling process, whereas the ground truth $p_{\text{test}}(y|\mathbf{x})$, over which we seek to optimize the performance of the model, are drawn from a Dirac delta distribution, even though it may be impossible to collect such a dataset in practice. In this case, the performance of the deployed model may decrease since the training process did not actually optimize the model's parameters based on the correct data [**10, 76**].

In this paper, we examine the case where we have some additional information about $p_{\text{train}}(y|\mathbf{x})$. More specifically, we focus on the task of regression with a deep neural network when labels are corrupted by heteroscedastic noise: $p_{\text{train}}(y|\mathbf{x}) \sim \mathcal{N}(\mu_y,\sigma_y)$ where μ_y is the ground truth label and σ_y is different for each sample. We assume that we have access to at least an estimate of the variance σ_y for each corrupted each label. This information is available if the labels are being generated by some stochastic process that is capable of also reporting uncertainty. We examine how the knowledge of the estimate of the label noise variance can be used to mitigate the effect of the noise on the learning process of a deep neural network. We refer our method as Batch Inverse-Variance (BIV)[1], which is based on inverse-variance weighting for linear regression (IV). We show that applying the inverse-variance loss allows for a strong empirical advantage over the standard L2 loss. However, it is unstable in presence of near ground-truth labels and not appropriate for all optimizers. BIV includes two mechanisms to stabilize the learning process in this case.

Our claimed contributions are threefold:

(1) An analysis of the labelling process giving rise to a heteroscedastic regression problem in supervised learning.

[1] https://github.com/montrealrobotics/BIV

(2) We present BIV, a loss function which uses the noise variance to minimize the performance loss due to the noise in a robust and reliable way while keeping control on the learning rate.
(3) An experimental study comparing the performances of BIV over L2 loss, IV, as well as over a threshold-based filtering method.

How best to use the information of the variance of heteroscedastically noisy labels in neural networks has not been studied in detail in the literature, maybe because neural networks are usually robust to strong label noise [**137**] or because the datasets traditionally do not include such information. We believe however that (1) the recent usage of deep neural networks in fields in which uncertainty plays a major role, such as robotics [**159**], as well as (2) the combined advances in uncertainty estimation of the neural network output [**77, 90**] and in complex deep learning systems such as deep reinforcement learning where the labels are provided by another neural network [**115, 61**], make this problem highly relevant.

The outline of the paper is as follows: In section 2, we describe the task of regression with heteroscedastic noisy labels. In section 3, we position our work among the existing literature on learning with noisy labels. In section 4, we explain the challenges of using IV in neural networks, and introduce BIV as an approach to improve its robustness. In section 5, we describe the setup for the experiments we made to validate the benefits of using BIV, and we present and analyze the results in section 6.

3.2. Heteroscedastic noisy labels in regression

Consider an unlabelled dataset with inputs $\{\mathbf{x}_i\}$. To label it, one must apply to each input $\mathbf{x}_i$ an instance of a label generator LG_j which should provide its associated true label y_i. This label generator has access to some features $\mathbf{z}_i \in \mathcal{Z}$ correlated to $\mathbf{x}_i$. We define $LG_j : \mathcal{Z} \to \mathbb{R}$. When the labelling process is not exact and causes some noise on the label, the noisy label of $\mathbf{x}_i$ provided by LG_j is defined as $\tilde{y}_{i,j}$. Noise on a measured or estimated value is often represented by a Gaussian distribution, based on the central limit theorem, as most noisy processes are the sum of several independent variables. Gaussian distributions are also mathematically practical, although they present some drawbacks as they can only represent uni-modal and symmetric noise. We model:

$$\tilde{y}_{i,j} = y_i + \delta_{y_{i,j}} \text{ with } \delta_{y_{i,j}} \sim N(0, \sigma^2_{i,j}) \tag{3.2.1}$$

$\sigma^2_{i,j}$ can be a function of $\mathbf{z}_i$ and LG_j, without any assumption on its dependence on one or the other. We finally assume that the label generator is able to provide an estimate of $\sigma^2_{i,j}$, therefore being re-defined as $LG_j : \mathcal{Z} \to \mathbb{R} \times \mathbb{R}_{\geq 0}$. The training dataset is formed of triplets $(\mathbf{x}_i, \sigma^2_{i,j}, \tilde{y}_{i,j})$, renamed $(\mathbf{x}_k, \sigma^2_k, \tilde{y}_k)$ for triplet k for simplicity. This setup describes many labelling processes, such as:

Crowd-sourced labelling. In the example case of age estimation from facial pictures, labellers Alice and Bob are given $\mathbf{z}_i = \mathbf{x}_i$ the picture of someone's face and are asked to estimate the age of that person. Age is harder to estimate for older people come (5 and 15 years of age are harder to confuse than 75 and 85) suggesting a correlation between $\sigma^2_{i,j}$ and $\mathbf{z}_i$. But Alice and Bob may also have been given different instructions regarding the precision needed, inducing a correlation between $\sigma^2_{i,j}$ and LG_j. Finally, there may be some additional interactions between $\mathbf{z}_i$ and LG_j, as for example Alice may know Charlie, recognize him on the picture and label his age with lower uncertainty. Both labellers can provide an estimation of the uncertainty around their labels, for example with a plus-minus range which can be used as a proxy for standard deviation.

Labelling from sensor readings, population studies, or simulations. Imagine you want build a dataset of pictures $\mathbf{x}_i$ from a camera on the ground labelled with the position y_i of a drone in the sky. To estimate the position of the drone at the moment the picture was taken, you could use state estimation algorithms based on the Bayes' filter [**159**]. These algorithms take as an input $\mathbf{z}_i$ the measurements from the drone's sensors, and provide a full posterior distribution over the state, sometimes under a Gaussian assumption for Kalman filters for example. The uncertainty depends on the precision of the sensors, the observability of a given state, the precision of the dynamic model, and the time since sensor signals were received. Similarly, studies based on population such as polling or pharmaceutical trials have quantified uncertainties based on the quantity and quality of their samples. It is also possible to train on simulators, as in climate sciences [**133**] or in epidemiology [**7**], and some of them provide their estimations' uncertainty based on the simulation procedure and the inclusion of real measurements in the model.

$$\Big((s,a), \sigma^2_T, T(s,a)\Big)$$

Using predictions from a neural network in complex neural architectures. In deep reinforcement learning for example, the critic network learns to predict a value from a state-action pair under the supervision of the heteroscedastic noisy output of a target network plus the reward [**114, 61**]. While the estimation of the uncertainty of the output of a neural network is not an easy task, it is an active field of research [**52, 130**]. There, $\mathbf{z}_i$ is the state-action pair at the next step, and LG_j the target network being updated over time. The prediction is a mix of aleatoric and epistemic uncertainties as defined by [**77**] which are dependent on both $\mathbf{z}_i$ and LG_j.

We could not find any current dataset that provides such label uncertainty information for regression. However, as it is precious information, we argue that it should actually be provided when possible. In classification, [**174, 173**] took a step in this direction by providing a "confidence" score from 0 to 255 for each pixel in the KITTI-360 dataset.

3.3. Related work

Noise on labels amounts to a loss of information. When the noise is significant enough, it leads to overfitting and lower model performance [**102, 181**]. This effect is more prevalent in small data settings [**161**]. Four possible strategies exist in the literature to tackle this problem for neural networks: detection, correction, robustness, or re-weighting.

- *Detection* consists of identifying noisy labels and ignoring them in the learning process. These methods are often based on the observation that neural networks first fit on consistent, non-noisy data [**10**], thus converging to a higher loss on the noisy samples [**134, 144**]. Other methods use several neural networks to co-teach each other [**62, 180**] or dropout to estimate the consistency of the data [**134**]. However, in the case of imbalanced training datasets, higher loss can also be the signature of a non-noisy but rare sample. [**29**] address this ambiguity by regularizing different regions of the input space differently.
- *Correction* strategies go further: once noise is detected, the noisy labels are changed to probability distributions. Such an operation requires a noise model. [**54, 82, 107, 156, 178**] learn it jointly with the parameters, assuming a correlation between the noise and the input, the labels, or both.
- *Robust loss functions* are less sensitive to noise. [**102**] proposed to avoid overfitting due to noise by ignoring samples during the training when the prediction error is reasonable. [**118**] compute the loss assuming knowledge of example-independent mislabelling probabilities in binary classification, and then optimize these hyperparameters with cross-validation. More recent works are based on reverse cross-entropy [**165**] or curriculum loss [**105**]. Others leverage a distillate of the information gathered from a subset of clean labels to guide training with noisy labels [**97**].
- *Re-weighting* the samples is another efficient method for mitigating noise in datasets. [**101**] estimate the effective label probabilities as well as noise rates for a given input and use these estimates to weigh the samples using importance sampling. [**145**] go one step further by learning the weighting function through a meta-learning method. [**73**] control overfitting by adjusting sample weights in the training and validation mini-batches, increasing robustness to overfitting on noisy labels.

While most works that address noisy labels consider classification tasks [**148**], only some of these strategies can be generalized to regression. Heteroscedastic regression occurs when each label's noise is sampled from a different distribution. [**120**] tackle this problem in neural networks by jointly training a variance estimator based on the maximum likelihood of an underlying Gaussian model. [**77**] use the same idea to estimate the aleatoric (input-dependant) uncertainty of the network's prediction, while using dropout as a Bayesian approximation for

the epistemic uncertainty (due to the learning process) as in [**52**]. [**140**] use a re-weighting approach to regression, identifying noisy samples using the Parzen windowing method.

Similarly to [**140**], our method tackles heteroscedastic regression in neural networks using a re-weighting approach. The main distinction between our work and most of the related literature is that, while we do not require that the noise variance is a function of the input or of the label, we do assume that we have access to the noise variance, or at least an estimate of it. In addition, we do not seek to regress the variance of the model's prediction. This is significant compared to the previous works in both regression and classification as it proposes to answer the question: how to best use this additional information to train a deep neural network model?

Note that other regression models, such as Gaussian processes, have the ability to take the uncertainty of the label into account built into their framework. However, they require the design of a kernel to define distances between inputs, which is difficult in high dimensional inputs such as images. Additionally, they scale poorly with the number of samples [**22**].

3.4. Batch Inverse-Variance Weighting

3.4.1. Inverse variance for linear models

The task of heteroscedastic linear regression, where the model is linear, is solved by optimizing a weighted mean square error (WMSE) with inverse-variance weights, which is the optimal solution as per the Gauss-Markov theorem [**143**]:

$$WMSE = \sum_{k=0}^{n} \frac{(y_k - \mathbf{x_k} \cdot \beta)^2}{\sigma_k^2} \tag{3.4.1}$$

where β is the vector of parameters used as linear coefficients. This is also the solution to maximum likelihood estimation for parameters β given the training examples [**47**].

While the solution to such an optimization is known for linear regression ($\beta^* = (\mathbf{x}^T\mathbf{w}\mathbf{x}^{-1})\mathbf{x}^T\mathbf{w}\mathbf{y}$), we could apply it to gradient-based methods for neural networks by using the Inverse Variance (IV) loss function for each sample:

$$\mathcal{L}_{\text{IV}}(\mathbf{x}_k, \tilde{y}_k, \theta) = \frac{(f(\mathbf{x}_k, \theta) - \tilde{y}_k)^2}{\sigma_k^2} \tag{3.4.2}$$

However, using $\mathcal{L}_{\text{IV}}(\mathbf{x}_k, \tilde{y}_k, \theta)$ in a gradient descent setup brings two problems:

(1) Near ground-truth samples, with very small σ_k^2, have a disproportionate learning rate with respect to the others. They risk to cause overfitting while other samples are ignored.

(2) In gradient-based optimizers, the learning rate impacts the optimization process and should be controllable by the practitioner. In IV loss, the average of the sample

weights $1/\sigma_k^2$ acts as a constant multiplied to the learning rate, making such control harder. While this would be less important in adaptive learning rate optimizers such as Adam [**78**] as any constant multiplied to the loss is cancelled, it would still be problematic in continual or reinforcement learning tasks where the noise variance distribution can evolve.

3.4.2. Batch Inverse-Variance weighting for heteroscedastic noisy labels

To address these issues, we introduce the Batch Inverse-Variance (BIV) loss function:

$$\mathcal{L}_{\text{BIV}}(D_i, \theta) = \left(\sum_{k=0}^{K} \frac{1}{\sigma_k^2 + \epsilon} \right)^{-1} \sum_{k=0}^{K} \frac{\mathcal{L}\left(f(\mathbf{x}_k, \theta), \tilde{y}_k\right)}{\sigma_k^2 + \epsilon} \tag{3.4.3}$$

The first distinction with respect to the IV loss of (3.4.2) is that it is computed on the whole batch instead of only being defined on a single sample. This allows to normalize the loss based on the weights of the other samples in the batch. The gradient's scale is thus in dependent from the noise variance distribution, allowing better learning rate control and to use BIV in methods relying on the dynamics of the learning process.

Note that consistency is verified as, when σ_k^2 is identical for each sample, this formulation leads to empirical risk minimization normalized by the number of samples in the mini-batch.

Another difference with the IV loss is the smooth lower bound on the variance, ϵ, which allows to incorporate samples with (near) ground-truth labels without ignoring the other samples. ϵ allows to control the effective batch size EBS, which, according to [**79**], is equal to

$$EBS = \frac{\left(\sum_i^N w_i\right)^2}{\sum_i^N w_i^2} = \frac{\left(\sum_i^N \frac{1}{(\sigma_i^2 + \epsilon)}\right)^2}{\sum_i^N (\frac{1}{(\sigma_i^2 + \epsilon)})^2} \tag{3.4.4}$$

A higher ϵ increases EBS by reducing the relative difference between the weights, thus reducing the effect of BIV. We found that ϵ can be set between 0.01 and 0.1 for normalized losses. More details can be found in appendix A.2.1.

These two elements allow us to overcome the challenges related to inverse variance weighting applied to gradient descent described in section 3.4.1.

3.4.3. Cutoff: filtering heteroscedastic noisy labels

In order to compare the BIV approach to a baseline, we introduce the Cutoff loss. As we do not assume that there is a correlation between $\mathbf{x}_k$ and σ_k^2, most correction and re-weighting algorithms as presented in Section 3.3 are not applicable. Additionally, most

robust loss functions are specifically designed for classification problems. We thus compare BIV to a detection and rejection strategy.

In heteroscedastic regression, an important difference from classification with noisy labels is that all labels are corrupted, albeit not at the same scale. Defining which labels to ignore is therefore a matter of putting a threshold on the variance. As we have access to this information, detection strategies such as the ones used in section 3.3 are not necessary. Instead, we simply use an inverse Heaviside step function as a weight in the loss function:

$$w_k = \mathbf{1}_{\sigma_k^2 < C} \tag{3.4.5}$$

where the threshold C is a hyper-parameter. Similarly to equation (3.4.3), we normalize the loss in the mini-batch by the sum of the weights, equal here to the number of samples considered as valid. As this filtering is equivalent to cutting off a part of the dataset, we refer to this method as 'Cutoff', and consider it to be a relevant baseline to compare BIV against.

3.5. Experimental Setup

To test the validity of the BIV loss (3.4.3) approach, we compared its performance with the classical L2 loss as well as cutoff loss (3.4.5) on two datasets. We refer to ground-truth (GT) labels when training with L2 on noise-less data as the best performance that could be achieved on this dataset.

Unfortunately, we did not find any existing dataset for regression where label uncertainty is associated to the samples. We therefore used two UCI datasets [**42**] for regression cases, and artificially added noise to them. UTKFace Aligned&Cropped [**149**] (UTKF), is a dataset for image-based age prediction. In the Bike Sharing dataset [**45**], the task is to predict the number of bicycles rented in Washington D.C., from structured data containing the date, hour, and weather conditions. For UTKF, a convolutional neural network was used to predict the age, while a simple multi-layer perceptron was used for BikeSharing. More details about the datasets and models can be found in appendix A.1.

3.5.1. Noise generation

To produce the datasets $\{\mathbf{x}_k, \sigma_k^2, \tilde{y}_k\}$ with noise as described in section 3.2, we use a two-step process which does not assume any correlation between the noise and the state.

(1) The noise variance σ_k^2 is sampled from a distribution $P(\sigma^2)$ which only has support for $\sigma^2 \geq 0$

(2) $\tilde{y}_k$ is sampled from a normal distribution $\mathcal{N}(y_k, \sigma_k^2)$.

$P(\sigma^2)$ has a strong effect on the impact of BIV or Cutoff. For example, if it is a Dirac delta and all variances are the same, BIV becomes L2. We evaluate BIV on three different

types of $P(\sigma^2)$. The average noise variance μ_P was chosen empirically so that the lowest test loss achieved by L2 is doubled compared to the ground-truth label case: $\mu_P = 2000$ for UTKF and 20000 for BikeSharing.

Uniform distribution. The uniform distribution is characterized by its bounds a,b. Its expected value μ_P is the average of its bounds, and its variance $V = (b-a)^2/12$. As P only has support for $\sigma^2 \geq 0$, the maximum variance $V_{\max}$ is when $a = 0$ and $b = 2\mu_P$. While such a distribution is not realistic, it is simple conceptually and allows for interesting insights.

"Binary uniform". A more realistic distribution, which can also help us understand the effects of BIV, is when the data is generated from two regimes: low and high noise. We call the "binary uniform" distribution a mixture of two uniform distributions balanced by parameter p. With probability p, the label is in a low noise regime: $\sigma^2 \sim U(0,1)$, with expected value $\mu_l = 0.5$. With probability $1-p$, the label is in a high noise regime: $\sigma^2 \sim U(a_h,b_h)$. a_h and b_h are chosen such that the average is μ_h and the variance of the high-noise distribution is determined by $V_h \in [0, V_{\max}]$. Note that, if we want a given expected value of the whole distribution μ_P, the value of μ_h changes depending on p: $\mu_h = (\mu_P - p\mu_l)/(1-p)$

Therefore, the high-noise expected value μ_h of the noise variance σ^2 in a distribution with high p will be higher than the one for a low p, for the same value of μ_P. In other words, a higher p means more chance to be in the low-noise regime, but the high-noise regime is noisier.

Gamma distributions. While the mixture of 2 uniform distributions ensures support in the low noise region, it is not continuous. We therefore also propose to use a Gamma distribution with shape parameter α. If we want to control the expected value μ_P, we adjust $\beta = \alpha/\mu_P$.

For a fixed expected value μ_P, lower α and β mean that there is a stronger support towards low variance noise, but the tail of the distribution spreads longer on the high-noise size. In other words, a lower α means more chances to have low-noise samples, but when they are noisy, the variance is higher. When $\alpha \leq 1$, the highest support is at $\sigma^2 = 0$.

3.5.2. Evaluating the performance of the model

The objective of BIV is to improve the performance at predicting the true label y_i. While a non-noisy test dataset may not be available in a real application, we aimed here at determining if BIV performs better than L2, and therefore measured the performance of the network using mean square error on ground-truth test data.

3.6. Experimental Results and Analysis

3.6.1. For L2 loss, mean variance is all that matters

Before looking at the results for BIV, we share an interesting insight for L2 loss with noisy labels which helps simplifying the analysis of the results. Under the unbiased, heteroscedastic Gaussian-based noise model presented in section 3.5.1, the only parameter of distribution $P(\sigma)$ that mattered to describe the performance of the L2 loss is its average μ_P, which is also the variance of the overall noise distribution. Independently of the distribution type, and the values of V, p and V_h, or α, as long as μ_P is equal, the L2 loss trained neural networks had the same performance. This is shown in Figure 1. For the sake of clarity, all the curves in this section were smoothed using moving average with a 35 steps window, and the shaded area represents the standard deviation over several runs.

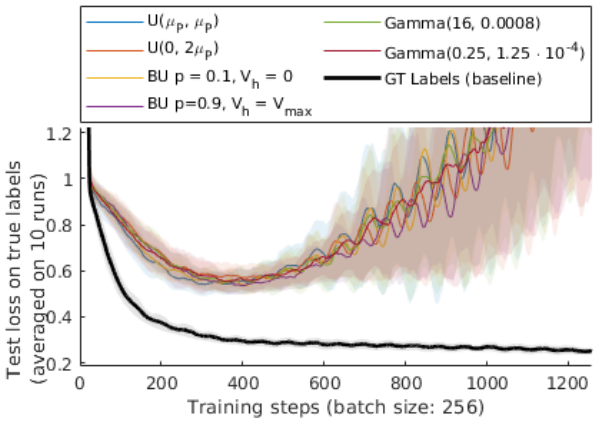


Fig. 1. Similar performance of the L2 loss on UTKF for different $P(\sigma^2)$ with constant $\mu_P = 2000$. No matter the distribution type or parameters, the performance is similar.

3.6.2. High and low variance noise regimes: BIV acts as a filter

With the binary uniform distribution, the noise is split in two regimes, with high or low variances. In this case, our results show that BIV performs better than L2, and actually similarly to the cutoff loss presented in section 3.4.3 with a threshold $C = 1$.

Figure 2 compares the test losses on UTKF with different values of p for $V_h = 0$. While the L2 curves are strongly impacted by the noise, both the BIV and cutoff losses lead to better and very similar performances for a given p. When $p = 0.3$, there are not a lot of information that can be used, and the performance is still impacted by the noise. When $p = 0.9$, there is nearly as much near-ground-truth data as in the noiseless case, and the performance is comparable.

In the case of binary uniform distributions, BIV is acting as a filter, cutting off labels which are too noisy to contain any useful information.

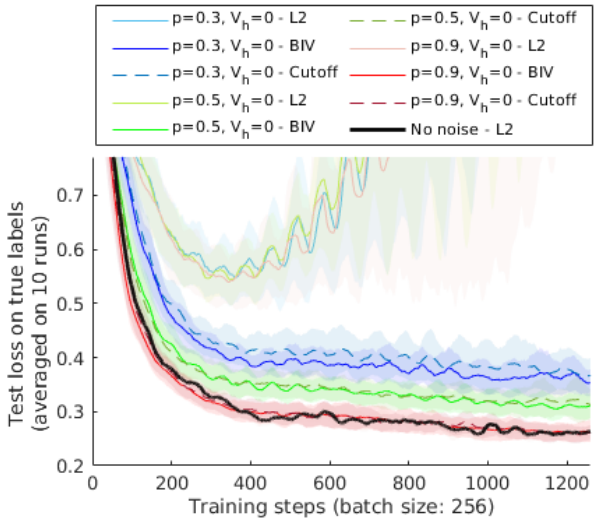


Fig. 2. Comparison between BIV, Cutoff and L2 losses for binary uniform distributions of variance with different ps for $\mu_P = 2000$ and $V_h = 0$ on UTKF.

3.6.3. Continuous, decreasing noise variance distribution: the advantage of BIV

On Gamma distributions, there is no clear threshold to define which information to use. When $\alpha \leq 1$, BIV shows a strong advantage compared to both L2 and cutoff. Figure 3 shows the results in both the BikeSharing and the UTKF datasets for Gamma distributions with $\alpha = 1$.

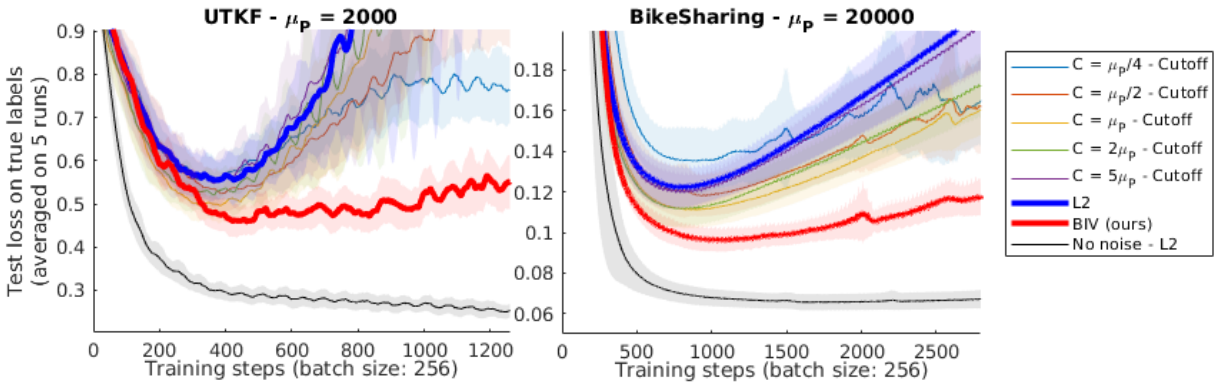


Fig. 3. Comparison between the performances of BIV, L2, and different cutoff values on both datasets where the noise variance follows a Gamma distribution with $\alpha = 1$.

In both datasets, when the cutoff parameter C is too low ($\mu_P/4$ and $\mu_P/2$), there is not enough data to train the model. When C is too high ($2\mu_P$ and $5\mu_P$), the data is too noisy and the curves go close to the original L2 loss. Even at the best case ($C = \mu_P$), cutoff is not better than BIV. This is because, in contrast to cutoff, BIV is able to extract some information from noisier samples while avoiding to over-fit on them.

In Table 1, we present the lowest value of the test loss curves for the different methods with other α parameters for the Gamma distributions over both datasets. BIV consistently leads to the best performances, regardless of $P(\sigma^2)$. The plots showing these runs can be found in the appendix A.2.3.1. BIV is less sensitive to hyperparameters than cutoff, as it avoids the need to choose the right cutoff parameter for each distribution $P(\sigma^2)$. BIV's own hyperparameter ϵ can be set between 0.01 and 0.1 for any dataset with a normalized output, as shown in appendix A.2.1, and be ready to use. As ϵ can be seen as a minimal variance, scaling it for other label distributions is straightforward: it suffices to multiply it by the variance of the label distribution. Here, it was set at 0.05.

Table 1. Lowest test loss for different α on two datasets, for BIV, L2 and several cutoff losses. The test loss with standard deviation is computed as the average over 5 runs. In every case, BIV loss led to the lowest value. The best C value differs based on α.

	$\alpha = 1$		$\alpha = 0.5$		$\alpha = 0.25$	
	UTKF	Bike	UTKF	Bike	UTKF	Bike
$C = \mu_P/20$	0.79±.08	0.327±.056	0.48±.04	0.125±.010	0.38 ±.05	0.092±.008
$C = \mu_P/4$	0.55±.04	0.135±.016	0.45±.04	0.097±.006	0.39±.03	0.085±.006
$C = \mu_P$	0.50±.04	0.111±.009	0.48±.04	0.097±.008	0.43±.03	0.088±.006
$C = 5\mu_P$	0.55±.06	0.120±.009	0.54±.05	0.111±.012	0.51±.04	0.107±.009
L2	0.56±.05	0.122±.010	0.56±.05	0.116±.011	0.55±.05	0.119 ±.012
BIV (ours)	**0.46**±.03	**0.096**±.006	**0.40**±.03	**0.088**±.006	**0.33**±.02	**0.079**±.006
IV	0.99±.03	0.120±.021	1.65±.03	0.554±.041	N.A.	N.A.
GT labels	0.25±.02	0.066±.004	0.25±.02	0.066±.004	0.25±.02	0.066±.004

The benefit of BIV over L2 is clearly higher when α is lower. This is due to an increase in the support for low-variance noise in $P(\sigma^2)$. The more BIV can count on low-noise elements and differentiate them from high noise ones, the better it can perform. This is consistent with results from section 3.6.2, and with other experiments we have run. For example, when $\alpha > 1$, the highest support of P is not at $\sigma^2 = 0$. BIV was less able to improve the performance compared to L2.

We also ran the experiment with uniform distributions: the performance is better when variance V is closer to V_{max} (and a to 0). But even when $V = V_{max}$, as there is less support in low noise variance than for Gamma distributions with $\alpha \leq 1$, the improvement is less important.

In all cases, BIV was performing consistently better than L2 and at least better than cutoff in all the experiments we ran. More details on these results can be found in appendix A.2.2.

3.6.4. Ablation study: comparison to IV

While we have shown in the previous parts that inverse variance weighting is indeed performing better than L2 or cutoff, we focus here on the role of both normalization and ϵ in BIV loss (3.4.3) compared to a simple IV loss (3.4.2) using an ablation study. The results of using IV on a Gamma distributions with α values of 1, 0.5 and 0.25 are shown in table 1. It is clear that BIV performs better, but also that IV is struggling when α is small, which is when there is a higher probability that a sample's variance is very close to zero. Actually, the training process would diverge for $\alpha = 0.25$ on every seed for both BikeSharing and UTKF datasets.

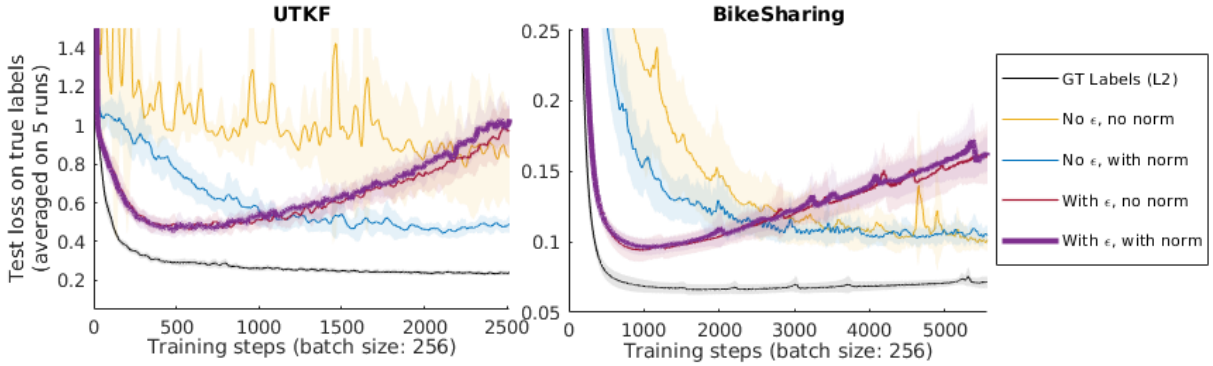


Fig. 4. Ablation study for ϵ and normalization for both datasets where $P(\sigma^2)$ is a Gamma distribution with $\alpha = 1$.

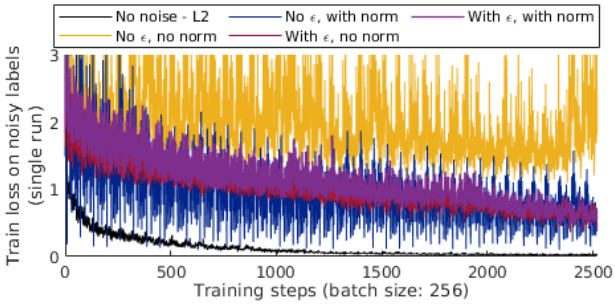


Fig. 5. Training loss for ablation study for ϵ and normalization for both datasets where $P(\sigma^2)$ is a Gamma distribution with $\alpha = 1$. Note that, because the normalization value depends on ϵ, these curves should not be compared quantitatively but quantitatively, by looking at their respective variability.

More details can be seen in figure 4. Here, the advantage of BIV, with ϵ and normalization, is clear over IV, without ϵ nor normalization. The effect of ϵ is significant. It induces not only the lowest test loss, but also learns faster by preventing low-variance samples to create high variability in the training loss. In figure 5 showing the training loss, it is evident that with ϵ, the variability of the loss is smaller. The impact of normalization is more subtle.

In this case, the optimizer we used was Adam [**78**]. Such optimizers are insensitive to a constant multiplied to the loss function, which is cancelled in the update process. Therefore, if the normalization constant is actually similar in every batch, it is equivalent to simply normalizing with the number of samples. This is what is happening when $\epsilon \neq 0$: the size of the minibatches and the limited variation in the noise variances makes this normalization close to effect-less. However, this is not the case when $\epsilon = 0$, where some mini-batches include such a small-variance sample that it makes the sum of the weights a lot higher than in other batches, which are actually ignored. With normalization, the effect of this near-ground truth sample is limited to the minibatch itself, which still allows for some learning when mini-batches are reshuffled and the samples have a chance to be taken into account at another epoch.

In the case of another optimizer such as stochastic gradient descent, the effect of normalization could be a lot more important, as it would also regulate the learning rate, regardless of the average sum of weights, and thus the distribution of noise variances in the dataset.

3.6.5. Robustness

We identified two factors that may impact the performances of BIV: the size of the mini-batches and the accuracy of the noise variance estimation. We tested the robustness of BIV when these factors are different than during the previous experiments.

Size of the mini-batches. In equation 3.4.3, each weight is normalized based on the assumption that the distribution of noise variances in the mini-batch is representative of the whole training dataset. With smaller mini-batches, the variability of the normalization constant between mini-batch should be bigger. However, this does not affect BIV, which still performs very well in these cases, as presented in section A.2.4.1.

Noisy variances. Because the noise variance σ_i^2 is often estimated, the method needs to be robust to errors in σ_i^2's. A model for the noise of σ_i^2 can be a Gaussian for which the variance is proportional to σ_i^2. In this case, results show that the effect of moderate to high levels of noise on BIV is not significant. More details can be seen in section A.2.4.2

3.7. Conclusion

We have proposed Batch Inverse-Variance, a mini-batch based approach to account for the variance of the label noise in heteroscedastic regression tasks for neural networks. BIV is able to extract more information from the noisy dataset than L2 loss or threshold-based filtering approaches, and consistently outperforms them on both structured and unstructured datasets, by learning faster and reaching a lower test loss. The method is robust to low-variance, near ground-truth samples, which is not the case for naively applying an

inverse-variance loss. It also enables reliable control of the learning rate, which is particularly important in tasks such as continual or reinforcement learning. BIV can be easily implemented in any regression setup, making it a versatile and ready to use algorithm.

An element which would need further investigation is the impact of such weights on the performance of the model when the noise distribution is not uniformly distributed in the input space, to ensure that some regions of the input space are not doubly penalized by a higher level noise and a slower learning compared to the other regions.

Chapter 4

Prologue to Article 2

This article was published and presented as a spotlight paper at the Tenth International Conference on Learning Representations (ICLR) 2022. A preliminary version was presented at the RL4RealLife workshop at ICML 2021. It was a joint work with Kaustubh Mani and Liam Paull. Kaustubh Mani was then an intern at the Robotics and Embodied AI Lab.

In this article, we propose to use uncertainty prediction in regression and the BIV loss presented in Article 1 to improve the sample efficiency of deep reinforcement learning.

The idea for this paper came after following the reinforcement learning class of Prof. Doina Precup at McGill University. With my background in state estimation for robotics, I could not help but notice the parallels between the policy evaluation update from a target in RL and the belief update from a measurement in a recursive estimator such as a Kalman filter. Recursive estimation measurement updates are significantly more precise when they consider the uncertainty of the measurement This paper aims at applying the same concept to RL.

Contribution statement: I produced the core idea behind this work and developed the BIV loss as a first step towards this project. The algorithm was developed jointly with Kaustubh Mani. We implemented it step-by-step, testing every design choice before moving forward. Kaustubh mainly worked on the code, while I was designing, running and analyzing the intermediate experiments. The final experiments and analyzis were doing jointly with Kaustubh. Liam Paull supervised the process, provided guidance and advice, and gave several ideas. I prepared the manuscript and the rebuttal, reviewed by Kaustubh and Liam. Kaustubh prepared the code so that it is open source, tested, and documented. Finally, I prepared the poster and videos presenting the paper online at ICML's workshop and ICLR 2022.

Chapter 5

Article 2: Sample Efficient Deep Reinforcement Learning via Uncertainty Estimation

Résumé

Dans les algorithmes d'apprentissage par renforcement (RL) profonds sans modèle, l'utilisation d'estimations de valeur bruitées pour superviser l'évaluation et l'optimisation des politiques nuit à l'efficacité en matière d'échantillonnage. Comme ce bruit est hétéroscédastique, ses effets peuvent être atténués en utilisant des pondérations basées sur l'incertitude dans le processus d'optimisation. Les méthodes existantes reposent sur des ensembles échantillonnés, qui ne capturent pas tous les aspects de l'incertitude. Nous fournissons une analyse systématique des sources d'incertitude dans la supervision bruyante qui se produit en RL, et nous introduisons le RL à variance inverse, un cadre bayésien qui combine les ensembles probabilistes et la pondération Batch Inverse Variance. Nous proposons une méthode où deux méthodes complémentaires d'estimation de l'incertitude tiennent compte à la fois de l'incertitude sur la valeur Q et de la stochasticité de l'environnement afin de mieux atténuer les impacts négatifs de la supervision bruitée. Nos résultats montrent une amélioration significative en termes d'efficacité en matière d'échantillonnage sur des tâches de contrôle discrètes et continues.

Mots clés: apprentissage par renforcement, efficacité en matière d'échantillonage, estimation d'incertitude

Abstract

In model-free deep reinforcement learning (RL) algorithms, using noisy value estimates to supervise policy evaluation and optimization is detrimental to the sample efficiency. As this noise is heteroscedastic, its effects can be mitigated using uncertainty-based weights in the

optimization process. Previous methods rely on sampled ensembles, which do not capture all aspects of uncertainty. We provide a systematic analysis of the sources of uncertainty in the noisy supervision that occurs in RL, and introduce inverse-variance RL, a Bayesian framework which combines probabilistic ensembles and Batch Inverse Variance weighting. We propose a method whereby two complementary uncertainty estimation methods account for both the Q-value and the environment stochasticity to better mitigate the negative impacts of noisy supervision. Our results show significant improvement in terms of sample efficiency on discrete and continuous control tasks.
Key words: reinforcement learning, sample efficiency, uncertainty estimation

5.1. Introduction

Deep reinforcement learning (DRL) methods have proven to be powerful at solving sequential decision-making tasks across domains [**147, 123**]. Combining the flexibility of the reinforcement learning framework with the representational power of deep neural networks enables policy optimization in complex and high-dimensional environments with unknown dynamics models to maximize the expected cumulative reward [**154**].

An important limitation of DRL methods is their sample inefficiency: an enormous amount of data is necessary and makes training expensive. This makes applying DRL in the real world challenging, for example in robotics [**155, 43**]. In tasks like manipulation, sample collection is a slow and costly process [**100**]. It is even more expensive in risk-averse applications like autonomous driving [**80**].

Among the current state-of-the-art approaches to improve learning efficiency, a promising direction is to exploit the prevalence of uncertainty in the underlying DRL algorithm. By adopting a Bayesian framework, we can consider the sampled quantities in DRL as random variables and leverage information about their distributions to improve the learning process [**124**]. In this paper, we consider the particular problem of unreliable supervision in the temporal difference update and the policy optimization process. In DRL, value predictions are used to supervise the training: in temporal difference-based algorithms, they are included in bootstrapped target values which are used as labels; in actor-critic frameworks, the policy is trained to optimize them. That these value predictions are noisy slows the learning and brings instability [**84, 85**]. The amount of noise in the supervision depends on the uncertainty of the value prediction, which evolves during the training process and depends on the state (and action) evaluated. It is therefore *heteroscedastic*.

While there is an extensive body of literature focused on using the uncertainty of the value prediction to guide the exploration/exploitation trade-off [**40, 150, 125, 129, 32, 124, 50, 127, 48, 36, 72, 9**], there are very few works focused on leveraging it to mitigate the impact of unreliable supervision.

Distributional RL [**16**] considers the value function as a distribution to be learned as such. It is orthogonal to our proposition: we consider the uncertainty of the labels used to learn a scalar value function. In the offline RL setting, where the dataset is limited, uncertainty-weighted actor-critic (UWAC) [**171**] uses inverse-variance weighting to discard out-of-distribution state-action pairs using Monte Carlo dropout [**52**] for uncertainty estimation.

Closer to our work, [**92**] propose SUNRISE, in which each sample of the Bellman backup in the TD update step is weighted to lower the importance of the targets which have a high standard deviation. The weights $w(s',a')$ are computed based on a sigmoïd of the negative standard deviation $\hat{Q}_{\text{std}}(s',a')$ scaled by a temperature hyperparameter T, and then offset such that they are between 0.5 and 1: $w(s,a) = \sigma(-\hat{Q}_{\text{std}}(s',a') * T) + 0.5$. The uncertainty of the target is estimated by sampled ensembles. While SUNRISE proposes other contributions such as an exploration bonus, the heuristic weighting scheme and the limitations of sampled ensembles in capturing the predictive uncertainty leave space for improvement in the mitigation of the effects of unreliable supervision.

We propose inverse-variance reinforcement learning (IV-RL). IV-RL also uses weights to reduce the importance of uncertain targets in training. It does so by addressing the problem from two viewpoints. First, we use variance networks [**77**], whose loss function for regression is the negative log-likelihood instead of the L2 distance. For a given state-action pair (s,a), the network learns the target's noise, due for example to the stochasticity of the environment or the update of the policy. It then naturally down-weights the highly noisy samples in the training process. Second, we use variance ensembles [**90**] to estimate the uncertainty of the target due to the prediction of $Q(s',a')$ during the temporal-difference update. We merge the predicted variances of several variance networks through a mixture of Gaussians, which has been shown to be a reliable method to capture predictive uncertainty [**128**]. We then use Batch Inverse-Variance (BIV) [**109**], which has been shown to significantly improve the performance of supervised learning with neural networks in the case of heteroscedastic regression. BIV is normalized, which makes it ideal to cope with different and time-varying scales of variance. We show analytically that these two different variance predictions for the target are complementary and their combination leads to consistent and significant improvements in the sample efficiency and overall performance of the learning process.

In summary, our contribution is threefold:

(1) We present a systematic analysis of the sources of uncertainty in the supervision of model-free DRL algorithms. We show that the variance of the supervision noise can be estimated with two complementary methods: negative log-likelihood and variance ensembles.

(2) We introduce IV-RL, a framework that accounts for the uncertainty of the supervisory signal by weighting the samples in a mini-batch during the agent's training. IV-RL uses BIV, a weighting scheme that is robust to poorly calibrated variance estimation.[1]
(3) Our experiments show that IV-RL can lead to significant improvements in sample efficiency when applied to Deep Q-Networks (DQN) [**113**] and Soft-Actor Critic (SAC) [**61**].

In section 5.2, we introduce BIV as a weighting scheme for heteroscedastic regression, and variance ensembles as an uncertainty estimation method. We analyse the sources of uncertainty in the target in section 5.3, where we also introduce our IV-RL framework. We finally present our experimental results in section 5.4.

5.2. Background and preliminaries

5.2.1. Batch Inverse-Variance Weighting

In supervised learning with deep neural networks, it is assumed that the training dataset consists of inputs x_k and labels y_k. However, depending on the label generation process, the label may be noisy. In regression, we can model the noise as a normal distribution around the true label: $\tilde{y}_k = y_k + \delta_k$ with $\delta_k \sim \mathcal{N}(0, \sigma_k^2)$. In some cases, the label generation process leads to different variances for the label noises. When these variances can be estimated, each sample is a triplet $(x_k, \tilde{y}_k, \sigma_k^2)$.

Batch Inverse-Variance (BIV) weighting [**109**] leverages the additional information σ_k^2, which is assumed to be provided, to learn faster and obtain better performance in the case of heteroscedastic noise on the labels. Applied to L2 loss, it optimizes the neural network parameters θ using the following loss function for a mini-batch D of size K [2]:

$$\mathcal{L}_{\text{BIV}}(D, \theta) = \left(\sum_{k=0}^{K} \frac{1}{\sigma_k^2 + \xi} \right)^{-1} \sum_{k=0}^{K} \frac{(f_\theta(x_k) - \tilde{y}_k)^2}{\sigma_k^2 + \xi} \tag{5.2.1}$$

This is a normalized weighted sum with weights $w_k = 1/(\sigma_k^2 + \xi)$. Normalizing in the mini-batch enables control of the effective learning rate, especially in cases where the training data changes over time, such as in DRL. By focusing on the relative scale of the variances instead of their absolute value, it also provides robustness to poor scale-calibration of the variance estimates.

As explained in [**109**], ξ is a hyperparameter that is important for the stability of the optimization process. A higher ξ limits the highest weights, thus preventing very small variance samples from dominating the loss function for a mini-batch. However, by controlling the discrimination between the samples, ξ is also key when the variance estimation is not

[1] The code for IV-RL is available at `https://github.com/montrealrobotics/iv_rl`.
[2] We replaced ϵ in [**109**] with ξ to avoid confusion with the reinforcement learning conventions.

completely trusted. It provides control of the effective mini-batch size EBS, according to:

$$EBS = \frac{\left(\sum_k^K w_k\right)^2}{\sum_k^K w_k^2} = \frac{\left(\sum_k^K \frac{1}{(\sigma_k^2 + \xi)}\right)^2}{\sum_k^K \frac{1}{(\sigma_k^2 + \xi)^2}} \tag{5.2.2}$$

For example, imagine a mini-batch where most samples have very high variances, and only one has a very low variance. If $\xi = 0$, this one low-variance sample is effectively the only one to count in the mini-batch, and EBS tends towards 1. Increasing ξ would give more relative importance to the other samples, thus increasing EBS. With a very high ξ compared to the variances, all weights are equal, and EBS tends towards K; in this case, the BIV loss tends towards $L2$.

Tuning the ξ parameter. The simplest way to set ξ is to choose a constant value as an additional hyperparameter. However, the best value is difficult to evaluate a priori and can change when the profile of variances changes during a task, as is the case in DRL.

It is instead possible to numerically compute the value of ξ which ensures a minimal EBS for each mini-batch. This method allows ξ to automatically adapt to the different scales of variance while ensuring a minimal amount of information from the dataset is accounted for by the algorithm. The minimal EBS is also a hyper-parameter, but it is easier to set and to transfer among environments, as it is simply a fraction of the original batch size. As such, it can be set as a batch size ratio.

5.2.2. Estimating the uncertainty of a neural network prediction

The predictive uncertainty of a neural network can be considered as the combination of aleatoric and epistemic uncertainties [**77**]. Aleatoric uncertainty is irreducible and characterizes the non-deterministic relationship between the input and the desired output. Epistemic uncertainty is instead related to the trained model: it depends on the information available in the training data, the model's capacity to retain it, and the learning algorithm [**70**]. There is currently no principled way to quantify the amount of task-related information present in the input, the training data, or the model. The state of the art for predictive uncertainty estimation instead relies on different sorts of proxies. These sometimes capture other elements, such as the noise of the labels, which we can use to our advantage. We focus here on the relevant methods to our work.

5.2.2.1. Sampled ensembles. Several networks independently train an ensemble of size N that can be interpreted as a distribution over predictions. The expected behaviour is that different networks will only make similar predictions if they were sufficiently trained for a given input. The sampled variance of the networks' outputs is thus interpreted as epistemic uncertainty. It is possible to include a random Bernoulli mask of probability p to

each training sample, to ensure that each network undergoes different training. This method, used by [**36**] and [**92**], has the same principle as single network Monte-Carlo dropout [**52**]. As the variance is sampled, the standard deviation is usually on the same scale as the prediction.

When used at the very beginning of the training process, sampled ensembles present one particular challenge: as the networks are initialized, they all predict small values. The initial variance, instead of capturing the lack of knowledge, is then underestimated. To address this problem, Randomized Prior Functions (RPFs) enforce a prior in the variance by pairing each network with a fixed, untrained network which adds its predictions to the output [**127**]. RPFs ensure a high variance at regions of the input space which are not well explored, and a lower variance when the trained networks have learned to compensate for their respective prior and converge to the same output. The scale of the prior is a hyper-parameter.[3]

5.2.2.2. Variance networks. With variance networks, the uncertainty is predicted using loss attenuation [**120, 77**]. A network outputs two values in its final layer given an input x: the predicted mean $\mu(x)$ and variance $\sigma^2(x)$. Over a minibatch D of size K, the network parameters θ are optimized by minimizing the negative log-likelihood of a heteroscedastic Gaussian distribution:

$$\mathcal{L}_{LA}(D,\theta) = \sum_{k=0}^{K} \frac{1}{K} \frac{\left(\mu_\theta(x_k) - y(x_k)\right)^2}{\sigma_\theta^2(x_k)} + \ln \sigma_\theta^2(x_k) \tag{5.2.3}$$

Variance networks naturally down-weight the labels with high variance in the optimization process. The variance prediction is trained from the error between $\mu_\theta(x)$ and $y(x)$. Hence, noise on the labels or changes in the regression task over time will also be captured by loss attenuation. As the variance is predicted by a neural network, it may not be well-calibrated and may be overestimated [**83, 96, 20**]. This can (1) give wrong variance estimates but also (2) affect the learning process by ignoring a sample if the variance estimate is too high.

5.2.2.3. Variance ensembles. Variance ensembles, or deep ensembles [**90**], are ensembles composed of variance networks. The predictive variance is given by a Gaussian mixture over the variance predictions of each network in the ensemble. This method, when trained, is able to capture uncertainty more reliably than others [**128**], with $N = 5$ networks in the ensemble being sufficient. Similarly to sampled ensembles, variance ensembles suffer from underestimated early epistemic variance estimation. However, compared to variance networks, they seem empirically less prone to calibration issues (1) in the final variance estimation because the mixture of Gaussians dampens single very high variances, and (2) in the learning process because even if one network does not learn correctly, the others will.

[3]Using RPFs in the context of DRL can destabilize the exploration due to a significant initial bias added to the value of each state-action. It must be compensated for with uncertainty-aware exploration, such as Bootstrap [**125**], where it leads to a significant improvement in performance for exploration-based tasks.

For better readability, we will use the terms var-network and var-ensemble in the rest of the paper.

5.2.3. Uncertainty and exploration in DRL

While our work focuses on using uncertainty estimates to mitigate the impact of unreliable supervision, we can take advantage of the structure in place to better drive the exploration/exploitation trade-off. In particular, we used BootstrapDQN [**125**] for exploration. In this method, a single network is sampled from an ensemble at the beginning of each episode to select the action. This method is improved with the previously described RPFs [**124**]. In continuous settings, we instead followed [**92**] and added an Upper Confidence Bound (UCB) exploration bonus based on uncertainty prediction. As the variance in UCB is added to Q-values, it must be calibrated: we evaluate it with sampled ensembles. Possible interactions between exploration strategies and IV-RL are discussed in Appendix B.9.

5.3. Inverse-Variance Reinforcement Learning

5.3.1. Target uncertainty in Reinforcement Learning

Many model-free DRL algorithms use temporal difference updates. In methods such as DQN [**113**], PPO [**142**] and SAC [**61**], a neural network is trained to predict the Q-value of a given state-action pair $Q^\pi(s,a)$ [4] by minimizing the error between the target $T(s,a)$ and its prediction $\hat{Q}(s,a)$. $T(s,a)$ is computed according to Bellman's equation:

$$T(s,a) = r + \gamma \bar{Q}(s',a') \tag{5.3.1}$$

s' and r are sampled from the environment given (s,a), and a' is sampled from the current policy given s'. $\bar{Q}(s',a')$ is predicted by a copy of the Q-network (called the target network) which is updated less frequently to ensure training stability. The neural network's parameters θ are optimized using stochastic gradient descent to minimize the following $L2$ loss function:

$$\mathcal{L}_\theta = \left|\left|T(s,a) - \hat{Q}_\theta(s,a)\right|\right|^2 \tag{5.3.2}$$

5.3.1.1. The target as a random variable. The target $T(s,a)$ is a noisy approximation of $Q^\pi(s,a)$ that is distributed according to its distribution $p_T(T|s,a)$. The generative model used to produce samples of $T(s,a)$ is shown in Figure 1, and has the following components:

(1) if the reward r is stochastic, it is sampled from $p_R(r|s,a)$ [5];

(2) if the environment dynamics are stochastic, the next state s' is sampled from $p_{S'}(s'|s,a)$;

[4] In this paper, we focus on Q-values, although the principle can also apply to state values.

[5] Stochastic reward can happen for example if the reward is corrupted or estimated [**138, 27**]

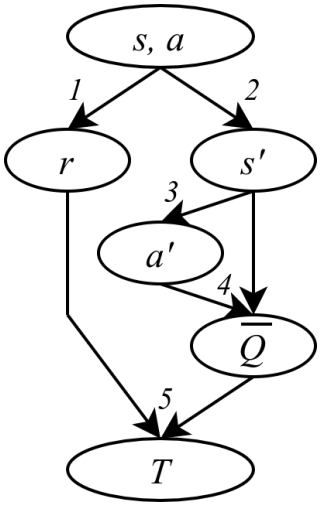


Fig. 1. Bayesian network representing the target sampling process

(3) if the policy is stochastic a' is sampled from the policy $\pi(a'|s')$;
(4) $\bar{Q}$ is a prediction from a var-network $p_{\bar{Q}}(\bar{Q}|s',a')$;
(5) T is deterministically generated from r and $\bar{Q}$ by equation (5.3.1).

As the variance of the noise of $T(s,a)$ is not constant, training the Q-network using $\mathcal{L}_\theta$ as in equation (5.3.2) is regression on heteroscedastic noisy labels.

5.3.1.2. Variance of the target. As seen in section 5.2.1, BIV can be used to reduce the impact of heteroscedastic noisy labels in regression, provided estimates of the label variances. We thus aim to evaluate $\sigma_T^2(T|s,a)$ based on the sampling process described in section 5.3.1.1. r and Q-value estimation are independent given s and a, hence:

$$\sigma_T^2(T|s,a) = \sigma_R^2(r|s,a) + \gamma^2 \sigma_{S'A'\bar{Q}}^2\left(\bar{Q}|s,a\right) \tag{5.3.3}$$

where $p_{S'A'\bar{Q}}$ is the compound probability distribution based on components 2-4 in Figure 1:

$$p_{S'A'\bar{Q}}(\bar{Q}|s,a) = \iint p_{\bar{Q}}(\bar{Q}|s',a') p_{S'A'}(s',a'|s,a) da' ds' \tag{5.3.4}$$

where $p_{S'A'}(s',a'|s,a) = p_{A'}(a'|s') p_{S'}(s'|s,a)$. Using the law of total variance, the variance of $\bar{Q}$ is given by:

$$\sigma_{S'A'\bar{Q}}^2\left(\bar{Q}|s,a\right) = \mathbb{E}_{S'A'}\left[\sigma_{\bar{Q}}^2\left(\bar{Q}|s',a'\right)|s,a\right] + \sigma_{S'A'}^2\left(\mathbb{E}_{\bar{Q}}\left[\bar{Q}|s',a'\right]|s,a\right) \tag{5.3.5}$$

Plugging (5.3.5) into (5.3.3) gives:

$$\sigma_T^2(T|s,a) = \gamma^2 \underbrace{\left(\mathbb{E}_{S'A'}\left[\sigma_{\bar{Q}}^2\left(\bar{Q}|s',a'\right)|s,a\right]\right)}_{\text{Predictive variance of Q-network}} + \underbrace{\gamma^2\left(\sigma_{S'A'}^2\left(\mathbb{E}_{\bar{Q}}\left[\bar{Q}|s',a'\right]|s,a\right)\right) + \sigma_R^2(r|s,a)}_{\text{Policy and environment induced variance}} \tag{5.3.6}$$

We can identify two distinct components in equation 5.3.6 that contribute to the overall variance of the target. The first is the (expectation of the) variance that is due to the

uncertainty in the neural network prediction of the value function, $\mathbb{E}_{S'A'}[\sigma^2_{\bar{Q}}\left(\bar{Q}|s',a'\right)]$. The second is the uncertainty due to the stochasticity of the environment and of the policy, $\sigma^2_R(r|s,a) + \gamma^2\sigma^2_{S'A'}\left(\mathbb{E}_{\bar{Q}}\left[\bar{Q}|s',a'\right]\right)$.

Uncertainty in neural network prediction. For a given policy π and a given s',a', the agent may not have seen enough samples to have an accurate approximation of $Q^\pi(s',a')$. This corresponds to an epistemic source of uncertainty that should be captured by sampled ensembles. However, as π is updated, the regression target, $Q^\pi(s',a')$, is also changing. This can be interpreted as variability in the underlying process which will be captured by var-networks. We can thus combine both sampling-based and var-network methods and evaluate $\sigma^2_{\bar{Q}}\left(\bar{Q}(s',a')\right)$ with var-ensembles.

We assume that the estimate of $\sigma^2_{\bar{Q}}(\bar{Q}|s',a')$ given a sampled (s',a') is unbiased and can therefore use it to directly approximate the expectation $\mathbb{E}_{S'A'}[\sigma^2_{\bar{Q}}\left(\bar{Q}|s',a'\right)]$. These values are used in the BIV loss $\mathcal{L}_{BIV}$ (equation 5.2.1) across a mini-batch sampled from the replay buffer. In this case, ξ is used to control the trust in the variance estimation, as explained in section 5.2.1.

Stochastic environment and policy. The other potential source of variance in the target is the result of the stochasticity of the environment encapsulated by $p_R\left(r|s,a\right)$ and $p_{S'}(s'|s,a)$ and of the policy represented by $\pi(a'|s')$. Note that in model-free RL, we have no explicit representation of $p_R\left(r|s,a\right)$ or $p_{S'}(s'|s,a)$, which are necessary to estimate this source of uncertainty.

$Q^\pi(s,a)$ is defined as the expected value of the return. As a result, even in the case where $\bar{Q}(s',a') = Q^\pi(s,a)$ in (5.3.1), there is still noise over the value of the *target* that is being used as the label due to the stochasticity of the environment and policy that generate r, a' and s'. Assuming a zero mean and normally distributed noise, this underlying stochasticity of the generating process is well-captured by a var-network with a loss attenuation using the negative log-likelihood formulation described in equation (5.2.3).

5.3.1.3. Loss function for IV-RL. Based on the motivation above, we propose our IV-RL loss over minibatch D of size K as a linear combination of $\mathcal{L}_{\text{BIV}}$ and $\mathcal{L}_{\text{LA}}$, balanced by hyperparameter λ:

$$\begin{aligned}\mathcal{L}_{\text{IVRL}}(D,\theta) &= \mathcal{L}_{\text{BIV}}(D,\theta) + \lambda\mathcal{L}_{\text{LA}}(D,\theta)\\ &= \sum_{k=0}^{K}\left[\left(\sum_{j=0}^{K}\frac{1}{\gamma^2\sigma^2_{\bar{Q},j}+\xi}\right)^{-1}\frac{\left(\mu_{\hat{Q}_\theta}(s_k,a_k) - T(s_k,a_k)\right)^2}{\gamma^2\sigma^2_{\bar{Q},k}+\xi}\right.\\ &\qquad\left.+\frac{\lambda}{K}\left(\frac{\left(\mu_{\hat{Q}_\theta}(s_k,a_k) - T(s_k,a_k)\right)^2}{\sigma^2_{\hat{Q}_\theta}(s_k,a_k)} + \ln\sigma^2_{\hat{Q}_\theta}(s_k,a_k)\right)\right]\end{aligned} \tag{5.3.7}$$

The var-network with parameters θ predicts both $\mu_{\hat{Q}_\theta}(s_k,a_k)$ and $\sigma^2_{\hat{Q}_\theta}(s_k,a_k)$ for each element k of the mini-batch. $\sigma^2_{\bar{Q},k}$ is instead provided by the target var-ensemble which predicts $\bar{Q}(s'_k,a'_k)$ when estimating the target $T(s_k,a_k)$. In $\mathcal{L}_{\text{BIV}}$ of equation (5.2.1), σ^2_k is replaced by $\gamma^2\sigma^2_{\bar{Q},k}$ according to equation (5.3.6). Empirically, the value of λ is usually optimal around 5 or 10. Its impact is studied more in details in appendix B.4. High-variance samples generated from the target estimation will be down-weighted in the BIV loss, while high-variance samples due to the stochasticity of the underlying environment and policy will be down-weighted in the LA loss. In the remainder of the paper, we show how this loss can be applied to different architectures and algorithms, and demonstrate that in many cases it significantly improves sample efficiency.

5.3.2. Q-value uncertainty and actor-critic structures

In section 5.3.1, we discussed how the target's uncertainty can be quantified and how the Bellman update can then be interpreted as a heteroscedastic regression problem. This is applicable in most model-free DRL algorithms, whether they are based on Q-learning or policy optimization. In the special case of actor-critic algorithms, the state-values or Q-values predicted by the critic network are also used to train the policy π_ϕ's parameters by gradient ascent optimization. An estimate of their variance can also be used to improve the learning process.

The objective is to maximize the expected Q-value:

$$\mathbb{E}_{s\sim D, a\sim\pi_\phi(s)}[Q(s,a)] \tag{5.3.8}$$

where D is the state distribution over the probable agent trajectories. The expectation is computed by sampling a mini-batch of size K from a replay buffer containing state-action pairs (s_k,a_k). The Q-value is approximated by the critic as $\hat{Q}(s_k,a_k)$. The actor's parameters ϕ are then trained to maximize the unweighted average:

$$1/K\sum_{k=0}^{K}\hat{Q}(s_k,a_k) \tag{5.3.9}$$

If we instead consider the critic's estimation $\hat{Q}(s_k,a_k)$ as a random variable sampled from $p_{\hat{Q}}(\hat{Q}|s,a)$, with mean $\mu_{\hat{Q}}(s_k,a_k)$ and variance $\sigma^2_{\hat{Q}}(s_k,a_k)$, we can instead infer the expected value in equation (5.3.8) using Bayesian estimation [**117**]:

$$\left(\sum_{k=0}^{K} 1/\sigma^2_{\hat{Q}}\left(s_k,a_k\right)\right)^{-1}\sum_{k=0}^{K}\frac{1}{\sigma^2_{\hat{Q}}(s_k,a_k)}\mu_{\hat{Q}}(s_k,a_k) \tag{5.3.10}$$

The normalized inverse-variance weights are a direct fit with the BIV loss in equation 5.2.1: we therefore also use BIV to train the actor. As the target and the Q-networks have the

same structure, $\sigma^2_Q(s_k,a_k)$ can be estimated using the same var-ensembles used in section 5.3.1.2.

5.3.3. Algorithms

We have adapted IV-RL to DQN and SAC to produce IV-DQN and IV-SAC. Comprehensive pseudo-code, as well as implementation details for each algorithm, can be found in appendix B.2.

5.4. Results

We have tested IV-RL across a variety of different environments, using IV-DQN for discrete tasks and IV-SAC for continuous tasks. To determine the effects of IV-RL, we compare its performance to the original DQN and SAC, but also the ensemble-based baselines upon which IV-DQN and IV-SAC were built: BootstrapDQN and EnsembleSAC. We also include SUNRISE [**92**].

All ensemble-based methods use an ensemble size of $N = 5$. Unless specified otherwise, each result is the average of runs over 25 seeds: 5 for the environment $\times$ 5 for network initialization. The hyperparameters are the result of a thorough fine-tuning process explained in appendix B.3. As it uses ensembles, IV-RL is computationally expensive: computation time is discussed in Appendix B.5.

5.4.1. IV-DQN

We tested IV-DQN on discrete control environments selected to present different characteristics. From OpenAI Gym [**25**], LunarLander is a sparse reward control environment and MountainCar is a sparse reward exploration environment. From BSuite[6] [**126**], Cartpole-Noise is a dense reward control environment. The BootstrapDQN baseline follows [**124**] as explained in section 5.2.3. The reported SUNRISE results are obtained by applying the SUNRISE weights to BootstrapDQN. The learning curves are shown in figure 2, where the results are averaged on a 100 episode window.

IV-DQN significantly outperforms BootstrapDQN and SunriseDQN on control-based environments. However, in the exploration-based Mountain-Car where the challenge is to find the state that produces some reward, there is very little additional information in every other state for IV-DQN to improve over BootstrapDQN. All weighting schemes thus perform comparably.

Table 1 shows the median number of episodes necessary to reach a given score for which the environment is considered as solved. IV-DQN shows a clear improvement over baselines

[6]The results from BSuite are averaged over 20 runs with different environment and network seeds

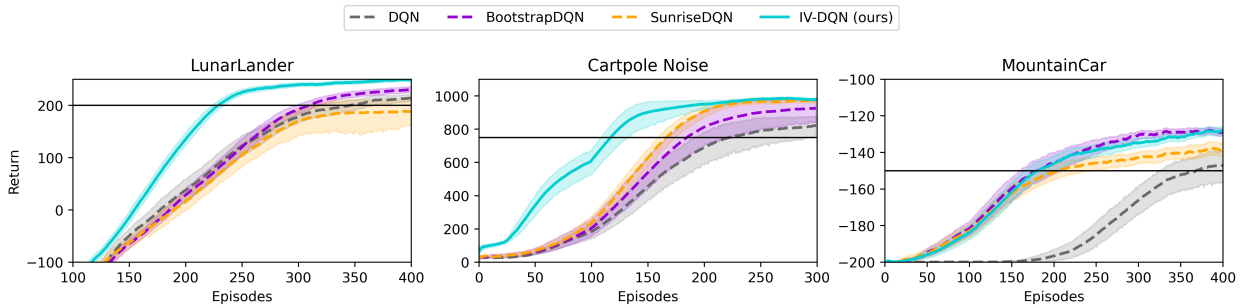


Fig. 2. Using IV-RL shows improved performance over baseline methods in Cartpole Noise and LunarLander, and does not impact MountainCar.

in sample efficiency in both control-based environments but fails to improve the exploration in Mountain Car.

Table 1. 25th - 50th - 75th percentiles of the number of episodes necessary for the return averaged with a 100-episode window to reach the solved score on different environments. IV-DQN shows significant improvements in sample efficiency when the environment is not exploration-based.

	LunarLander (200)	Cartpole-Noise (750)	MountainCar (-150)
DQN	296 - 316 - 349	171 - 193 - max	304 - 333 - 403
BootstrapDQN	287 - 305 - 317	160 - 174 - 196	**134 - 149 - 206**
SunriseDQN	291 - 309 - 368	155 - 165 - 175	152 - 197 - 257
IV-DQN (ours)	**220 - 227 - 239**	**105 - 112 - 117**	142 - 163 - 200

In Figure 3 we perform an ablation study for LunarLander and Cartpole-Noise. In L2 + VarEnsembleDQN, the loss function $\mathcal{L}_{\text{BIV}}$ in equation 5.3.7 is replaced by the $L2$ loss. In BIV + BootstrapDQN, sampled ensembles are used instead of var-ensembles, and trained only with $\mathcal{L}_{\text{BIV}}$. More details about these ablations are found in appendix B.2. In both cases, the use of BIV or var-ensembles improves the performance, and their combination leads to the best sample efficiency. In Cartpole-Noise, var-ensembles carry most of the improvement, maybe because the low amount of states leads to low epistemic uncertainty. We also show that using one single var-network as opposed to var-ensembles is sub-optimal. This is likely due to the unstable nature of the single uncertainty estimate, which affects the learning process. More details are shown in appendix B.6 and B.8. The use of var-ensembles stabilizes the variance estimation, as explained in section 5.2.2.3.

5.4.2. IV-SAC

IV-SAC was applied to different continuous control environments from OpenAI Gym [**25**] as implemented by MBBL [**164**]. EnsembleSAC is a baseline using ensembles with an UCB exploration bonus [**92**]. Table 2 shows the average return after 100k and 200k steps on 25

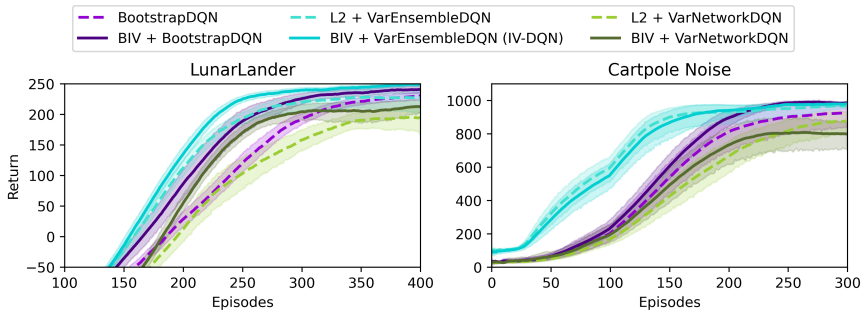


Fig. 3. Ablation study: depending on the environment, the BIV or the var-ensemble component is the most important factor of improvement.

seeds. We note that the addition of an ensemble instead of a single network, along with the uncertainty-based exploration bonus, already allows the performance to increase compared to SAC. Except in Ant, the SUNRISE weights do not seem to lead to consistently better results. In comparison, IV-SAC leads to significant improvements in performance.

Table 2. Performance at 100K and 200K time steps (100 and 200 episodes) for several robotics environments in OpenAI Gym. The results show the mean and standard error over 25 runs.

		Walker	HalfCheetah	Hopper	Ant	SlimHuman.
100k steps	SAC	-392 ± 187	3211 ± 136	322 ± 177	724 ± 29	815 ± 74
	EnsembleSAC	-389 ± 209	3938 ± 112	1597 ± 152	852 ± 27	1043 ± 31
	SunriseSAC	46 ± 213	3879 ± 204	1618 ± 195	834 ± 130	1077 ± 41
	IV-SAC (ours)	**857 ± 231**	**4260 ± 118**	**2237 ± 116**	**948 ± 46**	**1122 ± 34**
200k steps	SAC	371 ± 189	3978 ± 148	1635 ± 162	985 ± 82	1020 ± 52
	EnsembleSAC	1337 ± 278	4757 ± 92	2652 ± 91	981 ± 65	1294 ± 39
	SunriseSAC	1423 ± 295	4785 ± 228	2572 ± 119	**1462 ± 154**	1383 ± 56
	IV-SAC (ours)	**3009 ± 193**	**5451 ± 151**	**2889 ± 50**	1281 ± 101	**2023 ± 114**

This improvement in performance after a fixed amount of training steps can be explained by an improved sample efficiency. This can be seen in figure 4. Even when IV-SAC's return is not significantly better than the baselines at 200k steps, such as in Ant or Hopper, it clearly is learning faster, which is also reflected by the scores at 100k steps.

Similarly to IV-DQN, we can separate the contribution from BIV and loss attenuation, as shown in the two first plots of figure 5. While both BIV and var-ensembles alone have a significant impact in Walker, it's only their combination that brings the most improvement in HalfCheetah. Similarly to the discrete control case, a simple var-network leads to a small improvement over SAC, which is discussed in appendix B.8. We also show in figure 5 that the BIV weights provided with var-ensemble variance estimations than the SUNRISE weights, or even the inverse variance weights of UWAC [**171**]. The normalization of BIV allows it to better cope with changes in the variance predictions, as seem in appendices B.6 and B.7.

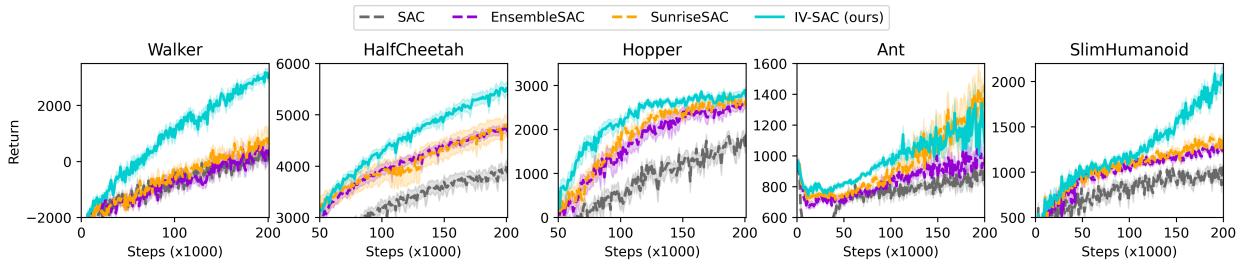


Fig. 4. IV-SAC learns faster and leads to significantly better results than the baselines.

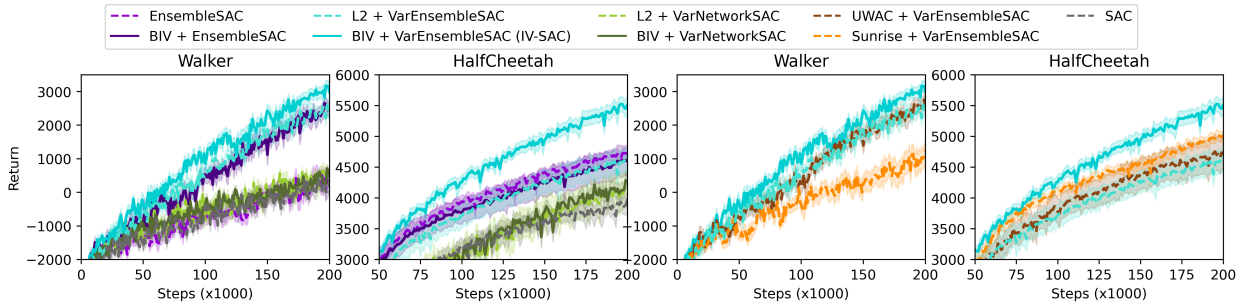


Fig. 5. Ablation Study: (first two figures) Impact of using different uncertainty estimation methods (last two figures) Comparing different weighting schemes with var-ensembles.

5.5. Conclusion

We present Inverse Variance Reinforcement Learning (IV-RL), a framework for model-free deep reinforcement learning which leverages uncertainty estimation to enhance sample efficiency and performance. A thorough analysis of the sources of noise that contribute to errors in the target motivates the use of a combination of Batch Inverse Variance (BIV) weighting and variance ensembles to estimate the variance of the target and down weight the uncertain samples in two complementary ways. Our results show that these two components are beneficial and that their combination significantly improves the state of the art in terms of learning efficiency, in particular for control tasks. We applied IV-RL to DQN and SAC, but it can be adapted to any model-free algorithm. Model-based, active or continuous learning could also benefit from the underlying ideas presented in this paper. We thus believe that IV-RL is a significant step to enable the application of DRL to real-world problems such as robotics, where sample efficiency is the main challenge.

Chapter 6

Prologue to Article 3

This article was submitted at the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2023. Preliminary versions were presented at the NeurIPS 2021 Workshop: Tackling Climate Change with Machine Learning and the NeurIPS 2022 Workshop: Reinforcement Learning for Real Life. It was a joint work with Philippe Maisonneuve, Tianyu Zhang, Hadi Nekoei, Jorge Montalvo Arvizu, Liam Paull and Antoine Lesage Landry. Philippe was a master student supervised by Antoine, a professor at the electrical engineering department of Polytechnique Montreal. Tianyu and Hadi were PhD students at Université de Montréal and Mila, supervised respectively by Yoshua Bengio and Sarath Chandar. Jorge was a PhD student at University of Liège (Belgium) with Louis Wehenkel.

As I wanted to move towards applied research in RL, I looked for an application which would focus on one of the biggest challenges of our time – climate change. Antoine's research focused on algorithms to tackle the challenges of integrating renewable energy sources in power grids. This had been identified as a key field where RL could be used to tackle climate change [**136**]. After discussion, we decided to launch a project of MARL for fast-timescale demand response with air conditioners.

Contribution statement: Antoine and I identified the problem. I formalized it, based on Antoine's domain expertise. I designed and programmed most of the simulator, together with Philippe for signal simulation. Philippe implemented the baseline classical algorithms. Tianyu Zhang supported the project definition and implemented baseline learning algorithms together with Jorge and Hadi. I implemented the final learning algorithms. I designed all experiments, and ran most of them, except the robustness ones which were ran by Philippe. I analyzed the results, but discussed them with the whole group. I prepared the manuscript with the help of Tianyu and Philippe for figures and the appendix. Tianyu, Hadi, Liam and Antoine reviewed it. Antoine and Liam globally supervised the project and gave advice. I prepared the poster and videos presenting the paper at the different workshops.

Chapter 7

Article 3: Multi-Agent Reinforcement Learning for Fast-Timescale Demand Response of Residential Loads

Résumé

Pour intégrer de grandes quantités de ressources énergétiques renouvelables, les réseaux électriques doivent être capables de faire face à des variations rapides et de grande amplitude de la production d'électricité. La régulation de la fréquence par la réponse à la demande offre la possibilité de coordonner des charges temporellement flexibles, comme les climatiseurs, pour contrer ces variations. Les approches existantes pour le contrôle discret avec des contraintes dynamiques ont du mal à fournir des performances satisfaisantes pour la sélection d'actions à échelle de temps rapide avec des centaines d'agents. Nous proposons un agent décentralisé entraîné avec une optimisation de politique proximale multi-agents avec une communication localisée. Nous explorons deux cadres de communication : l'un conçu manuellement, l'autre appris par une communication multi-agents ciblée. Les politiques résultantes sont performantes et robustes pour la régulation de fréquence, et s'adaptent de manière fluide à un nombre arbitraire de maisons pour des temps de traitement constants.

Mots clés: apprentissage par renforcement multi-agents, réseaux électriques, énergies renouvelables, réponse par la demande, régulation de fréquence

Abstract

To integrate high amounts of renewable energy resources, electrical power grids must be able to cope with high amplitude, fast timescale variations in power generation. Frequency regulation through demand response has the potential to coordinate temporally flexible loads, such as air conditioners, to counteract these variations. Existing approaches for discrete control with dynamic constraints struggle to provide satisfactory performance for fast timescale

action selection with hundreds of agents. We propose a decentralized agent trained with multi-agent proximal policy optimization with localized communication. We explore two communication frameworks: hand-engineered, or learned through targeted multi-agent communication. The resulting policies perform well and robustly for frequency regulation, and scale seamlessly to arbitrary numbers of houses for constant processing times.
Key words: multi agent reinforcement learning, power grids, renewable energy, demand response, frequency regulation

7.1. Introduction

To achieve the United Nations' climate change target of limiting global warming at +1.5 °C, global electricity generation must transition from fossil fuels to renewable energy sources such as wind turbines and solar panels. In 2019, according to the International Energy Agency, electricity and heat production accounted for 40% of global emissions [**3**], as 64% of it is generated from burning fossil fuel [**1**]. The electricity sector must thus move from a conventional, fuel-burning paradigm to a renewable, natural phenomenon-based generation, e.g., wind turbines and solar photovoltaics. Renewable energy generation is subject to short-term, high-amplitude variations, referred to as intermittency. As an example, a cloud passing will lead to a sudden drop in the solar-based generation, followed by a sharp increase when the sky becomes clear again. These changes can happen at the scale of a few seconds, and create major challenges for power grid operators: to ensure the stability of the electric grid, a near-perfect balance between the power demand and the generation is critical [**86**]. In other words, power generation and consumption must be equal at all times. Hence, trading a constant, deterministic generation for an intermittent, uncertain one exacerbates the need for power balancing. At the second timescale, this balancing task is referred to as frequency regulation [**19, 157**].

On the power generation side, solutions such as stocking the excess energy in batteries or supporting it with fossil fuel plants require large investments and are not renewable respectively. Alternatively, demand response programs [**146**] can be introduced to mitigate renewable intermittency [**157**]. The demand response approach aims at adjusting the power demand to meet the supply by coordinating loads temporally. These loads must be flexible, i.e., capable of modulating their consumption while fulfilling their own purpose. This does not apply to, for example, computer monitors, which must be fully powered when they are in use. Thermostatic loads, such as heating, air conditioning or water heaters, are instead ideal candidates: they usually do not need to be turned on at all times, as long as the temperature of the air/water is within the user's preference range [**26**]. They are also widely deployed and they represent a significant part of global power consumption [**2, 110**]. The frequency regulation objective differs from peak-shaving for which the objective is load shifting over,

e.g., a day. In our case, the goal is to leverage the loads' flexibility to balance out high-frequency variations in power generation.

In this paper, we focus on the task of fast timescale demand response for frequency regulation using *residential air conditioners*. This presents several physical and algorithmic constraints: (1) air conditioners are **discretely** powered, i.e., ON or OFF, which limits the control flexibility; (2) they are subject to hardware **dynamic constraints** such as lockout: once turned OFF, they must wait some time before being allowed to turn back ON to protect the compressor; (3) as the context is residential, privacy is important and **communications should be limited**; (4) to provide enough power flexibility to the grid, a large aggregation of loads must be considered: the method must be **scalable**; (5) for easier implementation, the control should also be **decentralized** with **localized communications** ; (6) the decisions must be taken at a **few seconds timescale**; and finally (7) the control algorithm should be able to **cope with uncertainty** in the future regulation signal.

These constraints impede the deployment of classical methods. Greedy algorithms are centralized and have difficulty accounting for long-term dynamic constraints [**95**]. Standard model predictive control is also centralized, and even decentralized versions solve a multi-period combinatorial optimization problem that does not scale with the number of agents [**75, 31**]. We propose to tackle this problem by using multi-agent reinforcement learning (MARL) to learn decentralized and scalable policies (4) with discrete and constrained control (1, 2) and limited and localized communications (3, 5). Once learned, these policies can take the best decisions in real time (6) based on expected value over uncertainty (7). As this problem combines the most important current challenges of MARL, i.e., communication, long-term credit assignment, coordination, and scalability [**58**], it is also an interesting benchmark for MARL algorithms. We train our agents with Multi-Agent Proximal Policy Optimization (MA-PPO) [**179**] with Centralized Training, Decentralized Execution (CTDE). [**81**]. Two local communication frameworks are tested – hand-engineered and learned – and both outperform the baselines. Our main contributions are threefold:

- an open source, multi-agent environment[1] simulating the real-world problem of frequency regulation through demand response at the second timescale. The simulator is compatible with the OpenAI Gym [**25**] framework.
- two decentralized, fast-responding agent trained by MA-PPO. The first one has a hand-engineered communication strategy, whereas the second one learns what data to share through Targeted Multi-Agent Communication (TarMAC) [**39**]. Both agents outperform baselines on two-day simulations.
- an in-depth analysis of the dynamics, communications, scalability and robustness of the trained agents.

[1]The code will be made available at `https://github.com/ALLabMTL/MARL_for_fast_timescale_DR`.

In the next section, we describe prior work in the field of demand response and MARL. In Section 7.3, we describe the environment and formulate the problem. The classical and learning-based methods are described in Section 7.4. Finally, Section 7.5 presents the experimental results and analyses of the agents' performance, dynamics, robustness, and scalability.

7.2. Related Works

Frequency regulation through demand response is commonly tackled by model predictive control (MPC) [**170, 93, 121, 75, 108, 110**], where the best action is chosen based on trajectory prediction over a given horizon, sometimes combined with machine learning [**44, 91, 4**]. Apart from [**99**], these works do not consider short-term dynamic constraints such as lockout. MPC approaches rely on mixed-integer programming, which does not scale sustainably with higher numbers of agents, preventing control at fast timescales. Moreover, these works generally require a centralized entity to access residences' data, leading to confidentiality issues. An alternative method of multipliers-based distributed MPC approach was proposed in [**31**]. This approach did not consider the lockout constraint and is not compatible with fast timescale decision-making as it requires multiple centralized communication rounds at each time step in addition to solving several optimization problems and converting continuous setpoints to binary actions.

To tackle these problems, online optimization (OO) approaches [**94, 185**] have been used because of their high computational efficiency and scalability. In particular, [**95**] deploys OO for frequency regulation with binary control settings as is the case for ACs. However, these methods rely on greedy optimization and their lack of foresight leads to limited performance when facing dynamic constraints. Reinforcement learning (RL) methods have been developed to address the longer timescale power balance problems such as peak shaving through demand response [**6**] or coordination of loads and generators [**135, 176**]. The CityLearn environment [**162**] proposes a standard environment for multi-agent RL (MARL) for demand response, upon which are developed methods such as [**131**] to regulate the voltage magnitude in distribution networks using smart inverters and intelligent energy storage management, and [**163**] for load shaping of grid-interactive connected buildings. The AlphaBuilding ResCommunity environment [**166**] then implements detailed thermal models. Both CityLearn and AlphaBuilding ResCommunity, however, consider longer timescale control, which makes them inadequate for high-frequency regulation and removes the ACs' lockout and binary constraints. The PowerGridworld [**21**] environment, a more flexible alternative to CityLearn, allows fast-timescale simulation but does not provide a detailed thermal model of loads, options for lockout or binary control, or classical baseline approaches to compare with. High-frequency regulation has been addressed by MARL, but only on the power generation side [**172**]. We are unaware of any example in the literature deploying MARL

for frequency regulation with demand response, with second-timescale control and flexible binary loads such as ACs which are subject to hardware dynamic constraints like a lockout.

More generally, MARL has been developed for collaboration both in virtual environments such as Dota 2 [**123**], Hide and Seek [**13**] or Hanabi [**51**], and in real-world environments such as traffic light control [**168**], single-house energy management [**5**] or ride-sharing [**132**]. MARL problems pose several additional challenges to the RL settings [**58**], such as the non-stationarity of the environment, the need to learn coordination and communication, or the scaling of the training and deployment. Multi-agent adaptations of known RL algorithms, such as online PPO [**142, 179**], or offline DDPG [**98, 104**] and DQN [**115**], have led to strong performance in many problems. However, some particular problems, such as the ones requiring communication with large numbers of agents, need specialized algorithms [**74**]. TarMAC [**39**], for example, uses an attention mechanism to aggregate messages based on their importance.

7.3. Problem Formulation

7.3.1. Environment

The environment is a simulation of an aggregation of N houses, each equipped with a single air conditioning (AC) unit. The outdoor temperature $T_{o,t}$ is assumed to be the same for every house, i.e., they are co-located in the same geographical region, and is simulated as sinusoidal with a one-day period. Unless otherwise specified, the maximal temperature of 34 °C is reached at 6 pm and the minimal temperature of 28 °C at 6 am. $T_{o,t}$ is thus always above the target indoor temperature T_T of 20 °C so that the household can offer its flexibility to the grid. The environment model is updated every 4 seconds. Thermostatic loads modeled as multi-zone units and equipped with more than a single AC [**8**] is a topic for future work. More details about the environment are given in Appendix C.3. A notation table is provided in Appendix C.1.

7.3.1.1. House thermal model. Each house $i = 1,2,\ldots,N$ is simulated using a second-order model based on Gridlab-D's Residential module user's guide [**71**]. At time t, the indoor air temperature $T^i_{h,t}$ and the mass temperature $T^i_{m,t}$ are updated given the house characteristics θ^i_T (wall conductance U^i_h, thermal mass C^i_m, air thermal mass C^i_h and mass surface conductance H^i_m), the outdoor temperature $T_{o,t}$, and the heat $Q^i_{a,t}$ removed by the AC. By default, the thermal characteristics are the same for each house and model a 100 square meter, 1-floor house with standard isolation. During training and deployment, the initial mass and air temperatures are set by adding a positive random noise over the target temperature. Although it is not used by default, the solar gain $Q_{s,t}$ can also be added to the simulation, as seen in Appendix C.3.1.1.

7.3.1.2. Air conditioners. Once again based on Gridlab-D's guide [**71**], air conditioner i's heat removal capacity $Q^i_{a,t}$ and power consumption $P^i_{a,t}$ are simulated based on the AC characteristics θ^i_a, which include their cooling capacity K^i_a, their coefficient of performance COP^i_a and the latent cooling fraction L^i_a. The model and parameters are also described in Appendix C.3.2. Additionally, a hard dynamic constraint is set to protect the compressor: after being turned OFF, it needs to wait a given amount of time before being allowed to turn ON again [**184**]. This constraint is referred to as the lockout. By default, the lockout duration $l^i_{\max}$ is set to 40 seconds.

7.3.1.3. Regulation signal. The power system operator sends to the aggregator a signal ρ_t, which covers the complete aggregated load consumption: the systems we cannot control such as computers, washing machines, or lights, and the flexible power consumption, in our case, the ACs. Let, $\rho_t = D_{o,t} + s_t$ where $D_{o,t}$ is the power demand for the non-controllable loads and s_t is the objective aggregated AC power consumption, i.e., the flexible load. We define $D_{a,t}$ as the power needed by the ACs to satisfy their thermal objectives, i.e., to keep the temperature around the target. To focus on the high-frequency variations of the power generation, we assume that s_t is well behaved at low frequencies, i.e., its mean in the 5 minutes scale is $D_{a,t}$. A 0-mean, high-frequency variation $\delta_{s,t}$ is added to represent renewable intermittency the aggregator wants to mitigate. We model the regulation signal as $s_t = D_{a,t} + \delta_{s,t}$.

The aggregation flexible power consumption is the sum of all of the ACs' consumption: $P_t = \sum_i^N P^i_{a,t}$. The objective is to coordinate the ACs in the aggregation so that P_t tracks s_t.
Base signal. To compute the average needed power $D_{a,t}$, we created a dataset of the average power needed over a 5-minute period by a bang-bang controller without lockout – which is optimal for temperature – for all combinations of discrete sets of the relevant parameters. At each time step, we interpolate the average power demand of each AC from this dataset and sum them to compute $D_{a,t}$. In practice, the base signal would be estimated or obtained from historical data. The aggregator would then consider its value when committing to track a signal s_t. This ensures that the required power adjustment is enough to maintain the houses at acceptable temperatures while providing flexibility to the grid.
Modelling high-frequency variations. The high-frequency variation $\delta_{s,t}$ is modelled with 1-D Perlin noise [**89**], a smooth, procedurally generated 0-mean noise. The Perlin noise produces $\delta_{p,t} \in [-1, 1]$, and we have $\delta_{s,t} = D_{a,t}\beta_p\delta_{p,t}$ where β_p is an amplitude parameter set to 0.9. Our Perlin noise is defined by 5 octaves and 5 octave steps per period of 400 seconds; it thus is the sum of noises with periods of 80, 40, 20, 10 and 5 seconds. More details are given in Appendix C.3.3.2.

7.3.1.4. Communication between agents. To achieve coordination between agents, they must be able to communicate. For the agent implementation to be decentralized,

flexible, and privacy-preserving, we consider limited and localized communications. This enables, for example, devices communicating with simple radio-frequency emitters, without the need for any further infrastructure. As such, we limit the communication to a number N_c of neighbours. This is in line with the low-deployment investment argument for using demand response for frequency regulation.

7.3.2. Decentralized Partially Observable Markov Decision Process

In this section, we formalize the above environment as a decentralized, partially observable Markov decision process (Dec-POMDP) characterized by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. Let $\mathcal{S}$ be the global state, $\mathcal{A} = \prod_{i=1}^{N} \mathcal{A}^i$ the joint action space, and $\mathcal{O} = \prod_{i=1}^{N} \mathcal{O}^i$ the joint observation space. $\mathcal{O}^i$ partially observes $\mathcal{S}$. $\mathcal{P}$ describes the environment's transition probabilities, $\mathcal{R}$ the reward function for each agent and γ the discount parameter.

7.3.2.1. State, transition probabilities and actions. The state of the environment $X \in \mathcal{S}$ and its transition probabilities $\mathcal{P}$ are unknown to the agent. They are simulated by the environment dynamics described in Section 7.3.1. Each agent i's action $a_t^i \in \mathcal{A}^i$ is a binary decision to control the AC status. If the remaining lockout time l_t^i is above zero, the ON action will be ignored by the AC. In practice, a backup controller within the AC would prevent the ON decision from being implemented.

7.3.2.2. Observations and communications. By default, agent i receives observation $o_t^i = \{T_{h,t}^i, T_{m,t}^i, T_T^i, \omega_t^i, l_t^i, s_t/N, P_t/N\}$ at time step t, where $T_{h,t}^i$, $T_{m,t}^i$ and T_T^i are the indoor air, mass, and target temperatures, ω_t^i is the ON or OFF status of the AC, l_t^i is its remaining lockout time, s_t/N is the per-agent regulation signal and P_t/N is the per-agent total consumption of the aggregation.

Each agent i communicates with its N_c neighbours. The messages' sizes are not hard limited but should be small, and their contents are not constrained. We define the set of all of agent i's N_c neighbours as M^i. By default, we organize the agents in a 1-dimensional structure: $M^i = \{i - \lfloor N_c/2 \rfloor, i - \lfloor N_c/2 \rfloor + 1, \ldots, i, \ldots, i + \lfloor N_c/2 \rfloor - 1, i + \lfloor N_c/2 \rfloor\} \backslash \{i\}$.

7.3.2.3. Reward. For each agent i, the reward r_t^i is computed as the weighted sum of the penalties due to its air temperature being different from the target, which is unique to the agent, and to signal tracking, which is common across all agents. This scenario is therefore cooperative with individual constraints. We normalize the reward with $\alpha_{\text{temp}} = 1$ and $\alpha_{\text{sig}} = 3 \times 10^{-7}$: a 0.5 °C variation is penalized as much as a 912 W per-agent error (each agent consumes 6000 W).

$$r_t^i = -\left(\alpha_{\text{temp}} \left(T_{h,t}^i - T_{T,t}^i \right)^2 + \alpha_{\text{sig}} \left(\frac{P_t - s_t}{N} \right)^2 \right)$$

7.4. Classical and learning-based algorithms

7.4.1. Classical baselines

To the best of our knowledge, there is no classical baseline that performs well under all the constraints enumerated in Section 7.1. However, simple algorithms can optimize selected objectives, and we use them as baselines for the results of the MARL agent.

7.4.1.1. Bang-bang controller. The bang-bang controller (BBC) turns the AC ON when the air temperature $T^i_{h,t}$ is higher than the target T^i_T, and OFF when it is lower. This is a decentralized algorithm, which does not consider demand response but near-optimally controls the temperature. When the lockout duration $l^i_{\max}$ is 0, the BBC optimally controls the temperature, but does not account for the signal. As the base signal $s_{0,t}$ is computed to allow optimal temperature control, BBC's signal tracking error is mainly due to the high-frequency variations of the signal.

7.4.1.2. Greedy myopic. The greedy controller is a centralized algorithm that solves a knapsack problem [**38**] where the size of the collection is the regulation signal, the weight of each AC is its consumption $P^i_{h,t}$, and its value is the temperature difference $T^i_{h,t} - T^i_T$. At each time step, ACs are chosen based on a value priority computed by $(T^i_{h,t} - T^i_T)/P^i_{h,t}$, until the aggregation's consumption P_t is higher than the regulation signal s_t. Because it does not plan for the future, the greedy myopic approach quickly runs out of available ACs as most of them are in the lockout. However, with a 0-lockout duration $l^i_{\max}$, it is optimal to track the signal s_t, and controls the temperature in second priority. We implement the greedy myopic approach as it is better adapted to these settings than the OO approach described in Section 7.2. Indeed, OO only uses past state information and must be implemented in a strictly online fashion. Both frameworks are myopic, and struggle similarly with the lockout constraint.

7.4.1.3. Model predictive control. Model predictive control, or MPC, in its nominal form is a centralized algorithm modeling the environment and identifying the actions which will lead to the highest sum of rewards over a given time horizon of H time steps. As the signal is randomly generated, MPC assumes a constant future signal over horizon H, and optimally solves the trajectory with lockout. However, because it is a large-scale combinatorial optimization problem, it scales poorly with the number of agents N and with a horizon H. In the best case the complexity is polynomial, but it is exponential in the worst case. As a result, we were not able to run the MPC for more than 10 agents for $H = 60$s, and had to increase the time step between each action to 12 seconds. More details are provided in Appendix C.4.1.

7.4.2. Learning-based methods

We deploy two algorithms using deep reinforcement learning, namely MA-DQN and MA-PPO, both using the CT-DE paradigm. While MA-DQN only uses hand-engineered communications, MA-PPO was implemented with two communications paradigms: hand-engineered and learned. Details about the architectures and hyperparameters are provided in Appendix C.4.2.

7.4.2.1. Centralized Training, Decentralized Execution. The CT-DE paradigm [**81**] assumes that information is shared during the training of the agents, while they execute actions only based on their decentralized observations. This reduces the non-stationarity of the environment [**58**] and stabilizes the training. In our case, all agents are homogeneous, which allows the use of parameter sharing [**59**]. As such, all ACs are controlled by identical instances of the same policy trained from the shared experience of all agents.

7.4.2.2. MA-DQN. Multi-agent Deep Q-Network (MA-DQN) is the CT-DE adaptation of DQN [**115**], an off-policy algorithm made for discrete action spaces. A DQN agent mainly consists of a Q-network predicting the Q-value of action-observation pairs $(a_t^i, \tilde{o}_t^i)$ for every possible a_t^i. During training, at time step t, the transition $\Theta_t^i = \{\tilde{o}_t^i, a_t^i, r_t^i, \tilde{o}_{t+1}^i\}$ of every agent is recorded in a common replay buffer. This replay buffer is sampled to train the Q-network to predict $Q(a_t^i, \tilde{o}_t^i)$ supervised with target $T(a_t^i, \tilde{o}_t^i)$ according to Bellman's optimality equation:

$$T(a_t^i, \tilde{o}_t^i) = r_t^i + \gamma \max_a Q(a, \tilde{o}_{t+1}^i).$$

Actions are selected as a_t^i with maximal predicted Q-value given an input $\tilde{o}_t^i$. ϵ-greedy exploration is added during training.

7.4.2.3. MA-PPO. Multi-agent Proximal Policy Optimization (MA-PPO) [**179**] is the CT-DE adaptation of clipped PPO [**142**], an on-policy, policy-gradient algorithm. The agent jointly learns a policy $\pi_\theta(a_t^i|\tilde{o}_t^i)$, also called an actor, and a value function $V_\phi(\tilde{o}_t^i)$, also called a critic. At each epoch, the policy is fixed and the transitions $\Theta_t^i = \{\tilde{o}_t^i, a_t^i, \pi_{\theta_t}(a_t^i|\tilde{o}_t^i), r_t^i\}$ for all agents are recorded together for one or several episodes of length H. For each Θ_t^i, a return $G_t^i = \sum_{\tau=0}^{H-t} \gamma^\tau r_{t+\tau}^i$ is computed based on future experience. Then, the new policy parameters θ_{t+1} are trained over the stored memory to optimize the clipped PPO objective $\mathcal{L}(\tilde{o}_t^i, a_t^i, \theta_{t+1}, \theta_t)$, maximizing the advantage $A^{\pi_{\theta_t}}(\tilde{o}_t^i, a_t^i) = G_t^i - V_\phi(\tilde{o}_t^i)$ under the constraint of proximity around the previous policy. The critic parameters ϕ are then trained so that $V_\phi(\tilde{o}_t^i)$ predicts the return G_t^i. The memory is erased and a new epoch starts.

Exploration is handled by the inherent stochasticity of the policy. In the CT-DE setting, V_ϕ, which is only used during training, is given additional information about the states of other agents.

7.4.2.4. Communications.

Hand-engineered communications. For MA-DQN and the hand-engineered MA-PPO, the messages are designed based on the state of each agent, effectively providing a wider observability of the general state. Agent j's message $m_{j,t}$ contains the current difference between its air and target temperatures $T^j_{h,t} - T^j_T$, its remaining lockout time l^j_t, and its current status ω^j_t. The messages $\{m^i_{j,t}\}_{\forall j \in M_i}$ from agents $j \in M^i$ are concatenated with the observations o^i_t to create the input $\tilde{o}^i_t$ of the neural networks. Message $m^i_{j,t}$ from agent j to agent i is at a fixed place in the $\tilde{o}^i_t$ vector based on its relative position $i - j$. MA-PPO with hand-engineered communication will be referred to as MA-PPO-HE.
Targeted Multi-Agent Communication. To allow agents to learn to communicate, we implement TarMAC [**39**] in MA-PPO. TarMAC is an attention-based targeted communication algorithm where each agent outputs a key, a message and a query. The key is sent along with the message to the other agents, which then multiply it with their query to compute the attention they give to the message. All messages are then aggregated using the attention as a weight. The three modules – key, message, query – are trained. TarMAC allows more flexibility to the agents: it does not restrict the contents of the communication, and it allows agents to communicate with a different number of houses than they were communicating with during training. More details are available in Appendix C.4.2.1. We refer to this version as TarMAC-PPO.
No communication. It is also possible to train agents without communication. In this case, it only observes o^i_t. This agent is referred to as MA-PPO-NC.

7.4.2.5. Agent training. The learning agents were trained on environments with $N_{\text{tr}} = \{10, 20, 50\}$ houses and communicating with $N_{\text{c}_{\text{tr}}} = \{9, 19, 49\}$ other agents. We trained every agent on 16 different seeds: 4 for environment and 4 for network initialization. They were trained on 3286800 time steps, equivalent to 152 days, divided in 200 episodes. Each episode is initialized with each house having a temperature higher than the target, sampled from the absolute value of a 0-mean Gaussian distribution with $\sigma = 5°\text{C}$. We carefully tuned the hyperparameters through a grid search, as shown in Appendix C.4. The contribution of this paper is to demonstrate that learning-based methods can lead to high performance on the problem of high frequency regulation. We therefore do not compile statistics over the trained agents; instead, for each situation, we selected the two best agents over the seeds based on test return, and report the best score from these two over the benchmark environment.

7.5. Results and analysis

7.5.1. Metrics of performance

We deploy the agents on a benchmark environment with N_{de} houses on trajectories of 43200 steps, i.e., two full days. We measure their performance with the per-agent root mean

Table 1. Performance of the different agents, computed over 10 environment seeds.

		$N_{\text{de}} = 10$			$N_{\text{de}} = 50$			$N_{\text{de}} = 250$			$N_{\text{de}} = 1000$		
Per-agent RMSE		Signal (W)	T. (°C)	Max T. (°C)	Signal (W)	T. (°C)	Max T. (°C)	Signal (W)	T. (°C)	Max T. (°C)	Signal (W)	T. (°C)	Max T. (°C)
No l.o	Greedy	194 ± 1	0.04	0.06	70 ± 1	0.03	0.05	63 ± 1	0.03	0.052	63 ± 1	0.03	0.05
	BBC	806 ± 147	0.02	0.03	392 ± 50	0.02	0.04	310 ± 11	0.02	0.03	272 ± 12	0.02	0.03
40s l.o	Greedy	2668 ± 14	0.87	0.93	3166 ± 12	1.09	1.15	3313 ± 12	1.16	1.22	3369 ± 15	1.18	1.24
	BBC	830 ± 207	0.05	0.09	426 ± 63	0.05	0.10	318 ± 7	0.05	0.10	296 ± 4	0.05	0.10
	MPC	344 ± 96	0.07	0.12	-	-	-	-	-	-	-	-	-
	MA-DQN	541 ± 86	0.05	0.09	321 ± 24	0.05	0.10	246 ± 8	0.05	0.11	234 ± 4	0.05	0.12
	MA-PPO-HE	253 ± 1	0.04	0.08	161 ± 8	0.04	0.08	127 ± 2	0.04	0.11	122 ± 3	0.05	0.13
	TarMAC-PPO	**247 ± 3**	**0.04**	**0.07**	**158 ± 2**	**0.04**	**0.09**	**115 ± 1**	**0.05**	**0.13**	**101 ± 2**	**0.05**	**0.14**
	MA-PPO-NC	434 ± 2	0.06	0.08	215 ± 1	0.06	0.14	132 ± 1	0.06	0.16	107 ± 1	0.06	0.17

square error (RMSE) between the regulation signal s_t and aggregated power consumption P_t. We also measure the temperature RMSEs – one (T.) for all agents, one (Max T.) of the maximal temperature error of the aggregation – to evaluate thermal control. Every house's temperature is initialized differently, so we start computing the RMSE when the temperature is controlled, after 5000 steps. For context, a single AC consumes 6000 W when turned ON. Due to the MPC's computing time, its performance is evaluated differently, as explained in Appendix C.4.1. Unless mentioned otherwise, the results are the mean and standard deviation over 10 environmental seeds.

7.5.2. Performance of agents

Table 1 shows the performance of different agents in environments with and without lockout with N_{de} of 10, 50, 250 and 1000 houses.

The per-agent signal RMSE generally goes down when N_{de} increases. This is due to the lower relative discretization error, but also because, with more agents, errors have more chances to cancel each other, as explained in Appendix C.5. As expected, BBC controls the temperature well, but does not track the signal. Without lockout, the greedy myopic shows near-optimal signal tracking, where errors are due to discretization. It also maintains good control of the temperature. With lockout, however, it fails, as it runs out of available agents. The MPC gives good results for 10 agents, but its performance is limited by the lower control frequency of 12 seconds. It could not be run on $N_{\text{de}} = 50$ for computing time reasons. DQN controls the temperature well but is only slightly better than BBC on the signal. Both PPO agents show significantly better performance, and TarMAC-PPO outperforms MA-PPO-HE at high N_{de}. The results without communication will be discussed in Section 7.5.5.

Figure 1 shows the behaviour of each agent over two days for 50 houses. The mean offset captures the error's bias by averaging the differences such that positives and negatives cancel each other, while the mean error is the mean of the absolute differences. The signal and consumption are very high at the beginning due to the initial situation, and then follow the sinusoidal pattern of the outdoor temperature when the indoor temperature is regulated. Without lockout, the BBC shows low temperature and signal offsets, with a significant signal

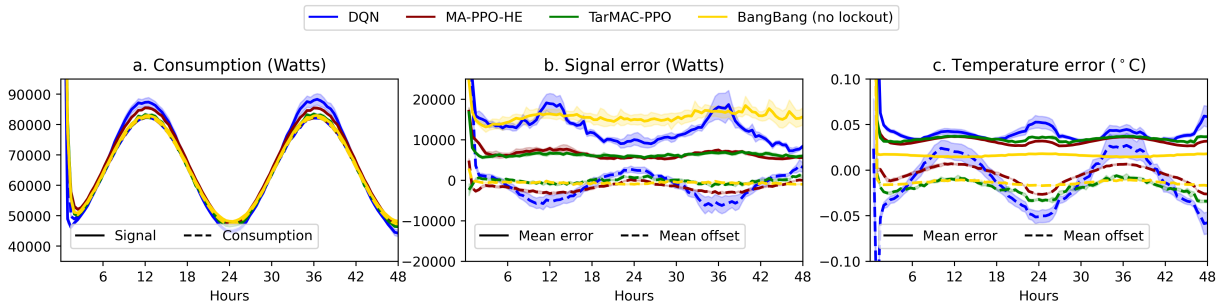


Fig. 1. MA-PPO-HE and TarMAC-PPO outperform DQN and BBC for signal and temperature over 2 days with $N_{\text{de}} = 50$ agents.

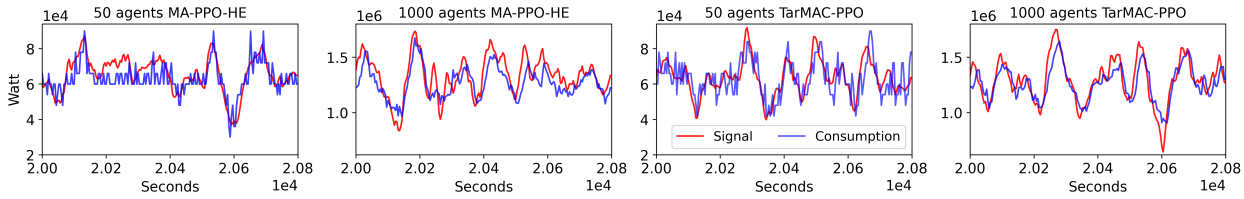


Fig. 2. Both MA-PPO policies scale seamlessly in the number of agents: signal and consumption on 800s for $N_{\text{de}} = 50$ and 1000.

error, as it does not track high-frequency variations of the signal. With the lockout, it underconsumes as explained in Section 7.4.1.1, leading to a positive temperature offset, and the base signal rises to compensate. As the signal variation amplitude is high, this does not strongly affect the error.

The DQN agent has a smaller signal offset and error, especially at night when the amplitude of the signal variations is lower. During the day, the signal error is still significant. Both MA-PPO agents, on the other hand, have a near-0 offset in signal and temperature. Their signal error is also significantly lower than the others, because they are able to track the high-frequency variations.

7.5.3. Scalability with number of agents

As can be seen in Table 1, the PPO agents, and TarMAC-PPO especially, scale gracefully with the number of agents. Figure 2 shows the consumption and signal over 800 seconds for both types of agents deployed over $N_d = 50$ and 1000 over 800 seconds. For $N_d = 50$, the agents do not perfectly match the signal. However, the same agent does better on 1000 houses. Indeed, as the environment is homogeneous, the local strategy scales smoothly by averaging out errors. The best performing agents for TarMAC-PPO were trained on environments with $N_{\text{tr}} = 10$ houses. With MA-PPO-HE, it is often the agents trained on $N_{\text{tr}} = 20$ houses that had the best results. Training with $N_{\text{tr}} = 50$ houses probably makes the credit assignment harder. This is shown in Figure 3.

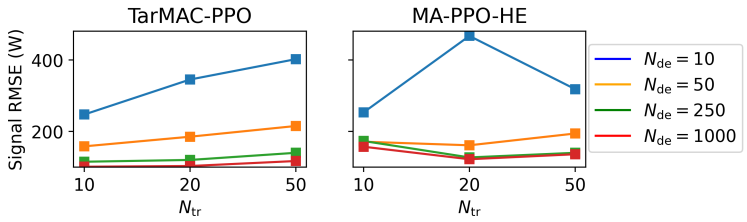
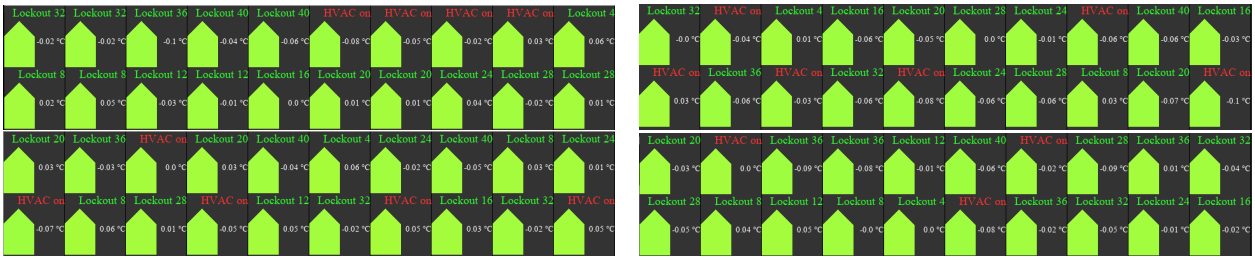


Fig. 3. Training with more agents N_{tr} does not lead to better performance, even when deployed on large N_{de}.

7.5.4. PPO agents' dynamics

As visualized in Figure 4, both MA-PPO-HE and TarMAC-PPO policies keep the ACs in lockout or ON, and never OFF. This is optimal for temperature control: an agent needing to be OFF to warm up after lockout, would not have had the time to warm up during the lockout and was thus ON for too long beforehand. The agents turn ON as soon as they can, but control when they turn OFF based on the context and the messages of other agents.



(a) MA-PPO-HE **(b)** TarMAC-PPO

Fig. 4. State of 20 houses controlled with two different PPO agents. The number on the top right is the remaining lockout time. (Left) Two different agents of MA-PPO-HE with $N_{\text{c}_{\text{de}}} = 19$ show a "20-house" (up) and a "3-house" (down) pattern. (Right) Two different TarMAC-PPO agents show no such pattern.

A fascinating feature of the learned policies is the cyclic behaviour used by MA-PPO-HE agents for coordination. As shown in Figure 4, the ACs turn ON one after the other based on their positions in the aggregation, with a repetition pattern. This happened for each MA-PPO-HE agent we trained, although the pattern period or moving direction was different. These patterns allow the agents to coordinate thanks to the stable message structure, i.e., the fixed relative position of agent j's message to agent i in the $\tilde{o}_t^i$ vector. The TarMAC-PPO agents, on the other hand, do not seem to follow a pattern in their collective behavior. Indeed, as the messages are aggregated, they do not contain information about the structure of the neighbours. The coordination is done through flexible message contents.

7.5.5. Communications

The agents need communications to coordinate and get the best results. In theory, the more agents to communicate with, the better the performance because the observability of the environment is improved. In practice, this is not always the case, as shown in Figure 5. For TarMAC-PPO, communicating with 9 neighbours often leads to the best performance. Higher values of $N_{c_{de}}$ can lead to a reduction of the weight of important messages in the aggregation. For MA-PPO-HE, communicating with 19 agents yields better results than with 49. Indeed, in MA-PPO-HE, the agents must communicate with the same amount of neighbours at deployment than during training: $N_{c_{de}} = N_{c_{tr}}$. During training, communicating with more agents increases the credit assignment difficulty as it increases the input size with non-controllable elements. It is also clear in Figure Figure 5 that agents trained to communicate do not cope well when not communicating. Figure 6 shows the performance of a TarMAC agent trained with $N_{tr} = 10$ houses and $N_{c_{tr}} = 9$ neighbours deployed on an environment with $N_{de} = 50$ houses, when changing the number $N_{c_{de}}$ of neighbours it can communicate with. The performance is bad at low communication but stabilizes around 7 or 8 agents.

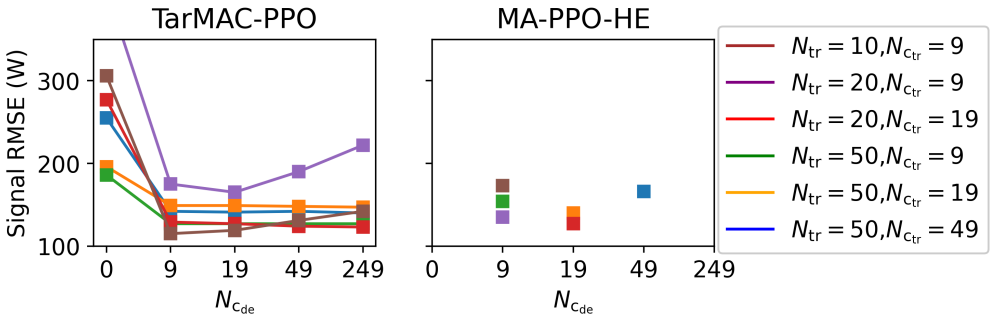


Fig. 5. TarMAC-PPO's performance does not increase after $N_{c_{de}} = 9$ neighbours, while MA-PPO-HE is better with $N_{c_{de}} = 19$ neighbours, for $N_{de} = 250$ houses.

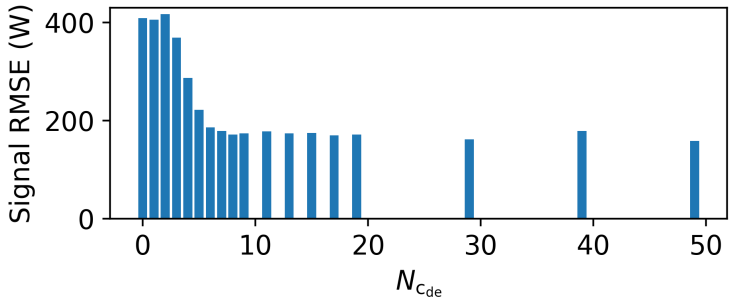


Fig. 6. A TarMAC-PPO agent performs well as long as it communicates with $N_{c_{de}} = 7$ neighbours or more, on $N_{de} = 50$ houses.

It is, however, possible to train an agent without communication to do better than Bang-Bang control, as shown by the performance of MA-PPO-NC in Table 1. Without coordinating with the others, an agent can learn to act well on average to minimize the signal error. When there are only a few agents, as when $N_{\text{de}} = 10$ or 50, this does not perform very well. However, the performance gap decreases when N_{de} increases: a good average policy will do well when applied on many agents. Another way to see this is that, with large N_{de}, each agent's importance becomes negligible in the final result. As such, the group can be seen as a single average agent, and the problem cam be posed as a mean field game [**177, 151**].

Interestingly, MA-PPO-HE at high N_{de} does better with communication defects. This may be because the MA-PPO-HE coordination leads to locally biased policies, which do not benefit from the averaging effect reducing the relative error when N_{de} increases.

7.5.6. Robustness

All the results presented were produced under certain assumptions, such as homogeneous houses and ACs, consistent outdoor temperature and signal profiles, and faultless communication. If such agents were to be deployed in the real world, they would be confronted with situations where these conditions are not satisfied. In this section, we evaluate the robustness of our trained agents to different disturbances in the deployment conditions.

7.5.6.1. Faulty communications. As previously demonstrated, communications are key for good performance of the agents. In this robustness test, we simulate defective communications. At every time step, each message m_j^i is defective with a probability p_d. In the case of TarMAC-PPO, this leads to the message not being received. For MA-PPO-HE, every element of the message is set to 0. We tested the best agents for $N_{\text{de}} = 10$, 50, 250 and 1000 houses with $p_d = 0.1$ and 0.5, as seen in Table 2. MA-PPO-HE agents' coordination is based on their stable communication structure. As a result, it copes badly with defective communications. Interestingly, when N_{de} is higher, the impact decreases, even leading to better performance at $N_{\text{de}} = 1000$ houses. This may be due to the fact that the resulting policies cannot coordinate locally and are less locally biased. The TarMAC-PPO handles perfectly temporary defects in communication as its messages are aggregated. This is the case even with $p_d = 0.5$ and when the agent communicates with $N_{\text{c}_{\text{tr}}} = 9$ neighbours only.

7.5.6.2. Heterogeneous houses and ACs. In reality, different houses have different thermal characteristics. The ACs also do not always have the same rated cooling power or lockout duration. We deployed the best trained MA-PPO-HE and TarMAC-PPO agents for 50-house environments that do not comply with these assumptions on different scales, to evaluate their robustness to separate disturbances. We also trained new agents on environments with these conditions, to see if the agents could learn to cope with heterogeneity.

Table 2. Performance under faulty communication (5 seeds)

		$N_{\text{de}} = 10$			$N_{\text{de}} = 50$			$N_{\text{de}} = 250$			$N_{\text{de}} = 1000$		
Per-agent RMSE		Signal (W)	T. (°C)	Max T. (°C)	Signal (W)	T. (°C)	Max T. (°C)	Signal (W)	T. (°C)	Max T. (°C)	Signal (W)	T. (°C)	Max T. (°C)
MA-PPO-HE	$p_d = 0$	253 ± 1	0.04	0.08	161 ± 8	0.04	0.08	127 ± 2	0.04	0.11	122 ± 3	0.05	0.13
	$p_d = 0.1$	504 ± 2	0.07	0.14	207 ± 1	0.04	0.11	138 ± 2	0.05	0.13	118 ± 1	0.05	0.14
	$p_d = 0.5$	597 ± 2	0.10	0.19	274 ± 1	0.06	0.15	148 ± 1	0.06	0.151	115 ± 2	0.06	0.17
TarMAC-PPO	$p_d = 0$	247 ± 3	0.04	0.07	158 ± 2	0.04	0.09	115 ± 1	0.05	0.13	101 ± 2	0.05	0.14
	$p_d = 0.1$	246 ± 2	0.04	0.07	158 ± 2	0.04	0.09	115 ± 2	0.05	0.12	101 ± 1	0.05	0.14
	$p_d = 0.5$	248 ± 2	0.04	0.07	159 ± 3	0.04	0.09	115 ± 2	0.05	0.13	101 ± 1	0.05	0.14

Table 3. Performance under house and AC heterogeneity

	MA-PPO-HE		MA-PPO-HE-T	
Per-agent RMSE	Signal (W)	Max T. (°C)	Signal (W)	Max T. (°C)
Without heterogeneity	161 ± 8	0.08	-	-
House thermal parameters	285 ± 8	0.17	222 ± 7	0.11
AC cooling power	292 ± 3	0.15	181 ± 3	0.14
Lockout duration	324 ± 9	0.15	246 ± 4	0.09
	TarMAC-PPO		TarMAC-PPO-T	
Without heterogeneity	158 ± 2	0.09	-	-
House thermal parameters	184 ± 2	0.12	174 ± 2	0.11
AC cooling power	187 ± 2	0.16	185 ± 9	0.16
Lockout duration	192 ± 3	0.09	251 ± 4	0.08

The characteristics in question were observed by both agents as part of o_t^i, and part of the messages m_j^i in MA-PPO-HE. These agents are referred to with the -T suffix. The thermal characteristics heterogeneity was simulated by adding a Gaussian noise to element of θ_h^i for each house. This noise has a standard deviation of 50% of the original value (the final values cannot be negative). Heterogeneity in the ACs cooling capacities K_a^i was simulated by uniformly selecting for each house a value between 10, 12.5, 15, 17.5 and 20 kW. Finally, heterogeneity in the lockout duration $l_{\max}$ was tested by sampling uniformly between 32, 36, 40, 44 and 48 seconds.

The results are shown in Table 3. TarMAC-PPO is much more robust to heterogeneity in agents than MA-PPO-HE. This is because in MA-PPO-HE the coordination scheme is based on the stable dynamics of the agent's neighbours, especially with the lockout duration. TarMAC-PPO is instead more flexible with respect to different dynamics. For both agents, it is possible to reduce the effect of heterogeneity by training the agents on such environments and allowing them to observe the characteristics. This is different for heterogenity on the lockout duration, where TarMAC-PPO did not seem able to train satisfactorily on such conditions. An interesting observation is that the best TarMAC-PPO results were obtained when communicating with $N_{\text{c}_{\text{tr}}} = 49$ neighbours. When agents are heterogeneous, more neighbours are needed to have a representative input.

Table 4. Robustness on environment changes (5 seeds)

	MA-PPO-HE		TarMAC-PPO	
Per-agent RMSE	Signal (W)	Max T. (°C)	Signal (W)	Max T. (°C)
Same as training	161 ± 8	0.08	158 ± 2	0.09
Solar gain	190 ± 6	0.09	174 ± 2	0.10
Outdoor T. + 4°C	203 ± 4	0.11	198 ± 2	0.11
Outdoor T. - 4°C	170 ± 1	0.09	184 ± 2	0.12
Signal average + 30%	401 ± 2	0.11	302 ± 2	0.14
Signal average - 30%	337 ± 4	0.10	317 ± 1	0.11
Signal noise amplitude + 30%	188 ± 5	0.08	179 ± 3	0.09
Signal noise frequency + 100%	200 ± 4	0.08	198 ± 5	0.09

7.5.6.3. Other environments. We also tested our agents on environments differing from the training environment, with different outdoor temperature T_o, solar gain Q_s, too low or high average signal D_a, and higher or faster signal variations δ_s. As can be seen in Table 4, both agents are quite robust to such changes, with TarMAC-PPO usually leading to better results. The only case where the performance is strongly affected is when the signal is misbehaved, i.e., it is too low or too high to allow correct control of the temperature. In this case, there is a tradeoff between the signal and the temperature objectives. MA-PPO-HE seems to give higher priority to temperature control, leading to higher signal RMSE but a lower temperature error.

7.5.7. Processing time

In Table 5, we report the processing time for action selection of the baseline and trained agents. The results are shown for 25 times steps (100 seconds of simulation), except for the MPC which simulated 100 seconds with 10-time steps. They were computed on the 12-core, 2.2 GHz Intel i7-8750H CPU of a laptop computer.

Table 5. Computation time (s) for action selection, for 100 seconds of simulation. We report the time per-agents for a decentralized system and for the whole system otherwise.

Agent	Decentralized	$N_{\text{de}} = 10$	$N_{\text{de}} = 1000$
TarMAC-PPO	Yes	0.002	0.001
MA-PPO-HE	Yes	0.006	0.006
DQN	Yes	0.003	0.002
BBC	Yes	0.00001	0.00001
Greedy myopic	No	0.1	3.7
MPC - $H = 40$s	No	92.6680	-

As the decentralized, learned agents only need a single forward pass in a relatively small neural network, the time for action selection is sufficiently low for control when using 4-second time steps. Centralized approaches such as greedy myopic scale badly with many agents. MPC, already simplified with time steps of 12 seconds instead of 4, and a short horizon of 40 seconds, takes an unacceptable amount of time for more than 10 agents.

7.6. Conclusion

In this paper, we tackle the problem of high-frequency regulation with demand response by controlling discrete and dynamically constrained residential loads equipped with air conditioners with a decentralized, real-time agent trained by MA-PPO. We test two frameworks for local communication – fixed hand-engineered messages and learned targeted communication. The policies trained with few agents perform significantly better than baselines, scale seamlessly to large numbers of houses, and are robust to most disturbances. Our results show that MARL can be used successfully to solve some of the complex multi-agent problems induced by the integration of renewable energy in electrical power grids. Future works towards the application of such algorithms on real power systems could include sim2real transfer, integration of more complex flexible loads, as well as power grid safety issues.

Conclusion

The path towards performing and reliable real world autonomous agents will pass through deep reinforcement learning. The evident potential of this machine learning approach has not yet been realized. The recent successes of deep neural networks, combined with the always more affordable computation power and the constantly increasing gathering of data, have boosted its capabilities in the last decade. Tremendous research efforts are currently undertaken to tackle one of the current bottlenecks to its application on real world agents: training agents is expensive and risky.

In this thesis, I presented three pieces of work in this direction. In the first two articles, I proposed to improve sample efficiency of DRL algorithms through the probabilistic representation of its variables. The BIV loss developed in **Batch inverse-variance weighting for deep heteroscedastic regression** is a key ingredient in the IV-RL approach presented in **Sample efficient deep reinforcement learning via uncertainty estimation**. In **Multi-agent reinforcement learning for fast-timescale demand response with residential loads**, I show that DRL can be successfully applied to real world agent cases in power grid problems where engineered algorithms do not perform well.

As with most research projects, these three papers have limitations and open many questions for future work. They could also have broader applications than the objectives for which they were developed in this thesis.

In the first paper, BIV is inspired from the Gauss Markov theorem, which is optimal in linear regression. However, it is not proven optimal for stochastic gradient descent with neural networks. In addition, we introduce a new hyperparameter, ϵ, which needs to be optimized for better performance.

BIV however proved that accounting for the variance of the labels as an additional information allows to mitigate the effect of noisy labels on the learning process. It can be applied to any heteroscedastic regression problem where the label noise variances are known. This applies, for example, to measured labels, where uncertainty comes from sensor reading or population studies. It also describes the situation where labels are created from imperfect

models, such as hand-crafted simulators or in complex machine learning pipelines where labels are created by other models – DRL is an instance of the latter category.

Further research could improve BIV, for example by automating the selection of ϵ. Better performing solutions to the problem of heteroscedastic regression with known variance noise could come out of stronger theoretical analysis of the optimization problem.

In the second paper, IV-RL suffers from the complexity and the lack of reliability of current uncertainty estimation methods for deep neural networks predictions. It needed heavy engineering to ensure stability and robustness to over- or under-estimations. Its loss function, $\mathcal{L}_{\text{IVRL}}$, needs at the same time to train the variance networks through loss attenuation and to take into account the variance of the target using BIV. As a result, it does not combine both as an unique target uncertainty, as would have been optimal according to our analysis.

Despite this, IV-RL's results are yet another proof that the use of probabilistic representations in DRL is a key direction towards better performing and sample efficient algorithms. It adds to a body of many recent works who took different angles at the same question, and are steps towards a holistic Bayesian pipeline for DRL.

More reliable and information-theory grounded uncertainty estimation methods for regression with deep neural networks are a key to such probabilistic formulation for DRL. It would not only improve the performance, but also allow safer exploration and risk-aware deployment of DRL agents. DRL algorithms come in many flavors, from Q-learning to policy gradient and even model-based RL. An interesting research direction is to examine how probabilistic formulations can be applied to each of these families.

In the third paper, we did not deploy our MARL agents on real air conditioners. While we are confident that the agents are robust enough to imperfections in the simulator, this was not tested. We also made important efforts to ensure the infrastructure assumptions behind our project were realistic. They would need to be demonstrated on real world deployed agents.

Our approach however demonstrates the potential of MARL for complex problems in power-grid systems coping with the introduction of renewable energy sources. While these environments can be very well understood, the complexity and stochasticity of their dynamics may impede hand-crafted solutions to perform in a satisfying manner. In this case, RL can be a powerful tool, and could be a key factor in the energetic transition.

To increase the demand response potential of the approach, further work would consider the addition of additional flexible loads, such as water heaters or charging electric vehicles. In addition, to ensure a scalable and successful integration of such agents in power grids, safety issues must be taken into account, as well as their interactions with other, longer timescale programs for demand response. This may require solving more complex MARL

problems with heterogeneous or asymmetric agents.

Final remarks This thesis was built on state-of-the-art approaches in many different but currently blooming fields: supervised learning and regression with deep neural networks, uncertainty prediction and Bayesian deep learning, deep reinforcement learning and its Bayesian extensions, cooperative multi agents reinforcement learning, attention mechanisms...

I consider that the opportunity I was given, to explore such impressive advances, and to combine some of them to create something new, was an extraordinary privilege. Early in my academic career, I studied physics. I often wondered if the researchers of the beginning of the XXth century, when modern physics was being discovered, realized that they were at the forefront of a scientific revolution. Their research deeply transformed our society.

The research presented in this thesis allowed me to witness the same kind of historical moments. Since I started in 2018, every one of the fields I encountered has advanced by gigantic strides. The methods which were discussed and designed two years ago are now proven with astonishing large scale demonstrations. They are being quickly consolidated and readied for massive deployment. While there are still important challenges, I have no doubt the powerful algorithms developed in research laboratories will soon be found in many aspects of our daily lives. This will profoundly transform our society. I hope that the work in this thesis contributes to this transformation being for the better.

References

[1] International Energy Agency : Energy statistics data browser – data tools. Available on: `https://www.iea.org/data-and-statistics/data-tools/energy-statistics-data-browser` (Accessed on Sept 15, 2022).

[2] International Energy Agency : *The Future of Cooling*. 2018.

[3] International Energy Agency : *Greenhouse Gas Emissions from Energy: Overview*. 2021.

[4] Roya Ahmadiahangar, Tobias Häring, Argo Rosin, Tarmo Korõtko et João Martins : Residential load forecasting for flexibility prediction using machine learning-based regression model. *In 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, pages 1–4, 2019.

[5] Mehdi Ahrarinouri, Mohammad Rastegar et Ali Reza Seifi : Multiagent reinforcement learning for energy management in residential buildings. *IEEE Transactions on Industrial Informatics*, 17(1): 659–666, Jan 2021.

[6] Sally Aladdin, Samah El-Tantawy, Mostafa M. Fouda et Adly S. Tag Eldien : Marla-sg: Multi-agent reinforcement learning algorithm for efficient demand response in smart grid. *IEEE Access*, 8:210626–210639, 2020.

[7] Hannah Alsdurf, Yoshua Bengio, Tristan Deleu, Prateek Gupta, Daphne Ippolito, Richard Janda, Max Jarvie, Tyler Kolody, Sekoul Krastev, Tegan Maharaj et et al. : Covi white paper. *arXiv:2005.08502 [cs]*, May 2020.

[8] Uzma Amin, MJ Hossain et E Fernandez : Optimal price based control of hvac systems in multizone office buildings for demand response. *Journal of Cleaner Production*, 270:122059, 2020.

[9] Siddharth Aravindan et Wee Sun Lee : State-aware variational Thompson sampling for deep Q-networks. *In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 124–132. International Foundation for Autonomous Agents and Multiagent Systems, 2021.

[10] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio et et al. : A closer look at memorization in deep networks. *arXiv:1706.05394 [cs, stat]*, Jul 2017.

[11] Jean-Yves Audibert, Rémi Munos et Csaba Szepesvári : Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, Apr 2009.

[12] Peter Auer, Nicolò Cesa-Bianchi et Paul Fischer : Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.

[13] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew et Igor Mordatch : Emergent tool use from multi-agent autocurricula. Feb 2020. arXiv:1909.07528 [cs, stat].

[14] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare et Michael Bowling : The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, Mar 2020. arXiv:1902.00506 [cs, stat].

[15] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess et Timothy Lillicrap : Distributed distributional deterministic policy gradients. (arXiv:1804.08617), Apr 2018. arXiv:1804.08617 [cs, stat].

[16] Marc G. Bellemare, Will Dabney et Rémi Munos : A distributional perspective on reinforcement learning. *In Proceedings of the 34th International Conference on Machine Learning*, volume 70 de *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 06–11 Aug 2017.

[17] Richard Bellman : A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957.

[18] Daniel S. Bernstein, Robert Givan, Neil Immerman et Shlomo Zilberstein : The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, Nov 2002.

[19] Hassan Bevrani, Arindam Ghosh et Gerard Ledwich : Renewable energy sources and frequency regulation: survey and new perspectives. *IET Renewable Power Generation*, 4(5):438–457, 2010.

[20] Dhaivat Bhatt, Kaustubh Mani, Dishank Bansal, Krishna Murthy, Hanju Lee et Liam Paull : f-Cal: Calibrated aleatoric uncertainty estimation from neural networks for robot perception. *arXiv:2109.13913*, Sep 2021.

[21] David Biagioni, Xiangyu Zhang, Dylan Wald, Deepthi Vaidhynathan, Rohit Chintala, Jennifer King et Ahmed S. Zamzam : Powergridworld: A framework for multi-agent reinforcement learning in power systems, 2021.

[22] Christopher M. Bishop : *Pattern recognition and machine learning*. Information science and statistics. Springer, 2006.

[23] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu et Daan Wierstra : Weight uncertainty in neural networks. (arXiv:1505.05424), May 2015. arXiv:1505.05424 [cs, stat].

[24] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron et Olivier Simonin : Cooperative multi-agent policy gradient. *In* Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley et Georgiana Ifrim, éditeurs : *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, page 459–476, Cham, 2019. Springer International Publishing.

[25] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba : Openai gym, Jun 2016.

[26] Duncan S Callaway : Tapping the energy storage potential in electric loads to deliver load following and regulation, with application to wind energy. *Energy Conversion and Management*, 50(5):1389–1400, 2009.

[27] Jeffrey S. Campbell, Sidney N. Givigi et Howard M. Schwartz : Handling stochastic reward delays in machine reinforcement learning. *In 2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 314–319, 2015.

[28] Alex J. Cannon : Quantile regression neural networks: Implementation in r and application to precipitation downscaling. *Computers & Geosciences*, 37(9):1277–1284, Sep 2011.

[29] Kaidi Cao, Yining Chen, Junwei Lu, Nikos Arechiga, Adrien Gaidon et Tengyu Ma : Heteroskedastic and imbalanced deep learning with adaptive regularization. *arXiv:2006.15766 [cs, stat]*, Jun 2020.

[30] René Carmona, Mathieu Laurière et Zongjun Tan : Model-free mean-field reinforcement learning: Mean-field mdp and mean-field q-learning. (arXiv:1910.12802), Oct 2019. arXiv:1910.12802 [cs, math].

[31] Bingqing Chen, Jonathan Francis, Marco Pritoni, Soummya Kar et Mario Bergés : Cohort: Coordination of heterogeneous thermostatically controlled loads for demand flexibility. *In Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, page 31–40, Nov 2020. arXiv:2010.03659 [cs, eess].

[32] Richard Y. Chen, Szymon Sidor, Pieter Abbeel et John Schulman : UCB exploration via Q-ensembles. *arXiv:1706.01502*, Nov 2017.

[33] Tianshu Chu, Sandeep Chinchali et Sachin Katti : Multi-agent reinforcement learning for networked system control. (arXiv:2004.01339), Apr 2020. arXiv:2004.01339 [cs, stat].

[34] Xiangxiang Chu et Hangjun Ye : Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. (arXiv:1710.00336), Oct 2017. arXiv:1710.00336 [cs].

[35] CIBSE : *Guide A: Environmental Design*. Chartered Institution of Building Services Engineers, 8th édition, 2015.

[36] William R. Clements, Bastien Van Delft, Benoît-Marie Robaglia, Reda Bahi Slaoui et Sébastien Toth : Estimating risk and uncertainty in deep reinforcement learning. *arXiv:1905.09638*, Sep 2020.

[37] Will Dabney, Mark Rowland, Marc G. Bellemare et Rémi Munos : Distributional reinforcement learning with quantile regression. 2017.

[38] George B. Dantzig : Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, Apr 1957.

[39] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat et Joelle Pineau : Tarmac: Targeted multi-agent communication. *In Proceedings of the 36th International Conference on Machine Learning*, page 1538–1546. PMLR, May 2019.

[40] Richard Dearden, Nir Friedman et Stuart J. Russel : Bayesian Q-learning. 1998.

[41] Steven Diamond et Stephen Boyd : CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016. To appear.

[42] Dheeru Dua et Casey Graff : UCI machine learning repository, 2017.

[43] Gabriel Dulac-Arnold, Daniel Mankowitz et Todd Hester : Challenges of real-world reinforcement learning. *arXiv:1904.12901*, Apr 2019.

[44] Ivana Dusparic, Colin Harris, Andrei Marinescu, Vinny Cahill et Siobhán Clarke : Multi-agent residential demand response based on load forecasting. *In 2013 1st IEEE Conference on Technologies for Sustainability (SusTech)*, pages 90–96, 2013.

[45] Hadi Fanaee-T et Joao Gama : Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15, 2013.

[46] Di Feng, Lars Rosenbaum, Claudius Glaeser, Fabian Timm et Klaus Dietmayer : Can we trust you? on calibration of a probabilistic object detector for autonomous driving. (arXiv:1909.12358), Sep 2019. arXiv:1909.12358 [cs].

[47] G. R. Fisher : Maximum likelihood estimators with heteroscedastic errors. *Review of the International Statistical Institute*, 25(1/3):52, 1957.

[48] Sebastian Flennerhag, Jane X. Wang, Pablo Sprechmann, Francesco Visin, Alexandre Galashov, Steven Kapturowski, Diana L. Borsa, Nicolas Heess, Andre Barreto et Razvan Pascanu : Temporal difference uncertainties as a signal for exploration. *arXiv:2010.02255*, Oct 2020.

[49] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli et Shimon Whiteson : Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr 2018.

[50] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin et et al. : Noisy networks for exploration. *arXiv:1706.10295*, Jul 2019.

[51] Andrew Fuchs, Michael Walton, Theresa Chadwick et Doug Lange : Theory of mind for deep reinforcement learning in hanabi. Jan 2021. arXiv:2101.09328 [cs].

[52] Yarin Gal et Zoubin Ghahramani : Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *In Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1050–1059. PMLR, 20–22 Jun 2016.

[53] Xavier Glorot et Yoshua Bengio : Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, page 249–256. JMLR Workshop and Conference Proceedings, Mar 2010.

[54] Jacob Goldberger et Ehud Ben-Reuven : Training deep neural-networks using a noise adaptation layer. *In 5th International Conference on Learning Representations*, 2017.

[55] C. V. Goldman et S. Zilberstein : Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, Nov 2004.

[56] Ian Goodfellow, Yoshua Bengio, Aaron Courville et Yoshua Bengio : *Deep learning*, volume 1. MIT Press, 2016.

[57] Alex Graves : Practical variational inference for neural networks. *In* J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira et K. Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[58] Sven Gronauer et Klaus Diepold : Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, Apr 2021.

[59] Jayesh K. Gupta, Maxim Egorov et Mykel Kochenderfer : *Cooperative Multi-agent Control Using Deep Reinforcement Learning*, volume 10642 de *Lecture Notes in Computer Science*, page 66–83. Springer International Publishing, Cham, 2017.

[60] Gurobi Optimization, LLC : Gurobi Optimizer Reference Manual, 2022.

[61] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel et Sergey Levine : Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *In International Conference on Machine Learning*, pages 1861–1870. PMLR, Aug 2018.

[62] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang et Masashi Sugiyama : Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels. *arXiv e-prints*, page arXiv:1804.06872, avril 2018.

[63] Ali Harakeh, Jordan Hu, Naiqing Guan, Steven L. Waslander et Liam Paull : Estimating regression predictive distributions with sample networks. (arXiv:2211.13724), Nov 2022. arXiv:2211.13724 [cs].

[64] Hado van Hasselt, Arthur Guez et David Silver : Deep reinforcement learning with double q-learning. *In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. *arXiv:1512.03385 [cs]*, Dec 2015.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *In 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[67] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup et David Meger : Deep reinforcement learning that matters. *In Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[68] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar et David Silver : Rainbow: Combining improvements in deep reinforcement learning. (arXiv:1710.02298), Oct 2017. arXiv:1710.02298 [cs].

[69] Ronald A. Howard : Dynamic programming and markov processes. 1960.

[70] Eyke Hüllermeier et Willem Waegeman : Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110(3):457–506, Mar 2021.

[71] Betelle Memorial Institute : Gridlab-d wiki. `http://gridlab-d.shoutwiki.com/wiki/Main_Page` (Accessed on: Sept 15, 2022).

[72] Moksh Jain, Salem Lahlou, Hadi Nekoei, Victor Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov et Yoshua Bengio : DEUP: Direct epistemic uncertainty prediction. *arXiv:2102.08501*, Feb 2021.

[73] Simon Jenni et Paolo Favaro : Deep bilevel learning. *arXiv:1809.01465 [cs, stat]*, Sep 2018.

[74] Jiechuan Jiang et Zongqing Lu : Learning attentional communication for multi-agent cooperation. *In Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[75] Jin, Mohammed Olama, Teja Kuruganti, James Nutaro, Christopher Winstead, Yaosuo Xue et Alexander Melin : Model predictive control of building on/off hvac systems to compensate fluctuations in solar power generation. *In 2018 9th IEEE International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, pages 1–5, 2018.

[76] Kenji Kawaguchi, Leslie Pack Kaelbling et Yoshua Bengio : Generalization in deep learning. *arXiv:1710.05468 [cs, stat]*, Jul 2020.

[77] Alex Kendall et Yarin Gal : What uncertainties do we need in bayesian deep learning for computer vision? *Advances in Neural Information Processing Systems*, 30:5574–5584, 2017.

[78] Diederik P. Kingma et Jimmy Ba : Adam: A method for stochastic optimization. *In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[79] L. Kish : *Survey Sampling*. Wiley, 1965.

[80] Parth Kothari, Christian Perone, Luca Bergamini, Alexandre Alahi et Peter Ondruska : Drivergym: Democratising reinforcement learning for autonomous driving. *arXiv:2111.06889*, Nov 2021.

[81] Landon Kraemer et Bikramjit Banerjee : Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, May 2016.

[82] Jan Kremer, Fei Sha et Christian Igel : Robust active label correction. volume 84 de *Proceedings of Machine Learning Research*, pages 308–316, Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018. PMLR.

[83] Volodymyr Kuleshov, Nathan Fenner et Stefano Ermon : Accurate uncertainties for deep learning using calibrated regression. *In International Conference on Machine Learning*, pages 2796–2804. PMLR, 2018.

[84] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker et Sergey Levine : Stabilizing off-policy Q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32:11784–11794, 2019.

[85] Aviral Kumar, Abhishek Gupta et Sergey Levine : DisCor: Corrective feedback in reinforcement learning via distribution correction. *Advances in Neural Information Processing Systems*, 33, 2020.

[86] Prabha Kundur : Power system stability. *Power system stability and control*, pages 7–1, 2007.

[87] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt et Thomas Dandres : Quantifying the carbon emissions of machine learning. *arXiv:1910.09700*, 2019.

[88] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt et Thomas Dandres : Quantifying the carbon emissions of machine learning. (arXiv:1910.09700), Nov 2019. arXiv:1910.09700 [cs].

[89] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin et M. Zwicker : A survey of procedural noise functions. *Computer Graphics Forum*, 29(8):2579–2600, Dec 2010.

[90] Balaji Lakshminarayanan, Alexander Pritzel et Charles Blundell : Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in Neural Information Processing Systems*, 30, Nov 2017. arXiv: 1612.01474.

[91] Fiorella Lauro, Fabio Moretti, Alfonso Capozzoli et Stefano Panzieri : Model predictive control for building active demand response systems. *Energy Procedia*, 83:494–503, 2015. Sustainability in Energy and Buildings: Proceedings of the 7th International Conference SEB-15.

[92] Kimin Lee, Michael Laskin, Aravind Srinivas et Pieter Abbeel : SUNRISE: A simple unified framework for ensemble learning in deep reinforcement learning. *In International Conference on Machine Learning*, pages 6131–6141. PMLR, 2021.

[93] Young M Lee, Raya Horesh et Leo Liberti : Optimal hvac control as demand response with on-site energy storage and generation system. *Energy Procedia*, 78:2106–2111, 2015.

[94] Antoine Lesage-Landry et Joshua A Taylor : Setpoint tracking with partially observed loads. *IEEE Transactions on Power Systems*, 33(5):5615–5627, 2018.

[95] Antoine Lesage-Landry, Joshua A Taylor et Duncan S Callaway : Online convex optimization with binary constraints. *IEEE Transactions on Automatic Control*, 2021.

[96] Dan Levi, Liran Gispan, Niv Giladi et Ethan Fetaya : Evaluating and calibrating uncertainty prediction in regression tasks. *arXiv:1905.11659*, Feb 2020.

[97] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo et Li-Jia Li : Learning from noisy labels with distillation. *arXiv:1703.02391 [cs, stat]*, Apr 2017.

[98] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver et Daan Wierstra : Continuous control with deep reinforcement learning. *arXiv:1509.02971*, Jul 2019. arXiv:1509.02971 [cs, stat].

[99] M Liu et Y Shi : Model predictive control of aggregated heterogeneous second-order thermostatically controlled loads for ancillary services. *IEEE Trans. on Power Systems*, 31(3):1963–1971, 2015.

[100] Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin et Birgitta Dresp-Langley : Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. *Robotics*, 10(1):22, Jan 2021. arXiv: 2102.04148.

[101] Tongliang Liu et Dacheng Tao : Classification with noisy labels by importance reweighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(3):447–461, Mar 2016. arXiv: 1411.7718.

[102] Z. P. Liu et J. P. Castagna : Avoiding overfitting caused by noise using a uniform training mode. *In IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 3, pages 1788–1793 vol.3, 1999.

[103] Christos Louizos et Max Welling : Multiplicative normalizing flows for variational bayesian neural networks. (arXiv:1703.01961), Jun 2017. arXiv:1703.01961 [cs, stat].

[104] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel et Igor Mordatch : Multi-agent actor-critic for mixed cooperative-competitive environments. Mar 2020. arXiv:1706.02275 [cs].

[105] Yueming Lyu et Ivor W. Tsang : Curriculum loss: Robust learning and generalization against label corruption. *arXiv:1905.10045 [cs, stat]*, Feb 2020.

[106] Xiaoteng Ma, Li Xia, Zhengyuan Zhou, Jun Yang et Qianchuan Zhao : Dsac: Distributional soft actor critic for risk-sensitive reinforcement learning. (arXiv:2004.14547), Jun 2020. arXiv:2004.14547 [cs].

[107] Xingjun Ma, Yisen Wang, Michael E. Houle, Shuo Zhou, Sarah M. Erfani, Shu-Tao Xia, Sudanthi Wijewickrema et James Bailey : Dimensionality-driven learning with noisy labels. *arXiv:1806.02612 [cs, stat]*, Jul 2018.

[108] Mehdi Maasoumy, Borhan M Sanandaji, Alberto Sangiovanni-Vincentelli et Kameshwar Poolla : Model predictive control of regulation services from commercial buildings to the smart grid. *In 2014 American Control Conference*, pages 2226–2233. IEEE, 2014.

[109] Vincent Mai, Waleed Khamies et Liam Paull : Batch inverse-variance weighting: Deep heteroscedastic regression. *arXiv:2107.04497*, Jul 2021.

[110] J L Mathieu, S Koch et D S Callaway : State estimation and control of electric loads to manage real-time energy imbalance. *IEEE Trans. on Power Systems*, 28(1):430–440, 2012.

[111] Laetitia Matignon, Guillaume J. Laurent et Nadine Le Fort-Piat : Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, Feb 2012.

[112] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver et Koray Kavukcuoglu : Asynchronous methods for deep reinforcement learning. *arXiv:1602.01783 [cs]*, Jun 2016. arXiv: 1602.01783.

[113] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra et Martin Riedmiller : Playing Atari with deep reinforcement learning. *NIPS*, page 9, 2013.

[114] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski et et al. : Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[115] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg et Demis Hassabis : Human-level control through deep reinforcement learning. *Nature*, 518(75407540):529–533, Feb 2015.

[116] Aryan Mobiny, Pengyu Yuan, Supratik K. Moulik, Naveen Garg, Carol C. Wu et Hien Van Nguyen : Dropconnect is effective in modeling uncertainty of bayesian deep networks. *Scientific Reports*, 11(11):5458, Mar 2021.

[117] Kevin P. Murphy : *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[118] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar et Ambuj Tewari : Learning with noisy labels. *In* C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani et K. Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems*, volume 26, page 1196–1204. Curran Associates, Inc., 2013.

[119] J. A. Nelder et R. Mead : A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[120] D. A. Nix et A. S. Weigend : Estimating the mean and variance of the target probability distribution. *In Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60 vol.1, 1994.

[121] Mohammed M. Olama, Teja Kuruganti, James Nutaro et Jin Dong : Coordination and control of building hvac systems to provide frequency regulation to the electric grid. *Energies*, 11(7), 2018.

[122] Frans A. Oliehoek et Christopher Amato : *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer International Publishing, Cham, 2016.

[123] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski et Susan Zhang : Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680*, Dec 2019. arXiv: 1912.06680.

[124] Ian Osband, John Aslanides et Albin Cassirer : Randomized prior functions for deep reinforcement learning. *In Advances in Neural Information Processing Systems*, 2018.

[125] Ian Osband, Charles Blundell, Alexander Pritzel et Benjamin Van Roy : Deep exploration via bootstrapped dqn. *Advances in Neural Information Processing Systems*, 29:4026–4034, 2016.

[126] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh et et al. : Behaviour suite for reinforcement learning. *arXiv:1908.03568*, Feb 2020.

[127] Ian Osband, Benjamin Van Roy, Daniel J. Russo et Zheng Wen : Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019.

[128] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan et Jasper Snoek : Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in Neural Information Processing Systems*, 32:13991–14002, 2019.

[129] Deepak Pathak, Pulkit Agrawal, Alexei A Efros et Trevor Darrell : Curiosity-driven exploration by self-supervised prediction. *In International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

[130] Valentin Peretroukhin, Brandon Wagstaff et Jonathan Kelly : Deep probabilistic regression of elements of so(3) using quaternion averaging and uncertainty injection. *In CVPR Workshops*, 2019.

[131] Aisling Pigott, Constance Crozier, Kyri Baker et Zoltan Nagy : Gridlearn: Multiagent reinforcement learning for grid-aware building energy management, 2021.

[132] Zhiwei Qin, Hongtu Zhu et Jieping Ye : Reinforcement learning for ridesharing: An extended survey. Aug 2022. arXiv:2105.01099 [cs].

[133] Stephan Rasp, Michael S. Pritchard et Pierre Gentine : Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, Sep 2018.

[134] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan et Andrew Rabinovich : Training deep neural networks on noisy labels with bootstrapping. *arXiv:1412.6596 [cs]*, Apr 2015.

[135] Martin Roesch, Christian Linder, Roland Zimmermann, Andreas Rudolf, Andrea Hohmann et Gunther Reinhart : Smart grid for industry using multi-agent reinforcement learning. *Applied Sciences*, 10(19), 2020.

[136] David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Kording, Carla Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer Chayes et Yoshua Bengio : Tackling climate change with machine learning. (arXiv:1906.05433), Nov 2019. arXiv:1906.05433 [cs, stat].

[137] David Rolnick, Andreas Veit, Serge Belongie et Nir Shavit : Deep learning is robust to massive label noise. *arXiv:1705.10694 [cs]*, Feb 2018.

[138] Joshua Romoff, Peter Henderson, Alexandre Piché, Vincent François-Lavet et Joelle Pineau : Reward estimation for variance reduction in deep reinforcement learning. (CoRL):1–26, 2018.

[139] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband et Zheng Wen : A tutorial on thompson sampling. (arXiv:1707.02038), Jul 2020. arXiv:1707.02038 [cs].

[140] Y. Sai, R. Jinxia et L. Zhongxia : Learning of neural networks based on weighted mean squares error function. *In 2009 Second International Symposium on Computational Intelligence and Design*, volume 1, page 241–244, Dec 2009.

[141] Tom Schaul, John Quan, Ioannis Antonoglou et David Silver : Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[142] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford et Oleg Klimov : Proximal policy optimization algorithms. *arXiv:1707.06347*, Aug 2017.

[143] Cosma Rohilla Shalizi : *Advanced Data Analysis from an Elementary Point of View*, 2019. (Accessed November 13th, 2020).

[144] Yanyao Shen et Sujay Sanghavi : Learning with bad training data via iterative trimmed loss minimization. *arXiv:1810.11874 [cs, stat]*, Feb 2019.

[145] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu et Deyu Meng : Meta-weight-net: Learning an explicit mapping for sample weighting. *arXiv:1902.07379 [cs, stat]*, Sep 2019.

[146] Pierluigi Siano : Demand response and smart grids—a survey. *Renewable and sustainable energy reviews*, 30:461–478, 2014.

[147] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot et et al. : Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, Jan 2016.

[148] Hwanjun Song, Minseok Kim, Dongmin Park et Jae-Gil Lee : Learning from noisy labels with deep neural networks: A survey. *arXiv:2007.08199 [cs, stat]*, Jul 2020.

[149] Yang Song et Zhifei Zhang : *UTKFace, Large Scale Face Dataset*, 2017. (Accessed June 11, 2020).

[150] Malcolm Strens : A bayesian framework for reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning*, 02 2001.

[151] Jayakumar Subramanian, Raihan Seraj et Aditya Mahajan : Reinforcement learning for mean-field teams. *In Workshop on Adaptive and Learning Agents at the International Conference on Autonomous Agents and Multi-Agent Systems*, 2018.

[152] Sainbayar Sukhbaatar, Arthur Szlam et Rob Fergus : Learning multiagent communication with backpropagation. *Advances in Neural Information Processing Systems*, page 2252–2260, 2016.

[153] Richard S. Sutton : Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[154] Richard S. Sutton et Andrew G. Barto : *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, second edition édition, 2018.

[155] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford et et al. : The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4–5): 405–420, Apr 2018.

[156] Ryutaro Tanno, Ardavan Saeedi, Swami Sankaranarayanan, Daniel C. Alexander et Nathan Silberman : Learning from noisy labels by regularized estimation of annotator confusion. *In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 11236–11245. IEEE, Jun 2019.

[157] Josh A Taylor, Sairaj V Dhople et Duncan S Callaway : Power systems without fuel. *Renewable and Sustainable Energy Reviews*, 57:1322–1336, 2016.

[158] William R. Thompson : On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285, Dec 1933.

[159] Sebastian Thrun, Wolfram Burgard et Dieter Fox : *Probabilistic Robotics*. Intelligent robotics and autonomous agents. MIT Press, 2006.

[160] T. Tieleman et G. Hinton : RMSProp: Divide the gradient by a running average of its recent magnitude., 2012.

[161] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona et Serge Belongie : Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. *In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 595–604. IEEE, Jun 2015.

[162] Jose R. Vazquez-Canteli, Sourav Dey, Gregor Henze et Zoltan Nagy : Citylearn: Standardizing research in multi-agent reinforcement learning for demand response and urban energy management. Dec 2020. arXiv:2012.10504 [cs].

[163] Jose R. Vazquez-Canteli, Gregor Henze et Zoltan Nagy : Marlisa: Multi-agent reinforcement learning with iterative sequential action selection for load shaping of grid-interactive connected buildings. *In Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, BuildSys '20, page 170–179, New York, NY, USA, 2020. Association for Computing Machinery.

[164] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel et Jimmy Ba : Benchmarking model-based reinforcement learning. *arXiv:1907.02057*, Jul 2019.

[165] Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi et James Bailey : Symmetric cross entropy for robust learning with noisy labels. *arXiv:1908.06112 [cs, stat]*, Aug 2019.

[166] Zhe Wang, Bingqing Chen, Han Li et Tianzhen Hong : Alphabuilding rescommunity: A multi-agent virtual testbed for community-level load coordination. *Advances in Applied Energy*, 4:100061, 2021.

[167] Chris Watkins et Peter Dayan : Q-learning. *Machine Learning*, May 1992.

[168] Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu et Zhenhui Li : Colight: Learning network-level cooperation for traffic signal control. *In Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1913–1922, New York, NY, USA, Nov 2019. Association for Computing Machinery.

[169] Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard E. Turner, José Miguel Hernández-Lobato et Alexander L. Gaunt : Deterministic variational inference for robust bayesian neural networks. (arXiv:1810.03958), Mar 2019. arXiv:1810.03958 [cs, stat].

[170] Xiaoyu Wu, Jinghan He, Yin Xu, Jian Lu, Ning Lu et Xiaojun Wang : Hierarchical control of residential hvac units for primary frequency regulation. *IEEE Transactions on Smart Grid*, 9(4):3844–3856, 2018.

[171] Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua M Susskind, Jian Zhang, Ruslan Salakhutdinov et Hanlin Goh : Uncertainty weighted actor-critic for offline reinforcement learning. *In Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 11319–11328. PMLR, 18–24 Jul 2021.

[172] Lei Xi, Jianfeng Chen, Yuehua Huang, Yanchun Xu, Lang Liu, Yimin Zhou et Yudan Li : Smart generation control based on multi-agent reinforcement learning with the idea of the time tunnel. *Energy*, 153:977–987, 2018.

[173] Jun Xie, Martin Kiefel, Sun Ming-Ting et Andreas Gieger : Kitti-360, 2020.

[174] Jun Xie, Martin Kiefel, Ming-Ting Sun et Andreas Geiger : Semantic instance annotation of street scenes by 3d to 2d label transfer. *In Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[175] Aolin Xu et Maxim Raginsky : Minimum excess risk in bayesian learning. (arXiv:2012.14868), Dec 2020. arXiv:2012.14868 [cs, math, stat].

[176] Yaodong Yang, Jianye Hao, Yan Zheng, Xiaotian Hao et Bofeng Fu : Large-scale home energy management using entropy-based collective multiagent reinforcement learning framework. *In Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 2285–2287, Richland, SC, May 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[177] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang et Jun Wang : Mean field multi-agent reinforcement learning. Feb 2018.

[178] Kun Yi et Jianxin Wu : Probabilistic end-to-end noise correction for learning with noisy labels. *arXiv:1903.07788 [cs]*, Mar 2019.

[179] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen et Yi Wu : The surprising effectiveness of ppo in cooperative, multi-agent games. Jul 2021. arXiv:2103.01955 [cs].

[180] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W. Tsang et Masashi Sugiyama : How does disagreement help generalization against label corruption? *arXiv:1901.04215 [cs, stat]*, May 2019.

[181] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht et Oriol Vinyals : Understanding deep learning requires rethinking generalization. *arXiv:1611.03530 [cs]*, Feb 2017.

[182] Kaiqing Zhang, Zhuoran Yang et Tamer Başar : Multi-agent reinforcement learning: A selective overview of theories and algorithms. (arXiv:1911.10635), Nov 2019. arXiv:1911.10635 [cs, stat].

[183] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang et Tamer Başar : Fully decentralized multi-agent reinforcement learning with networked agents. (arXiv:1802.08757), Feb 2018. arXiv:1802.08757 [cs, math, stat].

[184] Wei Zhang, Jianming Lian, Chin-Yao Chang et Karanjit Kalsi : Aggregated modeling and control of air conditioning loads for demand response. *IEEE transactions on power systems*, 28(4):4655–4664, 2013.

[185] X Zhou, E Dall'Anese et L Chen : Online stochastic optimization of networked distributed energy resources. *IEEE Trans. on Automatic Control*, 65(6):2387–2401, 2019.

[186] Changxi Zhu, Mehdi Dastani et Shihan Wang : A survey of multi-agent reinforcement learning with communication. (arXiv:2203.08975), Mar 2022. arXiv:2203.08975 [cs].

Appendix A

Batch Inverse-Variance weighting: Supplementary material

In this appendix, we provide details about the datasets and neural network training used in Article 1. We also present additional results about the influence of ϵ, the behavior of BIV on different variance distributions, and the robustness of BIV on variance errors and mini-batch size.

A.1. Data sets and Neural Networks

A.1.1. UTKFace

A.1.1.1. Dataset description. The UTKFace Aligned&Cropped dataset [**149**] consists of 20,000 pictures of faces labelled with their age, ranging from 0 to 116 years. We use it in a regression setting: the network must predict the age of a person given the photo of their face. Unless described otherwise, 16,000 images were used for training, and 4,000 for testing.

Some images are in black and white and some are in color. The pixel dimension of each image is 200x200.

Both the pixels and the labels were normalized before the training, so that their mean is 0 and standard deviation is 1 over the whole dataset. The noise variances were correspondingly scaled, as well as the cutoff threshold if applicable.

A.1.1.2. Neural network and training hyper-parameters. The model that we used was a Resnet-18 [**65**], not pretrained. It was trained with an Adam optimizer [**78**], a learning rate of 0.001 over 20 epochs. A batch size of 256 was used in order to ensure the best performance for the L2 method with noisy labels as well as to reduce the time necessary to the training process.

A.1.2. Bike Sharing Dataset

A.1.2.1. Dataset description. The Bike Sharing Dataset [**45**] consists of 17,379 samples of structured data. It contains, for nearly each hour of the years 2011 and 2012, the date, season, year, month, hour, day of the week, a boolean for it being a holiday, a boolean for it being a working day, the weather situation on a scale of 4 (1: clear and beautiful, 4: stormy or snowy), the temperature, the feeling temperature, the humidity, and the wind speed, in the city of Washington DC. It also contains the number of casual, registered, and total bike renters for each hour as recorded on the Capital Bikeshare system.

We use it in a regression setting: the network must predict the total number of bike renters given the time and weather information. Unless described otherwise, 7,000 samples were used for training, and 3,379 for testing. We used less samples than available for training because the low-data situation, noise has a stronger effect on the performance. The minimal test loss achieved with 7000 noiseless samples was very close to the one with 14000 samples, hinting that the additional samples did not give a lot of additional information.

We applied some pre-processing on the data to make it easier for the network to learn. First, the date was normalized from a scale between day 1 to day 730 to a scale between 0 and 4π. Then, we provided the network with the cosine and the sine of this number. This allowed to have the same representation for the same days of the year, while having the same distance between any two consecutive days, keeping the cyclic nature of a year. A similar idea was applied to hours, normalized from 0 to 2π instead of 0 to 24, and with the cosine and sine given to the network. The day of the week, being a category, was given as a one-hot vector of dimension 7. We also removed the season and the month as it was redundant information with the date.

Overall, the number of features was 19:

1	Year
2-4	Date (sine and cos)
4-5	Hour (sine and cos)
6-12	Days of the week (one-hot vector)
13	Holiday boolean
14	Working day boolean
15	Weather situation
16	Temperature
17	Felt temperature
18	Humidity
19	Wind speed

We observed that the network was learning significantly faster and better provided with this format for the data.

Both the features and the labels were normalized before the training, so that their mean is 0 and standard deviation is 1 over the whole dataset. The noise variances were correspondingly scaled, as well as the cutoff threshold if applicable.

A.1.2.2. Neural network and training hyper-parameters. The model that we used was a multi-layer perceptron with 4 hidden layers, the first one with 100 neurons, then 50, 20, and 10. The activation function was ReLU. We did not use any additional technique such as batch normalization as it did not improve the performances.

The model was trained over 100 epochs on mini-batches of size 256 for similar reasons than explained in section A.1.1.2, using the Adam optimizer with learning rate 0.001.

A.2. Additional experiments

A.2.1. The influence of ϵ

In this section, we provide experimental results justifying our recommendation of the range of $[10^{-2}; 10^{-1}]$ for ϵ. In the experiments presented in this article, we have mainly used $\epsilon = 5 \times 10^{-2}$, and sometimes $\epsilon = 10^{-1}$. ϵ must be chosen as part of a trade-off between mitigating the effect of BIV with near ground-truth labels while keeping its effect with noisy labels. To better understand these results, it is important to remember that ϵ is added to the variance that is used in the loss function. When the labels are normalized - which is the case in our work -, the noise and its variance for each label is normalized too. The value we recommend for ϵ should therefore be valid for any normalized set of labels.

The main role of ϵ is to set a maximal weight to the samples and prevent a near-zero variance sample to effectively reducing the minibatch to itself, thus ignoring the other potentially valid samples. We tested several values of ϵ on both the UTKF and BikeSharing datasets, with $P(\sigma^2)$ being Gamma distributions with $\alpha = 1$ and $\mu_P = 2000$ and $\mu_P = 20000$ respectively.

The resulting graph can be seen in figures 1 and 2

In both experiments, the optimal value for ϵ is 0.05. Between 0.02 and 0.1, the performances are acceptable. When ϵ is too small, the learning process gives too much importance to the low-noise samples, as seen in section 3.6.4. When it is too high, it leans too much towards L2.

This can be seen in figure 3, which applies BIV with different value for ϵ on a binary distribution on UTKF.

The fact that in the three, the same values for ϵ are optimal, even though tasks as well as the noise distribution are different, shows that these values are valid for most cases with normalized labels.

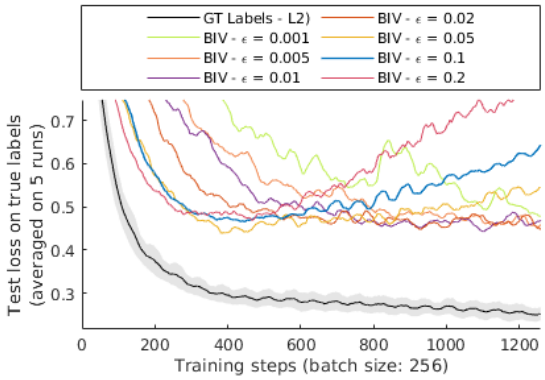


Fig. 1. Results of running BIV with different values of ϵ on UTKF with $P(\sigma^2)$ as a Gamma distribution with $\alpha = 1, \mu_P = 2000$.

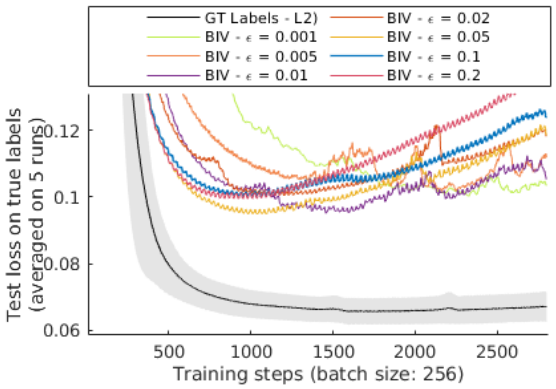


Fig. 2. Results of running BIV with different values of ϵ on BikeSharing with $P(\sigma^2)$ as a Gamma distribution with $\alpha = 1, \mu_P = 20000$.

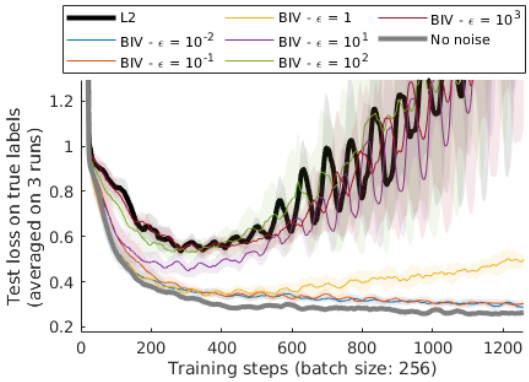


Fig. 3. Impact of ϵ when training with highly noisy labels using BIV loss on UTKF dataset. The variance was sampled through a binary uniform distribution with $p = 0.5$, $\mu_P = 2000$, and $V_h = 0$. Very high ϵ shows a loss of performance as BIV approaches the L2 results.

A.2.2. BIV on different distributions

A.2.2.1. Uniform distributions. We present in figure 5 the results of the experiment with uniform distributions in more details.

As explained in section 3.6.3, we observe that BIV and L2 have the same performances when $V = 0$ (and $a = b = \mu_P$). This is to be expected, as all samples have the exact same noise variance and thus the same weights. When $V = V_{max}$ ($a = 0$ and $b = 2\mu_P$), BIV has an advantage, as it is able to differentiate the samples and use the support of low-noise labels. When $V = V_{max}/2$ ($a = 0.293\mu_P$ and $b = 1.707\mu_P$), the difference between the samples is less important, and BIV only does a bit better than L2 on BikeSharing. On UTKF, the process has more variability and it is difficult to detect this effect.

In all cases, the benefit from using BIV is less important than with Gamma distributions with $\alpha \leq 1$, where the support on low-noise samples is higher.

A.2.3. BIV on Gamma distributions

A.2.3.1. $\alpha \leq 1$**.** As described in section 3.6.3, the smaller α, the better the performance of BIV and cutoffs. We show in figures 7 and 8 the curves that led to the numbers in Table 1. BIV consistently outperforms the other methods. The performance of cutoff methods strongly depends on C, and the best value of C is not the same for every distribution P.

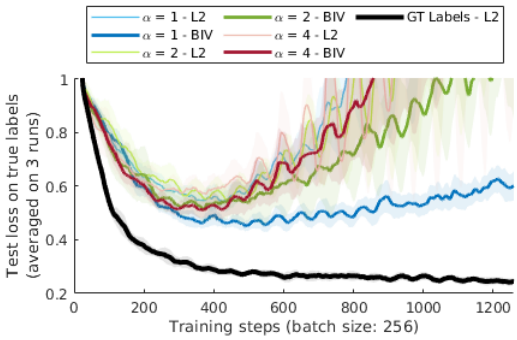


Fig. 4. Test loss on the UTKF dataset for L2 and BIV learning on Gamma function with $\alpha \geq 1$.

A.2.3.2. $\alpha > 1$**.** When $\alpha > 1$, the highest support of $P(\sigma^2)$ shifts towards μ_P. This makes the samples less distinguishable for BIV and therefore the benefits of using it are reduced. This is shown in Figure 4 on UTKF.

A.2.4. Robustness of BIV

A.2.4.1. Size of the mini-batches. In equation (3.4.3), the normalization constant is computed from the samples in the mini-batch. If the distribution of the noise variances in

mini-batch is representative of the whole training dataset, the relative weight given to each sample with respect to the others is the same than if the normalization was made over the whole dataset. The larger the mini-batch, the more representative it is. In our experiments, we used a size of 256, which is arguably high. We tested our algorithm with lower batch sizes, from 16 to 128, to see if it was a critical factor in the performances.

The results are presented in figure 9. In UTKF, the batch size does not make any significant difference in the performance with respect to the amount of samples seen, except for a slightly steeper overfitting once the best loss has been achieved. In BikeSharing, a smaller batch size makes the training faster with respect to the amount of samples, but with a higher minimal loss, for both L2 and BIV. While a larger batch size leads to a lower loss function, the effect of BIV compared to the corresponding L2 curve is not compromised by smaller batch-sizes.

Two main factors may explain this robustness. First, a mini-batch of size 16 seems to be already representative enough of the whole dataset for the purpose of normalization. Second, the fact that the mini-batches are populated differently at every epoch improves the robustness as a sample who would have been in a non-representative batch at one epoch may not be at another epoch. In any case, the size of the mini-batch is not a critical factor for BIV.

A.2.4.2. Noisy variance estimation. In many scenarios, the variance σ^2 from which the noise was sampled is estimated, or inferred from a proxy, and therefore prone to be noisy itself. We tested the robustness of our method to such variance noise. In this experimental setup, the value given to the BIV algorithm is disturbed by noise $\delta_{\sigma_i^2}$. We modelled this noise on σ_i^2 to be sampled from a normal distribution whose standard deviation is proportional to σ_i^2 with a coefficient of variance disturbance D_v:

$$\delta_{\sigma_i^2} \sim \mathcal{N}(0, D_v \sigma_i^2/9) \tag{A.2.1}$$

Dividing σ_i^2 by 9 allows to scale D_v so that, when $D_v = 1$, $\delta_{\sigma_i^2} < -\sigma_i^2$ is at 3 standard deviations from the mean.

We then compute the noisy variance, which needs to be positive, as $\tilde{\sigma}_i^2 = \left|\sigma_i^2 + \delta_{\sigma_i^2}\right|$. The noise is therefore biased, but when $D_v \leq 1$, this is negligible as it happens with probability less than 0.15%.

The results presented in figure 10 show that, when $D_v \leq 1$, BIV is robust to such noise. While a higher D_v leads to lower performance, the impact is small compared to the effect of BIV. However, when $D_v = 2$, which is an arguably high level of noise and leads to bias as explained previously, the beneficial effect of BIV is significantly affected in BikeSharing, and completely disappears in UTKF.

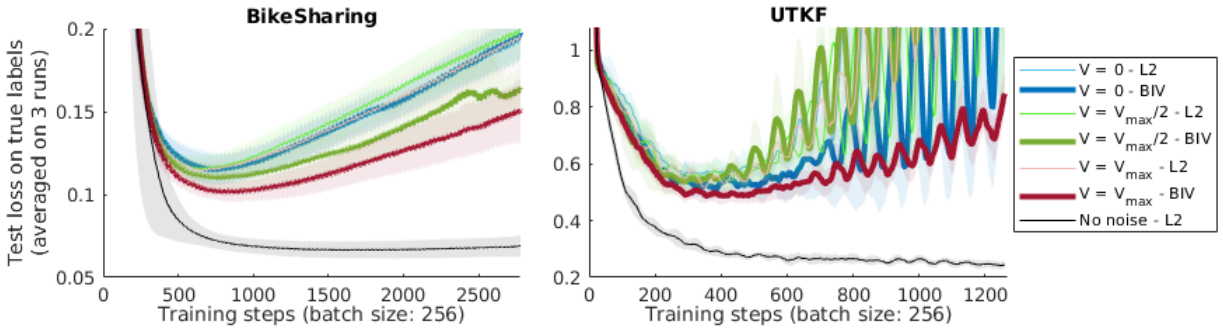


Fig. 5. Test loss for L2 and BIV learning on uniform with different variances V.

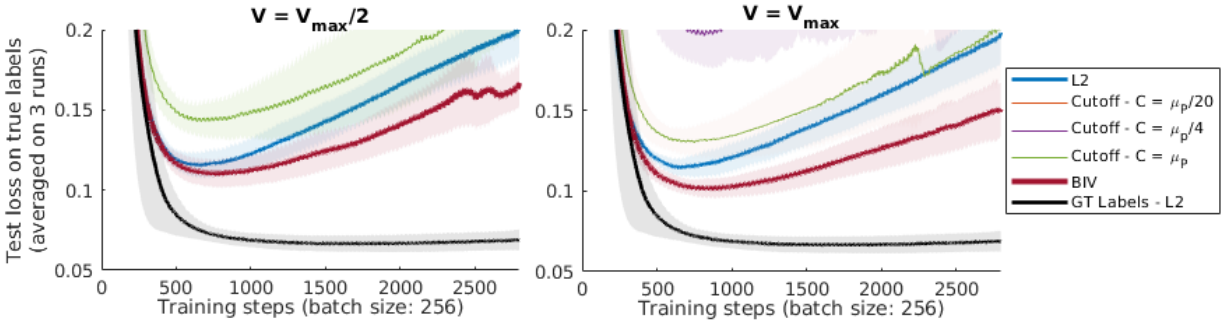


Fig. 6. On BikeSharing with $\mu_P = 20000$, using cutoff is not helpful in the uniform setting. This is due to the significant loss of information induced by such a strategy.

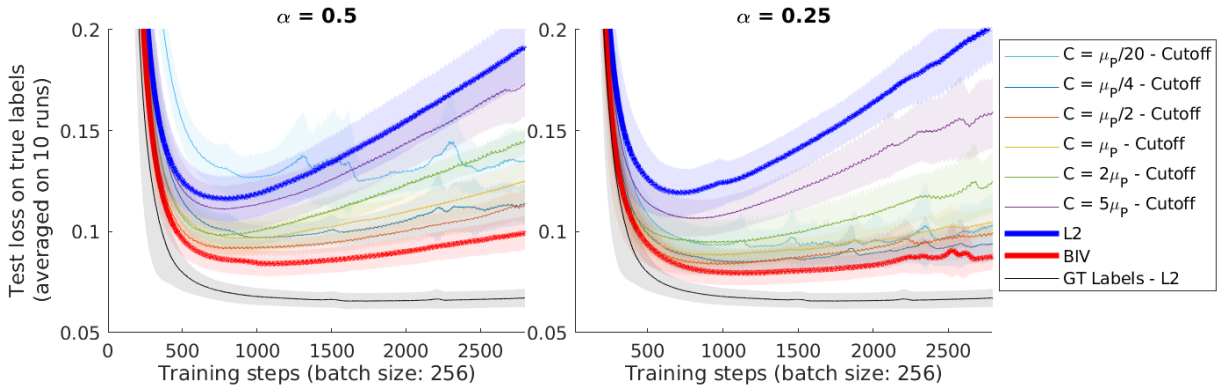


Fig. 7. Test loss on the BikeSharing dataset, with $\alpha \leq 1$

In this setting, we also show as shown in figure 6 that cutting off the noisy data is not a good strategy, as it always performs worse than L2.

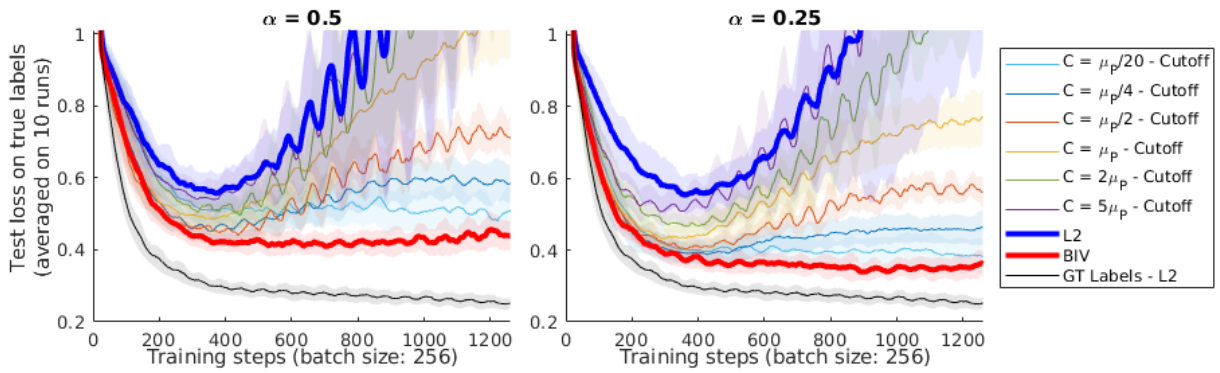


Fig. 8. Test loss on the UTKF dataset, with $\alpha \leq 1$

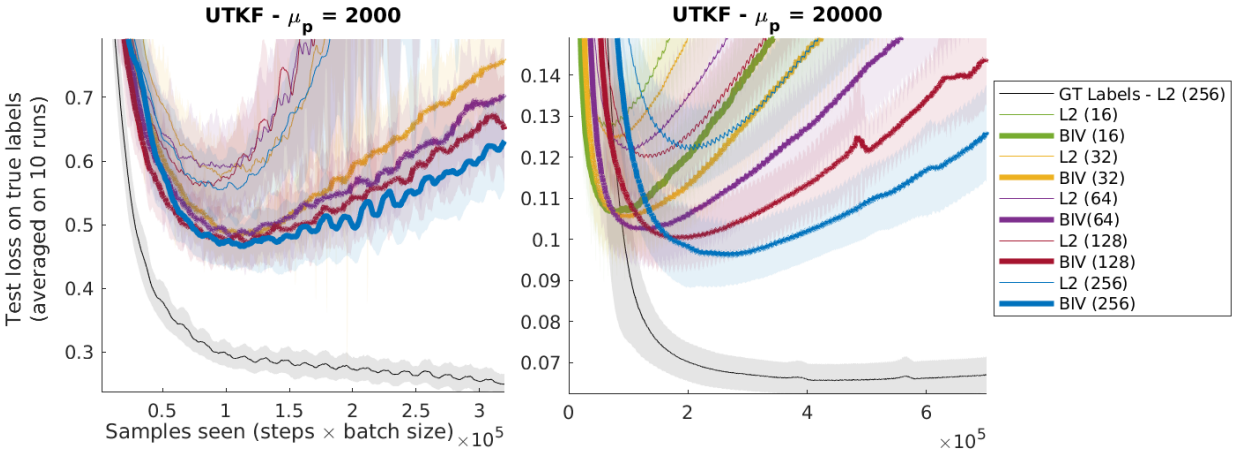


Fig. 9. BIV with different batch sizes in both UTKF and BikeSharing datasets.

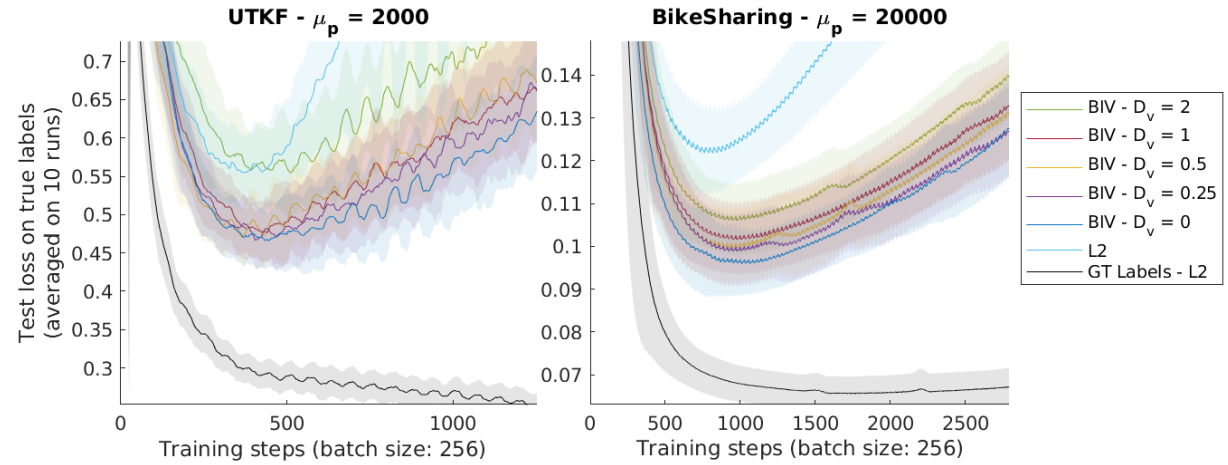


Fig. 10. Robustness of BIV to noise in the variance with different disturbance coefficients D_v.

Appendix B

Sample efficient reinforcement learning via uncertainty estimation: Supplementary material

In this appendix, we provide supplementary material for Article 2. We first provide an accounting of the carbon emissions of this research project. We then describe each algorithm in details. We also provide the hyperparameters used to train them, we discuss the influence of λ in $\mathcal{L}_{\text{IVRL}}$ and the computation time of our algorithm. We discuss the evolution of variance estimation, compare different weighting schemes and evaluate the performance of single variance networks. We finally discuss on the relationship between IVRL and optimism in face of uncertainty.

B.1. Carbon emissions of the research project

As a significant amount of electricity has been used to train and run the models for this work, we publish its estimated carbon footprint.

Experiments were conducted using a private infrastructure, which has a carbon efficiency of 0.028 kgCO_2eq/kWh. A cumulative of 12367 days, or 296808 hours, of computation was mainly performed on the hardware of type RTX 8000 (TDP of 260W). We assume full power usage of the GPUs, although this was not always the case. Total emissions are estimated to be 2160.76 kgCO_2eq of which 0 percents were directly offset. This is equivalent to 8730 km driven by an average car, or 1.08 metric tons of burned coal.

Estimations were conducted using the MachineLearning Impact calculator presented in [**87**].

B.2. Appendix - Algorithms in details

In this section, we are detailing the different algorithms described in the paper, and explaining the design choices that we made.

B.2.1. BootstrapDQN

BootstrapDQN [**125**], improved with Randomized Prior Function (RPF) [**124**], is an ensemble-based modification of DQN [**113**]. Instead of a Q- and a target network, we use a Q-ensemble and a target ensemble. It is the base algorithm upon which IV-DQN is built. The detailed steps at training are presented in algorithm 1[1].

Network architecture. The Q-ensemble and the target ensemble are initialized identically with $N = 5$ neural networks. We used multi-layer perceptrons with 2 ReLU-activated hidden layers of 64 nodes.

TD update. Following the implementation of [**125**], the networks are paired one to one between the Q- and the target networks during the TD update. The system can be considered as several DQNs working in parallel. At training time, the best action a' for each Q-network is selected based on the corresponding target network.

Exploration. During exploration at training time, one network of the ensemble is used to predict the best action for a whole episode. This follows [**125**], and is an approximation of Thompson sampling [**158**]. We added random prior functions [**124**] to tackle the variance underestimation at early training.

At test time, the best action a' is instead selected by a vote from the networks in the ensembles. Ties are broken randomly. Using the mean value across the ensemble was also tested, with slightly lower performances.

Masked replay buffer. To mask the replay buffer, each step saved in the replay buffer is associated with a N-sized boolean vector m which is used as a mask during training. Each element of m indicates if the corresponding network in the Q-ensemble should be trained using this sample. These elements of m are independently generated when the sample is saved, using a Bernoulli distribution with probability p_m. They are kept fixed during the training. The role of the mask is to ensure variability in the training of the different networks of the Q-ensemble and thus maintain a significant variance prediction.

B.2.2. IV-DQN

IV-DQN is built upon BootstrapDQN. The main differences are the use of var-networks in the ensemble, the computation of the variance in the TD update, and the use of $\mathcal{L}_{\text{IVRL}}$ when training the networks' parameters commonly used to represent epistemic uncertainty

[1]The presentation of algorithms is optimized for readability. During the implementation, many for-loops are replaced by tensor computation or parallel computing.

Algorithm 1 BootstrapDQN - Training

Input: RPF scale δ_{RPF}, minibatch size K, mask probability p_m, ensemble of N Q-networks $\mathcal{Q}$, ensemble of N target-networks $\mathcal{T}$, ensemble of N prior-networks P
for each episode **do**
  Pick a Q-network from $\mathcal{Q}$ using $i \sim \text{Uniform}\,(1..N)$ ▷ [**125**]
  for each time step **do**
    $a \leftarrow \arg\max_\alpha \left(\mathcal{Q}_i(s,\alpha) + \delta_{\text{RPF}} P_i(s,\alpha)\right)$ ▷ Random prior [**124**]
    Collect next-state s' and reward r from the environment by taking action a
    Sample bootstrap mask $M = m_l \sim \{\text{Bernoulli}(p_m) | l \in 1, ..., N\}$
    Add (s, a, s', r, M) in buffer $\mathcal{B}$
    $s \leftarrow s'$
    Sample minibatch D from replay buffer $\mathcal{B}$
    for j in $[1 \dots N]$ **do** ▷ Each member of the ensemble
      for each tuple $(s_k, a_k, s'_k, r_k, M_k)$ in minibatch D **do** ▷ Computing the targets
        $\hat{Q}_k \leftarrow \mathcal{Q}_j(a_k, s_k) + \delta_{\text{RPF}} P_j(s_k, a_k))$ ▷ Expected Q-value
        $a'_k \leftarrow \arg\max_a \left(\mathcal{T}_j(s'_k, a) + \delta_{\text{RPF}} P_j(s'_k, a)\right)$
        $T_k \leftarrow r_k + \gamma(\mathcal{T}_j(s'_k, a'_k) + \delta_{\text{RPF}} P_j(s'_k, a'_k))$ ▷ Target
      end for
      Mask minibatch D using M_j
      Update $\mathcal{Q}_j$'s parameters θ_j by minimizing $L2$ loss: $(\hat{Q} - T)^2$ over D
      Soft update of the $\mathcal{T}_j$'s parameters $\bar{\theta}_j \leftarrow (1-\tau)\bar{\theta}_j + \tau\theta_j$
    end for
  end for
end for

in ML. The pseudo-code for IV-DQN is given in algorithm 2, where the differences with BootstrapDQN are highlighted. At test time, action selection for IV-DQN is same as BootstrapDQN.

Var-networks architecture. Similar to BootstrapDQN, we used ensembles consisting of $N = 5$ networks. The var-networks were also multi-layer perceptrons with 2 ReLU-activated hidden layers of 64 nodes. The only difference is that, instead of a single head for a given (s,a), these networks now provide 2 heads to predict the mean μ_θ and the variance σ^2_θ of the Q-value. To differentiate these outputs of network $\mathcal{N}$ given a state-action pair (s,a), we write $\mathcal{N}^\mu(s,a)$ and $\mathcal{N}^{\sigma^2}(s,a)$.

Computing the uncertainty of $\bar{Q}(s',a')$. As justified in section 5.3, variance $\sigma^2_{\bar{Q}(s',a')}$ is estimated using var-ensembles. For each (s'_k, a'_k) pair used to train a single Q-network $\mathcal{Q}_j$ in ensemble $\mathcal{Q}$, the value of $\bar{Q}(s'_k, a'_k)$ is the mean $\mu_{k,j}$ predicted by the corresponding target-network $\mathcal{T}_j$. However, to evaluate $\sigma^2_{\bar{Q}}$, every target-network $\mathcal{T}_l$ in the target ensemble predicts a mean $\mu_{k,l}$ and a variance $\sigma^2_{k,l}$ for $\bar{Q}(s'_k, a'_k)$. Then, $\sigma^2_{\bar{Q},k}(s'_k, a'_k)$ is computed as the variance of the mixture of Gaussians composed of all the $\mu_{k,l}$ and $\sigma^2_{k,l}$:

$$\sigma^2_{\bar{Q},k} = \frac{1}{N}\sum_{l=1}^{N} \sigma^2_{k,l} + \mu^2_{k,l} - \mu^2_{k,\text{all}} \tag{B.2.1}$$

where $\mu^2_{k,\text{all}}$ is the average of all $\mu^2_{k,l}$'s.
Using $\mathcal{L}_{\text{IVRL}}$. Given $\sigma^2_{\bar{Q}}$, each network can now learn to predict mean μ and variance σ^2 during the TD update using $\mathcal{L}_{\text{IVRL}}$ instead of $L2$. To select the value of ξ, we specify a minimal effective batch size $MEBS$. We use equation (5.2.2) to determine if, for the values of $\sigma^2_{\bar{Q},k}$ and $\xi = 0$, the effective batch size EBS is lower than $MEBS$. If this is the case, ξ is increased until $EBS \geq MEBS$ using a Nelder Mead optimization method [**119**].

Algorithm 2 IV-DQN - Training. In red, the differences with BootstrapDQN.

Input: RPF scale δ_{RPF}, minibatch size K, mask probability p_m, LA weight λ, minimal effective batch size $MEBS$, initial state s, var-ensemble of N Q-var-networks $\mathcal{Q}$, var-ensemble of N target-var-networks $\mathcal{Q}$, var-ensemble of N prior-var-networks P
for each episode **do**
  Pick a Q-network from $\mathcal{Q}$ using $i \sim \text{Uniform}\,(1..N)$ ▷ [**125**]
  for each time step **do**
    $a \leftarrow \arg\max_\alpha \left(\mathcal{Q}_i(s,\alpha) + \delta_{\text{RPF}} P_i(s,\alpha)\right)$ ▷ Random prior [**124**]
    Collect next-state s' and reward r from the environment by taking action a
    Sample bootstrap mask $M = m_l \sim \{\text{Bernoulli}(p_m) | l \in 1, ..., N\}$
    Add (s, a, s', r, M) in buffer $\mathcal{B}$
    $s \leftarrow s'$
    Sample minibatch D from replay buffer $\mathcal{B}$
    for j in $[1 \dots N]$ **do**
      for each tuple $(s_k, a_k, s'_k, r_k, M_k)$ in mini-batch D **do**
        $\mu_{\hat{Q},k} \leftarrow \mathcal{Q}^\mu_j(a_k, s_k) + \delta_{\text{RPF}} P_j(s_k, a_k))$ ▷ Expected Q-value
        $a'_k \leftarrow \arg\max_a \left(\mathcal{T}_j(s'_k, a) + \delta_{\text{RPF}} P_j(s'_k, a)\right)$
        $T_k \leftarrow r_k + \gamma(\mathcal{T}_j(s'_k, a'_k) + \delta_{\text{RPF}} P_j(s'_k, a'_k))$ ▷ Target
        for l in $[1 \dots N]$ **do** ▷ μ and σ^2 for each network of the ensemble
          $\mu_{k,l} \leftarrow \mathcal{T}^\mu_l(s'_k, a'_k) + \delta_{\text{RPF}} P^\mu_l(s'_k, a'_k)$
          $\sigma^2_{k,l} \leftarrow \mathcal{T}^{\sigma^2}_l(s'_k, a'_k)$
        end for
        $\mu_{k,\text{all}} \leftarrow 1/N \sum_{l=1}^N \mu_{k,l}$ ▷ Average of the mean predictions
        $\sigma^2_{\bar{Q},k} \leftarrow 1/N \sum_{l=1}^N \sigma^2_{k,l} + \mu^2_{k,l} - \mu^2_{k,\text{all}}$ ▷ Variance of mixture of Gaussians
      end for
      Estimate ξ based on all $\sigma^2_{\bar{Q},k}$ so that $EBS \geq MEBS$ using Eq. (5.2.1)
      Mask minibatch using M_j
      Train $\mathcal{Q}_j$'s parameters θ_j with $\mathcal{L}_{\text{IVRL}}$ (5.3.7) using $\mu_{\hat{Q}}$, T, ξ and $\sigma^2_{\bar{Q}}$ on D
      Soft update of $\mathcal{T}_j$'s parameters $\bar{\theta}_j \leftarrow (1-\tau)\bar{\theta}_j + \tau\theta_j$
    end for
  end for
end for

B.2.3. Ablations for IV-DQN

In the ablation study presented in section 5.4.1, 4 new algorithms are presented. We detail here their differences with IV-DQN and algorithm 2.

BIV + BootstrapDQN. In BIV + BootstrapDQN, the networks used in the ensembles are single-head networks, similar to BootstrapDQN. Thus, these are sampled ensembles. $\sigma^2_{\bar{Q},k}$ is thus computed as the sampled variance of the networks' predictions. Additionally, instead of $\mathcal{L}_{\text{IVRL}}$, we simply use $\mathcal{L}_{\text{BIV}}$.

Compared to algorithm 2, lines 1 to 17 are the same. Line 18 is removed. Line 21 is changed by:
$\sigma^2_{\bar{Q},k} \leftarrow 1/N \sum_{l=1}^{N} (\mu_{\bar{Q},l} - \mu_{\bar{Q},\text{all}})^2$. Finally, in line 25, $\mathcal{L}_{\text{IVRL}}$ is changed to $\mathcal{L}_{\text{BIV}}$.
L2 + VarEnsembleDQN. In L2 + VarEnsembleDQN, we do not apply BIV weights. The difference with IV-DQN is that we simply change $\mathcal{L}_{\text{BIV}}$ for the $L2$ loss in $\mathcal{L}_{\text{IVRL}}$. The networks are still var-networks trained with the $\mathcal{L}_{\text{LA}}$ part of the $\mathcal{L}_{\text{IVRL}}$ loss, and the rest of the algorithm is the same as algorithm 2. In practice, the computation of $\sigma^2_{\bar{Q},k}$ and ξ are now useless, and lines 16 to 21 as well as line 23 can be removed.
BIV + VarNetworkDQN. In BIV + VarNetworkDQN, we use a single var-network instead of a var-ensemble. Concretely, $N = 1$. This means that at line 21, $\sigma^2_{\bar{Q},k}$ is effectively the variance output by the single target network.

Additionally, masks and RPF do not make sense here, so they are not applied. As the BootstrapDQN exploration cannot apply either, an ϵ-greedy strategy is used instead.
L2 + VarNetworkDQN. L2 + VarNetworkDQN is BIV + VarNetworkDQN, without BIV weights. $\mathcal{L}_{\text{BIV}}$ is replaced by the $L2$ loss in $\mathcal{L}_{\text{IVRL}}$.

B.2.4. IV-SAC

IV-SAC is built upon EnsembleSAC, which is inspired by SUNRISE [**92**]. We describe here the main elements of IV-SAC, and present the detailed process in algorithm 3.
Network architecture. The variance networks used in IV-SAC are multilayer perceptrons with flattened inputs and 2 ReLU-activated hidden layers of 256 nodes. Our implementation is based on rlkit[2].
Critic. SAC is a complex DRL algorithm with several details which are orthogonal to IV-RL, for example, the accounting of policy entropy in the definition of the Q-value. On the critic side, the implementation of IV-RL is mostly similar to IV-DQN's TD update.
Actor. Following the example of [**92**], we use an ensemble for the actor-network when using ensembles for the critic. Each network in the actor is linked to a pair of networks in the critic. When training, the Q-value and its variance for each state-action are computed similarly as when training the critic. As justified in section 5.3.2, we also apply the BIV loss function when training the actor. However, instead of the $L2$ error shown in equation (5.2.1), we use the SAC loss function for the actor. For a minibatch D of size K, we have:

[2]https://github.com/rail-berkeley/rlkit

$$\mathcal{L}_{\text{actor}}(D,\phi) = \left(\sum_{k=0}^{K} \frac{1}{\sigma^2_{\hat{Q}}(s_k,a_k)+\xi}\right)^{-1} \sum_{k=0}^{K} \frac{1}{\sigma^2_{\hat{Q}}(s_k,a_k)+\xi}(\alpha \ln(\pi_\phi(a_k|s_k)) - \mu_{\hat{Q}}(s_k,a_k)) \tag{B.2.2}$$

Action selection. The use of an ensemble for the d allows for different methods to select the next action given a state. In this respect, we follow again the example of [**92**].

During the training of the Q-network, we pair each Q-network with its corresponding actor. To encourage exploration, we additionally take advantage of the ensemble to implement UCB. Each policy network outputs an action. The mean Q-value $\mu_Q(s,a)$ and variance $\sigma^2_Q(s,a)$ for each action a are computed by sampling over all networks of the critic. The chosen action has the highest UCB score: $a^* = \arg\max_a$where Λ_{UCB} is a hyperparameter.

To estimate the standard deviation for UCB, we use sampled ensembles: this is because, for UCB, the variance estimation must be calibrated, in order to be correctly scaled with the Q-values. Var-ensembles are therefore disqualified.

During testing, the agent takes the average action, as sampled from the output of the mean prediction by the policy networks.

B.2.5. Ablations for IV-SAC

We also do ablation studies with our IV-SAC model (presented in section 5.4.2) where we show 6 additional variations. We detail here their differences with IV-SAC with respect to algorithm 3.

EnsembleSAC. EnsembleSAC is the base on which IV-SAC is built. It uses regular networks as critics instead of variance networks in the ensemble, and it does not consider uncertainty. Compared to algorithm 4, lines 1 to 17, and 20, 21, 26 to 29 are identical. Lines 18, 19, 22 and 24 are removed. In line 23 and 25, the usual losses of SAC are used instead of $\mathcal{L}_{\text{actor}}$ and $\mathcal{L}_{\text{BIV}}$.

BIV+EnsembleSAC. Similar to EnsembleSAC, this version uses regular networks as critic instead of variance networks in the ensemble. However, it is uncertainty aware: it estimates the variance of the prediction using sampled ensembles. It is thus similar to SunriseSAC [**92**], but with BIV weights instead of SUNRISE weights. $\sigma^2_{\bar{Q},k}$ and $\sigma^2_{\hat{Q},k}$ being computed as the sampled variance of the networks' predictions, we change these lines compared to algorithm 3:

- line 18 to: $\sigma^2_{\hat{Q},k} \leftarrow 1/N \sum_{l=1}^{N} (\mu_{\hat{Q},l} - 1/N \sum_{m=1}^{N} \mu_{\hat{Q},m})^2$
- line 19 to: $\sigma^2_{\bar{Q},k} \leftarrow 1/N \sum_{l=1}^{N} (\mu_{\bar{Q},l} - 1/N \sum_{m=1}^{N} \mu_{\bar{Q},m})^2$

Additionally, instead of $\mathcal{L}_{\text{IVRL}}$, we use $\mathcal{L}_{\text{BIV}}$ for the critic at line 25.

L2 + VarEnsembleSAC. Here, we do not apply BIV weights to both actor and critic losses. The critic networks are still var-networks trained with the $\mathcal{L}_{\text{LA}}$ part of the $\mathcal{L}_{\text{IVRL}}$ loss, and

Algorithm 3 IV-SAC - Training

Input: LA weight λ, UCB Λ_{UCB}, minimal effective batch size $MEBS$, policy networks π, ensemble of N Q-var-networks $\mathcal{Q}$, ensemble of N target-var-networks $\mathcal{T}$
for each episode **do**
  for each time step **do**
    Collect the action samples: $\mathcal{A} \leftarrow \{a_i \sim \pi_i(a|s) | i \in \{1,...,N\}\}$
    $a \leftarrow \arg\max_{a_i \in \mathcal{A}} [\mu(\mathcal{Q}(s, a_i)) + \Lambda_{\text{UCB}} \sigma(\mathcal{Q}(s, a_i))]$ ▷ UCB exploration
    Collect next-state s' and reward r from the environment by taking action a
    Sample bootstrap mask $M \leftarrow \{m_l \sim \text{Bernoulli}(p_m) | l \in \{1, ...,N\}\}$
    Store tuple (s, a, s', r, M) in buffer $\mathcal{B}$
    $s \leftarrow s'$
  end for
  for each optimization step **do**
    Sample minibatch D from the replay buffer $\mathcal{B}$
    for j in $[1 \dots N]$ **do** ▷ Each member of the ensemble
      for each tuple $(s_k, a_k, s'_k, r_k, M_k)$ in minibatch D **do** ▷ Computing the targets
        Sample next-action: $a'_{k,j} \sim \pi_j(a'|s'_k)$
        Compute Expected Q-value: $\mu_{\hat{Q},k} \leftarrow \mathcal{Q}_j^{\mu}(a_k, s_k)$
        Compute Target: $T_k \leftarrow r_k + \gamma \mathcal{T}_j^{\mu}(s'_k, a'_{k,j})$
        $\sigma^2_{\hat{Q},k} \leftarrow$ Predictive-Variance($\mathcal{Q}$,s_k,$a_{k,j}$) ▷ Algorithm 4
        $\sigma^2_{\bar{Q},k} \leftarrow$ Predictive-Variance($\mathcal{T}$,s'_k,$a'_{k,j}$) ▷ Algorithm 4
      end for
      Mask minibatch D using M_j
      Estimate ξ_a so that $EBS(\sigma^2_{\hat{Q}}) \geq MEBS$ using Eq. (5.2.2)
      Update π_j's parameters ϕ_j with $\mathcal{L}_{\text{actor}}$ (B.2.2) using $\mu_{\hat{Q}}$, π_j, ξ_a and $\sigma^2_{\hat{Q}}$
      Estimate ξ_c so that $EBS(\sigma^2_{\bar{Q}}) \geq MEBS$ using Eq. (5.2.2)
      Update $\mathcal{Q}_j$'s parameters θ_j with $\mathcal{L}_{\text{IVRL}}$ (5.3.7) using $\mu_{\hat{Q}}$, T, ξ_c and $\sigma^2_{\bar{Q}}$ on D
      Soft update of $\mathcal{T}_j$'s parameters $\bar{\theta}_j \leftarrow (1 - \tau)\bar{\theta}_j + \tau\theta_j$
    end for
  end for
end for

Algorithm 4 Variance Estimation for IV-SAC

procedure Predictive-Variance($\mathcal{N}$,s,a)
  Input: Var-networks ensemble: $\mathcal{N}$, State: s, Action: a
  for l in $\{1...N\}$ **do**
    $\mu_l \leftarrow \mathcal{N}_l^{\mu}(s,a)$
    $\sigma_l^2 \leftarrow \mathcal{N}_l^{\sigma^2}(s,a)$
  end for
  $\mu_{\mathcal{N}} \leftarrow 1/N \sum_{l=1}^{N} \mu_l$ ▷ Average of the mean predictions
  $\sigma^2_{\mathcal{N}} \leftarrow 1/N \sum_{l=1}^{N} \sigma_l^2 + \mu_l^2 - \mu^2_{\mathcal{N}}$ ▷ Variance of mixture of Gaussians
  return $\sigma^2_{\mathcal{N}}$
end procedure

the rest of the algorithm is the same as algorithm 3. In practice, the computation of $\sigma^2_{\bar{Q},k}$ and ξ are now useless, and lines 18, 19, 22 and 24 can be removed.
BIV + VarNetworkSAC. We use a single var-network instead of a var-ensemble as the critic. Concretely, $N = 1$. This means that at lines 18 and 19, $\sigma^2_{\hat{Q},k}$ and $\sigma^2_{\bar{Q},k}$ are effectively the variance output by the single-target network.
L2 + VarNetworksAC. VarNetwork is BIV + VarNetwork, without BIV weights in both actor and critic losses. $\mathcal{L}_{\text{BIV}}$ is replaced by the $L2$ loss in $\mathcal{L}_{\text{IVRL}}$.
UWAC + VarEnsemble. Here we change the weighing scheme in IV-SAC to the one proposed in [**171**], keeping everything else the same.
Sunrise + VarEnsemble. For this version we modify the weighing scheme in IV-SAC to the one proposed in [**92**], keeping everything else the same.

B.3. Appendix - Hyperparameter tuning

Deep reinforcement learning algorithms are known to be particularly difficult to evaluate due to the high stochasticity of the learning process and the dependence on hyperparameters. To thoroughly compare different algorithms, it is important to carefully tune the hyperparameters, based on several environments and network initialization seeds [**67**].

For every result presented in this paper, the hyperparameters for each algorithm were tuned using grid search. Each combination of hyperparameters was run five times, with different seeds on both initialization and environment. The best 3 or 4 configurations were then selected to be run 25 times - this time, the combinations of 5 environment and 5 initialization seeds. The configuration selected to be shown in the paper is the best of these configurations based on the 25 runs.

Table 1 describes the type and sets of parameters that were optimized for the relevant algorithms. Note that the learning rate lr, discount rate γ and target new update rate τ are also relevant for SAC. Because there was too many hyperparameters to tune, we kept for all SAC experiments the default values used in *rlkit*[3] implementation : $actorlr = 0.0003$, $criticlr = 0.0003$, $\gamma = 0.99$ and $\tau = 0.005$.

The hyperparameters used for each curve can be found in the configuration file of the code submitted as additional material.

B.4. Appendix - λ in $\mathcal{L}_{\text{IVRL}}$

As expressed in equation (5.3.7), the $\mathcal{L}_{\text{IVRL}}$ loss function we use to train the Q-value estimation is a combination of two components, $\mathcal{L}_{\text{BIV}}$ and $\mathcal{L}_{\text{LA}}$, balanced by hyperparameter λ.

[3]https://github.com/rail-berkeley/rlkit

Table 1. List and set of hyperparameters that were tuned when testing the algorithm.

Hyperparameter	Relevant algorithms	Set
Effective minibatch size	All	$\{32, 64, 128, 256\}$ for DQN+ $\{256, 512, 1024\}$ for SAC+
Learning rate lr	DQN+	$\{0.0005, 0.001, 0.005, 0.01\}$
Discount rate γ	DQN+	$\{0.98, 0.99, 0.995\}$
Target net update rate τ	DQN+	$\{0.001, 0.005, 0.01, 0.05\}$
e-decay	DQN+ with e-greedy exploration	$\{0.97, 0.98, 0.99, 0.991, 0.995\}$
Mask probability	All with masked ensembles	$\{0.5, 0.7, 0.8, 0.9, 1\}$
RPF prior scale	All with RPF	$\{0.1, 1, 10\}$
Loss attenuation weight λ	All with loss attenuation	$\{0.01, 0.1, 0.5, 1, 2, 5\}$
Minimal EBS ratio	All with minimal EBS	$\{16/32, 24/32, 28/32, 30/32, 31/32\}$ for DQN+ $\{0.5, 0.9, 0.95, 0.99\}$ for SAC+
Minimal variance ϵ	All with constant ϵ	$\{0.5, 1, 5, 100, 1000, 10000\}$
UCB weight λ_{UCB}	All SAC+ with UCB (0 means Bootstrap)	$\{0, 1, 10\}$ as suggested in [**92**]
SUNRISE temperature	All with SUNRISE	$\{10, 20, 50\}$ as suggested in [**92**]
UWAC β	All with UWAC	$\{0.8, 1.6, 2.5\}$ as suggested in [**171**]

We use Adam [**78**] optimizer: any constant multiplied to the loss function is absorbed and does not impact the gradient. Therefore, λ effectively controls the ratio between $\mathcal{L}_{\text{BIV}}$ and $\mathcal{L}_{\text{LA}}$.

The sensitivity to the value of λ depends on the scale of the return, the stochasticity of the environment, and the learning dynamics. As can be seen in figure 1, changing the values of λ between 1 and 10 lead to different results in LunarLander, HalfCheetah and Ant, while in SlimHumanoid, Walker and Hopper, the performance stays very similar.

In figure 2, we analyze how the performance of IV-RL methods change when λ is set to extreme values. Very high values of λ lead to sub-optimal results, first because $\mathcal{L}_{\text{BIV}}$ is practically ignored, but also because using variance ensembles alone may lead to a slower or less stable learning curve. Very low values of λ, on the other hand, also bring the performance down, first as $\mathcal{L}_{\text{LA}}$ is ignored, but also as the variance prediction is not trained: the uncertainty estimates provided to $\mathcal{L}_{\text{BIV}}$ are less reliable.

From figure 2, we can see that the effect of λ depends on the environment. For LunarLander, smaller λ leads to slower learning and sub-optimal performance in comparison to $\lambda = 10$. On the other hand, very high λ leads to poor performance and never reach the goal score of 200. For SlimHumanoid, both extremes show inferior performance to when $\lambda = 10$, but, although there are differences in the dynamics, they reach similar a score after 200k steps. In Walker, high λ is also slower, however, the impact of low λ is less important.

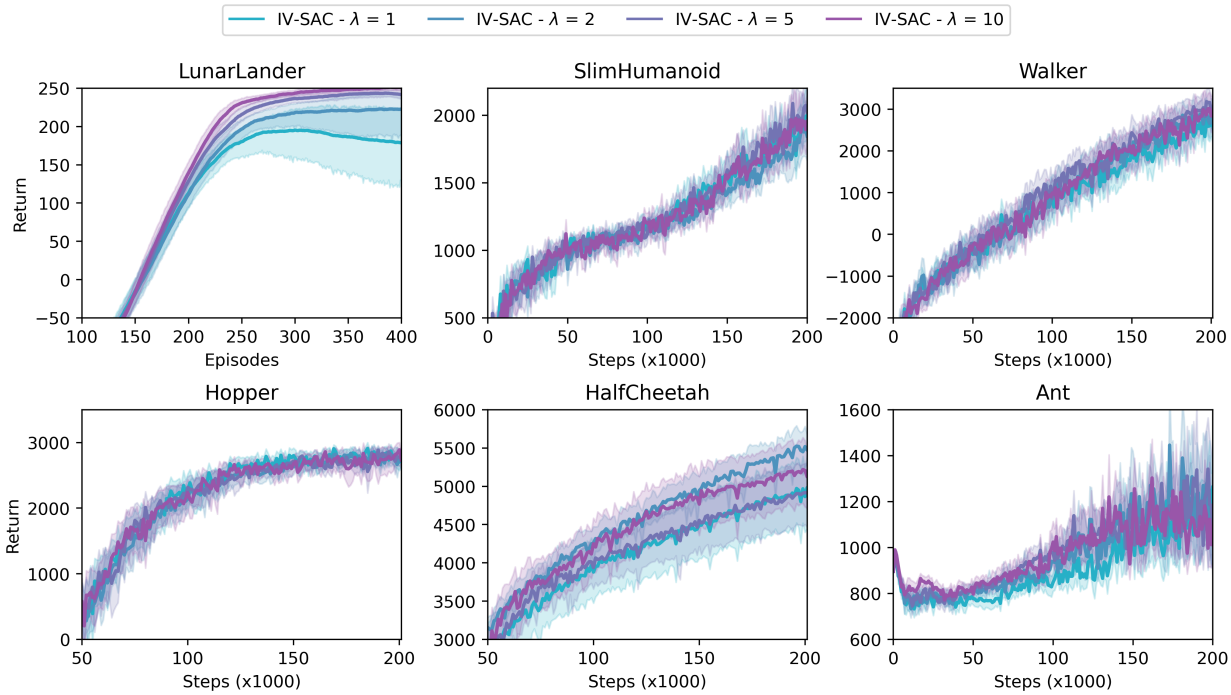


Fig. 1. IV-RL with λ values around the range of 1 to 10.

This can be attributed to the fact that, for Walker, the BIV component contributes more to performance than the LA component of IV-RL loss, as seen in figure 5.

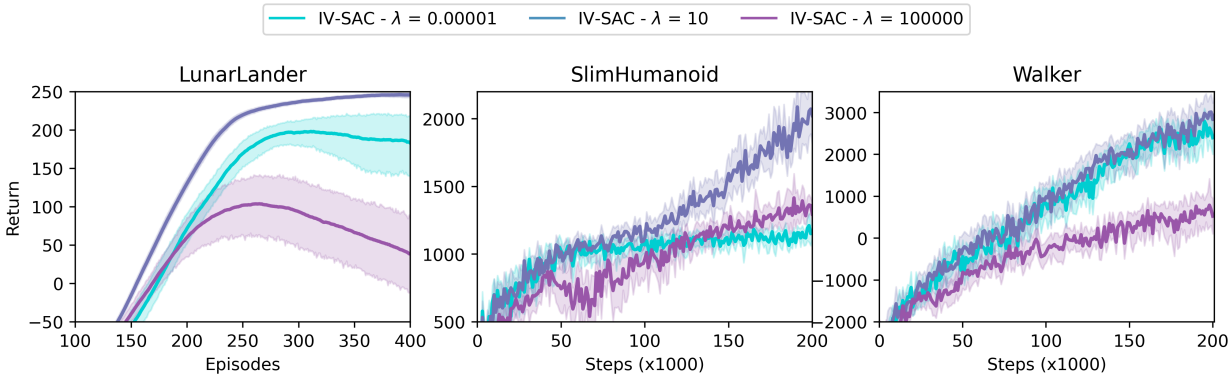


Fig. 2. IV-RL with extreme values of λ

B.5. Appendix - Computation time

As IV-RL uses ensembles, the computation time necessary for the training of the agents is significantly increased compared to single-network algorithms such as SAC or DQN. This can be seen in tables 2 and 3.

The computation of ξ for $MEBS$ as well as additional forward passes to evaluate the variance add some computational load compared to other ensemble-based methods such as BootstrapDQN and SunriseDQN.

While this can limit the applicability of IV-RL, we consider the cases where computation time is secondary to sample efficiency. This is the case when samples are the more expensive element, such as in the real world.

Table 2. Average training time for 1000 steps on walker2d environment.

	SAC	EnsembleSAC	SunriseSAC	IV-SAC (ours)
Runtime/1000 steps (s)	8.34	62.16	62.1	74.56

Table 3. Average training time per episode on LunarLander-v2 environment

	DQN	BootstrapDQN	SunriseDQN	IV-DQN (ours)
Runtime/ episode (s)	0.51	1.5	1.53	1.87

B.6. Appendix - Variance estimation

The difference of performance between IV-RL with sampled ensembles and variance ensembles is sometimes significant. Looking at the variance prediction can allow us to better understand this result.

B.6.1. IV-DQN

In figure 3, we show the mean of the sample variance estimations per mini-batch, depending on the variance estimation method.

Different profiles can be seen in LunarLander and Cartpole-Noise. In the first environment, the scale of the variance predicted by sampled ensembles with BootstrapDQN and variance ensembles does not differ significantly. In Cartpole-Noise however, the variance ensembles quickly capture much more uncertainty. This explains the difference in dynamics that can be seen in figure 3.

Although we do not have results for single variance networks in Cartpole-Noise, we can see in LunarLander why single variance networks are less reliable than variance ensembles. The mean variance they predict is very high, but, as can be seen in figure 4, the median is a lot lower. The mean is thus driven by extremely high variance predictions. Instead, sampled and variance ensembles show similar profiles for mean and median variance. The low median variance confirms that single variance networks fail to capture epistemic uncertainty, but also that they have very high volatility.

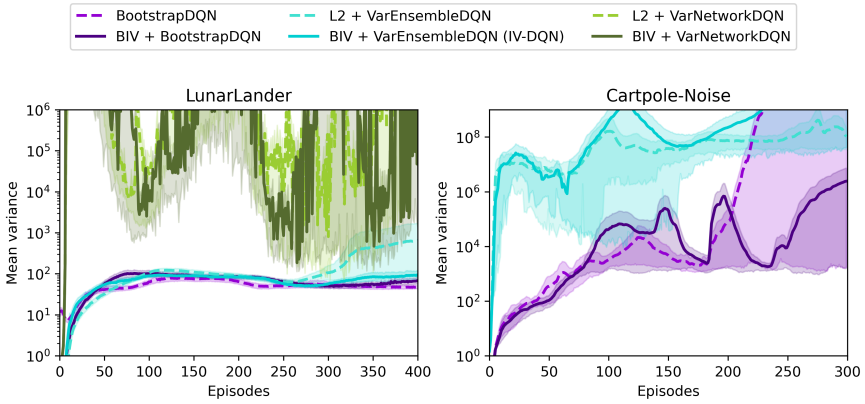


Fig. 3. Mean target variance predictions for discrete environments

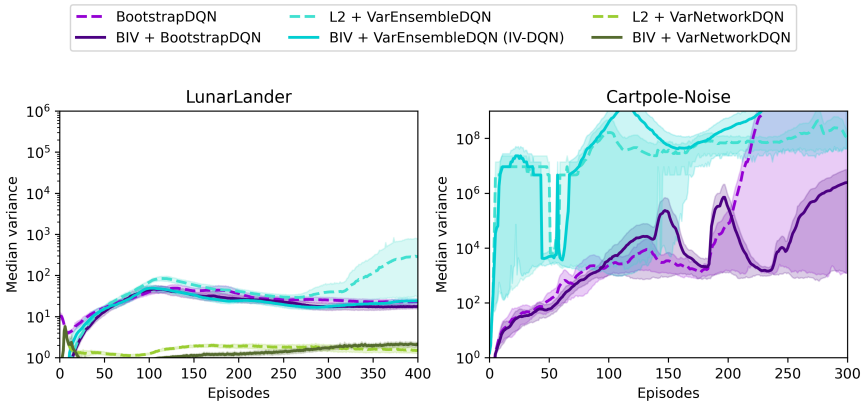


Fig. 4. Median target variance predictions on discrete environments

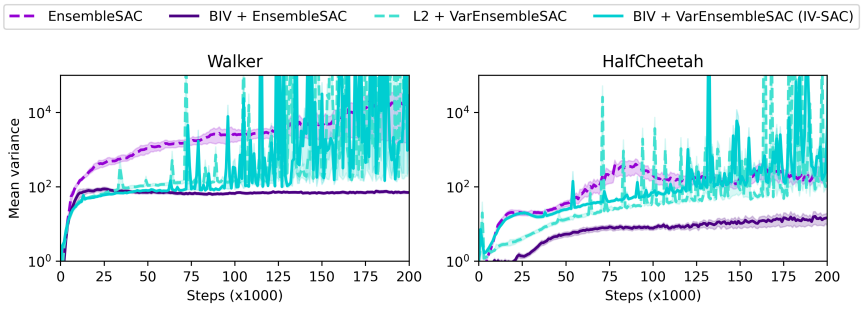


Fig. 5. Mean target variance predictions on continuous environments

B.6.2. IV-SAC

In the continuous control environments, the story is different, as can be seen in figures 5 and 6.

The first thing to notice is the volatility of the mean of the variance produced by variance ensembles. Although the scale is not completely unreasonable and is still close to the median, (on the opposite of variance networks in DQN), it still suffers from some calibration issues.

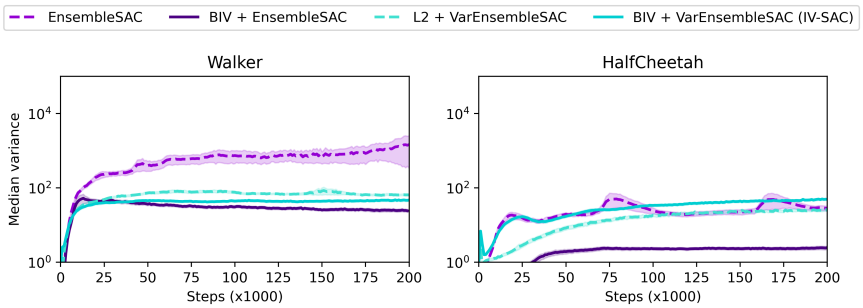


Fig. 6. Median target variance predictions on continuous environments

Second, we see that the epistemic variance in EnsembleSAC is higher than the one in BIV + EnsembleSAC. One hypothesis to explain this is that using IV reduces the impact of very noisy supervision which may confuse the networks and increase the variance in the ensemble. Variance ensembles do not seem to suffer from this.

B.7. Appendix - Weighting schemes

The study on the variance allows a better understanding of the advantage of the BIV weights [**109**] compared to the UWAC weights [**171**] and SUNRISE weights [**92**].

In these works, the weight w_k of a sample k in a minibatch of size K is given by:

$$w_{\text{BIV},k} = \left(\sum_{j=1}^{K} \frac{1}{\sigma_j^2 + \xi}\right)^{-1} \frac{1}{\sigma_k^2 + \xi} \tag{B.7.1}$$

$$w_{\text{UWAC},k} = 1/K \min\left(\frac{\beta}{\sigma_k^2}, 1.5\right) \tag{B.7.2}$$

$$w_{\text{SUNRISE,k}} = 1/K \text{Sigmoid}(-\sigma_k * T) + 0.5 \tag{B.7.3}$$

Figure 7 shows the result of applying such weights to variance ensembles. Note that the first two plots are repeated from figure 5 in section 5.4.2.

It is clear that the SUNRISE weights, which are based on a heuristic, are not optimal compared to the more theoretically grounded UWAC and BIV weights.

However, we can note that UWAC weights perform much better than SUNRISE, but are still somewhat inferior to BIV weights. Our understanding is that this is due to the changes in the scale of the variance estimation along the learning process, as seen in figures 3 and 5.

In UWAC, this first leads to continuous changes in the effective learning rate. For example, multiplying the variances by 2 divides the weights accordingly by 2. While the multiplication of the loss by a constant is captured by optimizers such as Adam [**78**] in the long term, it can still cause short term disturbances in the learning process. In comparison, the normalization of the BIV weights ensures the stability of the loss function.

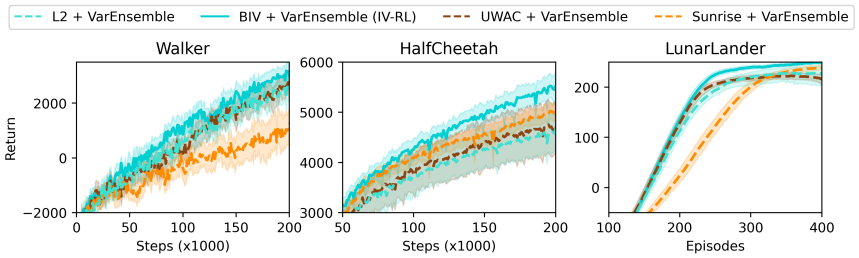


Fig. 7. Comparing BIV, UWAC and SUNRISE weights with var-ensemble

UWAC weights are also hard-clipped by the value 1.5, after being scaled by a constant hyperparameter β. When the scale of the variances changes, this can disturb the control over the discriminatory power of the weights. For example, if all variances are smaller than β, all weights are clipped and become identical at 1.5, and the weighting scheme does not discriminate between certain and uncertain samples. On the other hand, if most variances are very high compared to β, the weights will be very low. A single low-variance sample may thus completely outweigh all other samples, which will get ignored in the gradient update. The discrimination is then too strong. BIV instead uses ξ, which is dynamically computed to maintain a minimal effective batch size $MEBS$ as explained in section 5.2.1. The relative weights of the samples are thus preserved if the variances are multiplied by a constant. As such, the BIV weights maintain a stable discriminatory power in the face of changing variance scales along the learning process, leading to better results.

Note that the curves here present the combination of UWAC and SUNRISE with var-ensembles, which is different from the original works proposed by [**171**] (using MC-Dropout) and [**92**] (using sampled ensembles).

B.8. Appendix - Variance networks

What if, instead of using predictive ensembles, we simply replaced the Q-network by a variance network and trained it using the IV-RL loss?

Figure 8 shows the performances on 4 continuous environments, compared to SAC. We can see a slight improvement, but it is not statistically significant. Based on the insights given in section B.6, we make the hypothesis that it is due to the volatility of the variance estimation with a single variance network.

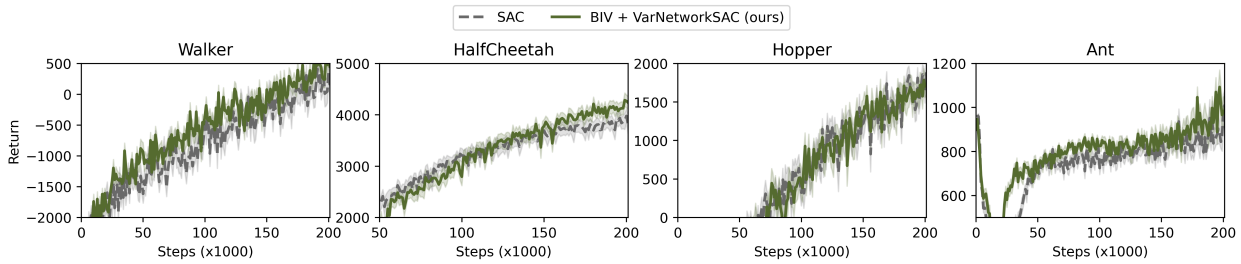


Fig. 8. Single variance networks combined with BIV lead to a very slight improvement.

B.9. Appendix - IV-RL and Optimism in the Face of Uncertainty

A question that arose during this research is how does IV-RL interact with the principle of Optimism in the Face of Uncertainty (OFU), which is often used to drive exploration in reinforcement learning, for example with Upper Confidence Bounds (UCB).

The principle behind IV-RL is to down-weight highly uncertain samples during the TD-update; the idea of OFU is to encourage the agent to choose actions with high uncertainty during exploration. Do these two approach go against each other?

First, we would like to distinguish the uncertainty of the *value* estimation at state-action pair $Q(s,a)$ used by OFU, and the uncertainty of the value estimation of the *target* $T(s,a)$ used by IV-RL. They are separate: we tell the agent to go where the $Q(s,a)$ is uncertain, and then to sample a target $T(s,a)$ by projecting over the next time step with $Q(s',a')$. $T(s,a)$ has its own uncertainty, which is not that of $Q(s,a)$. If the agent believes $T(s,a)$ is highly uncertain, it should not give it too much weight. $T(s,a)$ may also have low uncertainty: the agent will then give it more importance.

However, two factors come and blur this distinction. First, in the context of reinforcement learning where the experience is built with trajectories, the information the agent has received at (s',a') is correlated to the information it received at (s,a). Second, the use of function approximation may cause the uncertainty of $Q(s,a)$ and the uncertainty of $Q(s',a')$ to be similar, especially if (s,a) is close to (s',a') on the state-action space. These correlations may induce interactions between IV-RL and OFU, which may not be completely decoupled.

Another element needs to be considered: in both IV-DQN and IV-SAC, exploration is made following the Q-networks, whereas the value of $Q(s',a')$ to compute the target is estimated by target networks, which are delayed compared to Q-networks. This reduces the correlations between the uncertainties of $Q(s,a)$ and $T(s,a)$.

As an empirical element to this answer, we show in the paper that OFU and IV-RL work together: the IV-SAC experiments include a UCB bonus for exploration. It is to be noted that, if these two strategies can conflict, IV-RL proposes the use of hyperparameter $MEBS$

which allows us to control the intensity of its discrimination. By increasing the $MEBS$, one can give more importance to high-variance samples, and find an optimal balance.

Nevertheless, we believe this question is important for all strategies based on weighting samples according to variance in DRL and could be an interesting line of future investigation.

Appendix C

Multi-gent reinforcement learning for fast-timescale demand response of residential loads: supplementary material

In this appendix, we provide supplementary material for Article 3. We first present a notation table, as well as an estimation of the carbon footprint of running our experiments. We then provide details for the environment and the algorithms. We finish with a discussion of the effect of the number of agents in the environment on the performance measurement.

C.1. Notation

Tables 1 and 2 contain the different notations we use in this paper.

C.2. Carbon emissions of the research project

As a significant amount of electricity has been used to train and run the models for this work, we publish its estimated carbon footprint.

Experiments were conducted using a private infrastructure, which has a carbon efficiency of 0.049 kgCo_2eq/kWh. A cumulative of 10895 days, or 261480 hours, of computation was mainly performed on CPU of type Intel Xeon Processor E5-2683 v4 (TDP of 120W). We assume on average a power usage of half the TDP for CPUs.

The total emissions are estimated to be 628 kgCO_2eq of which 0% were directly offset. This is equivalent to 2550 km driven by an average car, or 314 kg of burned coal.

These estimations were conducted using the MachineLearning Impact calculator [**88**].

Table 1. Notation table - Part 1

Number of agents	N	Number of houses in cluster (general)
	N_{tr}	Number of houses in training environment
	N_{de}	Number of houses in test environment
	$N_{\text{c}_{\text{tr}}}$	Number of agents for communication during training
	$N_{\text{c}_{\text{de}}}$	Number of agents for communication at deployment
Temperatures	T_h	Indoor air temperature
	T_m	Indoor mass temperature
	T_o	Outside temperature
	T_T	Target indoor temperature
Signal and power	s_0	Base signal
	ρ	Power system operator signal
	s	Regulation signal
	D_a	Average power needed by the ACs
	D_o	Power needed by non flexible loads
	δ_s	Signal variation
	δ_p	Perlin noise
	β_p	Variation amplitude parameter
	P	Total cluster power consumption
AC state	ω	Status (ON or OFF)
	l	Time left for lockout
	P_a	Power consumption
	Q_a	Heat removed by the AC
House thermal model	θ_h	House thermal characteristics
	U_h	Outside walls conductance
	C_m	House thermal mass
	C_h	Air thermal mass
	H_m	Mass surface conductance
	θ_s	House lightning characteristics
	Q_s	Solar gain
AC model	θ_a	AC characteristics
	K_a	Cooling capacity
	COP_a	Coefficient of performance
	L_a	Latent cooling fraction
	$l_{\max}$	lockout duration

C.3. Environment details

C.3.1. Detailed house thermal model

The air temperature in each house evolves separately, based on its thermal characteristics θ_h, its current state , the outdoor conditions such as outdoor temperature and solar gain, and the status of the air conditioner in the house. The second-order model is based on Gridlab-D's Residential module user's guide [**71**].

Table 2. Notation table - Part 2

POMDP	$a, \mathcal{A}$	Action, action space
	$o, \mathcal{O}$	Observation, observation space
	$S, \mathcal{S}$	State, state space
	$r, \mathcal{R}$	Reward, reward function
	$\mathcal{P}$	Transition probabilities
	M	Set of communicating agents
	m_j^i	Message from j to i
	$\tilde{o}$	Concatenated observation and messages
	γ	Discount factor
	$\alpha_{\text{temp}}, \alpha_{\text{sig}}$	Weights in the reward function
Algorithms	H	Horizon
	Θ	Transition
	$Q(a,o), T(a,o)$	Q-value prediction (with Q or target network)
	π_θ	Policy parameterized by θ
	V_ϕ	Critic parameterized by ϕ
	G	Return
	A^π	Advantage for policy π

Using Gridlab-D's module, we model an 8×12.5 m, one level rectangular house, with a ceiling height of 2.5 m, 4 1.8-m^2, 2-layer, aluminum windows, and 2 2-m^2 wooden doors, leading to the following values presented in Table 3.

Table 3. Default house thermal parameters θ_h

U_h	2.18×10^2 W/K
C_m	3.45×10^6 J/K
C_h	9.08×10^5 J/K
H_m	2.84×10^3 W/K

To model the evolution of the house's air temperature $T_{h,t}$ and its mass temperature $T_{m,t}$, we assume that this temperature is homogeneous and do not consider the heat propagation in the house. We define the following variables:

$$\begin{aligned} a =& C_m C_h / H_m \\ b =& C_m (U_h + H_m)/H_m + C_h \\ c =& U_h \\ d =& Q_{a,t} + Q_{s,t} + U_h T_{o,t} \\ dT_{A0}/dT =& \left(H_m T_{m,t} - (U_h + H_m) T_{h,t} \right. \\ & \left. + U_h T_{o,t} + Q_{h,t} + Q_{s,t}\right) / C_h. \end{aligned}$$

The following coefficient are then computed:

$$\begin{aligned}
r_1 =&(-b + \sqrt{b^2 - 4ac})/2a \\
r_2 =&(-b - \sqrt{b^2 - 4ac})/2a \\
A_1 =&(r_2 T_{a,t} - dT_{A0}/dT - r_2 d/c)/(r2 - r1) \\
A_2 =&T_{h,t} - d/c - A_1 \\
A_3 =&(r_1 C_h + U_h + H_m)/H_m \\
A_4 =&(r_2 C_h + U_h + H_m)/H_m.
\end{aligned}$$

These coefficients are finally applied to the following dynamic equations:

$$\begin{aligned}
T_{a,t+1} =&A_1 e^{r_1 \delta t} + A_2 e^{r_2 \delta t} + d/c \\
T_{m,t+1} =&A_1 A_3 e^{r_t \delta t} + A_2 A_4 e^{r_2 \delta_t} + d/c.
\end{aligned}$$

C.3.1.1. Solar gain. It is possible to add the solar gain to the simulator. It is computed based on the CIBSE Environmental Design Guide [**35**].

The house's lighting characteristics θ_S, which include the window area and the shading coefficient of 0.67 are needed to model the solar gain, $Q_{s,t}$.

Then, the following assumptions are made:

- The latitude is 30°.
- The solar gain is negligible before 7:30 am and after 5:30 pm at such latitude.
- The windows are distributed evenly around the building, in the 4 orientations.
- All windows are vertical.

This allows us to compute the coefficients of a fourth-degree bivariate polynomial to model the solar gain of the house based on the time of the day and the day of the year.

C.3.2. Detailed air conditioner model

Once again based on the Gridlab-D Residential module user's guide [**71**], we model the air conditioner's power consumption $P_{a,t}$ when turned ON, and the heat retrieved from the air $Q_{a,t}$, based on its characteristics θ_H, such as cooling capacity K_a, coefficient of performance COP_a, and the latent cooling fraction L_a.

COP_a and L_a are considered constant and based on default values of the guide: $COP_a = 2.5$ and $L_a = 0.35$. We have:

$$\begin{aligned}
Q_{a,t} &= -\frac{K_a}{1 + L_a} \\
P_{a,t} &= \frac{K_a}{COP_a}.
\end{aligned}$$

We set K_a to 15 kW, or 50 000 BTU/hr, to be able to control the air temperature even with high outdoor temperatures. This is higher than most house ACs, but allows to have sufficient flexibility even at high outdoor temperatures (a 5kW AC would have to be always ON to keep a 20°C temperature when it is 38°C outside). This choice does not significantly affect our results: with lower outdoor temperatures, the problem is equivalent with lower AC power.

C.3.3. Regulation signal

C.3.3.1. Interpolation for the base signal. As described in Section 7.3.1.3, we estimate $D_{a,t}$ by interpolation. A bang-bang controller is ran without lockout for 5 minutes, and we compute the average power that was consumed. This gives a proxy for the amount of power necessary in a given situation.

A database was created by estimating $D_{a,t}$ for a single house for more than 4 million combinations of the following parameters: the house thermal characteristics θ_h, the differences between its air and mass temperatures $T_{a,t}$ and $T_{m,t}$ and the target temperature T_T, the outdoor temperature $T_{o,t}$, and the AC's cooling capacity K_a. If the solar gain is added to the simulation, the hour of the day and the day of the year are also considered.

When the environment is simulated, every 5 minutes, $D_{a,t}$ is computed by summing the interpolated necessary consumption of every house of the cluster. The interpolation process is linear for most parameters except for the 4 elements of θ_h and for K_a, which are instead using nearest neighbours to reduce the complexity of the operation.

C.3.3.2. Perlin noise. 1-D Perlin noise is used to compute $\delta_{\Pi,t}$, the power generation high-frequency element. Designed for the field of computer image generation, this noise has several interesting properties for our use case.

Perlin noise is most of the time generated by the superposition of several sub-noises called octaves. It is possible to restrict the span of the values that they can take. Thus, it is possible to test the agents in an environment taking into account several frequencies of non-regular noise, but whose values are restricted within realistic limits. Moreover, the average value of the noise can be easily defined and does not deviate, which ensures that for a sufficiently long time horizon, the noise average is 0.

Each octave is characterized by 2 parameters: an amplitude and a frequency ratio. The frequency represents the distance between two random deviations. The amplitude represents the magnitude of the variation. Normally the frequency increases as the amplitude decreases. This way, high-amplitude noise is spread over a wider interval and lower amplitude noise is more frequent and compact.

In our case, we use 5 octaves, with an amplitude ratio of 0.9 between each octave and a frequency proportional to the number of the octave.

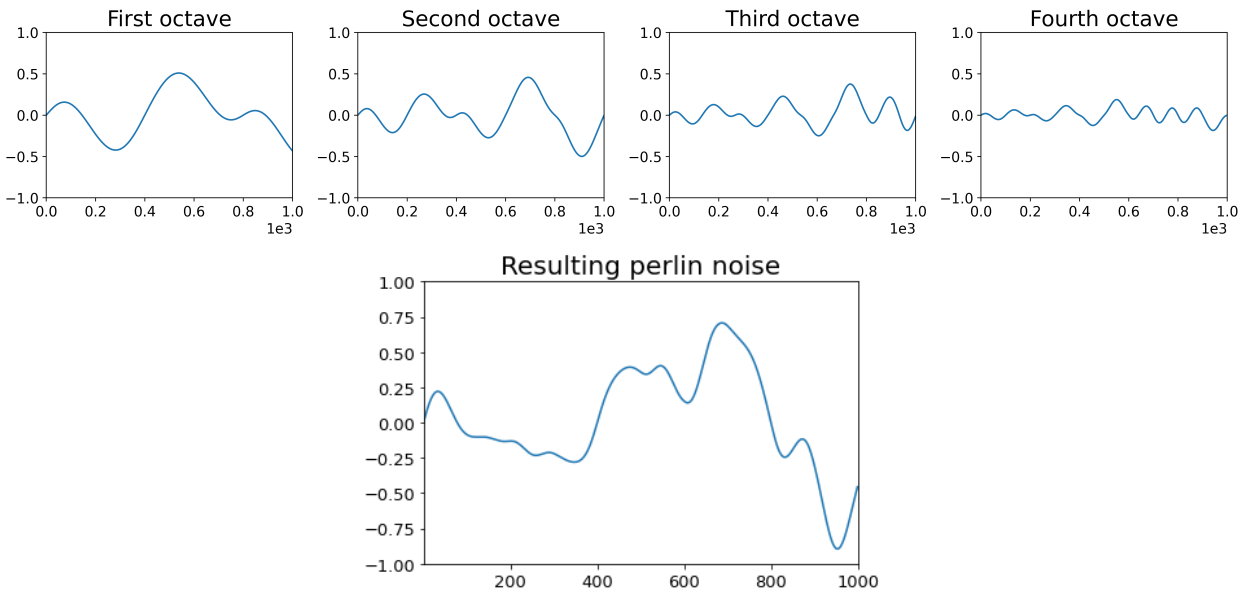


Fig. 1. Illustration of how several octaves add up to form Perlin noise. The frequency of the octaves increases as their amplitude decreases.

C.4. Algorithm details

C.4.1. Model Predictive Control

Our MPC is based on a centralized model. At each time step, information about the state of the agents is used to find the future controls that minimize the reward function over the next H time steps. The optimal immediate action is then communicated to the agents. At each time step, the algorithm calculates the ideal control combination for the H-time step horizon.

The cost function for both the signal and the temperature to minimize being the RMSE, the problem is modeled as a quadratic mixed-integer program. The solver used to solve the MPC is the commercial solver Gurobi [**60**] together with CVXPY [**41**]. Gurobi being a licensed solver, its exact internal behavior is unknown to us and it acts as a black box for our MPC. However, we know that it solves convex integer problems using the branch and bound algorithm. The speed of resolution depends mainly on the quality of the solver's heuristics.

The computation time required for each step of the MPC increases drastically with the number of agents and/or H. To be able to test this approach with enough agents and a rolling horizon allowing to have reasonable performance, it was necessary to increase the time step at which the agents make decisions to 12 seconds (instead of 4 for other agents).

It was impossible to launch an experiment with the MPC agent for 48 hours in a reasonable time. To compensate, we launched in parallel 200 agents having been started at random simulated times. In order to reach quickly the stability of the environment, the noise on the

temperature was reduced to 0.05°C. We then measured the average RMSE over the first 2 hours of simulation for each agent.

Despite this, it was impossible to test the MPC with more than 10 agents while keeping the computation time reasonable enough to be used in real time. That is to say, in a time shorter than the duration between two-time steps.

At each time step, the MPC solves the following optimization problem :

$$\min_{a\in\{0,1\}^{N\times H}} \sum_{t\in H} \alpha_{\text{sig}}(\sum_{i\epsilon N} P_{i,t} - s_0)^2 + \alpha_{\text{temp}} \sum_{i\epsilon N}(T_{h,t,i} - T_{t,t,i})^2,$$

such that it obeys the following physical constraints of the environment:

$$T_{h,t,i}, T_{m,t,i} = F_1(a_{i,t}, T_{h,t-1,i}, T_{m,t-1,i}) \ \forall\ t \in H, i \in N$$
$$P_{h,t,i} = a_{i,t} F_2(\theta_a^i) \ \forall\ t \in H, i \in N,$$

and the lockout constraint:

$$l_{max}(a_{i,t} - \omega_{i,t-1}) - \sum_{k=0}^{l_{max}} (1 - \omega_{i,t-k}) \leq 0 \ \forall\ t \in H, i \in N,$$

where F_1 and F_2 are convex functions that can be deduced from the physical equations given in Section C.3.

C.4.2. Learning-based methods

C.4.2.1. TarMAC and MA-PPO. The original implementation of TarMAC [**39**] is built over the Asynchronous Advantage Actor-Critic (A3C) algorithm [**112**]. The environments on which it is trained have very short episodes, making it possible for the agents to train online over the whole memory as one mini-batch.

This is not possible with our environment where training episodes last around 16000 time steps. As a result, we built TarMAC over our existing MA-PPO implementation. The same loss functions were used to train the actor and the critic.

The critic is given all agents' observation as an input.

The actor's architecture is described in Figure 2. Agent i's observations are passed through a first multi-layer perceptron (MLP), outputting a hidden state x. x is then used to produce a key, a value, and a query by three MLPs. The key and value are sent to the other agents, while agent i receives the other agents' keys and values. The other agents' keys are multiplied using a dot product with agent i's query, and passed through a softmax to produce the attention. Here, a mask is applied to impose the localized communication constraints and ensure agent i only listen to its neighbours. The attention is then used as weights for the values, which are summed together to produce the communication vector for agent i. For multi-round communication, the communication vector and x are concatenated and passed through another MLP to produce a new x, and the communication process is

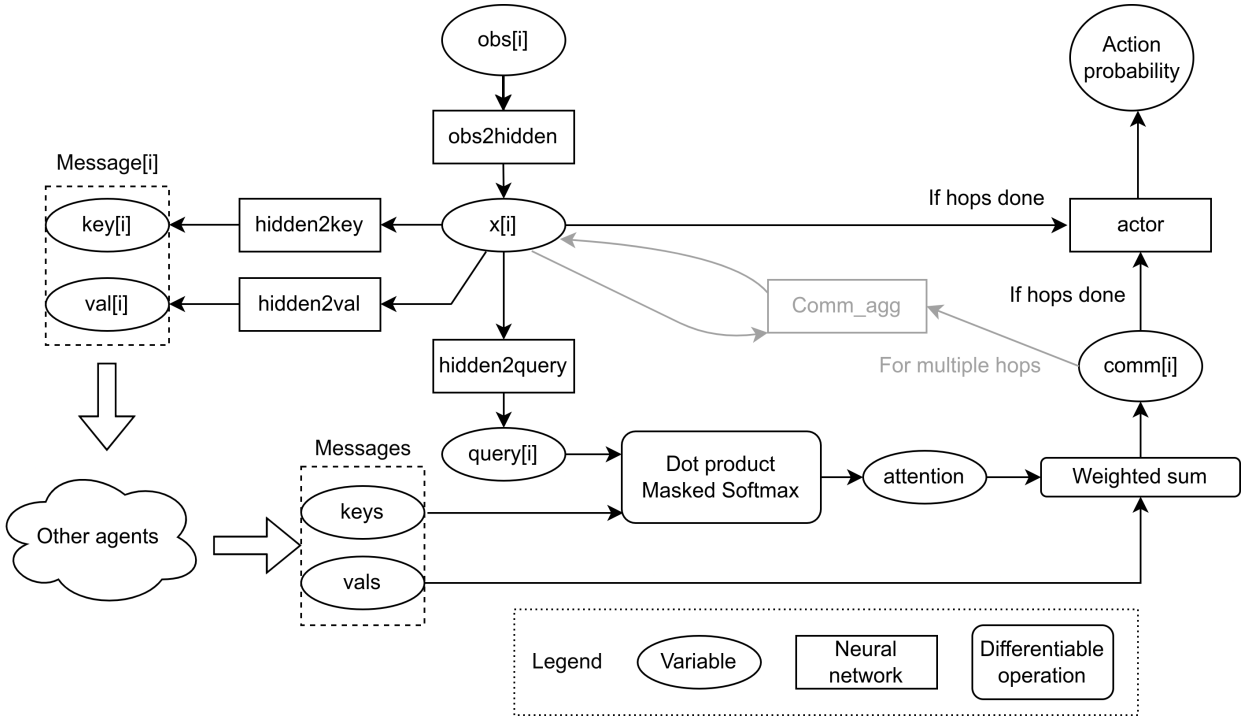


Fig. 2. Architecture of the TarMAC-PPO actor

repeated for the number of communication hops. Once done, the final x and communication vector are once more concatenated and passed through the last MLP, the actor, to produce the action probabilities.

We take advantage of the centralized training approach to connect the agents' communications in the computational graph during training. Once trained, the agents can be deployed in a decentralized way.

C.4.2.2. Neural networks architecture and optimization. For MA-DQN as well as for MA-PPO-HE, every neural network has the same structure, except for the number of inputs and outputs. The networks are composed of 2 hidden layers of 100 neurons, activated with ReLU, and are trained with Adam [**78**].

For TarMAC-PPO, the actor's *obs2hidden*, *hidden2key*, *hidden2val*, *hidden2query* and *actor* MLPs (as shown in Figure 2) all have one hidden layer of size 32. *obs2hidden* and *actor* are activated by ReLU whereas the three communication MLPs are activated by hyperbolic tangent. The hidden state x also has a size of 32.

The centralized critic is an MLP with two hidden layers of size 128 activated with ReLU. The input size is the number of agents multiplied by their observation size, and the output size is the number of agents.

For all networks, the inputs are normalized by constants to facilitate the training. The networks are optimized using Adam.

C.4.2.3. Hyperparameters. We carefully tuned the hyperparameters through grid searches. Table 4 shows the hyperparameters selected for the agents presented in the paper.

Table 4. Training hyperparameters

Hyperparameter	TarMAC-PPO	MA-PPO	DQN
Learning rate	0.001	0.001	0.0001
Mini-batch size	256	512	256
Clip parameter	0.2	0.2	-
Max grad norm	0.5	0.5	-
Number epochs	200	200	-
Number updates	10	10	-
Number episodes	200	200	-
Discount factor γ	0.99	0.99	0.99
Key vector size	4 or 8	-	-
Comm. vector size	8	-	-
Number comm. rounds	1	-	-
Buffer capacity	-	-	65536
ϵ decay	-	-	0.995
Min ϵ	-	-	0.01

C.5. N_{de} and per-agent RMSE

In this section, we discuss the relation between the per-agent signal RMSE of an aggregation of N homogeneous agents if N is multiplied by an integer $k \in \mathbb{N}$.

We consider the aggregation of size kN as the aggregation of k homogeneous groups g_j of N agents which consumes a power $P^j_{g,t} = \sum_i^N P^i_{a,t}$. We have: $P_t = \sum_i^{kN} P^i_{a,t} = \sum_j^k P^j_{g,t}$.

We assume that each group tracks an equal portion of the signal $s^j_t = s_t/k$. We assume that the tracking error $P^j_{g,t} - s^j_t$ follows a 0-mean Gaussian of standard deviation σ_g. This Gaussian error is uncorrelated to the noise of other groups.

It follows from the properties of Gaussian random variables that the aggregation signal error $P_t - s_t$ follows a Gaussian distribution of mean $\mu_k = 0$ and standard deviation $\sigma_k = \sqrt{k}\sigma_g$ for all $k \geq 1$ with $k \in \mathbb{N}$.

Hence the signal's RMSE of a group of kN agents, which is a measured estimation of σ_k, is approximately $\sqrt{k}$ times the RMSE of a group of N agents, which estimates σ_g. Finally, the per-agent RMSE is computed as the group's RMSE divided by the number of agents. We therefore have that the per-agent RMSE of kN agents is approximately $\sqrt{k}/k = 1/\sqrt{k}$ times the RMSE of N agents.

This discussion provides an intuitive explanation for the diminution of the relative RMSE when the number of agents increases. However, it is based on the assumption that the error of each group is not biased, which is not necessarily true with our agents. This explains why the RMSEs are not 10 times lower passing from $N_{\text{de}} = 10$ to $N_{\text{de}} = 1000$.