Université de Montréal

Entity-centric representations in deep learning

par Rim Assouel

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

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Juin, 2020

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Entity-centric representations in deep learning

présenté par:

Rim Assouel

a été évalué par un jury composé des personnes suivantes:

Pierre-Luc Bacon,	président-rapporteur
Yoshua Bengio,	directeur de recherche
Hugo Larochelle,	membre du jury

Mémoire accepté le: 28/08/2020

Résumé

L'incroyable capacité des humains à modéliser la complexité du monde physique est rendue possible par la décomposition qu'ils en font en un ensemble d'entités et de règles simples. De nombreux travaux en sciences cognitives montre que la perception humaine et sa capacité à raisonner est essentiellement centrée sur la notion d'objet. Motivés par cette observation, de récents travaux se sont intéressés aux différentes approches d'apprentissage de représentations centrées sur des entités et comment ces représentations peuvent être utilisées pour résoudre plus facilement des tâches sous-jacentes.

Dans la première contribution on montre comment une architecture centrée sur la notion d'entité va permettre d'extraire des entités visuelles interpretables et d'apprendre un modèle du monde plus robuste aux différentes configurations d'objets.

Dans la deuxième contribution on s'intéresse à un modèle de génération de graphes dont l'architecture est également centrée sur la notion d'entités et comment cette architecture rend plus facile l'apprentissage d'une génération conditionelle à certaines propriétés du graphe. On s'intéresse plus particulièrement aux applications en découverte de médicaments. Dans cette tâche, on souhaite optimiser certaines propriétés physico-chmiques du graphe d'une molécule qui a été efficace in-vitro et dont on veut faire un médicament.

Mots-Clés: apprentissage profond, apprentissage non supervisé, apprentissage de représentations, représentations d'objets, représentations de graphes, découverte de médicaments

Summary

Humans' incredible capacity to model the complexity of the physical world is possible because they cast this complexity as the composition of simpler entities and rules to process them. Extensive work in cognitive science indeed shows that human perception and reasoning ability is structured around objects. Motivated by this observation, a growing number of recent work focused on entity-centric approaches to learning representation and their potential to facilitate downstream tasks.

In the first contribution, we show how an entity-centric approach to learning a transition model allows us to extract meaningful visual entities and to learn transition rules that achieve better compositional generalization.

In the second contribution, we show how an entity-centric approach to generating graphs allows us to design a model for conditional graph generation that permits direct optimisation of the graph properties. We investigate the performance of our model in a prototype-based molecular graph generation task. In this task, called lead optimization in drug discovery, we wish to adjust a few physico-chemical properties of a molecule that has proven efficient in vitro in order to make a drug out of it.

Keywords: representation learning, unsupervised learning, deep learning, entity-centric representations, objects, graphs generation, graph neural networks, conditional generation, drug discovery.

Table des matières

Table des figures

Liste des tableaux

List of Abbreviations

ML Apprentissage Machine de l'anglais *Machine Learning*
MLE Estimation du Maximum de Vraisemblance de l'anglais *Maximum Likelihood Estimation*
MI Information Mutuelle de l'anglais *Mutual Information*
MLP Perceptron Multicouche de langlais *Multi Layer Perceptron*
RL Apprentissage par renforcement de l'anglais *Reinforcement Learning*
DRL Apprentissage par renforcement profond de l'anglais *Deep Reinforcement Learning*
VAE Auto-encodeur variationnel de l'anglais *Variational Autoencoder*
MSE Erreur quadratique moyenne, de l'anglais *Mean Square Error*
GAN Reseaux de Neuronnes adversariaux generatif de l'anglais *Generative Adversarial Networks*
GNN Reseaux de Neuronnes pour Graphs de l'anglais *Graph Neural Networks*
GMM Modèle à Mixture de Gaussiennes de l'anglais *Gaussian Mixture Model*

Acknowledgments

First and foremost, I would like to thank my supervisor Yoshua Bengio without whom I would not have considered joining this amazing research journey. Thank you for giving me the ideal amount of research freedom that allowed me to eventually find my own path and research interests. I am truly grateful to have such an inspiring and understanding person to guide me through this adventure. With you I made it through this first milestone and this is all just starting !

I would also like to thank all the amazing friends I made in Montreal who've been unconditionally supportive: Ahmed, Amal, Anne-Marie, Arthur, Vincent, Carla, Julie, Maxime, Salem, Gauthier, Adrien, Victor, Zhor, Nazia ... to name a few. Furthermore, I am very grateful for the many members of Mila, who I crossed paths with in my time pursuing a master's. I just love the vibe of the lab and I truly found a new home abroad thanks to all of you. It all started with Salem agreeing (did he have the choice ?) to be my buddy and answer the hundred questions I had a day, Ahmed and that couscous party, Gauthier and his shared taste for chocolate, Adrien always free for a chill Friday afternoon at the lab, Tristan that I still call Yann junior, and re-inventing objects with Evan. Those tea talks also kept me pretty busy the whole time.

To Benjamin and Prof. Satoh, thank you for taking a chance on me and giving the opportunity to discover Tokyo, research and the field of machine learning as part of my NII experience. To Pierre, Simon, Gilles and all the amazing Owkin people I met back in Paris without whom I wouldn't have even applied to Mila.

To Marwin, Mo and Amir, this amazing Benevolent.AI trio ! I know I was a stubborn collaborator from time to time but working with you on DEFactor was real fun. I not only learned a lot on drug discovery and graphs in general but it made me realize the importance of good and frequent communication in research.

Last but not least, a huge thank to my mom, Khadija, my brother, Amine, and my sister, Nour, for believing in me more than I do !

1 Introduction

1.1 Background

1.1.1 Machine Learning

The desire to understand human cognition has generated a variety of scientific disciplines. Cognitive science, neuroscience, and the study of machine learning and artificial intelligence (AI) are the most popular examples. The work in this thesis is situated in the field of machine learning, which is one of the most pursued branches in AI research. Machine learning deals with the question of buildingg systems and designing algorithms that learn from data and experience which is in contrast to the traditional approach in computer science where systems are explicitly programmed to follow a sequence of instructions.

More particularly in machine learning we seek to learn algorithms that are necessary to solve one or several tasks. In traditional computer science a programmer would specify the algorithm (the set of instructions) for the machine to satisfy the desired task whereas machine learning would shift this algorithm creation process to the computer based on some input data. Understanding this thesis will require basic knowledge of machine learning, and more specifically deep learning. For a full introduction to machine learning and deep learning, we encourage the reader to check out Bishop [2006], Murphy [2012], Goodfellow et al. [2016]

1.1.2 Unsupervised vs Supervised Learning

Machine learning algorithms are traditionally divided into two main learning paradigms: **supervised learning** and **unsupervised learning** methods. Supervised learning usually involves learning from a dataset of inputs x and their associated *label* y provided by humans. The goal of supervised learning is then to learn a mapping f linking x to y. Most of the existing algorithms will do so by trying to model the distribution $p(y|x)$ with $x, y \sim (X, Y)$ [Goodfellow et al., 2016].

On the hand, unsupervised learning designates learning paradigms that only make use of unlabeled input data $x \sim X$. The objective can vary from one task to another but it often involves estimating or sampling from the input distribution $P(X)$. Unsupervised learning can also be used as an intermediary to learn a *good* representation of the data which is of particular interest in this thesis. Four common goals of unsupervised learning are [Khemakhem et al., 2019]:

— **Density Estimation**
  Modelling the data distribution, $p_{data}(x)$, is often concerned with fitting a model $p_{model}(x)$ to estimate the data density.

— **Sampling**
  Sampling involves learning a model that allows you to perform approximate sampling from $p_{data}(x)$. This can be accomplished using density estimation if you learn a model you can sample from. Moreover, one could directly learn a generator function made specifically for sampling [Goodfellow et al., 2014a].

— **Underlying Structure of the Data**
  In this case, we are interested in revealing underlying structure of the data. We can often imagine the data is created by some unknown generative process that takes a few underlying high level concepts and combines them to form the raw data. In this case we are interested in discovering these hidden concepts that are part of this data generating process.

— **Downstream Task performance**
  In this case, we aim to learn transformations of the data, which we call representations, that are amenable to future prediction tasks. This is commonly called *representation learning*.

These four goals are interrelated, and not mutually exclusive. In this thesis we will focus on a certain type of representations that should help the downstream task performance when the compositional aspect is of major importance.

1.1.3 Representation Learning

The problem of learning is commonly approached by fitting a model to data with the goal that this learned model will generalize to new data or experiences. Traditionally, many machine learning algorithms are built on top of some features, extracted using a pre-defined procedure from the raw data format. The process of developing sophisticated feature extractors is often referred to as feature engineering. On the other hand deep learning (DL), addresses the learning problem by jointly learning representations of the raw input data and a predictive model for the task at hand. This is usually achieved by stacking multiple layers of differentiable non-linear transformations and by training such a model in an end-to-end fashion using gradient descent approaches.

Performance of machine learning algorithms is often dependent on the data representations they use for their input. A representation is traditionally [Goodfellow et al., 2016] defined as a transformation of the input data into another space, usually of a lower dimension before it is used by the machine learning algorithm. The field of representation learning designates all the methods used to learn these transformations from raw input data instead of having to specify a function to extract hand-crafted features [Lowe, 2004, Horn and Schunck, 1981]. As such representation learning has been defined as representing data in a way that will facilitate some downstream tasks [Bengio et al., 2013]. Recently, representation learning has become synonymous with unsupervised representation where we want to learn a representation useful for many tasks without knowing the task of interest beforehand.

1.1.4 What is a good representation ?

We want a representation which will by definition facilitate solving of future supervised downstream tasks that will use this representation as input. However, it is not straightforward to know a priori what the desirable features must be in order to result in high downstream performance. Nevertheless, extensive work in cognitive science shows that human perception and reasoning abilities are structured around objects [van Steenkiste et al., 2019]. Following this observation a recent line of work [Greff et al., 2017, van Steenkiste et al., 2018, Eslami et al., 2016, Kosiorek et al., 2018, Greff et al., 2019, Burgess et al., 2019] has focused on learning entity-centric representations of the input in an unsupervised way in order to reuse them in downstream tasks where the notion of entity is central. The scope of this thesis is centered around this kind of representations. In the next sections we will first motivate the two contributions and then describe a bit more the related work and background necessary to understand each of the two contributions.

1.2 Motivation

1.2.1 Objects and Representation Learning

The broad field of representation learning designates all the methods used to extract features from raw data that will be useful for one or several other downstream tasks [Bengio et al., 2012]. Those shared representations are thus crucial because they facilitate the transfer of learned knowledge from one task to another for which only a handful of examples are available. Specifically, a good representation is a representation that will make the learning of a downstream task easier.

Model-based RL [Chiappa et al., 2017, Sutton, 1991] is a good example where a learned representation of the world can be reused in order to solve different tasks of the same environment. However, model-based algorithms have to make accurate predictions about future states which can be very hard when dealing with high dimensional inputs such as images. On the other hand, extensive work in cognitive science shows that human perception and reasoning abilities are structured in terms of objects. Following this observation, a line of work has focused on unsupervised learning of object-centric representations from raw images with the hope that they can be reused in a modular way to solve many downstream reasoning tasks.

Specifically, Greff et al. [2017], Eslami et al. [2016], van Steenkiste et al. [2018], Kosiorek et al. [2018], Burgess et al. [2019], Greff et al. [2019] have focused on unsupervised ways to decompose a raw visual scene in terms of objects. They rely on a latent representation of the visual scene where the latent space is structured as a set of vectors. Each vector of the set is supposed to represent an "object" (which we refer to as an "entity") of the scene. These approaches can be categorized into two types of models: scene-mixture models and spatial-attention models. In scene-mixture models [Greff et al., 2017, Burgess et al., 2019, Greff et al., 2019], a visual scene is the result of a finite mixture of component images. They have the advantage of providing arbitrary complex segmentation maps of the objects that constitute the visual scene. As a result, important features such as scale and positions of the objects are only implicitly encoded. In contrast, spatial-attention models [Eslami et al., 2016, Kosiorek et al., 2018] propose to disentangle the ”where” and the ”what” in each object representation. Further improvements of the initial Attend-Infer-Repeat [Eslami et al., 2016] model have been then suggested to handle sequential data [Kosiorek et al., 2018] and improve their computational cost [Crawford and Pineau, 2019, Jiang et al., 2019].

1.2.2 First Contribution : SPECTRA

However, most of the recent contributions learn those slot-structured representations in a static way. This means that they do not use any information about the dynamics of the environment. It is however not clear how to disambiguate two adjacent objects without any temporal cues about their respective evolution, especially if they have the same color/appearance. Greff et al. [2019] exhibit the fact that many segmentation maps are correct for a single image and one mixture will be better than an other one depending on the goal we wish to achieve. Watters et al. [2019] study these representations in an RL context considering a visually simple Spriteworld environment. They introduce a method to learn a transition model that is applied to all the slots of their latent scene representation. Extending their work, the first contribution posits that slot-wise transformations should be sparsely applied and that the perception module should be learned jointly with the transition model in order to exhibit useful entity-centric representations. Veerapaneni et al. [2019] also later advocate for a joint training of the perception module and the transition model.

1.2.3 Graphs and Deep Learning

Those slot-structured representations are in fact an instance of a broader body of structured representations: graphs. Here, nodes of the graph correspond to visual entities, and edges are not specified in the static representation but could be added to represent relations between visual entities. Graphs and graph neural networks [Zambaldi et al., 2018] have proven particularly useful to perform structured reasoning when dealing with visual input. Another field where graphs are of crucial importance is one of drug-discovery. Specifically, generating novel molecules with optimal properties is an ongoing and unsolved challenge. Recent deep generative models [Olivecrona et al., 2017, Kusner et al., 2017, Jin et al., 2018, Gómez-

Bombarelli et al., 2016, Li et al., 2018a, You et al., 2018b] of graphs have shown promising ways of performing de-novo molecular design, and recent approaches have investigated ways to generate and optimize molecular graphs more efficiently.

In particular, sequential methods [Li et al., 2018a,b, You et al., 2018b] to graph generation aim to construct a graph by predicting a sequence of addition/edition actions of nodes/edges. Starting from a sub-graph (normally empty), at each time step a discrete transition is predicted and the sub-graph is updated. Because each step is a discrete and non-differentiable transformation of the current sub-graph, in order to optimize some properties of the molecular graph one needs to resort to RL-based optimization techniques but these have proven to suffer from high variance in gradients estimation.

1.2.4 Second Contribution : DEFactor

The main challenge here stems from the discrete nature of graph representations for molecules. This prevents us from using global discriminators that assess generated samples and backpropagate their gradients to guide the optimisation of a generator. This becomes a bigger hindrance if we want to either optimise a property of a molecule (graph) or explore the vicinity of an input molecule (prototype) for conditional optimal generation, an approach that has proven successful in controlled image generation [Mirza and Osindero, 2014]. The second contribution suggests a new framework for conditional graph generation that leverages an entity-centric approach to graph generation. Our approach biases the model towards learning a representation of each node (atom) of the graph (molecule) that contains enough information about the node itself and its neighbours such that simple learned similarity metrics can compare pairs of entities and retrieve the adjacency structure of the graph.

1.3 Visual Entity-centric Representations

1.3.1 Generative Modeling

The fields of learning representations and generative modeling are tied together because representations are often learned by performing posterior inference for a given generative model. The goal of representation learning can be described as learning a representation $z \in \mathbf{Z}$ which summarizes important information contained in some (high-dimensional) input $x \in \mathbf{X}$. The usual desiderata for good representations are that they have to be successful in solving downstream tasks (classification, RL, etc ..). Another desirable property of representation is their interpretability: to that extent, recent work focused on both the disentangling and the compositional aspect of learned representations. In the first contribution we are interested in the latter to model and explain a visual input as the composition of its constitutive entities. Namely, we are interested in learning visual entity-centric representations.

1.3.2 Variational Inference and Learning

Most of the related work we are interested in learn representation as part of a variational inference and learning framework that we introduce in this section. Let $\mathbf{x}$ be a set of observed variables, and $\mathbf{z}$ a set of latent variables and let $p(\mathbf{x}, \mathbf{z})$ be their joint distribution. Given a set of observation $x_1, x_2, ...x_N \in \mathbf{X}$ we want to maximize the marginal likelihood of the parameters i.e to maximize :

$$\log(p(\mathbf{x})) = \sum_{i=1}^{N} \log p(x_i) = \sum_{i=1}^{N} \log \int p(x_i, z) dz \tag{1.1}$$

The marginalization over the latent variable z makes this computation intractable in the general case where z is continuous. The idea of variational inference is to mitigate this intractability by maximizing a lower bound on this log-likelihood instead. A Variational Autoencoder (VAE) learns a latent variable model by maximizing an approximate lower bound on the marginal log-likelihood, $\log p(x) = \log \int p(x, z)dz$. The idea behind the lower bound derivation, called the evidence lower bound (ELBO), is to approximate the posterior $p(z|x)$ with a parametric model $q_\psi(z|x)$ such that :

$$\mathcal{L}_{ELBO} = \mathbb{E}_{z \sim q_\psi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\psi(z|x) \mid\mid p(z)) \leq \log p(x) \tag{1.2}$$

Where $p(z)$ is the prior distribution, often picked to be $\mathcal{N}(0, I)$ an isotropic Gaussian distribution. We parametrize both $q_\psi(z|x)$, which we call the encoder, and $p_\theta(x|z)$, called the decoder, with neural networks. Using the reparameterization trick, both models can be trained end-to-end to maximize this lower bound. When used to model a distribution over images, a VAE first encodes a sample x resulting in the mean and variance parameters of the posterior distribution over the latent variable. The latent variable z is sampled using the reparameterization trick. This latent variable z is then transformed by the decoder to obtain $\hat{x}$, a reconstruction of the input x. The negative ELBO, used as a loss, is then computed and both the encoder and decoder are updated end-to-end to minimize this loss like in any neural network.

Many of the models we are interested in use this variational inference and learning framework to learn good representations of visual inputs.

1.3.3 Slot-based Representations

In the scope of this thesis we are particularly interested in inductive biases for entities representations to emerge. van Steenkiste et al. [2019] ask the question of the requirements such representations should have. In order for entities to serve as primitives of compositional reasoning they posit they should be :

— **Universal** : Each entity representation should be able to represent any object regardless of position, class or other properties. It should facilitate generalization, even to unseen objects, which in practice means that its representation should be distributed and disentangled.

— **Multi-object** : It should be possible to represent multiple objects simultaneously, such that they can be related and composed but also transformed individually.

— **Common Format** : All objects should be represented in the same format, i.e. in terms of the same features. This makes representations comparable, provides a unified interface for compositional reasoning and allows the transfer of knowledge between objects.

Flat vector representations as used by standard VAEs are inadequate for meeting this requirements and for capturing the combinatorial object structure that many datasets exhibit. Let us consider an image composed of 3 coloured objects, each with its own properties such as shape, size, position, color and material. To split objects, a flat representation would have to represent each object using separate feature dimensions. But this neglects the simple and (to us) trivial fact that they are interchangeable objects with common properties. To achieve the kind of combinatorial generalization that is so natural for humans, van Steenkiste et al. [2019] argue that we should use a multi-slot representation where each slot shares a common representation format, and each would ideally describe an independent part of the input.

1.3.4 Scene-Mixture Models

A recent line of work has focused on slot-based architectural biases for visual entity-centric representations to emerge. Among them, we are particularly interested in scene-mixture models [Greff et al., 2017, Burgess et al., 2019, Greff et al., 2019, van Steenkiste et al., 2018] which model a visual scene with spatial Gaussian mixtures models. In these models, an input image $\mathbf{x} \in \mathbb{R}^D$ is represented by a set of $\mathbf{K}$ latent entities (slots) $\mathbf{z} \in \mathbb{R}^{K \times p}$ where each slot $\mathbf{z}_k \in \mathbb{R}^p$ is represented in the same way and is supposed to capture properties of one *entity* k of the visual scene. Each slot $\mathbf{z}_k$ is then decoded by the same decoder f_{dec} into a pixel-wise mean μ_{ik} and a pixel-wise assignment m_{ik} (non-negative and summing to 1 over k). Assuming that the pixels i are independent conditioned on $\mathbf{s}$, the conditional likelihood thus becomes:

$$p_\theta(\mathbf{x}|\mathbf{s}) = \prod_{i=1}^{D} \sum_k m_{ik} \mathcal{N}(\mathbf{x_i}; \mu_{ik}, \sigma^2) \text{ with } \mu_{ik}, m_{ik} = f_{dec}(z_k)_i.$$

Greff et al. [2016] first introduced this way of representing a visual scene and it has been recently further extended with an expectation maximization (EM) [Greff et al., 2017], a VAE [Burgess et al., 2019] and an iterative variational inference [Greff et al., 2019] approach.

Both Greff et al. [2017],Burgess et al. [2019], Greff et al. [2019] decode the latent slots with the same spatial mixture approach described above but they differ in the way they extract the slots representations from the visual input.

NEM - Neural Expectation Maximization The goal of NEM is to group pixels in the input that belong to the same object and capture this information efficiently in a distributed representation θ_k for each object. Each image $x \in \mathbb{R}^D$ is modeled as a spatial mixture of K components parametrized by $\theta_1, ..., \theta_K$. A neural network is used to transform these representations into parameters for the pixel-wise distributions :

$$\psi_{i,k} = f_\phi(\theta_k)_i$$

A set of binary variables encodes the unknown pixel true assignment s.t :

$$z_{i,k} = 1 \text{ iff pixel } i \text{ was generated by } k$$

The full likelihood of $\mathbf{x}$ given θ is :

$$p(\mathbf{x}|\theta) = \prod_i \sum_{z_i} p(x_i, z_i|\psi_i) = \prod_i \sum_k p(z_{i,k} = 1) p(x_i|z_{i,k} = 1, \psi_{i,k})$$

But as the marginalization over z complicates the process Greff et al. [2017] are instead interested in the generalized EM on the following lower bound of the full log-likelihood :

$$\mathcal{Q}(\theta, \theta^{\text{old}}) = \sum_z p(\mathbf{z}|\mathbf{x}, \psi^{\text{old}}) \log p(\mathbf{x}, \mathbf{z}|\psi)$$

Each iteration consists of 2 steps :

— **E-step** : computes $\gamma_{i,k} = p(z_{i,k} = 1|x_i, \psi_i^{\text{old}})$ which yields a new soft-assignment of the pixels to the components (clusters), based on how accurately they model x

— **M-step** : updates θ^{old} by taking a gradient ascent step on $\mathcal{Q}$ using the previously computed soft-assignments.

MONet - Multi-object Network With MONet Burgess et al. [2019] propose to amortize the inference step with a VAE approach. They use an attention module that will attend specific parts of the image : this module is sequential and outputs at each time step a mask such that all the input is explained by all the steps. Each mask is then fed to a component VAE, along with the input image. The mask indicates which part of the image the VAE should focus on representing via its posterior $q_\phi(z_k|x, m_k)$. The VAE is additionally required to model the attention masks over the K components.

In order for the MONet to be able to model scenes over a variable number of slots, they used a recurrent attention network α_ϕ for the masks decomposition process. A scope s_k indicates at each time step the proportion of each pixel that remains to be explained given all previous attention masks, where the scope for the next step is given by :

$$s_{k+1} = s_k(1 - \alpha_\phi(x, s_k))$$

The attention mask for step k is given by :

$$m_k = s_{k-1}\alpha_\phi(x, s_{k-1})$$

IODINE - Iterative Object Decomposition Inference Network Greff et al. [2019] argue that the standard feed-forward VAE inference approach is ill-suited for slot-based representation learning because we need to infer both the components and the mixing weights of the scene-mixture model and this is traditionally tackled as an iterative procedure. They consider Marino et al. [2018] powerful iterative amortized variational approach and adapt it to slot-based representation learning. The idea is to start with an arbitrarly guess for the posterior parameters θ_k and then iteratively refine them using the input, samples from the current posterior estimate as well as other easily computable auxiliary inputs (gradients wrt estimates, parameters, masks ...). The refinement network is parametrised with an LSTM. In principle it is enough to minimize the final negative ELBO $\mathcal{L}^T$ but they found it beneficial to use a weighted sum that includes earlier terms (and corresponding to the refinement steps t) :

$$\mathcal{L} = \sum_{t=1}^{T} \frac{t}{T} \mathcal{L}^{(t)}$$

where

$$\mathcal{L}^{(t)} = D_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p(z)) - \log \sum_k m_k^{(t)} \mathcal{N}(\mathbf{x}; \mu_k^{(t)}, \sigma^2)$$

All these spatial-mixture methods have the advantage of providing arbitrary complex segmentation maps of the objects that constitute the visual scene instead of fixed bounding boxes. As a result, important features such as scale and positions of the objects are only implicitly encoded. However, they only study perceptual groupings in static images and we argue, in the second contribution, that temporal cues are important to extract meaningful entities that can further be used in RL downstream tasks. Veerapaneni et al. [2019] validate this intuition and design an entity-centric dynamic latent variable framework for model-based RL emphasizing the fact that dynamics are important to disambiguate objects in a visual scene.

1.4 Molecular Graphs and Deep Learning

1.4.1 Molecular Graph

A graph is a powerful representation of relations between groups of entities. We are particularly interested in the way graphs are used to represent chemical compounds composed of atoms (nodes) linked together with typed chemical bonds (edges). Formally, a graph is an ordered pair $G = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V}$ is a non-empty set of vertices (also called nodes) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Additional information can be attached to both vertices and edges in the form of categories (e.g. atom and bond type for molecules).

1.4.2 Molecular Graphs Generation

Deep learning-based generative models have gained massive popularity recently and particularly in the field of images and text generation. The main idea behind most approaches is to collect an important number of unlabeled data from one domain and train a model to generate similar data points. Usually, the generative process (also called decoding) is conditioned on a random vector drawn from a simpler known prior distribution [Goodfellow et al., 2014b] and/or a point from a defined vector space that can encode other learned and pre-defined properties [Chen et al., 2016]. Two main challenges need to be tackled in the case of graph generation:

— Similar to text, graphs have a **discrete** nature. Sequential construction methods with autoregressive models thus involve discrete decision steps, which are not differentiable and thus problematic for gradient-based optimization methods common in DL

— Unlike words that compose a sentence, nodes in a graph are **unordered**. Consequently, even if we would like to decompose the generation into a sequence of conditional decisions as this is done in the teacher-forcing trick, there would be no canonical order of decisions.

Approaches that try to tackle the challenges posed by molecular graph generation can be splitted in two categories : sequential methods and non-sequential methods. Sequential methods to graph generation [You et al., 2018b, Li et al., 2018a, You et al., 2018a, Li et al., 2018b] aim to construct a graph by predicting a sequence of discrete addition/edition actions of nodes/edges. Starting from a sub-graph (usually empty), at each time step a discrete transition is predicted and the sub-graph is updated. Although sequential approaches enable us to decouple the number of parameters in models from the the maximum size of the graph processed, due to the discretisation of the final outputs, the graph is still non-differentiable w.r.t. to the decoder's parameters. This prevents us from directly optimising for the objectives we are interested in. In contrast to the sequential process Cao and Kipf [2018], Simonovsky and Komodakis [2018] reconstruct probabilistic graphs. These methods however make use of fixed size multi-layer perceptron layers in the decoding process to predict the graph adjacency and node tensors. This however limits their use to very small graphs of a pre-chosen maximum size. They therefore restrict their study and application to small molecular graphs ; a maximum number of 9 heavy atoms, compared to approximately 40 in sequential models.

In our second contribution, we propose a probabilistic graph decoding scheme that is end-to-end differentiable, computationally efficient w.r.t the number of parameters in the model and capable of generating arbitrary sized graphs.

1.4.3 Molecular Graph Optimization.

In the second contribution we are interested in generating graphs that have a structure that is plausible with the one of a molecule and with certain physico-chemical properties (e.g lead optimization). The aim here is to obtain molecules that satisfy a target set of objectives, for example activity against a biological target while not being toxic *or* maintaining certain properties, such as solubility. The most popular strategy has been to fine-tune a pre-trained generative model to produce/select molecules that satisfy a desired set of properties and the search can be done in the molecules space [Segler et al., 2017] or in the latent space [Gómez-Bombarelli et al., 2016, Kusner et al., 2017, Dai et al., 2018, Jin et al., 2018]. An orthogonal approach would be to cast the problem as a reinforcement learning setting using an efficient sequential-like generative scheme [You et al., 2018b]. In the second contribution we propose to cast the optimization of the molecular graph as a conditional generation problem.

2 SPECTRA : Sparse Entity-centric Transitions

SPECTRA : Sparse Entity-centric Transitions
Rim Assouel, Yoshua Bengio

This chapter presents a joint work with Yoshua Bengio. It was accepted to the NeurIPS Deep Reinforcement Learning Workshop (DRL Neurips 2019)

Affiliation
— Rim Assouel, Mila, Université de Montréal
— Yoshua Bengio, Mila,Université de Montréal

2.1 Abstract

Learning an agent that interacts with objects is ubiquitous in many RL tasks. In most of them the agent's actions have **sparse** effects : only a small subset of objects in the visual scene will be affected by the action taken. We introduce SPECTRA, a model for learning *slot-structured* transitions from raw visual observations that embodies this sparsity assumption. Our model is composed of a perception module that decomposes the visual scene into a set of latent objects representations (i.e. **slot-structured**) and a transition module that predicts the next latent set slot-wise and in a sparse way. We show that learning a perception module jointly with a sparse slot-structured transition model not only biases the model towards more *entity-centric* perceptual groupings but also enables intrinsic exploration strategy that aims at maximizing the number of objects changed in the agent's trajectory.

2.2 Introduction

Recent model-free deep reinforcement learning (DRL) approaches have achieved human-level performance in a wide range of tasks such as games [Mnih et al., 2015]. A critical known drawback of these approaches is the vast amount of experience required to achieve good performance. The promise of model-based DRL is to improve sample-efficiency and generalization capacity across tasks. However model-based algorithms pose strong requirements about the models used. They have to make accurate predictions about the future states which can be very hard when dealing with high dimensional inputs such as images. Thus one of the core challenge in model-based DRL is learning accurate and computationally efficient transition models through interacting with the environment. [Buesing et al., 2018] developed state-space models techniques to reduce computational complexity by making predictions at a higher level of abstraction, rather than at the level of raw pixel observations. However these methods focused on learning a state-space model that doesn't capture the compositional nature of observations: the visual scene is represented by a single latent vector and thus cannot be expected to generalize well to different objects layouts.

Extensive work in cognitive science [Baillargeon et al., 1985, Spelke, 2013] indeed show that human perception is structured around objects. Object-oriented MDP's [Diuk et al., 2008] show the benefit of using object-oriented representations for structured exploration although the framework as it is presented requires hand-crafted symbolic representations. [Bengio, 2017] proposed as a prior (the consciousness prior) that the dependency between high-level variables (such as those describing actions, states and their changes) be represented by a *sparse factor graph*, i.e., with few high-level variables at a time interacting closely, and inference performed sequentially using attention mechanisms to select a few relevant variables at each step.

Besides, a recent line of work [Greff et al., 2017, van Steenkiste et al., 2018, Eslami et al., 2016, Kosiorek et al., 2018, Greff et al., 2019, Burgess et al., 2019] has focused on unsupervised ways to decompose a raw visual scene in terms of objects. They rely on a **slot-structured** representation (see Figure 2.1) of the scene where the latent space is a set of vectors and each vector of the set is supposed to represent an "object" (which we refer to as "entity") of the scene. Watters et al. [2019] investigate the usefulness of slot-structured representations for RL. They introduced a method to learn a transition model that is applied to all the slots of their latent scene representation. Extending their work, we go further and posit that slot-wise transformations should be sparse and that the perception module should be learned jointly with the transition model.

We introduce **Sp**arse **E**ntity-**C**entric **Tra**nsitions (**SPECTRA**), an entity-centric action-conditioned transition model that embodies the fact that the agent's actions have sparse effects: that means that each action will change only a few slots in the latent set and let the remaining ones unchanged. This is motivated by the physical consideration that the agent's interventions are localized in time and space. Our contribution is motivated by three advantages:

- Sparse transitions enable transferable model learning. The intuition here is that the sparsity of the transitions will bias the model towards learning primitive transformations (e.g. how pushing a box affects the state of a box being pushed etc) rather than configuration-dependent transformations, the former being more directly transferable to environments with increased combinatorial complexity.
- Sparse transitions enable a perception module (when trained jointly) to be

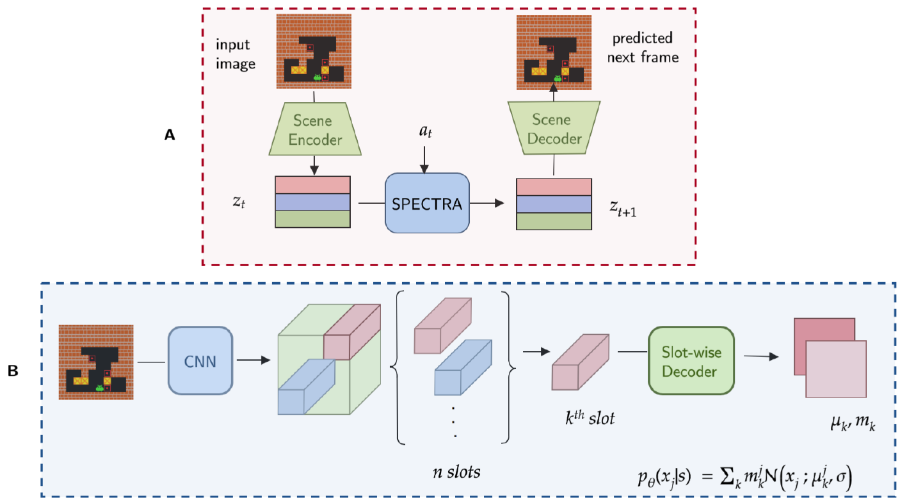


Figure 2.1 – **A**: SPECTRA. Illustration of an entity-centric transition model. **B**: Naive Perception module with a CNN-based encoder and a slot-wise decoder.

biased towards more meaningful perceptual groupings, thus giving potentially better representations that can be used for downstream tasks, compared to representations learned from static data.

- Sparse transitions enable an exploration strategy that learns to predict actions that will change the state of as many entities as possible in the environment without relying on pixels error loss.

2.3 Related Work

Unsupervised visual scene decomposition. Learning good representations of complex visual scenes is a challenging problem for AI models that is far from solved. Recent work [Greff et al., 2017, van Steenkiste et al., 2018, Eslami et al., 2016, Kosiorek et al., 2018, Greff et al., 2019, Burgess et al., 2019] has focused on learning models that discover objects in the visual scene. Greff et al. [2019] further advocate for the importance of learning to segment and represent objects jointly. Like us they approach the problem from a spatial mixture perspective. van Steenkiste et al. [2018] and Kosiorek et al. [2018] build upon Greff et al. [2017] and Eslami et al. [2016] respectively by incorporating next-step prediction as part of the training objective in order to guide the network to learn about essential properties of objects. As specified by van Steenkiste et al. [2019] we also believe that objects are task-dependent and that learning a slot-based representations along with sparse transitions bias the perception module towards **entity-centric** perceptual groupings and that those structured representations could be better suited for RL downstream tasks.

Slot-based representation for RL. Recent advances in deep reinforcement learning are in part driven by a capacity to learn good representations that can be used by an agent to update its policy. Zambaldi et al. [2018] showed the importance of having structured representations and computation when it comes to tasks that explicitly targets relational reasoning. Watters et al. [2019] also show the importance of learning representations of the world in terms of objects in a simple model-based setting. Zambaldi et al. [2018] focus on task-dependent structured computation. They use a self-attention mechanism [Vaswani et al., 2017] to model an actor-critic based agent where vectors in the set are supposed to represent entities in the current observation. Like Watters et al. [2019] we take a model-based approach: our aim is to learn task-independent slot-based representations that can be further used in downstream tasks. We leave the RL part for future work and focus on how learning those representations jointly with a sparse transition model may help learn a better transition model.

2.4 SPECTRA

Our model is composed of two main components: a **perception module** and a **transition module** (section 3.1). The way we formulated the transition implicitly defines an **exploration policy** (section 3.3) that aims at changing the states of as many entities as possible.

Choice of Environment. Here we are interested in environments containing entities an agent can interact with and where actions only affect a *few* of them. **Sokoban** is thus a good testbed for our model. It consists of a difficult puzzle domain requiring an agent to push a set of boxes onto goal locations. Irreversible wrong moves can make the puzzle unsolvable. Each room is composed of walls, boxes, targets, floor and the agent avatar. The agent can take 9 different actions (no-op, 4 types of push and 4 types of move).

Fully Observed vs Learned Entities. The whole point is to work with slot-based representations learned from a raw pixels input. There is no guarantee that those learned slots will effectively correspond to entities in the image. We thus distinguish two versions of the environment (that correspond to two different levels of abstraction):

- **Fully observed entities**: the input is structured. Each entity corresponds to a spatial location in the grid. Entities are thus represented by their one-hot label and indexed by their x-y coordinate. This will be referred to as the *fully observed setting*. There is no need for a perception module in this setting.
- **Raw pixels input**: the input is unstructured. We need to infer the latent entities representations. This will be referred to as the *latent setting*.

2.4.1 Model overview

The idea is to learn an action-conditioned model of the world where at each time step the following take place:

- **Pairwise Interactions**: Each slot in the set gathers relevant information about the slots conditioned on the action taken
- **Active entity selection** : Select slots that will be modified by the action taken
- **Update**: Update the selected slots and let the other ones remain unchanged.

Ideally, slots would correspond to unsupervisedly learned entity-centric representations of a raw visual input like it is done by Burgess et al. [2019], Greff et al. [2019]. We show that learning such perception modules jointly with the sparse transition biases the perceptual groupings to be entity-centric.

Perception module. The perception module is composed of an encoder f_{enc} and a decoder f_{dec}. The encoder maps the input image $\mathbf{x}$ to a set of $\mathbf{K}$ latent entities such that at time-step t we have $f_{enc}(\mathbf{x}^t) = \mathbf{s}^t \in \mathbb{R}^{K \times p}$. It thus outputs a **slot-based** representation of the scene where each slot is represented in the same way and is supposed to capture properties of one *entity* of the scene. Like Burgess et al. [2019], Greff et al. [2019] we model the input image $\mathbf{x}^t$ with a spatial Gaussian Mixture Model. Each slot s_k^t is decoded by the same decoder f_{dec} into a pixel-wise mean μ_{ik} and a pixel-wise assignment m_{ik}^t (non-negative and summing to 1 over k). Assuming that the pixels i are independent conditioned on $\mathbf{s}^t$, the conditional likelihood thus becomes:

$$p_\theta(\mathbf{x}^t|\mathbf{s}^t) = \prod_{i=1}^{D} \sum_k m_{ik}^t \mathcal{N}(\mathbf{x}_\mathbf{i}^\mathbf{t}; \mu_{ik}^t, \sigma^2) \text{ with } \mu_{ik}^t, m_{ik}^t = f_{dec}(s_k^t)_i.$$

As our main goal is to investigate how sparse transitions bias the groupings of entities, in our experiments we use a very simple perception module represented in Figure 2.1. We leave it for future work to incorporate more sophisticated perception modules.

Pairwise interactions. In order to estimate the transition dynamics, we want to select relevant entities (represented at time t by the set $\mathbf{s}^t \in \mathbb{R}^{K \times p}$) that will be affected by the action taken, so we model the fact that each entity needs to gather useful information from entities interacting with the agent (i.e. is the agent close ? is the agent blocked by a wall or a box ? etc..). To that end we propose to use a self-attention mechanism [Vaswani et al., 2017]. From the k-th entity representation s_k^t at time t, we extract a row-vector *key* K_k^t, a row-vector *query* Q_k^t and a row-vector *value* V_k^t conditioned on the action taken such that (aggregating the rows into corresponding matrices and ignoring the temporal indices):

$$\tilde{\mathbf{s}} = softmax(\frac{KQ^T}{\sqrt{d}})V$$

where the softmax is applied separately on each row. In practice we concatenate the results of several attention heads to use it as input to the entity selection phase.

Entity selection. Once the entities are informed w.r.t. possible pairwise interactions the model needs to select which of these entities will be affected by the action taken a^t. Selection of the entities are regulated by a selection gate [Hochreiter and Schmidhuber, 1997b, Cho et al., 2014] computed slot-wise as:

$$f_k^t = \sigma(MLP([\tilde{s}_k^t; a^t])) \tag{2.1}$$

where f_k^t can be interpreted as the probability for an entity to be selected.

Update. Finally, each **selected** entity is updated conditioned on its state s_k^t at time-step t and the action taken a^t. We thus simply have:

$$s_k^{t+1} = f_k^t f_\theta([s_k^t, a^t]) + (1 - f_k^t) s_k^t$$

f_θ is a learned action-conditioned transformation that is applied slot-wise. We posit that enforcing the transitions to be slot-wise and implicitly sparse will bias the model towards learning more primitive transformations. We verify this assumption in next subsection in the simpler case where the entities are fully observed (and not inferred with a perception module).

2.5 Experiments

In this work we demonstrate three advantages of entity-centric representations learned by SPECTRA:

– Implicitly imposing the transitions to be sparse will enable us to learn transition models that will transfer better to environments with increased combinatorial complexity. Section 4.1.
– Learning slot-based representations jointly with a sparse transition model will bias the perceptual groupings to be *entity-centric*. Section 4.2.
– Finally we investigate the usefulness of the implicit exploration scheme induced by SPECTRA when learning the model jointly. Section 4.3.

2.5.1 Learned Primitive Transformations

In this section we show that sparse selection in the transitions yields learned slot-wise transformations that are transferable to out-of-distribution settings with increased combinatorial complexity. We restrict ourselves to the **fully observed setting**. Like Zambaldi et al. [2018] the entities correspond to a spatial location in the 7×7 grid. Each entity s_k is thus described in terms of its label to which we append its x-y coordinate. The results in Figure 2.2 are intuitive ; to learn the right transitions with our formulation, the model is forced to:

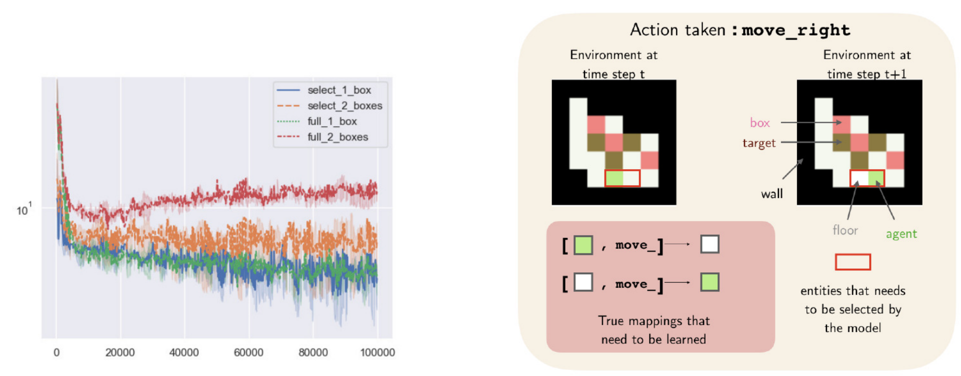


Figure 2.2 – **left**: Full and sparse settings are trained on environment containing one box and evaluated out-of-distribution on two boxes. We plotted the validation losses of both settings during training. The full connectivity architecture is unable to achieve out-of-distribution generalization to an environment with two boxes. **right**: Illustration of what the model has to learn in the fully observed setting: to be correct the model needs to map any concatenation of [agent,move] to a vacated position = floor and to select only the right entities to be changed. The learned mappings are general rules that are directly transferable to settings with more boxes.

– select only the relevant entities to be updated.
– learn the right primitive transformation (e.g. if the agent slot is selected to be modified by any of the *move* actions, then its position is vacated, so the model should map any concatenation of [*agent*, *move*] to the *floor* label etc...). See Figure 2.2, right.

Here entity representations are not learned and thus correspond to their labels. We thus train the model with a simple cross-entropy loss. We are interested in comparing two settings:

– **Sparse** setting: the transformation is still done slot-wise to selected entities only. Each slot contains the label and x-y coordinate of the entity only. The transformation is applied to a concatenation of the entity label and the action $[label, action]$.
– **Full** setting: the transformation is still done slot-wise but this time each slot in $\tilde{s}_t$ potentially contains information about all the other slots in the set. The transformation is applied to a concatenation of the entity representation $\tilde{s}_k^t$ and the action $[\tilde{s}_k^t, action]$. Thus we hypothesize that the transformation module will learn *configuration-dependent* rules (e.g. if an agent is close to a box and a wall, and 3 steps ahead there is a target to be reached, and it takes a move action to do so) that will not be easily transferable to environments

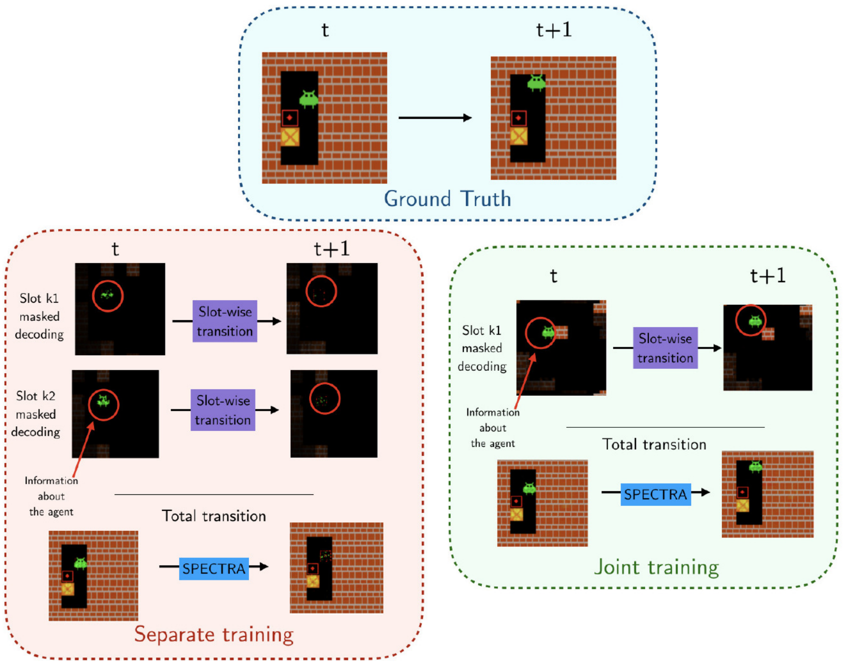


Figure 2.3 – Comparison of slot-wise masked decodings when the perception module is trained separately or jointly with the sparse transitions. We show the reconstruction associated with the slots that contain information about the agent. When the perception module is trained jointly, slots in the learned latent set are biased to be entity-centric (here agent-centric).

with increased complexity and a wider variety of contexts.

Both settings are illustrated in Figure 2.6. In Figure 2.2 we reported the evolution of training and evaluation losses of both the full and the sparse settings when the models are trained in a 7x7 environment with one box and evaluated in a 7x7 environment with two boxes.

2.5.2 Structured Representation Learning

In this section we demonstrate how learning a perception module along with sparse transitions will bias this module towards learning **entity-centric** perceptual groupings of the raw pixel input. In order to verify this intuition we compare in Figure 2.3 the reconstructions from the perception module when it is trained *separately* vs *jointly* with the sparse transition module. In this experiment the input is not structured anymore but just a raw 112x112x3 pixel image. We used a simple perception module as described in Figure 2.1.

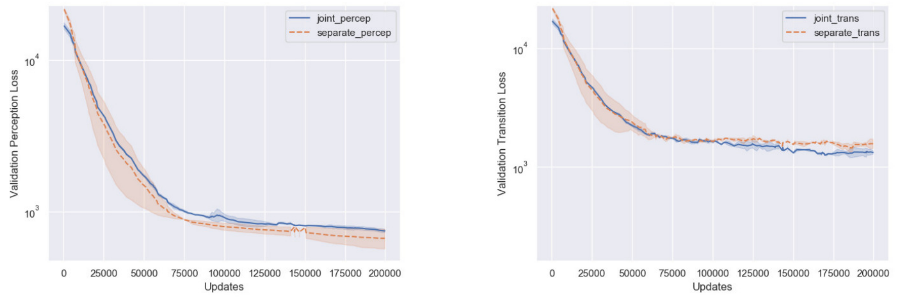


Figure 2.4 – Loss vs training updates, with training is done in pixel space, transitions are sampled randomly and results are averaged over 3 runs. **left**: Validation perception loss $\mathcal{L}_{percep}$ of joint and separate training **right**: Validation transition loss $\mathcal{L}_{trans}$ of joint and separate training. Separate training is better in terms of perception loss but joint training gives a better transition model. We posit that this is because the slots are biased to be **entity-centric** and transformations involving only relevant entities are easier to learn.

We thus distinguish two losses, a reconstruction loss

$$\mathcal{L}_{percep} = \sum_{i=1}^{D} \log \sum_{k} m_{ik}^{t} \mathcal{N}(\mathbf{x_i^t}; \mu_{ik}^{t}, \sigma^2)$$

and a transition loss

$$\mathcal{L}_{trans} = \sum_{i=1}^{D} \log \sum_{k} \hat{m}_{ik}^{t+1} \mathcal{N}(\mathbf{x_i^{t+1}}; \hat{\mu}_{ik}^{t+1}, \sigma^2)$$

with $\mu_{ik}^t, m_{ik}^t = f_{dec}(s_k^t)_i$, $s^t = f_{enc}(\mathbf{x}^t)$, $\hat{\mu}_{ik}^{t+1}, \hat{m}_{ik}^{t+1} = f_{dec}(\hat{s}_k^{t+1})_i$, and $\hat{s}_k^{t+1} = f_{trans}(s_k^t)$ is the future state predicted by the transition function.

f_{dec}, f_{enc} and f_{trans} are respectively the decoder, the encoder and the transition modules. For the joint training (resp. separate training) setting, gradients from $\mathcal{L}_{trans}$ are back-propagated through parameters of f_{enc} and f_{trans} (resp. f_{trans} only). In both settings, gradients from $\mathcal{L}_{percep}$ are back-propagated through parameters of f_{enc} and f_{dec}.

In Figure 2.3 we put particular attention on the masked reconstructions from slots containing *visual information* about the agent. We can directly notice that the perceptual groupings done by the encoder, when it is trained jointly with the transition module, are **agent-centric**: the information about the agent is contained in one slot only (whereas it is often contained in several slots in the separate training settings). Moreover, in Figure 2.4 we see the joint training setting leading to a better transition model: we hypothesize that the transformations are easier to learn specifically because they have to focus on the effects of the actions taken on entities, i.e., involving a few strongly dependent variables at a time rather than more global but more specific configurations involving all the variables in the state, as suggested by Bengio [2017].

2.5.3 Intrinsic Exploration Strategy

In many environments a uniformly random policy is insufficient to produce action and observation sequences representative enough to be useful for downstream tasks. In this paper we suggest to learn an exploration policy jointly with the model, based on an intrinsic reward that depends on the transition model itself and exploits its entity-centric structure to quantify the diversity of aspects of the environment modified by exploratory behavior. Our model learns to first **select** entities that will be changed and then learns how to **transform** the selected entities. Similar to the empowerment intrinsic objectives Klyubin et al. [2005], Kumar [2018], a natural exploration strategy in settings like Sokoban would be to follow trajectories that overall have as many entities being selected as possible. If the agent indeed never pushes a box on target when learning its transition model, it will not be able to transfer its knowledge to a task where it has to push all the boxes on all the targets. We thus suggest to learn a policy that **maximizes the number** of entities selected, as predicted by the current model. We alternate between policy update and model update.

We used a 10-step DQN for the exploration policy and have the DQN and the model share the same 1-step replay buffer. The DQN policy is ϵ-greedy with ϵ decaying from 1 to 0.3. In order to train the DQN we used the following intrinsic 1-step reward:

$$r(\mathbf{s}_t, a_t) = \sum_k \mathbb{1}_{(f_k^t \geq h)} \tag{2.2}$$

with h a chosen threshold for the update gate value. We expect this training strategy to promote trajectories with as many entities that will have their state changed as possible. We thus expect the agent to learn not to get stuck, aim for the boxes, push

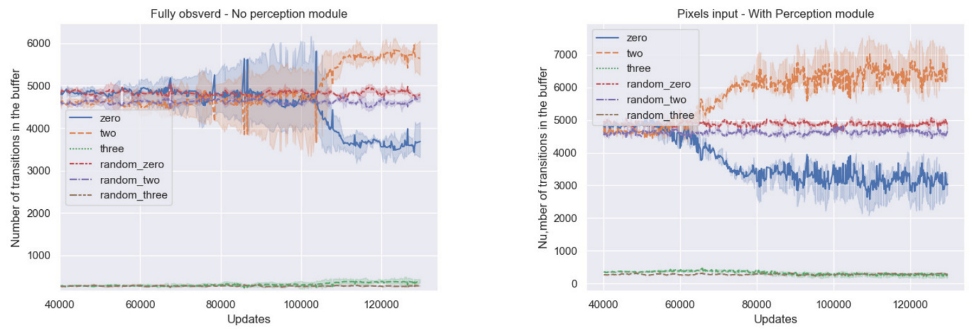


Figure 2.5 – Comparison is done against randomly sampled transitions. **left**: Number of entities changed in the 1-step buffer during training. As expected, the number of transitions with 2 spatial locations changed in the grid increases whereas the ones with no location changed decreases. We also notice a slight increase in the number of transitions with 3 spatial locations changed (corresponding to the agent moving a box!). Training is done in the *fully observed setting*. **right**: Training done in pixel space. Again here, the number of transitions with two spatial locations changed in the grid increases whereas the ones with no location changed decreases. However the number of of transitions with the agent that moves a box did not increase.

them etc... In order to validate that intuition, we first conduct experiments in the **fully observed setting**. In this setting we consider the following types of moves:

- ***valid_move***: Whenever the agent takes a *move* action in a valid direction, two entities will have their state changed: the initial location of the agent and the next one.
- ***valid_push*** : Whenever the agent takes a *push* action and a box is available to be pushed in the chosen directions, three entities will have their state changed: the initial location of the agent, the initial location of the box and the next location of the box.
- ***blocked_push*** : Whenever the agent takes a *push* action when there is no box to push in the chosen direction, nothing happens.
- ***blocked_move***: Whenever the agent takes a *move* action in a non-valid direction (against a wall, a box etc...), nothing happens.

With our suggested training strategy we expect the agent to promote trajectories with more transitions of type ***valid_move*** than ***blocked_move*** and ***blocked_push*** and hopefully with the number of ***valid_push*** transitions increased as well. During training, we thus monitor the true number of entities changed in the transitions stored in the shared 1-step buffer. We also performed the same experiment in the raw input pixels setting and monitored the true number of entities changed in the

1-step buffer during training. Results are reported in Figure 2.5 and confirm our hypothesis: the agent learns to avoid actions that will result in no changes in the environment (***blocked_push*** and ***blocked_move***).

2.6 Conclusion and Future Work

We have introduced **SPECTRA**, a novel model to learn a sparse slot-structured transition model. We provided evidence to show that sparsity in the transitions yields models that learns more primitive transformations (rather than configuration-dependent) and thus transfer better to out-of-distribution environments with increased combinatorial complexity. We also demonstrated that the implicit sparsity of the transitions enables an exploration strategy that aims at maximizing the number of entities that be will be modified on the agent's trajectory. In Figure 2.5 we showed that with this simple exploration strategy the agent leans to avoid actions that will not change the environment (*blocked_move* and *blocked_push*). Preliminary results in pixel space show that SPECTRA biases even a simple perception module towards perceptual groupings that are entity-centric. We anticipate that our model could be improved by incorporating a more sophisticated perception module. In the future we aim to use SPECTRA to investigate possible uses in model-based reinforcement learning.

2.7 Architecture and Hyperparameters

2.7.1 Fully observed setting

In the fully observed setting the input at time t is a set $\mathbf{o}^t \in \{0, 1\}^{N \times 7}$ corresponding to one-hot labels (that can be agent (off and on target), box (off and on target), wall, target and floor). of each entity in a 7×7 grid ($N = 49$). We also append their normalized $x - y$ coordinates so that the final input to the transition model is a set $\mathbf{s}^t \in \{0, 1\}^{N \times 9}$. Like detailed previously in Figure 2.6, the transition model is composed of two modules: the **selection** module and the **transformation** module.

In section 4.1 we also distinguished between the **sparse** and the **full** setting and they are described in 2.6. In the full setting, there is no more selection bottleneck and the transition module is a simple transformer-like architecture.

Selection module. The selection module is a transformer-like architecture. It takes as input at time step t the concatenation $\mathbf{e}^t = [\mathbf{s}^t, a^t]$ of the set $\mathbf{s}^t$ and the action a_t. The selection module is then composed of 2 attention heads where is head is stack of 3 attention blocks [Vaswani et al., 2017, Zambaldi et al., 2018]. The 3 blocks are 1-layer MLP that output key, query and value vectors of channels size 32, 64, 64 respectively. The first two blocks are followed by $RELU$ non linearities and the last one doesn't have any. The output of the attention phase is thus the concatenation of values obtained from the 2 attentions heads $\tilde{\mathbf{s}}^t \in \mathbb{R}^{N\times 112}$. To obtain the selection binary selection variables we then simply apply slot-wise a single layer MLP to the concatenation $\tilde{\mathbf{e}}^t = [\tilde{\mathbf{s}}^t, a^t]$ followed by a logSoftmax non-linearity in order to compute the log-probabilities of each entity to be modified by the action taken. The output of the selection module is thus a set of log-probabilities $\mathbf{l}^t \in \mathbb{R}^{N\times 2}$.

Transformation module. The transformation module is a simple shared 2-layers MLP that is applied slot-wise to the the concatenation $\mathbf{e}^t = [\mathbf{s}^t, a^t]$ of the input set $\mathbf{s}^t \in \{0,1\}^{N\times 9}$ and the action taken. It outputs channels of sizes 16, 7 respectively. The first layer is followed by a $RELU$ non-linearity and the last one by a logSoftmax non-linearity in order to compute the log-probabilities of the label of each predicted entity.

Full setting. In the full setting, we don't have a selection bottleneck anymore. The transformation module is thus directly applied to the output of the attention phase $\tilde{\mathbf{e}}^t = [\tilde{\mathbf{s}}^t, a^t]$. It consits this time of a simple shared 3-layers MLP that is applied slot-wise and outputs channels of sizes 64, 32, 7 respectively. The first two layers are followed by a $RELU$ non-linearity and the last one by a logSoftmax non-linearity .

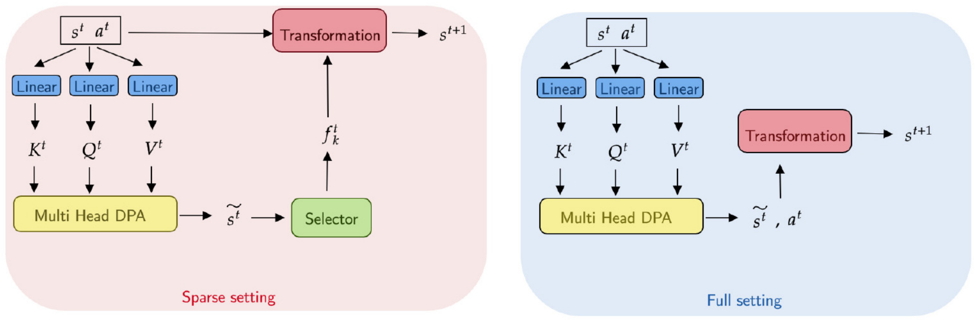


Figure 2.6 – Transition model with and without selection phase.

2.7.2 Latent setting

In the latent setting the input at time t is a raw pixels (RGB) image $\mathbf{o}^t \in \mathbb{R}^{112\times112\times3}$. In the latent setting, the transition model is composed of a **perception** module, a **selection** module and a **transformation** module.

Perception module. When dealing with unstructured input we first need a way to extract entities latent representations. For this work we used a very simple and naive perception module, with an encoder similar to what is done by Zambaldi et al. [2018], Santoro et al. [2017]. Like detailed in Figure 2.1, we use a CNN to parse pixel inputs into k feature maps of size $n \times n$, where k is the number of output channels of the CNN. We choose arbitrarily $n = 4$ and didn't perform any hyperparameter search for the CNN architecture. We then concatenate x and y coordinates to each k-dimensional pixel feature-vector to indicate the pixel's position in the map. We treat the resulting pixel-feature vectors as the set of entities $\mathbf{s}^t \in \mathbb{R}^{N\times k}$ where here $N = n^2 = 16$. We denote as $\mathbf{s}^t_{coord} \in \mathbb{R}^{N\times k+2}$ the entities set to which we have appended the x-y position in the map.

As our loss is a pixel loss we also need a decoder that decodes each entity $s^t_{k,coord}$ of the set $\mathbf{s}^t$ back to its corresponding mean μ^t_k and mask m^t_k. The CNN of the encoder outputs channels of size (16, 32, 32, 32, 32). All layers (except the last one) are followed by $RELU$ non-linearities. Kernel sizes are (3, 3, 4, 3) and strides (2, 2, 2, 2, 1). The decoder is composed of a 2-layers MLP followed by a stack of transposed convolutions. The MLP outputs channels of sizes $(7 \times 34, 7 \times 7 \times 34)$ with a $RELU$ non-linearity between the 2 layers. The output is then resized to $7 \times 7 \times 34$ map that will be fed to the convolution part. For the convolution part, it outputs maps of channel sizes (4, 4, 4, 4, 4) with $RELU$ non-linearities between each layer. The kernel sizes are (3, 3, 5, 4).

Selection and Tranformation modules. The selection and transformation module are very similar to the fully observed setting, except that they operate on the latent space, so we do not apply LogSofmax non-linearities for the transformation part. The input of the selection module is $\mathbf{s}^t_{coord}$ and the input to the transformation module is $\mathbf{s}^t$. The **selection** module is composed of 2 attention heads where is head is stack of 3 attention blocks [Vaswani et al., 2017, Zambaldi et al., 2018]. The 3 blocks are 1-layer MLP that output key, query and value vectors of channels size 34, 16, 16 respectively. The first two blocks are followed by $RELU$ non linearities and the last one doesn't have any. The output of the attention phase is thus the concatenation of values obtained from the 2 attentions heads $\tilde{\mathbf{s}}^t \in \mathbb{R}^{N \times 32}$. To obtain the selection binary selection variables we then simply apply slot-wise a 3-layers MLP of channels sizes 16, 32, 32 respectively to the concatenation $\tilde{\mathbf{e}}^t = [\tilde{\mathbf{s}}^t, a^t]$ followed by a Softmax non-linearity in order to compute the probabilities of each entity to be modified by the action taken. The output of the selection module is thus a set of probabilities $\mathbf{p}^t \in \mathbb{R}^{N \times 2}$. The **transformation** module is a simple 2-layers MLP of channels sizes 32,32 respectively with a $RELU$ non-linearity between the two layers.

2.8 Additional Visualisations

In this section we reported additional visualizations similar to Figure 2.3 where we monitor:

– Differences in slot-wise masked decodings of the perception module when it is trained jointly and separately from the sparse transitions.
– Differences in the slot-wise transformations earned by the transition model when it is trained separately and jointly with the perception module.

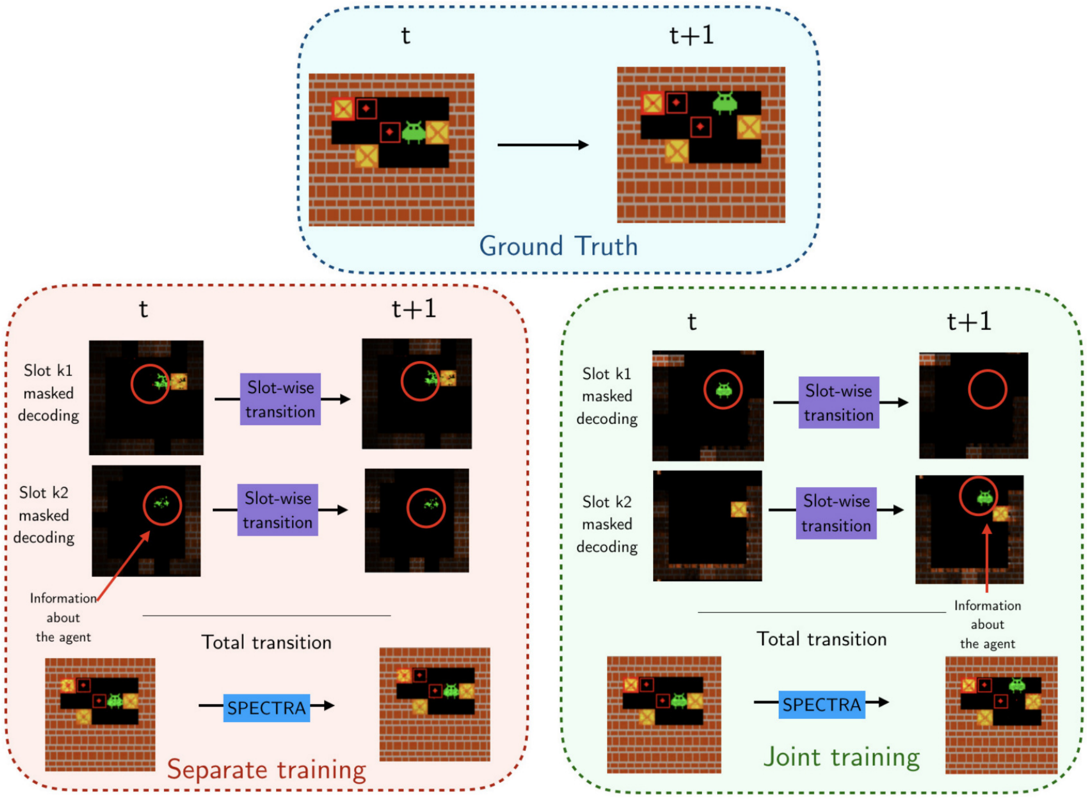


Figure 2.7 – Additional visualisations of masked decodings from joint and separate training settings.

We notice that joint training enables to learn slot-structured representation that are **entity-centric** and thus enable to learn better transition models. The transformations learned are especially visually more *interpretable*.

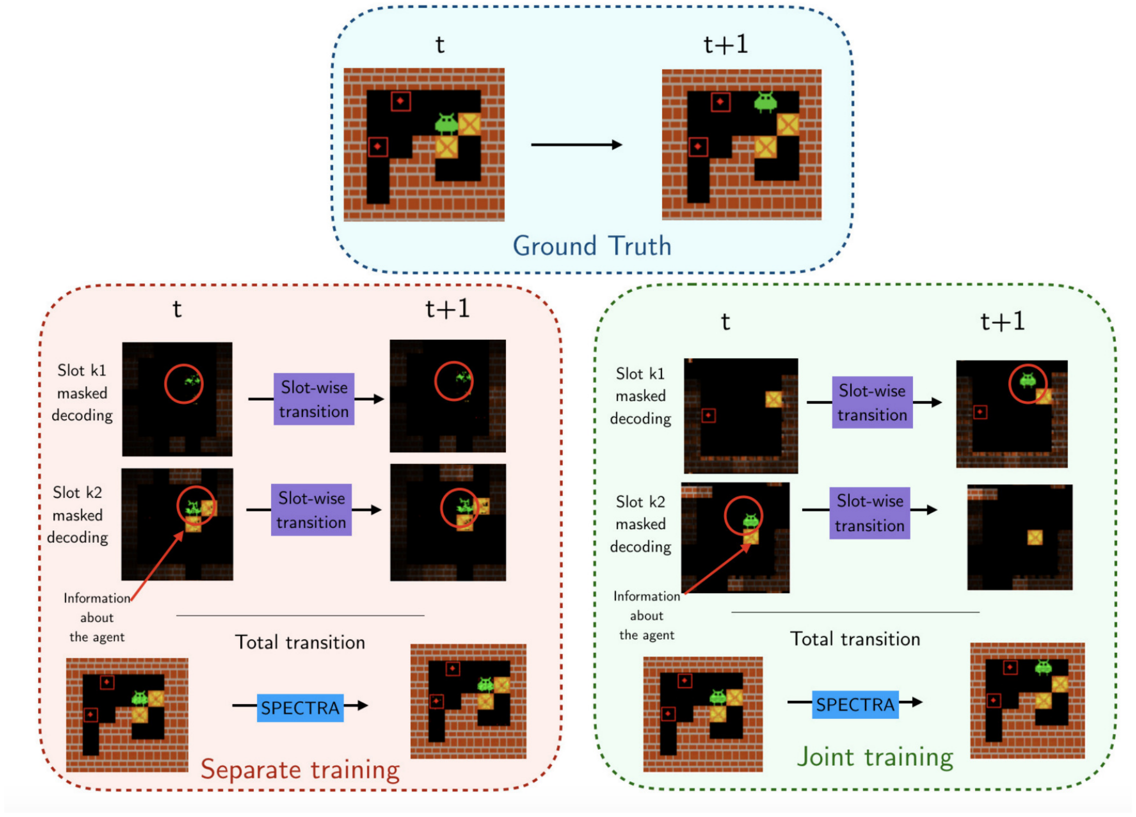


Figure 2.8 – Additional visualisations of masked decodings from joint and separate training settings.

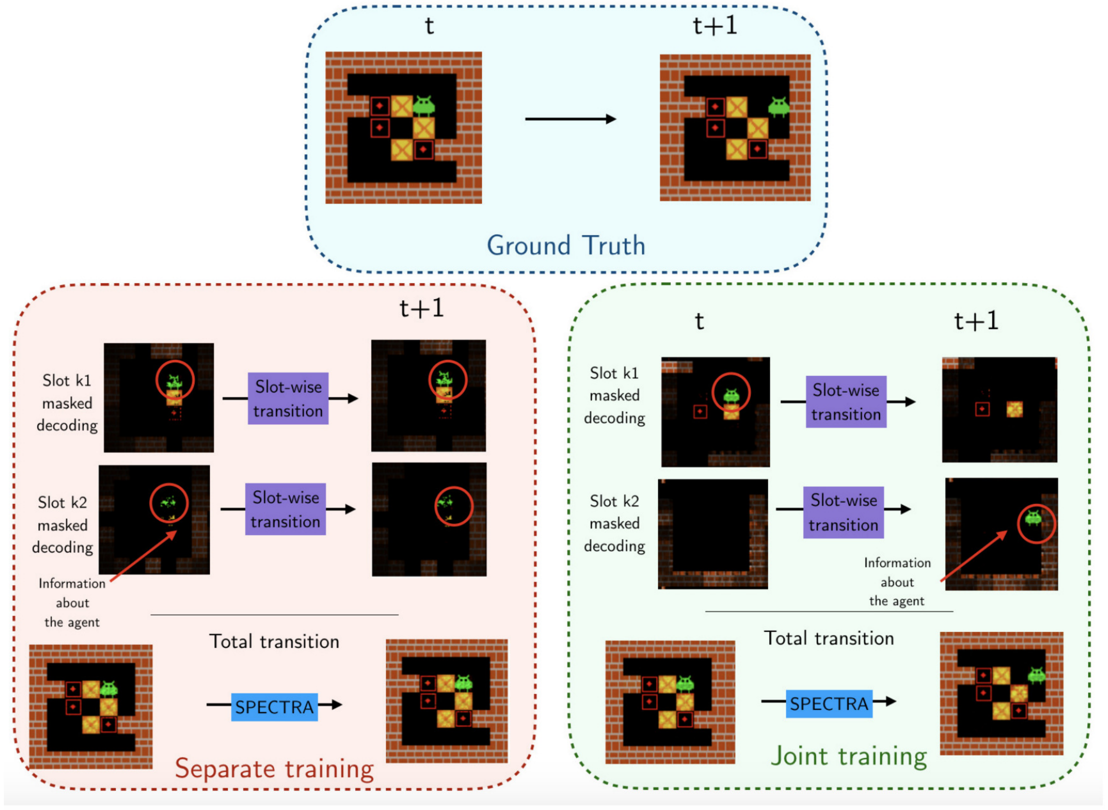


Figure 2.9 – Additional visualisations of masked decodings from joint and separate training settings.

3 DEFACTOR : Differentiable Edge Factorization-based Probabilistic Graph Generation

DEFACTOR : Differentiable Edge Factorization-based Probabilistic Graph Generation

Rim Assouel, Mohamed Ahmed, Marwin H Segler, Amir Saffari, and Yoshua Bengio

This chapter presents joint work with Mohamed Ahmed, Marwin H Segler, Amir Saffari, and Yoshua Bengio. It was accepted at the 2nd NeurIPS Workshop on Machine Learning for Molecules and Materials (MLMM Neurips 2018).

Affiliation

– Rim Assouel, Benevolent.AI, Mila, Université de Montréal
– Mohamed Ahmed,Benevolent.AI
– Marwin H Segler, Benevolent.AI
– Amir Saffari, Benevolent.AI
– Yoshua Bengio, Mila, Université de Montréal

3.1 Abstract

Generating novel molecules with optimal properties is a crucial step in many industries such as drug discovery. Recently, deep generative models have shown a promising way of performing de-novo molecular design. Although graph generative models are currently available they either have a graph size dependency in their number of parameters, limiting their use to only very small graphs or are formulated as a sequence of discrete actions needed to construct a graph, making the output graph non-differentiable w.r.t the model parameters, therefore preventing them to be used in scenarios such as conditional graph generation. In this work we propose a model for conditional graph generation that is computationally efficient and enables direct optimisation of the graph. We demonstrate favourable performance of our model on prototype-based molecular graph conditional generation tasks.

3.2 Introduction

We address the problem of learning probabilistic generative graph models for tasks such as the conditional generation of molecules with optimal properties. More precisely we focus on generating realistic molecular graphs, similar to a target molecule (the prototype).

The main challenge here stems from the discrete nature of graph representations for molecules; which prevents us from using global discriminators that assess generated samples and back-propagate their gradients to guide the optimisation of a generator. This becomes a bigger hindrance if we want to either optimise a property of a molecule (graph) or explore the vicinity of an input molecule (prototype) for conditional optimal generation, an approach that has proven successful in controlled image generation Odena et al. [2016], Chen et al. [2016].

Several recent approaches aim to address this limitation by performing indirect optimisation Jin et al. [2018], You et al. [2018a], Li et al. [2018a]. You et al. You et al. [2018a] formulate the molecular graph optimisation task in a reinforcement learning setting, and optimise the loss with policy gradient Yu et al. [2016]. However policy gradient tends to suffer from high variance during training. Kang and Cho Kang and Cho [2018] suggest a reconstruction-based formulation which is directly applicable to discrete structures and does not require gradient estimation. However, it is limited by the number of samples available. Moreover, there is always a risk that the generator simply ignores the part of the latent code containing the property that we want to optimise. Finally, Jin et al.Jin et al. [2018] apply Bayesian optimisation to optimise a proxy (the latent code) of the molecular graph, rather than the graph itself.

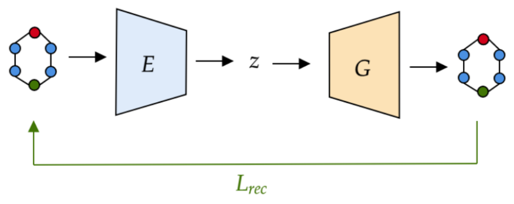
In contrast, Simonovsky and Komodakis Simonovsky and Komodakis [2018] and De Cao and Kipf Cao and Kipf [2018] have proposed decoding schemes that output graphs (adjacencies and node/edge feature tensors) in a single step, and so are able to perform direct optimisation on the probabilistic continuous approximation of a graph. However, both decoding schemes make use of fixed size MLP layers which restricts their use to very small graphs of a predefined maximum size.

Our approach (DEFactor) depicted in Figure 3.1 aims to directly address these issues with a probabilistic graph decoding scheme that is end-to-end differentiable, computationally efficient w.r.t the number of parameters in the model and capable of generating arbitrary sized graphs. We evaluate DEFactor on the task of constrained molecule property optimisation Jin et al. [2018], You et al. [2018a] and demonstrate that our results are competitive with recent results.

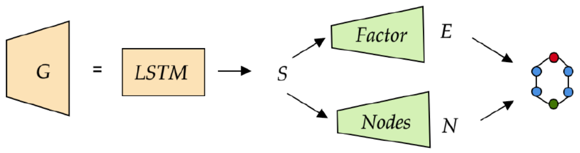
3.3 Related work

Lead-based Molecule Optimisation. The aim here is to obtain molecules that satisfy a target set of objectives, for example activity against a biological target while not being toxic *or* maintaining certain properties, such as solubility. Currently a popular strategy is to fine-tune a pretrained generative model to produce/select molecules that satisfy a desired set of properties [Segler et al., 2017].

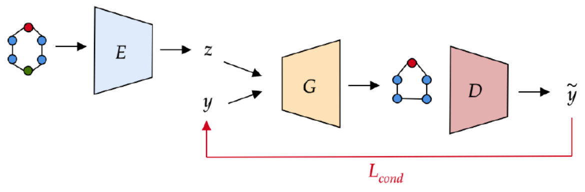
Bayesian optimisation is proposed to explore the learnt latent spaces for molecules [Gómez-Bombarelli et al., 2016], and is shown to be effective at exploiting feature rich latent representations [Kusner et al., 2017, Dai et al., 2018, Jin et al., 2018]. Li et al. [2018b,a] propose sequential graph decoding schemes whereby conditioning properties can be added to the input. However these approaches are unable to perform direct optimisation for objectives. Finally You et al. [2018a] reformulate the



(a) The full autoencoder (step **1** to **4**)



(b) Expanding the steps (**3** for the LSTM and **4** for the factorization and node decoding) of the generator G of the autoencoder



(c) The conditional setting a discriminator D that assesses the outputs and gives its feedback to the generator G. L_{rec} (resp. L_{cond}) refers to the reconstruction (resp. conditional) loss described in section 3.3.

Figure 3.1 – Overview of our molecule autoencoding ((a) and (b)) and conditional generation (c) process.

problem in a reinforcement learning setting, and objective optimisation is performed while keeping an efficient sequential-like generative scheme [You et al., 2018b].

Graph Generation Models. Sequential methods to graph generation [You et al., 2018b, Li et al., 2018a, You et al., 2018a, Li et al., 2018b] aim to construct a graph by predicting a sequence of addition/edition actions of nodes/edges. Starting from a sub-graph (normally empty), at each time step a discrete transition is predicted and the sub-graph is updated. Although sequential approaches enable us to decouple the number of parameters in models from the the maximum size of the graph processed, due to the discretisation of the final outputs, the graph is still non-differentiable w.r.t. to the decoder's parameters. This again prevents us from directly optimising for the objectives we are interested in.

In contrast to the sequential process Cao and Kipf [2018], Simonovsky and Komodakis [2018] reconstruct probabilistic graphs. These methods however make use of fixed size MLP layers when decoding to predict the graph adjacency and node tensors. This however limits their use to very small graphs of a pre-chosen maximum size. They therefore restrict study and application to small molecular graphs ; a maximum number of 9 heavy atoms, compared to approximately 40 in sequential models.

We propose to tackle these drawbacks by designing a graph decoding scheme that is:

- **Efficient**: so that the number of parameters of the decoder does not depend on a fixed maximum graph size.
- **Differentiable**: in particular we would like the final graph to be differentiable w.r.t the decoder's parameters, so that we are able to directly optimise the graph for target objectives.

3.4 DEFactor

Molecules can be represented as graphs $G = (V, E)$ where atoms and bonds correspond to the nodes and edges respectively. Each node in V is labeled with its atom type which can be considered as part of its features. The adjacency tensor is given by $E \in \{0, 1\}^{n \times n \times e}$ where n is the number of nodes (atoms) in the graph and e is the number of possible edge (bond) types. The node types are represented by a node feature tensor $N \in \{0, 1\}^{n \times d}$ which is composed of several one-hot-encoded properties.

3.4.1 Graph Construction Process.

Given a molecular graph defined as $G = (N, E)$ we propose to leverage the edge-specific information propagation framework described by Simonovsky and Komodakis [2017] to learn a set of informative embeddings from which we can directly infer a graph. Our graph construction process is composed of two parts:

- An **Encoder** that in
  - **step 1** performs several spatial graph convolutions on the input graph, and in
  - **step 2** aggregates those embeddings into a single graph latent representation.
- A **Decoder** that in
  - **step 3** autoregressively generates a set of continuous node embeddings conditioned on the learnt latent representation, and in
  - **step 4** reconstructs the whole graph using edge-factorization.

Figure 3.1 provides a summary of those 4 steps.

Steps 1 and 2: Graph Representation Learning. We use the Graph Convolutional Network (GCN) update rule [Kipf and Welling, 2016b] to encode the graph. Each node embedding can be written as a weighted sum of the edge-conditioned information of its neighbors in the graph. Namely for each l-th layer of the encoder, the representation is given by:

$$H^l = \sigma(\sum_e [D_e^{-\frac{1}{2}} E_e D_e^{-\frac{1}{2}} H^{l-1} W_e^l] + H^{l-1} W_s^l) \tag{3.1}$$

where E_e is the e-th frontal slice of the adjacency tensor, D_e the corresponding degree tensor and W_e^l and W_s^l are learned parameters of the layer.

Once we have the node embeddings we aggregate them to obtain a fixed-length latent representation of the graph. We propose to parametrize this aggregation step by an *LSTM* and we compute the graph latent representation by a simple linear transformation of the last hidden state of this **Aggregator**:

$$z = g_{agg}(f^e_{LSTM}(\{H^K\})). \tag{3.2}$$

Because the use of an LSTM makes aggregations permutation dependant, Like Hamilton et al. [2017], we adapt the aggregator using randomly permuted sets of embeddings and empirically validated that this did not affect the performance of the model significantly.

In the subsequent steps we are interested in designing a graph decoding scheme from the latent code that is both scalable and powerful enough to model the interdependencies between the nodes and edges in the graph.

Step 3: Autoregressive Generation. We are interested in building a graph decoding scheme that models the nodes and their connectivity (represented by continuous embeddings S) in an autoregressive fashion. This is in contrast to Simonovsky and Komodakis [2018], Cao and Kipf [2018], where each node and edge is conditionally independent given the latent code z. In practice this means that every detail of the interdependencies within the graph have to be encoded in the latent variable. We propose to tackle this drawback by autoregressive generation of the continuous embeddings $s = [s_0, s_1, ..., s_n]$ for n nodes. More precisely we model the generation of node embeddings such that:

$$p(s|z) = \prod_{i=1}^{n} p(s_i|s_{<i}, z). \tag{3.3}$$

In our model, the autoregressive generation of embeddings is parametrized by a simple Long Short-Term Memory (LSTM, [Hochreiter and Schmidhuber, 1997a]) and is completely deterministic such that at each time step t the LSTM decoder takes as input the previously generated embeddings and the latent code z which captured node-invariant features of the graph. Each embedding is computed as a function of the concatenation of the current hidden state and the latent code z such that:

$$h_{t+1} = f^d_{LSTM}(g_{in}([z, s_t]), h_t) \tag{3.4}$$

$$s_{t+1} = f_{embed}([h_{t+1}, z]), \tag{3.5}$$

where f^d_{LSTM} corresponds to the LSTM recurrence operation and g_{in} and f_{embed} are parametrized as simple MLP layers to perform nonlinear feature extraction.

Step 4: Graph Decoding from Node Embeddings. As stated previously, we want to drive the generation of the continuous embeddings s towards latent factors that contains enough information about the node they represent (i.e. we can easily retrieve the one-hot atom type performing a linear transformation of the continuous embedding) and its neighbourhood (i.e. the adjacency tensor can be easily retrieved by comparing those embeddings in a pair-wise manner). For those reasons, we suggest to factorize each bond type in a relational inference fashion [Zitnik et al., 2018, Kipf et al., 2018].

Let $S \in \mathbb{R}^{n \times p}$ be the concatenated continuous node embeddings generated in the previous step. We reconstruct the adjacency tensor E by learning edge-specific similarity measure for k-th edge type as follows:

$$p(E_{:,:,k}|S) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(E_{i,j,k}|s_i, s_j). \tag{3.6}$$

This is modeled by a set of edge-specific factors $U = (u_1, \cdots, u_e) \in \mathbb{R}^{e \times p}$ such that we can reconstruct the adjacency tensor as :

$$\tilde{E}_{i,j,k} = \sigma(s_i^T D_k s_j) = p(E_{i,j,k}|s_i, s_j), \tag{3.7}$$

where σ is the logistic sigmoid function, D_k the corresponding diagonal matrix of the vector u_k and the factors $(u_i) \in \mathbb{R}^{e \times p}$ are learned parameters.

We reconstruct the node features (i.e. the atom type) with a simple affine transformation such that:

$$\tilde{N}_{i,:} = p(N_i|s_i) = \text{softmax}(W s_i), \tag{3.8}$$

where $W \in \mathbb{R}^{p \times d}$ is a learned parameter.

Generating Graphs of arbitrary sizes. In order to generate graphs of different sizes we need to add what we call here an **Existence** module that retrieves a probability of a node belonging to the final graph for each of the embedding generated (in step 3). This module is parametrized as a simple MLP layer followed by a sigmoid activation and stops the unrolling of the embedding LSTM generator whenever we encounter a *non-informative* embedding. This module can be interpreted as an $< eos >$ translator.

3.4.2 Training

Teacher forcing. To make the model converge in reasonable time we adapt teacher-forcing on language models [Williams and Zipser, 1989] as follows. The training is thus done in 3 phases:

- We first pre-train the GCN part along with the embedding decoder (factorization, nodes and existence modules) to reconstruct the graphs. This corresponds to the training of a simple Graph AE as in Kipf and Welling [2016a] except that we also want to reconstruct the nodes' one-hot features (and not just the *relations*).
- We then append those two units to the embedding aggregator and generator while keeping them fixed. In this second phase, the embedding generator is trained using teacher forcing where at each time step t the *LSTM* decoder does not take as input the previously generated embedding but the *true* one that is the direct output of the pretrained GCN embedding encoder.
- Finally in order to transition from teacher-forcing to a fully autoregressive state we increasingly [Bengio et al., 2015] feed the LSTM generator more of its own predictions. When a fully autoregressive state is reached the pre-trained units are unfrozen and the whole model continues training end-to-end.

Log-Likelihood Estimates We train the autoencoder on the reconstruction error using the MLE with the estimate negative log-likelihood given by:

$$\mathcal{L}_{rec} = \mathcal{L}_X + \mathcal{L}_{\bar{X}} + \mathcal{L}_N \tag{3.9}$$

where X and $\bar{X}$ correspond to the existing and non existing edges in the adjacency tensor E, and N is the node features containing n nodes features such that:

$$\mathcal{L}_X = -\frac{1}{|X|} \sum_{(i,j)\in X} E_{i,j,:}^T \log(\tilde{E}_{i,j,:}) + (1 - E_{i,j,:})^T \log(1 - \tilde{E}_{i,j,:}) \tag{3.10}$$

$$\mathcal{L}_{\bar{X}} = -\frac{1}{|\bar{X}|} \sum_{(i,j)\in \bar{X}} \sum_k \log(1 - \tilde{E}_{i,j,k}) \tag{3.11}$$

$$\mathcal{L}_N = -\frac{1}{n} \sum N^T \log(\tilde{N}), \tag{3.12}$$

Since molecular graphs are sparse, we found that such separate normalisations were helpful for the training.

3.4.3 Conditional Generation and Optimisation

Model overview. Given the entangled latent code z for a given input molecular graph, we create a conditioned input (z, y) by augmenting z with a set of structured attributes y - the target properties of interest, such as physico-chemical property. The conditional generator is then trained on the combined reconstruction and property loss. At the end of a successful training we expect the decoder to generate samples that have the properties specified in y and to be *similar* (in terms of information contained in z) to the original query molecular graph (encoded as z). To do so we choose a mutual information maximization approach (detailed in section 3.8) that involves the use of discriminators that assess the properties $\tilde{y}$ of the generated samples and their feedback is used to guide the learning of the generator.

Discriminator Pre-Training In this phase we pre-train a discriminator to assess the property y of a generated sample so that we can backpropagate its feedback to the generator (the discriminator can be trained on another dataset and we can have several discriminators for several attributes of interest). In order to have informative gradients in the early stages of the training we have trained the discriminator on continuous approximations of the discrete training graphs (details insection **??**) so that our objective becomes:

$$\mathcal{L}_{dis} = \mathbb{E}_{(x,y)\sim\tilde{p}_{data}(x,y)}[-\log Q(y|x)], \tag{3.13}$$

where the graphs sampled from $\tilde{p}_{data}(x)$ are the probabilistic approximations of the discrete graphs from the training distribution $p_{data}(x)$.

The next step is to incorporate the feedback signal of the trained discriminator in order to formulate the property attribute constraint. The training is decomposed in two phases in which we learn to reconstruct graphs of the dataset (**MLE** phase) and to modify chemical attributes (**Variational MI maximization** phase).

Encoder Learning. The encoder is updated only during the reconstruction phase where we sample attributes y from the true posterior. The encoder loss is a linear combination of the molecular graph reconstruction ($\mathcal{L}_{rec}$) and the property reconstruction ($\mathcal{L}_{prop}$). The total encoder loss is:

$$\mathcal{L}_{enc} = \mathcal{L}_{rec} + \beta \mathcal{L}_{prop}. \tag{3.14}$$

where $\mathcal{L}_{rec} = \mathbb{E}_{(x,y) \sim p_{data}(x,y), z \sim E(z|x)}[-\log p_{gen}(x|z,y)]$ (using the log-likelihood estimates in **(7)**) and $\mathcal{L}_{prop} = \mathbb{E}_{(x,y) \sim p_{data}(x,y), z \sim E(z|x), x' \sim p_{gen}(x|z,y)}[-\log Q(y|x')]$. With $\beta \in [0,1]$ a hyperparameter of the model.

Generator Learning. The generator is updated in both reconstruction and conditional phases. In the **MLE** phase the generator is trained with same loss $\mathcal{L}_{enc}$ as the encoder so that it is pushed towards generating realistic molecular graphs. In the **MI maximization** phase we sample the attributes from a prior $p(y)$ s.t. we minimize the following objective: $\mathcal{L}_{cond} = \mathbb{E}_{x \sim p_{data}(x), y \sim p(y) z \sim E(z|x), x' \sim p_{gen}(x|z,y)}[-\log Q(y|x')]$,

$$\mathcal{L}_{gen} = \mathcal{L}_{rec} + \alpha \mathcal{L}_{cond} + \beta \mathcal{L}_{prop}. \tag{3.15}$$

where $\beta, \alpha \in [0,1]$ are hyperparameters of the model.

In this phase the only optimisation signal comes from the trained discriminator. Since there are no *realism* constraint specified in our model (see [Jin et al., 2018, You et al., 2018a]), there is a risk of "falling off the manifold". A possible solution to mitigate against it is to add a similarity discriminator trained to distinguish between the real probabilistic graph and the generated ones - so that when trying to satisfy the attribute constraint the generator is forced to produce valid molecular graphs. We leave this for future work.

3.5 Experiments

To compare with recent results in constrained molecular graph optimization [Jin et al., 2018, You et al., 2018a], we present the following experiments :

– **Molecular Graph Reconstruction**: We test the autoencoder framework on the task of reconstructing input molecules from their latent representations.
– **Conditional Generation**: We test our conditional generation framework on the task of generating novel molecules that satisfy a given input property.

Method	Reconstruction Accuracy
JT-VAE [Jin et al., 2018]	76.7
JT-AE (without stereochemistry)	69.9
DEFactor	**89.2**

Table 3.1 – Molecular graph reconstruction task. We compare the performance of our decoder in the molecular reconstruction task with the JT-VAE. The results for JT-VAE result is taken from Jin et al. [2018].. The JT-AE refers to an adapted version of the original model using the same parameters. It is however deterministic, and like DEFactor does not evaluate stereochemistry.

Here, we are interested in the octanol-water partition coefficient (LogP) optimization used as benchmark in [Kusner et al., 2017, Jin et al., 2018, You et al., 2018a].

– **Constrained Property Optimization**: Finally, we test our conditional autoencoder on the task of modifying a given molecule to improve a specified property, while constraining the degree of deviation from the original molecule. Again we use the LogP benchmark for the experiment.

Finally in the following section we use the 250K subset of the ZINC [Irwin et al., 2012] dataset, released byKusner et al. [2017], along with their given train and test splits.

Molecular graph reconstruction: In this task we evaluate the exact reconstruction error from encoding and decoding a given molecular graph from the test set. We report in Table 3.1 the ratio of exactly reconstructed graphs, where we see that the our autoencoder outperforms the JT-VAE [Jin et al., 2018] which has the current state-of-the-art performance in this task. In section 3.8.3 we report the reconstruction ratio as a function of the molecule size (number of heavy atoms).

Conditional Generation: In this task we evaluate the conditional generation formulation described in Section 3.3. For a given molecule m with an observed property value y, the goal here is to modify the molecule to generate a new molecule with the given target property value ; (m^*, y^*). New molecules are generated by conditioning the decoder on $(z; y^*)$, where z is the latent code for m. The decoded

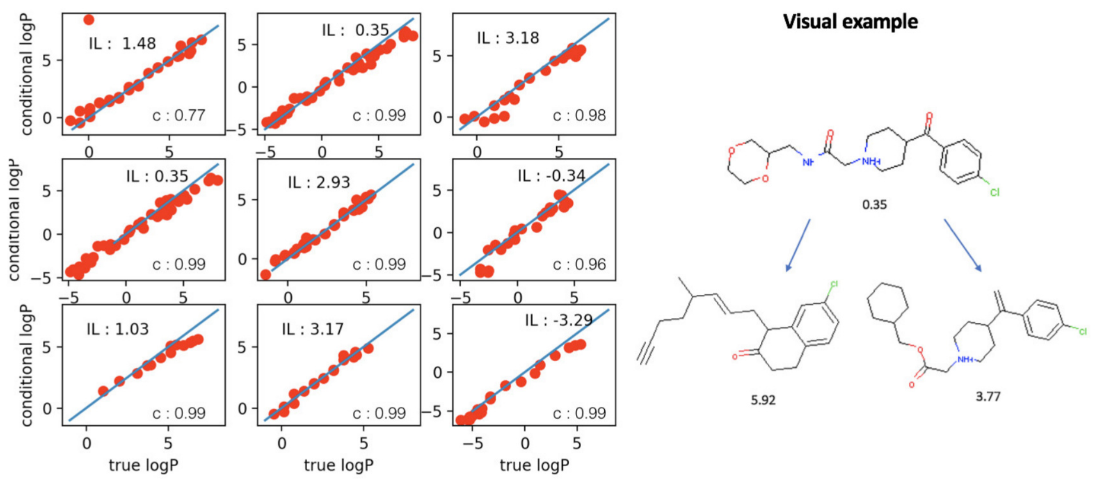


Figure 3.2 – Conditional generation: The initial LogP value of the query molecule is specified as *IL* and the Pearson correlation coefficient is specified as *c*. We report on the y-axis the conditional value given as input and on the x-axis the true LogP of the generated graph when translated back into molecule. For each molecule we sample uniformly around the observed LogP value and report the LogP values for the decoded graphs corresponding to valid molecules.

new molecule m^*, is ideally best suited to satisfy the target property. This is evaluated by comparing the property value of the new molecule with the target property value. A generator that performs well at this task will produce predicted molecules with property values that are close to the target. In these experiments, LogP was chosen as the desired property, and we use RDKIT [RDKit, online] to calculate the LogP values of generated molecules.

The scatter plots in Figure 3.2 give for a selected set of test molecules, the correlation of target property values against the evaluated property value of the correctly decoded molecules. Here, for each molecule the target set is defined by uniformly sampling around the observed property value for the molecule.

Constrained Property Optimization: In this section we follow the evaluation methodology outlined by Jin et al. [2018], You et al. [2018a], and evaluate our model in a constrained molecule property optimization. In contrast to Jin et al. [2018], due to the conditional formulation, does not need retraining for the optimisation task.

Given the 800 molecules with the lowest penalized LogP[1] property scores from

1. The penalized logP is octanol-water partition coefficient (logP) penalized by the synthetic accessibility (SA) score and the number of long cycles, see [Jin et al., 2018]

δ	JT-VAE			GCPN			DEFactor		
	Imp.	Sim.	Suc.	Imp.	Sim.	Suc.	Imp.	Sim.	Suc.
0.0	1.91± 2.04	0.28±0.15	97.5%	4.20±1.28	**0.32±0.12**	**100%**	**6.62±2.50**	0.20±0.16	91.5%
0.2	1.68± 1.85	0.33±0.13	97.1%	4.12±1.19	**0.34±0.11**	**100%**	**5.55±2.31**	0.31±0.12	90.8%
0.4	0.84± 1.45	**0.51±0.10**	83.6%	2.49±1.30	0.47±0.08	**100%**	**3.41±1.8**	0.49±0.09	85.9%
0.6	0.21± 0.71	0.69±0.06	46.4%	0.79±0.63	0.68±0.08	**100%**	**1.55±1.19**	**0.69±0.06**	72.6%

Table 3.2 – Constrained penalized LogP maximisation task: each row gives a different threshold similarity constraint δ and columns are for improvements (Imp.), similarity to the original query (Sim.), and the success rate (Suc.). Values for other models are taken from You et al. [2018a].

the test set, we evaluate the decoder by providing pairs of (z, y^*) with increasing property scores, and among the valid decoded graphs we compute:

– Their similarity scores (Sim.) to the encoded target molecule (called the prototype) ;
– Their penalized LogP scores. Note that in this setting the conditioning property values (y^*) are the unpenalized LogP scores. However, to evaluate the model we compute the penalized LogP scores to assess the model's ability to decode synthetically accessible molecules.
– While varying the similarity threshold values (δ), we compute the success rate (Suc.) for all 800 molecules. This measures how often we get a novel molecule with an improved penalized LogP score.
– Finally, for different similarity thresholds, for successfully decoded molecules, we report the average improvements (Imp.) and the similarity (Sim.) for the molecule that is most improved. We compare our results with Jin et al. [2018], You et al. [2018a].

The final results are reported in Table 3.2. As can be seen, although slightly behind GCPN [You et al., 2018a] w.r.t. success rates (Suc.), DEFactor significantly outperforms other models in terms of improvements (Imp.) achieved (by between 1.3× and 1.95× for thresholds 0.2 and 0.6 respectively, with respect to the next best model GCPN).

3.6 Future work

In this paper, we have presented a new way of modelling and generating graphs in a conditional optimisation setting such that the final graph being fully differentiable w.r.t to the model parameters. We believe that our *DEFactor* model will contribute to understanding and building ML-driven applications for de-novo drug design or generation of molecules with optimal properties, without resorting to methods that do not directly optimise the desired properties.

Note that a drawback of our model is that it uses an MLE training process which forces us to either fix the ordering of nodes or to perform a computationally expensive graph matching operation to compute the loss. Moreover in our fully deterministic conditional formulation we assume that chemical properties optimisation is a one-to-one mapping but in reality there may exist many suitable way of optimizing a molecule to satisfy one property condition while staying similar to the query molecule. To that extent it could be interesting to augment our model to include the possibility of a one-to-many mapping. Another way of improving the model could also be to include a validity constraint formulated as training a discriminator that discriminates between valid and generated graphs.

3.7 Models Comparison

Model	Inference	Parameters	Constrained	Probabilistic	No Retraining
MolGAN Cao and Kipf [2018]	✗	✗	✗	✓	NA
JT-VAE Jin et al. [2018]	✓	✓	✓	✗	✗
GCPNNYou et al. [2018a]	✗	✓	✓	✗	✓
DEFactor(Ours)	✓	✓	✓	✓	✓

Figure 3.3 – We report here a comparison of the abilities of previous recent models involving molecular graph generation and optimization

We are interested in the following features of the models :

- **Inference** : If the model is equipped or not with an inference network. To encode some target molecule like we do in the conditional setting.

- **Parameter-efficient** : If the number of parameters of the model depends on the graph sizes.
- **Constrained** : If the model is studied in a constrained optimization scenario : namely the case where we want to optimize a property while constraining the degree of deviation from the original molecule.
- **Probabilistic** : If the outptut of the model is a probabilistic graph s.t. it is differentiable w.r.t to the decoder's parameters.
- **No Retraining** : If we need to retrain/fine-tune/perform gradient-ascent each time we want to optimize a novel molecule.

3.8 Conditionnal setting

3.8.1 Graphs continuous approximation

For the pre-training of the discriminators we suggested to train them on continuous approximation of the discrete graphs that *ressembles* the output of the decoder. To that extent we used the trained partial graph autoencoder (used for the teacher forcing at the beginning of the training of the full autoencoder)

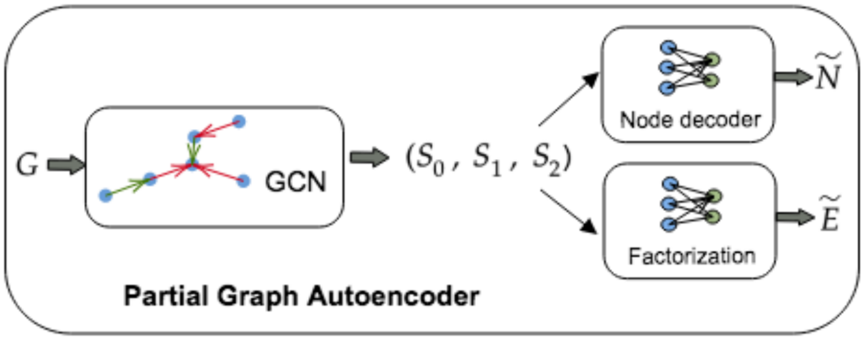


Figure 3.4 – Partial graph Autoencoder used for the pre-training part

3.8.2 Mutual information maximization

For the conditional setting we choose a simple mutual information maximization formulation. The objective is to maximize the MI $I(X;Y)$ between the target property Y and the decoder's output $X = G_\theta(Y)$ under the joint $p_\theta(X,Y)$ defined by the decoder G_θ. In the conditional setting G_θ is also conditioned on the encoded molecule z but for simplicity we treat it as a parameter of the decoder (and thus reason with one target molecule from which we want to modify attributes). We define the MI as:

$$\begin{aligned}
I(y; G_\theta(y)) &= \mathbb{E}_{x\sim G_\theta(y)}[\mathbb{E}_{y\prime\sim p_\theta(y|x)}[\log p_\theta(y\prime|x)]] + H(y)\\
&= \mathbb{E}_{x\sim G_\theta(y)}[D_{KL}(p_\theta(.|x)||Q(.|x))\\
&+ \mathbb{E}_{y\prime\sim p_\theta(y|x)}[\log Q(y\prime|x)]] + H(y)\\
&\geq \mathbb{E}_{x\sim G_\theta(y)}[\mathbb{E}_{y\prime\sim p_\theta(y|x)}[\log Q(y\prime|x)]] + H(y)
\end{aligned}$$

In our conditional setting we pre-trained the discriminators (parametrized by Q in the lower bound derivation) to approximate $p_{data}(y|x)$ which makes the bound tight only when $p_\theta(y_{paired}|x)$ is close to $p_{data}(y|x)$ and this corresponds to a stage where the decoder has maximized the log-likelihood of the data well enough (i.e. when it is able to reconstruct input graphs properly when z and y are paired). Thus, in the conditional setting we are maximizing the following objective:

$$\mathcal{L}_{cond} = \mathbb{E}_{x,y\sim p_{data}(x,y),z\sim E(x),y'\sim p(y)}[\log G_\theta(y,z) + I(y\prime; G_\theta(y\prime, z))]$$

3.8.3 Reconstruction as a function of number of atoms

Notice that as we make use of a simple LSTM to encode a graph representation, there is a risk that for the largest molecules the long term dependencies of the embeddings are not captured well resulting in a bad reconstruction error. We capture this observation in figure 4. One possible amelioration could be to add other at each step other non-sequential aggregation of the embeddings (average pooling of the emebeddings for example) or to make the encoder more powerful by adding some attention mechanisms. We leave those for future work.

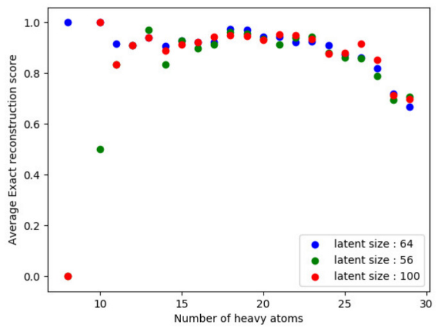
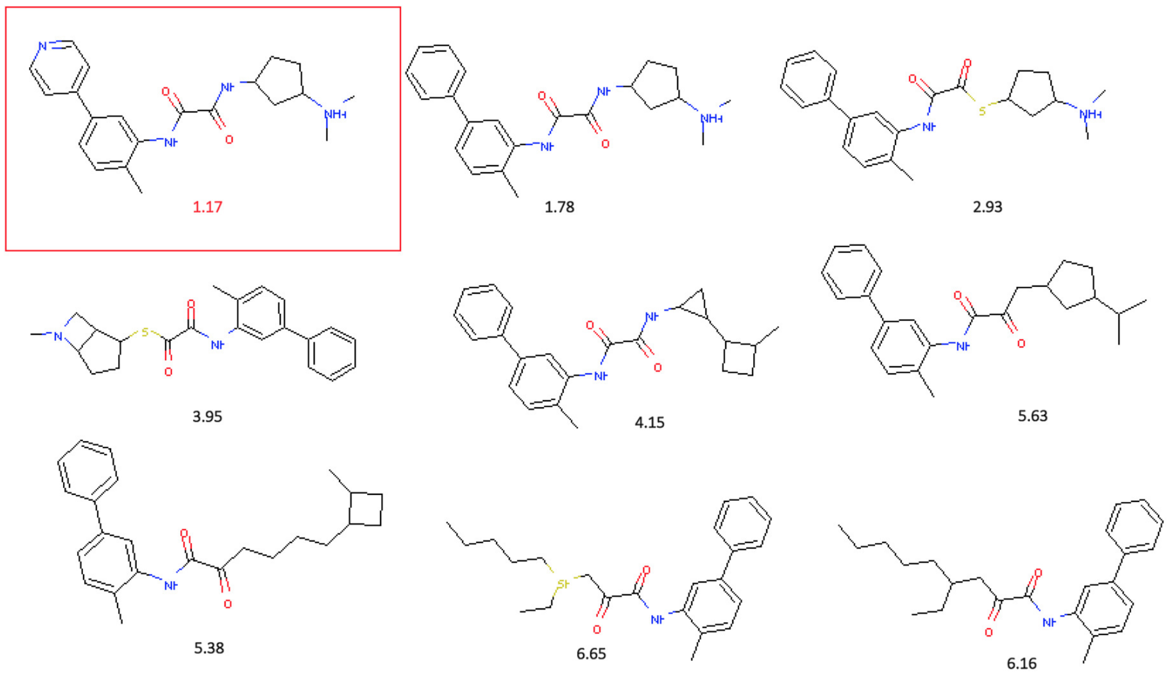


Figure 3.5 – Accuracy score as a function of the number of heavy atoms in the molecule(x axis) for different size of the latent code

3.8.4 Visual similarity samples



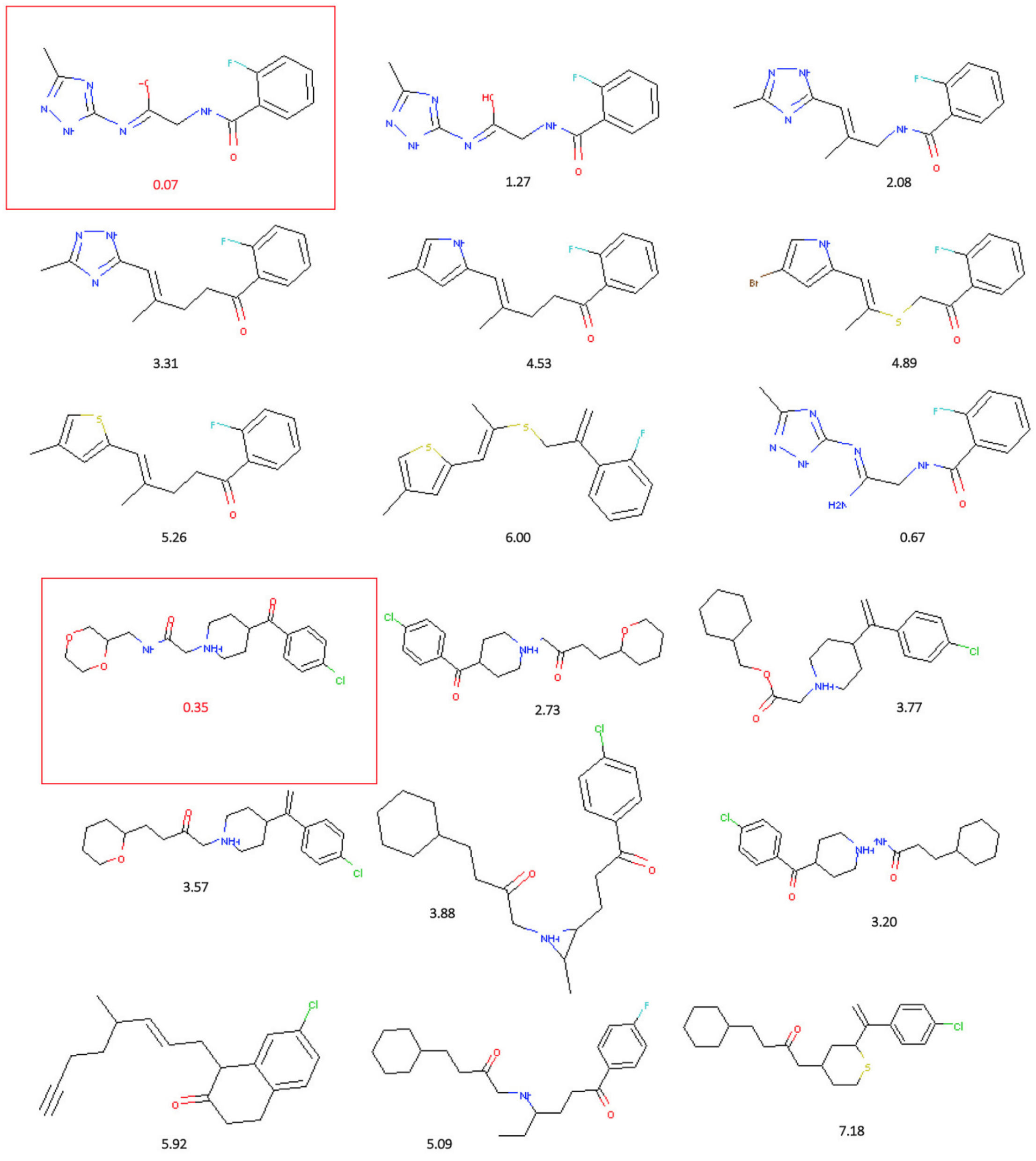


Figure 3.6 – LogP increasing task visual example. The original molecule is circled in red.

4 Conclusion

In this work we first introduced the field of representation learning and the recent tendency to bias recent models towards learning more structured representation. In particular we covered most recent works that focus on a specific kind of structure in representation learning: entity-centric representations. Visual entity-centric representations aim at decomposing a raw visual visual scene in terms of objects. They rely on a latent representation where the latent space is structured as a set of vectors where each vector ideally capture the properties of one entity of the visual scene.

We studied this kind of representations in our first contribution. We posit that temporal cues are important in order to disambiguate entities in a scene and further advocate for a joint learning of the perception and transition modules. We introduce an entity-centric action-conditioned transition model that translates the fact that the agent's actions have sparse effects. We intuitively motivate the need of sparsity in transitions by the physical consideration that the agent's interventions are localized in time and space.

These slot-structured representations are in fact a specific instance a more general kind of structured representations: graphs. We therefore introduced recent progresses made in the field of graph representation learning and generation. Graph generation have in particular proven useful in the field of drug discovery where discovery and generation of small molecules is of crucial importance.

In our second contribution we introduced a model for conditional graph generation that also leverages an entity-centric approach. The idea behind the design of the model is to learn a slot-structured representation of the graph where each slot corresponds to an atom of the molecule. We bias the slots towards containing enough information about their neighbours such that a learned pairwise similarity measure of the slots can retrieve the edge structure of the graph. We demonstrate favourable performance of our model on prototype-based molecular graph conditional generation tasks.

A core problem in machine learning is the one of inductive bias that asks the question of how we can build models that learn the right representations, abstractions, and skills that allow them to generalize to novel and unforseeable scenarios. Inductive bias in deep neural networks can come in many forms: the choice of model architecture, the training objective, and the optimization procedure to name a few. In this thesis we focused on a particular form of inductive bias which is the structure of the representation we use for our input data, and, more specifically entity-centric representations. However, entity-centric approaches to representation learning are still at their infancy. There is a clear surge of interest for entity-centric learning in deep learning and it seems clear that objects hold great potential for enabling more systematic generalisation, building compositional models of the world, and serving as grounding for language and symbolic reasoning. However, despite strong intuitions, a general definition of what constitutes an object is still lacking, and the precise notion of objects remains unknown. Future work will have to focus more on how the entity-centric aspect of representations can be defined and evaluated.

Bibliographie

Renée Baillargeon, Elizabeth Spelke, and Stan Wasserman. Object permanence in five-month-old infants. *Cognition*, 20:191–208, 09 1985. doi: 10.1016/0010-0277(85) 90008-3.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks, 2015.

Yoshua Bengio. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012. URL http://arxiv.org/abs/1206.5538.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):1798–1828, 2013.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Lars Buesing, Theophane Weber, Sebastien Racaniere, S. M. Ali Eslami, Danilo Rezende, David P. Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. Learning and querying fast generative models for reinforcement learning, 2018.

Christopher P. Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation, 2019.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs, 2018.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators, 2017.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Eric Crawford and Joelle Pineau. Exploiting spatial invariance for scalable unsupervised object tracking, 2019.

Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. *CoRR*, abs/1802.08786, 2018.

Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 240–247, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390187. URL http://doi.acm.org/10.1145/1390156.1390187.

S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014a.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014b.

Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hotloo Hao, Jürgen Schmidhuber, and Harri Valpola. Tagger: Deep unsupervised perceptual grouping. *CoRR*, abs/1606.06724, 2016. URL http://arxiv.org/abs/1606.06724.

Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization, 2017.

Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference, 2019.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules, 2016.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997a.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997b. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012. doi: 10.1021/ci3001277. URL https://doi.org/10.1021/ci3001277. PMID: 22587354.

Jindong Jiang, Sepehr Janghorbani, Gerard de Melo, and Sungjin Ahn. Scalor: Generative world models with scalable object representations, 2019.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation, 2018.

Seokho Kang and Kyunghyun Cho. Conditional molecular design with deep generative models, 2018.

Ilyes Khemakhem, Diederik P Kingma, and Aapo Hyvärinen. Variational autoencoders and nonlinear ica: A unifying framework. *arXiv preprint arXiv:1907.04809*, 2019.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016a.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016b.

Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. All else being equal be empowered. In *European Conference on Artificial Life*, pages 744–753. Springer, 2005.

Adam R. Kosiorek, Hyunjik Kim, Ingmar Posner, and Yee Whye Teh. Sequential attend, infer, repeat: Generative modelling of moving objects, 2018.

Navneet Madhu Kumar. Empowerment-driven exploration using mutual information estimation. *arXiv preprint arXiv:1810.05533*, 2018.

Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. URL http://proceedings.mlr.press/v70/kusner17a.html.

Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model, 2018a.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs, 2018b.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Joseph Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference, 2018.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

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, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2016.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de novo design through deep reinforcement learning, 2017.

RDKit, online. RDKit: Open-source cheminformatics. http://www.rdkit.org. [Online ; accessed 11-April-2013].

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter W. Battaglia, and Timothy P. Lillicrap. A simple neural network module for relational reasoning. *CoRR*, abs/1706.01427, 2017. URL http://arxiv.org/abs/1706.01427.

Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks, 2017.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders, 2018.

Elizabeth S. Spelke. Where perceiving ends and thinking begins: The apprehension of objects in infancy. 2013.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions, 2018.

Sjoerd van Steenkiste, Klaus Greff, and Jürgen Schmidhuber. A perspective on objects and systematic generalization in model-based RL. *CoRR*, abs/1906.01035, 2019. URL http://arxiv.org/abs/1906.01035.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Rishi Veerapaneni, John D. Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B. Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning, 2019.

Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P. Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration, 2019.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks, 1989.

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation, 2018a.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models, 2018b.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1609.html#YuZWY16.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Relational deep reinforcement learning, 2018.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics, 34:13, 457-466, 2018, 2018.