UC Irvine
UC Irvine Electronic Theses and Dissertations

Title
Towards Data Efficiency on Model-Based Reinforcement Learning: Model Confidence and Representation

Permalink
https://escholarship.org/uc/item/0121p992

Author
Nagata, Takashi

Publication Date
2022

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Towards Data Efficiency on Model-Based Reinforcement Learning: Model Confidence and Representation

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Takashi Nagata

Dissertation Committee:
Professor Emre O. Neftci, Chair
Professor Nikil D. Dutt
Professor Roy Fox

2022

Chapter 3 © 2022 IEEE
All other materials © 2022 Takashi Nagata

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor Professor Emre Neftci for his guidance, continuous support, and endless patience during my PhD journey. I also sincerely thank my dissertation committee, Professor Nikil Dutt and Professor Roy Fox. I would also like to thank Professor Jeff Krichmar and Professor Aaron Bornstein for serving on my PhD candidacy committee and for constructive feedback on my research ideas. In addition, I would like to thank Dr. Tsutomu Kumazawa for mentoring me.

My dissertation is based upon work supported by Intel Corporation and the National Science Foundation (NSF) under grant 1652159. I sincerely thank these funding sources that made my PhD studies possible.

I also owe my thanks to the following people and/or organizations.

The Neuromorphic Machine Intelligence Lab (NMI). Dan Barsever, Tim Hin Wai Lui, Kenneth Stewart, Nick Alonso, and Yue Yin. I would express my special appreciation to Jinwei Xing, my research fellow and co-author, separately. I am lucky to have you in my PhD journey. Your positive attitude and grit always inspired and encouraged me.

Old NMI folks. Dr. Georgios Detorakis, Travis Bartley, Andrew Hansen, Armaan Saini, Dr. Jordan Rashid, and Roman Parise. I enjoyed working with you. I miss our random chatting a lot.

Past and current members of the Cognitive Anteater Robotics Laboratory (CARL) and Dutt Research Group.

Amazon - Global Learning & Development and Visual Search & AR for my summer internships.

Colleagues at Hewlett-Packard Japan and Amazon Web Services Japan for their friendship and inspiration.

Other grad students at UCI for making my graduate school life much fun.

Professor Munehiro Takimoto and Professor Yasushi Kambayashi, my former advisor and co-author, respectively for always welcoming and encouraging me.

My family for always caring me and supporting my decision.

Finally, I would like to thank my wife and best friend, Yuki Nagata, for always being by my side, encouraging me, and sharing the journey. I am truly blessed to have you in my life.

Portions of Chapter 3 are reprinted with the permission from Takashi Nagata, Jinwei Xing, Tsutomu Kumazawa, and Emre Neftci. *Uncertainty Aware Model Integration on Reinforcement Learning.* To appear in the 2022 IEEE International Joint Conference on Neural

Networks (IJCNN).

VITA

Takashi Nagata

EDUCATION

Doctor of Philosophy in Computer Science	**2022**
University of California, Irvine	*Irvine, California*
Master of Science in Information Science	**2010**
Tokyo University of Science	*Tokyo, Japan*
Bachelor of Science in Information Science	**2008**
Tokyo University of Science	*Tokyo, Japan*

RESEARCH EXPERIENCE

Graduate Student Researcher	**2017–2022**
University of California, Irvine	*Irvine, California*

TEACHING EXPERIENCE

TA for CS178 Machine Learning & Data Mining	**Fall 2019/2020**
University of California, Irvine	*Irvine, California*
TA for ICS32 Programming with Software Libraries	**Fall 2018/Winter 2019**
University of California, Irvine	*Irvine, California*
Reader for CS132 Computer Networks	**Spring 2018**
University of California, Irvine	*Irvine, California*
Reader for ICS45J Programming in Java	**Winter 2018**
University of California, Irvine	*Irvine, California*
Reader for CS244 Introduction to Embedded Systems	**Fall 2017**
University of California, Irvine	*Irvine, California*

Book Chapters

- Nagata T., Takimoto M., Kambayashi Y. (2013) Cooperatively Searching Objects Based on Mobile Agents. In: Nguyen N.T. (eds) Transactions on Computational Collective Intelligence XI. Lecture Notes in Computer Science, vol 8065. Springer, Berlin, Heidelberg.

Conference Papers

- Nagata, T., Xing, J., Kumazawa, T., & Neftci, E.O. (2022). Uncertainty Aware Model Integration on Reinforcement Learning. IEEE International Joint Conference on Neural Networks (IJCNN),Padua, Italy.
- Xing, J., Nagata, T., Chen, K., Zou, X., Neftci, E.O., & Krichmar, J.L. (2021). Domain Adaptation In Reinforcement Learning Via Latent Unified State Representation. Proceedings of the AAAI Conference on Artificial Intelligence, 35(12):10452–10459, May 2021.
- Nagata, T., Takimoto, M., & Kambayashi, Y. (2009). Suppressing the Total Costs of Executing Tasks Using Mobile Agents. 2009 42nd Hawaii International Conference on System Sciences, 1-10.

ABSTRACT OF THE DISSERTATION

Towards Data Efficiency on Model-Based Reinforcement Learning:
Model Confidence and Representation

By

Takashi Nagata

Doctor of Philosophy in Computer Science

University of California, Irvine, 2022

Professor Emre O. Neftci, Chair

Humans can develop their internal model of the external world and use it for decision making. Reinforcement Learning (RL) is an optimization method to maximize the expected total reward on sequential decision-making problems. RL is divided into two approaches: a model-free approach directly learns optimal behaviors given the data, whereas a model-based approach builds the model of the environment and utilizes it for decision making. Although the Model-based approach is intuitive and appealing, it has several challenges to overcome, such as the model's inaccuracy or determining the effective model architecture. These challenges limit practical applications of the model-based RL. In this thesis, we first discuss how to integrate the model uncertainty into model-based RL and propose methods to use them. We apply the Monte Carlo dropout technique to the state transition model to estimate uncertainty. Our approach enables the algorithm to use model simulations effectively by filtering the simulation given the model uncertainty. We show that this scheme achieves speed-up of agents' policy learning in contrast to conventional ways to use model simulations without considering the uncertainty. In model-based RL, model architecture is another critical factor to consider. In this context, we then investigate variants of the Variational Autoencoder (VAE) and Generative Adversarial Networks (GANs), and then evaluate the combination of them, VAE/GAN, as the agents' state representation learning (SRL) methods. Acquiring a compact and efficient representation of the world for control is essential to help model-based RL agents overcome the curse of dimensionality. We evaluate the VAE/GAN architecture qualitatively and quantitatively, and show that the RL agent that learns a policy over the VAE/GAN embedding outperforms the one with the VAE embedding. We further discuss VAE/GAN and disentanglement. Taken together, the presented method and models provide the RL agent architecture to achieve better sample efficiency.

Chapter 1

Introduction

"I seem to have been only like a boy playing on the seashore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me."

— Isaac Newton

Some animals are believed to learn helpful representations and use them to determine the action [Tolman, 1948, Fast and Blaisdell, 2011, Fast et al., 2016, Blaisdell, 2019]. Humans develop the mental model of the world, which is the reflections of one's understanding of the world, called the "mental model" [Ha and Schmidhuber, 2018], and cognitive models in their brains throughout their experiences and can understand the world or plan behaviors using this model [Tani, 2016, Lake et al., 2016]. The model refers to any representation humans can consult with to determine their actions or predict how the environment reacts to their actions, and thus how rewarding the particular action is in the situation. This enables humans to leverage our knowledge from past experiences or even the knowledge we have not directly experienced and make complicated decision-making efficiently. *Artificial Intelligence* (AI) agent research aims to develop machines capable of thinking and acting like humans do. We have seen significant advancements in AI agent developments in recent years. Many practical applications, such as face recognition, automatic translation, or even autonomous driving systems in certain use cases have been deployed into our society and benefited people. Nevertheless, artificial systems require massive computation, are typically good at a particular task, and are not easily transferable to other domains. The brain's efficiency remains out of reach in many ways. Machine learning is one of the sub-fields of the broad AI research literature. It is typically divided into three different paradigms: supervised learning, unsupervised learning, and reinforcement learning. Although outstanding contributions have been made by the supervised learning approach in the past decade, supervised learning generally requires labeling and is notoriously data-intensive. It requires a high cost to prepare the data with correct labels, which is typically achieved by human efforts that are sometimes done with crowdsourcing. Unsupervised learning aims to find patterns in the data and cluster them, to acquire distributions from which the data is generated, or learns to project the input data onto another space, which is typically less dimensions compared to the original input space. Instead of learning from the data collected beforehand, reinforcement learning provides a unique way to train an artificial agent, either virtual or physical, to solve tasks by collecting the data from interactions with the environment. Although it avoids human labeling costs, the agents need to collect a large amount of data, which consists of a set: an state of the environment, agent's action, a scalar reward, and a next state. The data collection is achieved by agent's interaction with the environment, and consuming time and money. This characteristic is one of the critical challenges of reinforcement learning: sample inefficiency and the common motivation underlying this dissertation is how to improve sample efficiency in RL. In general, to improve sample efficiency, there are several approaches. The first approach is to think about more sophisticated exploration strategies, including intrinsic motivation [Burda et al., 2018a, Baranes and Oudeyer, 2013], where the agent gets an incentive to explore states with high prediction errors. Typically this is achieved by assigning the reward to do so. Alternatively, noise injection (for problems with continuous state space) is another approach that forcefully perturbs the agent's action [Fortunato et al., 2017, Plappert et al., 2017] instead of the simple ϵ-greedy exploration. Second, leveraging a teacher signal or supervision, which enables agents to learn a policy adaptively, such as curriculum learning [Bengio et al., 2009] where the agent starts learning in a simple task and gradually advances to the more complicated final task, or imitation learning where the expert knowledge is given and the agent learns the initial policy in supervised way[Ross et al., 2011]. Provide the agent with human guidance, which is called Human-in-the-loop RL, is one of the possible solutions as well. The third group, in which this dissertation belongs, focuses on the environment and ways to perceive the environment, including model-based RL and representation learning.

Model-based reinforcement learning (RL) is known to be sample efficient, in combination with function approximation techniques to acquire approximated models of environments. However, it is practically impossible in many scenarios to learn a fully accurate model of the world. Thus, consideration of the uncertainty of the model is a critical problem to avoid catastrophic consequences, such as the agent behaving dangerously in states where it gets overly confident due to the lack of experience. Yet, a clear solution is still missed. Although humans can cast doubt as a reaction to their mental simulations [Hamrick et al., 2015], it is challenging to build such functionality into the machines. In this dissertation, we first develop a method to bridge the gap between these two. Our method is based on Dyna architecture, which combines model-free learning with model learning, and uses the learned model to speed-up policy learning by model-based simulations. The core contribution of our approach is that the method uses Monte Carlo dropout (MC-dropout) [Gal and Ghahramani, 2016] in the agent's state transition model to calculate prediction variance and use it to measure the uncertainty.

We then focus on the representation of the environment. In Chapter 3, we tackle the sample efficiency problem from the model-based RL perspective and propose a Dyna architecture with a mechanism to mitigate this negative effect by using the confidence of the state transition model, which is calculated using MC-dropout. Recently, the rise of deep learning motivates the combination of RL agents with larger-scale approximate models using *Neural Networks* (NN). In environments, sensory input to the agent or observations, such as a sequence of RGB video frames, corresponds to state space. Even for reinforcement learning tasks with such huge state space, policies with deep network architecture show successful results that score higher than humans in video games or robotic manipulations. However, again, this comes with the cost of a sample inefficiency: the massive amount of agents' interaction with the environment. For tasks with such a huge state space, getting benefits by applying model-based RL techniques such as Dyna is difficult even with NNs, unlike toy tasks with a smaller state space (e.g., grid world search). We can imagine situations in which even minor changes of some pixels are recognized as different states. In that case, it takes a very long time to acquire enough data to train the model due to the size of the state space. In such cases, the impact of incorporating information from inaccurate models becomes more apparent as many states are unseen for the model before generating a simulation.

Chapter 4 focuses on the representation learning aspect to tackle the above challenge. Representation learning methods using a dimensional compression technique such as an Autoencoder have been widely studied [Hinton and Salakhutdinov, 2006, Vincent et al., 2008, 2010]. We propose a method to encode images into low-dimensional hidden vectors as a preliminary step to model learning instead of directly handling the input image sequences. Specifically, we investigate the disentanglement when compressing dimensions and propose a method to obtain more disentangled representations, which is beneficial for the agents that use the encoded vectors to learn more efficiently. In this study, we test the hypothesis of the effectiveness of disentanglement by comparing the performance of reinforcement learning agents trained with the codes generated by a trained dimensional compressor. Although recent advances of available computational resources enables us to utilize deep neural networks [LeCun et al., 2015], which can learn helpful representations of high-dimensional inputs, it is not efficient or practical for RL agents to take a raw input in large state space such as RGB image frame. For this reason, representation learning [Bengio et al., 2012, Lesort et al., 2018], which provides a way to encode states into compact representations, has been a key to handling rich visual inputs effectively. In Chapter 4, we introduce VAE/GAN, which combines *Variational Autoencoder* (VAE) [Kingma and Welling, 2013] and *Generative Adversarial Networks* (GAN) [Goodfellow et al., 2014], as a vision module for an RL agent. The scopes of these projects are shown in the Figure 1.1 We discuss potential future directions based on our findings and the current state-of-the-art in Chapter 5.

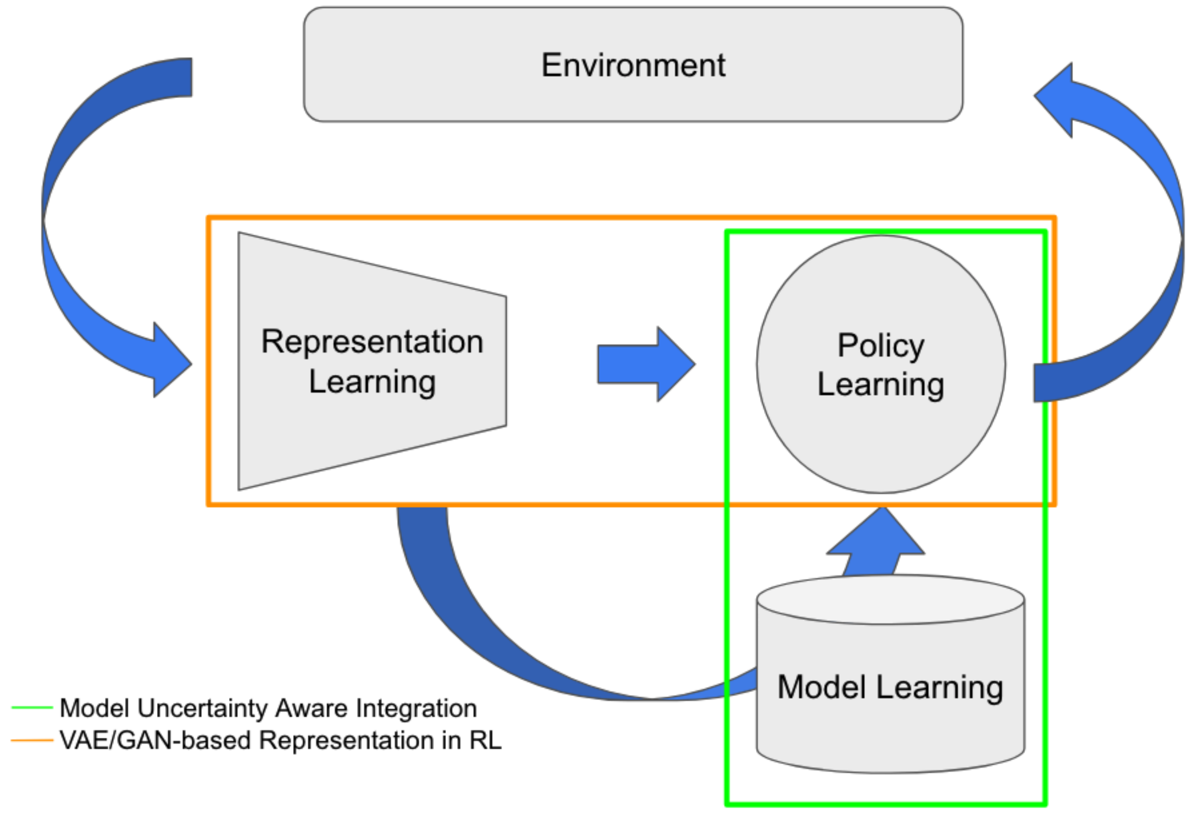
Environment
Representation Learning
Policy Learning
Model Learning
Model Uncertainty Aware Integration
VAE/GAN-based Representation in RL

Figure 1.1: Project scopes. The first project is a model-based RL, where the models are learned and used to optimize agents' behavior policy. In particular, we investigate how to handle uncertainty in models. The second project is about representation learning. The method representing the input data is a critical problem in machine learning and can significantly impact the RL agents' performance. We propose using the architecture consisting of Variational Autoencoder and Generative Adversarial Network, called VAE/GAN, and showing its effectiveness from a task performance and disentanglement point of view. The research projects share the motivation to improve sample efficiency.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning which considers a sequential decision-making problem [Sutton and Barto, 2018]. Unlike in supervised learning, there is no explicit target or label in RL problems. Instead, an entity that makes a decision called an agent learns optimal behavior to achieve a goal. The problem consists of an agent, an environment, states, and reward. The mathematical formulation called Markov Decision Process (MDP) is introduced in section 2.1.2 to formalize RL problems. The agent takes action at each time step in a state, observes a new state, and receives a reward, given a combination of the state and the action. The agent can update its behavior based on the acquired reward to achieve the optimal action sequences. In this context, 'optimal' corresponds to maximizing its cumulative reward throughout the task. There are two different formalizations of tasks regarding the length of a task: one is called an episodic or finite horizon task and the other is continuous or infinite horizon task. In episodic tasks, there is a set of states where the task terminates, called terminal states. For example, in a video game, the agent hit by an enemy

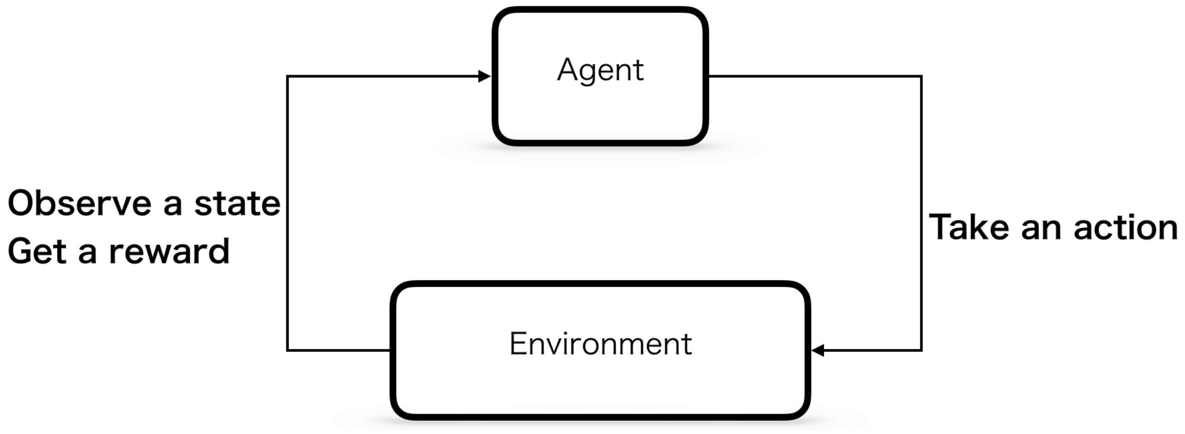

Figure 2.1: Reinforcement Learning Sequence. The agent takes an action based on the observed state. The environment state changes based on the action (e.g., the agent's position, joint angles, velocity, or a view changes), which is a new observation for the agent, and the environment also provides the agent with a scalar reward. The agent decides the next action given a new observation, and this process continues. Rewards give the agent cues on whether a taken action was good or not, and the agent tries to learn behaviors, which maximizes the total cumulative rewards. The figure is modified from [Sutton and Barto, 2018].

or reaches a goal terminates the task, returning it to an initial state. On the other hand, continuous or infinite horizon tasks, which do not have terminal states, last forever. In this case, we consider 'discounting' future rewards, which makes the task mathematically easy to handle. RL works in different kinds of applications such as games [Mnih et al., 2015, Silver et al., 2016, 2017, Schrittwieser et al., 2020], optimal control [Lazic et al., 2018], robotics [OpenAI et al., 2018, Rudin et al., 2021], recommender systems [Chen et al., 2019], transportation systems [Wang et al., 2018], or financial applications such as portfolio optimization [Wang et al., 2021]. Optimal control problems and RL have strong connections. In fact, the terms in RL correspond to the terms in control: agent/controller, environment/system, and action/control signal.

As shown in Figure 2.2, there are two high-level classifications of RL algorithms: model-free RL and model-based RL. In a model-free setting, the agent directly interacts with an environment to collect data. Further, the model-free family can be divided into two approaches: 1. Value-based methods, such as Q-learning [Watkins and Dayan, 1992], and

2. Policy-based methods such as REINFORCE [Williams, 1992]. The value-based approach learns the values of states or the value of actions at each state and determines the behavior based on them. On the other hand, a policy-based agent learns the policy which determines the action taken in each state. The *Actor-Critic method* combines the value function and policy optimization approaches. The actor optimizes its policy while a value estimation from a critic assesses the actor's actions. Model-based RL, another one of the two classifications of RL, considers state dynamics of the environment using a model to select actions. Model-based RL is one of the main topics of this dissertation, and we discuss the details of the model-based RL in later sections.

2.1.1 Deep Reinforcement Learning

The impressive success of the deep convolutional neural network on the ImageNet LSVRC-2010 task ignited a surge of "deep learning" research in many areas, with the recent extremely powerful computational resources such as GPUs[Krizhevsky et al., 2012]. The layers of neurons acquire abstracted and localized representations of an image [LeCun et al., 2015]. In reinforcement learning research, deep networks have also been applied and showed great success [Silver et al., 2016, Berner et al., 2019, Vinyals et al., 2019]. Such networks with several layers play a role in approximating several factors in RL, such as 1. an agent's policy, 2. the value of states, or 3. environmental dynamics. It was challenging to train an agent directly in high-dimensional input spaces like sequential image frames. However, deep convolutional networks showed their ability to learn a helpful representation from input data directly and became a key factor that led to the great success of the Deep Q-network (DQN) in the Atari retro game suite [Bellemare et al., 2012, Mnih et al., 2015]. These games have large state spaces (e.g., an RGB image with the size of $84 \times 84 \times 3$), and it used to be intractable or not practical to train the agent with a limited capacity. Further, an agent called AlphaGo, that plays the game of Go[Silver et al., 2016] drew wide attention. AlphaGo uses

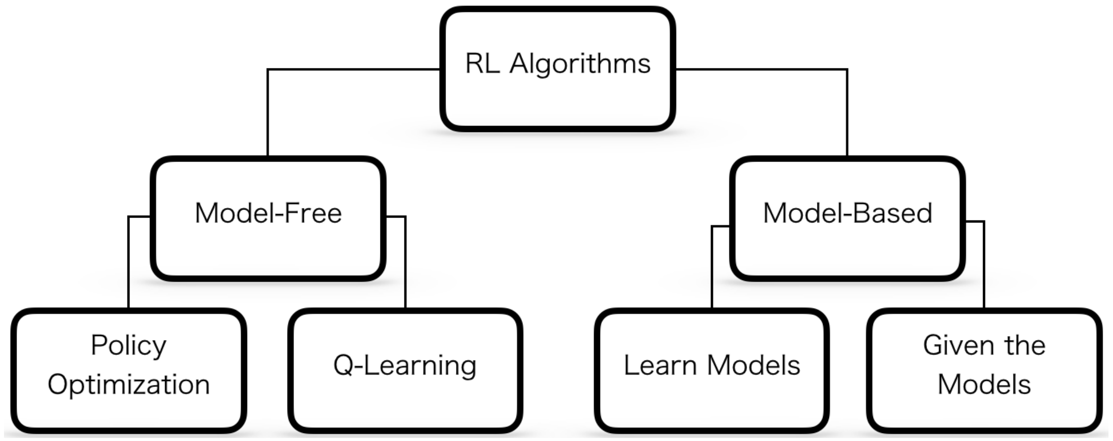

Figure 2.2: RL Taxonomy. At the top level, the tree branches based on whether the method is model-free or model-based. The model-free family consists of Q-learning and the policy gradient method. There are two distinctions in the model-based branch: whether the model is given or the model needs to be learned.

two deep neural networks. The first is a policy network to determine moves. The training process of this network consists of two phases, the first one is a supervised learning using expert human moves, and then the network is trained in games against itself called self-play. The second network is a value network, which evaluates board positions. Several handy benchmarking platforms, including classic games, retro video games, or physical simulation environments, became available and boosted RL developments in recent years [Bellemare et al., 2012, Brockman et al., 2016, Tassa et al., 2018, Todorov et al., 2012].

DQN has its basis in Q-learning [Watkins and Dayan, 1992], which belongs to one of the two families of RL algorithms: model-free RL. Model-free methods learn values of input space directly from trial and error to determine the optimal action sequence instead of considering how the system's dynamics work. Model-free methods scale to large applications well. The actor-critic methods has been also developed and succeeded in Deep RL in the benchmarking environment mentioned above [Mnih et al., 2016].

2.1.2 Markov Decision Process

The Markov Decision Process (MDP) is a formalization of sequential decision making. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is a set of states (continuous or discrete), which is a quantitative representation of the environment, $\mathcal{A}$ is a set of actions (continuous or discrete), $\mathcal{P}$ is a dynamics function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, 1]$, and $\mathcal{R}$ is the reward $\mathcal{R} \in \mathbb{R}$. The agent interacts with the environment by taking an action $A_t \in \mathcal{A}$ at time step t in a state $S_t \in \mathcal{S}$. $\mathcal{A}$ and $\mathcal{S}$ are either discrete or continuous and can be vectors depending on environments and tasks. As a consequence of the selected action, the agent gets feedback from the environment in the form of a new observation S_{t+1} and a scalar reward $R_t \in \mathcal{R}$. The state transition over an MDP given a selected action is determined by a function depending on the environment $\mathcal{P}$. The MDP assumes that all the past information is included in the current state, and a transition to the next state s' only depends on the current state s and an action to take a, which is called the *Markov property*. Thus, the optimal decision is based only on the current state and not on the agent's interactions in the past.

2.1.3 Rewards and Episodes

The reward provides the agent with the value of the action taken and works as a feedback signal to the agent. The goal is to maximize its cumulative reward from time step t and defined as follows:

$$G_t = R_t + \gamma R_{t+1} + ... + \gamma^K R_{t+K} = \sum_{k=0}^{K} \gamma^k R_{t+k} \tag{2.1}$$

where γ is a discount factor and $0 \leq \gamma \leq 1$ and K is the total number of steps. If the total number of steps is finite as in Eq. 2.1, it is called an *episodic task*, on the other hand, if K is infinite, it is called *continuing task*. If $\gamma = 0$, the agent only considers maximizing the immediate one-step reward. On the other hand, if $\gamma = 1$, it is possible that the agent exploits a small reward endlessly, which does not lead to the terminal states. Eq. 2.1 holds in either case when $0 < \gamma < 1$ because the power of γ eventually gets close to zero, and the sum of discounted rewards converges. The expected value of a one-step immediate reward is defined as

$$\mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r | s, a) \tag{2.2}$$

where p is a state transition probability from state s to $s' \in \mathcal{S}$ given the action a: $p(s_{t+1}|s_t, a_t)$ and r is the observed reward $r \in \mathcal{R}$. The state-value function representing the value of each state (expected cumulative reward from the given state) is defined as $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k R_{t+k+1}|S_t = s]$, for all $s \in S$. Similarly, the action-value function, which represents the value of taking an action a at a state s, is defined as $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$. Given the dynamics of the environment and the reward function, the optimal solution is computed by applying the following recursive relationship known as the Bellman equation:

$$\begin{aligned} v_\pi(s) = &\quad \mathbb{E}_\pi[G_t|S_t = s] \\ = &\quad \mathbb{E}_\pi[R_t + \gamma G_{t+1}|S_t = s] \\ = &\quad R_t + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\ = &\quad R_t + v_\pi(s_{t+1}) \end{aligned} \tag{2.3}$$

This recursive relationship holds for q_π as well, and also, it can be written as follows by using the *optimal state-value function* $v_*(s) \doteq \max_\pi v_\pi(s)$:

$$\begin{aligned} q_*(s, a) &\doteq \max_\pi q_\pi(s, a) \\ &= \mathbb{E}[r_t + \gamma v_*(s_{t+1})|s = s_t, a = a_t] \end{aligned} \tag{2.4}$$

Further, we define the *Bellman optimality equation* for v and q as follows:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_a \sum_{s', r} p(s', r|s, a)[r + \gamma v_*(s')] \end{aligned} \tag{2.5}$$

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(s', a')] \\ &= \sum_{s', r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{2.6}$$

Based on the Bellman equation, RL agents iteratively update the value function, and once the iteration converges, the optimal solution is given by greedily traversing the states. Although solving the Bellman optimality equation gives us the optimal policy, it is typically impossible to solve it directly because we do not know the exact dynamics of an environment, or it is computationally hard to look ahead to every possible combination of state and action to the goal. Thus, we need an approximated solution.

2.1.4 Exploration-Exploitation Trade-off

To approximate the values of states or state-action values for each state, the agent needs to try different actions and collect experiences. When selecting actions, the agent faces the dilemma of whether it should exploit the current knowledge or explore something else if it works well. This is an essential question in RL called "exploration-exploitation trade-off", and an effective exploration strategy has been one of the active research themes. Classical approaches include ϵ-greedy, where the agent takes a random action with the probability ϵ, or a random noise injection approach in continuous action spaces to perturb selected actions [Fortunato et al., 2017] or parameter [Plappert et al., 2017] to forcefully have the agent perform random actions. This approach includes some variants such as decaying ϵ-

greedy, where the exploration probability ϵ decay and eventually converges to a very small constant. The agent explores a lot in the early stage of the learning and will be more greedy towards the end of the episodes. Count-based approaches are also proposed [Tang et al., 2017, Bellemare et al., 2016, Ostrovski et al., 2017]. In addition, curiosity (or intrinsic motivation)-driven exploration is also an active study area [Pathak et al., 2017, Burda et al., 2018a, Zheng et al., 2018]. In those cases, agents are built with some internal driver, which leads them to explore. For example, it can be the novelty of observed states, preference for unexplored actions, or entropy. In the projects proposed in Chapters 3 and 4, we use the classical ϵ-greedy approach for our agent's exploration strategy. We further discuss the other strategies in Chapter 5 as a future direction. [Burda et al., 2018b, Ecoffet et al., 2019, Houthooft et al., 2016]

2.1.5 Temporal Difference and Value Optimization

Temporal-difference (TD) is defined as follows using reward and action-values of the current state and the next state:

$$TD = r + \gamma max_{a'} Q(s'a') - Q(s,a) \tag{2.7}$$

This tells us the difference of the estimated value of taking the action a at the state s; $Q(s,a)$, and the summation of the realized reward value r plus the discounted value of the estimated next action-value of taking action a' in state s'; $r + \gamma max_{a'} Q(s'a')$. This difference corresponds to the estimation error, and by using this quantity, Q-learning updates the estimate as

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s'a') - Q(s,a)) \tag{2.8}$$

where α is a step-size. Updating the values based on the Eq. 2.8 while collecting experiences with an exploration strategy eventually leads the agent to convergence if "an infinite number of episodes for each starting state and action" is obtained [Watkins and Dayan, 1992]. There are number of RL algorithms based on Q-learning have been proposed [Hasselt, 2010, Wang et al., 2015, Nair et al., 2015, Kapturowski et al., 2019].

2.1.6 Policy Optimization

Instead of learning the state-value or action-value function introduced above, it is possible to learn the optimal behavior directly. This family learns a policy, $\pi(A_t|S_t)$, which is the probability of A_t while in the state S_t. Agents do not need to look up the values of states but compute the probability distribution of the actions and draw one. We approximate this function with parameters $\boldsymbol{\theta}$, and the function is $\pi_{\boldsymbol{\theta}}$. Our problem is a parameter learning problem given the objective function $J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0)$, which corresponds to the cumulative reward by following the policy π_θ starting from the initial state. The parameter is updated based on the gradient. Thus, the update rule is written as $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} + \alpha \nabla J \boldsymbol{\theta}_{old}$, where α is a step size, and methods based on this approach are called *policy gradient methods*. One of the pioneering works in this approach is the REINFORCE algorithm [Williams, 1992]. Practically, the *policy gradient theorem* provides us with the analytical way to compute the derivative of $J(\theta)$ and the parameter update rule above is written as follows:

$$\boldsymbol{\theta_{t+1}} \doteq \boldsymbol{\theta_t} + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \tag{2.9}$$

where the effect of the policy change on the state distribution needs not be considered [Sutton et al., 2000, Sutton and Barto, 2018]. G_t in the Eq. 2.9 can be $G_t - b(S_t)$, where $b(S_t)$ is called a *baseline*, which can be any function as long as the function does not depend on the selected actions. One way is $G_t - \hat{V}(s_t)$, which calculates the difference between the realized

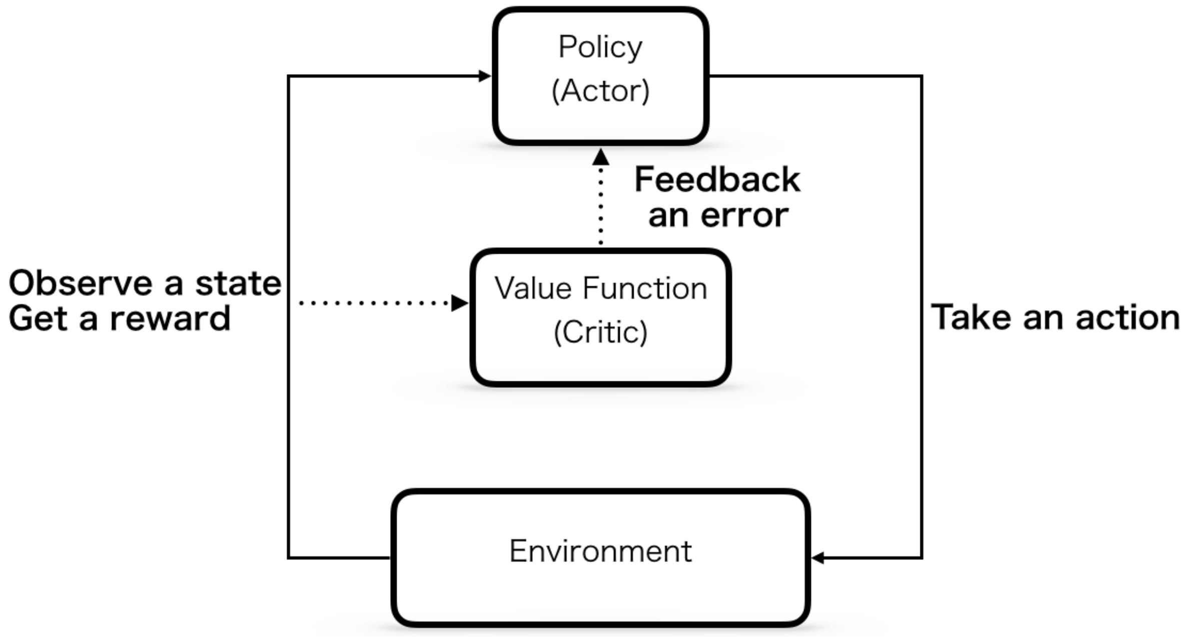

Figure 2.3: Actor-Critic architecture. The network consists of an actor and a critic. Given the newly observed state and reward, the critic calculate the difference between the realized value of the state/action and the estimated value of the state, and feedback it to the actor.

value of rewards and the estimate of the value at the state S_t. A method that combines value-based iteration and policy optimization is called *actor-critic* [Sutton and Barto, 2018, Mnih et al., 2016] (Figure. 2.3).

2.1.7 Proximal Policy Optimization

In Chapter 3 and 4, we use *Proximal Policy Optimization* (PPO) [Schulman et al., 2017], which is a policy gradient method, and it works in an actor-critic manner. A drawback of the policy gradient method is that the parameter update can be too large and result in performance degradation, or conversely, the update is too small and take longer to converge. *Trust Region Policy Optimization* (TRPO) [Schulman et al., 2015] is a policy gradient method that maximizes the following surrogate objective with the constraint:

$\underset{\theta}{argmax}\ \mathcal{L}(\theta_{old}, \theta)\ s.t.D_{KL}(\theta||\theta_{old}) \leq \delta$ where

$$\mathcal{L}(\theta_{old}, \theta) = E_{s,a\sim\pi_{\theta_{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A^{\pi_{\theta_{old}}}(s,a)\right] \tag{2.10}$$

and $A^{\pi_{\theta_{old}}}(s,a)$ is the advantage function that is ${}^{\pi_{\theta_{old}}}(s,a) = Q_\pi(s,a) - V_\pi(s)$. D_{KL} is the *Kullback–Leibler* (KL) divergence.

$$D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_{old}}(\cdot|s)\right) \tag{2.11}$$

The KL term avoids the new policy changing too much compared to the old policy. However, TRPO approximates the update, and the analytical solution requires Hessian-vector of the KL term, which is computationally expensive. PPO intends to reduce computation cost by replacing the KL constraints for parameter update with a clipping operation, and the objective function is defined as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}\big[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\big] \tag{2.12}$$

where $r_t(\theta)$ is the probability ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$, which is used in the TRPO as well.

2.2 Model-Based Reinforcement Learning

If an RL agent has a model of the environment, there are many different ways of using it. The agent can learn the policy from the model, explore efficiently or build an "intrinsic motivation," or counterfactual reasoning based on the acquired model is possible. Here, we focus on one of the drawbacks of the RL framework: sample inefficiency, which means the training process requires a large amount of data, and thus it requires a considerable cost to collect data (e.g. a robot destroys itself by entering a dangerous path). One intuitive solution

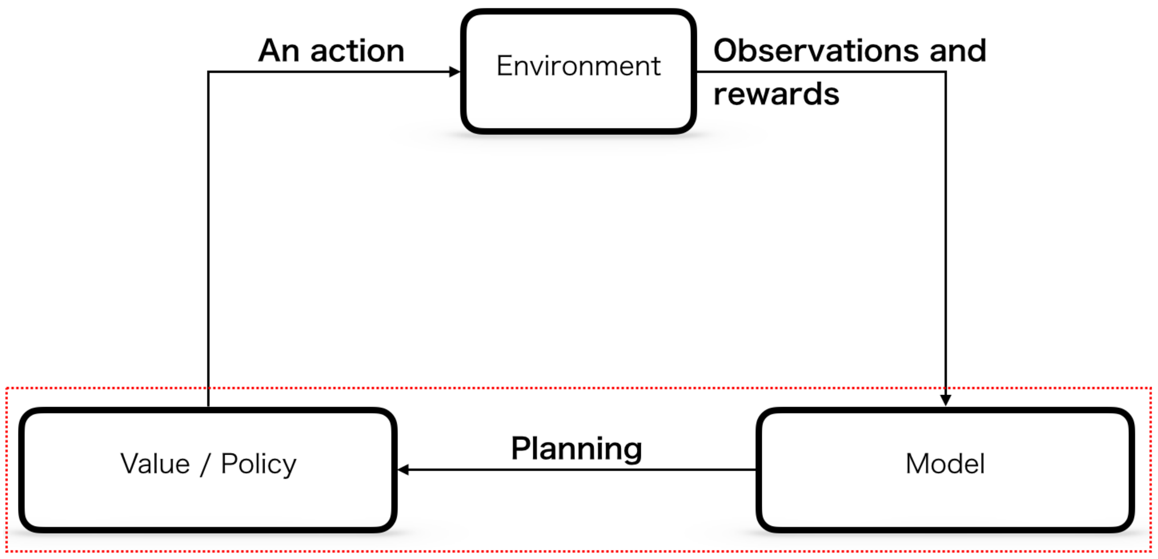

Figure 2.4: Model-based RL architecture. The model is used for planning. In accordance with [Sutton and Barto, 2018], 'planning' here refers to *"any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment"*

is to make the agent utilize some kinds of knowledge representation of the environment such as transition dynamics, and model-based RL has been studied to overcome the sample efficiency challenge. In model-based learning, the model is used to accelerate the agent's learning process by simulating the world dynamics. However, an accurate model is key to take full benefit of model-based RL [Gu et al., 2016] and generating accurate models is the challenging problem that needs to be solved. Although it is possible to define a model by hand for very small problems, it is not practical to build a model by hand for a larger scale problem. In addition, in typical RL setups, the agent assumes no prior knowledge and needs to learn the environment dynamics model in the first place. Thus, approximation techniques such as neural networks are widely adapted and have been an active area of research [Ha and Schmidhuber, 2018, Kaiser et al., 2019]. There are several different kinds of models. Following Lesort et al. [Lesort et al., 2018], we introduce three different categories of the models here (Figure. 2.5).

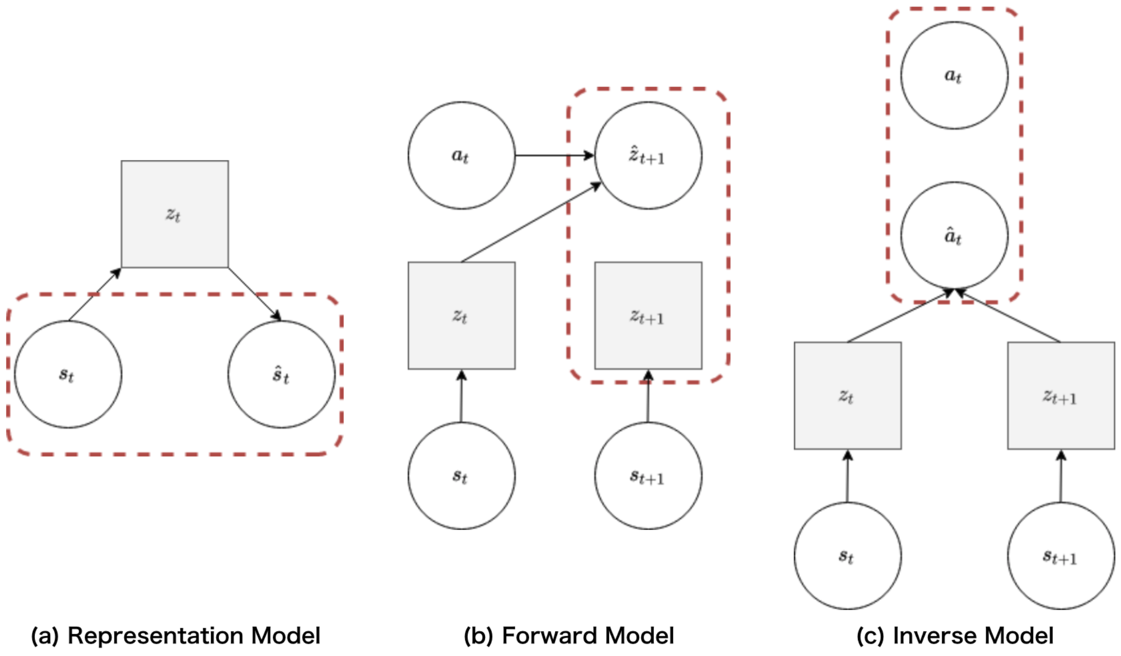

Figure 2.5: Different approaches to learn models. The figures are adapted from [Lesort et al., 2018] with some modifications. (a) learns the representation of a state by reconstructing. (b) predicts the next representation of the state given the current state and the action. (c) predicts what action was taken by comparing the current state and the observed next state.

Representation Model This model learns to encode inputs into latent code, typically a smaller dimension than the original input, and reconstruct the original input from the latent code. The model is trained by minimizing the reconstruction error. The most common example is autoencoder, and we discuss this objective later in this chapter.

Forward Model This learns the forward dynamics of the input states. Given the current state and action, the model predicts the next representation and minimizes the difference between the prediction and the observation.

Inverse Model This type of model predicts an action instead of the state. The input for this model is the current state (or state representation) and the next state. Then, the model predicts what the action was, which results in the forward dynamics from s_t to s_{t+1}.

One of the showcases of a combination of different models is World Models [Ha and Schmidhuber, 2018]. In this architecture, they first train a representation model of the environment using VAE. Once the representation model, which they call the vision module, is trained, a forward model using recurrent neural network (RNN) is learned. Input for the forward model is the encoded vectors from the representation model instead of the raw inputs. After those two models, they train a simple Multi-layer perceptron (MLP) controller with an evolutionary algorithm. World models successfully show that the agent can be trained entirely from the simulated environment using a car racing game and a first-person shooting game. Model-based RL for Atari [Kaiser et al., 2019] introduced Simulated Policy Learning (SimPLe), which outperforms state-of-the-art model-free algorithms at that time with a restriction of a budget to 100K time steps (*"corresponding to about two hours of play time"* according to the authors, which sounds very reasonable considering a human player's game play learning). Given that most of the model-free methods typically require more than millions of time steps, this is the significant reduction of the required budget. Their approach alternates model-free learning and the model-learning. This style was pioneered by Dyna, which we will talk about in the next section. World models in [Ha and Schmidhuber, 2018] trains the models once using the collected data, which is collected by running a random policy, but SimPLe iterates policy learning and the model learning similar to the Dyna architecture we describe next in 2.2.1.

2.2.1 Dyna

Dyna [Sutton, 1991] integrates model-free learning and model-based learning approaches as shown in Fig. 2.6. The agent learns its policy in a model-free way. Along with this direct learning path, the agent learns the model of the environment on the fly based on experiences. This distinguished Dyna from typical model-based approaches where one assumes complete knowledge of the environment dynamics, or the model is pre-trained. The agent uses the model to generate simulated transitions while it is trained. This boosts the agent's policy

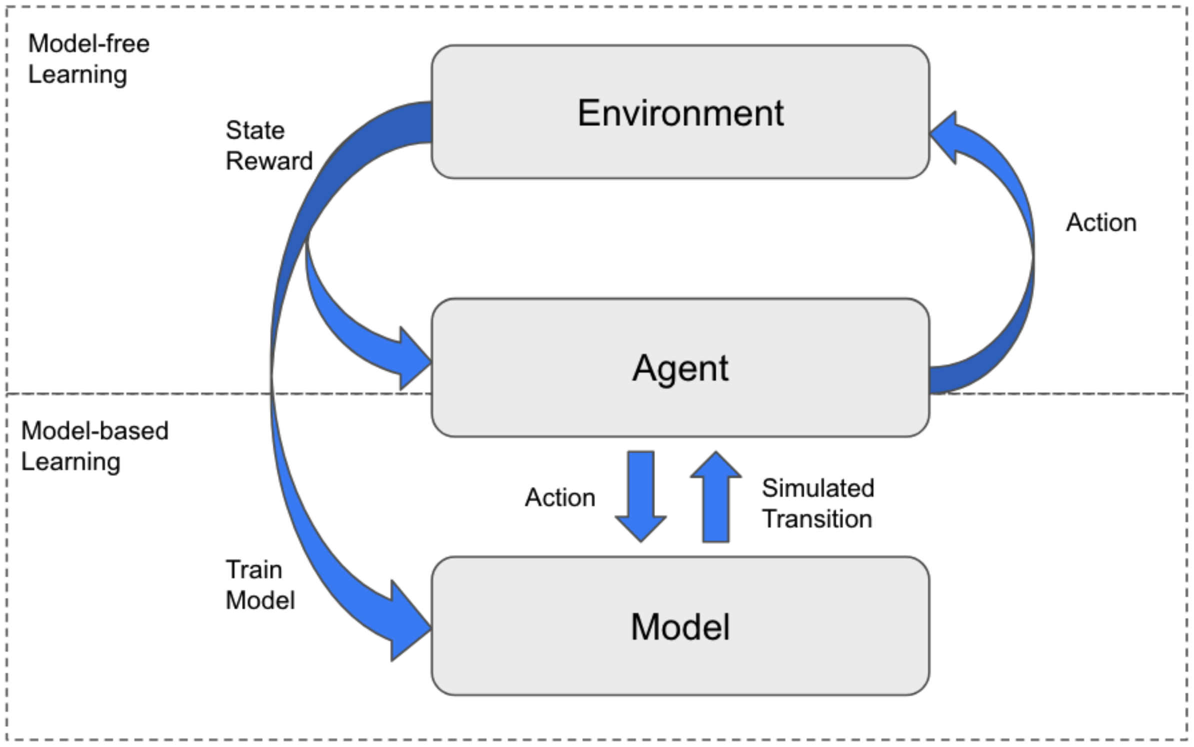

Figure 2.6: Dyna Architecture. The agent learns its policy in a model-free way. Along with this direct learning path, the agent learns the models of the environment on the fly based on experiences and uses the model to generate simulated transitions to extend experiences for the training.

learning speed by putting additional inputs for the policy. This is attractive, especially in a situation where physical interactions come at a cost, e.g., robotic navigation [Hayamizu et al., 2021] or dialogue agents [Peng et al., 2018].

In some model-based approaches, which train models prior to agent training, random rollouts is the typical approach to collect the data to train the model. However, these random rollouts can miss many possible states because the random actions can not lead the agent to as many states as possible. Dyna is an attractive approach for this reason as well since the architecture performs model learning as well while collecting data with the improved policy, which leads the agent to novel states. We will discuss this from a continual learning perspective in the Chapter 5.

2.3 Uncertainty in Deep Learning

As more deep learning-based real-world applications emerge, it is essential to know how uncertain predictions are for some fields, especially where erroneous interpretations of results lead subjects to fatal consequences such as medical diagnosis or self-driving vehicle to an accident. However, estimating the uncertainty of a deep neural network output is a notoriously difficult problem [Gal, 2016, Gawlikowski et al., 2021]. What standard deep learning provides us are point estimates of model parameters and deterministic predictions from the learned model.

Bayesian approaches offer methods to compute a posterior distribution $p(\theta|\mathcal{D})$, which expresses the distribution of the parameters of the network θ given data set $\mathcal{D}$ and enables us to evaluate the uncertainty in parameters [Bishop, 2006]. The entropy of this $p(\theta|\mathcal{D})$ encodes the model uncertainty but is intractable. Bayesian Neural Network (BNN) is one approach to approximate this [Blundell et al., 2015]. In BNNs, parameters are themselves random variables conforming to given probability distributions, and the inference process is stochastic instead of the point estimates with the constant parameters. However, these approaches introduce additional computational costs as we need to marginalize across all the possible parameters to compute the posterior distribution, which is intractable. Gal et al. [Gal and Ghahramani, 2016] showed that a neural network, which has dropout layers [Srivastava et al., 2014] before every weight layer, provides a way to measure uncertainty in predictions numerically by doing dropout both training and inference time.

There are different types of uncertainty in model-based reinforcement learning, and we discuss two of them here: uncertainty derived from environmental stochasticity and uncertainty related to model learning. These are called *aleatoric* and *epistemic*, respectively [Gal, 2016, Hüllermeier and Waegeman, 2019].

***Aleatoric* Uncertainty** This represents stochasticity in a system including sensor, and a likelihood $p(\mathcal{D}|\theta)$ in the Bayes rule captures this uncertainty. Since it comes from the underlying system, the level of uncertainty will remain the same even when the model gets unlimited data for training.

***Epistemic* Uncertainty** *Epistemic* uncertainty captures uncertainty caused by a lack of knowledge or data, which can be reduced by adding more data or unexplored information. Given this definition, *epistemic* is the one related to our model parameters. A posterior $p(\theta|\mathcal{D})$ encodes this uncertainty as its variance, which gets smaller if we can obtain infinite data.

Aleatoric and epistemic uncertainty is related to the RL tasks. The environment has stochasticity, and there is inherent aleatoric uncertainty in the agent's observations. The agent needs to reduce epistemic uncertainty by exploring and collecting as many experiences.

2.4 Representation Learning

In deep learning, learning input data representation is a critical area of study. How the learned data are represented significantly affects the performances of tasks [Bengio et al., 2012, LeCun et al., 2015], and many different methods have been proposed. 'Representation' here includes several things, such as the choice of network model architecture, an assumed data prior, or transformations. All these factors and the hierarchy of linear and non-linear transformations contribute to eventually acquiring the useful representations. Figure 2.7 is a visualization of learned representations of hand-written digit data [LeCun and Cortes, 2010] in 2D space. The original data is a 28×28 grayscaled image data, and those data is represented with 2D vectors. We hope to acquire "good" representations. A good representation has the following characteristics. It is easy for a classifier to separate the input data in a

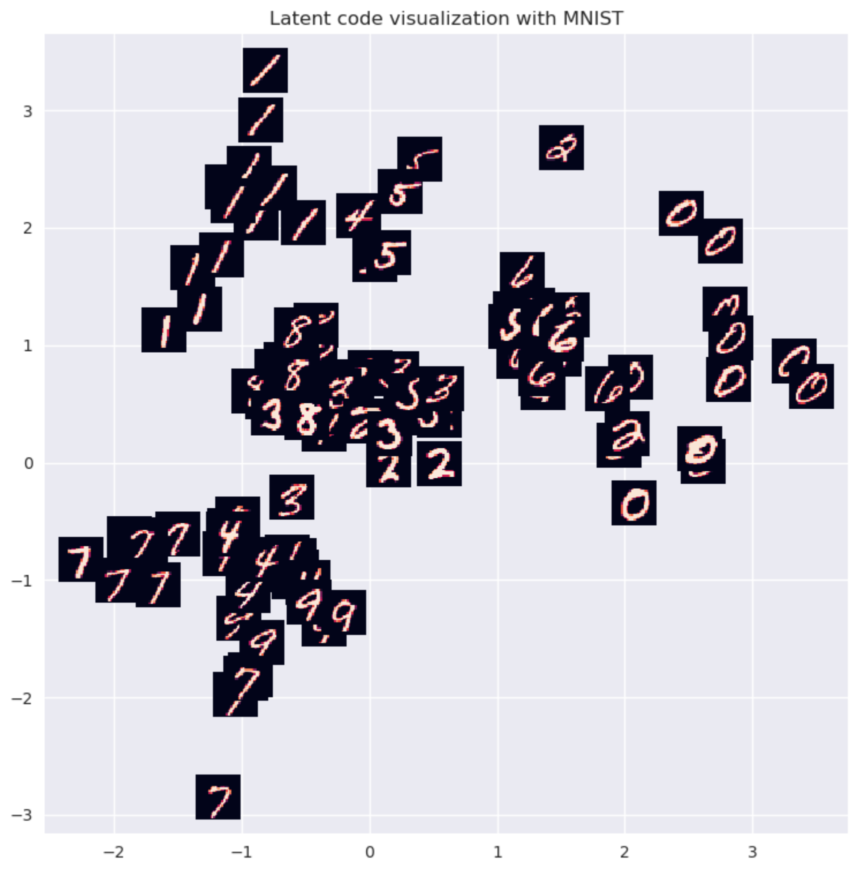

Figure 2.7: Visualization of learned representations by Variational Autoencoder using MNIST data set. The dimension of latent space is 2 in this example, and the horizontal and vertical axis corresponds to a value of the first and second dimensions in the latent space. For example, a vector [-0.7228127, 3.0308518] in the latent space is character '1', or [2.7340467, 1.6926069] corresponds to the character '0'. These vectors are the representation in the latent space, and the objective of the representation learning is to acquire this mapping from the original data space to the latent space.

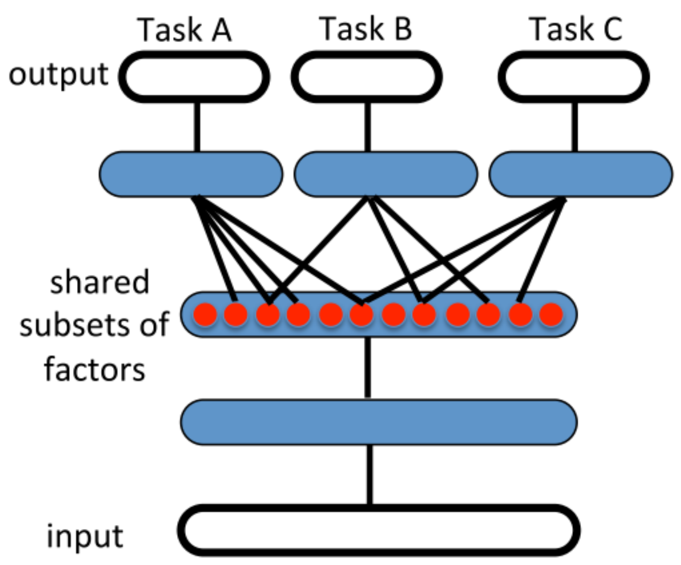

Figure 2.8: Example of learned representation, which is factorized and useful for multiple tasks sharing some factors. Task A, B, and C shares some factors (red circles) in the representation. Thus, the intermediate representation can be transferred, and it does not need to be learned for each task. Adapted from [Bengio et al., 2012] ©2012 IEEE.

transformed feature space. Further, it can be used in several different tasks sharing some properties. Since it factorizes the underlying data generative factor, it provides explainability or generalizability (Figure 2.8).

2.4.1 Generative Models

In machine learning, the two main approaches are discriminative and generative models. The discriminative model determines the class's posterior probability given the data: $p(C_k|x)$ where C_k means the k-th class. On the other hand, the generative models' approach is to model a distribution of the data given training samples: $P(x|C_k)$. This model can generate new data by sampling from the learned distribution. There are several approaches to learn generative models as shown in Figure 2.10. First, whether the model requires explicit density function or not is the important distinction. If the function is intractable, variational approximation or approximation using Monte Carlo are the approach to learn the model.

On the other hand, there is the implicit density branch, where the density function is not explicitly specified. GAN, which is discussed in Section (2.4.1) falls into this category.

Variational Autoencoder

Variational autoencoder is a generative model proposed in [Kingma and Welling, 2013]. This network learns a representation of the input data in an unsupervised way and has a similar architecture to autoencoder. During the training, it reconstructs input data and map them to a compact representation called latent vector. We assume that the data set $X = \{x^{(i)}\}_{i=1}^{N}$ with N i.i.d data points is generated from some underlying data generation rule, and a variable z is involved in that process, which corresponds to the latent vector. The network consists of a probabilistic encoder $q_\phi(z|x)$ and a probabilistic decoder $p_\theta(x|z)$. ϕ and θ are parameters of the encoder and the decoder, respectively. The encoder infers a latent vector z given the input data x. On the other hand, the decoder generates a data point x given the corresponding latent vector z. Both p_θ and q_ϕ produce a distribution. We assume that the prior distribution of z is an isotropic multivariate Gaussian $p_\theta(z) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. The distribution of the encoder is a Gaussian as well: $p_\theta(z|x) = \mathcal{N}(z|\mu, \sigma^2)$ where parameters μ and σ are determined by a function (e.g. a neural network) given an input (Figure 2.9). The training loss consists of a reconstruction loss and Kullback–Leibler divergence between $p_\theta(z|x)$ and $q_\phi(z|x)$. In recent studies, Variational Autoencoder (VAE) and its variants are often employed to learn a compact representation of visual input [van Hoof et al., 2016, Ha and Schmidhuber, 2018, Hafner et al., 2019]. The Variational Autoencoder consists of the encoder and the decoder. Given an input, the encoder predicts the parameters for Gaussian distribution with mean μ and variance σ^2. Given those parameters, a latent vector z is drawn from the Gaussian distribution, and the decoder reconstructs the input based on that latent vector. However, the sampling process stops the gradient propagation, and the backpropagation can not be used. VAE avoids this issue by the reparameterization trick.

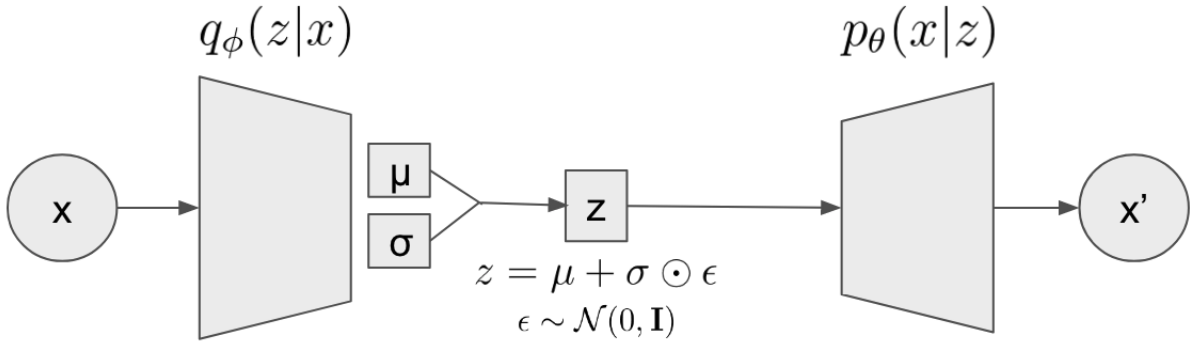
$q_\phi(z|x)$
$p_\theta(x|z)$
x
μ
σ
z
$z = \mu + \sigma \odot \epsilon$
$\epsilon \sim \mathcal{N}(0, \mathbf{I})$
x'

Figure 2.9: VAE Architecture. The network consists of a probabilistic encoder $q_\phi(z|x)$ and a probabilistic decoder $p_\theta(z|x)$. The encoder outputs a set of parameters for the Gaussian distribution, and a latent vector z is generated based on those parameters using the reparameterization trick so that it can perform backpropagation. The decoder then reconstructs the input. Training objective is to minimize the reconstruction loss and the KL-divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$.

Instead of sampling $z \sim N(\mu, \sigma^2)$, computing $z = \mu + \sigma^2 \times \epsilon$ makes gradient calculation possible, thus, the loss is backpropagated.

What we'd like to solve $p(\mathbf{Z}|\mathbf{X})$ which is a posterior of the model parameter θ given the collected data $\mathbf{X}$ where we assume that each data point x_i is i.i.d. Instead of the point estimate methods using maximum likelihood or maximum posterior (MAP) estimation, a probability distribution of θ given the data $\mathbf{X}$ can be calculated using the Bayes rule: $p(\mathbf{Z}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{Z}) \cdot p(\mathbf{Z})}{p(\mathbf{X})}$ which tells us *posterior* $\propto$ *likelihood* $\times$ *prior*. The Bayesian approach has several benefits over the point estimation; for example, it is robust to overfitting or provides uncertainty measure. However, the posterior $p(\mathbf{Z}|\mathbf{X})$ is intractable. To handle this issue, introducing a tractable approximated distribution $q(\mathbf{Z}) \sim p(\mathbf{Z}|\mathbf{X})$ is the core idea of the variational bayes.

Given the parameter θ, the evidence, which corresponds to the likelihood of the data x is

$$\begin{aligned}
\ln p(x;\theta) &= \ln \sum_{z} p(x,z;\theta) \\
&= \ln \sum_{z} q(z) \frac{p(x,z;\theta)}{q(z)} \\
&= \ln \mathbb{E}_{\mathbf{Z}\sim q}\Big[\frac{p(x,\mathbf{Z};\theta)}{q(\mathbf{Z})}\Big] \\
&\geq \mathbb{E}_{\mathbf{Z}\sim q}\Big[\ln \frac{p(x,\mathbf{Z})}{q(z)}\Big] \\
&= \mathcal{L}(q)
\end{aligned} \tag{2.13}$$

where the lower bound of the evidence $\mathcal{L}(q)$ is called Evidence Lower Bound (ELBO). Thus, the objective of the variational bayes is to maximize the ELBO. Further, Kullback-Leibler divergence (KL divergence) between $q(z)$ and $p(z|X)$ is written as equation 2.14, and our objective is to minimize the KL divergence of $q(z)$ and $p(z|X)$.

$$\begin{aligned}
KL[q(z)|p(z|X)] &= \sum q(z) \ln \frac{q(z)}{p(z|\mathbf{X})} \\
&= \sum q(z) \ln \frac{p(X)q(z)}{p(\mathbf{X},z)} \\
&= \sum q(z) \ln p(X) - \sum q(z) \ln \frac{p(\mathbf{X},z)}{q(z)} \\
&= \ln p(X) - L(q)
\end{aligned} \tag{2.14}$$

Generative Adversarial Networks

The GAN is a generative model which consists of two networks called the generator G and the discriminator D. These networks are parameterized by θ_g and θ_d, respectively. The generator produces new data $x = G(z)$, where z is drawn from a prior distribution $z \sim p(z) = \mathcal{N}(z; 0, I)$. The discriminator is a binary classifier, and it classifies whether the input data is real data or fake. The generator aims to minimize the probability of the synthesized data being detected as a fake. On the other hand, the discriminator's objective is to maximize the

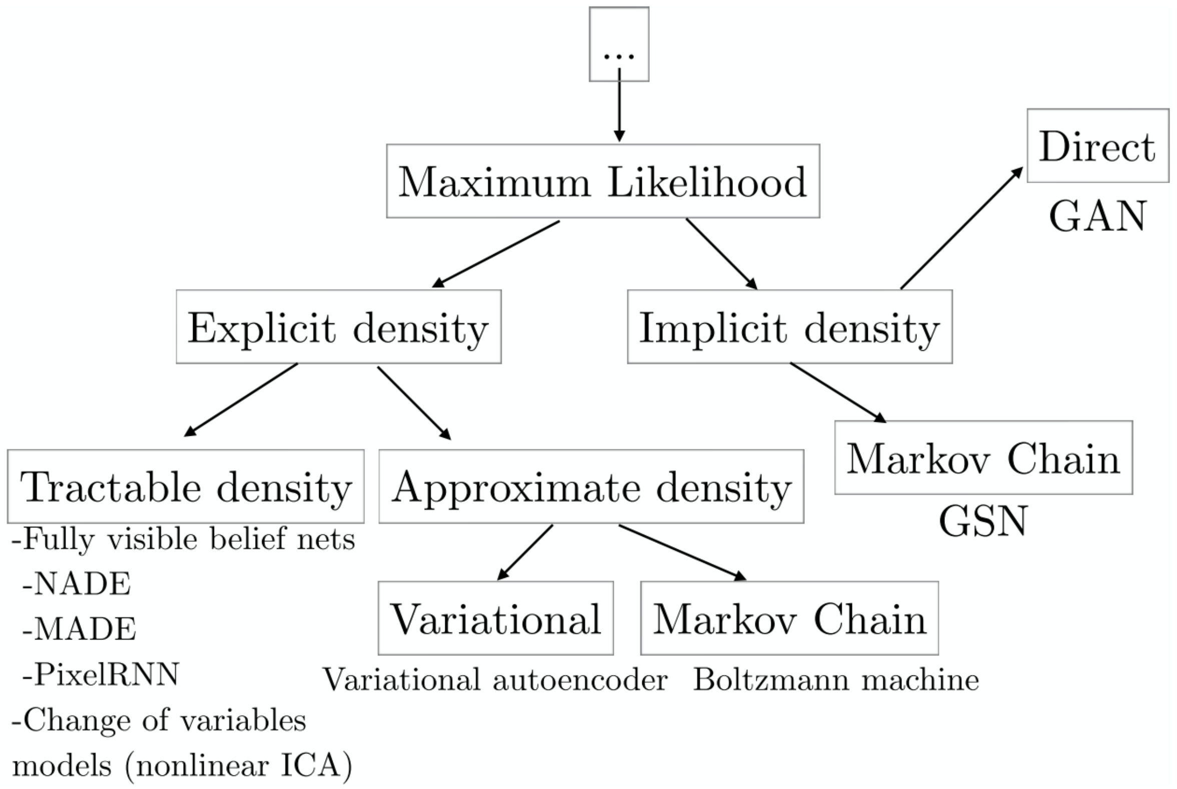

Figure 2.10: Deep generative models taxonomy adapted from [Goodfellow, 2017]. There are two broad classes: 1. Explicit density models, where a density function needs to be defined, and 2. Implicit density models that do not require such functions. GAN is in the latter family as it does not require defining a density function.

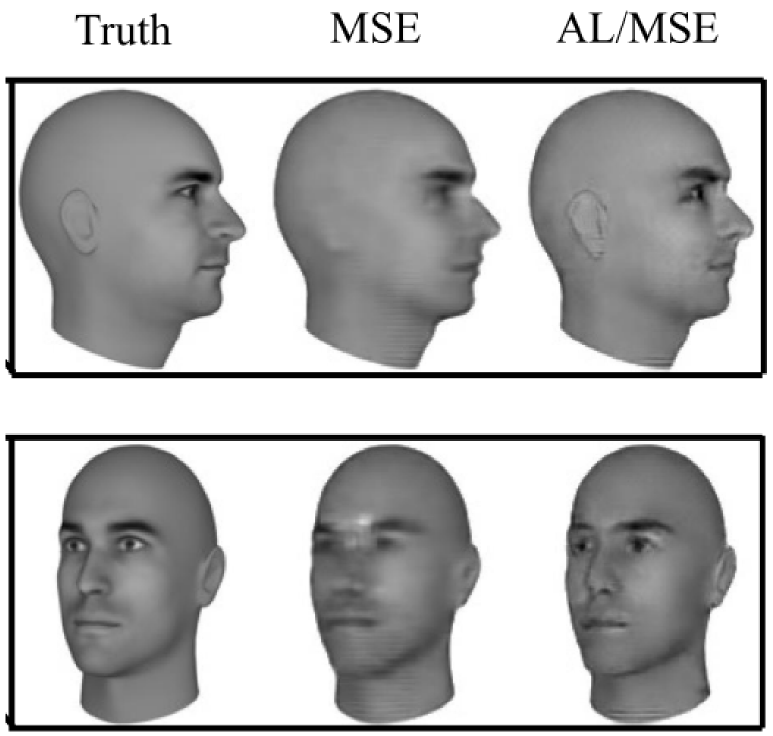

Figure 2.11: Comparison of the generated image characteristics between MSE and adversarial loss. Since the MSE losses consider all possible samples, it dent to generate blurry images. On the other hand, the adversarial losses only require one specific sample to consider, which makes the generated samples look sharper. The image adapted from [Lotter et al., 2015].

accuracy of classification. Together, the objective is written as a minimax game as follows:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \tag{2.15}$$

where $D(x)$ corresponds to the correct classification rate, and $1 - D(G(z))$ means the probability of the generator successfully fooling the discriminator. GAN is unique that it does not require to explicitly specify a density function $p_{model}(x; \theta)$ (Figure 2.10). Regarding the generated image quality, one distinction of GANs from VAE is that the GANs generate sharper images than VAE. This is because the squared loss in VAE considers the average of all possible samples and makes the generated image blurry (Figure 2.11).

2.4.2 Disentanglement

Disentanglement has been studied as one of the beneficial properties of a good representation. The idea of disentangled representation is that latent units of the representation are well-factorized, and each factor is less sensitive to the others [Bengio et al., 2012]. For instance, images have different factors such as lighting, scale, angle, or objects in the images have several properties, such as object material and shape. In human face images, a person's hair color, whether a person is wearing glasses or not, or facial expressions are examples of high-level properties. The data consists of combinations of these factors. These disentangled properties makes it possible to manipulate the generated data [Higgins et al., 2017a, Li and Mandt, 2018].

2.4.3 Representation Learning in Reinforcement Learning

Learning a compact and effective representation of the input space (e.g., RGB image frames), which express the state of the environment well, is critical for RL agents because policy learning will suffer from the curse of dimensionality. Although the combination of neural networks with deep layers and RL algorithms has shown the success of end-to-end learning in high-dimensional observations domains such as video games [Mnih et al., 2015, Silver et al., 2016, Mnih et al., 2016], the learned representation is strongly tied with the policy learning, and it is brittle to change of input. As an alternative approach to representation learning connected to policy learning, which is affected by reward signals, unsupervised representation learning approaches have emerged, commonly using VAE [van Hoof et al., 2016, Ha and Schmidhuber, 2018, Hafner et al., 2019]. The characteristics of a good state representation described in [Lesort et al., 2018] are as follows: 1) the representation should have Markov property, which means the current state includes all the past information and is informative enough for an agent for decision-making. 2) The representation is expressive enough to have actual values of states. 3) Generalizable to similar but not identical unseen states. Last but not least, 4) the representation dimension should be low for efficient estimation. Also, we hypothesize that the disentanglement of the representation will beneficial for the RL agent as well [Higgins et al., 2017a, Bengio et al., 2012, Ridgeway, 2016]. Representation learning methods relying on pixel level reconstruction do not have strong incentive to disentangled representation. [Higgins et al., 2017b] proposed DARLA (DisentAngled Representation Learning Agent), which has its vision module consists of the β-VAE, in which the hyperparameter β works to weigh disentanglement, and the denoising autoencoder [Vincent et al., 2010] with a modification of the VAE loss. [Anand et al., 2019] introduced mutual information to learn state representations so that the learned representation disentangles distinct features well.

Chapter 3

Confidence Based Model Integration

3.1 Introduction

The recent extraordinary success of model-free reinforcement learning (RL) techniques has demonstrated its capability for various types of problems such as board games, video games, or robotic manipulation [Mnih et al., 2015, Silver et al., 2017, OpenAI et al., 2018]. However, model-free learning requires huge amounts of interaction with the environment, making it *sample inefficient*. On the other hand, model-based RL is known to overcome the sample efficiency challenge. In model-based learning, the model is used to accelerate the agent's learning process by simulating the world dynamics. However, an accurate model is key to take full benefit of model-based RL [Gu et al., 2016] and generating accurate models is a challenging problem that needs to be solved. The agent assumes no prior knowledge in typical RL setups and needs to learn the environment dynamics model. Thus, approximation techniques such as neural networks are widely adopted and have been an active area of research [Ha and Schmidhuber, 2018, Kaiser et al., 2019]. There is often insufficient prior data to train the model in an actual situation, which causes problems such as inaccurate predictions or situations where the model needs to simulate states that fall outside the training data distribution. Under such a situation, one of the promising solutions is to build an agent that also learns the model.

Dyna (Section 2.2.1) was proposed as an architecture that integrates both model-free and model-based reinforcement learning [Sutton, 1991]. In this architecture, the agent learns a policy in a model-free way by directly interacting with the environment. At the same time, the agent learns a state transition dynamics of the environment using real experiences. The agent uses the learned model to simulate a trajectory and updates its policy based on both real experiences and simulated trajectories. It is known that the model greatly accelerates the agent's learning process, provided it is accurate [Miller et al., 1995]. The Dyna architecture has been studied in different types of tasks, from simple tabular problems to complex problems with large state spaces. In the latter case, recent advanced model-free methods have been employed [Peng et al., 2018, Angermueller et al., 2020]. However, the architecture inherits the challenge of making the model accurate enough to optimize the agent's policy. Oftentimes the learned model is inexact and can be detrimental to learning [Gu et al., 2016]. In this chapter, we propose a simple yet effective strategy to utilize the model while mitigating undesired outcomes from the inaccurate model simulations. We use the model's uncertainty (or how much the model is confident about each prediction) as a metric. This requires a model or technique that is capable of estimating uncertainty. We take a Bayesian inference approach that enables us to assess uncertainty quantitatively. Our approach uses Monte Carlo dropout (MC-dropout) [Gal and Ghahramani, 2016]. For each input, the model makes N predictions. If the prediction variance is beyond the given confidence interval of a given percentile, it is deemed incorrect and the simulation is rejected. We assume independent Gaussian distributions for the values of each dimension of states, therefore the same for the model simulations. Thus the confidence intervals can be estimated. The agent uses the generated samples if the model's prediction variance from each simulation is in the calculated confidence interval. On the other hand, the agent should not put a high value on the simulation where variance is out of the confidence interval and cause the degradation of policy learning. This mechanism works as a gate to determine whether the model simulations are used, and we expect the model to improve performance at early stages. To the best of our knowledge, the integration of model uncertainty estimation based on MC-dropout into Dyna-style architecture and filtering the simulation is novel.

We evaluated the proposed approach on robotic control tasks using the MuJoCo simulator [Todorov et al., 2012]. We tested our approach with Proximal Policy Optimization (PPO) [Schulman et al., 2017] for policy optimization, combined with Dyna style architecture which we call Dyna-PPO.

This chapter is based on previously published work:

Nagata, T., Xing, J., Kumazawa, T., & Neftci, E.O. (2022). "Uncertainty Aware Model Integration on Reinforcement Learning". In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN).

Portions are reprinted with permission, ©2022 IEEE.

3.2 Methods

In the Dyna architecture, the agent collects real experiences and uses them not only to improve its value function and the policy but also to improve the model. In this section, first, we review model-based RL and uncertainty in deep learning. Then, explain our method to model the dynamics of the environment and the reward structure. Finally, we introduce our strategy to integrate those models into the Dyna-PPO algorithm.

3.2.1 Model-Based Reinforcement Learning and Dyna

The model refers to some kind of representation by which an agent can get information about the world. There are several types of models such as state transition models: $p(s'|s, a)$ and reward models: $p(r|s, a)$. They are used to predict the next state s' or the immediate reward r given the current state s and the action a. The state transition models can be a density estimation problem, and the reward models are a regression problem. If the state space is small, e.g., a 9×6 grid [Sutton and Barto, 2018] world, a table of transition probabilities (tabular model) is sufficient to make a prediction based on the collected data. When the state space is large or continuous, tabular approaches are impractical or impossible. In this case, a function approximator, such as a neural network, can be used [Hafner et al., 2019] to map the inputs, the current state $s \in \mathbb{R}^{d_s}$ and the action $a \in \mathbb{R}^{d_a}$, to the next state $s' \in \mathbb{R}^{d_s}$, where d_s and d_a correspond to the dimension of states and the dimension of actions for each.

In model-based RL, the model can be used in different ways. One approach is to generate one-step samples $\hat{s}_{t+1} \sim p(\hat{s}_{t+1}|s_t, a_t)$ and use them for the training of the agent's policy. Another approach is to perform multi-step simulations. There, a model is used to look ahead several steps. Further, this can be used to simulate multiple different trajectories starting from the current state s_t, and the agent utilizes that information for planning. Monte-Carlo Tree Search (MCTS) is one popular approaches of this family [Coulom, 2007, Silver et al., 2017]. In all cases, the model offers sample efficiency for an agent compared to model-free learning approaches because the agent can use trajectories simulated by the model to optimize its policy rather than through interactions with the environment. Our approach corresponds to the multi-step simulation approach to generate simulated trajectories from sampled states, but it does not build a tree for planning as in MCTS.

Dyna integrates model-free learning and model-based learning approaches as shown in Figure 2.6. The agent learns its policy in a model-free way. Along with this direct learning path, the agent learns the model of the environment on the fly based on experience. This distinguishes Dyna from typical model-based approaches where one assumes complete knowledge of the environment dynamics, or the model is pre-trained. The agent uses the model to generate simulated transitions while it is trained. This boosts the agent's policy learning speed by putting additional inputs for the policy. This is attractive for situations where physical interactions come at a large cost, e.g., robotic navigation Hayamizu et al. [2021] or dialogue agents Peng et al. [2018]. Although the model can increase the agent's learning speed, this is the case only when the model is sufficiently accurate. An incorrect model causes performance degradation due to policy updates on inaccurate simulations. Thus guaranteeing the model's accuracy is a heavily studied topic. One of the approaches is the ensemble-based method, in which several models are used to make predictions to utilize the model better [Chua et al., 2018]. [Angermueller et al., 2020] fits their models to the available data after each iteration and evaluates the accuracy of each model by the coefficient of determination (R^2). Those models, of which the R^2 is greater than a hyperparameter τ, are qualified, and their ensembles are used. If there is no qualified model, the agent skips the model-based learning step in the current iteration to mitigate its negative effect. Ensemble methods require significantly more computations and parameters. This paper derives from a similar motivation: how we can best utilize our model-based approach while mitigating possible situations where the model works negatively. To evaluate the model to determine the reliability, we apply a dropout-based uncertainty measurement[Gal and Ghahramani, 2016], which requires fewer parameters and computation compared to ensemble methods. MC-dropout applies the dropout technique at inference time as well as training time. Thus, the predictions are stochastic, which enables prediction variance calculation. We further discuss MC-dropout in the next section. We propose using the prediction variance to determine whether the simulated experiences should be used for policy updates or discarded.

3.2.2 Model Uncertainty Estimation Using MC-Dropout

There are two types of uncertainty in model-based reinforcement learning: uncertainty derived from environmental stochasticity and uncertainty related to model learning. These are called aleatoric and epistemic, respectively [Chua et al., 2018]. Epistemic uncertainty can be further divided into two different classes of uncertainty. The first class is caused by insufficient exploration of the environment, while the other is caused by correlations in the experiences. In this project, we focus on epistemic uncertainty due to the model's inaccuracy.

Estimating the uncertainty of a deep neural network output is a notoriously difficult problem. Bayesian approaches offer methods to compute a posterior distribution $p(\theta|\mathcal{D})$, which expresses the distribution of the parameters of the network θ given data set $\mathcal{D}$. The entropy of this $p(\theta|\mathcal{D})$ encodes the model uncertainty but is intractable. The Bayesian Neural Network (BNN) is one approach to approximate this [Blundell et al., 2015]. In BNN, parameters are random variables conforming to given probability distributions, and the inference process is stochastic instead of the point estimates with the constant parameters. However, these approaches introduce additional computational costs as it requires marginalizing across all the possible parameters to compute the posterior distribution, which is intractable. Instead of directly solving the intractable calculation, we approximate the distribution with another distribution q_ϕ and think of minimizing the KL-divergence between the posterior distribution and q_ϕ. Treating this minimization problem as the maximization problem of ELBO (2.4.1) provides us the approximated solution, which is called *variational inference.* Gal et al. [Gal and Ghahramani, 2016] showed that a neural network equipped with dropout layers before every weight layer, provides a way to measure uncertainty in predictions numerically. Dropout is an effective technique to achieve the better generalizability of a neural network. Internally, the dropout corresponds to train sub-models by randomly dropping some units in a network. Following the Bernoulli distribution $Bern(p_i)$, some units are randomly blanked out. [Gal and Ghahramani, 2016] showed that the loss for the dropout

$L_{dropout} = \frac{1}{N}\sum_{i=1}^{N} E(y_i, \hat{y}_i) + \lambda \sum_{i=1}^{L}(||\theta||_2^2 + ||b_i||_2^2)$ corresponds to the posterior of θ in variational inference. The usual way to utilize dropouts is to apply them at training time. In their method, dropout layers are also applied to inference time, which makes predictions stochastic, and thus the variance of multiple forward passes are available. They developed the theoretical framework to relate this with the approximation of Bayesian inference to Gaussian processes. They also showed that uncertainty estimates provide an RL agent with information to determine whether to exploit the current knowledge or to explore further. There, a function to approximate action-value function (Q-function) with dropout layers combined with Thompson sampling was shown to improve an RL agents' learning efficiency and convergence compared to an epsilon greedy approach. Their contribution is on the policy learning part. On the other hand, our main target is to evaluate the learned state transition dynamics model. In our method, the model iterates N predictions for each input and makes a prediction by averaging those predictions with different network connections. This approach is similar to ensemble-like inference, with the difference that a single neural network is used rather than an ensemble of models. We use $p_i = 0.5$ in fully-connected layers in the state transition model.

3.2.3 Models

Our method uses three different models: the state transition model, the reward model, and the termination prediction model. The state transition model approximates the dynamical system underlying the environment. Given the simulated state transition, the reward model predicts a reward based on the transition, and the termination prediction model classifies whether the next state is a terminal state or not. The termination prediction model is necessary to stop trajectory rollout and assign rewards accordingly.

State Transition Models We use an Encoder-Decoder architecture, in particular, a Variational Autoencoder [Kingma and Welling, 2013] for our state transition model. Here, a current state $s \in \mathbb{R}^{d_s}$ and an action $a \in \mathbb{R}^{d_a}$ are inputs for the state transition model and the model predicts the next state $s' \in \mathbb{R}^{d_s}$. We assume that the values follow independent Gaussian distributions for each dimension of states. The encoder predicts the parameters for a Gaussian distribution with mean μ and variance σ^2: $\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$. The parameters are determined for each dimension of the input. Thus, both μ and σ are the vectors of R^{d_s} respectively. Instead of reconstructing the input, the decoder is trained to predict the next state. The decoder has a dropout layer and is used for both training and testing (Figure 3.1). For each input, a model performs the forward operation I times and the final prediction is the average of those predictions: $\hat{s}_{t+1} = \frac{1}{I}\sum_{i=1}^{I} f_i(\mathbb{R}^{d_s+d_a})$. This corresponds to MC-Dropout, which provides a variance of prediction for each input.

Reward Model and Termination Prediction Model These two models are conventional (none autoencoder) feedforward networks. The reward model is a regressor to predict the reward value, and the termination prediction model is a classifier to predict whether the state transition leads the agent to the terminal state or not. The reward model ($r_t \sim p(r|s, a)$) has 4 layers, whereas termination prediction model ($p(s^+)$) where $s^+ \in S$ is the terminal state has 2 layers. Each layer is followed by ReLU activation function.

3.2.4 Model Quality Aware Integration

All three models are trained while the agent collects data interacting with the environment. When updating the agent's weight, the agent makes the model predict state transitions from states randomly selected from the stored data. Starting from the randomly selected state, the simulation is continued until it reaches the terminal state. We iteratively run this process to get N simulated steps. Those simulated transitions $\langle s_t, a_t, \hat{r}_t, \hat{s}_{t+1}\rangle$ are used along with

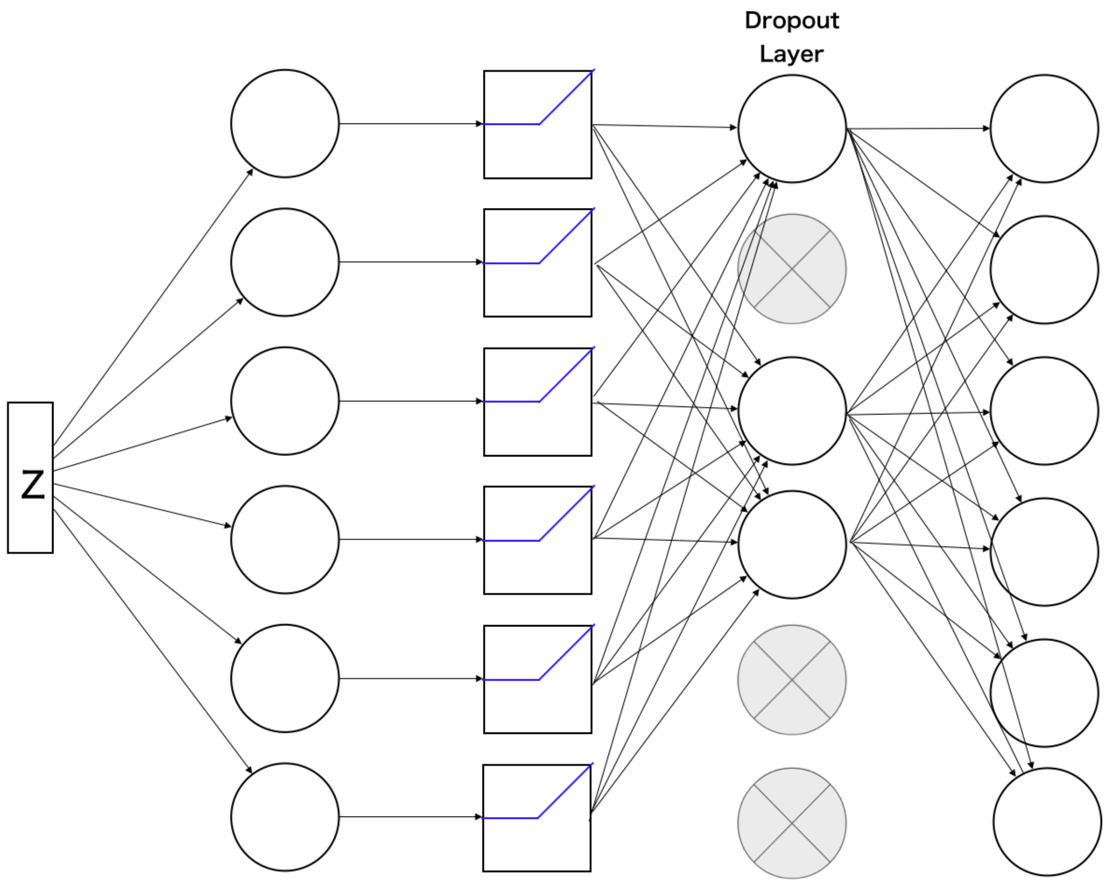

Figure 3.1: Decoder Architecture of the State Transition Model. The decoder has a dropout layer, and the output is stochastic instead of the point estimate. Thus, performing inference multiple times for each input enables prediction variance calculation.

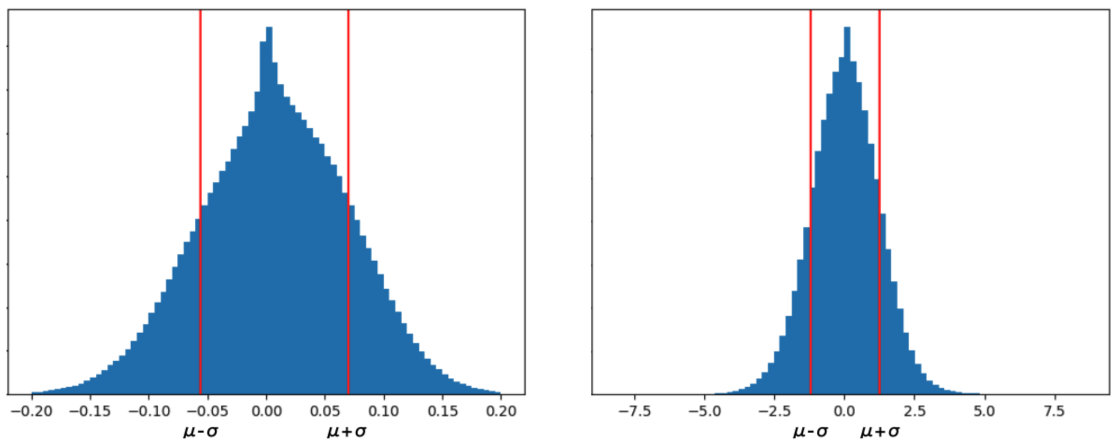

Figure 3.2: Examples of empirical distributions of values (observations) in some dimensions of states in the Hopper environment used to train the model. The agent collects N data steps in each epoch, and confidence intervals are calculated based on this distribution.

the real (observed) data to learn the policy. In our method, we assume that values in each dimension of states follow a Gaussian distribution, and we calculate the empirical mean and the variance of the real data as Fig. 3.2. Based on the distribution with the empirical mean and variance, the confidence intervals of $\pm\sigma, \pm 2\sigma, \pm 3\sigma$ centered on the empirical mean are determined. We verify whether each simulated transition falls into the interval. If the predicted state drifts off from the interval, it is rejected.

3.3 Experiments and Results

3.3.1 Dyna-PPO with Confidence Interval Checking

As a baseline algorithm to implement a Dyna-style model-free and model-based hybrid algorithm, we use PPO [Schulman et al., 2017] for our model-free algorithm. We selected PPO as it is a widely used algorithm, but any other on-policy algorithm would be suitable. As described in Section 2.1.7, PPO is a policy gradient algorithm that optimizes a surrogate loss, which clips a probability ratio of policy parameters before and after the update within

$[1-\epsilon, 1+\epsilon]$ so that the updated policy will not drastically change. The actions are chosen from a normal distribution. The parameters for the distribution, μ_p and σ_p, are determined by parametric function approximators given the state as follows: $\mu_p, \sigma_p = P_\theta(s)$ where P is a policy of the agent parameterized by θ and then an action is drawn from a Gaussian distribution parameterized by them: $a \sim \mathcal{N}(\mu_p,\, \sigma_p^2)$. The entire process is described in Algorithm 1.

As a whole, the Dyna-PPO with confidence interval checking works as follows: first, the agent interacts with the environment to collect T steps of real experiences. Then, to assess the model's prediction quality in the next step, we calculate the empirical distribution of states using that data. In addition, all three models are trained on that data. The state transition model $p(s_{t+1}|s_t, a_t)$ minimizes the reconstruction error between the predicted next state $\hat{s}_{t+1}$ and the observed next state s_{t+1} under the penalty of $KL(p\|q)$, where q is the distribution of simulated states and p is the empirical data distribution. The objective function for both the reward and the termination prediction models is a mean squared error between a predicted reward and the observed reward, and a classification error of termination state, respectively. Finally, generate N steps of simulated experience $\langle s_t, a_t, \hat{r}_t, \hat{s}_{t+1}\rangle$ from the three models. An initial state is randomly selected from the data, and simulated transitions are computed from the initial state. Once it reaches the terminal state, an initial state is randomly selected again, and simulations are generated until the total number reaches N. The generated simulations are evaluated during this phase based on the confidence interval calculated from the empirical distribution. We use the $\pm\sigma$ confidence interval to evaluate predictions made by the model. If any value of a simulated state is outside that interval, the simulation is rejected, and the model simulates the transition again. In case the model generates such poor simulations multiple times, the model simulation step is skipped altogether in that iteration. The whole process is illustrated in the Figure 3.3.

Algorithm 1 Dyna-PPO with simulation confidence

```
1: Initialize Hyperparameters [lr, $\gamma$, N, I]
2: Initialize $\pi_\theta(s)$ and models [$p(s,a)$, $r(s,a)$]
3: for epoch=1:M do
4:     Initialize empty dataset $\mathcal{D} \leftarrow []$
5:     // Collect Data
6:     while time step $t < 2048$ do
7:         $s_t \leftarrow current\_state$
8:         $a_t \leftarrow \pi_\theta(s_t)$
9:         Collect $\mathcal{D}_t \leftarrow \langle s_t, a_t, r_t, s_{t+1} \rangle$
10:    end while
11:    Calculate $\mu, \sigma \in R^{d_s}$
12:    // Model Learning
13:    Fit models on $\mathcal{D}$
14:    // Generate simulations using the model
15:    for n=1:N do
16:        $s_t \leftarrow current\_state$
17:        $a_t \leftarrow policy(s_t)$
18:        for i=1:I do
19:            $\hat{s}_{t+1}^{(i)} \sim p(s_t, a_t)$
20:        end for
21:        $\hat{\mu} \leftarrow \frac{1}{I} \sum_i \hat{s}_{t+1}^{(i)}$
22:        $\hat{r}_t \leftarrow r(s_t, a_t)$
23:        if $-\sigma < \hat{\mu} < \sigma$ then
24:            $\mathcal{D} \leftarrow \mathcal{D} \cup \{\langle s_t, a_t, \hat{r}_t, \hat{s}_{t+1} \rangle\}$
25:        end if
26:    end for
27:    Update policy $\pi_\theta$ on $\mathcal{D}$
28:    $\theta \leftarrow \theta_{updated}$
29: end for
```

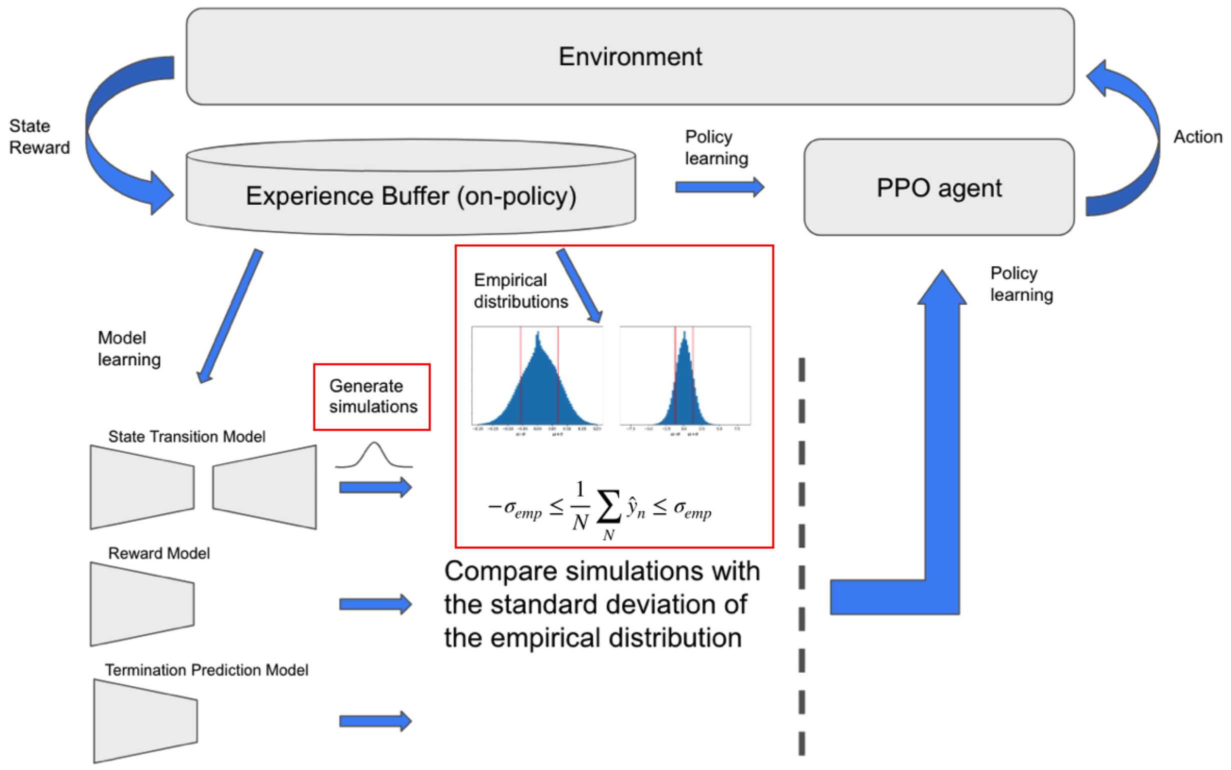
Environment
State
Reward
Action
Policy learning
Experience Buffer (on-policy)
PPO agent
Policy learning
Model learning
Empirical distributions
Generate simulations
State Transition Model
Reward Model
Termination Prediction Model
$-\sigma_{emp} \leq \frac{1}{N}\sum_{N}\hat{y}_n \leq \sigma_{emp}$
Compare simulations with the standard deviation of the empirical distribution

Figure 3.3: Dyna-PPO with Model Confidence Aware Integration.

3.3.2 Continuous Control Problems

We evaluate our approach on three different environments (Pendulum, Hopper and Swimmer) available in OpenAI Gym [Brockman et al., 2016]. All are control tasks with continuous state space and action space.

The evaluation criterion is the mean reward of different runs with random seeds. This experiment evaluates three different algorithms' performance: straight Dyna-PPO (no model steps), Dyna-PPO with model steps, and Dyna-PPO with confidence-based adjustments as described above. The number of model steps is 128. We evaluated 16, 32, 64, 128, and 256 model steps and empirically determined the value. We found that adding further model simulations does not necessarily improve the performance, and it is also a trade-off with the computation requirements.

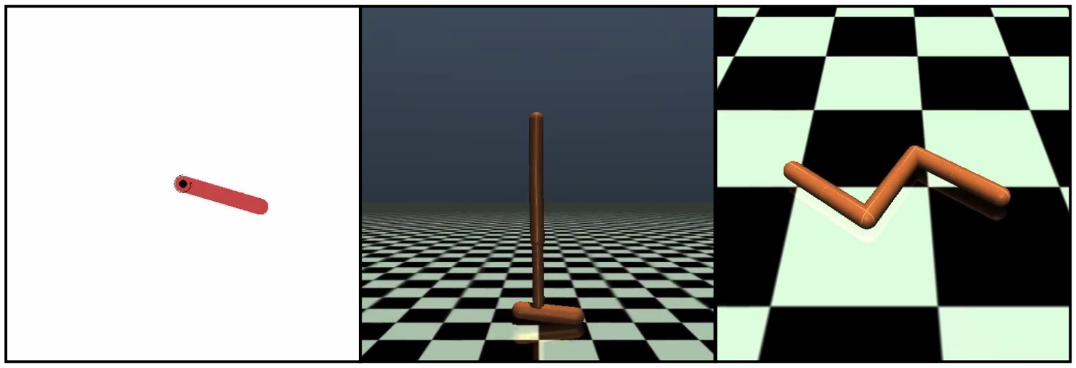

Figure 3.4: Tasks evaluated. Left: Pendulum. For the pendulum environment, the goal is to swing up the pendulum and keep that position as long as possible. Middle: Hopper. The goal is to keep making a one-legged robot hop forward. Right: Swimmer. The goal is to make a three-link robot swim forward as fast as possible. Each environment provides the agent with 3, 11, and 8-dimensional continuous observations, respectively.

Fig. 3.5 shows the performance results across all tasks. The graph shows that having a model lifts the performance throughout the iterations as we expect. Further, the performance of the Dyna-PPO with confidence interval checking outperforms the one without that mechanism. As the training proceeds, all three converge to a similar performance, which we expected for two reasons. First, the number of observations becomes sufficiently large by the time of convergence of baseline models. Second, as the model quality improves, it becomes less frequent that the generated simulation falls outside of the confidence interval of an empirical distribution. Thus, the effect of having the confidence checking mechanism becomes smaller. Performance improvements are summarized in the table below: The reported improvements

Performance improvement (%) over the baseline			
	Pendulum	Hopper	Swimmer
Model simulations	2.89	17.54	-4.44
+Confidence interval	12.28	31.84	2.42

Table 3.1: Performance improvements

are calculated as the average improvement ratio for all data points. $\frac{1}{E}\sum\frac{R^M-R^B}{R^B}$ where R^M is a cumulative reward in an epoch of the Dyna-PPO agent with model simulations, and R^B is the cumulated reward of the baseline (straight PPO). The table shows that the confidence interval-based approach outperforms both straight PPO and the standard Dyna approach.

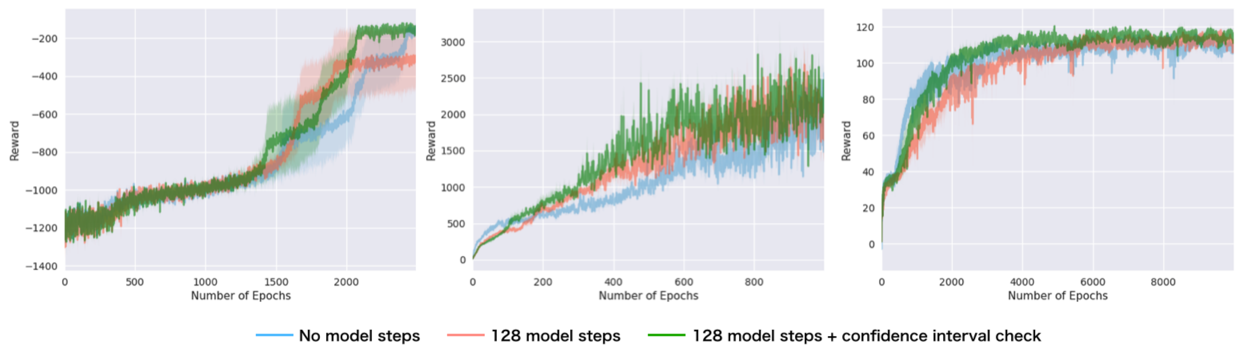

Figure 3.5: Comparison of PPO, Dyna-PPO learning curves with 128 model steps, and Dyna-PPO with 128 model steps with quality checking. Plots show training performance (rewards) over the number of collected episodes, and each line is a mean performance over random seeds.

In the Swimmer environment, the standard Dyna approach lags compared to the baseline approach, but confidence interval checking makes it outperform with respect to the baseline. Hyperparameters we used in the experiments are shown in the following table:

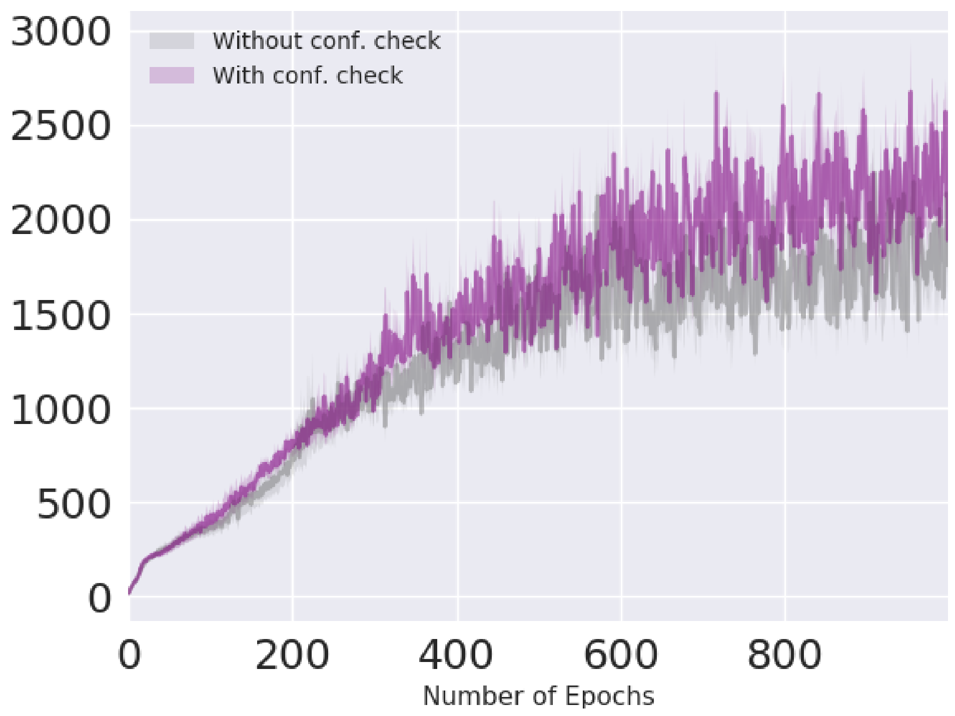

Figure 3.6: Comparison between our method without confidence interval checking (gray) and our proposed method with confidence interval checking (purple). 14.94% performance improvement averaged throughout the epochs in 5 different runs. Hopper environment is used in this experiment.

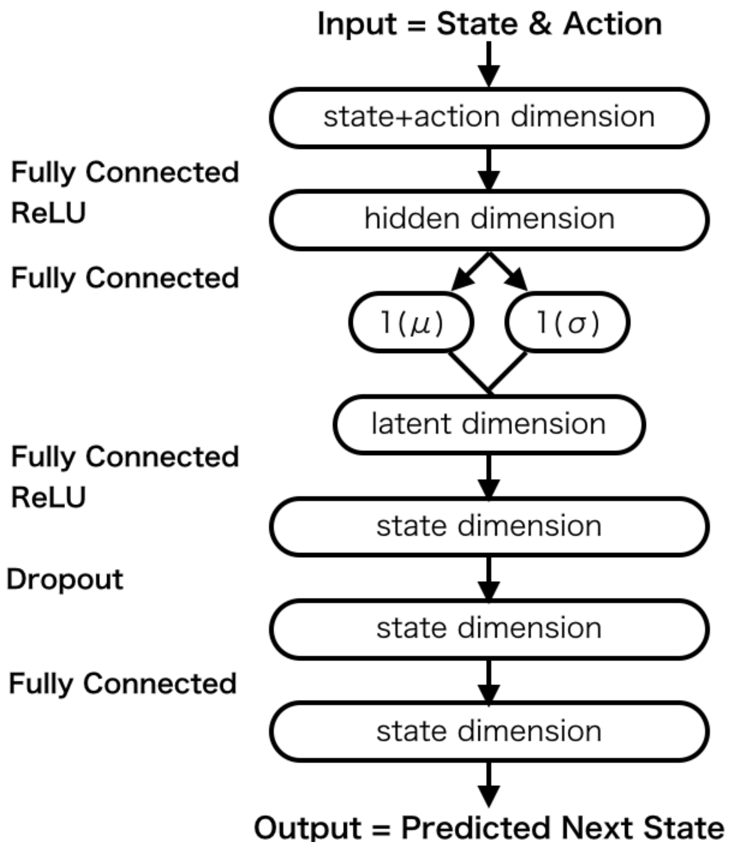

Figure 3.7: State transition model network details.

Hyperparameter	Value	
Horizon (T)	2048	Number of data to collect before the policy update.
Minibatch size for policy training	16	Number of training samples over which parameter update for the policy is performed.
Minibatch size for model training	128	Number of training samples over which parameter update for model learning is performed.
Learning rate for the Actor/Critic	3×10^{-4}	The learning rate used by Adam optimizer.
Discount (γ)	0.99	A discounting weight to compute the advantage.
Hidden size	100	The number of dimension for hidden layer (same for all models)
Generalized Advantage Estimation (λ)	0.98	Hyperparameter fro GAE ranging from 0 to 1.
Clipping parameter (ϵ)	0.2	This value is used to keep the probability ratio r_t in PPO policy update within the range of $[1-\epsilon, 1+\epsilon]$
Dropout rate	0.5	The ratio of randomly dropped nodes. This value is used for both training and inference.
MC dropout iteration	20	Inference is repeated this number so that the variance can be calculated.

Table 3.2: Hyperparameters used in Dyna-PPO. Dyna-PPO has two groups of parameters; parameters for PPO (from Horizon to λ in the table) and those for MC-dropout (dropout rate and the number of iterations). Values are determined empirically based on the original paper Schulman et al. [2017].

3.3.3 Ablation Study

To validate the effectiveness of our confidence-based approach, we conduct an ablation study here. We run Dyna-PPO with and without confidence interval check. We hypothesize that the absence of our confidence check mechanism deteriorates the reward acquired by the agent because our check mechanism filters inaccurate simulations based on the confidence intervals. Fig. 3.6 is the plot of acquired reward with adaptive Dyna-PPO with (purple) and without (gray) confidence interval check in the Hopper environment. We use the same set of random seeds for each run to ensure that the only difference in these two experiments is the existence of the checking mechanism of simulations. This result successfully shows the effectiveness of our approach as the cumulative reward of Dyna-PPO with confidence interval check (purple) outperforms.

3.4 Discussion

We propose a simple yet effective way to improve model-based reinforcement learning performance by utilizing the confidence interval of data distributions which is empirically determined. We assume that values for each dimension in states follow a Gaussian distribution. This is reasonable given that the distribution of the agent's observations through the interactions can be approximated with Gaussian distribution. We utilize a convenient mathematical property of the Gaussian, which is the confidence interval, to our method. On the other hand, utilizing numerical evaluation of uncertainty in our models [Maddox et al., 2019, Loquercio et al., 2020] possibly is a more broadly applicable approach. In the case of using variance as an uncertainty measure, how to normalize a value without hand-tuning for each task is one of the key future directions we plan, given that variance can have many different ranges for each. Another future direction is to investigate our method in environments where the underlying dynamics, such as the state transition dynamics or the reward structure, changes over time - adaptively adjusting whether using models or not makes the agent more robust to such environments. Although the standard Dyna-style architecture can eventually adapt to those environmental changes, the performance could sharply drop until the model re-learns the environment dynamics. We believe that discarding model simulations when simulations drift off from confident intervals leads model-based RL to work towards continual learning problems because it temporally avoids taking those simulations from the model before the change of the dynamics until the model is re-trained.

Chapter 4

Disentangled State Representation for Reinforcement Learning

4.1 Introduction

In deep learning, learning input data representation is a critical research area. How the learned data are represented significantly affects the performances of tasks [Bengio et al., 2012, Lesort et al., 2018], and many different methods have been proposed. In Reinforcement Learning (RL), inputs for an agent can be image frames such as video game screens [Mnih et al., 2015, Johnson et al., 2016, Kempka et al., 2016] or camera observations of a robot [OpenAI et al., 2018, van Hoof et al., 2016]. In such a case, agents have convolutional neural network layers to process RGB images before a feedforward network and learn the optimal behavior. In this architecture, the convolutional and the linear layers are strongly tied together in optimization. Recent works propose to learn a state representation using unsupervised methods first to represent input states in a low-dimensional space and learn the optimal behavior on top of that latent space [Ha and Schmidhuber, 2018, Hafner et al., 2019, Higgins et al., 2017b]. The learned representation consists of some underlying disentangled factors, which are common among the tasks, and it can be transferred to a new task with the same or similar input space. Since the representation and the agent's behavior policy are separately learned, what the agent needs to do in a new task is to re-train a policy on top of the existing learned representation instead of learning the entire network layers again. Further, the policy network's input has smaller dimensions than the original inputs (e.g., images), and the learning process can be quicker.

In recent studies, Variational Autoencoder (VAE, Section 2.4.1) [Kingma and Welling, 2013] and its variants are often employed to learn a compact representation of visual input [van Hoof et al., 2016, Ha and Schmidhuber, 2018, Hafner et al., 2019]. Variational Autoencoder consists of the encoder and the decoder. Given an input, the encoder predicts the parameters for Gaussian distribution with mean μ and variance σ^2. Given those parameters, a latent vector z is drawn from the Gaussian distribution, and the decoder reconstructs the input based on that latent vector. However, the sampling process stops the gradient propagation, and the backpropagation can not be used. VAE avoids this issue by the reparameterization trick. Instead of sampling $z \sim \mathcal{N}(\mu, \sigma^2)$, computing $z = \mu + \sigma^2 \times \epsilon$ makes gradient calculation possible, thus, the loss is backpropagated. GANs, another generative modeling method, which has been actively studied recently, also learn data representations [Chen et al., 2016, Karras et al., 2019]. Because of its adversarial loss, GANs generate sharper images compared to VAE [Goodfellow, 2017]. In addition, GAN is unique that it does not require any explicit density function for its training. However, the GAN has some disadvantages, such as training instability or a unique problem called mode collapse. Among those characteristics, the most important difference between VAE and GAN is that GAN does not learn the latent distribution. The combination of these two, called VAE/GAN, was proposed [Larsen et al., 2016]. This has several unique properties, which we discuss in the following section, and our method adopts VAE/GAN as the representation learning method for our RL agent.

There are several different ways of evaluating the learned representations in the representation learning literature, and it is not always as straightforward as accuracy in supervised learning problems. One intuitive approach is to use the performance of downstream tasks based on the idea that the good representation is the one by which the downstream task is easier to perform [Goodfellow et al., 2016, Higgins et al., 2017b]. For instance, the reinforcement learning agents' performance uses projected embeddings instead of raw inputs. In this paper, we also use this approach for the evaluation metric. Though it sounds natural to use downstream tasks for evaluation purposes, it is worth noting that there are several disadvantages to this approach, especially in RL. First, it requires high costs to run RL agents. Second, it is directly affected by uncertainty in RL agents' performance due to implementation details, hyperparameter selection, or even stochasticity in different runs [Lesort et al., 2018]. As another direction, disentanglement has been studied as one of the beneficial properties of a good representation. The idea of disentangled representation is that latent units of the representation are well-factorized, and each factor is less sensitive to the others [Bengio et al., 2012]. Figure 4.1 is the illustration of the idea of disentanglement. Here, the input is a video game frame from the car racing game, and the learned representation is a 4-dimensional vector. If, for instance, the first dimension of the vector expresses the car's position, the second dimension describes rotation, the third dimension corresponds to acceleration, and the last dimension is related to the road angle, respectively, this learned representation is well-disentangled. The data consists of combinations of these factors. These disentangled properties makes it possible to manipulate the generated data [Higgins et al., 2017a, Li and Mandt, 2018]. Disentanglement has been considered to benefit different tasks and studied [Ridgeway, 2016, Bengio et al., 2012], however, the lack of measurements for this disentanglement has been a challenge, and some methods were proposed, such as a classifier-based approach in β-VAE [Higgins et al., 2017a] or an information-theoretic approach in β-TCVAE (Total Correlation Variational Autoencoder) [Chen et al., 2018]. However, they assume that the ground factor of the data is available [Liu et al., 2015, Matthey et al., 2017],

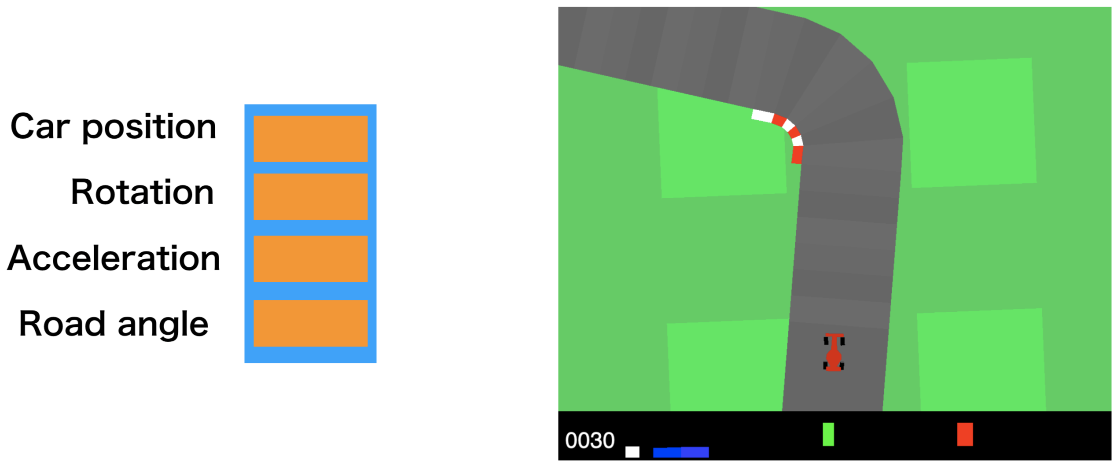

Figure 4.1: Example of disentangled representation. The learned representation in this example is a 4-dimensional vector, and each dimension corresponds to car position, rotation, acceleration, and road angle, respectively.

which is not a realistic assumption in many cases. InfoGAN, another information-theoretic approach, learns disentangled representations in an unsupervised way, but the evaluation is non-numeric, in which they vary only one latent factor and observe the corresponding change of a generated image [Chen et al., 2016]. Yet, the benefits of having a disentangled representation for an RL agent is not clear. In this paper, we evaluate our representation learning methods from the RL agent's performance point of view and disentanglement properties.

4.2 Methods

4.2.1 VAE/GAN

Although VAE shows the remarkable capability to learn data-generative factors, it computes element-wise loss, which is not convenient for some applications. For example, in control tasks where the input is image frames, a slight perturbation of a pixel typically does not matter for the objective (e.g., grasping a glass) but affects the loss. In the first place, it is computationally heavy. [Larsen et al., 2016] proposed to use a similarity metric to mitigate this problem. They use a discriminator for this purpose. As shown in the previous section, the discriminator's task is to distinct real and fake images. The output of the network can be used to determine how similar the generated images are. [Higgins et al., 2017b] replaces the reconstruction loss in the VAE objective with $\mathbb{E}_{q_\phi(z|x)} \|J(\hat{x}) - J(x)\|_2^2$ where $\hat{x} \sim p_\theta(x|z)$ and $J \in \mathbb{R}^{W\times H\times C} \to \mathbb{R}^N$ that maps the high-dimensional features into low-dimensional space of N deriving from the same motivation. They uses a pre-trained denoising autoencoder [Vincent et al., 2010] as the function J, and compare the z for the input $\hat{x}$ and x. We adopt the discriminator for our similarity measurement scheme but also use the latter idea as feature matching, which is described later in this section. Anther benefit of this architecture is the generated image quality. VAE works well to capture latent vector z, however, since there are many possible images considered for next states, an image generated by this family typically blurry. On the other hand, GANs are good to produce sharper images as the generator needs to generate a realistic one particular image for each instead of considering every possible images to minimize mean squared error. GANs architecture, however, the generator in particular has one drawback which is that the network doesn't have any way to learn z itself. z is sampled from a normal distribution randomly. Because of these missing points in each of VAE and GAN, we apply a model similar to VAE/GAN architecture as a part of the agent's vision. In this architecture, the generator plays a role of the decoder for VAE. The benefits of this architecture is two fold: 1)this generates more clear image than VAE, and 2)this has a latent vector z captures a real input instead of drawing a sample from a normal distribution. Further, they found that VAE/GAN can disentangle factors underlying the data. [Radford et al., 2016] also indicated that the latent space learned by GANs factorize some factors in it.

There are some drawbacks for VAE/GAN as well. First, computational costs are higher than GAN or VAE as it combines two of them. Second, GAN's training instability needs to be considered. Further, "No ground truth factor needed" can be a drawback, to put it the other way around because this limit the interpretability of the learnt latent representation. In our

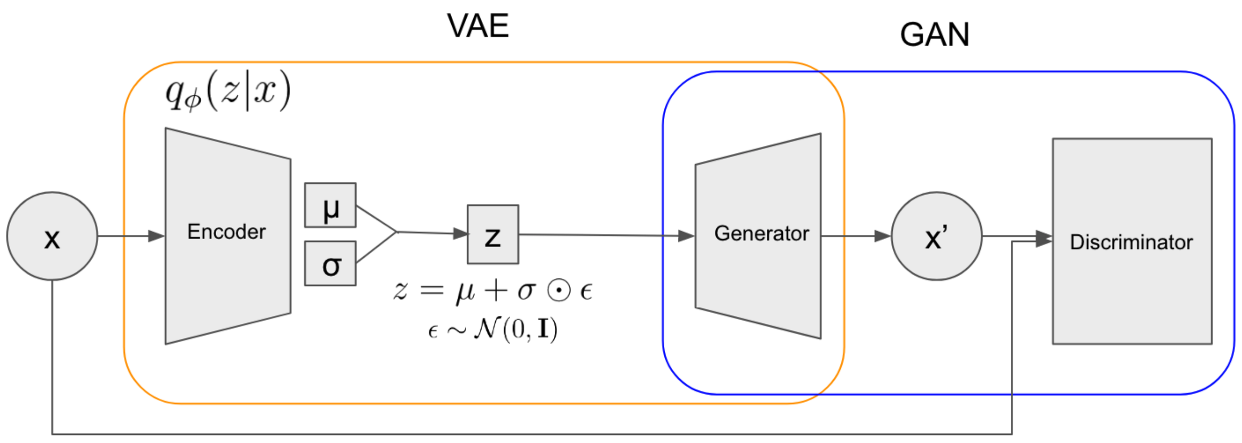
VAE
GAN
$q_\phi(z|x)$
x
Encoder
μ
σ
z
$z = \mu + \sigma \odot \epsilon$
$\epsilon \sim \mathcal{N}(0, \mathbf{I})$
Generator
x'
Discriminator

Figure 4.2: VAE/GAN network architecture. Input data is fed into the encoder first. The encoder outputs parameters for a Gaussian distribution (μ and σ), and a latent vector z is drawn from the distribution parameterized by them. Instead of a decoder in the standard VAE architecture, the generator generates data x' in VAE/GAN. The generated data is then fed into the discriminator along with the original input x. The discriminator determines whether an input is real or fake.

network, ”feature matching” is used to mitigate GAN's training instability [Salimans et al., 2016]. This technique is improve the generator's image generation quality by feeding x and $\hat{x}$ into the discriminator and minimizing mean squared errors between outputs of intermediate layer's output as possible instead of the outputs itself. This direct the generator to generate an image of which the discriminator shows similar output patterns in its intermediate layer. We found that this also works for ”mode collapse” problem. In this regard, our loss function is changed as follows:

$$\left\|\mathbb{E}_{x\sim p_{data}} f(X) - \mathbb{E}_{z\sim p_z(z)} f(G(z))\right\|_2^2 \tag{4.1}$$

where $f(x)$ denote activation on an intermediate layer of the discriminator. Not only the instability but also the technique help to reduce computational costs by not having pixel-wise comparison. The training of VAE/GAN is two-folds: VAE training and GAN training. VAE training phase is the same as the original proposal [Kingma and Welling, 2013] with Gaussian prior. In this phase, the discriminator tries to classify whether the input is the original X or the reconstruction $\hat{X}$. The loss is a squared loss of activation patterns of the layers, which leads the discriminator to be activated the same for actual data and generated data.

$$-(y\log(p) + (1-y)\log(1-p)) \tag{4.2}$$

The whole training process is described in Algorithm 2.

Algorithm 2 VAE/GAN Training

```
1: initialize model parameters for $\theta_{Enc}, \theta_{Dec}, \theta_{Dis}$
2: while Training do
3:     get input batch $X$ from replay buffer
4:     $Z \leftarrow Enc(X)$
5:     compute $KL$-divergence
6:     $\hat{X} \leftarrow Dec(Z)$
7:     compare layer activation pattern in the discriminator for $X$ and $\hat{X}$
8:     $Z \sim N(0,1)$
9:     $\hat{X} \leftarrow Dec(Z)$
10:    compute discriminator loss
11:    compute generator loss using feature matching
12:    update parameters
13: end while
```

4.2.2 Evaluation of Representations

A unique challenge of representation learning in combination with downstream tasks is evaluating learned representations. In classification tasks, for example, a network is evaluated numerically using a misclassification rate. Simply assess the representation from the synthesized data, in particular, visual quality for image inputs, does not necessarily reflect our aim well.

Downstream Task Performance One intuitive approach is to use the performance of the downstream task itself as the evaluation metric for the representations. For example, if an RL agents' with a particular representation performs better than that with other representation, that specific representation is better. However, in representation learning, in particular, it is challenging to evaluate the learned representation itself.

Disentanglement Measurement One possible approach to evaluate representations is to consider disentanglement. As a factorized representation, which means each factor in the latent representation is only sensitive to changes in a generative factor, which corresponds to the factor, while it would not for the changes of changes in other data generative factors [Bengio et al., 2012, Higgins et al., 2017a]. There are several approaches proposed to evaluate disentanglement in learned representations along with different network architectures. One of the early work, which is based on GANs, is InfoGAN Chen et al. [2016]. GAN's objective is defined as min-max game shown in Equation 2.15, and InfoGAN proposed to modify the game formulation as

$$\min_{G} \max_{D} V_I(D, G) = V(D, G) - \lambda I(c; G(z, c)) \tag{4.3}$$

where c is the latent code, which corresponds to the structured latent factors in the data distribution. Using the mutual information $I(X; Y)$, which measures how much information the Y provides in terms of X, the proposed method adds a term $I(c; G(z, c))$. λ is a hyper-parameter to control the weight for that mutual information term. Their evaluation method is an example of qualitative evaluation of representations. The idea is that projecting the code onto data space while changing a factor one by one and observing the generated data provides how each factor in the latent code corresponds to some factors in the data. They used MNIST, 3D Faces [Paysan et al., 2009], and 3D Chairs images [Aubry et al., 2014] for these experiments and showed that manipulation in the latent code results in the change of the corresponding factor in the images. On the other hand, there are several VAE-based approaches have been proposed. β-VAE [Higgins et al., 2017a] added a hypterparameter β

as a weight for the KL term in VAE loss as follows:

$$L = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta KL(q_\phi(z|x)||p(z)) \tag{4.4}$$

where $p(z)$. β constrains how much the loss imposes on latent factors to be statistically independent [Burgess et al., 2018]. This corresponds to the standard VAE when $\beta = 1$. Since the method is based on VAE, it is more stable than InfoGAN to train. They proposed a linear classifier-based approach to evaluate the representation from the disentanglement point of view. It fixes one of the generative factors and randomly samples all others. The goal of the classifier is to predict the index y of the generative factor that was kept fixed. To conduct the experiments, they also created a data set called dSprites [Matthey et al., 2017], where the data is generated based on color, shape, scale, rotation, x and y positions, which enables us to control generated data and fixed factors for each shape, thus, true labels for the classifier as mentioned above are provided. However, β-VAE has a trade-off between the reconstruction quality and disentanglement. The reconstructed image quality is worse when a larger β is specified, although it leads the learned representation to be disentangled. Thus, it tends to generate blurry images when β increases. Based on the β-VAE, [Kim and Mnih, 2018] proposed FactorVAE, which mitigates the drawback of the β-VAE. They use Total Correlation (TC) [Watanabe, 1960], $KL(q(z)||\overline{q}(z))$ where $\overline{q}(z) = \prod_j q(z_j)$, as a penalty to the VAE objective. TC measures the independence of each factor by computing the KL divergence between a product of each distribution and the joint distribution of them. In their study, they also proposed a metric for their representation based on the classifier-based approach in [Higgins et al., 2017a].

Further, on top of β-VAE approach, the information theoretic approach to evaluate disentanglement was proposed in [Chen et al., 2018] called Mutual Information Gap (MIG).

We first evaluate VAE/GAN with the dSprites dataset [Matthey et al., 2017]. Then, RL task's performance is evaluated. By showing these two, we can relate the disentangled representation helps the RL agent to improve its performance.

4.3 Experiments and Results

We evaluate our method using a car racing game environment from OpenAI Gym [Brockman et al., 2016]. In this environment, the car position on the screen is fixed to the bottom center, and the environment changes over time with the agent's action. The action is a 3-dimensional continuous vector, each of which corresponds to steering, where -1 equals to fully left, and +1 is fully right, acceleration, and breaking. Observation is RGB images of size (96, 96, 3). Other information, such as true speed, ABS sensor reading, steering wheel position, and gyroscope value, is shown at the bottom left. However, it is not provided as observed values and needs to be read and processed separately from the frames. As a preprocessing, these frames are resized to (64, 64, 3) in our experiments. Further, the additional information described above is not used in our experiments; thus, the agent learns the policy only from image frames of the track. The reward of -0.1 is given every frame, and +1,000/N is given for every track tile the agent visits, where N is the total number of tiles in the track. In addition, if the agent drifts off for a certain amount, a -100 reward is given, and the game is terminated as the agent died.

4.3.1 Qualitative Evaluation

We first generate 2,000 rollouts using a random policy and train VAE and VAE/GAN using that data in unsupervised way. The figure 4.4 compares generated images using VAE (left) and VAE/GAN (right) after 200 epochs of training. Although the adversarial loss provides sharp reconstruction [Goodfellow, 2017], both VAE and VAE/GAN generate qualitatively

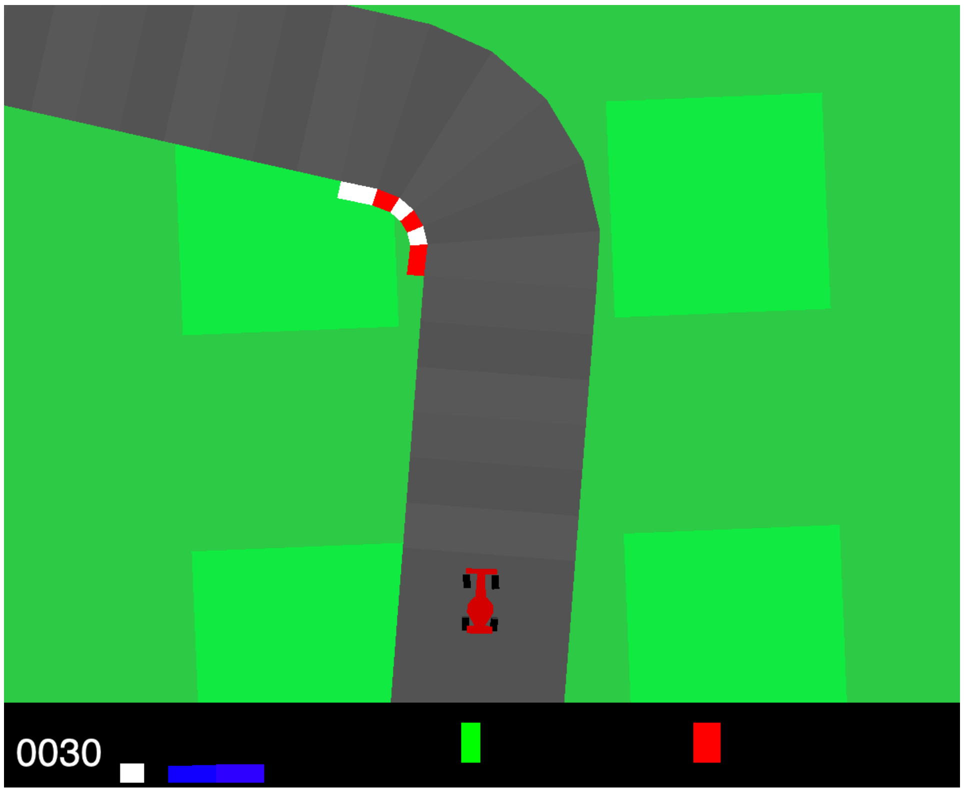

Figure 4.3: CarRacing-v0 environment in OpenAI Gym. The car (agent) location is fixed to the center bottom, and the scene changes based on the agent's actions. There are some indicators in the black area at the bottom. From left to right: true speed, four ABS sensors, steering wheel position, gyroscope.

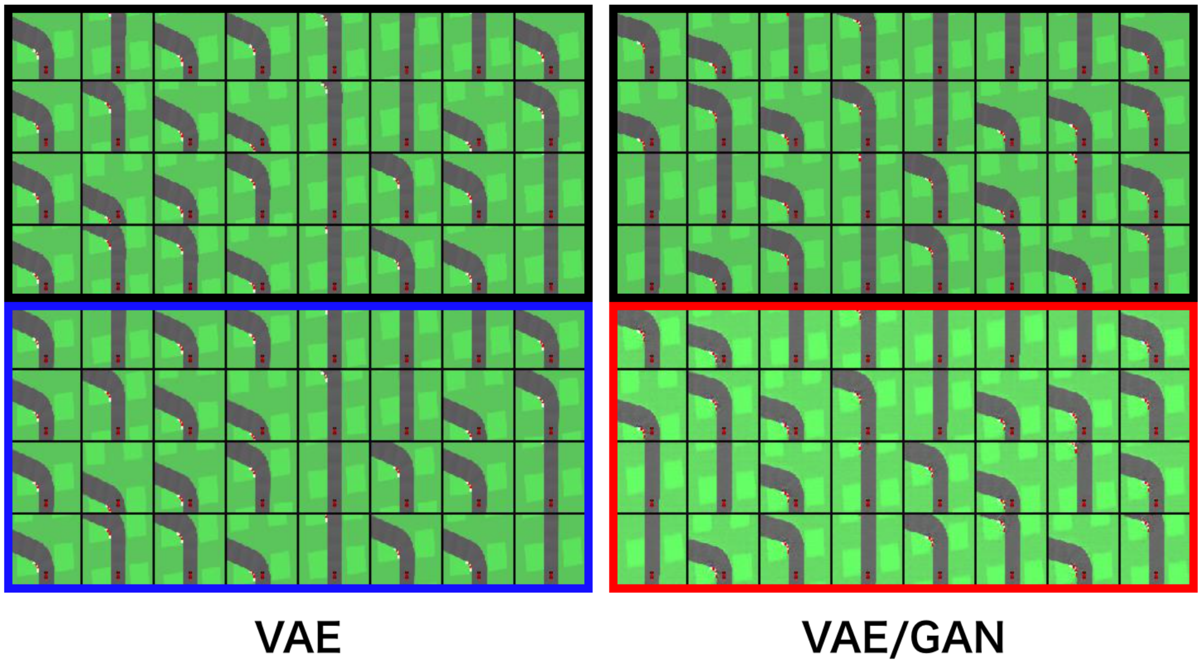

Figure 4.4: Reconstructed image comparison. Top four rows are original input images and the bottom four rows are reconstructions for each. Both VAE and VAE/GAN reconstruct images equally well.

equivalent images for this 2D game environment.

To confirm how the latent vectors are expressed, we project them onto a 2D surface using t-SNE [van der Maaten and Hinton, 2008]. The figure 4.5 shows the results. Left is the result of VAE, and the right is VAE/GAN. The VAE/GAN's result shows some clear clusters, such as corner images at the bottom left cluster or straight courses at the upper left part in the biggest cluster. Some other examples are shown in the figure 4.6. Although both VAE and VAE/GAN generate good images, the projection shows that the learned latent representations are quite different. We hypothesize that RL agents benefit from this separation in the latent space. In addition to the visual inspections in this section, we confirm the hypotheses numerically in the next section using several different approaches.

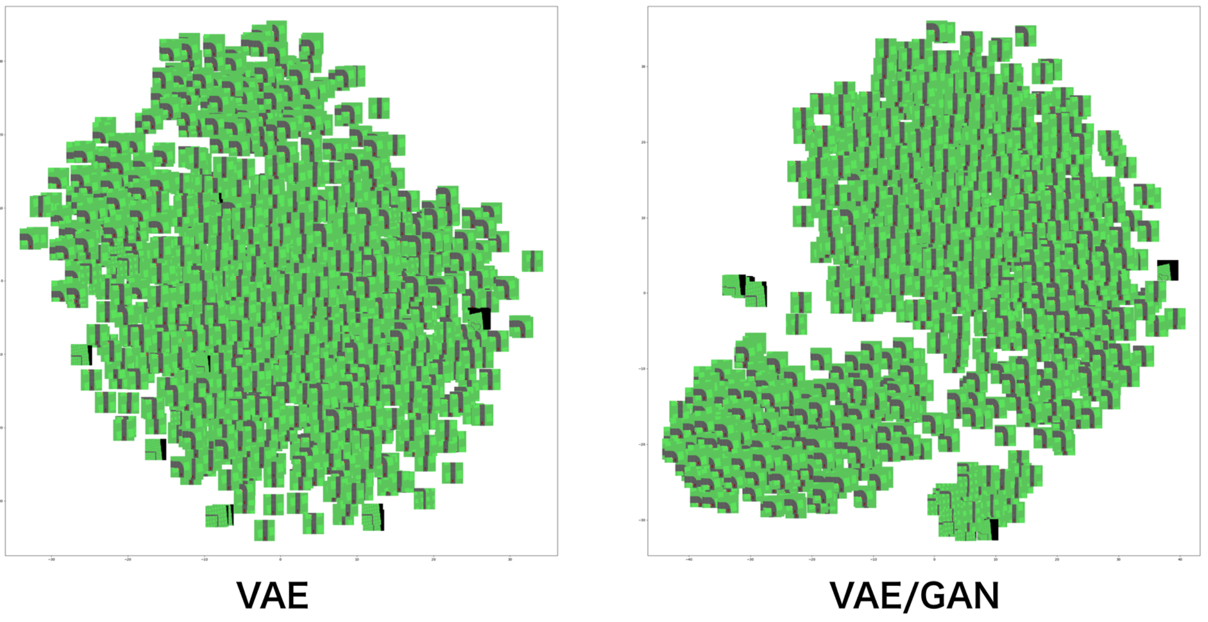

Figure 4.5: Latent vector mapped on 2D plane using t-SNE. The left figure (VAE/GAN) has some clear clusters with similar images. Based on this observation, we conjecture that the VAE/GAN embedding could benefit RL agents.

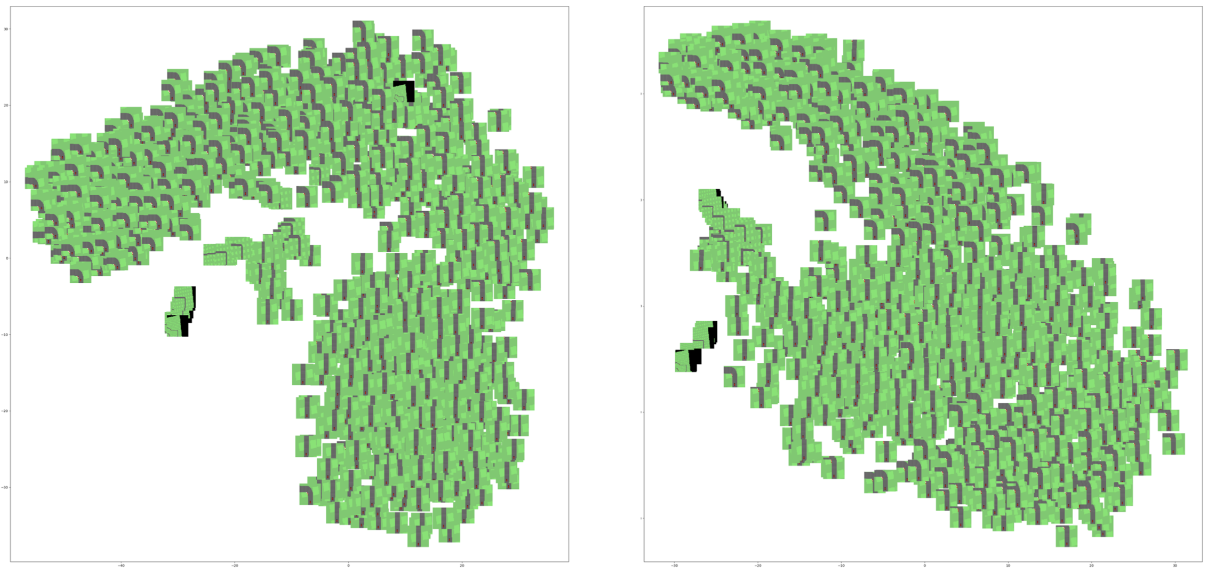

Figure 4.6: Some other examples of t-SNE projection of VAE/GAN embedding. There are several clusters in these examples as well.

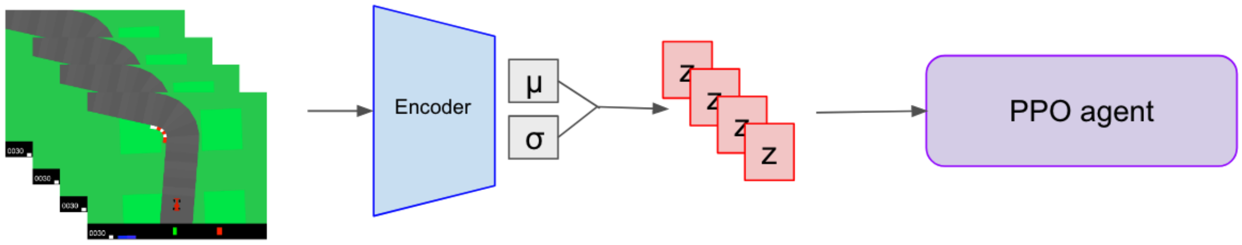

Figure 4.7: Experimental setup for the downstream task.

4.3.2 Quantitative Evaluation

Several approaches have been proposed to evaluate representations [Lesort et al., 2018]. Here, we first evaluate our representation by a downstream task performance. Then, conduct some experiments using disentanglement measures and discuss the results.

Task Performance

In the previous section, VAE and VAE/GAN are trained using the collected images. To confirm how effective those representations are for RL agents, we train the RL agent using those representations. Instead of row pixel inputs, the agent observe the latent vector of input images, and learns a policy based on them. z is 16-dimensional vectors in this experiment. In this experiment, four consecutive inputs are concatenated and form one input for the agent as shown in Figure 4.7. For policy learning, we use Proximal Policy Optimization [Schulman et al., 2017]. For our experiment, we stacked four consecutive frames. This technique is known as "frame stacking" and is widely used in game-playing RL agent research. Stacked frames give an agent useful information, such as the head-direction of the agent itself in the CarRacing environment. A selected action is repeated eight times in our setting. The network consists of convolutional networks followed by batch normalization [Ioffe and Szegedy, 2015] and ReLU [Nair and Hinton, 2010]. Details are in Figure 4.8 and hyperparameters used in the experiments are in Table 4.1.

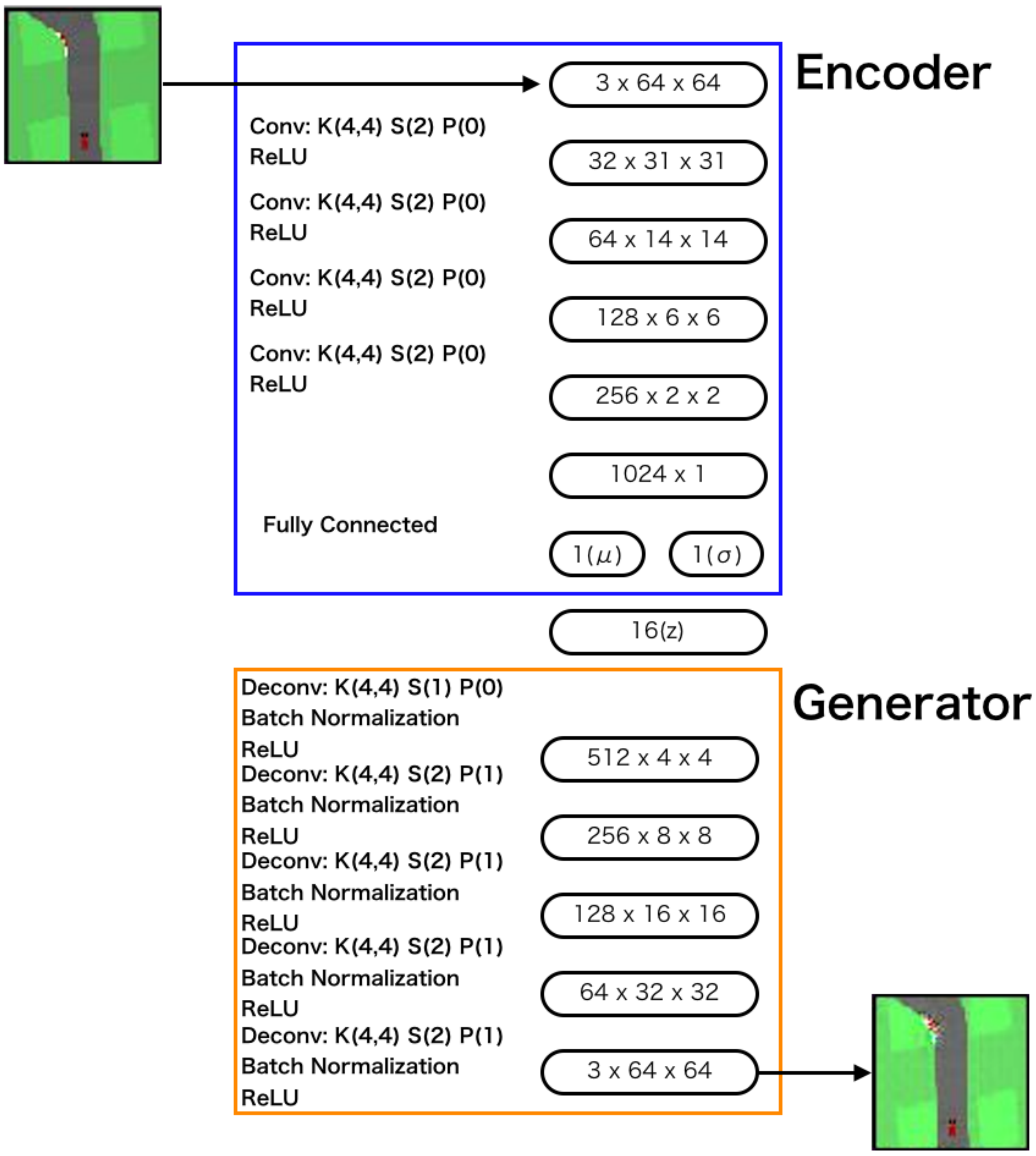

Figure 4.8: VAE/GAN network details. K, S, and P are kernel size, strides, and padding, respectively.

Hyperparameter	Value	
Latent dimension	16	Number of latent code (z) dimension.
Input screen size	64	Resize the screen size to 64 in the experiments from the original screen size of 94.
Minibatch size for VEA/GAN training	100	Number of training samples over which VAE/GAN parameter update is performed.
Learning rate	1×10^{-4}	The learning rate used in the experiments. The number is the same for all three networks: Encoder, Generator, and the Discriminator.
Learning rate for the policy learning	1×10^{-3}	Learning rate for policy learning.
Discount (γ)	0.99	A discounting weight to compute the advantage.
Horizon (T)	2000	Number of data to collect before the policy update.
Minibatch size for policy training	128	Number of training samples over which parameter update for the policy is performed.
Action repeat	8	Number to repeat the agent's action.
Image stacking	4	The number of sequential frames for the agent's input.
Clipping parameter (ϵ)	0.1	This value is used to keep the probability ratio r_t in PPO policy update within the range of $[1-\epsilon, 1+\epsilon]$

Table 4.1: Hyperparameters used in VAE/GAN training for CarRacing environment. Values are determined empirically.

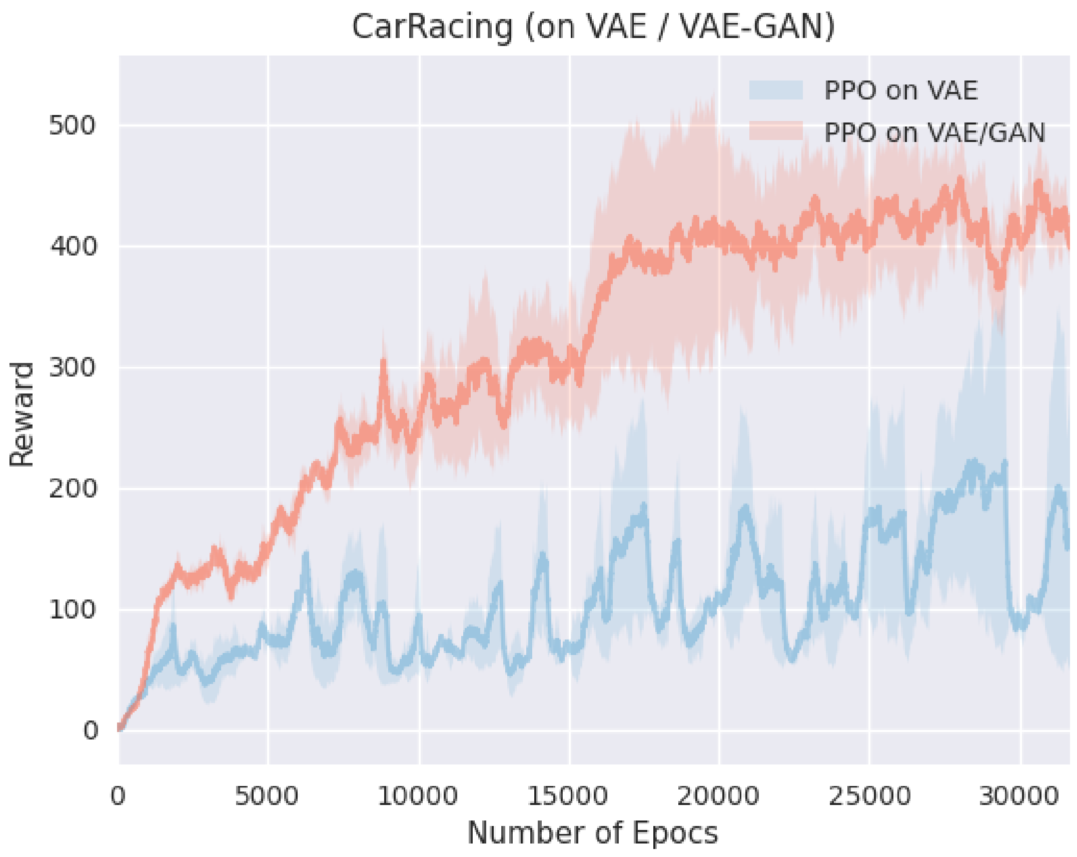

Figure 4.9: Performance comparison using PPO. The blue line is the PPO on the latent vector generated by VAE and the red line is on the latent vector generated by VAE/GAN. Values are the average of three runs with different random seeds.

Figure 4.9 shows the experimental results. We conducted three different runs for each with different random seeds and plot their average. The result indicates that the RL agent learns its policy using VAE/GAN encoding outperforms the one with VAE encoding. Although the performance of state-of-the-art RL agents in the CarRacing game environment can be much better, the result is a positive indicator as a comparison of VAE and VAE/GAN representations.

Disentanglement Measure

In addition to the task performance, we conduct experiments from the disentanglement point of view. We evaluate VAE/GAN using a classifier-based method proposed in [Higgins et al., 2017a]. However, it requires that the data generative factors be known beforehand, which is unavailable for our tasks. Thus, we train VAE and VAE/GAN on dSprites data set Matthey et al. [2017] and compare the performance. First, we generate data while fixing one factor and randomizing others. The data is fed into the encoder and latent codes are generated. Then, difference between pairs of encoded vectors are calculated, and we calculate the average of those differences, which is the input for the classifier. The classifier's task is to determine which factor is fixed for that data generation process (Figure 4.10).

We implemented a simple multi-layer perceptron with two linear layers, followed by a ReLU activation layer for each and a softmax layer for the output. Figure 4.11 shows the classification error (cross-entropy) transition throughout the 300 epochs of training. The metric value has randomness as it is affected by the data generation and the classifier training procedure. Thus, the value is an average of 10 runs with different random seeds. The graph shows that the VAE outperforms VAE/GAN, which contradicts our assumption given that VAE/GAN outperforms VAE in task performance. We conjecture the reason for this results for several reasons. First, Larsen et al., evaluated their representation qualitatively in CelebA data set [Liu et al., 2015], but it is not clear whether the reconstructed image space correlated to the latent space. Indeed, it is know that the reconstruction quality trade-off with the disentanglement as shown in [Higgins et al., 2017a]. Thus, the relationship between factors in image spaces and the latent space dimensions is further needs to be investigated. Secondly, there are mismatches between several different metrics [Zaidi et al., 2020]. Thus, it is possible that the representation which shows a good performance in one metric performs poorly for another metric. Further, the data set for task performance and disentanglement evaluation are different as the metric requires data generative factors to compute it. There is no de

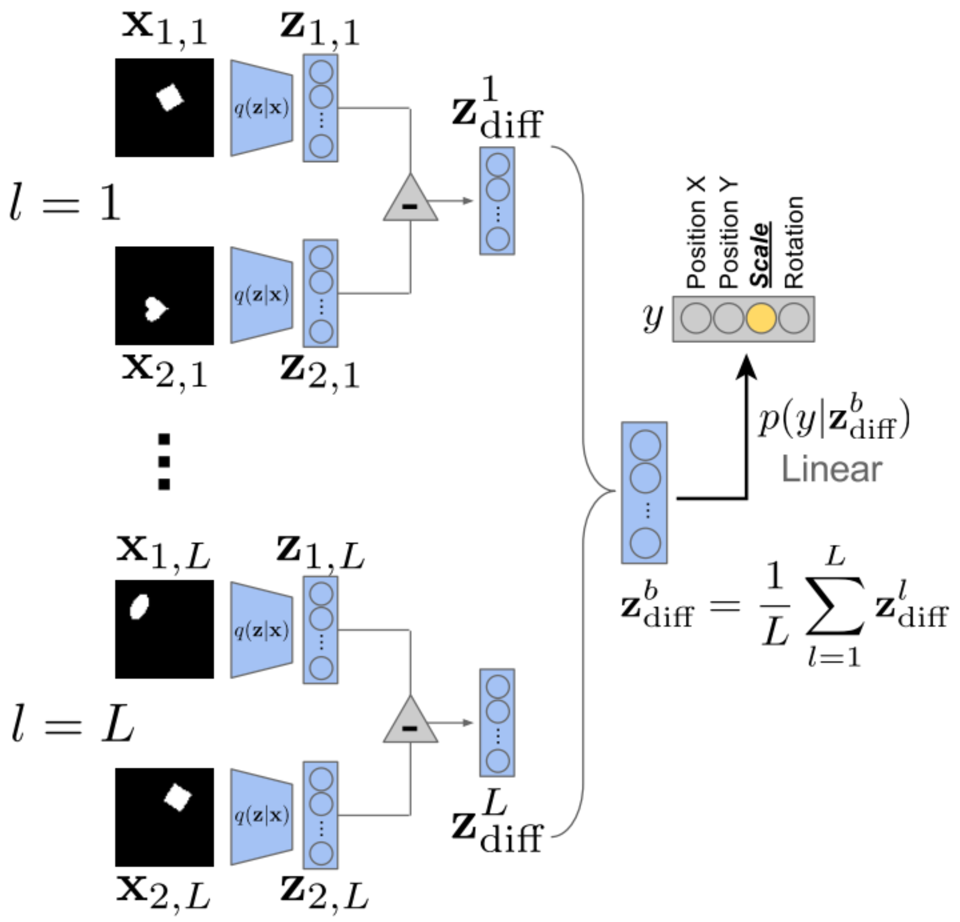

Figure 4.10: Classifier-based disentanglement metric proposed in [Higgins et al., 2017a]. The figure is adapted from the paper.

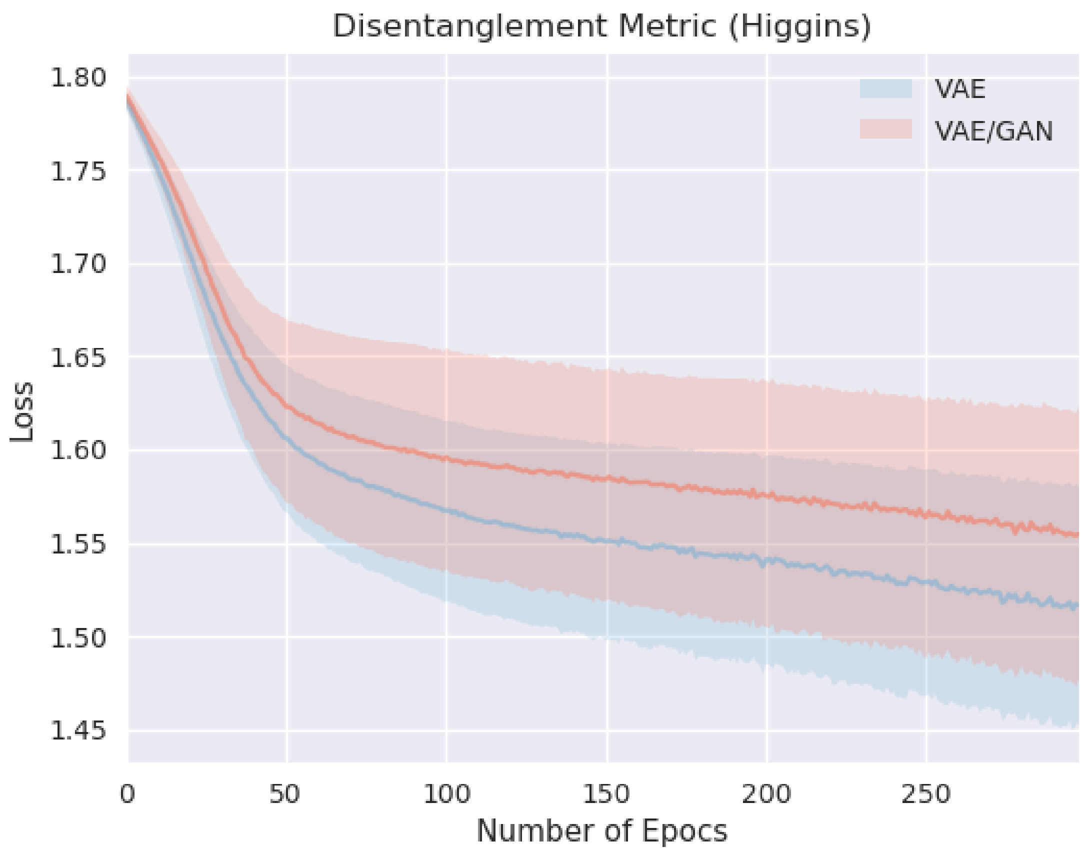

Figure 4.11: Metric score with 300 epochs of training. A classifier is an multi-layer perceptron with two linear layers.

facto standard to measure disentanglement in the field, and each research comes with a set of new representation learning method and metric, which is best suitable to capture the benefit of the representation. In such situation, there is a claim that challenges the idea of unsupervised learning of disentangled representations and ignited active discussions about whether it is possible to learn disentangled representations in an unsupervised way or not (or it is possible only when it comes with strong assumptions) [Locatello et al., 2018, Horan et al., 2021]. Given the area's current situation, further development of our understanding of the disentanglement and its metrics is necessary to fairly assess existing and newly proposed methods. In addition, searching for the effects of varying weights for each factor in the loss functions is another thing to analyze. As β-VAE puts the weight β to the KL-divergence term and shows that it changes the disentanglement capability of the model, detailed analyses for VAE/GAN from the loss function perspective are also a critical step for further understanding.

4.4 Conclusion

In this project, we investigated VAE/GAN as the RL agents' visual processing module and showed its potential benefit based on the car racing game's task performance. In addition, we evaluated the learned representation from the disentanglement point of view, and our results depicted some disparity between the downstream task performance and the disentanglement. Further, we surveyed the current situation of disentanglement metric research and highlighted issues in the field. Although it has been believed that the factors of the learned disentangled representation are beneficial when two or more tasks have common properties such as scaling, orientation, or even more specific factors like road or car, understanding what we measure in reality and developing the common methodology for evaluation, instead of the current "one metric per each proposal" is necessary. On top of it, one direction that needs to be evaluated is the transfer learning performance of the disentangled representation in RL tasks. We evaluated our approach using the 2D car racing simulation game, of which the original image quality is coarse and low. Testing the VAE/GAN-based RL agent in richer input image tasks is another interesting direction. GANs typically generate more realistic images and can benefit the agent to augment the data, as we discuss in the future directions chapter.

Chapter 5

Future Directions

In this chapter, we discuss future directions. Aside from the main projects that introduced in the previous chapters, we investigated some other research projects. Some preliminary experiments are discussed, along with the future directions.

5.1 Uncertainty-based Adaptive learning

When we consider real-world applications, RL agents' tasks may not be limited to a single task with a limited length of episodes, such as navigation in a constantly changing environment. For example, Sutton introduced the *Blocking Maze* and *Shortcut Maze* experiment in [Sutton and Barto, 2018], where the position of walls changes and the shortest path changes at some point of iterations. The agent needs to update its policy in this environment to catch up with the latest situation. This can happen indefinitely in real-world scenarios due to road constructions, big events, or parked cars blocking the learned shortest path. Effective exploration strategies are required to quickly adapt such environmental changes. Dyna-Q+ [Sutton and Barto, 2018] is one of the classical approaches to making an agent, which can quickly catch up with changes in the environmental dynamics by encouraging the agent to test state-action pairs that have not been tried for a while. Although this exploration strategy could eventually update the optimal actions, it is not sample-efficient. The agent needs to execute it, especially suppose that the change happens after the learning rate converges to a tiny constant. In that case, the agent must take that exploratory action many times until the value catches up with the latest situation. From this perspective, we focus on the learning rate and discuss the complementary solution for the challenge above. In Chapter 3, we proposed the RL agent architecture, which uses the uncertainty of models to determine whether the agent should rely on the model simulations or not. We investigate an approach to use the uncertainty to make our RL agent update its learning rate, and promote faster learning when the learned behavior needs to adapt to the environmental changes. In Dyna-Q+, for instance, the learning rate is typically decreased over the iterations, making it slow for RL agents to adjust their behavior to the new environmental dynamics even when it encounters a novel state.

There are several approaches regarding the learning rate update. Simple decaying is one classical approach, which works for stationary environments, but does not suit our tasks here. In addition, those methods require tuning in hyperparameters, such as an initial rate or a total number of steps by the learning rate converging to a fixed constant. Typically they are determined empirically. Another approach is the adaptive strategies such as the incremental delta bar delta (IDBD), temporal coherence learning (TCL), or Autostep [Sutton, 1992, Mahmood et al., 2012, Beal and Smith, 1999, Dabney and Barto, 2012].

The uncertainty-based approach has potential to making continual learning more efficient with the dynamic learning rate update. The idea is dynamically to adjust a learning rate using uncertainty. For example, the agent adjusts its learning rate as follows using the entropy of action probability:

$$lr = base\_lr + (entropy\ of\ action\ probability - const) \times uncertainty\_weight \tag{5.1}$$

where *base_lr* and *uncertainty_weight* are constant positive real variables. Here, we interpret the entropy of action probability distribution as the uncertainty. We conduct simple experiments using this learning rate update scheme and present the preliminary results below.

Two Bandit Machines Problem To show how the proposed learning rate adjustment works, we think about the problem of two bandit machines here. In this experiment, we consider two bandit machines, called A and B, with different initial reward probabilities 0.8 and 0.2 respectively. The agent action is to select one of them, and the objective is to maximize the expected total reward in a given number of iterations. After some iterations, 1,500 iterations in our initial experiment, the reward probabilities of each machine are flipped to 0.2 and 0.8; thus, the agent's optimal behavior must be changed. We expect that the adaptive learning rate scheme will help in two directions. First, the agent's action probability distribution is less peaky because it has not learned much yet, and a high entropy value increases the learning rate. Second, assume that the action probability distribution looks peaky in some states where a specific action leads the agent to the goal. When the reward structure change after the agent has learned the optimal behavior in the previous reward structure, the agent face situations where the optimal behavior is not optimal, and action probability become less peaky as the policy is updated. In this situation, the learning rate gradually increases, which accelerates the agent's adaptation speed. We compared the results using a Q-learning agent with and without a dynamic learning rate update scheme. The figure 5.2 is the plot of the probability to select machine A. 'Static' results from a fixed constant learning rate approach and 'Dynamic' is for the dynamic learning rate update

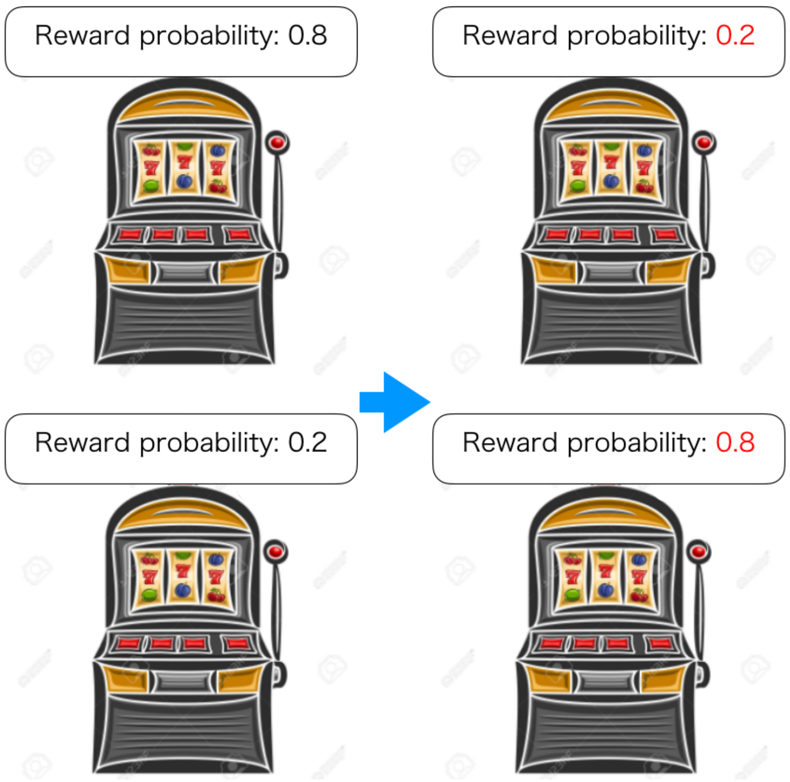

Figure 5.1: Bandit machine experiment with reward structure flipping. Initially, two bandit machines have a reward probability of 0.8 and 0.2, respectively (left). After some iterations, the reward probability is changed to 0.2 and 0.8, respectively (right).

approach. Starting from the reward probability of 0.8, the reward structure change to 0.2 for selecting machine A after 1,500 iterations. We see two observations here: 1. the Dynamic scheme reaches closer to 0.8 in the first half, and 2. the Dynamic scheme also gets closer to 0.2 faster than the Static in the last half of the experiment after the reward probability is swapped. The result indicates that dynamically updating the learning rate based on the action probability entropy makes the agent quickly adaptable to environmental change. To confirm that the results mentioned above are not caused by the learning rate of the static approach, we tested multiple learning rates ranging from 0.001 to 0.005 and compared them with the dynamic learning rate update mechanism. Figure 5.3 shows that the dynamic learning rate mechanism outperforms the static approach of any tested learning rate. This simple experiment shows the potential of this dynamic learning rate approach in environments where the agents keep working for a long time and adapt to the environmental change over time as quickly as possible. It is an interesting direction to develop a method for large-scale and more complicated tasks and evaluate it. RL agents with such capability will benefit the real-world applications stated above.

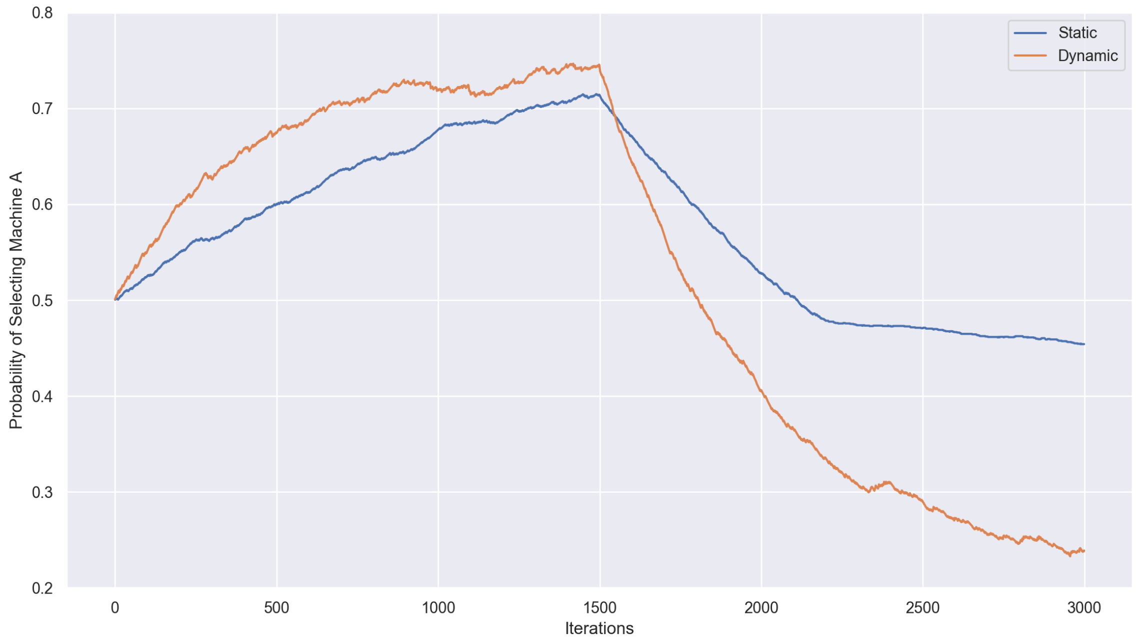

Figure 5.2: Bandit machine experiment results. The plots show the probability of selecting machine A, which initially has a reward probability of 0.8, and it is updated to 0.2 in the second half of the experiment. 'Static' refers to the agent with a constant learning rate. On the other hand, 'Dynamic' corresponds to the agent with a dynamic learning rate update mechanism. Both eventually get close to the optimal probability, but the dynamic mechanism helps to adapt the situation quickly.

One issue that needs to be addressed is that the uncertainty in the action selection itself does not capture aleatoric uncertainty. The agent needs to consider both to assess the uncertainty more precisely.

5.2 Generalizing RL agents with Data Augmentation using Models

Data augmentation is a widely used technique in computer vision, and recently it has been studied to apply to other domains such as natural language processing [Shorten and Khosh-

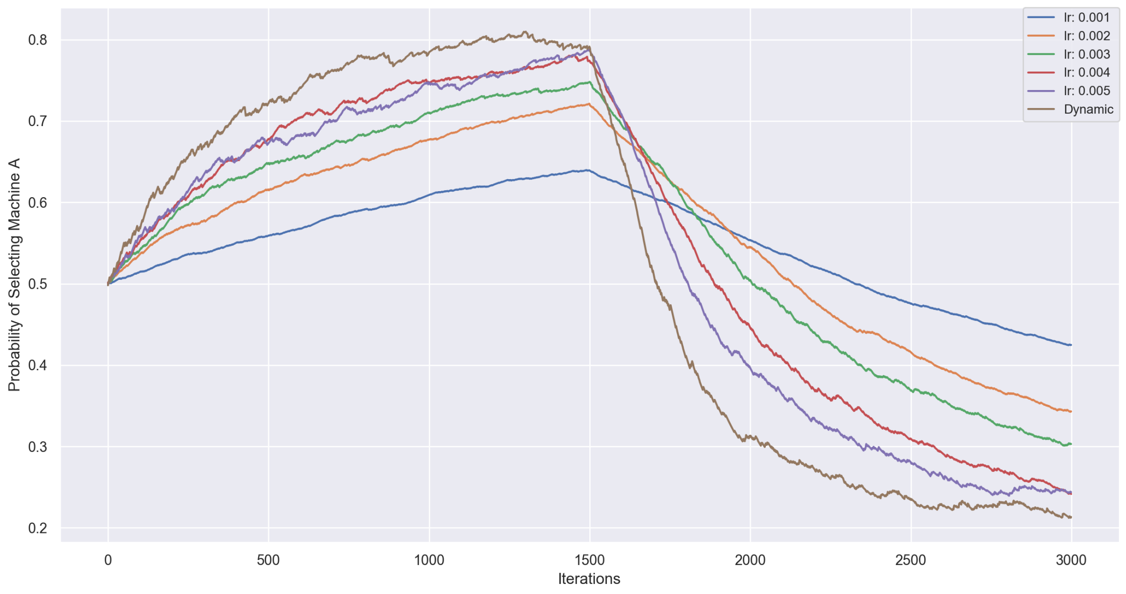

Figure 5.3: Comparison of different learning rates. Run multiple experiments with different learning rates from 0.001 to 0.005 and the dynamic learning rate. The dynamic scheme gets close to 0.8 first, and after the reward structure change (1,500 iterations), it also reaches the updated value fastest.

goftaar, 2019, Feng et al., 2021] and built into popular libraries [Abadi et al., 2016, Paszke et al., 2019, Wu et al., 2019]. It is known that augmenting a data set with some transformations, such as rotating, scaling, cropping, flipping vertically or horizontally, changing luminary, or adding some noise, improves the generalizability of a model, and leads the model to better testing performance. Not only for the generalizability, but it is also useful to simply add more data when it is hard to acquire a large amount of data. Generative models have been considered to be helpful for this purpose as it synthesizes new data from a learned distribution [Kingma et al., 2014, Sandfort et al., 2019, Sundaram and Hulkund, 2021]. Effectiveness of data augmentation in RL has also studied [Yarats et al., 2021].

In the VAE/GAN project, we proposed to use the combination of VAE and GANs to serve as the vision module for the RL agent. From the image generation point of view, this architecture has two benefits: 1. the generated images tend to be sharper than the reconstructions from VAE, and 2. it has the encoder, which the original GAN architecture does not have.

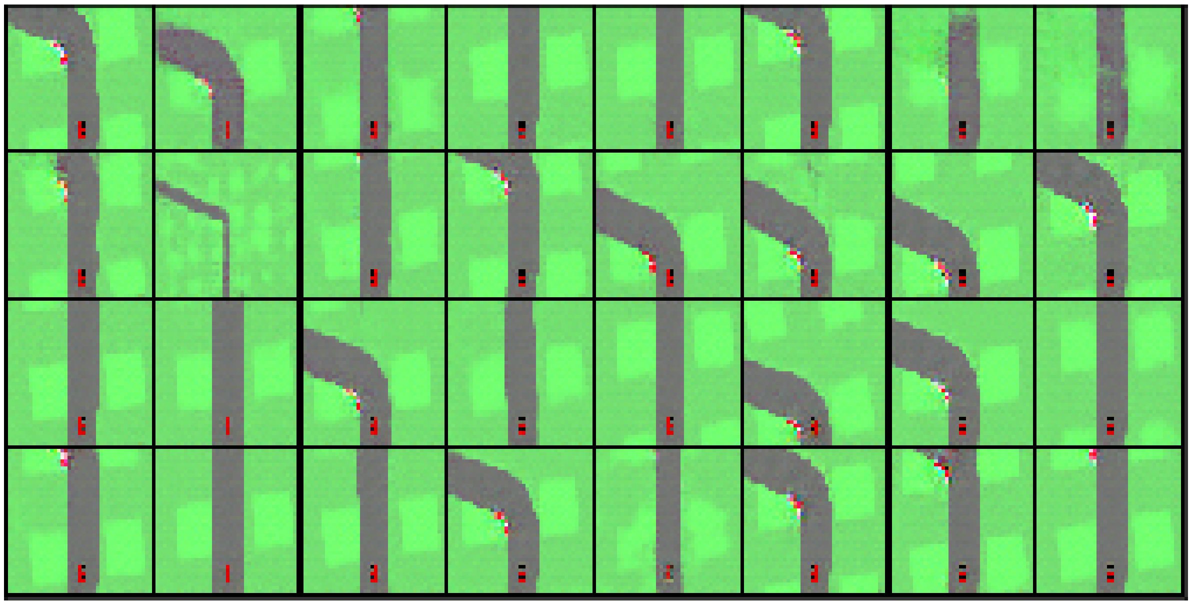

Figure 5.4: Examples of generated images with random latent vectors.

With these two benefits, one future direction of research is to perform data augmentation using VAE/GAN. For example, figure 5.4 is a sample of generated images. GAN itself can be used for this purpose, however, as stated before, it does not have encoder instead use the normal distribution to generate a latent code z. Further analyses about whether the assumption of the normal distribution prior limits the acquired representation capacity or not, and the benefits of having learned latent distribution is an important path to study.

We hope that this direction is extended with the forward prediction model. As a preliminary analysis, we implemented a forward prediction model using a recurrent neural network (RNN). This network takes a sequence of latent vector $z_t, z_{t+1}, ..., z_{t+n}$, and predict the next state's latent vector z_{t+n+1}. Figure 5.5 is an example of the original input frames (the first row), reconstructed images of predicted z (the second row), and the reconstruction of z, which is encoded by the encoder (the third row). The comparison of images in the second and third-row shows that the forward prediction model predicts the next observation's latent vector well. It enables us to augment the data with more control; for example, it allows us to generate trajectories we would like the agent to learn instead of independent frames.

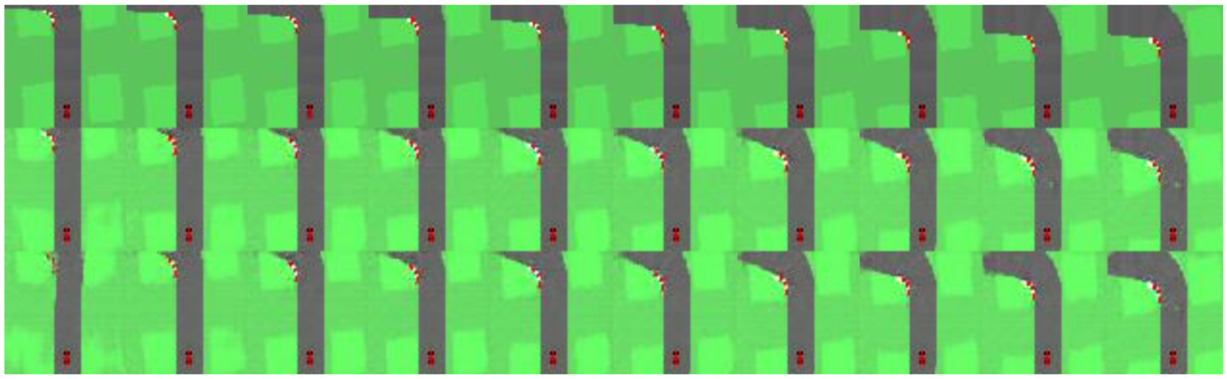

Figure 5.5: Forward prediction experiment. The first row is the frames of next observations (ground truth), the second row is the reconstructions from the predicted z' given the current observations, which we call $\hat{z}'$, and the third row is reconstructed images from latent code z' of next observations, encoded using the real observations. Based on this experiment, we can qualitatively conclude that the forward prediction model predicts z' well.

Chapter 6

Conclusion

Reinforcement learning provides us with autonomy as it does not require labeled data sets, which has a great potential to develop diverse real-world machine learning applications. However, it requires a large amount of interaction to collect data, and model-based RL is a solution to this sample inefficiency. Motivated to solve the sample inefficiency challenge, this dissertation presents the following two research directions from the model perspective in RL:

- How should RL agents better utilize models to train their policy under the condition that the models are threatened to be wrong because of insufficient training or data.
- What neural network architecture can serve as a representation model for the RL agents.

Chapter 3 proposed a novel training method of RL agent, which leverages the uncertainty estimation using MC-dropout in Dyna. We demonstrated that filtering less confident model simulations improved the agent's performance in the early phase of its training. This result suggests that considering model uncertainty makes an RL agent require less interaction with the environment, thus the agent is sample efficient. Chapter 4 proposed using VAE/GAN as the vision module for the agent. We analyzed VAE/GAN from RL task performance and disentanglement perspective. Our result shows that the RL agent that learns a policy over the VAE/GAN embedding outperforms the agent with the VAE embedding as its vision module. Although we expected that the network learns disentangled representation from our qualitative evaluation using t-SNE that shows clusters on a 2D surface, we concluded qualitatively, based on the classifier-based metric proposed in [Higgins et al., 2017a], that the network did not learn disentangled representation. Along with the experimental results, we highlighted the discussion that the existing disentanglement measures do not match each other. They may perform well in specific tasks and data sets but do not match the RL task performance. Further study about the disentanglement and RL task performance correlation is necessary. The bandit machines experiments in Chapter 5 discussed the benefit of dynamically adjusting learning rates. It speeds up the RL agent's policy update in the environments where the environment changes and the learned behavior needs to be updated. The data augmentation discussed in Chapter 5 aims to mitigate the high cost of collect data in RL. We hope the directions we propose help to develop sample efficient RL agent that requires fewer interactions with the environment and computational costs, which can be deployed in practical applications.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm. Unsupervised state representation learning in Atari. *CoRR*, abs/1906.08226, 2019. URL `http://arxiv.org/abs/1906.08226`.

Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy J. Colwell. Model-based reinforcement learning for biological sequence design. In *ICLR*, 2020.

Mathieu Aubry, Daniel Maturana, Alexei Efros, Bryan Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *CVPR*, 2014.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *CoRR*, abs/1301.4862, 2013. URL `http://arxiv.org/abs/1301.4862`.

Donald F. Beal and Martin C. Smith. Temporal coherence and prediction decay in TD learning. In Thomas Dean, editor, *IJCAI*, pages 564–569. Morgan Kaufmann, 1999. ISBN 1-55860-613-0. URL `http://dblp.uni-trier.de/db/conf/ijcai/ijcai99.html#BealS99`.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/afda332245e2af431fb7b672a68b659d-Paper.pdf`.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL `http://arxiv.org/abs/1207.4708`.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML '09*, 2009.

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`.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL `http://arxiv.org/abs/1912.06680`.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Aaron P. Blaisdell. Mental imagery in animals: Learning, memory, and decision-making in the face of missing information. *Learning & Behavior*, pages 1–24, 2019.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, 2015.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

Yuri Burda, Harrison Edwards, Deepak Pathak, Amos J. Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. *CoRR*, abs/1808.04355, 2018a. URL `http://arxiv.org/abs/1808.04355`.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. *CoRR*, abs/1810.12894, 2018b. URL `http://arxiv.org/abs/1810.12894`.

Christopher P. Burgess, Irina Higgins, Arka Pal, Loïc Matthey, Nicholas Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE. *ArXiv*, abs/1804.03599, 2018.

Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. Top-k off-policy correction for a reinforce recommender system. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.

Tian Qi Chen, Xuechen Li, Roger B. Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. *CoRR*, abs/1802.04942, 2018. URL `http://arxiv.org/abs/1802.04942`.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016. URL `http://arxiv.org/abs/1606.03657`.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo Tree Search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

Will Dabney and Andrew G. Barto. Adaptive step-size for online temporal difference learning. In *AAAI*, 2012.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019. URL `http://arxiv.org/abs/1901.10995`.

Cynthia D Fast and Aaron P. Blaisdell. Rats are sensitive to ambiguity. *Psychonomic Bulletin & Review*, 18:1230–1237, 2011.

Cynthia D Fast, Traci Biedermann, and Aaron P. Blaisdell. Imagine that! Cue-evoked representations guide rat behavior during ambiguous situations. *Journal of Experimental Psychology. Animal Learning and Cognition*, 42 2:200–11, 2016.

Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard H. Hovy. A survey of data augmentation approaches for NLP. *CoRR*, abs/2105.03075, 2021. URL `https://arxiv.org/abs/2105.03075`.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017. URL `http://arxiv.org/abs/1706.10295`.

Yarin Gal. *Uncertainty in Deep Learning.* PhD thesis, University of Cambridge, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/gal16.html`.

Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseo Lee, Matthias Humt, Jianxiang Feng, Anna M. Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, M. Shahzad, Wen Yang, Richard Bamler, and Xiaoxiang Zhu. A survey of uncertainty in deep neural networks. *ArXiv*, abs/2107.03342, 2021.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. *Corrosion*, page iii, 2014. ISSN 03787753. doi: 10.1016/B978-0-408-00109-0.50001-8. URL `https://papers.nips.cc/paper/5423-generative-adversarial-nets{%}0Ahttp://doi.wiley.com/10.1002/9781118472507.fmatter{%}0Ahttp://linkinghub.elsevier.com/retrieve/pii/B9780408001090500018`.

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

Ian J. Goodfellow. NIPS 2016 tutorial: Generative Adversarial Networks. *CoRR*, abs/1701.00160, 2017. URL `http://arxiv.org/abs/1701.00160`.

Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016. URL `http://arxiv.org/abs/1603.00748`.

David Ha and Jürgen Schmidhuber. World Models. *CoRR*, abs/1803.10122, 2018. URL `http://arxiv.org/abs/1803.10122`.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/hafner19a.html`.

Jessica B. Hamrick, Kevin A. Smith, Thomas L. Griffiths, and Edward Vul. Think again? The amount of mental simulation tracks uncertainty in the outcome. *Cognitive Science*, 2015.

Hado Hasselt. Double Q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL `https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf`.

Yohei Hayamizu, Saeid Amiri, Kishan Chandan, Keiki Takadama, and Shiqi Zhang. Guiding robot exploration in reinforcement learning via automated planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31(1):625–633, May 2021. URL `https://ojs.aaai.org/index.php/ICAPS/article/view/16011`.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. β-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017a.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1480–1490. PMLR, 06–11 Aug 2017b. URL `https://proceedings.mlr.press/v70/higgins17a.html`.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL `https://www.science.org/doi/abs/10.1126/science.1127647`.

Daniella Horan, Eitan Richardson, and Yair Weiss. When is unsupervised disentanglement possible? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 5150–5161. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/29586cb449c90e249f1f09a0a4ee245a-Paper.pdf`.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In *NIPS*, 2016.

Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *CoRR*, abs/1910.09457, 2019. URL `http://arxiv.org/abs/1910.09457`.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 July 2015. PMLR. URL `https://proceedings.mlr.press/v37/ioffe15.html`.

Matthew Johnson, Katja Hofmann, Tm Hutton, David Bignell, and Katja Hofmann. The malmo platform for artificial intelligence experimentation. In *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI - Association for the Advancement of Artificial Intelligence, July 2016. URL `https://www.microsoft.com/en-us/research/publication/malmo-platform-artificial-intelligence-experimentation/`.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019. URL `http://arxiv.org/abs/1903.00374`.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lyTjAqYX`.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019. doi: 10.1109/CVPR.2019.00453.

Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A Doom-based AI research platform for visual reinforcement learning. *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2649–2658. PMLR, 10–15 July 2018. URL `https://proceedings.mlr.press/v80/kim18b.html`.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL `http://arxiv.org/abs/1312.6114`.

Diederik P. Kingma, Danilo Jimenez Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014. URL `http://arxiv.org/abs/1406.5298`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Brenden M. Lake, Tomer David Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2016.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *33rd International Conference on Machine Learning, ICML 2016*, 4:2341–2349, 2016.

Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/059fdcd96baeb75112f09fa1dcc740cc-Paper.pdf`.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL `https://doi.org/10.1038/nature14539`.

Timothée Lesort, Natalia Díaz-Rodríguez, Jean Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018. ISSN 18792782. doi: 10.1016/j.neunet.2018.07.006.

Yingzhen Li and Stephan Mandt. A deep generative model for disentangled representations of sequential data. *CoRR*, abs/1803.02991, 2018. URL `http://arxiv.org/abs/1803.02991`.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Francesco Locatello, Stefan Bauer, Mario Lucic, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *CoRR*, abs/1811.12359, 2018. URL `http://arxiv.org/abs/1811.12359`.

Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, 5:3153–3160, 2020.

William Lotter, Gabriel Kreiman, and David D. Cox. Unsupervised learning of visual structure using predictive generative networks. *CoRR*, abs/1511.06380, 2015. URL `http://arxiv.org/abs/1511.06380`.

Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry P. Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *CoRR*, abs/1902.02476, 2019. URL `http://arxiv.org/abs/1902.02476`.

Ashique Rupam Mahmood, Richard S. Sutton, Thomas Degris, and Patrick M. Pilarski. Tuning-free step-size adaptation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2121–2124, 2012. doi: 10.1109/ICASSP.2012.6288330.

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dSprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

W. T. Miller, R. S. Sutton, and P. J. Werbos. *First Results with Dyna, an Integrated Architecture for Learning, Planning and Reacting*, pages 179–189. 1995.

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`.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/mniha16.html`.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015. URL `http://arxiv.org/abs/1507.04296`.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL `http://arxiv.org/abs/1808.00177`.

Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *CoRR*, abs/1703.01310, 2017. URL `http://arxiv.org/abs/1703.01310`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL `http://arxiv.org/abs/1912.01703`.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/pathak17a.html`.

Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3D face model for pose and illumination invariant face recognition. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 296–301, 2009. doi: 10.1109/AVSS.2009.58.

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2182–2192, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1203. URL `https://aclanthology.org/P18-1203`.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *CoRR*, abs/1706.01905, 2017. URL `http://arxiv.org/abs/1706.01905`.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1511.06434`.

Karl Ridgeway. A survey of inductive biases for factorial representation-learning. *ArXiv*, abs/1612.05299, 2016.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *CoRR*, abs/2109.11978, 2021. URL `https://arxiv.org/abs/2109.11978`.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf`.

Veit Sandfort, Ke Yan, Perry Pickhardt, and Ronald Summers. Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks. *Scientific Reports*, 9, 11 2019. doi: 10.1038/s41598-019-52737-x.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, L. Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588 7839:604–609, 2020.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 July 2015. PMLR. URL `https://proceedings.mlr.press/v37/schulman15.html`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

D. Silver, Julian Schrittwieser, K. Simonyan, Ioannis Antonoglou, Aja Huang, A. Guez, T. Hubert, L. Baker, Matthew Lai, A. Bolton, Yutian Chen, T. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, T. Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Shobhita Sundaram and Neha Hulkund. GAN-based data augmentation for chest X-ray classification, 2021. URL https://arxiv.org/abs/2107.02970.

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

Richard S. Sutton. Adapting bias by gradient descent: An incremental version of Delta-Bar-Delta. In *AAAI*, 1992.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, MA, USA, 2018. ISBN 0262039249.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000. URL https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3a20f62a0af1aa152670bab3c602feed-Paper.pdf.

Jun Tani. *Exploring Robotic Minds: Actions, Symbols, and Consciousness as Self-Organizing Dynamic Phenomena.* Oxford University Press, Inc., USA, 1st edition, 2016. ISBN 0190281065.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. DeepMind control suite. *CoRR*, abs/1801.00690, 2018. URL http://arxiv.org/abs/1801.00690.

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Edward C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948. ISSN 19391471. doi: 10.1037/h0061626.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL `http://jmlr.org/papers/v9/vandermaaten08a.html`.

Herke van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934, 2016. doi: 10.1109/IROS.2016.7759578.

Pascal Vincent, H. Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML '08*, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(110):3371–3408, 2010. URL `http://jmlr.org/papers/v11/vincent10a.html`.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 617–626, 2018. doi: 10.1109/ICDM.2018.00077.

Zhicheng Wang, Biwei Huang, Shikui Tu, Kun Zhang, and Lei Xu. DeepTrader: A deep reinforcement learning approach for risk-return balanced portfolio management with market conditions embedding. In *AAAI*, 2021.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL `http://arxiv.org/abs/1511.06581`.

Satosi Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of Research and Development*, 4(1):66–82, 1960. doi: 10.1147/rd.41.0066.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Julian Zaidi, Jonathan Boilard, Ghyslain Gagnon, and Marc-André Carbonneau. Measuring disentanglement: A review of metrics. *CoRR*, abs/2012.09276, 2020. URL https://arxiv.org/abs/2012.09276.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *CoRR*, abs/1804.06459, 2018. URL http://arxiv.org/abs/1804.06459.