Intrinsic Reward Driven Exploration for Deep Reinforcement Learning

Haitao Xu

a thesis submitted for the degree of

Doctor of Philosophy

at the University of Otago, Dunedin,

New Zealand.

31 August 2020

Abstract

Deep reinforcement learning has become one of the hottest research topics in machine learning. In reinforcement learning, agents interact with the environment and try to maximise the expected cumulative reward. The goal of reinforcement learning is to find a policy to maximise the agent's total cumulative rewards. Unfortunately, some environments can only provide extremely sparse rewards, so the agent needs to learn a strategy to explore in its environment more efficiently to find these rewards. However, it is known that exploration in complex environments is a key challenge of deep reinforcement learning, especially for tasks where rewards are very sparse.

In this thesis, intrinsic reward driven exploration strategies are investigated. The agent driven by this intrinsic reward can explore expeditiously, so as to find the sparse extrinsic rewards provided by the environment. Recently, surprise has been used as an intrinsic reward that encourages systematic and efficient exploration. We first define a novel intrinsic reward function called assorted surprise, and propose Variational Assorted Surprise Exploration (VASE) algorithm to approximate this assorted surprise in a tractable way, with the help of Bayesian neural networks. Then we apply VASE algorithm to continuous control problems and large scale Atari video games respectively. Experimental results show that VASE performs well across these tasks. Then we discover that all surprise based exploration methods will lose exploration efficiency in areas where the environmental transition is discontinuous. To solve this problem, we propose Mutual Information Minimising Exploration (MIME) algorithm. We show that MIME can explore as efficiently as surprise based methods in other areas of the environment but much better in areas with discontinuous transitions.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Prof. Brendan McCane and co-supervisor Dr. Lech Szymanski for forging my scientific personality. I am truly grateful to them for the freedom they gave to me and for their continuous support, trust throughout my PhD study.

I also wish to thank the Department of Computer Science, especially the technical and administrative staff. They have always been kind to help and deal with my requests. I am also grateful to Xiping Fu, David Wang, Russel Mesbah and Tapabrata Chakraborti, who gave me a lot of advice on how to start a PhD.

Finally, I'm forever indebted to my wife, my parents, my children, for their support and patience during the past three years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL), especially deep reinforcement learning (DRL) has achieved many amazing results and can defeat human beings in some fields like playing the game of Go (Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, *et al.*, 2016) and playing Atari video games (Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, Petersen, Beattie, Sadik, Antonoglou, King, Kumaran, Wierstra, Legg, and Hassabis, 2015). It has become one of the most popular research topics and attracted more and more interests from researchers recently.

As a type of machine learning (Bishop, 2006; Mohri, Rostamizadeh, and Talwalkar, 2018; Murphy, 2012) technique, reinforcement learning (RL) is a goal-oriented learning mechanism. The goal of RL is to seek a policy model so that the actions can be taken to maximise the agent's total cumulative reward. It tries to train the agent through many steps of decision-making to achieve this goal. For each step, the agent receives a reward from the environment (Sutton and Barto, 2018). The decision that is learnt by an RL agent in each step can be made by maximising the discounted sum of these rewards. However, it is a challenging problem to design this reward function because it depends on different environments and sometimes can be extremely sparse.

Deep reinforcement learning (DRL) is a powerful combination of deep learning (Bengio, Courville, and Vincent, 2013; LeCun, Bengio, and Hinton, 2015; Schmidhuber, 2015) and reinforcement learning. The "deep" here refers to multiple layers of artificial neural networks. The rise of deep learning accelerated progress in RL. With deep learning algorithms, DRL has been applied to different fields and industries such as resources management in computer clusters (Mao, Alizadeh, Menache, and Kandula, 2016), large scale traffic light control (Wei, Zheng, Yao, and Li, 2018;

Chu, Wang, Codecà, and Li, 2019), robotics (Kober, Bagnell, and Peters, 2013; Levine, Finn, Darrell, and Abbeel, 2016), chemistry (Zhou, Li, and Zare, 2017), personalised recommendations (Zheng, Zhang, Zheng, Xiang, Yuan, Xie, and Li, 2018), bidding and advertising (Jin, Song, Li, Gai, Wang, and Zhang, 2018) and games (Silver *et al.*, 2016; Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, Baker, Lai, Bolton, *et al.*, 2017; Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller, 2013).

1.1 Motivation

Recently, researchers have shown their interests in intrinsic motivation driven exploration in the field of DRL. The agent motivated by intrinsic reward can be trained to explore efficiently in its environment, even when the environment only provides sparse rewards. To explain how the intrinsic reward works in reinforcement learning exploration, the original reward signal r can be rewritten as

$$r = r^e + r^i, \tag{1.1}$$

where r^e represents the extrinsic reward from the environment, r^i represents the intrinsic reward. If the environment provides extremely sparse rewards, the agent can only obtain a zero reward ($r^e = 0$) at most time steps when it interacts with the environment. It can be seen that we can still use the intrinsic reward r^i to train the agent.

There are three main techniques for solving the problem of intrinsic reward-driven deep reinforcement learning exploration: The first evaluates state novelty (Kearns and Singh, 2002; Bellemare, Srinivasan, Ostrovski, Schaul, Saxton, and Munos, 2016), the second uses prediction error (Schmidhuber, 1991b,a; Achiam and Sastry, 2017; Pathak, Agrawal, Efros, and Darrell, 2017) and the third is based on information gain (Deci and Ryan, 2010; Houthooft, Chen, Duan, Schulman, De Turck, and Abbeel, 2016; Little and Sommer, 2013). Prediction error can be derived from surprisal (Traibus, 1961), and information gain method is highly related to Bayesian surprise (Itti and Baldi, 2006; Storck, Hochreiter, and Schmidhuber, 1995). It can be seen that the latter two types of methods are both related to surprise. Sometimes we collectively call them surprise-driven learning. This thesis mainly discusses the latter two methods.

There are many interesting but challenging problems that are worthy of study. First, different forms of surprise (surprisal for prediction error method, Bayesian surprise for

information gain method) have been chosen as the intrinsic reward to train the agent. However, each of them has its own shortcomings. For example, in surprisal method it is hard to process the stochasticity that happens in the environment. Bayesian surprise can handle the uncertainty but the proposed algorithms show that they can only apply to simple environments for the reason of expensive computation. Therefore, designing or selecting a suitable surprise function to represent intrinsic rewards is a worthwhile research direction. In addition, the designed intrinsic reward exploration algorithm should be easily applied to large scale problems like deep neural networks and complex environments. Furthermore, some environmental transitions are discontinuous or abrupt, it is difficult for the agent to predict future state and therefore the agent is continuously surprised by the transition. This results in an agent that gets stuck on the transition boundary. It is interesting to investigate how to generate an intrinsic reward when the agent's model simply tries to learn a representation of the world without prediction.

1.2 Thesis statement and contributions

The thesis is that it is possible to design intrinsic rewards to drive an agent to explore an environment where only extremely sparse rewards are provided.

In the case of a continuous environment, it is hypothesised that:

(I) The agent driven by assorted surprise will perform better than those driven by surprisal.

(II) Training speed of assorted surprise algorithm runs faster than Bayesian surprise based algorithm.

(III) In the environment where rewards can be obtained at each time step, adding intrinsic reward will not reduce the agent's exploration efficiency.

In the case of an environment with boundary transitions, it is hypothesised that:

(IV) Mutual information minimising exploration (MIME) will perform as well as surprise-based method in normal sparse reward environments.

(V) Mutual information minimising exploration (MIME) will perform better than surprise-based method in an environment where the transition is discontinuous.

The main contributions of this thesis are:

- I propose a new definition of surprise as the intrinsic reward. I also propose an algorithm VASE (Variational agent's surprise exploration) to implement this surprise to train the agent in reinforcement learning settings. VASE is a fast exploration strategy and solved by variational inference in Bayesian neural networks. Because the uncertainty in Bayesian neural network is reduced during training, VASE is as good as information gain methods in dealing with uncertain environments, such as continuous control environments. The agent driven by VASE can effectively explore continuous control environments and find sparse extrinsic rewards given by the environment.

- I applied the VASE algorithm to large scale problems. Let the agent trained by VASE play Atari video games. VASE works well on most Atari games with sparse reward. However, a new problem was found that when playing Montezuma Revenge game, whose transition boundary is not continuous, the agent was also stuck at the boundary between different rooms.

- I also propose an algorithm named MIME (Mutual Information Minimising Exploration). The agent trained by surprisal or VASE can be stuck on the environmental transition boundary. The MIME algorithm solves the problem of agents getting stuck at sharp boundary transitions. MIME agents learn as well as surprisal-agents in sparse reward environments and much better in environments that include sharp boundary transitions.

1.3 Limit of scope

In addition to exploration, there are other ways to use intrinsic rewards in reinforcement learning. Actually, Oudeyer, Kaplan, *et al.* (2008) have proposed a classification for all intrinsic reward methods (later Aubret, Matignon, and Hassas (2019) proposed a slightly different classification). The first category is the knowledge-based model (knowledge acquisition), the second category is the competency-based model (skill learning). Exploration belongs to knowledge acquisition. The method like empowerment (Salge, Glackin, and Polani, 2014) is also included in this category. Skill learning has methods like skill abstraction (Heess, Wayne, Tassa, Lillicrap, Riedmiller, and Silver, 2016) and curriculum learning (Co-Reyes, Liu, Gupta, Eysenbach, Abbeel, and Levine, 2018; Sharma, Gu, Levine, Kumar, and Hausman, 2019; Lee, Eysenbach, Parisotto, Xing, Levine, and Salakhutdinov, 2019). However, it is impossible to cover

all these problems during a single PhD. Limit of Scope in this thesis is that we only discuss "exploration methods" for intrinsic reward-driven reinforcement learning.

1.4 Thesis layout

This thesis consists of six chapters and one appendix. The rest of the thesis is structured as follows.

Chapter 2 presents related background and important concepts in deep reinforcement learning. These are necessary to understand the remaining chapters of this thesis.

The following three chapters are for different contributions to this thesis. Chapter 3 presents a new form of surprise as the agent's intrinsic motivation and applied it to the RL settings by using the VASE algorithm. VASE tries to approximate this surprise in a tractable way and train the agent to maximise its reward function. The agent is driven by this intrinsic reward, which can effectively explore the environment and find sparse extrinsic rewards given by the environment. Empirical results show in Chapter 3 that VASE performs well across various continuous control tasks with sparse rewards. Chapter 4 shows the results when we apply VASE to large scale environments: in this case Atari video games. Chapter 5 introduces a novel algorithm, MIME, whose purpose is to solve the problems we found in Chapter 4. The main difference between MIME agents and other surprise-driven RL agents is that MIME agents do not try to predict the future. Rather, they form a measure of how comfortable they are in a given environment. This is a simple idea, is easy to implement and most importantly it overcomes the limitations of surprisal getting stuck at transition boundaries.

Chapter 6 concludes the thesis and suggests possible future works.

Before I shifted my research direction to deep reinforcement learning, I initially started my PhD in the direction of support vector machines. I include this part of work as an appendix.

The source code for this thesis can be found online[1].

1.5 Publications

Some of the techniques mentioned in this thesis have previously been described in several publications, which are listed below

[1] `https://drive.google.com/open?id=1NRfVdrlqW7vDIgALHCSuIlnpQSOpkHlI`

- Xu, H., McCane, B. and Szymanski, L., 2019. VASE: Variational Assorted Surprise Exploration for Reinforcement Learning. arXiv preprint arXiv:1910.14351.
- Xu, H., McCane, B., Szymanski, L. and Atkinson, C., 2020. MIME: Mutual Information Minimisation Exploration. arXiv preprint arXiv:2001.05636.

I also did some work on support vector machines during my Ph.D. study, the results are published in

- Xu, H., McCane, B. and Szymanski, L., 2018, December. Twin Bounded Large Margin Distribution Machine. In Australasian Joint Conference on Artificial Intelligence (pp. 718-729). Springer, Cham.

Chapter 2

Background and related work

In this chapter, I will introduce the background and notations related to this thesis. I will first introduce some important concepts in information theory, these concepts can be used to define the intrinsic rewards discussed in this thesis. Since this thesis mainly discusses the importance of intrinsic reward exploration in deep reinforcement learning, I will then introduce reinforcement learning and deep reinforcement learning related algorithms, focusing on policy gradient algorithms like TRPO algorithm and PPO algorithm that will be used in the following chapters. Next, the variational inference method and Bayesian neural networks are introduced. They will first be used in Chapter 3 to derive the variational assorted surprise. At the end of this chapter, I will introduce and summarise some intrinsic reward exploration algorithms in the field of reinforcement learning in recent years, so that I can compare my methods to some of them in Chapter 3 to Chapter 5.

2.1 Information theory terminologies

In this section, I will introduce some terminologies in information theory field. All these terms will be used in the following chapters and are important to define the intrinsic reward functions.

2.1.1 Surprisal

Suppose probability distribution P is defined on probability space $\mathcal{X}$, $X \in \mathcal{X}$ is a random variable. In information theory (Shannon, 1948; Traibus, 1961), the surprisal

(also called information content or self-information)

$$-\log P_X(x) \tag{2.1}$$

quantifies the "surprise" degree of a particular event x that occurs from a random variable X. Through this definition, we can see that if an event x occurs with probability of 1, it will not bring any surprises or information. If an event that happened is unlikely to happen, it will generate a lot of information content and bring huge surprises.

2.1.2 Shannon entropy

The Shannon entropy $H(X)$ of the random variable X is defined as

$$\begin{aligned} H(X) &= \sum_{x\in\mathcal{X}} -P_X(x)\log P_X(x) \\ &= \sum_{x\in\mathcal{X}} P_X(x)(-\log P_X(x)) \\ &= \mathbb{E}_x[-\log P_X(x)]. \end{aligned} \tag{2.2}$$

It is the average level of "surprise" or "uncertainty" in the possible outcome x of the random variable X. Different to surprisal which quantifies the degree of surprise associated with a particular outcome x of a random variable X, entropy quantifies the average degree of the entire random variable over all its possible outcomes.

2.1.3 KL-divergence

The Kullback-Leibler divergence (also called relative entropy, KL-divergence) is a measure of how one distribution differs from the other. For probability distributions P and Q of a discrete random variable $X \in \mathcal{X}$, where $\mathcal{X}$ is the probability space that P and Q are defined on. The KL-divergence from Q to P is defined (Kullback and Leibler, 1951) as

$$D_{KL}[P\|Q] = \sum_{x\in\mathcal{X}} P_X(x)\log\frac{P_X(x)}{Q_X(x)}. \tag{2.3}$$

For continuous random variable X, the KL-divergence is defined as

$$D_{KL}[P\|Q] = \int_{\mathcal{X}} p(x)\log\frac{p(x)}{q(x)}dx, \tag{2.4}$$

where $p(x)$ and $q(x)$ are the probability density functions of P and Q respectively. Note that KL-divergence is asymmetric and does not satisfy the triangle inequality.

2.1.4 Mutual information

Let (X, Y) be a random vector with value over the space $\mathcal{X} \times \mathcal{Y}$. The mutual information (MI, also called information gain) is a measure of the interdependence between the two random variables X and Y. In other words, mutual information determines how different the joint distribution of the random vector (X, Y) is to the product of the marginal distributions of random variable X and random variable Y:

$$I(X;Y) = D_{KL}[P_{(X,Y)} \| P_X \otimes P_Y]. \tag{2.5}$$

In the discrete case, $P_{(X,Y)}$ denotes the joint probability mass function of random variables X and Y, P_X and P_Y are the marginal probability mass functions of X and Y respectively, the mutual information can be rewritten as

$$I(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} P_{(X,Y)}(x,y) \log \frac{P_{(X,Y)}(x,y)}{P_X(x) P_Y(y)}, \tag{2.6}$$

and for continuous case, $P_{(X,Y)}$ is now the joint probability density function of random variable X and random variable Y, P_X and P_Y are the marginal probability density functions of X and Y respectively, the mutual information is

$$I(X;Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} P_{(X,Y)}(x,y) \log \frac{P_{(X,Y)}(x,y)}{P_X(x) P_Y(y)} dx dy. \tag{2.7}$$

Mutual information satisfies the following equations:

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X), \end{aligned} \tag{2.8}$$

which is important to derive Bayesian surprise that we will discuss in following chapters. We show the proof of $I(X;Y) = H(Y) - H(Y|X)$ here, in discrete case:

$$\begin{aligned} I(X;Y) &= \sum_y \sum_x P_{(X,Y)}(x,y) \log \frac{P_{(X,Y)}(x,y)}{P_X(x) P_Y(y)} \\ &= \sum_y \sum_x P_{(X,Y)}(x,y) \log \frac{P_{(X,Y)}(x,y)}{P_X(x)} - \sum_y \sum_x P_{(X,Y)}(x,y) \log P_Y(y) \\ &= \sum_y \sum_x P_X(x) P_{Y|X=x}(y) \log P_{Y|X=x}(y) - \sum_y \left(\sum_x P_{(X,Y)}(x,y) \right) \log P_Y(y) \\ &= -\sum_x P_X(x) H(Y|X=x) - \sum_y P_Y(y) \log P_Y(y) \\ &= -H(Y|X) + H(Y) \\ &= H(Y) - H(Y|X). \end{aligned} \tag{2.9}$$

There is another concept called pointwise mutual information (PMI):

$$\text{pmi}(x;y) = \log \frac{P_{(X,Y)}(x,y)}{P_X(x)P_Y(y)}, \tag{2.10}$$

which is a measure of association of single events. Mathematically, mutual information is the expected value of pointwise mutual information.

In general, mutual information is symmetric and non-negative. Compared to linear dependence like the correlation coefficient, mutual information is more general.

2.2 Deep reinforcement learning

Reinforcement learning (RL) is a type of machine learning. Compared with the classic supervised and unsupervised learning problems of machine learning, the biggest feature of reinforcement learning is learning in interaction: Reinforcement learning is learning what to do—how to map situations to actions—to maximize reward signals (Sutton and Barto, 2018). The protagonist of RL is the *agent*. The *environment* is the world that the agent interacts with. The environment is the world that the agent interacts with. Currently, the development of deep learning allows reinforcement learning to solve previously difficult problems, such as learning to play video games directly from pixels. Deep reinforcement learning (DRL) algorithms are also applied to robotics, so that robot control policies can be learned directly from real-world camera inputs (Arulkumaran, Deisenroth, Brundage, and Bharath, 2017).

There are two outstanding success stories in the latest work in the DRL field. The first was the development of an algorithm that can learn directly from image pixels to play Atari 2600 (Bellemare, Naddaf, Veness, and Bowling, 2013) video games at superhuman level (Mnih *et al.*, 2015), which drove the revolution of DRL. This is the first convincing evidence that RL agents can be trained on raw high-dimensional observations. The other outstanding success was AlphaGo, which defeated the human world champion in 2016 (Silver *et al.*, 2016). The DRL algorithm has also been applied to robotics, and it is now possible to learn the robot's control policy directly from real-world camera inputs. (Levine, Pastor, Krizhevsky, Ibarz, and Quillen, 2018; Levine *et al.*, 2016). In addition, DRL has also used meta-learning algorithms ("learn to learn"), enabling them to generalize to different environments that they have never seen before (Duan, Schulman, Chen, Bartlett, Sutskever, and Abbeel, Duan *et al.*; Wang, Kurth-Nelson, Tirumala, Soyer, Leibo, Munos, Blundell, Kumaran, and Botvinick, 2016). Currently, DRL has also been used to handle almost all forms of machine

learning problems like designing the latest machine translation models (Zoph and Le, 2016; Zhang, Yang, and Başar, 2019) or building new optimization functions (Li and Malik, 2017).

In RL setup, a finite horizon discounted Markov decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, T, \rho_0)$, where: $\mathcal{S}$ is a *state* set contains the state of the world that the agent can perceive. $\mathcal{A}$ is an *action* set representing all the actions of the agent. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a *transition probability distribution* and $s_{t+1} \sim P(\cdot|s_t, a_t)$. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward* function defines the learning goal of the agent, denoted as $r_t = r(s_t, a_t)$. Although sometimes for convenience, it is also denoted as $r_t = r(s_t)$ or $r_t = r(s_t, a_t, s_{t+1})$. $t \in \{1, 2, \cdots, T\}$ here represents the current step. Each time the agent interacts with the environment, the environment returns reward, telling the agent that the action just performed is good or bad. $\gamma \in (0, 1]$ is a *discount factor*, T the *horizon* and ρ_0 an initial state distribution. A *policy* refers to agent's choice of action to be taken in state s. It is the core concept of RL problems and can be deterministic or stochastic. In this thesis, we only discuss stochastic policy. A stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ gives the probability with $\pi(a|s)$ of taking action a in state s. In deep reinforcement learning frameworks, the policy is often represented by a deep neural network. The input of this neural network is states and the output is actions. Let $\tau = (s_0, a_0, \cdots)$ denote the whole *trajectory*, which is a sequence of states and actions in the world. $s_0 \sim \rho_0$ the very first state of the world, $a_t \sim \pi(a_t|s_t)$. The goal of an agent is to find a policy π so as to maximises the expected *discounted total return*:

$$\mathbb{E}_{\tau \sim \pi}[R(\tau)], \tag{2.11}$$

where $\tau \sim \pi$ is shorthand for "for each state s_i in τ, the associated action $a_i \sim \pi(\cdot|s_i)$", $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ — a discounted sum of all rewards in a fixed horizon T. In this regard, RL aims to solve an optimisation problem. But in RL, agents need to use trial and error to learn the consequences of actions in the environment. Each interaction with the environment generates information that the agent uses to update its knowledge.

The *model* is a simulation of the environment. The model simulates the response of the environment after the agent samples the action. In RL, the method of using model and planning is called *model-based*, while the method of learning policy by trial-and-error instead of using model is called *model-free*. And in DRL, similarly to policy, the model is represented by a deep neural network too. The following chapters in the thesis only discuss model-based methods. The agent trained by the algorithms that we proposed always has a model to help it to learn the environment.

There are two main types of techniques for solving the RL problem: methods based on *value function* and methods based on *policy search*, which we call

- Value-Based methods,
- Policy-Based methods.

2.3 Value-Based methods

The value-based method estimates the value (expected return) at a given state. The value function, different from reward function that defines the quality of immediate return in an interaction, defines the average return of the action in the long run. Value function $V_\pi(s)$ is a term used to represent the long-term expected return of the strategy π under state s:

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s], \tag{2.12}$$

and action-value function $Q_\pi(s,a)$ is the long-term expected return of the strategy π taking action a under state s:

$$Q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]. \tag{2.13}$$

Both $V_\pi(s)$ and $Q_\pi(s,a)$ value functions obey special self-consistency equations called Bellman equations:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_{a\sim\pi(\cdot|s),s'\sim\mathcal{P}(\cdot|s,a)}[r(s,a) + \gamma V_\pi(s')] \\ Q_\pi(s,a) &= \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[r(s,a) + \gamma\mathbb{E}_{a'\sim\pi(\cdot|s)}[Q_\pi(s',a')]]. \end{aligned} \tag{2.14}$$

There are two other value functions. One is called the optimal value function, $V_*(s)$, which gives the expected return when you start in the state s and always act under the optimal policy:

$$V_*(s) = \max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s], \tag{2.15}$$

the other is called the optimal action-value function, $Q_*(s,a)$, which gives the expected return for taking action a in state s and thereafter following an optimal policy:

$$Q_*(s,a) = \max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]. \tag{2.16}$$

Both the optimal value function and optimal action-value function also obey Bellman equations:

$$\begin{aligned} V_*(s) &= \max_a \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}[r(s,a) + \gamma V_*(s')] \\ Q_*(s,a) &= \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}[r(s,a) + \gamma \max_{a'}[Q_*(s',a')]]. \end{aligned} \tag{2.17}$$

And

$$V_*(s) = \max_a Q_*(s,a). \tag{2.18}$$

The Bellman equation is the basis of the Q-learning (Watkins and Dayan, 1992; Arulkumaran *et al.*, 2017) algorithm, and the core idea of Q-learning is to update the Q-value iteratively via satisfying optimal Bellman equations.

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\delta, \tag{2.19}$$

where α is the learning rate and

$$\delta = r_t + \gamma \max_a Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t) \tag{2.20}$$

is called temporal difference (TD) error. Q-learning approximates Q_* directly, which can be achieved by minimizing TD errors from trajectories (See Algorithm 1).

Algorithm 1: Q-learning algorithm

Initialise $Q_0(s,a)$ for all s, a
Start from initial s
for $k = 0, 1, \cdots$ **do**
  Sample action a, get next state s'
  $Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\delta$
  Update $s \leftarrow s'$
  if s' *is terminal* **then**
    Sample new initial state s'
  end
end

2.3.1 Deep Q-Network (DQN)

Since deep neural networks (DNN) are universal function approximators, it is natural to use deep learning to approximate functions for deep reinforcement learning agents.

In the beginning, the value-based method in DRL only took simple states as its inputs. However, the current methods can solve complex environments (Mnih *et al.*, 2015; Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, and Kavukcuoglu, 2016; Oh, Chockalingam, Lee, *et al.*, 2016; Schulman, Levine, Abbeel, Jordan, and Moritz, 2015; Zhu, Mottaghi, Kolve, Lim, Gupta, Fei-Fei, and Farhadi, 2017).

DQN (Mnih *et al.*, 2015; Yang, Xie, and Wang, 2019) has achieved scores comparable to professional video game testers in many classic Atari 2600 video games. When designing, the final fully connected layer of DQN will output $Q_\pi(s;\cdot)$ for all possible action values in a group of discrete actions. The advantage of DQN is that it can use DNN to compactly represent high-dimensional observations and Q functions.

One of the most important parts of DQN is the function that approximates the value of Q. We try to find or approximate the value function first and then extract the policy. The representative methods based on DQN are: DQN (Mnih *et al.*, 2013), double DQN (van Hasselt, Guez, and Silver, 2016; Brim, 2020), dueling DQN (Wang, Schaul, Hessel, Van Hasselt, Lanctot, and De Freitas, 2016) and Rainbow (Hessel, Modayil, van Hasselt, Schaul, Ostrovski, Dabney, Horgan, Piot, Azar, and Silver, 2018).

Q-learning is a very classic algorithm of RL. Its DQN extensions also have achieved great success in the Atari game problem. However, the scope of application of DQN is still in low-dimensional, discrete action space. This is because DQN finds the maximum Q-value of each action, which is not applicable in continuous space. If the continuous action space is discretized, the action space will scale exponentially in its dimensionality. For example, if the dimension of continuous action space is 8, each action is divided into 10 discrete actions, and the dimension of action space will be expanded to 10^8 = 100000000. Even if some DQN variants can give a solution of continuous action, the second problem of DQN is that it can only give a deterministic action, not the probability value.

From another perspective, the value-based method like DQN is still indirectly seeking policies. A natural question to ask is why we don't directly seek policy? This is what policy-based methods do.

2.4 Policy-based methods

Policy-based methods have some advantages compared to value-based methods. For example, policy-based methods are much more effective in high-dimensional or continuous spaces. Policy-based methods also seek stochastic policies.

Policy-based reinforcement learning is an optimization problem. Given a policy π with the parameter ψ, we try to find best ψ that generates the best policy. This can be done by maximising the objective function $J(\pi_\psi)$

$$\max_{\psi} J(\pi_\psi) \quad = \quad \max_{\psi} \mathbb{E}_{\tau\sim\pi_\psi}[R(\tau)], \tag{2.21}$$

To solve optimisation problem Eq. 2.21, we can use two families of algorithms, one is called gradient-free algorithms, the other is gradient-based algorithms (Deisenroth, Neumann, Peters, *et al.*, 2013). The examples of gradient-free algorithms include hill climbing, simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983) and evolutionary algorithms (Moriarty, Schultz, and Grefenstette, 1999) etc. The gradient-free policy search method can optimize non-differentiable policies. However, in reinforcement learning, the most popular policy-based algorithm is gradient-based, which is called the policy gradient (PG) algorithm.

2.4.1 Policy gradient (PG) algorithm

Given policy function

$$\pi(a|s,\psi) = P(a_t = a|s_t = s,\psi), \tag{2.22}$$

The probability of generating a trajectory $\tau = (s_0, a_0, s_1, \cdots, s_{T+1})$ is

$$P(s_0, a_0, s_1, \cdots, s_{T+1}) = \rho_0(s_0)\prod_{t=0}^{T}\pi_\psi(a_t|s_t)P(s_{t+1}|s_t,a_t). \tag{2.23}$$

More generally, the probability of generating the trajectory τ under the policy π is expressed as

$$\pi_\psi(\tau) = \rho_0(s_0)\prod_{t=0}^{T}\pi_\psi(a_t|s_t)P(s_{t+1}|s_t,a_t). \tag{2.24}$$

Finding the gradient of Eq. 2.21 gives the policy gradient $\nabla_\psi J(\pi_\psi)$

$$\begin{aligned}\nabla_\psi J(\pi_\psi) &= \int \nabla_\psi \pi_\psi(\tau)R(\tau)d\tau \\ &= \int \pi_\psi(\tau)\nabla_\psi \log \pi_\psi(\tau)R(\tau)d\tau \\ &= \mathbb{E}_{\tau\sim\pi_\psi}[\nabla_\psi \log \pi_\psi(\tau)R(\tau)].\end{aligned} \tag{2.25}$$

Take the logarithm of both sides of Eq. 2.24,

$$\log \pi_\psi(\tau) = \log \rho_0(s_0) + \sum_{t=0}^{T}\log \pi_\psi(a_t|s_t) + \log P(s_{t+1}|s_t,a_t). \tag{2.26}$$

Then substitute into the gradient expression Eq. 2.25, the policy gradient becomes:

$$\nabla_\psi J(\pi_\psi) \quad = \quad \mathbb{E}_{\tau\sim\pi_\psi}[(\sum_{t=0}^{T} \nabla_\psi \log \pi_\psi(a_t|s_t))(\sum_{t=0}^{T} \gamma^t r(s_t, a_t))]. \tag{2.27}$$

This is an expectation. We can estimate it using the sample mean:

$$\nabla_\psi J(\pi_\psi) \quad \approx \quad \frac{1}{N}\sum_{i=1}^{N}[(\sum_{t=0}^{T} \nabla_\psi \log \pi_\psi(a_{i,t}|s_{i,t}))(\sum_{t=0}^{T} \gamma^t r(s_{i,t}, a_{i,t}))], \tag{2.28}$$

where N is the number of trajectories. According to the policy π_ψ, we can sample trajectories, and then calculate Eq. 2.28, and update the parameter ψ in one step according to the gradient ascent:

$$\psi \leftarrow \psi + \alpha \nabla_\psi J(\pi_\psi). \tag{2.29}$$

In summary, if the trajectory produces a high positive reward, we hope to increase the possibility of policy adoption. Conversely, if policies lead to higher negative rewards, we want to reduce their likelihood. The simplest policy gradient algorithm is called REINFORCE (Williams, 1992) or policy gradient with Monte-Carlo roll outs and can be summarised in Algorithm 2. We can see that REINFORCE algorithm relies on

Algorithm 2: REINFORCE (Policy gradient with Monte-Carlo rollouts)

Initialise policy neural network π_ψ
Reset the environment getting (s_0, r_0)
for $k = 0, 1, \cdots$ **do**
  Sample each trajectory τ from π_{ψ_k}
  Compute $\nabla_{\psi_k} J(\pi_{\psi_k})$ by Eq. 2.28
  Update π_{ψ_k} by $\psi_{k+1} \leftarrow \psi_k + \alpha \nabla_{\psi_k} J(\pi_{\psi_k})$
end

an estimated return by Monte-Carlo methods using trajectory samples to update the policy parameter ψ. i.e. play out the whole trajectory to compute the total rewards. The basic principle is to use gradient ascent to follow the policy with the largest increase in rewards. However, first-order optimizers are not very accurate for curved regions. The agent may become overconfident, make wrong moves and disrupt training progress. Trust region policy optimization (TRPO) (Schulman *et al.*, 2015) is trying to address this issue. TRPO updates the policy by taking the largest steps to improve the policy, while meeting special constraints to allow the old policy to be close to the new one.

Trust region policy optimization (TRPO)

The weakness of the policy gradient (PG) algorithm is the step size α. If the learning rate is not sensitive to the training process, the PG will severely encounter convergence problems. When the step size is inappropriate, the policy corresponding to the updated parameter is a worse policy. The updated parameters will be worse, so it is easy to cause the learning performance to deteriorate and eventually collapse. Therefore, proper step size is critical for reinforcement learning. What is the appropriate step size? The so-called appropriate step size means that when the policy is updated, the value of the return function cannot be worse. This is the problem that TRPO needs to solve. The idea is to decompose the return function that is relevant to new policy into a return function relevant to old policy and other terms. If the other terms are not less than zero, the new policy can guarantee that the return function is monotonic increasing. Let $J(\pi_\psi)$ be decomposed as

$$J(\pi_\psi) = J(\pi_{\psi_{old}}) + \mathbb{E}_{\tau\sim\pi_\psi}[\sum_{t=0}^{T}\gamma^t A_{\pi_{\psi_{old}}}(s_t, a_t)]. \tag{2.30}$$

Here we use $\pi_{\psi_{old}}$ to denote the old policy, π_ψ to denote the new policy, and $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ is called the *advantage function.*

We know that value function $V_\pi(s)$ is the average value of all action-value functions with respect to the action probability in this state s. The action-value function $Q_\pi(s,a)$ is a value function under a single action a. So the advantage function $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ denotes the advantage of the action-value function over the value function in the current state. If $A_\pi(s,a)$ is greater than zero, it means that the action is better than the average of actions. Otherwise, the current action a is not good.

Eq. 2.30 can be rewritten as

$$J(\pi_\psi) = J(\pi_{\psi_{old}}) + \sum_{t=0}^{T}\sum_{s} P(s_t = s|\pi_\psi)\sum_{a}\pi_\psi(a|s)\gamma^t A_{\pi_{\psi_{old}}}(s,a), \tag{2.31}$$

where $P(s_t = s|\pi_\psi)\pi_\psi(a|s)$ is the joint probability of (s,a), $\sum_a \pi_\psi(a|s)$ is the marginal distribution of the action a, that is, summing the entire action space at state s. $\sum_s P(s_t = s|\pi_\psi)$ finds the marginal distribution of the state s: the sum of the entire state space. If we define

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots, \tag{2.32}$$

then

$$J(\pi_\psi) = J(\pi_{\psi_{old}}) + \sum_{s}\rho_{\pi_\psi}(s)\sum_{a}\pi_\psi(a|s)A_{\pi_{\psi_{old}}}(s,a). \tag{2.33}$$

Note that the distribution of the state s at this time is generated by the new policy, which means it depends heavily on the new policy. For an improved new policy π_ψ, if $\sum_a \pi_\psi(a|s)A_{\psi_{old}}(s,a) \geq 0$, then π_ψ is better than old policy. However, if $\sum_a \pi_\psi(a|s)A_{\psi_{old}}(s,a) < 0$, it is hard to optimise $J(\pi_\psi)$.

TRPO first ignores the changes in the state distribution and still uses the state distribution corresponding to the old policy. This technique is the first approximation of the original cost function. In fact, when the old and new parameters are very close, it is reasonable to replace the new state distribution with the old state distribution. The original cost function becomes:

$$L_{\pi_{\psi_{old}}}(\pi_\psi) = J(\pi_{\psi_{old}}) + \sum_s \rho_{\pi_{\psi_{old}}}(s) \sum_a \pi_\psi(a|s) A_{\pi_{\psi_{old}}}(s,a). \tag{2.34}$$

Consider that $L_{\pi_{\psi_{old}}}(\pi_\psi)$ and $J(\pi_\psi)$ are functions of policy π_ψ, they are matched to first order (Kakade and Langford, 2002), that is,

$$\begin{aligned} L_{\pi_{\psi_{old}}}(\pi_{\psi_{old}}) &= J(\pi_{\psi_{old}}) \\ \nabla_\psi L_{\pi_{\psi_{old}}}(\pi_\psi)|_{\psi=\psi_{old}} &= \nabla_\psi J(\pi_\psi)|_{\psi=\psi_{old}}. \end{aligned} \tag{2.35}$$

So near ψ_{old}, the policy that can improve L can also improve the original return function J. Kakade and Langford (2002) proposed that the following bound holds:

$$J(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - C D_{KL}^{\max}(\pi_{old} \| \pi_{new}), \tag{2.36}$$

where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ and $D_{KL}^{\max}(\pi_{old}\|\pi_{new}) = \max_s D_{KL}(\pi_{old}(\cdot|s)\|\pi_{new}(\cdot|s))$. Here

$$\epsilon = \max_{s,a} A_\pi(s,a) \tag{2.37}$$

Eq. 2.36 gives the lower bound of $J(\pi)$, and if we denote $M_k(\pi) = L_{\pi_k}(\pi) - CD_{KL}^{\max}(\pi_k, \pi)$, then

$$\begin{aligned} J(\pi_{k+1}) &\geq M_k(\pi_{k+1}) \\ J(\pi_k) &= M_k(\pi_k) \\ J(\pi_{k+1}) - J(\pi_k) &\geq M_k(\pi_{k+1}) - M_k(\pi_k), \end{aligned} \tag{2.38}$$

which shows that if the new policy π_{k+1} maximises M_k, then $M_k(\pi_{k+1}) - M_k(\pi_k) \geq 0$, and $J(\pi_{k+1}) - J(\pi_k) \geq 0$. This new policy is the one we have been searching for. So the question can be formalized as

$$\max_\psi M_{\psi_{old}}(\pi) = L_{\pi_{\psi_{old}}}(\pi_\psi) - C D_{KL}^{\max}(\pi_{\psi_{old}, \pi_\psi}). \tag{2.39}$$

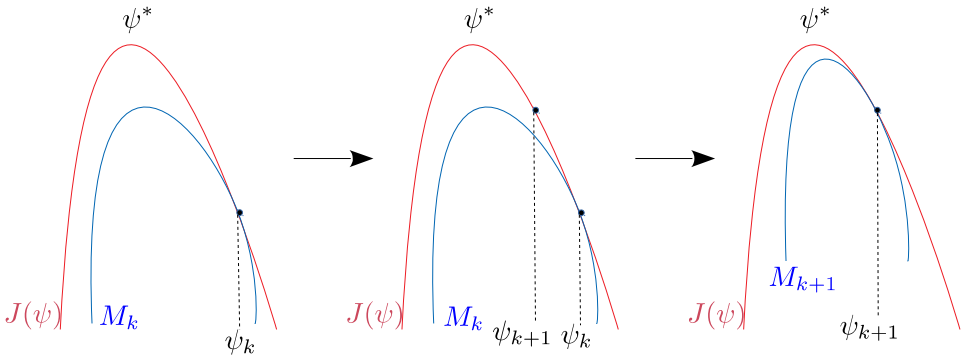


Figure 2.1: Monotonic improvement iteration

In each iteration, we search the best point of M and set it as the current policy. Then, we re-compute the lower bound M of the new policy and repeat this iteration. The policy will keep improving until it converges to a global or local optimal solution (See Figure 2.1).

In practice, if using the coefficient C, the step sizes would be very small. One way to take bigger steps in a robust way is to use KL-divergence between the new policy and the old policy, which is the trust region constraint:

$$\begin{aligned} \max_{\psi} \quad & L_{\pi_{\psi_{old}}}(\pi_{\psi}) \\ s.t. \quad & D_{KL}^{\max}(\pi_{\psi_{old}}, \pi_{\psi}) \leq \delta. \end{aligned} \tag{2.40}$$

From Eq. 2.34 and in practice the average KL-divergence

$$\bar{D}_{KL}^{\rho}(\pi_{\psi_1}, \pi_{\psi_2}) := \mathbb{E}_{s\sim\rho}[D_{KL}(\pi_{\psi_1}, \pi_{\psi_2})] \tag{2.41}$$

is used, then the Eq. 2.40 turns to

$$\begin{aligned} \max_{\psi} \quad & \sum_{s} \rho_{\pi_{\psi_{old}}}(s) \sum_{a} \pi_{\psi}(a|s) A_{\pi_{\psi_{old}}}(s,a) \\ s.t. \quad & \bar{D}_{KL}^{\rho_{old}}(\pi_{\psi_{old}}, \pi_{\psi}) \leq \delta, \end{aligned} \tag{2.42}$$

We can see that the action a is still generated by new policy π_{ψ}. But the parameter ψ of new policy π is unknown and so cannot be used to generate actions.

Now we need the second trick of TRPO, which processes action distributions using importance sampling:

$$\sum_{a} \pi_{\psi}(a|s) A_{\psi_{old}}(s,a) = \mathbb{E}_{a\sim\pi_{\psi_{old}}} \left[\frac{\pi_{\psi}(a|s)}{\pi_{\psi_{old}}(a|s)} A_{\pi_{\psi_{old}}}(s,a) \right]. \tag{2.43}$$

Based on the definition of $\rho(s)$, TRPO also uses $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\pi_{\psi_{old}}}}[\cdots]$ instead of $\sum_s \rho_{\pi_{\psi_{old}}}(s)[\cdots]$. Then Eq. 2.42 becomes the final theoretical optimization problem of TRPO:

$$\begin{aligned} \max_\psi \quad & \hat{L}_{\pi_{\psi_{old}}}(\pi_\psi) = \mathbb{E}_{s\sim\rho_{\psi_{old}}, a\sim\pi_{\psi_{old}}}\left[\frac{\pi_\psi(a|s)}{\pi_{\psi_{old}}(a|s)} A_{\pi_{\psi_{old}}}(s,a)\right] \\ s.t. \quad & \bar{D}_{KL}^{\rho_{old}}(\pi_{\psi_{old}}, \pi_\psi) \leq \delta, \end{aligned} \tag{2.44}$$

To solve 2.44 in practice, Taylor's expansion is used in both objective and constraint terms above:

$$\begin{aligned} \hat{L}_{\psi_k}(\psi) &\approx g^T(\psi - \psi_k) & g &\doteq \nabla_\psi \hat{L}_{\psi_k}(\psi)|_{\psi_k} \\ \bar{D}_{KL}(\psi, \psi_k) &\approx \tfrac{1}{2}(\psi - \psi_k)^T H(\psi - \psi_k) & H &\doteq \nabla_\psi^2 \bar{D}_{KL}(\psi, \psi_k)|_{\psi_k}, \end{aligned} \tag{2.45}$$

where g is the policy gradient, H is called Fisher Information Matrix (FIM) that can measure the sensitivity of the policy to the parameter ψ. Now the optimal problem of Eq. 2.44 becomes:

$$\begin{aligned} \psi_{k+1} = \quad & \text{argmax}_\psi \quad g^T(\psi - \psi_k) \\ & s.t. \qquad \frac{1}{2}(\psi - \psi_k)^T H(\psi - \psi_k) \leq \delta. \end{aligned} \tag{2.46}$$

This can be solved analytically by the Lagrangian dual method:

$$\psi_{k+1} = \psi_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g. \tag{2.47}$$

One problem is that this may not satisfy the KL constraint due to the approximate error introduced by the Taylor expansion. TRPO has modified this update rule by a backtracking line search (Algorithm 3)

$$\psi_{k+1} = \psi_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g, \tag{2.48}$$

where $\alpha \in (0,1)$ is the backtracking coefficient and j is the smallest non-negative integer, so that $\pi_{\psi_{k+1}}$ satisfies the KL constraint.

Finally, we put everything together for TRPO (Algorithm 4).

TRPO has been shown to be relatively powerful and suitable for domains with high-dimensional input. The combination of TRPO and generalised advantage estimation (GAE) (Schulman, Moritz, Levine, Jordan, and Abbeel, 2016) is still one of the most advanced RL technologies in continuous control. However, TRPO algorithms have some disadvantages. We can see that for each model parameter update, the computation of

Algorithm 3: Line search for TRPO

Compute policy step $\triangle_k = \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}^k}} \hat{H}_k^{-1} \hat{g}_k$

for $j = 0, 1, \cdots, L$ **do**

  Compute update $\psi = \psi_k + \alpha^j \triangle_k$

  if $\hat{L}_{\psi_k}(\psi) \geq 0$ *and* $\bar{D}_{KL}(\psi, \psi_k) \leq \delta$ **then**

    accept the update and set $\psi_{k+1} = \psi_k + \alpha^j \triangle_k$

    break

  end

end

Algorithm 4: TRPO: Trust region policy optimization

Initial policy parameter ψ_0

for $k = 0, 1, \cdots$ **do**

  Collect trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\psi_k)$

  Estimate advantage function $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

  Estimate policy gradient $\hat{g}_k$

  Use conjugate gradient iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

  Estimate proposed step $\triangle_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

  Perform line search to obtain final update $\psi_{k+1} = \psi_k + \alpha^j \triangle_k$

end

H is needed. Computing H is very time-consuming and it also requires a large number of samples to approximate H. Generally, compared to other policy gradient methods trained using a first-order optimizer, the sample efficiency of TRPO is lower. Due to this scalability issue, TRPO is impractical in large deep networks. Proximal policy optimization (PPO) was introduced to address these issues.

Proximal policy optimization (PPO)

Like TRPO, PPO also tries to take the update step as large as possible while the update does not lead to performance collapse. But different to TRPO that trains its optimisation problem by a second-order method, PPO chooses the first-order method and other tricks so that the new update policy is close to the old one. There are two variants of PPO: PPO-Clip and PPO-Penalty. In this thesis, we focus on PPO-Clip, which is very simple to implement in practice.

PPO updates policies by solving the following optimisation problem:

$$\psi_{k+1} = \text{argmax}_{\psi} L^{CLIP}_{\psi_k}(\psi), \tag{2.49}$$

where

$$L^{CLIP}_{\psi_k}(\psi) = \mathbb{E}_{\tau\sim\psi_k}[\sum_{t=0}^{T}(\min(\frac{\pi_\psi(a_t|s_t)}{\pi_{\psi_k}(a_t|s_t)}\hat{A}^{\pi_k}_t, \text{clip}(\frac{\pi_\psi(a_t|s_t)}{\pi_{\psi_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_k}_t)]. \tag{2.50}$$

ϵ is a small parameter, and the advantage function will be clipped if the ratio $\frac{\pi_\psi(a_t|s_t)}{\pi_{\psi_k}(a_t|s_t)}$ between the new policy π_ψ and the old policy π_{ψ_k} is out of the range of $(1-\epsilon)$ and $(1+\epsilon)$. The ratio is clipped at $1-\epsilon$ or $1+\epsilon$ depending on whether the advantage is positive or negative (Figure 2.2).

The PPO algorithm is summarised in Algorithm 5. Reducing computational costs while maintaining TRPO performance means that PPO is becoming increasingly popular in DRL tasks (Schulman, Wolski, Dhariwal, Radford, and Klimov, 2017; Heess, TB, Sriram, Lemmon, Merel, Wayne, Tassa, Erez, Wang, Eslami, *et al.*, 2017).

2.5 Variational inference

One of the core problems of modern statistics is to approximate a probability density that is difficult to calculate. This problem is particularly important in Bayesian statistics because modern Bayesian statistics relies on models where the posterior is difficult to compute (Blei, Kucukelbir, and McAuliffe, 2017).

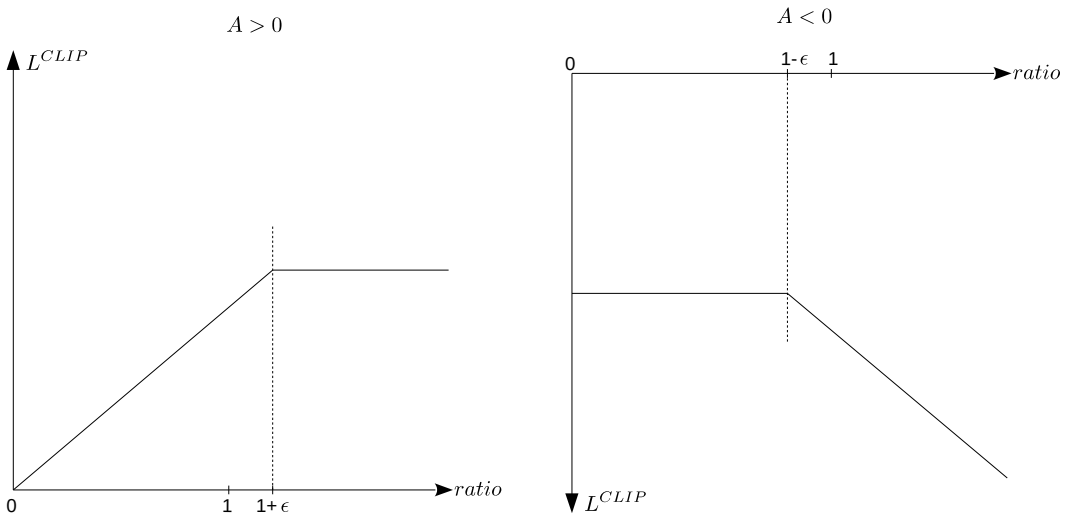


Figure 2.2: Surrogate function L^{CLIP} as a function of ratio.

Algorithm 5: Proximal policy optimization with clipped objective (PPO-CLIP)

Initial policy parameter ψ_0, threshold ϵ

for $k = 0, 1, \cdots$ **do**

Collect trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\psi_k)$

Estimate advantage function $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Compute policy update $\psi_{k+1} = \text{argmax}_\psi L_{\psi_k}^{CLIP}(\psi)$

by taking K steps of mini-batch SGD, where

$L_{\psi_k}^{CLIP}(\psi) = \mathbb{E}_{\tau\sim\psi_k}[\sum_{t=0}^{T}(\min(\frac{\pi_\psi(a_t|s_t)}{\pi_{\psi_k}(a_t|s_t)}\hat{A}_t^{\pi_k}, \text{clip}(\frac{\pi_\psi(a_t|s_t)}{\pi_{\psi_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k})]$

end

MCMC (Hastings, 1970; Gelfand and Smith, 1990) has been the main approximate inference paradigm for decades. It has been developed into an indispensable method for modern Bayesian statistics (Hastings, 1970; Geman and Geman, 1984; Gelfand and Smith, 1990; Robert and Casella, 2013).

However, when the data set is large or the model is very complex, we need an approximate algorithm faster than a simple MCMC algorithm. At this time, the variational inference is one good alternative to approximate Bayesian inference.

Variational inference (VI) is a machine learning method designed to approximate probability densities (Jordan, Ghahramani, Jaakkola, and Saul, 1999; Wainwright, Jordan, *et al.*, 2008). It is widely used in Bayesian inference to approximate the posterior density of the Bayesian model, which is an alternative method to Markov Chain Monte Carlo (MCMC) sampling. Moreover, variational inference is faster than MCMC, so it is easier to extend to large scale problems (Blei *et al.*, 2017).

Consider the joint density of observations x and latent variables z,

$$p(x, z) = p(z)p(x|z). \tag{2.51}$$

Latent variables are first extracted by the Bayesian model from prior density $p(z)$ and then related to the observations based on the likelihood $p(x|z)$. All Bayesian inference methods focus on computing the posterior $p(z|x)$. However, the computation of this posterior if often intractable when the Bayesian model is complex. So approximate inference is required.

The main idea of variational inference is to use optimization not to use sampling. Suppose we have a family of approximate densities $\mathcal{Q}$ over the latent variables. Variational inference tries to find the member of this family that minimizes the Kullback-Leibler (KL) divergence to the posterior $p(z|x)$ (See Figure 2.3):

$$q^*(z) = \underset{q(z)\in\mathcal{Q}}{\operatorname{argmin}}\, D_{KL}(q(z)||p(z|x)) \tag{2.52}$$

Then we approximate the posterior $p(z|x)$ with $q^*(z)$. We can see that the inference problem now turns into an optimization problem by variational inference. The difference between MCMC and variational inference is: MCMC sample a Markov chain and approximate the posterior with samples from the Markov chain; however, variational inference algorithms try to solve an optimization problem and approximate the posterior probability with the result of the optimization. MCMC methods often require more calculations. Variational inference methods can be faster than MCMC because they can use some optimisation tricks (Robbins and Monro, 1951; Welling and Teh,

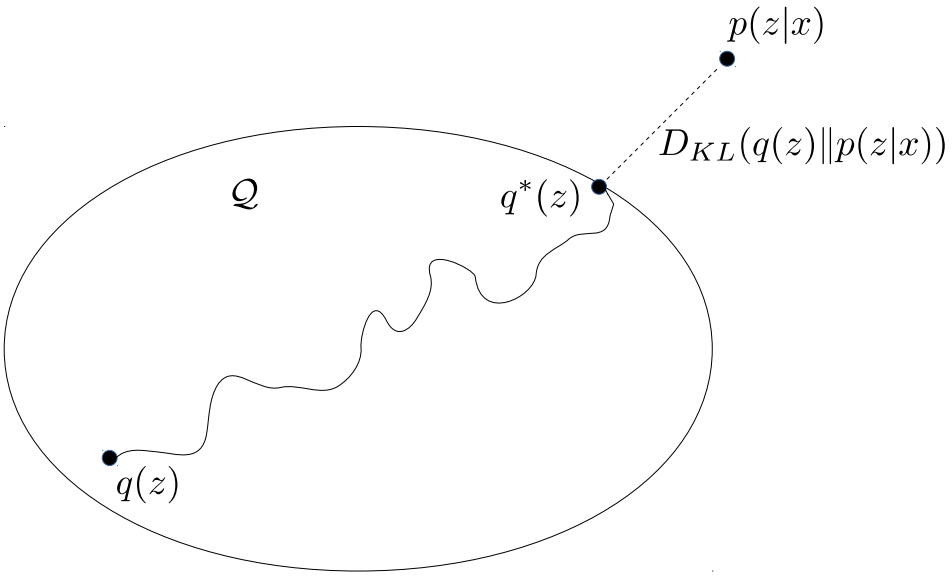


Figure 2.3: Variational inference.

2011; Ahmed, Aly, Gonzalez, Narayanamurthy, and Smola, 2012). The geometry of the posterior is another factor. When the posterior of a mixture model has multiple modes, variational inference performs better than MCMC technique, even for small datasets (Kucukelbir, Ranganath, Gelman, and Blei, 2015). It has been emphasized that both MCMC and variational inference techniques can be applied to compute intractable densities (Blei *et al.*, 2017; Tang and Agrawal, 2018).

The goal of variational inference is to find the best $q(z)$ that optimises Eq. 2.52. However, computation of the posterior $p(z|x)$ is intractable because

$$p(z|x) = \frac{p(z)p(x|z)}{p(x)}, \tag{2.53}$$

where the denominator of Eq. 2.53 (also called the *evidence*)

$$p(x) = \int p(x,z)dz \tag{2.54}$$

is hard to compute: a closed-form representation of the integral is not able to be computed in most instances or it needs exponential computational time to compute numerically.

Expanding the objective function of Eq. 2.52, we have

$$\begin{aligned} D_{KL}(q(z)\|p(z|x)) &= \mathbb{E}_q[\log q(z)] - \mathbb{E}_q[\log p(z|x)] \\ &= \mathbb{E}_q[\log q(z)] - \mathbb{E}_q[\log p(z,x)] + \log p(x). \end{aligned} \tag{2.55}$$

Denoting the variational lower bound (or evidence lower bound (ELBO)) function as:

$$\text{ELBO}(q) = \mathbb{E}_q[\log p(z, x)] - \mathbb{E}_q[\log q(z)], \tag{2.56}$$

we can see that minimizing the KL divergence is equivalent to maximizing the ELBO, because $\log p(x)$ is a constant of $q(z)$. Since the KL-divergence is always greater than or equal to zero (Kullback and Leibler, 1951), and from Eq. 2.55 and Eq. 2.56:

$$\log p(x) = D_{KL}(q(z)\|p(z|x)) + \text{ELBO}(q). \tag{2.57}$$

This means $\log p(x) \geq \text{ELBO}(q)$ for any $q(z)$, the ELBO lower-bounds the (log) evidence. Hence the ELBO function can also be rewritten as

$$\begin{aligned} \text{ELBO}(q) &= \mathbb{E}_q[\log p(x|z)p(z)] - \mathbb{E}_q[\log q(z)] \\ &= \mathbb{E}_q[\log p(x|z)] + \mathbb{E}_q[p(z)] - \mathbb{E}_q[\log q(z)] \\ &= \mathbb{E}_q[\log p(x|z)] - D_{KL}(q(z)||p(z)). \end{aligned} \tag{2.58}$$

To Maximise ELBO is to find a $q(z)$ that maximises the likelihood and at the same time $q(z)$ is as close as possible to the prior $p(z)$.

2.6 Bayesian neural networks

Bayesian neural networks (BNNs) are a combination of Bayesian statistics and deep learning. They includes uncertainty in deep learning model predictions. All weights in Bayesian neural networks (Graves, 2011; Blundell, Cornebise, Kavukcuoglu, and Wierstra, 2015) are probability distributions rather than fixed values.

The representation and calculation of learning must be robust under weight perturbations, but the amount of perturbations exhibited by each weight should also be learned in a coherent manner to explain the variability of the training data. Thus Bayesian neural networks actually train an ensemble of networks: each network has its weights drawn from a shared, learnt probability distribution. Bayesian neural network is a probabilistic model $p(y|x;\theta)$: given an input x, the BNN assigns a probability to each possible output y, using the set of parameters or weights θ. In the past, BNN models were rarely used because they require more parameters for optimization, which can make the model difficult to train. Recently, however, BNNs have become increasingly popular (Gal and Ghahramani, 2015; Shridhar, Laumann, and Liwicki, 2019; Liu, Zhao, Nacewicz, Adluru, Kirk, Ferrazzano, Styner, and Alexander, 2019; Islam,

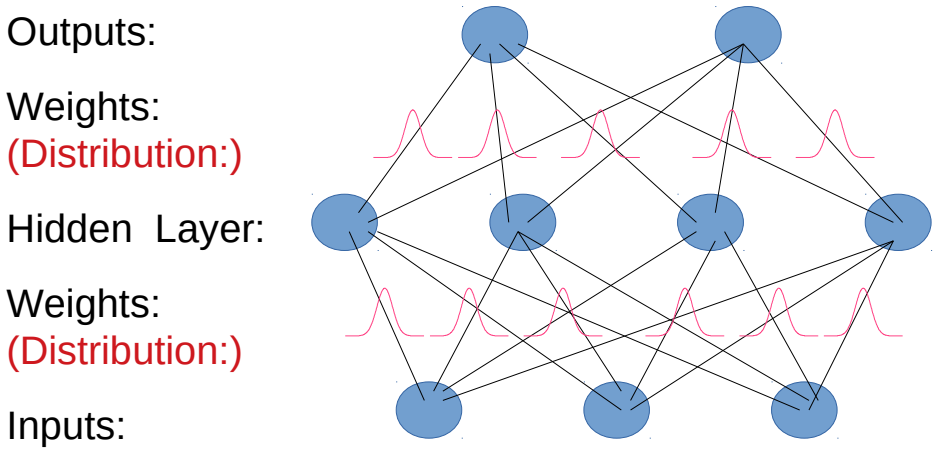


Figure 2.4: A Bayesian neural network example with one hidden layer.

2016; Zhao, Liu, Oler, Meyerand, Kalin, and Birn, 2018; Kendall, Badrinarayanan, and Cipolla, 2015; Mobiny, Nguyen, Moulik, Garg, and Wu, 2019), and new technologies are being developed to include uncertainty in models while using the same number of parameters.

Given a set of training examples $\mathcal{D} = (x_i, y_i)_i$, BNNs compute the posterior distribution of the weights $p(\theta|\mathcal{D})$ and use this distribution to compute predictions $\hat{y}$ about unseen data $\hat{x}$. The predictive distribution is given by

$$p(\hat{y}|\hat{x}) = \mathbb{E}_{p(\theta|\mathcal{D})}[p(\hat{y}|\hat{x};\theta)]. \tag{2.59}$$

Each possible setting of the weights, according to the posterior distribution, makes a prediction about the unknown label given the test data item $\hat{x}$.

Graves (2011); Blundell *et al.* (2015) and Hinton and Van Camp (1993) applied variational approximation to the Bayesian posterior distribution on the weights. Training by variational inference, a BNN can learn the parameters of these distributions instead of learning the weights directly. Based on the introduction of section 2.5, variational learning finds the parameters ϕ of a distribution on the weights $q(\theta;\phi)$ that minimises the Kullback-Leibler (KL)divergence with the true Bayesian posterior on the weights (He, Zhuang, Liu, He, and Lin, 2019). That is to say, find ϕ^*, to maximize the varia-

tional lower bound:

$$\begin{aligned}\phi^* &= \arg\min_{\phi}(D_{KL}[q(\theta;\phi)||p(\theta|\mathcal{D})]) \\ &= \arg\max_{\phi}\mathbb{E}_{\theta\sim q(\cdot;\phi)}[\log(\mathcal{D}|\theta)] - D_{KL}[q(\theta;\phi)||p(\theta)] \\ &= \arg\min_{\phi} D_{KL}[q(\theta;\phi)||p(\theta)] - \mathbb{E}_{\theta\sim q(\cdot;\phi)}[\log(\mathcal{D}|\theta)].\end{aligned}$$

The first term $D_{KL}[q(\theta;\phi)||p(\theta)]$ is the model complexity cost and the second term is the expected value of the likelihood.

Uncertainty in predictions caused by the uncertainty in weights is called epistemic uncertainty (Der Kiureghian and Ditlevsen, 2009; Matthies, 2007). More data can reduce this uncertainty. Therefore, in areas where there is little data, the epistemic uncertainty is higher, and in areas where there is more training data, it is lower. The uncertainty from the built-in noise in the data is called aleatoric uncertainty. Even if we get more data, we can't reduce it. In this thesis, we'll use a Gaussian distribution for the variational posterior, parameterised by $\phi = (\mu, \sigma)$. Here μ is the mean vector of the Gaussian distribution and σ the standard deviation. Instead of parameterising the neural network with weights θ directly, we train μ and σ and therefore double the number of parameters compared to a basic neural network.

2.7 Intrinsic reward driven exploration

Recall that in RL, at time step $t \geq 0$, the agent is in state $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$, receives extrinsic reward r_t^e and transitions to the next state $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. The objective of RL is to find a policy π to maximize the discounted cumulative reward:

$$J = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty}\gamma^t r_t^e], \tag{2.60}$$

where π is a mapping (neural network in this thesis) from a state to a distribution over actions so that $a_t \sim \pi(\cdot|s_t)$, $\gamma \in [0, 1)$ is the discount factor. These rewards r_t^e are considered as extrinsic rewards because they are provided by the environment and are task-specific.

Many amazing results have been obtained with the help of extrinsic rewards. However, when rewards are sparse in the environment, these methods are unsuccessful in most cases because the agent cannot subsequently learn the desired behaviour of the target task (Florensa, Held, Geng, and Abbeel, 2018). Moreover, behaviours learned

by agents in a special task is hardly reusable. Then it is difficult for agents to generalize abstract decisions in the environment (Sutton, Precup, and Singh, 1999; Bengio, Louradour, Collobert, and Weston, 2009; Aubret *et al.*, 2019). This requires us to add intrinsic rewards r_t^i to encourage the agent to explore efficiently to find the sparse extrinsic rewards (Schmidhuber, 1991b,a; Ryan and Deci, 2000; Silvia, 2012; Pathak *et al.*, 2017; Barto, 2013). The objective function of RL in Eq. 2.60 can be rewritten as:

$$J = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t (r_t^e + \eta r_t^i)], \tag{2.61}$$

where η is the trade-off between the intrinsic reward and extrinsic reward. Intrinsic rewards can be used to explain the need to explore the environment and to discover target states, and more broadly as a way to learn new skills, just like babies or other organisms spontaneously explore their environment (Gopnik, Meltzoff, and Kuhl, 1999; Georgeon, Marshall, and Ronot, 2011; Baranes and Oudeyer, 2013).

In recent years, intrinsic reward-driven exploration is increasingly used in reinforcement learning and can be grouped into three broad classes:

The first is to encourage agents to explore "new" states (Lopes, Lang, Toussaint, and Oudeyer, 2012; Poupart, Vlassis, Hoey, and Regan, 2006), which is also called "count-based" method. Counted-based method adds an intrinsic reward based on a state visit count function:

$$r_t^i = \frac{1}{n(s_t)}, \tag{2.62}$$

where $n(s_t)$ is the number of times agent has visited state s_t (counting the first visit as $n(s_t) = 1$). However, this method is obviously only computationally tractable for environments with relatively small number of discrete states.

The second belongs to the category of methods that predict the next state given the current state. The intrinsic reward is computed by the difference between the prediction and the true value of the next state, which can be derived from the definition of *surprisal*.

The third is to encourage agents to perform actions that reduce uncertainty in predictions of the consequences of their own behaviour (Houthooft *et al.*, 2016; Mohamed and Rezende, 2015; Schmidhuber, 1991b, 2010; Chentanez, Barto, and Singh, 2005). The uncertainty is reduced when the agent maximises the information gain about its belief of environmental dynamics. This is done equivalently by maximising the *Bayesian surprise* (Itti and Baldi, 2005, 2006).

Since the latter two classes (surprisal based and Bayesian surprise based) are both related to surprise, we can collectively call them "surprise driven exploration". This thesis will focus on surprise driven exploration and I will introduce surprisal and Bayesian surprise in details in the following section.

2.7.1 Surprise-driven reinforcement learning

All surprise driven exploration methods discussed in this thesis are based on model-based RL. In model-based RL, the agent maintains one or more models $M \in \mathcal{M}$, (where $M : \mathcal{S} \times \mathcal{A} \to \mathcal{S}, \mathcal{M}$ the model/hypotheses space), that predict the next state s_{t+1} based on the current state s_t and the action a_t about to be taken. The model prediction is denoted as $\widehat{s_{t+1}}$.

There has been a number of definitions of surprise, some of which have previously appeared in the literature. We use $U(s_{t+1})$ to denote the surprise in all cases in this thesis and to form the basis for intrinsic reward to encourage exploration of *unexpected* states (Houthooft *et al.*, 2016; Achiam and Sastry, 2017; Pathak *et al.*, 2017; Mohamed and Rezende, 2015; Schmidhuber, 1991b; Chentanez *et al.*, 2005). The agent is rewarded for surprise of the *unknown* as gauged by its model of the environment. Throughout training, the model is improved to be more accurate (and so less surprised) next time it encounters an already explored state. The hope is that this methodical approach to exploration will result in a speedier arrival of the agent at the states with non-zero extrinsic reward. In this thesis, the surprise is considered as graded rather than binary (surprised or not surprised), different values of $U(s)$ denote different levels of surprise.

Surprisal

One popular used surprise definition is the so-called *surprisal* (Traibus, 1961), which is the negative log-likelihood (NLL) of the next state in RL tasks

$$U_{\text{NLL}}(s_{t+1}) = -\log P(s_{t+1}|s_t, a_t). \tag{2.63}$$

In information theory, it is also called information content or self-information, which represents the quantity of information brought by new events. The quantity can interpret the level of "surprise" of this event. If an event with a low likelihood occurs, then the surprisal will be large.

Although Tribus's definition of surprisal does not explicitly mention conditional probability, there is always an implicit assumption that surprisal depends on the context

or model (Barto, Mirolli, and Baldassarre, 2013). So when focusing one specific model M, we can assume that the surprisal is conditioned on this model. Then we can rewrite Eq. 2.63 as

$$U^M_{\text{NLL}}(s_{t+1}) = -\log P(s_{t+1}|s_t, a_t, M). \tag{2.64}$$

Without causing confusion, we can use the term surprisal to mean both $U_{\text{NLL}}(s_{t+1})$ and $U^M_{\text{NLL}}(s_{t+1})$. And without special instructions, surprisal refers specifically to $U^M_{\text{NLL}}(s_{t+1})$.

Methods based on surprisal in recent years (Achiam and Sastry, 2017; Pathak *et al.*, 2017) all adopted Eq 2.64. We can see from Eq. 2.64 that maximising the surprisal is equivalent to maximising the negative log-likelihood of the state, given the model M. Agents choose actions on their environment to suppress the difference between their predictions and actual experience to avoid being surprised.

Surprisal $U^M_{\text{NLL}}(s_{t+1})$ is intuitive, simple and easy to compute, but it does not handle stochastic environments well, because it only measures the surprise at a single point (point estimate) so that the model can be over confident.

Bayesian surprise

An alternative is *Bayesian surprise*. In Bayesian surprise's definition, the agent captures its environment's background information by a prior belief distribution $\{P(M|s_t, a_t)\}_{M\in\mathcal{M}}$, where $\mathcal{M}$ is the model space. Bayesian surprise then measures the difference between the prior belief $P(M|s_t, a_t)$ and the posterior belief distribution $P(M|s_t, a_t, s_{t+1})$ over the model space $\mathcal{M}$, and the posterior belief distribution updated by Bayes' rule with the newly observed state (first used in RL by Storck *et al.* (1995)):

$$P(M|s_t, a_t, s_{t+1}) = \frac{P(M|s_t, a_t)P(s_{t+1}|s_t, a_t, M)}{P(s_{t+1}|s_t, a_t)} \tag{2.65}$$

Formally, Bayesian surprise is defined as the Kullback-Leiber (KL) divergence between the prior and the posterior beliefs about the dynamics of the environment (Itti and Baldi, 2005, 2006):

$$U_{\text{Bayes}}(s_{t+1}) = D_{KL}[P(M|s_t, a_t)||P(M|s_t, a_t, s_{t+1})]. \tag{2.66}$$

One problem with Bayesian surprise is that the agent does not express surprise until it updates its belief, which is inconsistent with the instantaneous response to surprise displayed by neural data (Faraji, 2016) and will also be time-consuming.

Faraji, Preuschoff, and Gerstner (2016) introduced a modification of Bayesian surprise referred to as the *confidence-corrected surprise* (CC). They first assumed the

agent has a flat prior (uniform distribution, the agent believes that all models are equally likely) belief, then for each novel state s_{t+1}, they gave the flat posterior belief distribution about M derived by Bayes' rule:

$$P^{\text{flat}}(M|s_t, a_t, s_{t+1}) = \frac{P(s_{t+1}|s_t, a_t, M)}{\int_M P(s_{t+1}|s_t, a_t, M)dM}. \tag{2.67}$$

The confidence-corrected surprise is a measure of difference between an agent's prior and this flat posterior belief,

$$U_{\text{CC}}(s_{t+1}) \quad = \quad D_{KL}[P(M|s_t, a_t)||P^{\text{flat}}(M|s_t, a_t, s_{t+1})]. \tag{2.68}$$

This flat posterior is computed by the Bayesian update rule and therefore is also time-consuming to compute.

Both surprisal and Bayesian surprise have already been applied as an intrinsic reward to deep reinforcement learning in recent years. To use Bayesian surprise for RL tasks, Houthooft *et al.* (2016) proposed a surprise-driven exploration strategy called VIME (variational information maximizing exploration). The idea is that agents should take actions to minimise dynamic uncertainty, which is equivalent to maximising the sum of reductions in entropy (Sun, Gomez, and Schmidhuber, 2011):

$$\sum_t (H(M|s_t, a_t) - H(M|s_t, a_t, s_{t+1})). \tag{2.69}$$

From Eq. 2.8 we have:

$$I(s_{t+1}; M|s_t, a_t) = H(M|s_t, a_t) - H(M|s_t, a_t, s_{t+1}), \tag{2.70}$$

We can see from Eq. 2.69 and Eq. 2.70 that for each time step t, the agent is trying to maximise the mutual information $I(s_{t+1}; M|s_t, a_t)$ between s_{t+1} and the model M. And

$$\begin{aligned}
& I(s_{t+1}; M|s_t, a_t) \\
= & \int_{\mathcal{S}} \int_{\mathcal{M}} P(s_{t+1}, M|s_t, a_t) \log \frac{P(s_{t+1}, M|s_t, a_t)}{P(s_{t+1}|s_t, a_t)P(M|s_t, a_t)} dM ds_{t+1} \\
= & \int_{\mathcal{S}} \int_{\mathcal{M}} P(M|s_t, a_t, s_{t+1})P(s_{t+1}|s_t, a_t) \log \frac{P(M|s_t, a_t, s_{t+1})P(s_{t+1}|s_t, a_t)}{P(s_{t+1}|s_t, a_t)P(M|s_t, a_t)} dM ds_{t+1} \\
= & \int_{\mathcal{S}} \int_{\mathcal{M}} P(M|s_t, a_t, s_{t+1})P(s_{t+1}|s_t, a_t) \log \frac{P(M|s_t, a_t, s_{t+1})}{P(M|s_t, a_t)} dM ds_{t+1} \\
= & \int_{\mathcal{S}} P(s_{t+1}|s_t, a_t) \int_{\mathcal{M}} P(M|s_t, a_t, s_{t+1}) \log \frac{P(M|s_t, a_t, s_{t+1})}{P(M|s_t, a_t)} dM ds_{t+1} \\
= & \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)}[D_{KL}[P(M|s_t, a_t, s_{t+1})||P(M|s_t, a_t)]].
\end{aligned} \tag{2.71}$$

The last line of Eq. 2.71 shows that maximising the mutual information $I(s_{t+1}; M|s_t, a_t)$ is equivalent to maximising

$$D_{KL}[P(M|s_t, a_t, s_{t+1})||P(M|s_t, a_t)], \tag{2.72}$$

which is a reversed form of Bayesian surprise $D_{KL}[P(M|s_t, a_t)||P(M|s_t, a_t, s_{t+1})]$ we defined in Eq. 2.66. Although KL-divergence is not symmetric, experiments in VIME paper showed that there is no significant difference between these two KL divergence variants when we choose them as the intrinsic reward.

VIME achieves significantly better performance compared to heuristic exploration methods across a variety of continuous control tasks with sparse rewards. However, to compute each reward, VIME needs to calculate the gradient at each time step, through a Bayesian neural network (BNN)(Graves, 2011; Blundell *et al.*, 2015) used to implement the agent's model. This requires a forward and a backward pass, which means if the trajectory length is T, VIME needs an extra T back gradient calculations. This will definitely result in reduced training speed. Achiam and Sastry (2017) compared VIME with surprisal and showed that although surprisal does not perform as well as VIME, it is much faster. So Burda, Edwards, Pathak, Storkey, Darrell, and Efros (2018) chose this surprisal as an intrinsic reward and apply it to train the agent in large scale RL environments.

So in summary, the intrinsic reward-driven exploration paradigm enhances an agent's ability to explore the environment, and learn skills independently of the specific task (Aubret *et al.*, 2019), and even create state representations with meaningful features. Since intrinsic reward-driven exploration does not need expert supervision, it is easy to generalise across environments.

Chapter 3

VASE: Variational Assorted Surprise Exploration

Note: Some portions of this chapter are taken from my own work (Xu, McCane, and Szymanski, 2019).

Exploration in environments with continuous control and sparse rewards remains a key challenge in reinforcement learning (RL). Recently, surprise has been used as an intrinsic reward that encourages systematic and efficient exploration. In this chapter, we introduce a new definition of surprise and its RL implementation named Variational Assorted Surprise Exploration (VASE). VASE uses a Bayesian neural network as a model of the environment dynamics and is trained using variational inference, alternately updating the accuracy of the agent's model and policy. Our experiments show that in continuous control sparse reward environments VASE outperforms surprisal and Bayesian surprise based methods.

3.1 Introduction

RL trains agents to act in an environment so as to maximise cumulative reward. The resulting behaviour is highly dependent on the trade-off between exploration and exploitation. During training, the more the agent departs from its current policy, the more it learns about the environment, which may lead it to a better policy; the closer it adheres to the current policy, the less time wasted exploring less effective options. How much and where to explore has an immense impact on the training and ultimately on what the agent learns. Designing exploration strategies, especially for increasingly complex environments, is still a significant challenge.

A common approach to exploration strategies is to rely on heuristics that introduce random perturbations into the choices of actions during training, such as ϵ–greedy (Sutton and Barto, 1998) or Boltzmann exploration (Mnih *et al.*, 2015). These methods instruct the agent to occasionally take an arbitrary action that may drive it into a new experience. Another way is through the addition of noise to the parameter space of the agent's policy neural network (Fortunato, Azar, Piot, Menick, Hessel, Osband, Graves, Mnih, Munos, Hassabis, Pietquin, Blundell, and Legg, 2018; Plappert, Houthooft, Dhariwal, Sidor, Chen, Chen, Asfour, Abbeel, and Andrychowicz, 2018), which varies the policy itself to a similar random exploration net effect. These strategies can be highly inefficient because they are a result of random behaviour, which is especially problematic in high dimension state-action spaces (common in discretised continuous state-action space environments) because of the curse of dimensionality. Random exploration is also extremely inefficient in environments with sparse rewards, where the agent ends up wandering aimlessly through the state-space, learning nothing until (by sheer luck) it chances upon a reward.

Not surprisingly, more methodical approaches were devised, which provide the agent with intrinsic rewards that encourage efficient exploration. The idea that agents should explore by intrinsic reward can be traced back to early 1990's. In 1991, Schmidhuber (1991a) proposed that an agent trained by reinforcement learning can translate mismatches between expectations and reality into curiosity/surprise rewards. The agents are driven to explore surprising aspects of the world, and hence to explore the environment efficiently. This idea has been inherited and carried forward for the next thirty years, especially in recent years. Thanks to increases in computing power, people have verified this idea in large-scale data and scenarios (Achiam and Sastry, 2017; Pathak *et al.*, 2017; Burda *et al.*, 2018; Houthooft *et al.*, 2016). As we introduced in Chapter 2 section 2.7, Pathak *et al.* (2017); Burda *et al.* (2018) and Achiam and Sastry (2017) choose surprisal as the intrinsic reward, which belongs to the prediction-error method class. And Houthooft *et al.* (2016) proposed an algorithm VIME that tries to maximise the Bayesian surprise, it belongs to the information gain method class.

In this chapter, we propose a new definition of surprise, which we call *assorted surprise*. The assorted surprise drives our agents' intrinsic reward function. It can also handle uncertainty well, but it is more efficient than Bayesian surprise-driven exploration.

To compute and use assorted surprise for guiding exploration, we propose an algorithm called VASE (Variational Assorted Surprise Exploration) in a model-based RL

framework (see Figure 3.1).

VASE alternates the update step between the agent's policy and its model of the environment dynamics. The policy is implemented with a multilayer feed-forward (MLFF) neural network and the dynamics model with a Bayesian neural network (BNN (Graves, 2011; Blundell *et al.*, 2015; Hinton and Van Camp, 1993)). We evaluate the performance of our method against other surprise-driven methods on continuous control tasks with sparse rewards. Our experimental results show VASE's superior performance.

3.2 Variational assorted surprise exploration (VASE)

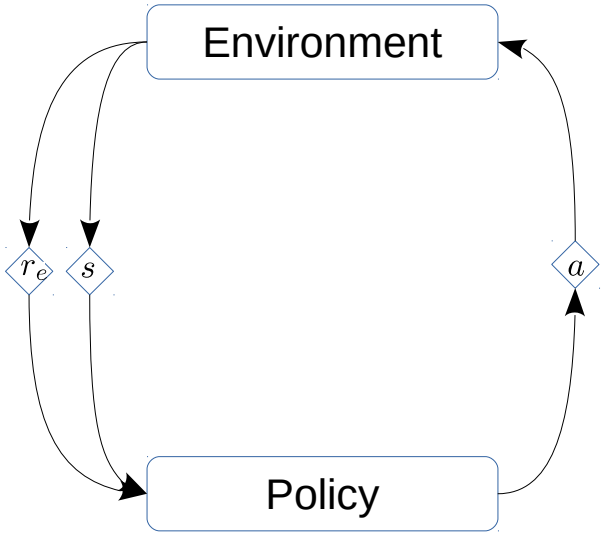
In this chapter, we focus on continuous control RL environments with sparse rewards. The state space and action space are both continuous spaces, and these environments only provide very sparse extrinsic rewards. So the agent needs an intrinsic-reward-driven strategy to explore the environment and collect these sparse extrinsic rewards provided by the environment.

3.2.1 Assorted surprise

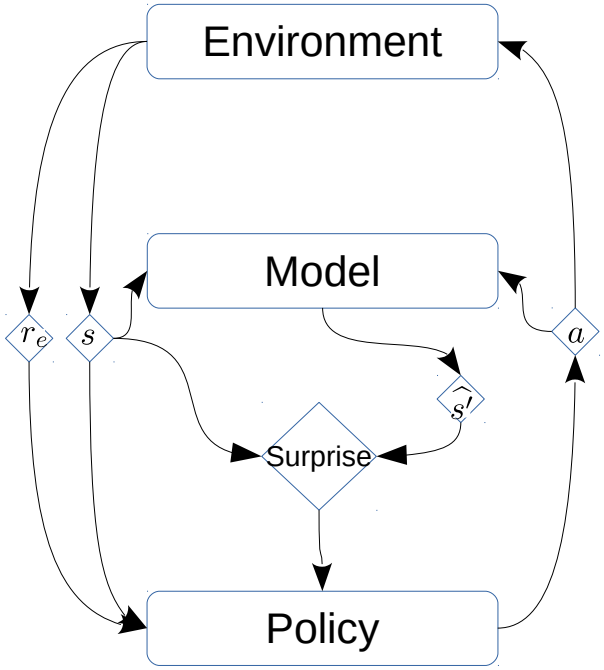
To conquer the numerous shortcomings of existing definitions for surprise discussed in Section 2.7.1, we propose a novel intrinsic reward called *assorted surprise*, and inspired by the idea of confidence-corrected surprise (Faraji *et al.*, 2016), we note that in RL the definition of surprise should have the following characteristics:

1. *subjectivity* – "subjectivity" here has two meanings. On the one hand, different observers will have different surprises about the same event. Similarly, if the agent has multiple models at the same time, then for the same state s_{t+1}, the surprisal $U^M_{\mathrm{NLL}}(s_{t+1})$ corresponding to each model is different. On the other hand, the agent should hold subjective beliefs about the environment captured through $\{P(M)\}$; the surprise depends on an agent's belief and this belief can be updated during learning;

2. *consistency* – based on the same belief, the agent should be more surprised by states with lower likelihood; on the contrary, when the agent observes a state that has lower likelihood, if it has a high confident (low entropy) belief at this time, then it should be more surprised.



(a) Model-free RL



(b) Model-based/surprise-driven RL

Figure 3.1: Model-free RL vs. model-based/surprise-driven RL with s for state, r_e for the extrinsic (environment-driven) reward, a for action, surprise for intrinsic (agent-driven) reward, the model of the environment makes a prediction of the next state $\widehat{s'}$.

3. *instancy* – the agent should be surprised immediately when it observes a new state from the environment, without the need to update its belief first.

Obviously Bayesian surprise $U_{\text{Bayes}}(s_{t+1})$ doesn't satisfy the instancy characteristic, because we can only compute the Bayesian surprise after the agent updates its belief. Furthermore, this update will also cause time-consuming. For the surprisal $U^{M}_{\text{NLL}}(s_{t+1})$, since it only considers one single model M, we always get same surprise based on the same event.

Our assorted surprise idea comes from ensemble learning (Opitz and Maclin, 1999; Rokach, 2010). Ensemble learning is a technique that trains multiple models to solve the same problem, and combines them to get better, more reliable predictions with lower variance and/or lower bias. We suppose the agent has created many models, each model can predict the next state s_{t+1} that generates different surprisal:

$$\begin{aligned}
M_1: &\quad U^{M_1}_{\text{NLL}}(s_{t+1}) = -\log P(s_{t+1}|s_t, a_t, M_1) \\
M_2: &\quad U^{M_2}_{\text{NLL}}(s_{t+1}) = -\log P(s_{t+1}|s_t, a_t, M_2) \\
&\quad \vdots \\
&\quad \vdots \\
M_k: &\quad U^{M_k}_{\text{NLL}}(s_{t+1}) = -\log P(s_{t+1}|s_t, a_t, M_k) \\
&\quad \vdots \\
&\quad \vdots
\end{aligned} \tag{3.1}$$

Based on ensemble learning theory (Kuncheva and Whitaker, 2003; Sollich and Krogh, 1996), when there are large differences among models, ensemble techniques tend to produce better results. So we ensemble these models together and give our assorted surprise (AS):

$$\begin{aligned}
& P(M_1|s_t, a_t)U^{M_1}_{\text{NLL}}(s_{t+1}) + P(M_2|s_t, a_t)U^{M_2}_{\text{NLL}}(s_{t+1}) + \cdots \\
= & \int_{\mathcal{M}} P(M|s_t, a_t)\left(-\log P(s_{t+1}|s_t, a_t, M)\right) dM \\
= & \ \mathbb{E}_{M\sim P(\cdot|s_t,a_t)}[-\log P(s_{t+1}|s_t, a_t, M)].
\end{aligned} \tag{3.2}$$

In addition, to address the above consistency characteristic, we subtract the entropy of $P(M|s_t, a_t)$, and use this final form as the intrinsic reward to drive the agent to explore, which we refer to as *assorted surprise for exploration* (ASE):

$$U_{\text{ASE}}(s_{t+1}) = \mathbb{E}_{M\sim P(\cdot|s_t,a_t)}[-\log P(s_{t+1}|s_t, a_t, M)] - \delta H\big(P(M|s_t, a_t)\big), \tag{3.3}$$

the first term, $\mathbb{E}_{M\sim P(\cdot|s_t,a_t)}[-\log P(s_{t+1}|s_t,a_t,M)]$, we call the assorted surprise term, and the second term, $H\big(P(M|s_t,a_t)\big)$, the confidence term. Hyper-parameter δ is a trade-off between the surprise term and confidence term.

We will next demonstrate that the assorted surprise satisfies all three characteristics we mentioned above. Subjectivity comes from the assorted surprise term in Eq. 3.3 because it is an expectation over the agent's belief in the veracity of each of the models. It is also the sum of the Bayesian surprise $U_{\text{Bayes}}(s_{t+1})$ and the surprisal $U_{\text{NLL}}(s_{t+1})$ (not $U^M_{\text{NLL}}(s_{t+1})$) as shown in the following lemma.

Lemma 1 (Assorted surprise is the sum of Bayesian surprise and surprisal)**.**

$$\mathbb{E}_{M\sim P(\cdot|s_t,a_t)}[-\log P(s_{t+1}|s_t,a_t,M)] = U_{Bayes}(s_{t+1}) + U_{NLL}(s_{t+1}) \tag{3.4}$$

Proof.

$$\begin{aligned}
& \mathbb{E}_{M\sim P(\cdot|s_t,a_t)}[-\log P(s_{t+1}|s_t,a_t,M)] \\
= & -\int_{\mathcal{M}} P(M|s_t,a_t)\log P(s_{t+1}|s_t,a_t,M)dM \\
= & -\int_{\mathcal{M}} P(M|s_t,a_t)\log \frac{P(M|s_t,a_t,s_{t+1})P(s_{t+1}|s_t,a_t)}{P(M|s_t,a_t)}dM \\
= & \int_{\mathcal{M}} P(M|s_t,a_t)\log \frac{P(M|s_t,a_t)}{P(M|s_t,a_t,s_{t+1})}dM - \int_{\mathcal{M}} P(M|s_t,a_t)\log P(s_{t+1}|s_t,a_t)dM \\
= & D_{KL}[P(M|s_t,a_t)||P(M|s_t,a_t,s_{t+1})] - \log P(s_{t+1}|s_t,a_t) \\
= & U_{\text{Bayes}}(s_{t+1}) + U_{\text{NLL}}(s_{t+1}),
\end{aligned}$$

□

This also means that assorted surprise is subjective due to the contribution of U_{Bayes}. However, the expectation in Eq. 3.3 does not require evaluation of $P(M|s_t,a_t,s_{t+1})$, so there is no requirement to update the agent's belief in order to compute assorted surprise thus satisfying the instancy characteristic.

The confidence term of Eq. 3.3, $H(P(M|s_t,a_t))$, is the entropy of $P(M|s_t,a_t)$. This term was added for confidence correction of assorted surprise so that satisfies the consistency requirement. A confident agent will have a low entropy, and therefore any surprising event according to the assorted surprise term will remain surprising. Whereas an uncertain agent will have a large entropy and their overall surprise will be reduced because they would be equally surprised by many events. That is, confident agents are more surprised when their beliefs are violated by unlikely events than uncertain agents.

Assorted surprise captures the positive elements of both Bayesian surprise and confidence-corrected surprise. It implicitly computes the difference in belief as in Bayesian surprise without needing to update the belief first, and it can be computed very fast as in confidence-corrected surprise without needing to maintain the idea of a naive observer. In this chapter, it is hypothesised that:

(3.1) The agent driven by assorted surprise will perform better than those driven by surprisal.

(3.2) Training speed of assorted surprise algorithm runs faster than Bayesian surprise based algorithm.

(3.3) In the environment where rewards can be obtained at each time step, adding intrinsic reward will not reduce the agent's exploration efficiency.

3.2.2 VASE implementation

We have discussed the advantages of assorted surprise, but before applying it to practical problems, we need to solve some issues. First, many ensemble methods like boosting (Schapire, 1990) and bagging (Breiman, 1996) combine small learners together to convert weak learners to strong ones (Zhou, 2012). However, this is infeasible if each model M is a deep neural network with a large number of parameters. We don't want to create so many big models. In addition, computing the posterior distribution $P(M|s_t, a_t)$ is intractable.

There are four possible regimes for creating the agent's model:

1. There is a single deterministic model
2. There is one model that produces a distribution over new states
3. There is a distribution of models, each of which is deterministic
4. There is a distribution of models, each of which produces a distribution over states.

As an example, the model for case 1 could be a traditional neural network, case 2 variational auto-encoder, case 3 Bayesian neural network and case 4 Bayesian variational autoencoder. In cases 2-3, the outcome is a distribution over states and we are free to choose the most convenient formalism. For this chapter we choose case 3 as in (Baldi and Itti, 2010). With the help of Bayesian neural network and variational inference technique introduced in Chapter 2, we can solve the above issues at the same time.

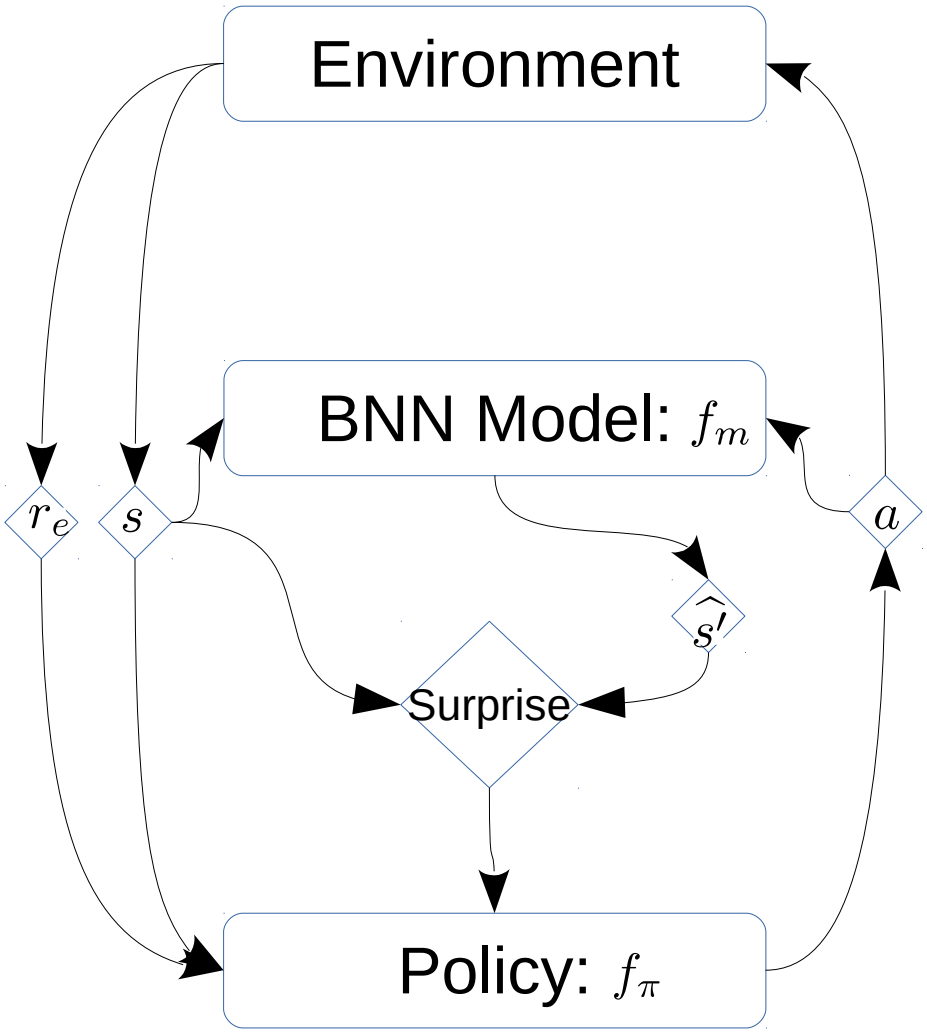


Figure 3.2: VASE-driven RL with s for state, r_e for the extrinsic (environment-driven) reward, a for action, surprise for intrinsic (agent-driven) reward, and $\widehat{s'}$ the output of BNN model.

We construct a BNN dynamics model $f_m(s_t, a_t, \Theta)$, where $\Theta \in \mathbf{\Theta}$ is a random variable describing the parameters of the model (see Figure 3.2), $\mathbf{\Theta}$ the parameter probability space. BNN can be seen as a distribution of models M, where a sample of network parameters θ according to distribution $P(\theta)$ is analogous to generating a single prediction of the next state according to $P(M)$. Note that each BNN has the same structure, which means $\mathbf{\Theta} \subset \mathcal{M}$. The prior distribution $P(\theta)$ changes to posterior $P(\theta|\mathcal{D})$ when BNN is trained by $\mathcal{D} = \{s_t, a_t, s_{t+1}\}$.

Since the posterior $P(\theta|s_t, a_t)$ in Eq. 3.3 is intractable, we turn to variational inference (Bishop, 2006) to approximate it with a fully factorised Gaussian distribution (Graves, 2011; Blundell *et al.*, 2015; Hinton and Van Camp, 1993)

$$q(\theta; \phi) = \prod_{i=1}^{|\Theta|} \mathcal{N}(\theta_i; \mu_i, \sigma_i^2), \tag{3.5}$$

where θ_i is the i^{th} component of θ, and $\phi_i = (\mu_i, \sigma_i)$. $q(\theta; \phi)$ is also called variational posterior distribution. The use of $q(\theta; \phi)$ in place of $P(\theta|s_t, a_t)$ changes the definition of surprise from Eq. 3.3 to one we call variational assorted surprise for exploration (VASE):

$$U_{\text{VASE}}(s_{t+1}) = \mathbb{E}_{\theta \sim q(\cdot;\phi)}[-\log P(s_{t+1}|s_t, a_t, \theta)] - \delta H(q(\theta; \phi)). \tag{3.6}$$

Since the output of the model for sample θ gives the prediction of the next state $\widehat{s}_{t+1} = f_m(s_t, a_t, \theta)$, we define $P(s_{t+1}|s_t, a_t, \theta)$ by measuring the deviation of $\widehat{s}_{t+1}$ from s_{t+1} under the assumption that states are normally distributed:

$$P(s_{t+1}|s_t, a_t, \theta) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-||\widehat{s}_{t+1} - s_{t+1}||^2/(2\sigma_c^2)}, \tag{3.7}$$

where σ_c is an arbitrarily chosen constant, $||\widehat{s}_{t+1} - s_{t+1}||$ is the norm of the difference vector between the prediction of the next state and the true next state.

N samples of $\theta \sim q(\cdot; \phi)$ give N predictions for the next state from the BNN, which allows us to estimate the first term of Eq. 3.6 with the average:

$$\mathbb{E}_{\theta \sim q(\cdot;\phi)}[-\log P(s_{t+1}|s_t, a_t, \theta)] \approx \frac{1}{N} \sum_{n=1}^{N} \left(-\log P(s_{t+1}|s_t, a_t, \theta^{[n]})\right), \tag{3.8}$$

where $\theta^{[n]}$ is the n^{th} sample of Θ drawn from $q(\theta; \phi)$ and $P(s_{t+1}|s_t, a_t, \theta^{[n]})$ is evaluated according to Eq. 3.7.

Since $q(\theta;\phi)$ is a fully factorised Gaussian distribution, the second term of Eq. 3.6 is straightforward to evaluate:

$$\begin{aligned} H\big(q(\theta;\phi)\big) &= \sum_{i=1}^{|\Theta|} H(\mathcal{N}(\theta_i;\mu_i,\sigma_i^2)) \\ &= \frac{1}{2}\sum_{i=1}^{|\Theta|}\log(2\pi e\sigma_i^2). \end{aligned} \tag{3.9}$$

The last thing remaining is to ensure $q(\theta;\phi)$ is as close as possible to $P(\theta|\mathcal{D})$. Variational inference uses Kullback-Leibler (KL) divergence for measuring how different $q(\theta;\phi)$ is from $P(\theta|\mathcal{D})$:

$$\begin{aligned} & D_{KL}[q(\theta;\phi)||P(\theta|\mathcal{D})] \\ = & \int_\theta q(\theta;\phi)\log\frac{q(\theta;\phi)}{P(\theta|\mathcal{D})}d\theta \\ = & D_{KL}[q(\theta;\phi)||P(\theta)] - \mathbb{E}_{\theta\sim q(\cdot;\phi)}[\log P(\mathcal{D}|\theta)] + \log P(\mathcal{D}). \end{aligned} \tag{3.10}$$

This difference is minimised by changing ϕ, which is equivalent to maximising the variational lower bound (Bishop, 2006):

$$L[q(\theta;\phi),\mathcal{D}] = \mathbb{E}_{\theta\sim q(\cdot;\phi)}[\log P(\mathcal{D}|\theta)] - D_{KL}[q(\theta;\phi)||P(\theta)], \tag{3.11}$$

which does not require evaluation of $P(\theta|\mathcal{D})$. In this thesis, the prior distribution of θ is taken to be

$$P(\theta) = \prod_{i=1}^{|\Theta|}\mathcal{N}(\theta_i;0,\sigma_m^2), \tag{3.12}$$

where σ_m is set to arbitrary value, and the expectation of log likelihood of $P(\mathcal{D}|\theta)$ is evaluated as in Eq. 3.8.

When training BNN, the local reparameterisation trick (Kingma, Salimans, and Welling, 2015) is used (See Figure 3.3). In VASE's original form, it samples from θ, which is a random node and approximated by $q(\theta;(\mu,\sigma))$ of the true posterior. Back-propagation algorithm cannot flow through a random node. A new parameter ϵ is introduced to reparameterise θ, so as to back-propagate through the deterministic nodes.

The entire training procedure is listed in Algorithm 6. In this chapter, we choose Trust Region Policy Optimization (TRPO) (Schulman *et al.*, 2015) that we introduced in Chapter 2 as our RL policy update algorithm in all of our experiments. However, it should be noted that our surprise-driven method can be embedded into any other RL algorithms.

Algorithm 6: Variational Assorted Surprise Exploration (VASE)

Initialise policy neural network f_π with parameters ψ

Initialise agent's BNN model f_m:

  Initialise $q(\theta;\phi)$ with parameters ϕ

  Initialise prior distribution $P(\theta)$

Initialise experience buffer R.

Reset the environment getting $(s_0, r_e(s_0))$.

for *each iteration* n **do**

  for *each time step* t **do**

    Get action $a_t \sim f_\pi(s_t, \psi)$

    Sample θ N times according to $q(\theta;\phi)$

    Evaluate N predictions $\widehat{s}_{t+1} = f_m(s_t, a_t, \theta)$

    Take action a_t getting $(s_{t+1}, r_e(s_{t+1}))$

    Compute intrinsic surprise $U_{\text{VASE}}(s_{t+1})$

    Construct cumulative reward $r(s_{t+1}) = r_e(s_{t+1}) + \eta U_{\text{VASE}}(s_{t+1})$

    Add new $(s_t, a_t, s_{t+1}, r(s_{t+1}))$ to R

  end

  Update f_m by maximising Eq. 3.11, with $\mathcal{D}$ sampled randomly from R

  Update f_π using TRPO.

end

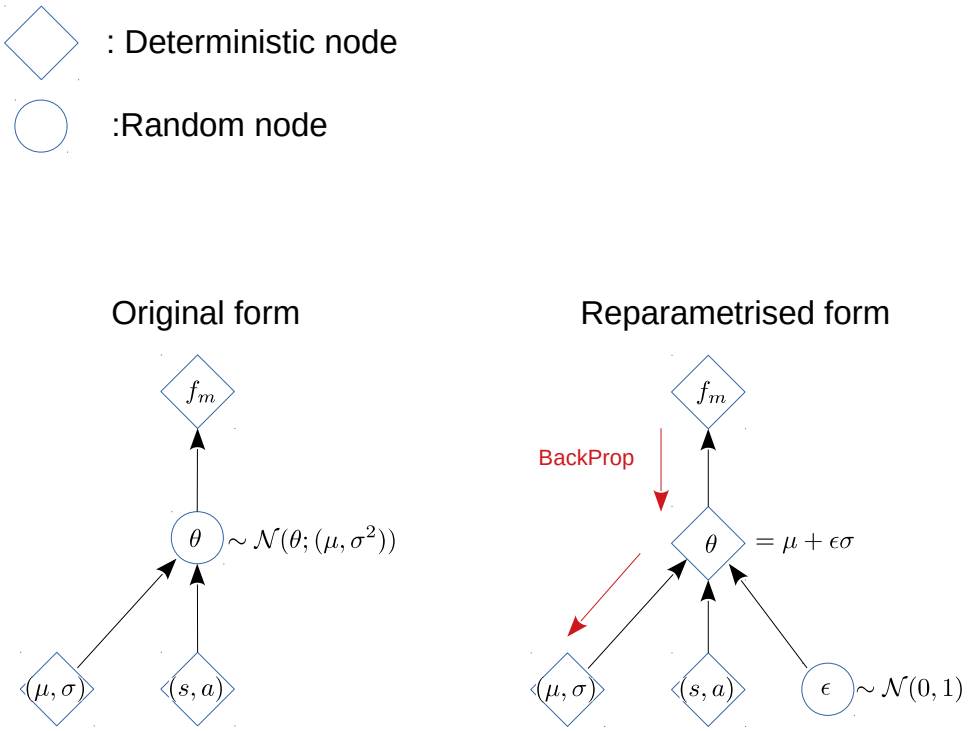


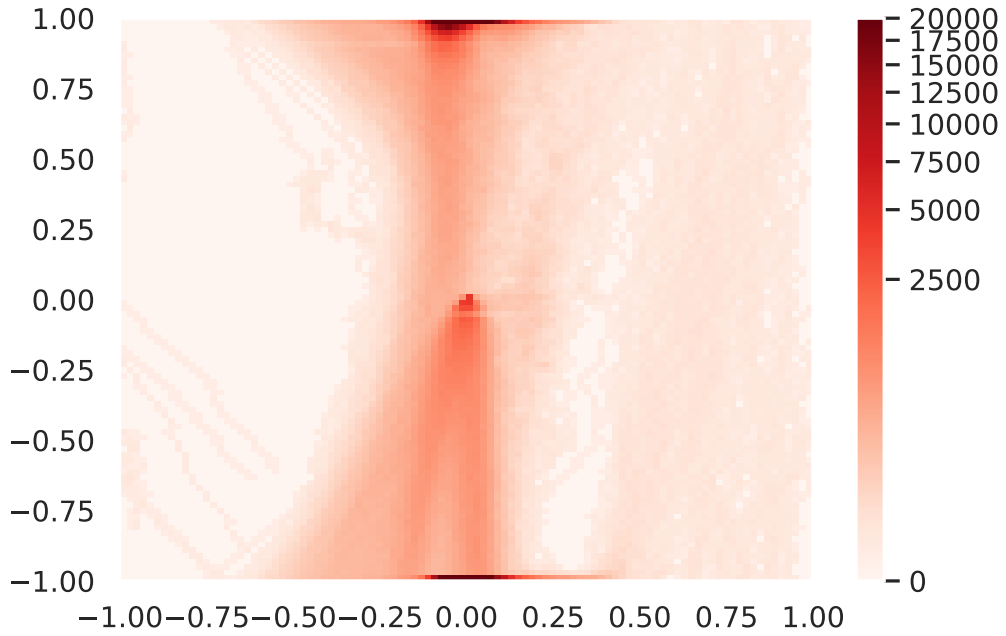
Figure 3.3: Reparameterisation trick.

3.3 VASE for continuous control tasks with sparse rewards

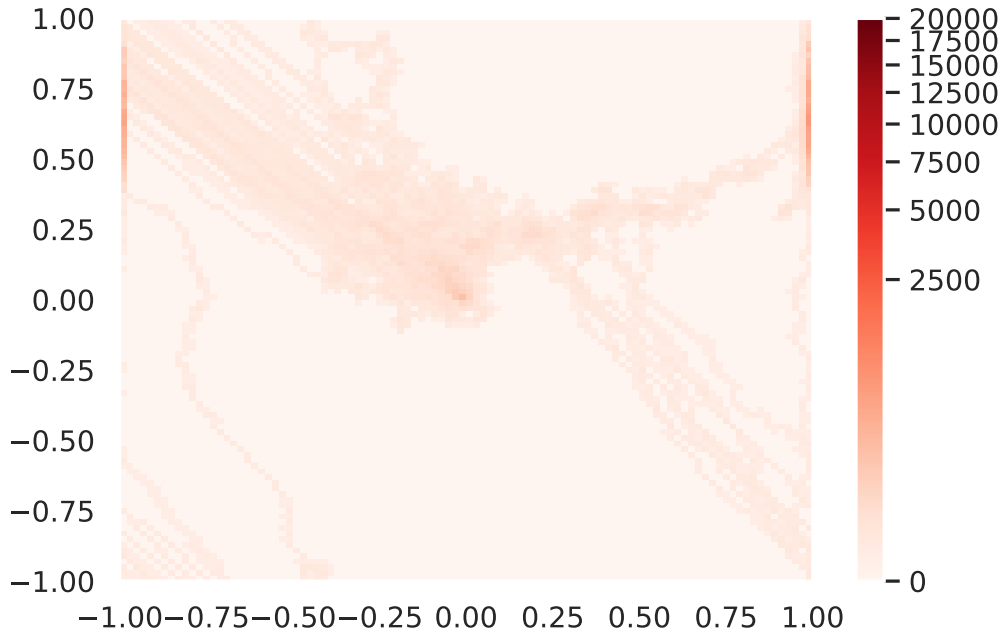
3.3.1 Visualising exploration efficiency

For illustrative purposes, we begin the experimental evaluation of VASE by testing it on a simple 2D Plane environment ($\mathcal{S} \subset \mathbb{R}^2, \mathcal{A} \subset \mathbb{R}^2$) which lends itself to a visualisation of the agent's exploration efficiency. The observation space is a square on the 2D plane ($(x, y) \in [-1, 1] \times [-1, 1]$), centred on the origin. The action is its velocity $(\dot{x}, \dot{y})$ that satisfies $|\dot{x}| \leq 0.01, |\dot{y}| \leq 0.01$. In this environment, the agent starts at the origin (0,0), and the only external reward can be found in a circle of radius 0.01 centred on (1, 1). When the agent enters the circle, an external reward can be found. The environment wraps around, so that there are no boundaries.

In this experiment, we train one agent and record the observation coordinate (x, y) in each step until it finds the non zero extrinsic reward. Figure 3.4 shows the heat map of motion track for the agent trained without surprise and with VASE surprise. Darker red colour represents a higher density, which means the agent explore more steps in this area. It is clear that random exploration strategy takes a long time (2,059,459



(a) TRPO (2,059,459 steps)

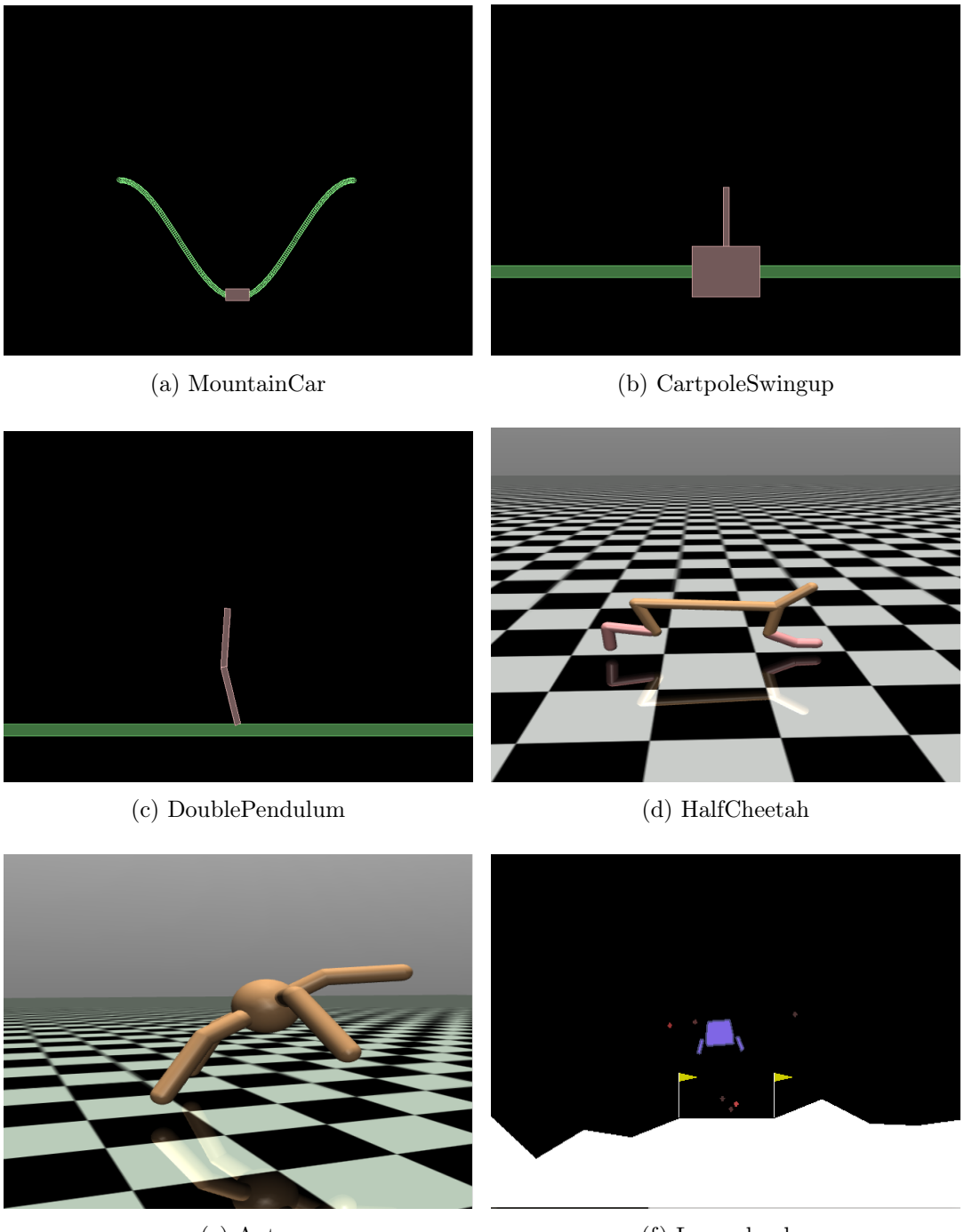


(b) TRPO+VASE (26,663 steps)

Figure 3.4: Exploration efficiency as a heatmap showing the number of states visited during training in 2DPointRobot environment until chancing upon the reward state with a) no surprise, b) VASE.

steps vs 26,663 steps) to find that first non-zero $r_e(s_t)$, whereas VASE does not spend time unnecessarily in random states.

3.3.2 Continuous state/action environments



(a) MountainCar (b) CartpoleSwingup

(c) DoublePendulum (d) HalfCheetah

(e) Ant (f) Lunar-lander

Figure 3.5: Six continuous control environments, all of which have a sparse reward.

To test the hypothesis (3.1) and (3.2) proposed in 3.2.1, next we evaluate VASE on five continuous control benchmarks with very sparse reward, including three classic tasks: sparse MountainCar ($\mathcal{S} \subset \mathbb{R}^3, \mathcal{A} \subset \mathbb{R}^1$), sparse CartPoleSwingup ($\mathcal{S} \subset \mathbb{R}^4, \mathcal{A} \subset \mathbb{R}^1$), sparse Doublependulum ($\mathcal{S} \subset \mathbb{R}^6, \mathcal{A} \subset \mathbb{R}^1$) and two locomotion tasks: sparse HalfCheetah ($\mathcal{S} \subset \mathbb{R}^{20}, \mathcal{A} \subset \mathbb{R}^6$), sparse Ant ($\mathcal{S} \subset \mathbb{R}^{125}, \mathcal{A} \subset \mathbb{R}^8$). These tasks were introduced in (Houthooft *et al.*, 2016). See simulations for more details.[1] We also evaluate VASE on LunarLanderContinuous ($\mathcal{S} \subset \mathbb{R}^8, \mathcal{A} \subset \mathbb{R}^2$) task (See Figure 3.5) to test hypothesis (3.3).

For the sparse MountainCar task, the car will climb a one-dimensional hill to reach the target. The target is located on top of a hill and on the right-hand side of the car. If the car reaches the target, the episode is done. The observation is given by the horizontal position and the horizontal velocity of the car. The agent car receives a reward of 1 only when it reaches the target.

For the sparse CartpoleSwingup task, a pole is mounted on a cart. The cart itself is limited to linear motion. Continuous cart movement is required to keep the pole upright. The system should not only be able to balance the pole, but also be able to swing it to an upright position first. The observation includes the cart position x, pole angle β, the cart velocity $\dot{x}$, and the pole velocity $\dot{\beta}$. The action is the horizontal force applied to the cart. The agent receives a reward of 1 only when $cos(\beta) > 0.9$.

For the sparse Doublependulum task, the goal is to stabilise a two-link pendulum at the upright position. The observation includes joint angles (β_1 and β_2) and joint speeds ($\dot{\beta}_1$ and $\dot{\beta}_2$). The action is the same as in CartpoleSwingup task. The agent receives a reward of 1 only when dist < 0.1, with dist the distance between current pendulum tip position and target position.

For the sparse HalfCheetah task, the half-cheetah is a flat biped robot with nine rigid links, including two legs and one torso, and six joints. 20-dimensional observations include joint angle, joint velocity, and centroid coordinates. The agent receives a reward of 1 when $x_{body} \geq 5$.

For the sparse Ant task, the ant has 13 rigid links, including four legs and a torso, along with 8 actuated joints. The 125-dim observation includes joint angles, joint velocities, coordinates of the centre of mass, a (usually sparse) vector of contact forces, as well as the rotation matrix for the body. The ant receives a reward of 1 when $x_{body} \geq 3$.

For Lunar-lander task, the agent tries to learn to fly and then land on its landing

[1] `https://drive.google.com/open?id=1kpBvOHPYNeaEvZB9Jo8bOiQtO7rErYQr`

pad. The episode is done if the lander crashes or comes to rest. The agent should get rewards of 200 when it solves this task.

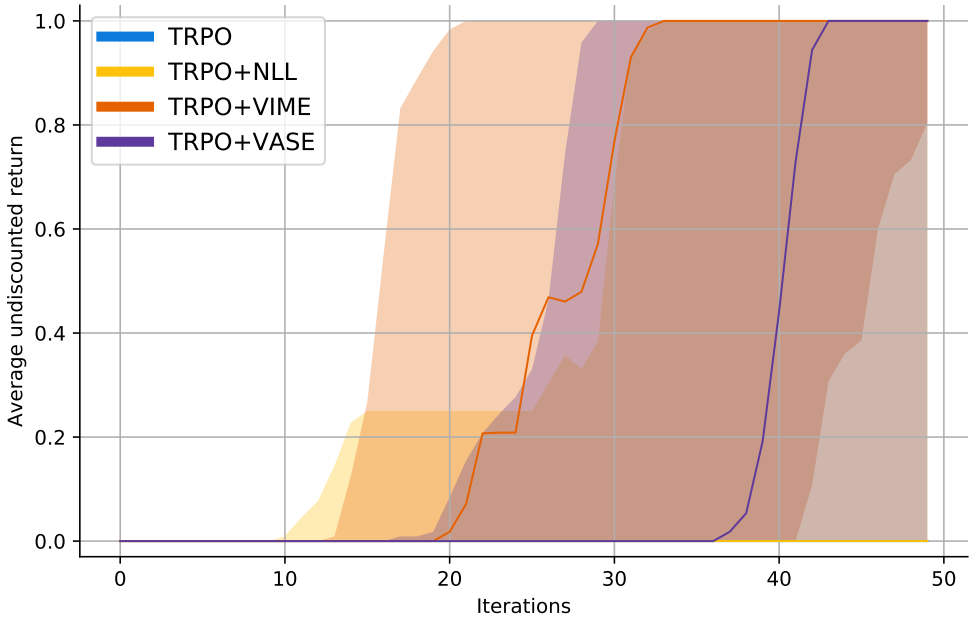
All the environments except MountainCar and 2DPointRobot tasks (they are two simple tasks, we do not need to normalise them) are normalised before the algorithm starts. Here normalise the task means normalise its observations, for each observation o:

$$o = \frac{(o - \mu_o)}{\sigma_o},$$

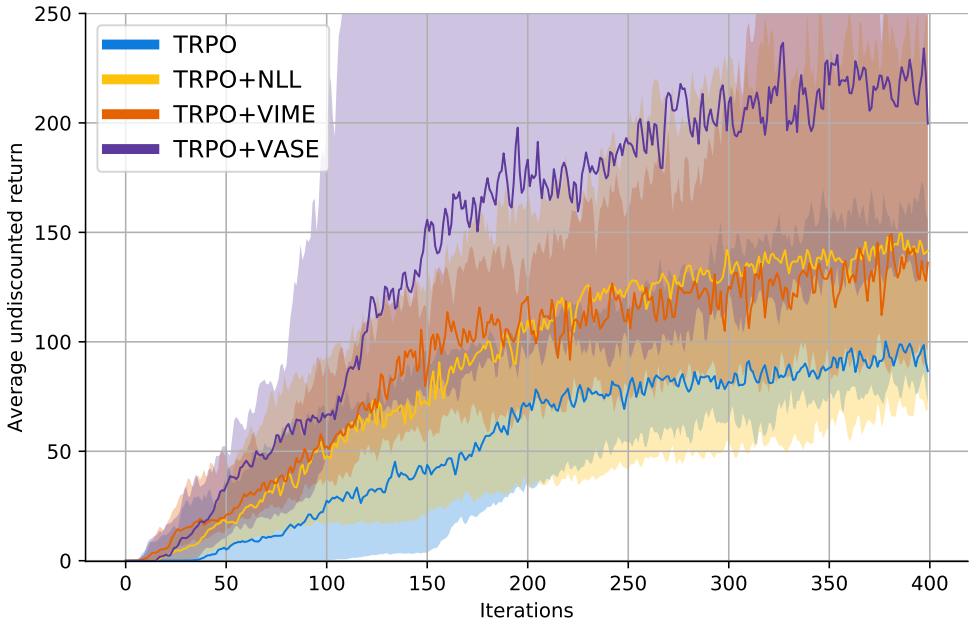
where μ_o and σ_o are the mean and standard deviation of observations. All observations and actions in these environments are continuous values. To compare with Achiam and Sastry (2017) and Houthooft *et al.* (2016), we use Trust Region Policy Optimization (TRPO) (Schulman *et al.*, 2015) method (Section 2.4.1) as our base reinforcement learning algorithm throughout our experiments, and we use the rllab (Duan, Chen, Houthooft, Schulman, and Abbeel, 2016) implementations of TRPO.

Similarly, in order to compare with surprisal and VIME(Achiam and Sastry, 2017; Houthooft *et al.*, 2016), the selection of hyper-parameters also refers to these methods. For example, the extrinsic reward and intrinsic reward trade-off η in Algorithm 6 is set to 10^{-4}, the number N of samples drawn to compute our surprise in Eq. 3.8 is set to 10. The prior distribution $P(\theta)$ is given by a Gaussian distribution from Eq. 3.12 with $\sigma_m = 0.5$. σ_c in Eq. 3.7 is set as 5. For the classic tasks: sparse MountainCar, sparse CartPoleSwingup, sparse DoublePendulum and sparse LunarLanderContinuous, the f_m has one hidden layer of 32 units. All hidden layers have rectified linear unit (ReLU) non-linearities. The replay pool R has a fixed size of 100,000 samples, with a minimum size of 500 samples. For the locomotion tasks sparse HalfCheetah and sparse Ant, the f_m has two hidden layers of 64 units each. All hidden layers have tanh non-linearities. The replay pool R has a fixed size of 5,000,000 samples. The Adam learning rate of f_m is set to 0.001. All output layers are set to linear. The batch size for the policy optimisation is set to 5,000. For f_π the classic tasks use a neural network with one layer of 32 tanh units, while the locomotion task uses a two-layer neural network of 64 and 32 tanh units. For baseline, the classic tasks use a neural network with one layer of 32 ReLU units, while the locomotion task uses a linear function. The maximum length of trajectory for LunarLanderContinuous is 1000, for all the other tasks, it is 500.

Figure 3.6 (a)-(e) shows the median performance of three classic control tasks and two locomotion tasks. All these tasks are with sparse rewards. Figure 3.6 (f) shows the median performance of LunarLanderContinuous task. The agent can easily obtain rewards from this task. The performance is measured through the average return

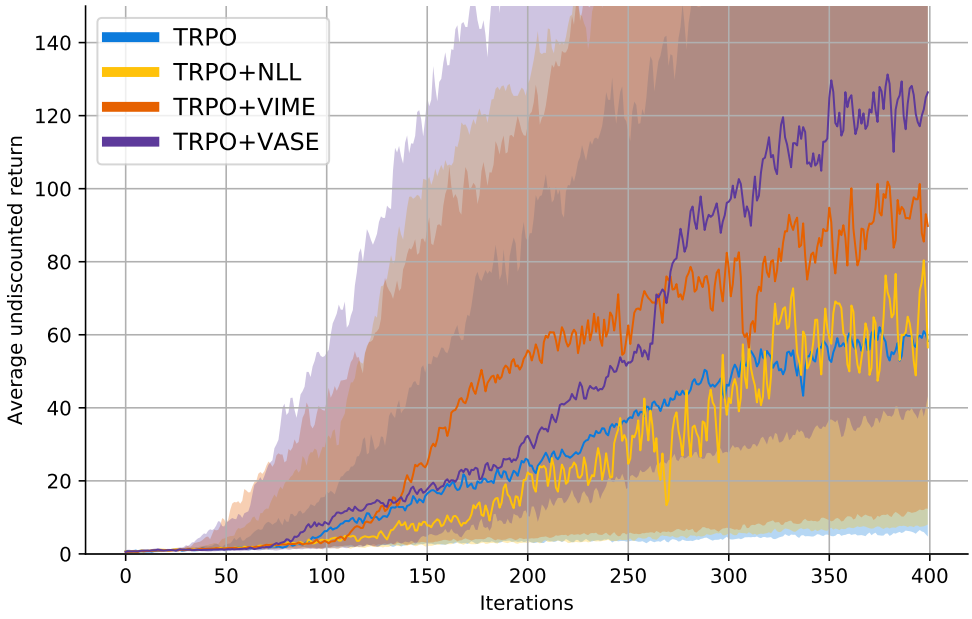


(a) MountainCar

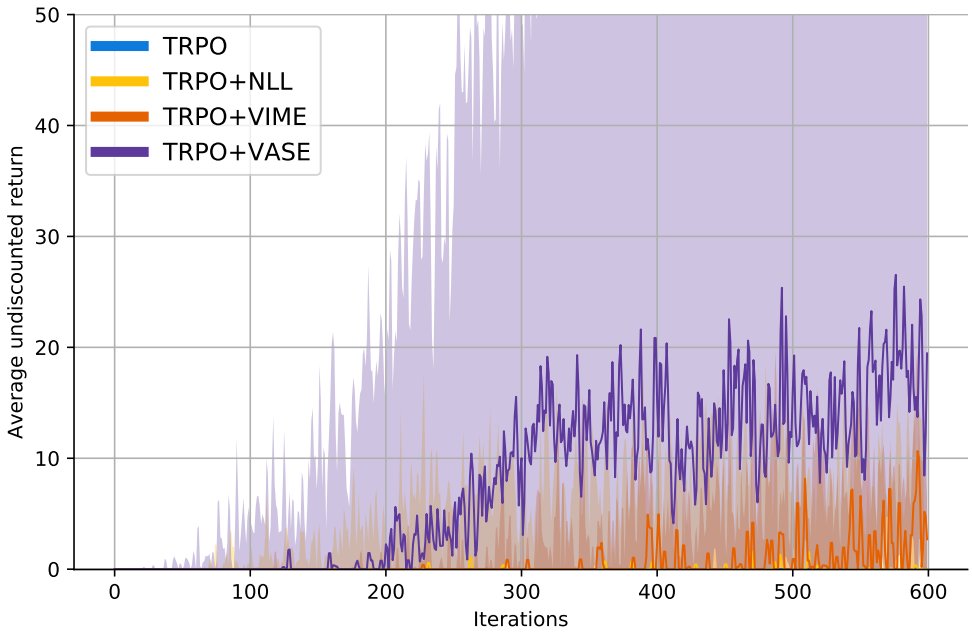


(b) CartpoleSwingup

Figure 3.6: Median performance for the continuous control tasks over 20 runs with a fixed set of seeds, with interquartile ranges shown in shaded areas. VIME, NLL and VASE use Bayesian surprise, surprisal, our surprise respectively (continued on next page).

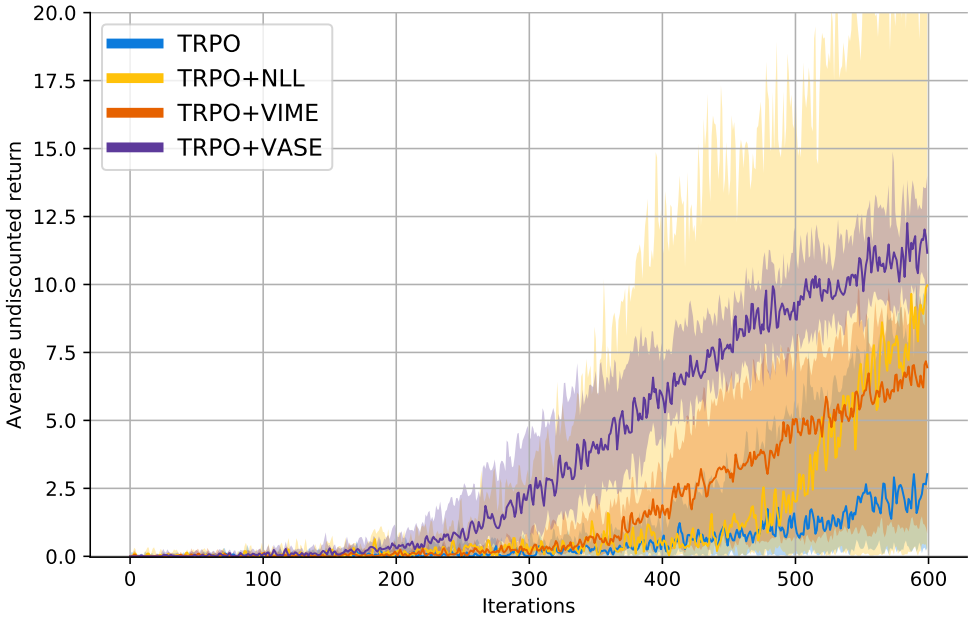


(c) DoublePendulum

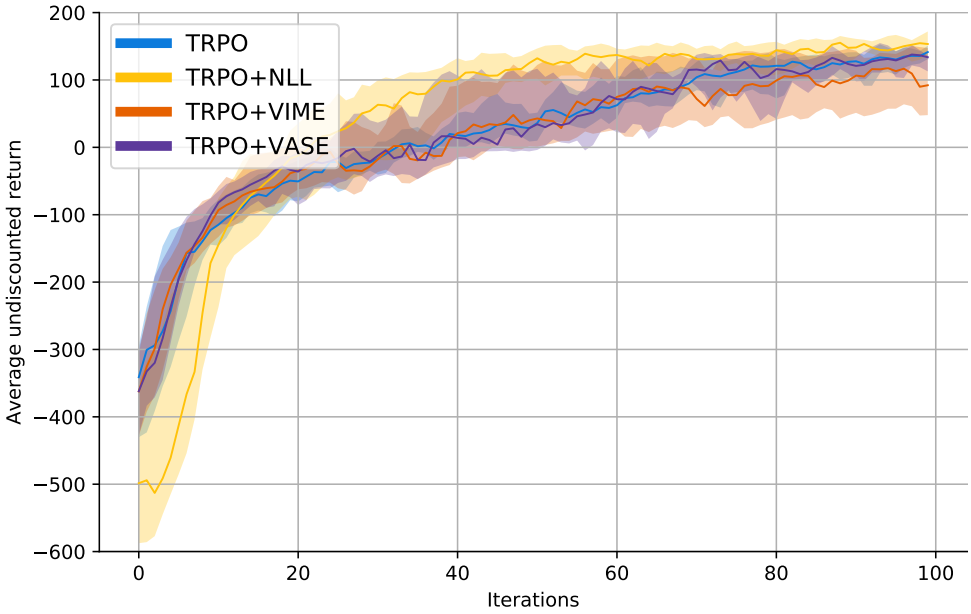


(d) HalfCheetah

Figure 3.6: Median performance for the continuous control tasks over 20 runs with a fixed set of seeds, with interquartile ranges shown in shaded areas. VIME, NLL and VASE use Bayesian surprise, surprisal, our surprise respectively (continued on next page).



(e) Ant



(f) Lunar-lander

Figure 3.6: Median performance for the continuous control tasks over 20 runs with a fixed set of seeds, with interquartile ranges shown in shaded areas. VIME, NLL and VASE use Bayesian surprise, surprisal, our surprise respectively (continued from previous page).

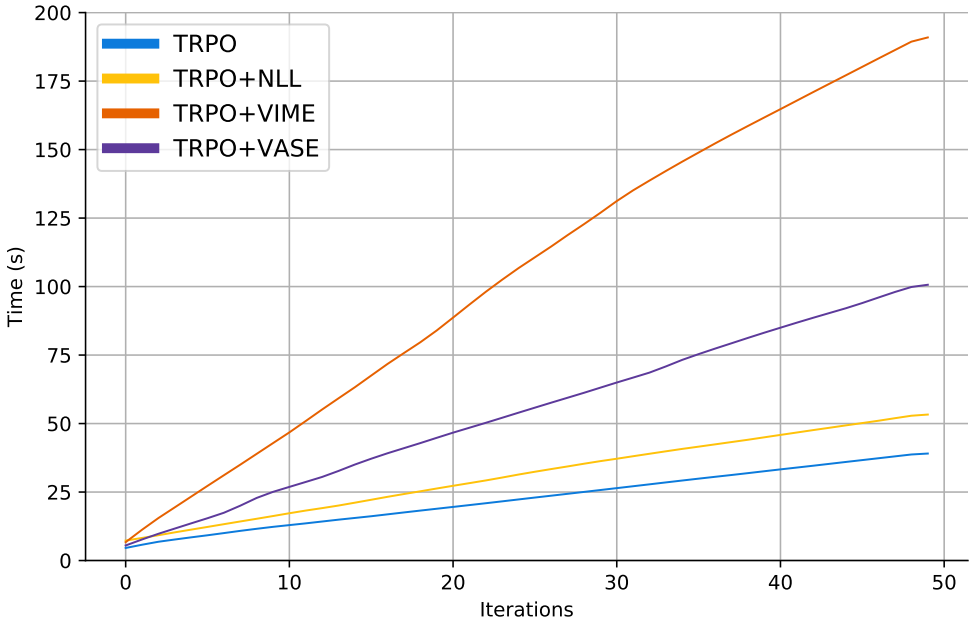
$\mathbb{E}_\tau[\sum_{t=0}^{T} r_e(s_t)]$, not including the intrinsic rewards. The median performance curves with shaded interquartile ranges areas as shown. Figure 3.7 shows the speed comparison on all tasks.

As can be seen from Figure 3.6 (a)-(e), VIME performs best for sparse MountainCar task. For the sparse DoublePendulum, VIME performs well initially, but is later surpassed by VASE. VASE shows good results in sparse CartpoleSwingup, sparse HalfCheetah and sparse Ant tasks. We can also see that VASE always performs better than NLL (suprisal) in all tasks. This supports hypothesis (3.1). However, it should be noted that the high variance of the return weakens the claim that the performance difference between methods is significant (See Ant task iteration 300-600 in Figure 3.6 (e)). To assess whether the return of VASE and NLL algorithms ranks differ in Ant task, for each iteration from iteration 300 to iteration 600, the returns of VASE and NLL algorithms are picked and paired from all different 20 trials, then the difference between each pair is computed. A non-parametric statistical hypothesis test called Wilcoxon signed-rank test (Wilcoxon, 1946) is chosen to test whether the differences between each pair follow a symmetric distribution around zero. The two-sided test has null hypothesis that the median of the return differences is zero against the alternative that it is different from zero. We applied the two-sided test and get the p-value of 0.046. Hence, we would reject null hypothesis at the significance level of 5%, concluding that there is a significant difference between the returns of VASE and NLL algorithms.

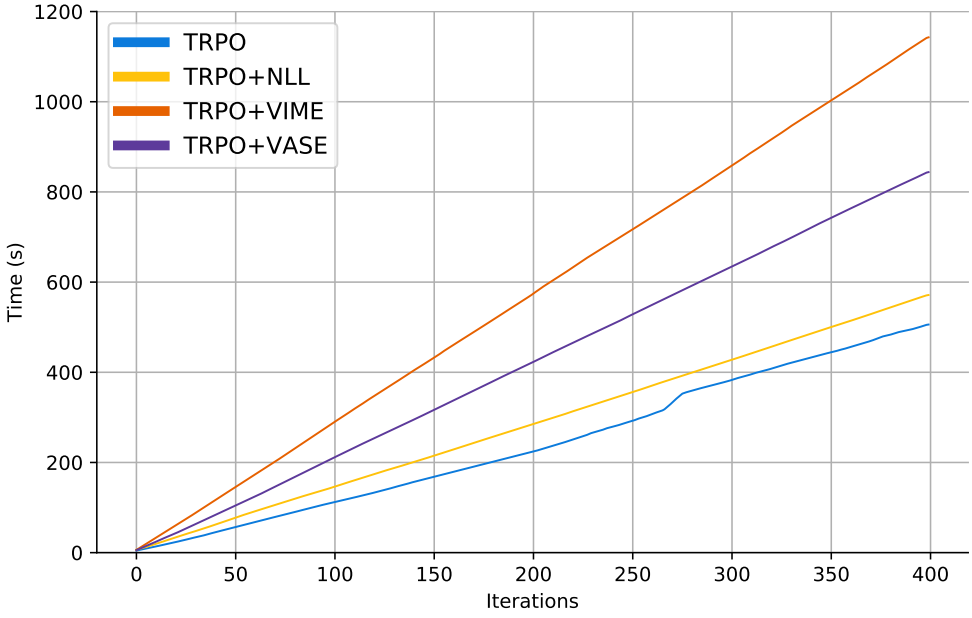
Figure 3.6 (f) shows that in LunarLanderContinuous task that has enough reward for the agent, all surprise-driven methods behave almost the same as the no-surprise method, which supports hypothesis (3.3).

Figure 3.7 shows us the speed test results. For VIME, it needs to calculate a gradient through its BNN at each time step to compute the Bayesian surprise reward. This is really time consuming. However, for our VASE algorithm, it does not need to compute this gradient. Figure 3.7 shows that VASE runs much faster than VIME, which supports hypothesis (3.2).

Finally, we also test how different values of trade-off δ that we used in Eq. 3.6 affects the performance of surprise U_{VASE} on all five sparse rewards environments. We know that the value of $H(q(\theta;\phi))$ depends not only on the distribution of each parameter θ_i, but also on the number of parameters $|\boldsymbol{\Theta}|$. Meanwhile, in the beginning stages of training, the entropy of each parameter θ_i is relatively large, therefore, we should take a relatively small value of δ. Figure 3.8 shows VASE performance based on δ chosen from $\{0, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1\}$. For each environment and each δ, the experiment

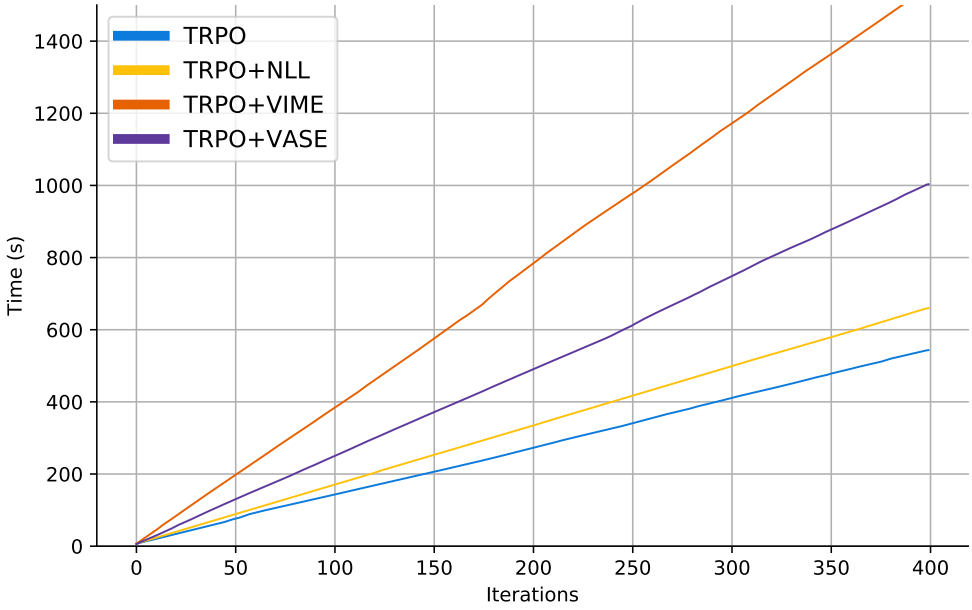


(a) MountainCar

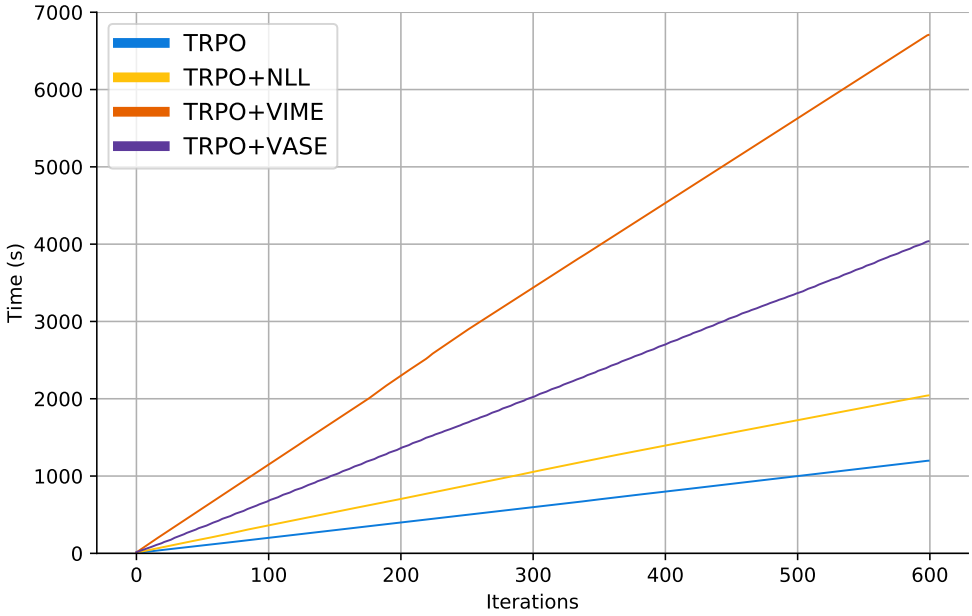


(b) CartpoleSwingup

Figure 3.7: Running time comparison on six environments. VIME, NLL and VASE use Bayesian surprise, surprisal, our surprise respectively (continued on next page).

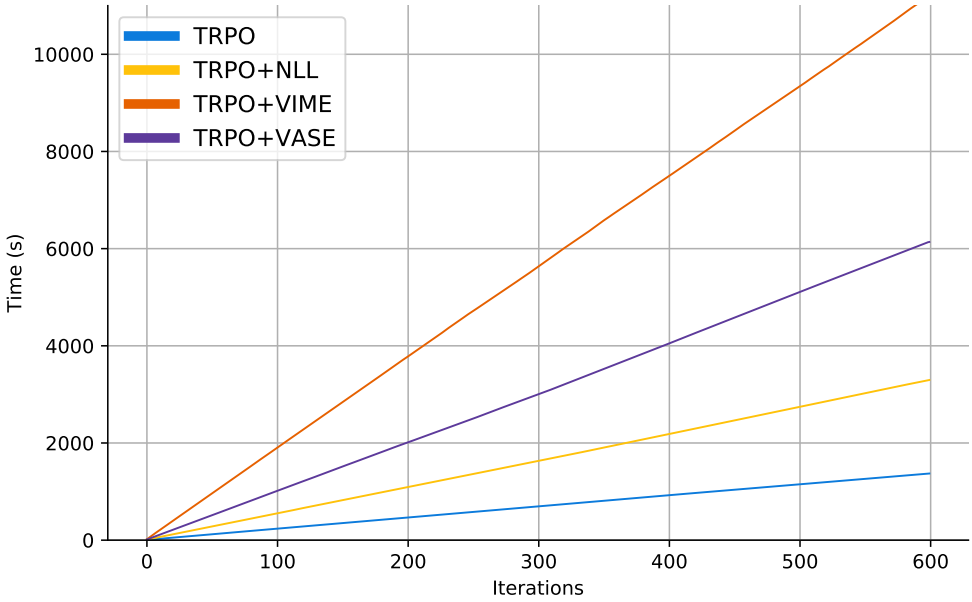


(c) DoublePendulum

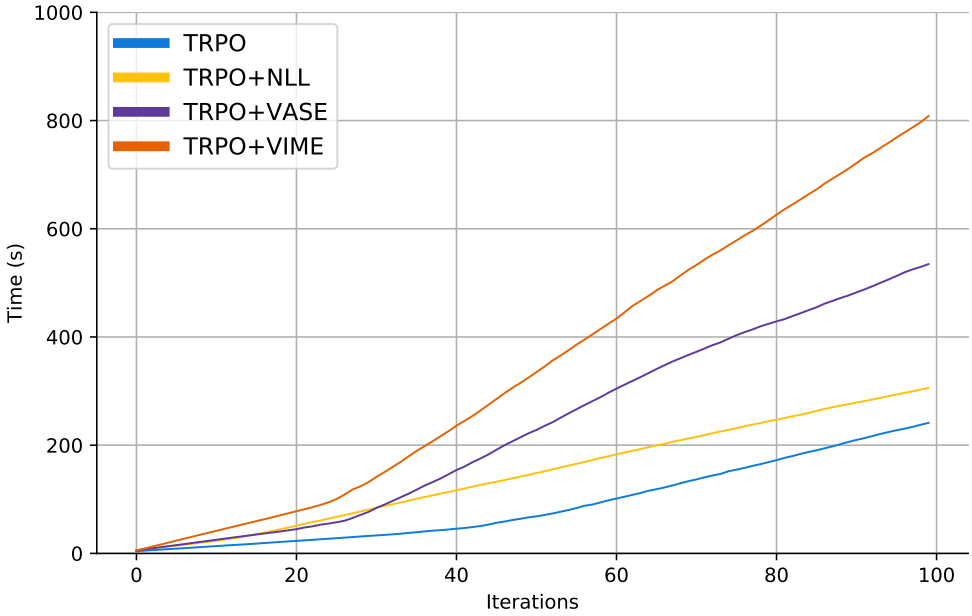


(d) HalfCheetah

Figure 3.7: Running time comparison on six environments. VIME, NLL and VASE use Bayesian surprise, surprisal, our surprise respectively (continued on next page).



(e) Ant



(f) Lunar-lander

Figure 3.7: Running time comparison on six environments. VIME, NLL and VASE use Bayesian surprise, surprisal, our surprise respectively (continued from previous page).

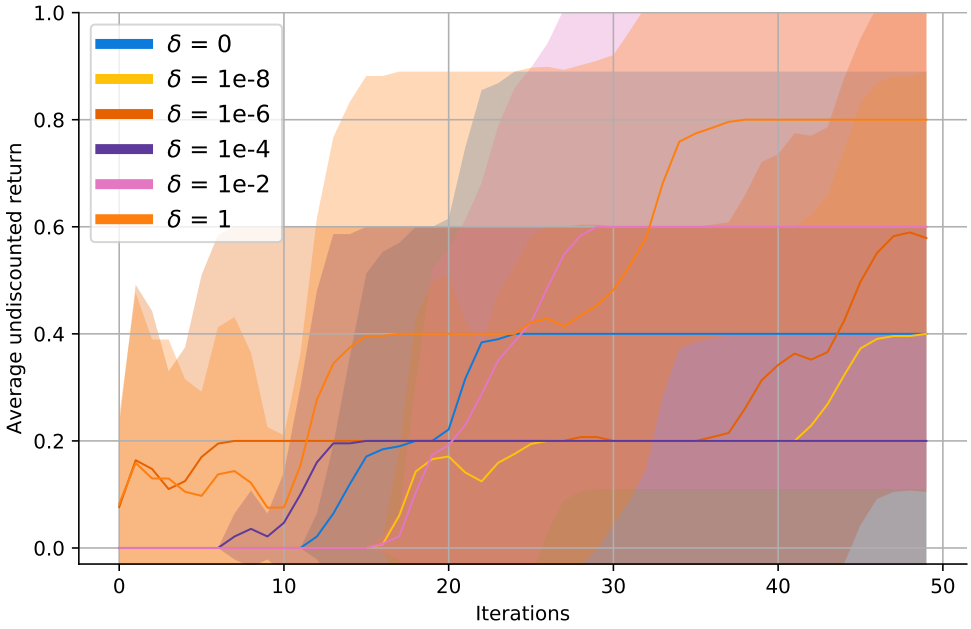
is done 5 times. We can see that different δ will lead to different average returns. It can also be seen from Figure 3.8 that for the classic continuous control environments CartpoleSwingup and DoublePendulum, the best average return is obtained when $\delta = 10^{-8}$, while for the locomotion tasks HalfCheetah and Ant, $\delta = 10^{-2}$.

3.4 Discussion: uncertainty in VASE

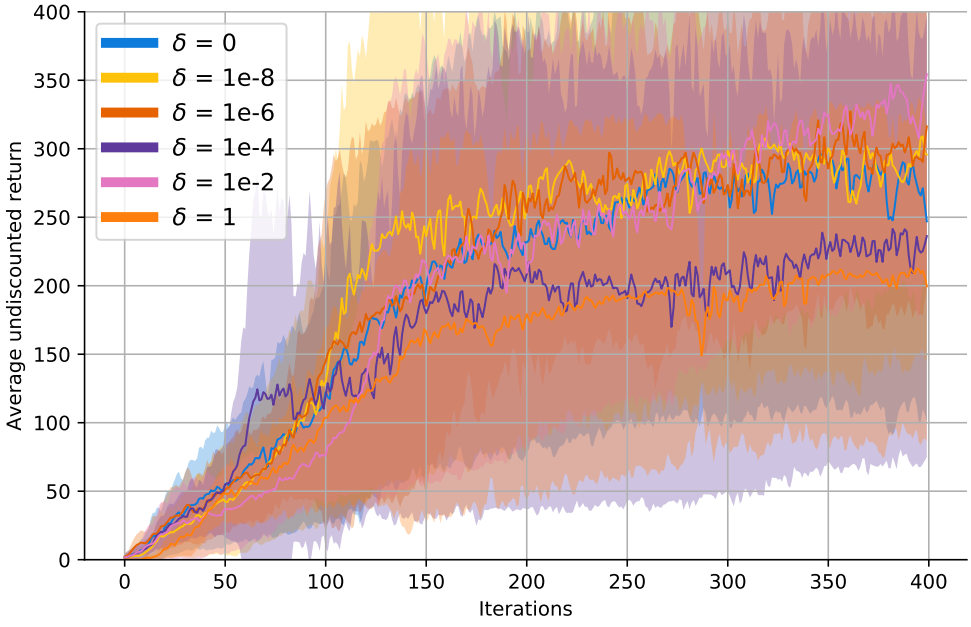
We proposed our VASE algorithm with the help of Bayesian neural network technique. Comparing Figure 3.2 and Figure 3.1, we can see the difference of the algorithm flow is that we choose the BNN to replace the conventional deep neural network. This is because we need a BNN to maintain the distribution $\{P(\theta)\}$. BNN is currently the latest technology used to estimate predictive uncertainty (Lakshminarayanan, Pritzel, and Blundell, 2017). In this section, we will discuss how uncertainty can help for exploration and how BNNs quantify predictive uncertainty.

There exists two sources of uncertainties in reinforcement learning; *aleatoric* uncertainty and *epistemic* uncertainty (Knight, 2012). In model based reinforcement learning, these uncertainties can affect the prediction of the model (Depeweg, Hernández-Lobato, Doshi-Velez, and Udluft, 2017; Chua, Calandra, McAllister, and Levine, 2018; Henaff, LeCun, and Canziani, 2019). Aleatoric uncertainty, or risk, captures the inherent randomness of the observations in environments. This random noise cannot be reduced even if more data were to be collected. Epistemic uncertainty, is the uncertainty caused by imperfect understanding of the environment(Der Kiureghian and Ditlevsen, 2009), which can be decreased when the model sees more data. Both types of uncertainties play an important role in reinforcement learning exploration problems. For example, epistemic uncertainty can help agents explore the environment more effectively because actions are chosen to explore unknown states. Nikolov, Kirschner, Berkenkamp, and Krause (2018) proposed an exploration strategy including both epistemic uncertainty and aleatoric uncertainty. They pointed out that aleatoric uncertainty should also be considered to capture the heteroscedastic[2](Kennedy, 2003) observation noise. This heteroscedastic observation noise is omnipresent in reinforcement learning, because of the interactions between the agent and the environment, no matter

[2]Consider the regression equation $y_i = x_i\beta + \epsilon_i, i = 1, \ldots, N$, $y_i = x_i\beta + \epsilon_i, i = 1, \ldots, N$, where the dependent random variable y_i equals the deterministic variable x_i times coefficient β plus a random disturbance term ϵ_i that has mean zero. The disturbances are homoskedastic if the variance of ϵ_i is a constant σ^2; otherwise, they are heteroskedastic.

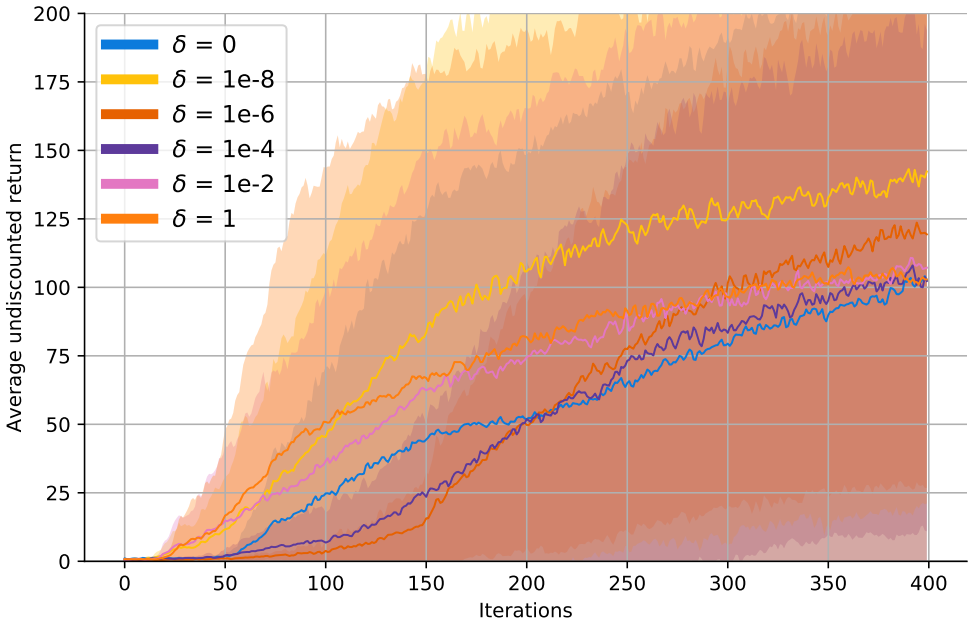


(a) MountainCar

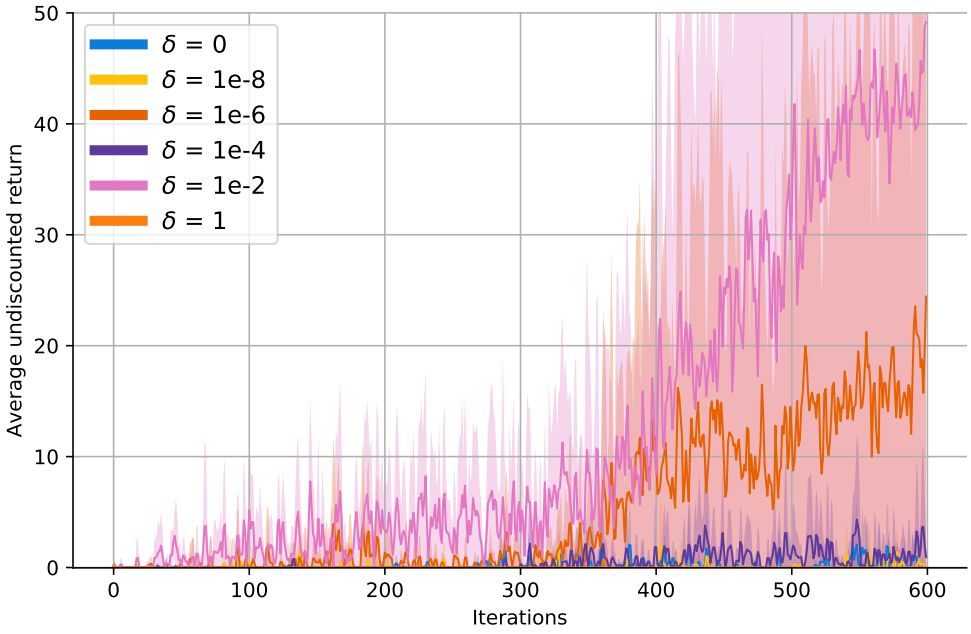


(b) CartpoleSwingup

Figure 3.8: Median performance for five sparse reward tasks with different δ chosen from $\{0, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1\}$, with interquartile ranges shown in shaded areas (continued on next page).

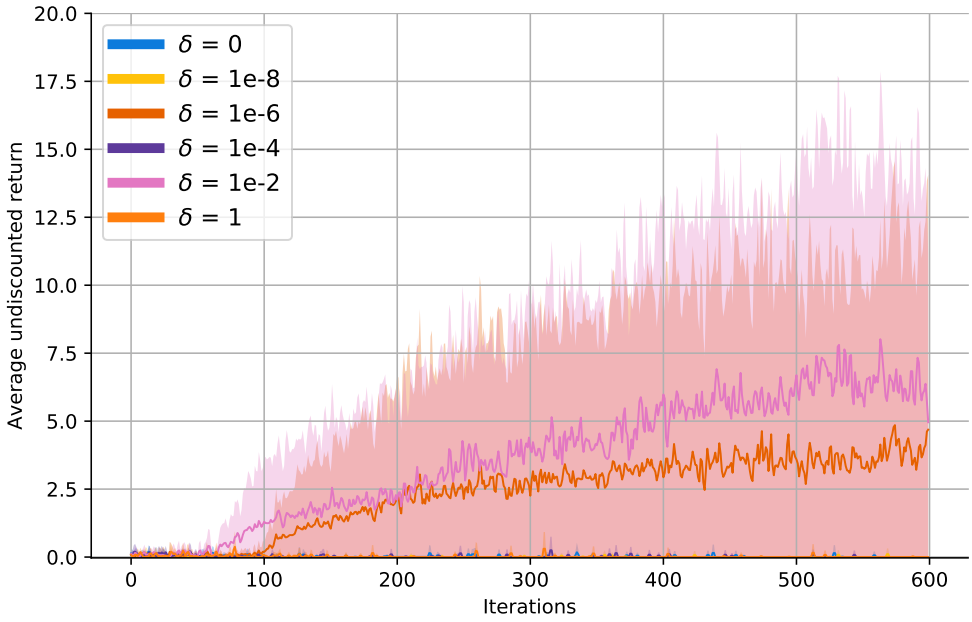


(c) DoublePendulum



(d) HalfCheetah

Figure 3.8: Median performance for five sparse reward tasks with different δ chosen from $\{0, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1\}$, with interquartile ranges shown in shaded areas (continued on next page).



(e) Ant

Figure 3.8: Median performance for five sparse reward tasks with different δ chosen from $\{0, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1\}$, with interquartile ranges shown in shaded areas (continued from previous page).

whether the environment is deterministic or stochastic, discrete or continuous. When designing risk-aware policies, aleatoric uncertainty is more informative than epistemic uncertainty (Clements, Robaglia, Van Delft, Slaoui, and Toth, 2019).

Recently, Gal (2016); Kendall and Gal (2017); Kwon, Won, Kim, and Paik (2018); Depeweg, Hernandez-Lobato, Doshi-Velez, and Udluft (2018) showed that training a BNN and estimating the predictive uncertainties can improve the quality of model prediction. Since we also chose BNNs as the prediction model in our VASE algorithm, we will show in this section that VASE quantifies both aleatoric and epistemic uncertainty, so that can drive the agent to explore better.

First, we can derive the variational predictive distribution of the BNN used in VASE algorithm as:

$$q_\phi(s_{t+1}|s_t, a_t) = \int_\Theta P(s_{t+1}|s_t, a_t, \theta) q(\theta; \phi) d\theta, \tag{3.13}$$

here $q(\theta; \phi)$ is the variational posterior distribution which is used to approximate the intractable posterior distribution.

Then the variational predictive uncertainty is quantified by the variance of the variational predictive distribution. We show in the following lemma that the variational

predictive uncertainty includes both aleatoric uncertainty and epistemic uncertainty:

Lemma 2 (The variational predictive uncertainty of Bayesian neural network includes both aleatoric uncertainty and epistemic uncertainty)**.**

$$
\begin{aligned}
&Var_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}] \\
&= \underbrace{\int_{\mathbf{\Theta}} Var_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}]q(\theta;\phi)d\theta}_{aleatoric} \\
&+ \underbrace{\int_{\mathbf{\Theta}} \{\mathbb{E}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}] - \mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2} q(\theta;\phi)d\theta}_{epistemic},
\end{aligned} \tag{3.14}
$$

Proof.

$$
\begin{aligned}
& \mathrm{Var}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}] \\
= \;& \mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}^{\otimes 2}] - \{\mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2} \\
= \;& \int_{\mathcal{S}} \widehat{s_{t+1}}^{\otimes 2} q_\phi(s_{t+1}|s_t,a_t)ds_{t+1} - \{\mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2} \\
= \;& \int_{\mathbf{\Theta}}\int_{\mathcal{S}} \widehat{s_{t+1}}^{\otimes 2} P(s_{t+1}|s_t,a_t,\theta)ds_{t+1}q(\theta;\phi)d\theta - \{\mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2} \\
= \;& \int_{\mathbf{\Theta}} \mathbb{E}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}^{\otimes 2}]q(\theta;\phi)d\theta - \{\mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2} \\
= \;& \int_{\mathbf{\Theta}} \{\mathrm{Var}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}] + \{\mathbb{E}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}]\}^{\otimes 2}\}q(\theta;\phi)d\theta \\
& - \int_{\mathbf{\Theta}} \{\{\mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2}\}q(\theta;\phi)d\theta \\
= \;& \int_{\mathbf{\Theta}} \mathrm{Var}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}]q(\theta;\phi)d\theta \\
& + \int_{\mathbf{\Theta}} \{\{\mathbb{E}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}]\}^{\otimes 2} - \{\mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2}\}q(\theta;\phi)d\theta \\
= \;& \underbrace{\int_{\mathbf{\Theta}} \mathrm{Var}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}]q(\theta;\phi)d\theta}_{aleatoric} \\
& + \underbrace{\int_{\mathbf{\Theta}} \{\mathbb{E}_{P(s_{t+1}|s_t,a_t,\theta)}[\widehat{s_{t+1}}] - \mathbb{E}_{q_\phi(s_{t+1}|s_t,a_t)}[\widehat{s_{t+1}}]\}^{\otimes 2} q(\theta;\phi)d\theta}_{epistemic},
\end{aligned}
$$

□

where "Var" denotes variance, $\otimes 2$ denotes vector product: $s^{\otimes 2} = ss^T$, and the last line of the proof can be derived by Eq. 3.13. The first term of last line in the proof is

the aleatoric uncertainty, it captures inherent randomness of an output $\widehat{s_{t+1}}$. We first have the variance of output $\widehat{s_{t+1}}$, based on the predictive distribution $P(s_{t+1}|s_t, a_t, \theta)$. The variance is multiplied with the variational posterior distribution $q(\theta; \phi)$ and then integrated over the parameter θ in the parameter space $\mathbf{\Theta}$. This gives us the average value of the output variability from the data set. Therefore, it can be regarded as the uncertainty evolved from the variability of the data set. The second term is epistemic uncertainty. This uncertainty comes from the variability of θ given data and can be decreased when the size of training data increases.

From the proof of Lemma 2 we showed that the output of the BNN in our VASE algorithm includes both aleatoric and epistemic uncertainty. It is interesting that if we look back to Eq. 3.5, the theorem that we showed the assorted surprise is the sum of the Bayesian surprise U_{Bayes} and the surprisal U_{NLL}. Obviously here the surprisal U_{NLL} can capture aleatoric uncertainty and the Bayesian surprise U_{Bayes} accounts for epistemic uncertainty, which means the assorted surprise that we proposed in this chapter also includes both types of uncertainty simultaneously. As discussed in this section, this will help the agent explore more efficiently in reinforcement learning environments. However, we need to point out that this assertion is not always true. For example, for environments that have an area with sharp discontinuous transitions (too noisy), the aleatoric uncertainty can even damage the exploration. This leads to the question: How to disentangle the aleatoric and epistemic uncertainty in BNN prediction or in assorted surprise, so that the exploration algorithm can adapt to different environments. We leave this as future work. We will show how to explore in the environment that has sharp discontinuous transition boundary in Chapter 5.

3.5 Conclusions

In this chapter, we chose a new form of surprise as the agent's intrinsic motivation and applied it to the RL settings by our VASE algorithm. VASE tries to approximate this surprise in a tractable way and train the agent to maximise its reward function. The agent is driven by this intrinsic reward, which can effectively explore the environment and find sparse extrinsic rewards given by the environment. We also discussed the exploration ability of VASE algorithm, based on the viewpoint of qualifying the aleatoric and epistemic uncertainty. Empirical results show that VASE performs well across various continuous control tasks with sparse rewards. We believe that VASE can be easily extended to deep reinforcement learning methods or learn directly from

pixel features. We leave that to next chapter to explore.

Chapter 4

VASE for Large Scale Problems: Playing Atari Video Games

Most reinforcement learning algorithms playing Atari games are trained by maximising extrinsic rewards, which are the game scores provided by the learning environment. However, when people play games, we are driven by intrinsic motivation, or to explore new game scenarios that surprise us. A good example is Minecraft game, we play but no extrinsic reward from the game is required. In this chapter, we applied our variational assorted surprise exploration (VASE) method that we proposed in the previous chapter to large scale problems: we let the agent play Atari video games driven by VASE. The extrinsic reward from game environments is *removed* so the agent playing Atari games is *only* trained by VASE. To investigate VASE performance, we choose three Atari games as training environments and also compare the VASE performance with *surprisal*. Although VASE algorithm was originally designed specifically for continuous space environments, our experimental results show surprisingly good performance on these Atari video games with discrete action spaces. Furthermore, from an exploration point of view, VASE driven agent can explore and find some solutions in games that surprisal driven agent cannot. For example, in Breakout game, VASE agent can even find a bug in the game so that it will never die.

4.1 Introduction

The recent advances in reinforcement learning (RL) have been made possible by the rise of deep learning. This has led to a resurgence of research in deep reinforcement learning (DRL). A well-known success story is that the DRL algorithms can play Atari

video games and play better than humans.

Atari video games, or Atari 2600 games were developed and manufactured by Atari, Inc. In 1977, Atari released the most famous video game machine: Video Computer System (VCS), later known as Atari 2600. In 2013, Bellemare *et al.* (2013) proposed a simple framework called Arcade Learning Environment (ALE). ALE allows researchers to design DRL algorithms to play Atari 2600 games. Currently, it supports more than 50 Atari games. The Atari 2600 in ALE has a high-dimensional visual input, which is 210×160 RGB video at 60 Hz.

When playing Atari games, the DRL algorithm usually uses the convolutional neural network (CNN) as its control policy. The policy CNN uses the original video data as its input, and various algorithms (such as DQN or policy gradient) to train the network. Policy networks only learn from video input, rewards and terminal signals, without any other game information.

Most of the successful DRL algorithms are trained by the "score" provided by the game. The agents trained by the RL algorithm performed well, even surpassing human performance. However, there is different learning behaviour between human and these trained agents: People explore the world based on their internal world model built from their previous learned experience. They are interested in the area that they have not explored before so that they are surprised by the difference between the new event and their knowledge. But the agents are trained by the rewards provided by the game environment. In addition, the reward function to train the agent is difficult to design and also not scalable, different game environments need to design different reward functions. These have prompted scholars to explore whether it is possible to train agents directly with intrinsic reward in reinforcement learning to play Atari 2600 games.

In 2018, Burda *et al.* (2018) proposed their surprisal driven exploration method based on the idea that these intrinsic surprisal rewards will bridge the gaps between sparse extrinsic rewards so that the agent can efficiently explore the game environment to find the next extrinsic reward (game score). They also showed their amazing result of training the agent only by the surprisal intrinsic reward, removing the extrinsic reward provided by the game.

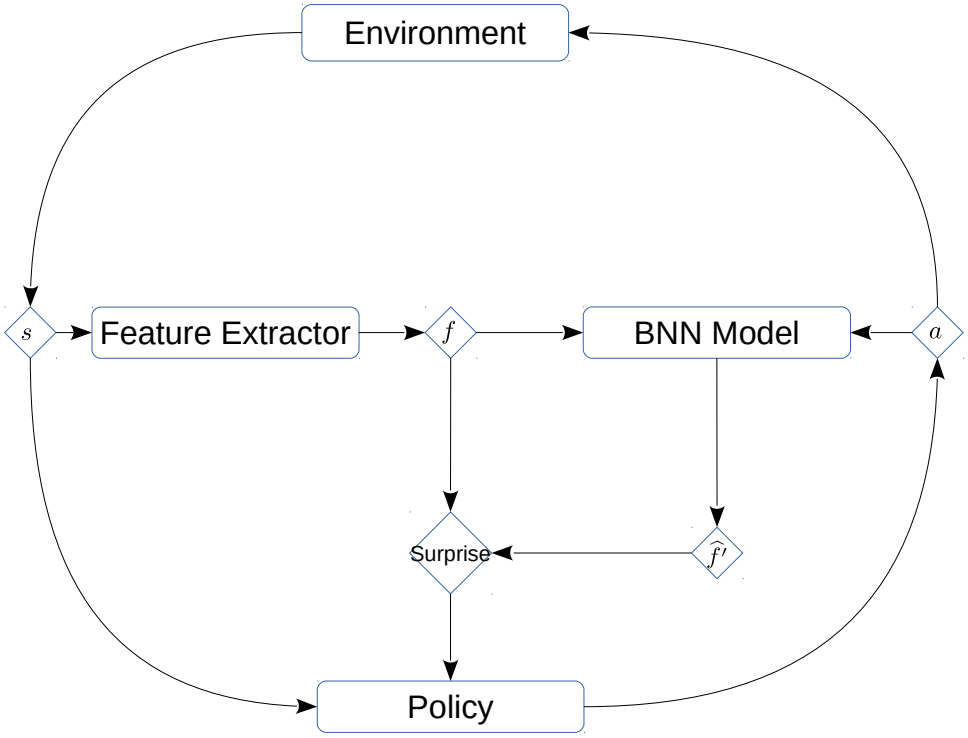


Figure 4.1: VASE-driven RL with s for state, a for action, f for the state feature that is extracted from the feature extractor network, and $\widehat{f}'$ for the BNN model prediction of the next state feature f. Note that the extrinsic reward r_e is removed.

4.2 Assorted Surprise for Atari games

In this chapter, we tried to apply our VASE algorithm to drive the agent to play Atari 2600 games. Similarly to Burda *et al.* (2018), we also remove the extrinsic reward that the game environment provides. The agent is only trained and driven by VASE, that is, it only learns to play the game by its intrinsic reward (See Figure 4.1). As introduced in the previous chapter, the assorted surprise is the ensemble inconsistency between the model prediction and observed environment outcome. To compute the assorted surprise reward, we need to build a BNN for the agent to predict future states.

The video frame is a high-dimensional pixel space input, and we need to add a feature extractor before the BNN model. In this way, we can reduce the size of the input space and avoid having the BNN model predict a huge pixel space every time. Pathak *et al.* (2017) has also shown that if the world model M is learned from an embedded space it can perform better than from the original pixel space. Many methods have been used (Raytchev, Yoda, and Sakaue, 2004; Kim, Jung, and Kim, 2002; Tenenbaum, De Silva, and Langford, 2000; Lima, Zen, Nankaku, Miyajima, Tokuda, and Kitamura,

2004) for dimension reduction. However, as data sizes increase, more effective dimensionality reduction methods are needed. Random Projections (RP) (Dasgupta, 2000) have been shown (Johnson and Lindenstrauss, 1984) as efficient and data-independent methods for linear dimensionality reduction. Giryes, Sapiro, and Bronstein (2016) extended RP to nonlinear case. They showed that deep neural networks (DNNs) with random Gaussian weights and ReLU activation function can preserve the metric structure of the data. Theory in Giryes *et al.* (2016) asserts deep neural networks with random gaussian weights can preserve the metric structure of the data, which guarantees that BNN (fully connected) can perform a stable embedding of the data. However, for convolutional layers with fixed random weights as a nonlinear random projector, although Burda *et al.* (2018) showed empirically that random feature convolutional neural networks (CNNs) perform very well across different Atari video games, there is currently no theoretical guarantee.

We first choose random feature convolutional network (CNN) to map the game frames to lower dimensional vector, followed by a fully connected Bayesian neural network (BNN). The weights of the CNN are fixed and sampled from Gaussian distribution $N(0, 1)$. Random feature CNN does RP behaviour so that we only need to train the weights of BNN layers, which can make the learning more efficient.

4.2.1 Compute assorted surprise

Through VASE algorithm we showed in Chapter 3, the BNN can be seen as a distribution of models, where a sample of network parameters θ according to distribution $P(\theta)$ is analogous to generating a single prediction of the next state. The prior distribution $P(\theta)$ changes to posterior when BNN is trained. The difference is that in this chapter, we add a CNN before the BNN model so that we can extract the feature from original pixel frames of game videos. Because the CNN network is fixed, the features are stable.

Denote $f(s_{t+1})$ as the feature representation of game video frame s_{t+1}. It is the final output of CNN layers. Then the variational assorted surprise we proposed in Chapter 3 changes to be computed in feature space:

$$\mathbb{E}_{\theta \sim q(\cdot|\phi)}[-\log P(f(s_{t+1})|f(s_t), a_t, \theta)]. \tag{4.1}$$

Note that the confidence term $H(q(\theta; \phi))$ in Eq. 3.6 is removed because it is hard to do a best hyperparameter search for its coefficient δ in large scale problems and we found in experiments that even when δ is set to zero, we can still get better results than surprisal.

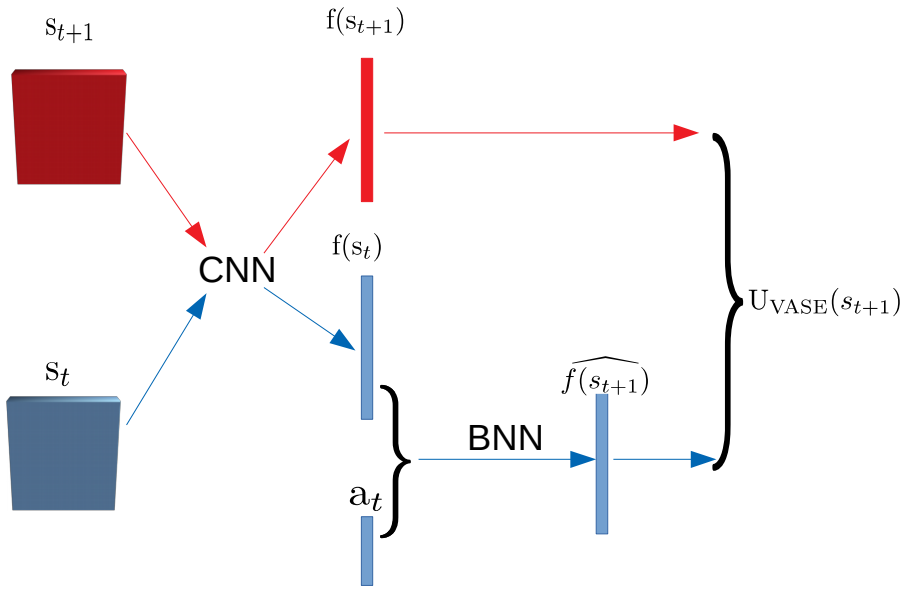


Figure 4.2: A block diagram shows how to compute assorted surprise.

The output of the BNN gives the prediction of the next state $\widehat{f(s_{t+1})}$, then $P(f(s_{t+1})|s_t, a_t, \theta)$ can be computed by:

$$P(f(s_{t+1})|s_t, a_t, \theta) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-||\widehat{f(s_{t+1})} - f(s_{t+1})||^2/(2\sigma_c^2)}, \tag{4.2}$$

under the assumption that states are normally distributed and σ_c is an arbitrarily chosen constant. (See Figure 4.2.)

4.3 Experiments and results analysis

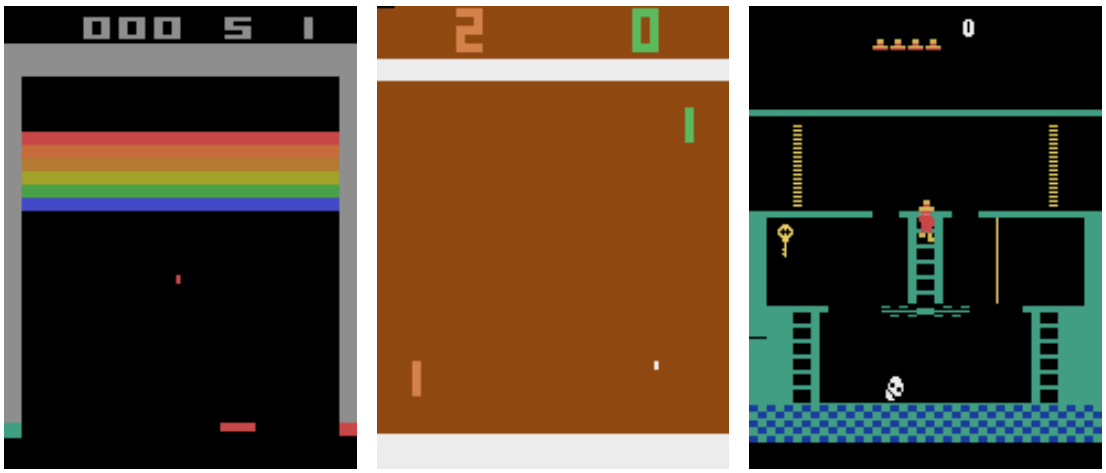
To investigate the agent behaviour driven by our variational assorted surprise, we chose three Atari video games, Breakout, Pong and Montezuma's Revenge as experimental environments. We also compared behaviour of the agent that is driven by Surprisal (Burda *et al.*, 2018). We didn't compare with VIME (Houthooft *et al.*, 2016) because VIME is time-consuming and not suitable for large scale problems (We will explain the details in section 4.3.3).

Breakout was first released by Atari, Inc. on May 13, 1976. The ported version of the Atari 2600 was programmed by Brad Stewart and published in 1978. In Breakout game, bricks are on the top of the screen and the game goal is to strike them and make them disappear. A ball is to move around the screen, and also to bounce off the top and two sides of the screen. The player has a movable paddle to bounce the ball. It

loses a life when the ball has missed the paddle and touches the bottom of the screen. There are six rows of bricks, and the player is given five turns to destroy the brick wall (Figure 4.3a).

Pong was released by Atari, Inc. in 1972. It is not only one of the earliest Atari video games, but also a commercially successful video game. In 1977, Atari published Pong game in its Atari 2600 version. Pong is a game that simulates table tennis. The player controls a paddle, moves it vertically so that they can compete against the other player who controls the opposing side paddle. The paddles are used to hit a ball. The ball then moves back and forth across the screen. The goal of the Pong game for each player is to achieve 21 points before the other player. The awarded points are gained when the other one fails to hit and return the ball (Figure 4.3b).

Montezuma's revenge was first released in 1984 on Atari 2600 platform, and also has its IOS and Android version published in 2012. The player controls a character called Panama Joe (a.k.a. Pedro), moving him from room to room, scoring points along the way (Figure 4.3c). Panama Joe can jump, run, slide down poles, and climb chains and ladders in the game. He has to face enemies like skulls, snakes and spiders, or some other traps and dangers like laser gates, fire pits and suddenly missing floors. The game goal is to gain game scores while collecting jewels or killing enemies.



(a) Breakout (b) Pong (c) Montezuma Revenge

Figure 4.3: Three Atari video games.

For Breakout, the episode length and the game score are highly correlated, because the agent has to play the game longer to gain high game scores. For Pong, however, the game score has nothing to do with the episode length. The agent can play long enough but still has a negative reward. But the agent has to play carefully to make sure not

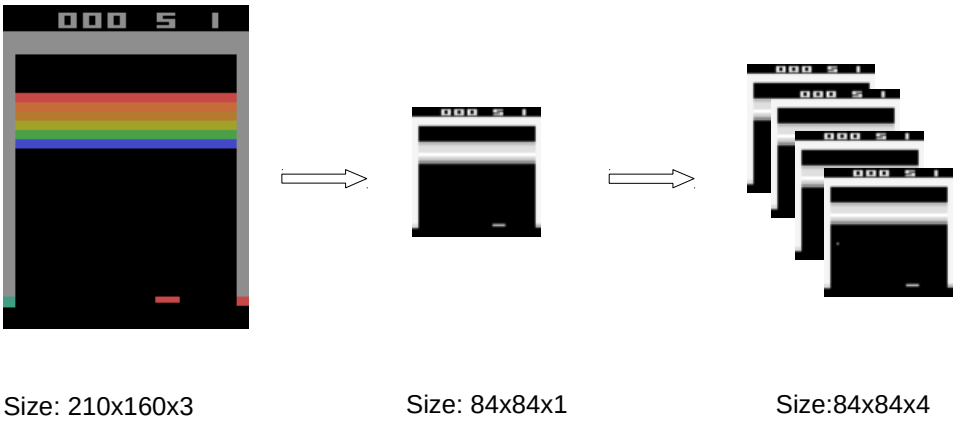


Figure 4.4: Data preparation: RGB image (size: 210x160x3) is resized to grey image (size: 84x84x1), and then stacked 4 historical observations to the current observation (size: 84x84x4).

to let the opposite player get 21 points. The Montezuma revenge game is similar to Pong. It has many rooms for the agent to explore and the agent can find many ways to play the game forever as long as it is still alive in the game. Since the policy network is trying to maximise the sum of surprise, when the surprise of different states is almost the same after training for a while, the policy will tend to maximise the episode length. So when compared to surprisal driven method, we not only compared the game score that the agent can collect from the game, we also compared the episode length to see which agent can live longer in the game.

4.3.1 Data preparation

In Chapter 3, the inputs of the policy and model network are the original environment state s. But because Atari game video frames are high dimensional pixel images, we need to make some preparation. First, we convert the colors of all the pixel images s_t to grayscale images x_t, cropped and downsampled their size to 84x84 (Figure 4.4). Then we chose 4 historical frames $[x_{t-3}, x_{t-2}, x_{t-1}, x_t]$ together and stacked them as the agent's stacked observation. The purpose of keeping and stacking 4 frames is to ensure that we can maintain the necessary information. For example, the direction, velocity and acceleration of the moving objects in the game environments. This can be done easily through the OpenAI baseline wrapper (Dhariwal, Hesse, Klimov, Nichol, Plappert, Radford, Schulman, Sidor, Wu, and Zhokhov, 2017).

Table 4.1: Policy network architecture.

Layer		Channel	Size	Kernel size	Stride	Activation
Input	Image	4	84x84			
1	CONV	32	20x20	8x8	(4,4)	LEAKY RELU
2	CONV	64	9x9	4x4	(2,2)	LEAKY RELU
3	CONV	64	7x7	3x3	(1,1)	LEAKY RELU
4	FC		512			NONE
5	FC		512			RELU
6	FC		512			RELU
Output	FC		No. of actions			NONE

4.3.2 Network architecture

Policy network architecture

The input of the policy network are the stacked frames, which are preprocessed by data preparation wrapper, and then three convolutional layers, each convolutional layer is followed by a leaky-relu activation function. Next are four fully connected linear layers. The number of policy network output equals to the number of valid actions which depend on the games (Table 4.1).

Feature extractor network architecture (Frozen weights)

The architecture of the feature extractor network is the same as the architecture of the convolutional layers and the first fully connected linear layer in the policy network. The difference is that after random initialization, the weights of the feature extractor network will be frozen during reinforcement learning (Table 4.2). The output layer has 512 units. It can be seen as the representation of the input observation frames in feature space.

BNN model architecture

The BNN model is located behind the feature extractor network and takes the output of the feature extractor network as its input. All layers in BNN model are fully connected layers. But as introduced in Chapter 2, the weights of BNN are probability distributions rather than a fixed value (Table 4.3).

Table 4.2: Feature Extractor architecture (Frozen weights).

Layer		Channel	Size	Kernel size	Stride	Activation
Input	Image	4	84x84			
1	CONV	32	20x20	8x8	(4,4)	LEAKY RELU
2	CONV	64	9x9	4x4	(2,2)	LEAKY RELU
3	CONV	64	7x7	3x3	(1,1)	LEAKY RELU
Output	FC		512			NONE

Table 4.3: BNN model architecture.

Layer		Size	Activation
Input	Output of FE	512	LEAKY RELU
1	FC	512	LEAKY RELU
2	FC	512	LEAKY RELU
3	FC	512	LEAKY RELU
Output	FC	512	NONE

4.3.3 Experimental results

All the games are played three times with different random seed. For all experiments in this chapter, PPO algorithm Schulman *et al.* (2017) (proximal policy optimisation) is selected. This is because PPO is better than TRPO at handling big data problems like playing Atari video games.

The end of the episode signal (the "death" signal), also used as the "done" signal in reinforcement learning, is removed to increase the game difficulty. That is to say, "death" is now just another normal environment transition to the agent. When the agent loses all its lives, it will just return to where the game started, but the RL algorithm will continue to train. As explained in Burda *et al.* (2018), the "death" signal could leak game information and the game become easy. If we don't remove it, we can simply design a policy that provides a positive reward at the step when the agent is alive and negative reward if the agent is dead. This policy can work and drive the agent to play well for many games.

The maximum episode length is set to 4500 frames. For all networks in our experiments, the learning rate is set as 0.0001. To stabilise the method performance, more

actors (128 parallel environments) are used to collect trajectories. In each environment, the agent takes the action, and then gets the returned s_{t+1}. We collected 128 triples (s_t, a_t, s_{t+1}) in each environment and then use these 128×128 triples to train the policy network and BNN at the same time for 3 epochs. Then the agent sample the trajectories again based on the updated policy. The number of total time steps is 1e8.

Since we mentioned the parallel game environments trick used in surprisal and VASE, here we explain why we didn't compare with VIME method in this chapter. To compute the Bayesian surprise in VIME methods, we have to compute Eq. 2.66 at each time step t. Each reward needs to calculate the gradient through VIME model, which needs to pass forward and backward. For each episode, the total extra time cost of calculating the gradient when calculating the VIME reward is

$$\text{Time}_{\text{VIME}}^{\text{back}} = Tt_b, \tag{4.3}$$

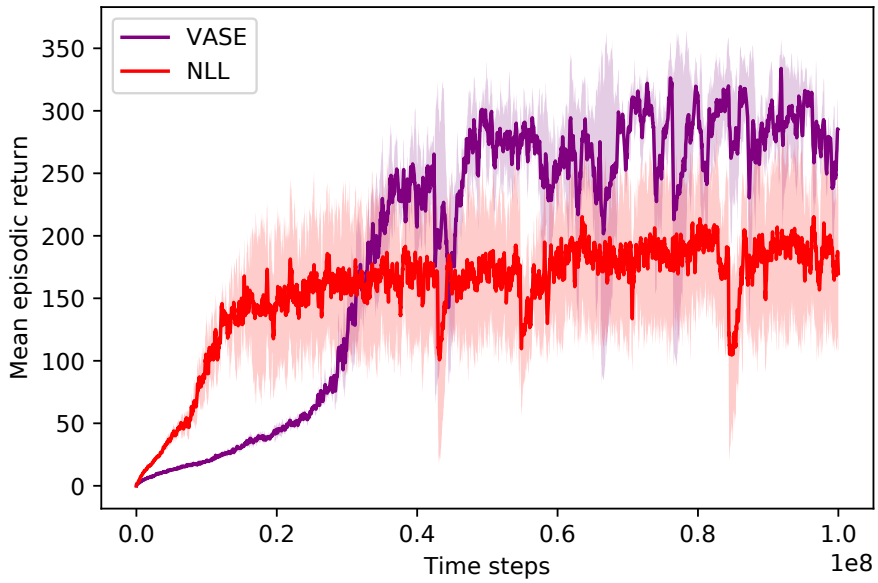
where T is the length of each episode, t_b is the time of each backward computation through the model. If we sample N episodes from N parallel environments, we have

$$\text{Time}_{\text{VIME}}^{\text{back}} = NTt_b. \tag{4.4}$$

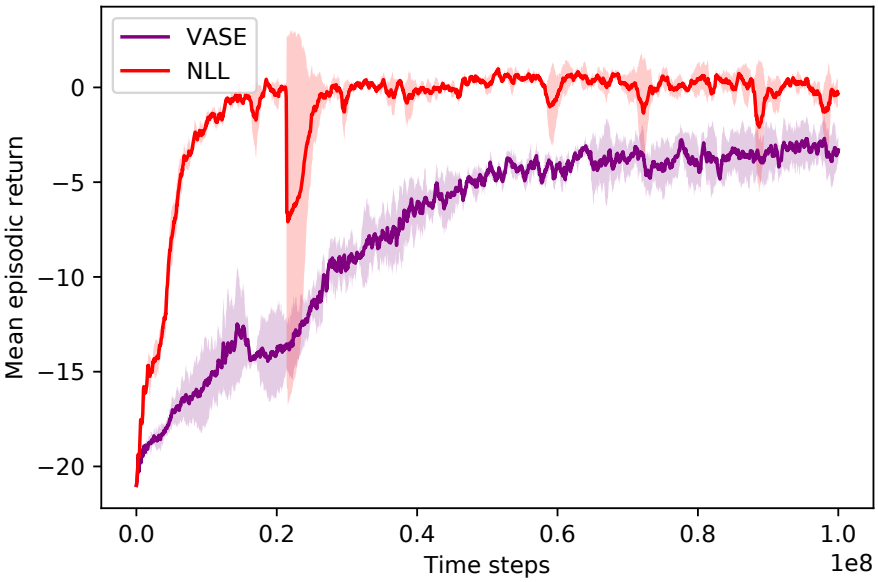
The larger N is, the longer the calculation takes, which means parallel environments cannot reduce the training time of VIME. We need to copy the BNN model M and use the copies to compute Bayesian surprise. That is what we have done in Chapter 3 when we use small BNN models for VIME algorithm. However, in this chapter, this is intractable since the Feature extractor network and Bayesian network model are large (millions of parameters).

The comparison result between VASE and surprisal method can be found in Figure 4.5 and Figure 4.6. Figure 4.5 shows mean extrinsic returns on Atari games with 3 different seeds, standard deviation shown in shaded areas. The agents were trained purely by intrinsic reward, without extrinsic reward or a "death" signal. Figure 4.6 compares the episode length.

We can see from Figure 4.5a that for Breakout, the agent trained by VASE can achieve a higher game score compared to surprisal (NLL). Figure 4.6a also shows that the episode length of VASE agent is also longer than NLL's. This is because the longer the agent is alive, the higher score it can get from the breakout game. In our practice, we found that the VASE agent can find some policies so that it will not die or survive forever in the game, and then it will stop controlling the ball to hit the bricks.

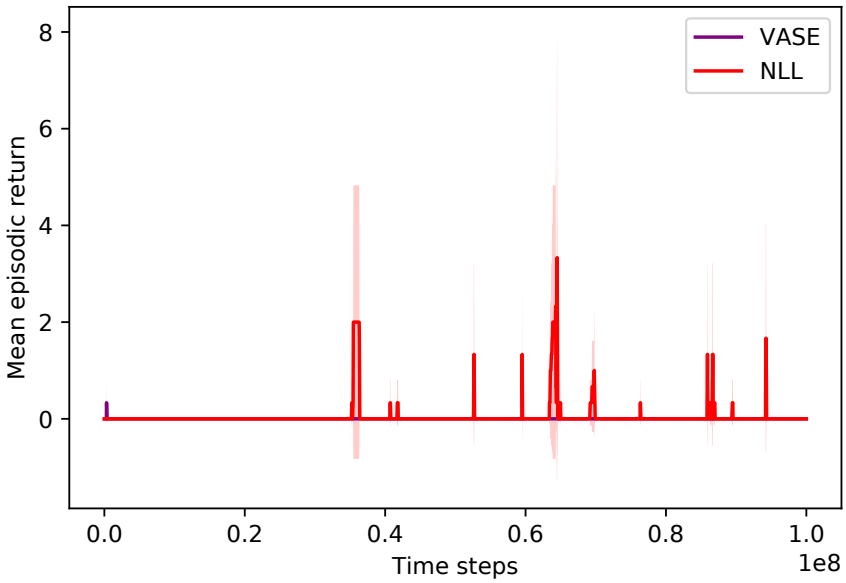


(a) Break out



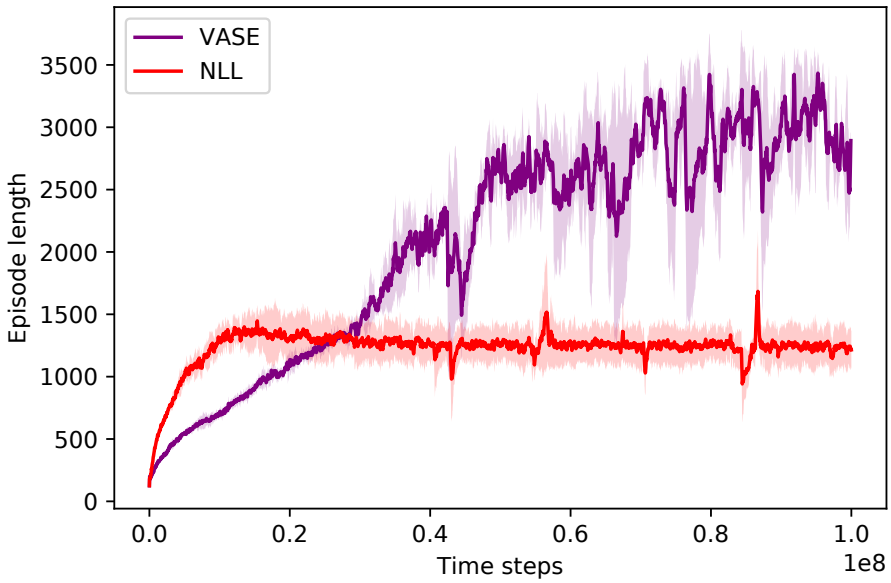
(b) Pong

Figure 4.5: Mean extrinsic returns on Atari games with 3 different seeds, standard deviation shown in shaded areas. The agents were trained purely by intrinsic reward, without extrinsic reward or a 'death' signal. NLL for surprisal method (continued on next page).

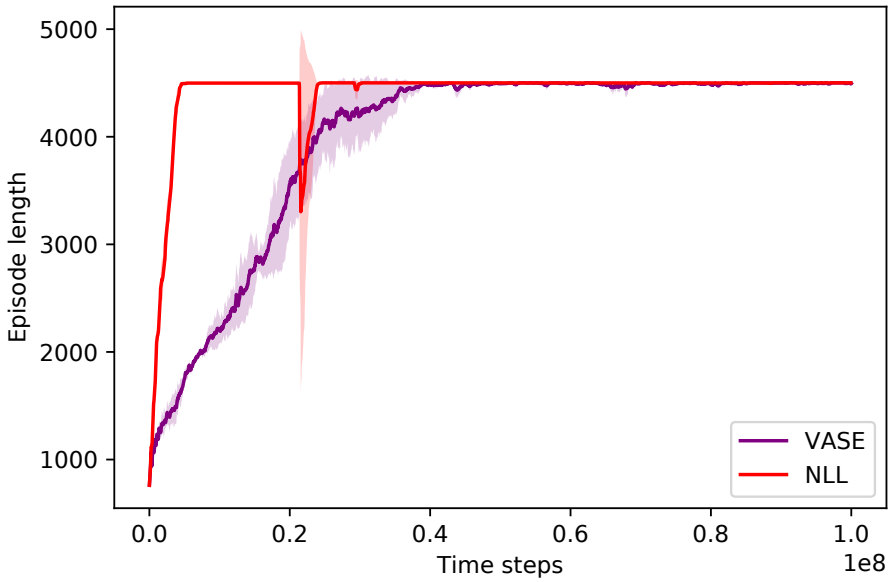


(c) Montezuma Revenge

Figure 4.5: Mean extrinsic returns on Atari games with 3 different seeds, standard deviation shown in shaded areas.The agents were trained purely by intrinsic reward, without extrinsic reward or a 'death' signal. NLL for surprisal method (continued from previous page).

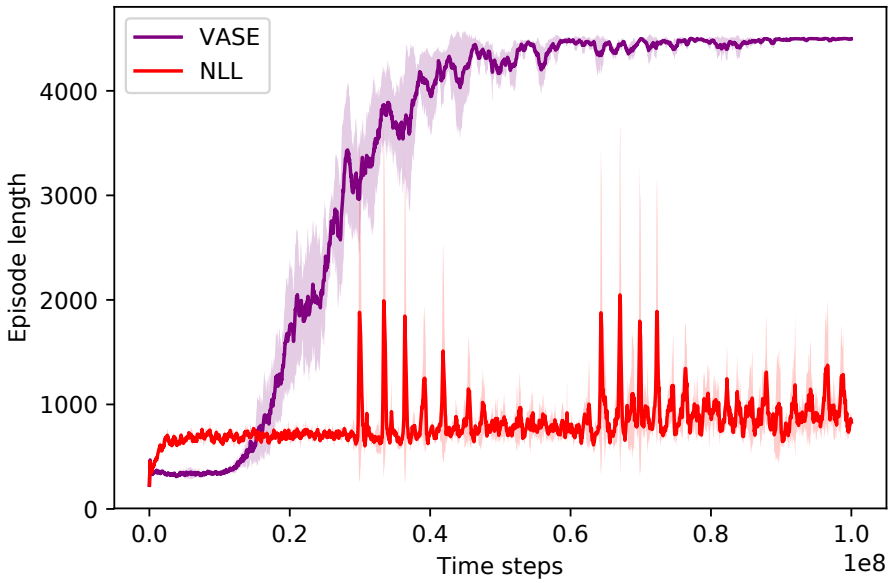


(a) Break out



(b) Pong

Figure 4.6: Mean episode lengths on Atari games with 3 different seeds, standard deviation shown in shaded areas. The agents were trained purely by intrinsic reward, without extrinsic reward or a 'death' signal. NLL for surprisal method (continued on next page).



(c) Montezuma Revenge

Figure 4.6: Mean episode lengths on Atari games with 3 different seeds, standard deviation shown in shaded areas. The agents were trained purely by intrinsic reward, without extrinsic reward or a 'death' signal. NLL for surprisal method (continued from previous page).

Sometimes, when the agent discovered that there was only one life left, it even chose to use bugs that will not let the ball roll out. (The ball disappeared)[1].

For Pong, the game score of VASE is lower than surprisal (Figure 4.5b). However, as we explained before, the agent will focus on how to live longer in the Pong game, so it doesn't matter if the game score is higher or not. Episodes of VASE and surprisal both reach the limit of the maximum number of episode length 4500 (Figure 4.6b). If we check the simulation videos[2], we can see that the policy that the VASE agent chose can let it play the game forever, the episode length is 4500 only because we manually set it as 4500. For surprisal trained agent, the video shows that the game score is still changing. There is still a trend that the agent wins or loses the pong game after these 4500 steps.

In the end, the Montezuma's Revenge seems too hard for both VASE and surprisal methods. They both almost haven't gotten any game scores (Figure 4.5c). But VASE still achieves its maximum episode length number (Figure 4.6c). The videos[3] show that VASE agent decides to jump back and forth safely on the platform instead of jumping down. However, the surprisal driven agent doesn't find a safe strategy for its long-term survival. After 1e8 iterations of training, it still chooses to jump down the platform and lose its life, so its episode length is particularly short.

To conclude, the VASE agent survives in the game longer than the surprisal agent. If the extrinsic game rewards are taken into consideration, it seems that both VASE and surprisal agents are not playing well on Montezuma's revenge game when they are only trained by intrinsic rewards. In order to further study whether intrinsic rewards can help to play Montezuma's revenge game, we have added extrinsic rewards (the game score) to the objective function. Now the agent no longer plays a pure exploration problem. It is driven by intrinsic reward to explore the game and then collect extrinsic rewards, and game policies are now trained by the sum of intrinsic and extrinsic rewards.

From Figure 4.7 we can see that both VASE and surprisal agents perform better than when they are trained only by intrinsic reward. The surprisal agent gets a maximum of 400 rewards and VASE agent even much better (2500). The VASE agent has also found more than 7 rooms. However, we also observed that after about 2e7 training steps, the episode reward has not changed for a long time. When we check

[1] `https://drive.google.com/open?id=1QFe1IAoDd9PyXlHtT8B54QsFmC3w8lqI`

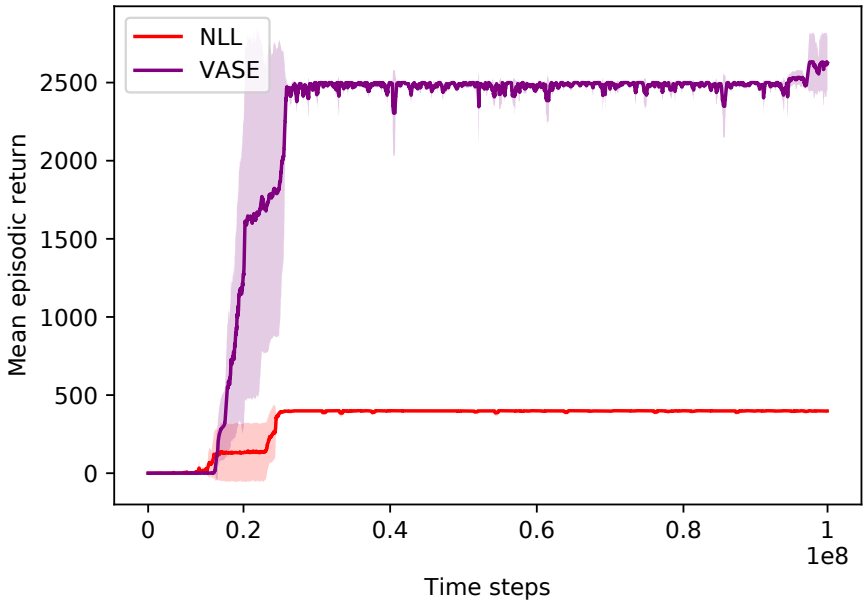
[2] `https://drive.google.com/open?id=1NcAyc4dWLH4PVaqOObQU_qUGwhXvejdD`

[3] `https://drive.google.com/open?id=1HTeioANVQZlrOQjtzwhoT9uWVRZ364-z`

the simulation videos,[4] we found that the agents were "stuck" in a specific location, thus stopping the exploration of the new room. This particular location is usually the boundary between rooms. For example, the surprisal agent will get stuck when walking from the first room to the next room (whether it goes to the room on the left or the room on the right). In this context, being stuck refers to the non-stop walking between these two rooms. In contrast, the VASE agent did not get stuck in the place where the surprisal agent got stuck. It happened relatively late. At this time, the VASE agent has explored several different rooms.

Now let's analyse why this stuck behaviour occurs. We know that the agent explores the game environment based on intrinsic rewards, and when we calculate the intrinsic rewards, whether it is VASE or surprisal, we are calculating the prediction error, which is the difference between the prediction of next observation and the true next observation of the game environment (this difference is calculated in the feature space). For Atari games, the agent's observation is the current game screen (pre-processed) or screen background. When the agent is exploring in the same room, the screen background does not change, but when the agent goes from one room to another, the screen background will be replaced by the background of the other room. This will make the surprise reward much larger than elsewhere. Therefore, this will drive the agent to linger here. To show what happens during the whole training process, we chose three episodes and visualise the surprise reward the agent got at each time step (Figure 4.8). These episodes are chosen from surprisal agent training results. From Figure 4.8 we can see that at the beginning episodes, the agent spent more than 2000 time steps to find the new room. Compared to other places, the surprise reward collected when the agent found the new room is very large. After training for a while, the surprise at the new room and the old boundary becomes smaller, but still large compared to other places (less than 2). When the agent has been trained for a long time, the boundary surprise has reduced to about 2, but the surprise reward at other places is also reduced (0.6-0.8), so the difference is still huge. From episode 3 we can see the agent move to the room bound very fast and then stuck there for almost all the episode. Of course, being stuck is not permanent, for example, the VASE agent appears to escape at 1e8 steps (See Figure 4.7). The agent has signs of escaping from where it is stuck. The surprisal agent should be able to escape from being stuck too, but it may take longer. But when the agent finds a new room, this stuck behaviour will appear again, which greatly reduces the efficiency of the agent's exploration.

[4] `https://drive.google.com/open?id=1K7GCVTw6RVDQ8_SAy24_HKRY-z0-uMew`



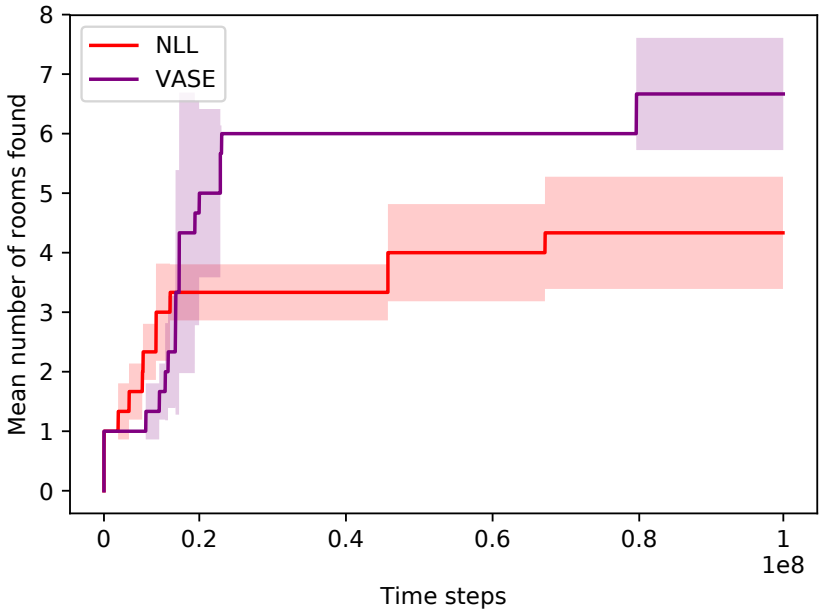


Figure 4.7: Montezuma's revenge episode return and number of rooms found across 3 seeds, agents are trained by intrinsic plus extrinsic rewards.

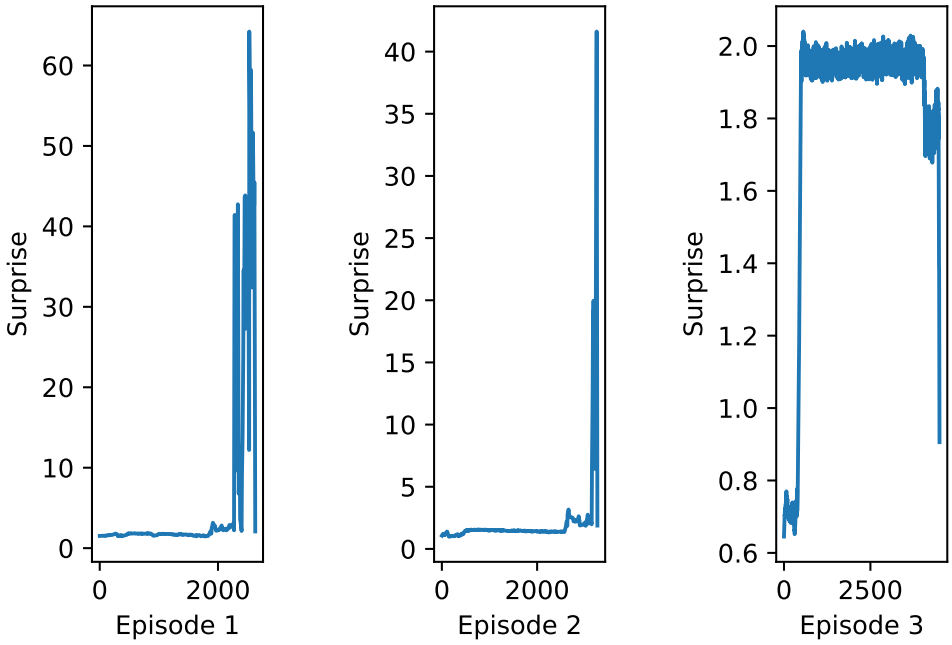


Figure 4.8: Three episodes show surprise rewards at each time step. Episode 1: the new room has just been found. Episode 2: the new room has been found for a while. Episode 3: the new room has been found for a long time.

4.4 Conclusions

In this chapter, we first showed that even only driven by assorted surprise reward, the agent can still play very well for some Atari games, especially for games whose scores are highly correlated with the game episode length, the agent driven by assorted surprise can find some solutions to make it play forever (and even find errors in the game). The agent does not need to know the reward or game score system. Our experimental results also show that both assorted surprise and surprisal driven agents are focusing on maximising the episode length rather than achieving the game score directly. The results of Montezuma's Revenge show that when the extrinsic reward is added, the agent can explore much better. However, it can be stuck at the boundary of rooms and so reduce the efficiency of exploration. In the next chapter, we will study how to find a way to deal with this kind of issue.

Chapter 5

MIME: Mutual Information Minimisation Exploration

Note: Some portions of this chapter are taken from my own work (Xu, McCane, Szymanski, and Atkinson, 2020).

5.1 Introduction

In Chapter 4, we used our VASE to train the agent playing Atari video games and compared it to a surprisal driven method. Both methods follow the same fundamental idea: A reward-maximising neural policy network π learns to generate action sequences; a separate neural network called the world model M learns to predict future states (s_{t+1}), given past inputs (s_t) and actions (a_t). In the absence of extrinsic reward, the policy network π maximises the same value function that the world model M minimises, that is, the intrinsic rewards. This adversarial learning mechanism motivates policy π to invent and generate experiments that lead to "novel" situations where the world model M cannot yet predict well.

However, since the world model M is trained to learn environment transitions, if some of those transitions are discontinuous or abrupt, it is difficult for the agent to predict s_{t+1} and therefore the agent is continuously surprised by the transition. This results in an agent that gets stuck on the transition boundary (See Figure 4.8). The length of time stuck on the boundary depends on the magnitude of change in the transition. The greater the transition change, the longer the time spent at the boundary.

Some current methods tend to avoid getting stuck at transition boundaries. Houthooft

et al. Houthooft *et al.* (2016) proposed VIME, which computes Bayesian-surprisal inspired by the idea of maximising information gain. But VIME is difficult to scale up to large-scale environments. Prediction improvement measures (Schmidhuber, 2006; Oudeyer, Kaplan, and Hafner, 2007; Lopes *et al.*, 2012; Achiam and Sastry, 2017) compute intrinsic reward by model learning progress and can avoid getting stuck in some situations where surprisal does. Even so, Achiam and Sastry (2017) show that prediction improvement measures do not explore as well as surprisal in ordinary sparse reward environments.

We propose Mutual Information Minimising Exploration (MIME) in this chapter. We show that MIME-agents can explore as well as surprisal-agents in sparse reward environments and much better in environments that include abrupt state transitions.

5.2 Background

In this chapter, we first make a brief review of surprisal-based method. We also draw a figure (Figure 5.1) to facilitate the intuitive understanding of the difference between our new method MIME and surprisal-based method.

5.2.1 Surprisal

Surprisal was already introduced in Chapter 2 (See Eq. 2.64). In practice, if we suppose the probability in surprisal is a Gaussian distribution, we can compute the intrinsic reward r_t^i in Eq. 2.61 with surprisal simply by:

$$r_t^i \sim \|M(s_t, a_t) - s_{t+1}\|^2. \tag{5.1}$$

In this chapter, in order to compare surprisal and MIME more intuitively, we use $M(s_t, a_t)$ to denote the prediction of the model M.

If we train an agent to explore the environment by surprisal, the world model M is trained with the environment transition $P(s_{t+1}|s_t, a_t)$. The world model then learns the mapping function from (s_t, a_t) to s_{t+1} and makes a prediction $M(s_t, a_t)$. When the environment has an area where the transition $P(s_{t+1}|s_t, a_t)$ from s_t to s_{t+1} is discontinuous, the agent cannot predict s_{t+1} well and gets a big surprise. The policy network π is trained by maximising the surprisal reward, so, at the next iteration, it will generate actions that drive the agent to this area again. Since piecewise continuous functions are hard to learn using continuous activation functions Selmic and Lewis

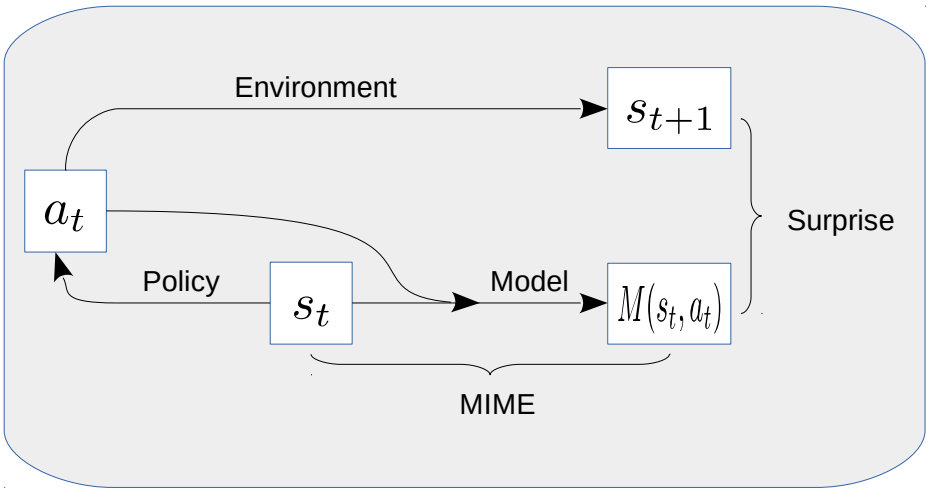


Figure 5.1: surprisal-driven V.S. MIME-driven, "Surprise" denotes the surprisal reward, "MIME" denotes the new intrinsic reward we proposed in this chapter.

(2002), the agent will be stuck at such transition boundaries for a long time. We have observed this stuck phenomena in Chapter 4.

Since the world model of VASE method we proposed in Chapter 3 is also trained with environment transition, the agent trained by VASE will also be stuck at these boundaries as we have seen in Chapter 4. When playing Montezuma's Revenge, both surprisal and VASE agents get stuck at room boundaries (See Figure 4.8), which reduces the exploration efficiency.

5.2.2 Random distillation network (RND)

Inspired by knowledge distillation (Hinton, Vinyals, and Dean, 2015), another exploration approach called random network distillation (RND) (Burda, Edwards, Storkey, and Klimov, 2018) was proposed in 2018 (See Figure 5.2). They chose a random initialised target network to generate the feature of each state, and a prediction network as a self-predictor to predict the target network's output. The difference between this prediction $\hat{f}_{t+1}$ and the target network's output f_{t+1} is regarded as the intrinsic reward to train the agent. Instead of predicting the next state given current state and current action as all surprise-based method does, the prediction network predicts the output of the target network for the next state. Here the mapped features f_{t+1} are produced by a CNN with frozen weights. Similar to the surprisal-driven idea, RND also focuses on future states. (Burda *et al.*, 2018) shows that the agent trained by RND does not get

stuck, it can explore much more efficiently than the agent trained by surprisal. So I will also compare MIME to RND in this chapter in environments where the transition is discontinuous.

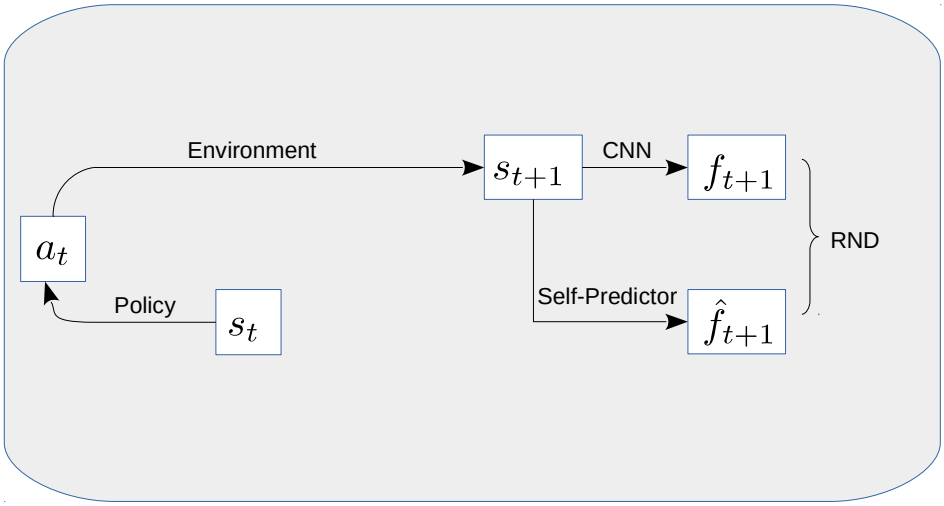


Figure 5.2: Random network distillation method.

5.2.3 Go-Explore

Go-Explore (Ecoffet, Huizinga, Lehman, Stanley, and Clune, 2019) is a new approach to solve hard exploration problems. Three principles are chosen in Go-Explore algorithms: (1) remember states (represented as a low-dimensional cell) that have previously been visited; (2) go back to a promising cell and explore from it; (3) solve simulated environments through exploiting any available methods. It achieved the-state-of-the-art results in Atari video games such as Montezuma's revenge game and Pitfall game. The author of Go-Explore hypothesised that the intrinsic reward driven exploration approach fails from two causes: detachment and derailment, which is interesting. However, it should be noted that the method designed in Go-Explore is only fit for deterministic environments like Atari video games and is difficult to be applied generally to other stochastic RL environments. Because it is almost impossible to return to a previous state in a stochastic environment. In addition, it will take a lot of computer resources (For example, 22 CPUs and 50 GB RAM in Go-Explore paper) to remember and use all previous visited states information. Go-Explore method also chose domain knowledge like the current room, current level, the position of the agent and the number of the keys held in Montezuma's revenge game to help the agent to explore. It also showed

that without domain knowledge, the agent even failed in Pitfall game. So RND is still state-of-the-art for playing Montezuma's revenge game if we only allow the RL agent to see pixels not the domain knowledge of internal environment information. There is also another controversy that the "returning to the previous state" strategy is just an exhaustive search, rather than reinforcement learning algorithm. Based on the above discussion, in this section, I will not compare MIME with Go-Explore.

5.3 MIME

In this chapter, we present a new exploration method named Mutual Information Minimisation Exploration (MIME). Similar to Bayesian surprise, MIME computes intrinsic reward by mutual information. However, instead of computing mutual information between past and future time steps in the trajectory, we compute mutual information between the input and output of the model M on the current state and current action. In other words, the world model simply tries to learn a representation of the world without prediction, but nevertheless incorporates information from the chosen action. Surprisingly, this works just as well as surprisal in common continuous transition environments, and much better in environments with abrupt transitions.

The mutual information (MI)-based approach has a long history in unsupervised feature learning and the infomax principle (Linsker, 1988; Bell and Sejnowski, 1995) for neural networks advocates maximizing the MI between input and output. In our problem, the expected mutual information (information gain) between the input (s_t, a_t) and output $M(s_t, a_t)$ of the model M is:

$$\begin{aligned}
& I((s_t, a_t), M(s_t, a_t)) \\
= \; & D_{KL}[P((s_t, a_t), M(s_t, a_t)) \| P(s_t, a_t) P(M(s_t, a_t))] \\
= \; & \int\int P((s_t, a_t), M(s_t, a_t)) \log \frac{P((s_t, a_t), M(s_t, a_t))}{P(s_t, a_t) P(M(s_t, a_t))} dM(s_t, a_t) d(st, at) \\
= \; & \int\int P(M(s_t, a_t)|s_t, a_t) P(s_t, a_t) \log \frac{P(M(s_t, a_t)|s_t, a_t)}{P(M(s_t, a_t))} dM(s_t, a_t) d(st, at) \\
= \; & \mathbb{E}_{(s_t, a_t) \sim D}[D_{KL}[P(M(s_t, a_t)|s_t, a_t) || P(M(s_t, a_t))]], \qquad (5.2)
\end{aligned}$$

where D is a dataset of tuples sampled from the environment and used for training. $P(M(s_t, a_t)|s_t, a_t)$ denotes the distribution of the output of model M, conditioned on the specific input (s_t, a_t). $P(M(s_t, a_t))$ is the distribution of entire output $M(s_t, a_t)$, i.e.,

$$P(M(s_t, a_t)) = \int P(M(s_t, a_t)|s_t, a_t) P((s_t, a_t)) d(s_t, a_t). \qquad (5.3)$$

In this way, if the probability $P(M(s_t, a_t)|s_t, a_t)$ is much greater than $P(M(s_t, a_t))$, we have the ability to distinguish the input only by the output, which means the output is a good representation of the input.

Since our purpose is to learn the best feature representation to reconstruct the input of model M, we train the model M by maximising 5.2:

$$\max_{\theta} \mathbb{E}_{(s_t,a_t)\sim D}[D_{KL}[P(M(s_t, a_t)|s_t, a_t)||P(M(s_t, a_t))]]. \tag{5.4}$$

Intuitively, one can view the world model as generating a notion of familiarity. When the agent is in state s_t that it has been visited many times before, and chooses action a_t that has been performed in that state before, it "feels" familiar and comfortable to the agent as the current state and action are well represented. However, the policy network π does an adversarial learning and tries to minimise the same mutual information and therefore encourages the agent to either choose actions it has not chosen before, or explore unfamiliar states. In other words, the agent is encouraged to get out of its comfort zone by minimising the mutual information in 5.2:

$$\min_{\pi} \mathbb{E}_{(s_t,a_t)\sim \pi}[D_{KL}[P(M(s_t, a_t)|s_t, a_t)||P(M(s_t, a_t))]]. \tag{5.5}$$

Minimising equation 5.5 is equivalent to minimising the pair-wise mutual information between the input and output of the model at each step t:

$$\log P(M(s_t, a_t)|s_t, a_t) - \log P(M(s_t, a_t)). \tag{5.6}$$

In summary, we train the model M by maximising the mutual information between its input and output to find the best feature representation. In the absence of extrinsic reward, the policy network π minimises exactly the same function that the model M is maximising. To be consistent with the previous reinforcement learning algorithms, the policy is updated by maximizing a reward function. We define the intrinsic reward r_t^i as the negative of Eq. 5.6.

Based on the above discussion, in this chapter, it is hypothesised that:

(5.1) MIME will perform as well as surprise-based method in normal sparse reward environments.

(5.2) MIME will perform better than surprise-based method in an environment where the transition is discontinuous.

5.4 Implementation

Computing the total probability $P(M(s_t, a_t))$ is intractable in practice (See Eq. 5.3), but since our purpose is to minimise $\log P(M(s_t, a_t)|s_t, a_t) - \log P(M(s_t, a_t))$, so in an approximate way, we choose to minimise the first term $\log P(M(s_t, a_t)|s_t, a_t)$. In other words, we only choose the first term from Eq. 5.6 as an approximate intrinsic reward:

$$r_t^i = -\log P(M(s_t, a_t)|s_t, a_t). \tag{5.7}$$

Similarly, the objective function in equation 5.4 for training the model M can be rewritten as:

$$\min_\theta -\frac{1}{D} \sum_{(s_t, a_t) \in D} \log P(M(s_t, a_t)|s_t, a_t). \tag{5.8}$$

If we suppose P is a Gaussian distribution and that M is autoencoder-like, then r_t^i can be written in a simple way:

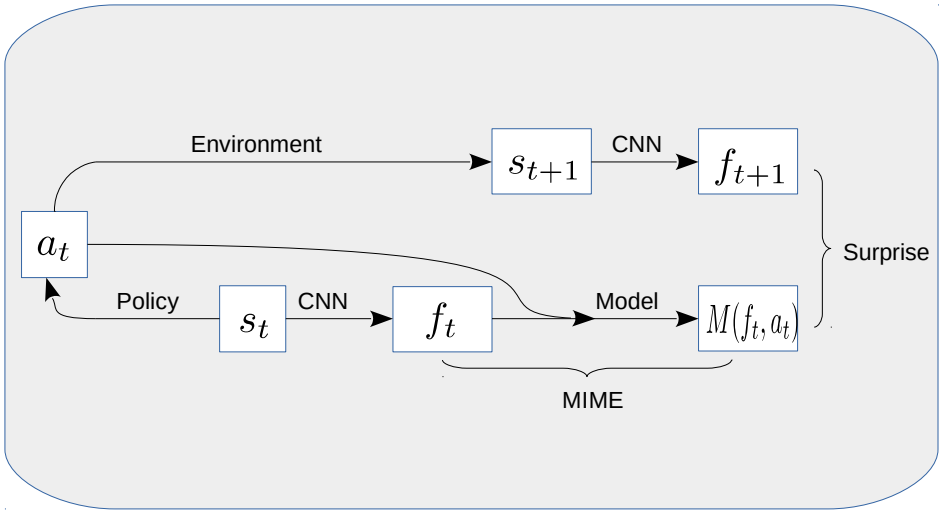
$$r_t^i \sim \|M(s_t, a_t) - s_t\|^2. \tag{5.9}$$

The entire training procedure is summarised in Algorithm 7. Figure 5.1 shows that the structure of surprisal-driven exploration is not changed, just the definition of intrinsic reward.

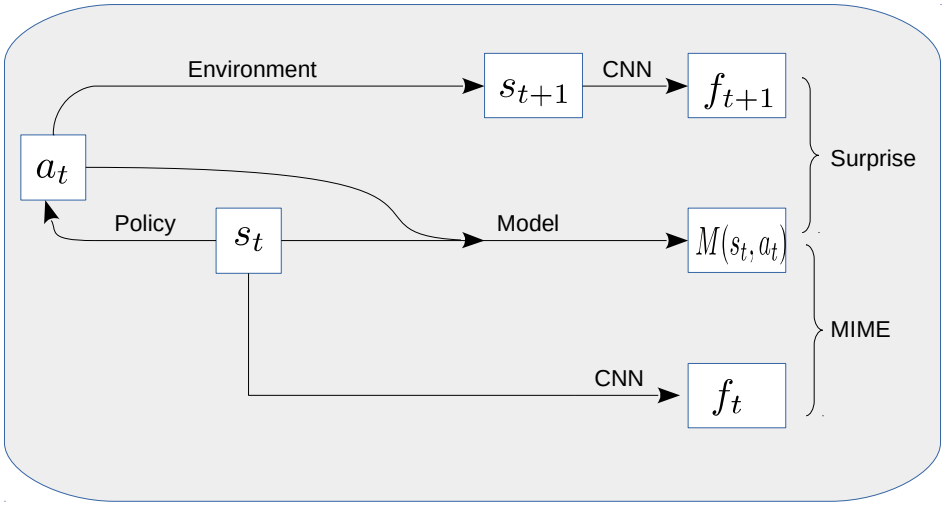
Algorithm 7: MIME-driven exploration for deep reinforcement learning

```
Initialise policy neural network π
Initialise world model M
Reset the environment getting (s_0, r_0))
for each iteration n do
    for each time step t do
        Get action a_t ~ π(·|s_t)
        Compute intrinsic reward r_t^i = ‖M(s_t, a_t) − s_t‖^2
        Construct cumulative reward r_t^e + η * r_t^i
        Take action a_t getting (s_{t+1}, r_{t+1}^e)
    end
    Update M by minimising the sum of r_t^i
    Update π by maximising the sum of r_t^e + η * r_t^i.
end
```

5.5 Experiments



(a) MIME-freeze-CNN layers



(b) MIME-trainable-CNN layers

Figure 5.3: Different structures used to compute intrinsic reward. Surprise denotes the surprisal reward.

For illustrative purposes, we begin with two simple experiments and visualise the agent's movements to show its exploration efficiency. Then to test hypothesis (5.1) and (5.2) proposed in Section 5.3, we implement MIME-driven exploration in three large-scale experiments: Gravitar, Doom, and Montezuma's Revenge. These three games have extremely sparse rewards and are a good test of exploration ability. Among them,

Gravitar is a normal sparse reward game environment to test hypothesis (5.1) and the other two environments have discontinuous transition boundaries to test hypothesis (5.2). All large-scale experiments are run three times with different seeds. Table 5.1 shows how we preprocessed the three large-scale environments. We use TRPO (Schulman *et al.*, 2015) in the two simple experiments and PPO (Schulman *et al.*, 2017) in large-scale games. In this chapter, we choose surprisal (not VASE) as the surprise-based comparator, because surprisal and VASE perform similarly (be stuck for a long time in the environments that have abrupt transitions, see Figure 4.7) and surprisal is computationally more efficient.

Table 5.1: Large-scale games environmental preprocessing.

Hyperparameter	Setting
Max and skip frames	4
Grey-scaling	True
Observation downsampling	(84, 84)
Max episode steps	4500
Terminal on loss of life	False

In the first two simple experiments, we follow the structure shown in Figure 5.1 and choose rllab (Duan *et al.*, 2016) as the platform to run the code. The model is a simple fully-connected neural network that has one hidden layer of 32 units and the number of units in output layer is equal to the dimension of state. The hidden layer has rectified linear unit (ReLU) non-linearities. The policy π is also a neural network that has one hidden layer of 32 units and tanh nonlinearities. For other hyper-parameter settings please check Table 5.2, where batch size refers to steps collected at each iteration.

Table 5.2: Hyperparameter setting for two simple experiments.

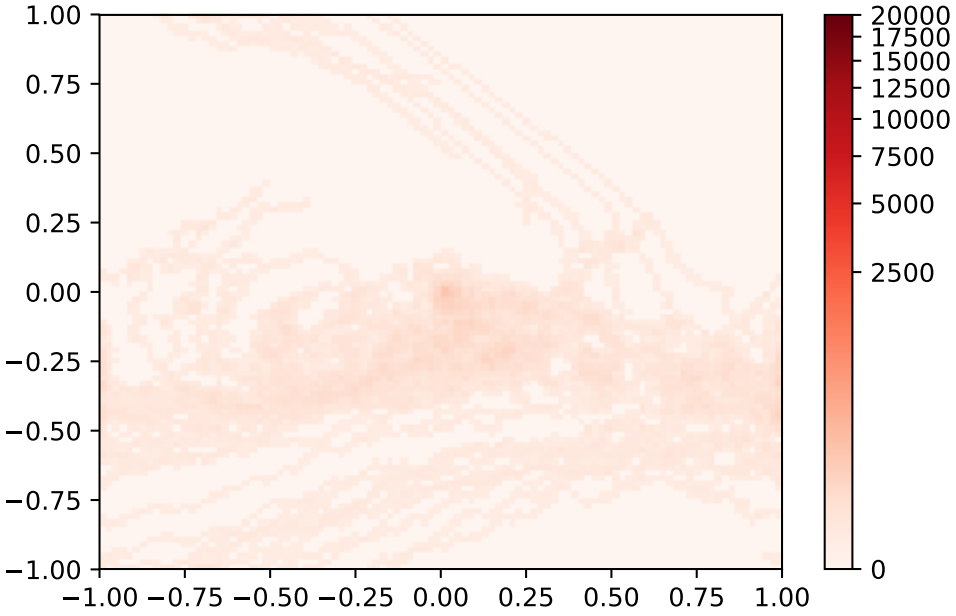
Hyperparameter	Setting
Batch size	5000
Max Rollout Length	500
Number of iterations	1000
Discount factor	0.99

In the three large-scale games, since the states are pixel-frames, if we also try the

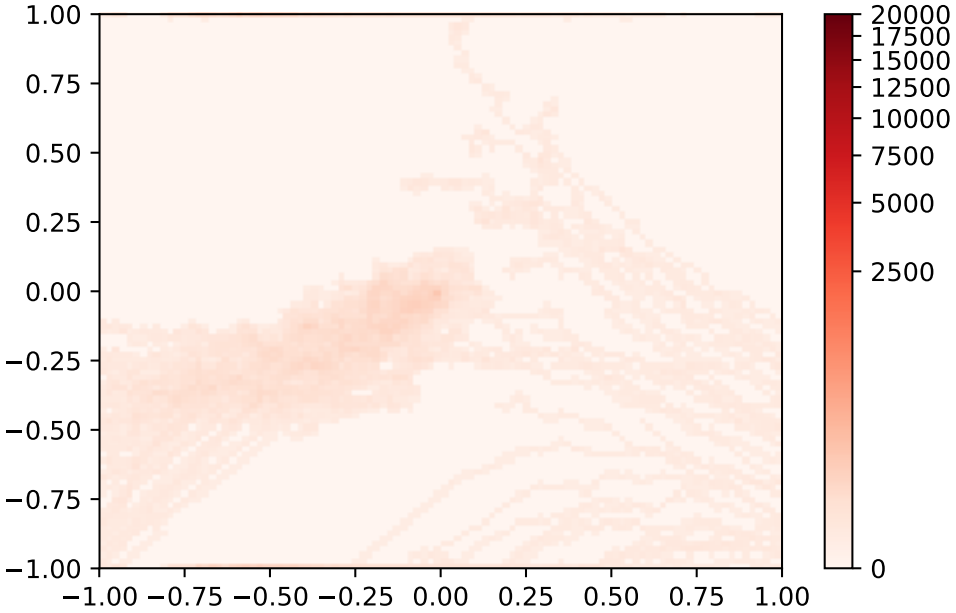
structure in Figure 5.1, we will compute the surprisal or mutual information from raw pixels. However, recent work Pathak *et al.* (2017); Burda *et al.* (2018) shows that if we map the raw pixels to a feature space first and compute the intrinsic reward in this space, we can get a much better result. The mapping can be any feature extractor such as a Variational Autoencoder (VAE) or a Convolution network (CNN) with weights frozen to randomly initialised values. In Chapter 4, we have chosen the latter for time efficiency reasons. We also use this CNN with frozen weights as the feature extractor in this chapter.

We use two different methods to compute MI in a CNN feature space (see Figure 5.3a and Figure 5.3b). In Figure 5.3a, the world model M is trained on the CNN feature space, but in Figure 5.3b the world model M is trained from pixels. Both of these two structures use a separate CNN network with frozen layers as a feature extractor so that we can compute intrinsic reward $\|M(f_t, a_t) - f_t\|^2$ or $\|M(s_t, a_t) - f_t\|^2$ in feature space. We also compare our results with RND in our large-scale experiments.

The feature extractor CNN in Figure 5.3 has three convolutional layers, which have 32, 64, 64 kernels, 8, 4, 3 kernel size and stride as 4, 2, 1, followed by a fully-connected linear layer with 512 output units. All the parameters in the CNN are fixed (no training). The world model in Figure 5.3a uses the output of the linear layer as its input and predicts future observations in feature space. It has two hidden non-linear layers and one linear output layer. All these fully-connected layers have 512 output units. The world model in Figure 5.3b on the other hand, uses the raw pixels as its input. It uses the same convolutional layer structure as the feature extractor CNN we introduced above, but the parameters are trained during learning. The self-predictor in Figure 5.2 has the same structure as the world model in Figure 5.3b. The difference is that the self-predictor only considers observations as its input and ignores actions. All the policy networks in Figure 5.2 and Figure 5.3 have the same structure. To compare RND as a baseline, we choose the same other hyper-parameter settings as RND in this paper (see Table 5.3). Please note that the trade-off between external rewards and internal rewards is set to 0.5, which is different from setting it to 10^{-4} in Chapter 3. This is because, in Chapter 3, all sparse environments only provide extrinsic rewards 1 after the agent finds the target. However, in the Atari video game in this chapter, the external rewards are usually hundreds or even thousands.



(a) Surprisal (22,919 steps)



(b) MIME (21,150 steps)

Figure 5.4: Exploration efficiency in 2DPlane environment until chancing upon the reward state.

Table 5.3: Hyperparameter setting for three large-scale games.

Hyperparameter	Setting
Rollout sampling length	128
Number of time steps	1e8
Number of optimization epochs	4
Reward trade off η	0.5
Number of parallel environments	32
Learning rate	1e-4
Discount factor for intrinsic reward	0.99
Discount factor for extrinsic reward	0.999
Frames stacked for policy	4
Frames stacked for model	1

5.5.1 2DPlane environment

This is the 2DPlane environment that we introduced in section 3.3.1. We choose this environment to show that MIME-agent has similar exploration ability with surprisal-agent.

In this experiment, we train one agent and record the observation coordinate (x, y) at each step until it finds the non-zero extrinsic reward. Figure 5.4 shows the heat map of location visits for the agent trained with surprisal reward and with MIME reward, respectively. Darker red represents a higher density, which means the agent takes more steps in this area. We can see that the exploration efficiency is similar between surprisal-agent and MIME-agent.

5.5.2 Passing through a wormhole

The wormhole experiment uses a three-dimensional environment with a sharp circular boundary (the wormhole) between an upper rectangular planar environment at $z = 1000$, and a lower second circular planar environment centred at the origin with radius 0.5 (See Figure 5.5). The observation space is 3D ($(x, y, z) \in \mathbb{R}^3$). The action is still a two dimensional vector: the velocity $(\dot{x}, \dot{y})$ that satisfies $\sqrt{\dot{x}^2 + \dot{y}^2} \leq 0.01$. The agent starts from the origin $(x = 0, y = 0, z = 0)$. When the agent crosses the boundary, it immediately transitions from one plane to the other. No extrinsic reward is provided in this environment. So this is a pure exploration problem. A uniform distribution of

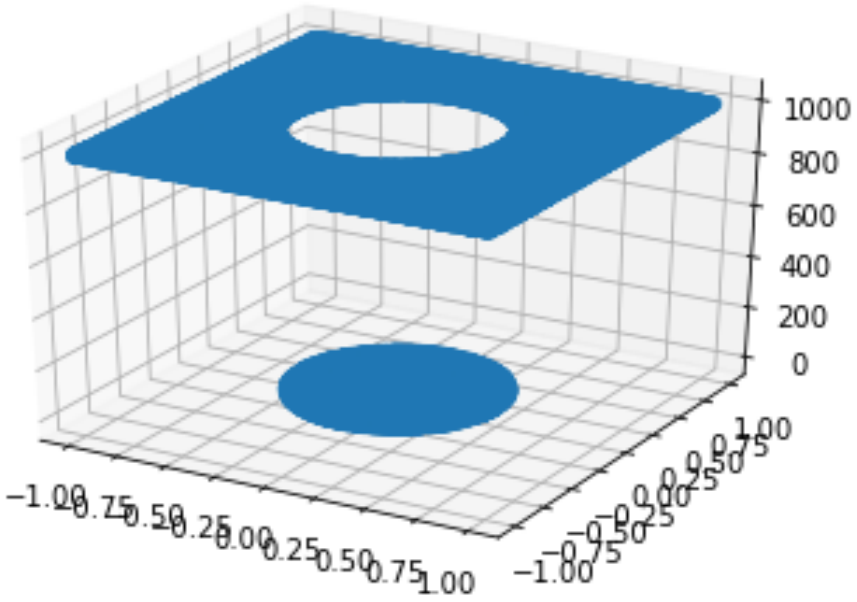
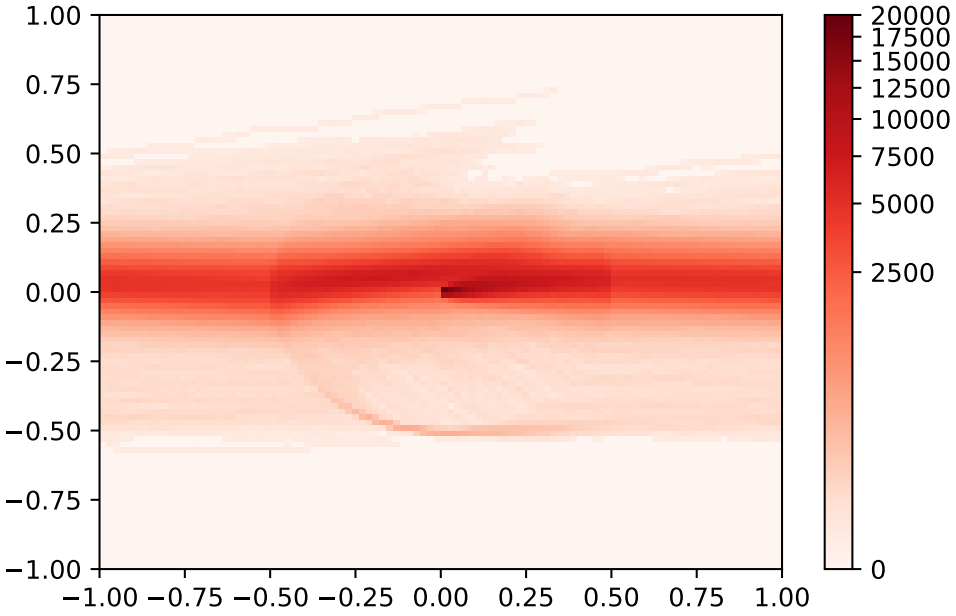
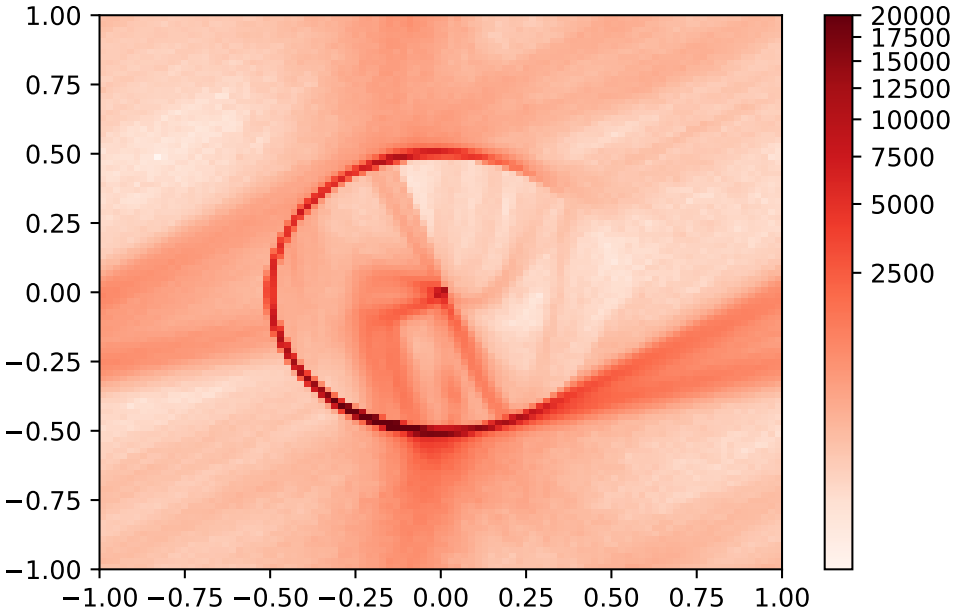


Figure 5.5: Pass through a wormhole: this is a three-dimensional environment with a sharp circular boundary (the wormhole) between an upper rectangular planar environment at $z = 1000$, and a lower second circular planar environment centred at the origin with radius 0.5. The agent starts from the origin $(x = 0, y = 0, z = 0)$. When the agent crosses the boundary, it immediately transitions from one plane to the other.

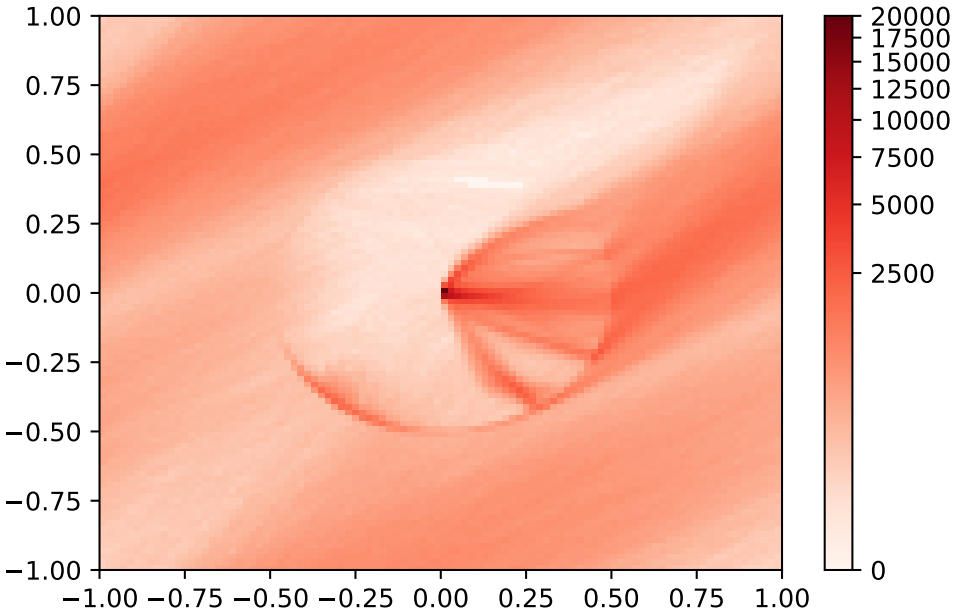


(a) No intrinsic reward



(b) Surprisal

Figure 5.6: Pass through a wormhole, 5 million steps (continued on next page).



(c) MIME

Figure 5.6: Pass through a wormhole, 5 million steps (continued from previous page).

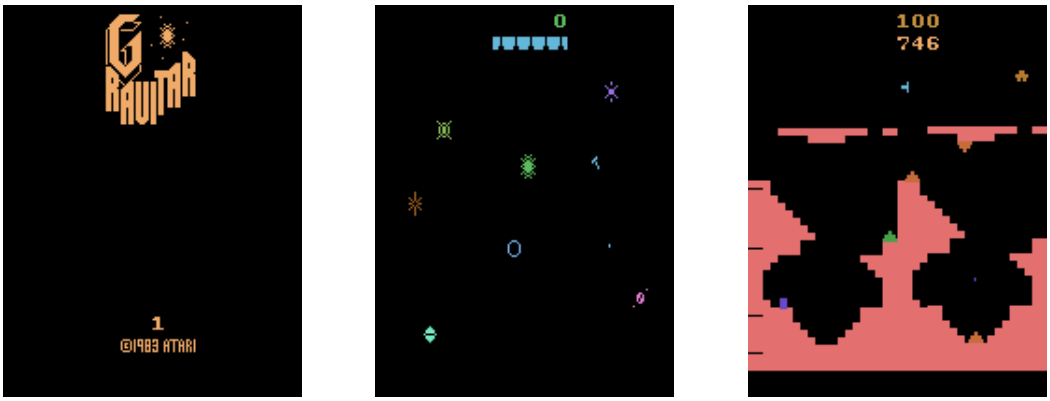
visits is the optimal exploration strategy. Similar to the previous experiment, darker red represents a higher density, which means the agent takes more steps in this area and loses exploration efficiency. All the agents explore 5,000,000 steps. Figure 5.6a to Figure 5.6c show the top view of the environment so that we can also visualise the agent's movements. We can see both MIME and surprisal agents are attracted by the boundary, but the time that surprisal agents stay at the boundary is much longer than the MIME agent. As can be seen from Figure 5.6a, the random exploration agent is not affected by the boundary, but the exploration efficiency is very low.

5.5.3 Large-scale games

In this subsection, we test MIME and compare it to surprisal and RND in three large-scale games: Gravitar, Doom, and Montezuma's Revenge. For MIME, we implement two different structures as shown in Figure 5.3a and Figure 5.3b respectively. All experiments run with 32 parallel environments.

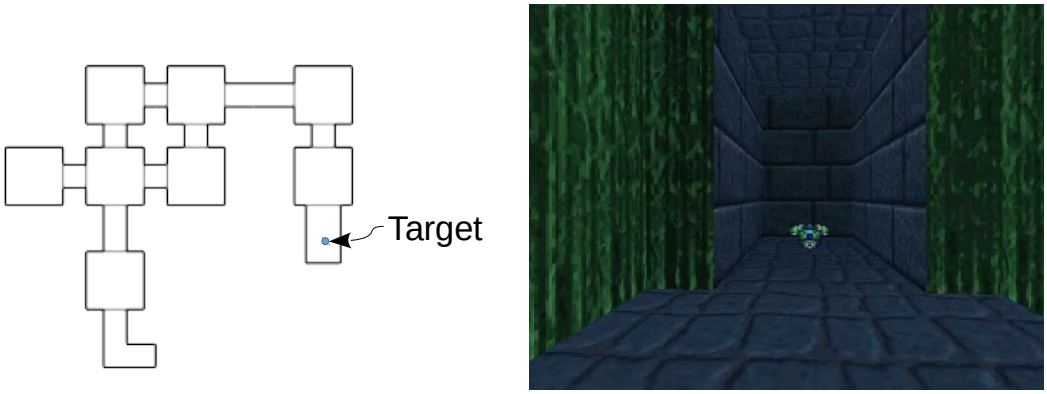
Gravitar (Figure 5.7) is an Atari game released in 1982. In the game, the player controls spacecraft to explore. There are two modes: the overworld (essentially space); and a sideview landscape when the spaceship enters a planet environment. The agent (spaceship) will be pulled slowly to the star in the overworld, and downward in the

side-view levels. The player will lose a life if the spacecraft crashes. If the fuel is out the game will also end. We chose this game to show a similar exploration ability between surprisal and MIME. It can be seen from Figure 5.9a that the agent trained by MIME, surprisal and RND performs similar in this game. The MIME agent that has frozen CNN layers also performs as good as the one with trainable CNN layers. This supports hypothesis (5.1)



(a) (b) (c)

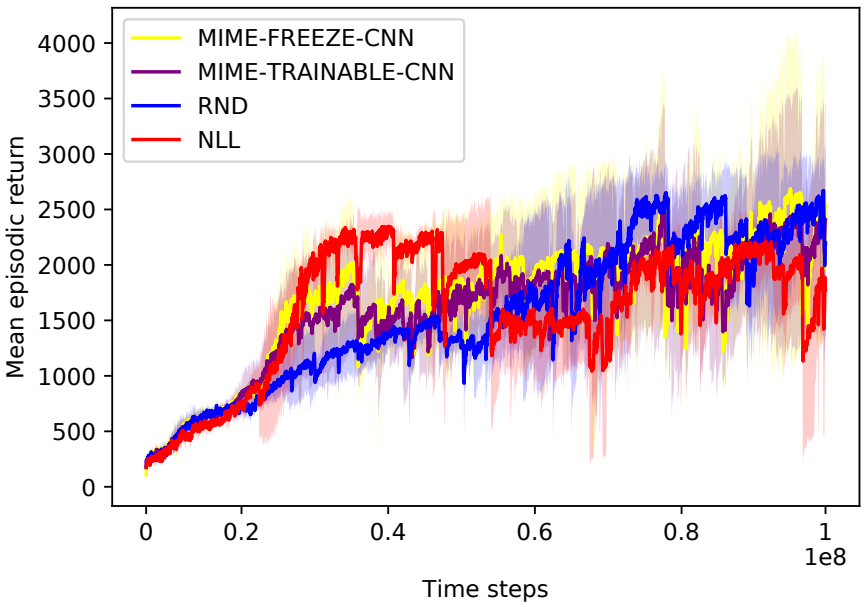
Figure 5.7: Gravitar game



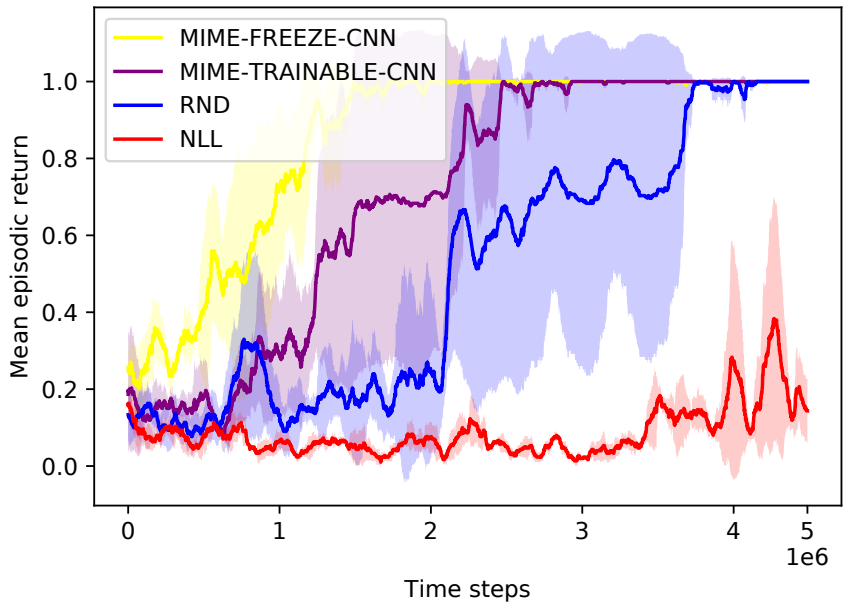
(a) Environment map (b) Target room

Figure 5.8: VizDoom scenario: "find my way home".

We choose the scenario named "find my way home" (Figure 5.8) in VizDoom game (Kempka, Wydmuch, Runc, Toczek, and Jaśkowski, 2016; Wydmuch, Kempka, and Jaśkowski, 2018) to train the agent to navigate in surroundings and reach his ultimate goal. The map is a series of connected rooms and one corridor with a dead end.

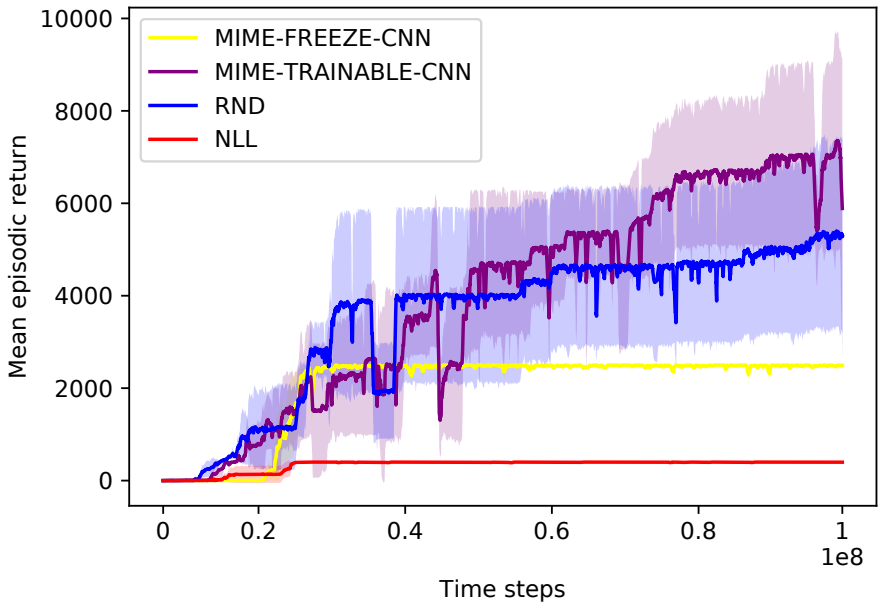


(a) Gravitar



(b) Doom with TV

Figure 5.9: Mean episodic return of MIME, surprisal (NLL), and RND on 3 hard exploration large-scale games. Curves are an average over 3 random seeds, with standard deviation shown in shaded areas. Horizontal axes show numbers of frames(continued on next page).



(c) Montezuma's Revenge

Figure 5.9: Mean episodic return of MIME, surprisal (NLL), and RND on 3 hard exploration large-scale games. Curves are an average over 3 random seeds, with standard deviation shown in shaded areas. Horizontal axes show numbers of frames (continued from previous page).

Each room has a different colour and texture. There is a green vest in one of the rooms (the same room every time). The agent is born in a randomly chosen room facing a random direction. When the agent explores in this map and finds the vest (the goal), it gets a 1 point reward. To show whether the agent trained by MIME or surprisal is stuck at the discontinuous area in this environment, we edit the map and add a TV (always shows changing frames) on the wall in one room. Here the changing frames are uniformly and randomly chosen from a frame pool of a fixed size and used to simulate discontinuous environmental transitions, as if someone is rapidly cycling through different TV channels. This can be done by a Zoom editor called Slade_3[1]. The experiment is run for 5 million frames. We observe that the surprisal agent gets stuck in the TV room, and only the agent born in a room between the TV room and the goal could find the goal, however, both MIME and RND driven agents can escape from the TV room[2]. This support hypothesis (5.2) that MIME will perform better than surprise-based method in an environment that has discontinuous transitions. As with the previous experiment, there is not much difference in the exploration ability between the agents driven from two different structures of MIME (See Figure 5.9b).

Montezuma's revenge game (See Figure 4.3c) is an Atari game that is known for its notorious difficulty. Most traditional RL algorithms (Mnih *et al.*, 2015; Hessel *et al.*, 2018) that are successful at other Atari games cannot solve Montezuma's revenge game due to its extremely sparse rewards. For example, in the first room of Montezuma's revenge, to get the first key (and this is also the first reward from the game), the agent needs to climb down a ladder, jump on a rope, climb down another ladder, jump over a skull, and then finally climb up another ladder. If one of the above steps failed, the agent may die in the game. This is just the first key in the first room and Montezuma's revenge game has 24 rooms in level 1. In recent years, Montezuma's revenge game has been seen as a grand challenge for RL researchers and of course, also inspired the development of some of RL algorithms (Ostrovski, Bellemare, van den Oord, and Munos, 2017; Aytar, Pfaff, Budden, Paine, Wang, and de Freitas, 2018; Burda *et al.*, 2018; Ecoffet *et al.*, 2019; Ecoffet, Huizinga, Lehman, Stanley, and Clune, 2020). This game has similar game mechanics as the wormhole environment we designed in subsection 5.5.2: when the agent moves into a new room, the state abruptly changes because the background of each room is different. Surprisal agents get stuck at the boundary between adjacent rooms because of this transition. In Figure 5.9c the surprisal agent can

[1] `http://slade.mancubus.net`

[2] `https://drive.google.com/open?id=1P15KBDhZisuml-X_KPc2RBSdYCNu0_yX`

only achieve a score of 400 because it is stuck at the boundary between the first and second rooms. We have also applied our VASE algorithm to Montezuma's Revenge in Chapter 4, and found that the VASE agent also can be stuck at room boundaries like surprisal agent. A MIME agent with trainable CNN layers performs somewhat better than RND agents, which also supports hypothesis (5.2). It is interesting to see that MIME agent with frozen CNN layers only scores 2500. The game offers a substantial reward for agents that return to room 2 with the reward from room 3 (The premise is to find the sword in the third room and return to the second room to kill one skull with the sword, see Figure 5.10). The frozen CNN MIME agent discovers this reward, and in the process, the policy encourages the agent to go back to room 2 after room 3. We also observe that the RND agent initially gets stuck in the same undesirable loop as the frozen CNN MIME agent. However, the RND agent does manage to escape from this loop in 2 out of 3 of the trials/seeds. We believe that extending training should allow the frozen CNN MIME agent to escape from this loop. For a simulated video of MIME agent play Montezuma's revenge, please check [3]

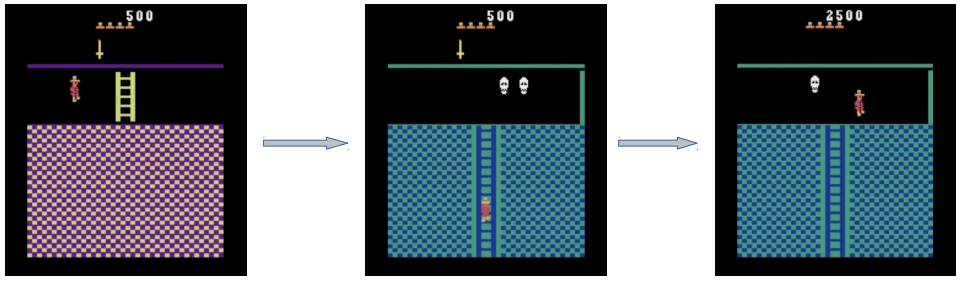


Figure 5.10: If the agent can find the sword in the third room and return to the second room to kill one skull with the sword, it will obtain a substantial reward (2000).

5.6 Discussion and Conclusion

One limitation of our approach is that when we maximise the mutual information, we maximise the KL divergence, which is theoretically without upper bound. For autoencoder-like world models, if the model learns the identity function, the network will be able to reproduce the input regardless of whether it has seen it before. Importantly, this will result in approximately the same intrinsic reward values for every

[3] `https://drive.google.com/open?id=1dPDj4ZdWzttfj3PHPUeuYgyGfm2u3crB`

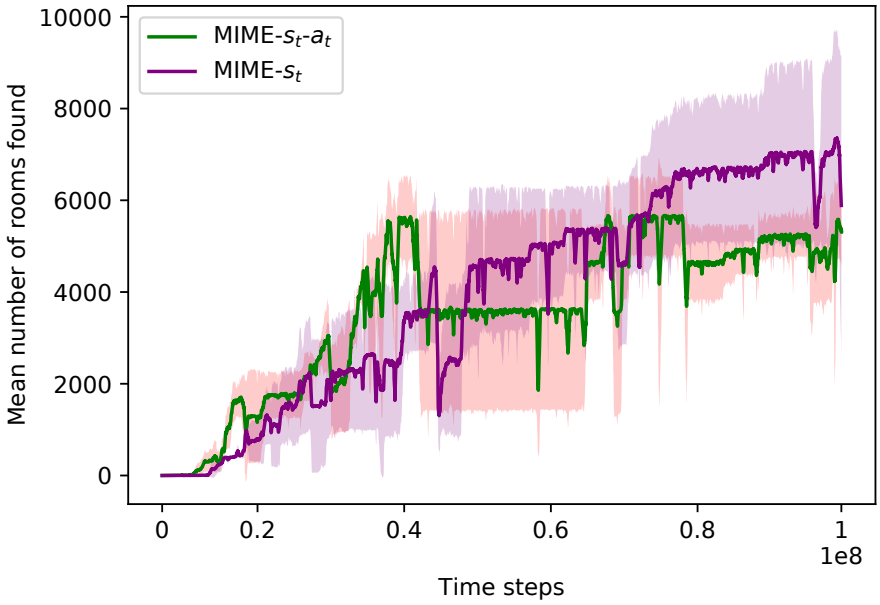
possible observation, regardless of its novelty. In practice, we found that when the policy adversarially steers the agent exactly towards areas where the states do not belong to the same distribution of the previous ones, the model will be inaccurate and the process of learning identity function will slow down.

We can see from Eq. 5.9 that when we compute the intrinsic reward r^i, we calculate the error between the model output and the input, which means the output $M(s_t, a_t)$ is a representation of s_t (f_t in feature space). What happens if we denote the output $M(s_t, a_t)$ as the representation of (s_t, a_t)? In other words, we compute the intrinsic reward by the following equation:

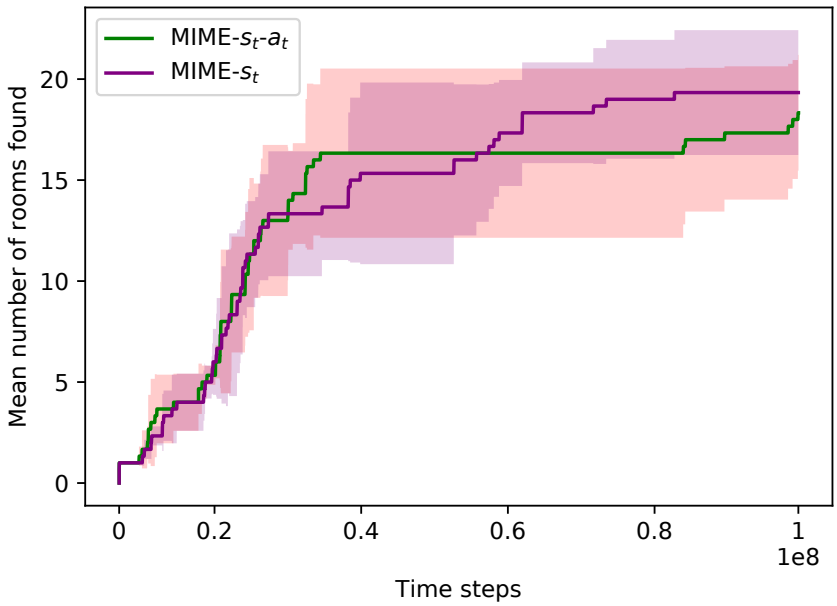
$$r_t^i \sim \|M(s_t, a_t) - (s_t, a_t)\|^2. \tag{5.10}$$

We tried an experiment on Montezuma's Revenge game and compute the intrinsic reward by Eq. 5.10. Figure 5.11 shows the results of average episode rewards and the average number of rooms found, trained by Eq. 5.9 and Eq. 5.10 respectively. We can see from Figure 5.11 that if we consider action a_t, the exploration performance is better in the first half of iterations, but worse in the last half. The reasons for this phenomenon need to be further studied. Because the action space of Atari video games is a discrete space, it also deserves to investigate whether there is a difference if we change the environment to one with continuous action space.

To conclude, the main difference between MIME agents and other common RL agents, is that MIME agents do not try to predict the future. Rather, they form a measure of how comfortable they are in a given environment. Whereas surprisal agents explore areas where prediction is poor, MIME agents explore areas where it has a poor world model. As a consequence, surprisal agents tend to seek out hard to learn transition boundaries, whilst MIME agents are encouraged to leave their comfort zone. This is a simple idea, is easy to implement and most importantly it overcomes the limitations of surprisal getting stuck at transition boundaries.



(a) Mean episodic return



(b) Mean number of rooms found

Figure 5.11: Compare results between Eq. 5.9 and Eq. 5.10, where MIME-s_t-a_t denotes the results of Eq. 5.10 and MIME-s_t denotes the results of Eq. 5.9.

Chapter 6

Conclusion

Reinforcement learning is a special machine learning mechanism. Agents need to interact with the environment at each time step to obtain information about new states and rewards. Rewards are important because all reinforcement learning algorithms require rewards to train good policies. However, some environments can only provide very sparse rewards. In this thesis, we have investigated how to design and generate intrinsic rewards and use them for deep reinforcement learning exploration. Driven by this intrinsic reward, the agent can efficiently explore to discover the sparse rewards obtained from the environment.

Now, we summarise contributions based on these different kinds of intrinsic rewards and their applications:

- **VASE: Variational Assorted Surprise Exploration**

  We first proposed a new definition of surprise that can be used as an intrinsic reward for deep reinforcement learning exploration, which we call assorted surprise. Surprise has been cast as a cognitive-emotional phenomenon that impacts many aspects from creativity to learning to decision-making. In reinforcement learning, a surprise-driven agent can learn to explore without knowing any reward system from the environment. Agents learn in a reinforcement learning environment by maximizing this "surprise".

  Before we presented our assorted surprise concept, there are two frequently used types of surprise intrinsic rewards in the field of reinforcement learning, one is surprisal, the other is Bayesian surprise. Surprisal is a measure of the information content of the outcome. It is the inconsistency between the model prediction and observed environment outcome. Bayesian surprise measures the amount of

information needed to transform the agent's prior belief into the posterior belief distribution. Surprisal is intuitive and simple to use. Bayesian surprise is good at the uncertainty in environments. However, both of them have shortcomings. Surprisal is not good at dealing with uncertainties because it is a point estimation method. Bayesian surprise can only be applied to small and simple problems because of its high computation cost. And Bayesian surprise can only be computed after the learning step.

Our assorted surprise idea comes from ensemble learning. Ensemble learning is a technique that trains multiple models to solve the same problem, and combines them to get better, more reliable predictions with lower variance and/or lower bias. In terms of our problem, we suppose the agent creates many models, and each model can predict the next state of the environment. Because the prediction is different between these models, each model gets different surprise (surprisal). We combine these surprises generated from different models together and give our assorted surprise definition: The assorted surprise is the expectation of these surprisals. By this definition, we can see that the assorted surprise doesn't have the shortcomings that surprisal and Bayesian surprise have. First, it is simple: it is only an expectation of suprisal, with cheap computation consumption. It is more robust compared to using a single model. In addition, the assorted surprise is computed as soon as the agent sees the next state, without the need to update the agent's belief like Bayesian surprise does. Furthermore, we also showed in Chapter 3 that the assorted surprise is the sum of surprisal and Bayesian surprise. We also add a confidence term, the entropy of the model parameters. We let our assorted surprise subtract this confidence term on the intuition that a confident agent model will have low entropy and will be more surprised when it sees an unlikely state.

Finally, in practice, we choose the Bayesian neural network as the agent's model to predict the next environmental state. This is because the expected value needs to be calculated when calculating our assorted surprise, which means that the probability of each model is required. However, this is intractable since the probability is a posterior distribution and updates at each time step. Combined with the variational inference technique, the Bayesian neural network can maintain this posterior distribution with a variational approximated distribution. Because the parameters in Bayesian neural network are all distributions, we can sample each agent's model based on this variational approximated distribution. That's

why we call our proposed exploration algorithm variational assorted surprise exploration (VASE).

- **VASE for continuous control task with sparse rewards**

  To show the performance of VASE algorithm, we first implemented it in a simple 2D exploration environment. Both state and action space in this 2D experiment are continuous. The experiment showed that with VASE algorithm, the agent can find the target in 26,663 steps. However, for the agent not driven by VASE, it needs about 2,059,459 steps. This experiment told us the agent could explore more efficiently driven by VASE algorithm. Then we applied VASE to five other environments. All these environments provide sparse reward. Among them, MontainCar, CartpoleSwingup and DoublePendulum are three classic continuous control environments. HalfCheetah and Ant are two locomotion tasks. We also applied VASE to LunarLanderContinuous environment. This environment is not a sparse reward environment, it provide reward to the agent at each time step. We chose this environment to investigate how VASE would perform in non-sparse reward environments. Our experimental results showed that VASE performed well on all these environments, compared to surprisal and Bayesian surprise.

- **VASE for Atari video games: Apply VASE to large scale problems**

  In previous work, the environments that we chose are simple. The dimensions of the state space and the action space of the environment are not high. Both the policy neural network and prediction model network are small (one hidden layer or two hidden layers fully connected). So to further study the exploration ability of VASE algorithm, we applied VASE to large scale problems: we let the agent play Atari video games.

  Atari video games are good environments to investigate the exploration ability of VASE algorithm on large scale problems, because its video frames are high dimensional pixel images (210x160x3). We first did some data preprocessing. Then we construct the neural networks to be used for policy, feature extractor and prediction model respectively. The assorted surprise is calculated in feature space, the output of the feature extractor network.

  In experiment, we first removed the game score that provided from the game, and trained the agent only by intrinsic reward. We chose three Atari video games, BreakOut, Pong and Montezuma's Revenge. Experimental results showed that

VASE driven agent played well and compared to surprisal driven agent, it always found some policies that can let it survive forever in the game, which the surprisal driven agent cannot find when training in the same time steps. In the end, the Montezuma's Revenge game seems too difficult to both VASE driven and surprisal driven agents. When we added game scores back as an extrinsic reward, both agents explored much better. However, we observed that the agents would be stuck at the boundary of rooms for a long time.

- **MIME: Mutual Information Minimisation Exploration**

  In previous work, we found that surprise (both surprisal and VASE) driven agents would get stuck at the boundary of rooms in Montezuma's Revenge. To solve this problem, we proposed a novel exploration strategy, which we call mutual information minimisation exploration (MIME). The agent's model network is trained by maximising this mutual information to find the best feature representation. On the contrary, the policy network is trained by minimising the mutual information, so that encourages the agent to explore unfamiliar states.

  We implemented MIME on three video games: Gravitar, Montezuma's Revenge and Zoom. Gravitar is a normal Atari video games with sparse reward. We chose this game to show that in the environment where the state transition is continuous, MIME driven agents can explore as well as surprisal. MIME performs much better than surprisal on Montezuma's Revenge and Zoom. For Montezuma's Revenge, the stuck phenomenon never happens any more at the boundary of rooms. For Zoom, we add a TV on the wall in one room. MIME driven agents will never be attracted by the TV and can always find the target, however, agents driven by surprisal will choose to stop at the TV room and struggle whether to continue to watch TV or leave that room.

6.1 Future work

In this thesis, I proposed different ways to define/design intrinsic reward functions in deep reinforcement learning, which can be used to drive the agent to explore its environment more efficiently. Different factors will affect the agent's exploration process. Due to time constraints, I cannot solve these challenges in a single Ph.D. The contribution of this article raises the following questions for future work:

- We showed in Chapter 3 that VASE can quantify both the aleatoric and epistemic uncertainty simultaneously. However, these two kinds of uncertainties are combined implicitly together. A way to explicitly decompose these uncertainties and assign different weights to aleatoric uncertainty and epistemic uncertainty during the training process would be worth pursuing.

- We choose Bayesian neural network as the prediction model of VASE algorithm. The model distribution $P(M)$ is maintained by the distribution of its parameters $P(\theta)$. The structure of the Bayesian neural network model is fixed. However, this can be extended to more general cases. We can choose different models that have different network structures. We can also try other possible regimes listed at the beginning of section 3.2.2.

- When we train Bayesian neural networks in VASE algorithm, we assume the variational posterior distribution is factorised regardless of whether the true posterior is a factorised distribution or not. The variational posterior learned under this assumption cannot capture the dependence on latent variables. We need to develop a new model with the assumption that the variational posterior distribution is dimensional-wise dependent.

- In Chapter 4, the parameters of feature extractor network (a CNN network) are randomly initialised and fixed and then not trained during the learning process. This can reduce the computation cost. Although experiments in this thesis showed good results, there is still no theoretical guarantee currently why a feature extractor network can work well with frozen parameters. It is worth studying whether there is a theoretical proof.

- For MIME algorithm, when we maximise the mutual information, we maximise the KL divergence, which is theoretically without upper bound. This could lead to an identity problem: the model network will just reproduce its input then the mutual information between the input and output would be the same for all different states. A possible research direction is to choose a divergence with an upper bound to compute the mutual information.

References

Achiam, J. and Sastry, S. (2017). Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*.

Ahmed, A., Aly, M., Gonzalez, J., Narayanamurthy, S., and Smola, A. J. (2012). Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*, 123–132.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

Aubret, A., Matignon, L., and Hassas, S. (2019). A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*.

Aytar, Y., Pfaff, T., Budden, D., Paine, T., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, 2930–2941.

Baldi, P. and Itti, L. (2010). Of bits and wows: a Bayesian theory of surprise with applications to attention. *Neural Networks*, *23*(5), 649–666.

Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, *61*(1), 49–73.

Barto, A., Mirolli, M., and Baldassarre, G. (2013). Novelty or surprise? *Frontiers in Psychology*, *4*, 907.

Barto, A. G. (2013). Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, 17–47. Springer.

Bell, A. J. and Sejnowski, T. J. (1995). An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, *7*(6), 1129–1159.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems 29*, 1471–1479.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798–1828.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 41–48.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, 1613–1622.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural computation*, *11*(7), 1493–1517.

Brim, A. (2020). Deep Reinforcement Learning Pairs Trading with a Double Deep Q-Network. In *10th Annual Computing and Communication Workshop and Conference, CCWC 2020, Las Vegas, NV, USA, January 6-8, 2020*, 222–227. IEEE.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018). Large-Scale Study of Curiosity-Driven Learning. In *International Conference on Learning Representations*.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. In *International Conference on Learning Representations*.

Chentanez, N., Barto, A. G., and Singh, S. P. (2005). Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17*, 1281–1288. MIT Press.

Chu, T., Wang, J., Codecà, L., and Li, Z. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, *21*(3), 1086–1095.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 4754–4765.

Clements, W. R., Robaglia, B.-M., Van Delft, B., Slaoui, R. B., and Toth, S. (2019). Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*.

Co-Reyes, J., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. (2018). Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, 1009–1018.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Dasgupta, S. (2000). Experiments with random projection. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, 143–151.

Deci, E. L. and Ryan, R. M. (2010). Intrinsic motivation. *The corsini encyclopedia of psychology*, 1–2.

Deisenroth, M. P., Neumann, G., Peters, J., *et al.* (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, *2*(1–2), 1–142.

Depeweg, S., Hernández-Lobato, J. M., Doshi-Velez, F., and Udluft, S. (2017). Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. *arXiv preprint arXiv:1710.07283*.

Depeweg, S., Hernandez-Lobato, J.-M., Doshi-Velez, F., and Udluft, S. (2018). Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, 1184–1193.

Der Kiureghian, A. and Ditlevsen, O. (2009). Aleatory or epistemic? Does it matter? *Structural safety*, *31*(2), 105–112.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). OpenAI Baselines. `https://github.com/openai/baselines`.

Dua, D. and Graff, C. (2017). UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 1329–1338.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl2: Fast reinforcement learning via slow reinforcement learning. 2016. *arXiv preprint arXiv:1611.02779*.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2020). First return then explore. *arXiv preprint arXiv:2004.12919*.

Faraji, M. (2016). *Learning with Surprise: Theory and Applications*. Ph. D. thesis, Ecole Polytechnique Fédérale de Lausanne.

Faraji, M., Preuschoff, K., and Gerstner, W. (2016). Balancing New Against Old Information: The Role of Surprise in Learning. *arXiv preprint arXiv:1606.05642*.

Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic Goal Generation for Reinforcement Learning Agents. In *International Conference on Machine Learning*, 1515–1528.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2018). Noisy Networks For Exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, *55*(1), 119–139.

Gal, Y. (2016). *Uncertainty in deep learning*. Ph. D. thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2015). Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

Gao, W. and Zhou, Z.-H. (2013). On the doubt about margin explanation of boosting. *Artificial Intelligence*, *203*, 1–18.

Gelfand, A. E. and Smith, A. F. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, *85*(410), 398–409.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 721–741.

Georgeon, O. L., Marshall, J. B., and Ronot, P.-Y. R. (2011). Early-stage vision of composite scenes for spatial learning and navigation. In *2011 IEEE International Conference on Development and Learning (ICDL)*, Volume 2, 1–6. IEEE.

Giryes, R., Sapiro, G., and Bronstein, A. M. (2016). Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, *64*(13), 3444–3457.

Gopnik, A., Meltzoff, A. N., and Kuhl, P. K. (1999). *The scientist in the crib: Minds, brains, and how children learn.* William Morrow & Co.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24*, 2348–2356. Curran Associates, Inc.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, *57*, 97–109.

He, J., Zhuang, F., Liu, Y., He, Q., and Lin, F. (2019). Bayesian dual neural networks for recommendation. *Frontiers of Computer Science*, *13*(6), 1255–1265.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., *et al.* (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. (2016). Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*.

Henaff, M., LeCun, Y., and Canziani, A. (2019). Model-predictive policy learning with uncertainty regularization for driving in dense traffic. In *7th International Conference on Learning Representations, ICLR 2019*.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. In S. A. McIlraith and K. Q. Weinberger (Eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 3215–3222. AAAI Press.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth ACM Conference on Computational Learning Theory*, 5–13. ACM Press.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, 1109–1117.

Islam, R. (2016). Active learning for high dimensional inputs using Bayesian convolutional neural networks. Master's thesis, Department of Engineering, University of Cambridge.

Itti, L. and Baldi, P. (2005). A principled approach to detecting surprising events in video. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Volume 1, 631–637.

Itti, L. and Baldi, P. F. (2006). Bayesian surprise attracts human attention. In *Advances in Neural Information Processing Systems 18*, 547–554.

Jin, J., Song, C., Li, H., Gai, K., Wang, J., and Zhang, W. (2018). Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2193–2201.

Johnson, W. B. and Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, *26*(189-206), 1.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, *37*(2), 183–233.

Kakade, S. M. and Langford, J. (2002). Approximately Optimal Approximate Reinforcement Learning. In C. Sammut and A. G. Hoffmann (Eds.), *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, 267–274. Morgan Kaufmann.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine learning*, *49*(2-3), 209–232.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Kendall, A., Badrinarayanan, V., and Cipolla, R. (2015). Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, 5574–5584.

Kennedy, P. (2003). *A guide to econometrics*. MIT press.

Khemchandani, R., Chandra, S., *et al.* (2007). Twin support vector machines for pattern classification. *IEEE Transactions on pattern analysis and machine intelligence*, *29*(5), 905–910.

Khemchandani, R. and Sharma, S. (2016). Robust least squares twin support vector machine for human activity recognition. *Applied Soft Computing*, *47*, 33–46.

Kim, K. I., Jung, K., and Kim, H. J. (2002). Face recognition using kernel principal component analysis. *IEEE signal processing letters*, *9*(2), 40–42.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, 2575–2583.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Knight, F. H. (2012). *Risk, uncertainty and profit*. Courier Corporation.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Kucukelbir, A., Ranganath, R., Gelman, A., and Blei, D. (2015). Automatic variational inference in Stan. In *Advances in neural information processing systems*, 568–576.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, *22*(1), 79–86.

Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, *51*(2), 181–207.

Kwon, Y., Won, J.-H., Kim, B. J., and Paik, M. C. (2018). Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. In *International Conference on Medical Imaging with Deep Learning, Amsterdam*, 4–6.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, 6402–6413.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. (2019). Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, *17*(1), 1334–1373.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, *37*(4-5), 421–436.

Li, K. and Malik, J. (2017). Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*.

Lima, A., Zen, H., Nankaku, Y., Miyajima, C., Tokuda, K., and Kitamura, T. (2004). On the use of kernel PCA for feature extraction in speech recognition. *IEICE TRANSACTIONS on Information and Systems*, *87*(12), 2802–2811.

Linsker, R. (1988). Self-organization in a perceptual network. *Computer*, *21*(3), 105–117.

Little, D. Y.-J. and Sommer, F. T. (2013). Learning and exploration in action-perception loops. *Frontiers in neural circuits*, *7*, 37.

Liu, Y., Zhao, G., Nacewicz, B. M., Adluru, N., Kirk, G. R., Ferrazzano, P. A., Styner, M., and Alexander, A. L. (2019). Accurate Automatic Segmentation of Amygdala Subnuclei and Modeling of Uncertainty via Bayesian Fully Convolutional Neural Network. *arXiv preprint arXiv:1902.07289*.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems 25*, 206–214. Curran Associates, Inc.

Luo, X. and Durrant, R. J. (2017). Maximum Margin Principal Components. *arXiv preprint arXiv:1705.06371*.

Mangasarian, O. L. and Musicant, D. R. (1999). Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, *10*(5), 1032–1037.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 50–56.

Matthies, H. G. (2007). Quantifying uncertainty: modern computational representation of probability and applications. In *Extreme man-made and natural hazards in dynamics of structures*, 105–135. Springer.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Mobiny, A., Nguyen, H. V., Moulik, S., Garg, N., and Wu, C. C. (2019). DropConnect Is Effective in Modeling Uncertainty of Bayesian Deep Networks. *arXiv preprint arXiv:1906.04569*.

Mohamed, S. and Rezende, D. J. (2015). Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, 2125–2133.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, *11*, 241–276.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nikolov, N., Kirschner, J., Berkenkamp, F., and Krause, A. (2018). Information-Directed Exploration for Deep Reinforcement Learning. In *International Conference on Learning Representations*.

Oh, J., Chockalingam, V., Lee, H., *et al.* (2016). Control of Memory, Active Perception, and Action in Minecraft. In *International Conference on Machine Learning*, 2790–2799.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, *11*, 169–198.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. (2017). Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 2721–2730. JMLR.org.

Oudeyer, P.-Y., Kaplan, F., *et al.* (2008). How can we define intrinsic motivation. In *Proc. of the 8th Conf. on Epigenetic Robotics*, Volume 5, 29–31.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, *11*(2), 265–286.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 16–17.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter Space Noise for Exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Platt, J. *et al.* (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, msr-tr-98-14, Microsoft Research.

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *ICML*, 697–704. ACM.

Rastogi, R., Sharma, S., and Chandra, S. (2017). Robust parametric twin support vector machine for pattern classification. *Neural Processing Letters*, 1–31.

Raytchev, B., Yoda, I., and Sakaue, K. (2004). Head pose estimation by nonlinear manifold learning. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Volume 4, 462–466. IEEE.

Reyzin, L. and Schapire, R. E. (2006). How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd international conference on Machine learning*, 753–760. ACM.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, *22*, 400–407.

Robert, C. and Casella, G. (2013). *Monte Carlo statistical methods*. Springer Science & Business Media.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, *33*(1-2), 1–39.

Ryan, R. M. and Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, *25*(1), 54–67.

Salge, C., Glackin, C., and Polani, D. (2014). Empowerment–an introduction. In *Guided Self-Organization: Inception*, 67–114. Springer.

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, *5*(2), 197–227.

Schapire, R. E., Freund, Y., Bartlett, P., Lee, W. S., *et al.* (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, *26*(5), 1651–1686.

Schmidhuber, J. (1991a). Curious model-building control systems. In *Proc. international joint conference on neural networks*, 1458–1463.

Schmidhuber, J. (1991b). A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, 222–227. MIT Press.

Schmidhuber, J. (2006). Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, *18*(2), 173–187.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, *2*(3), 230–247.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *ICML*, 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Y. Bengio and Y. LeCun (Eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Selmic, R. R. and Lewis, F. L. (2002). Neural-network approximation of piecewise continuous functions: application to friction compensation. *IEEE transactions on neural networks*, *13*(3), 745–751.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, *27*(3), 379–423.

Shao, Y.-H., Chen, W.-J., Wang, Z., Li, C.-N., and Deng, N.-Y. (2015). Weighted linear loss twin support vector machine for large-scale classification. *Knowledge-Based Systems*, *73*, 276–288.

Shao, Y.-H., Zhang, C.-H., Wang, X.-B., and Deng, N.-Y. (2011). Improvements on twin support vector machines. *Neural Networks, IEEE Transactions on*, *22*(6), 962–968.

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2019). Dynamics-Aware Unsupervised Discovery of Skills. In *International Conference on Learning Representations*.

Shridhar, K., Laumann, F., and Liwicki, M. (2019). A comprehensive guide to bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., *et al.* (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

Silvia, P. J. (2012). Curiosity and motivation. *The Oxford handbook of human motivation*, 157–166.

Sollich, P. and Krogh, A. (1996). Learning with ensembles: How overfitting can be useful. In *Advances in neural information processing systems*, 190–196.

Storck, J., Hochreiter, S., and Schmidhuber, J. (1995). Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the International Conference on Artificial Neural Networks*, Volume 2, 159–164.

Sun, Y., Gomez, F., and Schmidhuber, J. (2011). Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *International Conference on Artificial General Intelligence*, 41–51. Springer Berlin Heidelberg.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1-2), 181–211.

Tang, Y. and Agrawal, S. (2018). Boosting trust region policy optimization by normalizing flows policy. *arXiv preprint arXiv:1809.10326*.

Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*(5500), 2319–2323.

The Mathworks, Inc. (2017). *MATLAB version 9.2.0.538062 (R2017a)*. Natick, Massachusetts: The Mathworks, Inc.

Traibus, M. (1961). *Thermostatics and Thermodynamics: An Introduction to Energy, Information and States of Matter, with Engineering Applications*. New York: Van Nostrand.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In D. Schuurmans and M. P. Wellman (Eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 2094–2100. AAAI Press.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer Science & Business Media.

Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, Volume 1. Wiley New York.

Wainwright, M. J., Jordan, M. I., *et al.* (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, *1*(1–2), 1–305.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Wang, L., Sugiyama, M., Yang, C., Zhou, Z., and Feng, J. (2008). On the Margin Explanation of Boosting Algorithms. In R. A. Servedio and T. Zhang (Eds.), *21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, July 9-12, 2008*, 479–490. Omnipress.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, 1995–2003.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

Wei, H., Zheng, G., Yao, H., and Li, Z. (2018). Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2496–2505.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 681–688.

Wilcoxon, F. (1946). Individual comparisons of grouped data by ranking methods. *Journal of economic entomology*, *39*(2), 269–270.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Wydmuch, M., Kempka, M., and Jaśkowski, W. (2018). Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, *11*(3), 248–259.

Xu, H., Fan, L., and Gao, X. (2015). Projection twin SMMs for 2d image data classification. *Neural Computing and Applications*, *26*(1), 91–100.

Xu, H., McCane, B., and Szymanski, L. (2018). Twin Bounded Large Margin Distribution Machine. In *Australasian Joint Conference on Artificial Intelligence*, 718–729. Springer.

Xu, H., McCane, B., and Szymanski, L. (2019). VASE: Variational Assorted Surprise Exploration for Reinforcement Learning. *arXiv preprint arXiv:1910.14351*.

Xu, H., McCane, B., Szymanski, L., and Atkinson, C. (2020). MIME: Mutual Information Minimisation Exploration. *arXiv preprint arXiv:2001.05636*.

Xu, Y., Pan, X., Zhou, Z., Yang, Z., and Zhang, Y. (2015). Structural least square twin support vector machine for classification. *Applied Intelligence*, *42*(3), 527–536.

Yang, Z., Xie, Y., and Wang, Z. (2019). A theoretical analysis of deep Q-learning. *arXiv preprint arXiv:1901.00137*.

Zhang, K. and Kwok, J. T. (2010). Clustered Nyström method for large scale manifold learning and dimension reduction. *IEEE Transactions on Neural Networks*, *21*(10), 1576–1587.

Zhang, K., Yang, Z., and Başar, T. (2019). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *arXiv preprint arXiv:1911.10635*.

Zhang, T. and Zhou, Z.-H. (2014). Large margin distribution machine. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 313–322. ACM.

Zhang, T. and Zhou, Z.-H. (2018). Optimal Margin Distribution Clustering.

Zhao, G., Liu, F., Oler, J. A., Meyerand, M. E., Kalin, N. H., and Birn, R. M. (2018). Bayesian convolutional neural network based MRI brain extraction on nonhuman primates. *Neuroimage*, *175*, 32–44.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., and Li, Z. (2018). DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, 167–176.

Zhou, Z., Li, X., and Zare, R. N. (2017). Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, *3*(12), 1337–1344.

Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms*. CRC press.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, 3357–3364. IEEE.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Appendix A

Twin Bounded Large Margin Distribution Machine

Note: Some portions of this chapter are taken from my own work (Xu, McCane, and Szymanski, 2018).

In this chapter, I will introduce a classifier named twin bounded large margin distribution machine (TBLDM). The central idea of TBLDM is to seek a pair of nonparallel hyperplanes by optimizing the positive and negative margin distributions on the base of TBSVM (Shao, Zhang, Wang, and Deng, 2011). The experimental results indicate that the proposed TBLDM is a fast, effective and robust classifier.

A.1 Introduction

Support vector machines (SVMs) (Vapnik and Vapnik, 1998; Cortes and Vapnik, 1995) are powerful tools for pattern classification and regression. For the classical binary classification SVM, the optimal hyperplane can be obtained by maximizing a relaxed minimum margin, i.e., the smallest distance from data point to the classification boundary. This optimisation can be expressed as a quadratic programming problem (QPP). Margin theory (Vapnik, 2013) provides good theoretical support to the generalisation performance of SVMs and it has also been applied to many other machine learning approaches, such as AdaBoost (Freund and Schapire, 1997). There was, however, a long debate on whether margin theory plays a significant role in AdaBoost (Schapire, Freund, Bartlett, Lee, *et al.*, 1998; Breiman, 1999). It had been believed that a single-data-point margin such as minimum margin is not crucial (Reyzin and Schapire, 2006; Wang, Sugiyama, Yang, Zhou, and Feng, 2008). Gao and Zhou (2013) ended the long

debate and showed that margin distribution, characterized by margin mean and variance, is critical for generalisation in boosting. Inspired by these results, Zhang and Zhou (2014) first focused on the influence of the margin distribution for SVMs and proposed large margin distribution machine (LDM). The margin distribution heuristic can also be applied to clustering (Zhang and Zhou, 2018) and dimensionality reduction (Luo and Durrant, 2017).

The twin support vector machine (TWSVM) proposed by Khemchandani, Chandra, *et al.* (2007) seeks for two nonparallel boundary hyperplanes and attempts to make each of the two hyperplanes close to one class and far from the other as much as possible. TWSVM solves two smaller size QPPs instead of a single large QPP. This results in TWSVM being faster than SVM. An improved version of TWSVM, called twin bounded support vector machine (TBSVM) was proposed by Shao *et al.* (2011). TBSVM implemented the structural risk minimisation principle by introducing a regularization term. Based on statistical learning theory, TBSVM can improve the performance of classification of TWSVM. Recently, many extensions of TWSVM have been proposed, for details, see (Shao, Chen, Wang, Li, and Deng, 2015; Khemchandani and Sharma, 2016; Rastogi, Sharma, and Chandra, 2017; Xu, Pan, Zhou, Yang, and Zhang, 2015; Xu, Fan, and Gao, 2015).

In this chapter, we propose the twin bounded large margin distribution machine (TBLDM). Similar to LDM, the margin distribution of TBLDM is characterised by first and second order statistics and optimizing the margin distribution is realized by maximizing the margin mean and minimizing the margin variance simultaneously. However, TBLDM tries to optimise the positive and negative margin distributions separately. This is different from LDM, which optimised the whole margin distribution for all training points.

To begin with, we will first provide a brief background on SVM, TWSVM and LDM in Section 2. Our novel approach TBLDM for classification problems will be introduced in Section 3. In Section 4, we will make numerical experiments to verify that our new model is very effective in classification. Discussions and conclusions will be summarized in Section 5.

A.2 Notation and related work

Given the dataset $T = \{(x_i, y_i)\}_{i=1}^{l}$, where $x_i \in R^n$ is the i-th input sample and $y_i \in \{\pm 1\}$ is the class label of x_i. Let l_1 and l_2 be the numbers of samples belong-

ing to the positive and negative classes, respectively, such that $l = l_1 + l_2$. Denote $X = [x_1, \cdots, x_l] \in R^{n \times l}, A = [x_1^+, \cdots, x_{l_1}^+] \in R^{n \times l_1}$ and $B = [x_1^-, \cdots, x_{l_2}^-] \in R^{n \times l_2}$ as the entire, positive and negative sample matrices. Let $k : R^n \times R^n \to R$ be a kernel function with reproducing kernel Hilbert space (RKHS) $\widetilde{H}$ and nonlinear feature mapping $\phi : R^n \to \widetilde{H}$. Denote $\phi(A) = [\phi(x_1^+), \cdots, \phi(x_{l_1}^+)], \phi(B) = [\phi(x_1^-), \cdots, \phi(x_{l_2}^-)]$ as the positive and negative mapped sample matrices, the kernel matrix $K = \phi(X)^T \phi(X)$ where $\phi(X) = [\phi(x_1), \cdots, \phi(x_l)]$, $K_A = \phi(A)^T \phi(X) \in R^{l_1 \times l}, K_B = \phi(B)^T \phi(X) \in R^{l_2 \times l}, K(x, X) = [k(x, x_1), \cdots, k(x, x_l)] \in R^{1 \times l}$, $\forall x \in R^n$. and $y = (y_1, \cdots, y_l)^T \in R^l$. $y_A = (y_1^+, \cdots, y_{l_1}^+)^T \in R^{l_1}, y_B = (y_1^-, \cdots, y_{l_2}^-)^T \in R^{l_2}$.

A.2.1 Support vector machine (SVM)

SVM tries to find a hyperplane $f(x) = w^T \phi(x) = 0$, where f is linear and $w \in \widetilde{H}$ is a linear predictor. According to (Cortes and Vapnik, 1995) and (Vapnik, 2013), the margin of the individual sample (x_i, y_i) is defined as

$$\gamma_i = y_i w^T \phi(x_i), i = 1, \cdots, l. \tag{A.1}$$

In separable cases, all the γ_i will be non-negative. So we can get the geometric distance from each x_i to $w^T \phi(x) = 0$ by scaling each γ_i with $1/\|w\|$:

$$\hat{\gamma}_i = y_i \frac{w^T}{\|w\|} \phi(x_i), i = 1, \cdots, l.$$

For the separable case, SVM maximizes the minimum distance:

$$\begin{aligned} \max_w \quad & \hat{\gamma} \\ s.t. \quad & \hat{\gamma}_i \geq \hat{\gamma}, i = 1, \cdots, l. \end{aligned}$$

It can be written as

$$\begin{aligned} \max_w \quad & \frac{\gamma}{\|w\|} \\ s.t. \quad & \gamma_i \geq \gamma, i = 1, \cdots, l. \end{aligned}$$

We can simply set γ as 1 since it doesn't have influence on the optimization. Note that maximizing $1/\|w\|$ is equivalent to minimizing $\|w\|^2$, we can get the classic formulation of hard-margin SVM as follows:

$$\begin{aligned} \min_w \quad & \frac{1}{2}\|w\|^2 \\ s.t. \quad & y_i w^T \phi(x_i) \geq 1, i = 1, \cdots, l. \end{aligned}$$

For non-separable case, SVM can be written as

$$\begin{aligned} \max_{w,\xi_i} \quad & \gamma_0 - \bar{C}\sum_{i=1}^{l}\xi_i \\ s.t. \quad & \gamma_i \geq \gamma_0 - \xi_i, \\ & \xi_i \geq 0, i = 1, \cdots, l, \end{aligned}$$

where γ_0 is a relaxed minimum margin, ξ_i is slack variable and $\bar{C}$ is the trading-off parameter. The above formula can be rewritten as

$$\begin{aligned} \min_{w,\xi_i} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \\ s.t. \quad & y_i w^T\phi(x_i) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \cdots, l, \end{aligned}$$

where C is a trading-off parameter. We can see that SVMs for both separable and non-separable cases consider only single-data-point margins but not the whole margin distribution.

A.2.2 Twin bounded support vector machine (TBSVM)

Different from conventional SVM, TWSVM seeks for a pair of nonparallel hyperplanes $f_+(x) = w_+^T\phi(x) = 0$ and $f_-(x) = w_-^T\phi(x) = 0$. As an improved version of TWSVM, TBSVM consider the structural risk minimization principle by adding a regularization term. The training time of TBSVM is approximately four times faster than SVM. We introduce non-linear TBSVM in this subsection, for linear case and other details, see (Khemchandani *et al.*, 2007; Shao *et al.*, 2011). The unknown vectors $w_+, w_- \in R^n$ of TBSVM can be obtained by solving the following two QPPs:

$$\begin{aligned} \min_{w_+,\xi_2} \quad & \frac{c_1}{2}||w_+||^2 + \frac{1}{2}||\phi(A)^T w_+||^2 + c_3 e_2^T\xi_2 \\ s.t. \quad & -\phi(B)^T w_+ + \xi_2 \geq e_2,\ \xi_2 \geq 0, \end{aligned} \tag{A.2}$$

$$\begin{aligned} \min_{w_-,\xi_1} \quad & \frac{c_2}{2}||w_-||^2 + \frac{1}{2}||\phi(B)^T w_-||^2 + c_4 e_1^T\xi_1 \\ s.t. \quad & \phi(A)^T w_- + \xi_1 \geq e_1,\ \xi_1 \geq 0, \end{aligned} \tag{A.3}$$

where $c_1, \cdots, c_4 > 0$ are trade-off parameters, $\xi_1 \in R^{l_1}, \xi_2 \in R^{l_2}$ are slack variable vectors and $e_1 \in R^{l_1}, e_2 \in R^{l_2}$ are vectors of ones. A new input $\widetilde{x} \in R^n$ is assigned the

class k depending on which of the two hyperplanes it is closer to. That is, the class label $y_{\widetilde{x}}$ can be obtained by $y_{\widetilde{x}} = \arg\min_{k=\pm} \frac{|f_k(\widetilde{x})|}{||w_k||}$.

Similar to the definition of the margin of individual sample in (A.1), the positive and negative margin of individual sample can be formulated as

$$\gamma_j^+ = y_j^+ f_-(x_j^+) = y_j^+ w_-^T \phi(x_j^+), j = 1, \cdots, l_1, \tag{A.4}$$

$$\gamma_j^- = y_j^- f_+(x_j^-) = y_j^- w_+^T \phi(x_j^-), j = 1, \cdots, l_2, \tag{A.5}$$

respectively. We can see that TBSVM tries to maximize the minimal negative margin between the negative samples and positive decision hyperplane by (A.2) and maximize the minimal positive margin by (A.3).

A.2.3 Large margin distribution machine (LDM)

LDM tries to achieve a strong generalization performance by optimizing the margin distribution of samples on the basis of soft-margin SVM. The margin distribution is characterized by first- and second-order statistics. Optimizing margin distribution is realized by maximizing the margin mean and minimizing the margin variance simultaneously. Based on (A.1), the margin mean $\bar{\gamma}$ and the margin variance $\hat{\gamma}$ can be calculated by $\bar{\gamma} = \frac{1}{l}\sum_{i=1}^{l} \gamma_i$ and $\hat{\gamma} = \frac{1}{l}\sum_{i=1}^{l} (\gamma_i - \bar{\gamma})^2$. The unknown $w \in \widetilde{H}$ can be obtained by solving the following optimization problem:

$$\begin{aligned} \min_{w,\xi_i} \quad & \frac{1}{2} w^T w + \lambda_1 \hat{\gamma} - \lambda_2 \bar{\gamma} + C \sum_{i=1}^{l} \xi_i \\ s.t. \quad & y_i w^T \phi(x_i) \geq 1 - \xi_i,\ \xi_i \geq 0, i = 1, \cdots, l, \end{aligned}$$

where $\lambda_1, \lambda_2 > 0$ are the parameters for trading-off the margin variance, the margin mean and the model complexity. It is obvious that LDM can be reduced to soft-margin SVM when $\lambda_1 = \lambda_2 = 0$.

A.3 Twin bounded large margin distribution machine (TBLDM)

In this section, we will introduce our novel classification method named as twin bounded large margin distribution machine (TBLDM). Based on the concepts of positive margin and negative margin in (A.4) and (A.5), the positive margin mean $\bar{\gamma}^+$ and the

positive margin variance $\hat{\gamma}^+$ can be calculated by $\bar{\gamma}^+ = \frac{1}{l_1}\sum_{j=1}^{l_1}\gamma_j^+ = \frac{1}{l_1}y_A^T\phi(A)^T w_-$, and $\hat{\gamma}^+ = \frac{1}{l_1}\sum_{i=1}^{l_1}(\gamma_i^+ - \bar{\gamma}^+)^2 = w_-^T\phi(A)Q_1\phi(A)^T w_-$ respectively. Here $Q_1 = \frac{l_1 I_{l_1} - y_A y_A^T}{l_1^2}$ is a symmetric matrix. Since $Q_1^2 = \frac{1}{l_1}Q_1$, it can be concluded that Q_1 is a symmetric nonnegative definite matrix. Similarly, we can get the negative margin mean $\bar{\gamma}^-$ and the negative margin variance $\hat{\gamma}^-$ by $\bar{\gamma}^- = \frac{1}{l_2}y_B^T\phi(B)^T w_+, \hat{\gamma}^- = w_+^T\phi(B)Q_2\phi(B)^T w_+$, where $Q_2 = \frac{l_2 I_{l_2} - y_B y_B^T}{l_2^2}$ is also a symmetric nonnegative definite matrix.

A.3.1 TBLDM

Specifically, TBLDM seeks a pair of unknown vectors $w_+, w_- \in \widetilde{H}$ by maximizing the positive and negative margin mean and minimizing the positive and negative margin variance simultaneously, that is, by considering the following two optimization problems:

$$\begin{aligned} \min_{w_+,\xi_2} \quad & \frac{c_1}{2}\|w_+\|^2 + \frac{1}{2}\|\phi(A)^T w_+\|^2 - \lambda_1\bar{\gamma}^- + \lambda_2\hat{\gamma}^- + c_3 e_2^T\xi_2 \\ s.t. \quad & -\phi(B)^T w_+ + \xi_2 \geq e_2,\ \xi_2 \geq 0, \end{aligned} \tag{A.6}$$

$$\begin{aligned} \min_{w_-,\xi_1} \quad & \frac{c_2}{2}\|w_-\|^2 + \frac{1}{2}\|\phi(B)^T w_-\|^2 - \lambda_3\bar{\gamma}^+ + \lambda_4\hat{\gamma}^+ + c_4 e_1^T\xi_1 \\ s.t. \quad & \phi(A)^T w_- + \xi_1 \geq e_1,\ \xi_1 \geq 0, \end{aligned} \tag{A.7}$$

where $\lambda_1, \cdots, \lambda_4 > 0$ are the parameters for trading-off the margin variances, the margin means and the complexity of models. It is obvious that TBLDM can be reduced to the nonlinear TBSVM when $\lambda_1, \lambda_2, \lambda_3$ and λ_4 are equal to 0.

Substituting $\bar{\gamma}^+$, $\bar{\gamma}^-$ and $\hat{\gamma}^+$, $\hat{\gamma}^-$ into the models (A.6) and (A.7) , we can get the following two QPPs :

$$\begin{aligned} \min_{w_+,\xi_2} \quad & \frac{c_1}{2}\|w_+\|^2 + \frac{1}{2}\|\phi(A)^T w_+\|^2 - \frac{\lambda_1}{l_2}y_B^T\phi(B)^T w_+ + \lambda_2 w_+^T\phi(B)Q_2\phi(B)^T w_+ + c_3 e_2^T\xi_2 \\ s.t. \quad & -\phi(B)^T w_+ + \xi_2 \geq e_2,\ \xi_2 \geq 0, \end{aligned} \tag{A.8}$$

$$\begin{aligned} \min_{w_-,\xi_1} \quad & \frac{c_2}{2}\|w_-\|^2 + \frac{1}{2}\|\phi(B)^T w_-\|^2 - \frac{\lambda_3}{l_1}y_A^T\phi(A)^T w_- + \lambda_4 w_-^T\phi(A)Q_1\phi(A)^T w_- + c_4 e_1^T\xi_1 \\ s.t. \quad & \phi(A)^T w_- + \xi_1 \geq e_1,\ \xi_1 \geq 0, \end{aligned} \tag{A.9}$$

Due to $\widetilde{H} = \text{span}\{\phi(x_1), \cdots, \phi(x_l)\}$, we can let $w_+ = \phi(X)\beta_1$ and $w_- = \phi(X)\beta_2$, where

$\beta_1, \beta_2 \in R^l$ are coefficient vectors, and then we can deduce that

$$\begin{aligned} \|w_+\|^2 = \beta_1^T K \beta_1, \qquad & |w_-\|^2 = \beta_2^T K \beta_2, \\ \phi(A)^T w_+ = K_A \beta_1, \qquad & \phi(B)^T w_+ = K_B \beta_1, \\ \phi(A)^T w_- = K_A \beta_2, \qquad & \phi(B)^T w_- = K_B \beta_2, \\ f_+(x) = w_+^T \phi(x) = K(x, X)\beta_1, \quad & f_-(x) = w_-^T \phi(x) = K(x, X)\beta_2. \end{aligned} \tag{A.10}$$

Substituting (A.10) into the models (A.8), we have

$$\begin{aligned} \min_{\beta_1, \xi_2} \quad & \frac{c_1}{2}\beta_1^T K \beta_1 + \frac{1}{2}\beta_1^T K_A^T K_A \beta_1 - \frac{\lambda_1}{l_2} y_B^T K_B \beta_1 + \lambda_2 \beta_1^T K_B^T Q_2 K_B \beta_1 + c_3 e_2^T \xi_2 \\ s.t. \quad & -K_B \beta_1 + \xi_2 \geq e_2, \xi_2 \geq 0, \end{aligned} \tag{A.11}$$

$$\begin{aligned} \min_{\beta_2, \xi_1} \quad & \frac{c_2}{2}\beta_2^T K \beta_2 + \frac{1}{2}\beta_2^T K_B^T K_B \beta_2 - \frac{\lambda_3}{l_1} y_A^T K_A \beta_2 + \lambda_4 \beta_2^T K_A^T Q_1 K_A \beta_2 + c_4 e_1^T \xi_1 \\ s.t. \quad & K_A \beta_2 + \xi_1 \geq e_1, \xi_1 \geq 0. \end{aligned} \tag{A.12}$$

Let

$$\begin{aligned} G_1 &= c_1 K + K_A^T K_A + 2\lambda_2 K_B^T Q_2 K_B \in R^{l \times l}, \\ G_2 &= c_2 K + K_B^T K_B + 2\lambda_4 K_A^T Q_1 K_A \in R^{l \times l}. \end{aligned}$$

Obviously, G_1 and G_2 are symmetric nonnegative definite matrices. The models (A.11) and (A.12) can be rewritten as

$$\begin{aligned} \min_{\beta_1, \xi_2} \quad & \frac{1}{2}\beta_1^T G_1 \beta_1 - \frac{\lambda_1}{l_2} y_B^T K_B \beta_1 + c_3 e_2^T \xi_2 \\ s.t. \quad & -K_B \beta_1 + \xi_2 \geq e_2, \xi_2 \geq 0, \end{aligned} \tag{A.13}$$

$$\begin{aligned} \min_{\beta_2, \xi_1} \quad & \frac{1}{2}\beta_2^T G_2 \beta_2 - \frac{\lambda_3}{l_1} y_A^T K_A \beta_2 + c_4 e_1^T \xi_1 \\ s.t. \quad & K_A \beta_2 + \xi_1 \geq e_1, \xi_1 \geq 0. \end{aligned} \tag{A.14}$$

Considering the Lagrangian function of the model (A.13)

$$L_1(\beta_1, \xi_2, \alpha_1, \delta_1) = \frac{1}{2}\beta_1^T G_1 \beta_1 - \frac{\lambda_1}{l_2} y_B^T K_B \beta_1 + c_3 e_2^T \xi_2 - \alpha_1^T(-K_B \beta_1 + \xi_2 - e_2) - \delta_1^T \xi_2,$$

where $\alpha_1, \delta_1 \in R^{l_2}$ are nonnegative Lagrangian multipliers vectors, and letting $\partial L_1/\partial \beta_1 = \partial L_1/\partial \xi_2 = 0$, we get

$$\begin{aligned} G_1 \beta_1 &= \frac{\lambda_1}{l_2} K_B^T y_B - K_B^T \alpha_1, \\ c_3 e_2 - \alpha_1 - \delta_1 = 0 &\Rightarrow 0 \leq \alpha_1 \leq c_3 e_2. \end{aligned} \tag{A.15}$$

Without loss of generality, we can assume that G_1 is an invertible matrix; otherwise, it can be regularized, that is, it can be replaced by the matrix $G_1 + t_1 I_l$, where $t_1 > 0$ is a small positive number called regularized coefficient. Consequently, it can be deduced from (A.15) that

$$\beta_1 = G_1^{-1}(\frac{\lambda_1}{l_2} K_B^T y_B - K_B^T \alpha_1). \tag{A.16}$$

Submitting (A.16) and (A.15) into the Lagrangian function, we can obtain the Wolfe dual form of the model (A.13):

$$\begin{aligned} \min_{\alpha_1} \quad & \frac{1}{2}\alpha_1^T H_1 \alpha_1 - (\frac{\lambda_1}{l_2} H_1 y_B + e_2)^T \alpha_1 \\ s.t. \quad & 0 \le \alpha_1 \le c_3 e_2, \end{aligned} \tag{A.17}$$

where $H_1 = K_B G_1^{-1} K_B^T$. Similarly, we can get

$$\beta_2 = G_2^{-1}(\frac{\lambda_1}{l_1} K_A^T y_A + K_A^T \alpha_2), \tag{A.18}$$

and then the Wolfe dual form of the model (A.7) is:

$$\begin{aligned} \min_{\alpha_2} \quad & \frac{1}{2}\alpha_2^T H_2 \alpha_2 + (\frac{\lambda_3}{l_1} H_2 y_A - e_1)^T \alpha_2 \\ s.t. \quad & 0 \le \alpha_2 \le c_4 e_1, \end{aligned} \tag{A.19}$$

where $\alpha_2 \in R^{l_1}$ is a nonnegative Lagrangian multipliers vector and $H_2 = K_A G_2^{-1} K_A^T$. A new input $\widetilde{x} \in R^n$ is assigned the class i ($i = 1, 2$ denotes the positive and negative classes, respectively) depending on which of the two hyperplanes is closer to, that is, label $(\widetilde{x}) = \arg\min_{i=1,2} \frac{|K(\widetilde{x}, X)\beta_i|}{\sqrt{\beta_i K \beta_i}}$. The specific procedure is listed in Algorithm 1.

Algorithm 8: TBLDM

Input: Training set T, testing sample $\widetilde{x}$, kernel function $k : R^n \times R^n \to R$, model parameters $\lambda_i, \cdots, \lambda_4$ and $c_i, \cdots, c_4$, regularized parameters t_1, t_2 and kernel parameters;

1: Solve the QPP (A.17) and obtain the optimal solution α_1^*;
2: Compute β_1^* by (A.16) with $\alpha_1 = \alpha_1^*$;
3: Solve the QPP (A.19) and obtain the optimal solution α_2^*;
4: Compute β_2^* by (A.18) with $\alpha_2 = \alpha_2^*$;
5: For $\widetilde{x}$, predict its label by label $(\widetilde{x}) = \arg\min_{i=1,2} \frac{|K(\widetilde{x}, X)\beta_i^*|}{\sqrt{\beta_i^* K \beta_i^*}}$.

A.3.2 TBLDM for large scale datasets

It can be seen that we need to compute G_1^{-1} and G_2^{-1} and kernel matrix K, K_A, K_B before solving the dual problems (A.17) and (A.19). This is infeasible when the number of samples is significantly large both in terms of memory and computation. To effectively handle large scale problems, in this subsection, we first choose a kernel approximation method, Nyström method (Zhang and Kwok, 2010) to explicitly map features onto subspaces in the RKHS. In this case, the embedding features are obtained without constructing the complete kernel matrix for the data set. Given the kernel-specific embedding, we perform linear TBLDM. Because the inverse matrices of AA^T and BB^T still need to be computed to get the dual problem of linear TBLDM, we solve the primal problem of linear TBLDM here with stochastic gradient descent (SGD) algorithm.

Linear TBLDM is a special case of TBLDM with linear kernel function $k(u, v) = \langle u, v\rangle$ for any $u, v \in R^n$. In this case, the models (A.6) and (A.7) are reduced into the following two QPPs:

$$\begin{aligned} \min_{w_+,\xi_2} \quad & \frac{c_1}{2}\|w_+\|^2 + \frac{1}{2}\|A^T w_+\|^2 - \frac{\lambda_1}{l_2} y_B^T B^T w_+ + \lambda_2 w_+^T B Q_2 B^T w_+ + c_3 e_2^T \xi_2 \\ s.t. \quad & -B^T w_+ + \xi_2 \geq e_2,\ \xi_2 \geq 0, \end{aligned} \tag{A.20}$$

$$\begin{aligned} \min_{w_-,\xi_1} \quad & \frac{c_2}{2}\|w_-\|^2 + \frac{1}{2}\|B^T w_-\|^2 - \frac{\lambda_3}{l_1} y_A^T A^T w_- + \lambda_4 w_-^T A Q_1 A^T w_- + c_4 e_1^T \xi_1 \\ s.t. \quad & A^T w_- + \xi_1 \geq e_1,\ \xi_1 \geq 0. \end{aligned} \tag{A.21}$$

To solve formulas (A.20) and (A.21) in primal case, we express them equivalently as two unconstraint optimization problems:

$$\begin{aligned} \min_{w_+,\xi_2} g_1(w_+) = \quad & \frac{c_1}{2}\|w_+\|^2 + \frac{1}{2}\|A^T w_+\|^2 - \frac{\lambda_1}{l_2} y_B^T B^T w_+ + \lambda_2 w_+^T B Q_2 B^T w_+ \\ & + c_3 \sum_{i=1}^{l_2} \max\{0, 1 + w_+^T x_i^-\}, \end{aligned} \tag{A.22}$$

$$\begin{aligned} \min_{w_-,\xi_1} g_2(w_-) = \quad & \frac{c_2}{2}\|w_-\|^2 + \frac{1}{2}\|B^T w_-\|^2 - \frac{\lambda_3}{l_1} y_A^T A^T w_- + \lambda_4 w_-^T A Q_1 A^T w_- + \\ & c_4 \sum_{i=1}^{l_1} \max\{0, 1 - w_-^T x_i^+\}. \end{aligned} \tag{A.23}$$

If examples (x_i^+, y_i^+), (x_j^+, y_j^+), (x_k^+, y_k^+) are randomly sampled from the positive training set and (x_i^-, y_i^-), (x_j^-, y_j^-), (x_k^-, y_k^-) are randomly sampled from the negative training

set independently, it is straightforward to prove that

$$\begin{aligned}\nabla g_1(w_+, x_i^+, x_j^-, x_k^-) = \; & c_1 w_+ + l_1 x_i^+ {x_i^+}^T w_+ + 2\lambda_2 x_j^- {x_j^-}^T w_+ - 2\lambda_2 x_j^- {x_k^-}^T w_+ \\ & + \lambda_1 x_j^- + c_3 l_2 x_j^- \mathbb{I}(j \in I_1), \end{aligned} \tag{A.24}$$

$$\begin{aligned}\nabla g_2(w_-, x_i^-, x_j^+, x_k^+) = \; & c_2 w_- + l_2 x_i^- {x_i^-}^T w_- + 2\lambda_4 x_j^+ {x_j^+}^T w_- - 2\lambda_4 x_j^+ {x_k^+}^T w_- \\ & - \lambda_3 x_j^+ - c_4 l_1 x_j^+ \mathbb{I}(j \in I_2). \end{aligned} \tag{A.25}$$

are the unbiased estimation of $\nabla g_1(w_+)$ and $\nabla g_2(w_-)$ respectively. $\mathbb{I}(\cdot)$ is the indicator function that returns 1 when the argument holds, and 0 otherwise. I_1, I_2 are the index sets defined as $I_1 = \{j | w_+^T x_j^- > -1\}, I_2 = \{j | w_-^T x_j^+ < 1\}$. So we can update w_+, w_- by $w_+ \leftarrow w_+ - r_1 \nabla g_1(w_+, x_i^+, x_j^-, x_k^-)$ and $w_- \leftarrow w_- - r_2 \nabla g_2(w_-, x_i^-, x_j^+, x_k^+)$, r_1, r_2 are learning rates for each iteration of SGD algorithm.

The detailed procedure is listed in Algorithm 2.

Algorithm 9: Nyström + linear TBLDM for large scale problems

Input: Positive training set A, negative training set B, testing sample $\widetilde{x}$, model parameters $\lambda_1, \cdots, \lambda_4$ and $c_1, \cdots, c_4$ and learning rates r_1, r_2;

Get data embedding A_e, B_e and $\widetilde{x}_e$ by Nyström method;

while w_+, w_- *not converged* **do**

  Randomly select mini-batch $x_b^+ = \{x_i^+, x_j^+, x_k^+\}$ and $x_b^- = \{x_i^-, x_j^-, x_k^-\}$;

  for $x_b^+ \subset A_e$ *and* $x_b^- \subset B_e$ **do**

    Compute the gradient $\nabla g_1(w_+, x_i^+, x_j^-, x_k^-)$ by (A.24);

    Compute the gradient $\nabla g_2(w_-, x_i^-, x_j^+, x_k^+)$ by (A.25);

    $w_+ \leftarrow w_+ - r_1 \nabla g_1(w_+, x_i^+, x_j^-, x_k^-)$;

    $w_- \leftarrow w_- - r_2 \nabla g_2(w_-, x_i^-, x_j^+, x_k^+)$;

For $\widetilde{x}$, predict its label by label $(\widetilde{x}) = \arg\min_{i=\pm} \frac{|w_i^T \widetilde{x_e}|}{\|w_i\|}$.

A.4 Experiments and results analysis

In order to demonstrate the effectiveness of TBLDM, a series of comparative experiments with SVM, TBSVM and LDM are performed. The experiments focus on the aspects of classification accuracy and computational time on sixteen regular scale datasets and four large-scale datasets. These datasets are taken from UCI database (Dua and Graff, 2017) and real-world databases[1], respectively. All the computational time in-

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

volved is the sum of the training time and the testing time and all the classification accuracy involved is the testing accuracy, that is, the classification accuracy on testing sets.

A.4.1 Experiments on regular-scale datasets

The statistics of the regular-scale datasets are listed in the first four rows in Table A.1, where l and n denote the number and the dimensionality of samples, respectively. Gaussian radial basis function (RBF) kernel $k(u, v) = \exp(-\|u - v\|^2/\gamma)$ for $u, v \in R^n$

Table A.1: Statistics of datasets

Data set	l	n	Data set	l	n	Data set	l	n	Data set	l	n
australian	690	14	parkinsons	195	22	bupa	345	6	ringnorm	400	20
ecoli	336	7	sonar	208	60	german	1000	24	spect	80	22
haberman	306	3	transfusion	748	4	heart	270	13	twonorm	400	20
ionosphere	351	34	wdbc	569	30	monks2	432	6	wpbc	198	32
cod-rna	216948	8	ijcnn1	141691	22	skin	245057	3	w8a	64700	300

is selected and SMO (Platt *et al.*, 1998) algorithm is used for SVM, where $\gamma > 0$ is a kernel parameter. We use SOR solver (Mangasarian and Musicant, 1999) for fast training TBSVM; the source code of Zhang and Zhou (2014) for LDM; for TBLDM, the 'quadprog' toolbox in MATLAB (The Mathworks, Inc., 2017) is used to solve QPPs (A.17) and (A.19). All the experiments are operated in MATLAB. For the convenience of computation, we take all the model parameters $C, c_1, c_2, c_3, c_4 = 1$, the kernel parameter γ and λ_1, λ_2 are chosen from $[2^{-6}, 2^6]$ by using 5-fold cross validation method. Experiments are repeated for 5 times with random data partitions to calculate the average accuracies and variances. The experimental results are listed in Table A.2, from which we can see that for computational time, TBLDM is obviously faster than LDM except on spect and wpbc datasets, and faster than TBSVM on 12 datasets and similar on the remaining 4 datasets. For classification accuracy, TBLDM is higher than LDM on 11 datasets and same on wdbc data set, and is higher than TBSVM on 13 datasets. In addition, SVM only gets the highest classification accuracy on wdbc data set although its computational time is the fastest.

A.4.2 Experiments on large-scale datasets

The statistics of the large-scale datasets are listed in the last row of Table A.1. All of these four large-scale datasets are split into training and test parts. To compare with

Table A.2: Experimental results with regular size datasets

	SVM		TBSVM		LDM		TBLDM	
DATASETS	acc(mean±std)	time(s)	acc(mean±std)	time(s)	acc(mean±std)	time(s)	acc(mean±std)	time(s)
australian	0.8565±0.0262	0.0353	0.8574±0.0360	0.2015	0.8557±0.0258	1.4757	**0.8672±0.0194**	0.1500
bupa	0.6736±0.0591	0.0292	0.6986±0.0495	0.0818	0.6980±0.0542	0.1893	**0.7014±0.0366**	0.0820
ecoli	0.9637±0.0264	0.0189	0.9648±0.0331	0.0743	**0.9672±0.0232**	0.1785	0.9637±0.0175	0.0939
german	0.7224±0.0367	0.0611	0.7510±0.0211	0.5345	0.7590±0.0186	4.9356	**0.7724±0.0223**	0.4213
haberman	0.7333±0.0235	0.0241	0.7210±0.0203	0.1320	0.7353±0.0344	0.1337	**0.7380±0.0396**	0.0805
heart	0.8333±0.0367	0.0220	0.8356±0.0567	0.0577	0.8326±0.0461	0.0948	**0.8363±0.0464**	0.0491
ionosphere	0.9345±0.0327	0.0233	0.9248±0.0224	0.0688	**0.9441±0.0254**	0.2189	0.8872±0.0308	0.0511
monks2	0.7940±0.0450	0.0330	0.8065±0.0212	0.0595	0.8074±0.0391	0.3659	**0.8320±0.0448**	0.0509
parkinsons	0.9159±0.0387	0.0270	0.8995±0.0386	0.0399	0.9344±0.0421	0.0451	**0.9415±0.0378**	0.0358
ringnorm	0.9530±0.0273	0.0270	0.9560±0.0226	0.0594	**0.9675±0.0189**	0.3284	0.8485±0.0337	0.0448
sonar	0.8066±0.0460	0.0245	0.8489±0.0480	0.0467	0.8568±0.0519	0.0592	**0.8738±0.0449**	0.0296
spect	0.6900±0.1119	0.0217	0.6875±0.0633	0.0213	**0.7025±0.1094**	0.0038	**0.7025±0.1033**	0.0214
transfusion	0.7348±0.0262	0.0487	0.7628±0.0175	0.8928	**0.7939±0.0264**	1.8919	0.7839±0.0249	0.6641
twonorm	0.9725±0.0186	0.0195	0.9720±0.0158	0.0647	0.9695±0.0205	0.2988	**0.9730±0.0165**	0.0636
wdbc	**0.9761±0.0116**	0.0215	0.9708±0.0119	0.1345	0.9743±0.0146	0.8451	0.9743±0.0148	0.1185
wpbc	0.7627±0.0118	0.0306	0.7697±0.0176	0.0420	0.7988±0.0472	0.0428	**0.8002±0.0405**	0.0439

our method, we employ linear SVM, linear LDM and linear TBLDM after Nyström method. We choose Liblinear for linear SVM; the source code of Zhang and Zhou (2014) for linear LDM. A nonlinear SVM also runs directly on these large-scale datasets. For the convenience of computation, $C, c_1, c_2, c_3, c_4, \lambda_1, \lambda_2, \lambda_3, \lambda_4$ are all set to 1, γ that used for nonlinear SVM and Nyström method is set to the average squared distance between data points and the sample mean. The number of landmark points of Nyström method is chosen as $m = 50, 100$. Table A.3 tells us that all linear classifiers running after the Nyström method can get a close classification accuracy result compared to nonlinear SVM, even with such small number of landmark points m. However, we can see from Table A.4 that the running time of all linear classifier frameworks plus Nyström method are much faster than that of nonlinear SVM. Moreover, we can see that TBLDM is the fastest if we only compared the time running by three linear classifiers. In addition to nonlinear SVM, all classifiers labelled as SVM, LDM and TBLDM in Table A.3 and Table A.4 are linear.

Table A.3: Classification accuracy results on 4 large-scale datasets

		m=50			m=100		
DATASETS	Nonlinear-SVM	SVM	LDM	TBLDM	SVM	LDM	TBLDM
cod-rna	0.8778	0.8650	0.8542	0.8536	0.8651	0.8618	0.8541
ijcnn1	0.9840	0.9138	0.9050	0.9050	0.9357	0.9203	0.9159
skin	0.9756	0.9982	0.9972	0.9759	0.9985	0.9978	0.9807
w8a	0.9939	0.9696	0.9697	0.9698	0.9721	0.9709	0.9707

Table A.4: Time (seconds) comparison on 4 large-scale data sets

		m=50				m=100			
DATASETS	Nonlinear-SVM	Nyström	SVM	LDM	TBLDM	Nyström	SVM	LDM	TBLDM
cod-rna	358.88	0.41	0.50	0.49	0.33	0.71	0.55	0.53	0.34
ijcnn1	46.28	0.38	0.63	0.67	0.12	0.65	1.09	1.23	0.15
skin	1357.9	0.86	0.99	1.64	0.92	1.45	1.49	2.42	0.84
w8a	533.02	1.39	0.30	0.40	0.05	1.77	0.54	0.73	0.07

A.5 Conclusions

Inspired by the idea of LDM and TBSVM, in this chapter, we introduce the notions of positive margin and negative margin of samples and then present a novel classification method, TBLDM, by optimizing the positive and negative margin distributions. The experimental results on sixteen regular scale datasets and four large scale datasets indicate that, compared with SVM, TBSVM and LDM, the proposed TBLDM is a fast, effective and robust classifier. From the derivation process in Section A.3, we can see that the technique used in this chapter has a certain commonality. Therefore, it will be interesting to generalize the idea of TBLDM to regression models and other learning settings.