EFFICIENT REINFORCEMENT LEARNING THROUGH VARIANCE REDUCTION AND TRAJECTORY SYNTHESIS

BY

XIAOMING ZHAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Assistant Professor Jian Peng

ABSTRACT

Reinforcement learning is a general and unified framework that has been proven promising for many important AI applications, such as robotics, self-driving vehicles. However, current reinforcement learning algorithms suffer from large variance and sampling inefficiency, which leads to slow convergent rate as well as unstable performance. In this thesis, we manage to alleviate these two relevant problems. For enormous variance, we combine variance reduced optimization with deep Q-learning. For inefficient sampling, we propose novel framework that integrates self-imitation learning and artificial synthesis procedure. Our approaches, which are flexible and coud be extended to many tasks, prove their effectiveness through experiments on Atari and MuJoCo environment.

To my parents and Yuchen, for their unconditional love and support.

ACKNOWLEDGMENTS

I would first like to thank my advisor Professor Jian Peng of the Department of Computer Science at University of Illinois at Urbana-Champaign, for providing the perfect environment for this research. He constantly encouraged me to explore interesting and important problems. The constantly invaluable advice he provided steered me in the right direction when I worked on this thesis and other projects. I would also like to thank Yang Liu, Yuanyi Zhong, Yunan Luo, Jinglin Chen, Shibi He, Qing Ye and Wei Qian, for the inspiring discussions on this project as well as their help on conducting experimental studies. I owe a special thanks to the CS @ Illinois graduate advising team.

The views and conclusions contained in this thesis are those of the author(s) and should not be interpreted as representing any funding agencies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Interacting with the environment, when we talk about the nature of learning, is probably the first idea that comes to us. Infants learn to walk by tumbling to the ground; eaglets learn to fly through hopping between branches of trees. Although they do not have an explicit teacher, they do have direct connections to the around environment. Reinforcement learning (RL) is a computational approach to imitate the nature of learning from interactions. The fundamental idea behind RL is learning what to do by trials, namely learn how to map current situations to actions so as to achieve the final goal. However, an action does not only affect current situation but also has lasting effects for future scenarios. The above two characteristics, trial-and-error search and delayed rewards, are the two most important features that make RL distinct from other approaches [Sutton and Barto, 2018].

1.1 ELEMENTS OF REINFORCEMENT LEARNING

In every discussion of a reinforcement learning system, besides the environment and agent, there exist several key components: a policy, a reward, a value function and an optional model of the environment. The general mechanism of the reinforcement learning system is depicted in figure 1.1.

- **Policy**: a policy defines the agent's reaction to the environment. Sufficiently, it determines the behaviour of the agent.
- **Reward**: the reward is the realistic representation of the learning process's final goal. In each iteration of the interaction, the environment gives the learning agent a numeric value corresponding to the action agent takes. The general object of the agent is to maximize the cumulative rewards.
- **Value function**: the value function indicates what is a good choice in the long-term sense, compared to the reward, which is a feedback of the immediate situation. Generally speaking, the value function of a situation / state, is the cumulative rewards that the agent could gain starting from that situation / state.
- **Model of environment**: the model predicts what the environment will behave when receiving the action from agent. This model is not necessary for a reinforcement learning system. Actually, if we have a model for environment, it is called a *model-based* algorithm. Otherwise, the algorithm is a *model-free* one.

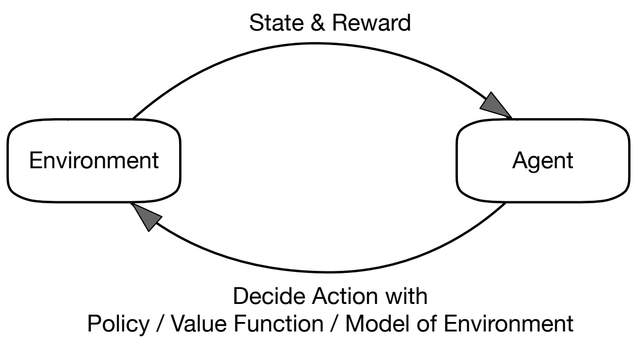


Figure 1.1: General Reinforcement Learning System

1.2 DIFFERENCE BETWEEN VARIOUS MACHINE LEARNING FRAMEWORKS

Reinforcement learning is a kind of framework used in machine learning community and it distinguishes itself from *supervised* learning and *unsupervised* learning.

Supervised learning is a kind of task that infers a function from *labeled* data and manages to generalize the inference from training examples to unseen data. In general, supervised learning requires domain knowledge for labeling the data. In reinforcement learning literature, there does not exist such expert knowledge which is external to the reinforcement learning system.

On the other hand, unsupervised learning concentrates on finding hidden structure from *unlabeled* data. Although reinforcement learning uses samples without label as well, its focus is laid on maximizing the future cumulative rewards instead of discovering hidden structure.

Reinforcement learning has been studied in many other fields for a long time, such as game theory [Lee, 2008], control theory [Bertsekas et al., 1995] and multi-agent system [Littman, 1994]. Recently, as the prevalence of deep learning [LeCun et al., 2015], there are plethora of work that focuses on deep reinforcement learning (DRL). It has been proven promising that DRL is able to achieve or even outperform human experts, such as Go [Silver et al., 2016, Silver et al., 2017], Atari games [Mnih et al., 2015]. However, there also exist severe problems in DRL training scheme. The large variance results in slow convergence rate and the extreme sample inefficiency makes that DRL is hard to be implemented in daily situation. In this thesis, we propose two new algorithms that aim to alleviate such problems. One work is based on variance reduced gradient descent [Zhao et al., 2019] and the other is developed from self-imitation policies [Gangwani et al., 2018].

The rest of the thesis is organized as follows. Chapter 2 gives background and problem definition of reinforcement learning. Chapter 3 introduces our new algorithm for variance reduction learning. A new algorithm for efficiently exploiting samples are explained in Chapter 4. Related work is discussed in Chapter 5. We conclude this study in Chapter 6.

CHAPTER 2: REINFORCEMENT LEARNING BACKGROUND

In this chapter, as an overview, we talk about some basic algorithms that are used in reinforcement learning field.

2.1 MARKOV DECISION PROCESS

Single agent reinforcement learning is usually formulated as a Markov decision process (MDP). The definition involves state space $\mathcal{S}$ and agent's action space $\mathcal{A}$. The agent sequentially interacts with the environment in discrete time steps $t = 0, 1, 2, \dots$. At every time step t, the agent gets a state $s_t \in \mathcal{S}$, which represents the situation of the environment. Based on that state s_t, the agent uses a policy π_θ to select the corresponding action $a_t \in \mathcal{A}$, where θ is the parameters of the policy. Hereafter, for simplicity, we omit θ. Policy π maps the state $s_t \in \mathcal{S}$ to either a distribution over $\mathcal{A}$ (stochastic policy), or a single action a_t (deterministic policy). Obviously, we have:

$$\sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) = 1, \forall s_t \in \mathcal{S}$$

As a consequence of the action a_t, the agent obtains a reward $r_t(s_t, a_t) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ per step. Besides, it will find itself in a new state s_{t+1}. The overall system is described in figure 2.1, which is a more explicit version of figure 1.1. Therefore, the interaction between agent and environment will provide a trajectory τ:

$$\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots\}$$

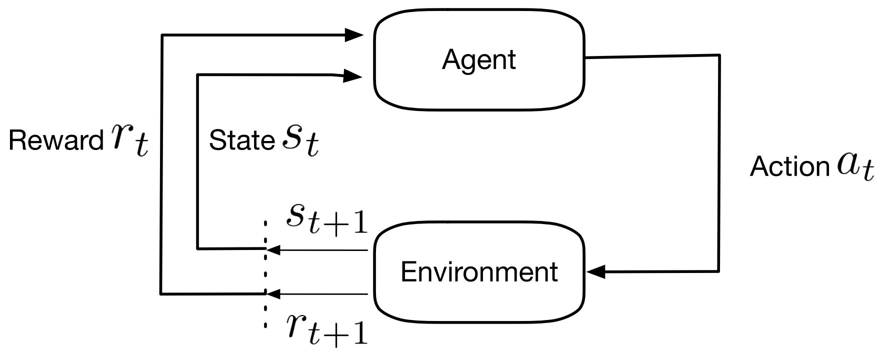


Figure 2.1: Markov Decision Process System [Sutton and Barto, 2018]

We could define the discounted return of a trajectory as $R(\tau) = \sum_{t=1}^{T} \gamma^t r(s_t, a_t)$. Here T is the time horizon of the trajectory, which could be finite or infinite ($T = \infty$). We should notice that $\gamma \in [0, 1]$ is a discount factor. In mathematical perspective, it is used to bound the infinite summation as $T \to \infty$. Meanwhile, from the practical perspective, it determines the effect of the reward in every step. If $\gamma = 1$, every reward will be equally important. However, if γ is small, only the recent rewards play important roles since $\gamma^t \to 0$ as $t \to T$. Based on $R(\tau)$, we could define the following functions:

$$V^\pi(s) \doteq \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{2.1}$$

$$Q^\pi(s, a) \doteq \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{2.2}$$

$$A^\pi(s, a) \doteq Q^\pi(s, a) - V^\pi(s) \tag{2.3}$$

where $V^\pi(s)$ is the value function at state s under policy π. Similarly, $Q^\pi(s, a)$ is the action value function at state s and action a under policy π. $A^\pi(s, a)$ is the advantage function, which measures the *advantage* that the agent takes action a at state s over average performance. $\tau \sim \pi$ is a shorthand to indicate the trajectory distribution depends on π. If we model the environment as an unknown system dynamics with $p(s_{t+1}, r_{t+1}|s_t, a_t)$ and an initial state distribution $p_0(s)$. The trajectory distribution could be formulated as:

$$p(\tau) = p_0(s_0) \sum_{i=0}^{T} \pi(a_t|s_t)p(s_{t+1}, r_{t+1}|s_t, a_t)$$

$$\sum_{s_{t+1}\in\mathcal{S}} \sum_{r_{t+1}\in\mathbb{R}} p(s_{t+1}, r_{t+1}|s_t, a_t) = 1, \forall s_t \in \mathcal{S}, \forall a_t \in \mathcal{A}$$

Hereafter, we assume that the reward is deterministic, namely $p(s_{t+1}, r(s_t, a_t)|s_t, a_t) = p(s_{t+1}|s_t, a_t)$. Now we could define the goal of the reinforcement learning in MDP settings. In general, the agent aims to maximize the expected discounted sum of rewards:

$$\max_\theta \; J(\theta) = V^\pi(s_0) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0] = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|s_0] \tag{2.4}$$

2.2 DEEP Q-LEARNING

From the Markov decision process definition, we have the following deduction:

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \\
&= \mathbb{E}_{\tau\sim\pi}[r_1 + \gamma R(\tau')|s_0 = s] \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[r(s,a) + \gamma \mathbb{E}[R(\tau')|s_0 = s']\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[r(s,a) + \gamma V^{\pi}(s')\right] \\
&= \mathbb{E}_{a,s'}\left[r(s,a) + \gamma V^{\pi}(s')\right]
\end{aligned}$$

This is the *Bellman equation* [Bellman, 2010] for value function. Similarly, we have Bellman equation for action-value function as:

$$Q^{\pi}(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}[Q^{\pi}(s',a')]]$$

If we define the optimal value function and action-value function as following:

$$\begin{aligned}
V^{\pi^*}(s) &= \max_{\pi} V^{\pi}(s) \\
Q^{\pi^*}(s,a) &= \max_{\pi} Q^{\pi}(s,a)
\end{aligned}$$

where π^* is the optimal policy. The optimal action at state s_t is $\arg\max_a \pi^*(a|s_t) = \arg\max_a Q^{\pi^*}(s_t,a)$. Thus, we have the *Bellman optimality equation* as:

$$V^{\pi^*}(s) = \max_{a} \mathbb{E}_{a,s'}\left[r(s,a) + \gamma V^{\pi^*}(s')\right] \tag{2.5}$$

$$Q^{\pi^*}(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \max_{a'} Q^{\pi^*}(s',a')] \tag{2.6}$$

If we update the action-value function corresponding to equation 2.6 iteratively, we reach core idea of the Q-learning algorithm [Watkins, 1989]:

$$Q^{\pi_{i+1}}(s_t,a_t) = \mathbb{E}_{s'}[r(s,a) + \gamma \max_{a'} Q^{\pi_i}(s',a')] \tag{2.7}$$

where Q^{π_i} is the action-value function in the ith iteration. If the policy π_i is determined by parameter θ_i, we could write $Q^{\pi_i}(s,a)$ as $Q(s,a;\theta_i)$. The action-value function and policy will converge $Q^{\pi_i} \to Q^{\pi^*}, \pi_i \to \pi^*$ as $i \to \infty$. In practice, it is impossible to directly compute equation 2.7 since the action-value function in each iteration is estimated separately. In reinforcement learning community, a function approximator is often used to estimate the Q^{π^*}. If we denote the target parameter as θ^-, we could try to reach the optimal action-value function by minimizing the mean squared error of $Q(s,a;\theta_i)$ with $y_i = r(s,a) + \gamma \max_{a'} Q(s',a';\theta_i^-)$ [Mnih et al., 2015]:

$$\begin{aligned} L(\theta_i) &= \mathbb{E}_{s,a}\left[(\mathbb{E}_{s'}[y_i|s,a] - Q(s,a;\theta_i))^2\right] \\ &= \mathbb{E}_{s,a,s'}\left[(y_i - Q(s,a;\theta_i))^2\right] + \mathbb{E}_{s,a}\left[\mathrm{Var}_{s'}(y_i)\right] \\ \nabla_{\theta_i} L(\theta_i) &= \mathbb{E}_{s,a,s'}\left[\left(r(s,a) + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right] \end{aligned}$$

Note that $L(\theta_i)$ changes between iterations and the target θ_i^- depends on the weights θ_i, thus it is different from supervised learning. To make stable training and encourage exploration, we integrate the above technique with experience replay [Lin, 1992] and ϵ-greedy [Sutton and Barto, 2018] to reach algorithm 2.1.

Algorithm 2.1: Deep Q-network (DQN)

```
   Initialize: replay memory $D$ of capacity $N$
   Initialize: action-value function $Q$ with parameter $\theta$
   Initialize: target action-value function $\widehat{Q}$ with target $\theta^- = \theta$
1  for episode = 1 to $M$ do
2      Initialize state $s_1$
3      for $t$ = 1 to $T$ do
4          if uniform probability $< \epsilon$ then
5              select a random action $a_t$
6          else
7              $a_t = \operatorname{argmax}_a Q(s_t, a; \theta)$
8          end
9      end
10     Observe $r_{t+1}$ and $s_{t+1}$
11     Store $(s_t, a_t, r_{t+1}, s_{t+1})$ into $D$
12     Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
13     if episode terminates at $j+1$ then
14         $y_j = r_j$
15     else
16         $y_j = r(s_j, a_j) + \gamma \max_{a'} \widehat{Q}(s_{j+1}, a'; \theta^-)$
17     end
18     Perform graident descent following equation 2.7 with respect to parameter $\theta$
19     Every $C$ steps reset $\widehat{Q} = Q$
20 end
```

2.3 POLICY GRADIENT

In this section, different from the *action-value* method introduced in section 2.2, we want to discuss an approach, policy gradient. Polciy gradient learns the policy parameter from the gradient of a scalar policy performance measurement. In this setting, value function is not necessary for selecting actions. Actually, all methods which align with the following formula could be categorized as policy gradient:

$$\theta_{i+1} = \theta_i + \alpha \widehat{\nabla F(\theta_i)}$$

where $\widehat{\nabla F(\theta_i)}$ is an estimate for the gradient of the measurement. Recall from equation 2.4, $J(\theta)$ could be treated as the target measurement. We could prove the following *Policy Gradient Theorem* [Sutton et al., 2000]:

Theorem 2.1. *For any Markov decision process, we have:*

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \sum_{s \in \mathcal{S}} \rho^{\pi}(s) \sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(a|s) \\ &= \mathbb{E}_{s \sim \pi} \left[\sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s,a) \right] \end{aligned} \tag{2.8}$$

where $\rho^{\pi}(s) = \sum_{t=0}^{T} \gamma^t p(s_t = s|s_0, \pi)$. $p(s_t = s|s_0,\pi)$ *is the probability that the agent reaches state* s *at time step* t *when starting from state* s_0 *under policy* π. $s \sim \pi$ *denotes the distribution that state* s *is encountered under policy* π.

Proof. First, for $\forall s \in \mathcal{S}$, we have:

$$\begin{aligned} \frac{\partial V^{\pi}(s)}{\partial \theta} &= \frac{\partial}{\partial \theta} \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s,a) \\ &= \sum_{a \in \mathcal{A}} \left\{ \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s,a) + \pi(a|s) \frac{\partial}{\partial \theta} Q^{\pi}(s,a) \right\} \\ &= \sum_{a \in \mathcal{A}} \left\{ \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s,a) + \pi(a|s) \frac{\partial}{\partial \theta} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s') \right] \right\} \\ &= \sum_{a \in \mathcal{A}} \left\{ \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s,a) + \gamma \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a) \frac{\partial V^{\pi}(s')}{\partial \theta} \right\} \qquad (2.9) \\ &= \sum_{x \in \mathcal{S}} \sum_{t=0}^{T} \gamma^t p(s \to x, t, \pi) \sum_{a \in \mathcal{A}} \frac{\partial \pi(a|x)}{\partial \theta} Q^{\pi}(x,a) \end{aligned}$$

where $p(s \to x, t, \pi)$ is the probability that going to state x at time step t when starting from state s under policy π. The last equality comes from unrolling equation 2.9 several times. Now we have:

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} V^{\pi}(s_0) \\
&= \sum_{s \in \mathcal{S}} \sum_{t=0}^{T} \gamma^t p(s_0 \to s, t, \pi) \sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s, a) \\
&= \sum_{s \in \mathcal{S}} \rho^{\pi}(s) \sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s, a) \\
&= \mathbb{E}_{s \sim \pi} \left[\sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s, a) \right]
\end{aligned}$$

Now, let's talk about a practical algorithm that derive from policy gradient theorem. Note that:

$$\begin{aligned}
\mathbb{E}_{s \sim \pi} \left[\sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} Q^{\pi}(s, a) \right] &= \mathbb{E}_{s \sim \pi} \left[\sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s, a) \frac{\partial \pi(a|s)}{\partial \theta} \frac{1}{\pi(a|s)} \right] \\
&= \mathbb{E}_{s, a \sim \pi} \left[Q^{\pi}(s, a) \frac{\partial \pi(a|s)}{\partial \theta} \frac{1}{\pi(a|s)} \right] \\
&= \mathbb{E}_{s, a \sim \pi} \left[Q^{\pi}(a|s) \frac{\partial \log \pi(a|s)}{\partial \theta} \right] \\
&= \mathbb{E}_{s, a \sim \pi} \left[\mathbb{E}_{\tau \sim \pi} \left[R(\tau) | s, a \right] \frac{\partial \log \pi(a|s)}{\partial \theta} \right] \\
&= \mathbb{E}_{s, a \sim \pi} \left[\mathbb{E}_{\tau \sim \pi} \left[R(\tau) \frac{\partial \log \pi(a|s)}{\partial \theta} \middle| s, a \right] \right] \\
&= \mathbb{E}_{\tau \sim \pi} \left[R(\tau | s, a) \frac{\partial \log \pi(a|s)}{\partial \theta} \right]
\end{aligned}$$

where $R(\tau|s, a)$ denotes that trajectory τ starts from state s and action a. Now we have the REINFORCE algorithm [Williams, 1992] in algorithm 2.2. The policy gradient theorem could be generalized by adding an arbitrary baseline function $b(s)$:

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \frac{\partial \pi(a|s)}{\partial \theta} (Q^{\pi}(s, a) - b(s)) \\
&= \mathbb{E}_{\tau \sim \pi} \left[(R(\tau|s, a) - b(s)) \frac{\partial \log \pi(a|s)}{\partial \theta} \right]
\end{aligned}$$

To see the correctness, we only need to notice that:

$$\begin{aligned}\sum_{a\in\mathcal{A}} b(s)\frac{\partial\pi(a|s)}{\partial\theta} &= b(s)\sum\frac{\partial\pi(s,a)}{\partial\theta} \\ &= b(s)\frac{\partial}{\partial\theta}\sum\pi(a|s) \\ &= b(s)\frac{\partial}{\partial\theta}1 = 0\end{aligned}$$

Although baseline function does not change the expected values of the target, it does reduce the variance of the update step in the learning process. In return, it will accelerate the training procedure by adding a $b(s)$ [Greensmith et al., 2004].

Algorithm 2.2: REINFORCE

```
  **Require:** policy $\pi$ with parameter $\theta$
  **Require:** learning rate $\alpha$
1 for episode = 1 to M do
2     Generate a trejectory $\tau = \{s_0, a_0, r_1, s_1, a_1, \dots\}$
3     for t = 0 to T do
4         $R = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
5         $\theta = \theta + \alpha R \cdot \frac{\partial \log \pi(a_t|s_t)}{\partial\theta}$
6     end
7 end
```

CHAPTER 3: VARIANCE REDUCTION OPTIMIZATION

In this chapter, we describe our work on variance reduction optimization for training reinforcement learning models [Zhao et al., 2019]. Large variance is one of the key problems that hauls reinforcement learning from implementing in daily usage. We leverage the strength of stochastic variance reduced gradient descent (SVRG) [Johnson and Zhang, 2013] to alleviate it. We integrate it with Adam optimization [Kingma and Ba, 2014] to obtain a Stochastic Variance Reduction for Deep Q-learning (SVR-DQN).

The remaining of this chapter are organized as two parts. First, we present our approach, including the algorithm description and analysis. Second, we present our experiment experimental results. This includes experiment settings, evaluation metrics and performance.

3.1 STOCHASTIC VARIANCE REDUCED GRADIENT DESCENT

Many machine learning problems are considering a finite sum optimization problem as following:

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

Let $w^* = \arg\min_w f(w)$ be the optimal solution, researchers are motivated to find a value w such that $|f(w) - f(w^*)| < \epsilon$, where ϵ is the accuracy measurement. In order to develop fast stochastic first-order methods, we should make sure that when the w gets closer to optimum as the training iterates, the variance of randomized updating direction decreases. A popular approach is gradient descent:

$$\begin{aligned} w_t &= w_{t-1} - \eta_t \nabla f(w_{t-1}) \\ &= w_{t-1} - \frac{\eta_t}{n}\sum_{i=1}^{n} \nabla f_i(w_{t-1}) \end{aligned}$$

where w_t and η_t are the parameter and learning rate at iteration t. However, it is expensive to evaluate all the n functions / samples in each iteration. We could use stochastic gradient descent (SGD) instead:

$$w_t = w_{t-1} - \eta_t \cdot \frac{1}{m}\sum_{j=1}^{m} \nabla f_{i_{(t,j)}}(w_{t-1}) \tag{3.1}$$

where m is the size of mini-batch sampled from total n instances and $i_{(t,j)} \in \{1, 2, \ldots, n\}$ is a randomly chosen index at iteration t. Actually, it is generalized to write in the following formation:

$$w_t = w_{t-1} - \eta_t g_t(w_{t-1}, \xi_t)$$

where ξ_t is a random variable that may depend on w_{t-1} , and the expectation $\mathbb{E}_{\xi_t}(g(w_{t-1}, \xi_t)|w_{t-1}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w_{t-1})$. Large variances come from the randomness encoded in $g_t(w_{t-1}, \xi_t)$. To alleviate this, stochastic variance reduced gradient descent (SVRG) maintains a snapshot of an estimated $\tilde{w}$, which is close to optimal w^*, every certain iterations. Moreover, an average gradient $\tilde{\mu}$ is calculated:

$$\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{w}) \tag{3.2}$$

which goes through all training samples. Obviously, the expectation $\tilde{\mu} - \frac{1}{m}\sum_{j=1}^{m} \nabla f_{i_{(t-1,j)}}(\tilde{w})$ is zero. We extend equation 3.1 as:

$$\begin{aligned} w_t &= w_{t-1} - \eta_t \left(\frac{1}{m}\sum_{j=1}^{m} \nabla f_{i_{(t,j)}}(w_{t-1}) - \frac{1}{m}\sum_{j=1}^{m} \nabla f_{i_{(t,j)}}(\tilde{w}) + \tilde{\mu} \right) \\ \mathbb{E}[w_t|w_{t-1}] &= w_{t-1} - \eta_t f(w_{t-1}) \end{aligned} \tag{3.3}$$

This is the core step of SVRG. Note that when w is close to $\tilde{w}$, the difference $f(w) - f(\tilde{w})$ is small. When both w_t and $\tilde{w}$ converge to the same optimal parameter w^*, then $\tilde{\mu} \to 0$ and $f(w) \to f(\tilde{w})$. Therefore, the variance of SVRG in update rule equation 3.3 is reduced and SVRG can find the more accurate gradient direction estimation.

3.2 STOCHASTIC VARIANCE REDUCTION DQN

In order to improve the performance of Adam optimization, we apply SVRG to find the accurate gradient direction based on the small stochastic training subset and propagate the optimized first-order information to Adam. The details of algorithm are listed in algorithm 3.1. Note that the ideal selection of mini-batch size b and SVRG inner loop iteration number m should satisfy the constraint: $b \times m \geq B$. Meanwhile, it is unnecessary to rescale g_s , since effective step size in Adam is invariant to the scale of the gradients [Kingma and Ba, 2014].

Algorithm 3.1: Stochastic Variance Reduction for Deep Q-network Optimization

```
   Require: $B$: size of training batch size
   Require: $b$: mini-batch size
   Require: $\eta$: learning rate
   Require: $m$: inner loop iterations
   Require: $\alpha$: Adam step size
   Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rate for moment estimates
 1 Initialize $\tilde{w}_1$ (parameter vector)
 2 Initialize $m_1 = 0$ (first moment vector)
 3 Initialize $v_1 = 0$ (second moment vector)
 4 for episode $s$ = 1 to $M$ do
 5     Sample batch of size $B$ from all samples (for simplicity, we just denote samples as $1, 2, \ldots, B$)
 6     $\tilde{\mu}_s = \frac{1}{B}\sum_{i=1}^{B} \nabla f_i(\tilde{w}_s)$
 7     $w_1 = \tilde{w}_s$
 8     for $t$ = 1, ..., $m$ do
 9         Uniformly sample a mini-batch $\{i_{(t,1)}, \ldots, i_{(t,b)}\}$
10         $w_t = w_{t-1} - \eta(\frac{1}{b}\sum_{j=1}^{b} \nabla f_{i_{(t,j)}}(w_{t-1}) - \frac{1}{b}\sum_{j=1}^{b} \nabla f_{i_{(t,j)}}(\tilde{w}_s) + \tilde{\mu}_s)$
11     end
12     $g_s = w_m - \tilde{w}_s$
13     $m_{s+1} = \beta_1 \cdot m_s + (1 - \beta_1) \cdot g_s$ (Update biased first moment estimate)
14     $v_{s+1} = \beta_2 \cdot v_s + (1 - \beta_2) \cdot g_s^2$ (Update biased second raw moment estimate)
15     $\hat{m}_{s+1} = m_{s+1}/(1 - \beta_1^{s+1})$ (Compute bias-corrected first moment estimate)
16     $\hat{v}_{s+1} = v_{s+1}/(1 - \beta_2^{s+1})$ (Compute bias-corrected second raw moment estimate)
17     $\tilde{w}_{s+1} = \tilde{w}_s - \alpha \cdot \hat{m}_{s+1}/(\sqrt{\hat{v}_{s+1} + \epsilon})$ (Update parameters)
18 end
```

3.3 APPROXIMATION GRADIENT ERROR VARIANCE REDUCTION

The Approximation Gradient Error(AGE) is the error in the gradient direction estimation of cost function $f(\tilde{w})$, where $\tilde{w}$ is the hyper-parameters of this function, which are optimized with gradient descent methods iteratively by minimizing the DQN loss (line 17 in algorithm 3.1). Given the certain learning samples preserved in experience buffer, the ideal gradient estimation of loss function is supposed to give the accurate learning direction (derivation value) leveraging the current information provided by those learning samples, thus the agent (DQN) can quickly converge to policy optimum by optimizing hyper-parameter at the gradient direction. However, AGE appears in gradient estimation process.

AGE is a result of several factors: Firstly, the sub-optimality of current hyper-parameters $\tilde{w}$ due to inexact minimization. Secondly, the constrained representing strength of DQN.

Thirdly, the limited representation number of the samples we used for deriving the gradients. Lastly, representation error due to unseen(un-stored) state transitions and policies caused by nite storage of Experience-Replay buffer. The AGE can cause the distortion of the gradient estimation, thus derive the agent policy to a worse one. The AGE can also cause a large variability of DQN performance and postpone the process when DQN gets to local optima. To analyze the AGE variance we first propose the variance of approximation gradient for one single sample.

We suppose the gradient estimation from one single sample is $\nabla f_i(\tilde{w}) = AGE_i + \nabla f_i(w^*)$, where i is one training sample from the replay buffer and w^* denotes the exact minimized parameter from current stored samples. We also denote that $\nabla f(w^*)$ is the optimal gradient direction given current stored samples, $\mathbb{E}(AGE_i) = 0$ and $\mathrm{Var}(AGE_i) = \sigma^2$. We have:

$$\begin{aligned}
\mathrm{Var}(\nabla f_i(\tilde{w})) &= \mathrm{Var}\left(AGE_i + \nabla f_i(w^*))\right) \\
&= \mathrm{Var}(AGE_i) + \mathrm{Var}(\nabla f_i(w^*)) \\
&= \sigma^2
\end{aligned}$$

Suppose that ∇f_i is L-Lipschitz continuous, namely:

$$f_i(\tilde{w}) \geq f_i(w^*) + \langle \nabla f_i(\tilde{w}), \tilde{w} - w^* \rangle + \frac{1}{2L} \|\nabla f_i(\tilde{w}) - \nabla f_i(w^*)\|^2$$

by summing this inequality above over all the training sample i, and divide LHS and RHS by n, we obtain the bound of approximation gradient error variance $\mathrm{Var}(\nabla f_i(\tilde{w}))$ that:

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(\tilde{w}) - \nabla f_i(w^*)\|^2 \leq 2L\left(f(\tilde{w}) - f(w^*)\right) \tag{3.4}$$

Recall that in algorithm 3.1, we have:

$$\begin{aligned}
g_{SVR-DQN} &= w_m - \tilde{w} \\
&= w_{m-1} - \eta \cdot \left(\frac{1}{b}\sum_{i=1}^{b} \nabla f_i(w_{m-1}) - \frac{1}{b}\sum_{i=1}^{b} \nabla f_i(\tilde{w}) + \tilde{\mu}\right) - \tilde{w} \\
&= w_0 - \sum_{i=0}^{m-1} \kappa_i - \tilde{w} \\
&= -\sum_{i=0}^{m-1} \kappa_i
\end{aligned}$$

where $\kappa_i = \eta \cdot \left(\frac{1}{b}\sum_{i=1}^{b}\nabla f_i(w_{m-1}) - \frac{1}{b}\sum_{i=1}^{b}\nabla f_i(\tilde{w}) + \tilde{\mu}\right)$ and the last equality comes from that $w_0 = \tilde{w}$. Note that at each iteration, mini-batch is uniformly sampled, for $i \neq j$, $\mathrm{Cov}(\kappa_i, \kappa_j) = 0$. Thus, we have:

$$\begin{aligned}
\mathrm{Var}(g_{SVR-DQN}) &= \mathrm{Var}(-\sum_{i=0}^{m-1}\kappa_i) \\
&= \sum_{i=0}^{m-1}\mathrm{Var}(\kappa_i) \\
&\leq \sum_{i=0}^{m-1}\mathbb{E}(\|\kappa_i\|^2)
\end{aligned}$$

The last inequality results from that for any random variable X, $\mathbb{E}[(X-Y)^2]$ reaches its minimum only if $Y = \mathbb{E}(X)$. From equation 3.2, we know that $\tilde{\mu} = \mathbb{E}_i(\nabla f_i(\tilde{w}))$, where $\mathbb{E}_i$ denotes take expectation over all instances. Based on the fact that $\nabla f_i(w^*) = 0$, we could write:

$$\tilde{\mu} = \mathbb{E}_i[\nabla f_i(\tilde{w}) - \nabla f_i(w^*)]$$

Now, we could derive the bound:

$$\begin{aligned}
\mathbb{E}(\|\kappa_i\|^2) &= \mathbb{E}\left[\left\|\eta \cdot \left(\frac{1}{b}\sum_{j=1}^{b}\nabla f_j(w_i) - \frac{1}{b}\sum_{j=1}^{b}\nabla f_j(\tilde{w}) + \tilde{\mu}\right)\right\|^2\right] \\
&= \eta^2 \cdot \mathbb{E}\left[\left\|\frac{1}{b}\sum_{j=1}^{b}(\nabla f_j(w_i) - \nabla f_j(w^*) + \nabla f_j(w^*) - \nabla f_j(\tilde{w}) + \tilde{\mu})\right\|^2\right] \\
&\leq \frac{2\eta^2}{b^2}\sum_{i=1}^{b}\mathbb{E}\left\|\nabla f_j(w_i) - \nabla f_j(w^*)\right\|^2 + \frac{2\eta^2}{b^2}\sum_{i=1}^{b}\mathbb{E}\left\|\nabla f_j(\tilde{w}) - \nabla f_j(w^*) - \tilde{\mu}\right\|^2 \\
&= \frac{2\eta^2}{b^2}\sum_{i=1}^{b}\mathbb{E}\left\|\nabla f_j(w_i) - \nabla f_j(w^*)\right\|^2 + \\
&\quad \frac{2\eta^2}{b^2}\sum_{i=1}^{b}\mathbb{E}\left\|\nabla f_j(\tilde{w}) - \nabla f_j(w^*) - \mathbb{E}\left[\nabla f_j(\tilde{w}) - \nabla f_j(w^*)\right]\right\|^2 \\
&\leq \frac{2\eta^2}{b^2}\sum_{i=1}^{b}\mathbb{E}\left\|\nabla f_j(w_i) - \nabla f_j(w^*)\right\|^2 + \frac{2\eta^2}{b^2}\sum_{i=1}^{b}\mathbb{E}\left\|\nabla f_j(\tilde{w}) - \nabla f_j(w^*)\right\|^2 \\
&\leq \frac{4L\eta^2}{b}(\nabla f(w_i) - \nabla f(w^*)) + \frac{4L\eta^2}{b}(\nabla f(\tilde{w}) - \nabla f(w^*))
\end{aligned} \tag{3.5}$$

$$\leq \frac{8L\eta^2}{b}\left(\nabla f(\tilde{w}) - \nabla f(w^*)\right) \tag{3.6}$$

Inequality 3.5 follows the equation 3.4 and inequality 3.6 follows that $f(w_i) - f(w^*) \leq \alpha^i(f(\tilde{w}) - f(w^*))$ [Johnson and Zhang, 2013]. Therefore, we have:

$$\begin{aligned} \mathrm{Var}(g_{SVR-DQN}) &\leq \sum_{i=0}^{m-1} \mathbb{E}(\|\kappa_i\|^2) \\ &\leq \frac{8Lm\eta^2}{b}(f(\tilde{w}) - f(w^*)) \end{aligned} \tag{3.7}$$

This shows that SVR-DQN is theoretically more efficient in AGE variance reduction than traditional DQN.

3.4 EXPERIMENTS

3.4.1 Experiment Settings

To demonstrate our methods effectiveness, we evaluate our proposed algorithm on a collection of 20 games from Arcade Learning Environment [Bellemare et al., 2013]. Due to the environment's complexity, it is extremely demanding to nd a good learning algorithm which is generally effective across all 20 games with game-specific hyperparameter tuning.

Similar to the previous work [Mnih et al., 2015], we utilize a neural network as the approximation of action value, taking raw images as input. The network architecture is a convolutional neural network with three convolutional layers and a fully-connected layer. The filter sizes of three convolutional layers are 8×8, 4×4 and 3×3 and the strides are 4, 2 and 1, respectively. All the convolutional layers are followed by a rectifier nonlinear layer [Hahnloser et al., 2000, Hahnloser and Seung, 2001]; The fully-connected layer contains 512 hidden units. To preprocess the raw image, we rescale it into 84×84 from the extracted Y channel.

Following the paper [Mnih et al., 2015], we use the ϵ-greedy scheme for exploration ,where ϵ is annealed linearly from 1.0 to 0.1 over the rst million frames. All the simulated transitions are stored in a sliding replay memory, and the algorithm performs gradient descent on mini-batches of 512 transitions sampled uniformly from the reply memory. We set the learning frequency to 128, which means the training process repeats every 128 mini-batches. We also apply a frame-skipping strategy where the network takes the four frames as an input. All experiments are performed on an NVIDIA GTX Titan-X 12GB graphics card. In this paper, we utilize the tunned version of Double DQN algorithm [Van Hasselt et al., 2016], as it somehow resolves the over-estimation issue in Q-learning.

3.4.2 Evaluation

Our proposed algorithm can obtain more accurate gradient estimation through the same batch of training samples compared to baseline, and we assume that more accurate gradient evaluation should result in more aggressive learning curves in the initial training stage. Though the previous work [Mnih et al., 2015] trained their agent using 200 million (200M) frames or 50M training iteration for each game, we choose to train our agent within only 40M frames or 10M training iterations, due to time constraints. Note that regarding evaluating the performance of SVR-DQN, our main concerns focus on the performance in initial stage. Instead of using Double DQN baseline results for those 20 games published from previous work, to obtain fair comparison, we replicate the baseline results using the same hyper-parameter setting, code base, and random seed initialization as SVR-DQN for 10M training frames. The only difference is that the our gradient estimator could lead to a smaller variance. Our experiments could be nished within two days.

Our evaluation procedure follows the description by [Mnih et al., 2015], at the beginning of each episode, the starting states are randomized by executing a random number special of no-op actions, which have no effect on the environment, for 30 times. This procedure is often referred as '30 no-op evaluation' to provide different starting points for the agent. Our agent is evaluated after a maximum of 5-minute game play, which contains 18,000 frames, with the usage of ϵ-greedy policy where $\epsilon = 0.05$. The rewards are averaged from 100 episodes. For each game, our agent is evaluated at the end of every epoch (160 epochs in total), and we select the best performance as the agents final result. To compare the performance of our algorithm to the Double DQN baseline across games, we apply the normalization algorithm proposed by [Van Hasselt et al., 2016] to obtain the normalized improvement score in percent as follows:

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{|\text{score}_{\text{Double-DQN}} - \text{score}_{\text{random}}|} \tag{3.8}$$

The detailed results could be found in figure 3.1 and 3.3

In summary, we adopt the Double DQN and random score reported by [Mnih et al., 2015] , the results are demonstrated in figure 3.1. We observe a better performance on 19 out of 20 games, which demonstrates the effectiveness of our proposed algorithm. We also give the summary statistics in terms of mean and median score in table 3.1. Compared to Adam, the median performance across 20 games increases from 63.13% to 118.02% and the mean

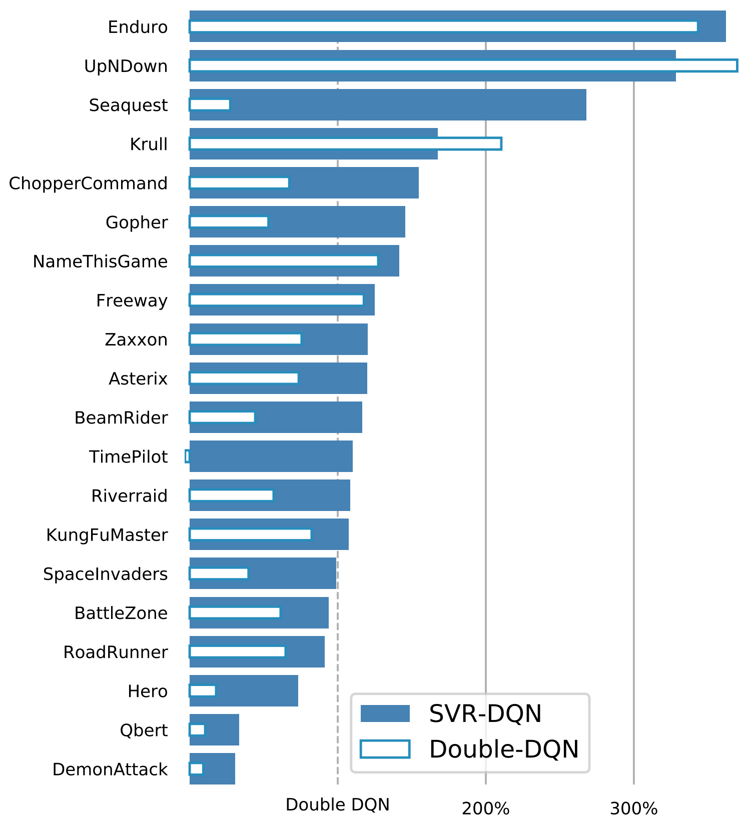


Figure 3.1: Normalized score on 20 Atari games, tested for 100 episodes per game. The blue bars denotes our SVR-DQN while the white bars denote the Adam optimizer, which is a baseline.

performance increases from 92.48% to 139.75%. Noteworthy examples include Seaquest (from 27.42% to 267.94%), Gopher (from 53.22% to 145.42%).

	Mean	Median
Double DQN	92.48%	63.13%
SVR-DQN	**139.75%**	**118.02%**

Table 3.1: Mean and median normalized scores.

We also conduct a comparison of the sample efciency and results could be found in figure 3.2. We observe that SVR-DQN boosts the performance on almost all games, and the sample efficiency of SVR-DQN is nearly twice as fast as original Double-DQN with Adam optimizer.

Also, the performances of three representative games are reported in figure 3.3. The three games include 'BeamRider', 'Freeway', 'Riverraid'. As can be seen in figure 3.3 that our proposed SVR-DQN method results in significant lower average gradient estimates, and the variance of gradient is largely reduced. We also observe that our method outperforms the baselines with a significant margin on the majority of the games, and SVR-DQN leads to

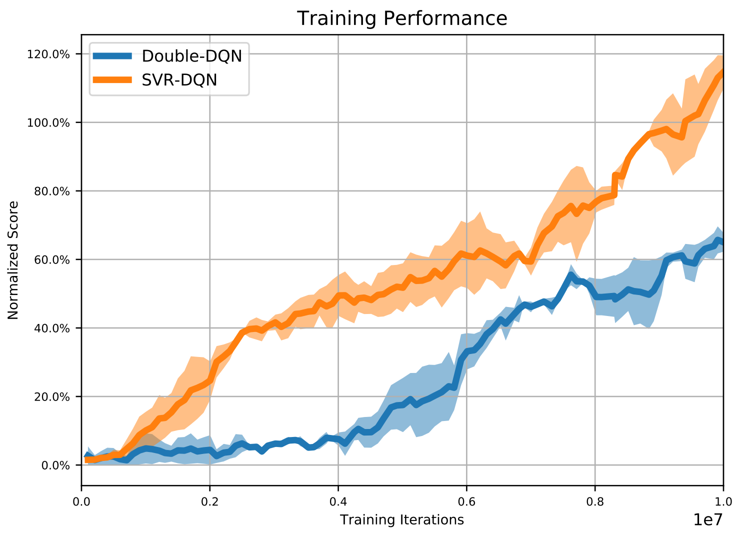


Figure 3.2: Summary plots of sample efficiency. Median over 20 games of the normalized score achieved so far. The normalized score is calculated in equation 3.8.

less variability between the runs of independent learning trials. For the game of Freeway, we see that the divergence of Double-DQN can be prevented by SVR-DQN. On the other hand, the performance of Double-DQN with Adam optimizer has a sudden deterioration in 4M iteration where the gradient variance suddenly increases.

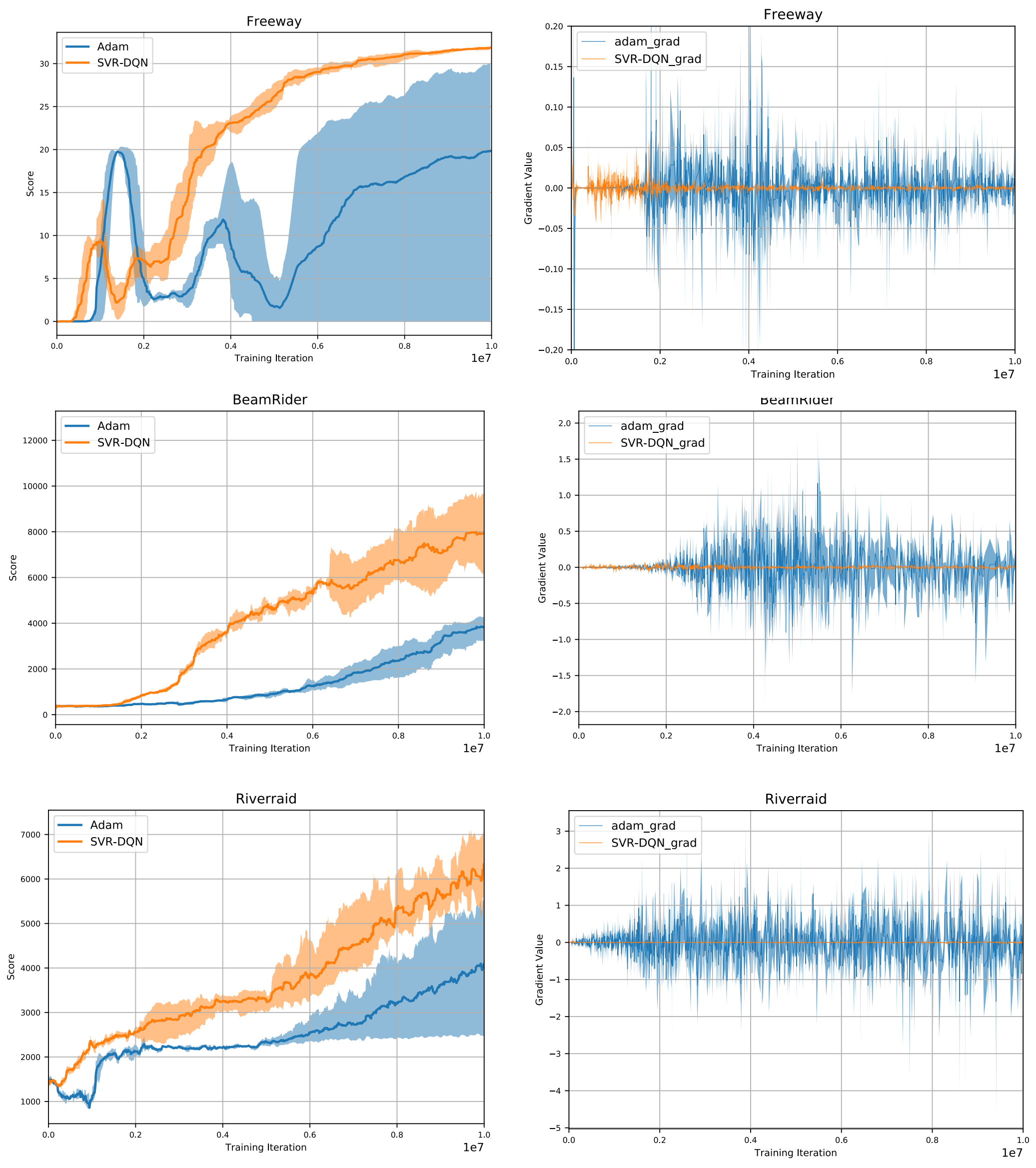


Figure 3.3: The left column shows the learning curves (in raw score) for the Double DQN with Adam optimizer (blue), SVR-DQN optimizer (yellow), on 3 games of the Atari benchmark suite. The bold lines are averaged over 6 independent learning trials (6 different seeds). The performance test using ϵ-greedy policy with 10 million iterations. The shaded area presents one standard deviation. The right column shows that when applied SVR-DQN, the variance of averaged gradient estimation is largely reduced, performance improves, and less variability is observed. 30 no-op evaluation is used and moving average over 4 points is applied. Here x-axis denotes the number of training frames while y-axis denotes the evaluation score in the game.

CHAPTER 4: ARTIFICIAL TRAJECTORIES DIVERGENCE MINIZATION

In this chapter, we describe our work on improving sample efficiency for training reinforcement learning models. Extreme inefficiency is another key problem that prevents reinforcement learning from large scale usage. We leverage the advantages of self-imitation policies [Gangwani et al., 2018] and the idea of Monte Carlo policy evaluation [Fonteneau et al., 2010, Fonteneau et al., 2013] to improve the training procedure.

The remaining of this chapter are organized as two parts. First, we present our approach, including the algorithm description and analysis. Second, we present our experiment experimental results. This includes experiment settings, evaluation metrics and performance.

4.1 SELF-IMITATION POLICIES

Recall in the proof of policy gradient theorem (equation 2.8), we have defined the value $p(s_t = s|s_0, \pi)$, denoting the probability that arriving state s when starting from state s_0 under policy π. Hereafter, we assume the initial state s_0 is fixed and we could omit the state s_0 in the notation as $p(s_t = s|\pi)$. Same as the policy gradient theorem, we could define the unnormalized γ-discounted state visitation distribution and state-action visitation distribution as:

$$\rho^\pi(s) = \sum_{t=0}^{T} \gamma^t p(s_t = s|\pi)$$
$$\rho^\pi(s,a) = \rho^\pi(s)\pi(a|s)$$

Although the policy $\pi(a|s)$ is given as a conditional distribution, the behaviour of policy is characterized by the state-action visitation distribution, which fully determines the expected return $\mathbb{E}_{\rho^\pi(s,a)}[r(s,a)]$. Therefore, it is a intuitive idea to learn a policy with good performance from an ‘expert’ by mimicing the expert’s action-state visitation distribution. Generative adversarial imitation learning (GAIL) [Ho and Ermon, 2016] extracts a policy from data as if it were obtained by following inverse reinforcement learning (IRL) [Ng et al., 2000]. More efficiently, previous work forgoes the external supervision and uses the past experience of the agent itself as the demonstration data [Gangwani et al., 2018]. To be specific, suppose we have the access to an expert policy π_E. In order to make the agent’s policy π similar to expert’s policy π_E, it is natural to minimize the divergence for the improvement of the agent's policy since $D(\rho^{\pi}, \rho^{\pi_E})$ captures the similarity between the two policies:

$$\min_{\pi} D(\rho^{\pi}, \rho^{\pi_E})$$

In practice, we could not have actual access to any 'expert' policies. However, we could maintain a set $\mathcal{M}_E$ which consists of selected trajectories with high returns in previous roll-outs. Instead, we could try to minimize the difference between $\rho^{\pi}(s,a)$ and the non-parametric empirical distributions $\{(s,a)\}_{\mathcal{M}_E}$:

$$\min_{\pi} D(\rho^{\pi}, \{(s,a)\}_{\mathcal{M}_E})$$

A good choice for such divergence will be Jensen-Shannon divergence D_{JS} similar to generative adversarial network (GAN) [Goodfellow et al., 2014]:

$$D_{JS}(\rho^{\pi}, \rho^{\pi_E}) = \max_{d_{\pi}(s,a), d_{\pi_E}(s,a)} \mathbb{E}_{\rho^{\pi}}\left[\log \frac{d_{\pi}(s,a)}{d_{\pi}(s,a) + d_{\pi_E}(s,a)}\right] + \mathbb{E}_{\rho_{\pi_E}}\left[\log \frac{d_{\pi_E}(s,a)}{d_{\pi}(s,a) + d_{\pi_E}(s,a)}\right] \tag{4.1}$$

where $d_{\pi}(s,a)$ and $d_{\pi_E}(s,a)$ are empirical estimation of ρ_{π} and ρ_{π_E}. To optimize the equation 4.1, we have the following theorem in [Gangwani et al., 2018]:

Theorem 4.1. *(**Gradient Approximation**) Let $\rho_{\pi}(s,a)$ and $\rho_{\pi_E}(s,a)$ be the state-action visitation distribution induced by two policies π and π_E respectively. Let d_{π} and d_{π_E} be the surrogates to ρ_{π} and ρ_{π_E}, respectively, obtained by solving equation 4.1. If the policy π is parameterized by θ, the gradient of $D_{JS}(\rho_{\pi}, \rho_{\pi_E})$ with respect to policy parameters (θ) can be approximated as:*

$$\begin{aligned} &\nabla_{\theta} D_{JS}(\rho_{\pi}, \rho_{\pi_E}) \approx \mathbb{E}_{\rho_{\pi}}\left[\nabla_{\theta} \log \pi_{\theta}(a|s) \widetilde{Q}^{\pi}(s,a)\right] \\ &\textit{where } \widetilde{Q}^{\pi}(s,a) = \mathbb{E}_{\rho_{\pi}(s,a)}\left[\sum_{t=0}^{T} \gamma^t \log \frac{d_{\pi}(s_t,a_t)}{d_{\pi}(s_t,a_t) + d_{\pi_E}(s_t,a_t)} \middle| s_0 = s, a_0 = a\right] \end{aligned} \tag{4.2}$$

The proof follows the principles in [Sutton et al., 2000] and could be found in [Gangwani et al., 2018]. Since the equation 4.2 is similar to policy gradient theorem (equation 2.8), we interpolate them together as:

$$\begin{aligned}\nabla J(\theta) &= (1-\nu)\mathbb{E}_{\rho_\pi(s,a)}\left[\nabla_\theta \log \pi_\theta(a|s) Q^r(s,a)\right] - \nu \nabla_\theta D_{JS}(\rho_\pi, \rho_{\pi_E}) \qquad (4.3)\\ &= \mathbb{E}_{\rho_\pi(s,a)}\left[\nabla_\theta \log \pi_\theta(a|s)\left[(1-\nu)Q^r(s,a) + \nu Q^{r^\phi}(s,a)\right]\right]\\ \text{where } Q^{r^\phi}(s,a) &= -\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t \log r^\phi(s_t,a_t)|s_0=s, a_0=a\right]\\ r^\phi(s,a) &= \frac{d_\pi(s,a)}{d_\pi(s,a)+d_{\pi_E}(s,a)} \qquad (4.4)\end{aligned}$$

Note that $-\log r^\phi(s,a)$ is high in the position of $\mathcal{S}\times\mathcal{A}$ where expert policy π_E visits more frequently than the agent's policy π. Equation 4.3 is the major updating step for training self-imitation policy.

4.2 MONTE CARLO POLICY EVALUATION

Suppose we have a set of transitions $\mathcal{F}_n = \{(s_l, a_l, r_l, s'_l\}_{l=1}^n$ from previous roll-outs under a *fixed* policy π. Note that here the subscription l only indicates the index instead of a timestep. We want to estimate the performance $J(\pi)$ from such a transition set. Under closed loop, the possibilities of utilizing model-free Monte Carlo-like approaches to solve this problem have been proven effective [Fonteneau et al., 2010, Fonteneau et al., 2013]. The main idea is to synthesize p artificial trajectories from the "broken trajectory" set $\mathcal{F}_n$ and average the returns of synthetical ones to estimate the policy performance:

$$\mathcal{R}(\mathcal{F}_n) = \frac{1}{p}\sum_{i=1}^{p}\sum_{t=0}^{T-1} r_{l_t^i} \qquad (4.5)$$

where $l_t^i \in \{1, 2, \ldots, n\}$. Now consider that s'_l and r_l are determined by s_l, a_l and ψ_l, where ψ_l is a random variable from an unobservable random process Ψ. The system dynamics could be written as:

$$\begin{aligned} s'_{l+1} &= f(s_l, a_l, \psi_l)\\ r_l &= g(s_l, a_l, \psi_l)\end{aligned}$$

where f and g are time-invariant functions. Let $\mathcal{P}_n = \{(s_l, a_l)\} \in (\mathcal{S}\times\mathcal{A})^n$ be state-action pairs extracted from $\mathcal{F}_n$. We could construct an ensembled set of samples:

$$\widetilde{\mathcal{F}}_n = \{(s_l, a_l, f(s_l, a_l, \widetilde{\psi}_l), g(s_l, a_l, \widetilde{\psi}_l)\}$$

where $\widetilde{\psi}_l$ is a disturbance of ψ_l. At this point, we define the expected value as:

$$E^{\pi}_{\mathcal{P}_n} = \mathbb{E}_{\Psi}[\mathcal{R}(\widetilde{\mathcal{F}})]$$

Now, for $\forall(s, s^-, a, a^-, \psi) \in \mathcal{S}^2 \times \mathcal{A}^2 \times \Psi$, we define Lipcshitz continuity conditions as:

$$\begin{aligned}
\|f(s, a, \psi) - f(s^-, a^-, \psi^-)\|_{\mathcal{S}} &\leq L_f \left(\|s - s^-\|_{\mathcal{S}} + \|a - a^-\|_{\mathcal{A}}\right) \\
|g(s, a, \psi) - g(s^-, a^-, \psi^-)| &\leq L_g \left(\|s - s^-\|_{\mathcal{S}} + \|a - a^-\|_{\mathcal{A}}\right) \\
\|a - a^-\|_{\mathcal{A}} &\leq L_{\mathcal{A}} \left(\|s - s^-\|_{\mathcal{S}} + \|a - a^-\|_{\mathcal{A}}\right)
\end{aligned}$$

where $\|\cdot\|_{\mathcal{S}}$ and $\|\cdot\|_{\mathcal{A}}$ denote the selected norm over state or action space. Actually, the distance metric between state-action pair could be written as $\Delta((s, a), (s^-, a^-)) = \|s - s^-\|_{\mathcal{S}} + \|a - a^-\|_{\mathcal{A}}$. Based on the distance metric, we define the k-sparsity as:

$$\alpha_k(\mathcal{P}_n) = \sup_{(s,a)\in\mathcal{P}_n} \Delta_k(s, a)$$

where $\alpha_k(s, a)$ represents the distance between (s, a) and its k-nearest neighbour. To this end, we list the main theorem [Fonteneau et al., 2010]:

Theorem 4.2. ***Bias of the Monte Carlo Policy Evaluation:***

$$\begin{aligned}
&|J(\pi) - E^{\pi}_{\mathcal{P}_n}| \leq C\alpha_{pT}(\mathcal{P}_n) \\
&\textit{where } C = L_g \sum_{t=0}^{T-1} \sum_{i=0}^{T-i-1} [L_f(1 + L_{\mathcal{A}})]^i
\end{aligned}$$

Theorem 4.3. ***Variance of the Monte Carlo Policy Evaluation:***

$$\begin{aligned}
Var^{\pi}_{\mathcal{P}_n} &= \mathbb{E}_{\Psi}\left[\left(\mathcal{R}(\widetilde{\mathcal{F}}) - E^{\pi}_{\mathcal{P}_n}\right)^2\right] \\
&\leq \left(\frac{\sigma(R)}{\sqrt{p}} + 2C\alpha_{pT}(\mathcal{P}_n)\right)^2 \\
\textit{where } C &= L_g \sum_{t=0}^{T-1} \sum_{i=0}^{T-i-1} [L_f(1 + L_{\mathcal{A}})]^i
\end{aligned}$$

Here $\sigma(R) = \mathbb{E}_{\Psi}[R(\tau)]$.

4.3 SELF-IMITATION LEARNING WITH ARTIFICIAL TRAJECTORIES

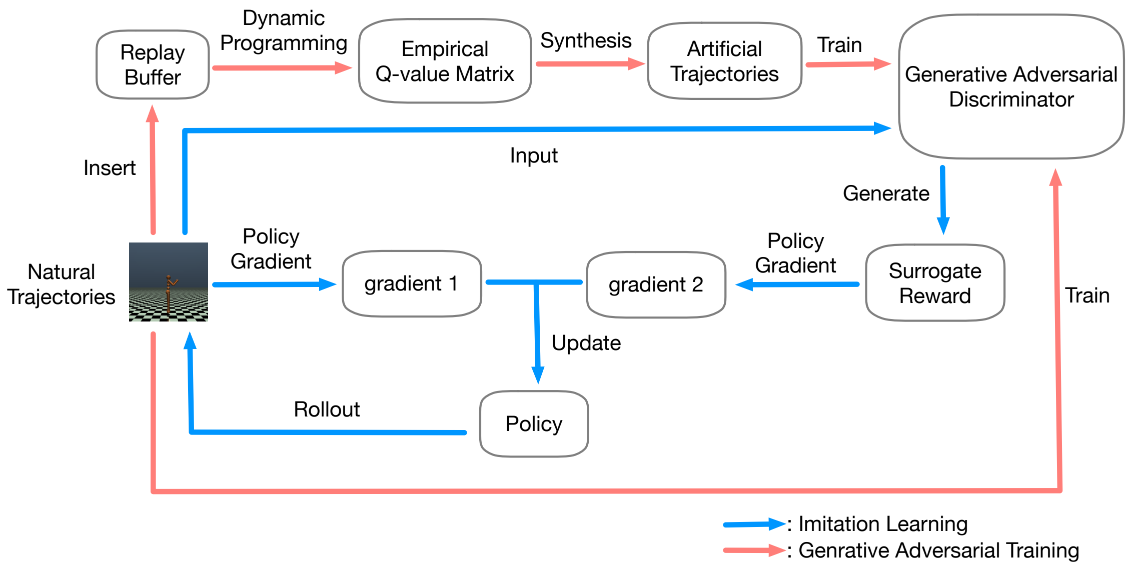


Figure 4.1: Pipeline for self-imitation learning with artificial trajectories. The whole pipeline consists of two parts, generative adversarial training and imitation learning.

To efficiently exploit roll-out samples, we utilize the idea behind Monte Carlo policy evaluation approach to generate artificial trajectories for representing empirical expert state-action visitation distribution. From the previous section, we know that if the discrepancy in the synthesis procedure is controlled well, the synthesized trajectories will be a good representation of expert policy.

To be more specific, we describe the pipeline for our approach in figure 4.1. It consists of two main components:

- **Generative adversarial training**: in each iteration of training, at first, the agent interacts with the environment with the current policy to get *natural* trajectories and true rewards. Then, we get the surrogate reward in equation 4.4 by feeding the natural trajectories into the generative adversarial discriminator. Next, we calculate the policy gradient $\nabla \log \pi(a|s) Q^{\pi}(s,a)$ with respect to true rewards. Meanwhile, we compute the policy gradient in equation 4.1 with respect to surrogate rewards. Finally, we update the parameters of policy according to equation 4.3 with weighted summation of two gradients computed before. The details of this component is introduced in algorithm 4.3.

- **Imitation learning**: within each learning circle, at first, we add the simulated natural transitions into replay buffer. Then, we use dynamic programming to calculate the empirical Q-value for each state-action pairs. Actually, the empirical Q-value is defined as the following:

$$\widehat{Q}(s,a) = r + \max_{(s^-,a^-)\in\mathcal{H}} \left\{ \widehat{Q}(s^-,a^-) \right\}$$
$$\text{where } \mathcal{H} = \{(s^-,a^-) \mid \Delta(s,s^-) \leq \delta, \forall (s^-,a^-) \in \mathcal{F}\}$$

Here $\mathcal{F}$ represents the replay buffer. (s,a,r,s') and $(s^-,a^-,r^-,s^{-\prime})$ are both transitions from natural trajectories. Although the form seems similar to action value function, the meaning is totally different since $s' \neq s^-$. The details of the dynamic programming are shown in algorithm 4.1. Next, we utilize the empirical Q-value matrix to generate artificial trajectories with ϵ-greedy exploration. This step is described in algorithm 4.2. Note that we do not consider action element in the distance measurement. The reason is that we want to minimize the discrepancy as well as encourage the exploration in the generation procedure. Similarly to GAN [Goodfellow et al., 2014], we could train the discriminator with transitions from both natural and synthetical trajecoties.

Algorithm 4.1: Dynamic Programming for Empirical Q-values

```
Require: $\mathcal{F} \sim$ replay buffer for transitions
Require: $\delta \sim$ the threshold for distance measure
Require: $T \sim$ the maximum length of trajectory
Require: $\Delta \sim$ the distance measure between two one-step transitions
1  Initialize the set $\mathcal{B} = \{\}$ for potential artificial trajectory initial state pair
2  Initialize the matrix $M$ with size $|\mathcal{F}| \times T$, where $|\mathcal{F}|$ is the cardinality of replay buffer
3  for $i = 1, \dots, |\mathcal{F}|$ do
4      if $(s_i, a_i, r_i, s'_i)$ is the last transition in its corresponding natural trajectory then
5          $M[i,T] = r_i$
6      end
7  end
8  for $t = T-1, \dots, 1$ do
9      for $i = 1, \dots, |\mathcal{F}|$ do
10         Compute $\mathcal{H} = \{j \mid \Delta(s_i, s_j) < \delta\}$
11         $M[i,t] = r_i + \max_{j\in\mathcal{H}}\{M[j,t+1]\}$
12         if $t = 1$ or $(s_i, a_i)$ is the initial transition in natural trajectory then
13             $\mathcal{B} = \mathcal{B} \cup \{(s_i, t)\}$
14         end
15     end
16 end
17 return matrix $M$ and $\mathcal{B}$
```

Algorithm 4.2: Synthesize Trajectory (SynTraj)

```
Require: $\pi \sim$ current policy
Require: $\delta \sim$ the threshold for distance measure
Require: $\epsilon_1, \epsilon_2 \sim$ the probability for $\epsilon$-greedy, $\epsilon_1 < \epsilon_2$
Require: $s_0 \sim$ starting state
Require: $N \sim$ the number of trajectories
Require: $T \sim$ the maximum length of trajectory
Require: $\mathcal{F} \sim$ set of $n$ one-step transitions
Require: $\Delta \sim$ the distance measure between two one-step transitions
Require: $M \sim$ the empirical Q-value matrix
Require: $\mathcal{B} \sim$ the potential initial pair for artificial trajectory
1  Initialize $i = 0$ (counter for number of trajectories)
2  for $(s_0, t) \in \mathcal{B}$ do
3      $\mathcal{G} = \mathcal{F}$, where $\mathcal{G}$ is the set of not yet used one-step transition in $\mathcal{F}$
4      $s_t^i = s_0$
5      while $t < T$ do
6          Set $a_t^i = \arg\max \pi(a|s_t^i)$
7          Compute $\mathcal{H} = \{j \mid \Delta(s_j, s_t^i) < \delta, \forall (s_j, a_j, r_j, s'_j) \in \mathcal{F}\}$
8          Let $j_{\max} = \underset{j \in \mathcal{H}}{\operatorname{argmax}} M[j, t+1]$
9          $\epsilon = \begin{cases} \epsilon_1, & (s_{j_{\max}}, a_{j_{\max}}, r_{j_{\max}}, s'_{j_{\max}}) \in \mathcal{G} \\ \epsilon_2, & \text{o.w.} \end{cases}$
10         if *uniform random value* $< \epsilon$ then
11             $l_t^i =$ uniformly random select from $\mathcal{H}$
12         else
13             $l_t^i = j_{\max}$
14         end
15         $s_{t+1}^i = s'_{l_t^i}$
16         $\mathcal{G} = \mathcal{G} \setminus \{(s_{l_t^i}, a_{l_t^i}, r(s_{l_t^i}, a_{l_t^i}), s'_{l_t^i})\}$
17         $t = t + 1$
18     end
19     i = i + 1
20 end
21 return $\{(s_{l_t^i}, a_{l_t^i}, r(s_{l_t^i}, a_{l_t^i}), s'_{l_t^i})\}_{t=0}^{t=T-1}$
```

Algorithm 4.3: Synthetic Trajectory self-Imitation LEarning (STILE)

```
Require: $\theta \sim$ initial policy parameters
Require: $\phi \sim$ initial discriminator parameters
Require: Any inputs needed for SynTraj in Algorithm 4.2
1  $\mathcal{F} = \{\}$, empty one-step transition set
2  $\mathcal{M}_S = \{\}$, empty replay memory for synthesis
3  for each iteration do
4      Generate batch of trajectories $\{\tau\}_1^b$ with two rewards for each transition:
         $r_1 = r(s, a)$ and $r_2 = -\log r^{\phi}(s, a)$
5      Add $\{\tau\}_1^b$ into $\mathcal{F}$
6      Synthesize trajectories $\mathcal{J} = \mathrm{SynTraj}(\pi_\theta, \epsilon, s_0, N, T, \mathcal{F}, \Delta)$
7      Choose the top $N$ artificial trajectories $\{\tau^s\}_1^N$ from $\mathcal{J}$
8      Update $\mathcal{M}_S$ using $\{\tau^s\}_1^N$ with priority queue threshold
       // Update policy $\theta$
9      for each minibatch do
10         Compute $g_1 = \nabla_\theta \eta^{r_1}(\pi_\theta)$ using $r_1$ with PPO objective
11         Compute $g_2 = \nabla_\theta \eta^{r_2}(\pi_\theta)$ using $r_2$ with PPO objective
12         Update $\theta$ with $(1 - \nu)g_1 + \nu g_2$ using ADAM
13     end
       // Update self-imitation discriminator $\phi$
14     for each epoch do
15         $s_1 \leftarrow$ Sample mini-batch of $(s, a)$ from $\mathcal{M}_S$
16         $s_2 \leftarrow$ Sample mini-batch of $(s, a)$ from $\{\tau\}_1^b$
17         Update $\phi$ with log-loss objective using $s_1, s_2$
18     end
19 end
```

4.4 EXPERIMENTS

4.4.1 Experiment Settings

To demonstrate the efficiency of our approach, we evaluate proposed algorithm in the simulation environment MuJoCo [Todorov et al., 2012] with OpenAI Gym [Brockman et al., 2016]. Due to its high-dimensional and continuous control setting, it is difficult to learn an agent with good performance. Following the pipeline of OpenAI Baselines [Dhariwal et al., 2017], we have all feed-forward networks with two layers of 64 hidden units and each of them has tanh non-linearity. We use the proximal policy optimization (PPO) [Schulman et al., 2017] of the clipped-surrogate version. The hyperparameters are as following:

- Horizon $T = 1000$
- Discount factor $\gamma = 0.99$

- GAE parameter [Schulman et al., 2015] $\lambda = 0.95$
- PPO internal epochs is 5
- PPO learning rate is 1×10^{-4}
- PPO mini-batch is 64
- Self-imitation learning policy weight factor $\nu = 0.8$
- Size of one-step transition set $n = 2048$

4.4.2 Evaluation

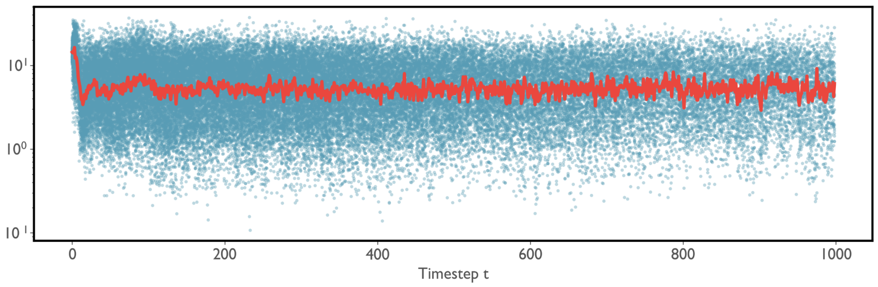


Figure 4.2: Distribution for distance between states within a same transition of environment Walker2d-v2.

Our proposed algorithm integrating self-imitation and artificial synthetical trajectories can obtain better or comparative performance from same data usage. The performance comparison are shown in figure 4.4. In each sub-figure, the learning curve are averaged on three random initiations (seeds). The shaded area depicts the standard deviation. Because different simulation environments have different characteristics, we fine-tune the distance measurement threshold δ for each environment. We found that artificial trajectories with much larger synthetical returns do not help training. We hypothesize the reason is the state-action visitation distribution is too far away from agent's natural policy. A good choice of distance threshold should make a balance between encouraging more acceptable connections between transitions (large threshold) and minimizing the discrepancy from natural trajectories (small threshold). In figure 4.3, we show the l_2 distance values of $\|s - s'\|_2$, where (s, a, r, s') is a transition from natural trajectory. Meanwhile, in figure 4.2, we plot the distribution of l_2 distance between different transitons from natural trajectories. The purpose is to know the rough statistics of distance and to make a good selection of distance threshold. Finally, we choose the threshold of 1.0, 1.5 and 0.1 for Ant-v2, HalfCheetah-v2 and Hopper-v2 respectively, which yields similar but better synthetical returns.

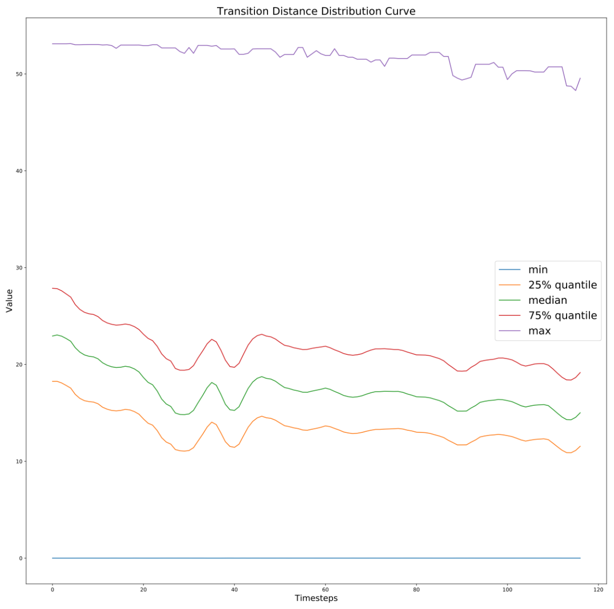


Figure 4.3: Distribution for distance between states from different transitions of environment Walker2d-v2.

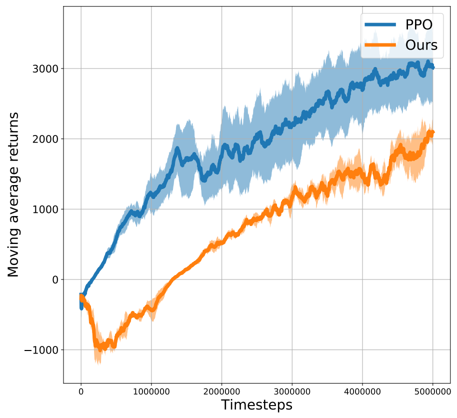
For distance metric, we evaluate l_2 and l_∞ norm separately. Actually, we set the constraints as:

$$\sqrt{\sum_{j=1}^{|\mathcal{S}|}(s_j - s_j^-)^2} \leq \delta, \quad \text{for } l_2$$

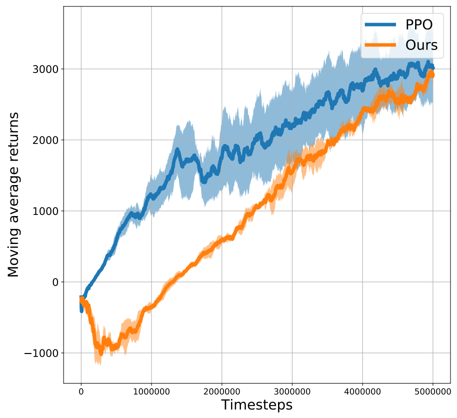
$$\sqrt{|\mathcal{S}| \max_j\{|s_j - s_j^-|^2\}} \leq \delta, \quad \text{for } l_\infty$$

where $s \in \mathcal{S}, s^- \in \mathcal{S}$ and $|\mathcal{S}|$ represents the dimension of state space. We found that l_∞ performs better than l_2 norm. This probably comes from that l_∞ constraint is more strict than l_2. It constrains the difference between each element instead of just a summation. Thus, l_∞ makes the discrepancy between natural and artificial trajectories really small.

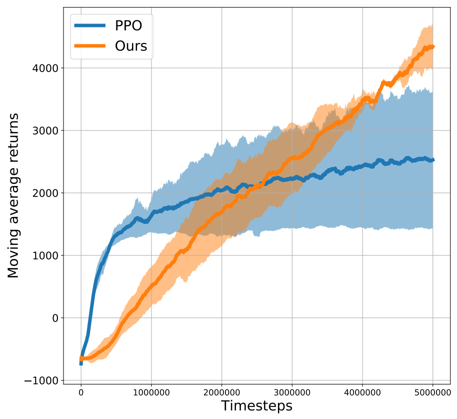
In addition, it is obvious that our approach has a much stable training performance since the variance is really small. We think the benefits come from that the artificial trajectories are similar to each other. As the training proceeds, the agent manages to minimize the divergence between its own policy and the 'expert' policy. Thanks to the similarity between 'expert' trajectories, the agent will not encounter big shift in the distribution and will not go through big variance.



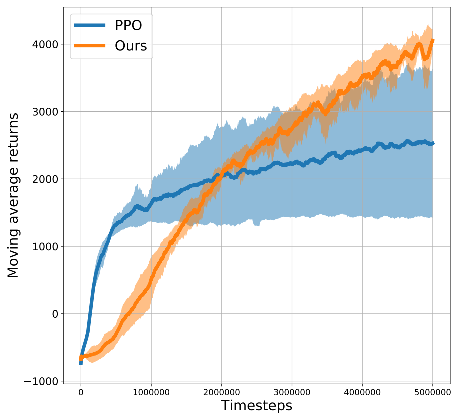
(a) Ant, l_2 norm with $\delta = 1.0$



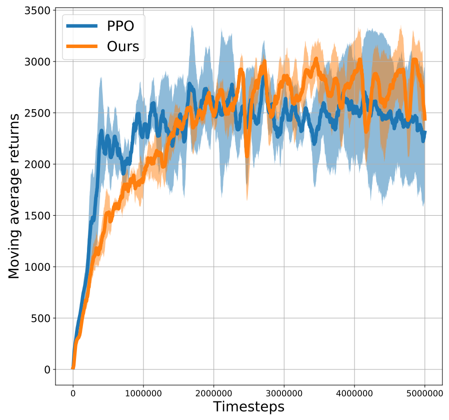
(b) Ant, l_∞ norm with $\delta = 1.0$



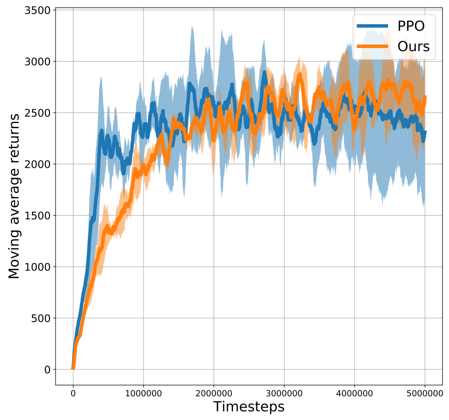
(c) HalfCheetah, l_2 norm with $\delta = 1.5$



(d) HalfCheetah, l_∞ norm with $\delta = 1.5$



(e) Hopper, l_2 norm with $\delta = 0.1$



(f) Hopper, l_∞ norm with $\delta = 0.1$

Figure 4.4: Evaluation on OpenAI Gym Ant-v2, HalfCheetah-v2 and Hopper-v2 environment, which are based on MuJoCo simulation engine.

CHAPTER 5: RELATED WORK

In this chapter, we discuss the related work for the proposed approaches, including variance reduction in optimization and reward learning.

5.1 VARIANCE REDUCTION IN DEEP REINFORCEMENT LEARNING

In recent years, numerous techniques have been proposed to improve the convergence and stability of deep reinforcement learning and optimization method plays a critical role. To accelerate the convergence rate and solve the challenges aforementioned, some important improvements are explored. AdaGrad [Duchi et al., 2011] adapts learning rate with respect to the frequency of parameters, and is well-suited for dealing with sparse data. RMSprop [Tieleman and Hinton, 2012] improves AdaGrad by resolving its radically diminishing learning rates. Adaptive Moment Estimate (Adam) [Kingma and Ba, 2014] combines the advantage of both AdaGrad and RMSprop while keeping momentum technique, empirically outperforming other adaptive learning method algorithms. Under the mechanics of variance control, representative methods such as SAG [Roux et al., 2012] and SDCA [Shalev-Shwartz and Zhang, 2013] are proposed. Recently, second-order statistics optimization algorithms are adopted [Battiti, 1992, Wang and Zhang, 2017]. However, second-order methods are infeasible in practice for high-dimensional training, such as neural network.

Since the existence of variance in deep Q learning procedure can largely deteriorate the performance of the network, there have been a number of techniques proposed to reduce varieties of variance in deep Q learning, such that the convergence, running time and stability are enhanced. The well-known variance in DQN is the Q learning overestimation error, which is first investigated by [Baird III, 1993], who has showed that since action values contain random errors distributed in the interval $[-\epsilon, \epsilon]$. Since the DQN target is obtained using max operator, the expected overestimation errors are bound by $\frac{n-1}{n+1}$, where n is the applicable action numbers given current state s. the intuition nature of overestimation error is that it can cause asymptotically sub-optimal policies, as shown by [Baird III, 1993] and later by [Van Hasselt et al., 2016] that noisy in Arcade Learning Environment [Bellemare et al., 2013] can lead to overestimation. The Double DQN is a possible way to tackle overestimation error which replaces the positive bias with a negative one, where two Q-network are applied for Q action selection and Q function value calculation respectively.

Another variance in DQN is the Target Approximation Error (TAE), which is investigated by [Anschel et al., 2017] that if we denote the term of gap between optimal estimated Q^*

value and current Q value $Q(s,a;\theta) - Q^*(s,a)$, and decompose it as $Q(s,a;\theta) - y_{s,a} + y_{s,a} - \widehat{y}_{s,a} + \widehat{y}_{s,a} - Q^*(s,a)$, where $y_{s,a}$ is the DQN target and $\widehat{y}_{s,a}$ is the true target. TAE is the gap between $Q(s,a;\theta)$ and $y_{s,a}$. The TAE is a result of sub-optimality of θ due to inexact minimization and limited representation power of DQN. An eficient method to reduce TAE variance is Average DQN [Anschel et al., 2017], the key idea is to use the K previously calculated Q-values to estimate the current action-value estimate. The averaging reduces the TAE variance and leads to more stability in learning process.

In addition to the aforementioned kinds of variance in deep Q learning, a recent explored variance is caused by noisy in reward signals in real experiment settings, which is investigated by [Romoff et al., 2018]. In order to reduce reward signal variance, a direct reward estimator $\widehat{R}(s_t)$ is proposed to update the discounted value function instead of sampled reward. Note that in discrete tabular case, the reward estimator is corresponding to using sampled mean.

Our stochastic variance reduction for deep Q learning method differs from all of the aforementioned approaches. The key idea of our method is to reduce the variance caused by approximate gradient estimation, and thus greatly improve the efciency and performance.

5.2 POLICY LEARNING WITH DIVERGENCE MININIZATION

Plenty of algorithms have been proposed through divergence minization. In order to minimizing the information loss between policy updates, Relative Entropy Policy Search (REPS) [Peters et al., 2010] restricts the KL-divergence between distribution of old and new policy state-action pairs. Being formulated in EM framework, policy search has several interesting approaches, for example RWR [Peters and Schaal, 2007]. As stated in a comprehensive exposition [Deisenroth et al., 2013], the M-step here mimics an imitation weighted by returns through minimizing trajectory distribution KL-divergence. Meanwhile, in the transfer learning field, adversarial training has been used to reduce gap between physical and simulated agents [Wulfmeier et al., 2017].

Usually, imitation learning, which trains a policy to produce similar trajectory distribution to demonstrator, requires an external supervision. Human-expert experience is used in autonomous driving [Bojarski et al., 2016] and drone control [Ross et al., 2013]. Combined with human expertise, better performance has been achieved in Atari [Hester et al., 2018] and robotics [Nair et al., 2018]. In the field of inverse reinforcement learning [Ng et al., 2000], human demonstration is also utilized to learn cost functions [Ho and Ermon, 2016]. In addition, non-human knowledge demonstration are as well being exploited, such as MCTS [Silver et al., 2016].

Exploration plays an important role in the success of reinforcement learning models. Methods based on state-action visitation count $N(s, a)$ assign extra bonus for rarely visited pair [Strehl and Littman, 2008]. However, for space with high dimension, function approximation is needed for estimation of pseudo-count based exploration [Fu et al., 2017]. Hindsight Experience Replay [Andrychowicz et al., 2017] combines Q-learning with additional goals, which represented extra rewards.

Our self-imitation learning approach from previous work by incorporating synthesis procedure. The artificial generation process encourages exploration and enjoy the benefits of past expriences.

CHAPTER 6: CONCLUSION AND FUTURE WORK

In this thesis, we study approaches to improve the efficiency for reinforcement learning. By utilizing stochastic variance reduced gradient descent, we improve the Adam optimizer's convergence rate and stable performance. On the other hand, we integrating self-imitation policies with artificial trajectories to exploit roll-out samples efficiently. Evaluation results of these two algorithms demonstrate significant improvement on digital games as well as physical simulation tasks.

Interesting future work includes integrating artificial trajectories with various exploration techniques in reinforcement learning community, such as curiosity-driven exploration. In addition, it is promising to extend synthetical trajectories idea to zeroth order framework, such as evolution strategies framework [Salimans et al., 2017]. Finally, one can combine variance reduction technique with artificial trajectories idea to further improve the agent's performance.

REFERENCES

[Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058.

[Anschel et al., 2017] Anschel, O., Baram, N., and Shimkin, N. (2017). Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 176–185. JMLR. org.

[Baird III, 1993] Baird III, L. C. (1993). Advantage updating. Technical report, WRIGHT LAB WRIGHT-PATTERSON AFB OH.

[Battiti, 1992] Battiti, R. (1992). First-and second-order methods for learning: between steepest descent and newton's method. *Neural computation*, 4(2):141–166.

[Bellemare et al., 2013] 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.

[Bellman, 2010] Bellman, R. (2010). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

[Bertsekas et al., 1995] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.

[Bojarski et al., 2016] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[Deisenroth et al., 2013] 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.

[Dhariwal et al., 2017] 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.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Fonteneau et al., 2010] Fonteneau, R., Murphy, S., Wehenkel, L., and Ernst, D. (2010). Model-free monte carlo-like policy evaluation. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 217–224.

[Fonteneau et al., 2013] Fonteneau, R., Murphy, S. A., Wehenkel, L., and Ernst, D. (2013). Batch mode reinforcement learning based on the synthesis of artificial trajectories. *Annals of operations research*, 208(1):383–416.

[Fu et al., 2017] Fu, J., Co-Reyes, J., and Levine, S. (2017). Ex2: Exploration with exemplar models for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2577–2587.

[Gangwani et al., 2018] Gangwani, T., Liu, Q., and Peng, J. (2018). Learning self-imitating diverse policies. *arXiv preprint arXiv:1805.10309*.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Greensmith et al., 2004] Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530.

[Hahnloser et al., 2000] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947.

[Hahnloser and Seung, 2001] Hahnloser, R. H. and Seung, H. S. (2001). Permitted and forbidden sets in symmetric threshold-linear networks. In *Advances in Neural Information Processing Systems*, pages 217–223.

[Hester et al., 2018] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573.

[Johnson and Zhang, 2013] Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[Lee, 2008] Lee, D. (2008). Game theory and neural basis of social decision making. *Nature neuroscience*, 11(4):404.

[Lin, 1992] Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

[Littman, 1994] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

[Nair et al., 2018] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE.

[Ng et al., 2000] Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2.

[Peters et al., 2010] Peters, J., Mulling, K., and Altun, Y. (2010). Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[Peters and Schaal, 2007] Peters, J. and Schaal, S. (2007). Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750. ACM.

[Romoff et al., 2018] Romoff, J., Piché, A., Henderson, P., Francois-Lavet, V., and Pineau, J. (2018). Reward estimation for variance reduction in deep reinforcement learning. *arXiv preprint arXiv:1805.03359*.

[Ross et al., 2013] Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE.

[Roux et al., 2012] Roux, N. L., Schmidt, M., and Bach, F. R. (2012). A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in neural information processing systems*, pages 2663–2671.

[Salimans et al., 2017] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

[Schulman et al., 2015] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

[Shalev-Shwartz and Zhang, 2013] Shalev-Shwartz, S. and Zhang, T. (2013). Accelerated mini-batch stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 378–385.

[Silver et al., 2016] 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 et al., 2017] 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.

[Strehl and Littman, 2008] Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

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

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[Wang and Zhang, 2017] Wang, J. and Zhang, T. (2017). Improved optimization of finite sums with minibatch stochastic variance reduced proximal iterations. *arXiv preprint arXiv:1706.07001*.

[Watkins, 1989] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

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

[Wulfmeier et al., 2017] Wulfmeier, M., Posner, I., and Abbeel, P. (2017). Mutual alignment transfer learning. *arXiv preprint arXiv:1707.07907*.

[Zhao et al., 2019] Zhao, W., Liu, Y., Zhao, X., Qiu, J., and Peng, J. (2019). Approximation gradient error variance reduced optimization. In *Workshop on Reinforcement Learning in Games (RLG) at The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.