UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence
Master Thesis

World Model Policy Gradient

by
Michal Nauman
11087501

January 4, 2021

48 EC
November 2019 - October 2020

Supervisors:
Prof. Frank van Harmelen
Dr. Eric Pauwels
Floris den Hengst

Assessor:
Prof. Herke van Hoof

ABSTRACT

In this thesis, we propose World Model Policy Gradient (WMPG), an approach to reduce the variance of policy gradient estimates using learned world models (WM's). In WMPG, a WM is trained online and used to imagine trajectories. The imagined trajectories are used in two ways. Firstly, to calculate a without-replacement estimator of the policy gradient. Secondly, the return of the imagined trajectories is used as an informed baseline. We compare the proposed approach with AC and MAC on a set of environments of increasing complexity (CartPole, LunarLander and Pong) and find that WMPG has better sample efficiency. Based on these results, we conclude that WMPG can yield increased sample efficiency in cases where a robust latent representation of the environment can be learned.

Contents

1 Introduction **7**
- 1.1 Problem statement . . . 8
- 1.2 Research goal . . . 9
- 1.3 Approach . . . 9
- 1.4 Contributions . . . 11
- 1.5 Thesis structure . . . 12

2 Background **13**
- 2.1 Markov decision processes . . . 14
- 2.2 Gradient-based policy search . . . 23
- 2.3 Deep reinforcement learning . . . 32

3 World model policy gradient **38**
- 3.1 Problem setting . . . 39
- 3.2 Proposed algorithm . . . 40
- 3.3 Algorithm training . . . 47
- 3.4 Related work . . . 50

4 Experimental setup **52**
- 4.1 Environments . . . 53
- 4.2 Benchmark algorithms . . . 56
- 4.3 Hyperparameter search . . . 58

5 Results **63**
- 5.1 CartPole . . . 64
- 5.2 LunarLander . . . 64
- 5.3 Pong . . . 65
- 5.4 Ablation studies . . . 66

6 Conclusions **69**
- 6.1 Discussion . . . 70
- 6.2 Conclusions and future work . . . 72

A Variance of MCA PG estimator **81**

B Unbiasedness of the without-replacement estimator **85**

C Variance of the without-replacement policy gradient estimator **87**

List of Figures

I WMPG learning cycle 10

I Agent-environment feedback loop 14
II Abstractions 20
III MDP bisimulation 21
IV MDP homomorphism 22
V World models and bisimulation 35

I WMPG Q-value approximation 43

I CartPole problem 53
II LunarLander problem 54
III Atari Pong problem 55
IV Pong preprocessing 55

I CartPole results 64
II LunarLander results 65
III Pong results 66
IV Different number of without-replacement actions 67
V Horizon comparison on CartPole 67
VI Horizon comparison on LunarLander 68

I Zero variance gradient update 82
II Approximated gradient update 1 83
III Approximated gradient update 2 84

List of Tables

I Hyperparameter tuning 58
II AC CartPole setting 59
III MAC CartPole setting 59
IV WMPG CartPole setting 60
V AC LunarLander setting 60
VI MAC LunarLander setting 61
VII WMPG LunarLander setting 61
VIII AC Pong setting 61
IX WMPG Pong setting 62

Acronyms and Symbols

AC Actor-Critic algorithm.

ALE Arcade Learning Environment.

DRL Deep Reinforcement Learning.

FNN Feedforward Neural Network.

MAC Mean Actor-Critic algorithm.

MC Monte-Carlo method.

MCA Monte-Carlo Approximation.

MCPE Monte-Carlo Policy Evaluation.

MCTS Monte-Carlo Tree Search.

MDP Markov Decision Process.

NN Neural Network.

PG Policy Gradient.

POMDP Partially-Observable Markov Decision Process.

RL Reinforcement Learning.

TD Temporal Difference.

TRPO Trust Region Policy Optimization.

WM World Models.

WMPG World Model Policy Gradient.

Chapter 1

Introduction

Advancements in reinforcement learning (RL) allowed for autonomous control in domains like transportation [Xia et al., 2016], trading [Nevmyvaka et al., 2006], industrial production [Gu et al., 2017] or even network optimization [Li and Malik, 2017]. While achieving superhuman results in many problems, the required number of interactions with the environment is often in the scale of hundreds of millions. Such number of samples is especially costly for physical systems, where speed of interaction is predetermined by real world and uninformed actions could potentially damage the agent. This poses a question: can each interactions with environment be more meaningful for learning? RL research seems to continuously answer this question - *yes* - but at a cost. Most often, the cost is expressed in additional computations performed during learning and more entities describing the model. Such approach is consistent with how Richard Sutton envisioned the future of RL: 'For the next tens of years RL will be focused on developing representations of the world'. Moreover, while there are good perspectives in faster simulated computations, speed of interacting with some physical environment will stay fixed.

One of the approaches taken by the RL community is focused on building desirable statistical properties of the estimators that are used for learning and control. It could imply reducing variance or bias of some random component. Most often, decreasing one comes at a cost of the other. For example, TD(λ) [Sutton and Barto, 2018] allows for direct balance between high-variance value estimate of Monte Carlo (MC) and high-bias value estimate of Temporal Difference (TD). Sometimes, methods are developed that only reduce variance without increasing bias. It is well known that in differentiable policy search variance can be reduced either by calculating advantage ([Williams, 1992]; and [Mnih et al., 2016]) or by increasing number of trajectories used for the policy gradient approximation, all without increasing bias ([Mnih et al., 2016]; [Asadi et al., 2017]; and [Kool et al., 2019a]). It was shown multiple times that lower variance of policy gradients increases the sample efficiency of RL agents ([Williams, 1992]; [Greensmith et al., 2004]; [Mnih et al., 2016]; and [Babaeizadeh et al., 2016]).

On the other hand, community is developing algorithms categorized under the name world models (WM) [Ha and Schmidhuber, 2018]. There, through interaction, agent learns a representation of the environment. After learning such representation, one can use most of the techniques that assume knowledge of a model describing the environment. As such, WMs can be leveraged for planning ([Hafner et al., 2018]; [van der Pol et al., 2020]); as a form of regularization for policy search ([Gelada et al., 2019]; [Hafner et al., 2019]); or to perform Monte Carlo Tree Search ([Silver et al., 2018]; [Moerland et al., 2018]). Another benefit of WMs is that, while values always represent certain policy, environment stays constant. As such, the modules of a WM have robust supervision signal. Those properties resulted in very good results, both in terms of sample efficiency and performance after convergence ([Kaiser et al., 2019]; [Hafner et al., 2018]; and [Hafner et al., 2019]).

In this thesis, we tackle the problem of sample efficient gradient-based policy search in reinforcement learning. We propose a method of simulating many trajectories starting from given state using a world model. We show that such approach reduces variance of gradients in differentiable policy search, at a cost of exponentially decaying bias. The code used to produce results presented in this thesis is available at `https://github.com/WMPG-paper/WMPG`. The GitHub contains a Jupyter notebook for fast reproduction.

1.1 Problem statement

We consider a RL agent learning parameters θ of differentiable function representing the policy π_θ. The agent updates parameter values, such that expected returns G are maximized for every state s. In such cases, it is well known that gradient of expected value can be approximated with expected value of gradient with the log-derivative trick [Williams, 1992]:

$$\nabla_\theta V^{\pi_\theta}(s) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[G(s|\tau)\right] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) G(s|\tau)\right] \tag{1.1}$$

This conceptual step is extremely important for general RL settings. Exact calculation of the gradient of expectation at state s would require performing all possible trajectories from π starting at s. Even if the number of trajectories is finite and expectation tractable, it might be inconvenient to 'rewind' the agent back to s, such that another trajectory could be tested. Therefore, a somewhat standard practice is to estimate $E_{\tau\sim\pi_\theta(\tau)}$ with single-sample Monte Carlo approximation [Silver et al., 2014] or with many samples generated by parallel agents [Mnih et al., 2016]. This yields a gradient estimator with variance proportional to $G(s|\tau)$ and inversely proportional to number of samples in the MC approximation. Unfortunately, the distributed approach yields rapidly diminishing returns of sample efficiency as the degree parallelization increases. This property is hard to tackle, as it stems from Monte Carlo approximation rather than the policy gradient itself. As an alternative to parallelization, two approaches have been proposed:

1. [Kool et al., 2019a] perform non-MC gradient approximation with many trajectories sampled without replacement. This is done by allowing the agent to rewind back to some state s and execute a different trajectory

2. [Asadi et al., 2017] consider policy gradient calculation that considers all actions at a given state. Here, values of trajectories are approximated using a Q-network

Both approaches show improvements in terms of performance given number of interactions with the environment. However, they have some drawbacks. Allowing agent to execute many trajectories from each state might be unrealistic if the learner is setup in a physical environment. The computations required for algorithm become also highly dependent on length of the episodes. On the other hand, exact calculation of gradient with a Q-network is unrealistic for discrete action spaces with a lot of actions, let alone continuous action spaces. It also inherits all problems of traditional value estimation, where value network represents all policies it was trained on [Fu et al., 2019].

1.2 Research goal

The goal of this thesis is to revisit the idea sampling many trajectories without replacement for policy gradient approximation, with the ultimate goal of proposing sample efficient method for finding optimal policy in discrete-action continuous-state MDPs. We answer the following questions:

1. What is the effect of approximating policy gradient for given state with Monte Carlo approximator??

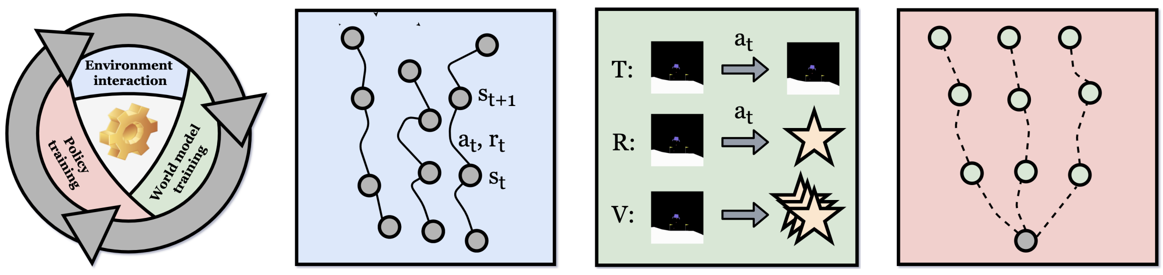
2. Can agent learn the optimal policy from trajectories simulated by an online trained world model?

3. Does simulating those trajectories with reward, transition and value networks perform better than single Q-network?

4. What is the diminishing effect of estimating the policy gradient with more samples? What can be achieved by including only few sampled trajectories?

1.3 Approach

Similarly to ([Kool et al., 2019a]; and [Asadi et al., 2017]), approach used in this thesis allows the agent to approximate the policy gradient by sampling without replacement some $N > 1$ actions at state s, with N potentially equal to the number of all actions in the environment. However, instead of allowing the learner to rewind back and execute other trajectories [Kool et al., 2019a] or learning a Q-network [Asadi et al., 2017], agent learns a world model of the environment it is acting in ([Ha and Schmidhuber, 2018]; [Hafner et al., 2019]; and [Kipf et al., 2019]). Then, the policy is evaluated basing on the unrolled world model, as shown in Figure I.



(a) (b) (c) (d)

Figure I: WMPG learning cycle. a) Agent gathers experience to train the WM, which is in turn used to estimate the policy gradient. b) Agent samples trajectories from the environment. The learning is triggered once enough experiences are gathered. c) Past transitions are used to train the transition and reward networks. Value is trained using only recent transitions. d) For every state in the experience batch, agent samples multiple actions without replacement and imagines the Q-value of those actions.

Values and Q-values under any policy can be related through their definitions. As such, output of a Q-network could be substituted with a compute on transition, reward and value networks, done according to Q-value definition. Such setting, while having more modules, offers many potential advantages. Firstly, contrary to Q-network, transition and rewards are policy-agnostic and generally do not change over time, thus giving stable supervision signal [Gelada et al., 2019]. Given convergence of reward and transition networks, number of interactions with the environment could be limited, with policy updates being done using simulated trajectories [Hafner et al., 2019]. Furthermore, training process could be compartmentalized, with potentially separate tuning and search for different networks representing the model [Ha and Schmidhuber, 2018]. Given changes to the MDP like different reward mapping, only some parts of the model would have to be retrained [Kipf et al., 2019]. Finally, introducing domain knowledge in the form of transition-reward-value relation would impose regularization on the entire model [Gelada et al., 2019].

The designed approach is tested against benchmark algorithms on three control problems: CartPole; LunarLander; and Atari Pong. Each of the environments allows to test different aspects of the proposed algorithm. CartPole and LunarLander represent problems where state representation does not have to be learned. As such, those environments allow to grade the approach independent of the problem of representation learning. On the other hand, in Atari Pong, agent is required to learn policy from pixel data. As such, it corresponds to more realistic problem setting where agent has to learn compact state representation parallel with the policy.

1.4 Contributions

Below, we list the contributions and novel aspects covered in this thesis.

We develop a framework for sample efficient reinforcement learning in an unknown environment. The proposed method unifies world model-based RL with differentiable policy search. This contribution is detailed in Chapter 3.

We show that the proposed approach allows for sampling without replacement trajectories starting from given state and without-replacement sampling approximation of policy gradient at that state. Since the multiple trajectories are simulated, the approach is applicable to physical environments. This contribution is detailed in Subsection 3.2.3.

We show that world models can be used for low-variance approximation of Q-values using TD(λ) on simulated trajectories. We propose that the policy rollout simulation could constitute an alternative approach for off-policy policy gradient This contribution is detailed in Subsection 3.2.2.

We consider the problem of choosing the number of samples in without-replacement sampling approximation of policy gradient at given state. We propose a set of heuristics for such choice. This contribution is detailed in Subsection 3.2.3.

We compare the proposed approach to other policy gradient algorithms on three control problems: CartPole; LunarLander; and Pong. We show that the proposed approach achieves extremely promising results with minimal hyperparameter tuning. This contribution is detailed in Chapter 5.

We include a detailed derivation of the policy gradient theorem for state-action pairs. We directly denote the Monte Carlo approximation of different random variables required for the policy gradient calculation. This contribution is detailed in Section 2.2.

We present a proof for unbiasedness of the without-replacement sampling estimator used in the proposed approach. The sketched proof represents a specific case of proofs shown in [Duffield et al., 2007] and [Kool et al., 2019a]. This contribution is detailed in Appendix B.

We derive the variance of the MC and without-replacement policy gradient approximators for various conditions. This contribution is detailed in Appendix C.

We develop an example MDP to show the effects of single-trajectory policy gradient approximation on policy probability updates. We argument why approximating state policy gradient with multiple trajectories is beneficial for sample efficiency. This contribution is detailed in Appendix A.

1.5 Thesis structure

Further, thesis is divided into five chapters: **Background**; **World model policy gradient**; **Experimental setup**; **Results**; and **Conclusions**.

In the second chapter, we introduce the concepts relevant for the contributions of this thesis. Firstly, we consider Markov Decision Processes, the formalism defining stochastic discrete-time control problems. Further, we describe the paradigm of gradient-based policy search. We conclude with introduction to deep reinforcement learning and world models.

In the third chapter, we introduce the main contribution of this thesis: world model policy gradient (WMPG) algorithm. We describe the main components of WMPG. We discuss how world model can be used to control two main sources of stochasticity in policy gradient estimation: approximation of Q-values and expected value of policy gradient at given state. Finally, we present pseudo-code for main calculations in the proposed approach.

In the fourth chapter, we discuss the experimental setup used in this thesis. Firstly, we describe the environments: CartPole, LunarLander and Pong. Further, we detail the implementation of benchmark algorithms that are used to evaluate the proposed approach. Finally, we discuss the evaluation metric and hyperparameter search scheme used for each environment and algorithm.

In the final two chapters, we show and comment on results of the conducted experiments. Firstly, we compare the proposed approach against benchmark algorithms on the main evaluation metric - number of episodes until certain performance is achieved. In ablation studies, we investigate the effect of variety of hyperparameters used in the proposed approach. We conclude with discussion of future work stemming from this thesis.

Chapter 2

Background

In this chapter, background relevant to the thesis contributions is explored. The chapter is divided into three sections: **Markov decision processes**; **Gradient-based policy search**; and **World models for deep reinforcement learning**.

In the first section, mathematical formalism for stochastic discrete-time control problems is introduced. The main entities that describe the Markov decision process are defined: states, actions, rewards, discount and transitions. Similarly, entities that are instrumental in achieving optimal behaviour are examined. Further, main approaches for finding optimal behaviour in a Markov decision process are explored: planning, where agent's knowledge of the model components is assumed; and reinforcement learning, where the agent does not know the model dynamics. Finally, the problem of state-space compression is introduced and basic approaches reviewed.

In the second section, gradient-based policy search is investigated. Firstly, the policy gradient objective is linked to maximization of state values. It is shown how log-derivative trick allows to approximate policy gradient without iterating over all actions of given state. Finally, usage of Monte Carlo approximation in policy gradient calculation is explained. Further, variance of Monte Carlo policy gradient is calculated. It is shown how variance of the policy gradient approximator depends on the amount of samples used for approximation. Finally, the principles of baseline subtraction are explained.

In the final section, techniques relevant for deep reinforcement learning are reviewed. Firstly, neural networks design and optimization is considered. Later, data preprocessing techniques used for DRL with high-dimensional input data. Finally, the problem of learning a world model is introduced.

2.1 Markov decision processes

Markov decision processes (MDPs) involve an *agent* or *learner* performing actions that yield *reward* dependent on the *state* of the environment that the agent is in, finally moving agent to a different state. This setting implies a loop-type relation between agent and environment that is shown in Figure III.

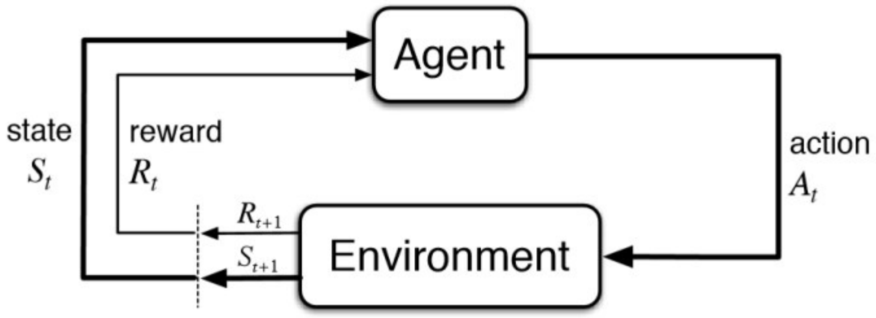


Figure I: Agent-environment feedback loop

Agent can evaluate the feedback generated through interaction with the environment. Learner has to consider that each action affects his future rewards possibilities. That complexity leads to a problem of delayed gratification, where agent has to trade-off the immediate reward with subsequent possibilities created by reaching some state of the environment. At some point, agent is expected to reach the *terminal* state, after which it will no longer perform any actions or continue from some starting state [Puterman, 2014]. The objective of the learner is to gather maximal possible rewards during the *episode*, that is before the terminal state is reached.

2.1.1 Components

The agent starts the episode in one of the starting states $s_{t=0}$ and decides to perform an action a, such that belongs to the set of legal actions A. Then, agent/environment is transitioned to a state $s_{t=1}$ with probability $p_{s'}$. For doing so learner receives a reward r with probability $p_{r'}$. Repeating this process creates a sequence referred to as *trajectory*: $(s_{t=0}, a_{t=0}, r_{t=0})$,$(s_{t=1}, a_{t=1}, r_{t=1})$,...,$(s_{t=T}, a_{t=T}, r_{t=T})$. It is assumed that probabilities of future state transitions and rewards are independent of past observations given knowledge about the present. That implies that for any time index $\forall t \in T$ and for all possible trajectories $P(S_{t+1} = s', R_{t+1} = r'|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(S_{t+1} = s', R_{t+1} = r'|s_t, a_t)$. This is referred to as 'Markov property' [Sutton and Barto, 2018].

State space S is a representation for all possible configurations of the environment. As such, it can be a finite discrete set of symbols or a continuous domain of numbers.

Action space A represents constraints on the agent's behaviour. It can consists of discrete symbols or continuous domain of numbers. If both the action and state spaces are discrete sets, then we refer to the MDP as 'finite'.

Transition mapping T defines future state probability distribution conditioning on state and action in previous step. For deterministic transitions MDP $p(s_{t+1}|s_t, a_t) = 1$ for some $s_t \in S$ and $a_t \in A$.

Reward mapping R defines reward probability distribution conditioning on state and action in previous step. Rewards are real valued numbers.

Discount factor γ is a hyperparameter that tunes the trade-off between immediate rewards and rewards in subsequent episodes. It is defined on domain of $[0, 1]$. For $\gamma = 0$ the agent is myopic, with total focus on immediate rewards.

2.1.2 Policy and Value

Assuming a MDP (S, A, T, R, γ), learner's objective is to maximize episodic *return* which is defined as:

$$G(s_0) = R_{(s_1,s_0,a_0)} + R_{(s_2,s_1,a_1)} + ... + R_{(s_T,s_{T-1},a_{T-1})} = \sum_{t=0}^{T-1} R_{(s_{t+1},s_t,a_t)} \tag{2.1}$$

Where t indices steps in the episode, with T being terminal step and s_0 is some starting state. It might be that the task has no terminal states. In that case, infinite horizon return can be calculated only for $\gamma < 1$:

$$G(s_0) = R_{(s_1,s_0,a_0)} + \gamma R_{(s_2,s_1,a_1)} + \gamma^2 R_{(s_3,s_2,a_2)} + ... = \sum_{t=0}^{\infty} \gamma^t R_{(s_{t+1},s_t,a_t)} \tag{2.2}$$

Policy

Is a mapping determining actions of the learner. Policy is referred to as *deterministic* if it outputs one action per state $\Pi : S \to A \in \mathbb{R}$. On the other hand, if it outputs a probability distribution over actions $\Pi : S \times A \to P(A|s)$ it is called *stochastic.* If the policy is designed to maximize expected returns according to agent's best knowledge, it is called *greedy.*

Value

To evaluate certain policy, we can ask what is the expected discounted return from following policy π in state s and thereafter. This formalization is referred to as *value* function:

$$V^{\pi}(s_i) = \mathbb{E}_{\tau\sim\pi,d}\left[G(s_i|\tau)\right] = \mathbb{E}_{s_{t+1}\sim\pi,d}\left[\sum_{t=i}^{\infty}\gamma^t R_{(s_{t+1},s_t,a_t)}\right] \tag{2.3}$$

Where $\mathbb{E}$ denotes expectations with respect to uncertainty connected to policy stochasticity (π) and MDP transitions (d). τ denotes a trajectory sampled from policy and transitions.

Q-value

Alternatively, we might ask what is the expected discounted return from performing action a in state s, assuming that learner will follow policy π in the following states. Such calculation is referred to as *Q-value* or *action value*:

$$\begin{aligned} Q^{\pi}(s_i,a_i) &= \mathbb{E}_{\tau\sim\pi,d}\left[G(s_i|\tau,a_i)\right] \\ &= \mathbb{E}_{s_{i+1}\sim d}\left[R_{(s_{i+1},s_i,a_i)} + \gamma\mathbb{E}_{\tau\sim\pi,d}\left[G(s_{i+1}|\tau)\right]\right] \\ &= \mathbb{E}_{s_{i+1}\sim d}\left[R_{(s_{i+1},s_i,a_i)} + \gamma V^{\pi}(s_{i+1})\right] \end{aligned} \tag{2.4}$$

2.1.3 Control in MDP

Maximizing expected discounted return for every state $s \in S$ implies finding a policy π^* from the set of all possible policies Π, such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $\forall s \in S$ and $\forall \pi \in \Pi$. Unfortunately, this is a hard problem. Naively, one could try out different policies until being satisfied with returns. But how to know that the chosen strategy is the best one?

Bellman equations

If the policy π^* is *optimal*, it follows $V^{\pi^*}(s) \geq V^{\pi}(s)$. Accordingly, we can note that:

$$V^{\pi^*}(s) = \max_{\pi\in\Pi} V^{\pi}(s) \quad for \quad \forall s \in S \tag{2.5}$$

Knowing that the optimal policy is the one that always chooses the highest valued actions, it follows that:

$$V^{\pi^*}(s) = \max_{a\in A} Q^{\pi^*}(s,a) \tag{2.6}$$

Above equation can be expanded with definition of Q-value. For ease of notation, assume the space of transitions to be finite:

$$V^{\pi^*}(s) = \max_{a\in A} E_{\pi}(G(s)|S=s, A=a) = \max_{a\in A}\sum_{s'} p(s'|s,a)\left(R_{(s',s,a)} + \gamma V^{\pi^*}(s')\right) \tag{2.7}$$

The above equation is known as the *Bellman's optimality equation* for value function [Bellman, 1957]. If all rewards $R_{(s',s,a)}$ and transitions probabilities $p(s'|s,a)$ are known, we can use the Bellman's equation to find unique optimal value functions via iterative calculation of the equation for all states. This approach is referred to as *dynamic programming.*

Planning

Assuming some arbitrary policy π, implied state values can be calculated via:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s'} p(s'|s,a) \left(R_{(s',s,a)} + \gamma V^{\pi}(s')\right) \tag{2.8}$$

After calculating $V^{\pi}(s)$ for all $s \in S$, there may exist a state s for which an adjustment to policy $\pi(s) \rightarrow \pi^*(s)$ would lead to bigger value in that state $V^{\pi^*}(s) > V^{\pi}(s)$. Then, the policy improvement theorem guarantees that $V^{\pi^*}(s) > V^{\pi}(s)\ \forall s \in S$ [Sutton and Barto, 2018]. This property guarantees that the planning algorithms listed below converge to an optimal policy.

Policy iteration

Once some change to the policy has been performed, state values can be recalculated and the policy can be improved again. This process yields a sequence of monotonically improving sequence of policies and values:

$$\pi^0 \xrightarrow{e} V^{\pi^0} \xrightarrow{i} \pi^1 \xrightarrow{e} V^{\pi^1} \xrightarrow{i} \pi^2 \xrightarrow{e} \ldots \xrightarrow{i} \pi^* \xrightarrow{e} V^{\pi^*} \tag{2.9}$$

Where e refers to *policy evaluation* (calculating $V^{\pi}(s)$ for all $s \in S$ given policy π) and i is *policy improvement* (finding a change for in policy π for some state s, such that improves $V(s)$). It is guaranteed that $V^{\pi^{i+1}}(s) > V^{\pi^i}(s)$ for all s and improvement steps. However, waiting for exact convergence of value function for every policy during learning can be costly.

Value iteration

Each cycle of policy iteration assumes evaluating the policy. That implies running Bellman backups until the calculated values indeed represent the chosen policy to some precision. However, it is possible that after some iteration i of policy evaluation the resulting greedy policy will not change. Then, all iterations of policy evaluation that go past i are not required for the algorithm to converge. [Puterman, 2014] shows that i can be restricted to any value, without loosing the convergence properties of the algorithm. A case of policy iteration for which $i = 1$ is referred to as value iteration. In practice, value iteration turns

Bellman optimality equation into a value update rule. As such, the algorithm outputs a greedy policy with respect to optimal state values.

Reinforcement learning

The planning algorithms listed above assume that learner has full knowledge of the environment's underlying MDP. However, such assumption is often very restrictive, as for many applied problems we do not have access to a well-defined model. Reinforcement learning (RL) approaches refer to a strategy of finding optimal policy by iterative informed interaction with the environment [Wiering and Van Otterlo, 2012]. The agent gradually builds knowledge about the values of states and the best policy. This implies that agent's policy at some point of training is dependent on the experience gathered until that point.

Value approximation

Value-based RL aims at calculating optimal value through heuristic based interaction with the environment. There are many ways of estimating state's value, such to balance bias and variance of estimations. *Off-policy* value methods use its exploration policy to generate actions, updating values towards some different target policy. Contrary to this is *on-policy*, where actions are picked from the some policy and values are calculated with respect to this policy. Below, main methods used to estimate state value are described.

Monte Carlo policy evaluation uses the fact that value $V^\pi(s)$ is equal to $\mathbb{E}_{\tau\sim\pi}\left[G(s|\tau)\right]$ [Sutton and Barto, 2018]. As such, to estimate value of some state s_i, one can perform a rollout of this policy and get the sampled $G(s)$. To estimate value under some policy π, algorithm performs 'rollouts' during which it executes π on the environment. The information about gathered rewards G is used to calculate best value estimate:

$$V^\pi(s_0) = \mathbb{E}_{\tau\sim\pi,d}\left[G(s_0|\tau)\right] = \lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\gamma^t R_{(s_{t+1},s_t,a_t)} \tag{2.10}$$

New information allows to adjust the learner's policy. As the sampled rollout comes directly from the policy distribution, the resulting value estimation is unbiased. However, this also implies that the learning is on-policy. The are two main drawbacks to this approach. Firstly, as the value estimate is approximated with a sampled rollout, the estimator will have non-zero variance given any stochasticity. Secondly, the sampled value is sampled from the distribution implied by some policy π. If the policy has changed, old values estimates stop being relevant.

Temporal difference assumes that values are calculated by summing the discounted rewards for some n steps and then adding the discounted value of the $1 + n^{th}$ step. Such approach is labeled as temporal difference with n steps (TD(n)) [Sutton and Barto, 2018]:

$$V_{TD(n)}(s_t) = R_t + \gamma R_{t+1} + ... + \gamma^n R_{t+n} + \gamma^{n+1} V(s_{t+n+1}) \tag{2.11}$$

Such approach allows to re-use knowledge from executing old policies with better effect - every time value is reevaluated, first n steps always represent the current policy. Furthermore, since variance is generated only on the first n steps, the total variance of estimation is lower than for MCPE. Unfortunately, in the beginning of learning the algorithm estimates $V(s_{t+n+1}$ wrongly. Thus, TD(n) introduces a bias that exponentially disappears as algorithm converges towards optimal values. Setting different n balances bias and variance of the value estimates. For $n = 1$, the stochasticity is restricted to the immediate reward. On the other hand, for n equal to the number of steps in an episode, algorithm is equal to MCPE. The obvious drawback of TD(n) is lack of good choice of n. TD(λ) addresses this issue, calculating final value estimate as exponentially-weighted average of TD(n) for different values of n [Sutton and Barto, 2018]. The equation determining this value estimate is given by:

$$V_\lambda(s_0) = (1 - \lambda) \sum_{t=1}^{T-1} \lambda^{t-1} V_{TD(t)} + \lambda^{T-1} G(s_0) \tag{2.12}$$

Where λ has values between $[0, 1]$. For $\lambda = 0$ the method collapses to TD(1); for $\lambda = 1$ the value follows MCPE estimate. For values in-between, value balances bias and variance of TD(n) for different n.

Updating knowledge

After gathering new experience, learner updates its model. New knowledge can be represented either as values or a policy.

Value-based RL assumes that learner bootstraps from the current estimate of the value function, according to $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$ [Sutton and Barto, 2018]. Two basic value-based approaches are *SARSA* and *Q-learning.* Both methods learn Q-values instead of state values. SARSA bootstraps Q-value estimates with the next state's Q-value under the executed policy. That makes SARSA an on-policy TD algorithm, and as such, it's Q-values converge to the Q-values under the executed policy. This implies that, for exploratory policies, the Q-values reflect the negative impact of non-optimal actions. Q-learning bootstraps Q-value estimates with the next state's Q-value under greedy policy. As such, Q-learning is an off-policy algorithm, converging towards greedy policy.

Policy gradient methods assume that policy is some function of state [Peshkin, 2003] that is differentiable wrt. its parameters. Then, policy function can be updated with gradient ascent to maximize $G(s)$ for every encountered s. Noting π_θ as the policy function parametrized by the vector θ, policy search becomes:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta, d} \left[\sum_{t=0}^{\infty} \gamma^t R_{(s_{t+1}, s_t, a_t)} \right] \tag{2.13}$$

More thorough look into policy search methods is done in the further section.

2.1.4 State abstractions

Giunchiglia and Walsh define abstraction as a process of creating mapping transforming the ground problem representation to an *abstract* one, such that predefined properties of the original problem are preserved [Giunchiglia and Walsh, 1992]. Thus, abstraction is a form of compression that is conditioned on the decision-maker's goal.

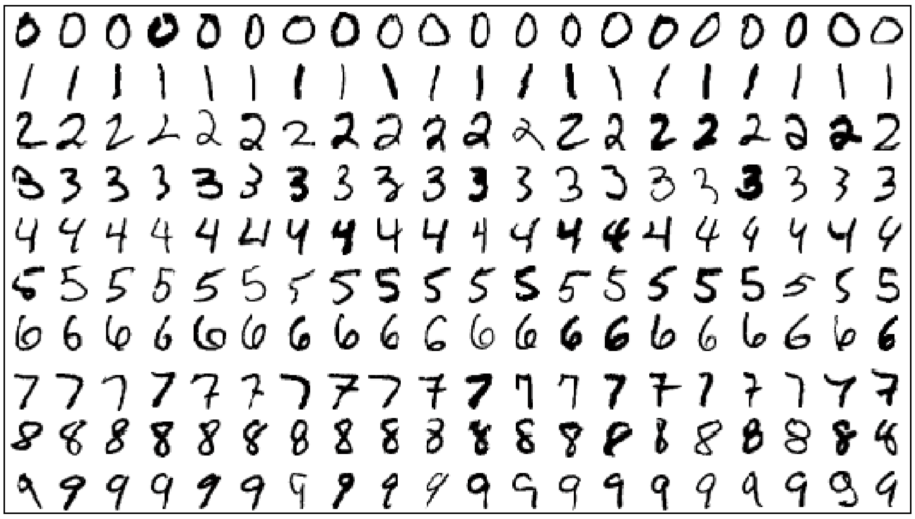


Figure II: How many different entities are on the picture?

Under abstraction model, two originally different entities can become equivalent, as long as this equivalence does not impair the decision-maker's goals. Relevant to this thesis are two abstraction models applied to MDPs: *bisimulation* and *homomorphism.* Both allow the learner to 'lift' the abstract policy back into the original problem, rendering both abstraction strategies *value-preserving* [Li et al., 2006]. In this subchapter, it is assumed that reward is a function of action and state $R_{(s,a)}$.

Bisimulation

The simplest notion of equivalence between states in referred to as bisimulation. First studied for general dynamic systems [Larsen and Skou, 1991], were later tailored and further studied for MDPs in [Li et al., 2006]. Bisimulation $B(s, z)$ is a binary relationship between two MDPs states s and z, such that:

$$B(s, z) \iff \forall a \in A. \left(R_{(s,a)} = R_{(z,a)} \wedge \forall X \in S/B.p(X|s, a) = p(X|s, a)\right) \tag{2.14}$$

From above definition, it follows that in order for two states to be bisimilar, two conditions have to be fulfilled for $\forall a \in A$. Firstly, rewards have to be exactly equal between the bisimilar states. Secondly, transition probability distribution has to be exactly equal under the bisimulation.

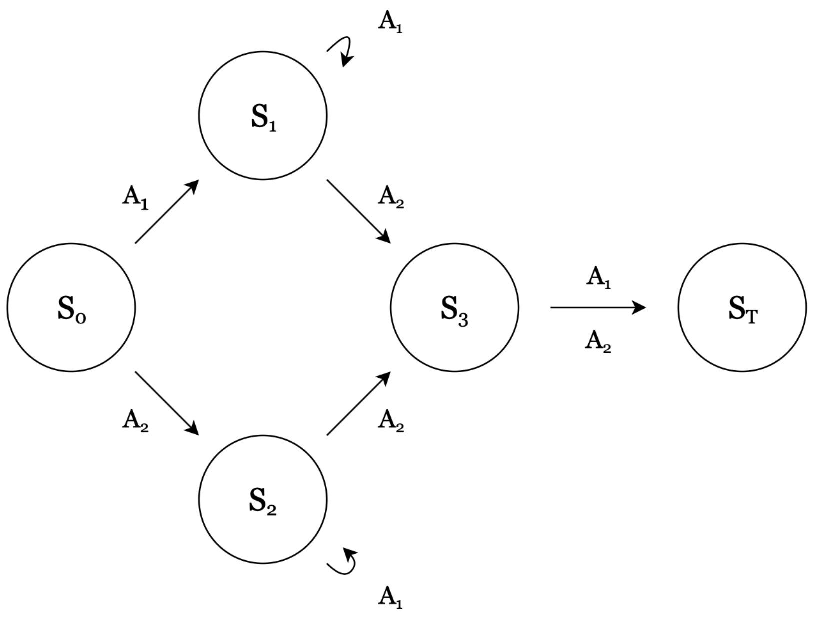


Figure III: Simple deterministic MDP; the learner does not have to distinguish S_2 from S_3 to follow the the policy - in both states A_2 leads to further states.

As follows from the definition, bisimilar states have the same state value $B(s, z) = 1 \Rightarrow (V^{\pi}(s) = V^{\pi}(z))$ for $\forall \pi \in \Pi$. Similarly, we can define bisimulation under policy π, where bisimulation is taken only wrt. the actions from the policy π. A particular case of such on-policy bisimulation is bisimulation under greedy policy:

$$B(s, z) \iff \max_{a \in A} . \left(R_{(s,a)} = R_{(z,a)} \wedge \forall X \in S/B.p(X|s, a) = p(X|s, a)\right) \tag{2.15}$$

One drawback of bisimulation is its binary nature. Only a tiny distortion of either rewards or transitions will result in states being no longer bisimilar. Moreover, finding exact MDP bisimulation is NP-complete [Biza et al., 2020]. To address this issue [Ferns et al., 2004] proposes a *bisimulation metric*, such that utilizes difference between rewards and distance between transition probability distribution to define the metric space:

$$B(s, z) = \max_{a \in A} . \left(\alpha_r (R_{(s,a)} - R_{(z,a)}) + \alpha_T D(p(X|s, a), p(X|z, a)) \right) \tag{2.16}$$

Where α_r is parameter weighting importance of rewards distance and α_T weights importance of transition distance. D is some probability distance metric, which is often assumed to be the *earth-mover* Wasserstein-1 distance [Villani, 2008]. This definition allows for finding *approximate* bisimulations of a given MDP, such where states are allowed to differ to some degree.

Homomorphism

For many practical cases, it might be that the environment exhibits symmetries of interest, but for different actions. In such case, bisimulation as defined above, would fail to compress those symmetries.

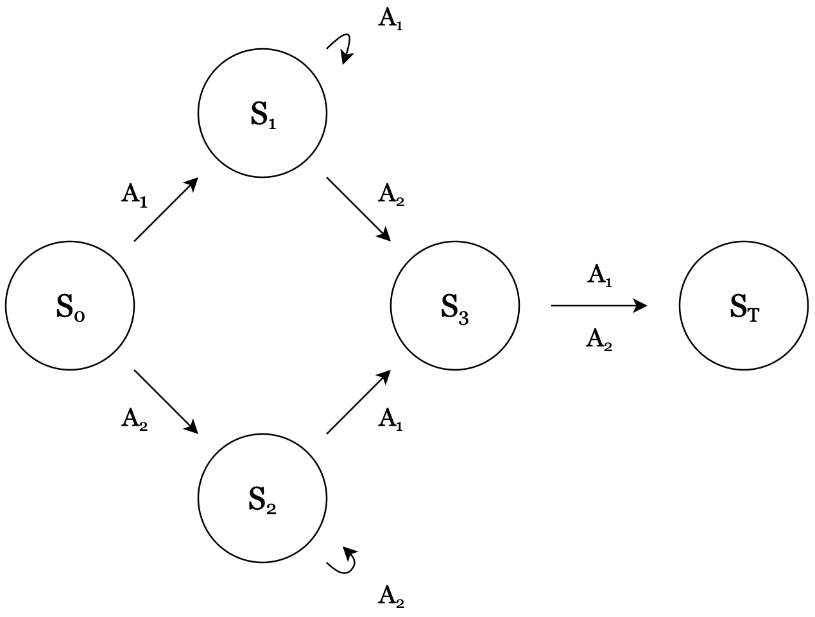


Figure IV: Simple deterministic MDP; no state is bisimilar because of different labels of actions that lead to S_3. Homomorphism would successfully compress the two states.

To remedy this, MDP homomorphisms define a mapping matching states that are bisimilar under certain action mapping. MDP homomorphisms were first defined as exact relation in [Ravindran and Barto, 2003] and [Ravindran and Barto, 2001]. Exact homomorphism H of MDP m is defined as a set of mappings $(\hat{Z}, \hat{A})$, such that:

$$\forall s \in S, \forall a \in A. \left(R_{(s,a)} = R_{(\hat{Z}(s), \hat{A}(a))} \wedge p(s'|s, a) = p(\hat{Z}(s')|\hat{Z}(s), \hat{A}(a)) \right) \tag{2.17}$$

Similarly to bisimulation, MDP homomorphism can be expanded to approximate relation, as shown in [Ravindran and Barto, 2004] and [Taylor et al., 2009]:

$$d(s, a, \hat{Z}, \hat{A}) = \alpha_r \left(R_{(s,a)} - R_{(\hat{Z}(s), \hat{A}(a))} \right) + \alpha_T D \left(p(s'|s, a), p(\hat{Z}(s')|\hat{Z}(s), \hat{A}(a)) \right) \tag{2.18}$$

Where (α_R, α_T) are weights measuring importance of reward and transition distances and D is the Wasserstein-1 distance.

2.2 Gradient-based policy search

Policy gradient (PG) algorithms are used to find a value-maximizing policy in any arbitrary MDP. PG methods define a policy function π_θ that is differentiable wrt. to its parameters θ [Williams, 1992]. Then, values of θ are optimized such that efficiency function $J(\theta)$ is maximized:

$$J(\theta) = V^{\pi_\theta}(s_0) = \sum_{a \in A} \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) = \sum_{\tau \in \omega} p(\tau|\pi_\theta, d) G(s_0|\tau) \tag{2.19}$$

Where $p(\tau|\pi_\theta, d)$ denotes probability of trajectory τ given policy and MDP transitions and ω denotes all possible trajectories. As follows from above, PG optimizes the policy with respect to the starting state values directly, such that the state values under the policy π_θ are maximal [Peshkin, 2003]. Such definition is convenient, since maximization of starting state values guarantees maximization of the following states as well [Bellman, 1957]. Assuming learning rate α, the policy parameters θ are updated with the gradient ascent rule:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{2.20}$$

Where $\nabla_\theta J(\theta)$ denotes gradient of the efficiency function $J(\theta)$ wrt. policy parameters θ and index t relates to an update step of a PG algorithm.

2.2.1 Policy gradient theorem

The policy gradient can be written as:

$$\nabla_\theta J(\theta) = \nabla_\theta V^{\pi_\theta}(s_0) = \sum_{a \in A} \nabla_\theta (\pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a)) \tag{2.21}$$

By first using derivative product rule and then applying the definition of state values, it can be generally rewritten:

$$\begin{aligned}\nabla_\theta V^{\pi_\theta}(s) &= \sum_{a\in A} \nabla_\theta \left(\pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right)\\ &= \sum_{a\in A} \left(\nabla_\theta \pi_\theta(a|s)\right) Q^{\pi_\theta}(s,a) + \pi_\theta(a|s)\left(\nabla_\theta Q^{\pi_\theta}(s,a)\right)\\ &= \sum_{a\in A} \left(\nabla_\theta \pi_\theta(a|s)\right) Q^{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s^{'}} \nabla_\theta p(s^{'}|s,a)(R_{(s^{'},s,a)} + \gamma V^{\pi_\theta}(s'))\end{aligned} \tag{2.22}$$

Where $s' \in S$ denotes some state that agent lands in after performing action a. Since both $p(s^{'}|s,a)$ and $R_{(s^{'},s,a)}$ are independent of the policy, their gradient is equal to zero, and thus the expression is simplified:

$$\nabla_\theta V^{\pi_\theta}(s) = \sum_{a\in A} (\nabla_\theta \pi_\theta(a|s)) Q^{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s^{'}} p(s^{'}|s,a)\gamma \nabla_\theta V^{\pi_\theta}(s') \tag{2.23}$$

Without loss of generality, assume $\gamma = 1$ and denote $\Psi(s) = \sum_{a\in A}\left(\nabla_\theta \pi_\theta(a|s)\right) Q^{\pi_\theta}(s,a)$. Expression is recursively expanded:

$$\begin{aligned}\nabla_\theta V^{\pi_\theta}(s) &= \Psi(s) + \sum_{a\in A} \pi_\theta(a|s) \sum_{s^{'}} p(s^{'}|s,a) V^{\pi_\theta}(s')\\ &= \Psi(s) + \sum_{a\in A} (\pi_\theta(a|s) \sum_{s^{'}} p(s^{'}|s,a)(\Psi(s') + \sum_{a^{'}\in A} (\pi_\theta(a^{'}|s^{'}) \sum_{s^{'}} p(s^{''}|s,a)(...)))\end{aligned} \tag{2.24}$$

Take some state $s^* \in S$. Then, $P(s^*|s,k,\pi_\theta,d)$ can denote the probability that agent is in state s^* after performing a sequence of exactly of k moves, starting from s, while following policy π_θ and given MDP transitions d. Given that, above can be rewritten:

$$\nabla_\theta V^{\pi_\theta}(s) = \sum_{s^*\in S} \sum_{k=0}^{\infty} P(s^*|s,k,\pi_\theta,d)\Psi(s^*) \tag{2.25}$$

Finally, substitute $t(s^*) = \sum_{k=0}^{\infty} P(s^*|s,k,\pi_\theta,d)$ as cumulative likelihood of visiting state s^* given policy π_θ and normalize wrt. S:

$$\begin{aligned}\nabla_\theta V^{\pi_\theta}(s) &= \sum_{s^*\in S} t(s^*)\Psi(s^*)\\ &= \sum_{s^*\in S}\sum_{s^*\in S} t(s^*) \frac{t(s^*)}{\sum_{s^*\in S} t(s^*)} \Psi(s^*)\\ &= \sum_{s^*\in S} t(s^*) \sum_{s^*\in S} \frac{t(s^*)}{\sum_{s^*\in S} t(s^*)} \Psi(s^*)\end{aligned} \tag{2.26}$$

Further, it can be denoted $p(s^*|\pi_\theta, d) = \frac{t(s^*)}{\sum_{s^* \in S} t(s^*)}$. Since $p(s^*|\pi_\theta, d)$ denotes a result of softmax normalization, it represents a stationary probability distribution. Additionally, $\sum_{s^* \in S} t(s^*)$ is a constant. As such, it can be excluded from the optimization problem, knowing that such monotonic transformation only re-scales the solution:

$$\begin{aligned} \nabla_\theta V^{\pi_\theta}(s) &= \sum_{s^* \in S} t(s^*) \sum_{s^* \in S} p(s^*|\pi_\theta, d)\Psi(s^*) \\ &\propto \sum_{s^* \in S} p(s^*|\pi_\theta, d)\Psi(s^*) \\ &= \sum_{s^* \in S} p(s^*|\pi_\theta, d) \sum_{a \in A} Q^{\pi_\theta}(s^*, a)\nabla_\theta \pi_\theta(a|s^*) \end{aligned} \tag{2.27}$$

Above equation pictures the mechanism of policy gradient, where the gradient information flowing from each state-action pair is scaled by the Q-value of this action under policy $Q^{\pi_\theta}(s^*, a)$; and the likelihood of landing in the related state $p(s^*|\pi_\theta)$:

$$\begin{aligned} \nabla_\theta J(\theta) &\propto \sum_{s^* \in S} p(s^*|\pi_\theta, d) \sum_{a \in A} Q^{\pi_\theta}(s^*, a)\nabla_\theta \pi_\theta(a|s^*) \\ &= \mathbb{E}_{s^* \sim \pi_\theta, d}\left[\sum_{a \in A} Q^{\pi_\theta}(s^*, a)\nabla_\theta \pi_\theta(a|s^*)\right] \end{aligned} \tag{2.28}$$

Where $\mathbb{E}_{s^* \sim \pi_\theta, d}$ denotes expectation wrt. the stationary state distribution under policy π_θ and MDP transitions d.

2.2.2 Log-derivative trick

Evaluating the sum $\sum_{a \in A} Q^{\pi_\theta}(s^*, a)\nabla_\theta \pi_\theta(a|s^*)$ requires approximation of $Q^{\pi_\theta}(s, a)$ for all state-action pairs, which is often infeasible. The log-derivative allows to express the summation over all actions as expectation over Q-value weighted with log-probability of an action. When denoting $J(\theta, s)$ as $J(\theta)$ for some s, it follows that:

$$\begin{aligned} \nabla_\theta J(\theta, s) &= \sum_{a \in A} Q^{\pi_\theta}(s, a)\nabla_\theta \pi_\theta(a|s) \\ &= \sum_{a \in A} \pi_\theta(a|s) Q^{\pi_\theta}(s, a)\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ &= \mathbb{E}_{a \sim \pi_\theta}\left[Q^{\pi_\theta}(s, a)\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}\right] \\ &= \mathbb{E}_{a \sim \pi_\theta}\left[Q^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(a|s)\right] \end{aligned} \tag{2.29}$$

This step is conceptually important, as it allows to calculate value of a policy as expectation of many rollouts, which is consistent with the constraint that agent often can perform only one action at a time. Moreover, the exact value of $Q^{\pi_\theta}(s,a)$ is often unknown. Then, the Q-value can be expressed as expected sum of returns $\mathbb{E}_{\tau\sim\pi_\theta,d}\left[G(s|\tau,a)\right]$:

$$\nabla_\theta J(\theta,s) = \mathbb{E}_{a\sim\pi_\theta}\left[\mathbb{E}_{\tau\sim\pi_\theta,d}\left[G(s|\tau,a_i)\right]\nabla_\theta \log\pi_\theta(a|s)\right] \tag{2.30}$$

Above equations allow to write final definition of PG estimator:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\nabla_\theta J(\theta,s)\right]\\ &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta,s,a)\right]\right]\\ &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta\log\pi_\theta(a|s)\right]\right]\\ &= \sum_{s\in S} p(s|\pi_\theta,d)\sum_{a\in A}\pi_\theta(a|s)Q^{\pi_\theta}(s,a)\nabla_\theta\log\pi_\theta(a|s)\end{aligned} \tag{2.31}$$

Alternative interpretation of the log-derivative trick stems from defining $V^{\pi_\theta}(s_0)$ as $\mathbb{E}_{\tau\sim\pi,d}\left[G(s|\tau)\right]$ at the start of the calculations. Denote ω as the number of all possible trajectories and use the definition of continuous expectations:

$$\begin{aligned}\nabla_\theta\mathbb{E}_{\tau\sim\pi_\theta,d}\left[G(s_0|\tau)\right] &= \nabla_\theta\int p(\tau|\pi_\theta,d)G(s_0|\tau)\,d\tau\\ &= \int\nabla_\theta p(\tau|\pi_\theta,d)G(s_0|\tau)\,d\tau\\ &= \int\pi_\theta(\tau)\frac{\nabla_\theta p(\tau|\pi_\theta,d)}{p(\tau|\pi_\theta,d)}G(s_0|\tau)\,d\tau\\ &= \int p(\tau|\pi_\theta,d)\nabla_\theta\log p(\tau|\pi_\theta,d)G(s_0|\tau)\,d\tau\\ &= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[\nabla_\theta\log p(\tau|\pi_\theta,d)G(s_0|\tau)\right]\end{aligned} \tag{2.32}$$

In such setting, the log-derivative trick allows to express gradient of the expectations as expectations of the gradient. The above notation is used for MDPs with continuous actions spaces, where there are infinite number of possible trajectories τ. However, it is pretty unpractical for implementation, as in general $p(\tau|\pi_\theta,d)$ are not known.

2.2.3 Monte Carlo approximation

The final expanded form of the policy gradient for stochastic environments is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\pi_\theta,d}\left[\mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta\log\pi_\theta(a|s)\right]\right] \tag{2.33}$$

Above expectations are extremely hard to calculate exactly. Even given full knowledge of the underlying MDP and $Q^{\pi_\theta}(s,a)$, the calculation would require iterating over all possible state-action pairs. Because of that, the expectations are approximated with Monte Carlo approximation (MCA) [Metropolis and Ulam, 1949]:

$$\mathbb{E}_{x\sim X}\left[f(x)\right] = \lim_{N\to\infty}\frac{1}{N}\sum f(x_n) \tag{2.34}$$

MCA is particularly useful when underlying probabilities are not known. Instead, when sampling infinitely many times from some distribution X, proportions of some value x in the sample reflect the probability of sampling x. As such, MCA approximates the underlying probabilities by sampling from the distribution. Given that the agent learns on-policy, i.e. the training data is distributed according to π_θ, MCA allows to drop both $\mathbb{E}_{s\sim\pi_\theta,d}$ and $\mathbb{E}_{a\sim\pi_\theta}$:

$$\nabla_\theta J(\theta) = \lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} Q^{\pi_\theta}(s_n,a_n)\nabla_\theta \log\pi_\theta(a_n|s_n) \tag{2.35}$$

Where $s_n \sim \pi_\theta, d$ and $a_n \sim \pi_\theta(s_n)$. Furthermore, assuming that $Q^{\pi_\theta}(s_n,a_n)$ are not known, the approximator can be written as:

$$\nabla_\theta J(\theta) = \lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} G(s_n|\tau,a_n)\nabla_\theta \log\pi_\theta(a_n|s_n) \quad \textit{with} \quad \tau\sim\pi_\theta,d \tag{2.36}$$

However, what is noted above as single sample size N, represents MCA of expectations over multiple uncertainties.

Static state distribution

Given policy π_θ and MDP transitions d, each state has some probability of being visited. By learning from policy rollouts, proportion of s in a batch converges to $p(s^*|\pi_\theta,d)$ as batch size goes to infinity:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\nabla_\theta J(\theta,s)\right]\\ &= \lim_{N_s\to\infty}\frac{1}{N_s}\sum_{n=1}^{N_s}\nabla_\theta J(\theta,s_n) \quad \textit{with} \quad s_n\sim\pi_\theta,d\end{aligned} \tag{2.37}$$

Where N_s represents number of samples used to approximate the expectation over static state distribution. In practice, N_s is the batch size of learning.

Policy distribution

After using the log-derivative trick, value of a policy in a state can be expressed as sample average of many rollouts of π_θ starting from state s:

$$\begin{aligned}\nabla_\theta J(\theta, s) &= \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta, s, a)\right] \\ &= \mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)\right] \\ &= \lim_{N_a\to\infty}\frac{1}{N_a}\sum_{n=1}^{N_a} Q^{\pi_\theta}(s,a_n)\nabla_\theta \log \pi_\theta(a_n|s) \quad \textit{with} \quad a_n \sim \pi_\theta\end{aligned} \tag{2.38}$$

Where N_a represents number of samples used to approximate value of policy in some state s. As noted by [Kool et al., 2019a], it is common practice to set N_a equal to 1. It is questionable whether MCA is an appropriate strategy for this approximation, as MCA is typically used when the underlying probabilities are not known. If both policy and the underlying MDP are deterministic, then both MCA and exact expectation calculation require 1 sample to calculate the state PG. However, if policy or the transitions are stochastic, then MCA requires strictly more samples than exact calculation of expectations combined with sampling without replacement ([Metropolis and Ulam, 1949]; [Kool et al., 2019b]; and [Shi et al., 2020]).

Distribution of Q-values

If Q-values are not known, they can be estimated as sample returns from executing action a in state s, and following π_θ thereafter:

$$\begin{aligned}Q^{\pi_\theta}(s,a) &= \mathbb{E}_{\tau\sim\pi_\theta,d}\left[G(s|\tau,a)\right] \\ &= \lim_{N_q\to\infty}\frac{1}{N_q}\sum_{n=1}^{N_q} G(s|\tau_q,a) \quad \textit{with} \quad \tau_q \sim \pi_\theta, d\end{aligned} \tag{2.39}$$

Where N_q represent number of samples used to approximate Q-value of action a in state s. In general, Q-values can be estimated with any strategy for value estimation mentioned in section 2.

Conclusion

The above analysis yields final policy gradient approximator, which given knowledge about Q-values is equal to:

$$\nabla_\theta J(\theta) \approx \frac{1}{N_s * N_a}\sum_{n_s=1}^{N_s}\sum_{n_a=1}^{N_a}\nabla_\theta \log \pi_\theta(a_{n_a}|s_{n_s})Q^{\pi_\theta}(s_{n_s},a_{n_a}) \tag{2.40}$$

With $s_{n_s} \sim \pi_\theta, d$ and $a_{n_a} \sim \pi_\theta$. If the Q-values are not known, the PG approximator is equal to:

$$\nabla_\theta J(\theta) \approx \frac{1}{N_s * N_a} \sum_{n_s=1}^{N_s} \sum_{n_a=1}^{N_a} \nabla_\theta \log \pi_\theta(a_{n_a}|s_{n_s}) \sum_{n_a=1}^{N_q} \frac{1}{N_q} G(s|\tau_q, a) \tag{2.41}$$

Where $\tau_q \sim \pi_\theta, d$.

2.2.4 Variance of the policy gradient estimator

In this part of the thesis, properties of policy gradient estimator variance are analyzed. It is assumed that Q-values under any policy are known, and thus they do not add to the uncertainty of the estimations. The variance $Var\left[\nabla_\theta J(\theta, s, a)\right]$ can be expanded via definition $Var\left[X\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2$:

$$\begin{aligned} Var\left[\nabla_\theta J(\theta, s, a)\right] &= \mathbb{E}_{a\sim\pi_\theta}\left[(\nabla_\theta J(\theta, s, a))^2\right] - \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta, s, a)\right]^2 \\ &= \mathbb{E}_{a\sim\pi_\theta}\left[(\nabla_\theta J(\theta, s, a))^2\right] - (\nabla_\theta J(\theta, s))^2 \\ &= \mathbb{E}_{a\sim\pi_\theta}\left[(Q^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(a|s))^2\right] - (\nabla_\theta J(\theta, s))^2 \\ &= \sum_{a\in A}\left(\pi_\theta(a|s)(Q^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(a|s))^2\right) - (\nabla_\theta J(\theta, s))^2 \end{aligned} \tag{2.42}$$

When $Var\left[\nabla_\theta J(\theta, s, a)\right]$ is considered, we measure the variance of PG for some specific state, as induced by sampling different actions in that state. Since the PG components are approximated with MC, the variance becomes additionally dependant on the number of samples used in approximation. Noting $\nabla_\theta \hat{J}(\theta, s)$ as MC approximator of $\mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta, s, a)\right]$ for some state s and noting the independence of sampled actions:

$$Var\left[\nabla_\theta \hat{J}(\theta, s)\right] = \frac{1}{N_a} Var\left[\nabla_\theta J(\theta, s, a)\right] \tag{2.43}$$

It follows that increasing N_a (number of sampled actions for state s) decreases the variance only for non-deterministic policies, for which $Var\left[\nabla_\theta J(\theta, s, a)\right] \neq 0$. Furthermore, N_a has a nonlinear effect on the variance of policy gradient estimator. The bigger $Var\left[\nabla_\theta J(\theta, s, a)\right]$ is, the bigger the variance reduction as a result of increasing N_a. On the other hand, if $Var\left[\nabla_\theta J(\theta, s, a)\right] \neq 0$ is held fixed, then variance reduction stemming from increase of N_a will diminish as $N_a \to \infty$:

$$\nabla_{Na} Var\left[\nabla_\theta \hat{J}(\theta, s)\right] = -\frac{1}{N_a^2} Var\left[\nabla_\theta J(\theta, s, a)\right] \tag{2.44}$$

This poses a hard limit on the performance of parallel learning, where adding more agents yields a diminishing return on the sample efficiency. The final PG approximator $\nabla_\theta \hat{J}(\theta)$ is calculated as sample average of $\nabla_\theta \hat{J}(\theta, s)$ for different sampled s. Denoting D_s as sample of s and making no assumptions about independence of concurrent samples, it can be written:

$$\begin{aligned} Var\left[\nabla_\theta \hat{J}(\theta)\right] &= \frac{1}{N_s^2}\sum_{s\in D_s}\sum_{s'\in D_s} Cov\left[\nabla_\theta \hat{J}(\theta,s), \nabla_\theta \hat{J}(\theta,s')\right] = \\ &= \frac{1}{N_s^2}\sum_{s\in D_s}\left(Var\left[\nabla_\theta \hat{J}(\theta,s)\right] + \sum_{s'\neq s\in D_s} Cov\left[\nabla_\theta \hat{J}(\theta,s), \nabla_\theta \hat{J}(\theta,s')\right]\right) \end{aligned} \tag{2.45}$$

With:

$$\begin{aligned} Var\left[\nabla_\theta \hat{J}(\theta,s)\right] &= \frac{1}{N_a} Var\left[\nabla_\theta J(\theta,s,a)\right] \\ &= \frac{1}{N_a}\left(\sum_{a\in A}\left(\pi_\theta(a|s)(Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s))^2\right) - (\nabla_\theta J(\theta,s))^2\right) \end{aligned} \tag{2.46}$$

In practice, the N_s samples are often dependent because the samples represent one or multiple trajectories. In such case, after knowing the starting state of some trajectory, the likelihood of some other states in the trajectory is affected by the MDP constraints. For example, it might be impossible to come back to some state s, and thus sample it again.

Baseline variance reduction

Variance of random variable X is smaller for aX than bX, as long as $|a| < |b|$. It is well established that this property can be leveraged to reduce the variance of policy gradient approximation ([Williams, 1992]; [Peshkin, 2003]; and [Sutton and Barto, 2018]). Technique known as the *baseline variance reduction* refers to a policy gradient transformation, where some value $b(s)$ is subtracted from Q-value of action a in state s according to:

$$\begin{aligned} \nabla_\theta J(\theta, b(s)) &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\nabla_\theta J(\theta,s,b(s))\right] \\ &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\sum_{a\in A}(Q^{\pi_\theta}(s,a) - b(s))\nabla_\theta \pi_\theta(a|s)\right] \\ &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\sum_{a\in A}(Q^{\pi_\theta}(s,a))\nabla_\theta \pi_\theta(a|s)\right] - \mathbb{E}_{s\sim\pi_\theta,d}\left[\sum_{a\in A} b(s)\nabla_\theta \pi_\theta(a|s)\right] \\ &= \mathbb{E}_{s\sim\pi_\theta,d}\left[\nabla_\theta J(\theta,s)\right] - \mathbb{E}_{s\sim\pi_\theta,d}\left[\sum_{a\in A} b(s)\nabla_\theta \pi_\theta(a|s)\right] \end{aligned} \tag{2.47}$$

Using Leibniz integral rule and assuming that $b(s)$ does not vary with a, it can be shown that the operation does not create bias in expectations:

$$\begin{aligned}\mathbb{E}_{s\sim\pi_\theta,d}\left[\sum_{a\in A} b(s)\nabla_\theta\pi_\theta(a|s)\right] &= \mathbb{E}_{s\sim\pi_\theta,d}\left[b(s)\nabla_\theta\sum_{a\in A}\pi_\theta(a|s)\right]\\ &= \mathbb{E}_{s\sim\pi_\theta,d}\left[b(s)\nabla_\theta 1\right] = \mathbb{E}_{s\sim\pi_\theta,d}\left[0\right] = 0\end{aligned} \tag{2.48}$$

Thus, it follows that:

$$\mathbb{E}_{s\sim\pi_\theta,d}\left[\nabla_\theta J(\theta,s)\right] = \mathbb{E}_{s\sim\pi_\theta,d}\left[\nabla_\theta J(\theta,s,b(s))\right] \tag{2.49}$$

As well as for some state s:

$$\mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta,s,a)\right] = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta,s,a,b(s))\right] \tag{2.50}$$

Above implies that, in expectation with regards to state distribution and the policy, baselined PG is equal to the PG without baseline. Further, variance of baselined policy gradient for some state s can be calculated:

$$\begin{aligned}&Var\left[\nabla_\theta J(\theta,s,a,b(s))\right] =\\ &= \mathbb{E}_{a\sim\pi_\theta}\left[(\nabla_\theta J(\theta,s,a,b(s)))^2\right] - \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta,s,a,b(s))\right]^2\\ &= \mathbb{E}_{a\sim\pi_\theta}\left[((Q^{\pi_\theta}(s,a)-b(s))\nabla_\theta\log\pi_\theta(a|s))^2\right] - \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta,s,a)\right]^2\\ &= \left(\sum_{a\in A}\pi_\theta(a|s)((Q^{\pi_\theta}(s,a)-b(s))\nabla_\theta\log\pi_\theta(a|s))^2\right) - (\nabla_\theta J(\theta,s))^2\end{aligned} \tag{2.51}$$

Now, the change of variance as a result of changes to b can be investigated with the derivative:

$$\begin{aligned}&\nabla_{b(s)}Var\left[\nabla_\theta J(\theta,s,a,b(s))\right] =\\ &= \nabla_{b(s)}\left(\left(\sum_{a\in A}\pi_\theta(a|s)((Q^{\pi_\theta}(s,a)-b(s))\nabla_\theta\log\pi_\theta(a|s))^2\right) - (\nabla_\theta J(\theta,s))^2\right)\\ &= \nabla_{b(s)}\left(\sum_{a\in A}\pi_\theta(a|s)((Q^{\pi_\theta}(s,a)-b(s))\nabla_\theta\log\pi_\theta(a|s))^2\right) - 0\\ &= \nabla_{b(s)}\mathbb{E}_{a\sim\pi_\theta}\left[(Q^{\pi_\theta}(s,a)^2+b(s)^2-2Q^{\pi_\theta}(s,a)b(s))\log\pi_\theta(a|s)^2\right]\\ &= \mathbb{E}_{a\sim\pi_\theta}\left[(2b(s)-2Q^{\pi_\theta}(s,a))\log\pi_\theta(a|s)^2\right]\\ &= 2(\mathbb{E}_{a\sim\pi_\theta}\left[b(s)\log\pi_\theta(a|s)^2\right] - \mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\log\pi_\theta(a|s)^2\right])\end{aligned} \tag{2.52}$$

Thus, given that $\mathbb{E}_{a\sim\pi_\theta}\left[b(s)\log\pi_\theta(a|s)^2\right] \le \mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\log\pi_\theta(a|s)^2\right]$, variance is indeed reduced by introducing b. The extreme point of variance as function of b falls on $\mathbb{E}_{a\sim\pi_\theta}\left[b(s)\log\pi_\theta(a|s)^2\right] = \mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\log\pi_\theta(a|s)^2\right]$. Combining above with equation 2.51 reveals that indeed, this extreme point corresponds to minimum of variance wrt. b. As shown above, baseline can reduce variance of policy gradient, as induced by sampling different actions at a given state.

2.3 Deep reinforcement learning

Deep reinforcement learning (DRL) attracted a lot of attention in the recent years, with some models achieving human or superhuman level of control on various domains ([Mnih et al., 2015]; [Silver et al., 2017]; and [Vinyals et al., 2019]). DRL is a subsection of reinforcement learning, where deep neural networks are used to approximate the components of the underlying RL problem.

2.3.1 Neural networks

Neural networks (NNs) are graph-based computing systems that allow to approximate any non-linear mapping to an arbitrarily small error [Bishop, 2006]. NNs build complex representations by composing a sequence of simpler transformations of states from previous layer. Operations in each layer of the network are computed by *cells*, according to:

$$y = f\left(\sum_i \omega_i x_i + b\right) \tag{2.53}$$

Where y is the output of a cell; f is some non-linear activation function; x_i and ω_i is the input data and the corresponding parameter; and b is the bias. The source of the input data x_i is determined by the edge structure of the network. The design of the network's structure reflects the prior belief about the data or the problem at hand.

Network structures

The design of the network's structure reflects the prior belief about the data or the problem at hand. Feedforward neural networks (FNNs) do not assume any temporal or spatial structure in the data [Bishop, 2006]. FNN is sometimes called *fully-connected* network, as each cell inputs all cells from the previous layer, with input data being treated as the first layer. On the other hand, recurrent neural networks (RNNs) assume that the data is generated sequentially, i.e. that it forms a directed graph [Greff et al., 2016]. The network maintains a memory vector which is sequentially updated with information from new samples.

Activations

There is a wide class of activation functions and network structures that guarantee that the resulting NN will be an *universal approximator* ([Gorban and Wunsch, 1998]; [Schäfer and Zimmermann, 2006]; [Hanin, 2019]; and [Heinecke et al., 2020]). This property guarantees that increasing number of parameters in the network decreases the approximation error up to an arbitrarily small number and lays a theoretical guarantee that there exists a architecture that is expressive enough for mapping of arbitrary complexity. Activation functions refer to the non-linearities in the network. Activations are fundamental to expressive power of a NN, as without them NN would perform only linear transformations. On the other hand, usage of activations makes the optimization problem non-concave, increasing the difficulty of the optimization task itself.

Network training

The output of the NN is evaluated against real value of the approximated function with a *loss* function $\mathcal{L}$. First, ω is initialized randomly, preferably such that the distribution of the data in each layer is standard Gaussian. Then, ω is iteratively updated with gradient descent:

$$\omega_{t+1} = \omega_t - \alpha \nabla_\omega \mathcal{L}(\omega) \tag{2.54}$$

Often, it is impossible to evaluate $\nabla_\omega \mathcal{L}(\omega)$ for the entire data at once, and thus it is computed in *batches* of samples. Then, similarly to MCA, iterating over the entire dataset guarantees that the gradient estimation converges to the true gradient [Becker et al., 1988]. The stochasticity of batch gradient approximation can have adverse effects on the learning. For example, it might be that the expected value of gradient wrt. some parameter ω_i is equal to 0, i.e. $\nabla_{\omega_i} \mathcal{L}(\omega) = 0$ when calculated over the entire data. Unfortunately, given that batch approximation of $\nabla_{\omega_i} \mathcal{L}(\omega)$ is continuously distributed, sampled $\nabla_{\omega_i} \mathcal{L}(\omega)$ is always non-zero, and thus ω_i is continuously updated in directions that eventually cancel out. This unnecessary compute can be alleviated by maintaining a per-parameter learning rate and approximating first moments of the gradients wrt. individual parameters. Then, gradients with stronger expected values and low variance can be have higher learning rate. Similarly, learning rates of gradients that are oscillating around zero can be themselves set to 0. This mechanism is used by various *optimizers* ([Duchi et al., 2011]; [Tieleman and Hinton, 2012]; and [Kingma and Ba, 2014]).

2.3.2 RL with high-dimensional sensory data

Often, compact MDP representation of a problem is not known. Then, a common practice is to define MDP over sensory data that captures all the variation in the problem representation ([Mnih et al., 2015]; and [Xia et al., 2016]). Example of that is finding optimal control in video games, with state representation defined over graphic content of the game's screen. While the information density of such screen is often huge, the features required to follow an optimal policy are often compact. Fine-tuning all components of the data pipeline in such high-dimensional settings seems to highly impact results of the experiments [Mnih et al., 2015], with a huge variety of techniques being considered.

Rescaling

Many sandbox environments have different scale of images, with some sizes outright tailored towards human preference ([Bellemare et al., 2013]; and [Brockman et al., 2016]). As such, the information density of input can often be reduced by down-scaling the frames, without impairing performance of a converged agent [Mnih et al., 2015].

Stacking

When looking at a single frame, some environments might become partially-observable. For example, single frame of game *Pong* does not reveal the velocity or direction of the ball. Such environments can be transformed to fully-observable by stacking multiple frames as one observation.

Skipping

Many sandbox RL environments run 30/60 frames per second and allow the agent to execute an action every frame. This results in lengthy episodes with sparse reward signal and only atomic changes to the environment after taking each action. Frame skipping refers to the number of times an action is repeated before the agent is allowed to choose a new action and is shown to be an important hyperparameter for Arcade Learning Environment [Braylan et al., 2015].

Non-terminating

It seems natural to terminate an episode when the game is over. Interestingly, as shown in ([Mnih et al., 2015]; and [Machado et al., 2018]) not terminating episodes can result in better performance for off-policy learners. Non-terminating is not practical when used with Monte Carlo value approximation, as its computational complexity is dependent on the length of episodes.

2.3.3 World models

WM assume a reinforcement learning setup in which agent tries to learn the underlying MDP ([Ha and Schmidhuber, 2018]; [Kaiser et al., 2019]; and [Kipf et al., 2019]). The approach is shown to offer many advantages over model-free RL: compressed structured representation reduces size of the search and creates a strong inductive bias for generalization in novel environment configurations; compartmentalization of networks allows for transfer learning between different reward and transition mappings. Creating a structured representation of the environment is a challenging problem. While there are variety of approaches for representation learning in RL, three strategies seem to be dominant in the recent literature: *reconstruction*; *bisimulation*; and *information bottleneck*.

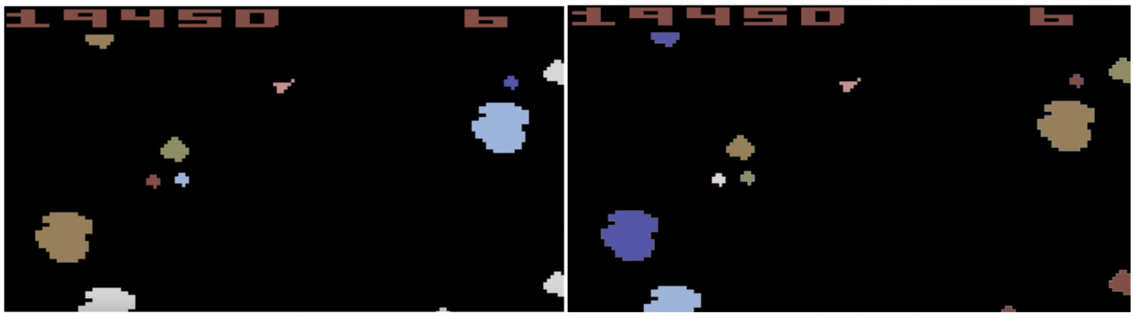


Figure V: Since the asteroids are coloured differently, the image representation of the two states is different. From the perspective of optimal policy, the states are identical

Reconstruction

The first introduced method of training world model learns compact latent representations by reconstructing observations of the MDP ([Ha and Schmidhuber, 2018]; and [Hafner et al., 2018]). There, the high-dimensional frame is first encoded into some compact latent embedding, then to be decoded back to dimensionality of original input. Loss is calculated as some distance metric between the original and decoded frames. As such, the lower the reconstruction loss, the lower the information loss due to compression. This fact gives guarantees regarding the quality of encoded representations.

The reconstruction approach seems to be dominant in recent literature, especially when sample efficiency of the RL algorithm is measured ([Ha and Schmidhuber, 2018]; [Hafner et al., 2018]; [Igl et al., 2018]; [Hafner et al., 2019]; and [Kaiser et al., 2019]). It it is not clear why reconstruction performance is better than other WM training strategies, but there are few recurrent arguments ([Ha and Schmidhuber, 2018]; [Hafner et al., 2019] and [Kaiser et al., 2019]). Firstly, the reconstruction supervision signal is independent of other components of the agent. Secondly, with reconstruction trained independently to other components, hyperparameters of the encoder can be searched with ease. Furthermore, with AEs being a relatively active research topic, the reconstruction offers a variety of good practices. However, as noted in ([Kaiser et al., 2019]; and [Kipf et al., 2019], placing representation learning loss in the pixel space has various failure modes. Firstly, small objects of great importance for the policy (for example ball in *Pong*) induce relatively low reconstruction loss if encoded badly. Similarly, bad encoding of visually rich background that is minimally important for the policy inflates reconstruction loss greatly.

Bisimulation

Bisimulation loss is directly related to MDP bisimulation as discussed in (REF) and was introduced in [Gelada et al., 2019]. Bisimulation is a mixed loss function defined over two entities:

$$\mathcal{L} = \alpha_T \mathcal{L}_T + \alpha_R \mathcal{L}_R \tag{2.55}$$

Where indices $_T$ and $_R$ denote portions of loss created by transitions and rewards respectively. The transition loss is defined as Wasserstein-1 distance between latent representation of real transitions from S and modeled transitions from latent representation of S according to:

$$\mathcal{L}_T = D(T(Z(S), A) - Z(S')) \tag{2.56}$$

The loss greatly simplifies for environments with deterministic transitions. Then, the loss is calculated as distance between latent representation of S' and predicted transition from latent representation of S. The reward loss is calculated as distance between the real reward and predicted reward via latent embedding of S:

$$\mathcal{L}_R = |R(Z(S), A) - R| \tag{2.57}$$

Where R denotes real reward gathered from the environment. With bisimulation loss, all components of the WM are trained simultaneously. With supervision signal being dependent on model outputs as in Q-network, the WM trains towards 'moving' target. Additionally, as noted by [Kipf et al., 2019], if there is not enough variation in the reward signal, the loss has a trivial solution with all states mapped to the same point. Kipf proposed contrastive loss, where additionally to bisimulation, model trains towards distance between embedding of different states. This calculation is done according to energy hinge loss [LeCun et al., 2006].

Information theory

The final discussed approach defines state abstraction as information theoretic process [Shannon, 1948]. While there is no established approach toward learning WM with information theory tools, a relatively often used approach utilizes variational information bottleneck (VIB) between three random variables X, Y and Z [Tishby et al., 2000]:

$$VIB = I(Z, Y) - \beta I(X, Y) \tag{2.58}$$

Where $I(Z, Y)$ and $I(X, Y)$ denotes the mutual information between (Z, Y) and (X, Y) respectively, measuring how different the joint distribution is from the product of marginals. β denotes the importance of compression relative to prediction and is an artifact of stating the problem as Lagrangian maximization of $I(Z, Y)$ under constraint of $I(X, Y) = x$. For discrete random variables, the calculation is done according to:

$$I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p_{(X,Y)}(x, y) \log \frac{p_{(X,Y)}(x, y)}{p_X(x) p_Y(y)} \tag{2.59}$$

As such, the calculation of VIB requires joint distributions of (Z, Y) and (X, Y), as well as marginals of Z, Y and X. Most often, VIB assumes learning an encoding of X, denoted as Y, such that the predictive power of Y on Z is maximal, while minimizing similarity between original data X and the encoding Y. As such, VIB can be interpreted as a conditioned compression scheme. In the context of world models, there does not seem to be a standard VIB form used in experiments. Biza et al. considers learning state representation, such that maximizes mutual information between the encoding and state rewards or values for deterministic, discrete-action MDPs [Biza et al., 2020]. Authors show that, given a Gaussian mixture model encoder with enough parameters, the resulting embedding can correspond to bisimulation of the original MDP. Alternative approach was used in [Hafner et al., 2019], with VIB target of form:

$$VIB = I(Z, (S, R)) - \beta I(Z, i) \tag{2.60}$$

Where Z denotes WM state embedding, S denotes state observation; R denotes rewards and i denotes time-stamp index of the state observation within trajectory. The time index is used because underlying environment is assumed to be partially-observable, and thus state representation is build with a sequential encoder. Then, minimizing mutual information between WM state embedding and time-stamp is supposed to regularize the recurrent component of the network. The VIB approach performed worse than reconstruction in most of the experiments performed in [Hafner et al., 2019].

Chapter 3

World model policy gradient

In this chapter, the algorithm proposed in this thesis is introduced. The chapter is divided into six sections: **Problem statement**; **Proposed algorithm**; **Algorithm training**; and **Related work**.

In the first section, the problem setting is described. Similarities and differences to traditional policy gradient methods are reviewed and discussed. Further, the two core mechanisms of proposed approach are linked to two main sources of stochasticity in PG: Q-values and expectation over actions at given state.

In the second section, proposed algorithm is described. Firstly, components of the algorithm are introduced: world and behaviour models. Further, principles through which agent uses the world model to approximate Q-values and the policy gradient are explained. Finally, the without-replaceme sampling approach that is used for approximation of expected value of state policy gradient is discussed.

In the third section, the training procedure for the proposed approach is described in detail. The pseudo-code for training world model, value and policy networks is presented.

In the final section, the related research is described.

3.1 Problem setting

Similarly to traditional PG setting, agent tries to maximize the gathered returns during learning, training with experience of (s, a, r, s') stemming from interaction with a deterministic MDP with continuous state representation and discrete action space. From policy gradient theorem, it follows that such optimization target can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \pi_\theta, d} \left[\nabla_\theta J(\theta, s) \right] = \mathbb{E}_{s \sim \pi_\theta, d} \left[\mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta J(\theta, s, a) \right] \right] \tag{3.1}$$

With:

$$\nabla_\theta J(\theta, s, a) = Q^{\pi_\theta}(s, a) \nabla_\theta \log \pi_\theta(a|s) \tag{3.2}$$

Regular PG assumes that the expectation wrt. action in some state $\mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta J(\theta, s, a) \right]$ is calculated with MC as well. There are multiple reasons for such procedure. Firstly, Q-values are often unknown and have to estimated. If they are estimated as sampled returns, then approximating Q-value for some state-action pair requires the agent to execute the policy starting from that state-action pair. In regular setting, agent can execute only one action in state s before being transitioned to some state s'. Executing other action would require the agent to get back to state s, which is unpractical for many RL environments. Similarly, assuming the agent knows Q-value of each action in state s, it might be that set of available actions in the underlying MDP is too big for exact calculations. For above reasons, the expectation is often approximated with a single-sample estimation. However, if one assumes drawing some amount of samples without replacement from a known non-deterministic policy, then approximating mean with MC becomes inefficient ([Metropolis and Ulam, 1949]; and [Kool et al., 2019a]).

The method proposed in this chapter considers policy gradient approximation using multiple trajectories generated by the world model. As such, the policy is updated according to signal generated by the reward, transition and value approximators. This thesis hypothesizes that such approach could increase the sample efficiency of policy search through two mechanisms.

Low variance approximation of the state policy gradient

Transition and reward functions allow the agent to simulate any amount of trajectories starting from s. Given some amount of trajectories sampled without replacement, agent can calculate the state PG with Horvitz-Thompson estimator [Horvitz and Thompson, 1952]. As shown in [Williams, 1992] and Appendix A, exactly calculated gradient yield policy updates that change probabilities of individual actions proportionally to their Q-values. While sign of the gradient flowing through each action depends solely on the sign of it's Q-value, softmax derivative allows to redistribute the probability density according to the gradient's magnitude. In Appendix A, it is shown that the above property is partially lost due to MCA, due to the fact that magnitudes of gradients of the non-sampled actions are not known. As such, all non-sampled actions are corrected uniformly through softmax, dependent solely on the Q-value of the performed action. This property slows down the policy convergence. For example, if the gradient is approximated from an action with Q-value greater than the baseline, the probability of all other actions will be decreased equally - even those which are better than the executed action. Symmetrically, learning from a trajectory worse than the baseline will increase probability of all other actions, even those with smaller Q-values. Obviously, this is not a problem if the agent samples the optimal action. However, since the policy often starts stochastic, the optimal action might not be sampled often in the early stages of training. This thesis investigates whether approximating the state PG with multiple simulated trajectories sampled without replacement increases the sample efficiency of gradient-based policy search, by increasing probability that the optimal trajectory is sampled for any stage of the training.

Low variance approximation of Q-value

A standard approach is to estimate the Q-values with the gathered returns [Williams, 1992]. Furthermore, traditional PG methods use single sample Q-value estimates. Such procedure adds to uncertainty of PG calculations, as MC value estimates have high variance. This is especially problematic for early stages of training when the policy has high entropy, as single-sample Q-value might be not-representative of the policy. To this end, the world model allows the agent to unroll any policy from any state, arbitrary amount of times. As such, given some state-action pair and policy π, agent can approximate Q-value using multiple policy rollouts. Using multiple rollouts of a policy in Q-value approximation reduces variance of the estimator and allows the agent to diagnose promising trajectories early into the training. Similarly, given batch of states, agent can perform multiple policy updates, recalculating the Q-values for each policy on the way. This mechanism mimics policy iteration [Sutton and Barto, 2018], albeit calculated only for the batched states.

3.2 Proposed algorithm

The proposed approach differentiates from traditional PG methods from three perspectives: *algorithm components*; *Q-value approximation method*; and *policy gradient approximation method*. In the following subsections, we introduce the proposed algorithm in detail.

3.2.1 Algorithm components

The agent consist of three main components: *world model*; *behaviour model*; and *memory.*

World model

The world model learns the behaviour-agnostic information from agent-environment interaction. As such, the WM consist of three modules: *state embedding*, *transition* and *reward mappings*.

State embedding Z compresses the original state representation into smaller dimensionality. State dimensionality reduction allows the agent to learn more robust reward and transition approximators, as they operate on lower dimensionality ([Oh et al., 2015]; and [Hafner et al., 2019]). The FNN modelling state embedding can be trained independently of reward and transition functions with reconstruction or jointly with bisimulation loss. The state embedding network is parametrized with ν and denoted as Z_ν.

Transition function T_κ approximates the transitions of the underlying MDP and allows the agent to simulate transition after performing some action. Given that the underlying MDP is deterministic, transition can be modelled with a FNN, such that $T : Z \times A \rightarrow Z'$. Transition function parameters κ are trained with mean absolute distance loss against the real transitions from the environment.

Reward function R_o maps state-action pairs to rewards, such that $R : Z \times A \rightarrow \mathbb{R}$. For non-deterministic MDPs, the FNN representing reward mapping would have to be additionally conditioned on the following state representation Z'. Similarly to transition function, R_o is trained with mean absolute distance against the real rewards from the environment.

Behaviour model

Behaviour model learns optimal policy, as well as state values under that policy. It is composed from two modules: *policy* and *value functions*.

Policy π_θ outputs a discrete probability distribution over all possible actions conditioning on the input state z. Similarly to traditional policy gradient, it is trained to maximize the expected state values.

Value V_ϕ maps state representation z to some real number. Similarly to any other AC approach, value network can be trained using TD or MCPE methods.

Memory

Memory stores the agent interactions with the environment and is built by two modules: *rollout memory* and *policy-agnostic memory*.

Rollout memory stores the most recent trajectories, and thus the data represents samples from the current policy. This on-policy data is first used to update the value and policy networks, then is transferred to the policy-agnostic memory.

Policy-agnostic memory stores some amount of recent transitions, not necessarily coming from the most recent policy. This data is used to train the world model components of the agent.

3.2.2 Q-value approximation

The second enhancement to traditional PG methods is the method of approximating the Q-values. The Q-value can be defined in terms of future state values. Denote $R(s, a)$ and $T(s, a)$ as MDP reward and transition mappings respectively, with $s' = T(s, a)$. Then, Q-value of some state-action pair is given by:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \mathbb{E}_{a' \sim \pi} \left[Q^{\pi}(T(s, a), a') \right] = R(s, a) + \gamma V^{\pi}(T(s, a)) \tag{3.3}$$

Following above definition, on-policy Q-values can be approximated either by a single Q-network, or by reward, transition and value networks combined. Furthermore, given transition, reward mappings and some state-action pair (s, a), any policy can be unrolled some n steps into the future:

$$G(\tau | s, a, n) = R(s, a) + \gamma R(T(s, a), a') + ... + \gamma^n V^{\pi}(T^n(s, a)) \tag{3.4}$$

Where τ denotes a sampled sequence of n transitions given the policy π, $T^n(s, a)$ represents final state after executing τ and a' stems from the analyzed policy. In such setting, each $G(\tau|s, a)$ represents a sampled return from some policy, with expected value of $G(\tau|s, a)$ being equal to $Q^{\pi}(s, a)$. Given different values of n, Q-value can be approximated with TD(λ), allowing to directly trade-off bias and variance of estimates.

Single-step — all-bias no-variance

First, we consider calculating $Q^{\pi}(s, a)$ according to equation 3.3, such that the true MDP mappings are approximated with model components described in Subsection 3.2.1:

$$\hat{Q^{\pi_\theta}}(s, a) = R_o(Z_\nu(s), a) + \gamma V_\phi(T_\kappa(Z_\nu(s), a)) \tag{3.5}$$

Thus, Q-value of state-action pair (s, a) is equal to the reward from performing a in s added to the discounted value of the future state s'. Such calculation has zero variance, as for every possible combination of $(\theta, \phi, \nu, \kappa, o)$ always returns a single estimate of Q-value. Since all model parameters start initialized with arbitrary values and converge during learning, the calculation has exponentially decaying bias, which might not reach zero if the model is stuck on a local optimum [Sutton and Barto, 2018]. Even if the reward and transition networks approximate the MDP with no error, it might be that the value network 'remembers' values from the previous policies.

Many steps — bias-variance mix

Alternatively, the bias of above calculations can be reduced by approximating n step returns according to equation 3.4. With horizon lengths up till number N, the Q-value can be calculated with TD(λ), according to:

$$Q_{\lambda}^{\pi\theta}(s,a) = (1-\lambda)\sum_{n=1}^{N-1}\lambda^{n-1}G(\tau|s,a,n) + \lambda^{N-1}G(\tau|s,a,N) \tag{3.6}$$

Where:

$$G(\tau|s,a,n) = R_o(Z_\nu(s),a) + \gamma R_o(T(Z_\nu(s),a),a') + ... + \gamma^n V^{\pi_\theta}(T^n(Z_\nu(s),a)) \tag{3.7}$$

In this setting, we approximate Q-values with exponentially-weighted average of the n-step simulated returns for different values of n. Similarly to single-step simulation (3.2.2) the approach has bias, since the agent's networks are used in calculations as well. Contrary to the single-step approach, the returns are calculated by sampling from the policy. Therefore, each Q-value estimation is sampled from the implicit Q-value distribution. The variation allows the model to evaluate state-action pairs multiple times where each evaluation yields different policy gradient. The above approach is a strict generalization of no-bias approach, as when setting $N = 1$ or $\lambda = 0$ the calculation of many-steps becomes equivalent to the single step.

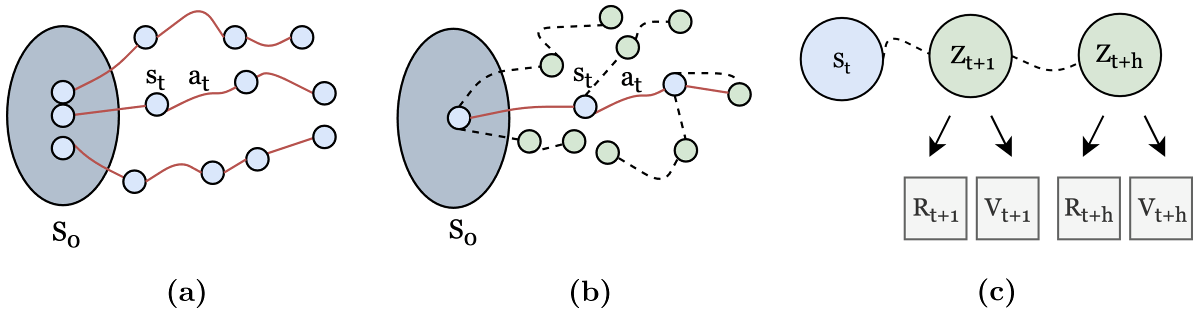


Figure I: WMPG Q-value approximation. a) Trajectories are sampled from the environment (represented by red lines). The data is used to train transition, reward and value networks. b) Agent samples without replacement some amount of actions for each state and imagines a trajectory of fixed length (represented by the dashed line). The sampled actions might be different than the actions performed in the environment. c) The Q-value of the actions sampled without replacement is evaluated using TD(λ).

3.2.3 Policy gradient approximation

The policy gradient at state s is equal to:

$$\nabla_\theta J(\theta, s) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta J(\theta, s, a)\right] \tag{3.8}$$

We propose to approximate the above expectation with simulated trajectories using the components described in Subsection 3.2.1. This allows the agent to sample actions without replacement [Kool et al., 2019a]: agent samples k actions without replacement at every considered state. For any $k \leq |A|$, the expectation can be calculated efficiently with without-replacement Horvitz-Thompson estimator ([Horvitz and Thompson, 1952]; [Hesterberg, 1988]; [Duffield et al., 2007]; and [Kool et al., 2019a]). Then, the expectation is approximated with:

$$\nabla_\theta \hat{J}(\theta, s) = \sum_{i=1}^{k} \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} \nabla_\theta J(\theta, s, a_i) \tag{3.9}$$

Where i indicates the concurrent actions sampled without replacement simulated with the WM and $\Omega(a_i|\pi_\theta, k, s)$ represents probability that action a_i was sampled without replacement given k samples and policy π_θ in state s. For any fixed k the estimator is an unbiased approximator of the exact expectation. The approximator rescales the probability density of each sample, such that the non-sampled domain is approximated with samples drawn without replacement. The unbiasedness proof presented in Appendix B is equivalent to result shown in [Duffield et al., 2007] (albeit for the case where weights represent probabilities), as well as the result presented in [Kool et al., 2019a] (without introducing Gumbel distribution). Conveniently, for $k = 1$, the approximator becomes equal to MC estimator:

$$\frac{\pi_\theta(a|s)}{\Omega(a|\pi_\theta, 1, s)} = 1 \tag{3.10}$$

Similarly, for $k = |A|$, calculation becomes exact expectation calculation:

$$\frac{\pi_\theta(a|s)}{\Omega(a|\pi_\theta, |A|, s)} = \pi_\theta(a|s) \tag{3.11}$$

The approximator allows to leverage the fact that policy probabilities are known. As such, it has lower variance than MCA as long as the policy is not fully deterministic.

Choosing k

Whereas MC PG approximator has strictly diminishing variance reduction stemming from increase of samples, this is not necessarily the case for the without-replacement sampling estimate. As shown in Appendix C, MC approximation of PG at given state is a sum of independent random variables divided by a constant. When approximating the state

PG with without-replacement sampling, the approximation becomes a weighted sum of random variables.

The estimator from Equation 3.9 is an unbiased estimator of expectation for any fixed size of k. Here, we postulate that it does not imply that k should have fixed size over the course of training. This is the case because we treat final PG approximator as sample average of independently approximated policy gradients for batched states, given some policy. Thus, PG at every state is approximated with an unbiased estimator of PG in that state, with k fixed for individual state, but different between states. In principal, the best approximation of gradient at given state is calculated when all actions are sampled. However, the benefits from increasing k disappear completely for deterministic policies, where sampling one action is enough for complete evaluation. Assuming a stochastic policy it seems natural to ask: is there a way of choosing k such that the quality of the approximation balances the required compute? The analysis of variance of without-replacement sampling approximator as a function of number of samples is a highly non-trivial task and beyond the scope of this thesis. Below, we develop heuristics that stem from different assumptions about the behaviour of PG variance dependent on k, in the context of world model trajectory simulation.

Constant k

In the simplest case, k may be treated as a hyperparameter, with the same k actions being sampled over the course of training. This setup is considered by [Kool et al., 2019a].

Decaying k

If the policy is deterministic, sampling single action is equivalent to calculating exact expectation. As the policy starts stochastic and decreases entropy over the training, it might be that decaying k over the training would decrease the required compute, while having only minor effects on the training.

Growing k

Since the transitions are sampled from the world model, quality of the PG approximation depends on the bias of simulated Q-values. As such, it might be that k should be increased, as the quality of WM prediction increases.

Policy-dependant k

It seems natural that k should be related to entropy of the policy distribution. If for certain state policy is deterministic, then sampling $k > 1$ yields no effect. On the other hand, if the policy is random, it might be beneficial to simulate as many actions as possible. This intuition yields a simple heuristic:

$$k(s) = \sum_{a \in A} f(\pi_\theta(a|s)) \tag{3.12}$$

With:

$$f(\pi_\theta(a|s)) = \begin{cases} 1 \quad if \quad \pi_\theta(a|s) \geq \frac{1}{A} - \epsilon \\ 0 \quad if \quad \pi_\theta(a|s) < \frac{1}{A} - \epsilon \end{cases} \tag{3.13}$$

Where ϵ is the slack parameter. The heuristic has a simple interpretation: algorithm samples one action for every action with policy probability bigger than $\frac{1}{A} - \epsilon$. For example, with $\epsilon = 0$ and 5 available actions, agent would sample 5 actions only for perfectly random policy. If only one action was more likely that $\frac{1}{A} = 20\%$, then agent samples one action from the policy.

Baseline variance reduction

WMPG uses the baseline variance reduction scheme developed in [Kool et al., 2019a], where variance is reduced by introducing a baseline dependent on analyzed trajectories. Then, returns of each state-action are reduced with without-replacement sample value estimate build from the entire sample of state-actions. Since the without-replacement weights depend on all actions, introducing such baseline requires bias correction with action dependent term:

$$\nabla_\theta \hat{J}(\theta, s, b(s)) = \\ \sum_{i=1}^{k} \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} \left((1 + \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} - \pi_\theta(a_i|s)) Q^{\pi_\theta}(s, a_i) - b(s) \right) \nabla_\theta \log \pi_\theta(a_i|s) \tag{3.14}$$

With:

$$b(s) = \sum_{i=1}^{k} \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} Q^{\pi_\theta}(s, a_i) \tag{3.15}$$

The estimator greatly simplifies if k is equal to the number of actions in the MDP:

$$\nabla_\theta \hat{J}(\theta, s, b(s)) = \sum_{a \in A} \nabla_\theta \pi_\theta(a|s) \left(Q^{\pi_\theta}(s, a) - \sum_{a \in A} \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \right) \tag{3.16}$$

This form of baseline yields reduced variance, as shown in Equation 2.52, it also is does not create bias [Kool et al., 2019a].

Without-replacement weight normalization

The weighted probabilities do not have to sum to one. As such, the importance weights can scale the gradients between different states. To alleviate this, we use the normalization scheme detailed in [Kool et al., 2019b]:

$$\nabla_\theta \hat{J}(\theta, s, b(s)) = \sum_{i=1}^{k} \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} \left(\frac{Q^{\pi_\theta}(s, a_i)}{W_i(s)} - \frac{b(s)}{W(s)} \right) \nabla_\theta \log \pi_\theta(a_i|s) \tag{3.17}$$

With:

$$W(s) = \sum_{i=1}^{k} \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} \tag{3.18}$$

And:

$$W_i(s) = W(s) - \frac{\pi_\theta(a_i|s)}{\Omega(a_i|\pi_\theta, k, s)} + \pi_\theta(a_i|s) \tag{3.19}$$

The normalized estimator is no longer unbiased, but remains consistent. It is also reported to yield more stable training due to lower variance ([Kool et al., 2019b] and [Kool et al., 2019a]). After calculating $\nabla_\theta \hat{J}(\theta, s, b(s))$ for each state in the learning batch D_s of size N_s, final PG is approximated with MC:

$$\nabla_\theta \hat{J}(\theta, b(s))) = \frac{1}{N_s} \sum_{D_s} \nabla_\theta \hat{J}(\theta, s, b(s)) \tag{3.20}$$

The non-normalized non-baselined version of WMPG is defined through policy gradient theorem and as such is unbiased per definition. However, similarly to AC, WMPG uses approximations of Q-values. As such, the WMPG approximation of PG is unbiased, as long as the approximation of Q-value is unbiased. Since the networks calculating Q-value are initialized with random parameters, the early approximations of Q-values are biased. Similarly to regular TD calculations, the bias is exponentially disappearing during training, with limit at 0 when the networks are converged [Sutton and Barto, 2018].

3.3 Algorithm training

This section details the algorithm and its separate components. We assume that the agent gathers an on-policy experience tuple (s, a, r, s') at every time step as is usual in PG training. After there are B tuples available, with B equal to the desired batch size, learning iteration is triggered:

Algorithm 1 Learning iteration

Require: memory, world model WM, behaviour model BM, world model iterations I, batch size B

```
1: function LEARNING_ITERATION()
2:     while $i < I$ do
3:         $b_w$ = policy_agnostic_memory($B$)
4:         train_WM($b_{wm}$)
5:     end while
6:     $b_b$ = rollout_memory($B$)
7:     train_value($b_b$)
8:     train_policy($b_b$)
9: end function
```

Where b_w and b_b denote batches of data used to train the world and behaviour models respectively. The world model training loop can be activated multiple times because data is sampled from the entire experience buffer and is independent of the policy. The training loops of individual components are investigated in the further subsections.

3.3.1 World model

This thesis treats WM learning as an orthogonal problem. Two methods of WM learning are considered: *no-state* and *reconstruction* learning.

No-state learning assumes that the original state representation does not have to be compressed. As such, the transition and reward functions directly input MDP state representation and learn through mean absolute distance towards real transitions and rewards.

Reconstruction learning allows to learn the state embedding function independently from T and R. In this approach, variational auto-encoder (VAE) is pre-trained with some number of transitions generated with random policy ([Ha and Schmidhuber, 2018]; [Kaiser et al., 2019]; and [Kipf et al., 2019]).

Algorithm 2 WM learning

Require: state embedding function Z_ν, transition function T_κ, reward function R_o, learning-rate α, samples from experience: states S, transitions S' and rewards R

```
1: function TRAIN_WM()
2:     $\kappa \leftarrow \kappa - \alpha \nabla_\kappa |T_\kappa(Z_\nu(S), A) - S'|$
3:     $o \leftarrow o - \alpha \nabla_o |R_o(Z_\nu(S), A) - R|$
4: end function
```

3.3.2 Value

Value network can be trained with any value estimation technique (e.g. TD, MC). In this thesis, value network is trained with mean absolute distance towards MC on-policy rollout values sampled from the agents memory.

Algorithm 3 Value learning

Require: rollout memory, state embedding function Z_ν, value function V_ϕ, learning-rate α, samples of on-policy visited states S and values $V^{\pi_\theta}_{MC}(S)$

```
1: function TRAIN_VALUE()
2:     $\phi \leftarrow \phi - \alpha \nabla_\phi |V_\phi(S) - V^{\pi_\theta}_{MC}(S)|$
3: end function
```

3.3.3 Policy

Policy network is trained with a PG target wrt. to simulated trajectories generated by the WM. Below pseudo-code depicts policy updates.

Algorithm 4 Policy learning

Require: rollout memory, state embedding function Z_ν, value function V_ϕ, learning-rate α, samples of on-policy visited states S

```
1: function TRAIN_POLICY()
2:     $A$=SWOR$(\pi_\theta, S, k)$
3:     $Q^{\pi_\theta}(S'_{S,A})$=simualate_values$(S, A, h, \lambda)$
4:     $\beta_{S,A}$=calculate_weights$(\pi_\theta, S, A, \lambda)$
5:     $\nabla_\theta J(\theta, S)=\sum_{a \in A} \beta_{S,a} \nabla_\theta J(\theta, S, a)$
6:     $\nabla_\theta J(\theta)=\frac{1}{B}\sum_{s \in S} \nabla_\theta J(\theta, s)$
7:     $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
8: end function
```

Where SWOR(π_θ, S, k) denotes sampling without replacement k actions in states S given policy π_θ and calculate_weights refers to calculations detailed in equation 3.9. The TD(λ) simulation is implemented with the pseudo-code below:

Algorithm 5 Simulated Q-values

Require: transition function T_κ, reward function R_o, horizon of imagination h, temporal difference λ, batch of encoded states Z and sampled actions A

```
1: function SIMULATE_VALUES()
2:     $i = 0$
3:     while i < h do
4:         $Z_i = T_\kappa(Z, A)$
5:         $\hat{R} = \hat{R} + \gamma^i R_o(Z, A)$
6:         $V_i = \hat{R} + \gamma^{i+1} V_\phi(Z_i)$
7:         $A_i = \pi_\theta(Z_i)$
8:         if i+1 $\neq$ h then
9:             $V_\lambda = V_\lambda + \lambda^i * V_i$
10:        end if
11:        if i+1 = h then
12:            $V_\lambda = V_\lambda + (1 - \lambda)\lambda^i * V_i$
13:        end if
14:        $(Z, A) = (Z_i, A_i)$
15:        $i = i + 1$
16:    end while
17: end function
```

The above approach allows for efficient calculation of TD(λ), with complexity which is linear wrt. the horizon length.

3.4 Related work

The proposed approach unifies two lines of research in DRL: *robust policy gradient approximation*; and *learning with a world model.*

3.4.1 Robust policy gradient approximation

Mean Actor Critic calculates exact state policy gradient by approximating Q-values of all actions with a Q-network [Asadi et al., 2017]. Kool et al. use multiple executed trajectories to approximate state PG with normalized without-replacement estimator.

Asynchronous actor critic [Mnih et al., 2016] calculates MC over samples generated in parallel by many agents using the same policy.

3.4.2 Learning with a world model

Simulating multi-step returns has been considered by a variety of prior research. Value Prediction Network (VPN) [Oh et al., 2017] and Model-based Value Extension (MVE) [Feinberg et al., 2018] use multi-step returns to approximate Q-values and achieve better sample efficiency than Deep Q-learning. PLANET [Hafner et al., 2018] and Simulated Policy Learning (SimPLe) [Kaiser et al., 2019] learn the model dynamics and plan within the simulated environment, achieving state of the art sample efficiency on multiple discrete task environments. AlphaGo [Silver et al., 2017] combines value prediction with planning, assuming knowledge of the underlying MDP. MuZero [Schrittwieser et al., 2019] uses simple heuristics to perform MCTS in the latent space. Dreamer [Hafner et al., 2019] uses deterministic policy gradients simulated by a world model and achieves state of the art on multiple continuous action environments.

Chapter 4

Experimental setup

In this chapter, the experimental setup is introduced. The chapter is divided into three sections: **Environments**; **Benchmark algorithms**; and **Hyperparameter search**.

In the first section, the environments on which the proposed algorithm is compared against benchmark approaches are described. Firstly, two classic control problem with compact state representations are introduced. Furthermore, Atari Pong with high-dimensional state representation is discussed.

In the second section, the implementation of benchmark algorithms is detailed.

In the third section, the hyperparameter search scheme used for each environment is detailed.

4.1 Environments

The proposed approach is tested against benchmark algorithms on two-types of tasks: *classic control*; and *frame-based control.*

4.1.1 Classic control

Classic control environments have compact state representation. This property allows to evaluate the proposed algorithm partially orthogonally to the problem of representation learning, as the World Model consist of only reward and transition mappings. As such, state embedding is assumed to be an identity mapping.

CartPole

CartPole is a classic control problem first introduced in [Barto et al., 1983]. In it, a rigid pole is attached to a cart operating on a bounded 1-dimensional plane. The pole is moving vertically, depending on the force that the agent applies to the cart. Goal of the agent is to balance the pole for as many steps as possible.

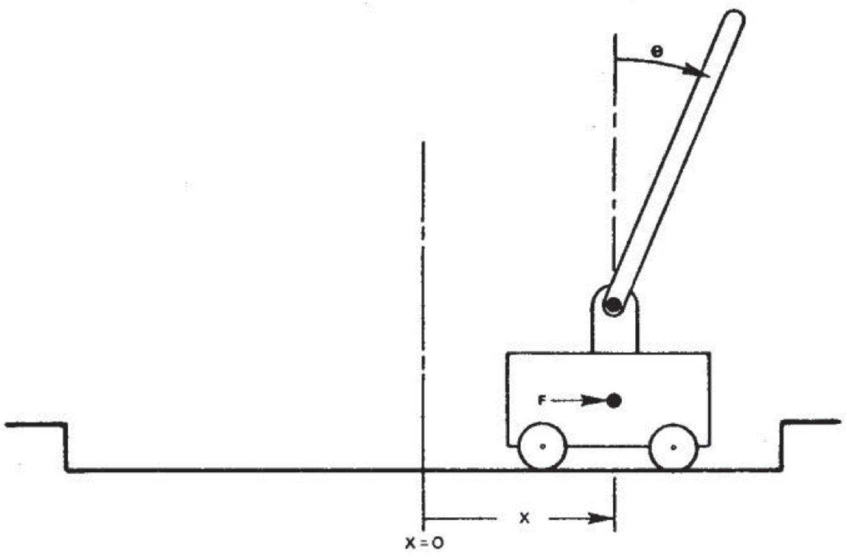


Figure I: CartPole schematic used in [Barto et al., 1983]. Cart starts the simulation around position $x = 0$ of bounded 1-dimensional plane. Actor can apply fixed amount of force F towards one of the two directions of x, with the goal of balancing the pole

CartPole model is represented by four continuous state variables (x, $\hat{x}$, θ, $\hat{\theta}$) where: x is the position of the cart on the plane; $\hat{x}$ is cart velocity; θ is the angle between the pole and a vertical line; and $\hat{\theta}$ showing the rate of change of the angle θ. Agent can apply fixed amount of force to the left or to the right, creating a discrete action space of size 2. Agent receives constant reward equal to 1, until termination. Termination occurs if the pole falls, if the cart touches the boundary or if the episode lasts 200 steps.

LunarLander

LunarLander simulates the problem of rocket trajectory optimization. In it, a lander embedded in 2d space manipulates three engines to land on a pad, without damaging the rocket. Agent gathers negative reward for using main engine (-0.3 point), side engines (-0.03 point) or for crashing the lander (-100 points). Positive rewards are mapped to successful landing (100 points) and for each leg contact with the ground (10 points). The landing pad is always located on the same coordinates.

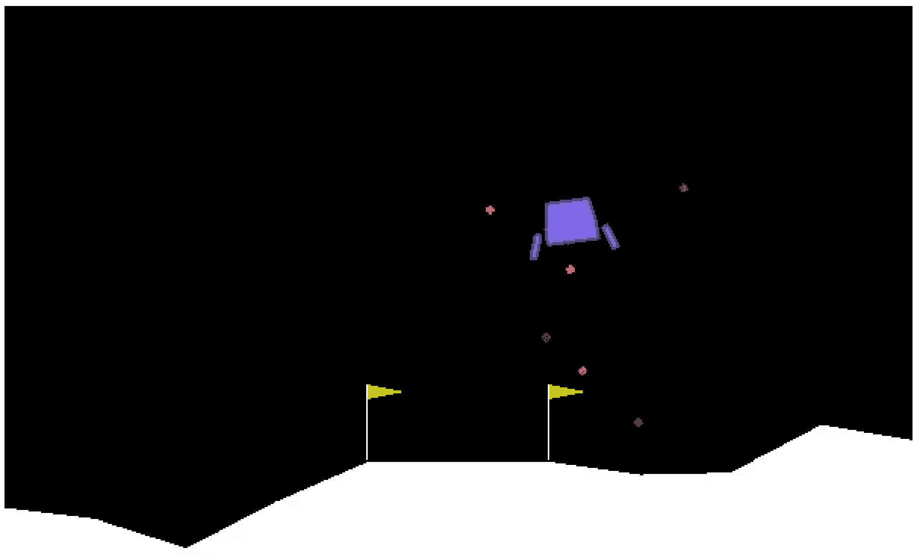


Figure II: Example frame from the LunarLander environment

Originally, LunarLander was a continuous action problem, but Pontryagin's maximum principle allows to prune the action space from from all actions except full-engine throttle or engine off. As such, the action space consist of four actions: fire any of the three engines or do nothing. The state-space is represented with a continuous vector of 8 values representing the position, velocity and direction of the lander.

4.1.2 Atari

Atari is a well-known sandbox environment for testing DRL algorithms. With dozens of different games that exhibit properties like partial-observability or sparse reward mappings, the agent is expected to find optimal policy from either the image or RAM data. We choose to test the proposed approach on visual representation, implemented in OpenAI gym under name *PongDeterministic-v4*. Atari experiments are run with WMPG and AC.

Pong

Pong is table tennis simulator embedded on a two-dimensional plane. Agent controls one in-game paddle by moving it vertically across the screen. The goal of the game is to reach score of 21 against computer controlled paddle. As such, to successfully control Pong, agent has to learn the dynamics of the ball, as well as the computer controlled paddle. The state observation is an image of the screen with size of (210, 160, 3).

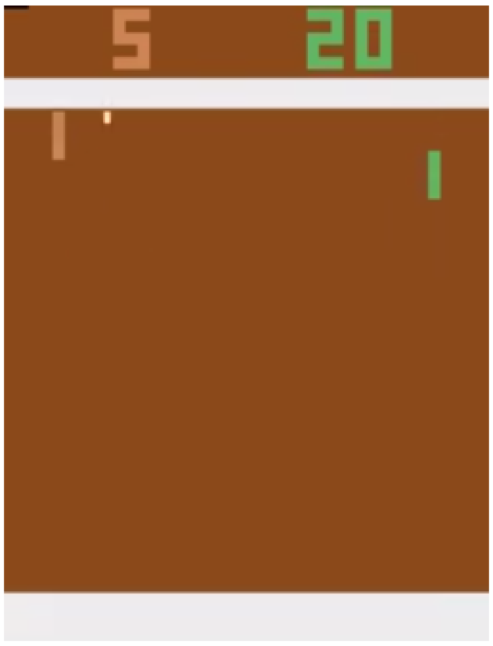


Figure III: The observation is an RGB image of the screen, which is of size (210, 160, 3)

To reduce the compute load of all algorithms, the static portions of the screen are deleted, entire frame is downsized by the factor of two and colours are turned black and white. An example preprocessed frame is shown on Figure IV.

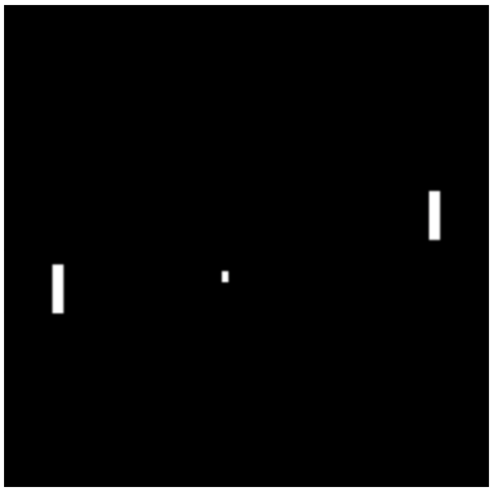


Figure IV: Preprocessed observation is a black and white image of size (80, 80)

Finally, to address the known mode of failure of compressing Pong frames, we enlarge the ball by the factor of 2 [Kaiser et al., 2019]. Observing a single frame from Pong makes the environment partially-observable, as the direction and velocity of the ball is not known. To relief this, four consecutive frames are concatenated. Furthermore, the proposed approach uses a pretrained state embedding function. As such, it inherently searches for the policy in state representation of smaller dimensionality. To this end, we decide to train the benchmark algorithms on the compressed world model representation of the environment as well, with the same state embedding function. Such procedure allows for better comparison and isolation of the effects of representation learning and the proposed approach. The state embedding function is trained with 20 000 frames collected according to random policy, disregarding frames where ball is not visible. The model is trained for 50 epochs or until the validation loss increases for two epochs in a row.

4.2 Benchmark algorithms

All models are implemented with two interacting classes: *rollout buffer*; and *model*. Rollout buffer manages the experience gathered by interacting with the environment and model stores and trains the neural networks used by the agent. All experiment components are implemented using default Python 3.6 stack complemented with NumPy [Walt et al., 2011] and PyTorch [Paszke et al., 2017] packages.

4.2.1 Actor Critic

In actor-critic (AC) approaches, agent is composed of two structures: policy π_θ (actor); and value v_ϕ (critic). In this setup, critic network is used to create an estimate of value of a given state s, such that it can be used as a baseline ([Schulman et al., 2015b], [Mnih et al., 2016]; and [Asadi et al., 2017]). The critic is trained with some form of a distance loss, such that it learns to estimate $V^{\pi_\theta}(s)$, which itself is approximated by one of the value approximation techniques (MCPE; TD; TD(λ)). Each of those techniques implies certain mix of bias and variance for the critic. In this thesis, values are approximated with MC. AC agent is implemented using two independent FNNs with independent optimizers. The pseudo-code for AC training function is shown below:

Algorithm 6 Train networks AC

Require: value function V_ϕ, policy function π_θ, samples of on-policy visited states S, performed actions A and values $V_{MC}^{\pi_\theta}(S)$

```
1: function TRAIN_NETWORKS()
2:     $Adv(S) = V_{MC}^{\pi_\theta}(S) - V_\phi(S)$
3:     $\theta \leftarrow \theta + \alpha_\theta Adv(S) \nabla_\theta \log \pi_\theta(A|S)$
4:     $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi |V_\phi(S) - V_{MC}^{\pi_\theta}(S)|$
5: end function
```

The agent is initialized by defining network architectures, optimizers, discount factor and learning rates.

4.2.2 Mean Actor Critic

Similarly to original implementation, MAC Q-value estimations are anchored towards MC Q-value approximation. MAC is implemented using two independent FNNs and optimizers. Q-network is defined to be conditioned on the action, such that it outputs a scalar for state-action pair. The pseudo-code for MAC training function is shown below:

Algorithm 7 Train networks MAC

Require: Q-value function Q_ϕ, policy function π_θ, samples of on-policy visited states S and Q-values $Q_{MC}^{\pi_\theta}(S, A)$

```
1: function TRAIN_NETWORKS()
2:     $\theta \leftarrow \theta + \alpha_\theta Q_\phi(S, A) \nabla_\theta \pi_\theta(A|S)$
3:     $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi |Q_\phi(S, A) - Q_{MC}^{\pi_\theta}(S, A)|$
4: end function
```

The agent is initialized by defining network architectures, optimizers, discount factor and learning rates. Moreover, following the original implementation, agent is allowed to perform multiple network updates in one learning episode [Asadi et al., 2017], similarly to generalized policy iteration [Sutton and Barto, 2018]. As such, MAC has two additional hyperparameters: general update iterations and value update iterations. General update iteration defines how many times entire learning loop is run on the same batch of states. Value update iteration determines how many times Q-network is updated in one learning loop.

4.3 Hyperparameter search

To minimize the effects of hyperparameter setting on conclusions, grid search [Bishop, 2006] is performed. For each hyperparameter setting, average cumulative training reward and its variance over many seeds is recorded. For each model, best performing setting is chosen. Each method has a different number of tunable hyperparameters, with the implemented version having 7, 8 and 18 for AC, MAC and WMPG respectively. The proposed algorithm having most dimensions to search over, complete search creates a probability of spurious results that stem from the number of configurations searched. As the proposed method has many common hyperparameters with AC, the search for those is done only for AC, while WMPG uses the optimal setting found in AC grid search. Additionally, all methods use the same size of policy network and all networks use ReLU activations. All tunable hyperparameters for the implemented methods are listed in the table below.

Name	Description	AC	MAC	WMPG
π_{size}	Policy network size	x	x	x
V_{size}	Value network size	x	(x)	x
T_{size}	Transition network size			(x)
R_{size}	Reward network size			(x)
O_{π}	Optimizer for π	(x)	(x)	x
O_V	Optimizer for value	(x)	(x)	x
O_T	Optimizer for transition			(x)
O_R	Optimizer for reward			(x)
α_{π}	Learning rate for π	(x)	(x)	x
α_V	Learning rate for value	(x)	(x)	x
α_T	Learning rate for transition			(x)
α_R	Learning rate for reward			(x)
I_G	General learning iterations		(x)	(x)
I_V	Value learning iterations		(x)	(x)
I_G	WM learning iterations			(x)
h	Horizon length			(x)
k	Sampled actions			x
λ	TD(λ) controller			(x)
# of configs	CartPole	81	486	126
	LunarLander	9	24	36
	Pong	8	NA	4

Table I: Number of tunable hyperparameters; x denotes that parameter is tunable; (x) denotes that parameter was tuned for experiments in this thesis

4.3.1 CartPole

Simplicity of the environment allows to test many hyperparameter settings for each method. Each model's policy network has fixed size of 1 hidden layer with 32 neurons. For AC and WMPG value network has fixed size of 1 layer and 64 neurons. All models use discount factor of 0.99 and a batch size of 32. Each configuration results are averaged over 10 seeds.

AC

AC grid search focuses on optimizer and learning rates. Three optimizers (Adam; RMSprop; and Adadelta) and three learning rates (0.005; 0.0025; and 0.00125) are tested per network, yielding grid with 81 elements. Table below shows hyperparameter values for the best performing configuration:

π_{size}	V_{size}	α_π	α_V	O_π	O_V	I_G
32	64	.0025	.005	RMS	RMS	1

Table II: Best performing AC configuration on CartPole

MAC

As the Q-network is additionally conditioned on action, it might require more parameters than a regular value network. As such, the search is done over network sizes of Q-network Q_{size} (1 layer with 64 units; 2 layers with 32 units each; and 2 layers with 64 units each), optimizers O_π and O_Q (Adam; RMSprop; and Adadelta), learning rates α_π and α_Q (0.005; 0.0025; and 0.00125); value learning iterations I_V (1; and 3), and general learning iterations I_G (1; 3; and 5). To reduce the grid size, same type of optimizer is used for both networks. Concluding, search is done over 486 models. Best performing configuration is detailed below:

π_{size}	Q_{size}	α_π	α_Q	O_π	O_Q	I_G	I_V
32	[64,64]	.00125	.005	RMS	RMS	3	3

Table III: Best performing MAC configuration on CartPole

WMPG

The learning rates, optimizers and networks sizes for both policy and value are taken from the AC grid search. To further reduce the search scope, the sizes of transition and reward networks are fixed at 1 layer with 64 units. Similarly, the same type of optimizer is used for both WM networks, with the same starting learning rate. Since the environment has only two actions, the state gradient is calculated exactly. Moreover, the horizon length and lambda should be independent of other hyperparameters. As such, the search is first done for single-step horizon and $\lambda = 1$. Then, given found learning rates, optimizers and iterations, additional values for horizon λ are tested. The pruning process results in search over 108+18 configurations, with optimizer for WM (Adam; RMSprop; and Adadelta), learning rate for WM (0.005; 0.0025; and 0.00125), value learning iterations (1; and 3), WM learning iterations (3; and 5) and general learning iterations (1; 3; and 5), horizon (5; 10; and 15) and λ (0.25; 0.5; and 0.75).

T_{size}	R_{size}	α_T	α_R	O_T	O_R	I_G	I_V	I_{WM}	h	λ
64	64	.005	.005	Adam	Adam	5	3	5	15	0.75

Table IV: Best performing configuration of WMPG. Other settings are taken from AC

4.3.2 LunarLander

Due to bigger state representation, policy network sizes of all models are increased to 64. All models use a discount factor of 0.99 and batch size of 64. Moreover, all models use a multiplicative learning rate scheduler with multiplier of 0.95 and step size of 125. Each configuration results are averaged over 5 seeds.

AC

Due to increased cost of the hyperparameter search on LunarLander, domains of search are decreased. Following OpenAI RL benchmark implementations [Brockman et al., 2016] both policy and value network use the RMSprop optimizer. Final search is done over the learning rates for each network, with the search domain of (0.005; 0.0025; 0.00125) yielding a grid of 9 configurations. The best performing configuration is detailed in the table below.

π_{size}	V_{size}	α_π	α_V	O_π	O_V	I_G
64	64	.0025	.0025	RMS	RMS	1

Table V: Best performing AC configuration on LunarLander

MAC

Similarly to LunarLander AC, RMSprop optimizer is used for both networks. Final search is done over learning rates (0.005; 0.0025), Q-network size (1 layer 64 units; 2 layers 64 units), general learning iterations (1; and 5) and value learning iterations (1; and 3). This results in a grid of 24 configurations. The best performing configuration is detailed in the table below.

π_{size}	Q_{size}	α_π	α_V	O_π	O_V	I_G	I_V
64	[64,64]	.0025	.005	RMS	RMS	5	3

Table VI: Best performing MAC configuration on LunarLander

WMPG

Similarly to CartPole WMPG, learning rates, optimizers and network sizes are fixed to be equal to AC best configuration. The WMPG search is done for WM learning rates (0.005; 0.0025), horizons (3; 6; 9), λ (0.25; 0.5; 0.75) and k (4; decaying). The best performing WMPG setup is described in the table below.

T_{size}	R_{size}	α_T	α_R	O_T	O_R	I_G	I_V	I_{WM}	h	λ
96	96	.005	.005	Adam	Adam	4	3	5	3	0.75

Table VII: Best performing configuration of WMPG. Other settings are taken from AC

With k decaying by one every 250 episodes.

4.3.3 Pong

Due to greater computational cost of experiments, Pong hyperparameter search is more restricted and focuses on the aspect of representation learning. All models use a discount factor of 0.99, frame-skip of four and batch size of 512. Moreover, all models use a multiplicative learning rate scheduler with multiplier of 0.99, step size of 5. Each configuration is run with 1 seed.

VAE

Two architectures were considered: convolutional and fully-connected variational auto-encoder. For each architecture, two sizes of latent bottleneck were tested (16; and 8). The best performing VAE configuration has two fully-connected layers with sizes of 512 and 16 units respectively and a mirrored decoder.

AC

We search two AC configuration (with entropy coefficients of 0.001 and 0.01). The best performing configuration is detailed in the table below.

π_{size}	V_{size}	α_π	α_V	O_π	O_V	I_G
512	512	.001	.001	RMS	RMS	1

Table VIII: Best performing AC configuration on Pong

With entropy coefficient equal to 0.01.

WMPG

Similarly to previous environments, the best performing configuration of AC is reflected in WMPG. We fix the transition and reward network sizes to twice the size of policy and value networks. Moreover, all networks take a stack of four previous frames as input. We choose to use Adam optimizer for WM components with learning rate twice larger than policy and value networks, as this setup was working well for previous experiments. We tested variants of WMPG differ with k (1; 2) and the entropy coefficient (0.01; none). The WMPG hyperparameter setting is described in the table below.

T_{size}	R_{size}	α_T	α_R	O_T	O_R	I_G	I_V	I_{WM}	h	λ	k
1028	1028	.002	.002	Adam	Adam	5	1	1	5	0.75	1

Table IX: WMPG hyperparameter setting. Other settings are taken from AC

Chapter 5

Results

In this chapter, the results of experiments are presented. The chapter is divided into four sections: **CartPole**; **LunarLander**; **Pong**; and **Ablations**.

In the first three chapters, the proposed approach is evaluated against benchmark algorithms on three environments of increasing complexity.

In the final chapter, the ablation studies are presented. The proposed approach is evaluated for varying setup of Q-value simulation, as well as different number of actions sampled.

5.1 CartPole

We investigate the average accumulated rewards of the best performing configurations. Figure I summarizes the results over 50 seeds.

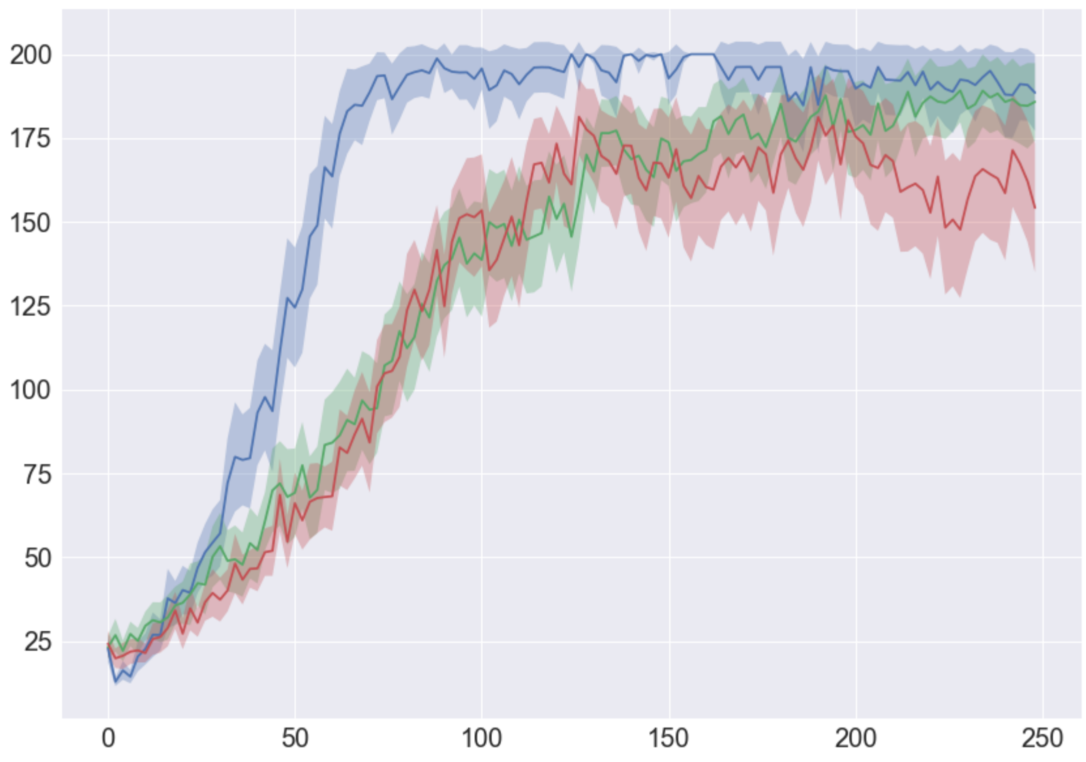


Figure I: Average reward accumulated during training over 50 random seeds. blue is WMPG; green is AC; and red is MAC. y-axis denotes average accumulated reward and x-axis denotes episode number; shadow denotes distance of two standard deviations of the mean

The proposed approach averages at highest training returns, with convergence to the optimal policy around 60th episode. Furthermore, the WMPG distance of two standard deviations of the mean does not overlap with other models, suggesting significantly better performance. Contrary to results reported by [Asadi et al., 2017], MAC was not found to be performing better than regular AC. While the performance of MAC is consistent between this thesis and [Asadi et al., 2017], Asadi et al. report slower convergence on AC. With CartPole being defined as solved when an algorithm averages on more than 190 reward over 50 episodes, WMPG solves the task on 130th episode; while AC and MAC do not solve the task in the constrained amount of environment interactions.

5.2 LunarLander

To accommodate the complexity of the LunarLander environment, the algorithms are run for 1000 episodes. Figure II shows average cumulative rewards gathered during training over 10 seeds.

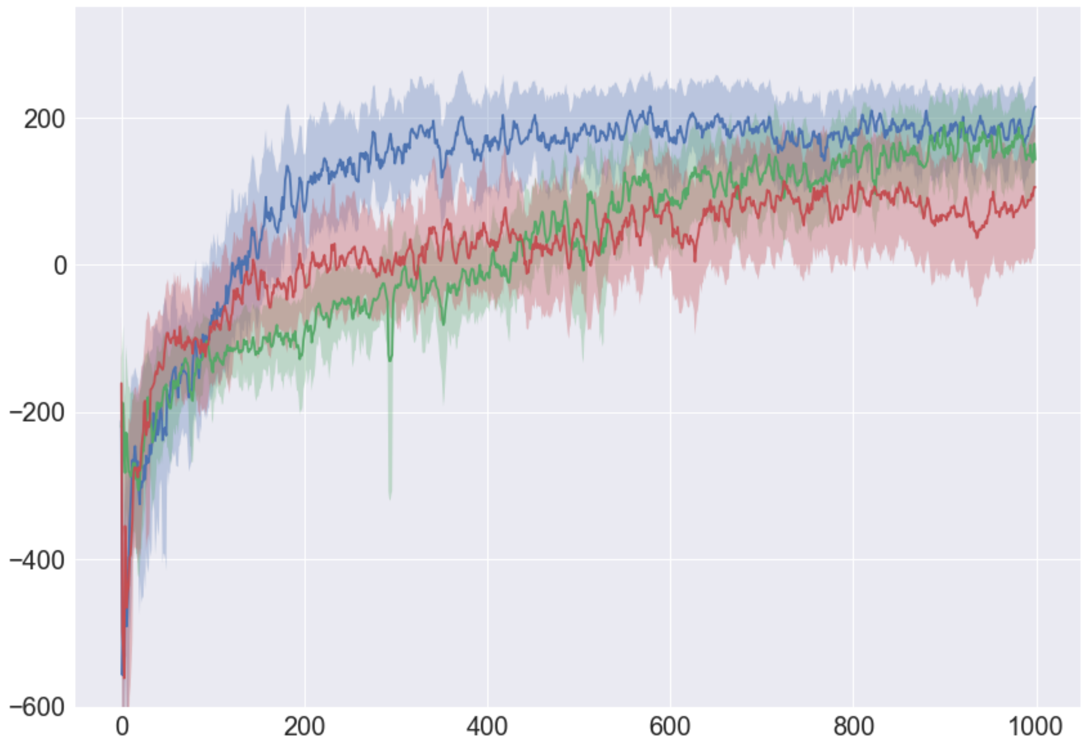


Figure II: Average reward accumulated during training over 10 random seeds; green is AC. blue is WMPG; and red is MAC. y-axis denotes average accumulated reward and x-axis denotes episode number; shadow denotes distance of two standard deviations of the mean; results are slightly smoothened for readability

Similarly to CartPole, proposed method averages at highest returns during training. The experiment indicates that WMPG is converging to the optimal policy faster than the benchmark algorithms, albeit the distance of two standard deviations does not overlap only on an interval between 200th and 400th episode. All WMPG seeds achieve very low returns in the first few episodes, which corresponds to world model producing random output. The k decay every 250th episodes does not produce visible changes in the tempo of the convergence.

5.3 Pong

As shown on the Figure III, WMPG performed better on the two tested seeds. We also observe that VAE-AC learns faster as compared to regular no-VAE AC results reported in ([Mnih et al., 2015]; [Schulman et al., 2015a]; [Asadi et al., 2017]; and [Wu et al., 2017]). This result is expected, given much smaller dimensionality of the policy search problem and is reflected in other research building on WMs ([Hafner et al., 2018]; [Igl et al., 2018]; [Kaiser et al., 2019] and [Hafner et al., 2019]). However, we observe only one seed converging to optimal policy of 21 reward per episode. Thus, we hypothesize that learning from a compressed VAE representation seems to facilitate local optimas with suboptimal policies.

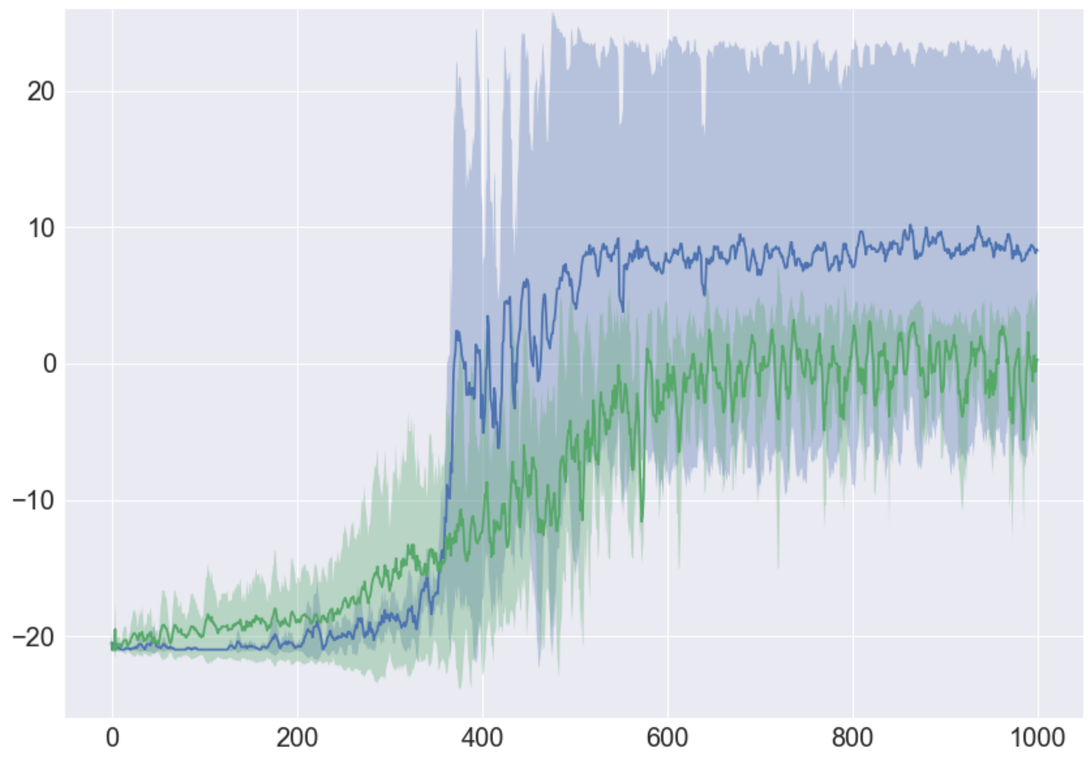


Figure III: Average reward accumulated during training over 2 random seeds. blue is VAE-WMPG; and green is VAE-AC. y-axis denotes average accumulated reward and x-axis denotes episode number; shadow denotes the distance of one standard deviation; results are slightly smoothened for readability

5.4 Ablation studies

In the subsections below, we investigate different aspects of the proposed approach. Firstly, we compare performance of WMPG for different number of sampled actions. Further, we investigate the learning curves of WMPG given different horizon lengths and λ.

5.4.1 Different k

As argumented in Subsection 3.2.3, exact calculation of state policy gradient is not necessarily the best choice. Besides increasing the required compute, exact calculations might impair the model by simulating gradients on low-likelihood trajectories that are unknown for the world model, and thus create a bias. Figure IV details the performance of WMPG given different number of actions sampled without replacement.

The confidence intervals of both settings on CartPole are overlapping. This implies that exact gradient calculation does not have any effect on the performance of WMPG

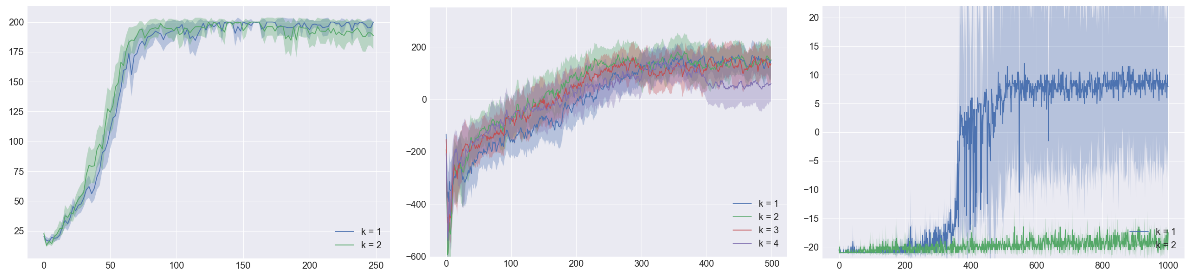

(a) CartPole; 50 seeds **(b)** LunarLander; 10 seeds **(c)** Pong; 2 seeds

Figure IV: WMPG learning curves given different horizon lengths and λ. All models use $h = 3$ and $\lambda = 0.75$. blue samples one action ($k = 1$); green samples two actions ($k = 2$); red samples three actions ($k = 3$); and purple samples four actions ($k = 4$). y-axis denotes average accumulated reward and x-axis denotes episode number; shadow denotes the distance of two standard deviations of the mean. Single action WMPG is baselined with value function function output for the current state

on CartPole. This result is consistent with the argumentation laid in Appendix A: with only two actions, sampling one action with a baseline yields policy gradient with very low variance. Similarly to CartPole, we do not observe statistically significant differences between performance of different WMPGs on LunarLander. Single action WMPG seems to be performing slightly worse than multi-trajectory versions. We believe that this performance difference can be explained by different baseline strategy used for $k = 1$ and $k \neq 1$. The analysis of only 500 episodes does not allow for any conclusions about effects of different k in the later stages of training. Interestingly, sampling all actions on Pong environment leads to almost random performance.

5.4.2 Horizon length and λ

Further, we investigate WMPG performance on the on CartPole environment given different horizon length and λ:

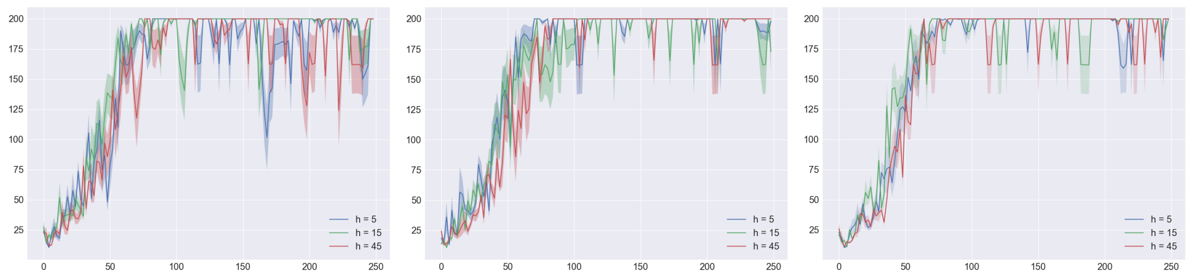
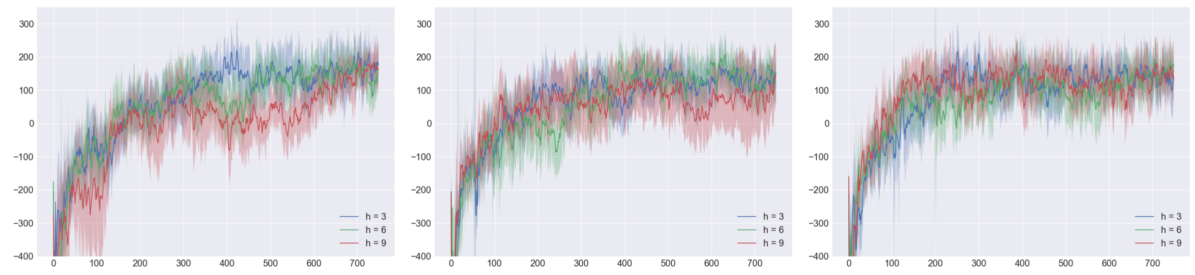

(a) $\lambda = 0.25$ **(b)** $\lambda = 0.5$ **(c)** $\lambda = 0.75$

Figure V: WMPG learning curves given different horizon lengths and λ. y-axis denotes the average reward accumulated over 8 seeds and x-axis denotes the episode number. Colours denote different imagination horizon lengths. Shadow denotes the distance of two standard deviations of the mean. Colours denote different horizon lengths, with [blue]: $h = 5$; [green]: $h = 15$; and [red]: $h = 45$

Interestingly, longer horizon lengths did not translate to bigger variance of the returns on CartPole environment.



(a) $\lambda = 0.25$ **(b)** $\lambda = 0.5$ **(c)** $\lambda = 0.75$

Figure VI: WMPG learning curves given different horizon lengths and λ. y-axis denotes the average reward accumulated over 8 seeds and x-axis denotes the episode number. Colours denote different imagination horizon lengths. Shadow denotes the distance of two standard deviations of the average. Colours denote different horizon lengths, with [blue]: $h = 3$; [green]: $h = 6$; and [red]: $h = 9$

As shown on Figure VI, the complexity of LunarLander environment translates to bigger hyperparameter dependency of WMPG.

Chapter 6

Conclusions

In the final chapter of this thesis, conclusions from the performed experiments are drawn. The chapter is divided into two sections: **Discussion**; and **Conclusions and future work**.

In the first section, experimental results are revisited in the context of the original research questions. Further, we discuss alternative design choices for WMPG that were not tested in this thesis.

In the second section, future work perspectives are visited and concluding remarks are drawn.

6.1 Discussion

As we have shown, world model-based policy gradient approximation is a promising approach for reducing variance of the gradient approximator by allowing the agent to simulate additional trajectories. Furthermore, as stems from the experimental results shown in Section 5, WMPG has potential for increased sample efficiency in a variety of RL environments.

6.1.1 Research questions revisited

We start this section by revisiting our research questions.

What is the effect of approximating policy gradient for given state with Monte Carlo approximator?

As shown in Appendix A, stochasticity of MC PG approximator increases the amount of samples required for policy convergence. We have shown that exactly calculated PG yields policy probability changes which are proportional to the Q-values of individual actions. Further, we sketched two effects that single-sample MC approximation has on the policy changes in a given state:

- Given infinitesimal learning rate, the direction of the policy probability change of performed action is dependent on the Q-value of that action
- The magnitude of the policy probability changes of non-performed actions is independent of their Q-values

The first effect was mentioned in [Williams, 1992] and allows to easily pinpoint why baseline variance reduction would effect variance given known Q-values. To the best of our knowledge, the second effect was not pinpointed in the earlier research. We show the second effect to be true on an example MDP, but we fail to prove it for more general cases.

Can agent learn optimal policy from trajectories simulated by an online trained world model?

As shown in Section 5, policy can converge learning solely from trajectories simulated by an online trained world model. WM-based learning is a promising approach that was shown to achieve good results in both context of planning [Hafner et al., 2018] and RL [Hafner et al., 2019]. WMs are also known to have problems with representation learning ([Kipf et al., 2019]; and [Kaiser et al., 2019]). To that end, we have shown that, marginalizing the problem of representation learning, WM-based approach yield superior results as compared to traditional PG algorithms.

Does simulating those trajectories with reward, transition and value networks perform better than single Q-network?

As shown in Section 5, WMPG achieved better average results than our implementation of MAC. The main differences between MAC and WMPG lie in the way that Q-values are approximated. Whereas MAC approximates the Q-values of all actions for given state with a Q-network trained with MCPE, WMPG allows to unroll the policy arbitrary amount of steps and calculate the Q-values with TD(λ) for any number of actions. We explain better robustness of the WMPG approximator twofold. Firstly, TD(λ) is known to solve tasks which are unsolvable with MCPE and is generally preferable to MCPE/TD(1) [Sutton and Barto, 2018]. Secondly, compartmentalization of the training into transition, reward and value functions serves as a regularization and allows for better generalization [Gelada et al., 2019]. We hypothesize that both mechanisms lead to better robustness of the estimate, as well as better reactiveness of the Q-value approximations to the policy changes.

What is the effect of estimating the policy gradient with more samples? What can be achieved by including only few sampled trajectories?

As shown in Subsection 2.2, the variance reduction stemming from adding more samples in MC approximation has diminishing returns. This puts a hard cap on returns stemming from parallel agents [Mnih et al., 2016] or vine estimation in TRPO [Schulman et al., 2015a]. Without-replacement sampling estimation of state PG is a promising alternative and was shown to achieve good results [Kool et al., 2019a]. In Appendix C, we have shown that variance reduction of without-replacement sampling estimate does not necessarily diminish with bigger amount of samples. This is the case because state PG approximator is no longer a sum of independent random variables, but a weighted sum dependent random variables with weights smaller than unity. In Subsection 3.2.3, we discuss why sampling more actions per state in the context of WMPG might lead to sub-optimal results and propose to sample variable amount of actions depending on the policy entropy in given state. In Section 5, we show that changing the amount of sampled actions over the course of training was the best performing setting on LunarLander environment. Finally, we show that in the context of WM simulated trajectories, simulating the entire action space does not yield favourable results as compared to sampling a subset of the action space.

6.1.2 Alternative design choices

In the paragraphs below, we discuss the alternative design choices that we could not test within the scope of this thesis.

Representation learning

The proposed approach can be used with any world model learning strategy. As such, it might be valuable to investigate the algorithm's behaviour with different loss functions placed on the world model components. The reconstruction approach used in this thesis is the most crude strategy, which is known to have many modes of failure [Kaiser et al., 2019]. A promising alternative is the hinge bisimulation introduced in [Kipf et al., 2019] and further developed in [van der Pol et al., 2020].

Bootstrapping with real Q-value

The approach proposed in this thesis approximates Q-values with the algorithm components, namely transition, reward and value approximators. As such, the gathered returns are used only to train the value approximator. One could consider using the gathered return for the policy gradient approximation on the executed state-action pairs. In such setting, only the non-executed actions would be simulated with the world model. We hypothesize that such approach could be helpful in the early stages of training in particular, as then the world model output is least reliable.

Different supervision of the value approximator

The proposed approach supervises value approximator with MC policy rollouts. As discussed in [Sutton and Barto, 2018], the value approximator can be trained with a variety of techniques. Therefore, one could consider supervising the value network in WMPG with TD or TD(λ) in particular (as those values are already calculated). Such approach would allow for off-policy learning in WMPG, by collecting data according to some exploratory policy and updating the networks according to on-policy values simulated by the world model.

6.2 Conclusions and future work

The unification of world model and policy gradient frameworks is an exciting research direction, offering potential in sample efficiency, exploration and explainable behaviour. WM-based algorithms operating on high-dimensional spaces exhibit very good convergence time as compared to traditional approaches at the cost of lower stability. Those stability issues seem to be connected to state-compression, which in the context of WMs is often learned independently of the policy ([Ha and Schmidhuber, 2018]; [Kipf et al., 2019]; and [Kaiser et al., 2019]). As such, learning robust environment representations independently of optimal control is an active research topic. Work done in this thesis is orthogonal to the problem of WM learning. Therefore, any advances in representation and WM learning will translate to better robustness of WMPG on high-dimensional spaces.

One potential track of future work is to accommodate WMPG for environments with different assumptions. For partially-observable environments, the state embedding function can be represented by a recurrent network. In such setting that state embedding maps a sequence of observations into latent state representation. This approach has been proven successful in multiple experiments ([Igl et al., 2018]; and [Hafner et al., 2019]). The without-replacement sampling strategy used in this thesis does not directly translate to continuous action spaces. Due to infinitesimal probabilities of individual actions in continuous action space, without-replacement sampling would most likely lead to numerical instability. Furthermore, the approach would be somewhat inconsistent with the dominating strategy of defining policy as deterministic [Silver et al., 2014]. As such, we consider that some strategy of policy discretization would allow to incorporate policy probabilities on continuous action space, without risking numerical underflow. Furthermore, we think that without-replacement sampling approximation of state PG is a promising approach, as it allows to circumvent the diminishing returns of parallelization with MC. Thus, we suggest that future work focuses on establishing heuristics for choice of amount of samples without replacement in the context of without-replacement sampling PG approximation.

On that note, we conclude this thesis. We have proposed an approach of unifying gradient-based policy search with world models framework. We have demonstrated that the proposed approach can achieve superior sample efficiency as compared to AC and MAC. The experimental results paired with the existing research provide a strong incentive for further research in WM-based RL.

Bibliography

[Asadi et al., 2017] Asadi, K., Allen, C., Roderick, M., Mohamed, A.-r., Konidaris, G., Littman, M., and Amazon, B. U. (2017). Mean actor critic. *stat*, 1050:1.

[Babaeizadeh et al., 2016] Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., and Kautz, J. (2016). Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256.*

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

[Becker et al., 1988] Becker, S., Le Cun, Y., et al. (1988). Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37.

[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, 1957] Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning.* springer.

[Biza et al., 2020] Biza, O., Platt, R., van de Meent, J.-W., and Wong, L. L. (2020). Learning discrete state abstractions with deep variational inference. *arXiv preprint arXiv:2003.04300.*

[Braylan et al., 2015] Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. (2015). Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.*

[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.*

[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(7).

[Duffield et al., 2007] Duffield, N., Lund, C., and Thorup, M. (2007). Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6):32–es.

[Feinberg et al., 2018] Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. (2018). Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*.

[Ferns et al., 2004] Ferns, N., Panangaden, P., and Precup, D. (2004). Metrics for finite markov decision processes. In *UAI*, volume 4, pages 162–169.

[Fu et al., 2019] Fu, J., Kumar, A., Soh, M., and Levine, S. (2019). Diagnosing bottlenecks in deep q-learning algorithms. *arXiv preprint arXiv:1902.10250*.

[Gelada et al., 2019] Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. (2019). Deepmdp: Learning continuous latent space models for representation learning. *arXiv preprint arXiv:1906.02736*.

[Giunchiglia and Walsh, 1992] Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial intelligence*, 57(2-3):323–389.

[Gorban and Wunsch, 1998] Gorban, A. N. and Wunsch, D. C. (1998). The general approximation theorem. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 2, pages 1271–1274. IEEE.

[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.

[Greff et al., 2016] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

[Gu et al., 2017] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

[Ha and Schmidhuber, 2018] Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462.

[Hafner et al., 2019] Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

[Hafner et al., 2018] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*.

[Hanin, 2019] Hanin, B. (2019). Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7(10):992.

[Heinecke et al., 2020] Heinecke, A., Ho, J., and Hwang, W.-L. (2020). Refinement and universal approximation via sparsely connected relu convolution nets. *IEEE Signal Processing Letters*.

[Hesterberg, 1988] Hesterberg, T. C. (1988). *Advances in importance sampling*. PhD thesis, Citeseer.

[Horvitz and Thompson, 1952] Horvitz, D. G. and Thompson, D. J. (1952). A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685.

[Igl et al., 2018] Igl, M., Zintgraf, L., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep variational reinforcement learning for pomdps. *arXiv preprint arXiv:1806.02426*.

[Kaiser et al., 2019] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

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

[Kipf et al., 2019] Kipf, T., van der Pol, E., and Welling, M. (2019). Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*.

[Kool et al., 2019a] Kool, W., van Hoof, H., and Welling, M. (2019a). Buy 4 reinforce samples, get a baseline for free!

[Kool et al., 2019b] Kool, W., Van Hoof, H., and Welling, M. (2019b). Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. *arXiv preprint arXiv:1903.06059*.

[Larsen and Skou, 1991] Larsen, K. G. and Skou, A. (1991). Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28.

[LeCun et al., 2006] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

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

[Li et al., 2006] Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a unified theory of state abstraction for mdps. In *ISAIM*.

[Machado et al., 2018] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, 44(247):335–341.

[Mnih et al., 2016] 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*, pages 1928–1937.

[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):529–533.

[Moerland et al., 2018] Moerland, T. M., Broekens, J., Plaat, A., and Jonker, C. M. (2018). A0c: Alpha zero in continuous action space. *arXiv preprint arXiv:1805.09613*.

[Nevmyvaka et al., 2006] Nevmyvaka, Y., Feng, Y., and Kearns, M. (2006). Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871.

[Oh et al., 2017] Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[Peshkin, 2003] Peshkin, L. (2003). Reinforcement learning by policy search.

[Puterman, 2014] Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[Ravindran and Barto, 2001] Ravindran, B. and Barto, A. G. (2001). Symmetries and model minimization in markov decision processes.

[Ravindran and Barto, 2003] Ravindran, B. and Barto, A. G. (2003). Relativized options: Choosing the right transformation. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 608–615.

[Ravindran and Barto, 2004] Ravindran, B. and Barto, A. G. (2004). Approximate homomorphisms: A framework for non-exact minimization in markov decision processes.

[Schäfer and Zimmermann, 2006] Schäfer, A. M. and Zimmermann, H. G. (2006). Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, pages 632–640. Springer.

[Schrittwieser et al., 2019] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019). Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.

[Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

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

[Shah and Kroese, 2018] Shah, R. and Kroese, D. P. (2018). Without-replacement sampling for particle methods on finite state spaces. *Statistics and Computing*, 28(3):633–652.

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

[Shi et al., 2020] Shi, K., Bieber, D., and Sutton, C. (2020). Incremental sampling without replacement for sequence models. *arXiv preprint arXiv:2002.09067*.

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms.

[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):354–359.

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

[Taylor et al., 2009] Taylor, J., Precup, D., and Panagaden, P. (2009). Bounding performance loss in approximate mdp homomorphisms. In *Advances in Neural Information Processing Systems*, pages 1649–1656.

[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.

[Tishby et al., 2000] Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. *arXiv preprint physics/0004057.*

[van der Pol et al., 2020] van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020). Plannable approximations to mdp homomorphisms: Equivariance under actions. *arXiv preprint arXiv:2002.11963.*

[Villani, 2008] Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

[Vinyals et al., 2019] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

[Walt et al., 2011] Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in science & engineering*, 13(2):22–30.

[Wiering and Van Otterlo, 2012] Wiering, M. and Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12:3.

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

[Wu et al., 2017] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288.

[Xia et al., 2016] Xia, W., Li, H., and Li, B. (2016). A control strategy of autonomous vehicles based on deep reinforcement learning. In *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, volume 2, pages 198–201. IEEE.

Appendix A

Variance of MCA PG estimator

To contextualize why decreasing variance would translate to faster learning, a toy MDP with a single state and three actions (a_1, a_2, a_3) is considered. In this MDP, after performing any action in any state, agent deterministically executes trajectory τ_a with Q-value equal to the index of the action. Given full knowledge, it is clear that a_3 is the optimal action, with $Q(s, a_3) = 3$. Similarly, a_1 is the action that yields least return. However, when the agent starts learning, it has no knowledge of the environment and thus, it follows a random policy. To aid the analysis, variance of policy gradient updates to the agent's policy is linked by defining:

Definition 1 *If agent updates policy network π_θ with $\theta \to \hat{\theta}$, where $\hat{\theta} = \theta + \alpha\nabla_\theta J(\theta)$, then $\zeta(\tau^*, \nabla_\theta)$ denotes absolute changes to probability of trajectory τ^* which resulted from gradient update ∇_θ:*

$$\zeta(\tau^*, \nabla_\theta) = \pi_{\hat{\theta}}(\tau^*) - \pi_\theta(\tau^*) \tag{A.1}$$

As such, $\zeta(\tau^*, \nabla_\theta) < 0$ implies that the probability of trajectory τ^* has decreased after performing policy gradient update. Analogically, if $\zeta(\tau^*, \nabla_\theta) > 0$, then the probability of executing τ^* according to the learner's policy has increased.

In the following paragraphs, properties of policy gradient are investigated in the context of $\zeta(\tau^*, \nabla_\theta)$. It is shown that, when the gradient is calculated exactly, expected value of $\zeta(\tau^*, \nabla_\theta)$ is proportional to the returns from performing τ^*. Furthermore, the effects of single-sample PG MCA on $\zeta(\tau^*, \nabla_\theta)$ are investigated. It is shown that MCA shifts the distributions of $\zeta(\tau^*, \nabla_\theta)$ of all trajectories except the one that was executed. For both cases, a simple setup is used, where π_θ is represented by a multilayer perceptron with a single hidden layer, ReLU activation units and softmax normalized output, with $\zeta(\tau^*, \nabla_\theta)$ being calculated either exactly or with single-trajectory MCA over 10 000 different policy configurations.

A.1 Exact policy gradient calculation

First, behaviour of $\zeta(\tau^*, \nabla_\theta)$ when the policy gradient at state s is calculated exactly is considered:

$$\nabla_\theta J(\theta, s) = \sum_{\tau=1}^{\omega} \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) G(s|\tau) = \nabla_\theta \sum_{i=1}^{3} \pi_\theta(\tau_i) G(s|\tau_i) \tag{A.2}$$

By calculating the policy gradient exactly for some state s and known $G(s|\tau)$, the approximation variance is reduced to zero. The only variance in the gradient stems from different values of θ. Values of $\zeta(\tau, \nabla_\theta)$ are investigated for all three trajectories of the example MDP, given exact calculation of the gradient according to equation A.2, for different policy settings. Results are shown in Figure I.

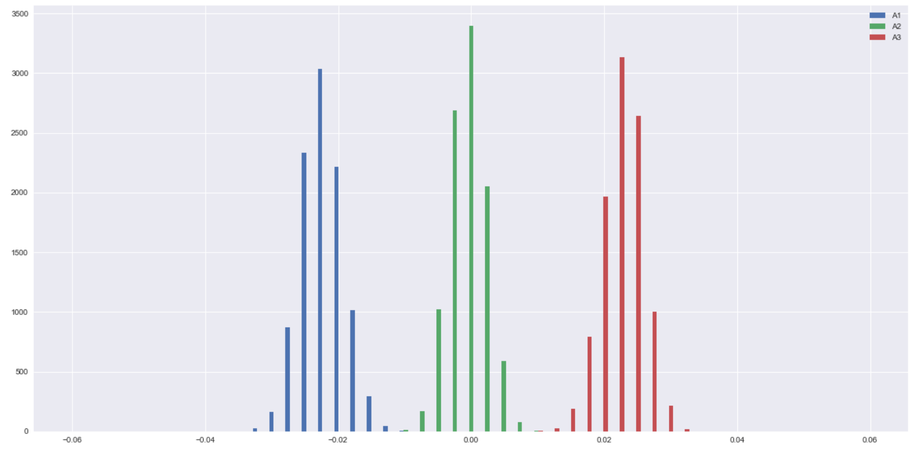


Figure I: Change of probability of an action after performing an exact gradient update. $y - axis$ represents count of observations; $x - axis$ represent value of $\zeta(\tau, \nabla_\theta)$. blue is τ_1; green is τ_2; and red is τ_3.

When calculating the policy gradient exactly, values of $\zeta(\tau, \nabla_\theta)$ are centered around zero. Given some trajectory τ, value of $\zeta(\tau, \nabla_\theta)$ is proportional to the return of this trajectory, with $\mathbb{E}_{\pi_\theta \sim \Pi}[\zeta(\tau_1, \nabla_\theta)] < 0$; $\mathbb{E}_{\pi_\theta \sim \Pi}[\zeta(\tau_2, \nabla_\theta)] = 0$ and $\mathbb{E}_{\pi_\theta \sim \Pi}[\zeta(\tau_1, \nabla_\theta)] > 0$. Similarly, the more distanced the values of $G(s|\tau)$ from 0, the bigger the variance of policy updates. This is visible in the fact that τ_2 has more peaked distribution given the same number of updates as other trajectories.

A.2 Policy gradient approximation

Values of $\zeta(\tau, \nabla_\theta)$ shown in Figure I stem from exact gradient calculation, and thus have zero approximation stochasticity. However, it is popular to approximate the policy gradient needed for a single update with a single trajectory [Kool et al., 2019a]. As follows from equation 2.43, such operation increases the variance of policy gradient estimator, which in turn increases the variance of $\zeta(\tau, \nabla_\theta)$ wrt. θ. Following the standard practice, the policy gradient is approximated with a single sample MCA:

$$\nabla_\theta^\tau = \nabla_\theta J(\theta, s|\tau) = \nabla_\theta \log \pi_\theta(\tau) G(s|\tau) \quad \textit{with} \quad \tau \sim \pi_\theta \tag{A.3}$$

Now, it is clear that the policy gradient is dependent solely on the trajectories that were sampled in the approximation process. This affects the values of $\zeta(\tau, \nabla_\theta)$, as the policy is updated with $\hat{\theta} = \theta + \alpha \nabla_\theta^\tau$. To explicitly account for this effect in the notation, denote $\zeta(\tau^*, \nabla_\theta^\tau)$ to represent changes to $\pi(\tau^*)$ after updating the policy with gradient approximated with experience τ. The figure II depicts $\zeta(\tau^*, \nabla_\theta^{\tau^*})$, that is change of probability of a trajectory $\pi(\tau^*)$ after approximating the policy gradient with exactly this trajectory τ^*.

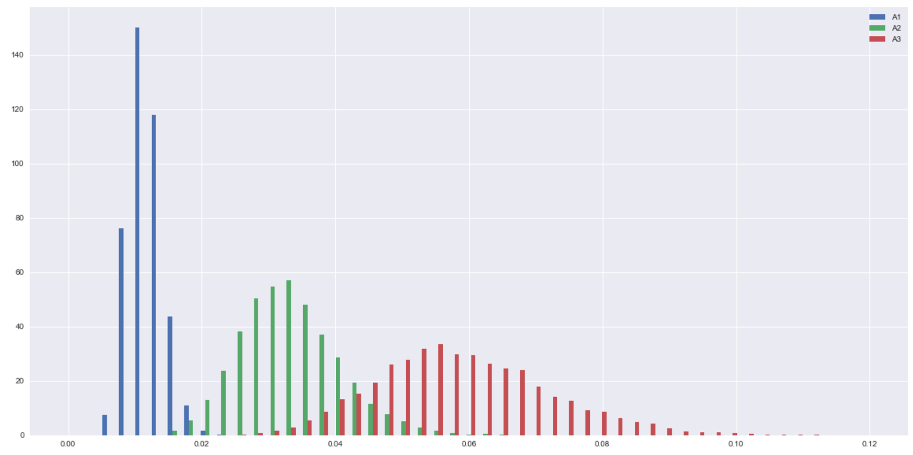


Figure II: Change of probability of a trajectory after performing it. $y - axis$ represents count of observations; $x - axis$ represent value of $\zeta(\tau^*, \nabla_\theta^{\tau^*})$. blue is τ_1; green is τ_2; and red is τ_3.

Figure II shows that approximating the gradient with a single trajectory indeed increases the variance of policy gradient estimator. Moreover, $\zeta(\tau^*, \nabla_\theta^{\tau^*})$ is no longer centered around zero and is positive for all trajectories. That implies that after executing and learning from the action with lowest return (τ_1), algorithm still increases probability of sampling it in the next episode. This stems from the definition of the policy gradient:

Definition 2 *If agent updates the policy network parameters $\theta \xrightarrow{\tau} \hat{\theta}$ by gradient approximated with experience τ, where $\hat{\theta} = \theta + \alpha \nabla_\theta^\tau J(\theta, s)$ and $J(\theta, s) = \log \pi_\theta(\tau) G(s|\tau)$, then for infinitesimally small learning rate α it follows that:*

$$\begin{aligned} \zeta(\tau, \nabla_\theta^\tau) > 0 &\iff G(s|\tau) > 0 \\ \zeta(\tau, \nabla_\theta^\tau) = 0 &\iff G(s|\tau) = 0 \\ \zeta(\tau, \nabla_\theta^\tau) < 0 &\iff G(s|\tau) < 0 \end{aligned} \tag{A.4}$$

This stems from the fact that $\hat{\theta}$ in $\pi_{\hat{\theta}}(\tau^*) - \pi_\theta(\tau^*)$ per definition changes in the direction *of the gradient. The direction of the gradient depends solely on the sign of $G(s|\tau)$, given that $\log \pi_\theta(\tau)$ represent log-likelihood. Detailed proof is presented in [Williams, 1992].*

In the example MDP, it is the case that $G(s|\tau) > 0$ for all τ. If all actions are constantly updated towards the same direction, then how does the policy gradient converge? The magnitude of changes to $\pi(\tau)$ depend on $\log \pi_\theta(\tau)$ and $G(s|\tau)$. The bigger $G(s|\tau)$, the bigger on average $\zeta(\tau, \nabla_\theta^\tau)$ [Sutton and Barto, 2018]. Thus, algorithm converges by increasing the probability of a trajectory more, the bigger the return of this trajectory. This leads to a inefficiency, where performing an exploratory action with comparatively low return decreases probability of all other action, and thus requires the agent to perform greedy trajectories to leverage the information from exploration. One potential solution to this problem is to subtract baseline from state returns. Then, the algorithm can leverage above property and reduce the probability of trajectories that perform worse than some defined level of returns represented by the baseline.

After centering the returns in the example MDP around 0, $\zeta(\tau_1, \nabla_\theta^{\tau_1})$ indeed becomes negative. Since $\sum_\tau \pi(\tau) = 1$, it follows that under new policy some other trajectories are more probable. As shown in Figure I, if the gradient was calculated exactly, then it follows that the probability mass would be transferred towards other trajectories proportionally to the return that comes from executing that trajectory. Transfer of this probability mass in the case of MC approximation is investigated below.

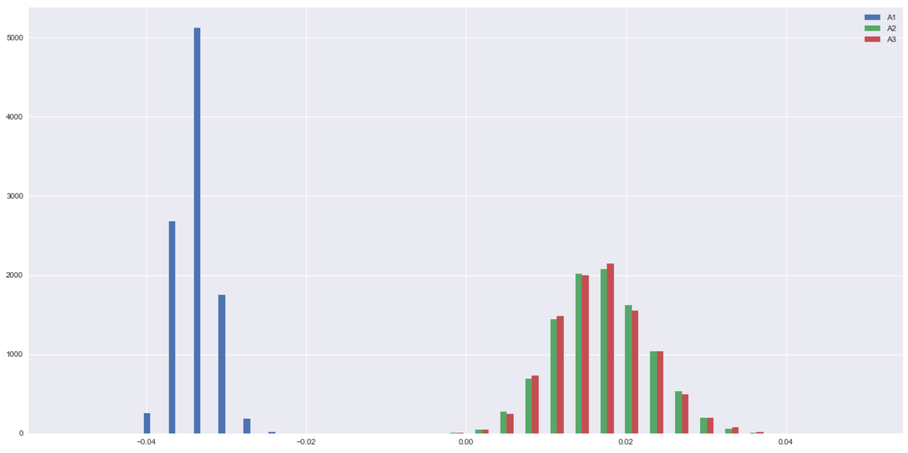


Figure III: Change of probability of trajectories after learning from τ_1. $y-axis$ represents count of observations; $x - axis$ represent value of $\zeta(\tau, \nabla_\theta^{\tau_1})$. blue is τ_1; green is τ_2; and red is τ_3. Here, returns are centered around zero.

Figure III shows that the value of $\zeta(\tau_2, \nabla_\theta^{\tau_1})$ and $\zeta(\tau_3, \nabla_\theta^{\tau_1})$ are independent of $G(\tau_2|s)$ and $G(\tau_3|s)$ respectively. This follows from the gradient approximation strategy, where the gradient of some trajectory τ^* is approximated with τ.

Appendix B

Unbiasedness of the without-replacement estimator

We assume X to be a discrete random variable with domain of $|\mathcal{A}|$ values $(x_1, ..., x_{|\mathcal{A}|})$ and respective probabilities $(p_1, ..., p_{|\mathcal{A}|})$. The expected value of X is given by:

$$\mathbb{E}\left[X\right] = \sum_{a=1}^{|\mathcal{A}|} p_a x_a \tag{B.1}$$

We denote $\hat{x}$ as the following estimator of $\mathbb{E}\left[X\right]$:

$$\hat{x} = \sum_{i=1}^{k} \beta_i x_i \tag{B.2}$$

Where k is the number of without-replacement samples drawn from X, x_i is the i^{th} sample drawn without replacement and β_i is a weight independent of determined by x_i. We want to show that for any constant value of $k \leq |\mathcal{A}|$, $\hat{X}$ is unbiased is expectation.

$$\mathbb{E}\left[\hat{X}\right] = \mathbb{E}\left[X\right] \tag{B.3}$$

We first note that $\hat{X}$ is defined as k without-replacement samples from X. Therefore, $\hat{X}$ has a domain of $\binom{\mathcal{A}}{k} = K$ values denoted as $\hat{X} = (\hat{x}_1, ..., \hat{x}_K)$ and respective probabilities $(q_1, ..., q_K)$. Therefore:

$$\mathbb{E}\left[\hat{X}\right] = \sum_{j=1}^{K} q_j \hat{x}_j = \sum_{j=1}^{K} q_j \sum_{i=1}^{k} \beta_i x_i = \sum_{j=1}^{K} q_j \beta_1 x_1^j + q_j \beta_2 x_2^j + ... + q_j \beta_k x_k^j \tag{B.4}$$

Now, we note that x_a can be sampled in some number of $\hat{x}$. We denote the set of indices of $\hat{X}$ that contain x_a as L. Then, by expanding the sum:

$$\mathbb{E}\left[\hat{X}\right] = \sum_{a=1}^{|\mathcal{A}|} \beta_a x_a \sum_{j \in L} q_j \tag{B.5}$$

The above operation explicitly assumes that β_a is independent of j. Combining Equations B.1, B.3 and B.5 yields:

$$\sum_{a=1}^{|\mathcal{A}|} p_a x_a = \sum_{a=1}^{|\mathcal{A}|} \beta_a x_a \sum_{j \in L} q_j \tag{B.6}$$

Thus, when $\beta_a = \frac{p_a}{\sum_{j \in L} q_j}$, the estimator is an unbiased estimator of $\mathbb{E}\left[X\right]$.

Appendix C

Variance of the without-replacement policy gradient estimator

Here, we show why calculating policy gradient without replacement might lead to a better variance reduction stemming from increase of samples. To not overload the notation, we assume the Q-values to be known. Given a batch of states $D_s = (s_1, s_2, ..., s_{|D_s|})$, gradient $\nabla_\theta \hat{J}(\theta)$ is calculated with batch average. Without assuming independence between sampled states, variance of the policy gradient estimator is equal to:

$$Var\left[\nabla_\theta \hat{J}(\theta)\right] = \frac{1}{|D_s|^2} \sum_{s \in D_s} \left(Var\left[\nabla_\theta \hat{J}(\theta, s)\right] + \sum_{s' \neq s \in D_s} Cov\left[\nabla_\theta \hat{J}(\theta, s), \nabla_\theta \hat{J}(\theta, s')\right] \right) \tag{C.1}$$

Where $\nabla_\theta \hat{J}(\theta, s)$ denotes evaluation of the policy gradient at state s. If $\nabla_\theta \hat{J}(\theta, s)$ is calculated via MC with $|D_a|$ samples, then:

$$Var\left[\nabla_\theta \hat{J}^{mc}(\theta, s)\right] = \frac{Var\left[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)\right]}{|D_a|} \tag{C.2}$$

Taking derivative wrt. $|D_a|$ reveals that the decrease of variance stemming from taking more samples is rapidly diminishing. Furthermore, if the variance is non-zero, then $Var\left[\nabla_\theta \hat{J}^{mc}(\theta, s)\right]$ converges to 0 for $|D_a| = \infty$. If we treat $|D_a|$ as the number of actions sampled without replacement, then for $|D_a| = |\mathcal{A}|$ (ie. when sampling all actions) $\nabla_\theta \hat{J}(\theta, s)$ is calculated with exact expectations:

$$\nabla_\theta \hat{J}^{wr}(\theta, s) = \sum_{a \in |\mathcal{A}|} \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \tag{C.3}$$

Thus, given $|D_a| = |\mathcal{A}|$ and known Q-values, it follows that $Var\left[\nabla_\theta \hat{J}^{wr}(\theta, s)\right] = 0$. As such, assuming $|D_a| < |\mathcal{A}|$ it might be the case that $Var\left[\nabla_\theta \hat{J}^{wr}(\theta, s)\right] < Var\left[\nabla_\theta \hat{J}^{mc}(\theta, s)\right]$. Borrowing the notation from Appendix B, the variance of without-replacement estimator is equal to:

$$Var\left[\nabla_\theta \hat{J}^{wr}(\theta, s)\right] = \sum_{j=1}^{K} q_j \left(\sum_{i=1}^{|D_a|} \beta_i x_i\right)^2 - \left(\nabla_\theta J(\theta, s)\right)^2 \tag{C.4}$$

For further reading on the without-replacement estimation variance, we point the reader to ([Horvitz and Thompson, 1952]; [Kool et al., 2019b]; [Shah and Kroese, 2018]).