Beyond Learning Curves

Understanding Stochasticity and Learned Solution Modes in Reinforcement Learning

by

Matthew Wilson

B.S., University of Utah, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

September 2022

© Matthew Wilson, 2022
The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

**Beyond Learning Curves: Understanding Stochasticity and Learned Solution Modes in Reinforcement Learning**

submitted by Matthew Wilson in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

**Examinining Committee:**

Michiel van de Panne, Professor, Computer Science, UBC  
*Supervisor*

Jeff Clune, Professor, Computer Science, UBC  
*Supervisory Committee Member*
Abstract

While deep reinforcement learning (Deep RL) algorithms have been used to successfully solve challenging decision making and control tasks, their behavior often remains poorly understood. Studies and comparisons between algorithms are often done through impoverished and partial signals such as learning curves and individual rollout videos. In this work, we follow along a tradition of work which dives deeper into why exactly algorithms produce different rewards from run to run on different tasks. We aim to go beyond learning curves and develop a more holistic view of both the optimization landscape of particular environments and the multimodal behaviors that algorithms produce for given environments. To this end, we develop a set of tools for comparing many runs of deep reinforcement learning algorithms and rollouts from a single policy. We use these to answer a broad range of questions about RL.
Lay Summary

Reinforcement learning (RL) is an area of artificial intelligence inspired by trial-and-error learning in animals and humans. Just as one might give a dog a reward for doing a trick and a punishment for misbehaving, researchers give artificial intelligence (AI) agents rewards based on their performance on a desired task. Through much experience and feedback, and if all goes well, AI agents trained using reinforcement learning are able to gradually learn to solve a desired task. This thesis develops a set of ideas and tools for further understanding and comparing different approaches to reinforcement learning so we can make further progress in the field. Specifically, we aim to develop more nuanced ways of understanding and comparing RL algorithm and resultant policy behavior that go beyond mere learning curves: instead of simple mean and standard deviation summaries of task performance, we explore in detail the specific behaviors that algorithms produce.
Preface

The work presented in this thesis was developed in collaboration with my advisor Michiel van de Panne. I proposed, implemented, and investigated a variety of ideas in the early exploratory phases of the research. The ideas in the thesis resulted from focusing on a subset of the ideas and emerged from ongoing discussions with Michiel. I realized concrete versions of the visualization methods, the design of the test environments, and the final experiments. Michiel contributed feedback related to the goals, experiments, and the writing.

A version of the work presented in Section 5.2 has been published in SIGGRAPH 2022 article titled Learning to Get Up, by Tianxin Tao, Matthew Wilson, Ruiyu Gou, and Michiel van de Panne [34]. My contribution was the method and code for visualizing the humanoid trajectories in 3D space using t-SNE. Tianxin developed the curriculum-based algorithm and did much of the writing. Ruiyu produced the visualizations for the paper. Michiel provided organizational help for the project, feedback related to the ideas and experiments, and contributed to the writing.
# Table of Contents

Abstract ................................................................. iii  
Lay Summary ............................................................ iv  
Preface ................................................................. v  
Table of Contents ................................................... vi  
List of Tables ......................................................... viii  
List of Figures ....................................................... ix  

1 Introduction ......................................................... 1  
  1.1 Research Questions ............................................ 3  
  1.2 Thesis Overview ............................................. 4  

2 Background ......................................................... 7  
  2.1 Neural Networks and Deep Learning ....................... 7  
  2.2 Reinforcement Learning ................................... 8  
    2.2.1 Sources of Stochasticity ............................. 9  
    2.2.2 Challenges of RL ................................... 10  
    2.2.3 DeepRL Algorithms ................................. 10  
  2.3 t-distributed Stochastic Neighbor Embedding (t-SNE) ... 11  

3 Related Work ..................................................... 12  
  3.1 Understanding and Visualizing Optimization Landscapes .. 12
3.2 Empirical Studies and Critiques of RL ........................................ 13
3.3 Policy Similarity Metrics ......................................................... 13
3.4 Visualizing Trained Policy Behavior ......................................... 16
3.5 Quality-Diversity Optimization Algorithms ................................. 17

4 Methods ................................................................. 19
4.1 Visualizing Policies as Points ............................................... 19
4.2 Visualizing Policies as State Space Trajectories ......................... 22
4.3 Custom Environments and Quantitive Metrics ........................... 23

5 Visualizing the Solution Modes Learned by RL ........................... 25
5.1 Visualizing Policies as Points ............................................... 25
5.1.1 LunarLander ............................................................... 26
5.1.2 BipedalWalker ........................................................... 26
5.1.3 HalfCheetah ............................................................... 27
5.2 HumanoidStandup ............................................................ 28

6 Understanding How RL Algorithms Behave for Tasks with Multiple
   Solution Modes ............................................................... 34
6.1 Custom Environments ......................................................... 34
6.1.1 DoubleSlit ................................................................. 35
6.1.2 DoubleSlitTrap ............................................................ 35
6.1.3 FourCorners ............................................................... 36
6.2 FourCorners Alternatives ..................................................... 38
6.3 Multimodal solutions .......................................................... 39
6.4 Discussion ................................................................. 40

7 Conclusion ................................................................. 41

Bibliography ................................................................. 43
List of Tables

Table 6.1  Empirical results on the Four Corners variants using TD3, as computed for 30-35 runs on each of Normal, Gapped, Weighted.  

| Table 6.1 | Empirical results on the Four Corners variants using TD3, as computed for 30-35 runs on each of Normal, Gapped, Weighted. | 39 |
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Learning curves for HalfCheetah</td>
<td>5</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Visualizations of SAC and TD3 runs for HalfCheetah environment</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>1D illustration of the type of optimization landscape discussed in this work</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Visualization of an arbitrary landscape that might be encountered during optimization</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Visualization of an optimization landscape of a ResNet with and without skip connections</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Visualization of HalfCheetah optimization for different algorithms</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Greydanus et al. [9] visualize saliency maps on the Atari Breakout game to determine what the agent is paying attention to. (permission pending)</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Hilton et al. [13] use neural network interpretability tools to determine what the agent assigns positive and negative attribution to. (permission pending)</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Zahavy et al. [39] construct t-SNE plots on Atari games in a fashion related to ours in Figure 4.2. Each point represents a specific game state. (permission pending)</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Example toy task of point mass</td>
<td>20</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Visualization of policies as state space trajectories</td>
<td>21</td>
</tr>
</tbody>
</table>
Figure 4.3 Cartoon 1D landscape of a minimization problem.

Figure 5.1 Top: rendering of LunarLander environment, showing the agent at the beginning of an episode rollout. The goal is to safely descend and land between the yellow flags. Bottom: annotated t-SNE plot, showing the clusters of behavior that the algorithms tend to fall into.

Figure 5.2 Top: rendering of BipedalWalker environment, showing the agent at the beginning of an episode rollout. The goal is to actuate the legs so as to locomote to the end of the map on the right side. Bottom: annotated t-SNE plot, showing the clusters of behavior that the algorithms tend to fall into.

Figure 5.3 Top: rendering of HalfCheetah environment, showing the beginning of a rollout. The goal is to actuate the legs so as to locomote to the end of the map on the right side. Bottom: annotated t-SNE plot, showing the clusters of behavior that the algorithms tend to fall into. There are only two easy-to-interpret clusters: one where the agent either flips on its back and attempts to thrash forward, and one where the agent runs forward normally with varying speeds. Most PPO runs fall into Cluster 1, while SAC and TD3 policies mostly fall into Cluster 2, with some exceptions.

Figure 5.4 Breakdown of t-SNE plot by the different algorithms. We can see for this problem that different algorithms fall into very different solutions. PPO tends to fall into a local optima of flipping on its back. SAC often achieves more optimal running gaits, but occasionally falls into the same poor solution that PPO does (shown by the two red dots in (b)). TD3 policies form another distinctive set of behaviors, often achieving strong running gaits.
Figure 5.5  Top: rendering of HumanoidStandup environment, showing the agent standing at the end of a rollout. In this environment, the agent starts dropped on the ground and must learn to stand up. Right: 3D t-SNE plot generated by processing state trajectories, as described in Section 4.2. The line between to the left shows the state corresponding to the image in the t-SNE plot. Bottom: Rollout frame-by-frames showing the different rollouts visualized in the t-SNE plot. We can see that the dark blue policy gets up from a sitting position, while every other policy gets up from the stomach. This is reflected in the t-SNE plot, where the points go in different directions (left for sitting getup, up and to the right for prone getup). We also see a cycle in the light blue trajectory where it falls down and repeats part of the motion to successfully stand up.

Figure 6.1  Example custom environments. From left to right: DoubleSlit, DoubleSlitTrap, Four Corners, Four Corners Gapped.

Figure 6.2  DoubleSlit environment, along with t-SNE plot produced from a mixture of PPO, SAC, and TD3 runs, where x and y axes are produced by t-SNE and not scaled. There are two distinct clusters that correspond to the agent (1) going up, and (2) going down through the chosen slit.

Figure 6.3  DoubleSlitTrap environment, along with t-SNE plot, similar to the previous figure. Again, we see two distinct clusters that correspond to the agent (1) going up, and (2) going down through the chosen slit. However, in this case, the lower-slit solution is clearly suboptimal. TD3 and SAC mostly find the optimal solution, while PPO with these hyperparameters usually finds the suboptimal solution.
Figure 6.4  FourCorners environment, along with annotated t-SNE plot for PPO, SAC, and TD3 runs. Similar to previous figure (dots and axes are output by t-SNE and then manually annotated on top). We know that this environment has many possible modes and we see the clusters generated via t-SNE correspond well to these. We have annotated these clusters along with the direction they chose, for example SE corresponds to choosing the South-East corner (down and to the right), and local min correspond to solutions where the agent gets stuck in the inner box region and does not reach the goal.

Figure 6.5  Solutions maintained at convergence by SAC, for three different runs on standard FourCorners environment; shown are the trajectories in the environment that the agent follows. These diagrams are produced by doing 4 rollouts, slightly perturbing the initial conditions so that the agent starts slightly closer to each of the gaps, then running the policy from that point onwards. We consider a solution to be “maintained” by an agent if it chooses that solution when initialized closer to it.
Chapter 1

Introduction

Recently, deep reinforcement learning (DeepRL) has been successful at solving a wide variety of challenging control and decision making problems. DeepRL algorithms now represent the state of the art for playing complex board games such as Go [31] and Chess [30], video games e.g., Atari from pixels [22], and strategy and multi-agent cooperation games such as StarCraft II [38] and DOTA2 [2]. They are also able to learn to control simulated and real robots to locomote and manipulate objects from raw state and image observations [11, 35].

Despite recent advances, however, deep reinforcement learning algorithms remain challenging to apply. Beyond the many practical issues faced in deep learning and optimization, RL environments often have local minima and complex task landscapes that agents have to explore and optimize. Popular algorithms can vary substantially in performance from run to run, due to multiple sources of stochasticity, including neural network weight initialization, stochastic policy sampling, environment stochastic initial state, and lastly, stochastic dynamics. Such issues are well documented in past work on practical challenges in RL, on the sensitivity of results to specific features of environments [26], algorithm hyperparameters [12], and the need to average across random seeds.

The status quo for comparing and understanding algorithms is through the learning curves and rollout videos that they produce. Researchers will often run an RL algorithm several times and plot the mean and standard deviation bounds as a function of experience, i.e., environment steps. This helps reduce the effect of
spurious differences in algorithm performance, but it reduces the information we can glean from those runs to a low resolution signal, which does not account for specific solution modes (how good or bad individual runs were and what exactly the behavior was), nor the detailed effects of stochasticity (how exactly certain random seeds succeeded or failed). Figure 1.1 is an example of such a learning curve plot, showing results for two different algorithms, using 5 seeds each. We see that information is lost about the exact behavior of how the deep RL algorithms behave on the HalfCheetah environment. For example, there is no way to determine why the RL algorithm Soft Actor Critic (SAC) might produce higher variance on this task, for example. However, by manually inspecting the runs in Figure 1.2, we find that in 2 cases out of 5, SAC [10] finds a solution mode where the character flips on its back. The extra variance in the plot is a result of SAC actually finding two distinct solution modes—one of them being highly suboptimal. While the information lost from learning curves can be recovered by manually inspecting video rollouts, this requires time and effort and makes it challenging to compare many runs of an algorithm and it impedes quickly understanding the landscape. It also remains challenging to understand differences across many policies or even rollouts of a single policy for complex problems, where the state space is hard to visualize and varying initial states can lead to various state space trajectories.

To regain some of the information being lost with learning curves and individual rollout videos, we follow along a set of work that aims to better understand things and we develop several tools to compare algorithms in a richer way. These include some visualization methods for comparing and understanding the behavior of algorithms across many runs, and for understanding the behavior of individual policy rollouts. Figure 1.2 shows an example of the better understanding that can be gained from going beyond learning curves and developing these different visualization approaches. In it, we can quickly visualize the behavior of different algorithms and which modes they tend to find. Relatedly, we aim to understand the effects of stochasticity and the solution modes that RL algorithms tend to produce, beyond what learning curves and individual rollout videos can provide. We study the interplay between environments and algorithms, the sources of stochasticity, and the solution models that deep RL algorithms tend to find. We also develop custom environments with specific optimization landscapes for studying behavior
of algorithms and optimization performance quantitatively.

1.1 Research Questions

With the above context in mind, we pose several research questions, to help guide the discussion and which motivate the development of tools and experiments throughout this document. We answer some of these in depth and relegate others to future work.

RQ1. What is a meaningful way to define the similarity of a set of policies, beyond simply looking at a single number that describes their expected performance?

RQ2. What is the diversity of policies resulting from running a single algorithm with many random seeds? What tools are needed to observe the resulting diversity, aside from the default of simply observing different final performance scores?

RQ3. For a given RL problem, how can we characterize its optimization landscape in a meaningful way (qualitatively and quantitatively)?

RQ4. What solutions and local minima do well-established RL methods such as SAC, TD3, and PPO find?

RQ5. How many solution modes do common RL methods maintain in practice for problems that have highly disparate multimodal solutions?

RQ6. Of the four sources of stochasticity (weight initialization, action sampling, environment initial state, and environment dynamics), which have the most influence on the resulting diversity?

RQ7. How does the choice of hyperparameters affect the solution modes that algorithms fall into?

RQ8. How much do multiple solution modes interfere with each other when trying to learn a single policy that attempts to discover, and then maintain, the multiple solution modes?
RQ9. Reward engineering / shaping plays an important role in overcoming local minima. Once a solution has settled into a dominant mode, can we drop the shaping terms so as to purely focus on the “pure” optimization objective?

RQ10. Can the answers to the questions above inform the design of improved algorithms?

In this thesis, we focus our attention on RQ1-RQ5, and we leave RQ6-RQ10 for future work.

1.2 Thesis Overview

The remainder of this thesis is structured as follows. We provide a general background to relevant topics in Section 2. We cover related work in Section 3. We discuss the tools that we develop for studying the behavior of RL algorithms in Section 4, including two means of visualizing policies as points and as trajectories in state space as well as a quantitative approach. We use these tools to understand the optimization landscape of different environments in Section 5. We then develop custom environments with known landscapes in order to better understand the multimodal behaviors of different RL algorithms in Section 6. Lastly, we provide conclusions in Section 7.
Figure 1.1: A visualization of learning curves on HalfCheetah for Soft Actor Critic (SAC) [10] and Twin-Delayed Deep Deterministic Policy Gradient (TD3) [7], illustrating the type of information that learning curves obfuscate. Means and standard deviations bounds for five seed runs of each algorithm are shown. We see that the SAC learning curve has a wider variance and sometimes performs better, while TD3 has lower variance but never performs as well as the best SAC run, as can be see by the shaded regions. However, we don’t get any insight into why this is the case.
Figure 1.2: Visualizations of SAC and TD3 runs for HalfCheetah environment, all in the same t-SNE space, based on similarity of policies (top: SAC, bottom: TD3, left: dots colored by reward, right: rollout images overlayed in our user interface). More similar policies—according to our metric discussed in Section 4.1—tend to locate closer together. By plotting many runs on a t-SNE plot, we can much more clearly show the multiple modes that algorithms fall into. For this environment, we can see that 2 runs of SAC fall into a red region which corresponds to the agent flipping on its back. This type of plot gives much more information about the behavior of the algorithms at a high-level and goes farther in describing the variance of SAC on this task, than standard learning curves.
Chapter 2

Background

This thesis studies the behavior of deep reinforcement learning algorithms, which have a long history going back at least to Tesauro’s TD-Gammon [36], which used neural networks and self-play to train an agent to achieve master-level performance on the board game of Backgammon. The popularity and widespread application of deep reinforcement learning algorithms has been spurred by the recent successes of deep learning [17] and its use to learn function-approximated versions of traditional reinforcement learning [33] algorithms. Here we provide a brief review of deep learning and reinforcement learning theory, as well as the methods we are going to use to visualize and understand them.

2.1 Neural Networks and Deep Learning

Deep learning has emerged as a promising paradigm for developing software by providing data and experience to train a system rather than designing and manually engineering all aspects of it. Deep learning broadly involves designing three components: a neural network architecture that defines how inputs are fed and outputs produced, a training objective and a related procedure that is used to adjust the neural network weights, and a process that produces data and feeds it to the network.

Broadly speaking, deep learning can be divided into three categories: supervised learning, unsupervised learning, and reinforcement learning. Supervised
learning refers to the case where there exists an input set input $X$ and an output label or value $Y$, and we wish to optimize the network $f(X)$ to produce the corresponding label, $Y$. Unsupervised learning refers to the case where we only have samples from a dataset $X$ and we wish to learn patterns within the data or to generate new examples from the same distribution that $X$ was generated from. A related term is self-supervised learning, when we generate labels by using some structure of $X$ itself—for example, predicting future values in a sequence of $X_{t+1}$, given past values in that sequence $X_0$. Finally reinforcement learning refers to the case where an agent acts in an environment, trying to maximize reward in a sequential decision process. Reinforcement learning is the focus of this thesis, and arguably quite different from supervised learning and unsupervised learning because the learning process influences the distribution from which it learns.

2.2 Reinforcement Learning

Algorithm 1 Pseudocode for basic reinforcement learning interaction

1: $s_t \leftarrow p(s_0)$  \hspace{1cm} $\triangleright$ s = env.reset()
2: while episode not over do
3: $a_t \leftarrow \pi(s_t)$  \hspace{1cm} $\triangleright$ Query policy for action given current state
4: $s_{t+1}, r_t \leftarrow F(s_t, a_t), R(s_t, a_t, s_{t+1})$  \hspace{1cm} $\triangleright$ next_state, reward = env.step(a_t)
5: end

Reinforcement learning generally assumes the standard formalization of a Markov Decision Process (MDP) [33]. An MDP can be defined by a state space $S$, an action space $A$, a transition function $F : (S, A) \rightarrow S'$, and a reward function $R : (S, A, S') \rightarrow \mathbb{R}$, where $S'$ represents a state from the next timestep after $S$. Some environments have stochastic transitions, represented by noise in $F$. Additionally, we may also assume an initial state distribution $p(s_0)$, which defines what initial state the agent starts in. Pseudocode for this interaction is shown in Algorithm 1.

Intuitively, we assume we have an agent that chooses actions in an environment so as to maximize expected rewards over time, up to some episode horizon. In deep reinforcement learning, we assume that agent’s policy $\pi(s_t)$ is defined by a neural network, where the input is the current state, $s_t$, and the output is an action, $a_t$, to be sent to the environment at every step. To help learn the policy, we may also use
a critic network, $V(s_t)$ or $Q(s_t, a_t)$, that predicts the expected summation of future rewards from the current state onwards, and can be used to provide better gradients to train the policy. Based on feedback from the reward signal of the environment, the policy network is progressively optimized to produce action sequences that achieve high reward in the environment.

2.2.1 Sources of Stochasticity

There are several sources of stochasticity in RL that can lead to varying outcomes, even when running the same algorithm. These are:

1. Neural network weight initialization. Neural network weights are randomly initialized to small values in order to break symmetry and allow convergence. Many papers have studied the importance of this in enabling convergence [18], but the effects in DeepRL are relatively understudied.

2. Stochastic policy. To ensure exploration, most algorithms will inject noise into the actions that the policy $\pi$ outputs, either through Gaussian noise or some $\epsilon$-greedy approach [33].

3. Initial state distribution $p(s_0)$. To provide extra challenge to the task or robustness to trained policies, environments are often designed with some randomization in the initial state.

4. Environment transition dynamics $F : (S, A) \rightarrow S'$. Similarly, environment dynamics can be random due to explicit design decisions or uncertainty in the underlying state.

The combined effect from all of these sources can lead to vast differences in outcome even for the same algorithm. Because of these well-known large effects, it is common practice to run deep reinforcement learning algorithms several times to obtain an estimate of mean performance, which can then be used for comparison with other algorithms. This is a useful practice, but is still highly imperfect approach, which motivates our search for better ways to understand the performance and behavior of DeepRL algorithms.
2.2.2 Challenges of RL

Compared to supervised learning and unsupervised learning, reinforcement learning (RL) is more general and faces additional challenges. RL does not assume a fixed and curated dataset\(^1\). The agent must interact with the environment and learn from those interactions. This requires balancing the exploration-exploitation trade-off, ensuring that interesting and useful data is chosen to learn from, and avoiding spurious collapses and instabilities in the policy. Additionally, the agent is only provided with possibly-delayed feedback about the longer term consequences of every decision. It must deal with the credit assignment problem, learning how past actions played a role in future rewards. Understanding and developing robust deep reinforcement learning algorithms remains a large area of open research.

2.2.3 DeepRL Algorithms

There are many possible approaches to solving RL problems with deep learning, but a few algorithms have become well-established over the last several years. All of these are so called actor critic methods, which rely on both a policy \(\pi(s_t)\) to produce actions and a value function \(V(s_t)\) or \(Q(s_t, a_t)\) to estimate future return.

*Proximal Policy Optimization (PPO).* PPO is a popular policy gradient algorithm [28], used to solve a variety of tasks including video games and robotic control. PPO works by learning networks that approximate a value function and an optimal policy. It uses the value network in order to compute an Advantage function that indicates whether an action led to a return that was above or below average. It then computes the gradient of the policy network and modulates the gradients by the Advantage function, \(\nabla a [A(s, a) \log \pi_a(s)]\) and takes a gradient ascent step. This leads to above average action choices being made more probable while below average action choices are made less probable. PPO also uniquely includes a clipping term in its loss function, in order to ensure suitable step sizes for the policy updates.

\(^1\)With the exception of batch/offline RL, where the dataset must originate from some interactions, but is assumed given.
**Twin Delayed DDPG (TD3)**. TD3 is a Deep Deterministic Policy Gradient [20] (DDPG) style algorithm [7]. It estimates a state-action value network $Q(s,a)$ using Bellman updates [33], and then propagates gradients back through that network to update the policy. It computes gradients $\nabla_\theta Q(s, \pi_\theta(s))$, and uses those to do a gradient update on the policy network. Beyond DDPG, TD3 adds a few tricks to stabilize the algorithm and make it work better with function approximators, including maintaining two value networks and taking the minimum of the two to reduce maximization bias [33], using delayed value networks to improve stability, and adding noise to the target action to smooth changes in $Q$.

**Soft Actor Critic (SAC)**. SAC is a similar algorithm to TD3 in that it builds off DDPG, however SAC also features entropy regularization [10]. The policy is optimized not just to maximize the Q-function, but also to maximize entropy in the policy output distribution. This helps incentivize greater exploration.

### 2.3 t-distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE [37] is a method for reducing high-dimensional spaces to a 2 or 3 dimensional spaces so that they can readily visualized. It is designed so that nearby points in the high-dimensional space are nearby in the latent space. However, the converse is not necessarily true, i.e., far away points in high dimensions are not guaranteed to be far away in low dimensions. t-SNE works by constructing a pairwise probability distribution over points in the original high dimensional space such that points that are close in the space yield high probability and points that are far away yield low probability. It then randomly initializes these points in a lower dimensional space and constructs a similar probability distribution in the low dimensional space. It finally minimizes the KL divergence between these two distributions, optimizing with respect to the coordinates of the points in the low-dimensional space. Compared to the original stochastic neighbor embedding paper [14], t-SNE uses the Student’s t-distribution, instead of a standard Normal distribution.
Chapter 3

Related Work

This thesis builds off a broad body of research in optimization, deep learning, deep RL, and visualization. In this section, we describe the relation between our work and several related categories of work.

3.1 Understanding and Visualizing Optimization Landscapes

A useful way to think about optimization problems is through the high-dimensional landscapes they define, with features like peaks, valleys, plateaus, basins, and ridges. Illustrative landscapes in 1D and 2D domains are in Figure 3.1 and Figure 3.2, respectively. These landscapes, often encountered in deep learning and deep RL, are often too difficult to visualize and understand directly. However, several works develop indirect means of studying and visualizing them [8, 15, 19]. These approaches generally use low-dimensional slices and approximate local estimates to determine the landscape of the neural network loss or motion optimization objective.

In this work, we face the similar challenge of understanding a high dimensional and complex optimization problem in an interpretable way. We consider the entire interaction between the environments, i.e., the optimization problems to be solved, RL algorithms, and the stochasticity involved. Rather than simply observing the peaks and values of the neural network optimization, we seek to explore the specific
behaviors that agents produce across different runs, and to understand how the algorithms behave in settings with multimodal solutions.

Figure 3.1: 1D illustration of the type of optimization landscape discussed in this work, with maxima at $x = -1.5, -0.2, 0.9$ and minima at $x = -1.1, 0.4$, and the edges of the plot.

3.2 Empirical Studies and Critiques of RL

Deep RL faces several additional challenges beyond standard deep supervised learning from fixed datasets. Deep RL algorithms are known to be sensitive to subtle differences in implementation and hyperparameters choices [12]. There have been several works discussing the challenges of RL [16], as well as limitations and complexities that arise [26]. Our work seeks to develop a better understanding of environments and algorithms, one that goes beyond learning curves. We examine the behavior of RL algorithms and study the solutions they find across multiple runs.

3.3 Policy Similarity Metrics

To better understand the differences between algorithms, it helps to have a quantitative metric to compare the behavior of different policies. Broadly, there are two
Figure 3.2: Visualization of an arbitrary landscape that might be encountered during optimization [3] (permission pending).

common ways of comparing policies: (1) via their state visitation and (2) via the action distribution that the learned policies produce at different states.

**State visitation similarity**

State visitation methods face the constraint that different policies generally will not enter the same states at the same time along a trajectory. To cope with this, they must either do some form of alignment like Dynamic Time Warping (DTW) [24] or a distillation approach, as in [40]. In this latter work, they learn autoencoders, $D_{\pi}(s)$, that take in and try to reconstruct the state. They can then measure the difference in state visitation between two policies by measuring the error between the autoencoders and their reconstructions; for policies that have not visited the same states, they will have larger errors for the mismatched states. This can be used to find the novelty or pairwise similarity between policies.
Figure 3.3: Visualization of an optimization landscape of a ResNet with and without skip connections [19] (permission pending). The x and y axes represent a slice of the decision variables while the z axis represents the value of the loss being optimized.

Policy action distribution similarity

Recent policy distribution methods include Agrawal et al. [1] and Sun et al. [32]. Agrawal et al. [1] use a probability pseudometric between policies which accounts for the similarity of how policies behave both in the short term and long term. They in turn use this to form an embedding space to help learn generalizable policies. Sun et al. [32] use a Wasserstein metric to compare the distribution of policies. They use this to seek out novelty to cover more and learn more about the state space.

As compared to prior work, which mostly have the objective of defining and optimizing some novelty or diversity metric [6, 29], our sole aim is to compare the behavior of the algorithms. We also face the additional constraint of dealing with different types of algorithms. For example PPO and SAC use very different methods of representing variance around a mean for continuous policies. This leads to our choice of using a policy distribution similarity, similar to Sun et al. [32], but with fixed variance instead of the full Wasserstein metric.
3.4 Visualizing Trained Policy Behavior

A large aspect of understanding RL algorithms is understanding the behaviors they produce. Several papers focus on gaining a deeper understanding of the behavior and properties of trained agents. Past work has done this by either creating visual saliency maps and seeing what the vision network pays attention to [9, 13, 27], as illustrated in Figure 3.5 and Figure 3.6, or through dimensionality reduction approaches to explore the different behaviors that the agent exhibits [21, 39] (see Figure 3.7). Our work falls into the latter category. Our Figure 4.2 is in some ways similar to Figure 3.7 from [39]. In contrast to prior work, we study the potentially multimodal landscape of many runs and algorithms together. We develop custom environments and provide multiple ways to interactively explore algorithm and the resulting policy behaviors.
3.5 Quality-Diversity Optimization Algorithms

Quality-diversity is a body of literature around developing optimization algorithms that both optimize traditional objectives, but also incentivize for greater diversity or novelty. The intuition is drawn from biology, where many agents exist in different niches that they are optimizing within. By having a large population of agents with strong diversity, these algorithms are highly adaptive and discover solutions that traditional algorithms do not [4, 5, 23].
**Figure 3.7:** Zahavy et al. [39] construct t-SNE plots on Atari games in a fashion related to ours in Figure 4.2. Each point represents a specific game state. (permission pending)
Chapter 4

Methods

In order to better understand the behavior of algorithms and the solution modes they tend to produce, we develop a set of tools for studying them. These include visualizing policies as points in a low-dimensional space (Section 4.1), visualizing states along policy rollouts as points in a low-dimensional space (Section 4.2), and by developing custom environments and optimization landscapes and quantitatively measuring the performance of algorithms on these environments (Section 4.3). This set of tools enables us to study RL beyond learning curves, and to start to answer the research questions we have posed.

4.1 Visualizing Policies as Points

When comparing algorithms, we would like to be able to view the high-level picture of how different algorithms behave and what solution modes they fall into across many runs so that we can better understand general patterns. Research question RQ1 specifically motivate this and the solutions are relevant for studying RQ4, and RQ7.

Thus, our first means of understanding RL algorithms is by embedding policies as points in a low-dimensional space, shown in Figure 4.1. We develop a metric for comparing different policies and we use this metric to create a t-SNE plot such that similar policies are nearby in space. These can be used to compare many policies and the solution modes that algorithms may favor. It allows for the identification
Figure 4.1: Top: example task where a point mass starts in the center and must move to exit one of the four gates and reach one of the four corner goals, shown as red lines. Bottom: visualization we develop where policies are embedded in a low-dimensional space, where each point represents a single run of an algorithm and the policy is produces, along with the rewards it achieves. Demonstration video: https://youtu.be/LyQn7p32cbg. Red colors correspond to low normalized reward. Green colors correspond to high normalized reward.

We employ a symmetric KL divergence metric on policy action distributions, as
Figure 4.2: A visualization we develop where policies are viewed as state space trajectories. Each point in the space represents the state that a policy rollout visits for the HumanoidStandup environment. Colors correspond to unique policies. Demonstration video: https://youtu.be/J8nlUj6BD6c.

computed pairwise between all policies. To account for differences in how the different RL algorithms parameterize policy noise, we opt for a fixed variance Normal distribution ($N(\pi(s), 0.1)$) to compute log probabilities. The metric $D_\pi$ is given by the following equations. The KL divergence of policy $A$ from is $B$ is given by:

$$KL(A||B) = \sum_{s \sim S_A} E_{\pi_a}[\log(\pi_a) - \log(\pi_b)]$$

Since the KL divergence is asymmetric, we also compute it the other way:

$$KL(B||A) = \sum_{s \sim S_B} E_{\pi_b}[\log(\pi_b) - \log(\pi_a)]$$

And finally these definitions can be combined to define the symmetric KL di-
vergence as follows:

\[ D_\pi(A, B) = KL(A||B)^2 + KL(B||A)^2 \]

\( D_\pi(A, B) \) is computed over the distribution of states that are induced by each policy in the rollout (\( s_A \) is the state distribution induced by \( \pi_a \), and \( s_B \) is the state distribution induced by \( \pi_b \)), and it is symmetric, so \( D_\pi(A, B) = D_\pi(B, A) \). Given \( D_\pi(A, B) \) for each pair of policies, we use t-SNE to embed the points in a 2D or 3D grid. Depending on the task, we use up to 90 runs of the algorithm. We plot this interactively using D3.js to enable us to explore different properties of the clusters and filter by these properties, such as by algorithm and hyperparameter.

The potential limitations of this KL divergence approach are that agents may vary in what initial paths they take. For example if Agent A goes North, Agent B goes South initially, but otherwise they mostly would behave similarly given the state they are in. It would take some further study to determine the effects of this, and it may be more sensitive on some environment than others.

### 4.2 Visualizing Policies as State Space Trajectories

If we want to understand the fine differences in motion behavior between two policies, this can be challenging if the state space is high-dimensional, like in the case of the HumanoidStandup environment. Research question RQ1 and RQ2 help motivate this, and the solution will be useful for studying RQ3, RQ4, and RQ5.

Thus, our next means of understanding the solution modes is visualizing policies as state space trajectories, shown in Figure 4.2. We run the deterministic trained policies in an environment and we collect the rollouts, i.e., state sequences, that they produce. We then compute a 3D t-SNE mapping where each state is a point in the space, and the trajectories are distinguished by color. These can be used to compare a handful of rollouts, to obtain fine-grained insights into the behaviors that policies produce. We can visually identify common bottleneck regions that may be shared by all policies, for example. This is especially useful for environments with high-dimensional state spaces, such as Humanoid.

More formally, to compute the t-SNE mapping, given some set of rollouts \((r_1, r_2, \ldots, r_N)\), consisting of sequences of states \((s_1, s_2, \ldots, s_T)\), we feed all \( N \times T \)
states into t-SNE. In our studies we use the states of the many joints of the humanoid robot. For handling even higher dimensional spaces such as images, we could compute neural network features based on a pre-trained image feature extractor.

Once we have our states, each of them correspond to a different in our t-SNE space. In this case, the comparisons are computed by t-SNE itself [37], using the formula:

\[
p_{ji} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2 \sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2 \sigma_i^2}\right)}
\]

The variance \(\sigma_i\) is computed used a binary search to produce a user specified perplexity \(Perp(P_i) = 2^{-\sum_j p_{ji} \log_n p_{ji}}\). These \(O(n^2)\) comparisons are then used to compute a low dimensional embedding. The embedding dimension is a chosen parameter, usually set to 2D or 3D for easy visualization. Further details are described in Section 2.3, or in the t-SNE paper itself [37].

NOTE: we always use the Policies as Points t-SNE plots in this work, except for on the humanoid getup task.

### 4.3 Custom Environments and Quantitive Metrics

Finally, environments may have complex and unknown optimization landscapes, making it hard to isolate properties and behaviors of different RL algorithms. To isolate these characteristics, another tool for understanding the behavior of algorithms is by constructing environments that induce specific optimization landscapes and solution modes. By controlling these aspects, we can also quantitatively compare different algorithms by looking at the solution modes they find.

For example, different environments may induce different breadths and depths of local policy optima that the agent or RL algorithm must navigate. We provide a 1D illustration of this in Figure 4.3. Understanding the behavior of RL algorithms in response to multimodal optimization landscapes remains and understudied area, despite the fundamental nature of explore-vs-exploit in RL problems. We construct our FourCornersWeighted environment to induce different breadths of gaps that an agent can pass through to reach a goal, and we can collect statistics about the frequency with which different algorithms find the different goals. This is relevant to
research question RQ5 and potentially RQ8. We cover experiments on our custom environments in Section 6.
Chapter 5

Visualizing the Solution Modes Learned by RL

We now apply the methods of the previous chapter to standard RL tasks and using standard RL algorithms. Of relevance to RL practitioners is the optimization landscape of the environments they are studying. We demonstrate the visualization of policies as points in order to answer RQ1 and RQ2. We also investigate RQ3 and RQ4, which can be restated as: If algorithm A performs better than algorithm B, on average, is this because of different modes being discovered with different frequencies? Are there many local minima that an algorithm will fall into? Are there adjustments that can be made to make the environment more tractable and make learning more efficient?

For our algorithms, we use Stable Baselines 3 [25] default implementations of PPO [28], TD3 [7], and SAC [10]. We chose these algorithms because they are fairly popular in the literature and they have standard implementations available online. The stopping criteria for training these was 500k environment interaction steps.

5.1 Visualizing Polices as Points

We study the solution modes that PPO, TD3, and SAC tend to fall in to, across many runs on standard OpenAI Gym tasks. This can give us a better understanding
of the environments, and in particular, the natural optimization landscapes that they have and how modern RL algorithms tend to interact with them in practice.

5.1.1 LunarLander

The LunarLander (LunarLanderContinuous-v2) environment (Figure 5.1) is based on the Atari game, where the goal is to safely land a propulsion lander on the surface of the Moon. The observation space consists of 8 values, including the XY position, XY velocity, angle, angular velocity, and leg ground contacts. The action space consists of 2 values, the main throttle and the left/right engine throttle. Reward is computed as a squared distance to the goal, along with some shaping terms for maintaining contact with the ground.

On this environment, all algorithms generally produce successful policies, as seen in Figure 5.1. These points consist of a mixture of PPO, SAC, and TD3 runs, and we did not observe any significant difference between their behaviors, either visually or in terms of the t-SNE visualization. The policies generally form two clusters: near the top, the best solutions waste minimal fuel. They quickly drop down and only use the fuel required to not crash. Cluster 2 in the diagram represent less efficient solutions that make more frequent use of the left-right boosters, or use a slower descent velocity.

5.1.2 BipedalWalker

The BipedalWalker (BipedalWalker-v3) environment (Figure 5.2) is a custom environment authored by OpenAI where the goal is to choose actions for a simulated box2D physics-based character to locomote as quickly as possible across the ground to the right. The observation space consists of 24 values, including angle and velocity of all 5 links, ground contacts, as well as a simple lidar-like measurement. The action space consists of 4 values, 1 for each joint of the robot, including the hips and knees for the two legs. Reward is given for achieving positive progress in the x-direction, along with some shaping terms for keeping the body level, and some energy minimization terms.

Figure 5.2 shows the solution points consist of a mixture of PPO, and SAC runs. We can see they fall into five distinct clusters. SAC performed better during
our runs, leading to Clusters 1, 2, and 3. Each of these clusters represent good solutions, but vary in their gaits, with Clusters 2 and 3 having different legs in front, and Cluster 1 having an alternating leg running gait. SAC is able to find better solutions (Clusters 1, 2, 3), while the Stable Baselines implementation of PPO remains slightly suboptimal (Clusters 4, 5). The clusters are formed based on the gait that the character uses. Some policies keep a single leg in front and are further clustered based on the choice of leading leg. Other gaits use more symmetric alternating left-right steps. These all can be seen in the graph, with more optimal and less optimal solution variants.

5.1.3 HalfCheetah

The HalfCheetah (HalfCheetah-v3) environment (Figure 5.3) is another 2D simulated locomotion task, which uses the Mujoco simulator. The goal is to choose joint actuations to locomote a Cheetah-like character to the right. The observation space consists of 17 values, including the position and velocity of the links and joints of the character. The action space consists of the 6 values, which are the activations of each of the 6 joints (3 per each leg, consisting of hip, knee, foot). Reward is computed as forward velocity minus a control cost for joint activations.

For this environment, shown in Figure 5.3, PPO consistently finds a local optima solution where the Cheetah flips on its back (Cluster 1), or where it has a jerky walking motion. SAC sometimes also finds local optima, but also finds fast gait motions. TD3 also finds fast gait motions, which vary slightly from each other and from SAC.

Along with the annotated plot, we also show the breakdown per algorithm in Figure 5.4. By filtering the points by algorithm, we can visualize clear trends between the different algorithms and the default hyperparameter settings we used.

By examining our policies behaviors as both points in a shared space and by their state space trajectories in t-SNE plots, we are able to learn more about the optimization landscape and solution modes of the environments. For LunarLander (Figure 5.1), we find that there are some local minimas that involve using more gas then is necessary in order to ensure a safe landing. For BipedalWalker (Figure 5.2), we find that there are various local optima for safe but slow locomotion policies;
we are also able to see the different solution modes that correspond to using the front leg first, back leg first, or alternating legs. For HalfCheetah (Figure 5.3), we find the alluring mode of the cheetah flipping on its back. In response to RQ2 and RQ4, we find that different algorithms tend to produce distinctive solutions, and for any given algorithm, the learned solutions can be quite diverse, i.e., not normally distributed for performance.

5.2 HumanoidStandup

The HumanoidStandup environment (Figure 5.5) is a 3D simulated humanoid character task, where the agent is dropped on the ground and it must learn how to stand up and stay balanced. The observation space consists of 68 values, including measurements of positions and velocities of all the many joins. The action space consists of 21 values, for the activations of the joints. Reward is given for raising the body to a certain height. To obtain more realistic motions, the policies for this environment were generated using a curriculum based approach, where the torque limits were gradually reduced to produce more realistic motions.

3D humanoid environments such as this are very high-dimensional and complex, and understanding the learned policy performance can be challenging. To further explore this, we compare and contrast learned policies by visualizing their induced trajectories, as seen in Figure 5.5. We see that many rollouts tend to get up from the stomach and enter a similar bottleneck state, while one of the trajectories ends up going in a different direction. We can also view a cycle that happens in the light blue trajectory, where it attempts to stand up, falls, and then successfully stands up.

By examining various trained policies and visualizing their trajectories on a 3D t-SNE plot, we are able to observe the bottleneck states in this environment and the different solution modes that algorithms tend to find. We can see that similar behaviors, like pushing up from the stomach followed similar strategies, while standing up from a sitting position went in a different direction. In response to RQ5, we find that certain environment and algorithm combinations yield many solution modes, but these can be effectively categorized based on bottleneck states.
Figure 5.1: Top: rendering of LunarLander environment, showing the agent at the beginning of an episode rollout. The goal is to safely descend and land between the yellow flags. Bottom: annotated t-SNE plot, showing the clusters of behavior that the algorithms tend to fall into.
Figure 5.2: Top: rendering of BipedalWalker environment, showing the agent at the beginning of an episode rollout. The goal is to actuate the legs so as to locomote to the end of the map on the right side. Bottom: annotated t-SNE plot, showing the clusters of behavior that the algorithms tend to fall into.
Figure 5.3: Top: rendering of HalfCheetah environment, showing the beginning of a rollout. The goal is to actuate the legs so as to locomote to the end of the map on the right side. Bottom: annotated t-SNE plot, showing the clusters of behavior that the algorithms tend to fall into. There are only two easy-to-interpret clusters: one where the agent either flips on its back and attempts to thrash forward, and one where the agent runs forward normally with varying speeds. Most PPO runs fall into Cluster 1, while SAC and TD3 policies mostly fall into Cluster 2, with some exceptions.
Figure 5.4: Breakdown of t-SNE plot by the different algorithms. We can see for this problem that different algorithms fall into very different solutions. PPO tends to fall into a local optima of flipping on its back. SAC often achieves more optimal running gaits, but occasionally falls into the same poor solution that PPO does (shown by the two red dots in (b)). TD3 policies form another distinctive set of behaviors, often achieving strong running gaits.
Figure 5.5: Top: rendering of HumanoidStandup environment, showing the agent standing at the end of a rollout. In this environment, the agent starts dropped on the ground and must learn to stand up. Right: 3D t-SNE plot generated by processing state trajectories, as described in Section 4.2. The line between to the left shows the state corresponding to the image in the t-SNE plot. Bottom: Rollout frame-by-frames showing the different rollouts visualized in the t-SNE plot. We can see that the dark blue policy gets up from a sitting position, while every other policy gets up from the stomach. This is reflected in the t-SNE plot, where the points go in different directions (left for sitting getup, up and to the right for prone getup). We also see a cycle in the light blue trajectory where it falls down and repeats part of the motion to successfully stand up.
Chapter 6

Understanding How RL Algorithms Behave for Tasks with Multiple Solution Modes

Standard benchmark environments such as those used in Section 5 can be used to understand and compare algorithms. However, their optimization landscapes are unknown and potentially complex, so it is difficult to know a priori when a learned policy is optimal or when it may be stuck in a local optima. To provide an easier means of studying differences and how they perform for tasks having multimodal solutions, we develop toy environments with specific optimization landscape properties.

6.1 Custom Environments

We consider a set of 2D environments with a 2D point mass agent having second order dynamics, using the Box2D simulator. The point mass is actuated by a force applied in the X and Y directions. The initial states are deterministic, as are the goals which are marked by red X’s or lines. Reward is computed as the L2 distance between the current XY coordinates and the goal XY coordinates: \( R = ||s - g||_2 \). Various initial states, goals, and walls lead to different behaviors. The dynamics and initial state are deterministic.
6.1.1 DoubleSlit

The agent starts on the left-center of the map and there are two slits the agent can pass through to reach a goal on the right (Figure 6.2). For this and following environments, we use a mixture of PPO, SAC, and TD3 runs. As expected, the algorithm run on DoubleSlit produce two solution modes. In the t-SNE plot of Figure 6.2, these are split left and right diagram, with a majority choosing down. We would expect each solution to be chosen 50/50; we are unsure what leads to the discrepancy, but it could be due to some slight offset in the environment making the down solution ever-so-slightly more optimal. The variance within the left and right clusters are due to minor differences in how the agent navigates to the slits and goal.

6.1.2 DoubleSlitTrap

Like DoubleSlit, but the agent starts closer to the lower slit, and this slit leads to a local minima (Figure 6.3). The local maxima for passing through the lower slit has an approximate cumulative reward of -31, while the global maxima for passing through the upper slit and terminating by reaching the goal has an approximate cumulative reward of -21. In practice, we note that well-behaving algorithms eventually stumble upon the globally optimal solution (moving up) through their exploration. In our experiments, visualized in the t-SNE plot of Figure 6.3, we see that PPO mostly find a local minima by going through the lower slit, while SAC and TD3 tend to find the optimal solution.
Figure 6.2: DoubleSlit environment, along with t-SNE plot produced from a mixture of PPO, SAC, and TD3 runs, where x and y axes are produced by t-SNE and not scaled. There are two distinct clusters that correspond to the agent (1) going up, and (2) going down through the chosen slit.

6.1.3 FourCorners

The agent begins in the middle (Figure 6.4). There are equally valid 4 goals: one in each corner. The agent must exit the inner square by passing through one of the slits. There are thus 8 global solution modes (depending on which gate and corner the agent chooses) and 4 local minima (going into the corners of the inner square). Because the environment is deterministic and the agent always starts in the exact
Figure 6.3: DoubleSlitTrap environment, along with t-SNE plot, similar to the previous figure. Again, we see two distinct clusters that correspond to the agent (1) going up, and (2) going down through the chosen slit. However, in this case, the lower-slit solution is clearly suboptimal. TD3 and SAC mostly find the optimal solution, while PPO with these hyperparameters usually finds the suboptimal solution.

In the middle, we expect well-behaving algorithms to find each of the 8 global solution modes in equal proportion, but they may also get stuck in one of the easier-to-find local minima and never escape. In the t-SNE plot of Figure 6.4, the red solution mode dots are the various local minima and the green solution mode dots are the various global minima, where the agent choose from the various 8 possibilities (NW, NE, EN, ES, SW, SE, WN, WS).
Figure 6.4: FourCorners environment, along with annotated t-SNE plot for PPO, SAC, and TD3 runs. Similar to previous figure (dots and axes are output by t-SNE and then manually annotated on top). We know that this environment has many possible modes and we see the clusters generated via t-SNE correspond well to these. We have annotated these clusters along with the direction they chose, for example SE corresponds to choosing the South-East corner (down and to the right), and local min correspond to solutions where the agent gets stuck in the inner box region and does not reach the goal.

6.2 FourCorners Alternatives

In FourCorners, each of the global solutions are equally “discoverable” and optimal. In this section, we study the effects of modifying the discoverability and optimality of the solutions to determine the effects on the algorithm behavior over many runs.
FourCornersGapped uses slits that get progressively wider going clockwise around. This provides a way to study this problem of discoverability or breadth of local minima. A default hypothesis would be that the discoverability of a solution should directly correlate with how often the algorithm converges on that solution.

FourCornersWeighted uses equally sized slits, but the slits provide progressively more optimal reward values going clockwise around. If the agent exits out the North slit, it gets a reward of 10 when it reaches any corner. If the agent exits out the East slit, it gets a reward of 15; South is 20, and West is 30. A reasonable base hypothesis would be that the optimality of a solution should directly correlate with how often the algorithm converges on that solution.

For both these environments, we run between 30 and 35 runs with TD3, and measure the results of which solution the agent converged to. Ignoring solutions which converged to the locally optimal solutions, we display the results in Table 6.1. The results, while somewhat lacking in sample size, point to confirming the default hypotheses. It seems that both optimality and discoverability are strongly correlated with the probability that the algorithm converges to that solution.

### Multimodal solutions

For FourCorners, we also investigate whether different solution modes stay active during training of the policy. To test this, we add slight perturbation to initial states and view the slit that the agent follows starting from that position. If the policy is strongly unimodal, the agents will converge to a single slit. If not, the slight perturbation will lead to different gaps being chosen. Results are shown in
Figure 6.5, where we see modes are often limited to 1 or 2 candidates. No policies we observed kept all 4 modes open at a time, even though they had equal reward values.

(a) Unimodal solution, always going SW. (b) Bimodal solution, either going WS or NE. (c) Bimodal solution, either going NW or ES.

Figure 6.5: Solutions maintained at convergence by SAC, for three different runs on standard FourCorners environment; shown are the trajectories in the environment that the agent follows. These diagrams are produced by doing 4 rollouts, slightly perturbing the initial conditions so that the agent starts slightly closer to each of the gaps, then running the policy from that point onwards. We consider a solution to be “maintained” by an agent if it chooses that solution when initialized closer to it.

6.4 Discussion

In this section, we introduced our custom environments and use them to study the behavior of RL algorithms in a more controlled setting. The various point mass environments create simple, known optimization landscapes that let us study algorithms. We find that the algorithms are quite sensitive to the “breadth” of solution modes, and as expected tend to more easily fall into the modes that are easier to randomly find and explore. We also find that varying the discoverability (breadth) of the solution in this case has a similar effect to increasing the reward (depth) of a solution. Both end up increasing the number of runs that find that solution in the expected proportions. We also find that algorithms tend to maintain only 1-2 solution modes at a time, and do not consider other possibilities when placed slightly closer to those gaps.
Chapter 7

Conclusion

In this work, we study deep reinforcement learning environments and algorithms, to better understand their optimization landscapes, solution modes, and the effects of stochasticity. We motivate the need to go beyond standard learning curve analysis and we pose a set of specific research questions to answer. In thinking about and answering these questions, we develop a set of interactive visualizations and custom environments to better explore the behavior of these systems, including t-SNE visualizations and simple 2D pointmass goal-reaching tasks.

Specifically, in response to the questions we posed in the introduction:

1. We show that agents can be meaningfully compared and clustered based on relatively simple comparisons of their state and action distributions (Sections 4, 5, 6). This helps guide further analysis of the other questions.

2. Both SAC and TD3 produced very diverse agent behaviors, which is not obvious from learning curves, but which becomes clear by visualizing t-SNE plots (Sections 4, 5). An example of this is provided in Section 1, where t-SNE plots enable us to see the instances where SAC sometimes produces a HalfCheetah agent that flips on its back.

3. We looked at several ways to characterize optimization landscapes of RL algorithms, including using both policies as points and policies as state space trajectories (Sections 4.1, 4.2). The different approaches are suitable to different problems. For simple environments, policies as points was sufficient,
but for complex state space environments like HumanoidStandup, it was useful to visualize the state space trajectories in more detail.

4. In our investigations, we found that both TD3 and SAC consistently produced high-performing behaviors that cover a variety of modes in the environments (Sections 5, 6). Their results appear distinct of some of the t-SNE plots, but visually, the behaviors are often hard to distinguish. While they sometimes also converged to locally optimal solutions, they did this far less frequently than the implementation of PPO that we used. Even with learning rate tuning, it was challenging to consistently produce good behavior with PPO.

5. We found that on the FourCorners environment, SAC tends to maintain at most 1-2 solution modes out of the 8 possible (Section 6.3). This suggests an unnecessary narrowing of the solution space; this would make it challenging for the agent to adapt if the environment became non-stationary and possible solutions became unavailable. Further study would be necessary to see if this applies more broadly.

6. We ran preliminary tests to isolate the impact of different sources of stochasticity. The random seed of the weight initialization did not seem to play a large role. Evaluating the individual roles of the other sources of stochasticity, i.e., the initial state, action sampling, and environment dynamics remains an area for future work.

We answer several of the questions that we sought to answer in the introduction, along with related ones that came up during investigation. We see it as exciting and valuable future work to better further understand these algorithms and expand the scope where they can be easily applied.
Bibliography


