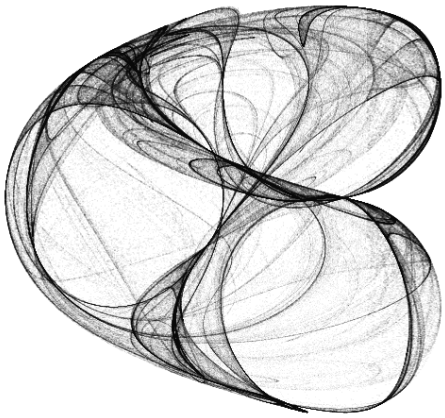
TOWARDS INTERPRETABLE REINFORCEMENT LEARNING
INTERACTIVE VISUALIZATIONS TO INCREASE INSIGHT

SHUBY V. DESHPANDE



CMU-RI-TR-20-61

Submitted in partial fulfillment of the requirements for the degree of Master of Science (Research) in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, United States of America

December 2020

COMMITTEE:
Jeff Schneider, *Chair*
Deepak Pathak,
David Held,
Benjamin Eysenbach

Aedificatio latere montis.

ABSTRACT

Visualization tools for supervised learning (SL) allow users to interpret, introspect, and gain an intuition for the successes and failures of their models. While reinforcement learning (RL) practitioners ask many of the same questions while debugging agent policies, existing tools are not a good fit for the RL setting as these tools address challenges typically found in the SL regime. Whereas SL involves a static dataset, RL often entails collecting new data in challenging environments with partial observability, stochasticity, and non-stationary data distributions. These unique characteristic of the RL framework necessitate the creation of alternate visual interfaces to help us better understand agent policies trained using RL. In this work, we design and implement an interactive visualization tool for debugging and interpreting RL. Our system identifies and addresses important aspects missing from existing tools and we provide an example workflow of how this system could be used, along with ideas for future extensions. We explain one such extension under development to increase insight into the learning dynamics of actor-critic learning algorithms by visualizing optimization landscapes.

ACKNOWLEDGMENTS

Life is a nonholonomic[1] system. Thus, it'd be impossible for me to actually express gratitude to all the individuals who've shaped me to become who I am today. I'll try to recount some experiences anyway.

Yogesh and Ishwar Sir: Thank you for providing us with a peaceful sanctuary within which to learn and grow. Your willingness to spend countless hours with us discussing the marvels of our universe, the behind the scenes tour of the GMRT (Antarctica!), and the candor of all our conversations are amongst the most memorable times of my research journey. Special thanks to Anna for the incredible food. I haven't found meals (and coffee!) as delicious and affordable anywhere else yet.

David: Thank you for believing in and taking a chance on a naive starry-eyed undergrad coming from an unheard-of institution halfway around the world. That summer at JPL/Caltech never really ended in my heart; memories of it continue to serve as a constant inspiration to "Dare Mighty Things".

Venkat: Thank you for teaching me how to question the results of experiments and teaching me the value of pushing through to take research to "completion". Memories of MSR and Lavelle Road continue to bring back a warm flood of memories. Flying back and forth 24 times in 6 months was intense, but totally worth it.

The entire team at Scale: Thank you for showing me the magic of relentless dedication and having an internal locus of control. Special thank you to Chiao for teaching me (by example) the process of rigorous thinking and the virtues of "getting in the weeds". The honest feedback you provided was bitter medicine, but I needed it.

Cyrus: I'm really grateful that our paths crossed that day at RoboMovie Night (just checked; it was Robot & Frank!). Amigos por Siempre.

The two kind souls I met on a beach in Loíza: Thank you for reminding me of all that is magical about research and academia.

Jeff: Thank you for providing me with the freedom to pursue what seemed at the time a pretty crazy idea. The chat we had at NeurIPS convinced me to return from my mini "sabbatical", and work on turning these ideas into a thesis. It has truly been an honor to learn and grow under your mentorship.

Ben: Thank you for bringing rigor to all the crazy ideas we've engaged with. Every single one of our chats has been nothing short of memorable and has consistently generated a fountain of ideas taking more time than the actual meeting to reflect on.

Archit, *Rathin*, *Shadab*, *Sushil*: Expressing gratitude for what we share might take up an entire thesis, so I won't attempt to start (yet anyway). I still remember the first time I met each of you. We've traveled a long ways since then, but in many ways I think we're all still the same: naive, foolish, and hungry to learn. All of you are a testament to the limitless potential everyone has latent within them, that can blossom regardless of where and when you've been instantiated, if you only dare to believe and fight.

Aai, *Baba*, *Po*: One of my favorite quotes[2] is: "It's the sides of the mountain which sustain life, not the top.". I now realize this quote is not quite accurate. The sides of a mountain wouldn't exist without its base. Thanks for everything.

1 A nonholonomic system is a system whose state depends on the path taken to achieve it. In some sense, it's the conceptual opposite of a Markovian process.

2 From Zen and Art of Motorcycle Maintenance by Robert Pirsig.

CONTENTS

LIST OF FIGURES

1

INTRODUCTION

1.1 REINFORCEMENT LEARNING (RL)

How might we train a dog to fetch a ball? One way perhaps is to somehow teach the dog human language, explain the known laws of physics, and then tell it to fetch the ball. Alternatively, we could show the dog many valid sets of actions, and ask it to replicate the same actions. This example might seem contrived, but it does highlight the fundamental difference between two different approaches to pursuing AI. As such, this latter paradigm of "learning by example" has resulted in many experimental advances in recent years and is the one we shall focus on in this thesis.

This approach of learning example, machine learning, can be taxonomized in many different ways (Figure 1.1) based on the interplay between data (e.g. examples), and loss function (e.g. signal). One such paradigm which was inspired by the field of behaviorist psychology is reinforcement learning (RL). RL attempts to formalize this process of training "agents" to maximize a "reward". This "reward" is generated using a function to incentivize behavior that we care about and penalizes behavior that we wish to discourage. In comparison to other paradigms of machine learning, where the loss function is a strong signal to guide model behavior, in the reinforcement learning setting the reward function is at best a proxy signal to guide the agent towards optimal behavior. Thus, in many ways, the formulation in this form results in a harder problem to solve. These notions of "optimality", "reward function", and further technical terms found later in this chapter will be defined later, but for now, serve as points to anchor the discussion. For a quick tour of reinforcement learning, we encourage reading these[1] guides[2], and for a more in-depth coverage we recommend the standard reference [50].

1.2 RECENT ADVANCES

In recent years, systems trained using reinforcement learning have seen impressive results in applications ranging from games (Atari [31], Go [47], Starcraft [54], DOTA [35]) to robot manipulation [25]. These advances can be attributed to many different factors from algorithmic contributions to experimental developments. Chief among these experimental developments has been the use of deep neural networks within reinforcement learning systems as powerful function approximators. These advances are paralleled in the broader machine learning field where the use of deep neural networks has fueled many impressive advances due to the ability of these family of functions to learn high dimensional models from large amounts of data [26]. However, these rapid advances have come at a resulting cost, namely the sacrifice of interpretability.

1 https://spinningup.openai.com/en/latest/spinningup/rl_intro.html
2 https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

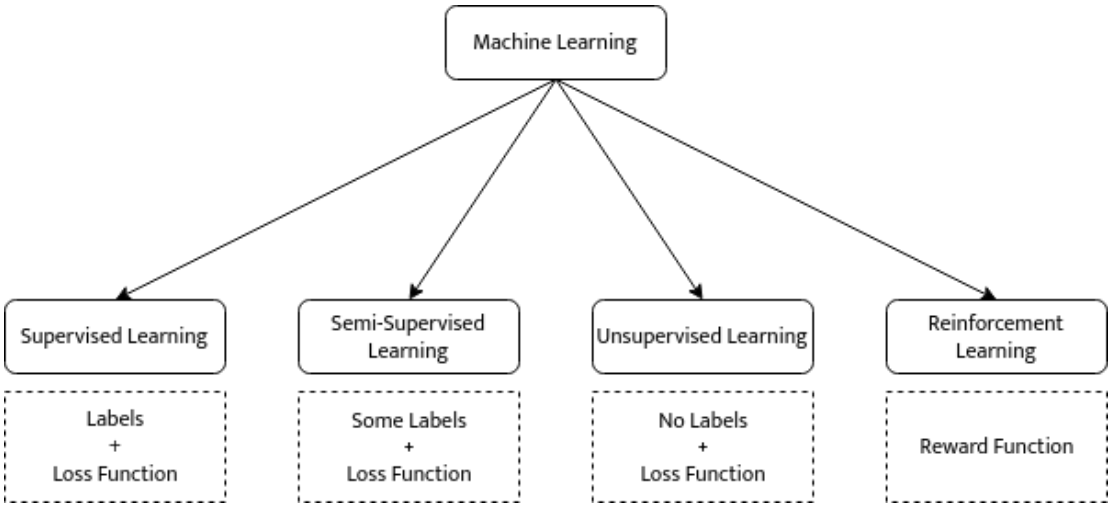


Figure 1.1: **Machine Learning Paradigms Taxonomy** One way to think about the different machine learning paradigms is by contrasting on the presence of labeled data and a loss function. This leads to a figure as shown that compares the different approaches: supervised learning, semi-supervised learning, unsupervised learning, reinforcement learning.

Inferences made using these high dimensional models are often hard to understand and trust [10].

Visualization systems have played an important role in overcoming these challenges. Many tools exist for addressing this challenge in the *supervised learning* setting, which find usage in tracking metrics [1, 43], generating graphs of model internals [56], and visualizing embeddings [28]. However, there is no corresponding set of tools for the reinforcement learning setting. At first glance, it appears we may repurpose existing tools for this task. However, we quickly run into limitations that arise due to the intent with which these tools were designed. Reinforcement learning (RL) is a more interactive science [32] compared to supervised learning, due to a stronger feedback loop between the researcher and the agent. Whereas supervised learning involves a static dataset, RL often entails collecting new data. To fully understand an RL algorithm, we must understand the effect it has on the data collected. Note that in supervised learning, the learned model does not affect a fixed dataset.

1.3 VISUALIZATION SYSTEMS

At their core visualization systems, consist of two components: *representation* and *interaction*. *Representation* is concerned with how data is mapped to a representation and then rendered. *Interaction* is concerned with the dialog between the user and the system as the user explores the data to uncover insights [57]. Though appearing to be disparate, these two processes have a symbiotic influence on each other. The tools we use for representation affect how we interact with the system, and our interaction affects the representations that we create. Thus, while designing visualization systems, it is important to think about the application domain from which the data originates, in this case, reinforcement learning.

The efficacy of the resulting system can be evaluated along the following three dimensions, as proposed by Beaudouin-Lafon [6], and adapted here for relevance:

- *descriptive power*: the ability to describe existing representations.

(a) **Representative Tool** (Tensorboard)

(b) **Vizarel**

Figure 1.2: **Tool Comparison** Comparing between a representative tool for debugging RL in the existing ecosystem **(L)**, and Vizarel (ours) **(R)**, highlights the difference in design intent between both systems. **L** Illustrates the plotting of scalar metrics such as policy loss or mean reward. **R** Illustrates an interactive interface designed to easily show the correspondence between scalar metrics and states (images). The points here are individual states in the replay buffer and the interactive scalar plot is a single trajectory, which we explain in further detail in Chapter 3.

- *evaluative power*: the ability to analyze alternative representations.

- *generative power*: the ability to generate new representations.

Figure 1.2 illustrates the differences between a representative tool from the existing ecosystem and the tool we have designed. The former was designed for the *supervised learning* setting and has shown promise for use in *reinforcement learning*. However, we argue that there exists a large space of unexplored interfaces that could help aid the process of debugging RL algorithms and trained policies. We explore one such solution that is designed around the spatial and temporal aspects of training RL agents. This approach might help increase understanding, interpretability, and thereby serve as a complement to tools in the existing ecosystem.

1.4 PRIOR WORK

As stated earlier, existing visualization tools for machine learning primarily focus on the supervised learning setting. However, the process of designing and debugging RL algorithms might benefit from a different set of tools, that can complement the strengths and overcome the weaknesses of offerings in the current ecosystem. In the rest of this section, we highlight aspects of prior work upon which our system builds. To the best of our knowledge, there do not exist visualization systems built for interpretable reinforcement learning that effectively addresses the broader goals we have identified. There exists prior work, aspects of which are relevant to features which the current system encapsulates, that we now detail.

Visual Interpretability

Related work for increasing understanding in machine learning models using visual explanations includes: feature visualization in neural networks [33, 48, 60], visual analysis tools for variants of machine learning models [21, 22, 24, 49, 58], treating existing methods as composable building blocks for user interfaces [34], and visualization techniques for increasing explainability in reinforcement learning [5, 30, 42]

Explaining agent behavior

There exists related work that tries to explain agent behavior. Amir and Amir [3] summarize agent behavior by displaying important trajectories. Waa et al. [55] introduce a method to provide contrastive explanations between user derived and agent learned policies. Huang et al. [18] show maximally informative examples to guide the user towards understanding the agent objective function. Hayes and Shah [17] present algorithms and a system for robots to synthesize policy descriptions and respond to human queries.

Explainable reinforcement learning

Puiutta and Veith [36] provide a survey of techniques for explainable reinforcement learning. Related work in this theme includes [9, 13, 20, 29, 37, 39, 46]

Similar to Amir and Amir [3], Huang et al. [18], and Waa et al. [55], this work is motivated by the aim to provide the researcher with relevant information to explore a possible space of solutions while debugging the policy. Similar to Hayes and Shah [17], we present a functioning system that can respond to human queries to provide explanations. However, in contrast, the interactive system we present is built around the RL training workflow, and designed to evolve beyond the explanatory use case to complement the existing ecosystem of tools [1, 43]. In contrast to the techniques surveyed in Puiutta and Veith [36], the contribution here is not on any single technique to increase interpretability, but a whole suite of visualizations built on an extensible platform to help researchers better design and debug RL agent policies for their task.

1.5 CONTRIBUTIONS OF THIS THESIS

The rest of this thesis describes our attempt at constructing Vizarel [3], an interactive visualization system to help RL researchers better understand algorithms and debug RL policies, and help RL researchers pose and answer questions of this nature. Towards these goals, we identify features that an interactive system for interpretable reinforcement learning should encapsulate and build a prototype of these ideas. We complement this by providing a walkthrough example of how this system could fit into the RL debugging workflow and be used in a real scenario to debug a policy.

Using existing tools, we can plot descriptive metrics such as cumulative reward, TD-error, and action values, to name a few. However, it is harder to pose and easily answer questions such as:

- How does the agent state-visitation distribution change as training progresses?
- What effect do noteworthy, influential states have on the policy?
- Are there repetitive patterns across space and time that result in the observed agent behavior?

These are far from an exhaustive list of questions that a researcher may pose while training agent policies, but are chosen to illustrate the limitations created by our current set of tools that prevent us from being able to easily answer such questions.

3 Vizarel is a portmanteau of visualization + reinforcement learning.

2

BACKGROUND

As alluded to in the introduction, reinforcement learning is a machine learning paradigm that formalizes the notion of training agents that interact with an environment. For a more in depth coverage of the subject, we refer the reader to [50]. In this chapter, we provide a brief primer and introduce notation [1] that is used in later parts of this thesis.

2.1 RL PRELIMINARIES

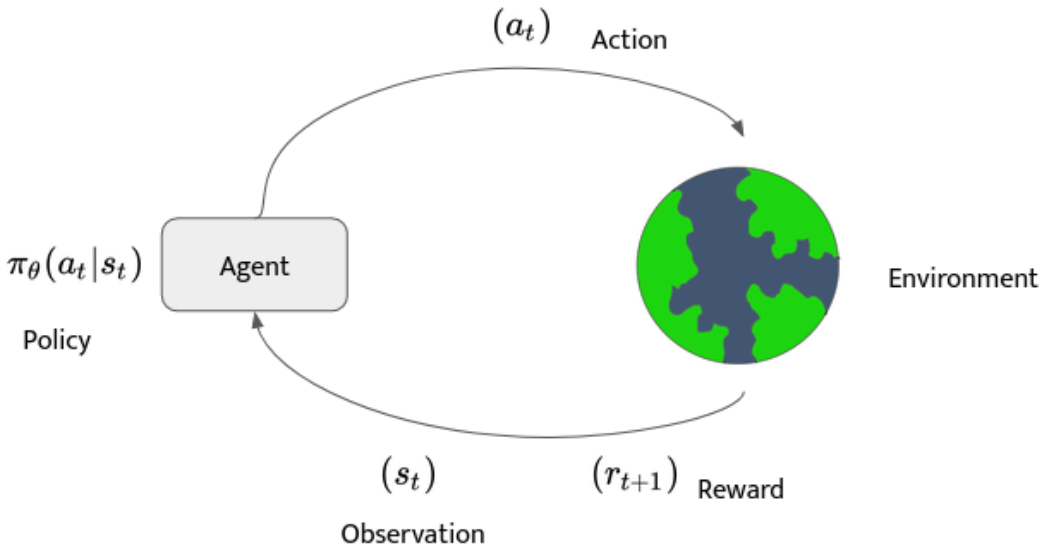


Figure 2.1: **RL Framework** Figure illustrating the reinforcement learning framework. The agent interacts with the environment at every timestep t by observing a state s_t and taking an action a_t. After taking the action the agent recieves a reward r_t.

2.1.1 *States, Actions, Rewards*

??

The main objects in RL are the *agent* and the *environment*. The environment is the world external to the agent. At every timestep t the agent interacts with the environment, observes a state s_t, and decides to take an action a_t after seeing this observation. The environment is updated after the agent takes an action a_t, but may evolve independently depending on the specific situation.

1 This section borrows from the excellent online resources that provide a quick tour of reinforcement learning.

Conditional on this action a_t, the environment returns a reward r_t which provides the agent with a signal regarding the quality of its action. The goal of the agent is to maximize the cumulative reward, also known as the *return*, and is what we look at next.

2.1.2 *Returns*

The *return* (G) is the cumulative sum of rewards (R) receieved by the agent.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.1}$$

where the subscript t is used to indicate timesteps, γ is a scalar quantity called the discount factor $\in [0, 1]$, that is used to control the time horizon over which rewards are meaningful.

Intuition: A relatively large discount factor, incentivizes the agent to care about the long term consequences of its actions, whereas a relatively small discount factor promotes hedonistic behavior. You can tweak this "window of importance" yourself by adjusting γ, and observing its' effect on the sum of the geometric series, which is $\frac{1}{1-\gamma}$. For example, setting $\gamma = 0.99$, yields a window of 100 frames.

2.1.3 *Policies*

The policy function is what encapsulates the behavior of the agent. The simplest form of a policy could even be a lookup table that maps states to actions. One step up in complexity could be a deterministic function that maps states to actions such as $a_t = \mu(s_t)$.
The policy function in its most general form (currently) is a stochastic function such as: $a_t \sim \pi(*|s_t)$.
Note, the policy function is often learnt using parameterized functions (such as a neural network), which is denoted by using a subscript such as θ or ϕ

$$a_t = \mu_\theta s_t \tag{2.2}$$
$$a_t \sim \pi_\theta(*|s_t) \tag{2.3}$$

Intuition: Think of the policy as a function returning a response to the following question: Which action should I take after seeing this state?

2.1.4 *Trajectories*

A trajectory is a sequence of states and actions the agent takes while interacting with the environment: $\tau = (s_0, a_0, s_1, a_1, ..., s_t, a_t)$
The initial state is sampled from the *start-state distribution*: $s_0 \sim \rho_0(*)$
State transitions (how the environment state changes) can occur either due agent actions, or due to an natural evolution of the environment, and can be either deterministic (2.4) or stochastic (2.5)

$$s_{t+1} = f(s_t, a_t) \tag{2.4}$$
$$s_{t+1} \sim f(*|s_t, a_t). \tag{2.5}$$

2.1.5 *Maximizing Expected Return*

The goal of an agent trained within the RL framework is to maximize return. With a stochastic policy and/or a stochastic environment, this turns into a problem of maximizing the *expected* return. The probability of observing the given trajectory under the current policy is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.6}$$
$$\tag{2.7}$$

where T is the length of the trajectory. The expected return is then

$$R(\pi) = \int_\tau P(\tau|\pi)G(\tau)d\tau = \mathbb{E}_{\tau\sim\pi}[G(\tau)] \tag{2.8}$$
$$\tag{2.9}$$

The goal of the agent is then to maximize this expected return

$$\pi^* = \arg\max_\pi R(\pi) \tag{2.10}$$

where π^* is the optimal policy.

2.1.6 *Value Functions*

The *value* of a state is defined as the expected return the agent would receive if it follows the current policy π forever after the current state s_t. Formally the value is specified as:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[G(\tau)|s_0 = s_t]. \tag{2.11}$$

A related quantity is the *state-action value function* which specifies the expected return if the agent takes an action a_t in state s_t and then follows the current policy π forever after. Formally the state-action value function is specified as:

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[G(\tau)|s_0 = s_t, a = a_t]. \tag{2.12}$$

In the case of following the optimal policy (π^*), the corresponding value function and state-action value function are denoted by V^* and Q^* respectively, where the optimal policy is the policy that maximizes the expected return

$$\max_\pi \mathbb{E}_{\tau\sim\pi} \tag{2.13}$$

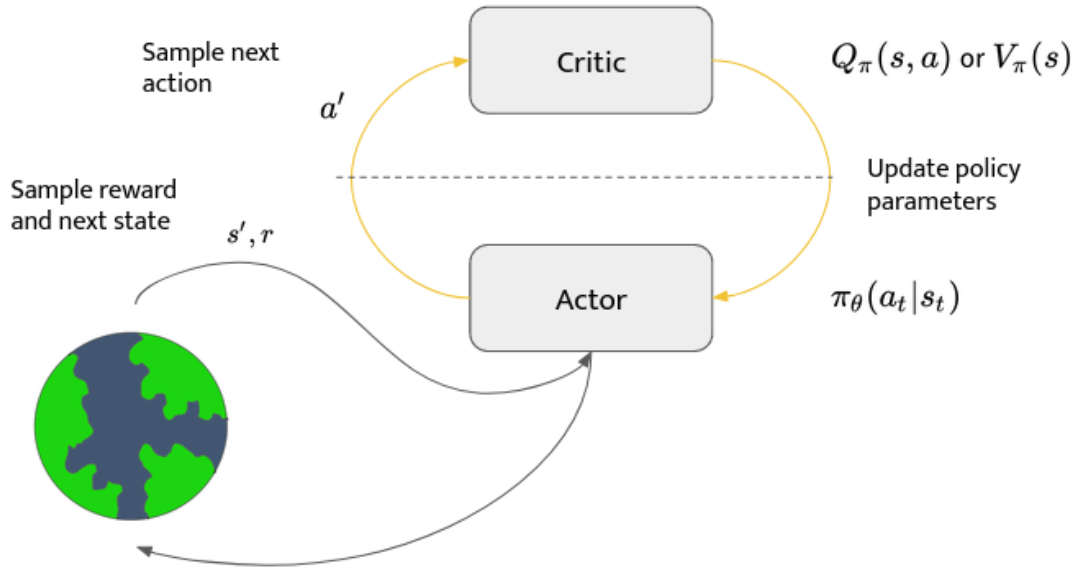


Figure 2.2: **Actor Critic Framework** An illustration of the actor critic learning framework. Here a **critic** update the value function parameters and the **actor** updates the policy function parameters according to a gradient signal from the critic.

Intuition: You can think of both the value functions as the agent learning a model of the world. e.g. how much reward am I likely to accumulate after seeing this state (value function)? how much reward am I likely to accumulate after (randomly or deterministically) taking this action, and then strictly following my policy (state-action value function)?

2.1.7 *TD Error*

TD Learning is a class of RL algorithms that learn the value function by bootstrapping from a current estimate of the value function. This is done by updating the value function V towards an estimated return (*TD Target*). The magnitude of the updates is controlled by a learning rate α:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{2.14}$$

where the quantity $(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$ is known as the *TD Error*.

Intuition: The TD error is a measure of how "far off" your current value function estimate is from the true value function. As such it is a proxy to the model prediction error, where the model the agent has learnt is of the reward it expects to receive after seeing the current state.

2.2 ACTOR CRITIC FRAMEWORK

The actor critic framework is a class of algorithms where we learn both the value function (or state-action value function), and the policy function. Methods belonging

to this framework can have update rules and nuances which differentiate them from each other, but are bound by this principle of learning both the value function and the policy function.

The following are general update rules for learning both functions

Actor

$$\theta \leftarrow \theta + \alpha_\theta Q(s,a;w)\nabla_\theta \ln \pi(a|s;\theta) \tag{2.15}$$

Critic

$$G_{t:t+1} = r_t + \gamma Q(s',a';w) - Q(s,a;w) \tag{2.16}$$

$$w \leftarrow w + \alpha_w G_{t:t+1} \nabla_w Q(s,a;w) \tag{2.17}$$

where,

θ : actor parameters, w : critic parameters,
γ : discount factor, α_θ : actor learning rate,
α_w : critic learning rate, G : return

2.3 LADDER OF ABSTRACTION

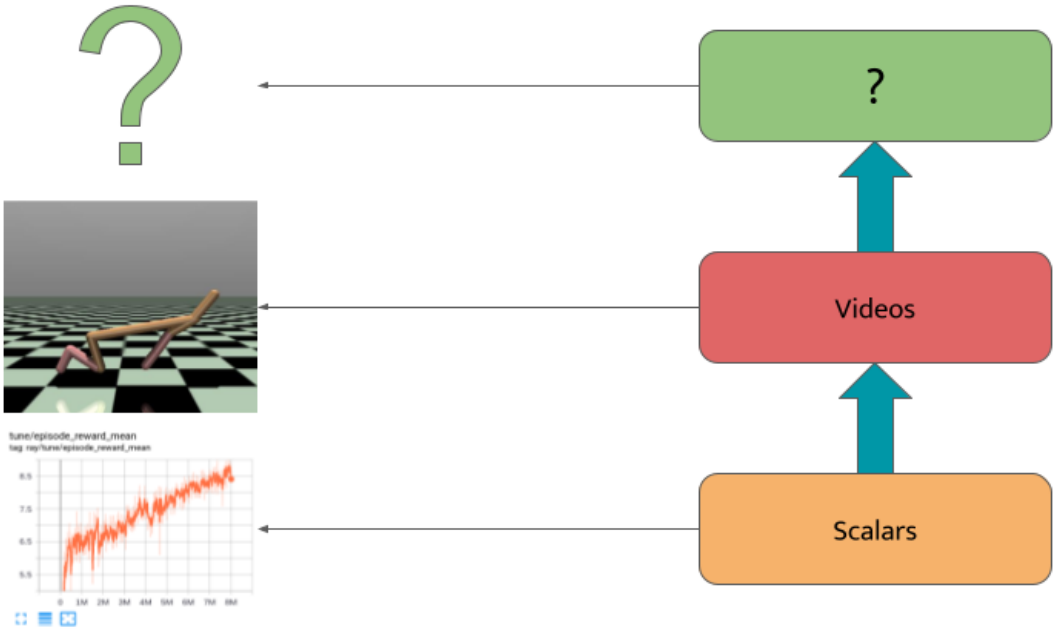


Figure 2.3: **Ladder of Abstraction** Sorting solutions in the existing ecosystem by increasing degrees of abstraction. We ask whether there lies anything higher in this "ladder of abstraction", a term coined by S.I. Hayakawa in Language in Thought and Action to highlight and contrast abstraction within language. We ask whether such an interface could provide an additional source of insight to the practitioner.

As briefly covered in the introduction, there currently exist tools that are used to debug RL algorithms and policies trained using RL. It is useful to think about how

these tools relate to each other, and where they lie in the space of all possible solutions. One way to think about this is to sort them by increasing degrees of abstraction. At the lowest level, are tools such as Tensorboard [51] which are commonly used to plot scalar metrics. One step above are video logs, which are often used to visualize the exact sequence of actions being taken by the agent

Arranging these by increasing degrees of abstraction leads to a figure such as that shown in Figure 2.3. Visualizing existion solutions in this space leads one to question whether the tools we've adopted from the supervised learning setting are a good fit for the RL setting. Could there exist better tools that provide more insight into the problem? Could these tools serve as complements to tools in the existing ecosystem?

2.4 UNDERSTANDING THE RL PROBLEM

To think about what such a solution might look like, it's important to identify which attributes of the RL problem are unique relative to other paradigms of machine learning. We idenfity one such dimension of comparison that we think leads to nice insights, but there might exist others.

2.4.1 *Spatial Interaction*

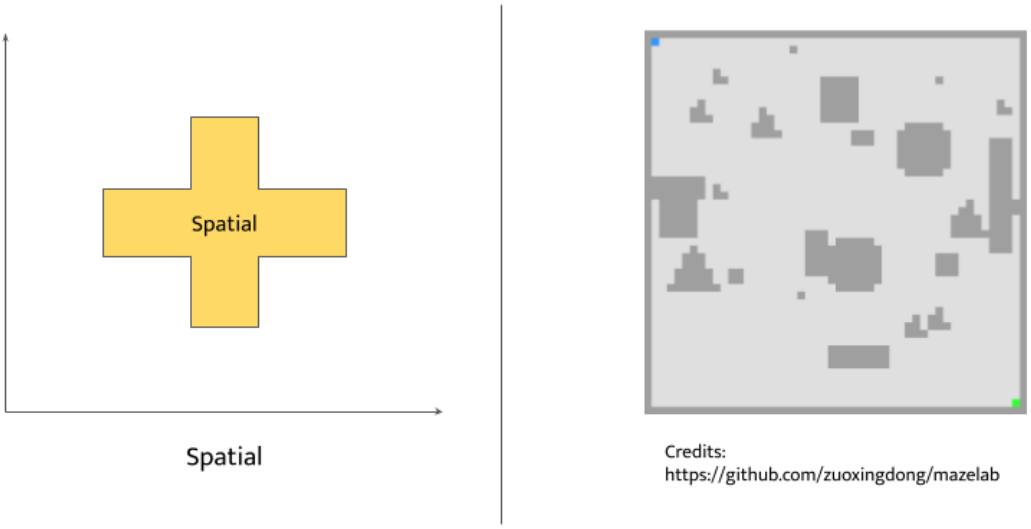


Figure 2.4: **Spatial Interaction** The spatial dimension is one axis of comparison along which to think about designing tools for insight into the RL problem. Here the agent (blue) is trying to navigate through a maze to reach a goal state (green), and can take actions to move around (up, down, left, right). A simple grid world such as this is already characterized by a strong spatial influence, which transfers across to similar problems (e.g. games, driving, robotics).

The RL framework is often characterized by a strong spatial influence, due to the notion of an agent interacting with an environment. This environment is frequently a simulation of the real world or a game environment which comes packaged with a notion of dimensions relative to which position is measured [2]. Of course this spatial

2 Note: We intentionally use the more abstract terms of (identifying) dimensions and (taking) measurements to highlight connections to what are very often the first steps carried out when interacting with physical systems.

interaction is by no means a prerequisite. For example, there exist scenarios where a policy trained using RL is being used for healthcare assitance. Another example is for ad caption generation it is unclear whether there exists a spatial component at all. However, note that even in these cases there exists dimensions and measurements along these dimensions, and thus we can easily repurpose the same ideas.

2.4.2 *Temporal Interaction*

Another dimension which is common to nearly all RL setups, is that of temporal interaction, which is formulated in the original RL framework **??**. This implies that there is a strong influence of time at every step and thus should be an important attribute to consider when designing a tool specifically for the RL problem.

We can then fuse these two axes of comparison (spatial and temporal) to arrive at interactions which lie in between these two (e.g. spatio-temporal). These concepts are used later in Chapter 3, when explaining the system we've designed.

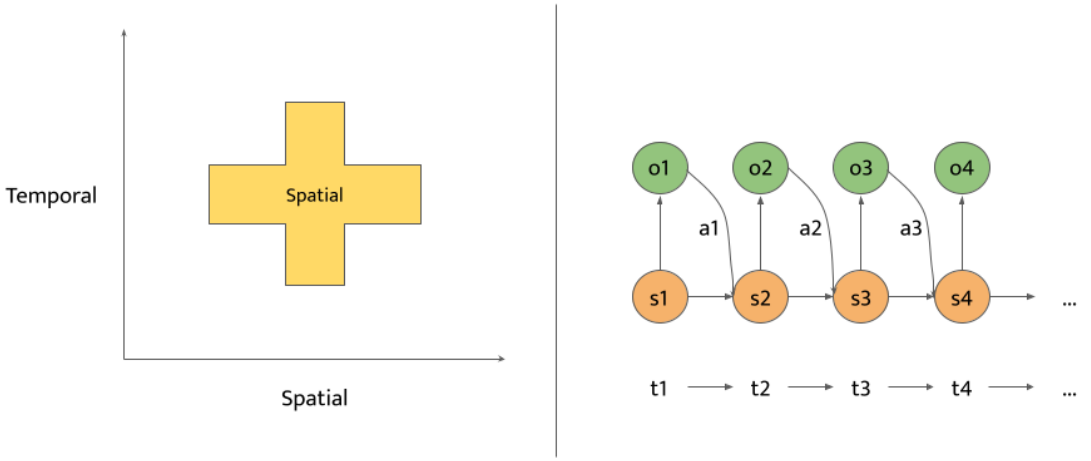


Figure 2.5: **Temporal Interaction** The temporal dimension is shown here by visualizing a Markov Decision Process (MDP). Here the agent interacts with the environment by taking an action a_t at every timestep t which causes a state change.

These background concepts cover most of what we'll need to understand and contextualize the ideas from the later chapters.

3

VIZAREL: INTERACTIVE VISUALIZATIONS FOR RL

This section describes how our interactive visualization system (Vizarel), is currently designed. The system offers different views that allow the user to analyze agent policies along spatial and temporal dimensions (described later in further detail). The tool consists of a set of *viewports*, that provide the user with different representations of the data, contingent on the underlying data stream. Viewports are generated by chaining together different visualization elements, such as:

1. *image buffers*: visualize observation spaces (image and non-image based)
2. *line plots*: visualize sequentially ordered data, such as action values or rewards across time
3. *scatter plots*: to visualize embedding spaces or compare tensors along specified dimensions
4. *histograms*: visualize frequency counts of specified tensors or probability distributions

The current implementation provides core viewports (detailed further), but can easily be extended by the user to generate additional viewports to explore different visualization ideas. This design naturally leads to the idea of an ecosystem of plugins that could be integrated into the core system, and distributed for use among a community of users to support different visualization schemes and algorithms. For example, the user could combine image buffers and line plots in novel ways to create a viewport to visualize the the state-action value function [50]. In the rest of this section, we provide details and distinguish between two types viewports currently implemented in Vizarel: temporal viewports and spatial viewports. Discussion on viewports beyond these has been deferred to the appendix. Comprehensive information on adding new viewports is beyond the scope of the thesis, but has been described at length in the system documentation[1].

3.1 TEMPORAL VIEWS

Temporal views are oriented around visualizing the data stream (e.g. images, actions, rewards) as a sequence of events ordered along the time dimension. We have implemented three types of temporal viewports: state viewports, action viewports, and reward viewports, which we now detail.

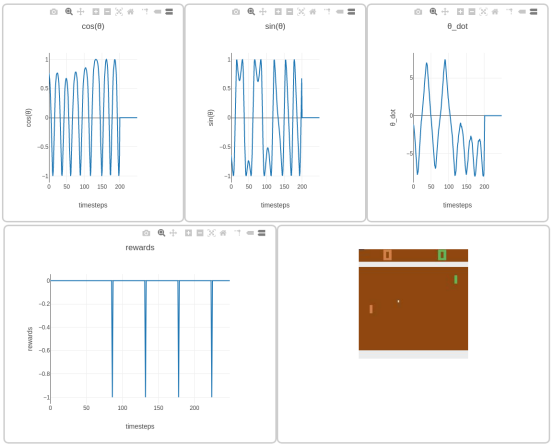


Figure 3.1: **State + Action Viewports (T)** Visualizing the state viewport for the inverted pendulum task. This representation overlayed with another state viewport similar to (b), provides the user with better intuition about the correspondence between states and actions for non image state spaces. **(B)** Visualizing the action viewport for the Pong environment [7]. Hovering over instantaneous timesteps dynamically updates the state viewport (3.1.1) and shows the corresponding rendered image for the selected state. This representation provides the user with intuition about the agent policy, and could help subsequent debugging.

3.1.1 *State Viewport*

For visualization, we can classify states as either image-based or non-image based. The type of observation space influences the corresponding viewport used for visualization. We provide two examples that illustrate how these differing observation spaces can result in different viewports. Consider a non-image based observation space, such as that for the inverted pendulum task. Here, the state vector $\vec{s} = \{\sin(\theta), \cos(\theta), \dot{\theta}\}$, where θ is the angle which the pendulum makes with the vertical.

We can visualize the state vector components individually, which provides insight into how states vary across episode timesteps (Figure 3.1). Since images are easier for humans to interpret, we can generate an additional viewport using image buffers, that tracks changes in state space to the corresponding changes in image space. Having this simultaneous visualization is useful since it now enables us to jump back and forth between the state representation which the agent receives, and the corresponding image representation, by simply hovering over the desired timestep in the state viewport.

For environments that have higher dimensional state spaces, such as that of a robotic arm with multiple degrees of freedom, we can visualize individual state components. However, since this may not be intuitive, we can also generate an additional viewport to display an image rendering of the environment to help increase interpretability.

3.1.2 *Action Viewport*

The action viewport is used to visualize how the actions chosen by the agent vary across the episode (Figure 3.1). Consider the Pong environment [7], where the action a at timestep t, corresponds to the direction in which the paddle should move. A visualization such as the one shown in Figure 3.1 can be generated to show the correspondence

1 Vizarel is planned to be released as an open source tool

between actions and states. This allows the user to easily identify states marked by sudden action transitions, and thus aid debugging. This idea can easily be extended to agents with stochastic actions, where we could generate a viewport using histograms to visualize the change in action distribution over time.

For higher dimensional action spaces we can use a technique similar to the one used for state viewports, to generate multiple viewports that track individual action dimensions, for example joint torques for a multilink robot.

3.1.3 *Reward Viewport*

The reward viewport is used to visualize how the rewards received by an agent vary across the episode. A user can look at the reward viewport together with the state viewport to understand and find patterns across state transitions that result in high reward. For many environments, the reward function consists of components weighted by different coefficients. These individual components are often easier to interpret since they are usually correspond to a physically motivated quantity tied to agent behaviors that we wish to either reward or penalize. For example, in autonomous driving environments the reward can be formulated as a function of speed, collision penalties, and the distance from an optimal trajectory [2].

In situations where we have access to these reward components, we can generate multiple viewports each of which visualize different components of the reward function. The viewports discussed so far can be combined to provide the user more insight into the correspondence between states (state viewport), actions (action viewport), and the components of the reward function (reward viewport) that the agent is attempting to maximize. Such a visualization could help alert researchers to reward hacking [4].

3.2 SPATIAL VIEWS

Spatial views are oriented around visualizing the data stream as a spatial distribution of events. We have implemented three types of spatial viewports: replay buffer viewports, distribution viewports, and trajectory viewports, that we now describe.

3.2.1 *Replay Buffer Viewport*

The replay buffer stores the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ in a buffer $B = \{e_0, e_1, ..., e_T\} \forall i \in [0, T]$. For off-policy algorithms, the replay buffer is of crucial importance, since it effectively serves as an online dataset for agent policy updates [11]. In the supervised learning setting, there exist tools[2] to visualize datasets, that provide the user with an intuition for the underlying data distribution. The replay buffer viewport aims to provide similar intuitions for the reinforcement learning setting by visualizing the distribution of data samples in the replay buffer.

Since the individual elements of the replay buffer are at least a four-dimensional vector e_t, this rules out the possibility of generating viewports to visualize data in the original space. We can instead visualize the data samples by transforming the points [28] to a lower-dimensional representation. This technique helps visualize the distribution of samples in the replay buffer, which is a visual representation of the replay buffer diversity [8].

2 https://github.com/PAIR-code/facets

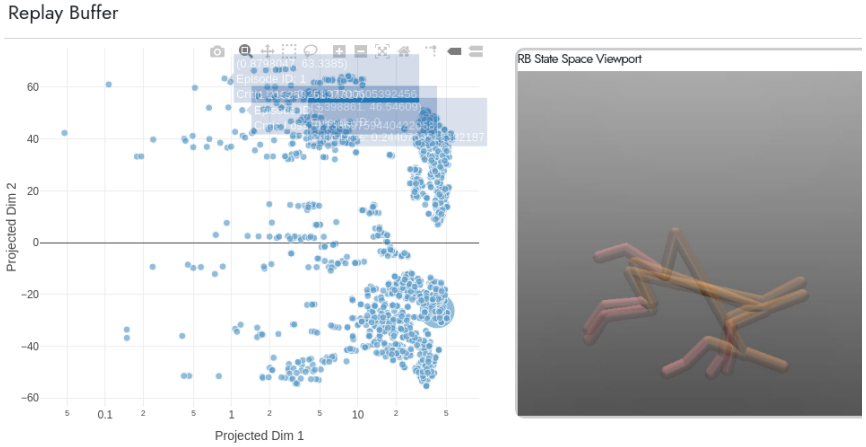


Figure 3.2: **Replay Buffer Viewport** Projecting the contents of the replay buffer into a 2D space for easier navigation, and clustering states based on similarity. This viewport provides a visual representation for replay buffer diversity and can help in subsequent debugging. Hovering over points in the replay buffer dynamically updates the generated state viewport (3.1.1), and shows the rendered image for the corresponding state (animation depicted using overlay).

The size of the replay buffer can be quite large [61], which can lead to difficulties while navigating the space of points visualized in the replay buffer viewport. To nudge [52] the user towards investigating samples of higher potential interest, we scale the size of points in proportion to the absolute normalized TD error [50], which has been used in past work [44] as a measure of sample priority during experience replay.

Moreover, the replay buffer viewport can be combined with the state viewport to simultaneously visualize an image rendering of the state, by tracking changes as the user hovers over points in the replay buffer viewport (Figure 3.2).

3.2.2 *Distribution Viewport*

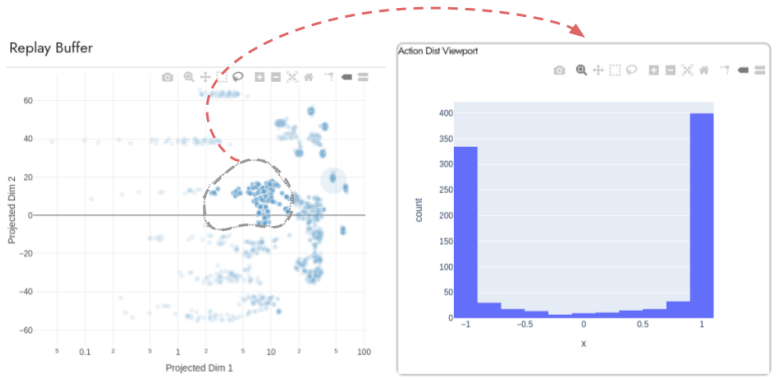


Figure 3.3: **Distribution Viewport** Using the lasso tool to select a group of points (dashed gray line) in the replay buffer viewport (3.2.1), dynamically updates (dashed red line) the distribution viewport (3.2.2) by computing and plotting the distribution of values for the specified tensor (e.g. actions or rewards).

The distribution viewport (Figure 3.3) complements the replay buffer viewport by allowing the user to select clusters of data samples and ask questions regarding the distribution of action, rewards, and other relevant tensors for the selected group of points.

Using the distribution viewport, users might ask questions like:

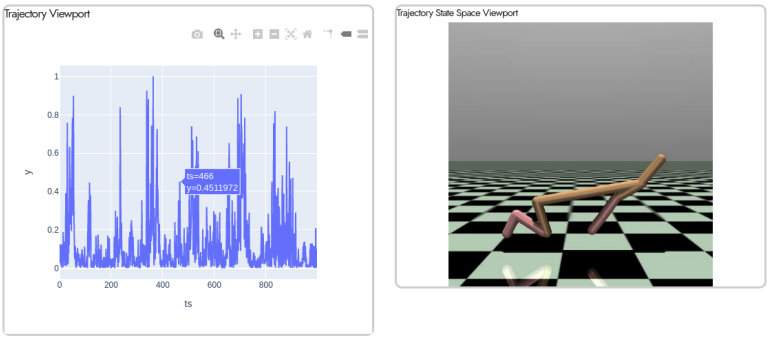


Figure 3.4: **Trajectory Viewport** Selecting points in the replay buffer viewport (3.2.1), causes the trajectory viewport (3.2.3) to dynamically update and plot the absolute normalized TD error values over the length of the trajectory. Hovering over points in the trajectory viewport, allows the user to view a rendering of the state corresponding to that timestep in the generated state viewport (3.1.1).

- What is the distribution of actions the agent took for these groups of similar states?
- What is the distribution of rewards for the state action transitions?
- What is the overall diversity of states which the agent has visited?

If the updates to the agent policy result in better task performance, the entropy of the action distribution should reduce over time (discounting any external annealing caused due to exploration), which can be easily verified through this viewport. In the limit, the distribution of actions for a group of similar points should converge to a Dirac distribution, since the optimal policy for an infinite horizon discounted MDP is a stationary distribution [38]. In practice, observing the distribution converging around the mean value could indicate a promising policy training experiment.

For multi-dimensional action spaces, the viewport could be repurposed to display the variance of the action distribution, plot different projections of the action distribution, or use more sophisticated techniques such as projection pursuit [19].

3.2.3 *Trajectory Viewport*

A fusion of the components from the spatial and temporal views leads to a *spatio-temporal view*, an example of which is the trajectory viewport (Figure 3.4). The replay buffer viewport alone visualizes the spatial nature of the points in the replay buffer but does not display the temporal nature of trajectories. Being able to switch between spatial and temporal views is crucial when understanding and debugging policies. This is supported by selecting points in the replay buffer viewport, which then retrieves the corresponding trajectory.

The X coordinate in the trajectory viewport represents the timestep, and the Y coordinate is the absolute TD error, normalized to lie within $[0, 1]$. Hovering over points in the trajectory viewport retrieves an image rendering of the corresponding state in the state viewport. This correspondence enables the user to easily navigate through and visualize action sequences in the trajectory that consistently have a high TD error, thus speeding up debugging of policies.

3.2.4 *Tensor Comparison Viewport*

For environments that have higher dimensional action spaces, it is hard for the user to understand how neighboring points in the replay buffer viewport differ. This becomes especially relevant for diagnosing clusters of points that have a higher TD error. The tensor comparison viewport (Figure 3.5) enables the user to easily select points and then compare them along the dimensions of interest, which for example could be actions. Dimensions that have a standard deviation beyond a specified threshold are automatically highlighted, which enables the user to focus on the dimensions of interest.

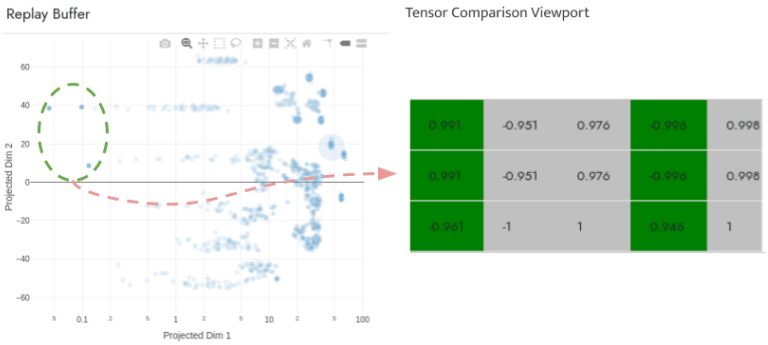


Figure 3.5: **Tensor Comparison Viewport** Selecting points (dashed green line) in the replay buffer viewport (3.2.1), and generating (dashed red line) the tensor comparison viewport, allows the user to compare tensors (e.g. actions or states), where dimensions of higher variance are automatically highlighted. This could lead to faster debugging in environments where each dimension corresponds to physically intuitive quantities.

3.3 CONSTRUCTING VIEWPORTS

We describe how a user could create a new viewport through an example. However, we defer an extended discussion to the system documentation, since reading the source code and give the user more insight. As mentioned in Section **??**, viewports are generated by chaining together different visualization elements. Various viewports we introduced along with the visualization elements they make use of are:

1. *image buffers*: state viewport
2. *line plots*: action viewport, reward viewport, trajectory viewport
3. *scatter plots*: replay buffer viewport
4. *histograms*: distribution viewport

Note that these primitives are not fixed and are bound to change if the creation of different viewports necessitates their expansion. However, we've found them to be a good starting point to provide the minimal functionality required to construct new viewports. For example the construction of a *saliency map viewport*, could be done using an *image buffer* and a *line plot*.

There exist utilities in the system to handle the rollout of agent policies, and storage of generated metadata. However, the user would still need to provide code to generate saliency maps[3]. Once the metadata has been generated, the user specifies which viewports to generate (e.g. core viewports and custom viewports) along with a visual layout for the dashboard. The system then generates an interactive interface with the

3 for which there exist open source tools

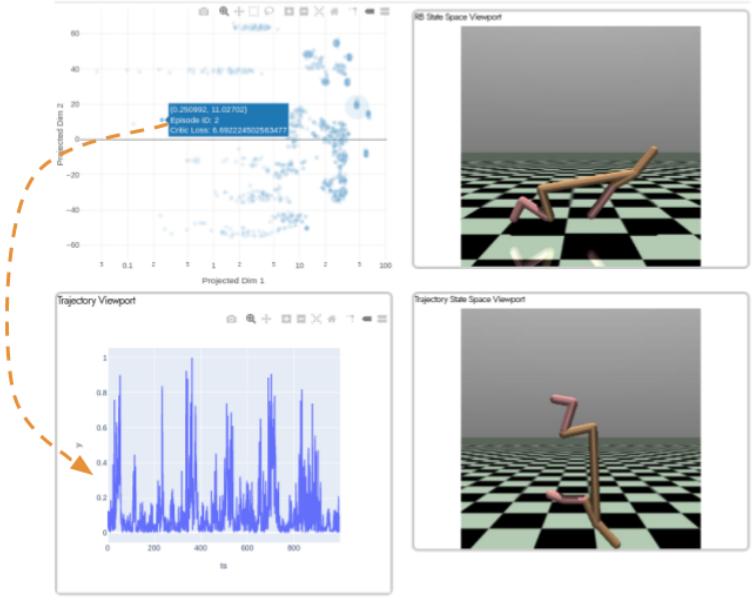


Figure 3.6: **Spatio-Temporal Interaction** Visualizing the replay buffer viewport (3.2.1) (spatial view), and trajectory viewport (3.2.3) (temporal view), along with overlays to independently track image renderings of states in both as a state space viewport (3.1.1). Navigating between these viewports allows the user to observe both agent spatial and temporal behavior, which could facilitate better insights during debugging.

specified viewports and layout, that the user can use to debug the agent policy and perform further analysis.

This viewport could further be incorporated as a plugin or extension to the core system and distributed to a community of users in the future. The exact details for this are not concrete yet, since we expect there to emerge a robust process through iterative design changes, as the tool finds broader usage.

3.4 WALKTHROUGH OF DEBUGGING AGENTS USING VIZAREL

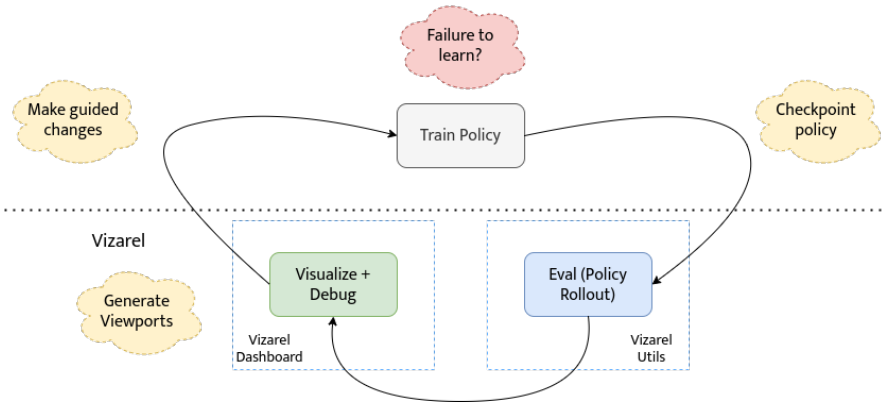


Figure 3.7: **Vizarel Workflow Diagram** Typical steps during policy debugging, and how the designed system fits into this workflow. The system takes as input a policy saved during a checkpoint and evaluates the policy through a specified number of rollouts. This data is then visualized through viewports specified by the user, that are used for debugging the policy through making guided changes.

We now detail an example workflow of how the system can be used in a real scenario. Figure 3.7, illustrates how Vizarel fits into an RL researcher's policy debugging work-

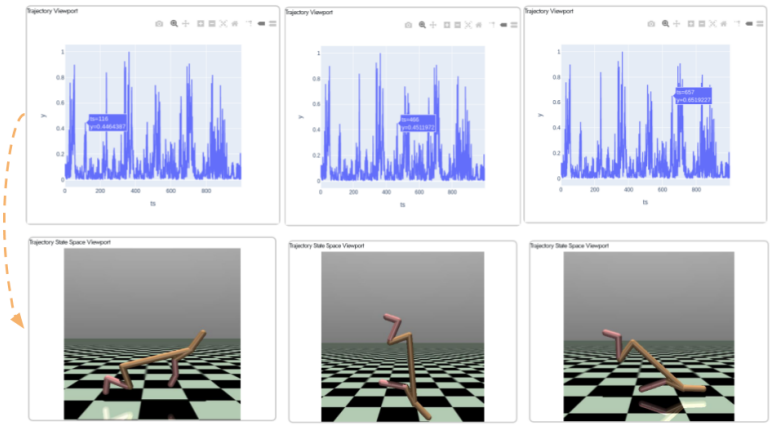


Figure 3.8: **Comparing TD error along an agent trajectory** Visualizing the trajectory viewport (3.2.3), allows the user to compare the TD error at different timesteps along the trajectory, along with the associated state viewport (3.1.1). An example interaction is visualized here by hovering over regions of potential interest in the trajectory viewport. This simultaneous view allows the user to easily compare and draw similarities between action sequences which cause large changes in TD error.

flow. Training a successful agent policy often requires multiple iterations of changing algorithm hyperparameters.

To speed up and increase the intuitiveness of this process, the researcher can load a stored checkpoint of the policy into the system, and evaluate a specified number of policy rollouts. Empirically, we've found that there should be enough rollouts to ensure sufficient coverage of the state space, since this influences the scope of questions which can be posed during debugging (e.g. through the replay buffer viewport). These rollouts can then be visualized and interacted with through specifying the required data streams and generating different viewports.

Figure 3.6, shows an example of replay buffer, state, and trajectory viewports generated for a policy trained using DDPG on the HalfCheetah task. The high variance in the TD error suggests the presence of critic overestimation bias [53], which could be remedied by using algorithms known to reduce the impact of this issue [12, 16]. Figure 3.8 shows how the user can compare the TD error along the agent trajectory. Hovering over regions of potential interest in the trajectory viewport allows the user to find action sequences that cause high variance in the TD error. A similar technique could be used to visualize clusters of states in the replay buffer space with high TD error (Figure 3.2). This approach could enable the user to identify patterns in states across space or time that persistently have high TD error, and design methods to mitigate this [4].

Another approach the user could take is to generate a distribution viewport (Figure 3.3), and identify the distribution of actions in the vicinity of states with a high TD error. If similar states persistently have a higher action and/or reward variance, this suggests that the usage of variance reduction techniques could help learning [41, 45]. Once promising avenues for modification have been identified, the user can make guided changes, and retrain the policy.

3.5 ALGORITHMS

Algorithm 1: Procedure to generate samples for the replay buffer viewport

Result: Data samples from replay buffer transformed from original space $\mathbb{R}^n$ to lower dimensional space $\mathbb{R}^d$, where $n >> d$

Load data samples d_i from replay buffer into memory;
Where, $d_i = (s_i, a_i, r_i, s_{i+1})$, $i \in (0, T)$ and T is the episode timestep;
for *i=0; i < T; i++* **do**
    // create data matrix from replay buffer samples;
    data[i] = d_i;
end
transform = compute_transform(data, type);
// where, type ∈ [PCA, TSNE, UMAP]
reduced_points = transform(data);

Algorithm 2: Procedure to compute visual size of data samples in the replay buffer viewport

Result: Visual size of data samples in replay buffer viewport transformed in proportion of their influence on the agent policy

Load data samples d_i from replay buffer into memory;
Where, $d_i = (s_i, a_i, r_i, s_{i+1})$, $i \in (0, T)$ and T is the episode timestep;
Load sample metadata m_i into memory;
Where, m_i stores the TD error for each sample (for relevant algorithms) ;
for *i=0; i < T; i++* **do**
    data[i] = m_i;
end
d_min = min(data);
d_max = max(data);
for *i=0; i< T; i++* **do**
    // p[i] stores the normalized TD error value, and (c_min, c_max) is the range to which values are mapped;
    p[i] = $\frac{p[i]-d\_min}{d\_max-d\_min} \times (c\_max - c\_min) + (c\_min)$;
    // r[i] stores the radius of the point to be plotted in the replay buffer viewport ;
    r[i] = $\sqrt{\frac{p[i]}{\pi}}$;
end

3.6 ALTERNATE ENVIRONMENTS: HARD EXPLORATION

We've also experimented with using this tool for hard-exploration tasks such as Montezuma's Revenge. Figure 3.9, shows how the replay buffer viewport can be used to visualize the distribution of data samples in the replay buffer. Since, the observations returned from the environment are images, we extract the embeddings computed by the feature extraction model in the agent policy, and use these for the projection technique described in the Appendix (Algorithms).

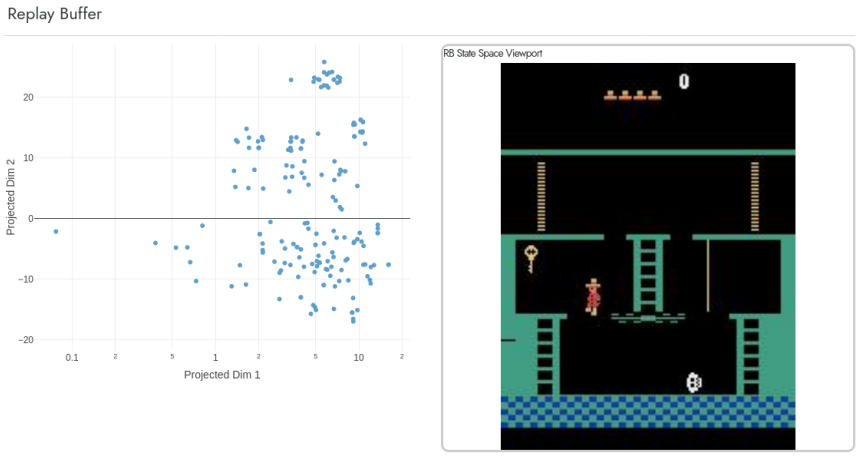


Figure 3.9: **Visualizing the replay buffer for hard exploration tasks** Tasks such as Montezuma's revenge are classic examples of hard exploration tasks. Here we show how the replay buffer viewport, can help visualize the distribution of data samples in the replay buffer.

3.7 USER STUDY

We conducted a user study[4] from RL users for feedback and an extended evaluation of potential use cases a tool such as the one described and implemented in this paper would serve.

These questions were of both a numerical and subjective type. We now list both types along with preliminary results

3.7.1 *Numerical questions*

- On a scale of 0-10, do you think Vizarel would help you identify bugs in your RL algorithms? **Average: 7.5**
- On a scale of 0-10, do you think Vizarel would help you identify improvements in your RL algorithms? **Average: 7.0**
- On a scale of 0-10, do you think Vizarel would help you understand whether your RL algorithms are working as intended? **Average: 8.0**

3.7.2 *Subjective questions*

- Are there specific settings where you think this tool might help answer questions that you might otherwise not easily been able to?
  - To show the effects of changes in reward function coefficients
  - Surfacing important points in the agent trajectory history
- Which features do you think are missing and would be a useful addition to have?
  - Add the capability to search over the replay buffer viewport and filter events based on search criteria.
  - Provide a documented approach to load in agent policies

4 Results from this are preliminary, as the survey is still in progress

- More details on how to create plugins
- Display agent trajectories in the replay buffer viewport

Future work along this direction would include creating test scenarios for debugging, and running an A/B test for users, contrasting their experience with existing tools vs the proposed tool along dimensions of efficacy in debugging RL algorithms.

3.8 PERFORMANCE

We've run measured preliminary performance metrics to help provide insight into how much overhead running this system would create (based on v4 of the system). These numbers were collected for a visualization of an agent trained using DDPG on the HalfCheetah-v2 task, about 35% of the way to task completion. The vizarel interface was generated on a machine with an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processor, with a 4TB HDD, and 128 GB RAM.

Tasks	**Time**
Generate Viewports	45 sec
Load Dashboard	5 sec
Policy Rollouts	2 minutes
Logging Overhead Fraction (Relative to Tensorboard)	1

Note that the policy rollout time is conditional on the length of the episode trajectory. These were collected for the HalfCheetah-v2 task.

3.9 FUTURE WORK

In this section, we introduced a visualization tool, Vizarel, that helps interpret and debug RL algorithms. Existing tools which we use to gain insights into our agent policies and RL algorithms are constrained by design choices that were made for the supervised learning framework. To that end, we identified features that an interactive system for debugging and interpreting RL algorithms should encapsulate, built an instantiation of this system which we plan to release as an open source tool, and provided a walkthrough of an example workflow of how the system could be used.

There are multiple features under development that contribute towards the core system. One feature is the integration of additional data streams such as saliency maps [15] to complement the state viewport. Another is designing the capability to use the system in domains that lack a visual component, for example in healthcare [59] or education [40]. An extension is to add search capabilities that allow the user to easily traverse, query, and identify regions of interest in the replay buffer viewport.

Vizarel suggests a number of avenues for future research:

1. We hypothesize that it could help design metrics that better capture priority during experience replay [44].
2. It could help the researchers create safety mechanisms early on in the training process through identifying patterns in agent failure conditions [4].
3. Another possible research direction this tool catalyzes is the construction of reproducible visualizations through plugins integrated into the core system.

We anticipate that the best features yet to be built will emerge through iterative feedback, deployment, and usage in the broader reinforcement learning and interpretability research communities.

4

VISUALIZING ACTOR CRITIC LEARNING DYNAMICS

As described in Chapter 3, the system designed is built with extendability in mind. These extensions can take the form of simple visualizations or can emerge through more involved research efforts. In this chapter we describe an instance of the latter.

4.1 MOTIVATION

Optimization algorithms for machine learning have often made use of geometric insights to design more optimal convergence behavior. These algorithms are often used to optimize the behavior of complex models that exist in extremely high dimensional spaces. However, humans aren't able to easily visualize higher dimensional spaces, and translate their insights from everyday lower dimensional space to the spaces in which these models exist. Hence, in recent years we've witnessed a multitude of techniques that try to address this problem. Recent work has even made inroads into attempting to visualize the nature of this higher dimensional loss landscape.

Thus, since optimization algorithms are often highly tied to geometry, understanding the geometry underlying the optimization problem can lead to better insights, and potentially better priors. Extending this claim to the reinforcement learning (RL) framework, we argue that understanding the geometry of the RL could lead to better optimization algorithms. To do so, we must first gain insights into the difficulty of the RL optimization problem.

4.2 DIFFICULTY OF OPTIMIZATION

Unlike the supervised learning setting, where the loss function provides a strong signal for gradient updates, the reward function in the RL framework is at best a weak proxy to encourage positive behavior, which adds to the difficulty of the optimization problem.

However even for the RL setting, the difficulty of the optimization lies on a continuum. For algorithms such as REINFORCE, the loss function doesn't change. For deep Q learning, with target networks, the loss function is non stationary but changes slowly (e.g. every k timesteps). On the far end of the spectrum, is the actor critic formulation, where the loss function changes at every timestep. One proxy measure we propose to gain insights into the complexity of the optimization is to look at the variance of gradient updates across time for similar states under these different algorithms.

As suggested by the figure, the actor critic optimization problem lies further along the difficulty continuum. This makes sense considering that REINFORCE is closer to the supervised learning framework, whereas algorithms in the actor critic family are more closely related to bilevel optimization or two player games such as that of generative adversarial networks. Despite this difficulty, recent research has shown RL agents

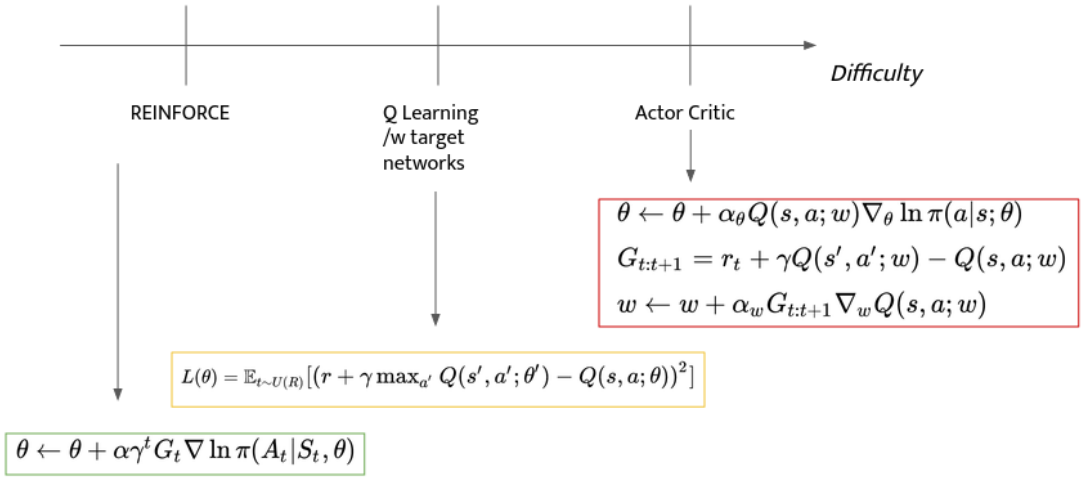


Figure 4.1: **Plotting increasing optimization difficulty** Figure showing different RL algorithms / frameworks in an increasing order of optimization difficulty. REINFORCE (also known as Monte-Carlo Policy Gradient), has a fixed loss function. Deep Q-learning is trained with a target network that is updated every k timesteps, where k is a user defined hyperparameter. In the Actor Critic framework, the loss function is influenced by the critic, which is updated at every timestep.

trained with the actor critic framework to display remarkable performance across a suite of benchmark tasks. However, actor critic algorithms suffer with stability issues during training, a phenomenon which is remedied with a host of different techniques.

We ask, whether these drawbacks are to some degree a function of not having geometric insight into the nature of the actor critic optimization problem. Through the proceeding sections, we seek to impart additional clarity into the nature of the actor critic loss landscape and how this affects the learning algorithm.

4.3 PRIOR WORK

4.3.1 *Two Player Games*

As shown in past work [23], the actor critic family of algorithms are a special case of the broader bilevel optimization framework, examples of which are GANs. There are striking parallels between actor critic (AC) optimization and GANs. In the GAN formulation, the generator network optimizes a loss defined by the discriminator network, whereas in the AC framework, the actor network optimizes a policy with respect to a value function learnt by the critic network.

Here the critic plays the role of the loss function, which defines the loss landscape over which the actor optimizes the policy. However, this optimization is done over extremely high dimensional parameter spaces, which makes it difficult to easily gain insight into the nature of the optimization problem.

4.3.2 *Loss Landscape*

Recent work has made inroads into visualizing the loss landscape of neural networks in the supervised learning setting. We apply similar techniques to compute the loss

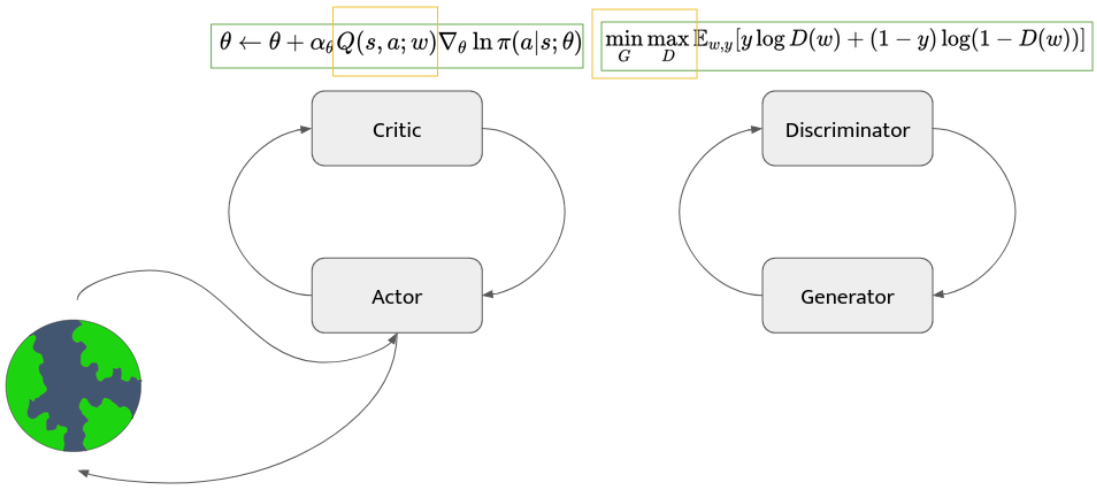


Figure 4.2: **Parallels between Actor Critic and GANs** Figure highlighting the similarities between the actor critic framework and generative adversarial networks. In the GAN formulation, the generator network optimizes a loss defined by the discriminator network, whereas in the AC framework, the actor network optimizes a policy with respect to a value function learnt by the critic network.

landscape for actor critic optimization, where the loss function is non-stationary and in fact changes at every timestep.

As the visualization suggests, the optimization problem is much harder than the supervised learning setting, where the landscape does not change over time, due to a fixed loss function. Even for the supervised learning setting, the ease of training is dependent on network architecture, optimization algorithm, initialization, and many other design choices. In the RL setting, there exist many proposed techniques to increase stability of training including: target networks, entropy regularization, learning rates (typically much smaller in RL than in ML), action noise, parameter noise, Double DQN, Dueling DQN, TD3 min Q trick, Gradient clipping.

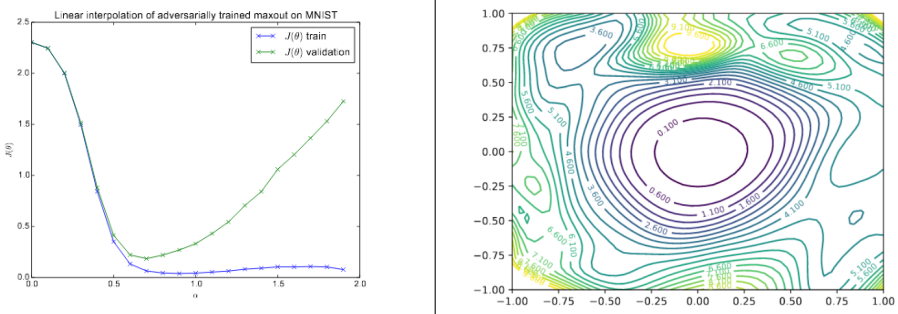
Visualization schemes such as that shown in Figure 4.3, provide a means for us to gain further insight into how these proposed factors affect the loss landscape, and thus the difficulty of the optimization problem. In the past such schemes have been used to visualize 1D and 2D projections of loss functions for supervised learning. The technique we use is closely related to this prior work and is what we briefly survey for context next.

4.3.2.1 *1D Loss Visualization*

The basic algorithm here as described in [14] is:

1. take two sets of parameters θ and θ'
2. generate a line joining these points (e.g. $\theta(\alpha) = (1-\alpha)\theta + \alpha\theta'$) where α is a scalar value.
3. plot values of the loss along this line $f(\alpha) = L(\theta(\alpha))$

Figure 4.3 shows an example of such a plot of linearly interpolating the loss of neural networks trained on the MNIST dataset [14]



(a) **1D Loss Surface** (b) **2D Loss Surface**

Figure 4.3: **Visualizing loss surfaces** Figure illustrating the procedure described in [14, 27] to plot the loss surface between two parameters in 1D and the loss surface in 2D for a neural network trained for digit classification on the MNIST task.

4.3.2.2 *2D Loss Visualization*

The basic algorithm here as described in [27] is:

1. Choose a parameter vector θ as the center of the graph
2. Choose random direction vectors η and δ
3. Plot the loss surface 1D ($f(\alpha) = L(\theta^* + \alpha\delta)$) 2D ($f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta)$)

We follow a similar procedure as that described in [27]. Note here that the loss function changes at every timestep hence, we expect to see different loss surfaces for every state input in the agent trajectory as opposed to a static loss surface as that expected for supervised learning.

4.3.2.3 *Visualizing the Actor Critic Loss*

Applying a similar process as that described in Section 4.3.2.2, yields a plot of the loss surface such as that shown in Figure 4.4.

4.4 VISUALIZING LEARNING ITERATES

A related idea is to project the intermediate function parameters onto this loss surface. This allows us to observe the optimization trajectory across time. Note, that for a fixed loss function, this would result in static loss contours, with the trajectory tracing a path across this loss surface. For an optimization problem such as that found in the actor critic framework, the optimization trajectory should be more complicated, due to changing loss surfaces. Experimental evidence suggests that the optimzation is taking place on a lower dimensional manifold, with the trajectory tracing out a path such as that shown in Figure 4.5.

4.5 FUTURE / ONGOING WORK

Since this research direction is still a work in progress, here are some questions that we've been thinking about, and we think we'd be able to answer through an approach such as this:

- What is the effect of the aforementioned stabilization tricks on the loss landscape?

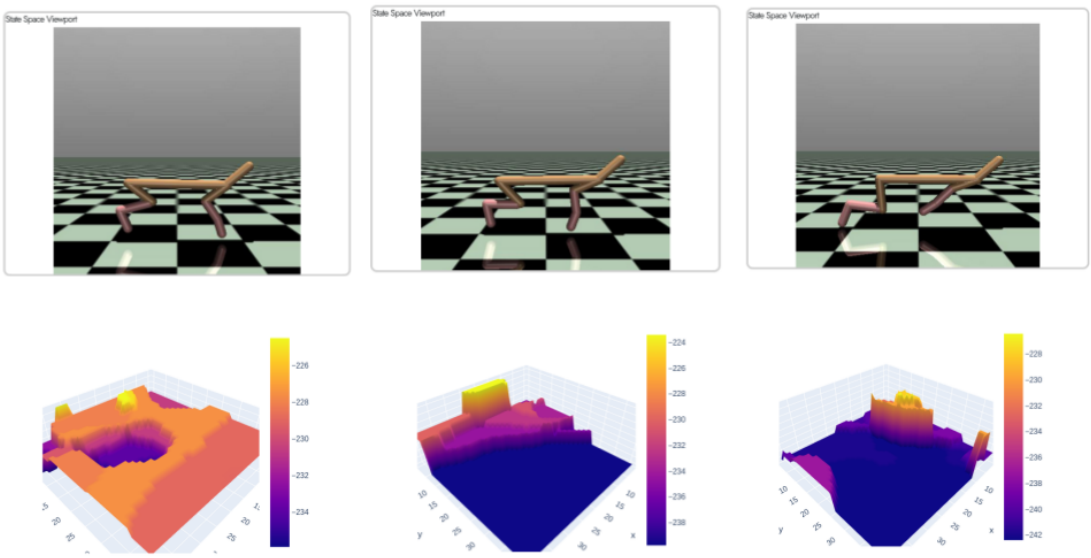
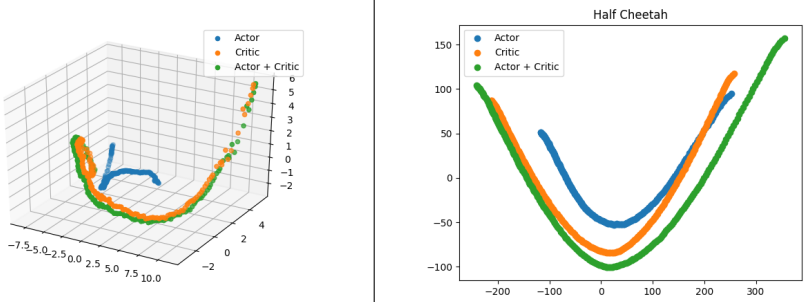


Figure 4.4: **Visualizing Actor Critic Loss Landscapes** These plots were made on consecutive timesteps in the agent trajectory. The drastic discontinuities in the landscape between similar states provides a visual representation of the difficulty of the optimization problem.

- Which of these heuristics end up having the same effect?
- Does actor critic optimization provably take place on a lower dimensional loss surface (as suggested by experimental evidence)?
- Are there ways to make the loss surface more invariant to state changes (or weakly variant)?



(a) **Plotting learning iterates (3D)**

(b) **Plotting learning iterates (2D)**

Figure 4.5: **Plotting learning iterates** Projecting the policy checkpoints to a lower dimensional surface. It is interesting to note the similarity between this result and that in [27], in the shapes of the overall learning trajectories.

5

CONCLUSION

In the preceding chapters, we looked at a novel system designed for the reinforcement learning framework. There are many distinctive features of this system, but a common thread is the ability to generate interactive visualizations for the RL framework. We believe this system could help RL researchers better understand RL algorithms and debug RL policies. Towards this end, we provided a walkthrough example of how this system could fit into the RL debugging workflow.

The system is designed with extensibility in mind, with Vizarel serving as a core around which to integrate different plugins. These plugins could take the form of simple visualizations, or be the result of more sustained research investigations. We provided an example of the latter, that we believe will help increase insight into actor-critic optimization.

We believe that the best research tools have a symbiotic relationship with the user. They help us interpret our results, but also guide us towards better questions to ask and hypotheses to confirm. We wonder what new questions a system like this would enable us to ask and what types of metadata we should start storing instead. This is but one example of what such a system design could look like, and what it could enable. We are uncertain whether Vizarel will be the last tool in this line of inquiry. However, we are certain that the best designs will continue to emerge from this process of exploration.

BIBLIOGRAPHY

[1] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." In: *arXiv:1603.04467 [cs]* (Mar. 2016). arXiv: 1603.04467.

[2] Tanmay Agarwal, Hitesh Arora, Tanvir Parhar, Shubhankar Deshpande, and Jeff Schneider. "Learning to Drive using Waypoints." en. In: *Workshop on Autonomous Driving, NeurIPS* (2019), p. 7.

[3] Dan Amir and Ofra Amir. "HIGHLIGHTS: Summarizing Agent Behavior to People." en. In: *17th International Conference on Autonomous Agents and Multiagent Systems* (2018), p. 9.

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete Problems in AI Safety." In: *arXiv:1606.06565 [cs]* (July 2016). arXiv: 1606.06565.

[5] Akanksha Atrey, Kaleigh Clary, and David Jensen. "Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning." In: *arXiv:1912.05743 [cs]* (Feb. 2020). arXiv: 1912.05743.

[6] Michel Beaudouin-Lafon. "Designing interaction, not interfaces." en. In: *Proceedings of the working conference on Advanced visual interfaces - AVI '04*. Gallipoli, Italy: ACM Press, 2004, p. 15. ISBN: 978-1-58113-867-2. DOI: 10.1145/989863.989865.

[7] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents." In: *Journal of Artificial Intelligence Research* 47 (June 2013). arXiv: 1207.4708, pp. 253–279. ISSN: 1076-9757. DOI: 10.1613/jair.3912.

[8] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babu ˇska. "Experience Selection in Deep Reinforcement Learning for Control." en. In: *Journal of Machine Learning Research* (2018), p. 56.

[9] Davide Calvaresi, Amro Najjar, Michael Schumacher, and Kary Främling, eds. *Explainable, Transparent Autonomous Agents and Multi-Agent Systems: First International Workshop, EXTRAAMAS 2019, Montreal, QC, Canada, May 13–14, 2019, Revised Selected Papers*. en. Vol. 11763. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-30390-7 978-3-030-30391-4. DOI: 10.1007/978-3-030-30391-4.

[10] Finale Doshi-Velez and Been Kim. "Towards A Rigorous Science of Interpretable Machine Learning." In: *arXiv:1702.08608 [cs, stat]* (Mar. 2017). arXiv: 1702.08608.

[11] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. "D4RL: Datasets for Deep Data-Driven Reinforcement Learning." In: *arXiv:2004.07219 [cs, stat]* (June 2020). arXiv: 2004.07219.

[12] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods." en. In: *arXiv:1802.09477 [cs, stat]* (Oct. 2018). arXiv: 1802.09477.

[13] Yosuke Fukuchi, Masahiko Osawa, Hiroshi Yamakawa, and Michita Imai. "Autonomous Self-Explanation of Behavior for Interactive Reinforcement Learning Agents." In: *Proceedings of the 5th International Conference on Human Agent Interaction* (Oct. 2017). arXiv: 1810.08811, pp. 97–101. DOI: 10.1145/3125739.3125746.

[14] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. "Qualitatively characterizing neural network optimization problems." en. In: *arXiv:1412.6544 [cs, stat]* (May 2015). arXiv: 1412.6544.

[15] Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. "Visualizing and Understanding Atari Agents." In: *arXiv:1711.00138 [cs]* (Sept. 2018). arXiv: 1711.00138.

[16] Hado V. Hasselt. "Double Q-learning." In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta. Curran Associates, Inc., 2010, pp. 2613–2621.

[17] Bradley Hayes and Julie A. Shah. "Improving Robot Controller Transparency Through Autonomous Policy Explanation." en. In: *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. Vienna Austria: ACM, Mar. 2017, pp. 303–312. ISBN: 978-1-4503-4336-7. DOI: 10.1145/2909824.3020233.

[18] Sandy H. Huang, David Held, Pieter Abbeel, and Anca D. Dragan. "Enabling Robots to Communicate their Objectives." In: *Robotics: Science and Systems XIII* (July 2017). arXiv: 1702.03465. DOI: 10.15607/RSS.2017.XIII.059.

[19] Peter J. Huber. "Projection Pursuit." en. In: *The Annals of Statistics* 13.2 (June 1985), pp. 435–475. ISSN: 0090-5364. DOI: 10.1214/aos/1176349519.

[20] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. "Explainable reinforcement learning via reward decomposition." In: *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*. 2019.

[21] Minsuk Kahng, Pierre Y. Andrews, Aditya Kalro, and Duen Horng Chau. "ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models." In: *arXiv:1704.01942 [cs, stat]* (Aug. 2017). arXiv: 1704.01942.

[22] Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. "Interactive optimization for steering machine classification." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Atlanta, Georgia, USA: Association for Computing Machinery, Apr. 2010, pp. 1343–1352. ISBN: 978-1-60558-929-9. DOI: 10.1145/1753326.1753529.

[23] Vijaymohan Konda. "Actor Critic Algorithms." en. In: (Mar. 2002).

[24] Josua Krause, Adam Perer, and Kenney Ng. "Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models." In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. San Jose, California, USA: Association for Computing Machinery, May 2016, pp. 5686–5697. ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858529.

[25] Oliver Kroemer, Scott Niekum, and George Konidaris. "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms." In: *arXiv:1907.03146 [cs]* (Nov. 2020). arXiv: 1907.03146.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." en. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539.

[27] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. "Visualizing the Loss Landscape of Neural Nets." en. In: *arXiv:1712.09913 [cs, stat]* (Nov. 2018). arXiv: 1712.09913.

[28] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE." In: 9 (2008), pp. 2579–2605.

[29] Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. "Distal Explanations for Model-free Explainable Reinforcement Learning." In: *arXiv:2001.10284 [cs]* (Sept. 2020). arXiv: 2001.10284.

[30] Sean McGregor, Hailey Buckingham, Thomas G. Dietterich, Rachel Houtman, Claire Montgomery, and Ronald Metoyer. "Facilitating testing and debugging of Markov Decision Processes with interactive visualization." In: *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Atlanta, GA: IEEE, Oct. 2015, pp. 281–282. ISBN: 978-1-4673-7457-6. DOI: 10.1109/VLHCC.2015.7357227.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning." In: *arXiv:1312.5602 [cs]* (Dec. 2013). arXiv: 1312.5602.

[32] Emre O. Neftci and Bruno B. Averbeck. "Reinforcement learning in artificial and biological systems." en. In: *Nature Machine Intelligence* 1.3 (Mar. 2019), pp. 133–143. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0025-4.

[33] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization." en. In: *Distill* 2.11 (Nov. 2017), e7. ISSN: 2476-0757. DOI: 10.23915/distill.00007.

[34] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. "The Building Blocks of Interpretability." en. In: *Distill* 3.3 (Mar. 2018), e10. ISSN: 2476-0757. DOI: 10.23915/distill.00010.

[35] OpenAI et al. "Dota 2 with Large Scale Deep Reinforcement Learning." In: *arXiv:1912.06680 [cs, stat]* (Dec. 2019). arXiv: 1912.06680.

[36] Erika Puiutta and Eric MSP Veith. "Explainable Reinforcement Learning: A Survey." In: *arXiv:2005.06247 [cs, stat]* (May 2020). arXiv: 2005.06247.

[37] Nikaash Puri, Sukriti Verma, Piyush Gupta, Dhruv Kayastha, Shripad Deshmukh, Balaji Krishnamurthy, and Sameer Singh. "Explain Your Move: Understanding Agent Actions Using Specific and Relevant Feature Attribution." In: *arXiv:1912.12191 [cs]* (Apr. 2020). arXiv: 1912.12191.

[38] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. eng. Wiley series in probability and statistics. OCLC: 254152847. Hoboken, NJ: Wiley-Interscience, 2005. ISBN: 978-0-471-72782-8.

[39] Siddharth Reddy, Anca D. Dragan, Sergey Levine, Shane Legg, and Jan Leike. "Learning Human Objectives by Evaluating Hypothetical Behavior." In: *arXiv:1912.05652 [cs, stat]* (Dec. 2019). arXiv: 1912.05652.

[40] Siddharth Reddy, Sergey Levine, and Anca Dragan. "Accelerating Human Learning with Deep Reinforcement Learning." en. In: (2017), p. 9.

[41] Joshua Romoff, Peter Henderson, Alexandre Piché, Vincent Francois-Lavet, and Joelle Pineau. "Reward Estimation for Variance Reduction in Deep Reinforcement Learning." In: *arXiv:1805.03359 [cs, stat]* (Nov. 2018). arXiv: 1805.03359.

[42] Christian Rupprecht, Cyril Ibrahim, and Christopher J. Pal. "Finding and Visualizing Weaknesses of Deep Reinforcement Learning Agents." In: *arXiv:1904.01318 [cs]* (Apr. 2019). arXiv: 1904.01318.

[43] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. "Vega-Lite: A Grammar of Interactive Graphics." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 341–350. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2599030.

[44] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized Experience Replay." In: *arXiv:1511.05952 [cs]* (Feb. 2016). arXiv: 1511.05952.

[45] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation." In: *arXiv:1506.02438 [cs]* (Oct. 2018). arXiv: 1506.02438.

[46] Pedro Sequeira and Melinda Gervasio. "Interestingness Elements for Explainable Reinforcement Learning: Understanding Agents' Capabilities and Limitations." In: *Artificial Intelligence* 288 (Nov. 2020). arXiv: 1912.09007, p. 103367. ISSN: 00043702. DOI: 10.1016/j.artint.2020.103367.

[47] David Silver et al. "Mastering the game of Go with deep neural networks and tree search." en. In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature16961.

[48] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps." In: *arXiv:1312.6034 [cs]* (Apr. 2014). arXiv: 1312.6034.

[49] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M. Rush. "LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks." In: *arXiv:1606.07461 [cs]* (Oct. 2017). arXiv: 1606.07461.

[50] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[51] *TensorBoard: TensorFlow's Visualization Toolkit*. 2020. URL: https://github.com/tensorflow/tensorboard.

[52] Richard H. Thaler and Cass R. Sunstein. *Nudge: improving decisions about health, wealth, and happiness*. Rev. and expanded ed. New York: Penguin Books, 2009. ISBN: 978-0-14-311526-7.

[53] Sebastian Thrun and Anton Schwartz. "Issues in Using Function Approximation for Reinforcement Learning." en. In: *Proceedings of the Fourth Connectionist Models Summer School* (1993), p. 9.

[54] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." en. In: *Nature* 575.7782 (Nov. 2019), pp. 350–354. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-1724-z.

[55] Jasper van der Waa, Jurriaan van Diggelen, Karel van den Bosch, and Mark Neerincx. "Contrastive Explanations for Reinforcement Learning in terms of Expected Consequences." In: *arXiv:1807.08706 [cs, stat]* (July 2018). arXiv: 1807.08706.

[56] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B. Viegas, and Martin Wattenberg. "Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow." In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 1–12. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744878.

[57] Ji Soo Yi, Youn ah Kang, John Stasko, and J.A. Jacko. "Toward a Deeper Understanding of the Role of Interaction in Information Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1224–1231. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2007.70515.

[58] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. "Understanding Neural Networks Through Deep Visualization." en. In: *ICML Deep Learning Workshop* (2015), p. 12.

[59] Chao Yu, Jiming Liu, and Shamim Nemati. "Reinforcement Learning in Healthcare: A Survey." In: *arXiv:1908.08796 [cs]* (Apr. 2020). arXiv: 1908.08796.

[60] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." In: *arXiv:1311.2901 [cs]* (Nov. 2013). arXiv: 1311.2901.

[61] Shangtong Zhang and Richard S. Sutton. "A Deeper Look at Experience Replay." In: *arXiv:1712.01275 [cs]* (Apr. 2018). arXiv: 1712.01275.

DECLARATION

Pittsburgh, PA, United States of America, December 2020

Shuby V. Deshpande