INFLUENCING EXPLORATION IN ACTOR-CRITIC REINFORCEMENT LEARNING ALGORITHMS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Andrew Richard Gough

June 2018

COMMITTEE MEMBERSHIP

TITLE:	Influencing Exploration in Actor-Critic Reinforcement Learning Algorithms
AUTHOR:	Andrew Richard Gough
DATE SUBMITTED:	June 2018
COMMITTEE CHAIR:	Franz Kurfess, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	John Seng, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Maria Pantoja, Ph.D. Professor of Computer Science

ABSTRACT

Influencing Exploration in Actor-Critic Reinforcement Learning Algorithms

Andrew Richard Gough

Reinforcement Learning (RL) is a subset of machine learning primarily concerned with goal-directed learning and optimal decision making. RL agents learn based on a reward signal discovered from trial and error in complex, uncertain environments with the goal of maximizing positive reward signals. RL approaches need to scale up as they are applied to more complex environments with extremely large state spaces. Inefficient exploration methods cannot sufficiently explore complex environments in a reasonable amount of time, and optimal policies will be unrealized resulting in RL agents failing to solve an environment

This thesis proposes a novel variant of the Actor-Advantage Critic (A2C) algorithm. The variant is validated against two state-of-the-art RL algorithms, Deep Q-Network (DQN) and A2C, across six Atari 2600 games of varying difficulty. The experimental results are competitive with state-of-the-art and achieve lower variance and quicker learning speed. Additionally, the thesis introduces a metric to objectively quantify the difficulty of any Markovian environment with respect to the exploratory capacity of RL agents.

ACKNOWLEDGMENTS

Thanks to:

- Dr. Kurfess, a truly special professor, advisor, and friend. Without his guidance, passion for AI, and great humor this thesis work would not have been possible.
- Dr. Pantoja and Dr. Seng for their advice and direction, especially regarding my ignorance of hardware requirements in the Fall.
- The Cal Poly Computer Science Department for enabling me to do the best work of my life and transform as a person.
- Tedd Akdag, the sys-admin for our department who I credit for my newly acquired system administrator skills.
- Julie Workman, whose encouragement and love for programming kept me in the major freshman year.
- My girlfriend Camille Posard, for her unwavering love, support, and gracefully handling the ups and downs of (thesis) life.
- Last but not least, my parents, family, and friends; I have endless gratitude for the village who raised me and built the foundation upon which I stand.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

A precursor to general artificial intelligence is the construction of systems that are equipped with generalized learning mechanisms and the capability to develop optimal decision making strategies. Reinforcement Learning (RL) is a subset of machine learning primarily concerned with goal-directed learning and optimal decision making [30, 39]. The field is largely inspired by psychology and neuroscience where it is believed that biological learning is positively reinforced by rewards of dopamine [10] in the brain, or negatively reinforced by sending painful signals to the brain [35]. Therefore, RL agents learn based on a reward signal discovered from trial and error in complex, uncertain environments with the goal of maximizing positive reward signals.

Often times, a good example can help build the foundations of a solid mental model. For an intuitive and familiar example, consider Phil, a man about to prepare breakfast. Closely examined, even this apparently mundane activity reveals a complex web of conditional behavior and interlocking goal-subgoal relationships: walking to the cupboard, opening it, selecting a cereal box, then reaching for, grasping, and retrieving the box. Other complex, tuned, interactive sequences of behavior are required to obtain a bowl, spoon, and milk jug. Each step involves a series of eye movements to obtain information and to guide reaching and locomotion. Rapid judgments are continually made about how to carry the objects or whether it is better to ferry some of them to the dining table before obtaining others. Each step is guided by goals, such as grasping a spoon or getting to the refrigerator, and is in service of other goals, such as having the spoon to eat with once the cereal is prepared and ultimately obtaining nourishment. Whether he is aware of it or not, Phil is accessing information about the state of his body that determines his nutritional needs, level of hunger, and food preferences.

Regarding optimal decision making, the human brain performs fairly well despite handling a firehose of sensorimotor information. These RL agent brains can theoretically consider all of this sensorimotor information with ease and flawlessly execute the next ideal action without hesitation. Where these agents lack is the ability to efficiently explore unknown state spaces and factor in long-term vs. short-term rewards. State-of-the-art approaches stand no chance against increasingly complex environments if they are utilizing inefficient exploration methods [6].

Historically, RL required a lot of abstraction and simplification of environments in order for an agent to engage with the environment in a reasonable amount of time. Due to these simplified, dumbed down environments, RL was not considered a practical method to solving complex optimization problems. Thanks to recent advances in computer vision and deep neural networks as function approximators, end-to-end model-free RL based on purely visual input has been realized [29, 30]. End-to-end reinforcement learning is the process of going from direct observation of an environment, to direct predictions of optimal actions, and looping until the problem is solved or the max time steps are achieved, with minimal pre and post processing of the problem. Model free means that the agent has no preconception of the dynamics or physics of the environment it is operating in. If we were to compare end-to-end RL to human flight with the Wright Brothers, the schematic for the plane was proved by Deepmind [29] in 2013 introducing a cutting edge RL algorithm utilizing a convolutional neural network architecture to process and play Atari games purely from raw visual data. It could take off and get some air down the runway, but for the most part would come bumping back down after a brief flight.

By 2015 super human performance [30] was achieved across a majority of the Atari Learning Environment (ALE) [7] games. This new, enhanced plane design could hold its own and lift off the runway in many different situations, varying weather / varying environments proving that an end-to-end RL agent can compete with humans in increasingly complex environments.

1.2 Problem

The fundamental trade-off between exploration and exploitation is arguably the most important problem within the field, demanding research for efficient exploration capabilities. When an agent learns to control a new, unknown environment, two opposing objectives need to be combined. First, in order to identify a (sub-)optimal controller, the environment must be sufficiently explored. The second objective is utilizing the experience gained during learning to maximize potential future reward, or minimize the cost of learning (in terms of negative reward). Thus for efficient learning, actions should be generated such that a) the environment is explored and b) pain is avoided. This quintessential balance between exploration and exploitation demands efficient exploration capabilities, maximizing the effect of learning while minimizing the costs of exploration and learning time.

Despite major advances in the field, exploration strategies have had little innovation as strategies developed [42] in the early 1990s are still considered state-of-the-art in production code [13]. For environments where the state space is relatively tame and rewards are somewhat constant, undirected exploration methods are surprisingly effective, but can be improved upon. RL approaches need to scale up as they are being applied to more complex environments with extremely large state spaces [45]. Inefficient exploration methods cannot sufficiently explore these environments in a reasonable amount of time, and optimal policies will be unrealized resulting in RL agents failing to solve an environment or converging on local optima.

1.3 Contributions

My contributions include:

- A comprehensive examination of state-of-the-art RL algorithms, Deep Q-Network (DQN) and Actor-Advantage Critic (A2C) in a modern Atari 2600 reinforcement learning benchmark.
- Design, implementation, and validation of a novel variant of the A2C algorithm.
- A proposed Exploration Difficulty Metric (EDM) to objectively quantify the difficulty of RL environments with respect to the exploratory capacity of RL agents.

1.4 Thesis Outline

This document will first introduce the topic, problem, and contributions of this thesis project in Chapter 1. Then it will cover the relevant background information in Chapter 2 to help readers become familiar with RL in an accelerated fashion. In Chapter 3, related works will be introduced and I will distinguish between the similarities and differences of their works and mine. Chapter 4 will explain design decisions and describe the specific software implementation details of a novel modification to A2C while also introducing the Exploration Difficulty Metric. Chapter 5 explains how experiments are conducted, what metrics are considered, and provides analysis to determine the value of my work. Finally, future work and a conclusion analyzing the impact of my work are discussed in Chapter 6.

Chapter 2

BACKGROUND

2.1 Markov Decision Process

In order for an autonomous agent to learn, it must act on its own accord and understand the consequences of its actions. For this to be possible, it must be able to sense the state of its environment, and affect that environment through a set of actions. Goals or rewards provide context to these actions from which to learn and it is the relationship between actions and rewards which must be learned. Sutton and Barto [39] claim that a Markov Decision Process (MDP) must be able to capture and represent three aspects of an environment - state, action, and reward. Formally, an MDP is a mathematical framework for modeling any sequential decision making process [3]. In this framework, actions influence not just immediate rewards, but also subsequent states and future rewards. Reinforcement learning utilizes this framework to define the interactions between a learning agent and its environment.

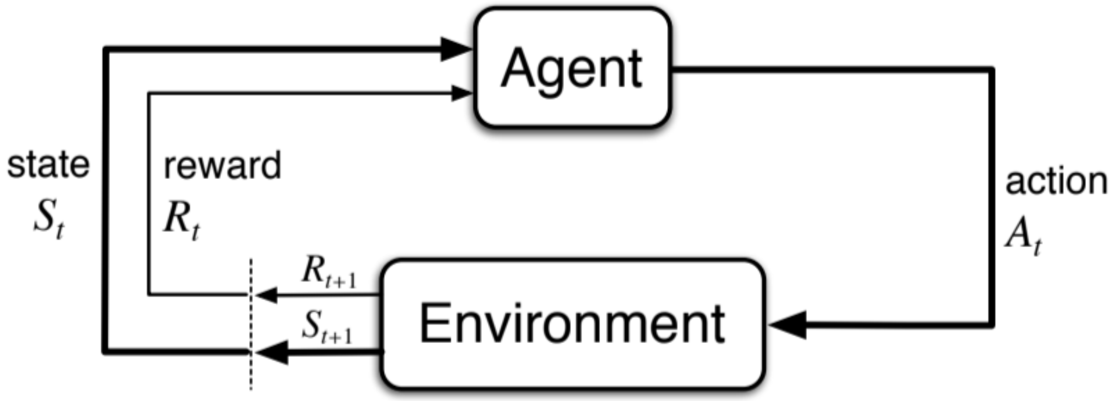


Figure 2.1: Diagram of a Markov Decision Process

Finite MDP's model stochastic, discrete-time and finite action space control problems [8]. An MDP is a 4-tuple entity represented below:

$$\mathcal{M} = \langle A, S, P, R(s, a) \rangle$$

Where A is a set of legal actions available to the agent. This set is assumed to be finite, and that all actions are available to the agent at each state. P is a function that defines the probability of transitioning to state s0 after action a is taken when the agent is in state s. Because of the Markov property, the probability of a transition to state s0 only depends on the prior state and action made. The function R(s, a) determines the probability of receiving reward r after choosing action a in state s. More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3 ... At each time step t, the agent receives some representation of the environment state, St S, and on that basis selects an action, At A(s). One time step later, as a consequence of the selected action, the agent receives a numerical reward Rt+1 R and finds itself in a new state St+1. The agent's interactions with the MDP give rise to a sequence or trajectory that begins like this: S0, A0, R1, S1, A1, R2, S2, A2, R3,.... MDP's are intended to represent the essential features of any artificial intelligence problem. For the rest of this paper each environment will be modeled as a finite MDP.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a computational approach to understanding and automating goal-directed learning and decision making [39]. To understand RL it is helpful to understand the fields that study optimal decision making and contribute to solving it. The following sections will discuss the origins of RL and formally describe the framework in which RL solves Markov Decision Processes.

2.2.1 Origins of Reinforcement Learning

RL is an attempt to solve the science of optimal decision making. Unsurprisingly, many fields of science contribute to the study of optimal decision making and tackle the same problems that RL hopes to solve. In life, there are often many solutions to a problem, the challenging task is finding the optimal solution to a problem. General engineering for centuries has studied optimal control policies for control of locomotion systems and robotics. Neuroscientists study optimal decision making through studying the brain and the examination of chemical interactions that form reward systems driving human motivation. Psychology comes into play with the idea of classical conditioning, Pavlov's dogs, and the idea that reinforcing behaviors drives learning in biological systems. Mathematics influences the field in terms of optimization and operations research. Finally economics touches on the topic of decision making with game theory, decision theory, and bounded rationality. RL researchers examine all of these fields and take inspiration to implementation when designing new algorithms, with the ultimate goal of solving intelligence.

2.2.2 Reinforcement Learning Framework

There are several major distinctions between reinforcement learning and other machine learning paradigms. First, most successful machine learning applications to date have required large amounts of hand-labeled training data. Second is that there is no supervisor, only a reward signal. RL algorithms must be able to learn from a scalar reward signal that is frequently sparse and noisy. Third, feedback is not guaranteed to be instantaneous and is often delayed. Your data set is not a fixed entity divided into training and test, an RL dataset is a dynamic system that your agent is creating through interaction and those actions influence the data it receives. The data is sequential in nature. Additionally, a daunting challenge for RL agents to learn a successful control policy is that they are often interacting with stochastic problems and learning non-stationary probability distributions.

Components of an RL System

Beyond the agent and the environment definitions of the Markov Decision Process, there are four main sub-elements of a reinforcement learning system.

- Policy: an agent's behavior function, how it selects actions to take
- Reward Signal: a scalar feedback signal from the environment indicating how rewarding a given state is
- Value Function: an agent's estimate of how good a state or an action is to take
- Model: an agent's internal representation of an environment; this component is optional, the algorithms examined in this thesis do not contain a model and are referred to as model-free RL algorithms

A *policy* defines the learning agent's way of behaving at a given moment. Essentially a policy is a mapping from perceived states of the environment to actions to be taken from that state. It corresponds to what in psychology would be called a set of stimulus-response rules or associations. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A *reward signal* defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single, scalar number called the *reward*. The agent's primary objective is to maximize net reward it receives over the course of its lifetime. The reward signal defines what is considered good and bad in a given environment. In a biological system, we might think of rewards as similarities to pleasure and pain. They are the immediate and defining features of the problem faced by the agent. The reward signal dictates how an agent will learn and adapt to a problem. It is the primary basis for altering the policy, for example if an action selected by the policy is followed by low or negative reward, then the policy may change to select a different action given that state in the future. When designing an environment for a reinforcement learning agent, it is particularly challenging to engineer the reward system and reward signals in a way such that the agent can generalize as much as possible and not follow a human-biased strategy for solving the environment. In general, reward signals are stochastic functions of the state of the environment and the set of actions available to the agent.

Where a *reward signal* dictates what is good in the immediate sense, the agent's *value function* estimates what is good in the long run. Essentially, the *value* of a state is the total amount of reward an agent can expect to accumulate in the future starting with the current state. Whereas rewards determine the immediate desirability of environmental states, values indicate a long-term prediction of future desirability of states from the current state. For example, a given state may always yield a low immediate reward but still have a high value because it is generally followed by other states that are associated with high rewards. Or the reverse could be true, indicating that an agent is on a possibly harmful trajectory given the current state. To circle back to human psychology, positive rewards are comparable to immediate pleasure and negative rewards are comparable to immediate pain. Whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are with our current state. Without rewards there could be no values, and the sole purpose of estimating values and learning a value function is to obtain more reward. However, value estimates are what most heavily influence strategy and decision making. Policy is often based on value judgments, it is better to choose actions that bring about the highest value states, not the highest reward, because these actions often generate the largest amount of reward for an agent in the long run. As you can imagine, it is much harder to determine values than it is to determine rewards. Rewards are given directly to the agent from an environment where values must be estimated and updated from sequences of observations that an agent makes over the duration of its lifetime. The most important component of practically all reinforcement learning algorithms is the efficient learning and estimation of its value function.

Finally, an element of some reinforcement learning systems is a *model* of the environment. The model is an internal representation of the environment built by the agent that mimics the behavior of an environment. More generally, it enables inferences to be made about how the environment will behave. For example, given a state and action, the model attempts to predict the future state and future reward resulting in the execution of the action. Models are used to help in future planning, or how the agent will decide on a course of action by predicting future states before they are actually realized. The class of reinforcement learning methods that use models are called *model-based* methods, as opposed to simpler, lighter weight *model-free* methods that explicitly learn through trial-and-error. Reinforcement learning methods vary from low-level trial-and-error learning agents to high-level predictive planning.

2.3 Classic Reinforcement Learning Approaches

Ultimately, all reinforcement learning algorithms can be categorized as Value-Based, Policy-Based, or a combination of the two, in the family of Actor-Critic methods.

2.3.1 Overview

Value based approaches attempt to predict and learn the value of being in a certain state which in turn influences the policy towards high value states or actions. Policy based approaches attempt to directly predict and learn the optimal policy, or the optimal action to take given a state. Both approaches have the same desired end result, providing an agent that can optimally perform within the environment it was trained in.

Matrix of RL Components and Methods			
	Value-Based	Policy-Based	Actor-Critic
Policy	Epsilon-greedy	Learned	Learned
Value Function	Learned	None	Learned
Model	Model-free	Model-free	Model-free

Table 2.1: Description of RL Methods

2.3.2 Value Based Methods

At a high level, value-based RL methods aim to learn an accurate value function for a given environment. Two prominent and relevant methods detailed below, Temporal Difference Learning and Q-Learning both rely on predictions of value to solve problems. Both methods have demonstrated success in various applications and prove that agents based solely on the predictions of future value are viable.

Temporal Difference Learning

In 1988 Richard Sutton, the grandfather of RL, introduced Temporal Difference (TD) learning [38] as a form for machine learning agents to solve complex sequential tasks via learning a value function. Temporal difference learning utilizes a function estimation *V(S)* which takes as a parameter the current state, and outputs the value of that state. This function is updated over time and uses TD error to update its predictions. TD learning differs from Monte-Carlo and supervised learning methods in that it 'learns' or 'bootstraps' itself from every single prediction made at each time-step. For example, in TD-Gammon [41] a neural network was used to predict the value function, or probability of winning in each state and selected actions based off that prediction, this implementation of TD learning was competitive with the best human players in the world. Figure 2.2 describes the neural network architecture used in TD-Gammon.

TD learning is only relevant in multi-step prediction problems, not for use in one step or classification problems. You may think this is a limitation, but most of everything we do as humans is a multi-step prediction and can be modeled as a Markov Decision Process. TD learning makes repeated predictions about a long term outcome, you can use TD learning in long term problems for example in predicting stock prices, considering every day as each event happens, updating your predictions daily. Predictions are made against discounted views of the future. Richard Sutton believes that learning to predict is the only scalable type of learning, this type of learning for generalized multi-step prediction problems may also be key to understanding how humans and biological systems learn.

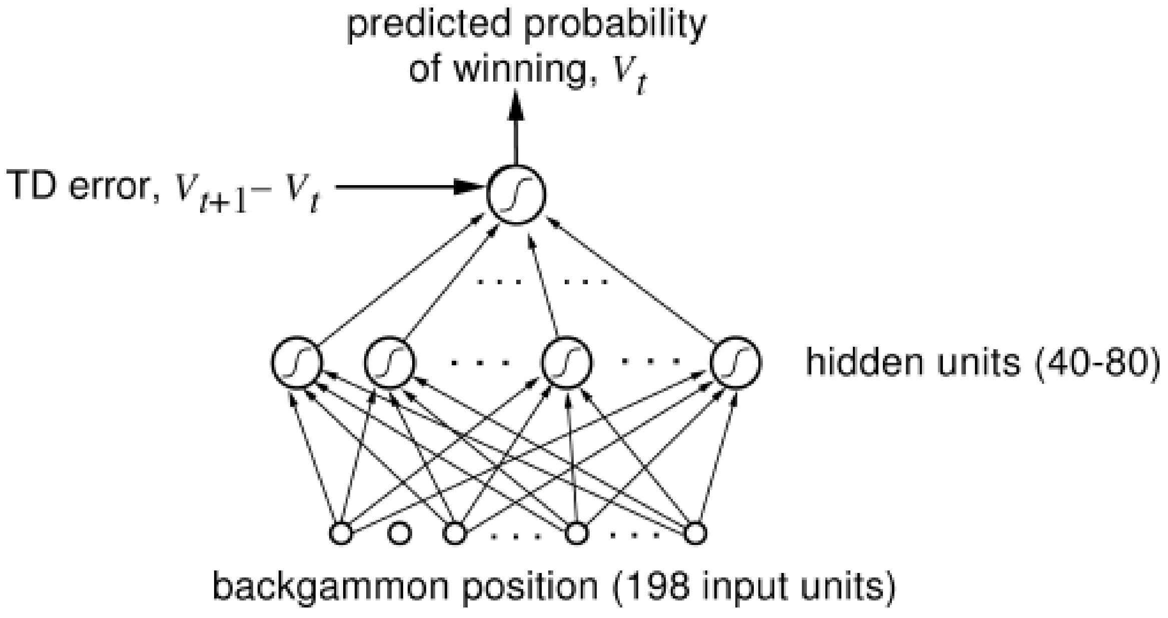

predicted probability of winning, V_t
TD error, $V_{t+1} - V_t$
hidden units (40-80)
backgammon position (198 input units)

Figure 2.2: Neural network architecture used in TD-Gammon

Q-Learning

In 1992 Watkins et al. [46] introduced Q-Learning. A form of TD learning, that extended the value function from *V(S)* to *Q(S, A)*, for a given policy. Where S is the current state, and A is a possible action, the result of the Q(S,A) (Q-value call) for that specific state-action combo represents the estimated value yielded from taking that action from that state. The simplest implementation of Q-learning is in a tabular form, having a Q-table hold every state-action combo possible in the environment, and over time this table is updated to become more accurate.

The Q-learning algorithm can be summarized as

```
initialize  $s_0$
for         $t = 1, 2, 3 \ldots$
            choose an action $a_t$, let's say $\epsilon$-greedy w.r.t. $Q$
            execute $a_t$
            $Q(s_t, a_t) = (1 - \alpha_t)Q(s_t, a_t) + \alpha_t \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a)\right]$    (2.1)
```

The α value is the discount on future rewards, after executing an action, the algorithm updates the Q-table and uses the α value to very slightly discount future reward versus immediate reward, and have that change over time.

Instead of using solely a state-value estimation to guide a policy, Q-learning utilizes state-action-value estimations, to determine which possible action will yield the most long term value given the current state. Q-learning also updates itself differently than TD-learning in that the update is based off *argmax(Q(s',a')* where *a'* is iterated through every action of the legal set of actions possible. So the Q-value of the previous state-action combo is updated based off the maximum possible Q-value of the new state. Q-learning is an off-policy learning method, whereas traditional TD-learning is an on-policy learning method.

2.3.3 Policy Based Methods

Prior to policy based methods we focused on value based methods. These value based methods, TD-learning and Q-learning approximate the value of being in a certain state (TD-learning) or estimates the action-value of taking an action and using those values to determine action selection (Q-learning). On the other hand, policy based methods directly learn a policy [40], which is a probability distribution over all possible actions with respect to the current state. This enables algorithms to select actions directly without referencing an estimated value function.

Concepts

Policy based methods have their advantages and disadvantages. Situationally, policy based approaches may be more computationally efficient; for example in Pong it is easier to know if the paddle should move up or down based on the state or knowing the ball's position rather than computing the q-values of going up and down based on that state. Another advantage is that policy based methods have stronger convergence properties [40]. Meaning that they are guaranteed to converge at least to a local optimum. The nature of selecting actions based on a probability distribution is highly effective in high-dimensional or continuous action spaces where it could be computationally intensive or intractable to compute every q-value for continuous actions such as angular positioning vs. discrete action possibilities (ex. up, down, left, right). Policy methods can also learn stochastic policies more easily due to the learned policy probability distribution. Some disadvantages are that policy gradient methods take small, smooth steps towards a better policy, so they generally require much more data to converge on an optimal policy. Evaluating a policy is typically sample inefficient and has higher variance than value based methods.

Policy Gradient

In 2000, Richard Sutton returned once more with a paper detailing the first successful policy based reinforcement learning algorithm, the Policy Gradient. Sutton et al. [40] proposed an alternative to value based methods in which the policy is explicitly represented by its own function approximator. This approximator is independent of the value function, and is updated according to the gradient of expected reward with respect to the policy parameters. For example, let θ denote the vector of policy parameters and let ρ denote the performance of the corresponding policy (e.g., the average reward per step). Then in the *policy gradient* approach, the policy parameters are updated approximately proportional to the gradient:

$$\Delta\theta \approx \alpha \partial\rho / \partial\theta$$

Where α is a positive-definite step size. If the above can be achieved then θ can usually be assured to converge to a locally optimal policy in the performance measure ρ. Unlike in value-based approaches, small changes here in θ can only cause small changes in the policy and in the state-visitation distribution. This paper foreshadows and predicts the marriage of neural network function approximation and reinforcement learning.

2.3.4 Actor-Critic Methods

Actor-critic based algorithms [21] learn both policies and value functions. The *actor* is the component that learns the policy function, and the *critic* is the component that learns the value function. The critic uses a TD-learning algorithm to learn the state-value function for the actor's current policy. The value function allows the critic to critique the actor's action choices by sending δ (TD-errors) to the actor. The TD-error, δ can be considered a surprise measure of the algorithm, how off its predictions end up being from the truth. A positive δ means that the action was good, because it led to a state with a better-than-expected value. A negative δ means that the action was bad because it led to a state with worse-than-expected value. Based on this feedback, the actor can continually update its policy more effectively than with just policy based information.

By combining both value and policy functions the actor-critic methods get the best of both worlds. The learned value function, the *critic* helps update the actor's policy parameters in a direction of continuous performance improvement. While the learned policy function, the *actor* is a policy-gradient based method, and brings with it the desirable convergence properties of policy based methods. The two components combined deliver faster convergence due to variance reduction when compared to purely policy based methods.

2.4 Neural Networks

The following sections will discuss neural networks and the role they play in powering deep reinforcement learning.

2.4.1 Origins

Despite renewed popularity, neural networks are not new ideas. The concepts date back to the early 1940s [26, 33]. Recent advances in hardware and data availability have enabled neural nets to reach their full potential and produce state of the art results in fields such as computer vision [22, 27, 36] speech recognition [12, 16] and machine translation [5, 11]. Since then, neural net applications have continued to advance the state of the art in fields ranging from reinforcement learning [29, 30] to data center cooling optimization [15] leading to a 40% reduction in energy bills.

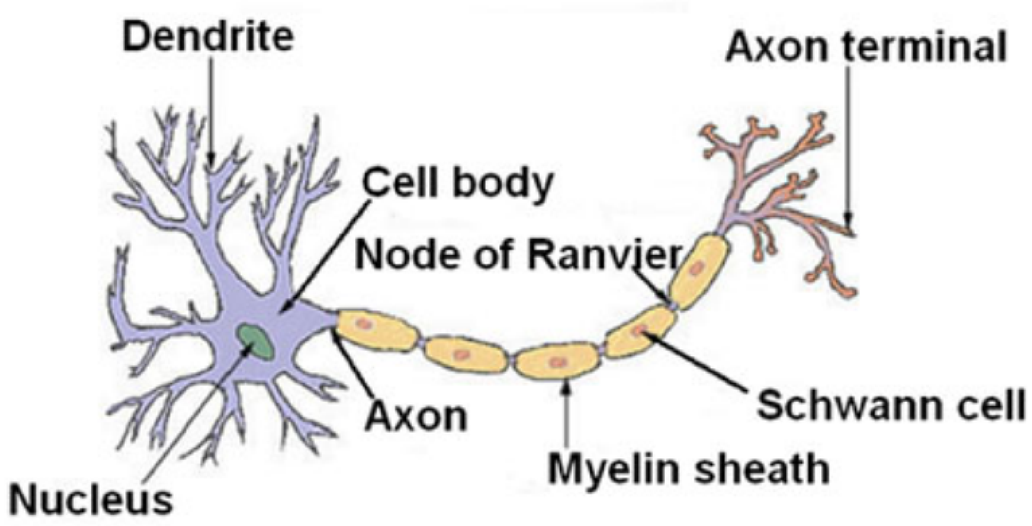


Figure 2.3: The structure of a biological neuron

Artificial neural networks are loosely inspired by our own biological systems [18]. Neurons, the main components of our central nervous systems, are cells specialized for processing and transmitting information utilizing electrical and chemical signals.

Neurons typically have a cell body, dendrites, and a single axon. Dendrites are the structures that branch from the cell body to receive input from other neurons. The axon is a channel that carries the neuron's output to other neurons. The neuron's output is dictated by a sequence of electrical pulses called action potentials that flow across the axon. These action potentials can also be referred to as activations. The synapse is a structure located at the end of an axon that mediates the communication of one neuron to another. The synapses "fire" when the action potential along the axon is large enough.

2.4.2 Usage and Intuition

Intuitively, one can think of neural networks as generalized "black box" function approximators. They can make simple predictions, like whether or not a hot dog is in an image, or very complex predictions like predicting optimal policies for deep reinforcement learning agents.

2.4.3 Concepts

The artificial neuron is the basic building block of a neural network. Figure 2.4 displays an artificial neuron. They represent a single computational element that takes in inputs, computes a weighted sum of the inputs, and passes that weighted sum to an activation function. Dependent on the activation function the neuron will output some value, often times a step function or a sigmoid function will be used to normalize the output between [-1, 1].

Networks are often represented by at least three layers. First is the input layer, where the input is fed into the network. The last layer is the output layer, where the final result or prediction of the network lies. In between the input and output layers can be any number of hidden layers, that serve as transformations in the

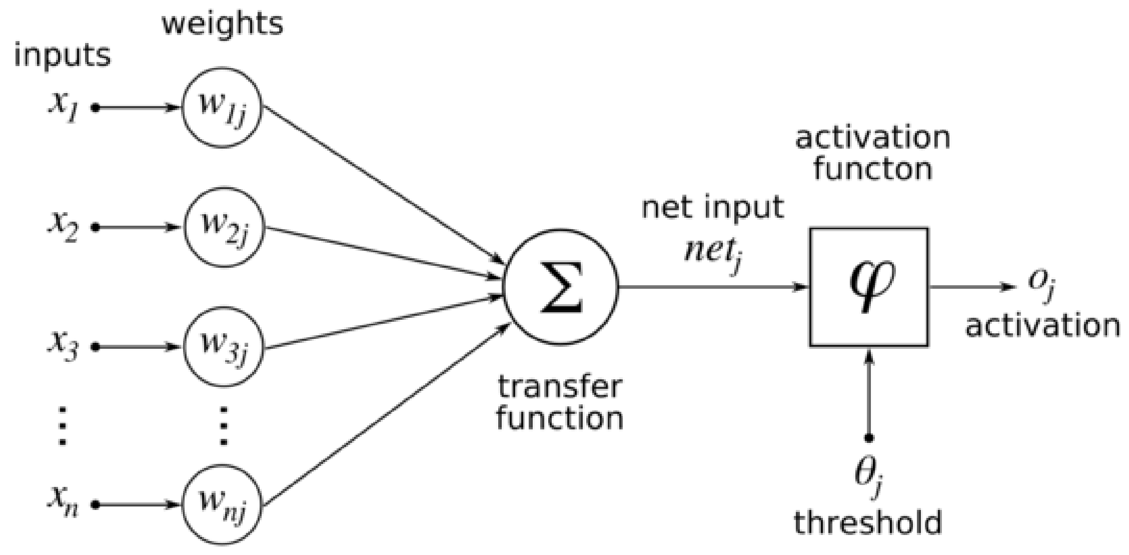


Figure 2.4: The artificial neuron model

computational graph of the network. If the data flows from input to the output layer, it is considered a feed-forward neural network. Figure 2.5 shows a multi-layered feed-forward network. If there are nodes that send data to previous layers, it is considered a recursive neural network.

Critical components of any neural network are the loss function, and back propagation algorithm. The loss function computes how 'off' a prediction was from some oracle, or truth value, or what the true prediction should have been. The back propagation algorithm updates all the internal weights of the neural network with the goal of reducing future loss, thus improving accuracy of the network. Back propagation algorithms are generally based on gradient descent methods. Gradient descent is easily understood in 2 dimensions, it utilizes a normal derivative to find the steepest direction to descend in order to achieve a local min or max given the parameters. In 3 dimensions, the goal of training a network is to find the minimum error with respect to the 3 dimensions. You can visualize the state space as a surface in which the neural network is trying to alter the three parameters to find a local min or max in the error surface. In higher dimensional space, multi-variable calculus takes the

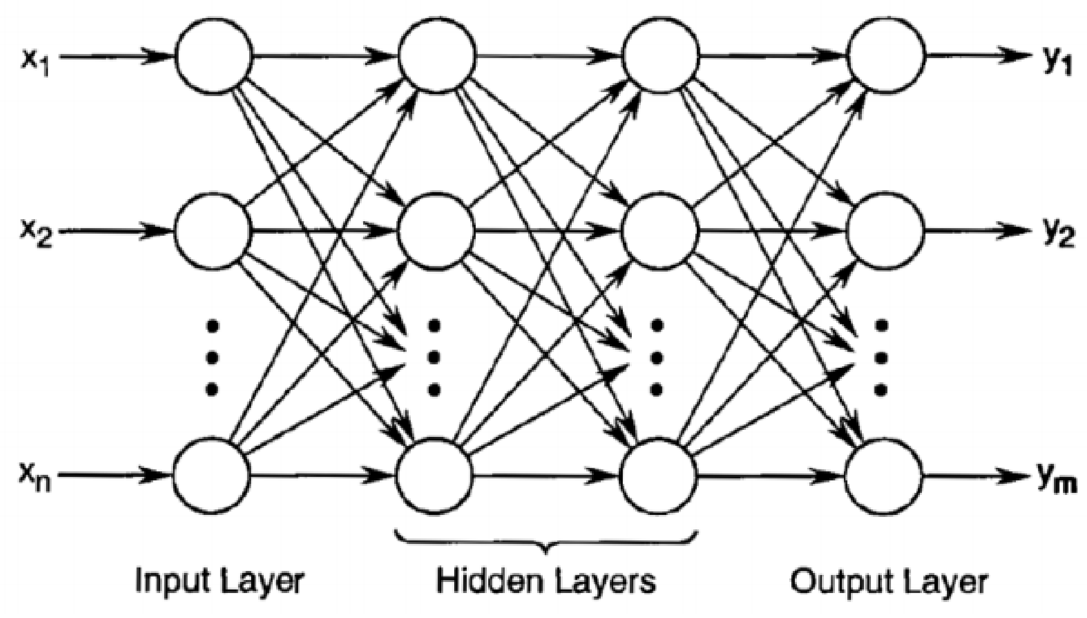


Figure 2.5: Typical multi-layered feed-forward network [18]

derivative of a vector (a matrix filled with the partial derivatives of that vector) to compute the gradient of that vector. And then uses that gradient to steer potentially hundreds of thousands of parameters all in a desired direction. Without updating the internal weights of the network, a neural net would be nothing more than a fancy way to perform matrix multiplications, and no *learning* would occur.

2.5 Deep Reinforcement Learning with Atari

The following sections discuss how deep learning and the Atari 2600 console helped reinforcement learning overcome serious limitations and elevated the field with super-human level performance in a complex environment.

2.5.1 Motivation and Deep Learning

A long-standing challenge for RL researchers is scaling up methods to solve real-world problems. Two major obstacles stand in the way, first is that the real world

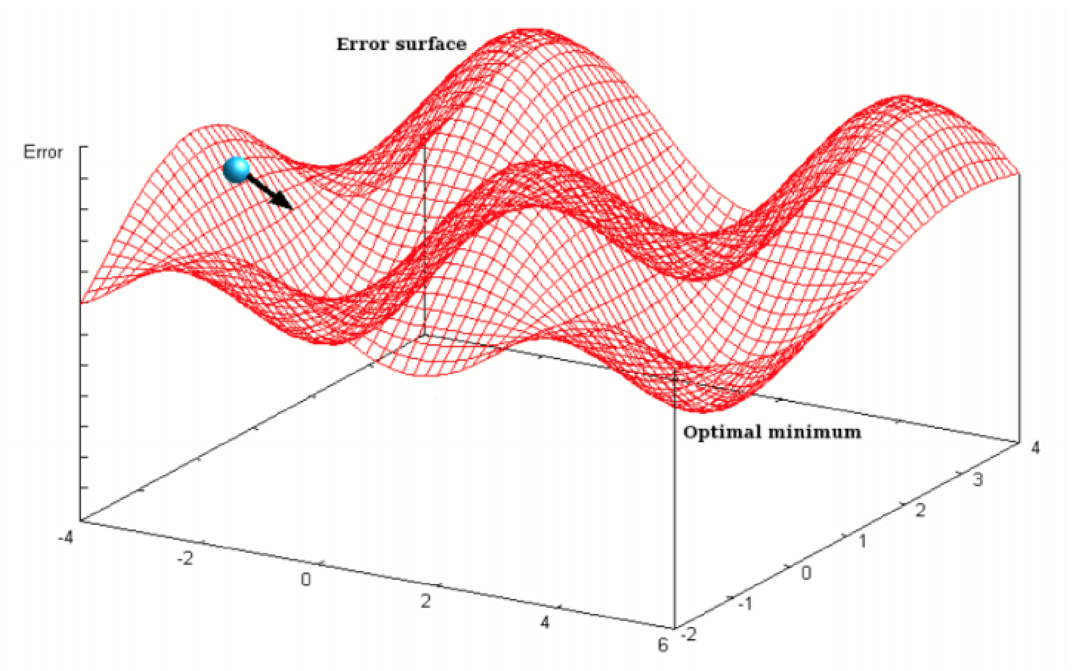


Figure 2.6: Visualizing a 3-dimensional error surface when training a neural network

is immensely complex and real world problems often require human manipulation to become "RL friendly". Second, the agent's hunger for training data is nearly insatiable and that data can often be difficult or expensive to attain. Neural networks as function approximators in deep RL help enable and scale RL approaches to increasingly complex problems. Tabular implementations of classical methods fail in arbitrarily large state spaces. For example, in Starcraft 2 [45] the player selects actions among a combinatorial space of approximately 10^8 possibilities (depending on the game resolution), using a point-and-click interface. If the state space is large enough, you cannot expect to converge on an optimal policy or optimal value function even with infinite time and computer memory. Rather than push tabular methods and computer memory to the limits, the field shifted to approximating good enough solutions with the application of deep neural networks.

Successful RL applications of the past have relied on abstracted features and dimensionality reduction creating a sandbox-like environment [9, 31]. For example, in most classical control problems such as Cartpole or Hillclimber the environment is boiled down to a few features, such as angle of the pole, coordinates of the cart, height and slope of the cart. Environments of that nature are so handcrafted and thus will rarely appear in the real world, and often are used to criticize the applicability of RL. Some of the earliest successful applications of RL had relied on carefully handcrafted sets of features based on human knowledge and intuition about a specific problem in order to be solved. Until recent advances in deep learning, RL mainly relied on problem specific engineering.

In 2013, for the first time ever Deepmind and Mnih et al. [29] demonstrated that a convolutional neural network can enable a generalized learning algorithm to successfully play Atari games and learn optimal control policies directly from raw video data in complex RL environments. This paper was groundbreaking and paved the path for deep reinforcement learning. The network was not provided with any game specific information or hand crafted visual features and had no knowledge of the internal state of the emulator. It learned from nothing but raw video input, reward signals, and a set of possible actions, just as a human player would. Even more impressive is that the network architecture and all hyperparameters used for training were kept constant across all five selected Atari games.

2.5.2 Atari Learning Environment

The Atari 2600, released in 1977 was the first mass-marketed video game console that brought the gaming revolution from the arcade into the homes of everyday consumers [7]. The Atari console introduced some of the most iconic video games of the era such as Pong, Space Invaders, Asteroids, and Breakout. The platform provided a total of 49 Atari 2600 games that varied in goals, complexity, controls, and aesthetics. Although modern video games are far more complex, Atari games are still interesting and fun for human players. Diversity within the games provide challenging environments for both humans and reinforcement learning agents to master. In early 2013 Bellemare et al. [7] produced the Atari learning Environment (ALE) introducing a challenging benchmark composed of the Atari 2600 games. This benchmark has since stood as tool to evaluate the general competency and learning efficacy of generalized AI algorithms. The ALE serves as an emulator for the Atari collection giving agents access to raw sensory information, frames of 210 x 160 pixels rendered at 60hz. A possible set of 18 actions mapped to each action a human could perform on the physical controller. The ALE also grants access to the score, or reward of each game. The ALE transformed Atari from a warm and distant memory, to a hot research test-bed for designing and testing artificially intelligent algorithms that could generalize across the wide range of problems and environment offered. The benchmark presents significant research challenges for reinforcement learning, model learning, model-based planning, imitation learning, transfer learning, and intrinsic motivation.

For the first time ever end-to-end reinforcement learning directly from raw visual data became possible. The ALE provides machine learning agents with the infrastructure to operate and observe Atari games directly from pixels. By framing each game as an MDP, reinforcement learning techniques can be applied to solve these environments and produce agents capable of super-human performance. This sort of benchmark and learning infrastructure meant that an RL system did not need to rely on human designed features to fit a tabular form, the systems themselves became general learning platforms similar to how biological humans would interact with the game.

In addition, the benchmark also helped solve the data hunger of RL algorithms. In real time, humans can process 60 frames per second (FPS) relatively well (Atari system is a 210 x 160 RGB video at 60 hz (1 hz = 1 FPS)). Within a simulated gaming environment, users can increase the render rate, or FPS up to 6000 (dependent on

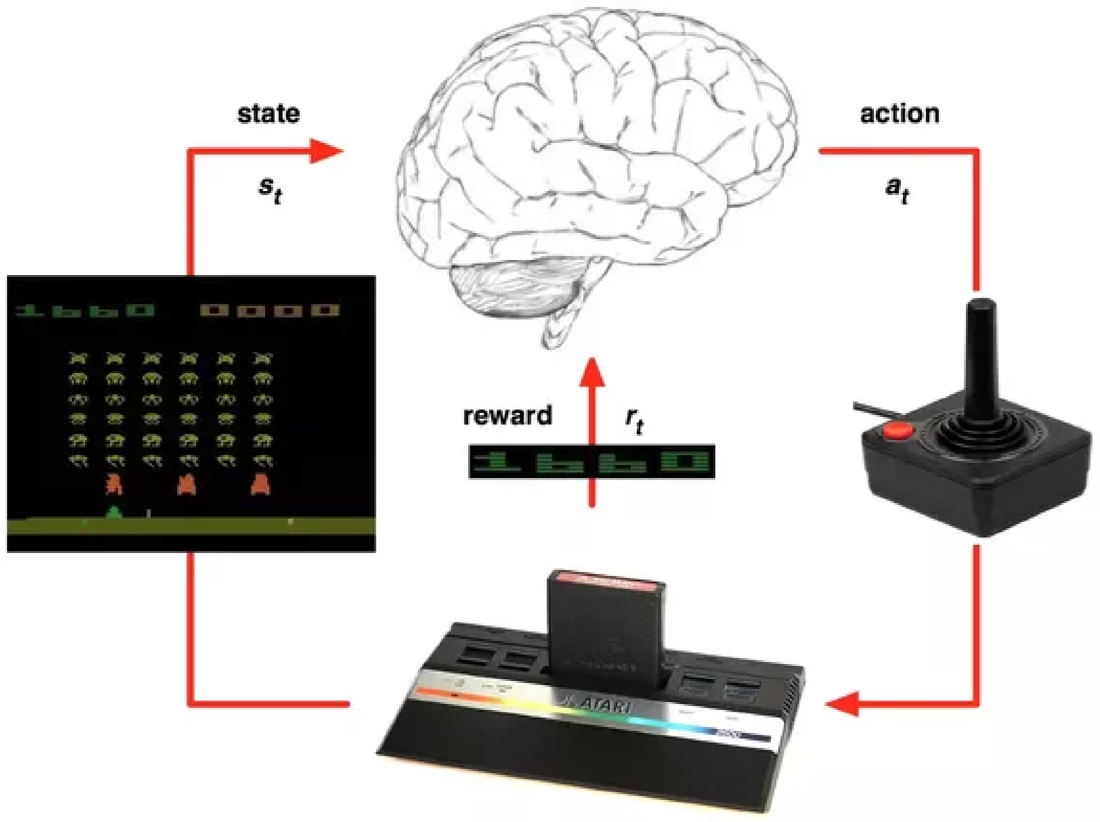


Figure 2.7: Reinforcement learning loop using the Atari console

their hardware of course). The max FPS of the ALE enables algorithms to play an unfathomable amount of games in a small amount of time. For example, it is not uncommon for an RL experiment to run for 200 million frames [30] or an equivalent 38.58 days of non-stop human experience playing the game. Considering an 8 hour day work week, it would take a human 308.64 working days, or just under 62 working weeks to complete such a task.

To summarize, scaling up RL algorithms to more complex tasks has been a long standing challenge for researchers. Thanks to simulated environments and deep learning, the variety and complexity of solvable RL environments has grown from text based games, to simple board games and control problems, to complex board games involving long term strategy. In recent years, RL algorithms are achieving super human performance in complex video games, playing as a human would utilizing raw video output, without requiring hand-crafted feature manipulation. A majority of these advances can be attributed to increased hardware power enabling lightning fast simulated environments and deep learning.

2.6 Modern Reinforcement Learning Methods

2.6.1 Deep Q-Network

The Deep Q-Network (DQN) [29] paved the way as the first massively successful combination of deep neural networks trained on raw visual data and reinforcement learning. It achieved super-human performance [30] on a large fraction of the games in the ALE. Figure 2.10 details the performance of DQN compared to the best linear learning models, with a demarcation of where DQN achieves human-level and greater performance. DQN not only attained superior results to any related works, but achieved unprecedented generality and robustness in terms of its architecture. At a high-level, DQN extends Q-learning beyond a tabular representation (a massive matrix containing every state and action combination with its estimated reward) to a convolutional neural network architecture used to approximate the state-action values over the set of possible actions in Atari.

Implementation

The first step is a minor pre-processing step applied to the raw frames. Working directly with raw Atari 2600 frames, 210 x 160 pixel images with a possible 128 color combination palette is extremely demanding in terms of computational and memory requirements. Thus, there is a pre-processing step applied to these frames aimed to reduce dimensionality. The frames are grayscaled and reduced to an 84 x 84 pixel frame.

The exact architecture of the model is shown below in Figure 2.8:

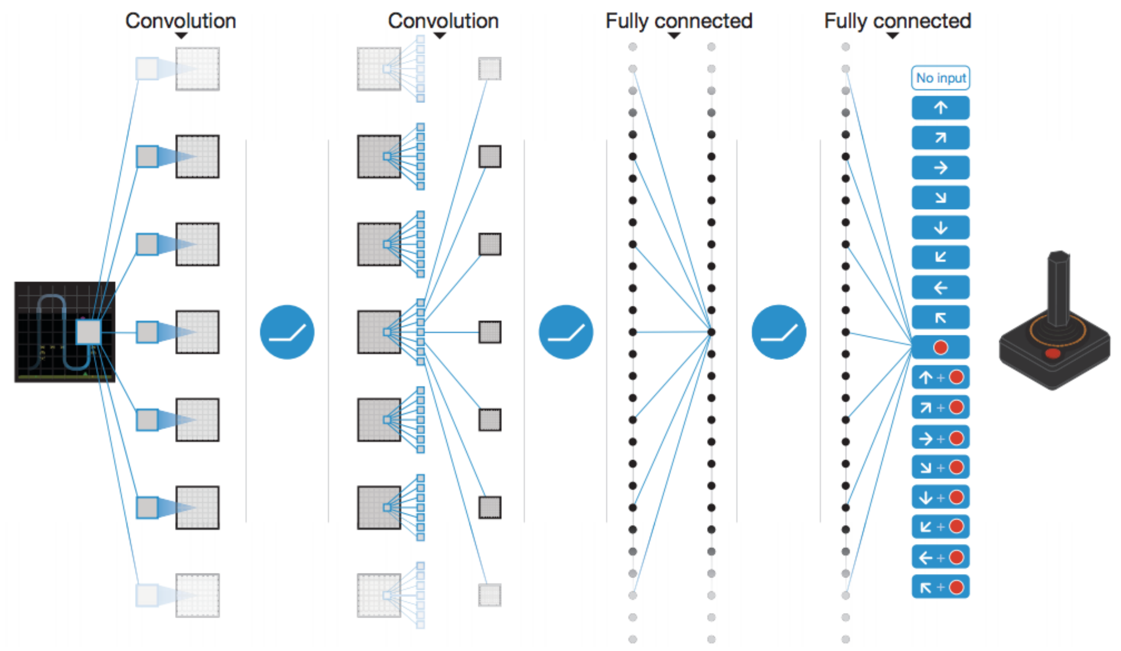


Figure 2.8: Visualization of DQN neural network architecture [30]

The actual inputs to the network consist of a concatenation of 4 subsequent frames [29], this procedure is necessary in order to maintain and convey spatial and temporal dependencies. For example, in Pong the agent needs to know the direction and velocity of the projectile. The first layer of the network convolves 32 filters of 8 by 8 with stride 4 and applies a rectified nonlinearity activation function [19, 32]. The second layer convolves the previous layers input with 64 filters of 4 x 4 with stride 2, followed by another rectified nonlinearlity. This is followed again by a third convolutional layer that convolves 64 filters of 3 x 3 with stride 1 followed by another rectified nonlinearity activation function. The final hidden layer is a fully connected layer of 512 rectifier units. The output layer is a fully-connected linear layer with a single output unit for each valid action for that environment. The number of valid actions per environment in ALE range from 4 to 18.

They performed experiments on 49 Atari 2600 games. A different copy of the network was trained on each game, with the identical architecture, learning algorithm, and hyperparameter settings, showing that their approach was robust enough to work on a variety of games. The agent had access to only the visual images as input, the game-specific score, and the number of actions with no correspondence to what they do, for example the agent was not told that action 1 was mapped to up on the controller. The final piece of information the agent had access to was a life count if applicable for that environment. As the magnitude of rewards vary greatly from game to game, Mnih et al. [30] found it necessary to clip the rewards from -1 to 1, leaving 0 rewards unchanged. Clipping the rewards limited the scale of the error derivatives and made it possible to use the same learning rate across all games.

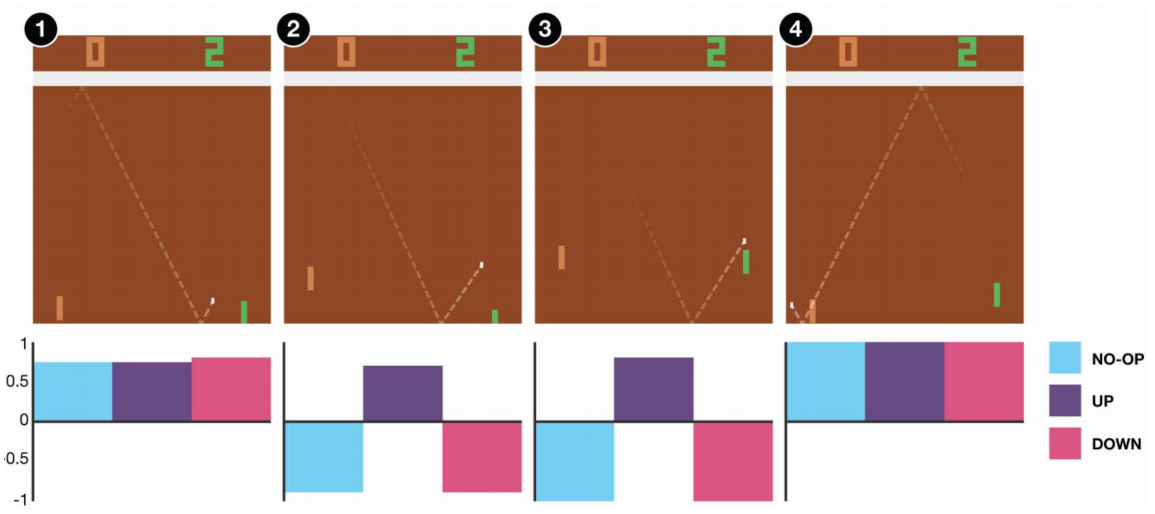


Figure 2.9: Visualization of learned q-values for Pong (y-axis on graph corresponds to value estimate) [30]

Figure 2.9 shows a visualization of the learned q-values for four different states of Pong. You can see in state 1 the agent believes that all three actions will yield reward, with moving down being slightly more rewarding, so it selects that action. In state 2 the agent predicts that going up will yield positive reward, where a no-operation or going down will yield negative reward. Based off the prediction in state 2, the agent moves up, and in state 3 the prediction is again that moving up will yield positive value. The agent also seems to be aware that hitting the ball with the sharpest edge of the paddle will send the ball flying with the greatest velocity, further increasing its chance to score. In the fourth state, the agent knows that any of the three possible actions will yield a full positive reward of 1.0 because the ball is beyond the opponent's paddle.

Key Features of DQN

DQN relies on two important features to stabilize the learning process.

First, to alleviate the problems of correlated data and non-stationary distributions, Deepmind utilized an experience replay mechanism [24] which randomly samples interactions from its history to learn from, instead of learning on-line in real time. This decorrelation of learning and interaction smooths the training distribution over the history of past interactions, essential to stabilizing the learning of DQN. Learning doesnt begin until the replay buffer is filled with at least 50,000 randomly selected actions. Deepmind's implementation stores the last 1M experience tuples (s, a, r, s) in the replay buffer at any given moment, with a first in first out replacement schedule.

Second, the learning network is different from the target network. This means that the network making decisions of what actions to take next can be considered a frozen, or check-pointed version of the learning network. The target network is updated once every 10,000 training steps. The reasoning behind this feature is that if the target network was constantly updated, the agent could end up 'chasing its own tail' so to speak.

2.6.2 Asynchronous Actor Advantage Critic

Following the success of 2015's super-human agent DQN, Minh's team at Deepmind went back to the drawing board to see if they could improve those results with a new approach. In 2016 Minh et al. [28] implemented a series of asynchronous reinforcement learning algorithms in attempts to outperform DQN. The most prominent algorithm to come out of this was the Asynchronous Actor Advantage Critic (A3C). This method is an asynchronous version of the actor-critic RL approach mentioned in Chapter 2.3.4. A3C surpassed the state of the art on the Atari domain while training for half the time on a single multi-core CPU instead of on a GPU. A3C also succeeded in continuous motor control RL environments whereas previous value based methods struggled.

Note that I will be using A2C later on in my thesis paper, which removes the Asynchronous part from A3C. OpenAI proved that a synchronous version of the Actor-Advantage Critic approach configured for a GPU provided equivalent performance in a shorter amount of run time.

Implementation

The overall architecture of the algorithm is visualized in Figure 2.10. The core ideas of A3C is that there is a global network, and there are workers. Each worker interacts with its own environment. Each worker has their own network that estimates both a value function and a policy. Recall a policy is a probability distribution over all actions for a given state. The pre-processing and first few layers of the network are identical to DQN. However, instead of a Q-value prediction for each possible action at the end of the network, there are two new layers added to the network. First is the policy layer that has a softmax output for the policy. Second is a value layer that consists of one linear output for the value function. In their paper, each worker is run on a separate processor thread. So there should be no more workers than threads on a given CPU. However, the optimal amount of workers as decided by the team at Deepmind is 16.

Every 1,000 time-steps, the global network examines each worker and computes the gradient of that workers network. These gradients are used to update the global network in a way that considers the optimal direction of all workers in the population. The new global network parameters are then copied by all workers in the population getting everyone back on the same page. This cycle continues until the end of training. Unlike DQN where a single agent represented by a single neural network interacts with a single environment, A3C uses multiple worker agents which each have their own set of network parameters. Since each agent interacts with its own copy of the environment, this provides a more robust strategy that a single agent network. Additionally, since the agents are using the value functions (the critic) to update the policy (the actor) the performance of the network is improved more intelligently than traditional policy gradient methods or value based methods alone.

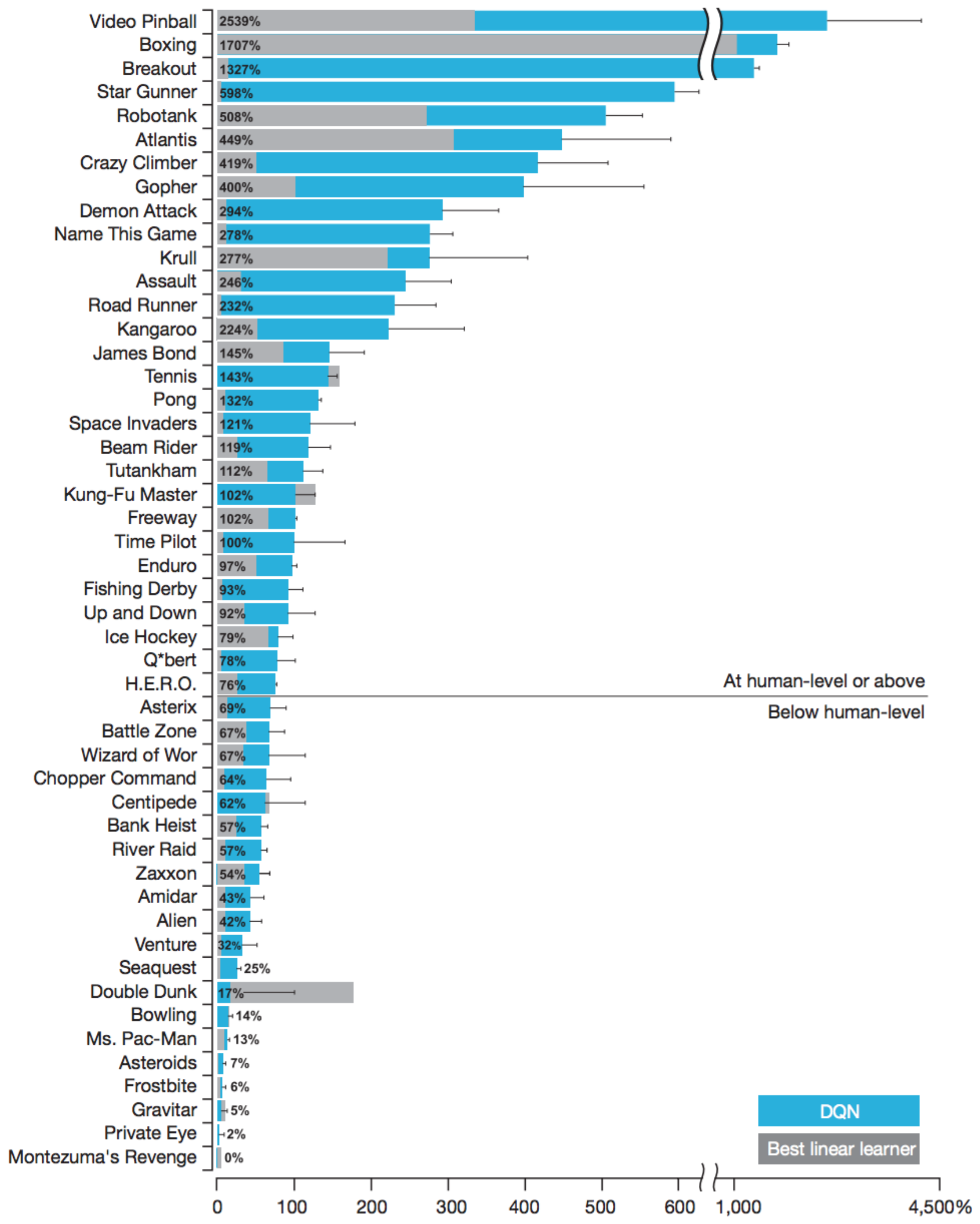


Figure 2.10: Results summary of DQN vs. human level performances across 49 Atari games [30]

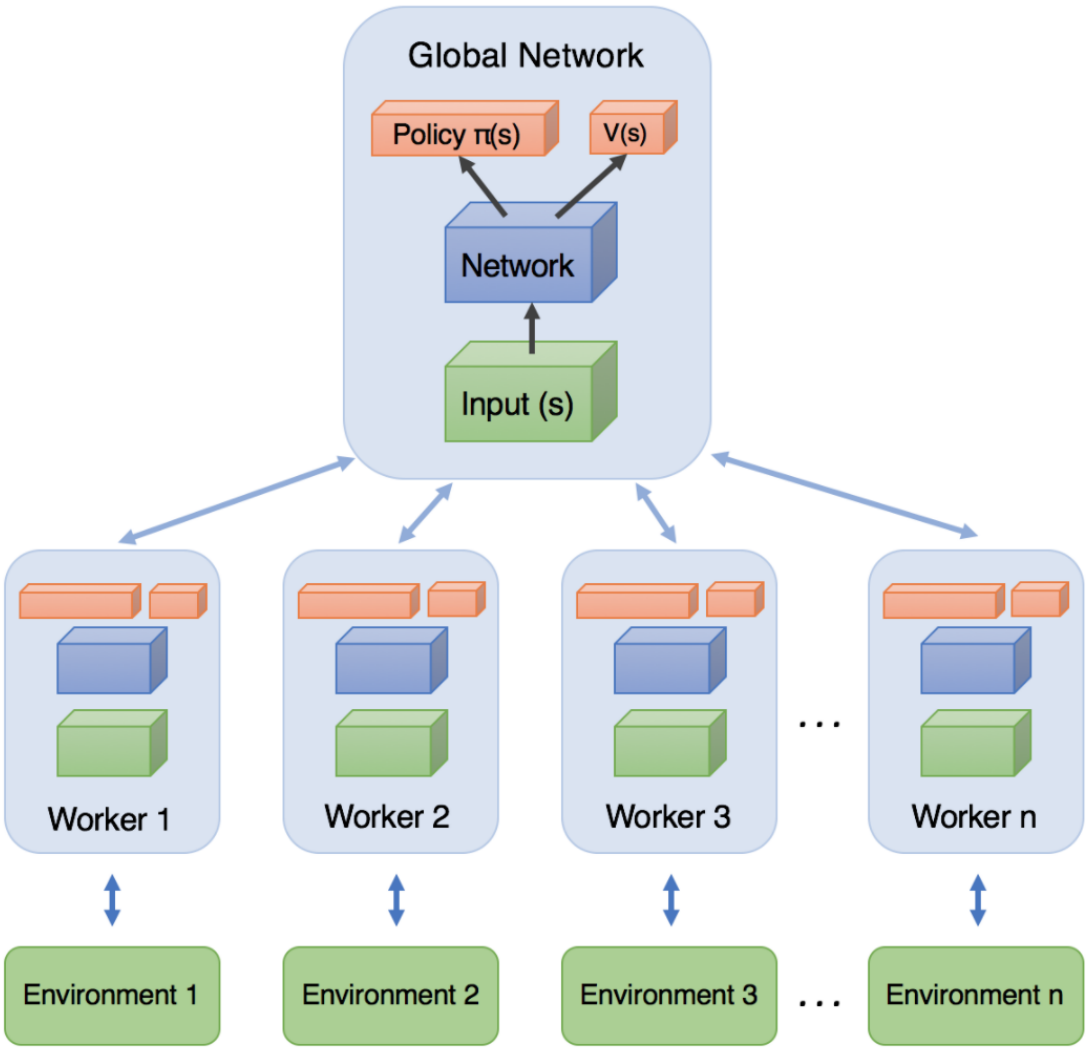


Figure 2.11: The architecture of the A3C algorithm

Chapter 3

RELATED WORKS

The key aspect of this chapter is to highlight related works in the field of Reinforcement Learning, specifically focusing on works that influence exploratory behavior within RL algorithms. For each paper, I will describe the differences between their works and mine, and mention any cross-over between theories.

3.1 Efficient Exploration in Reinforcement Learning

In 1992 Sebastian Thrun [42] released the paper Efficient Exploration in Reinforcement Learning. His paper was the first to distinguish between two families of exploration strategies: undirected and directed exploration. Thrun evaluated the role of exploration in active learning and described several strategies in depth for exploration in finite, discrete, reinforcement learning environments. The paper addresses the issues of *efficient exploration* in a reinforcement learning framework. Thrun investigates several exploration techniques still as relevant today as they were in 1992.

Undirected exploration utilizes no exploration specific knowledge and ensures exploration by combining randomness into action selection, whereas directed exploration relies on knowledge about the learning process itself. The computational costs of undirected exploratory methods are often slim to none, whereas directed exploration can end up demanding substantial compute resources and can be very difficult to generalize across environments. Thrun extensively studies exploration within Q-learning and value based algorithms, and heavily favors directed exploration in action space. Although inspired by his work, my work differs in that I focus on exploration within policy based methods and primarily try to maximize the effectiveness of undirected exploration.

3.2 Parameter Space Exploration Methods for A3C and DQN

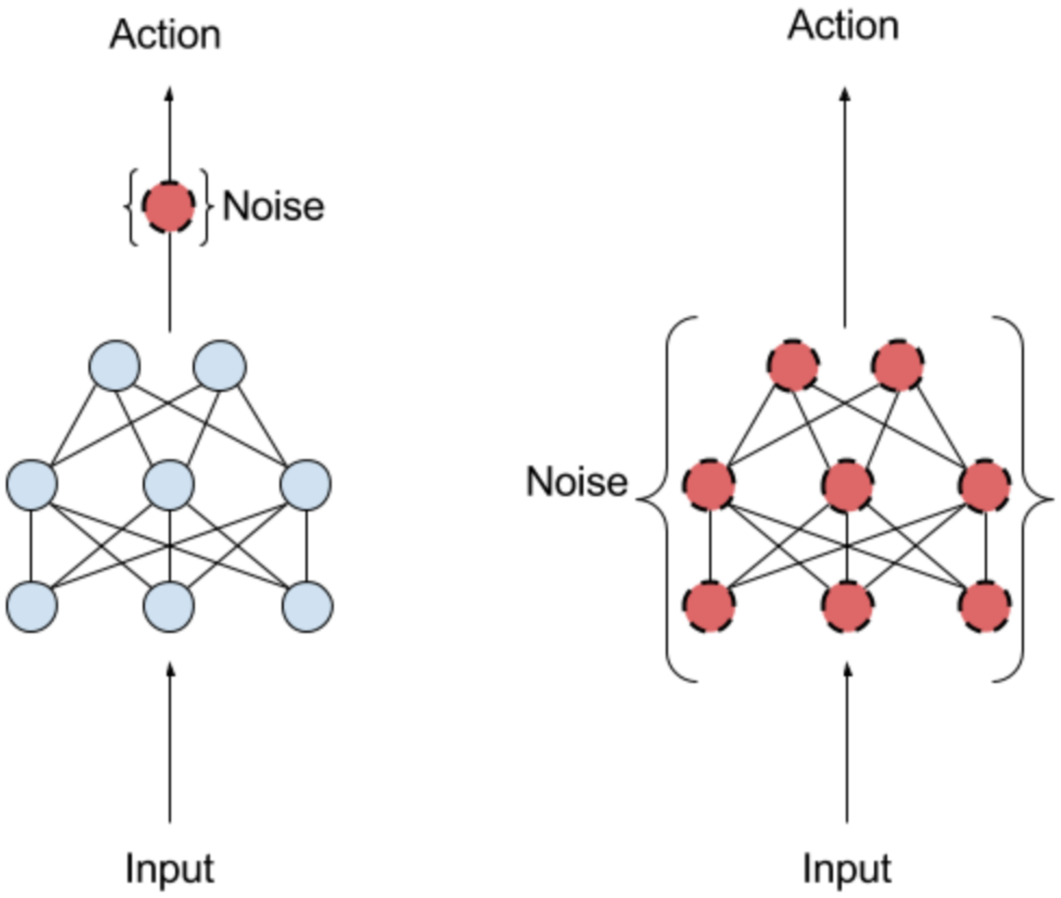


Figure 3.1: Visualization of action space vs. parameter space exploration [34]

Both Deepmind [14] and OpenAI [34] released papers on February of 2018 introducing versions of A3C and DQN that rely on parametric noise added to the weights of their neural networks for exploration rather than conventional action-space exploration methods. The variants essentially add learned noise to the weights for each layer of their neural networks, leaving biases independent. This noise is added with the intention of positively influencing exploration. This direction of research is similar to mine in the sense that they are utilizing undirected exploration to expose agents to novel states but differs in the sense that they are targeting parameter space instead of action space. The Deepmind paper evaluated the performance of Noisy Net on 57 Atari games against the performance of epsilon-greedy DQN and vanilla A3C. The performance of Noisy-Net-A3C against vanilla A3C was mediocre, with only a handful of games having significant performance increases and a few games resulting in a significant decrease in performance. Nonetheless, both papers introduced novel parameter space exploration methods for A3C and DQN in the Atari domain.

3.3 Directed Diversity Driven Exploration for A2C

Hong, et al. [17] of the National Tsing Hua University in Taiwan released Diversity Driven Exploration for Deep Reinforcement Learning in February of 2018. These researchers utilize information theory to encourage exploration in A2C. This paper is most similar to my research in terms of goals and experimental framework, but their implementation took a directed exploration approach rather than undirected. The main objective of their work is to encourage an agent to explore different behaviors during training. The authors modify the loss function of A2C to take into account the distance of the last five policies. Recall, that a policy is a probability distribution over all possible actions representing the likelihood of selecting a particular action. The authors use Kullback-Leibler divergence, which is a measure of how one probability distribution diverges from another [20]. Using KL divergence one can estimate the distance between two probability distributions, in this case (policy represented by π) π and π' are the distributions in question. The loss function modification encourages agents to update π with gradients towards directions such that π diverges from the previous five policies. The modification resulted in several positive qualities including more efficient learning curves to difficult Atari games and the ability to escape deceptive local optima. Most importantly, their agent proactively sought out new policies, increasing the opportunity to visit novel states even in the absence of reward signals from its environment. The paper concludes with experimental results on six Atari 2600 games, including three games I experiment with: Freeway, Montezumas Revenge, and Q*Bert. The results of the experiments compare DQN, A2C, Noisy Net, and diverse-A2C show that the their A2C modifications outperform both vanilla A2C and the Noisy-Net [14] paper detailed above.

Chapter 4

IMPLEMENTATION

This chapter is organized in the following sections where I will discuss:

(i) Tools and infrastructure that were critical to the foundation of my research.

(ii) Discussion of design decisions for modifying A2C and the specific software implementation details associated with my novel variant, Annealing-A2C.

(iii) An introduction to the exploration difficulty metric (EDM) to help classify environments based on exploratory challenges that they pose for RL algorithms.

4.1 Tools and Infrastructure

This section will discuss two of OpenAI's open source projects, Gym and Baselines. Both projects greatly empowered my research and the infrastructure provided enabled me to stand on the shoulders of giants. Thanks to these projects I was able to focus on experimentation and implementation of new algorithms and did not have to reinvent the wheel while doing so.

4.1.1 OpenAI Gym

OpenAI Gym [9] is an open source project that is simultaneously an infrastructure platform and a collection of reinforcement learning environments. Ranging from the Atari 2600 games, to classic control problems, to complex 3-d robotics simulators, there are hundreds of different RL environments to choose from. All Gym environments have a standardized API enabling the plug and play of one algorithm across any environment within the Gym. The origins of OpenAI Gym come largely out of frustration. RL has a long history, but until recent advances in deep learning, it required lots of problem-specific engineering. Deepmind's 2013 and 2015 Atari results [29, 30] Pieter Abbeel's robotics work [2, 23], and AlphaGo [37] all used generalized deep RL algorithms which all made minimal assumptions about their environment, and thus can be applied in other environments. The problem was that very few RL environments were plug and play so to speak. RL is very general, encompassing all problems that involve making a sequence of decisions: for example, controlling a robot's motors so that it is able to run and jump, making business decisions like pricing and inventory management, or playing video games and board games. However, RL research is often slow and disorganized due to the decentralization and inconsistency between benchmarks and papers.

In supervised learning, progress had been driven by large labeled datasets like ImageNet [22]. In RL, the closest equivalent would be a large and diverse collection of environments. Prior to OpenAI Gym the existing open-source collections of RL environments lacked variety, and were often difficult to even set up and use. Additionally, the lack of standardization across environments used in publications led to anger in the community over inability to reproduce results. Subtle differences in the problem definition, such as the reward function or the set of actions, can drastically alter a tasks difficulty. This issue makes it difficult to reproduce published research and compare results from different papers that use decentralized environments.

4.1.2 OpenAI Baselines

OpenAI Baselines [13] is a set of high-quality implementations of reinforcement learning algorithms. Following in suit with OpenAI's open-source nature, an open-source repository contains all of these implementations. A common place for implementations make it easier for the research community to replicate, refine, and identify new ideas. It also helps by leaning on the community to create new algorithms and catch bugs in existing ones, as well as confirming and reproducing state-of-the-art experiments. The amount of support that I was able to receive from the forums of this github project was paramount to my success in this thesis project.

4.2 Design and Implementation of Novel Algorithm Modifications

4.2.1 Design

Intuitively, contemplating exploration as a human, it is apparent that a diverse set of people working together on a new task can result in efficient discovery and learning of a problems state space. Within a human workforce, there would be little to no innovation if everyone had identical upbringings, educations, and experiences. For example, if you put ten humans into ten individual copies of a maze, each with the exact same upbringings and problem solving strategies, how effectively or differently would they explore the maze? These beliefs influenced my design of a population-based, diversity-driven exploration strategy in action-space. Thus, A2C seemed best suited for this experimental research due to the population of workers each interacting with their own environment. One appeal of actor-critic methods is their explicit separation of policy and value function parameters, which leads to a richer behavior space. This very separation, however, often leads to deficient exploratory behavior. Current research influencing exploration within actor-critic methods lies primarily in parameter space noise, and adding entropy to its policy to try shake the model out of local optima. As of conducting this research and writing the paper, I have yet to see any attempts influencing exploration in A2C with action-space strategies. Vanilla A2C selects actions via sampling a probability distribution in a greedy manner, rarely selecting low probability actions. Thus greedily explores a state space which can lead to sub-optimal performance, and inefficient learning curves. My goal is to modify the A2C algorithm via action-space exploration such that more states are experienced by the model in an aggressive fashion. Through exposure to as many states as possible early on in training, the value function and policy function should theoretically discover reward surfaces earlier, thus improving performance and sample efficiency. Annealing epsilon-greedy, naive yet practical, has produced state-of-the-art results in DQN [30] and multi-armed-bandit [42] problems. Adding an annealing epsilon value to encourage exploration in an element-wise (per worker) fashion versus a batch (total population) randomization was the direction taken to maximize diverse experiences within the workers respective environments.

4.2.2 Implementation

At a high level, the implementation put in place an aggressive exploration phase for A2C, with an epsilon value starting at 100% and annealing to 0% over the first 5% of training. At 11M time-steps per experiment, this value anneals to zero at the 550K time-step mark. Two main components implemented these modifications.

1. A linear scheduler for an arbitrary point of training to stop exploration
2. Modification of the Runner class function: run()

The A2C implementation uses a class Run which facilitates the interactions between the actor-learners and the gym environments. Essentially, the class executes actions on the environments and receives resulting rewards and new observations. Additionally, run() resets an environment if a terminal signal has been received. The function loops for five iterations, called a mini-batch, storing all data generated by the 16 actor-learners in the mini-batch. This data is used by the model later for learning and ultimately increases model accuracy.

There are two primary steps in this process, the predictions for each agent and the execution of those predictions. My modification adds a third procedure to this process in between the prediction and execution stages. All three steps are detailed below.

Prediction Stage

The primary function within the Run class, run(), begins by passing the model a vector of observations. This observation vector holds 16 current observations (each observation is 4 concatenated frames of 84 x 84 pixels). This vector is passed to the global A2C network to produce policy and value predictions for those states. The output received is a vector of length 16 containing the action predictions and value predictions for each agent. The actions are then used in the execution step whereas the values are stored for learning after the batch is complete.

Exploration Stage

Before training, a variable *explorationEnd* is initialized to an integer that corresponds with 5% of training. In the standardized experiments for A2C used in this paper, that value is 550,000.

In training, if the current step-count is less than *explorationEnd*, then the exploration phase kicks in. The exploration phase consists of a loop iterating through each index of the action prediction vector. For each action, a random number is generated between 0 and 99. If that number is less than the current epsilon value, that action is replaced by a random integer within the range of legal actions in that environment. Else, the initial action prediction remains the same. This added exploration phase does not threaten the stability of A2C's ability to learn, because the action vector is intercepted and if the actions are changed, that change is reflected throughout the execution and learning process of the algorithm.

Execution Stage

The actions generated, whether following the policy function prediction, or hijacked and set in the exploration phase, are finally executed on their respective environments. The execution of actions return three vectors (vector length based on number of agents in population): new observations, resulting rewards, and termination signals for each environment. This process is repeated for the duration of the mini-batch and at the end of the run() function the algorithm begins the learning process on the data yielded from the mini-batch.

4.3 Exploration Difficulty Metric

The Atari environments available for reinforcement learning algorithms range vastly in difficulty. Bellemare et al. [6] roughly classified the games in a taxonomy based on exploration difficulty. I propose the exploration difficulty metric (EDM) to quantify exactly how difficult the Atari environments are. The metric needs a set of episodes computed by running a completely random policy on an environment for 1M timesteps. The metric is expressed by the simple equation below, as a ratio of non-zero reward episodes, over the total amount of episodes from the set.

$$edm = count(nonZeroRewardEpisodes)/(count(totalEpisodes)$$

The ratio gives researchers an idea of exactly how difficult reward is to find via undirected (random) exploration. The metric can also be extended beyond the Atari environments. Any reinforcement learning environment has the potential for a random policy to interact, thus enabling EDM to be computed for any environment which is posed as an RL environment. Essentially, EDM suggests how sparse or constant reward is found in an environment via purely random action selection. This information can help provide researchers with insight as to what exploration strategies may work better than others in that environment.

Chapter 5

VALIDATION

This section answers the following questions:

(i) Can a quantifiable measure indicate how difficult an Atari game is for reinforcement learning with respect to exploration?

(ii) How does DQN compare to A2C in terms of overall performance, sample efficiency, and variance/stability?

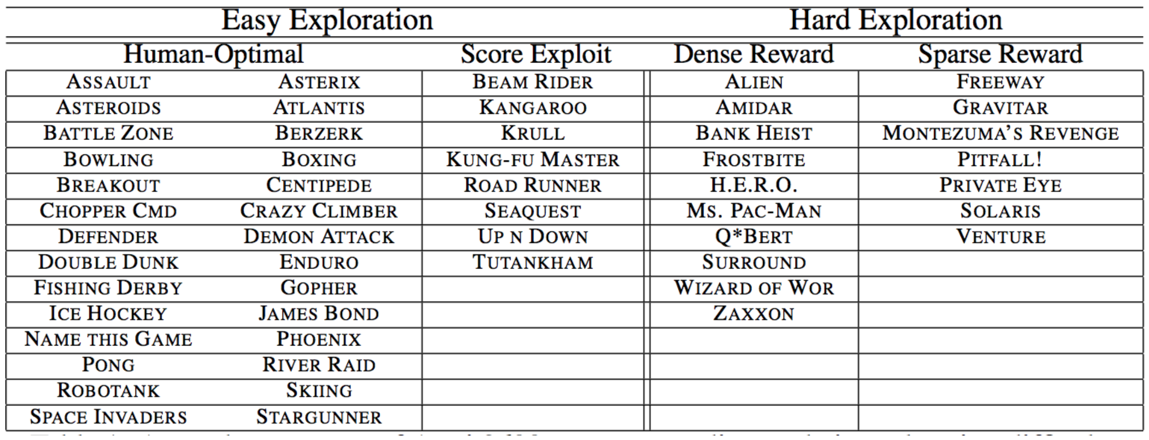
(iii) Does my proposed Annealing-A2C improve upon vanilla A2C with respect to overall performance, sample efficiency, and variance?

5.1 Experimental Framework

The following sections will explain my experimental framework. First, I explain the reasoning behind the selection of six Atari 2600 games for experimentation. Second, I describe the conditions and hardware that powered the experiments. Finally, for the sake of reproducibility I detail the three algorithms (DQN, A2C, and Annealing-A2C) exact architectures and hyperparameter values used for experimentation.

5.1.1 Selected Environments

The Atari Learning Environment provides a robust set of games to play. I selected a set of games ranging from easy to hard difficulty according to the taxonomy [6] of Atari games, loosely categorized on exploration difficulty. The figure below shows Bellemare's taxonomy.



Easy Exploration			Hard Exploration	
Human-Optimal		Score Exploit	Dense Reward	Sparse Reward
Assault	Asterix	Beam Rider	Alien	Freeway
Asteroids	Atlantis	Kangaroo	Amidar	Gravitar
Battle Zone	Berzerk	Krull	Bank Heist	Montezuma's Revenge
Bowling	Boxing	Kung-fu Master	Frostbite	Pitfall!
Breakout	Centipede	Road Runner	H.E.R.O.	Private Eye
Chopper Cmd	Crazy Climber	Seaquest	Ms. Pac-Man	Solaris
Defender	Demon Attack	Up n Down	Q*Bert	Venture
Double Dunk	Enduro	Tutankham	Surround	
Fishing Derby	Gopher		Wizard of Wor	
Ice Hockey	James Bond		Zaxxon	
Name this Game	Phoenix			
Pong	River Raid			
Robotank	Skiing			
Space Invaders	Stargunner			

Figure 5.1: Exploration Taxonomy of Atari Environments [6]

The first two games selected, **Space Invaders** and **Pong**, are categorized as easy, or human optimal. The games under this label have relatively constant rewards throughout the duration of the game, making them fairly easy and solvable for RL with basic exploration strategies.

The third and fourth games to test, **Ms. Pac-Man** and **Q*Bert**, fall under hard exploration, with dense rewards. This means that reward signals are grouped together, and when an agent realizes these rewards they are massive in comparison to the rest of the reward signals within that environment.

The final set of games selected, **Freeway** and **Montezumas Revenge** are classified as hard exploration environments with sparse reward. A sparse reward environment means that the agent may play the game for tens of thousands of timesteps without receiving a non-zero reward. As is the case with Montezumas Revenge, an infamous game that is the focus of many exploration research efforts. Sparse reward problems are the most challenging environments within the ALE because of the inherent difficulty RL agents face when crediting sequences of actions to rewards are so far and few in between. As a result, no super-human performance has been achieved on these games.

5.1.2 Experimental Conditions

The following sections detail the hardware used in experimentation and overall experiment requirements.

Hardware

The Cal Poly Massively Parallel Accelerated Computing (MPAC) Lab machines were utilized for these experiments. There are 37 machines available in this lab, the hardware capabilities of each machine is listed below.

- CPU: 28 cores, Intel Xeon CPU E5-2695 v3 @ 2.30GHz
- Memory: 32 GB memory, 66 GB swap
- GPU: GeForce GTX 980, 4 GB memory, 2048 cuda cores

Experimental Requirements

The DQN experiments ran for 10M timesteps each, producing 40M frames. The duration of these experiments lasted 28 hours on this hardware setup.

The A2C experiments ran for 11M timesteps each, producing 44M frames. The duration of these experiments lasted 2 hours on this hardware setup.

Both algorithms completely consumed GPU memory. The CPU core usage sat between 30 and 40 percent for all 28 cores, and DQN consumed 99% of RAM while A2C consumed very little RAM.

5.1.3 Algorithms and Hyperparameters

DQN

Each experiment runs on the aforementioned hardware with the following architecture. For the Atari experiments, the network architecture as described in Mnih et al. (2015) is used. This consists of 3 convolutional layers (32 filters of size 8 x 8 and stride 4, 64 filters of size 4 x 4 and stride 2, 64 filters of size 3 x 3 and stride 1) followed by 1 fully connected hidden layer with 512 units followed by a linear output layer with one unit for each action. Rectified linear unit (ReLU) activation functions are used in each layer, while layer normalization (Ba et al., 2016) is used in the fully connected part of the network. For observations, each frame is down-sampled to 84 84 pixels, after which it is converted to grayscale. The actual observation to the network consists of a concatenation of 4 subsequent frames. This setup is identical to what is described by Mnih et al. (2015).

Hyperparameters differ slightly to Mnih 2015, as those would cause RAM to explode. The target networks are updated every 1000 timesteps. The Q-value network is trained using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of (0.0001) and a batch size of 32. The replay buffer holds up to 50,000 state transitions. Epsilon is linearly annealed from 10% to 1% over the first 1M time-steps. In all cases 50,000 random actions are performed to collect initial data for the replay buffer before training starts. Lamdba is set to 0.99, and rewards are clipped to be in [-1, 1], additionally the gradients for the output layer of Q are clipped to be within [-1, 1].

A2C

Each experiment runs on the aforementioned hardware with the following architecture and hyperparameters.

The A2C algorithm uses 16 actor-learner agents running on a single machine. Each agent has their own environment to interact with, each seeded sequentially increasing by 1 starting from 0. The experiments use the same input preprocessing as Mnih et al. 2015. Each frame is down-sampled from 160 x 210 pixels to 84 x 84 pixels, after which it is converted to grayscale. The actual inputs to the network consist of a concatenation of 4 subsequent frames. The agents utilize a shared network architecture as described in Mnih et al. (2016). This consists of 3 convolutional layers (32 filters of size 8 x 8 and stride 4, 64 filters of size 4 x 4 and stride 2, 64 filters of size 3 x 3 and stride 1) followed by 1 fully connected hidden layer with 512 units followed by a linear output layer with one unit for each possible action for that environment (min 4, max 18). ReLUs are used in each layer, while layer normalization (Ba et al., 2016) is used in the fully connected part of the network.

Any cross-over hyperparameters with DQN are kept the same as listed above for A2C with one exception. The learning rate is fixed at (0.001).

Annealing-A2C

The architecture of Annealing-A2C is identical to A2C. The hyperparameters are kept the same as vanilla A2C listed above. However, a new hyperparameter is introduced to each actor-learner agent, epsilon. This value is initially set to 100% and linearly annealed to 0% over the first 5% of training, in this case at 550,000 time-steps.

5.2 Results and Analysis

This section contains graphs and analysis of the experiments conducted. For each environment, four algorithms were run, a completely random agent, a DQN agent, an A2C agent, and my Diverse-Annealing-A2C variant. The graphs for each experiment display episode rewards on the y-axis and display time-steps on the x-axis maxing out at 11M time-steps. Each point on the graph corresponds to a single episode generated by the agent at that particular time-step.

Exploration Difficulty Metric

Prior to observing the results of the algorithms across the games, refer to Table 5.1 for the computations of the Exploration Difficulty Metric for each game. A quick glance at this table can indicate how difficult each environment is from an exploratory perspective.

Final Average Reward Agent-Environment Matrix	
Environment	EDM Score
Space Invaders	99.78%
Pong	100%
Ms. Pacman	99.86%
Q*Bert	93.25%
Freeway	0.0027%
Montezuma	0.0031%

Table 5.1: Atari EDM Summary

The scores for both Space Invaders and Pong are at or are very close to 100% which tells us that for virtually every episode generated by a random policy, we are receiving more or less a constant reward signal. This score signifies that a simple or naive exploration strategy is sufficient to explore and learn within that environment. These scores are also consistent with the performance on agents on those two environments as shown below. The score for Ms. Pacman is slightly misleading, because Ms. Pacman is a 'dense' reward game meaning that if you clear the whole level, then a huge reward bonus is applied. The EDM score of Ms. Pacman is too high because of the very small constant rewards obtained by eating a tiny node in the maze. The score of Q*Bert indicates that a fair amount of episodes (almost 7%) resulted in absolutely zero reward, which makes sense because the agent can only score a 0 point game if it jumps off the edge of the map in the beginning of the game. Freeway and Montezuma, both games have extremely sparse reward signals and the EDM score shows just that. Out of thousands of episodes played with a random policy only a few yielded non-zero rewards. As scores stray further south of 100% the complexity of the state-space and difficulty of that environment increase exponentially.

Atari Results

Refer to Table 5.2 for a summary of all examined algorithms final performance in each environment.

5.2.1 Space Invaders

Space Invaders is a game that involves control of a space ship that can move laterally across the bottom of the screen, and can fire projectiles upwards with the goal of destroying enemy ships. There are opponent space ships that fly in a formation across the screen laterally, getting closer to the player controlled ship over time. The

Final Average Reward Agent-Environment Matrix				
Environment	Random	DQN	A2C	Annealing
Space Invaders	144.97	595.10	680.16	**714.43**
Pong	-20.30	-5.59	**20.28**	19.93
Ms. Pacman	239.97	1701.37	**2078.69**	2002.88
Q*Bert	156.15	484.51	5669.59	**6747.15**
Freeway	0	**9.09**	0	0
Montezuma	0	0	0	0

Table 5.2: Results Summary

game ends when the player runs out of ships (3 lives total). Refer to the appendix for a screen shot showing the player view of Space Invaders.

Figure 5.2 shows an agent with a random policy, the line through the plot indicates the agent's average performance. The random agent finished training with an average reward per episode of 144.97. This plot indicates that Space Invaders has a somewhat constant reward signal easily discoverable by random exploration.

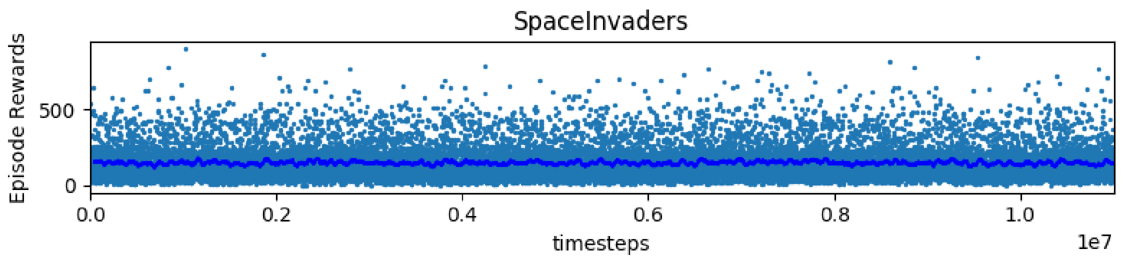


Figure 5.2: Space Invaders: Random policy with variance

Figure 5.3 shows the DQN agent's performance in Space Invaders. A notable point in this graph is after the 6M time-step mark there is a significant failure in policy, this is most likely due to a very poor target network update. It is able to recover but variance around the average is high, keeping the final average performance down at 595.10.

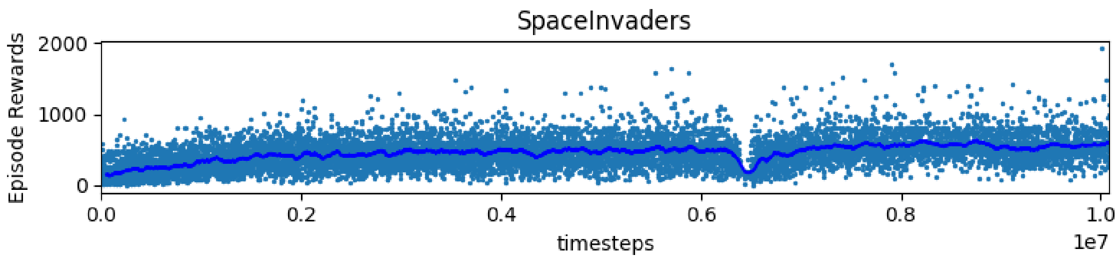


Figure 5.3: Space Invaders: DQN with variance

Figure 5.4 shows A2C's performance in Space Invaders. The learning curve of A2C vs. DQN is much slower and steadier. No apparent failures in policy, and over time the agents in the population are able to attain very high reward games between 1000 and 2000 reward. This indicates that the agent was not finished learning and if left to run longer would have yielded a stronger final performance. The agent finished with an average episode reward of 680.16 significantly beating DQN.

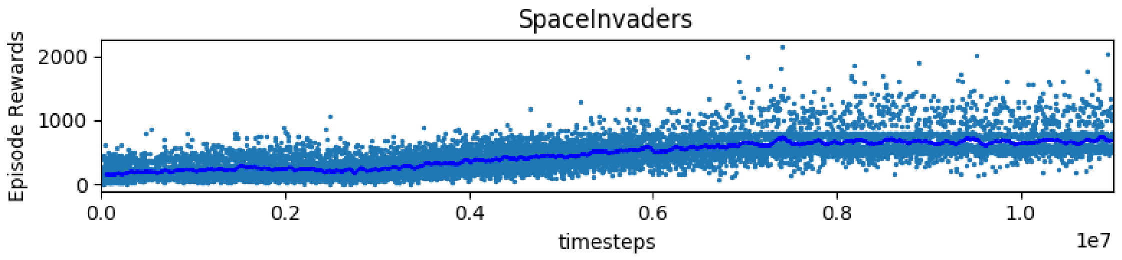


Figure 5.4: Space Invaders: A2C policy with variance

Figure 5.5 is a comparison of the random agent (blue), DQN (yellow) and A2C (green). DQN is able to achieve a better performance quicker than A2C but struggles in the long run due to a policy failure. A2C has a strong and consistent learning curve, however requires more time initially to hone in on a rewarding policy.

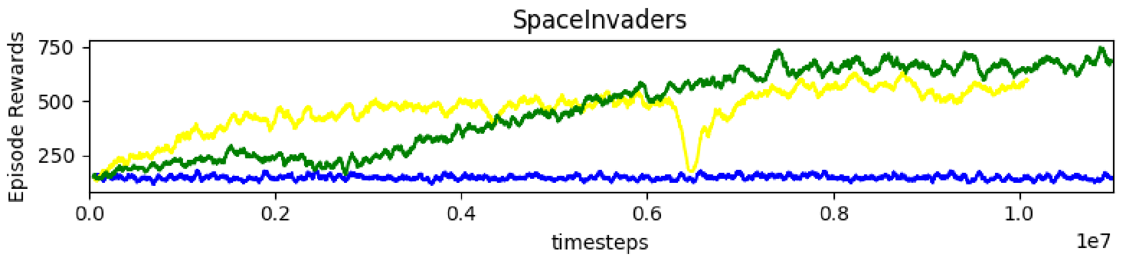


Figure 5.5: Space Invaders: Random (blue) vs. DQN (yellow) vs. A2C (green)

Figure 5.6 is a comparison of the random agent (blue), A2C (green) and Annealing-A2C (red). The learning curve of vanilla and my variant are very similar here, and I think this can be attributed to the constant reward signal of Space Invaders, thus Space Invaders can be solved with very little exploratory activity. However, Annealing-A2C has a higher peak performance and ends with a slightly stronger final performance of 714.43 over A2C's 680.16.

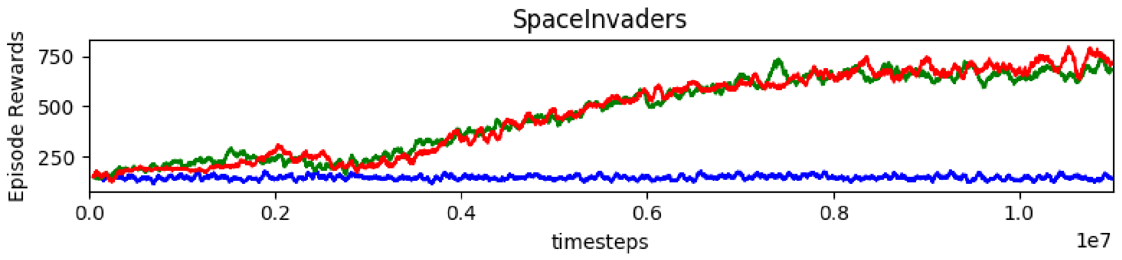


Figure 5.6: Space Invaders: Random (blue) vs. A2C (green) vs. A2C Annealing (red)

5.2.2 Pong

Pong is a game where the player controls a paddle that moves up and down on the right side of the screen. There is an opponent paddle on the left side of the screen with the same actions available as the player. The objective is to hit the ball on the screen past the opponent paddle, first player to 21 points wins. Refer to the appendix for a screen shot showing the player view of Pong.

Figure 5.7 shows an agent with a random policy playing Pong, the line through the plot indicates the agent's average performance. The random agent finished training with an average reward per episode of -20.30. This displays the very cut and dry reward structure of Pong.

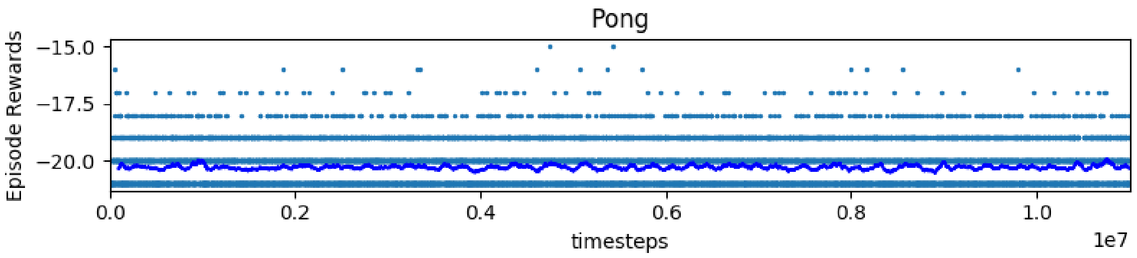


Figure 5.7: Pong: Random policy with variance

Figure 5.8 shows the performance of a DQN agent playing Pong. The notable part of this graph is the extreme variance that DQN has throughout training. Even at the end of training the graph is full of scores ranging from -21 to 20. So it can still completely lose a game without scoring but can also win a game. The final average reward for DQN in Pong was -5.59, indicating that there are still more losses than wins for the agent. It was unable to converge on a solution for Pong in 10M timesteps.

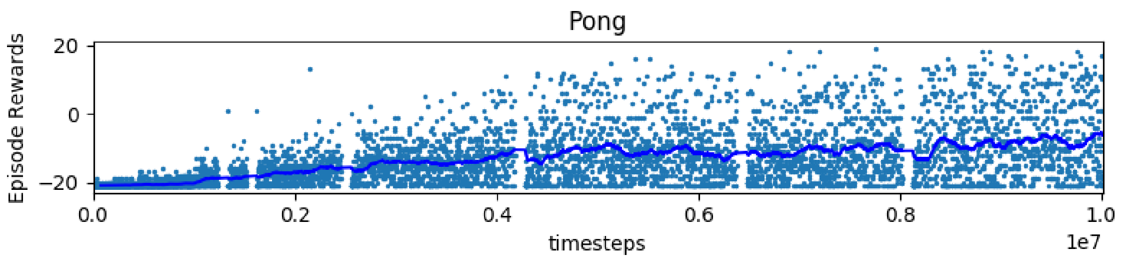


Figure 5.8: Pong: DQN with variance

Figure 5.9 shows the performance of A2C playing Pong. The agent successfully converged on a solution just before the 8M time-step mark. Aside from the medium to high degree of variance, this is close to an ideal learning curve. The agent finished with a final performance of 20.28, indicating that virtually every game can be won at the end of training.

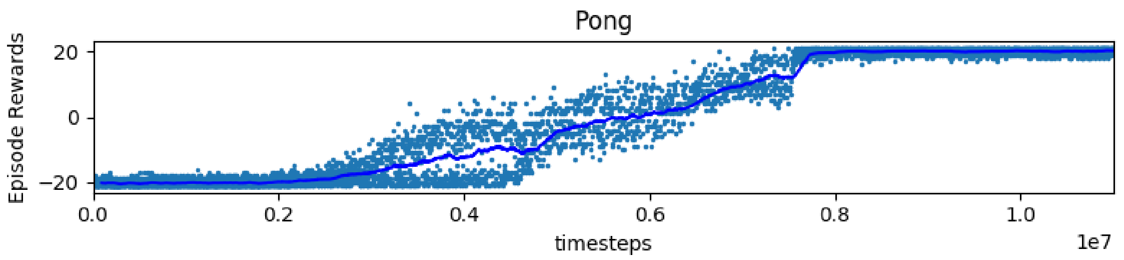


Figure 5.9: Pong: A2C policy with variance

Figure 5.10 shows the performance of Annealing-A2C playing Pong. Two things are very promising about this graph. First is that the agent converges on a solution just after 6M time-steps, much earlier than vanilla A2C. Second is that the variance is much tighter than the vanilla version. I believe this is due to the aggressive exploration of state-space in the annealing exploratory period. Pong has a relatively small state-space and the tight learning curve indicates that the agent efficiently explored that space.

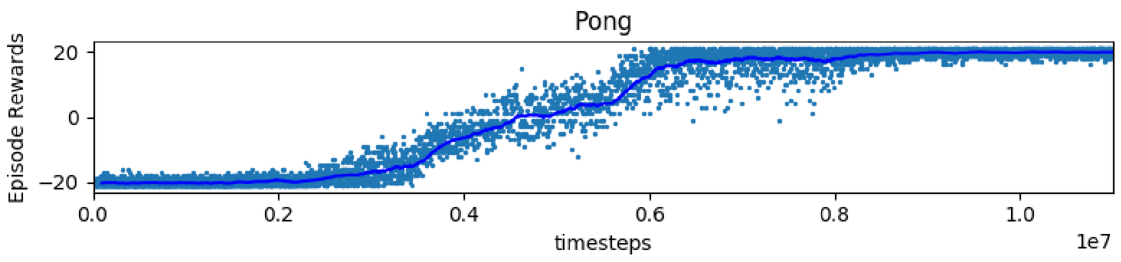


Figure 5.10: Pong: A2C-Annealing policy with variance

Figure 5.11 is a comparison of the random agent (blue), DQN (yellow) and A2C (green). A2C is clearly the dominant method for solving Pong. If the hardware available had the capabilities to use Deepmind's hyperparameters for DQN it would have stood a better chance.

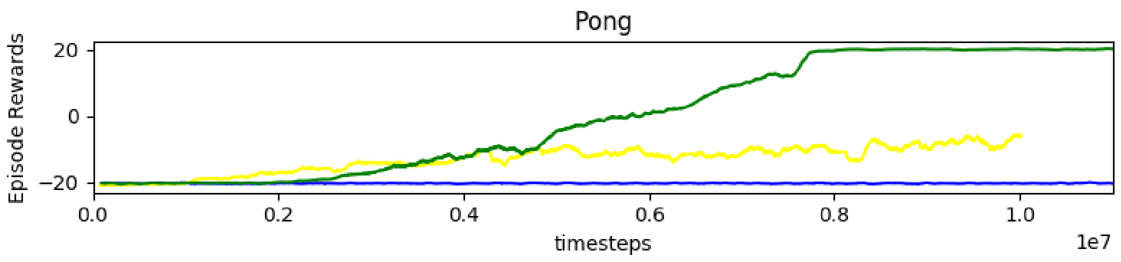


Figure 5.11: Pong: Random (blue) vs. DQN (yellow) vs. A2C (green)

Figure 5.12 is a comparison of the random agent (blue), A2C (green) and Annealing-A2C (red). This graph highlights the more efficient learning curve of Annealing-A2C over vanilla A2C and indicates that the annealing period increases sample efficiency of the method.

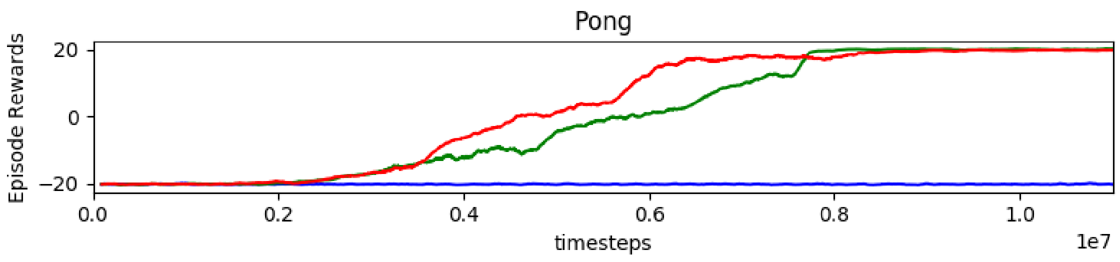


Figure 5.12: Pong: Random (blue) vs. A2C (green) vs. A2C Annealing (red)

5.2.3 Ms. Pacman

Ms. Pacman is a game that is essentially a copy of the famous arcade game Pacman. The player controls a character (Ms. Pacman) from a top down view through a maze. In the maze there are enemies that will try to eat Ms. Pacman. Trails of coins are spread throughout the maze and Ms. Pacman must eat all the coins to clear the level. The game ends when the player loses all three lives. Refer to the appendix for a screen shot showing the player view of Ms. Pacman.

Figure 5.13 shows an agent with a random policy playing Ms. Pacman, the line through the plot indicates the agent's average performance. The random agent finished training with an average reward per episode of 239.97. The agent was able to obtain small constant reward, but failed to realize the dense reward signals hidden deeper in state space of Ms. Pacman.

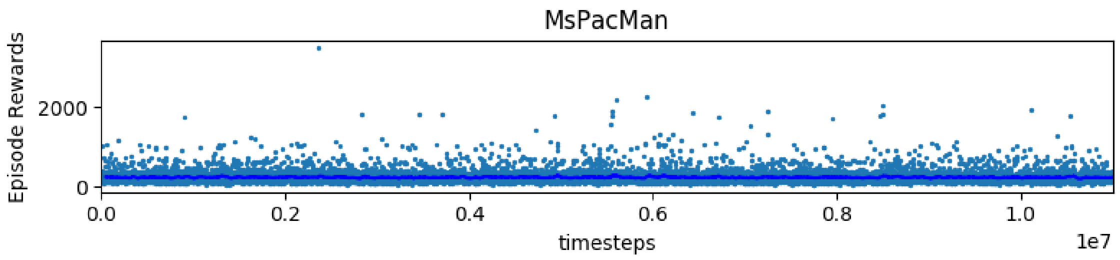


Figure 5.13: Ms. Pacman: Random policy with variance

Figure 5.14 shows the learning curve of DQN in Ms. Pacman. DQN as in the previous games displays pretty nasty variance, with episodes scoring incredibly low late in the training process. Thus the agent finishes below both A2C and Annealing-A2C with a final performance of 1701.37.

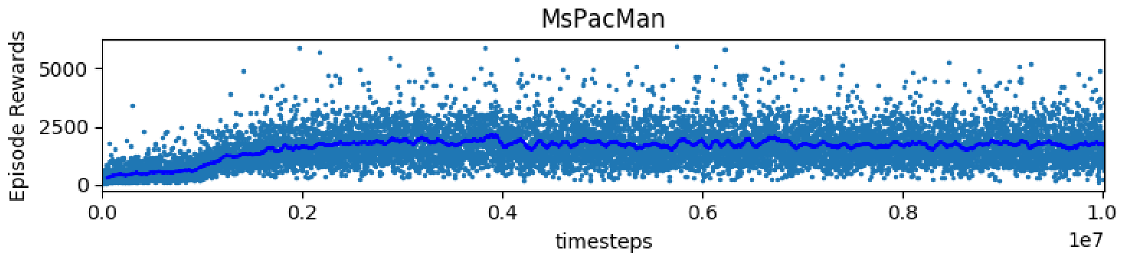


Figure 5.14: Ms. Pacman: DQN with variance

Figure 5.15 shows the learning curve of A2C in Ms. Pacman. An alarming trend occurs after 8M time-steps where average performance starts to decrease. This could be due to a variety of reasons but it indicates that the algorithm lacked the exploratory capability to avoid that performance dip. The final performance ended at 2078.69.

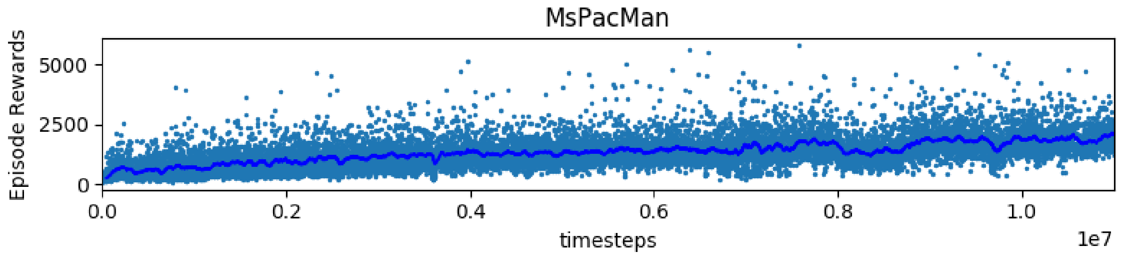


Figure 5.15: Ms. Pacman: A2C policy with variance

Figure 5.16 is a comparison of the random agent (blue), DQN (yellow) and A2C (green). Consistent with previous experiments, A2C has a much slower but steadier learning curve than DQN, aside from the policy failure at 8M time-steps.

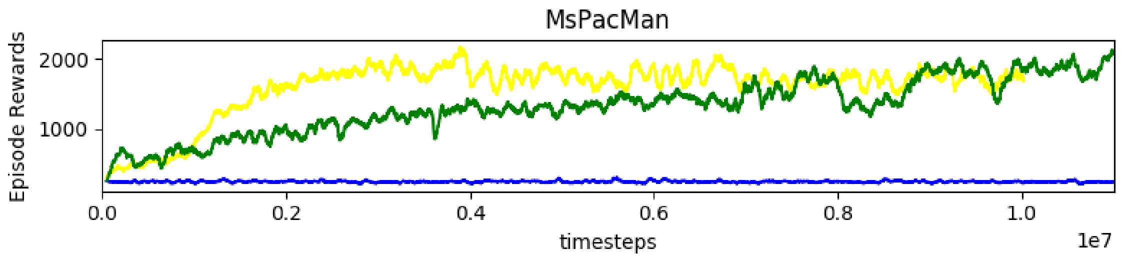


Figure 5.16: Ms. Pacman: Random (blue) vs. DQN (yellow) vs. A2C (green)

Figure 5.17 is a comparison of the random agent (blue), A2C (green) and Annealing-A2C (red). Consistent with other previous experiments, the Annealing-A2C variant was able to learn quicker than vanilla A2C. It avoided the policy failure that A2C faced and achieved higher peak performances than the vanilla version. However, at the very end of training Annealing-A2C finished with a 2002.88 average reward where A2C finished at 2078.69 overtaking the annealing variant. This overtaking and plateauing of performance by the Annealing-A2C agent could have been due to a lack of late game exploratory action, preventing it from clearing the Ms. Pacman maze. It could also be attributed to oscillating around the natural plateau given the hyperparameters of the algorithm.

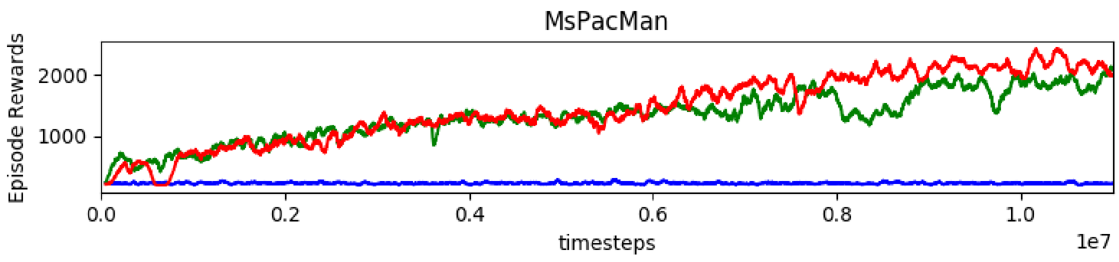


Figure 5.17: Ms. Pacman: Random (blue) vs. A2C (green) vs. A2C Annealing (red)

5.2.4 Q*Bert

Q*Bert is a game where a player controls a character in a unique environment. Each game starts with your character at the top of a pyramid structure, the objective is to jump onto each platform of the structure without falling off or getting eaten by an enemy that follows the player character. There is no 'safety rail' keeping the player within the structure and is free to jump off the structure ending the game at any point. Refer to the appendix for a screen shot showing the player view of Q*Bert.

Figure 5.18 shows an agent with a random policy playing Q*Bert, the line through the plot indicates the agent's average performance. Q*Bert is a dense reward game and it appears that the random agent failed to find any dense reward. Thus, the random agent finished training with an average reward per episode of 156.15.

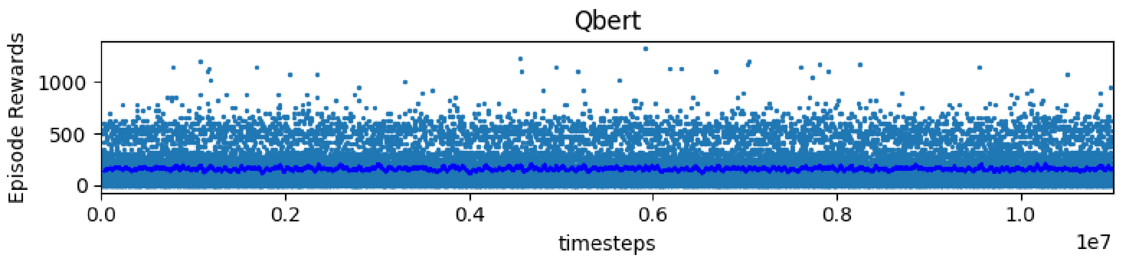


Figure 5.18: Q*bert: Random policy with variance

Figure 5.19 shows a DQN agent playing Q*Bert. What is interesting is that the learning curve shows that dense reward was found early in training around 1M timesteps but the experiences were not utilized moving forward. Perhaps the experience replay buffer was sampled in a way that the agent never learned from the few important reward yielding episodes. Ultimately the relatively flat average reward line indicates that the agent failed to learn anything in this environment, and ended with a final performance of just above the random policy with 484.51.

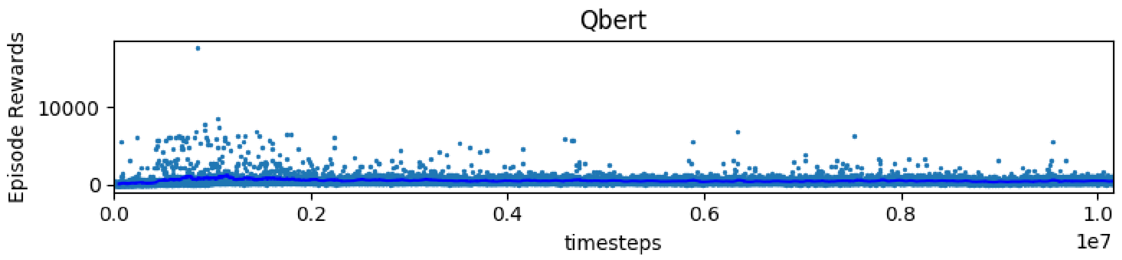


Figure 5.19: Q*bert: DQN with variance

Figure 5.20 shows the A2C agent playing Q*Bert. This graph shows the 'step-ladder' like dense reward structure of the environment. As the dense rewards are discovered, the algorithm trends upwards until all agents are performing at that level.

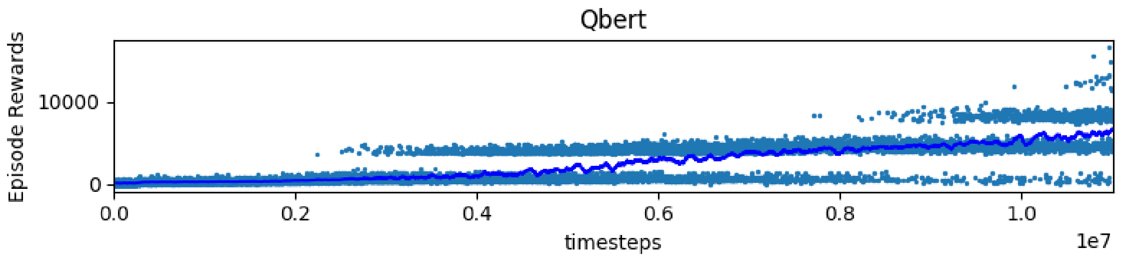


Figure 5.20: Q*bert: A2C policy with variance

Figure 5.21 is a comparison of the random agent (blue), DQN (yellow) and A2C (green) in the Q*Bert environment. This graph shows that DQN failed to learn entirely, and that A2C had a fairly rough time holding onto a good policy, as indicated by the severe drops in average reward around 7M time-steps and 9M time-steps.

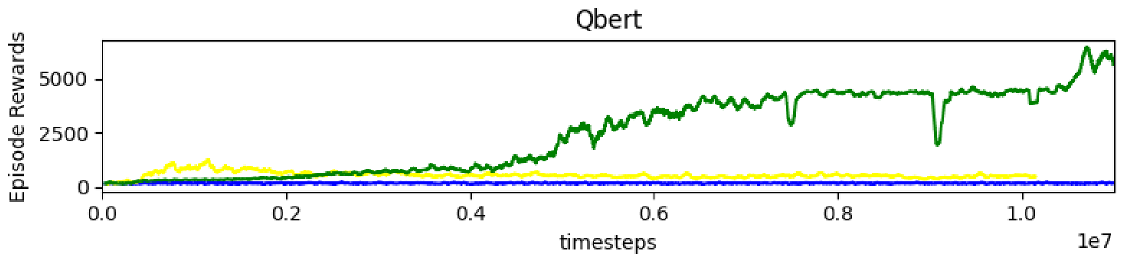


Figure 5.21: Q*bert: Random (blue) vs. DQN (yellow) vs. A2C (green)

Figure .11 is a comparison of the random agent (blue), A2C (green) and Annealing-A2C (red). This graph indicates that the annealing variant benefits from an exploratory phase that prevents policy failure. In Q*Bert agents can jump off the ledge of the environment ending the episode and resulting in very little reward. This indicates that the annealing exploratory phase may have exposed the policy and value functions to these pitfalls and resulted in avoiding harmful situations. Ultimately, Annealing-A2C ended with higher peak performances and an overall performance of 6747.15 over A2C 5669.59. The annealing variant had its greatest success here in a challenging dense reward environment.

5.2.5 Freeway

Freeway is a game which depicts a busy freeway with cars constantly driving across each lane. The player controls a character that must attempt to cross the road, re-

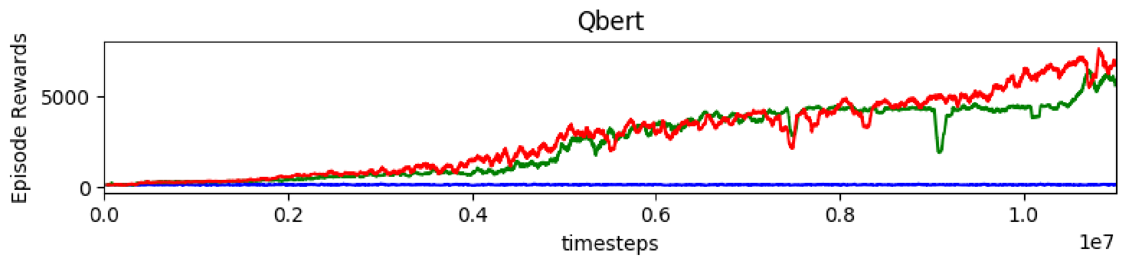


Figure 5.22: Q*bert: Random (blue) vs. A2C (green) vs. A2C Annealing (red)

ceiving a reward for successfully crossing the road. The game was allegedly developed at the same time as Frogger, an eerily similar concept. The main difference in Q*Bert to Frogger is that cars do not kill your character, a collision just sends the character back a few lanes. Refer to the appendix for a screen shot showing the player view of Freeway.

Graphs for Freeway to be included in Appendix, no relevant data concerning my algorithm validation, notable that only DQN was able to learn Freeway, which agrees with other paper results.

5.2.6 Montezuma's Revenge

Montezuma's Revenge is the most challenging game of the six in terms of state space and goals. The player controls an explorer who must navigate from room to room by getting to the far side of the current screen. The explorer may travel for many rooms without yielding any reward. There are many perilous obstacles in each room that may kill the explorer ending the episode. Refer to the appendix for a screen shot showing the player view of the first room in Montezuma's Revenge.

Graphs for Montezuma's Revenge to be included in Appendix, no relevant data concerning my algorithm validation, no algorithms were able to learn this environment, confirms itself as the most challenging environment in the Atari domain.

Chapter 6

CONCLUSION AND FUTURE WORKS

I hope that my systematic experimentation of DQN and A2C prove helpful in educating and demonstrating to others the strengths and weaknesses of both algorithms across a variety of Atari environments. Starting out in this field, I grew frustrated with the lack of comparison studies like this that were reproducible. Therefore, an intention throughout this thesis project was to make my work as transparent and reproducible as possible and I have all experimental data and associated commands used to run them and create graphs available on my github repository [link]. In addition to a transparent and reproducible github repository for the project, I wanted to contribute to a problem I noticed in the field. This led to the development of a quantifiable measure to describe an environments difficulty with respect to exploration, the Exploration Difficulty Metric (EDM). The EDM proved to be both an objective and meaningful measure supported by the success or failure of learning by DQN and A2C in the Atari environments quantified. Its inherent generality makes it extensible to any RL environment, for which I hope the metric can aid future researchers when studying and classifying RL environments

In conclusion, my Annealing-A2C variant hijacks the policy of A2C and exposes the algorithm to states that it would otherwise not visit. This effect is amplified with a population of 16 workers on their own trajectories thanks to an aggressive exploratory period in the first 5% of training. This phase enables the value function and policy approximation to learn more efficiently as shown in the results section above. The annealing variant didnt have a massive impact on final performance, but it had a notable impact on sample efficiency and was able to achieve peak performance quicker than the original algorithm. An inherent benefit of exploration with a population is that the individuals within the population are able to try out many different directions. Once one or a few directions are found to be promising, the algorithm can push all workers in the promising direction, a possibility beyond the scope of single-agent algorithms. This work is not conclusively stating that population based methods (A2C) are hands down better than single agent methods (DQN) but offer more possibilities in regards to exploration. Thus, this work does demonstrate the benefits of diverse, exploratory populations for deep reinforcement learning. This work also proves that action-space exploration for actor-critic methods is worthwhile for consideration in future research.

BIBLIOGRAPHY

[1] Cal Poly Github. `http://www.github.com/CalPoly`.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[3] J. H. Andreae and P. M. Cashin. A learning machine with monologue. *International Journal of Man-Machine Studies*, 1(1):1–20, 1969.

[4] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate. A bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 19–26. AUAI Press, 2009.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

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

[8] R. Bellman and R. Kalaba. *Dynamic programming and modern control.* Academic Press, 1992.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[10] E. S. Bromberg-Martin, M. Matsumoto, and O. Hikosaka. Dopamine in motivational control: rewarding, aversive, and alerting. *Neuron*, 68(5):815–834, 2010.

[11] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[12] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2012.

[13] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[14] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017.

[15] J. Gao and R. Jamidar. Machine learning applications for data center optimization. *Google White Paper*, 2014.

[16] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[17] Z. Hong, T. Shann, S. Su, Y. Chang, and C. Lee. Diversity-driven exploration strategy for deep reinforcement learning. *CoRR*, abs/1802.04564, 2018.

[18] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[19] K. Jarrett, K. Kavukcuoglu, Y. LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[20] J. M. Joyce. Kullback-leibler divergence. In *International Encyclopedia of Statistical Science*, pages 720–722. Springer, 2011.

[21] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[24] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[25] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017.

[26] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[27] V. Mnih. *Machine learning for aerial image labeling.* PhD thesis, University of Toronto (Canada), 2013.

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

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

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

[31] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[32] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[33] W. Pitts. Some observations on the simple neuron circuit. *The bulletin of mathematical biophysics*, 4(3):121–129, Sep 1942.

[34] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. *CoRR*, abs/1706.01905, 2017.

[35] W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[36] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Computer Vision and*

Pattern Recognition (CVPR), 2013 IEEE Conference on, pages 3626–3633. IEEE, 2013.

[37] D. Silver and D. Hassabis. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 2016.

[38] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[39] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. The MIT Press, 2012.

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

[41] G. Tesauro. Td-gammon: A self-teaching backgammon program. In *Applications of Neural Networks*, pages 267–285. Springer, 1995.

[42] S. B. Thrun. Efficient exploration in reinforcement learning. 1992.

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

[44] M. Tokic. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *Annual Conference on Artificial Intelligence*, pages 203–210. Springer, 2010.

[45] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft ii: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

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

.0.1 Q*Bert

.0.2 Montezuma's Revenge

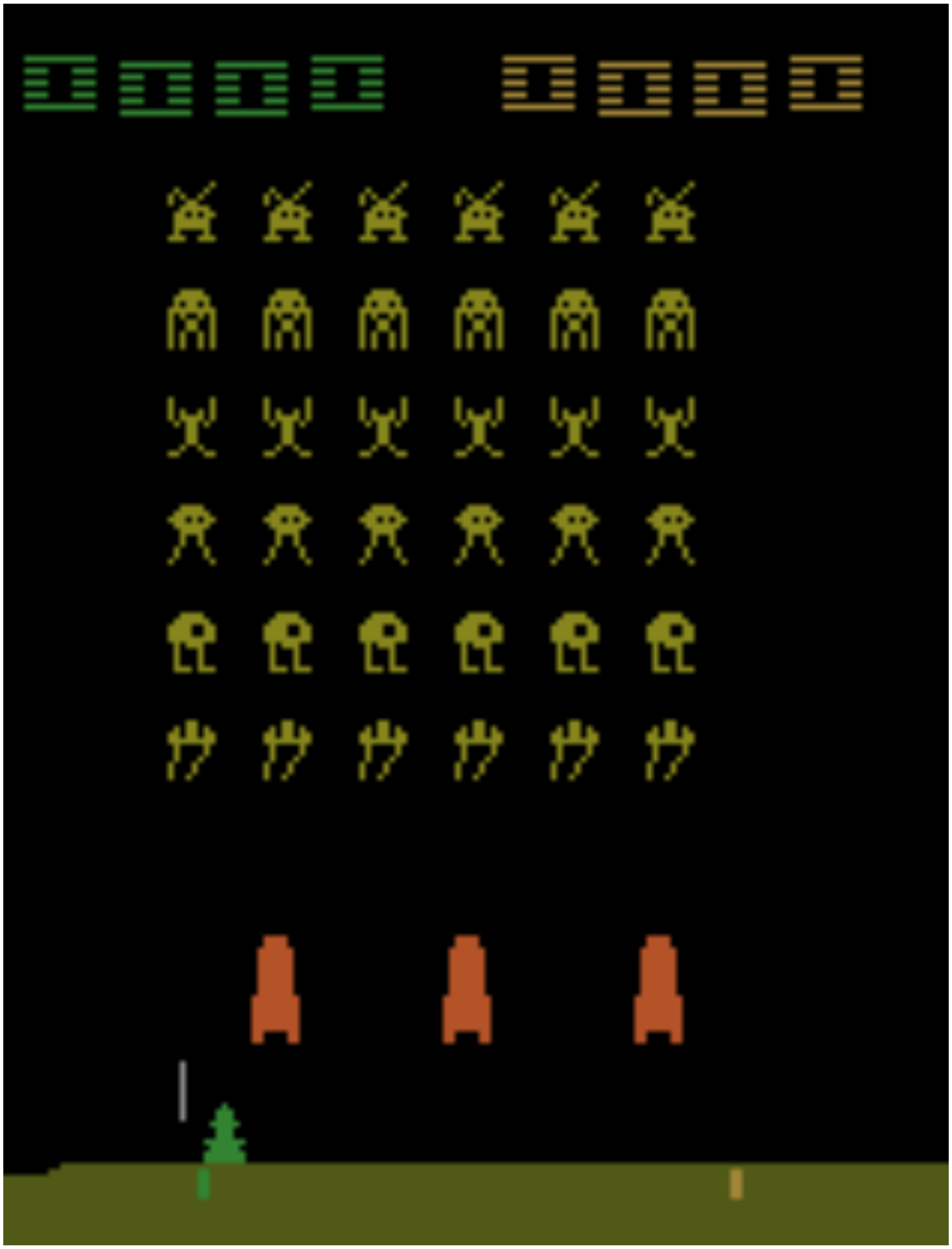


Figure .1: Player View of Space Invaders

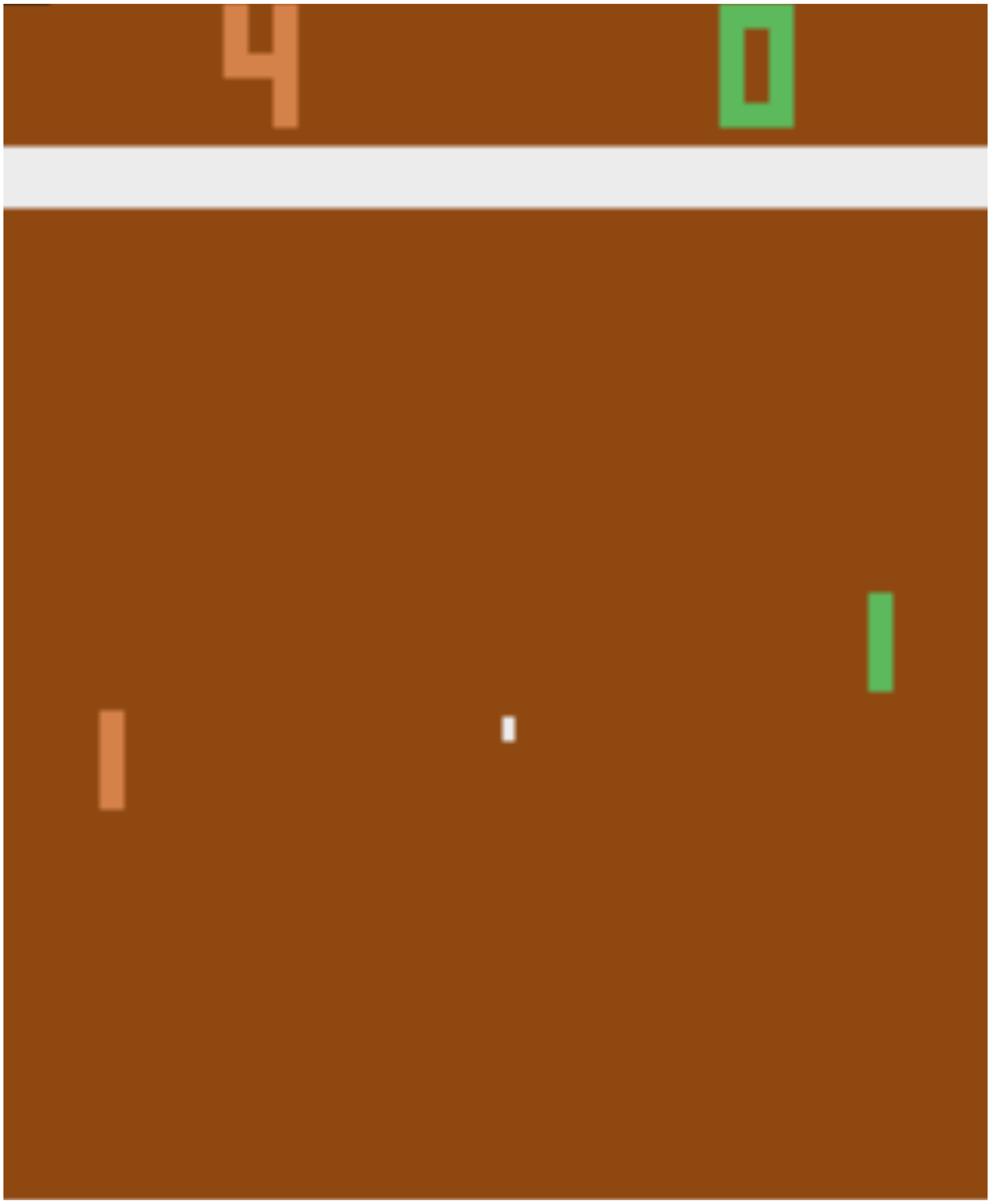


Figure .2: Player View of Pong

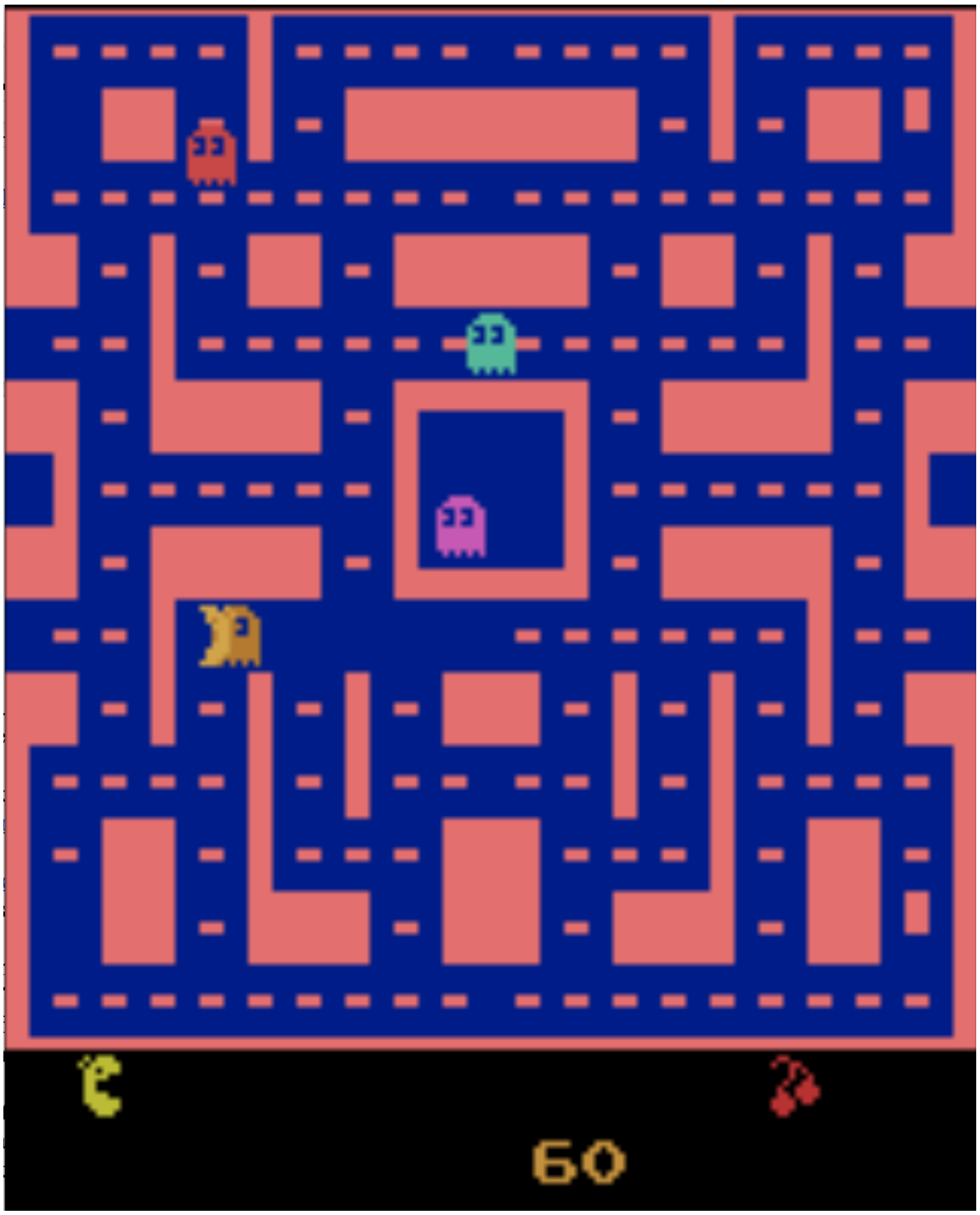


Figure .3: Player View of Ms. Pacman

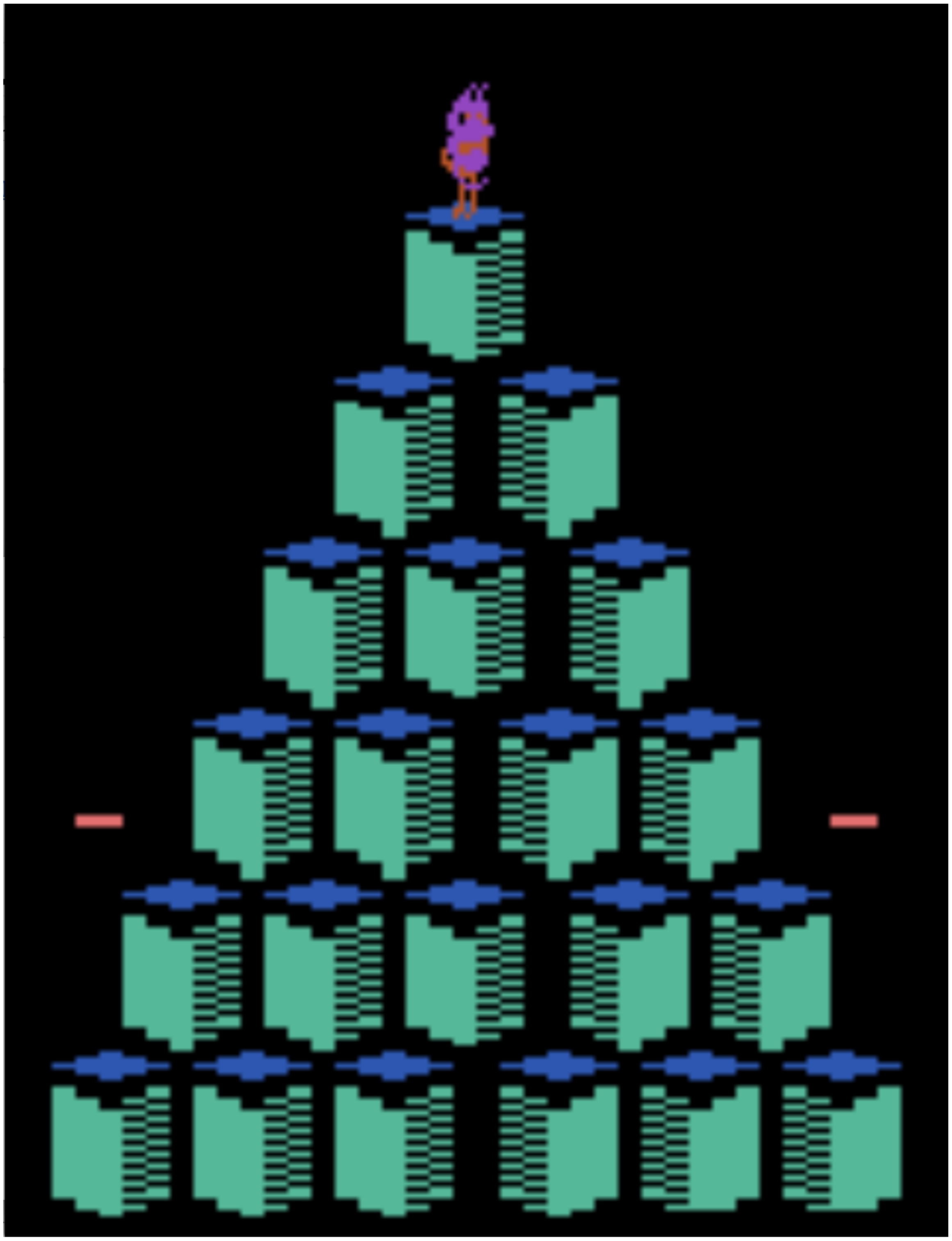


Figure .4: Player View of Q*Bert

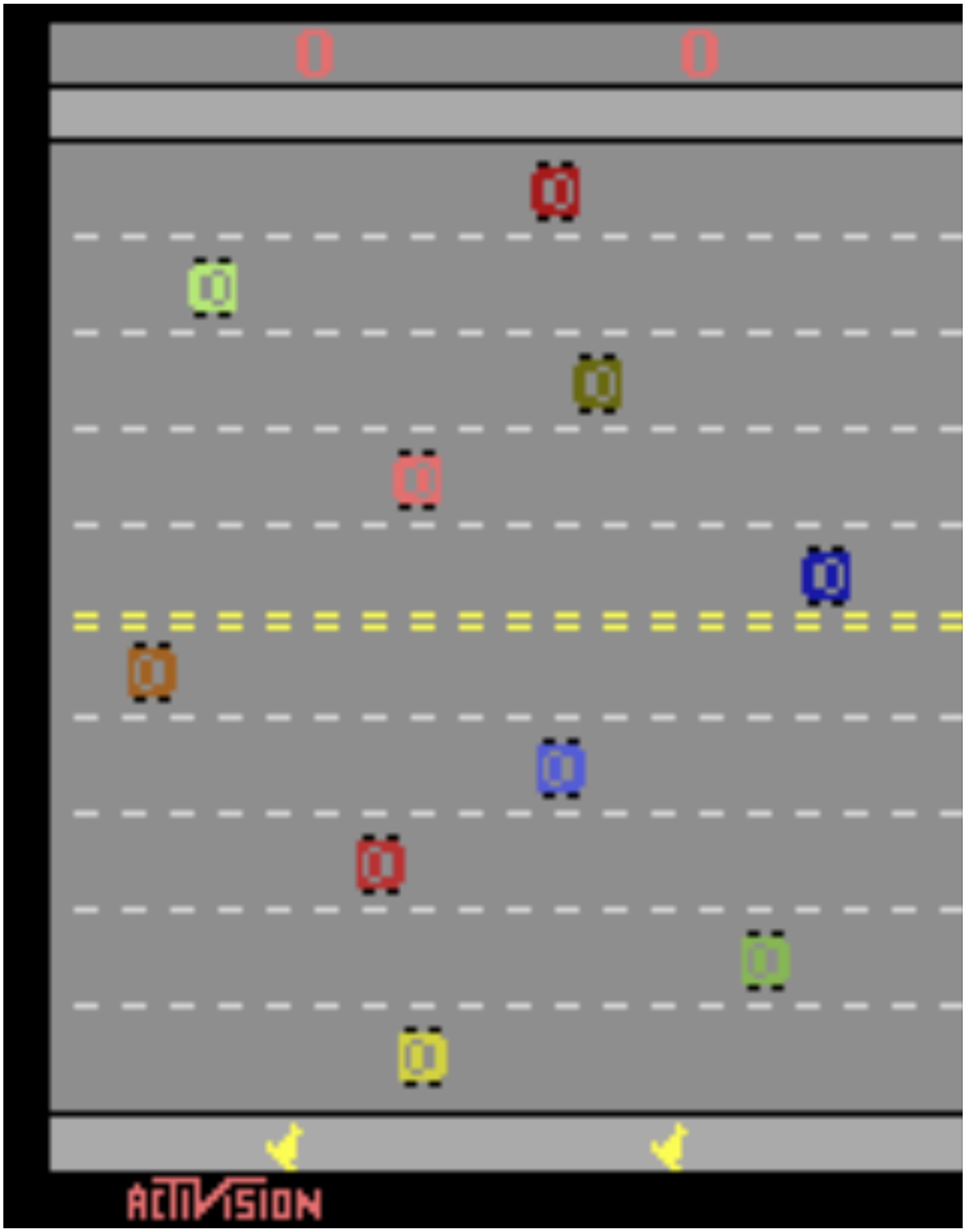


Figure .5: Player View of Freeway

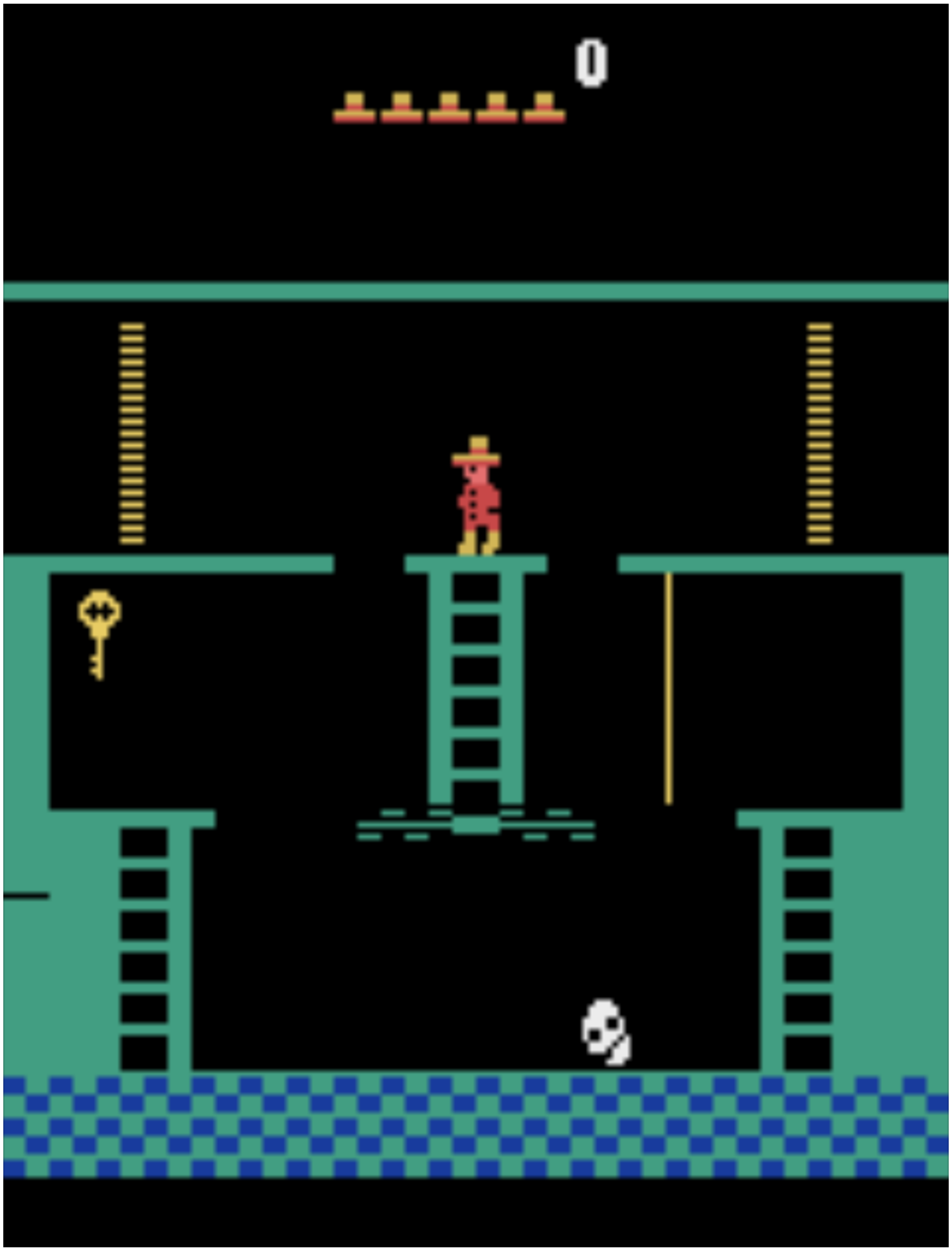


Figure .6: Player View of Montezuma's Revenge

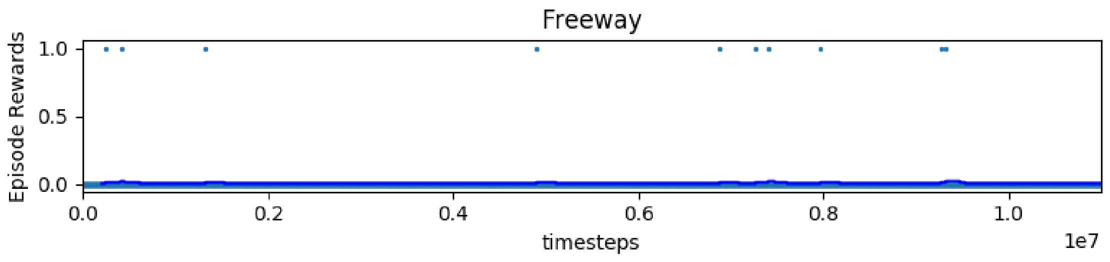


Figure .7: Freeway: Random policy with variance

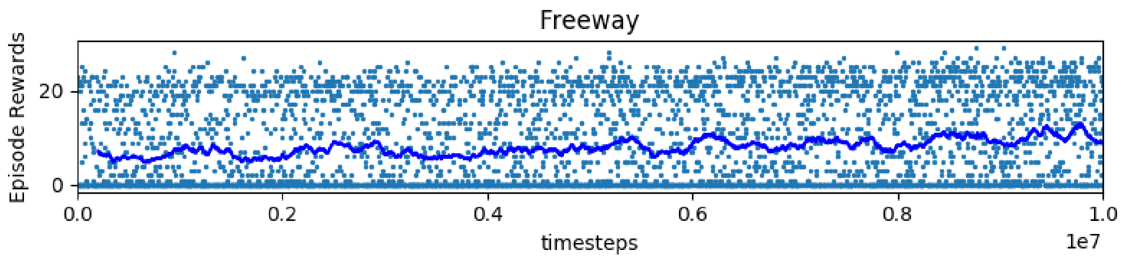


Figure .8: Freeway: DQN with variance

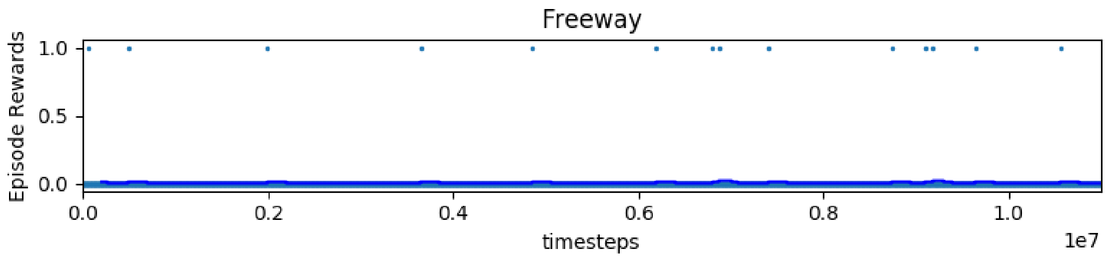


Figure .9: Freeway: A2C policy with variance

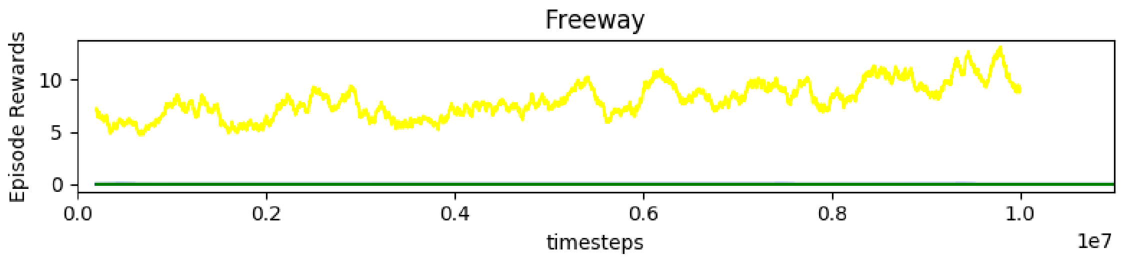


Figure .10: Freeway: Random (blue) vs. DQN (yellow) vs. A2C (green)

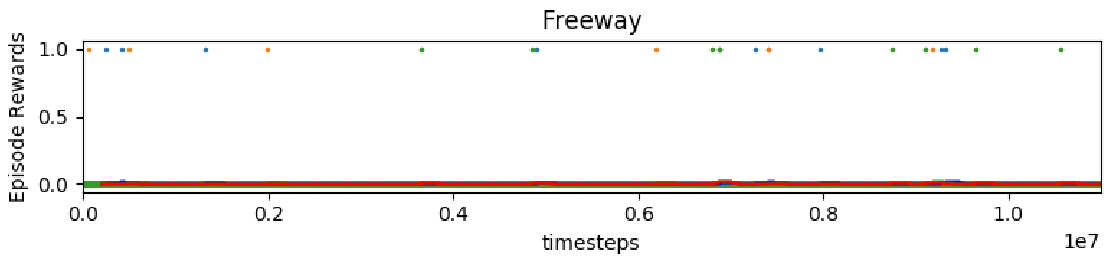


Figure .11: Freeway: Random (blue) vs. A2C (green) vs. A2C Annealing (red)

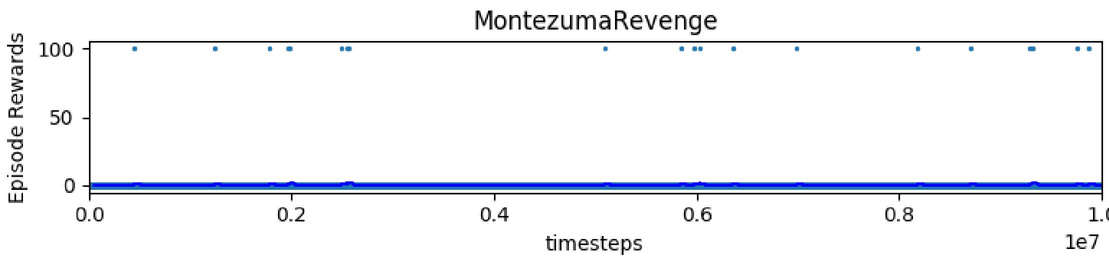


Figure .12: Montezuma's Revenge: Random policy with variance

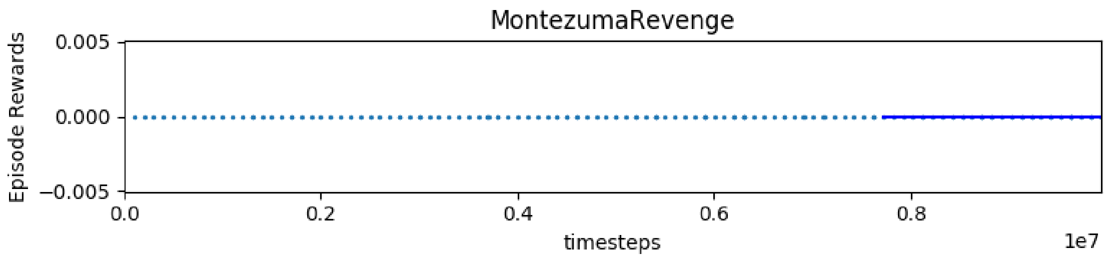


Figure .13: Montezuma's Revenge: DQN with variance

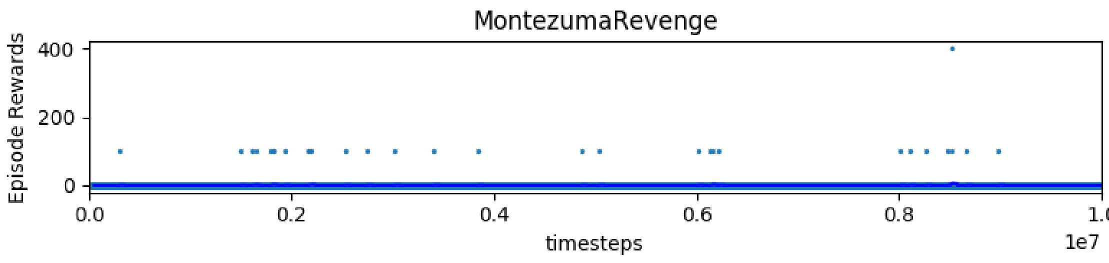


Figure .14: Montezuma's Revenge: A2C policy with variance