Continuous Multilevel Actions in Reinforcement Learning

by

Daniel Mitchell

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Multilevel action selection is a reinforcement learning technique in which an action is broken into two parts, the type and the parameters. When using multilevel action selection in reinforcement learning, one must break the action space into multiple subsets. These subsets are typically disjoint and their union is equal to the original action space. When doing action selection, the subset, representing the action type, is chosen separately from the exact value of the action itself, the parameter values. The majority of research into multilevel action selection focuses on applying it to problems with conceptually distinct action types, such as robot soccer, where an agent can run, turn, tackle, or shoot. However, this is not the only application. In this thesis I focus on a different application of multilevel action selection, where I break down a simple one-dimensional action space into action types in order to focus on specific areas. The goal is to improve learning time by focusing on one area of the action space and disregarding all actions outside of that area, reducing the number of actions to search through. Once an agent has enough experience with actions from a type leading to poor return, it can generalize its experience to the entire action type and instead favour the other types which are more rewarding. I attempt to solve the mountain car and cart pole problems, which I chose for their simple action spaces with a conceptual difference between forwards and backwards thrust. I find that in these problems, multilevel action selection can improve the performance, measured by total return, of a reinforcement learning algorithm.

Acknowledgements

I wish to acknowledge my supervisor Martha Steenstrup for assistance throughout the process of writing this thesis.

I would additionally like to acknowledge Roshan Shariff and Travis Dick, who's code base formed the basis for the majority of experiments I ran. This was a great help as it saved me the time of writing my own simulator, and provided several illuminating insights into techniques for efficiently coding reinforcement learning agents.

Contents

List of Tables

List of Figures

Glossary

$Q(s, a)$
: The action value function of action a when in state s

$V(s)$
: The state value function for state s

$V_d(s)$
: The action type value function of action type d when selecting the high level action from a multi level action space in state s

α_θ
: The step size of the policy parameters

α_θ
: The decay rate of the policy parameter trace

$\alpha_{\mathbf{w}}$
: The step size of the critic weights

$\alpha_{\mathbf{w}}$
: The decay rate of the critic weight trace

α_r
: The step size of the average reward

δ
: The TD Error

θ
: The policy parameters in an actor critic algorithm

$\mathbf{w}$
: The critic weights in an actor critic algorithm

$\mathbf{x}(s)$
: The tilecoded representation of state s

$\mathbf{z}^\theta$
: The policy parameter trace in an actor critic algorithm

$\mathbf{z}^{\mathbf{w}}$

The critic weight trace in an actor critic algorithm

CATV

Critic Action Type Values. An algorithm using multilevel action selection in which the action types are chosen based on the values of the critics for the action parameters associated with that type

Double Actor-Critic

An algorithm using multilevel action selection in which there are two sets of actors and critics, one for the action types and one for the action parameters

Rewarding Actions

Actions which lead to a high overall return relative to other actions

Time Step

A discrete point in time at which an agent has to select an action

Unrewarding Actions

Actions which lead to a low overall return relative to other actions

Chapter 1

Introduction

In this thesis, I investigate the use of multilevel action selection in reinforcement learning. In multilevel action selection, an action comprises two parts, the *action type* and the *action parameters*. In general, all actions in reinforcement learning can be considered as multilevel, with a single type covering the entire action space, and the parameters representing the actual actions. Consider a task of moving over uneven terrain. You can walk in any direction, or jump over or onto an obstacle. When walking, you choose your speed and direction, when jumping, you choose your distance, height, and direction. In this case, walking or jumping represent the action types, while the specifics of speed, direction, distance, and height represent the parameters. The action type represents what to do, and the action parameters represent how to do it. A problem arises when the actions parameters are from a continuum, or even a discrete set with a very large cardinality, instead of a small discrete set of actions. As Masson et al. [15] say, "If we use a continuous action space, we lose the ability to consider differences in kind: all actions must be expressible as a single vector. If we use only discrete actions, we lose the ability to finely tune action selection based on the current state." The solution is to use a hybrid discrete continuous action space, with the discrete part for the action types, and the continuous part for the action parameters.

One of the earliest implementations of multilevel action selection was Rachelson et al. [17]. Their work is not focused on choosing between multiple action

types, but rather includes an option for the agent to either wait for some period of time, or take an action. This wait action effectively creates a second action type, although it has no parameters attached to it.

The majority of the literature on the topic of multilevel action selection is in the form of parametrized action Markov decision processes (PAMDPs), and focuses on problems such as robot soccer [11][15][27][7][25][1], platform [15][25][1], human robot interaction [12], or King of Glory (a multiplayer online battle arena game) [27]. In PAMDPs, the final action the agent takes is a pair comprised of the chosen action type and its parametrization.

The most common method of selecting action types is q learning with epsilon greedy selection [1][25][11][27], although in some cases [12] softmax is used instead. Fan et al. [7] use actor critic instead to choose action types, with a single critic for both the action types and action parameters, and two actors, one for the action types and one for the action parameters.

While in these cases there are two or more conceptually distinct actions types which do not take their actions from the same continuum, this thesis focuses on using multilevel action selection to split an action space in which all actions lie on a single continuum into multiple parts. For example, if the domain of an action space is from -1 to 1, this can be split into a positive and negative action type with domains from 0 to 1 and -1 to 0 respectively. The other major difference between my work and PAMDPs is that in my work all action types have the same parameters, however in the existing literature, they are different.

In some states, there may be clusters of rewarding actions and unrewarding actions. In this case, the whole area of the action space containing rewarding actions can be favoured, and other areas of the action space focussed on less. I will show that breaking up an action space can be used to reduce the time it takes for an agent to learn from a given state by favouring the action type with good actions. The agent will still initially choose actions from the unrewarding action type, but when they all give poor return, it can determine not to choose those actions quicker than without multilevel action selection, where it would have to explore the whole space corresponding to the unrewarding action type.

When solving a task using multilevel action selection, the choice of action parameter values can be done just like any other action selection in reinforcement learning, but instead of choosing from the entire action space, the agent can only choose from the subspace associated with the chosen action type. However, the the choice of action type and action parameter values can be done in any order. You can first choose an action type, and then the parameter values from that type, or you can first choose the parameter values for each type, and then the type based on the specific values of the parameters. To demonstrate this, look at the above example again. If you are faced with a boulder in your path, you can either jump over it or walk around it. You could first decide if you want to jump or walk, then work out the details of whichever you choose. Alternately, you could first ask "How would I jump over this?" and "How would I walk around it?", then once you have a way of doing each, decide which is better.

I first give a brief overview of the aspects of reinforcement learning pertinent to this thesis.

Next, I focus on the specifics of multilevel actions and how I am using them. This includes a definition of multilevel actions and a discussion of what situations it is appropriate to use them in. I also cover various methods of implementing multilevel actions and what must be considered when doing so.

Then I explain the problems I tested on and the specifics of the experimental setup. The experiments were done on the mountain car and cart pole environments, chosen because they have action spaces that can intuitively be used with multilevel action selection. They involved comparing the performance of multiple implementations of multilevel actions against each other, measured by the total return, as well as how they compare with non multilevel action selection methods.

Chapter 2

Brief Overview of Reinforcement Learning

In this chapter, I go over the basics of reinforcement learning and the aspects of it relevant to the rest of the thesis.

2.1 Basics of Reinforcement Learning

Reinforcement learning is learning through trial and error interaction with the environment in order to maximize a reward signal. Reinforcement learning has its roots in animal learning theory, and has many similarities to the methods by which animals learn.

The agent is not restricted to a specific solution that it is taught by an expert. This means the set of solutions a reinforcement learning agent is capable of finding is potentially bigger than the set of solutions humans have discovered. It is therefore possible to learn new and innovative solutions through reinforcement learning that would not otherwise be found if agents were directly taught solutions.

2.2 How Reinforcement Learning Works

The basic behaviour of a reinforcement learning agent is as follows. The agent first makes an observation of its environment. Then it selects and takes an action based on this observation. The environment may change based on the agent's action. The agent observes the environment again and interprets part

of its observation as a reward (indicating how good the action was). The other part of this observation is used as the context for selecting the next action as this cycle repeats.

An agent's learning process is based on the following components, the policy, the reward, and the value function. That is, how it will decide what to do next, how well it is performing, and how it evaluates its current situation. There are two additional concepts, the state and the action, which may be used by the agent in the policy and value function, and by the environment to determine the reward.

The state is the representation of the agent's current observation of its environment, as well as potentially a summary of its previous interactions with the environment. Some agents do not observe the state, but these cases are not discussed here. It is important to keep in mind the distinction between the agent's observation and the true state of the environment. In some cases they are identical, but in some the agent is missing information. On the other hand, the agent may keep track of its past actions, in which case the agent's state can have more information than the environment's state. For example, if an agent is a person walking down the street, the state they observe is limited to what they can see in front of them (for the sake of this example, ignore other senses like hearing). The environment's state, on the other hand, also includes everything behind them and out of view.

Additionally, in some problems it may not be necessary to maintain a state representation in order to take appropriate actions. This is usually in problems where the environment does not change from one action to the next, so it is not necessary to keep track of.

The actions are the choices that the agent can make at each timestep. In some cases, the available actions may depend on the current state, in others the same set of actions are universally available to the agent regardless of its state. Actions may be a discrete set of choices (such as 'go forward', 'turn left', and 'turn right'), or a continuum of choices that can be selected from (such as move forward with some speed between 0 and the maximum speed).

The reward is a numerical signal representing how good the most recently

taken action was for the agent (i.e. how much it moves the agent towards the goal as defined by the experimenter). The agent's goal is to maximize a function of the reward signal it receives, such as the average reward over time, or the maximum total reward. It is important to note that an agent does not have any concept of having solved the problem as set by the experimenter; its only feedback is the reward signal it receives. When an experimenter decides on what reward signal to use for a reinforcement learning problem, it is important to make sure there are no ways that the agent can achieve a high reward other than by solving the experimenter's problem.

The return is another concept closely related to reward. The agent receives a reward every time it takes an action, and the return is the sum of the reward from the current time over the rest of the agent's lifetime.

The value function expresses how good a state or action is for the agent. It maps either a state, action, or a state-action pair to a number, with a higher result being more favourable. The value function is based on the agent's history of interactions with the environment, the actions it has taken and the rewards it has received. It is equivalent to how good a state or action is to the agent in the long term, as opposed to the present. This is similar to the concept of return, which represents the total reward that the agent expects to receive following its policy from the current state. To see the importance of this distinction between long term return and immediate reward, consider a case where the agent can move to a state from which it can receive a really high reward, but doing so requires taking an action with a low reward first. This is a temporary sacrifice in reward for a long term gain. Figure 2.1 is an example of this. Taking the top path (action 1), may at first look more appealing than the bottom path (action 2) because it has higher reward. However, when you consider the state resulting from each action, the total reward over two time steps will be two for the top path but nine for the bottom path.

The policy determines how the agent decides what action to choose. It is analogous to a function mapping the the current state to the action which the agent takes. In some cases, an agent learns its policy as it interacts with the environment. In other cases, an agent has a fixed policy that never changes.

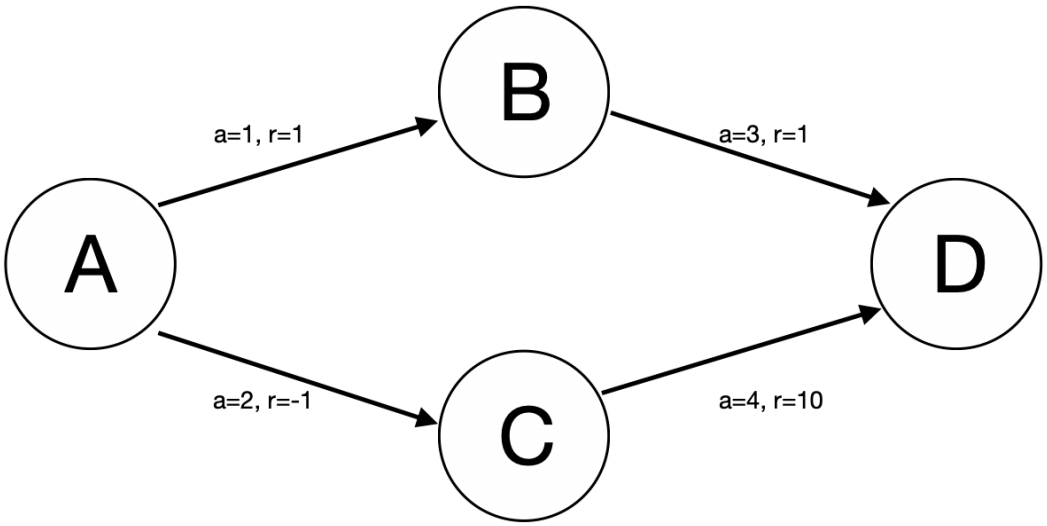


Figure 2.1: Return vs. Reward

What type of policy an agent has is determined by the experimenter. Some policies select the action based directly on the value function, choosing the one that will lead to the highest return. Agents using other policies, such as those in policy gradient methods, don't directly choose actions with the value function. They instead use a parameterized policy for selecting actions, but may learn its parameter values through the value function. An agent may also have a policy which does not follow the most rewarding action, but instead tries to fully explore the state and action space, being more likely to choose actions it has not taken recently.

2.3 Exploration vs. Exploitation

A policy that always chooses the best action, i.e. the action it currently thinks will lead to the best return, is known as a greedy policy. Oftentimes however, such a policy is not the optimal policy[1] an agent could use.

The agent may either exploit its current knowledge by taking the most rewarding action, or explore the environment by trying new actions. This trade off between exploration and exploitation is essential, because if the agent never explores, it may think it has a good policy for receiving a high return,

[1]The optimal policy is the hypothetical policy that would lead to equal or higher return than all other policies from any state.

but miss out on discovering one that is even better. On the other hand, there is a cost to exploring. If an agent spends too much time exploring, it may miss out on opportunities to take high valued actions and its total return will be low.

Additionally, if the environment is nonstationary, such that the most rewarding action from a state changes drastically over time, it is important to keep exploring, as the best actions now may not be ideal in the future. In a stationary environment, the amount of exploration can be decreased over time as the agent gains more experience and is closer to finding the optimal policy. This is because the better the policy the agent is following, the more likely it is to choose the action leading to the highest possible return. Since the agent cannot know if it's current policy is optimal or not, it should never completely stop exploration.

An example of this trade off between exploration and exploitation is the epsilon greedy policy. Every time the agent chooses an action, there is a small chance, represented by epsilon, that it chooses randomly instead of taking its currently highest valued action. In a stationary environment, as the agent gains experience in how rewarding certain action are, epsilon can be decreased, causing exploration to happen less often. Another example is actor-critic on a continuous action space, where an action is chosen from a probability distribution, calculated based on parameters learned by the agent, with the mean representing the most preferred action.[2] This causes every action to have some amount of exploration, but this exploration is focused closer to the most preferred action (the closer an action is to the most preferred action, the more likely it is to be explored). As the agent gains experience, it will learn parameters leading to a smaller variance of the distribution, causing the chosen actions to be closer to the preferred action.

[2]Actor-critic learns preferences for actions instead of values

2.4 Continuing and Episodic Problems

There are two types of reinforcement learning problems: episodic and continuing. Episodic problems have a definite end point or goal state. The agent solves the problem by reaching that goal in the most rewarding manner it can. Once the agent reaches the goal, it will start the problem over again (keeping its knowledge) from the start point in a new episode. These problems are most like games or isolated tasks which have a clear start and end.

On the other hand, continuing problems do not have a defined end point. The agent will keep going indefinitely. There are no success or failure states in continuing problems, just actions resulting in high or low reward which the agent will try to take and avoid respectively. Most real world problems can be classified as continuing problems.

2.5 Large State and Action Spaces

The agent selects actions in the simplest case based on the action values. For example, the agent may select the action with the highest value, with a small probability of choosing a random action instead for the purposes of exploration. An alternative is to use a policy gradient algorithm, where instead of using the action values, the agent uses a function mapping a state action pair to the probability of choosing that action from that state. A type of policy gradient algorithm is actor-critic, which has two parts, the actor which is responsible for selecting the action, and the critic which determines how good a state is. The critic will then give feedback to the actor based on how good of a state the agent ended up in after taking an action. If the critic says that the resulting state is good, then the actor will know to increase its preference for the most recent action from the previous state.

When the state and action spaces are small, one can use tabular methods. These are when instead of a continuous function mapping the state to a value, a simple one to one lookup table is used. Every state has a value associated with it stored in the agent's memory, and when the agent reaches a state, it simply looks up what that state's value is.

One issue that may arise is when the state or action spaces are very large is that it is no longer practical to keep track of every state-action pair individually. Tabular methods are still possible, but they may be inefficient both in terms of memory and running time. A simple solution is coarse coding, where a the state space is broken up into buckets, equivalence classes representing a range of states that are close together. Every state falling within one bucket is treated the same, resulting in a new state space with fewer elements. These buckets may overlap, so that a single state falls into more than one bucket. If the buckets are disjoint, then it is known as a state aggregation.

A more complex method is tile coding, which can create a finer grained approximation of the original state space than coarse coding. To do this, we can use multiple state aggregations, or tilings, layered over the space, each with a slight offset. Any state in the original space will fall into one tile in each tiling, and the new state representation is the set of these tiles.

The cardinality of the action space can affect the manner in which actions are selected. If the cardinality is 'small', then each action can be looked at individually and compared to each other. If there is a very large (but still finite) amount of actions, this is technically still possible, but becomes inefficient due the number of comparisons to be made. When the action space has an infinite cardinality, this method of comparing actions is no longer possible, and we need a different method of choosing actions.

The goal is to transform action spaces with this large or infinite cardinality into an alternate space of low cardinality. One possible solution is coarse coding, where you can discretize (for a continuous action space) or group (for a discrete action space) the action space into small sized buckets, resulting in a smaller amount of actions spread out along the domain of the action space.[3] The problem with this is that the buckets have to be large enough to reduce the computational complexity (by having fewer buckets to choose between) while also being small enough to maintain the precision of the original action space (i.e. being able to discriminate between two actions close together

[3]Note that any action space with an unbounded domain would still require infinitely many buckets (or have at least one bucket of infinite size).

which lead to different results). This also requires all the actions in the action space be in one dimension (or at least a small amount of dimensions). If the actions to choose between are totally disparate (e.g. they are not from the same dimension, such as run, jump, or turn around) which cannot be grouped together, this method is unlikely to work. However, most such action spaces relatively small sized, so this is not usually an issue. In the edge case of a large disparate action space, these solutions are unlikely to work, however these problems are not examined in the rest of this thesis.

When choosing actions from a continuum without first discretizing the space, the simple tabular method of looking at individual actions is no longer possible. An alternative to this is to use a function mapping the action to some metric used to indicate which action the agent prefers. One type of function that can be used for this is a probability distribution over the action space with the mean at the most preferred action [26]. An action can then be sampled from this distribution, and the agent will update the parameters of the distribution based on the resulting state and reward. The exact parameters will depend on the type of distribution being used, for a Gaussian distribution, they are the mean and standard deviation.

2.6 Multiple Dimensions

An action space is multidimensional if there are two actions which have different parameters. For example, the accelerations of 1 $\frac{m}{s^2}$ and -1 $\frac{m}{s^2}$ have the same parameters and are from the same dimension, but the acceleration 1 $\frac{m}{s^2}$ and the angular acceleration 1 $\frac{\text{rad}}{s^2}$ are not. When working with problems with actions spaces where the agent can select actions from multiple dimensions, there are two cases to consider. First is the case where the agent can act in multiple dimensions simultaneously. For example, consider a robot arm where the rotation of each joint is its own dimension, the agent can choose to move multiple joints at the same time. In this case, to determine the action, the agent can determine its action individually in each dimension, and then final action will be the combination of all of them together. The second case is

where the agent must can only act in a single dimension at a time. For example, consider travelling by bicycle. At some points the agent has to get off and walk the bike, at others it can ride the bike, however it can only ever pick one or the other and cannot simultaneously do both. In this case the agent has to choose between its available actions in each dimension to see which dimension contains the most preferred action. If the action space is small, this can easily be done by directly comparing the actions.

Now consider a problem with actions in multiple dimensions, some of which are continuums. Each individual action dimension may have a corresponding 'best' action, but only one may be selected as the agent's action for the current time step. The policy gradient method of choosing the action from a single probability distribution no longer works, as the actions are taken from multiple continuums, so the distribution cannot be defined over all of them. We need a method of choosing a multilevel action which combines the finite choice of which dimension, or action type, to use with the infinite choice of action, or parameter values, from any one dimension's action space. Additionally, there may be cases where the action space is very large, but in a given state, only a small subspace will have actions that are preferable, because all actions outside of that subspace are much less rewarding than those inside of it. In this case, splitting the space up and favouring the more rewarding subspace can potentially reduce the search time to find the optimal action, as the agent is only considering a fraction of the total action space.

Chapter 3

Approaches to Multilevel Actions

This chapter will explore the concept of multilevel action selection in continuous action spaces and how it can be applied to solve problems. The first section covers what multilevel action selection is and how it is used. Then I will cover the reasons to use multilevel action selection and what situations it is advantageous in. Then I give a brief overview of prior work regarding multilevel action selection and the advantages and disadvantages of these different methods. The next section then goes into detail on the specific algorithms I have developed and the way they are implemented.

Recall the example from chapter 1 of moving by either walking or jumping. Since you cannot jump and walk at the same time, a combination of the two actions is not possible (for contrast, compare this to reaching for an object with your arm; there are multiple joints that can all move independently and simultaneously). We need a method to represent this continuous action space which is broken into two disparate sets.

3.1 Definition of Multilevel Actions

Multilevel action selection is a type of action selection in which the agent has two (or more) levels of decision making. Each time the agent selects an action, it chooses both a high level and low level action. The high level action represents an action type, whereas the low level action represents parameter values for that type. The subset of the action space the action parameter values are drawn from depends on which action type is chosen. If the agent

selects an action from one action type, the action cannot also be from another action type at the same time.

3.2 Reasons to Use Multilevel Actions

An obvious question to ask is what advantage is there from using multilevel action selection instead of just choosing the action parameters directly.

In the case of discrete actions, choosing low level actions directly is still a possibility, however we have to be aware of how big the action space is. If each action type has only a couple actions associated with it, then multilevel actions are probably not necessary. However, as the size of the action space for each action type increases, the total number of actions increases even faster at a rate proportional to the number of action types. Consider the case of a continuous action space that has been discretized with a very high cardinality. In these cases, using multilevel actions breaks up the action spaces so fewer actions need to be considered at once. By breaking them into separate action types, then we only need to select an action from among the actions corresponding to that type, rather than from the entire action space.

In continuous action spaces, we can use the same justification to split the action space so we have smaller spaces to search over, however there is a more important factor. If the different action types are in different dimensions (i.e. they cannot be drawn from the same continuum), then we cannot combine them together in one action space the same way we could with discrete action spaces. In the case of bounded action spaces, one may expect to be able to combine them by concatenating one after the other. If we have two continua both ranging from 0 to 1, you would put the second one after the first so the range 1 to 2 corresponds to the second continuum. The problem is an algorithm like actor-critic which selects actions from a distribution will have strange behaviour around the point where the continua are concatenated. If a distribution has a mean just less than one (in the first continuum), the tail of the distribution may be greater than one and often select actions there (in the second continuum).

We can use any continuous action selection method within any one continuum, but we need a method of choosing between the multiple continuums. By associating each continuum with an action type, we can use multilevel actions to solve such problems. We can also use this technique to break up single continuum into multiple parts. This may be advantageous if two or more parts of the continuum are significantly different in terms of which states they produce good actions from. This means for any one state, the part of the continuum which contains highly rewarding actions can be favoured over equally exploring the entire action space. The agent will need to spend time exploring all action types, in order to find which are rewarding and which are unrewarding, but after a few experiences in each, it can determine which types are which quicker than if it had to explore the whole action space. There should still be some amount of exploration among the action types, so the agent does not get stuck only looking at a single action type when more rewarding actions are still undiscovered in others.

3.3 Methods of Choosing Multilevel Actions

The way to select a multilevel action depends greatly on what learning algorithm you are using. If you have an action value method, the possibilities open to you will be different than with a policy gradient method.

One method of selecting multilevel actions is to first select an action type. The action type selection is based on which type the agent decides to have more rewarding actions. Then based on which action type was chosen, the agent will only consider the corresponding action parameters for selection. In the example above, this would correspond to first choosing how to move: walk or jump. Then once that decision is made, the agent will look at the details of the chosen action type and determine the specifics, such as the speed and direction. [25][27]

An alternate method is to select action parameters from each type separately. Then, the agent will compare each of the selected action parameters, and choose a type based on their action values. The advantage is that the

choice of the action type will be based on the actual action to be taken, not just on the action type. This means any exploration that may take place is already factored in to the action values when the agent chooses the action. The downside is that it is dependant on action values, which may not be a part of the learning algorithm being used, especially if the action parameters are continuous. Going back to the example, this would mean the agent calculates both the best way it can walk and the best way it can jump. The agent will then compare these two, whether walking in that specific way is better or worse than making the specific jump. [1][15]

3.4 Algorithms

When designing a learning algorithm which incorporates multilevel action selection, how to select the actions is the first thing to consider. First, how should the action parameters be selected? This can be done with any learning algorithm, such as the action value or actor-critic methods mentioned previously in this thesis, given some adjustments to handle multiple action spaces split between the different action types. Ways to do this include duplicating the procedure for action selection to have one corresponding to each action type (e.g. in actor-critic, have multiple actors and critics), or include the selected action type as an extra feature of the state space that is considered when selecting the action parameters. It is also possible to select action parametrizations with different algorithms for each action type. For example, if one action type only contains a few discrete actions, but another contains a continuum, the parametrizations for the discrete action type can be chosen with a tabular method and the parametrizations for the continuous action type can be chosen with actor-critic.

Next, consider how to select the action types. While some problems have a large number of action types, I will assume it is sufficiently small in the problems discussed in this thesis. The action type selection can be done with more options than the action parameter value selection. The action type and action parameter selection may be completely independent, never sharing

information with each other, or they may share some elements between them (e.g. in actor-critic, the two levels could share a critic, or each have their own).

When I designed my algorithm, I specifically wanted to use multilevel action selection with tasks that had continuous action spaces. I also wanted to see how well I could do with a simple algorithm, using linear function approximation with binary features. I chose to use actor-critic [4] at the low level, with one actor for each action type.

For the action type selection, I tried several methods. One of the most promising ones were using a completely distinct algorithm for action type selection, having a second actor and critic to select action types. I refer to this as the double actor-critic algorithm, shown in algorithm 1. I also tried tying the two levels closer together by using a separate critic for each action type and using them to represent *action type values*. I then used these values to choose the action type with epsilon greedy selection. I refer to this as the critic action type value (CATV) algorithm, shown in algorithm 2. The comparisons between these can be found in chapter 4. I looked at some other methods as well, such as learning action type values with SARSA and choosing action types either epsilon greedily or randomly weighted by their action type values, but these were abandoned due to poor performance and not experimented on further.

The general form of the algorithm for selecting high and low level actions is as follows. Depending on its current state, the agent chooses an action type by comparing the action type values or preferences. Then, it uses the actor associated with that type to select the action. Based on the observed response to the selected action, the agent updates the actor and critic of the selected action type. If using eligibility traces, the unused actor also needs to be updated in order to decay the eligibility trace properly.

Algorithm 1 Double Actor-Critic algorithm for multilevel action selection with average reward

Input: two differentiable policy parametrizations $\pi(a|s,\theta)$ and $\hat{\pi}(a|s,\hat{\theta})$
Input: a feature vector $x(s)$
Algorithm Parameters: actor eligibility trace decay λ_θ, critic eligibility trace decay $\lambda_{\mathbf{w}}$, actor weight step sizes α_θ and $\hat{\alpha}_\theta$, critic weight step sizes $\alpha_{\mathbf{w}}$ and $\hat{\alpha}_{\mathbf{w}}$, average reward step size α_r, number of action types d

for i from 1 to d **do**
    Initialize policy parameters $\theta_i \in \mathbb{R}^n$ and critic weights $\mathbf{w}_i \in \mathbb{R}^m$
    $\mathbf{z}_i^\theta \leftarrow \mathbf{0}^n$ (actor eligibility trace)
    $\mathbf{z}_i^{\mathbf{w}} \leftarrow \mathbf{0}^m$ (critic eligibility trace)
Initialize action type policy parameters $\hat{\theta} \in \mathbb{R}^{\hat{n}}$ and action type critic weights $\hat{\mathbf{w}} \in \mathbb{R}^{\hat{m}}$
$\mathbf{z}^{\hat{\theta}} \leftarrow \mathbf{0}^{\hat{n}}$ (action type actor eligibility trace)
$\mathbf{z}^{\hat{\mathbf{w}}} \leftarrow \mathbf{0}^{\hat{m}}$ (action type critic eligibility trace)
Initialize $\bar{r} \in \mathbb{R}$
Choose initial state s
while True **do**
    Choose action type D from $\hat{\pi}(D|s,\hat{\theta})$
    Choose action parameters a from $\pi_D(a|s)$
    Take action a, observe state s' and reward r
    $\hat{\delta} \leftarrow r - \bar{r} + \hat{\mathbf{w}}\mathbf{x}(s') - \hat{\mathbf{w}}\mathbf{x}(s)$
    $\delta \leftarrow r - \bar{r} + \mathbf{w}_D\mathbf{x}(s') - \mathbf{w}_D\mathbf{x}(s)$
    $\bar{r} \leftarrow \bar{r} + \alpha_r\delta$
    for i from 1 to d **do**
        $\mathbf{z}_i^{\mathbf{w}} \leftarrow \lambda_{\mathbf{w}}\mathbf{z}_i^{\mathbf{w}}$
        $\mathbf{z}_i^\theta \leftarrow \lambda_\theta\mathbf{z}_i^\theta$
    $\mathbf{z}_D^{\mathbf{w}} \leftarrow \mathbf{z}_D^{\mathbf{w}} + \mathbf{x}(s)$
    $\mathbf{z}_D^\theta \leftarrow \mathbf{z}_D^\theta + \frac{\nabla_\theta \pi_D(a|s,\theta)}{\pi_D(a|s,\theta)}$
    $\mathbf{z}^{\hat{\mathbf{w}}} \leftarrow \lambda_{\mathbf{w}}\mathbf{z}^{\hat{\mathbf{w}}} + \mathbf{x}(s)$
    $\mathbf{z}^{\hat{\theta}} \leftarrow \lambda_\theta\mathbf{z}^{\hat{\theta}} + \frac{\nabla_{\hat{\theta}} \hat{\pi}_D(a|s,\hat{\theta})}{\hat{\pi}_D(a|s,\hat{\theta})}$
    for i from 1 to d **do**
        $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_{\mathbf{w}}\delta\mathbf{z}_i^{\mathbf{w}}$
        $\theta_i \leftarrow \theta_i + \alpha_\theta\delta\mathbf{z}_i^\theta$
    $\hat{\mathbf{w}} \leftarrow \hat{\mathbf{w}} + \alpha_{\mathbf{w}}\hat{\delta}\mathbf{z}^{\hat{\mathbf{w}}}$
    $\hat{\theta} \leftarrow \hat{\theta} + \alpha_\theta\hat{\delta}\mathbf{z}^{\hat{\theta}}$
    $s \leftarrow s'$

Algorithm 2 Critic Action Type Values (CATV) algorithm for multilevel action selection with average reward

Input: a differentiable policy parametrization $\pi(a|s)$
Input: a feature vector $x(s)$
Algorithm Parameters: actor eligibility trace decay λ_θ, critic eligibility trace decay $\lambda_{\mathbf{w}}$, actor weight step size α_θ, critic weight step size $\alpha_{\mathbf{w}}$, average reward step size α_r, number of action types d

for i from 1 to d **do**
    Initialize policy parameters $\theta_i \in \mathbb{R}^n$ and critic weights $\mathbf{w}_i \in \mathbb{R}^m$
    $\mathbf{z}_i^\theta \leftarrow \mathbf{0}^n$ (actor eligibility trace)
    $\mathbf{z}_i^{\mathbf{w}} \leftarrow \mathbf{0}^m$ (critic eligibility trace)
Initialize $\bar{r} \in \mathbb{R}$
Choose initial state s
while True **do**
    Choose action type D from $\{1, 2, ...d\}$ epsilon greedily according to $\underset{i}{\text{argmax}}(\mathbf{w}_i \mathbf{x}(s))$
    Choose action parameters a from $\pi_D(a|s)$
    Take action a, observe state s' and reward r
    $D' \leftarrow \underset{i}{\text{argmax}}(\mathbf{w}_i \mathbf{x}(s'))$
    $\delta \leftarrow r - \bar{r} + \mathbf{w}_{D'} \mathbf{x}(s') - \mathbf{w}_D \mathbf{x}(s)$
    $\bar{r} \leftarrow \bar{r} + \alpha_r \delta$
    for i from 1 to d **do**
        $\mathbf{z}_i^{\mathbf{w}} \leftarrow \lambda_{\mathbf{w}} \mathbf{z}_i^{\mathbf{w}}$
        $\mathbf{z}_i^\theta \leftarrow \lambda_\theta \mathbf{z}_i^\theta$
    $\mathbf{z}_D^{\mathbf{w}} \leftarrow \mathbf{z}_D^{\mathbf{w}} + \mathbf{x}(s)$
    $\mathbf{z}_D^\theta \leftarrow \mathbf{z}_D^\theta + \frac{\nabla_\theta \pi_D(a|s,\theta)}{\pi_D(a|s,\theta)}$
    for i from 1 to d **do**
        $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_{\mathbf{w}} \delta \mathbf{z}_i^{\mathbf{w}}$
        $\theta_i \leftarrow \theta_i + \alpha_\theta \delta \mathbf{z}_i^\theta$
    $s \leftarrow s'$

Chapter 4

Experiments and Analysis of Results

This chapter details the experiments I ran to test multilevel action selection. My hypothesis is that using multilevel actions can improve the performance of an algorithm over the single level action variant. For these experiments, I measure performance by the total return after a set number of time steps, averaged over 30 different runs, each with a different random seed. I also present learning curves comparing the average reward of both algorithms for all time steps of the trial. I chose these metrics to evaluate performance as the agent's goal is to maximize a reward signal, and these are two ways of representing that maximization (average over time, and total). These runs are performed on both actor-critic with and without multi level action selection, with two different methods of choosing the action type with multi level action selection. I tested various setting for different parameters: the step sizes for the actor, the critic, and the average reward, and the trace decay rates for the actor and critic for the purposes of analyzing the algorithm's sensitivity to different parameters. In the case of the multi level action selection algorithm, I also tested epsilon when choosing based on the critic values, and the high level step sizes when choosing with a second actor and critic for the action types. Statistical analysis is done to determine how sensitive the algorithm is to different parameters, and how the different algorithms compare.

4.1 Problems to Test Algorithms

When choosing a problem to test on, since I was interested in examining how multilevel action selection can be used to split up a continuous action space, the first quality I looked for was that action space was continuous. Additionally, for the sake of conceptual simplicity, I preferred problems with action spaces that could be broken up into two or more continua that have an obvious conceptual difference between them, although this is not a requirement of the algorithm. For example, an action choosing a movement speed ranging from a negative value (going backwards) to a positive value (going forwards) has a natural split between forwards and backwards movement.

I also favoured simple problems with relatively simple state and action spaces, as I wanted to show this method can work with simple algorithms. Another advantage of simple problems is that when analyzing, it is easier to understand the cause of any specific results. I ran the experiments on multiple problems to get a better idea of how it performs in general, instead of results tailored to a specific problem. The two problems I ended up choosing were mountain car and cart pole, shown in figure 4.1.

While the simplest version of both problems have a finite action space of forward and backward thrust, they have a been adapted to use actions from a continuum between the maximum thrust in the forward and backward directions, with the backward maximum thrust being the negative of the forward maximum thrust. These problems both have an action space involving movement in two directions, forward and backward. This gave a natural split in the action space at zero, resulting in one 'dimension' for positive acceleration and one for negative acceleration[1].

Splitting the action space of cart pole and mountain car resulted in simpler problems than concocting one that had two truly different dimensions (such as rotation and movement). This also had the added benefit of involving another

[1]While this leaves out zero, this is not a problem as when choosing actions from the original continuum, the agent has a vanishingly small chance of ever selecting exactly zero. Additionally, as the agent can still choose actions arbitrarily close to zero, these will have effectively no difference in outcome in the simulation from choosing exactly zero.

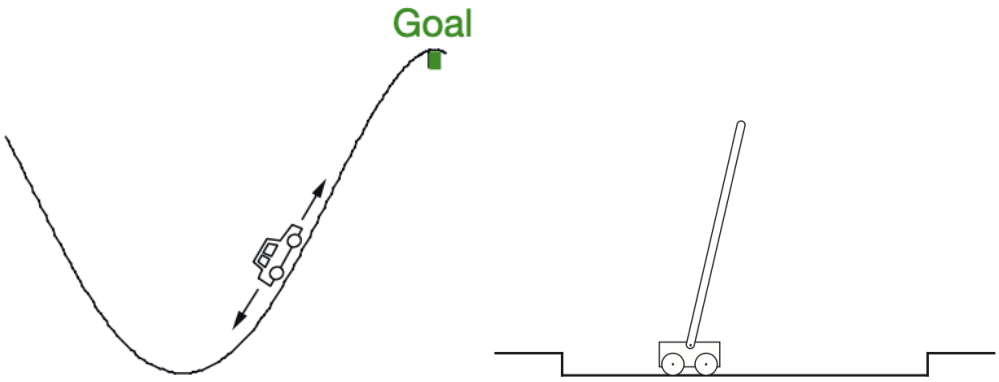


Figure 4.1: The mountain car and cart pole problems [21]

potential application of multilevel actions: breaking a large space into smaller spaces may speed up learning since some areas of the action space may be all lead to a poor return, so another area, where the actions leading to a high return may all be concentrated, can be favoured.

4.2 Cart Pole

The first problem I tested on was cart pole. In this formulation of cart pole, there is a cart on a one dimensional track with a pole attached to it initially pointing upwards. The pole is attached by a hinge so it can swing down in the same direction as the track. The objective is to prevent the pole from falling for as long as possible. A failure is defined as whenever the pole swings past a certain threshold from vertical, or the cart hits the end of the track. The agent's action is the thrust applied to the cart, and ranges from the maximum backward thrust to the maximum forward thrust. The reward is 0 for each timestep the pole stays up, with -1 when it falls, or the cart runs out of track (see figure 4.1). The state information available to the agent is the position and velocity of the cart, as well as the angle and angular velocity of the pole. The cart starts close to the middle of the track with the pole vertical. There is a small, uniformly distributed, random value added to the position and velocity of the cart and the pole. The maximum size of this random value is about 12% of the maximum value for the pole, and about 2% of the maximum value for the cart. For these experiments, I treat cart pole as a continuing problem;

when a failure occurs, instead of starting a new episode, the cart and pole are reset to their starting positions (with the random offsets recalculated) and the agent continues. This resetting of the position happens immediately, when the agent takes an action leading to a failure, the next state it transitions to is with the cart and pole in the new starting positions. Learning continues uninterrupted over this transition.

4.2.1 Experimental Setup

Both my multilevel action selection algorithm and an actor-critic algorithm without splitting the action space were tested and compared. This actor-critic algorithm without multilevel action selection is the effectively identical to my algorithms if the number of action types was reduced to one. This makes it the natural algorithm to use for comparison, as differences in performance will be solely due to the use of multilevel action selection, and not the method of selecting the action parameters. The multilevel action selection algorithm was set up to have one action type associated with forward movement (positive actions) and the other with backwards movement (negative actions). Each algorithm was run 30 times with different random seeds for each (every run has two random seeds, one for the agent and one for the environment), with each run lasting for a total of 10,000 time steps. Clouse and Utgoff [3] used 10,000 time steps as the boundary for when an agent has solved the cart pole problem, so it was initially included in my simulation as a maximum episode length. When I switched from episodic to continuous, it became the experiment length instead. It ended up being both long enough to show differentiation between the algorithms, but short enough to be simulated quickly, so I stuck with that value. The metric for comparison was the total return over the whole run, averaged between the different seeds.

The state space was tile coded, with 16 tilings of 10 tiles each. These values were chosen as a middle ground between accuracy of representing the state space (higher numbers) and quickness of both simulation and time to learn (lower numbers). The tilings were done for each state space dimension individually, as well as each pair of two dimensions. A single tiling over all

four dimensions was not used due to the impact on computation time, and preliminary tests do not show any notable gains in performance from using the four way tiling. Additionally, a single always on tile was included in the tile coding.

The traces were set up to be set to zero automatically after a set number of time steps. This resulted in a limit for how many past actions were included in the trace, which improved computation time over leaving all values in the trace until they decay below a certain value. It should be noted that removing features after some amount of time is only different from removing them at some threshold value in cases where the same features get visited again before being removed. In this case, the weight from the first visit will be removed, but not the second. The size of the trace (and thus the number of steps that the record of a past action sticks around) was dependant on the value of the trace decay rate λ. [20]

The initial values of the policy parameters were set to have a mean in the middle of the action space and a standard deviation one eighth the width of the action space. The critic weights were initialized to zeros.

For this problem, I mainly present results from the double actor critic algorithm for multilevel action selection. Experiments were run for both it and critic action type values, but double actor critic clearly performed better in all cases, so I only include one trial of critic action type values for the sake of comparison.

In addition to comparing the performance of a multilevel action algorithm to a regular actor-critic algorithm, I also ran some tests comparing the relative performance for values of certain parameters, to analyze the algorithm's sensitivity to these values. This included the decay rate of the eligibility traces, the step sizes, and whether or not to truncate the ends of the normal distribution used for action selection.[2]

[2]If the distribution is truncated, any weight lying outside the bounds of the action space gets redistributed over the domain of the action space, maintaining the relative probabilities. If it is untruncated, any selection in the tails of the distribution outside the bounds of the action space gets clipped to the bounds of the action space, effectively putting all the weight of the tail right at the bounds.

4.2.2 Parameter Sensitivity Analysis

The results shown in the figures throughout this section are plots of the total return the agent receives after 10,000 time steps, for various values of parameters. The results are plotted with error bars indicating the standard error. These values were compared using a t-test to determine if they are statistically significantly different. Specifically, either Student's t-test (when the variances of the two populations was expected to be equal) or Welch's t-test (when they were expected to be different) were used. To determine if the variances are equal, the Bartlett or Levene tests were used (using Bartlett when the data followed a normal distribution, Levene when it did not). This test was computed pairwise for each pair of values in a table, and additionally between the max value in each table. In these tests, a p-value of less than 0.05 was the condition to reject the null hypothesis that they have an identical mean. Tables for the data used in the plots throughout this section can be found in appendix A.

The first experiment was the comparison of the trace decay rates (λ). This was run with with $\alpha_\theta = 0.001, \alpha_{\mathbf{w}} = 0.001$ for the action parameter step sizes, $\hat{\alpha}_\theta = 0.01, \hat{\alpha}_{\mathbf{w}} = 0.001$ for the action type step sizes, and $\alpha_r = 0.05$ for the average reward step size. In figure 4.2, out of 36 pairwise comparisons, only 9 were not significantly different. These all correspond to the cases where λ_θ is the same and only $\lambda_{\mathbf{w}}$ differs. Therefore, I conclude that the trace decay rate of the critic weight trace has a statistically significant effect on the algorithm's performance, while the trace decay rate of the policy parameter trace does not.

Figure 4.3 shows the results for various values of the step sizes for action type selection. This was run with with $\alpha_\theta = 0.001, \alpha_{\mathbf{w}} = 0.001$ for the action parameter step sizes, $\lambda_\theta = 0.75, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates, and $\alpha_r = 0.05$ for the average reward step size. $\hat{\alpha}_\theta$ is the step size for the actor, and $\hat{\alpha}_{\mathbf{w}}$ is the step size for the critic. Of the 120 pairwise comparisons, 52 of them have a p-value of less than 0.05. Of these, 48 are comparisons between $\hat{\alpha}_{\mathbf{w}} = 0.1$ and the other values of $\hat{\alpha}_{\mathbf{w}}$. A further 3 are between the best

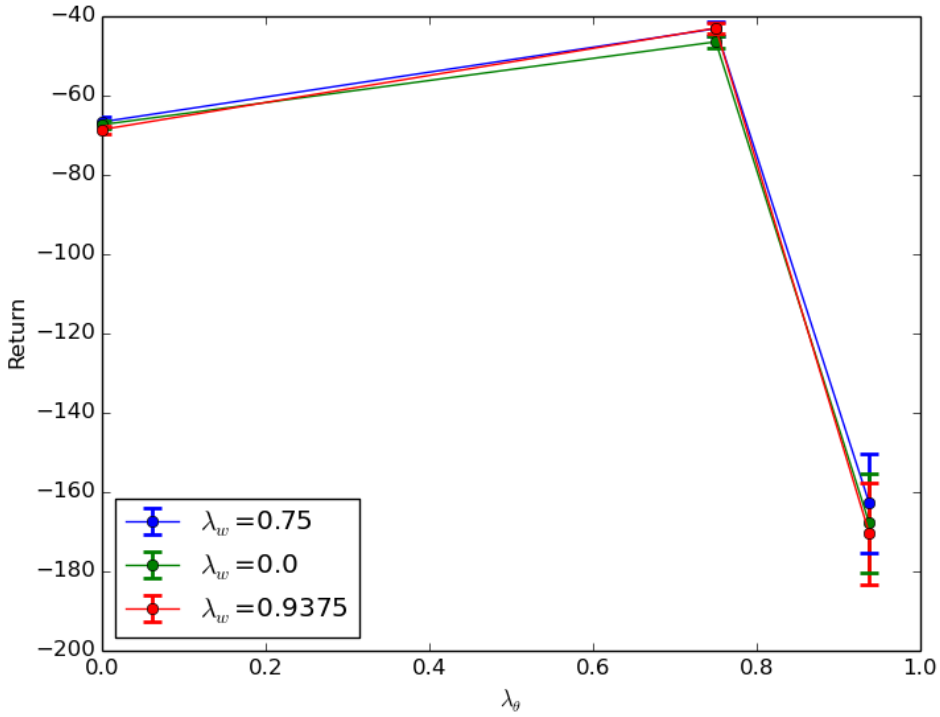


Figure 4.2: Total return after 10,000 time steps for various trace decay rates in cart pole

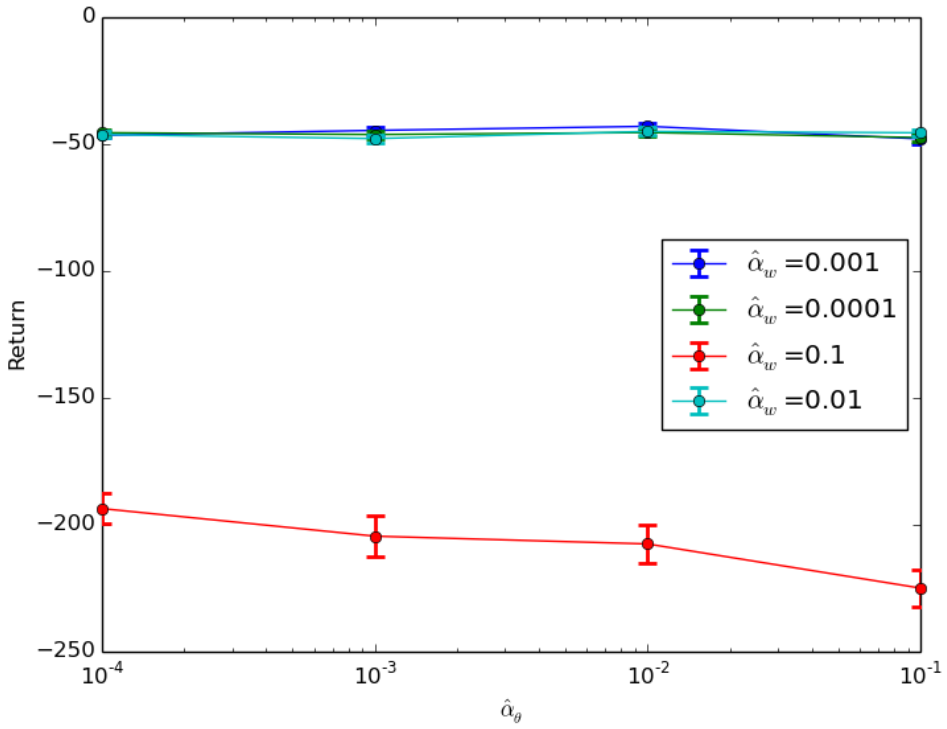


Figure 4.3: Total return after 10,000 time steps for various step sizes for action types in cart pole

performing values of $\hat{\alpha}_\theta = 0.01, \hat{\alpha}_\mathbf{w} = 0.001$ and the three worst performing values excluding those with $\hat{\alpha}_\mathbf{w} = 0.1$. The final is between the best and worst performing values of $\hat{\alpha}_\mathbf{w} = 0.1$. From this I conclude that there is some significance of the step sizes at the high level, but overall, the algorithm is not very sensitive to it, aside from $\hat{\alpha}_\mathbf{w} = 0.1$ having significantly poorer performance.

Figure 4.4 shows the results for various step sizes for the average reward. This was run with with $\alpha_\theta = 0.001, \alpha_\mathbf{w} = 0.001$ for the action parameter step sizes, $\hat{\alpha}_\theta = 0.01, \hat{\alpha}_\mathbf{w} = 0.001$ for the action type step sizes, and $\lambda_\theta = 0.75, \lambda_\mathbf{w} = 0.9375$ for the trace decay rates. Of the 36 pairwise comparisons, 34 of them are statistically significantly different. The only two not significantly different are between $\alpha_r = 0.0005$ and $\alpha_r = 0.001$, which are adjacent, and $\alpha_r = 0.5$ and $\alpha_r = 0.005$ which are on different sides of the peak at $\alpha_r = 0.05$. I conclude that the algorithm's performance is dependant on the step size for the average

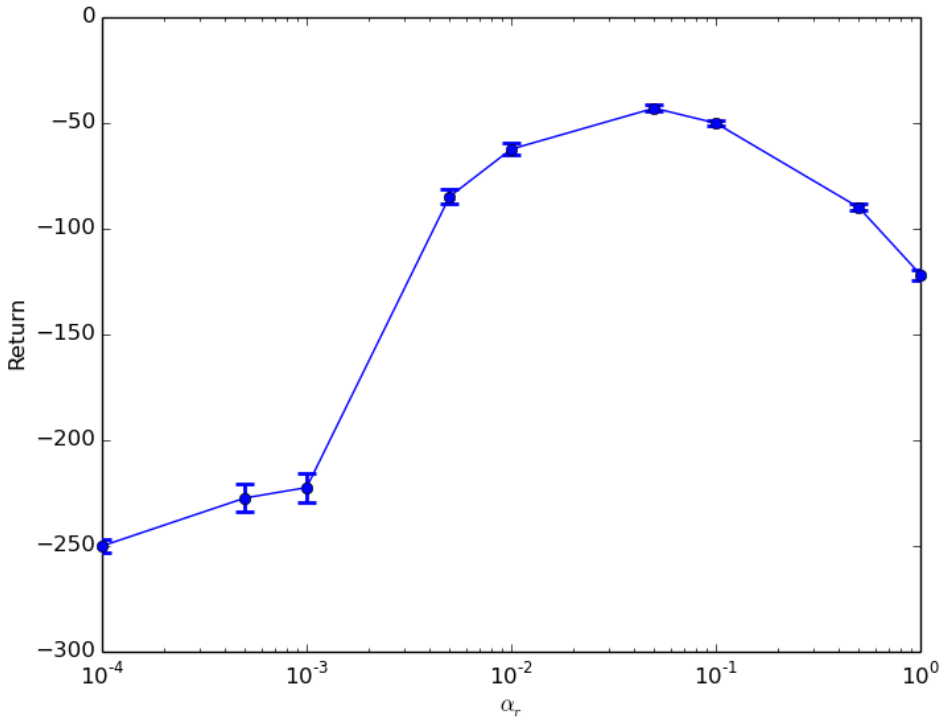


Figure 4.4: Total return after 10,000 time steps for various step sizes for the average reward in cart pole

reward, with fairly high sensitivity.

Figure 4.5 shows the results for various values of epsilon when using the low level critic values instead of a second high level actor and critic. This was run with with $\alpha_\theta = 0.001, \alpha_{\mathbf{w}} = 0.001$ for the action parameter step sizes, $\lambda_\theta = 0.75, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates, and $\alpha_r = 0.05$ for the average reward step size. In this test, 13 out of 28 pairwise comparisons had a p-value less than 0.05. Of these, $\epsilon = 0.1$ and $\epsilon = 0.05$ are statistically significantly different from all other values of ϵ. The other values of ϵ do not have any significant difference between them. I conclude that the algorithm is not very sensitive to the value of ϵ, outside of the area immediately around the best performing value.

Figure 4.6 shows the results for various values of the step sizes for action parameter selection. This was run with with $\hat{\alpha}_\theta = 0.01, \hat{\alpha}_{\mathbf{w}} = 0.001$ for the action type step sizes, $\lambda_\theta = 0.75, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates, and

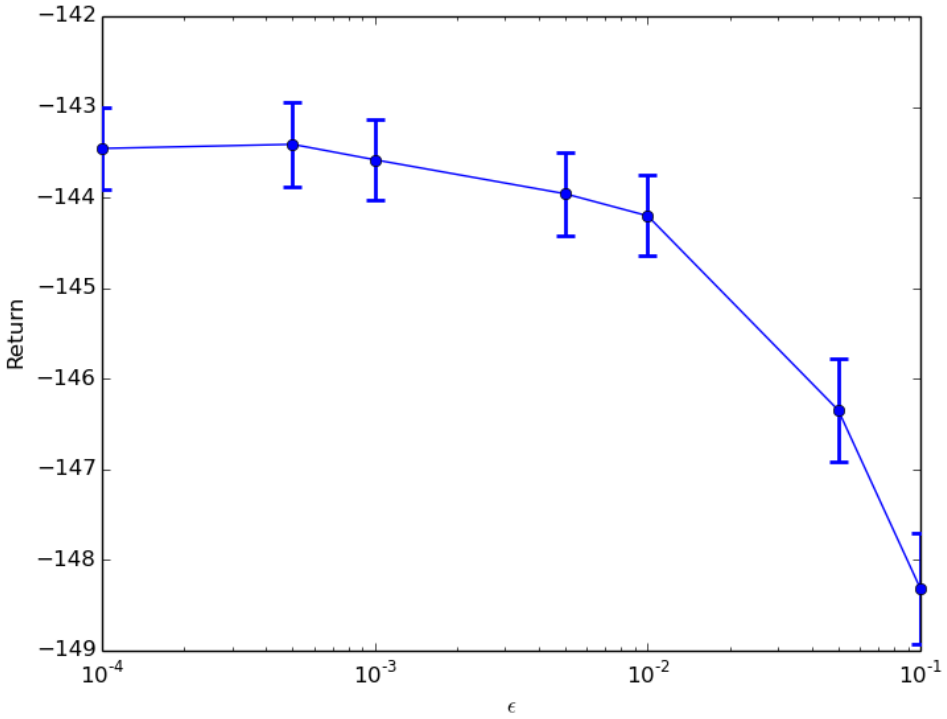


Figure 4.5: Total return after 10,000 time steps for various values of epsilon in cart pole

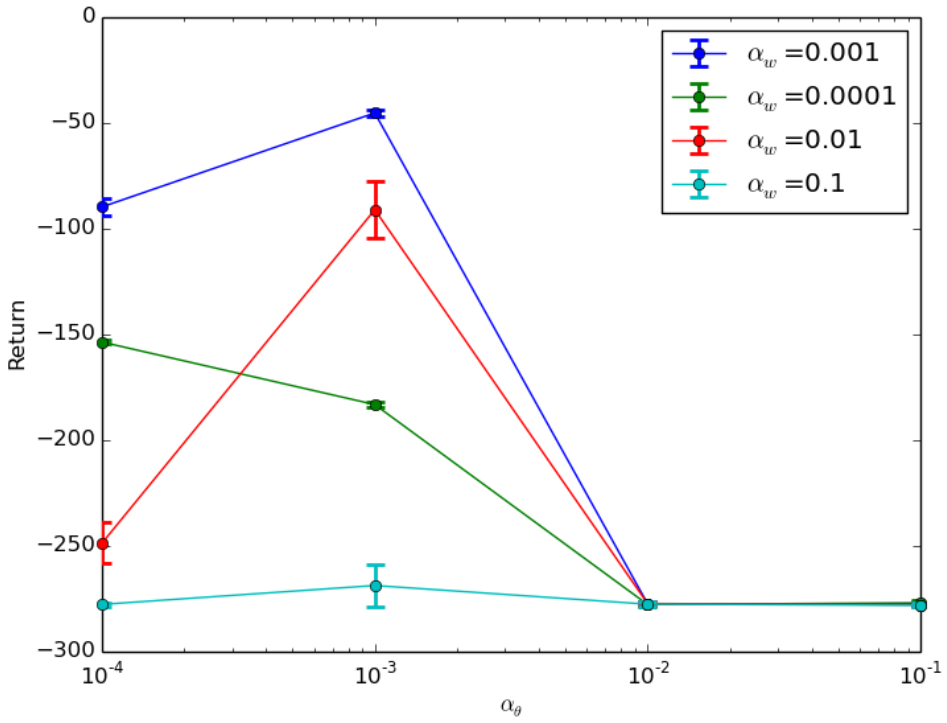


Figure 4.6: Total return after 10,000 time steps for various step sizes for low level actions in cart pole

$\alpha_r = 0.05$ for the average reward step size. For the comparisons where $\alpha_\theta = 0.1$ or $\alpha_\theta = 0.01$ in both trials, there no significantly different comparisons. For the comparisons where $\alpha_\theta = 0.001$ or $\alpha_\theta = 0.0001$ in both trials, the only significantly different comparisons are when $\alpha_{\mathbf{w}} = 0.1$ and between $\alpha_\theta = 0.001, \alpha_{\mathbf{w}} = 0.01$ and $\alpha_\theta = 0.0001, \alpha_{\mathbf{w}} = 0.001$. Comparing the values in between these two groups, all values are significantly different from each other except for when $\alpha_{\mathbf{w}} = 0.1$. I therefore conclude that the performance of the algorithm is significantly affected by the actor step size, and affected by the critic step size only for sufficiently small values of the actor step size.

4.2.3 Algorithm Comparison Analysis

Figures 4.7 and 4.8 show the results of using a the same algorithm with a non truncated distribution and using regular actor-critic algorithm without multi level actions respectively. Figure 4.9 shows the comparison between the three

algorithms. This was run with with $\alpha_\theta = 0.001, \alpha_{\mathbf{w}} = 0.001$ for the action parameter step sizes, $\hat{\alpha}_\theta = 0.01, \hat{\alpha}_{\mathbf{w}} = 0.001$ for the action type step sizes, $\alpha_r = 0.05$ for the average reward step size, and $\lambda_\theta = 0.75, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates. Comparing the results of these algorithms for each of the settings for the step sizes shows a statistically significant difference between the multi level and single level action algorithms for all values of the steps sizes, except for $\alpha_\theta = 0.0001, \alpha_{\mathbf{w}} = 0.01$. The difference between using truncated and non truncated distributions was never significant for any of the values. Additionally, comparing the best performing step sizes for each algorithm also results in a statistically significant difference between multi level and single level actions, but not between truncated and non truncated. From this, I conclude that the use of truncated or non truncated distributions does not have any significant affect on the performance of the algorithm, but using multilevel action selection can improve performance over not using it.

Figure 4.10 shows the comparison between the learning curves over time between multilevel (non truncated) and single level action selection. Note that in this plot, average reward refers to the mean of the reward received over all time steps so far, not the average reward tracked by $\bar{r}$ in the agent. This shows that both algorithms follow a similar curve, and are flattening out at roughly the same rate. In order to keep average reward steady, an agent must keep the pole up for a number of time steps equal to the negative of the inverse of the current average reward (see appendix C for how this value was calculated). At the beginning, the average reward is zero, so the agent would need to keep the pole up infinitely in order to prevent the average reward from decreasing. This is why near the beginning the curve shows a sharp decrease in average reward. As time goes on, the average reward decreases, and the agent can keep the pole up longer, increasing the chance the agent will meet this threshold. Once the agent meets and exceeds the threshold, the average reward will start to increase, until it is high enough that the agent can't keep the pole up for long enough anymore. The multilevel algorithm meets the threshold first and starts gaining average reward before the single level algorithm does.

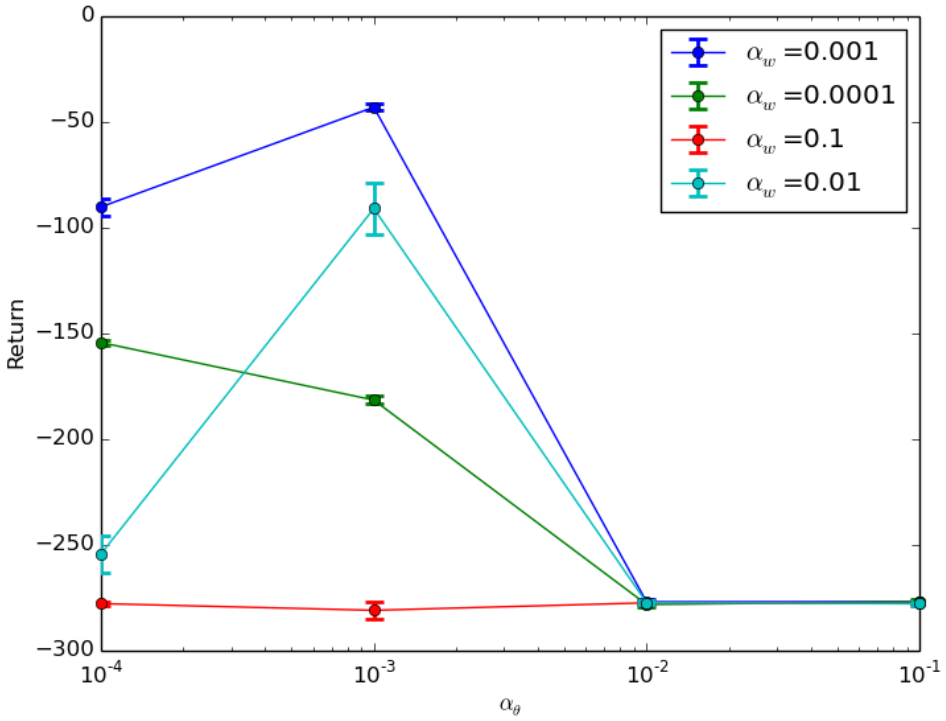


Figure 4.7: Total return after 10,000 time steps for various step sizes for low level actions with non truncated distributions in cart pole

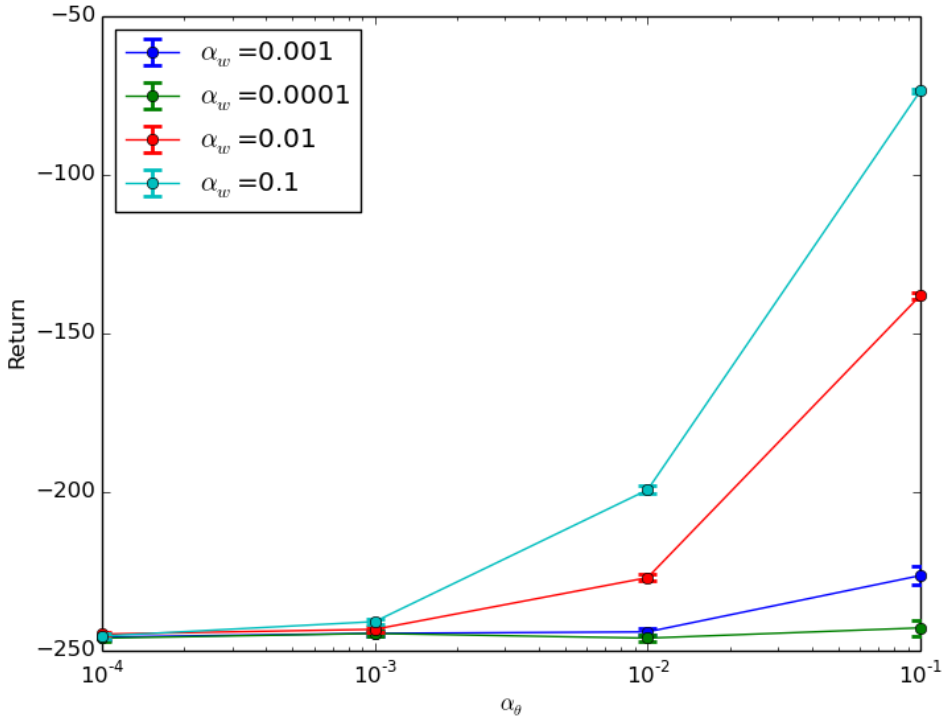


Figure 4.8: Total return after 10,000 time steps for various step sizes without using multilevel actions in cart pole

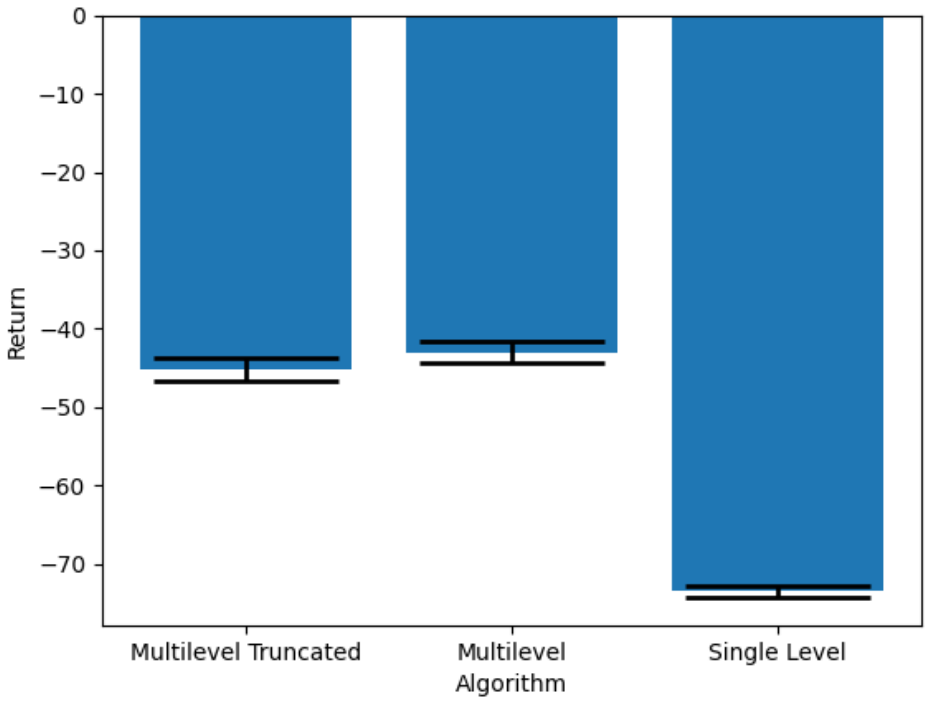


Figure 4.9: Comparative performance on the cart pole problem for different algorithms

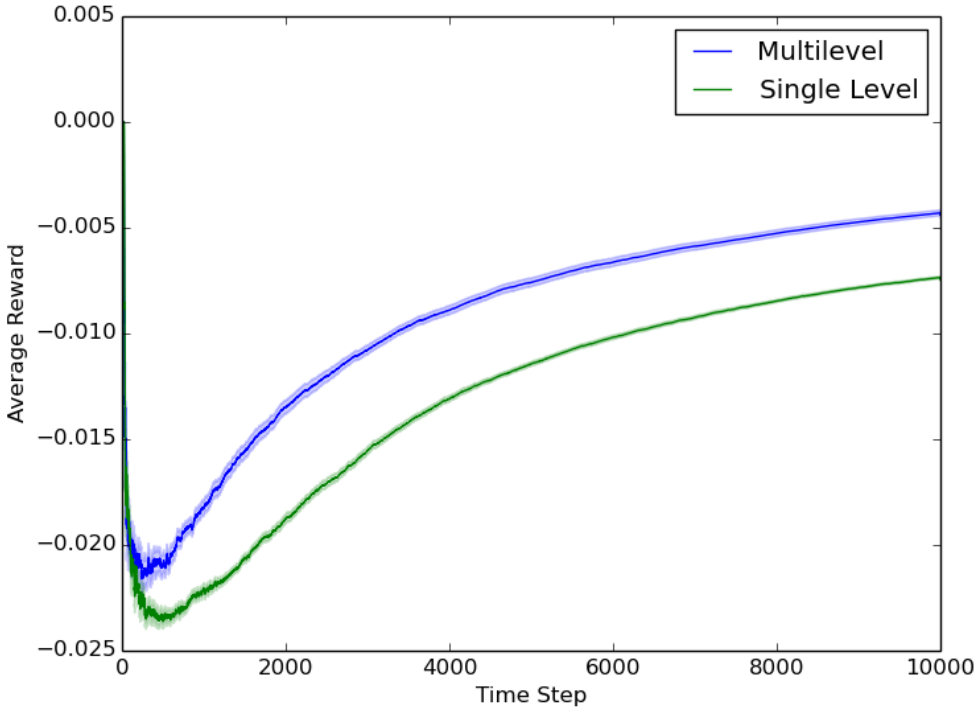


Figure 4.10: Comparative learning curves on the cart pole problem for multi-level and single level action selection.

4.3 Mountain Car

The algorithms were also tested on the mountain car problem. This formulation of mountain car involves a car trying to reach the top of a mountain, but its engine is not powerful enough to go directly. It is in a valley with another mountain behind it. It can however learn to build up speed by driving up one side of the valley, then reversing direction and driving up the other, until it reaches the top (see figure 4.1). Some versions of the problem have discrete actions of either forward or backward, but the version used here is continuous. The agent takes an action between the maximum backward thrust and the maximum forward thrust, and the reward is -1 for each time step, with a reward of 0 instead upon reaching the top. The state information provided to the agent is the position of the car along the ground, as well as its current velocity. Mountain car can be be either an episodic or a continuing problem, but I have treated it as continuing for these experiments. This was because in order to run it as episodic, there needs to be a max time for the episodes (so a configuration which can never learn to get up the hill won't run forever). Setting this max time could be problematic, as there are some configurations which do eventually learn, but take a long time to reach the top the first time. These configurations could never learn in episodic, but in a continuing setup where only the maximum time is set allows them to eventually learn and produce usable data.

4.3.1 Experimental Setup

Both my multi level action algorithm and a regular actor-critic algorithm without splitting the action space were tested and compared. The multi level action algorithm was set up to have one action type associated with forward movement (positive actions) and the other with backwards movement (negative actions). Each algorithm was run for 1,000,000 time steps 30 times each with different starting seeds. I ran each trial of mountain car longer than cart pole because for some parameter settings, the agent is very slow to find a solution, so the longer trials result in fewer parameter settings never finding a solution

in the allotted time, giving more meaningful comparisons. I found 1,000,000 time steps to be the best middle ground for giving most configurations a chance to learn, but not pushing the total experiment time too long. The metric for comparison between algorithms was the total return over all episodes averaged between the seeds.

The state space was tile coded, with 16 tilings of 10 tiles each. These values were chosen as a middle ground between accuracy of representing the state space (higher numbers) and quickness of both simulation and time to learn (lower numbers). The tilings were done for both state space dimensions individually, as well as for the two dimensions together. Adding the individual tilings did not have a large impact on computation time, as they are only one dimensional so small in size, so there was no drawback to including the added precision. Additionally, a single always on tile was included in the tile coding.

The traces were set up to zero automatically after a set number of time steps. This resulted in a limit for how many past actions were included in the trace, which improved computation time over leaving all values in the trace until they decay below a certain value. It should be noted that removing features after some amount of time is only different from removing them at some threshold value in cases where the same features get visited again before being removed. In this case, the weight from the first visit will be removed, but not the second. The size of the trace (and thus the number of steps that the record of a past action sticks around) was dependant on the value of the trace decay rate λ.

The initial values of the policy parameters were set to have a mean in the middle of the action space and a standard deviation one eighth the width of the action space. The critic weights were initialized to zeros.

For this problem, I focused on using critic action type values as my method of choosing action types. I did this due to performance issue, double actor critic was too slow to produce the results needed to make any sort of statistically significant claim in a timely manner.

Due to the nature of the problem and reward structure, the return for a run will be a very large negative number. In order to make results easier to

compare, instead of reporting the return, I instead report the number of times the agent reaches the top within the time limit. This is equal to the number of time steps plus the total return.

4.3.2 Parameter Sensitivity Analysis

The results shown in the figures throughout this section are plots of the total return the agent receives after 1,000,000 time steps, for various values of parameters. The results are plotted with error bars indicating the standard error.

The first experiment was a comparison of the trace decay rates. This was run with with $\alpha_\theta = 0.01, \alpha_{\mathbf{w}} = 0.01$ for the action parameter step sizes, $\epsilon = 0.1$ for the action type selection, and $\alpha_r = 0.0005$ for the average reward step size. The results are shown in figure 4.11. Of 36 pairwise comparisons, 5 were not significantly different from each other. Of these, one was between the two trials with zero return. The remaining four were all cases where $\lambda_{\mathbf{w}}$ was the same and only λ_θ differed. From this, I conclude that the trace decay rate has an effect on the algorithm's performance, with the critic trace decay rate having a stronger effect than the policy parameter trace decay rate.

The next experiment analyzed was the values of epsilon while choosing the action type with results in figure 4.12. This was run with with $\alpha_\theta = 0.01, \alpha_{\mathbf{w}} = 0.01$ for the action parameter step sizes, $\alpha_r = 0.0005$ for the average reward step size, and $\lambda_\theta = 0, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates. Of 28 pairwise comparisons, 6 of them do not have a statistically significant difference. These are from two clusters of three trials each which are not significantly different from each other. The first cluster involved $\epsilon = 0$, $\epsilon = 0.0001$, and $\epsilon = 0.0005$, and the second involved $\epsilon = 0.005$, $\epsilon = 0.01$, and $\epsilon = 0.05$. Therefore, the value of ϵ is statistically significant between low and high values, but when values are close enough, there is not a significant difference.

Figure 4.13 shows the results for various values of the average reward step size. This was run with with $\alpha_\theta = 0.01, \alpha_{\mathbf{w}} = 0.01$ for the action parameter step sizes, $\epsilon = 0.1$ for the action type selection, and $\lambda_\theta = 0, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates. Of the 36 pairwise comparisons, only 4 are not significantly

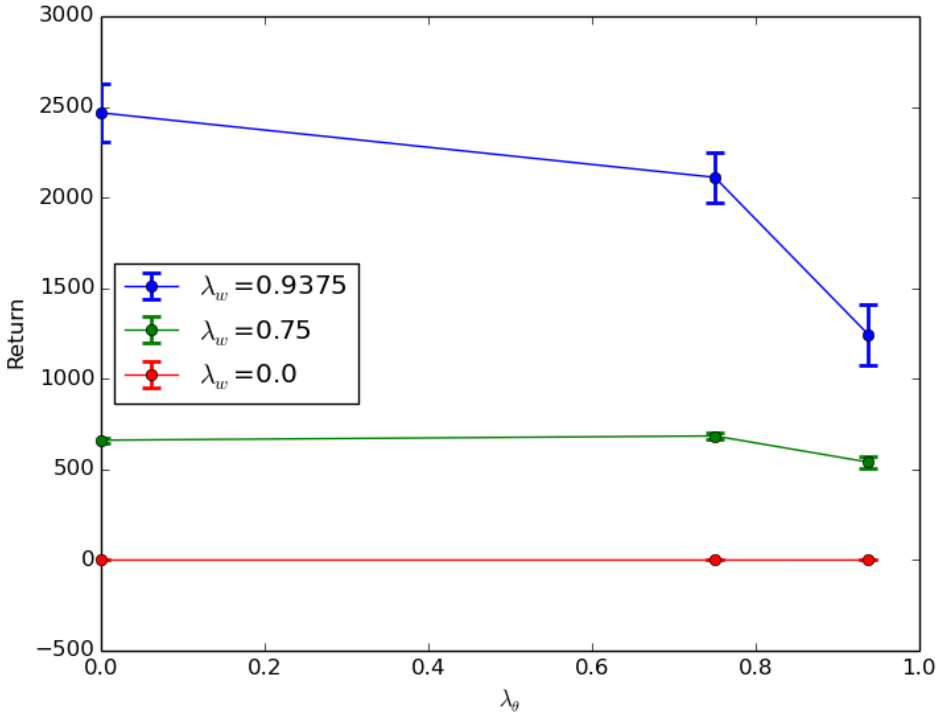


Figure 4.11: Total successes after 1,000,000 time steps for various trace decay rates in mountain car

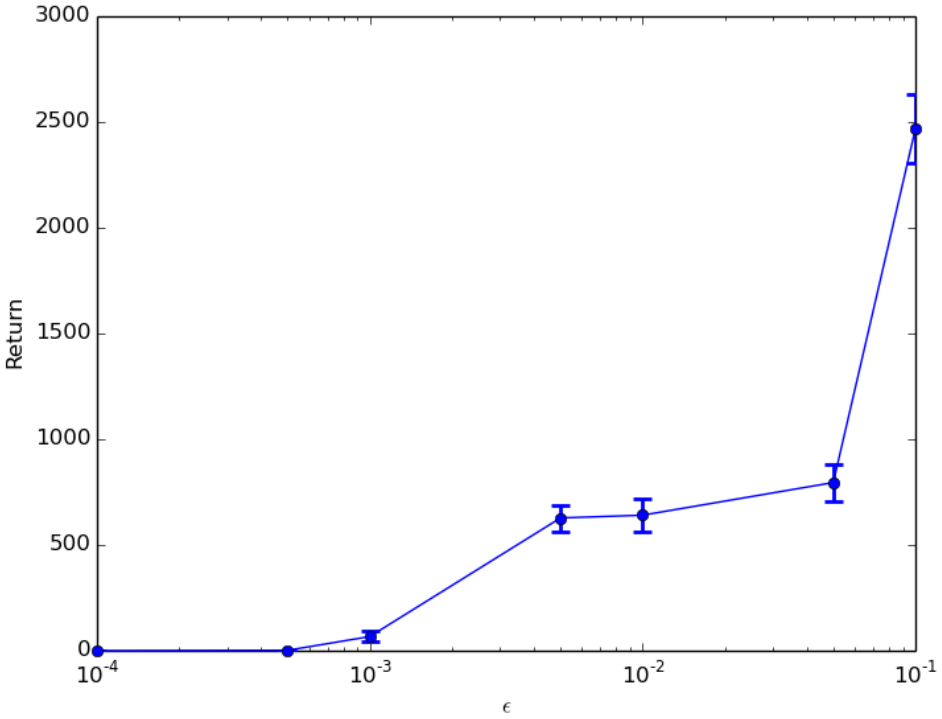


Figure 4.12: Total return after 1,000,000 time steps for various values of epsilon in mountain car

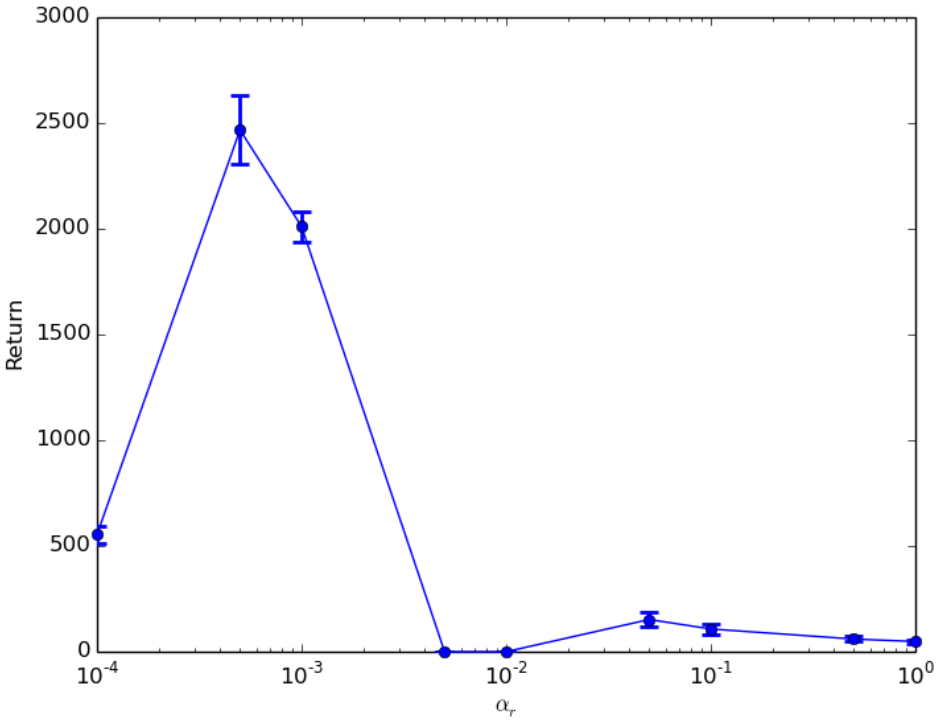


Figure 4.13: Total return after 1,000,000 time steps for various step sizes for the average reward in mountain car

different. Of these, one is comparing between the two values that lead to zero return. The remaining three are between $\alpha_r = 1$ and $\alpha_r = 0.5$, $\alpha_r = 0.5$ and $\alpha_r = 0.1$, and $\alpha_r = 0.1$ and $\alpha_r = 0.05$ As all of these cases are between middle ground values, there is a significant difference in performance based on the average reward step size, with a smaller step sizes tending to be better, except at the extreme.

Figure 4.14 shows the results for various values of the step sizes for low level action selection. This was run with with $\epsilon = 0.1$ for the action type selection, $\alpha_r = 0.0005$ for the average reward step size, and $\lambda_\theta = 0, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates. Of 120 pairwise comparisons, 28 are between the different values leading to zero return. Other than these, there are only 18 comparisons which are not significantly different. Of these, eight are comparing $\alpha_\theta = 0.1, \alpha_{\mathbf{w}} = 0.0001$ to the eight values which resulted in zero return. The other 10 are from a set of five values none of which are significantly different from

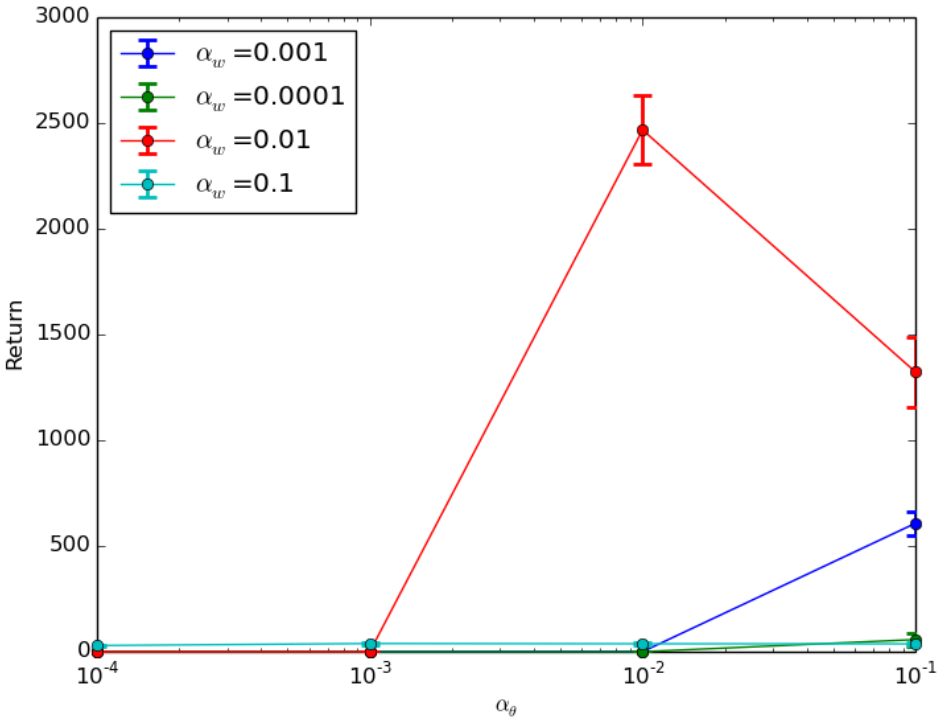


Figure 4.14: Total return after 1,000,000 time steps for various step sizes for low level actions in mountain car

each other, $\alpha_\theta = 0.0001, \alpha_{\mathbf{w}} = 0.1$, $\alpha_\theta = 0.1, \alpha_{\mathbf{w}} = 0.1$, $\alpha_\theta = 0.01, \alpha_{\mathbf{w}} = 0.1$, $\alpha_\theta = 0.001, \alpha_{\mathbf{w}} = 0.1$, and $\alpha_\theta = 0.1, \alpha_{\mathbf{w}} = 0.0001$. From this, I conclude that the step sizes have a significant effect on the performance of the algorithm.

4.3.3 Algorithm Comparison Analysis

Figures 4.15 and 4.16 show the results of using a the same algorithm with a truncated distribution and using regular actor-critic algorithm without multi level actions respectively. The results in figure 4.15 are overlapping, with only $\alpha_{\mathbf{w}} = 0.1$ giving non zero returns, see appendix A for the exact values. Figure 4.17 shows the comparison between the three algorithms. This was run with with $\epsilon = 0.1$ for the action type selection, $\alpha_r = 0.0005$ for the average reward step size, and $\lambda_\theta = 0, \lambda_{\mathbf{w}} = 0.9375$ for the trace decay rates. Comparing the results of these algorithms for each of the settings for the step sizes shows a statistically significant difference between the multilevel, multilevel truncated,

and single level action algorithms for almost all values of the steps sizes. In a few cases, the T test could not be completed due to arithmetic errors arising from identical values. Ignoring these cases, the only cases not significantly different are $\alpha_\theta = 0.1, \alpha_{\mathbf{w}} = 0.0001$ between the truncated and non truncated multilevel algorithms, $\alpha_\theta = 0.1, \alpha_{\mathbf{w}} = 0.01$ between the non truncated multilevel algorithm and the single level algorithm, and $\alpha_\theta = 0.1, \alpha_{\mathbf{w}} = 0.1$ between the single level algorithm and both multilevel algorithms. Additionally, comparing the best performing step sizes for each algorithm also results in a statistically significant difference between all algorithms. From this, I conclude that using multilevel actions cannot match the performance of regular actor-critic. Additionally, since it had so many zero returns, I conclude the use of truncated distributions does not work for the mountain car problem. Figure 4.18 shows the comparison between the learning curves over time between multilevel (non truncated) and single level action selection, with results averaged over 30 trials. This shows two major factors which are holding back the performance of the multilevel algorithm. First is that multilevel starts learning much later on average, with the point where the average reward first moves above -1 representing when the agent first reached the top of the hill. Second is the dip that the occurs around 400,000 time steps in the graph. Such a dip occurs in roughly the same place in all trials. See appendix B for a selection of learning curves from individual trials to see how they compare. Such dips could potentially be from unfortunately timed exploration at the action type level, if the agent is forced to go backward when a forward action would be ideal or vice versa, although I would expect the agent to be able to recover from this quite quickly (on the order of a couple hundred timesteps required to start over and go up the hill again, not the hundred thousand seen in this graph). Further analysis is required to determine the true source of this dip. Although the curves for neither trial have reached an asymptote by the end of the 1,000,000 time steps, they appear to both be flattening out at about the same rate.

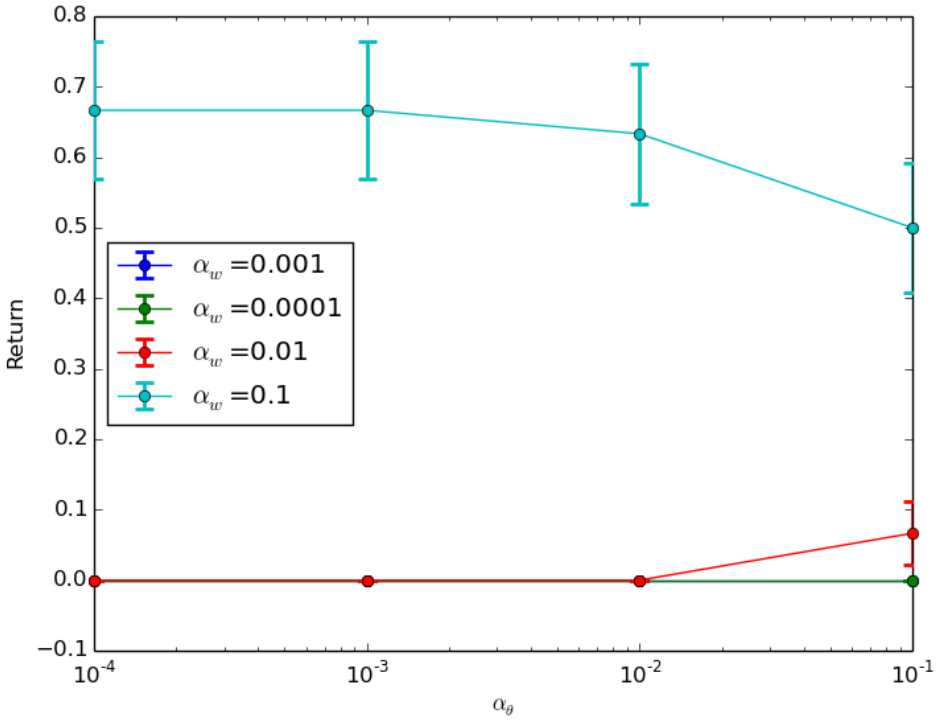


Figure 4.15: Total return after 1,000,000 time steps for various step sizes for low level actions with truncated distributions in mountain car

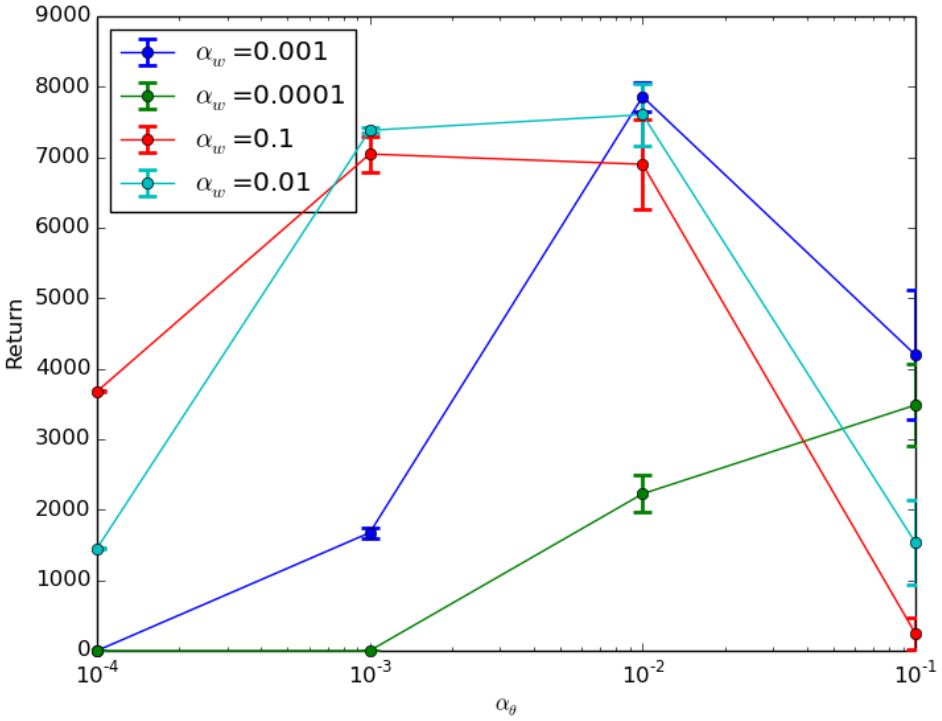


Figure 4.16: Total return after 1,000,000 time steps for various step sizes without using multi level actions in mountain car

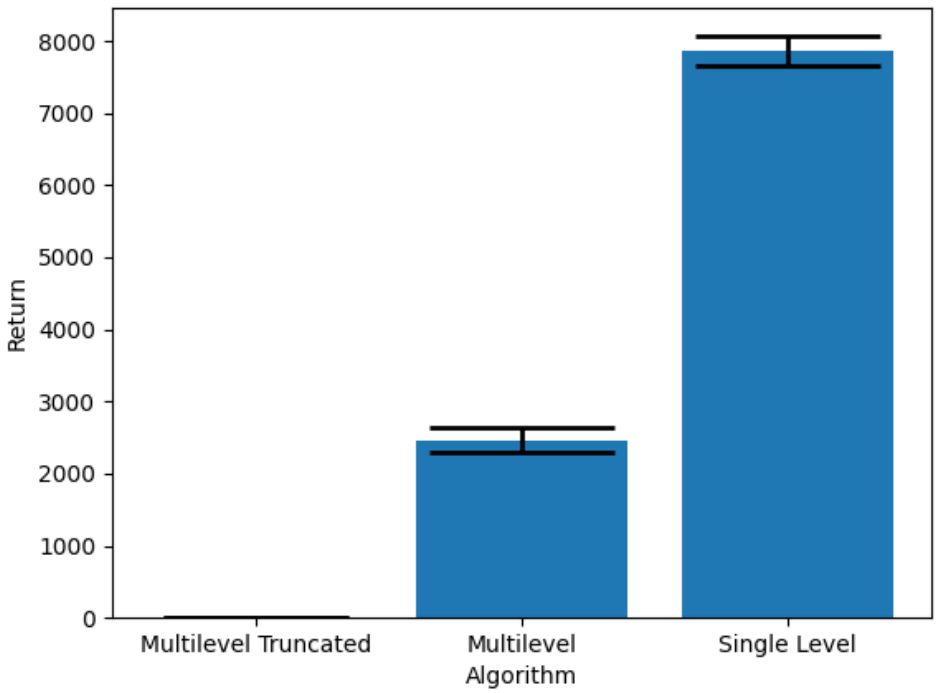


Figure 4.17: Comparative performance on the mountain car problem for different algorithms

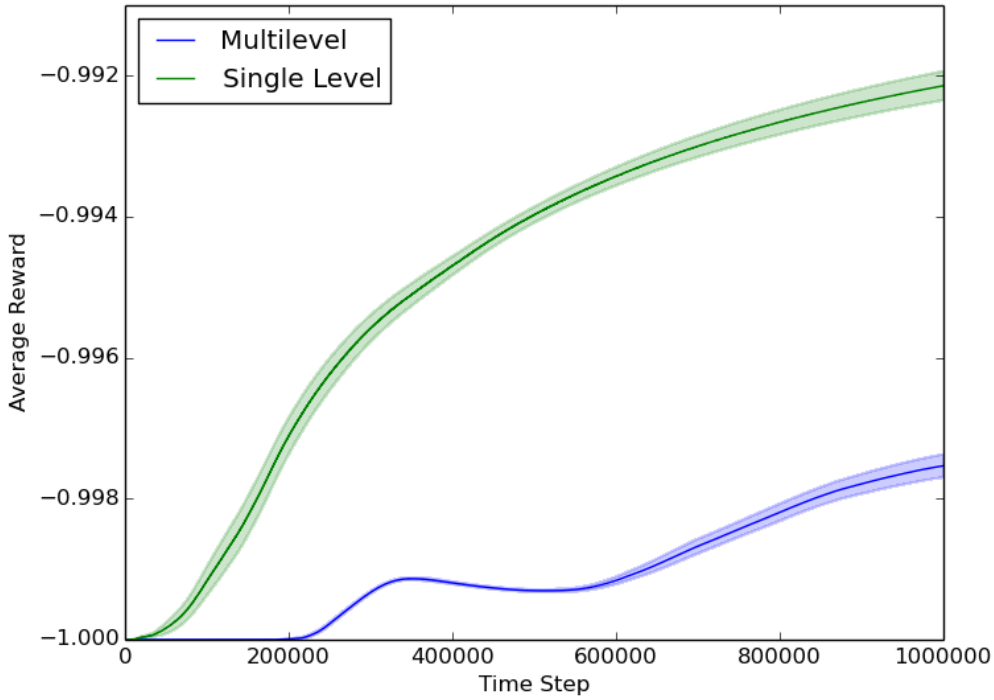


Figure 4.18: Comparative learning curves on the mountain car problem for multilevel and single level action selection.

4.4 Software

The software used to run the experiments used in this thesis can be found at this git repository [16]. I based this code on an existing simulator for a different problem [19]. The cart pole and mountain car problems were used for the experiments, and were modelled based on those from OpenAI Gym, with some alterations such as the reward signal. I used C++ for the purposes of efficiency in the simulator, with some additional scripts in Bash and Python. The statistical analysis was done using the SciPy python library [23]. In order to save on computation, instead of keeping a full record of the traces, they were truncated after a set number of time steps, and set to zero. The method for doing this is from [19].

Chapter 5

Conclusion

In this thesis I explore the usage of multilevel action selection. I looked at problems which had a single dimension action space to see if breaking up the action space with multi level actions could improve performance. Some problems have action spaces which are naturally broken up into multiple action types that lie in different dimensions, such as the example from chapter 1, but I did not explore these problems in my thesis.

My experiments were done on the cart pole and mountain car problems. I chose these problems for their simple action spaces with a conceptual difference between forwards and backwards thrust. They involve an action space ranging from the maximum backward thrust to the maximum forward thrust which I split up into positive and negative actions types for the purpose of multi level actions. In these experiments I was able to see an increase in the total return after a set amount of time by using a multi level actions instead of a regular actor-critic algorithm, depending on the problem the agent is solving.

Specifically, the agent was able to achieve significantly better performance on the cart pole problem using multilevel action selection, but could not match the performance of regular action selection on the mountain car problem. A notable difference between these problems is that mountain car is about reaching a goal, and cart pole is about preventing failure. In mountain car, once the agent reaches the goal, it starts again from the bottom, and tries to reach the goal again. In cart pole, once the agent can successfully balance the pole, the pole will never fall over (except for random chance from exploration) and

the agent will never restart the problem. I suspect that an agent which can learn a solution faster will perform better on cart pole, but an agent which can better iterate on and improve its solution will perform better on mountain car, however this would require further experiments to confirm.

With only two problems tested, I do not have enough information to speculate how well multilevel action selection will generalize to other problems. However, the performance on the cart pole problem shows promise, and I believe it is worth further investigation on other similar problems.

While the numerical result for the total return did increase, another factor to consider is computation time. Algorithms with multilevel action selection took considerably longer to run those without to simulate the same amount of time. This means that multilevel action selection may be ill suited for real world situations in which action selection is done live and decisions have to be made in a time sensitive manner.

There are other aspects I intended to investigate but dropped due to time constraints. These include further analysis of performance other than just total return such as the time taken to find a solution, and the quality of the final solution (return after solution has been learned). Further analysis of the learning curves would also be useful, including running them for sufficiently long until the results reach their asymptote, and analyzing the cause of the dip in the multilevel algorithm's performance on mountain car. It would also be illuminating to reproduce these tests on more problems to get a better idea of when multilevel action selection can help versus when it will be detrimental. There were also more ways of choosing action types that may have advantages over the ones used in this thesis, such as using a single critic for both the action type and action parameters, and using a tabular action value method such as SARSA to choose the action types.

The results of this thesis show promise for the application of multi level actions for the purpose of improving performance in at least some specific circumstances. While they may not always be the best way to approach a problem, it is worthwhile to have them in the toolkit alongside other reinforcement learning methods.

References

[1] C. J. Bester, S. D. James, and G. D. Konidaris, "Multi-pass q-networks for deep reinforcement learning with parameterised action spaces," *arXiv preprint arXiv:1905.04388*, 2019.

[2] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *International conference on machine learning*, PMLR, 2017, pp. 834–843.

[3] J. A. Clouse and P. E. Utgoff, "A teaching method for reinforcement learning," in *Machine learning proceedings 1992*, Elsevier, 1992, pp. 92–101.

[4] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *2012 American Control Conference (ACC)*, IEEE, 2012, pp. 2177–2182.

[5] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *arXiv preprint arXiv:1205.4839*, 2012.

[6] O. Delalleau, M. Peter, E. Alonso, and A. Logut, "Discrete and continuous action representation for practical rl in video games," *arXiv preprint arXiv:1912.11077*, 2019.

[7] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," *arXiv preprint arXiv:1903.01344*, 2019.

[8] Y. Fujita and S.-i. Maeda, "Clipped action policy gradient," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1597–1606.

[9] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," in *Australasian joint conference on artificial intelligence*, Springer, 1999, pp. 417–428.

[10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[11] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *arXiv preprint arXiv:1511.04143*, 2015.

[12] M. Khamassi, G. Velentzas, T. Tsitsimis, and C. Tzafestas, "Active exploration and parameterized reinforcement learning applied to a simulated human-robot interaction task," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2017, pp. 28–35.

[13] R. Kumaraswamy, M. Schlegel, A. White, and M. White, "Context-dependent upper-confidence bounds for directed exploration," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[14] Q. Lan, S. Tosatto, H. Farrahi, and A. R. Mahmood, "Model-free policy learning with reward gradients," *arXiv preprint arXiv:2103.05147*, 2021.

[15] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[16] D. Mitchell, *Simulator*, `https://github.com/daniel-mitchell/lunarlander`, 2022.

[17] E. Rachelson, P. Fabiani, and F. Garcia, "Timdppoly: An improved method for solving time-dependent mdps," in *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, IEEE, 2009, pp. 796–799.

[18] J. C. Santamaria, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive behavior*, vol. 6, no. 2, pp. 163–217, 1997.

[19] R. Shariff, *Lunar lander*, `https://github.com/roshanshariff/lunarlander`, 2021.

[20] R. Shariff and T. Dick, *Lunar lander: A continous-action case study for policy-gradient actor-critic algorithms*, 2013.

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

[22] H. Van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement learning*, Springer, 2012, pp. 207–251.

[23] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: `10.1038/s41592-019-0686-2`.

[24] H. Wang and Y. Yu, "Exploring multi-action relationship in reinforcement learning," in *Pacific Rim International Conference on Artificial Intelligence*, Springer, 2016, pp. 574–587.

[25] E. Wei, D. Wicke, and S. Luke, "Hierarchical approaches for reinforcement learning in parameterized action space," *arXiv preprint arXiv:1810.09656*, 2018.

[26] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Reinforcement learning*, pp. 5–32, 1992.

[27] J. Xiong, Q. Wang, Z. Yang, *et al.*, "Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space," *arXiv preprint arXiv:1810.06394*, 2018.

[28] S. Zhang, W. Boehmer, and S. Whiteson, "Generalized off-policy actor-critic," *Advances in neural information processing systems*, vol. 32, 2019.

Appendix A

Tables

This appendix contains the tabular data for the figures shown in chapter 4.

$\lambda_{\mathbf{w}}$ \ λ_θ	0.9375	0.75	0
0.9375	-170.4	$-42.9\overline{6}$	$-68.5\overline{6}$
0.75	-162.8	-43.1	$-66.6\overline{3}$
0	$-167.8\overline{3}$	$-46.4\overline{3}$	$-67.\overline{3}$

Table A.1: Total return after 10,000 time steps for various trace decay rates in cart pole

$\hat{\alpha}_{\mathbf{w}}$ \ $\hat{\alpha}_\theta$	0.0001	0.001	0.01	0.1
0.0001	$-45.4\overline{6}$	-46.2	-45.4	47.4
0.001	$-46.4\overline{6}$	$-44.5\overline{6}$	$-42.9\overline{6}$	-47.9
0.01	-46.2	-47.8	$-44.9\overline{6}$	-45.5
0.1	-193.6	$204.4\overline{6}$	$-207.5\overline{3}$	$-224.9\overline{3}$

Table A.2: Total return after 10,000 time steps for various step sizes for high level actions in cart pole

α_r	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1	0.5	1
Return	$-250.0\overline{3}$	-227.3	$-222.3\overline{6}$	$-84.7\overline{6}$	$-62.3\overline{6}$	$-42.9\overline{6}$	$-50.0\overline{3}$	$-89.9\overline{3}$	$-121.6\overline{3}$

Table A.3: Total return after 10,000 time steps for various step sizes for the average reward in cart pole

ϵ	0	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1
Return	-143.48	-143.46	-143.41	-143.58	-143.96	-144.2	-146.35	-148.31

Table A.4: Total return after 10,000 time steps for various values of epsilon in cart pole

$\alpha_{\mathbf{w}}$ \ α_θ	0.0001	0.001	0.01	0.1
0.0001	$-154.2\overline{6}$	$-181.4\overline{6}$	$-278.1\overline{6}$	-276.8
0.001	$-90.1\overline{3}$	$-42.9\overline{6}$	-276.9	$-276.8\overline{3}$
0.01	$-254.2\overline{3}$	-90.9	-277.3	-277.8
0.1	$-277.7\overline{3}$	-280.9	$-277.3\overline{6}$	$-277.7\overline{3}$

Table A.5: Total return after 10,000 time steps for various step sizes for low level actions in cart pole

$\alpha_{\mathbf{w}}$ \ α_θ	0.0001	0.001	0.01	0.1
0.0001	$-153.5\overline{6}$	$-183.1\overline{3}$	-277.7	-276.9
0.001	$-89.\overline{6}$	$45.2\overline{3}$	$-277.\overline{3}$	-277.3
0.01	$-248.\overline{6}$	-91	$-277.2\overline{3}$	$-277.8\overline{3}$
0.1	$-277.6\overline{3}$	-268.7	$-277.5\overline{3}$	$-278.0\overline{6}$

Table A.6: Total return after 10,000 time steps for various step sizes for low level actions with truncated distributions in cart pole

$\alpha_{\mathbf{w}}$ \ α_θ	0.0001	0.001	0.01	0.1
0.0001	$-246.0\overline{3}$	$-244.5\overline{6}$	$-246.0\overline{3}$	-242.8
0.001	-245.7	$-244.5\overline{3}$	$-244.0\overline{6}$	-226.3
0.01	$-244.7\overline{6}$	$-243.2\overline{6}$	$-226.9\overline{6}$	-138
0.1	$-245.3\overline{6}$	$-240.8\overline{3}$	-199.3	-73.5

Table A.7: Total return after 10,000 time steps for various step sizes without using multi level actions in cart pole

$\lambda_{\mathbf{w}}$ \ λ_θ	0.9375	0.75	0
0.9375	1245.1	2111.5	$2467.4\overline{6}$
0.75	$540.4\overline{6}$	$684.\overline{3}$	$661.3\overline{6}$
0	$0.0\overline{6}$	0	0

Table A.8: Total successes after 1,000,000 time steps for various trace decay rates in mountain car

ϵ	0	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1
Return	0	0	$1.8\overline{3}$	66.9	627.7	$640.8\overline{3}$	795.1	$2467.4\overline{6}$

Table A.9: Total return after 1,000,000 time steps for various values of epsilon in mountain car

α_r	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1	0.5	1
Return	$553.9\overline{6}$	$2467.4\overline{6}$	$2010.2\overline{3}$	0	0	152.2	107.5	$59.8\overline{3}$	$48.2\overline{6}$

Table A.10: Total return after 1,000,000 time steps for various step sizes for the average reward in mountain car

$\alpha_{\mathbf{w}}$ \ α_θ	0.0001	0.001	0.01	0.1
0.0001	0	0	0	57.3
0.001	0	0	0	$607.1\overline{6}$
0.01	0	0	$2467.4\overline{6}$	$1322.8\overline{6}$
0.1	$29.1\overline{6}$	38.7	$37.5\overline{3}$	$37.2\overline{6}$

Table A.11: Total return after 1,000,000 time steps for various step sizes for low level actions in mountain car

$\alpha_{\mathbf{w}}$ \ α_θ	0.0001	0.001	0.01	0.1
0.0001	0	0	0	0
0.001	0	0	0	0
0.01	0	0	0	$0.0\overline{6}$
0.1	$0.\overline{6}$	$0.\overline{6}$	$0.6\overline{3}$	0.5

Table A.12: Total return after 1,000,000 time steps for various step sizes for low level actions with truncated distributions in mountain car

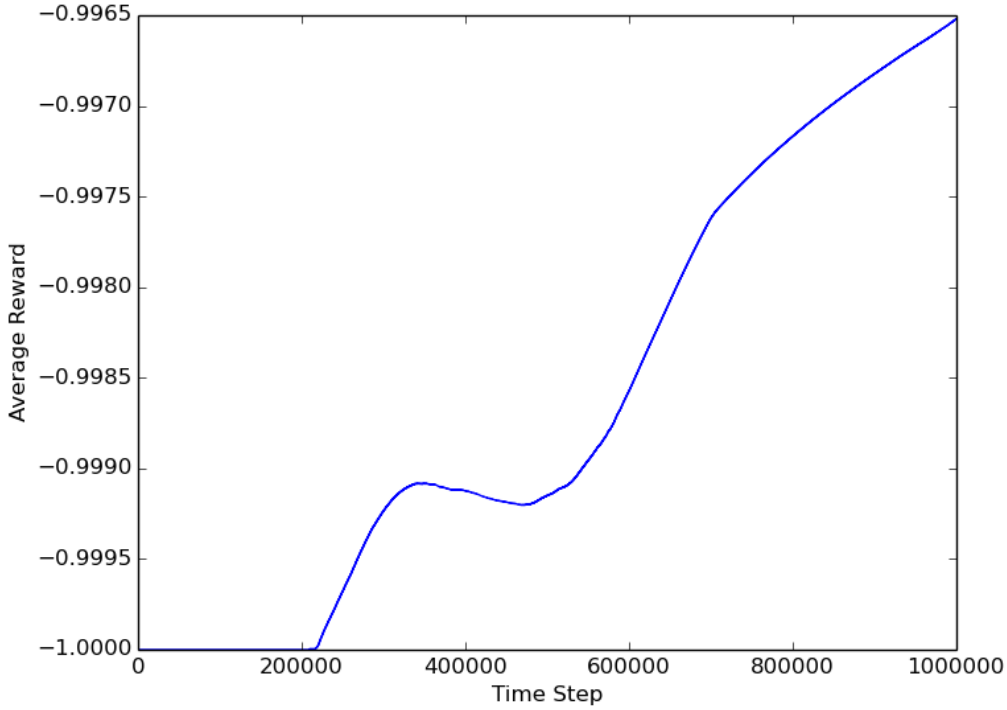
$\alpha_{\mathbf{w}}$ \ α_θ	0.0001	0.001	0.01	0.1
0.0001	0	0	$2225.8\overline{3}$	$3485.0\overline{6}$
0.001	0	$1674.5\overline{6}$	$7858.9\overline{6}$	$4196.5\overline{6}$
0.01	$1451.7\overline{3}$	$7382.6\overline{3}$	$7604.0\overline{3}$	$1533.8\overline{3}$
0.1	$3678.0\overline{6}$	7046.1	$6900.\overline{3}$	$244.0\overline{3}$

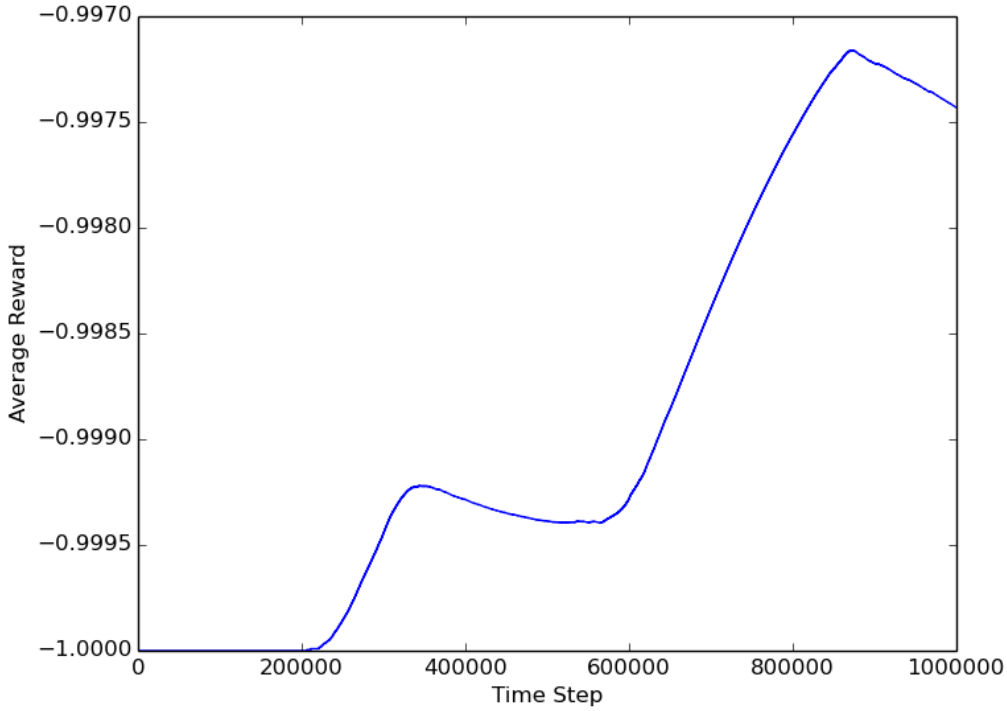
Table A.13: Total return after 1,000,000 time steps for various step sizes without using multi level actions in mountain car

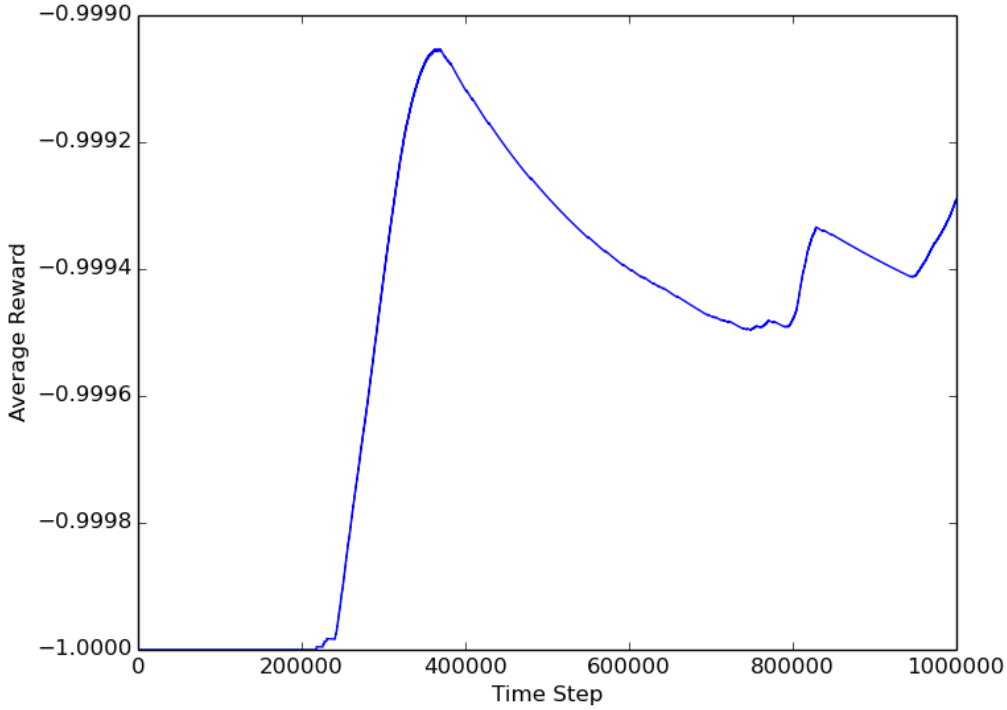
Appendix B

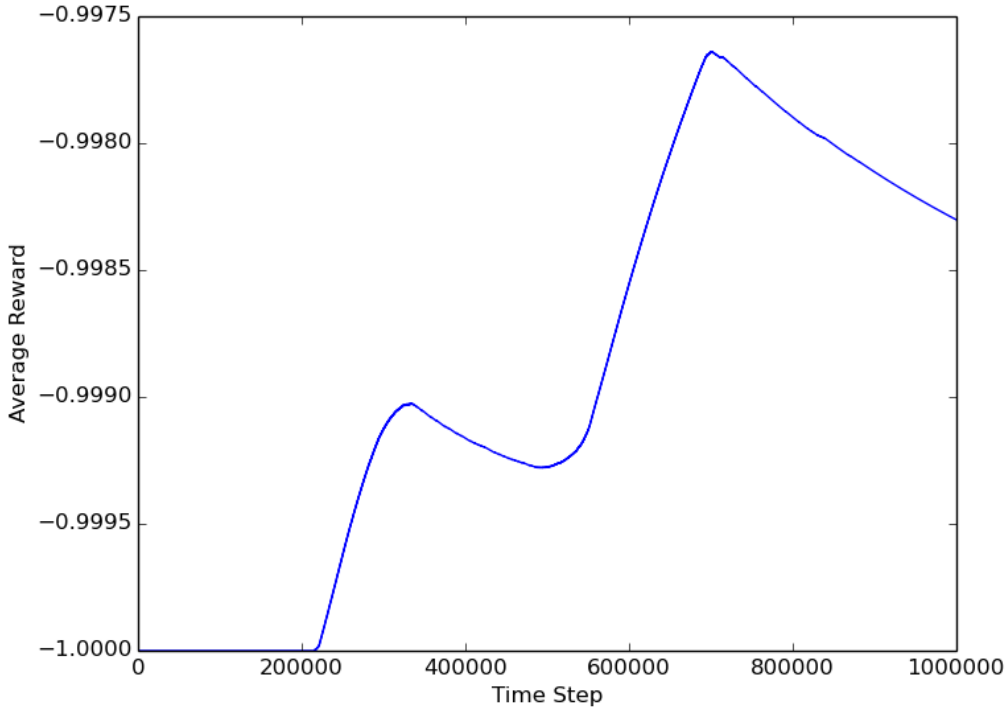
Mountain Car Learning Curves

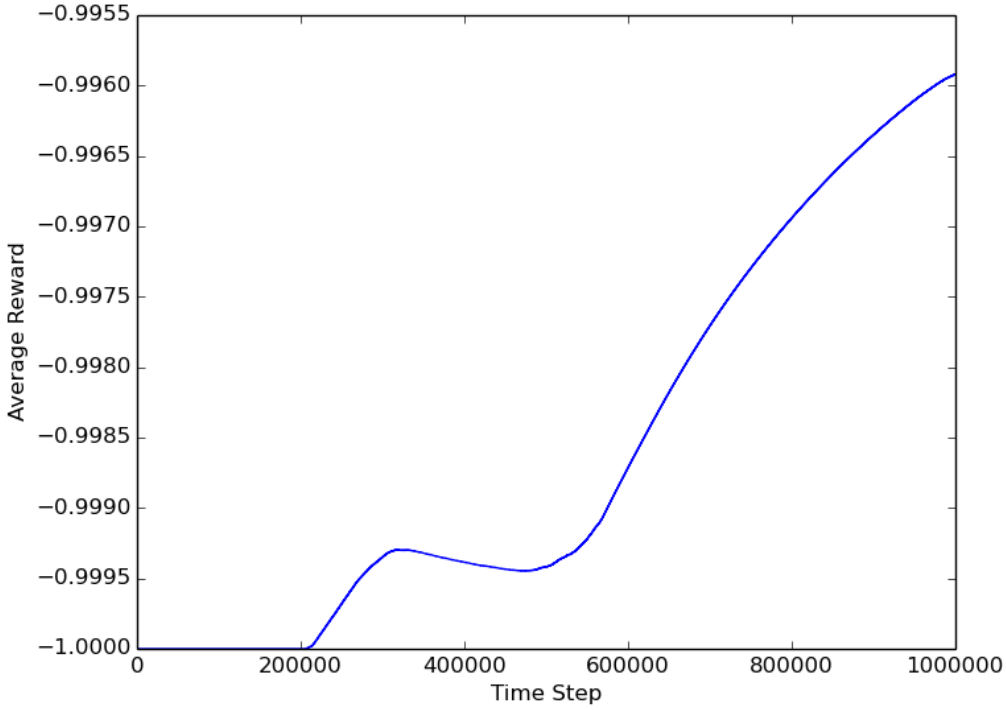
This appendix contains a few of the learning curves for single runs of mountain car with multilevel action selection to show the differences and similarities between runs.


−0.9965
−0.9970
−0.9975
−0.9980
−0.9985
−0.9990
−0.9995
−1.0000
Average Reward
0
200000
400000
600000
800000
1000000
Time Step


−0.9970
−0.9975
−0.9980
−0.9985
−0.9990
−0.9995
−1.0000
Average Reward
0
200000
400000
600000
800000
1000000
Time Step


−0.9990
−0.9992
−0.9994
−0.9996
−0.9998
−1.0000
Average Reward
0
200000
400000
600000
800000
1000000
Time Step


−0.9975
−0.9980
−0.9985
−0.9990
−0.9995
−1.0000
Average Reward
0
200000
400000
600000
800000
1000000
Time Step


−0.9955
−0.9960
−0.9965
−0.9970
−0.9975
−0.9980
−0.9985
−0.9990
−0.9995
−1.0000
Average Reward
0
200000
400000
600000
800000
1000000
Time Step

Appendix C

Average Reward Calculations

This appendix contains the calculations determining the conditions for the average reward to remain the same from one failure to the next in cart pole. r is the return at time t. Δt is the time until the next failure.

$$\frac{r}{t} = \frac{r-1}{t+\Delta t}$$
$$\frac{r(t+\Delta t)}{t} = r-1$$
$$t+\Delta t = \frac{t(r-1)}{r}$$
$$\Delta t = \frac{t(r-1)}{r} - t$$
$$\Delta t = t(\frac{r-1}{r} - 1)$$
$$\Delta t = \frac{-t}{r}$$