Reinforcement Learning with Function Approximation in Continuing Tasks: Discounted Return or Average Reward?

Lucas Descause

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2019

Abstract

Discounted returns, a common metric used in Reinforcement Learning to estimate the value of a state or action, has been claimed to be ill-defined for continuing tasks using value function approximation [11]. Instead, average rewards have been proposed as an alternative [11, 9]. We conduct an empirical comparison of discounted returns and average rewards on three different continuing tasks using various value function approximation methods. Results show a clear performance advantage for average rewards. Through further experiments we show that this advantage may not be caused by partial observability, as hypothesized by the claims against discounted returns [11, 9]. Instead, we show that this advantage may be due to average reward values requiring a simpler function to estimate than discounted returns.

Acknowledgements

I would like to thank my supervisor, Stefano Albrecht, for his support and guidance throughout this project.

Table of Contents

Chapter 1

Introduction

Reinforcement Learning aims to solve decision-making problems using only trial and error experience. An artificial agent observes the problem environment and must choose which action to perform to maximize its rewards. Maximizing immediate rewards does not always result in an optimal policy as such behavior could lead away from more distant but greater rewards. It is, therefore, necessary to define which rewards to maximize. Discounted returns [11] have been most commonly used to measure the immediate and future returns caused by an action. This measure considers both recent and distant rewards and assigns different weights to rewards based on recency. Typically, a higher weight is given to recent rewards.

Value Function Approximation (VFA), using models such as neural networks, can be used to estimate the returns of possible actions given the state of the environment. This allows RL to be used with more complex problems containing large state spaces by generalizing the learned policy to unseen states. It has been suggested [11], that discounted returns are ill-defined when used with VFA in tasks with no end state where the decision problem continues endlessly. We refer to such tasks as continuing tasks. Average rewards, a metric which weights recent and distant rewards equally, has been proposed to replace discounted returns for continuing tasks using VFA.

There has been theoretical work [9, 11] arguing that discounted returns are ill-defined when using VFA in continuing tasks. To our knowledge, however, the practical difference between discounted returns and average rewards in this setting has not been investigated. We conduct an empirical comparison of both methods across different continuing tasks and different VFA to verify whether depreciating discounted returns in this setting is justified. We limit our experiments to action-value methods and use both Asynchronous Q-learning [7] and SARSA [7] for our comparison. We justify the use of Asynchronous Methods as they have been shown to obtain state-of-the-art results on common benchmarks [7]. For VFA both linear models and neural networks are used as they are the most commonly used methods of function approximation in RL [11].

Furthermore, we implemented three continuing tasks of various complexity for our experiments. Open-source RL environments are widely available for episodic tasks but to our knowledge, few or no continuing tasks are available. The implemented tasks are made available publicly to facilitate research of continuing RL tasks.

To evaluate the different algorithms, we freeze the policy during and after training and measure the average reward obtained by the greedy policy. The experimental results suggest that average rewards outperform discounted returns for simple VFA (such as a linear model) but that both obtain similar performance when VFA is complex enough (for example a neural network with sufficient depth and number of hidden units). We hypothesize that simple VFA causes lower observability, as it causes more state aggregation, which is in turn responsible for the lower performance of discounted returns. We design further experiments using Partially Observable Markov Decision Processes (POMDPs) [9] to confirm this, however, results seem to indicate that partial observability is not the cause of the performance difference between discounted returns and average rewards. Finally, we investigate whether discounted return values require a more complex function to estimate than average reward values.

The report is structured as follows: Ch. 2 provides the necessary background and related work. Ch. 3 introduces the methodology for the empirical comparison between discounted returns and average rewards. The results and analysis of the comparison are presented in Ch. 4. In Ch. 5 we investigate through further experiments whether partial observability is the cause of the performance difference between discounted returns and average rewards. In Ch. 6, we investigate whether average rewards are easier to learn than discounted returns. Finally, Ch. 7 concludes our analysis.

Chapter 2

Background

RL consists of an agent learning a policy π that maximizes rewards using experience gathered by interacting with an environment. Environments are modeled by a Markov Decision Process (MDP) defined by a set S of states, a set A of actions, the transition probabilities $P(S_{t+1}|A_t, S_t)$ and a reward function $r(S_t, A_t)$. At each time step, the agent observes the current state S_t, performs action A_t, moves to a new state S_{t+1} and obtains a reward R_{t+1}. Note that the environment follows a Markov assumption meaning that the transition probabilities and rewards depend only on the current state and action. Consequently, the agent only requires to observe the current state to choose an action.

The aim of Reinforcement Learning is to find an optimal policy defined by the mapping $State \rightarrow Action$ that maximizes the rewards obtained by the agent. The background provided in this section is mostly issued from Sutton and Barto [11].

2.1 Episodic vs. Continuing Tasks

RL tasks can be either episodic or continuing. Episodic tasks are defined by any task with an end state. This includes many games such as Chess or Pac-Man, as in both, the game can end when the player loses or wins. Continuing tasks do not contain any end states and can continue indefinitely. This includes many business processes as a business continues to run indefinitely and has no end or goal state.

2.2 Discounted Returns and Average Rewards

Both discounted returns and average rewards, aim to quantify the return (present and future rewards) obtained by the agent. We define G_t as the returns obtained after time step t. A simple definition of returns would be to sum all future rewards, however we would end up with an infinite sum when the task has no end state. To avoid this discounted returns define the return as the sum of discounted rewards:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \text{ defined recursively as: } G_t = R_t + \gamma G_{t+1}$$

where $T = \infty$ in the continuing case. The discount factor $0 < \gamma < 1$ is necessary to ensure that the sum converges to a finite value.

Average rewards, on the other hand, define G_t as the sum of the differences between the rewards and the average rewards:

$$G_t = \sum_{k=0}^{T} (R_{t+k+1} - r(\pi)) \text{ defined recursively as: } G_t = (R_t - r(\pi)) + G_{t+1}$$

where $r(\pi)$ is equal to the average reward obtained by policy π.

2.3 Value Estimation

The value of a state or action is defined as the expected return obtained from that state or action. We define:

- $Q_\pi(S_t, A_t) = E[G_t \mid S_t, A_t]$ the value of taking action a while in state s when following a policy π.

- $V_\pi(S_t) = E[G_t \mid S_t]$ as the value of being in state s when following a policy π.

If we can perfectly estimate $Q_\pi(S_t, A_t)$ we can greedily choose the action with maximum value for the current state to obtain an optimal policy. We describe two methods to estimate these action-values, Q-Learning [11] and SARSA [11]. For both methods we use bootstrapping to estimate $Q_\pi(S_t, A_t)$ using:

$$Q_\pi(S_t, A_t) = E\Big[G_t \mid S_t, A_t\Big] = E\Big[R_t + \gamma G_{t+1} \mid S_t, A_t\Big]$$

Additionally, for both methods the mean is estimated using an incremental weighted mean $\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1})$ where μ_k is the mean estimate at step k and x_k is the k^{th} observation. High values of α can be used to give a higher weight to recent observations. Q-Learning and SARSA differ in their estimate of G_{t+1}.

2.3.1 Q-Learning

Discounted Returns

We estimate G_{t+1} by $\max_a Q(S_{t+1},a)$: the expected return at state S_{t+1} assuming we follow a greedy policy. Additionally, a weighted iterative mean is used to approximate the expected value of the returns. This leads to the following recursive update rule to estimate $Q(S_t,A_t)$.

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \underbrace{[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)]}_{\delta}$$

where δ represents the error between the new and previous estimate. Additionally, α can be interpreted as the learning rate.

Average Rewards

Average rewards differs from discounted returns only in the definition of the returns which leads to the following update rule for average rewards:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \underbrace{[(R_{t+1} - r(\pi)) + \max_a Q(S_{t+1},a) - Q(S_t,A_t)]}_{\delta}$$

$r(\pi)$ is defined as the incremental average of the returns and is updated every step using $r(\pi) \leftarrow r(\pi) + \beta\delta$ where a high β gives a higher weight to recent returns in the average.

2.3.2 SARSA

SARSA differs from Q-Learning only in the estimation of G_{t+1}. We estimate G_{t+1} with $Q_\pi(S_{t+1},A_{t+1})$, the expected return at state S_{t+1} assuming we follow policy π. This leads to the following update rules:

Discounted Returns

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \underbrace{[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]}_{\delta}$$

Average Rewards

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \underbrace{[(R_{t+1} - r(\pi)) + Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]}_{\delta}$$

2.4 Exploration

An optimal policy chooses the action with maximum value for each state. The action-values are, however, initially unknown. It is, therefore, necessary to explore different states and actions to evaluate their values. On the other hand, it is also necessary to exploit the agent's current experience and value estimations in order to obtain rewards and gradually improve the policy. This is known as the trade-off between exploration and exploitation.

It is common to use ε-greedy policies [11] to introduce exploration. An ε-greedy policy chooses the action with maximum value with probability $1-\varepsilon$ and a random action with probability ε. It is common to initialise $\varepsilon = 1$ and to anneal it to a low value during training. This allows for a large amount of exploration at the beginning of training but later focuses exploration on more relevant states and actions.

2.5 Value Function Approximation (VFA)

The update rules given in the previous section are used for tabular methods: the action value $Q(S_t, A_t)$ is stored for every state and action encountered. When the state space is large it can be difficult to explore the entire space and to store every value. Additionally, tabular methods do not allow to generalize the policy to unseen states. VFA can be used to learn a function that approximates $Q(S_t, A_t)$ and allows to generalize to unseen states. The function uses a feature vector of the current state as input and outputs the value of each action for that state. A differentiable function with weights **w** is typically used as it allows the use of gradient methods.

Instead of directly updating $Q(S_t, A_t)$ like in the tabular case, we now update **w** by performing gradient descent to minimize the error, $\frac{1}{2}\delta^2$ (we define δ in Section 2.3):

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \frac{1}{2}\delta^2}{\partial \mathbf{w}} = \mathbf{w} + \alpha \delta \nabla Q(S_t, A_t, \mathbf{w})$$

Note that we only differentiate the current estimate $Q(S_t, A_t, \mathbf{w})$ and not the target ($R_t + \max_a Q(S_{t+1}, a)$ in the case of discounted Q-learning). For this reason, these methods are referred to as semi-gradient methods [11].

2.6 Asynchronous Methods

Some problems arise when combining RL with VFA. Mnih et al. [7] propose Asynchronous Methods to address these problems and to combine RL and neural networks VFA successfully. The first problem arises due to the approximation of a moving target ($R_{t+1} + \gamma \max_a Q(S_{t+1}, a, \mathbf{w})$ in the case of discounted Q-learning). With every weight update, we aim to get our current estimate closer to the target, but updating the weights of the current estimate causes the target to change. To get around this issue, we use two value functions. One for the target and one which we update. Every I_{target} steps, we update the target VFA with the new weights.

The second problem is caused by the correlation between consecutive updates. When updates are performed on consecutive state-actions the updates become correlated which inhibits learning. To avoid this, Asynchronous Methods use multiple parallel agents each with their own environment. All parallel agents update the same VFA weights, but since they are likely to explore different parts of the state space at any given time, the updates will not be correlated. Additionally, running multiple agents in parallel has shown to improve convergence time.

2.7 Problems with Discounted Returns

2.7.1 General Problems

Schwartz [8] identifies some of the limitations of discounted returns. First, discounted returns are strongly related to the net present value in economics [10]. Obtaining capital X today is worth more than obtaining it tomorrow as that capital could be invested today to yield a higher sum the next day. Discounting in this context serves a clear purpose and is justified. However, in many tasks, rewards are unrelated to capital which possibly makes discounting obsolete.

Additionally, a possible justification for discounting includes the finiteness of the agent's life. When end states are present, there is a possibility that the agent will 'die' before reaching any future rewards. Distant rewards are, therefore, worth less because there is a chance the agent will not reach them. This does not, however, justify the use of discounting for continuing tasks as there are no end states.

Finally, the average reward or cumulative reward is usually used as a performance metric to evaluate RL methods. If we aim to maximize these undiscounted metrics, it makes little sense to use discounted returns which aims to maximize a discounted metric.

Schwartz [8] suggests that discounting may only be used to obtain finite values from an infinite sum of rewards. These arguments suggest that discounted returns may be inappropriate in continuing tasks and in tasks where rewards are unrelated to capital.

2.7.2 Problems with VFA

Singh et. al [9] argue that discounted returns should not be used with POMDPs, MDPs where the agent may not be able to distinguish between every state. Action-value methods aim to maximize V(s) for each s which may be impossible when partial observability is introduced.

The POMDP in Figure 2.1 is introduced as an example [9]. Assuming initial values of 0, choosing action A in state 1 increases $V(1)$ but decreases $V(2)$. On the other hand, choosing action B in state 1 increases $V(2)$ but decrease V(1). This suggests that no policy can maximize the values for each state, and that discounted returns are, therefore, inappropriate in POMDPs.

Singh et. al [9] argue that in the episodic case, it is sufficient to optimize the value of the starting state. In the continuing case, however, they propose to optimize the mean value over the policy state distribution:

$$\sum_{s \in \mathcal{S}} \mu_\pi(s) V_\pi(s)$$

where $\mu_\pi(s)$ is the probability of being in state s when following policy π. Optimizing this metric has been shown to be equivalent to optimizing average rewards [11]. They, therefore, argue that average rewards should be used in POMDPs.

But how are POMDPs related to VFA? The function approximation method is often not complex enough to model a 1 to 1 mapping between states and values, like tabular methods, and is required to generalize. This generalization can lead VFA to predict similar values for similar states, causing a behavior similar to state aggregation and the introduction of partial observability. Sutton and Barto [11], therefore, claim that discounted returns should not be used with VFA in continuing tasks.

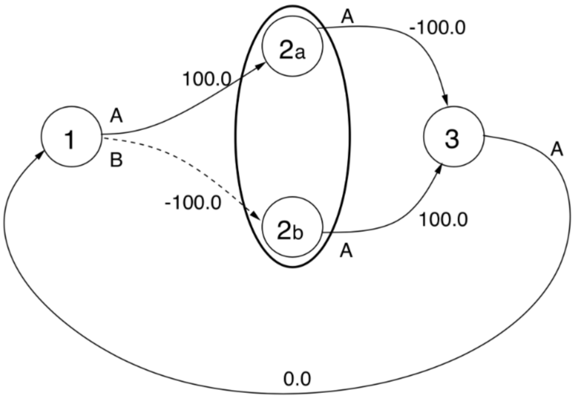

Figure 2.1: Example POMDP [9]. State 2a and 2b are indistinguishable by the agent.

2.8 Related Work

To our knowledge, there has not been any empirical comparison of discounted returns and average rewards using VFA in continuing tasks. However, Schwartz [8] and Mahadevan [5] have compared both methods using tabular Q-Learning and have found that average rewards achieve better final performance than discounted returns when evaluated using average rewards. Additionally, both comparisons use tasks with relatively small state spaces. Because these comparisons do not use VFA, they cannot be used to assess whether discounted returns should be discontinued for continuing tasks using VFA.

Tsitsiklis and Van Roy [13] suggest analytically that using discounted returns (with high γ) is equivalent to using average rewards in continuing tasks using linear VFA. They mention, however, that this claim should be verified with numerical experimentation, which we deliver in our experiments.

Chapter 3

Empirical comparison: Methodology

3.1 Tasks

The different algorithm configurations described in this chapter are compared using three different tasks of different complexity. Using multiple tasks ensures that the results generalize to tasks in different domains and of different complexity. This section provides a description of the tasks used. (The tasks have been made publicly available for future research: https://github.com/Lucas-De/RL-Continuing-Tasks)

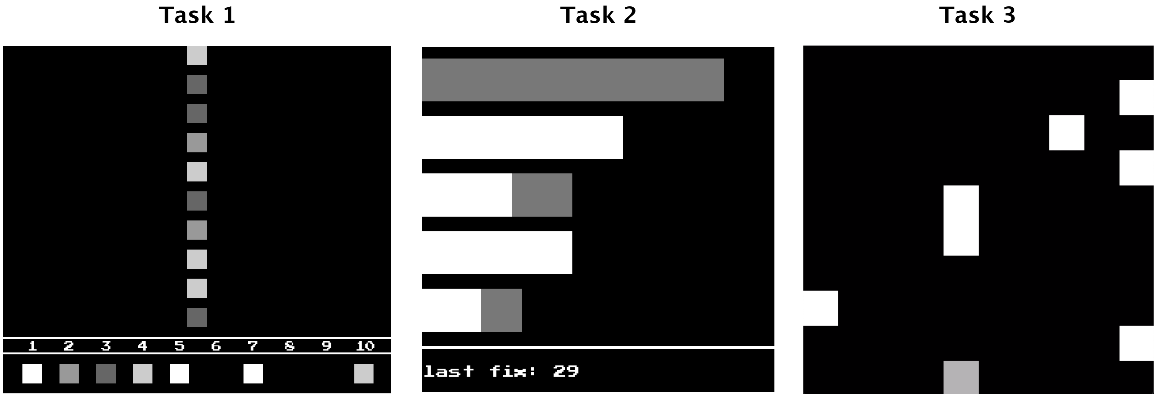

Figure 3.1: Visualisation screenshots of task 1, 2 and 3

3.1.1 Task 1

The first task is inspired by the queuing problem described by Sutton and Barto [11]. The environment consists of a queue containing customers of uniformly distributed priorities of 1, 2, 4 and 8. These customers can be assigned to 10 servers. The agent must decide whether to assign the customer at the front of the queue to a server or reject the customer. Assigning a customer of priority p to server results in a reward of p. If all servers are full and the agent accepts a customer, the customer is automatically rejected. We introduce a reward of -1 in this event in order to avoid policies where the agent always accepts customers. At each time step, each server has a probability of 0.06 of becoming free. Finally, the queue remains full at all times.

The state representation consists of a vector of size 11 containing the priority of the first 10 customers in the queue, as well as the number of free servers. This results in over 1.04×10^7 unique states. The agent has two actions available: accept, reject. This results in over 2.08×10^7 different action-values. While it is possible to represent a policy using tabular methods for a state space of this size, VFA allows generalizing to unseen states and therefore may require less exploration and less training steps. A tabular method, on the other hand, would require at least 2.08×10^7 training steps to update each action value once. This clearly justifies the use of VFA for this task.

3.1.2 Task 2

The second task is inspired by the factory simulation introduced by Mahadevan et. al [6]. The environment consists of a machine that can produce one of 5 products and store them in buffers of respective sizes (30, 20, 15, 15 and 10). Demands for each product have different stochastic inter-arrival times and prices (9, 7, 16, 20 or 25). A demand arrival for a product results in a reward equal to the product price. Failures can occur randomly at any point and force a repair resulting in a reward of -5000 and the interruption of all production. Failures can be avoided by performing regular maintenance. Performing maintenance results in a reward of -500 and stops production for a shorter time than a repair. We refer to Appendix A for the details regarding the distributions of demand arrivals, failure arrivals, repairs times and maintenance times.

After the completion of production, the end of a repair, or the end of maintenance, the agent must decide to produce one of the 5 products or to perform maintenance. When production starts for a product, it may not stop until the buffer is full or a failure occurs.

The state representation consists of a vector of size 6 containing the number of products present in each buffer, as well as the time since the previous repair or maintenance. Because this last feature is continuous and unbounded, the state space is theoretically infinite. We, however, attempt to estimate its size if tabular methods were to be used. Assuming we only use integer values in the range of [50-750] for the time since last repair/maintenance, we obtain a state-space of size 9.45×10^8. Since the agent can perform 6 actions this results in 5.67×10^9 action values. Assuming we store each action-value as a Python float, which occupies 24bytes, this would result in a minimum memory requirement of 136 gigabytes. The nature of the state-space, as well as its size, clearly justifies the use of VFA.

3.1.3 Task 3

For the last task, we implemented a simple game where cubes fall from the top of a 10x10 pixel screen. The agent must move left, right, or remain in its current position to collect the squares and obtain a reward.

- Each row contains a white square with probability 0.5. The square is placed uniformly in one of the 10 columns.
- At each time step, each row is shifted down by one pixel.
- The grey square on the bottom row represents the agent
- At each time step, the agent can move left or right by one pixel or stay in its current position. If the agent and a white square are in the same position, a reward of 1 is received.

As there are more white squares than the agent can collect, the agent must not only learn to collect the white squares but also learn which ones to collect to maximize future rewards. (A video example is available at: https://youtu.be/P1GFhcgVdV8)

The state-representation consists of a 10x10 pixel matrix where 0 represents a black pixel and 255 for a white pixel. The first 9 rows may each have 11 different configurations (the white square can be present in one of 10 positions or be absent from the row) while the bottom row may have 100 (10 positions for the grey square × 10 positions for the white square, including the absence of one). This results in a state-space of size 2.33×10^{11}.

We chose this task as it allows the use of image state-representation and a larger state space than the two previous tasks. This task also provides a relatively different domain to task 1 and 2. Additionally, the state-representation remains simple enough to satisfy the time requirements for this project. A more complex task may require a more complex VFA which would increase training time and may not allow to run the desired experiments in the time available.

3.2 Algorithms

We implemented Asynchronous Q-Learning [7] and Asynchronous SARSA [7] each with discounted returns and average rewards. Asynchronous methods were chosen as they show state of the art results on common RL benchmark tasks. We chose to use both Q-Learning and SARSA to allow to draw conclusions about both on and off-policy methods. Both methods are implemented using both linear function approximation and neural networks. This allows to investigate the effect of VFA complexity of discounted returns and average rewards.

We reuse the parameters from the original paper on Asynchronous methods [7] with the exception of parameters present in the update equation for discounted returns (α, β) and average rewards (α, γ). We chose to perform a random search for these parameters as they are expected to have the most effect on performance (we provide more details for this in the evaluation section). We use 16 parallel asynchronous agents for each run as it results in faster convergence [7]. Finally, we use the same exploration method as the original paper: each parallel agent has a different exploration rate $\varepsilon_i = 1$ which is annealed to 0.1, 0.01, 0.5 with respective probabilities 0.4, 0.3, 0.3 over the first 4 million steps. Each run for task 1 and 2 consists of 5 million steps as this showed to be enough to obtain convergence. For task 3, each run was limited to 10 million steps to allow the completion of experiments in the available time. While this is not sufficient to observe the limit behavior, it is sufficient to observe significant differences between the methods compared.

3.3 Value Function Approximation

For task 1 and 2 we use a linear model (defined as a neural network with no hidden layer) and a 2 hidden layer neural network with ReLu [4] activation. We use $(length_{input} - length_{output})/2$ neurons in each hidden layer to attempt to scale the complexity of the network to the complexity of the task.

For task 3, we use a similar linear model, and a convolutional neural network (CNN) [3]. Because the task is more complex and the initial neural network showed relatively poor performance, we added an additional more complex neural network. The architectures of the two CNNs are described in Figure 3.2.

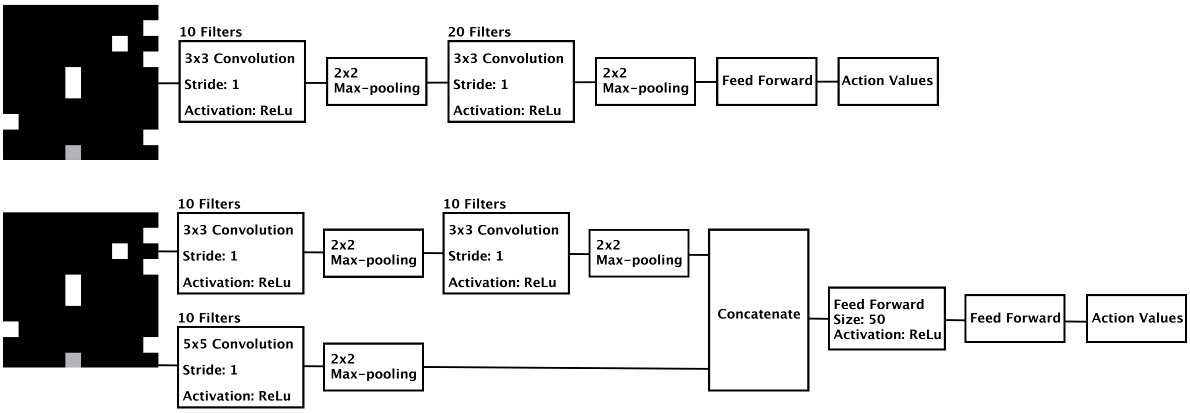

Figure 3.2: Architecture of the two CNNs used in task 3. V1 (top) and V2 (bottom). The leftmost block represents an image input for task 3

3.4 Evaluation

3.4.1 Parameter Optimization

It is important when comparing two methods empirically, that both methods are optimized to the same extent. To ensure that every algorithm configuration is optimized to a similar extent, we perform a random search over the respective parameters for each configuration. In the case of discounted returns, we search over the learning rate $\alpha_{discounted}$ and the discount factor γ. For average rewards, we perform the search over the learning rate $\alpha_{average}$ and the parameter β (Mahadevan [5] highlights the sensitivity of average rewards to these parameters). The parameters $\alpha_{discounted}$, $\alpha_{average}$ and β were sampled from a $LogUniform(10^{-5}, 10^{2})$ distribution. The discount factor γ must approach 1 for discounted returns to approximate average rewards, it was therefore sampled from $1 - LogUniform(10^{-5}, 10^{2})$. A random search of 100 samples is performed for each algorithm configuration on each task. We chose to use random search as it is computationally cheap, can be run in parallel and can provide better results than grid search [1].

3.4.2 Evaluating Policies

Every one million steps during training, we freeze the value function weights and perform an offline evaluation of the resulting greedy policy (Colas et al. [2] advocate comparing RL methods offline rather than using the performance during training). To do so, we perform 5 runs (each with random initialization and different seed) of 50,000 steps and measure the average reward obtained in each run. It is necessary to perform this for more than a single run as different initialisation and seeds could lead to different sections of the state space being explored.

We compute the average reward for each run by dividing the cumulative reward by the number of steps. The average reward obtained of each of the 5 runs is then averaged and is used as a performance metric for the greedy policy.

3.4.3 Comparison of Discounted Returns and Average Rewards

To compare the two methods, we select the 30 runs from the parameter search with the best greedy post-training performance (measured as described above). We average this performance metric over the 30 best runs and provide a 95% confidence interval for the mean. This is done for every one million steps to form a learning curve and compare convergence properties of the two methods. Finally, we use a Kolmogorov-Smirnov test to test the null hypothesis that both the 30 best runs for discounted returns and average rewards have a post-training performance drawn from the same distribution. This statistical test was chosen as it makes no assumption on the distribution of the two samples. Colas et al. [2] recommend using a Bonferroni correction due to an increased chance of incorrectly rejecting the null hypothesis when performing multiple tests. Because we perform 14 tests and want a significance of 0.05, we use a significance of $0.05/14$ for each test.

Chapter 4

Empirical comparison: Results

The results presented in Figures 4.1, 4.2 and 4.3 show the results of the empirical comparison. The grid search results are present in Appendix B. In this chapter, we aim to identify performance differences and their possible causes.

4.1 VFA Complexity

The results for task 1 and 2 (with the exception of Q-Learning in task 2) show a clear advantage for average rewards when linear VFA is used. This runs counter to the hypothesis presented by Tsitsiklis and Van Roy [13]. Discounted returns and average rewards achieve similar performance when the neural network is used however. This could indicate that both methods are equally performant when VFA is complex enough. In task 3, average rewards consistently outperform discounted returns, even when using the most complex VFA, which does not seem to confirm this hypothesis. It is possible that neither of the neural networks used is complex enough to observe similar performance between discounted returns and average rewards. Alternatively, it is possible that their performance will indeed be similar after full convergence.

As discussed in Ch.2, a linear model, having fewer parameters, can distinguish between fewer states than a neural network. This causes a higher level of state aggregation which is equivalent to decreasing the observability of the environment. As Singh et. al [9] indicate, discounted returns seem to be inappropriate when partial observability is introduced as optimizing the value in each state becomes impossible. We hypothesize that the difference in performance between discounted returns and average rewards is due to the introduction of partial observability by a VFA too simple to distinguish between a sufficient amount of states. We investigate this further in Ch. 5.

4.2 On-policy vs Off-policy

Both task 1 and 2 do not show any noticeable differences in final performance or convergence between Q-Learning and SARSA. In task 3, however, while average rewards have similar performance for both Q-Learning and SARSA, discounted returns show significantly poorer results for Q-Learning. Thrun et al. [12] have suggested that the combination of VFA, Q-learning and discounted returns could lead to an over-estimation of action-values and result in poor performance. This claim is a possible explanation for the different behaviors observed in Q-Learning and SARSA.

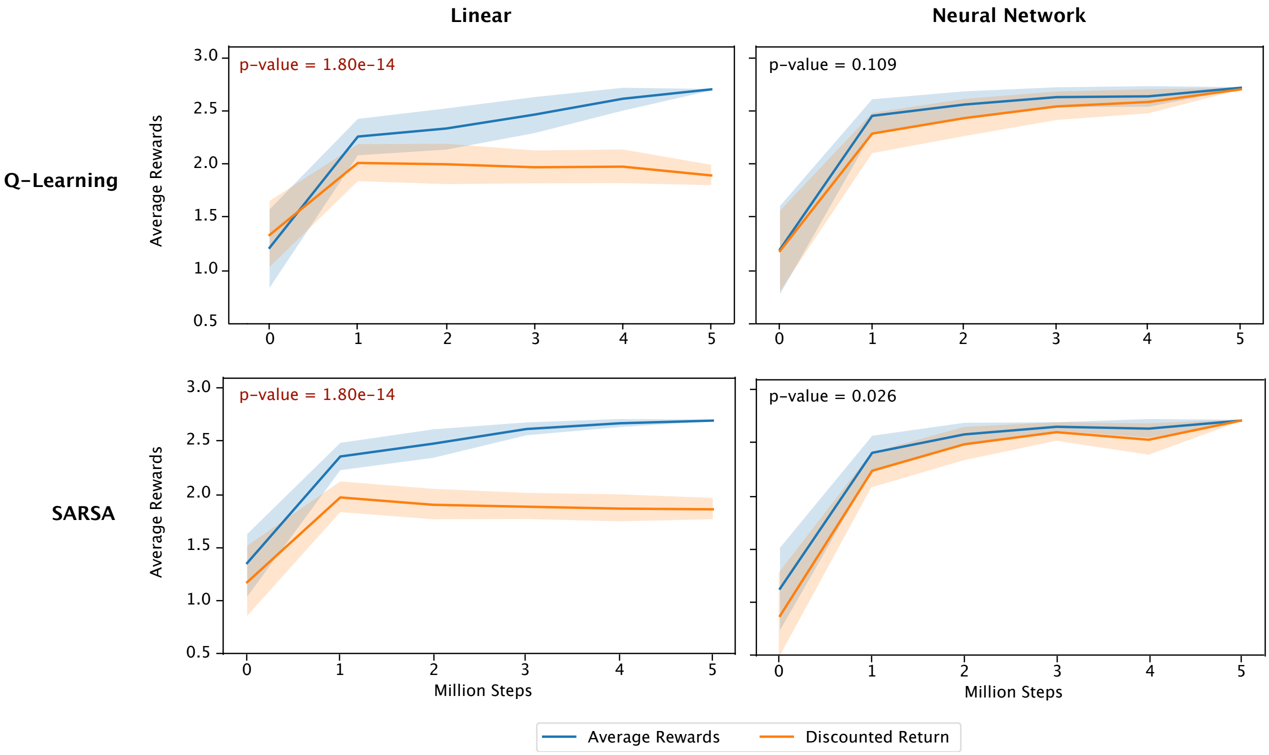

Figure 4.1: Learning Curves for task 1. Each curve represents the average reward averaged over the 30 best of the given algorithm configuration. The shaded areas represent the 95% confidence interval for the mean. The p-value for the Kolmogorov-Smirnov test is presented in red if the difference in final performance is significant

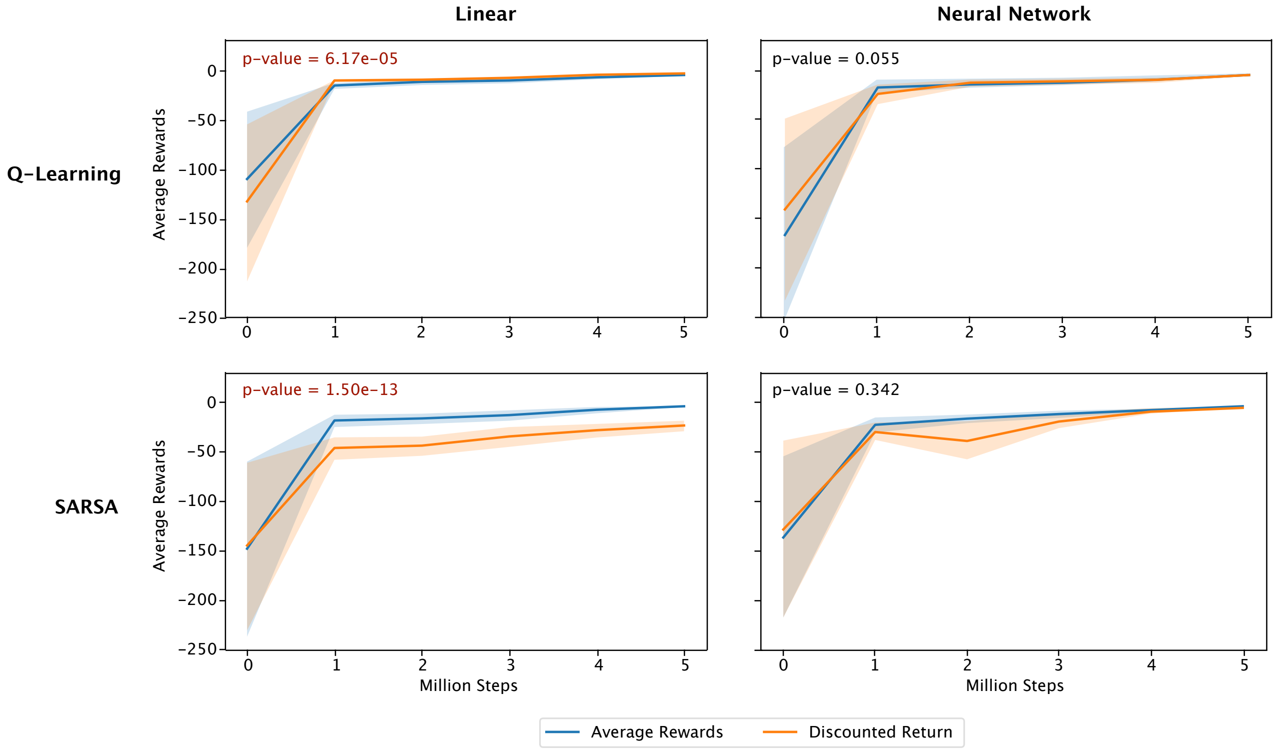

Figure 4.2: Learning Curves for task 2. Each curve represents the average reward averaged over the 30 best of the given algorithm configuration. The shaded areas represent the 95% confidence interval for the mean. The p-value for the Kolmogorov-Smirnov test is presented in red if the difference in final performance is significant

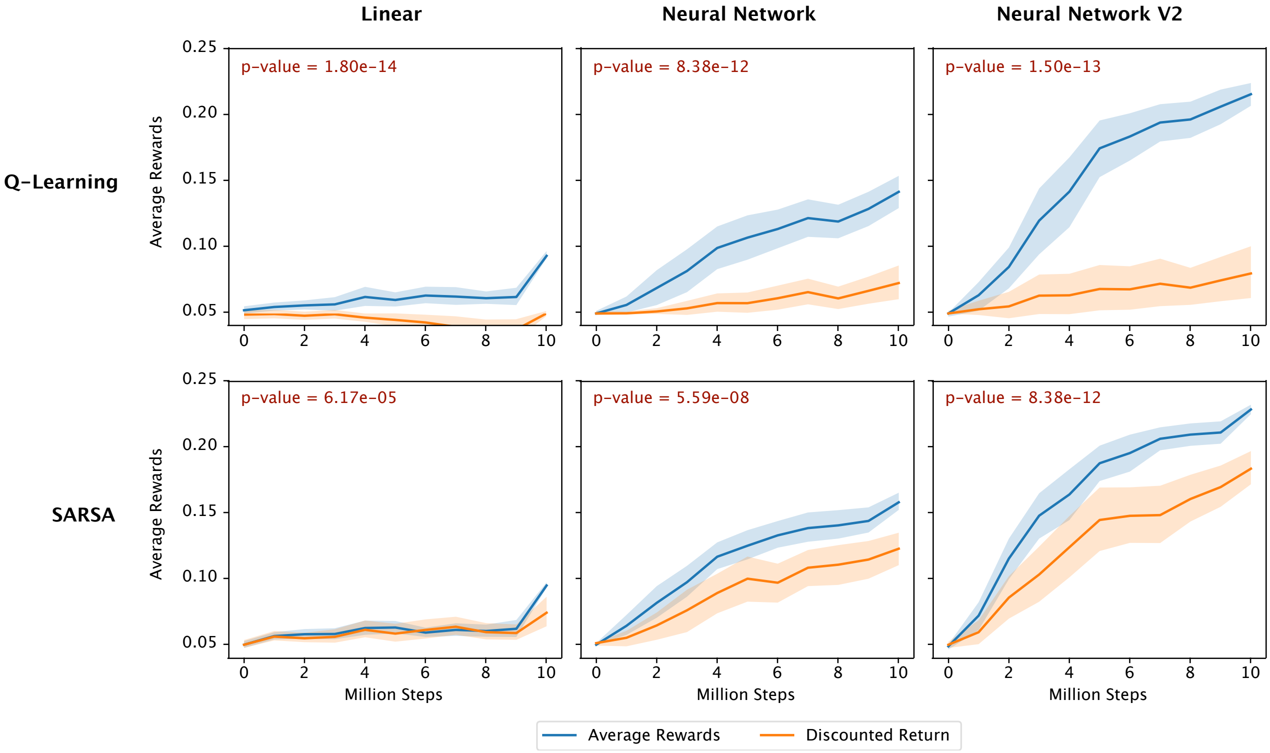

Figure 4.3: Learning Curves for task 3. Each curve represents the average reward averaged over the 30 best of the given algorithm configuration. The shaded areas represent the 95% confidence interval for the mean. The p-value for the Kolmogorov-Smirnov test is presented in red if the difference in final performance is significant

Chapter 5

POMDP: Discounted Returns or Average Rewards

5.1 Methodology

In the previous chapter, we hypothesized that the performance difference between discounted returns and average rewards in continuing tasks using VFA was due to the introduction of partial observability by simple VFA. To test this hypothesis, we dismiss VFA and use tabular methods with manually introduced partial observability. If the hypothesis is correct, we expect to see a higher performance for average rewards and the performance difference between discounted returns and average rewards to increase as the observability decreases.

We use simplified versions of the three tasks from the VFA experiments to allow the use of tabular methods. For each task, we compare discounted returns and average rewards on three different versions of the task, each with different levels of observability. To introduce partial observability, we create task-specific state aggregations. For each task, observability and algorithm configuration, we perform a grid search similar to the one described in Ch.3. The 30 best runs are then used for comparison. Performance is measured similarly to Ch.3 using average rewards measured offline after 5 million training steps (a high number of steps was used to ensure convergence).

Task 1: To allow the use of tabular methods, we reduce the task's state space. We describe the high, medium and low observability version of the state space.

- High: The state space consists of the priority of the first 3 customers in the queue as well as the number of free servers.

- Medium: The state space is similar to the high version, other than the number of free servers feature which is replaced with a binary feature indicating whether at least one server is free.
- Low: The state space is similar to the medium version but we aggregate priorities of 1, 2 (low) and priorities of 4, 8 (high).

Task 2: To allow the use of tabular methods, we simplify the task by using 2 rather than 5 product types. The different observability versions of the state space are defined as follows:

- High: The state space consists of the number of products in each buffer as well as the time since last repair/maintenance. For this last feature, we aggregate all states into bins of 10.
- Med: The state space is similar to the high version, however, we aggregate the number of products in each buffer using bins of size 5. The time since last repair/maintenance is aggregated into bins of size 100.
- Low: The number of products in each buffer is aggregated into bins of size 10. The time since last repair/maintenance is aggregated into bins of size 300.

To aggregate a feature x into bins of size K, we transform x using the following function:

$$f(x) = \lfloor x/K \rfloor$$

Task 3: To allow the use of tabular methods, we simplify the task by using an environment of 4×4 pixels (instead of 10×10 pixels as in the original task). The different observability versions of the state space are defined as follows:

- High: The state space consists of the values of all 16 pixels.
- Med: The pixels in positions (0,0), (1,1), (2,2) are unobservable (their value is always 0)
- Low: The pixels in positions (0,0), (1,1), (2,2), (0,3), (1,2), (2,1) are unobservable (their value is always 0)

5.2 Results

Figure 5.1 shows the comparison between all algorithm configurations on each task and level of observability. While the performance of average rewards is consistently higher or equal to discounted returns for task 2 and 3, this is not the case for task 1. These results run counter to the claim made by Singh et al. [9] stating that discounted returns should not be used with POMDPs. When tabular methods are used, these experiments show that neither average rewards or discounted returns consistently outperforms the other and that the best option may be dependent on the task.

The performance difference between average rewards and discounted returns does not seem to increase as observability decreases. In fact, the performance of discounted returns on Task 1 seems to increase as observability decreases. These results run counter to our hypothesis indicating that the introduction of partial observability may be the cause for the superior performance of average rewards on continuing tasks using VFA.

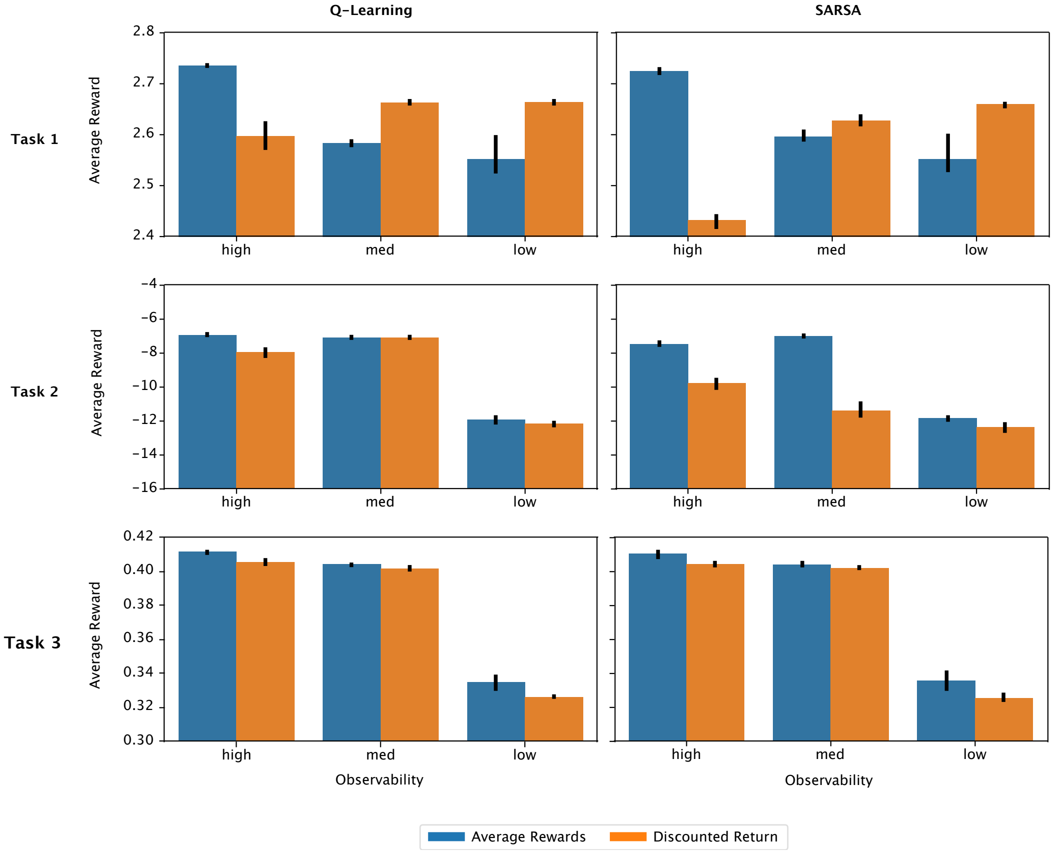

Figure 5.1: Mean greedy performance after training. The error bars represent the 95% confidence interval for the mean

Chapter 6

VFA Complexity: Discounted Returns vs. Average Rewards

6.1 Hypothesis

The experiments presented in the previous chapter seems to indicate that the introduction of partial observability by simple VFA is not the cause for the performance difference between discounted returns and average rewards. In this chapter, we aim to investigate an alternative reason for this performance difference.

Schwartz [8] indicates that a possible advantage of average rewards is linked to the linearity of undiscounted values in unichain domains with constant rewards. We provide an example in Figure 6.1. The example shows that a linear function is sufficient to map states to action-values in the example MDP when using average rewards but not when using discounted returns. This may partly explain why average rewards show a better performance when linear VFA is used and why when using a complex enough VFA both average rewards and discounted returns can achieve similar performance.

Sutton and Barto [11] mention that depending on the task, either action-values or policy gradient policies might be easier to represent using function approximation and consequently obtain a performance advantage. We make a similar hypothesis, stating that average rewards action-values are generally easier to estimate than discounted returns and therefore obtain higher performance when simple VFA is used.

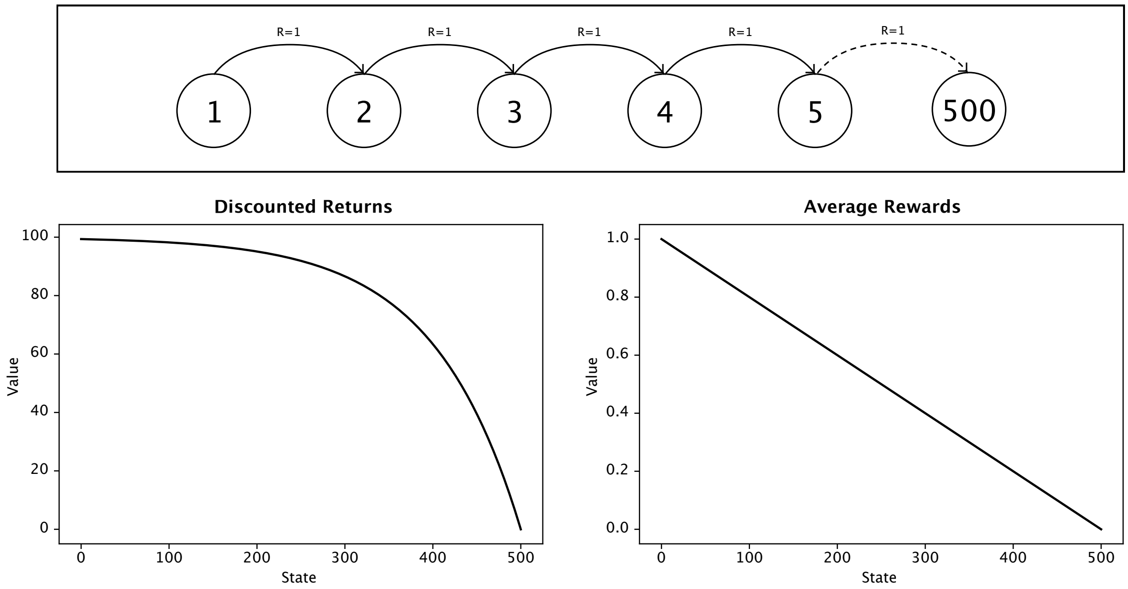

Figure 6.1: (top) Example unichain MDP with constant reward. (bottom-left) Theoretical discounted returns action-values for the example MDP. (bottom-right) Theoretical average rewards action-values for the example MDP.

6.2 Methodology

To investigate this hypothesis, we use supervised learning to learn action-values from tabular methods. First, we find 30 pairs of discounted returns and average rewards tabular runs with similar performance from the random search described in Ch.5. To do so we take the 30 best discounted returns runs and pair each with the average reward run with the closest post-training greedy performance. We then train a linear model (defined by a neural network with no hidden layer) to learn the $state \rightarrow action\text{-}values$ mapping from the tabular policies. Finally, we evaluate the obtained linear policies offline using average rewards (as described in Ch.3). We expect that while the tabular pairs have (by design) similar performance, their resulting linear policies will show a performance advantage in favor of average rewards (due to the action-values being easier to learn).

We chose to perform this experiment on the high observability version of task 3 as it is the task with the biggest state-space for which we have trained tabular policies. Once again we repeat the experiments for both Q-Learning and SARSA to see whether the observed behavior generalizes to both on and off-policy methods. We used 75% of the states as training data and 25% as validation. We trained the linear models using ADAM with early stopping.

6.3 Results

Figure 6.2 shows that while the tabular (discounted returns, average rewards) pair performances are similar (the points approximately follow the line $x = y$), the resulting linear policies generally show an advantage for average rewards. This seems to suggest that average reward action-values are easier to estimate.

For a different task, the discounted returns action-values may be easier to estimate resulting in better performance. The results are, therefore, not fully conclusive as only one task has been tested.

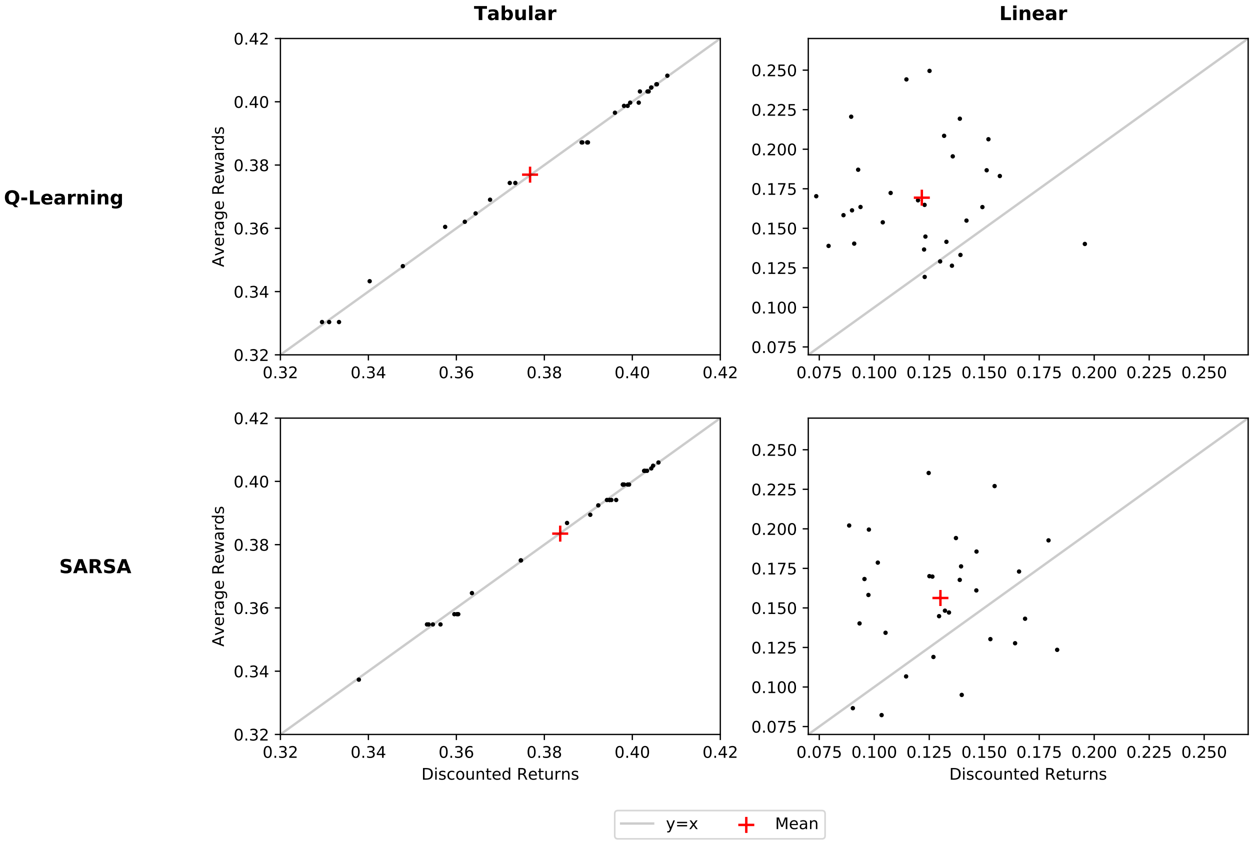

Figure 6.2: Discounted returns and average rewards performance using tabular methods and a linear supervised model.

Chapter 7

Conclusions

This thesis provides an empirical comparison of discounted returns and average rewards on continuing tasks using value function approximation. Additionally, it aims to identify potential causes for the performance difference between these methods.

The empirical comparison indicates that average rewards show superior final performance than discounted returns when linear models are used. Furthermore, it is hypothesized that when VFA is complex enough both methods obtain similar results. The results, however, remain not fully conclusive for several reasons. First, there are exceptions to these observations: in the case of Q-Learning with the linear model in task 2, discounted returns outperform average rewards. Additionally, the parameter search conducted was limited. It is possible that average rewards and discounted returns are sensitive to the parameters related to Asynchronous Methods. Also, Mahadevan [5] shows that average rewards are very sensitive to the exploration strategy. We, however, limited our comparison to ε-greedy exploration. Finally, while the results are relatively consistent across the three tasks, different results could be obtained for different tasks in different domains. This opens different possibilities for future research to extend our work. The results do, however, currently advocate for the use of average rewards in continuing tasks using VFA.

We investigated two possible causes for the performance difference between average rewards and discounted returns. First, we hypothesized that VFA introduced partial observability which in turn causes problems with discounted returns. The experiments presented in Ch. 5, however, do not support this hypothesis. Finally, we hypothesized that average rewards action-values were simpler to estimate and required a simpler network, leading to better performance when simple VFA is used. The experiments presented in Ch. 6 coincide with the hypothesis and open the way for future research.

Bibliography

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A hitchhikers guide to statistical comparisons of reinforcement learning algorithms. 2019.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[5] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22:159–195, 1996.

[6] Sridhar Mahadevan, Nicholas Marchalleck, Tapas K Das, and Abhijit Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Machine learning-international workshop then conference*, pages 202–210, 1997.

[7] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[8] Anton Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the tenth international conference on machine learning*, volume 298, pages 298–305, 1993.

[9] Satinder P Singh, Tommi Jaakkola, and Michael I Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Machine Learning Proceedings 1994*, pages 284–292. 1994.

[10] Ezra Solomon. *The arithmetic of capital budgeting decisions*, volume 29. 1956.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[13] John N Tsitsiklis and Benjamin Van Roy. On average versus discounted reward temporal-difference learning. *Machine Learning*, 49:179–191, 2002.

Appendix A

Task 2 Parameters [6]

- Demand arrival process for each product i is Poisson(λ_i)
- Production time for each product follows distribution Gamma(d_i, λ_i)
- Time between failures has Gamma(k, μ) distribution
- Time required for maintenance has a Uniform(a, b) distribution
- Time for repair has a Gamma(r, δ)distribution

Time Bet. Failures (k, μ)	Repair Time (r, δ)	Maint. Time (a, b)
(6,0.02)	(2,0.04)	(20,40)

Table A.1: Production inventory system parameters [6].

Product	Prod. Time (d, λ)	Demand Arrival (γ)	Buffer Size	Unit Reward
1	(8,8/1)	1/6	30	9
2	(8,8/2)	1/9	20	7
3	(8,8/3)	1/21	15	16
4	(8,8/4)	1/26	15	20
5	(8,8/5)	1/30	10	25

Table A.2: Production inventory product parameters [6].

Appendix B

VFA Experiments: Random Search

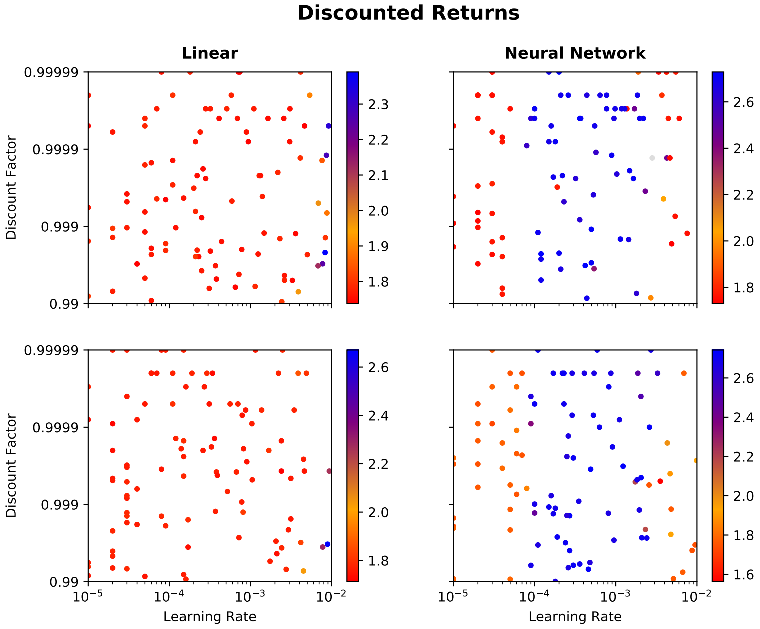

Figure B.1: Random parameter search for Task 1, Discounted Returns

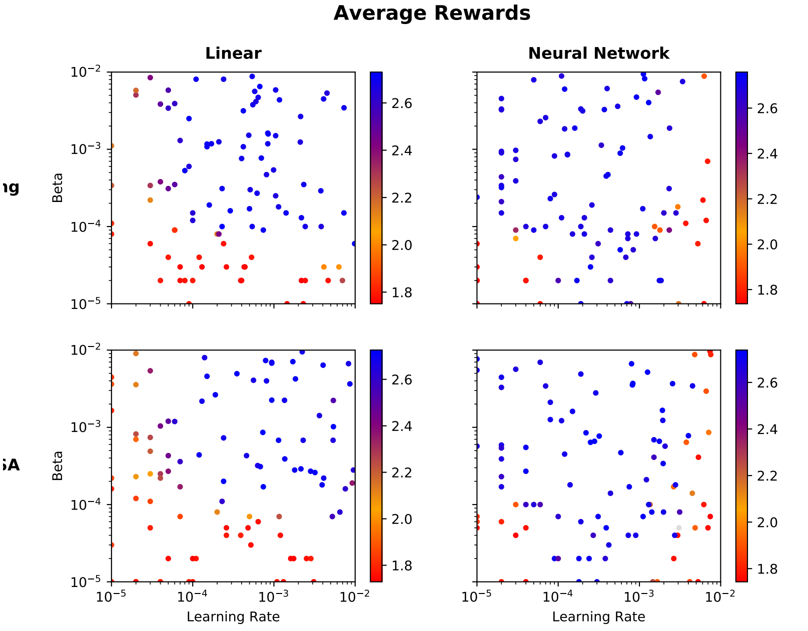

Figure B.2: Random parameter search for Task 1, Average Rewards

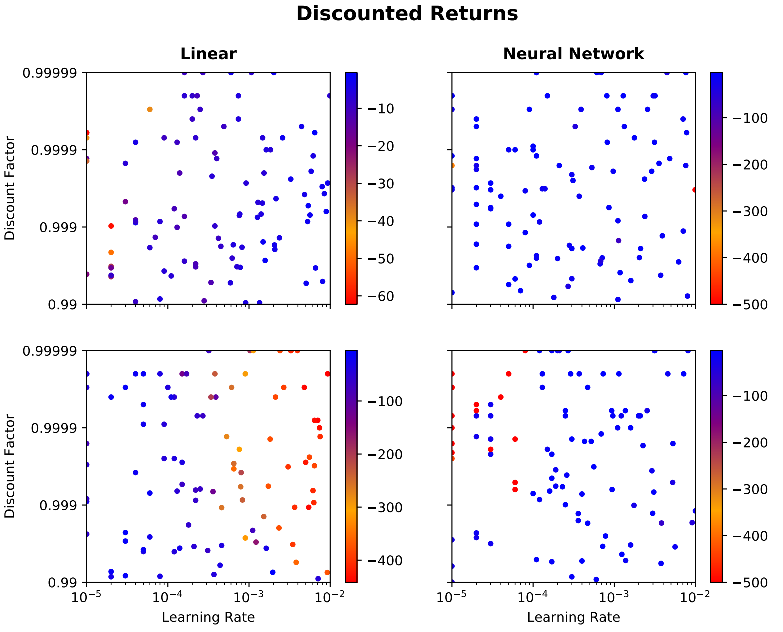

Figure B.3: Random parameter search for Task 2, Discounted Returns

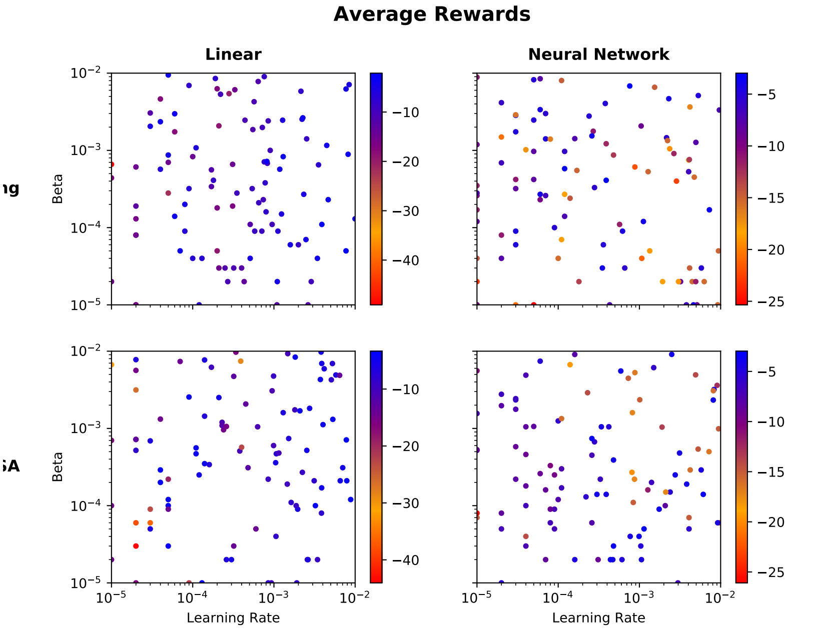

Figure B.4: Random parameter search for Task 2, Average Rewards

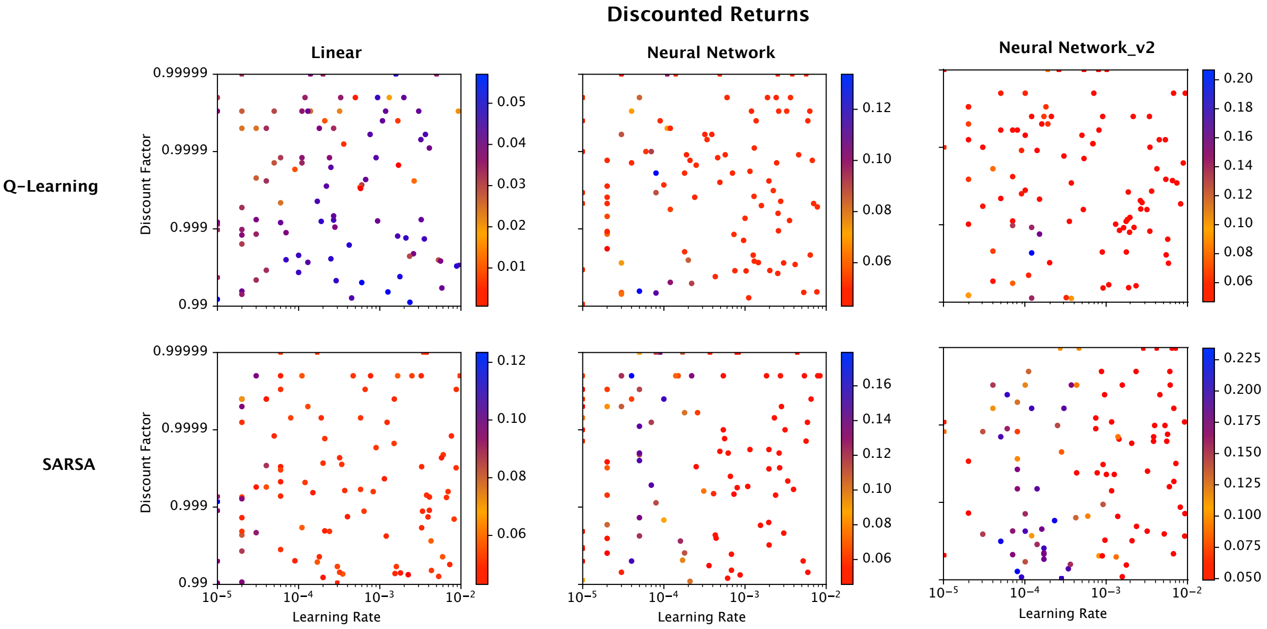

Figure B.5: Random parameter search for Task 3, Discounted Returns

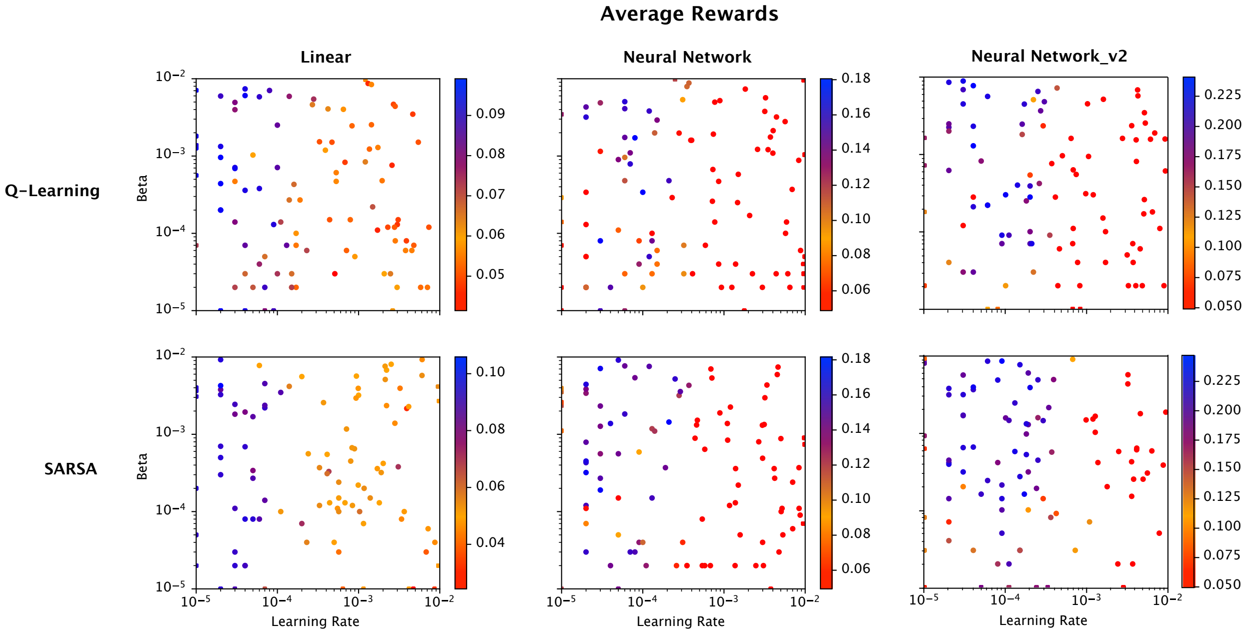

Figure B.6: Random parameter search for Task 3, Average Rewards