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

Panagiotis Kyriakou

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2021

Abstract

Reinforcement learning is a machine learning sub-field, involving an agent performing sequential decision making and learning through trial and error inside a predefined environment. An important design decision for a reinforcement learning algorithm is the return formulation, which formulates the future expected returns that the agent receives after following any action in a specific environment state. In continuing tasks with value function approximation (VFA), average rewards and discounted returns can be used as the return formulation but it is unclear how the two formulations compare empirically. This dissertation aims at empirically comparing the two return formulations. We experiment with three continuing tasks of varying complexity, three learning algorithms and four different VFA methods. We conduct three experiments investigating the average performance over multiple hyperparameters, the performance with near-optimal hyperparameters and the hyperparameter sensitivity of each return formulation. Our results show that there is an apparent performance advantage in favour of the average rewards formulation because it is less sensitive to hyperparameters. Once hyperparameters are optimized, the two formulations seem to perform similarly.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Panagiotis Kyriakou*)

Acknowledgements

I would like to thank my supervisor Stefano Albrecht as well as my co-supervisors Josiah Hanna and Lukas Schäfer for their guidance and feedback over the course of the project proposal and the dissertation.

Table of Contents

1 Introduction **1**

2 Background **4**

2.1 Reinforcement Learning 4

2.2 Markov Decision Processes 4

2.3 Continuing and Episodic Tasks 5

2.4 Average Rewards and Discounted Returns 6

2.5 Learning Algorithms 7

2.5.1 Q-Learning 8

2.5.2 SARSA 8

2.5.3 Overestimation Bias and Double Q-Learning 9

2.6 Value Function Approximation 9

2.6.1 Linear VFA 10

2.6.2 Polynomial Basis Function VFA 10

2.6.3 Deep VFA 11

2.7 Related Work 11

3 Methodology **14**

3.1 Learning Algorithms 14

3.2 VFA Methods 15

3.3 Tasks 16

3.3.1 Task 1: Access-Control Queuing 16

3.3.2 Task 2: Factory Production Simulation 16

3.3.3 Task 3: Catching Falling Objects 18

3.4 Hyperparameter Selection 19

3.5 Evaluation 19

3.6 Comparison 20

3.7 Implementation Details 21

4 Empirical Comparison Results **23**

4.1 Best 30% of Hyperparameters Results 23

4.2 Best Hyperparameter Results 29

4.3 Hyperparameter Sensitivity 34

5 Conclusions **38**

Bibliography **40**

A Appendix A: Learning Algorithms Pseudocode **44**

B Appendix B: Hyperparameter Samples **46**

Chapter 1

Introduction

Reinforcement learning is a machine learning sub-field that involves sequential decision making inside a predefined environment. In reinforcement learning, an agent is trained through trial and error to perform decisions in order to solve a particular task. The agent receives a reward signal as feedback based on its chosen action and the environment states the agent is in. The agent's goal is to maximise that reward signal over time. Reinforcement learning has had some impressive accomplishments [1, 2, 3], which is why it has grew in popularity over the past decade [4].

In many reinforcement learning methods, in order for the agent to make informed decisions at each state of the environment, the agent needs to predict the return that it will receive following a policy from that state. The way that the future expected returns are formulated is referred to as the return formulation and it is a very important design decision when creating a reinforcement learning algorithm. That is because it can affect the performance of the agent on the reinforcement learning task greatly. Discounted returns are the most common return formulation, which weight the future rewards based on their immediacy (short term rewards are more valued that long term rewards). Average rewards is another return formulation, which does not differ the weight between short term and long term rewards, regarding them both as equally important.

In reinforcement learning, the environment complexity can vary, with more complex environments having more states the agent can be in or more actions the agent can perform. In many traditional methods, the expected return of each action from each state is stored in a matrix with the action space and the state space being the matrix dimensions. Unfortunately, when an environment is complex and its state space or action space are large, it is impractical to use a matrix. That is, because it will need

a lot of time to be fully updated with accurate estimates of the expected returns and because it might not be able to fit in the computer's memory. For these reasons, value function approximation is used. Value function approximation involves the use of a parameterised function of environment features, whose parameters are learned during training.

At the time of writing, there has been limited work investigating the empirical performance difference between the two return formulations mentioned above. Theoretical work by Tsitsiklis and Van Roy [5] and Sutton and Barto [6] suggests that the two return formulations should perform similarly, but neither work has been supported by empirical results. On the other hand, empirical work performed by Mahadevan [7] and Schwartz [8], suggests that average rewards outperform discounted returns in terms of average rewards acquired at convergence and speed of convergence. In both of these cases though, simple environments were used that did not require the use of value function approximation. In 2019, an empirical comparison between the two return formulations with VFA was performed by Descause [9]. In that comparison it was shown that average rewards outperform discounted returns and it was hypothesised that that might have been because the action-values were easier to estimate when average rewards were used. Unfortunately, limited evidence was provided to support that claim.

This work is a continuation of the work done by Descause [9]. As discussed in our IPP [10], we aim to answer the following research question: How do average rewards and discounted returns compare empirically and why are empirical performance differences seen in practice ?

Given the results of Mahadevan [7], Schwartz [8], and Descause [9], we initially hypothesise that average rewards will indeed perform better than discounted returns in terms of average rewards acquired at convergence and in terms of the speed of convergence. We also hypothesize that the difference in performance is caused by the action-value estimation difficulty being different across the return formulations, as discussed by Descause [9]. This difficulty difference means that we expect the performance difference to get smaller as the VFA complexity is increased. In order to assess our hypotheses, we experiment with four different VFA methods of varying complexity and with three different learning algorithms on three different tasks of varying complexity. We investigate the performance of each formulation in terms of the average rewards that we acquired at the time the VFA had converged and in terms of how many steps were needed for the VFA to converge. This was performed by

looking at the average training curve of the best performing 30% of the hyperparameter combinations sampled. We also consider the performance of each formulation with near optimal hyperparameters by looking at the training curve of the single best performing hyperparameter combination. The results from these two experiments do not support our initial hypotheses and thus we form a new hypothesis relating the differences observed in performance of the two formulations to hyperparameter sensitivity. We further validated this new hypothesis by looking at the distributions of the average rewards acquired at convergence for both formulations and observing that the distributions for the average reward formulation were more narrow and centred around higher values than the distributions for the discounted return formulation.

This report is divided into chapters. In Chapter 2, we give an overview and explain all background concepts that are used in this work. In Chapter 3, we provide a description of the methods used for the the experiments and their implementations. In Chapter 4, we present and discuss the the results from all experiment setups. Finally, in Chapter 5, we summarize our findings and give some conclusive remarks.

Chapter 2

Background

2.1 Reinforcement Learning

The field of machine learning is commonly segmented into three general categories that describe the way that machine learning models learn to solve a task. These three categories are supervised learning, unsupervised learning and reinforcement learning. The first two categories learn using collected and pre-processed data that is available before the model starts being trained. Reinforcement learning (RL) involves an agent solving a task through sequential decision making inside a predefined environment. As it can be understood from the name, the desired behaviour of the agent is learned through reinforcement (rewarding desired behaviour and penalising undesired behaviour), similar to the way that humans learn to behave. In the past decade, reinforcement learning has grown in popularity [4] with very impressive achievements such as playing rule based games with superhuman performance [1], learning how to walk without human input [2] or learning how to manipulate a robotic arm to solve a Rubik's cube [3]. A more formal definition of RL problems and their components can be seen in Section 2.2.

2.2 Markov Decision Processes

In RL, problems are defined using Markov Decision Processes (MDPs) [11]. MDPs are described by Sutton and Barto [6] as a formalisation of sequential decision making in which actions not only affect immediate states and immediate rewards but also future states and future rewards. An MDP is defined through its four components, the state space S, the action space A, the environment dynamics, and the reward space

R. In MDPs, the entity which does the learning and is making decisions is called the agent. At each time-step t, the agent interacts with the environment, which includes everything that is not the agent [6]. This interaction comprises of the agent being in a state $S_t \in S$ taking an action $A_t \in A$ and the environment responding by giving the agent a new state to be in $S_{t+1} \in S$ and a reward $R_t \in R$. The state S_{t+1} and the rewards R_t returned to the agent at each time-step t are decided by the environment dynamics which are defined as a conditional probability:

$$P(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

Additionally, the environment dynamics include the transition function defined as:

$$P(s'|s, a) \doteq Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in R} P(s', r|s, a)$$

as well as, the expected rewards function defined as:

$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} P(s', r|s, a)$$

An MDP is controlled through a policy π. A policy is a mapping between each state and the probabilities of selecting each action from that state [6]. A policy can be understood informally as the agent behaviour. As described in Sutton and Barto [6], solving an RL task means finding a policy that maximises the reward acquired by the agent.

2.3 Continuing and Episodic Tasks

RL problems can widely vary and for that reason they are divided into two main categories, describing the task's temporal characteristics: continuing tasks and episodic tasks. Episodic tasks are the most common type of tasks in reinforcement learning and are characterised by the notion of a final time-step T or a final environment state S_T which practically divides the training time into episodes. An example of such a task is the game of chess with the final state being a checkmate or a stalemate position. Continuing tasks on the other hand don't have the notion of a final time-step or environment state. In such tasks, as the name suggests, the agent interacts with the environment continuously and indefinitely ($T = \infty$). An example of such a task is stock trading, which is performed indefinitely with no moment at which the trading stops.

2.4 Average Rewards and Discounted Returns

As we described earlier, in reinforcement learning the agent learns through reinforcement. Informally, as described by Sutton and Barto [6], the agent's goal is to maximise the total amount of reward it receives from the environment through their interaction. Thus, at every time-step, the agent goal is to choose the action which maximises the expected return G_t. For that reason, the way that the return is formulated is a very important part in solving an RL problem.

A naive approach to return formulations would be to sum all the expected future rewards as shown below.

$$G_t \doteq R_{t+1} + R_{t+2} + ... + R_T$$

Such formulation would not be practical because those future rewards would grow infinitely if the task is continuing or cyclical and there is no final state. This is impractical because, based on the environments reward function, G_t can be equal to ∞, $-\infty$ or 0 for every t. For that reason, in most reinforcement learning environments, discounted returns are used as the return formulation. This return formulation introduces the concept of discounting, which is weighting future rewards based on their immediacy. That means that immediate rewards given to the agent are more important (have higher weight - lower discount factor) than potential future rewards. The discounted return is defined in the equation below.

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

where $0 < \gamma < 1$ is the discount factor.

Another return formulation that exists for such continuing and cyclical tasks is the average rewards formulation. In such formulation, the incremental average of the rewards while following a policy, is used to evaluate the policy. The average reward formulation is defined in the equation below:

$$G_t \doteq \sum_{k=0}^{\infty} R_{t+k+1} - r(\pi)$$

where $r(\pi)$ is the incremental average reward obtained by following policy π. This is updated every time-step using the update rule below:

$$r(\pi) = r(\pi) + \beta * \delta$$

with β is a weight variable that inflates the values of recent returns in the average and δ being the error term from the action-value function update (further explained in Section 2.5).

In some tasks, the use of discounting is crucial for the agent to be able to solve a task successfully. An example of such task would be stock investing and stock trading. In investing, immediate monetary gains are seen as more valuable than future monetary gains because the money earned can be re-invested in the market which may lead to even more gains overall.

In other tasks though, prioritising the immediacy of the reward is not important for solving the task, but may even actually hinder the agent's performance. In such tasks, average rewards are more commonly used. An example of such a task is Tetris [12]. In Tetris, the agent is tasked to form horizontal lines by placing differently shaped blocks inside a rectangular space. When horizontal lines are formed the blocks that made those lines disappear and the score increases. The agent is able to acquire short term mediocre reward by forming a complete line as fast as possible. But, the agent can also acquire the long-term high reward by waiting and forming multiple complete lines at the same time (known as a Tetris).

2.5 Learning Algorithms

As mentioned previously, the policy that achieves maximal return is learned through training. During training, two functions are learned when using some RL learning algorithms, the state-value function and the action-value function. The former, maps each state to the expected return following policy π from that state and is defined below:

$$V(S_t) := \mathbb{E}[G_t|S_t]$$

The latter function, maps the expected return of each action at each state in the environment to the respective action and state. This is defined formally below:

$$Q(s_t, a_t) := \mathbb{E}[G_t|s_t, a_t]$$

Where G_t, is the expected return, a_t is the action evaluated at time-step t and s_t is the environment state evaluated at time-step t. During training, the value of $Q(S_t, A_t)$ is learned for each action and state, and thus by the end of training, it is hoped that the agent can choose the best action for every state. Below we go over three learning algorithms used to estimate the action-value function $Q(S_t, A_t)$. For all algorithms, we

consider an epsilon-greedy policy. With such policy, the agent chooses its next action uniformly at random with a probability ε and greedily (chooses the action with the highest action-value function) with a probability $1-\varepsilon$.

2.5.1 Q-Learning

Q-Learning [13] falls into the category of off-policy learning algorithms. This means that the optimal action-value function is directly approximated independently of the agent's policy [6]. In the case of Q-Learning, the next action considered in the update rule is only chosen greedily. The update equation of Q-Learning using the two different return formulations can be seen below:

Discounted Returns:

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

where α represents the learning rate and γ the discount factor.

Average Rewards:

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

where α again represents the learning rate and δ_t represents the Temporal Difference error between the previous estimate of the action-value function $(Q(S_t,A_t))$ and the current, better estimation of the action-value function $(R_{t+1} + \gamma * max_a Q(S_{t+1},a)$ or $(R_{t+1} - r(\pi)) + max_a Q(S_{t+1},a))$.

The Q-Learning algorithm pseudocode can be found in Appendix A.

2.5.2 SARSA

Contrary to Q-Learning, SARSA [13] is an on-policy learning algorithm. This means that the next action in its update rule is chosen according the agent's epsilon-greedy policy. The update rules of SARSA using the two different return formulations can be seen below.

Discounted Returns:

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

Average Rewards:

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

The SARSA algorithm pseudocode can be found in Appendix A.

2.5.3 Overestimation Bias and Double Q-Learning

It has been argued by Van Hasselt et al. [14], Van Hasselt [15] and Dankwa and Zheng [16], that when Q-Learning is used with discounted returns, the agent overestimates the value function which leads to instability, reduction in the speed of convergence and sometimes reduction in final performance. This is often referred to as "Overestimation Bias" and it was first addressed by Van Hasselt [15], by the introduction of Double Q-Learning.

Double Q-Learning, similarly to Q-Learning, is an off-policy algorithm which uses two different action-value functions Q_A and Q_B during training. In practice, one action-value function is used to update the other action-value function. Which of the two action-value functions is updated at the current time-step is chosen uniformly at random. The update rule (when updating Q_A) for Double Q-Learning can be seen below:

Discounted Returns:

$$A' \leftarrow max_a Q_A(S_{t+1},a)$$

$$Q_A(S_t,A_t) \leftarrow Q_A(S_t,A_t) + \alpha \underbrace{[R_{t+1} + \gamma * Q_B(S_{t+1},A') - Q_A(S_t,A_t)]}_{\delta_t}$$

Average Rewards:

$$A' \leftarrow max_a Q_A(S_{t+1},a)$$

$$Q_A(S_t,A_t) \leftarrow Q_A(S_t,A_t) + \alpha \underbrace{[(R_{t+1} - r(\pi)) + Q_B(S_{t+1},A') - Q_A(S_t,A_t)]}_{\delta_t}$$

The Double Q-Learning algorithm pseudocode can be seen in Appendix A.

2.6 Value Function Approximation

As it was previously discussed, reinforcement learning problems are formulated using MDPs. MDPs are partly defined by the states S and the actions A that can be taken from each state.

The expected return G_t of each action in each state is given by the action-value function $Q(S_t, A_t)$. Traditionally, the action-value function is represented by a table or matrix (referred to as the Q-table) whose dimensions are the environment's states and the environment's actions. Methods that use such data structure for the action-value function are called Tabular RL methods. In complex environments though, such as chess whose state space complexity is 10^{42} [17], the state space and the action space can be so large, that tabular RL methods are impractical. That is because of two main reasons. Firstly, finding a policy that solves the task might require the agent to visit all environment states and perform all actions multiple times in order to fully update the Q-table. If the state and action space is too large, then that would take too much time and thus the learning algorithm would not be able to converge within a reasonable amount of time. Secondly, if the environment is too complex then the look-up table will not be able to fit into the computers memory.

For these reasons, Value Function Approximation (VFA) is used instead of a look-up Q-table. In VFA, a parameterised function of different environment features is used to approximate the Q-table. During training, the parameters of this function are estimated in order for the VFA to approximate the real value function. In the subsections below, we explain the different VFA methods used in this report.

2.6.1 Linear VFA

The simplest, both conceptually and mathematically, VFA method that is seen in this report's experiments is linear VFA. In linear VFA, the approximate value function is a linear function of the environment state feature vector $x(s) = (x_1(s), x_2(s), ..., x_d(s))$. Linear VFA approximates the value function by taking the inner product of the state's features vector $x(s)$ and a weight vector $\mathbf{w}$, which is the same size as the features vector. Linear VFA's formula can be seen below:

$$\hat{v}(s,w) \doteq w^T x(s) \doteq \sum_{i=1}^{d} w_i x_i(s)$$

Where $\hat{v}(s,w)$ is the approximate state-value function, w is the weight vector, $x(s)$ is the state's feature vector and d is the length of the weight and feature vector.

2.6.2 Polynomial Basis Function VFA

Polynomial Basis Function VFA is not a completely different VFA method but rather a method to increase the complexity of the functions that can be approximated by

linear VFA. This is done by expanding the state's feature vector through raising the features to a degree as well as taking the product of the combination of features. For example, if the state's s' feature space consists of two features a and b, the original feature vector will be $x(s') = [a, b]$ while the expanded feature vector when 2nd degree polynomial basis functions are used will be $x(s') = [a, b, a^2, b^2, ab]$. By the same logic, when 3rd degree polynomial basis functions are used, the feature space will be $x(s') = [a, b, a^2, b^2, ab, a^2b, ab^2, a^3, b^3]$.

2.6.3 Deep VFA

Deep learning VFA refers to the idea of using an artificial neural network (NN) to approximate the value function. This idea was popularised by Mnih et al. [18] with the introduction of Deep Q-Networks and the achievement of playing Atari games with human level performance. NNs can be thought of as a network of computational units that are arranged in layers. These computational units can perform linear computations and non-linear computations through the use of activation functions. The connections between the units have weights, similar to the ones in linear VFA, which are tuned through training to best approximate the value function. Each layer in an NN aims at automatically extracting and weighting different features from the original feature vector input or from the output of the layer previous to it. The first and simplest type of neural network is the feedforward neural network, whose names refers to the computations going "forward" from layer to layer [19].

VFA methods, like the ones explained above, solve the problems caused by complex environment as they do not have such large memory requirements. They also do not require the agent to visit all states because a single update to the functions parameters affects the function's output for the whole state space.

2.7 Related Work

The literature that has been produced which compares the empirical performance of average rewards and discounted returns is limited. Theoretical work on the topic was performed by Tsitsiklis and Van Roy [5], in which it was argued that the performance of the two return formulations should be similar. The arguments presented in paper though, apply to policy evaluation and not policy improvement and thus do not guar-

antee the same results in our experimental setup nor with the rest of the empirical work done. Also, the arguments in the paper are purely theoretical and there are no empirical results to support them. A similar argument is done by Sutton and Barto [6], who argue that in theory the two return formulations should not have differences in performance, as optimizing discounted returns over the steady state distribution is similar to optimizing average rewards. That is, because the two return formulations are inversely proportional to each other by a factor of $1-\gamma$ as the average of the discounted returns is always $r(\pi)/1-\gamma$.

The empirical work that is done on the topic argues that the two return formulations perform differently, contradicting previous theoretical work. An empirical comparison was made by Schwartz [8], in their 1993 paper which introduces R-Learning, a Q-Learning alternative using average rewards [8]. In the paper, the authors find a performance advantage for average rewards but the paper only involves simple tasks that do not require VFA. These results were validated by Mahadevan [7], who conducted more experiments with R-Learning on simple tasks with no VFA. Again it was concluded that with the appropriate tuning, R-Learning was performing better than Q-Learning in terms of average reward acquired as well as time needed for convergence. Additionally, in his MSc thesis, Descause [9] studied the empirical comparison between the two return formulations in continuing tasks with VFA and further validated that there was a performance difference between the two formulations. The author experimented with two different learning algorithms (Q-Learning and SARSA) and two different VFA methods (linear and deep learning) on three tasks. A consistent performance advantage was observed for average rewards again both in terms of average rewards acquired when the VFA had converged and the time needed for the VFA to converge. The performance difference between the formulations was getting smaller as the VFA complexity increased. It was hypothesised that the performance difference was caused by the action-values being easier to estimate when using average rewards and even though some empirical evidence supported the hypothesis it was not complete.

In general, there has been an increase in the number of papers studying average rewards and introducing average reward based learning algorithms. In their recent review, Dewanto et al. [20] go over a variety of model-free learning algorithms as well as tasks in continuing environments. Additionally, two papers [21, 22] were presented in 2021 International Conference on Machine Learning, again introducing learning algorithms that optimised average rewards rather than discounted returns.

As mentioned in Chapter 1, this work is an extension of the work by Descause [9].

We extend the work by experimenting with Double Q-Learning, in order to eliminate the effect that overestimation bias might have on the results. We also extend the work by experimenting with two additional VFA methods, 2nd and 3rd degree polynomial basis VFA, which should additionally test whether the performance difference between the formulation is getting smaller as VFA complexity increases. Moreover, we train all algorithms for more time-steps than what was done by Descause [9], in order to ensure all VFA methods convergence in all experiment configurations. Finally, we investigate the hyperparameter sensitivity as a factor of the formulations' performance.

Chapter 3

Methodology

3.1 Learning Algorithms

We conduct experiments with the three algorithms that were introduced in Chapter 2, namely, Q-Learning, SARSA and Double Q-Learning. Q-Learning and SARSA were chosen for the sake of completeness, given that they are off-policy and on-policy algorithms. As explained in earlier sections, Double Q-Learning was chosen to alleviate instabilities and performance drops caused by overestimation bias. Because overestimation bias is something caused by the combination of Q-Learning and one of the return formulations we are comparing, it was important to eliminate any of its effects when comparing the two return formulations. This should provide more concrete results and make drawing conclusion from results easier.

As done by Descause [9], all the parameters of the three algorithms were reused from the paper by Mnih et al. [13] except for the learning rate α, the discount factor γ and the weight factor β. The methodology for sampling these hyperparameters for the comparison is explained in Section 3.4. Asynchronous methods with multiple parallel agents were used for training for all algorithms. Such methods allow the simultaneous training of multiple parallel agents across multiple environment instances while using and updating the same value function [13]. Asynchronous methods were used in order to prevent correlated updates to the value function. Correlated updates occur when the environment feature vector between multiple VFA updates is very similar, which can lead to overfitting [13, 18]. This is very common as consecutive states have similar feature vectors. Correlated updates slow down or even prevent learning [13]. Thus, asynchronous methods enabled learning and reduced convergence time. For all algorithms, 4 parallel agents were trained for 6 million steps across all of them, as

this was enough to assess performance at convergence. As done by Descause [9], we used the same exploration method that was used by the original asynchronous methods paper [13]: each parallel agent's ε parameter was annealed to either 0.1, 0.001 or 0.5 with probabilities 0.4, 0.3 and 0.3 respectively. In this work though, the use of random seeds, kept the value that each parallel agent's ε was annealed to consistent across experiments with the different return formulations. This ensured that no performance advantage is given to either formulation based on the values that ε was annealed to. Also, the epsilon decay is performed linearly over the last 80% of the training steps.

3.2 VFA Methods

For the return formulation comparison, four different VFA methods were experimented with. These include linear VFA, linear VFA with 2nd degree polynomial basis functions, linear VFA with 3rd degree polynomial basis functions and deep learning VFA. Similarly to what was done by Descause [9], linear VFA was modeled as a feed-forward neural network consisting of no hidden layers. The deep learning VFA was modelled as a feed-forward neural network with 2 ReLU [23] activated, hidden layers whose architectures changed with every task in order to scale to its complexity. This was done by using $(size_{input} - size_{output})/2$ neurons in each of the 2 hidden layers.

For task 3, a Convolutional Neural Network [24] architecture was used for the deep learning VFA, similar to the one used by Descause [9]. The architecture involved parallel processing with two parallel, ReLU activated, convolution layers. The first convolution layer has a 3x3 mask and is followed by a Max Pooling layer [25] with a 2x2 mask, another convolution layer with a 3x3 mask and another Max Pooling layer with a 2x2 mask. The second convolution layer has a 5x5 mask and is followed by just one Max Pooling layer with a 2x2 mask. The output of the two parallel convolutional layers was concatenated and passed through a ReLU activated, feed-forward neural network with 50 neurons.

In order to use polynomial basis functions, the state's feature vector was expanded based on the degree of the basis functions. Additionally, the neurons of the linear model input layer were increased to equal the size of the expanded state feature vector.

All VFA models were used with all learning algorithms.

3.3 Tasks

Three tasks of varying complexity were used. These tasks are the same tasks that were used by Descause [9]. We chose to experiment with the same tasks because it would aid in any result comparison between this work and the work of Descause [9] as it would remove one factor of variability if different conclusions are drawn. All tasks are continuing tasks with large enough state and action spaces that warranted the use of VFA. More details on each task are given in the following subsections.

3.3.1 Task 1: Access-Control Queuing

The first task is an access-control queuing task originally proposed by Sutton and Barto [6]. The task consists of a queue in which customers of different priorities wait before getting assigned to one of ten servers. Each customer has priority 1, 2, 4, or 8 with all being equally likely. The reinforcement learning agent has to decide whether or not he will assign the customer at the front of the queue in a server. The reward that the agent receives when a customer is successfully assigned to a server is equal to the customer's priority. The agent receives a reward of -1 if a customer is accepted when all servers are full. At each time-step, a server becomes free with a probability of 0.06.

The state feature vector is an 11 element vector containing the priorities of the first 10 elements of the queue and the number of available servers. Given 4 possible priorities per customer and 10 servers in total, the state space consists of around $1.048 * 10^7$ different states. Two actions are available to the agent at every time-step. The total number of action-values is thus $2.096 * 10^7$. A visual representation of the environment of task 1 can be seen in Figure 3.1.

3.3.2 Task 2: Factory Production Simulation

The second task is a factory production simulation task originally proposed by Mahadevan et al. [26]. The task involves a factory machine that can produce either one of five products. Each product can be assigned to a buffer whose size is conditioned on which of the five products it stores (30, 20, 15, 15 and 10). Each product has different demand frequency and a different price (9, 7, 16, 20 or 25). Production failures can occur which force a repair and halt production until repair is finished. Production failures are avoided by periodic maintenance. During maintenance, no production can be performed but that is for a shorter period than a repair. When no maintenance, no

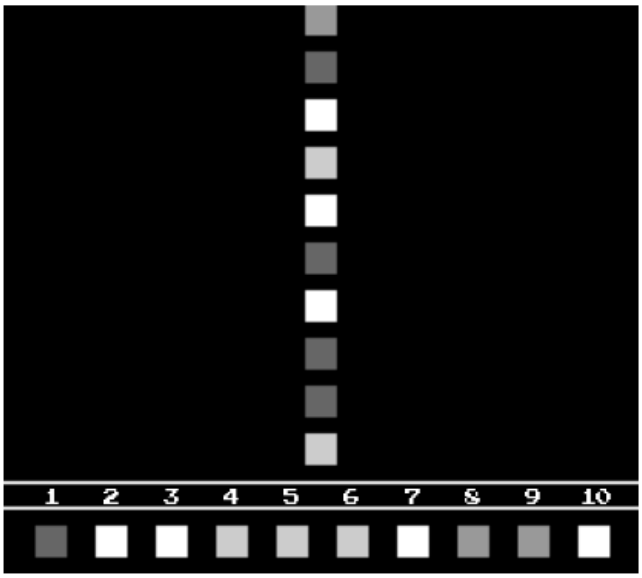

Figure 3.1: A visualisation of task 1, with the first 10 customers in the queue shown vertically and the servers shown horizontally. Each customer has a different colour representing their priority, with lighter colours indicating higher priority.

production or no repair is performed, the agent must decide which of the five products to produce or whether to start performing maintenance.

If production of either one of the five products is chosen, the production cannot stop unless there is a production failure or the product's buffer is full. The agent is given reward equal to a products price if demand for that product arrives and if there is any of that product stored in its buffer. The agent is given reward of -5000 if a production failure occurs and is also given a reward of -500 if maintenance is performed.

The state feature vector is of size 6 and consists of the number of products in each of the 5 product buffers and the time until the last repair or maintenance. Because, the last feature is theoretically not bounded, we cant compute the size of the state space, as it would be infinite. Assuming the last feature is on average in the range of [0, 1000], the state space would be equal to $1.35 * 10^9$. The action space consists of 6 actions. Thus the total number of action-values would be $8.1 * 10^9$. A visualisation of the environment's state representation can be seen in Figure 3.2.

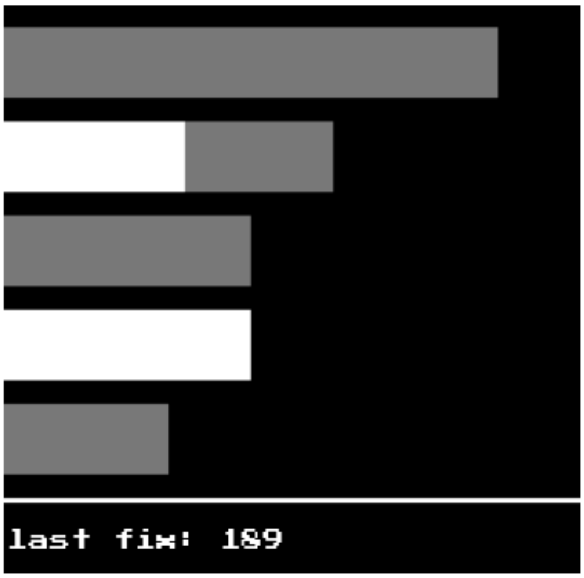

Figure 3.2: A visualisation of task 2. The buffers and the amount of products they have stored in them can be seen stacked vertically, while the number of days before a maintenance or repair can be seen at the bottom.

3.3.3 Task 3: Catching Falling Objects

The third task is a game where the agent has to catch as many falling objects as possible and was introduced by Descause [9]. The task's environment is a 10x10 grid where objects fall vertically. The agent stays in the bottom row and can move only horizontally in that row. Objects are placed in any row with 0.5 probability and are uniformly placed across columns. At each time-step, the objects fall by one row, thus moving towards the agent. If the agent is at the same column as an object, when the object has reached the final row, a reward of 1 is acquired by the agent. The state's feature vector is the flattened 10x10 grid, thus a vector of length 100. Objects are represented by the value of 255 on the feature vector. The first 9 rows can have 11 different configurations each, as for each row an object can be present in one of the 10 columns or be absent completely. The last row can have 100 configurations as there are 10 different positions for the agent (1 for each column) and objects may appear in 10 different positions along the row. Taking all this into account, the state space consists of $2.35 * 10^{11}$ states. Given that three actions are available to the agent (moving left, moving right or not moving) at each time-step, the total number of action-values is $7.07 * 10^{11}$. A visualisation of the environment's state representation can be seen in Figure 3.3.

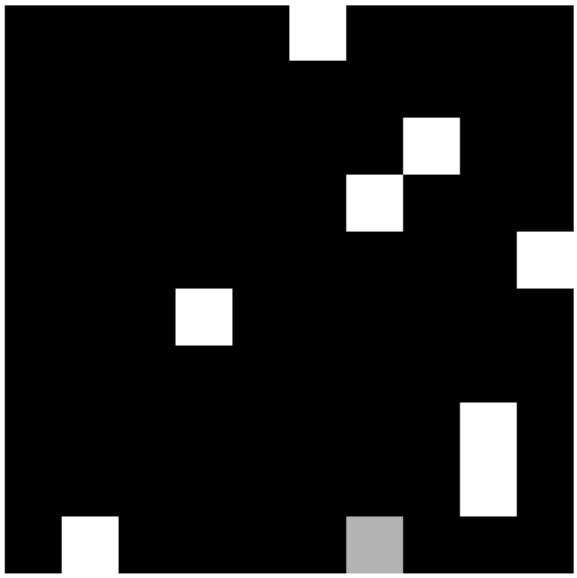

Figure 3.3: A visualisation of task 3. The white blocks represent falling objects while the grey block represents the agent.

3.4 Hyperparameter Selection

Hyperparameter selection was performed by running multiple runs of each learning algorithm, VFA and task combination, each of which had randomly sampled parameters. For task 1 and task 2, 100 sample runs were executed while for task 3, 50 sample runs were executed. This hyperparameter search was done in order to ensure that all methods tested and ultimately compared would be optimised to a similar extent. As discussed previously, the hyperparameter that were tuned were the ones relating to the update rules of the learning algorithms, namely the learning rate α, the weight factor β, and the discount factor γ. The two former parameters were sampled from a log-uniform distribution with 10^{-5} as the lower bound and 1 as the upper bound. The latter hyperparameters was sampled again from a log-uniform distribution but with 0 as the lower bound and $1 - 10^{-5}$ as the upper bound. The results of the random hyperparameter sampling are visualized using scatter-plots in Appendix B.

3.5 Evaluation

In order to compare the performance of each formulation, we had to compare the agents on the greedy policy that they have learned without random exploratory actions. In

order to do this, every $200,000$ steps for task 1 and 2 and every $1,000,000$ steps for task 3, the VFA parameters were saved and 5 runs, of 25,000 steps each, were run using the saved parameters. The final performance result would be the average of these 5 runs. In order to maintain consistency between the evaluation runs for each return formulation, 5 random seeds were used for the environments random events in these evaluation runs.

3.6 Comparison

The comparison between the two formulations is done both qualitatively and quantitatively.

As mentioned in Chapter 1 and in our project proposal **??**, our hypothesis is that average rewards will perform better at convergence, in terms of average rewards acquired, and will converge faster than discounted returns. We also hypothesize that as the complexity of the VFA increases, the performance difference between the formulations should decrease. In order to test these hypotheses we will compare the performance of the two return formulations from the best performing 30% of the hyperparameter combinations found for each experiment configuration (combination of algorithm and VFA method). This means that we will report the performance of the top 30 hyperparameter combinations for task 1 and task 2 and of the top 15 hyperparameter combinations for task 3. This experiment setup allows the investigation of the performance variability across multiple hyperparameters in terms of both the speed of convergence and average reward acquired at convergence. This setup also allows the validation of the results by Descause [9] as a similar setup was used in that work.

In order to further validate any performance differences that were seen from the results of the first experimental setup and verify that such performance difference can be seen when the optimal hyperparameters for each experiment configuration are chosen, we also compare the performance from the single best performing hyperparameter combinations for each experiment configuration. In order to eliminate the variability in performance that can occur across runs, we run each experiment configuration with its optimal hyperparameters 10 times with 10 different random seeds. The use of random seeds ensures that random environment events stay consistent when the different return formulations are tested in each experiment configuration.

For both of the last two experiment setups, training curves will be created that show the average performance, as well as highlight the 95% confidence interval. This will

ensure that both the average rewards acquired at convergence and speed of convergence can be assessed. Additionally, it will show the training stability of each experiment configuration which will allow investigating the advantages of Double Q-Learning over regular Q-Learning.

Lastly, we will analyze the hyperparameter sensitivity of each return formulation. This is done by looking at the distribution of average rewards acquired at VFA convergence through histograms.

In terms of quantitative evaluation, we use the Kolmogorov-Smirnov test [27] to assess the following null hypothesis: the final performance of the two return formulations comes from the same distribution. The Kolmogorov-Smirnov test was chosen because it's a statistical test for continuous distributions and because it does not make any prior assumptions about the distributions [27]. We use the Kolmogorov-Smirnov test on the results of the best performing 30% of the hyperparameters and the single best performing hyperparameters.

3.7 Implementation Details

All implementations of the methods that are discussed in this chapter were made using Python. Libraries such as Numpy [28] and MatPlotLib [29] were used for the creation and visualisation of the environments as well as the learning algorithms. For all the VFA methods, the PyTorch [30] machine learning framework and Scikit-Learn's [31] preprocessing module was used for expanding feature vectors when polynomial basis VFA was used. In order to run all experiments in time, we utilized the university's computational cluster "Eddie", which was accessed through the university's VPN service and an SSH connection.

In Table 3.1, the number of parameters that have to be optimized for each VFA method on each task is shown. It can be seen that the feature space expands greatly with the increase of the degree in the polynomial basis VFA. In terms of number of parameters, the 2nd and 3rd degree polynomial basis VFA methods are actually more complex than the deep learning VFA. This does not necessarily translate to better performance, as deep neural networks are known to be able to approximate complex functions with fewer training steps and fewer parameters [19].

Task/VFA	Task 1	Task 2	Task 3
Linear	24	42	303
2nd Degree Polynomial	156	168	15,453
3rd Degree Polynomial	728	504	530,553
Deep	128	126	6,428

Table 3.1: The number of parameters that need to be optimized for the VFA method on each task

Chapter 4

Empirical Comparison Results

The results of the empirical comparison can be seen in the figures and tables below. The discussion of these results is done in the following sections.

4.1 Best 30% of Hyperparameters Results

The average training curve of the best performing 30% of hyperparameter combinations for task 1 can be seen in Figure 4.1 for task 2 it can be seen in Figure 4.2 and for task 3 is can be seen in Figure 4.3.

From the results on task 1 it can be seen that there is a performance advantage for average rewards both in terms of the average rewards acquired when the VFA has converged as well as in terms of the time it takes for the VFA to converge. In terms of variability of performance, between the best performing 30% of hyperparameter combinations, it can be seen that in general, discounted returns have a wider 95% confidence interval. The performance gap between the two return formulations seems to be increasing as we increase the degree of the polynomial basis functions, but when the deep neural network VFA is used the difference in performance is minimal. Interestingly, double Q-learning does seem to stabilise the performance of regular Q-learning when discounted returns are used but it also seems not to be performing as well as regular Q-learning in terms of average rewards acquired at convergence. The statistical test results for task 1 (Table 4.1) also show that indeed the two return formulations have different performance at convergence with the null hypothesis being rejected for all experiment configurations.

The results that can be extracted from task 2 are not as clear or consistent as the ones from task 1. This can be because of the environments increased complexity and

its large negative rewards in the case of repairs (-5000) and maintenance (-500), which in the former case stem from random environment events. In some experiment configurations, such as the experiments with linear VFA, 2nd degree polynomial VFA and deep learning VFA, the performance of the two return formulations is similar at the point of convergence. The time needed for convergence in these configurations is different for each return formulation, but there is no consistency in which of the two formulations is faster to converge. In the case of the 3rd degree polynomial VFA, for Q-Learning and Double Q-Learning, discounted returns outperform average rewards both in terms of speed of convergence and average rewards acquired at convergence. When SARSA and 3rd degree polynomial VFA were used, discounted returns seemed to diverge and thus have much worse performance than average rewards. Also in that experiment configuration, the 95% confidence interval is very large indicating much variability in performance between the best 30% of hyperparameters. The statistical test results for task 2 (Table 4.2), validate the qualitative observations from the training curves, indicating that in some cases (Q-Learning with linear, 2nd degree polynomial basis and deep learning VFA) the performance between the two return formulations is similar and in other cases (Q-Learning and SARSA with 3rd degree polynomial basis VFA), it is not. Interestingly, in the case of Double Q-Learning with 3rd degree polynomial VFA, the statistical test shows that the two formulations have similar performance while the respective training curve in Figure 4.2, shows a large performance gap. This disagreement could be because of the large performance variability for average rewards that is indicated by the large 95% confidence interval.

The results from the experiments on task 3 (Figure 4.3), are again not as clear and consistent as the ones from task 1. From the experiment configurations involving linear VFA and polynomial basis functions VFA, it can be observed that after 20 million training steps, the VFA has not converged and thus the average rewards acquired at the end of training are similar to the average rewards acquired at the beginning of training. When deep Learning VFA is used, a significant improvement in the average rewards acquired at convergence can be seen. For the linear VFA, it can be argued that more training steps might have been needed as the performance increases sharply at the end of training. For the 2nd and 3rd degree polynomial basis VFA it can be argued that because of the large number of trained parameters, again more training steps are needed in order for the VFA to converge to a well-performing value function. None of the return formulations performs better consistently when looking at the these three VFA methods, and no observation can be made for the speed of convergence as no

configuration fully converged. When looking at the deep learning VFA, an advantage of average rewards can be seen both in terms of speed of convergence and the average rewards acquired at convergence. For both return formulations on deep learning VFA though, the 95% interval is again very large indicating much variability in performance across the best 30% of hyperparameters. The observations from the training curves for task 3 are in line with the statistical test results (Table 4.3). For the first three VFA methods, the null hypothesis is mostly accepted as the task is not solved and performance does not improve for any of the formulations, thus remaining similar. Interestingly, the null hypothesis is also mostly accepted for deep learning VFA. That can be explained by the large variability in performance for each formulation, which is similar to what was observed in the results of task 2.

The results discussed above validate the hypothesis that average rewards will have better empirical performance than discounted returns but our findings across all tasks, algorithms and VFA methods do not support our second hypothesis and are not as consistent as the ones presented by Descause [9]. In that work, it was observed that using a more expressive VFA made the gap of performance between the formulations smaller. In our results, this is not consistently seen with the higher order polynomial VFA often increasing the performance gap compared to linear VFA. As mentioned above, this might be explained by the large number of trained parameters for the polynomial basis VFAs.

A more consistent observation in our results is that in the cases where performance was significantly lower for either formulation, the 95% confidence interval was large. This indicates that there is variability in performance across the top 30% of hyperparameters, which is affecting the average training curve and can explain any differences that can be seen in the performance. This large variability can mostly be seen with discounted returns and across all tasks, algorithms and VFA methods. From this observation, we form the new hypothesis that the performance of discounted returns is more sensitive to hyperparameters, and that in the case of optimal hyperparameters, both return formulations should perform similarly.

In order to further investigate this new hypothesis, we experiment with the single best performing hyperparameter combination for each experiment configuration on each task. The results from such experiment show the performance when hyperparameters are close to optimal and thus should eliminate hyperparameter sensitivity as a performance factor.

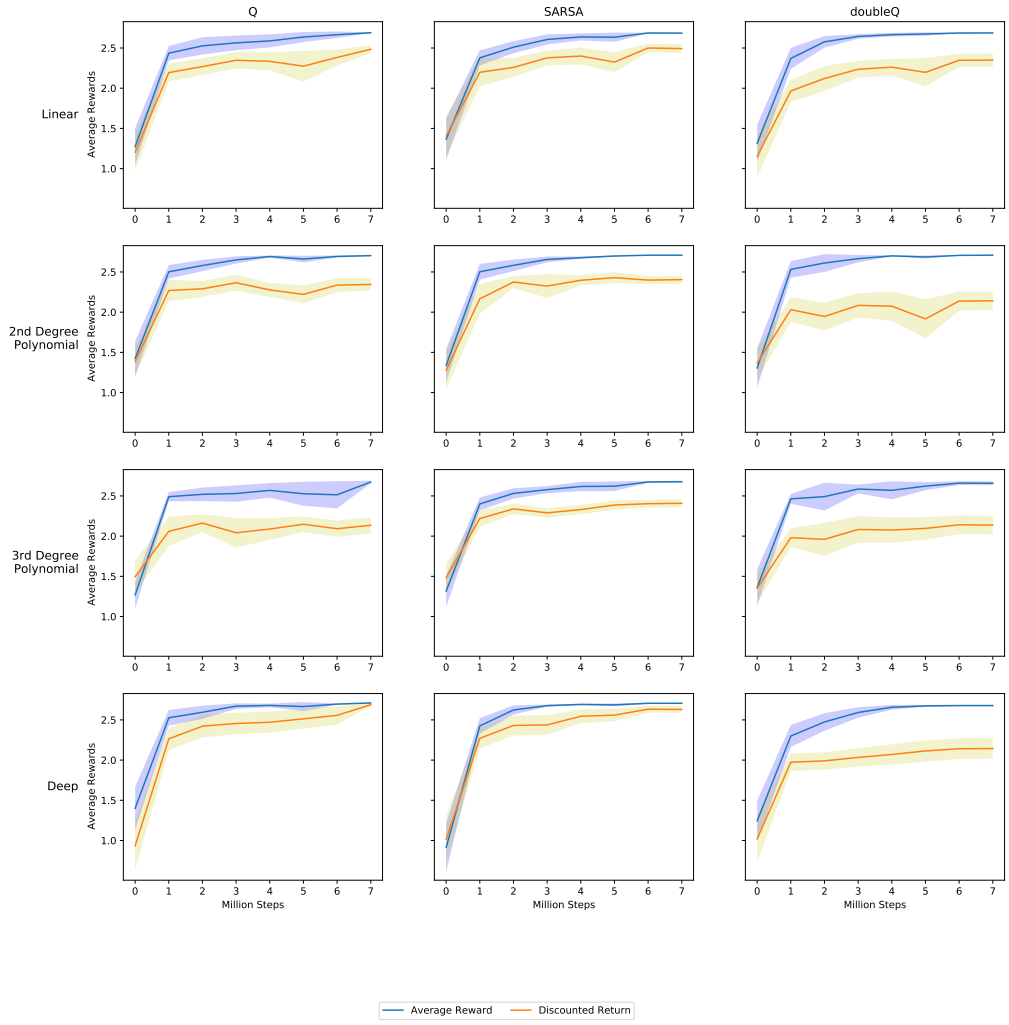

Figure 4.1: The average training curve for the best performing 30% of hyperparameter combinations (out of the 100 sampled) of each experiment configuration on task 1. The shaded area around each curve represent the 95% confidence interval.

Algorithm/VFA	Q-Learning	SARSA	Double Q-Leanring
Linear	2.9e-14	1.1e-15	2.9e-14
2nd Degree Polynomial	2.9e-14	1.6e-17	1.6e-17
3rd Degree Polynomial	8.2e-12	5.7e-13	8.2e-12
Deep	0.0065	8.7e-5	9.2e-11

Table 4.1: The results of the Kolmogorov-Smirnov statistical test for task 1 when taking into account the average performance across the best performing 30% of hyperparameters. The colour indicates whether the null hypothesis can be rejected. Red indicates rejection of the null hypothesis and green indicates acceptance

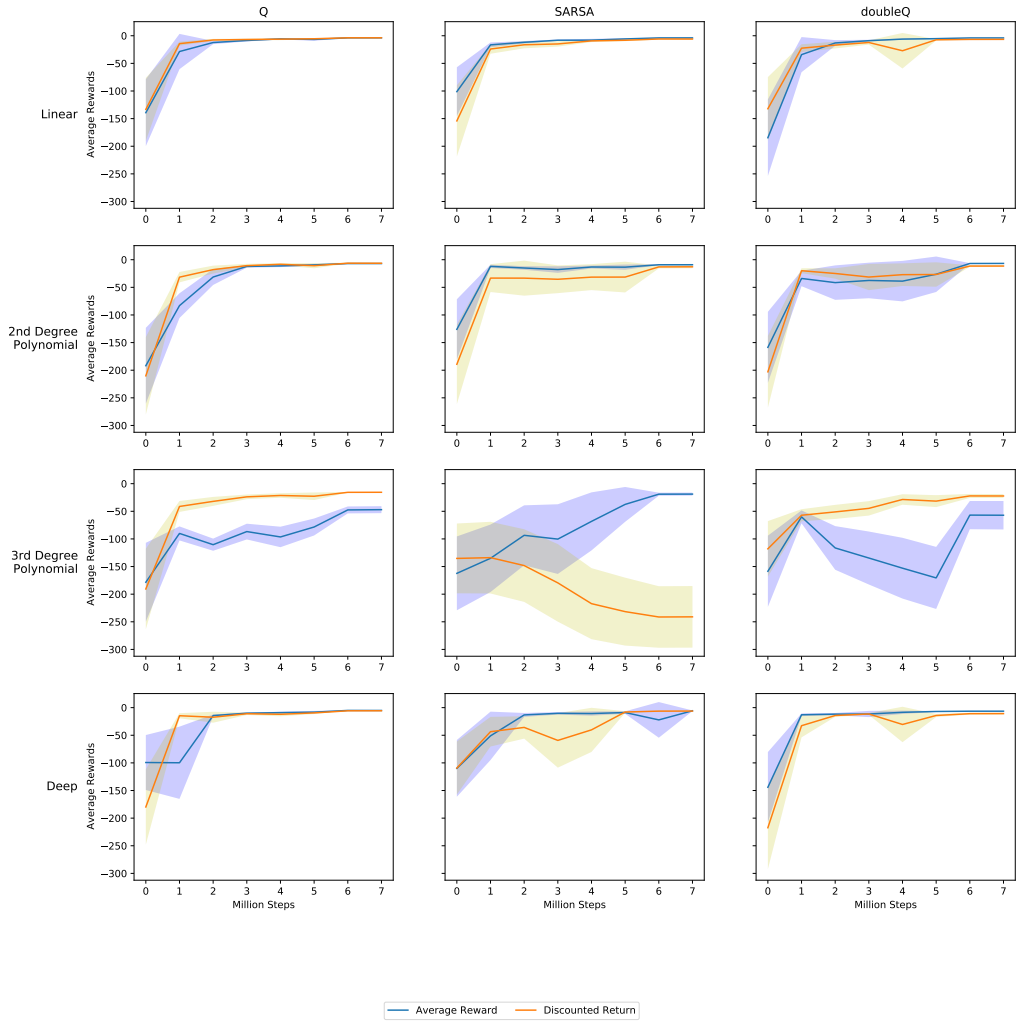

Figure 4.2: The average training curve for the best performing 30% of hyperparameter combinations (out of the 100 sampled) of each experiment configuration on task 2. The shaded area around each curve represent the 95% confidence interval.

Algorithm/VFA	Q-Learning	SARSA	Double Q-Leanring
Linear	0.135	0.0346	1.3e-6
2nd Degree Polynomial	0.3929	0.0008	1.3e-6
3rd Degree Polynomial	5.7e-13	6.5e-9	0.0709
Deep	0.3929	0.2391	2.3e-5

Table 4.2: The results of the Kolmogorov-Smirnov statistical test for task 2 when taking into account the average performance across the best performing 30% of hyperparameters. The colour indicates whether the null hypothesis can be rejected. Red indicates rejection of the null hypothesis and green indicates acceptance

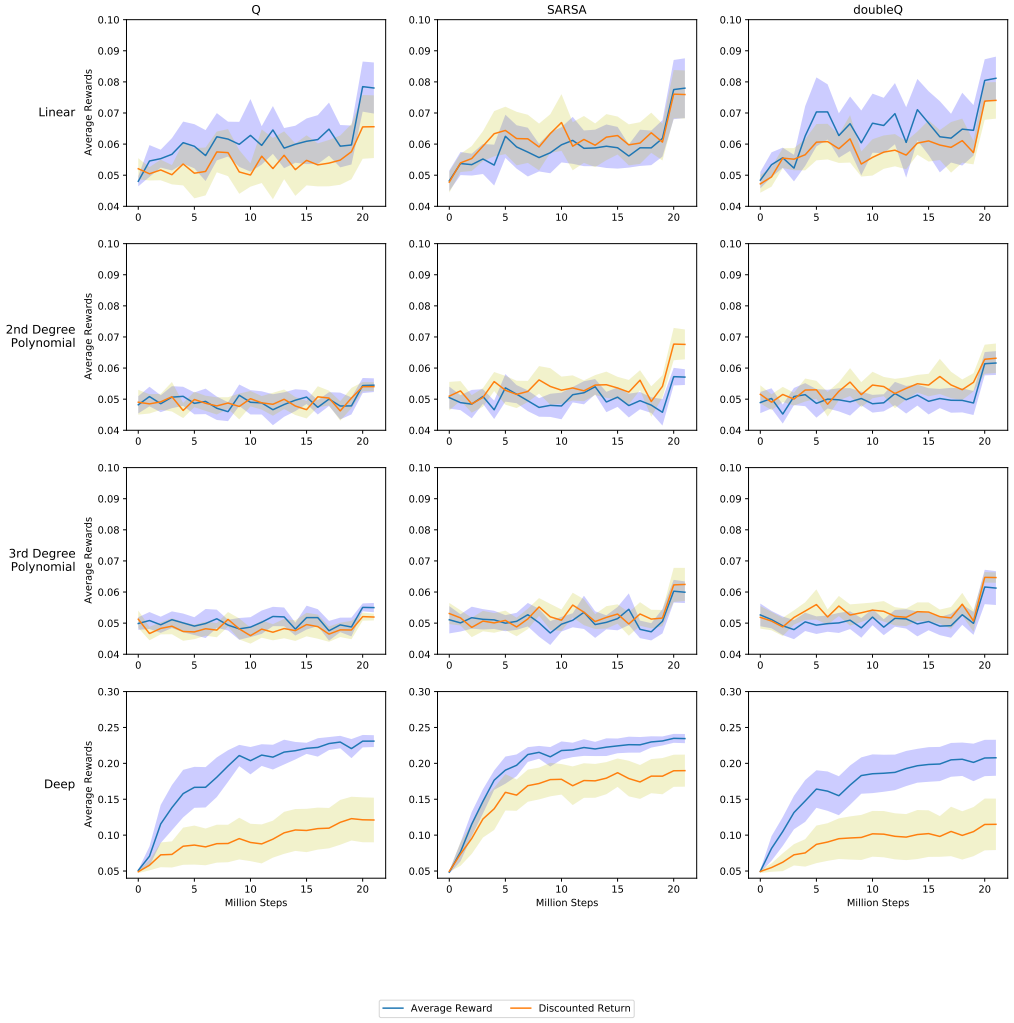

Figure 4.3: The average training curve for the best performing 30% of hyperparameter combinations (out of the 50 sampled) of each experiment configuration on task 3. The shaded area around each curve represent the 95% confidence interval.

Algorithm/VFA	Q-Learning	SARSA	Double Q-Leanring
Linear	0.0003	0.9578	0.2391
2nd Degree Polynomial	0.9578	0.0346	0.8080
3rd Degree Polynomial	0.0003	0.2391	0.0009
Deep	0.0025	0.1350	0.1250

Table 4.3: The results of the Kolmogorov-Smirnov statistical test for task 3 when taking into account the average performance across the best performing 30% of hyperparameters. The colour indicates whether the null hypothesis can be rejected. Red indicates rejection of the null hypothesis and green indicates acceptance

4.2 Best Hyperparameter Results

As discussed in Section 4.1, the new hypothesis from our initial results is that average rewards are less sensitive to hyperparameters than discounted returns. To further validate this hypothesis, we can try to eliminate the hyperparameter sensitivity factor from the experiments and investigate whether there is still a performance gap between the two return formulations. The results from task 1, 2 and 3 can be seen in Figure 4.4, 4.5 and 4.6 respectively.

The results on task 1 show that the performance of the two formulations is very similar. The only time where that is not observed is for the 2nd and 3rd degree polynomials where the performance gap between the two formulations is increasing in favour of the average rewards. This performance difference, could be because the hyperparameters tested in this experiment setup might not be the actual best performing hyperparameters from the 100 sampled, but randomly displayed the best performance when sampling was performed. This can also explain why the 95% confidence interval is larger for discounted returns in these cases. The quantitative results that can be seen in Table 4.4, show that the difference in average rewards acquired at convergence between the two formulations is statistically significant in 50% of the experiment configurations, which is not enough to conclude that there is any difference in performance between the formulations. As it was observed in the previous experiment setup, when looking at the training curve of Q-Learning and Double Q-Learning with discounted return, we can see an increase in training stability and average rewards acquired at convergence for Double Q-Learning. This also indicates that the difference in performance between discounted returns and average rewards in the case of Q-Learning, could be attributed to the overestimation bias discussed in Section 2.5.

Similarly to the previous experiment setup in Section 4.1, it is very much difficult to draw conclusions from the results on task 2 as there is a lot of instability and variance in performance across runs. In some cases, such as when Q-Learning is paired with linear VFA or SARSA is paired with 2nd degree polynomial basis VFA, the final performance of discounted returns is better than of average rewards. In other cases, such as when SARSA is paired with linear VFA or 3rd degree polynomial basis VFA the opposite applies. This instability is also reflected in the statistical test results (Table 4.5) where the null hypothesis is rejected in 75% of the experiment configurations. This instability and sub-optimal performance might have been caused by the random seeds and the fact that, as mentioned in Section 4.1, task 2 has large negative rewards that may easily

skew the average reward acquired downwards. Also, another cause could again be that the hyperparameters used in these configurations were not the actual best performing.

On task 3, similarly to the results of task 1, the results are much more easily interpretable. In most cases, the performance difference between the two formulations in terms of average rewards acquired at convergence is minimal except for when Double Q-Learning was paired with deep learning VFA, when SARSA was paired with polynomial basis functions and when Q-Learning was paired with 3rd degree polynomial basis VFA. Neither of the two formulations was performing better consistently across these four experiment configurations, indicating that it could be again because non-optimal hyperparameters were chosen, as discussed previously. These observation are supported by the statistical test results (Table 4.6) which show that in the majority of the experiment configurations, the null hypothesis is accepted.

Overall, the results from these experiments seem to contradict our initial hypotheses but actually support our new hypothesis that was formed in Section 4.1. Indeed, when hyperparameter variability was removed across runs of the same experiment configuration, the difference in acquired reward at convergence between the two formulations was significantly reduced or even statistically eliminated. By comparing the results from the statistical tests across the two experiment setups, we can also see that the null hypothesis is accepted more often when the hyperparameter variability is removed than when the performance of multiple hyperparameters is averaged. The observation that when hyperparameters are optimal, then the difference in performance across return formulations also validates the theoretical arguments made by Tsitsiklis and Van Roy [5] who argued that discounted return should perform similar to average rewards under the conditioned that the hyperparameters are tuned.

In order to further investigate and validate that our new hypothesis from Section 4.1 is correct, in Section 4.3 the distribution of the average reward acquired when the VFA has converged will be investigated for both return formulation on all experiment configurations.

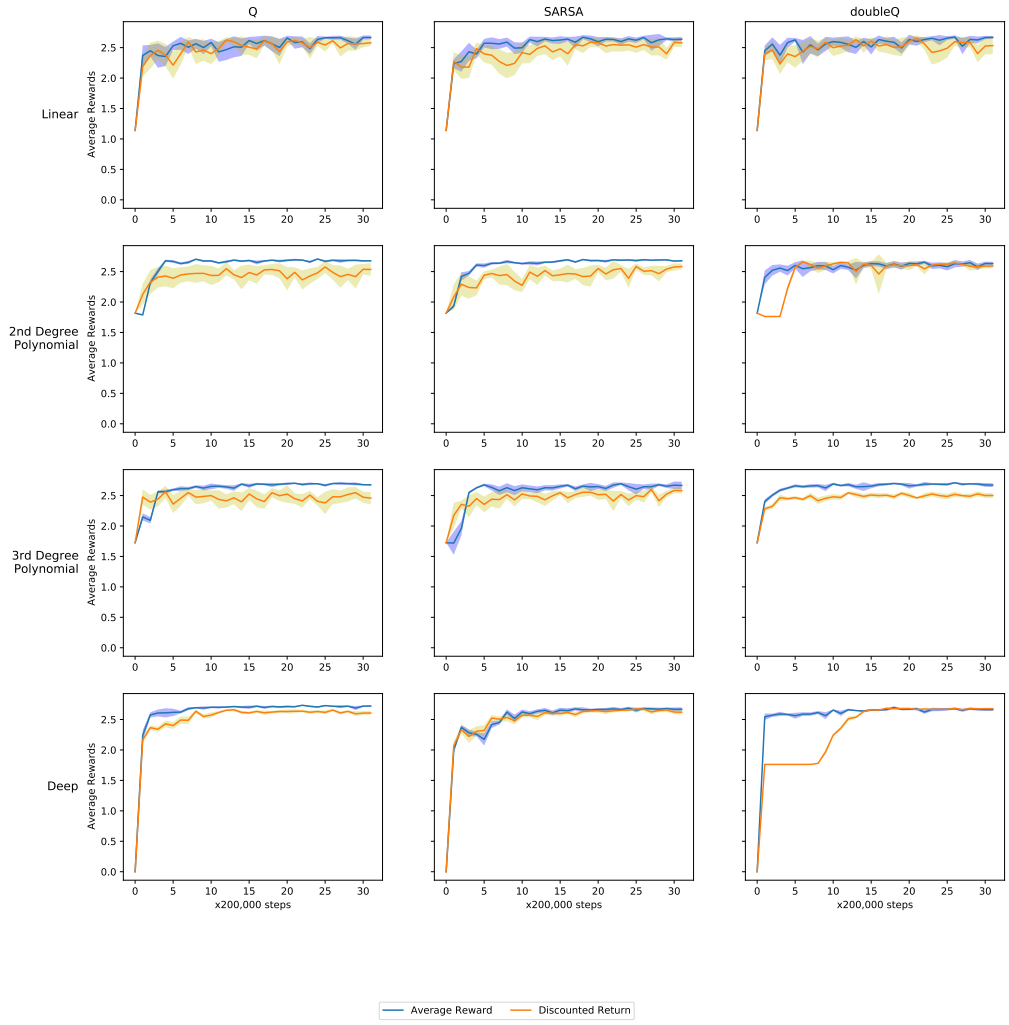

Figure 4.4: The training curve for the single best performing hyperparameter combination (out of the 100 sampled) of each experiment configuration on task 1. The shaded area around each curve represent the 95% confidence interval.

Algorithm/VFA	Q-Learning	SARSA	Double Q-Leanring
Linear	0.1678	0.7869	0.4175
2nd Degree Polynomial	0.0002	0.0002	0.1678
3rd Degree Polynomial	0.0002	0.0123	0.0002
Deep	1.082e-5	0.1678	0.0524

Table 4.4: The results of the Kolmogorov-Smirnov statistical test for task 1 when taking into account the performance of the single best performing hyperparameters. The colour indicates whether the null hypothesis can be rejected. Red indicates rejection of the null hypothesis and green indicates acceptance

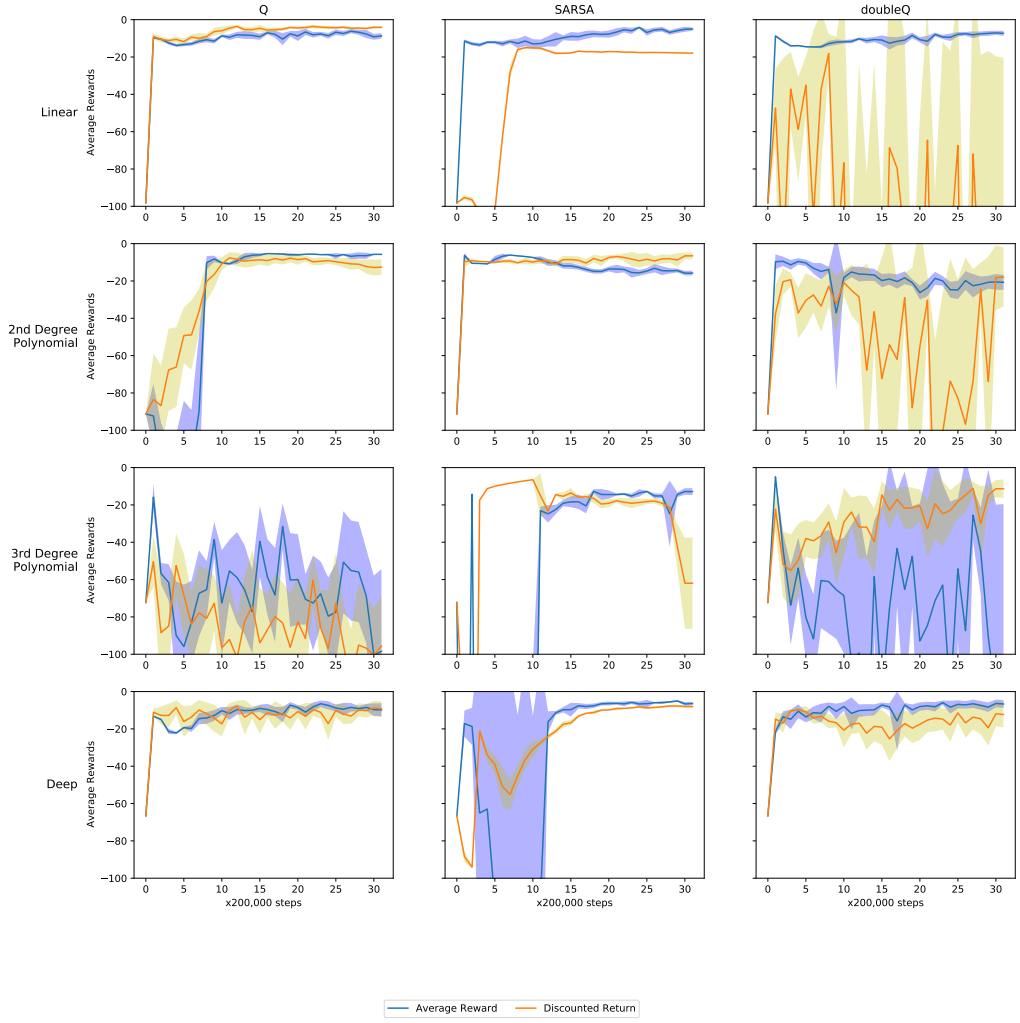

Figure 4.5: The training curve for the single best performing hyperparameter combination (out of the 100 sampled) of each experiment configuration on task 2. The shaded area around each curve represent the 95% confidence interval.

Algorithm/VFA	Q-Learning	SARSA	Double Q-Leanring
Linear	0.0002	1.082e-5	0.0123
2nd Degree Polynomial	0.0002	0.0002	0.0123
3rd Degree Polynomial	0.9944	1.082e-5	0.0021
Deep	0.7869	0.0123	0.7869

Table 4.5: The results of the Kolmogorov-Smirnov statistical test for task 2 when taking into account the performance of the single best performing hyperparameters. The colour indicates whether the null hypothesis can be rejected. Red indicates rejection of the null hypothesis and green indicates acceptance

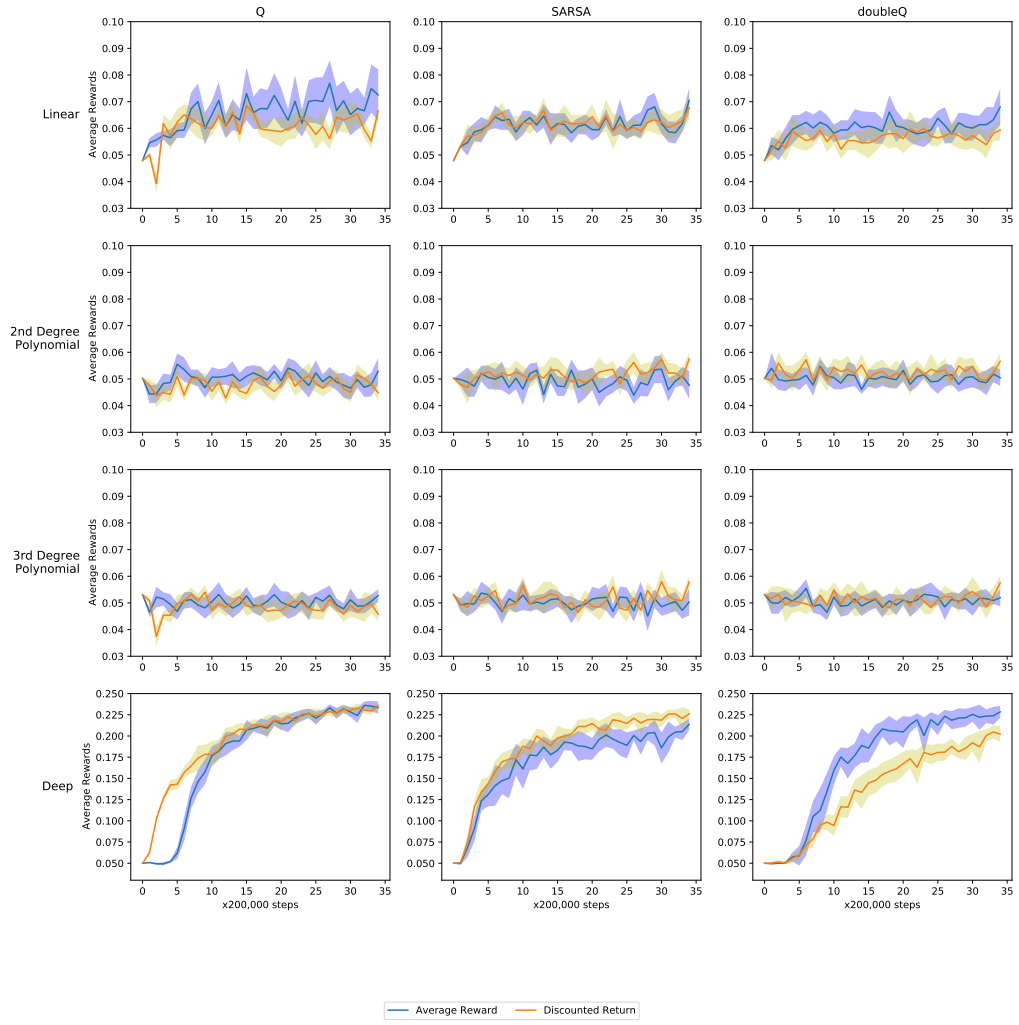

Figure 4.6: The training curve for the single best performing hyperparameter combination (out of the 100 sampled) of each experiment configuration on task 3. The shaded area around each curve represent the 95% confidence interval.

Algorithm/VFA	Q-Learning	SARSA	Double Q-Leanring
Linear	0.1678	0.7869	0.1678
2nd Degree Polynomial	0.0524	0.0021	0.0524
3rd Degree Polynomial	0.0021	0.0123	0.0524
Deep	0.4175	0.0524	0.0123

Table 4.6: The results of the Kolmogorov-Smirnov statistical test for task 3 when taking into account the performance of the single best performing hyperparameters. The colour indicates whether the null hypothesis can be rejected. Red indicates rejection of the null hypothesis and green indicates acceptance

4.3 Hyperparameter Sensitivity

As it was discussed in Sections 4.1 and 4.2, the performance advantage in favour of average rewards that was observed with our initial experiment setup and in the results of Descause [9], seems to be significantly reduced when near optimal hyperparameters are used for each return formulation. This observation goes against our initial hypotheses but supports the new hypothesis that was formulated in the end of Section 4.1. Our new hypothesis states that average rewards are less sensitive to hyperparameters than discounted returns, and that is why a performance difference is seen when averaging the best performing 30% of hyperparameter combinations. In order to further validate this hypothesis we will investigate the distribution of the average rewards acquired at convergence for both return formulations. The spread of the distribution will show whether a distribution is sensitive to hyperparameters. Thus, we expect the performance distribution of the average rewards to be more narrow and be centred at a higher average reward than the distribution of discounted returns.

The histograms from task 1 (Figure 4.7) indeed show that the distribution is more narrow and positioned on higher rewards in the case of the average rewards formulation. This shows that when average rewards are used as the return formulation, different hyperparameter combinations have a similar, high performance at convergence. On the other hand, when discounted returns are used, some hyperparameters achieve high performance but the similarity in performance between the different hyperparameters is not as great, thus lowering the overall average performance.

The same results can mostly be seen in the histograms from task 2 (Figure 4.8). In most cases, the average rewards formulation's distribution is more narrow and centred higher than the distribution of discounted returns. An exception to this is when Q-Learning is paired with 3rd degree polynomial basis VFA, which reflects the observation from the training curve of the same experiment configuration in Figure 4.2. This exception, can be again justified by the large negative rewards of task 2, which are often received after random environment events (such as a production failure).

The histograms from task 3 (Figure 4.9) are more difficult to parse than the ones from task 1 and task 2 mainly because the first three VFA methods had trouble solving the task and thus the distributions are centered similarly on low scores. In the case of deep learning VFA, the distributions for the average reward formulation are significantly more narrow and centered towards higher values than the distributions for the discounted return formulation. This is again in line with what was expected and what

was seen for the previous tasks.

Overall, the results that can be extracted from the performance distributions of both formulations are in line with our expectations. The results fully support our new hypothesis that average rewards are less sensitive to hyperparameters than discounted returns, as their distributions were more compact and overall centred on higher reward values. This, along with the results from Section 4.2, provide good evidence in favour of our new hypothesis. These results also provide some explanation for the results of Descause [9] and of Section 4.1, where a significant difference in performance between the two return formulations was seen.

Figure 4.7: Final performance histograms for the best performing 30% of runs of each experiment configuration on task 1 (average rewards: top, discount returns: bottom)

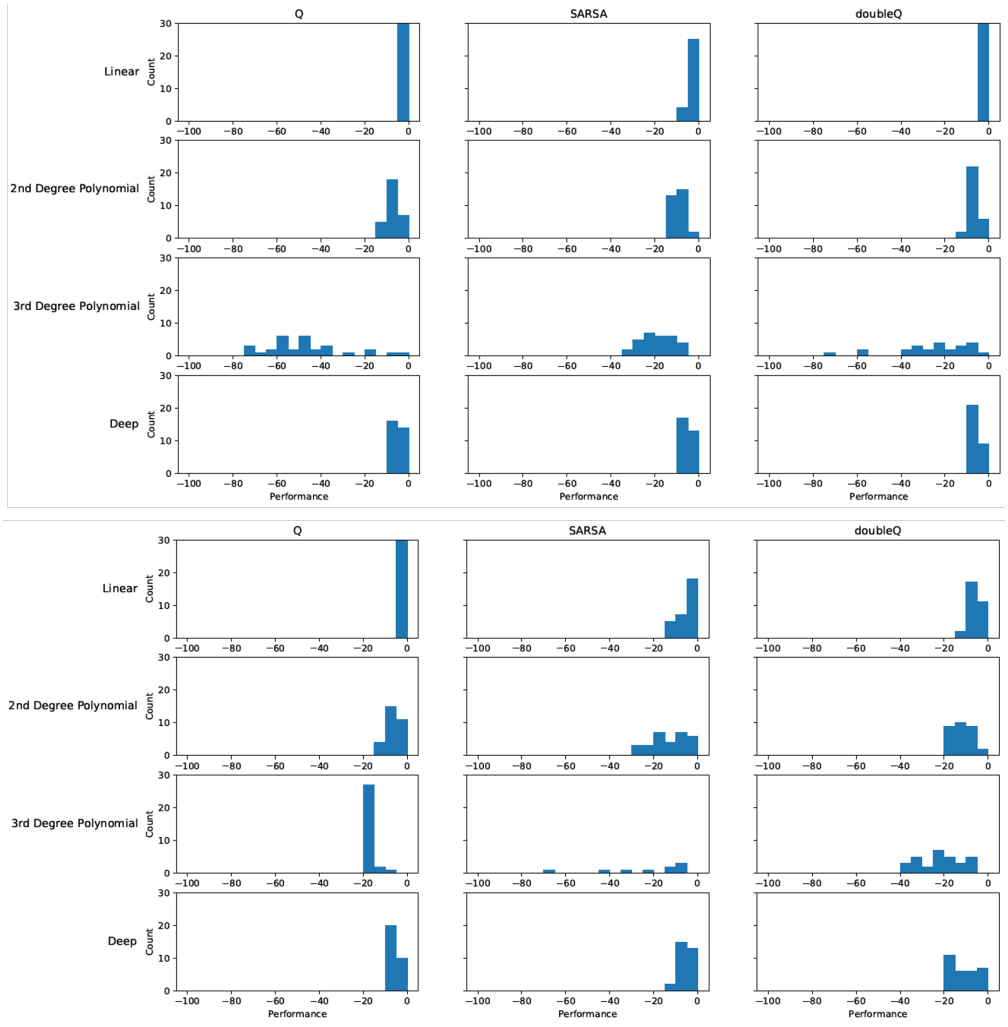

Figure 4.8: Final performance histograms for the best performing 30% of runs of each experiment configuration on task 2 (average rewards: top, discount returns: bottom)

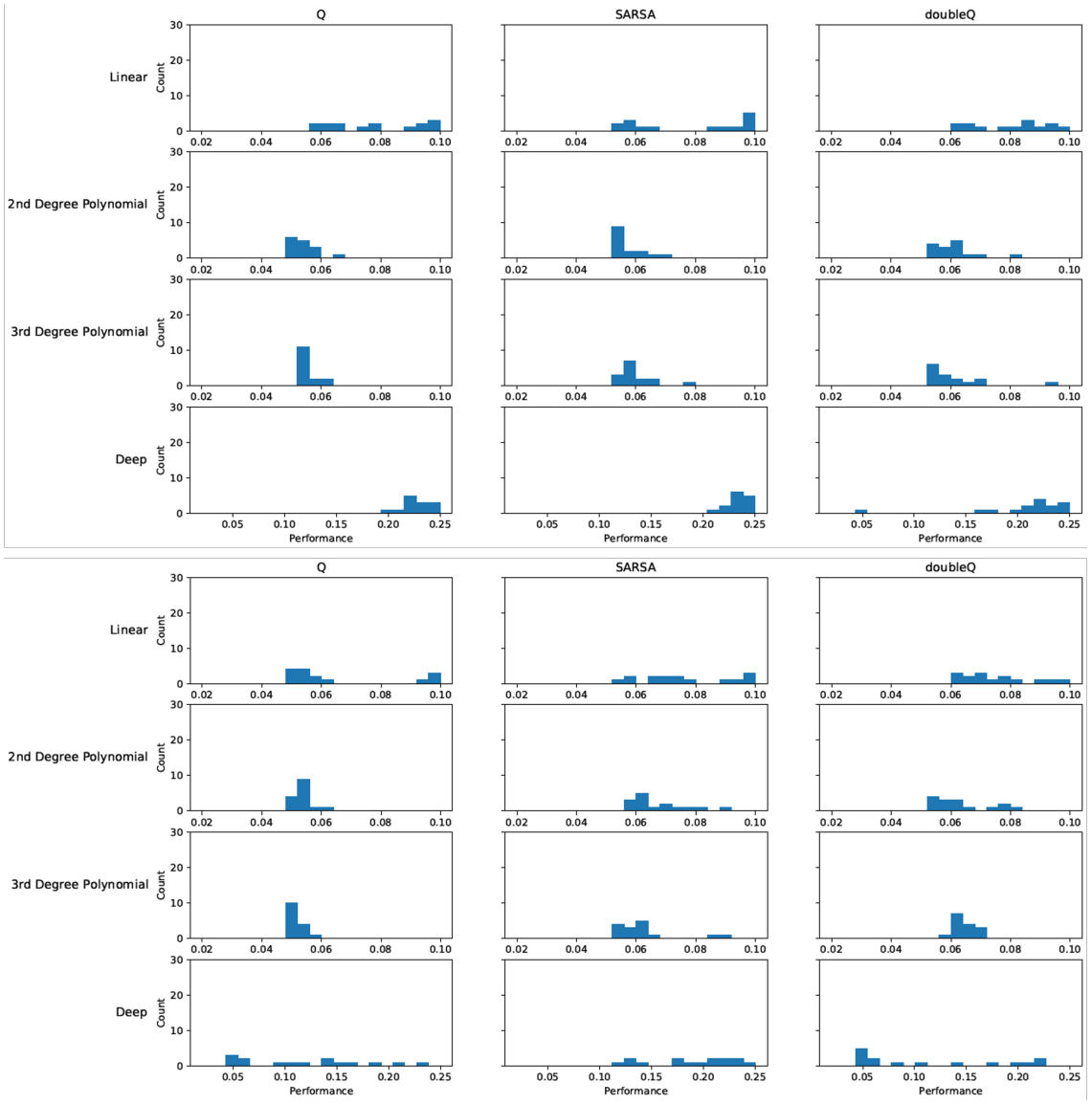

Figure 4.9: Final performance histograms for the best performing 30% of runs of each experiment configuration on task 3 (average rewards: top, discount returns: bottom)

Chapter 5

Conclusions

In this dissertation, we empirically compared the performance of average rewards and discounted returns in complex environments that required the use of VFA. Our investigation was setup to answer our research question: Is there a difference in the empirical performance of the two formulations and why do we see performance differences in practice?

We started with hypotheses that were motivated by previous empirical work done by Schwartz [8], Mahadevan [7] and Descause [9]. Our first hypothesis was that average rewards would have better performance than discounted returns in terms of the average rewards acquired at convergence and the speed of convergence. Our second hypothesis was that the difference in performance would be because of the difference in the difficulty of action-value estimation for each return formulation. Our results from our first experiment, where we looked at the training curves of the best performing 30% of hyperparameter combinations out of all sampled, partially supported our hypotheses and they were not entirely matching the results of Descause [9]. Our results did indeed show a difference in performance between formulations, which was statistically validated using the Kolmogorov-Smirnov test. The performance difference was in favour of average rewards but it did not get smaller as the VFA complexity was increased. This latter observation, along with the observation that the 95% confidence interval on discounted returns was often larger than on average rewards, motivated the formulation of a new hypothesis. The new hypothesis attributed the performance difference to hyperparameter sensitivity and proposed that when optimal hyperparameters are chosen, both formulations should acquire similar average rewards at convergence and converge at the same speed. We then experimented with the single best performing hyperparameter combinations out of all sampled and observed that in many cases

the performance of the two formulations was similar, which we again statistically validated using the Kolmogorov-Smirnov test. Our third and final experiment setup, at which the average reward acquired distributions were examined for each formulation, also supported our new hypothesis with the distribution of average rewards being more narrow and on higher values than the one of discounted returns.

A drawback of our investigation concerned the optimality of the hyperparameters chosen as the best performing ones in our second experiment setup. This is what mainly caused the instability of some results in Section 4.2, which prevented us from drawing conclusions from these results. This problem was caused by only training once with each hyperparameter combination when hyperparameter random sampling was performed. Because of that, the final performance may have been influenced positively by random environment events which could have made some hyperparameter combinations appear as the best performing ones while in reality they were not. Additionally, another drawback involved the number of sampled hyperparameter combinations for experiments on task 3, which was 50. Given the large range from which we sampled hyperparameters, 50 samples are too little for near-optimal hyperparameters to be found.

To conclude, our experimental results indicate that indeed a performance difference can be seen between the two return formulations. This performance difference though is because of the difference in hyperparameter sensitivity. With near optimal hyperparameter tuning, the two formulations should perform similarly. We believe that this work can be further extended by using more tasks, sampling more hyperparameter combinations (especially in the case of experiments on task 3) and training with each sampled hyperparameter combination multiple times. Also, experimentation with different VFA methods, especially with ones that have no exploding feature space (like polynomial basis functions do), would be a worthwhile addition.

Bibliography

[1] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, and et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. ISSN 0028-0836. doi: 10.1038/s41586-020-03051-4.

[2] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning, 2019.

[3] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019.

[4] Karen Hao. We analyzed 16,625 papers to figure out where ai is headed next, 2019. URL `https://www.technologyreview.com/2019/01/25/1436/we-analyzed-16625-papers-to-figure-out-where-ai-is-headed-next/`.

[5] John N. Tsitsiklis and Benjamin Van Roy. On average versus discounted reward temporal-difference learning. *Machine Learning*, 49(2):179–191, Nov 2002. ISSN 1573-0565. doi: 10.1023/A:1017980312899. URL `https://doi.org/10.1023/A:1017980312899`.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[7] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1):159–195, Mar 1996.

ISSN 1573-0565. doi: 10.1007/BF00114727. URL https://doi.org/10.1007/BF00114727.

[8] Anton Schwartz. A reinforcement learning method for maximizing undiscounted rewards. pages 298–305, 12 1993. ISBN 9781558603073. doi: 10.1016/B978-1-55860-307-3.50045-9.

[9] Lucas Descause. Reinforcement learning with function approximation in continuing tasks: Discounted return or average reward? Master's thesis, 2019.

[10] Panagiotis Kyriakou. Informatics project proposal: Reinforcement learning with function approximation in continuing tasks: Discounted return or average reward? Master's thesis, 2021.

[11] RICHARD BELLMAN. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. ISSN 00959057, 19435274. URL http://www.jstor.org/stable/24900506.

[12] Nintendo of America. Tetris, 1989.

[13] 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 Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/mniha16.html.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. 09 2015.

[15] Hado Van Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[16] Stephen Dankwa and Wenfeng Zheng. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. pages 1–5, 08 2019. doi: 10.1145/3387168.3387199.

[17] CLAUDE E. SHANNON. Programming a computer for playing chess. *Philosophical Magazine*, 41, Mar 1949.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] Vektor Dewanto, George Dunn, Alireza Eshragh, Marcus Gallagher, and Fred Roosta. Average-reward model-free reinforcement learning: a systematic review and literature mapping. 10 2020.

[21] Yiming Zhang and Keith W. Ross. On-policy deep reinforcement learning for the average-reward criterion, 2021.

[22] Yi Wan, Abhishek Naik, and Richard S. Sutton. Learning and planning in average-reward markov decision processes, 2021.

[23] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[24] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. *Object Recognition with Gradient-Based Learning*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-540-46805-9. doi: 10.1007/3-540-46805-6_19. URL `https://doi.org/10.1007/3-540-46805-6_19`.

[25] Giorgos Tolias, Ronan Sicre, and Hervé Jégou. Particular object retrieval with integral max-pooling of cnn activations, 2016.

[26] Sridhar Mahadevan, Nicholas Marchalleck, Tapas K. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proceedings of the 14th International Conference on Machine Learning*, pages 202–210. Morgan Kaufmann, 1997.

[27] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms, 2019.

[28] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[29] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Appendix A

Appendix A: Learning Algorithms Pseudocode

Algorithm 1 Q-Learning with discounted returns

Initialize $Q(s,a)$, for all $s \in S$, $a \in A$, arbitrarily
Initialize S
while current step $<$ maximum steps **do**
    Choose A from S using ε-greedy policy on Q
    Take Action A, observe R and S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma * \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
end while

Algorithm 2 SARSA with discounted returns

Initialize $Q(s,a)$, for all $s \in S$, $a \in A$, arbitrarily
Initialize S
Choose A from S using ε-greedy policy on Q
while current step $<$ maximum steps **do**
    Take Action A, observe R and S'
    Choose A' from S' using ε-greedy policy on Q
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma * Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$, $A \leftarrow A'$
end while

Algorithm 3 Double Q-Learning with discounted returns

Initialize $Q_A(s,a)$ and $Q_B(s,a)$, for all $s \in S$, $a \in A$, arbitrarily
Initialize S
while current step $<$ maximum steps **do**
  Choose A from S using ε-greedy policy on Q_A and Q_B
  Take Action A, observe R and S'
  Choose randomly either $UPDATE(A)$ or $UPDATE(B)$
  if $UPDATE(A)$ **then**
    $a^* = argmax_a Q_A(S',a)$
    $Q_A(S,A) \leftarrow Q_A(S,A) + \alpha[R + \gamma * Q_B(S',a^*) - Q_A(S,A)]$
  else
    $a^* = argmax_a Q_B(S',a)$
    $Q_B(S,A) \leftarrow Q_B(S,A) + \alpha[R + \gamma * Q_A(S',a^*) - Q_B(S,A)]$
  end if
  $S \leftarrow S'$
end while

Appendix B

Appendix B: Hyperparameter Samples

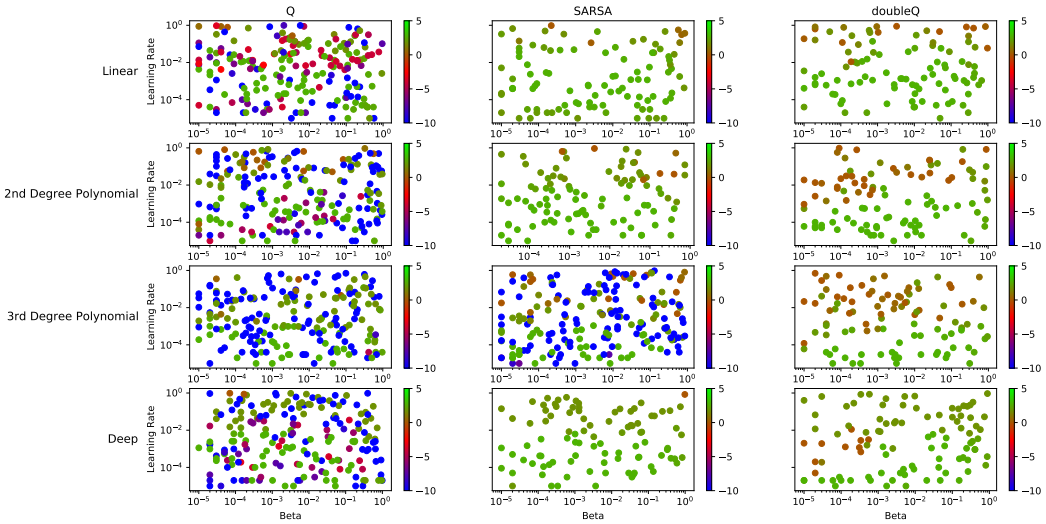

Figure B.1: Randomly sampled hyperparameters for average rewards on task 1

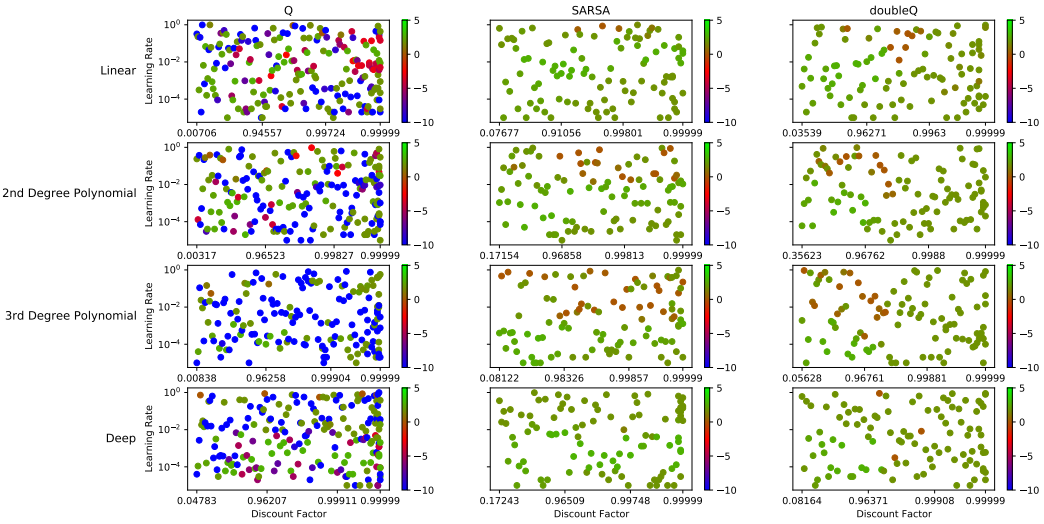

Figure B.2: Randomly sampled hyperparameters for discounted returns on task 1

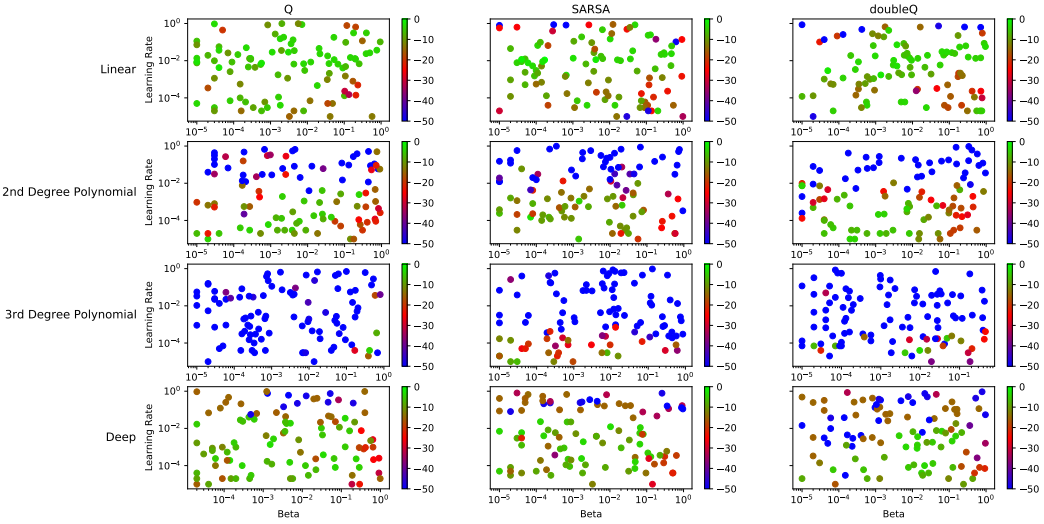

Figure B.3: Randomly sampled hyperparameters for average rewards on task 2

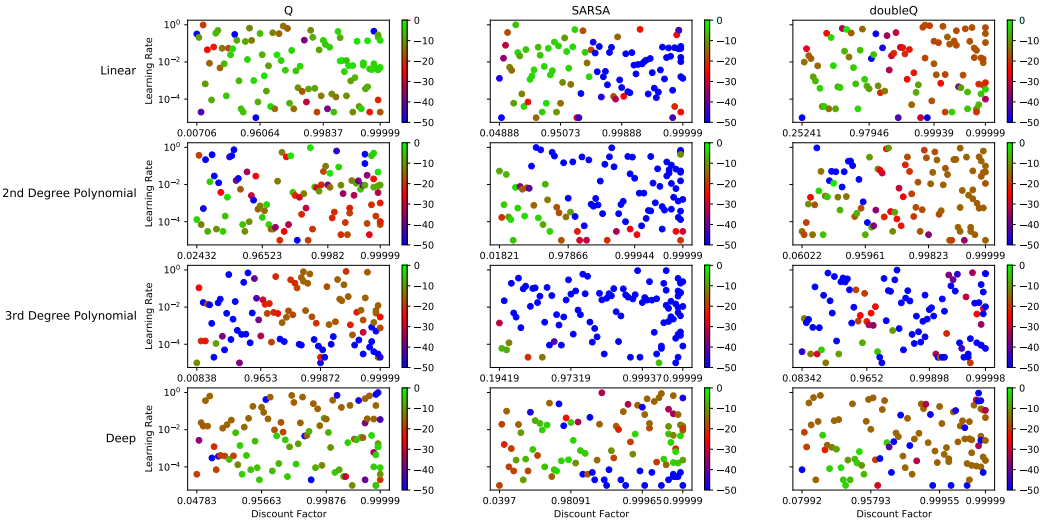

Figure B.4: Randomly sampled hyperparameters for discounted returns on task 2

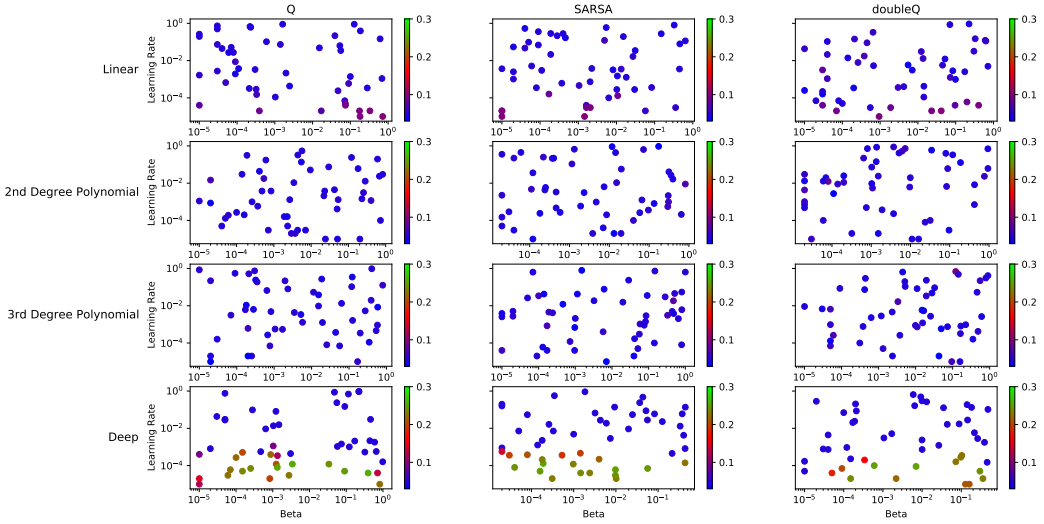

Figure B.5: Randomly sampled hyperparameters for average rewards on task 3

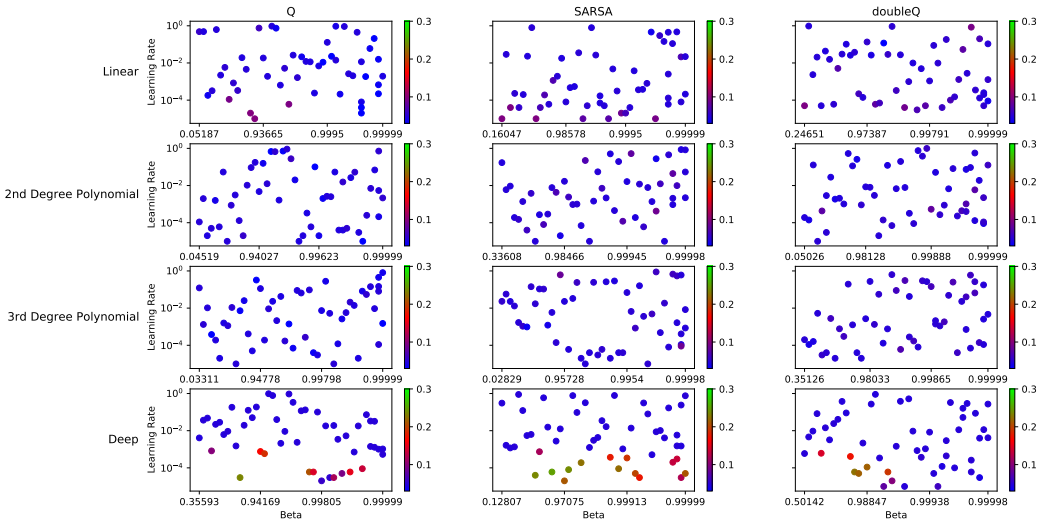

Figure B.6: Randomly sampled hyperparameters for discounted returns on task 3