An exploration of the differences between Semi- and Full-gradient TD-learning

David Wessels, David Knigge, Daan Le, Paul ten Kaate

October 2020

Abstract

In this project we explore the differences between full- and semi-gradient temporal difference learning (TD) in function approximation. To this end, we experiment with both linear function approximation and Deep-Q-Network (DQN) function approximation, on two classic reinforcement learning settings; the MountainCar and CartPole problems. Furthermore, we explore the impact of weight initialisation on the convergence and performance after convergence of the full- and semi-gradient approaches. Our experiments show that in linear function approximation settings, semi-gradient significantly outperforms full-gradient TD learning, whereas when using DQN function approximation, full-gradient TD learning seems to outperform semi-gradient learning, depending on the environment and task it is learning. Furthermore, weight initialisation in both methods seems to impact variance in the learning process, but does not seem to significantly alter the convergence speed or performance of the resulting policy of any methods. Lastly, we discuss limitations of our research, and give suggestions for future research ventures in this field.

1 Introduction

Over recent years, research in the field of reinforcement learning (RL) has yielded exciting results, with deep reinforcement learning algorithms achieving human and even super-human control over a range of tasks [1, 8]. Much of this recent progress finds its roots in the introduction of temporal difference methods. In this project we explore characteristics of temporal-difference learning when it is incorporated in approximate solution methods, where the value function is approximated as a parameterised functional form with a set of weights. Prototypical TD-learning approximation methods like semi-gradient TD(0), and semi-gradient SARSA update these weights by ignoring their influence on the bootstrapping target, to break the circular dependency that is otherwise expected to cause divergence in learning relevant approximations [6]. In this paper we explore this intuition by comparing semi-gradient approximation methods with their full-gradient counterparts, in which the true gradient of the error function is used in a weight update step. We compare their performance and rate of convergence on classic reinforcement learning environments. Furthermore, as the impact of the circular dependence between weights and reward signal may intuitively be heavily dependent on the absolute values of the weights at the start of the learning process, we assess the impact of weight initialisation on performance and convergence as well.

2 Background

2.1 Temporal-difference methods

One idea central to the recent progress in RL is that of *temporal-difference* (TD) learning [2]. Other than in dynamic programming approaches, TD learning is able to incorporate information obtained from experience directly, without any underlying knowledge of the environment. Furthermore, in contrast to other model-free approaches, such as Monte Carlo (MC) methods, it enables performing *backup operations* not after conclusion of an episode, but rather incorporates a current estimate of the future rewards in an update step, before the final outcome of the episode is known. The use of this current estimate of the value function is known as *bootstrapping*, an approach that has allowed a dramatic speed-up in convergence in many problem instances, when compared to MC methods [3]. The error which is minimised in temporal-difference learning approaches, the TD-error δ_t in timestep t, is given by:

$$\delta_t = R_{t+1} + \gamma v(s_{t+1}) - v(s_t) \tag{1}$$

Where R_{t+1} is the reward obtained in state s_{t+1}, γ is the temporal discount factor, and v is the value function, returning the total expected value for a given state under the current policy.

2.2 Temporal-difference learning in approximate solution methods

In many problem instances the number of possible states of the world, or the fact that we do not have all information about which state the world is currently in, prevents us from using tabular methods to learn desirable policies. In these cases, we can define the approximate value function $\hat{v}(s, w)$ by a set of weights w to be applied on a set of features representing the current state, instead of a table of values in which each state has an entry. We now aim at minimising the mean squared value error ($\bar{VE}$), given by taking the weighted average of the square of value prediction errors at every state:

$$\bar{VE} = \sum_{s=1}^{S} \mu(s)[v_\pi(s_{t+1}) - \hat{v}(s_t, w)]^2 \tag{2}$$

Where $v_\pi(s)$ is the true underlying value function, and $\mu(s)$ is a weight corresponding to the importance of the error in state s. As we do not know the true underlying value function, we instead, as with tabular TD-learning methods, use bootstrapping to approximate it. The target v_π is replaced in equation 2 with an approximation under the current approximate value function: $R_{t+1} + \gamma\hat{v}(s_{t+1}, w)$. In analogy to equation 1, we then end up with:

$$\bar{VE} = \sum_{s=1}^{S} \mu(s)[R_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)]^2 \tag{3}$$

In semi-gradient TD(0), weight updates are now performed using the following formula, assuming we are using SGD and sampling states under distribution μ [4]:

$$w_{t+1} = w_t + \alpha[R_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)]\nabla\hat{v}(s_t, w) \tag{4}$$

This method is named a semi-gradient approach, since the dependence of the target $R_{t+1} + \hat{v}(s_{t+1}, w)$ on w is not incorporated in the weight update. The full gradient of the error function is not used in the weight update. If we would simply take the gradient of the error shown in equation 3, we would expect the following equation:

$$w_{t+1} = w_t - \alpha[R_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)](\gamma\nabla\hat{v}(s_{t+1}, w) - \nabla\hat{v}(s_t, w)) \tag{5}$$

The main motivation for semi-gradient methods over full-gradient methods is that theoretically, taking the full gradient would create circular dependence; the target value for a given state is dictated by the weights, which are being updated using gradient information obtained from the target value. Therefore, the loss function is dependent on the weights which in turn means that optimising with relation to the loss function does not guarantee that our approximate value function approaches the true value function [9].

However, semi-gradient methods have their own drawbacks. It should be noted that, since semi-gradient TD methods are not "true" gradient descent methods, their convergence to even a local optimum is not guaranteed, because of the bias induced in using the bootstrapping method described above. For this reason, we aim to explore the practical consequences of the discrepancy between full and semi-gradient TD-learning on convergence. As explained above, we expect the weight initialisation in the full-gradient case to have a large impact on whether or not the learning method will convergence for a given environment, and at which rate it converges, since the larger the randomly initialised weights in absolute value are, the larger their contribution on the learning signal will be at the start of the learning process. Therefore, we experiment with different magnitudes of a classical initialisation scheme; a normal distribution [10].

Furthermore, in approximate TD-learning the learning signal essentially consists of two parts; the reward obtained by performing an action at the current state and the bootstrapped approximation for the expected reward in future states following the current policy the current action. The difference between full- and semi-gradient is whether the gradient of this bootstrapped approximation is taken into account in the update rule, which is exactly the difference between equations 4 and 5. Since the learning signal at the start of the learning process consists of the reward signal and the - at this point - noisy bootstrapped approximation for future reward, we expect the magnitude and type of reward obtained after actions to greatly impact the learning process for full-gradient TD methods. For example, intuitively, a larger reward may drown out the noise from the bootstrapped expected reward at the start of the learning process. To investigate this intuition we explore two environments with very distinct reward schemes; discussed in the next section.

3 Experimental setup

To test the differences of the semi-gradient and the full-gradient method, several experiments are conducted. In this section, we give a detailed explanation of the explored setups and environments. All corresponding Python implementations can be found at our repository [14].

3.1 Function approximation methods

3.1.1 Linear function approximation

First we will explore semi-gradient SARSA with linear approximation functions. We look into this important special case of function approximation, since semi-gradient TD(0) in on-policy learning has been shown to converge to near a local minimum under linear approximation [4]. We wish to assess empirically whether we can find cases in which the convergence is impacted by using the full-gradient instead of the semi-gradient update rule. For SARSA, we implemented the semi-gradient algorithm as described in Reinforcement Learning for A.I.[1]. Since the original state features as given by our environments, discussed in the next section, are low-dimensional (2 and 4 dimensions respectively), we create a set of non-linear features using basis functions[11]. After discussing with our TA, we chose Radial Basis Functions (RBF) to create the interactions between variables, as they create smoothly varying features. For a more detailed overview of the implementation of Radial Basis Functions see appendix A.

3.1.2 Deep Q-network function approximation

Next, as their application is largely responsible for much of the recent high-profile RL breakthroughs [1], we will explore approximate Q-Learning using a Deep-Q-Network with non-linear approximation functions. We are interested in seeing the impact of full-gradient TD learning on the quality and convergence of learning in these high-complexity cases. For the Deep-Q-Network implementation, we make use of the PyTorch Python package, and adapted the code from lab 4 DQN of the Reinforcement Learning course[2]. We used a DQN consisting of two hidden layers, which consist of 24 and 48 neurons respectively. The two hidden layers are followed by a ReLU activation function. We made use of a Replay Memory of 1000 states for the CartPole environment and 20000 states for the MountainCar environment. The MountainCar replay memory needs to be larger to adjust for the larger number of steps made on average in this environment per episode. A more detailed description of the environments is given next.

3.2 Environments

Both these models will tested on the CartPole and Mountain Car problem from the OpenAIGym [7]. The first is a continuous control task in which the objective is survival and every action leads to a positive reward, as long as it does not trigger the stopping condition. In the second, moving through the environment incurs a negative reward, which only stops when a specific goal state in the environment is reached. We think the previously explained circular dependence causes a noisy training signal, which may drown out the "true" signal we want to learn from; the reward, and so we chose these two environments based on their difference in reward signal. In the MountainCar environment the agent will obtain many small negative rewards before ending the episode when a certain point in space is reached, we expect this to translate to a slower learning process in full-gradient learning when compared to semi-gradient learning, as this would mean that the impact of the noisy bootstrap signal may be overpowering the reward signal. Eventually we expect the agent to learn to move quickly toward the goal state, which would mean that the absolute values for the approximate expected value function for a state also diminishes, which in turn would reduce its influence as noisy signal on the gradient. Essentially, exploration is encouraged in this environment due to the negative rewards, which may cause noisy gradients for states that have not been visited before. After some trial runs, we noticed the DQN implementation had difficulty learning in the MountainCar environment. We suspected the aforementioned reward scheme may have caused this, and so we adapted the environment to better suit DQN learning. This is done by adding trajectories which reach the goal state more often to the replay memory and adding a positive reward upon reaching the goal state. The specifics of these alterations can be found in the Appendix C. The CartPole environment works with positive rewards. Here, exploration is discouraged, when compared to the MountainCar environment. Once a positive reward is obtained, weight updates will be made to encourage behaviour that lead to this reward. The differences in reward scheme make these environments two interesting applications for our experiments. We train both models using a semi-gradient method and a full-gradient method. The speed of convergence and the performance of the eventual policies of the two methods are compared. A more detailed overview of the environments' properties is shown in appendix B.

[1]Reinforcement Learning for A.I. Page 209, n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

[2]Reinforcement Learning - Lab 4: `https://canvas.uva.nl/courses/17399/assignments/178284`

	Linear	Deep-Q-Network
Full-gradient	Zero-vector	$\mathcal{N}(0, 0.1)$
	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.01)$
	$\mathcal{N}(0, 1)$	$\mathcal{N}(0, 0.001)$
Semi-gradient	Zero-vector	$\mathcal{N}(0, 0.1)$
	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.01)$
	$\mathcal{N}(0, 1)$	$\mathcal{N}(0, 0.001)$

Table 1: The different setups we experimented with, in each cell the used three weight initialisations are given for the combination of approximation method (columns) and weight update method (rows).

3.3 Hyperparameters

In Reinforcement Learning, the hyperparameters settings are a central part of the quality of a model. This means that doing a thorough grid search for the hyperparameters is an integral part of doing RL research [13]. For the linear approximation method we go through this process and document the optimal hyperparameters. In appendix D, a detailed exploration of different hyperparameter settings is given. For the MountainCar environment we find the optimal values for alpha and epsilon to be 0.005 and 0.1 respectively. For the CartPole environment we find values of 0.01 and 0.1 respectively. These values are used throughout the further linear approximation experiments. The hyperparameters in the linear approximation model are the learning rate - which scales the weight update of the gradient-, epsilon - which indicates the probability of taking a random action in each state-, and the weight initialisation which will be discussed separately in the next section. The value of epsilon determines the rate of exploration versus exploitation of the model's policy. Exploration is something we want early in the learning process, because we haven't yet learned accurate approximations for all states, while in the later stages of the learning process, we want more exploitation. Therefore epsilon is lowered linearly throughout the learning process from the starting epsilon value to 0. Lastly, the discount factor gamma is taken to be 1 in both of these environments, as ultimately both environments are capped at a maximum number of steps, which makes them both finite control problems.

For the DQN, although preferably we would do the same grid-search for hyperparameters as for the linear approximation method, training a model takes a long time, which makes doing a grid search computationally infeasible for the current project. Therefore we will use the settings found in the code from lab 4 DQN of the Reinforcement Learning course. This means we use a learning rate of 0.001 and a discount rate gamma of 0.8. To compare the differences between full- and semi-gradient, the same hyperparameters used in full-gradient experiments are used in the semi-gradient experiments.

3.4 Weight initialisation

Lastly, we look into the impact of different weight initialisation schemes on the convergence of full-gradient descent when compared to semi-gradient methods. Since the weight initialisation impacts the reward signal, and the reward signal dictates the direction of gradient updates, we think that weight initialisation greatly impacts the possible occurrence of divergence from the true value function. We attempt three different magnitudes weight initialisations, intuitively representing three levels of reward signal noise at the start of learning. For the linear approximation we use zero-vector, normally distributed with $\mathcal{N}(0, 0.01)$, and normally distributed with $\mathcal{N}(0, 1)$. For neural networks the number of weights is larger, thus the weight initialisation should be of lower magnitude [12]. Because of the non-linearities in a neural network, the weights should also be initialised non-zero. We therefore check weight nationalisations distributed by $\mathcal{N}(0, 0.1)$, $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0, 0.001)$.

Given that the weight initialisation is random, we will run these experiments 10 times, after which we check the statistical significance of the results by investigating the spread of the results. In short, besides the hyperparameter search described in appendix D, an overview of the experiments we run to investigate difference between full- and semi-gradient methods can be found in table 1.

3.5 Evaluation

We evaluate each experimental setup based on two criteria. First, their convergence speed to a policy, for which we will look at the point in the learning process at which the reward plateaus (in both environments the reward is synonymous to the number of steps before the end of an episode). Second, we look at the performance of each experimental setup, which is measured by the reward the agent is able to obtain for a given episode. As explained, each setup will be ran for 10 different seeds, which can be found at our github repository [14]. All resulting graphs shown in the next section show a mean over these 10 different runs, and a spread in variance over these different runs, encoding the robustness of the experimental setup to random noise.

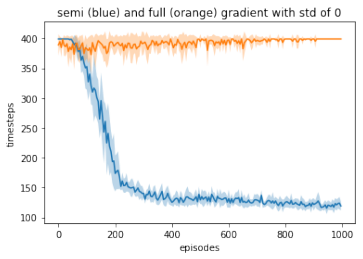
4 Results

4.1 Linear function approximation

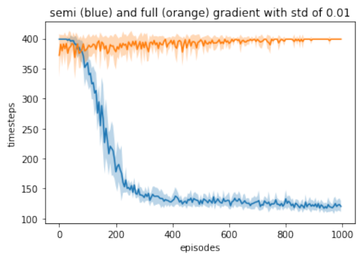
The results on the MountainCar problem for linear function approximation are shown in Figure 1. One can see that the semi-gradient method converges after approximately 200 episodes, whereas the full-gradient method diverges. Furthermore one can see that a higher standard deviation in the weight initialisation causes higher variance across the runs. Besides causing more variance, weight initialisation does not seem to have a significant impact on convergence. We also note that the variance over 10 runs, at least for zero-vector and $\mathcal{N}(0, 0.1)$ initialisation, is very small, which makes us confident these results are significant.

We were surprised that the full-gradient approach did not converge in the MountainCar environment. We suspected this may have to do with the value of the discount factor, so we investigated the effect of the discount factor on the performance of the linear approximation model using full gradient. It can be seen that with a discount factor of 0.99 instead of 1, the full-gradient model does learn to reach the goal. With a lower discount factor, the model shows a steadier learning curve, with less variance between episodes. Due to time constraints, we did not investigate the effect of a different discount factor on the semi-gradient implementation

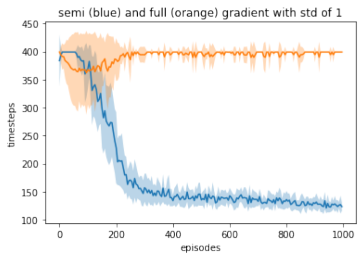
Next, the performance of the linear approximation method for the CartPole environment is shown in figure 2. We can see that across all weight initialisations, the full-gradient approach learns an effective policy more quickly, but the semi-gradient approach seems to converge to a slightly better performing policy for zero-vector weight initialisation and initialisation by $\mathcal{N}(0, 0.1)$. It should be noted that the variance across runs is quite high, but nevertheless we deem the difference between full- and semi-gradient on these weight initialisations to be so large that they should be considered significant results.



(a)

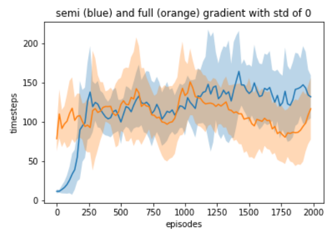


(b)

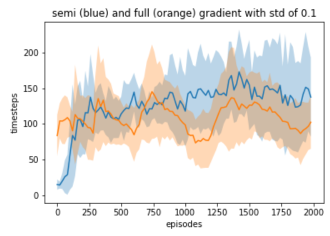


(c)

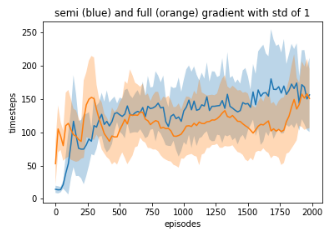
Figure 1: Performance of the Linear approximation on the MountainCar task for weight initialisation with different standard deviations averaged over 10 runs. The vertical axis displays the time steps it took to reach the goal for which a lower value is better.



(a)

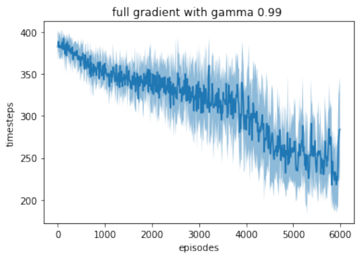


(b)

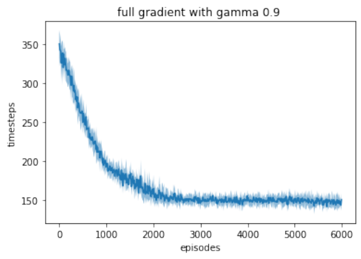


(c)

Figure 2: Performance of the Linear approximation on the CartPole task for weight initialisation with different standard deviations averaged over 10 runs. The vertical axis displays the time steps it took to reach the goal for which a higher value is better.



(a)

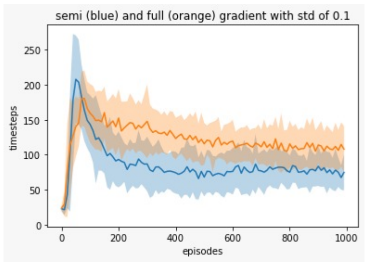


(b)

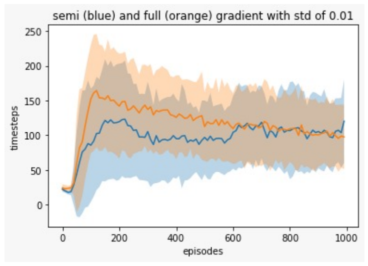
Figure 3: Convergence of the Linear approximation on the MountainCar task using a full gradient with different discount factors.

4.2 Deep-Q-Network function approximation

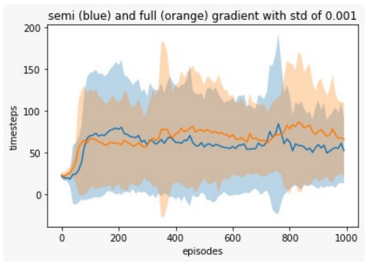
The results on the CartPole problem for function approximation with a Deep Q-network are shown in Figure 4. In figure 4 we stated three different plots, regarding the three different weight initialisations. Recall, that for the cartpole problem the goal is to maximise the amount of steps, so higher is better. In contrast to our predictions, one can see that overall the full-gradient model performed slightly better, except for the weight initialisation with a standard deviation of 0.01, where both models seem to converge to the same policy. For both models the variance in the learning process decreases, when the weights where initialised with a standard deviation closer to one. The performance and the convergence seems quite dependent on the weight initialisation for both models. When the weights are initialised with a standard deviation of 0.1, the full-gradient model converges to a better performing policy than the semi-gradient method. For smaller weight initialisations, the high variance shown as spread in the graphs makes us feel the difference between full- and semi-gradient methods are negligible.



(a)



(b)



(c)

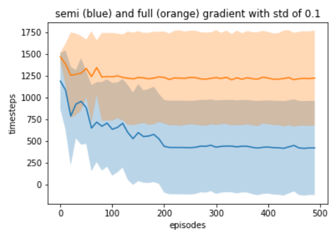
Figure 4: Convergence of the DQN on the CartPole task for weight initialisation with different standard deviations averaged over 10 runs. The vertical axis displays how many time steps the car kept up the pole for which a higher value is better.

The results of on the MountainCar problem for function approximation with a Deep Q-network are shown in figure 5. In figure 5 we show three different plots, corresponding to the three different weight initialisations. For the mountain car problem the goal is to minimise the amount of steps of reaching the goal, so here, a lower value is better. The maximum number of steps in these experiments is capped at 1500 steps. The results from this experiment seem to convey our prediction that the semi-gradient model would outperform full-gradient. In all setups, semi-gradient is able to reach better mean performance in terms of number of steps. We do note that these results show huge variances over different runs, especially for the semi-gradient method with weight initialisations 0.01 and 0.001. The full-gradient method for these intialisations seems stuck at around the maximum number of steps. After investigating, we notice that for individual runs, whether or not they converge highly depends on whether or not the goal state is found early on in the training process. We interpret these results in the following section.

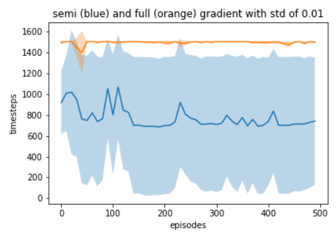
5 Discussion & Conclusion

5.1 Linear function approximation

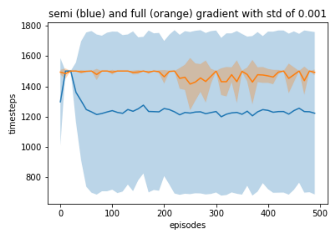
The results of the linear approximation make it seem like the difference between performance of the semi- and full-gradient methods is highly dependent on the environment. For the MountainCar problem, we can clearly see that the



(a)



(b)



(c)

Figure 5: Convergence of the DQN on the MountainCar task for weight initialisation with different standard deviations averaged over 10 runs. The vertical axis displays how many time steps the car kept up the pole for which a higher value is better.

full-gradient method is unable to learn, whereas the semi-gradient method converges nicely. We suspect this is due to the discount factor gamma of 1. This value of gamma does not pose a problem for semi-gradient learning, but in full-gradient learning this value has the consequence that both components of the TD-error - the target approximation and the approximation of the current state, as seen in equation 5 - contribute equally to the direction of the gradient used as weight update. Early on in the learning process, this introduces a lot of noise, as the gradient of the target does not have relevance to the problem yet due to the random weight initialisation. With a gamma value of 1, linear full-gradient learning in the MountainCar environment seems unable to overcome this noise. In semi-gradient learning the impact of this inaccurate target is less, as the approximation part of the target value now only contributes to the magnitude of the weight update, not the gradient direction in which the weights are updated. This suspicion is supported by the further experiments with other gamma values detailed in section 4.1. For lower discount rates, we saw that the full-gradient method can converge, which indicates that a higher difference between the target approximation and the approximation of the value for the current state results in a better gradient signal. However, we would think the CartPole environment would suffer from the same problems. Since for the CartPole environment the full-gradient approach does not seem to have such troubles, we are wary to give too much weight to this conclusion and suggest more environments should be tested to substantiate this claim. For the different weight initialisations, we see a higher deviation in the results but no change in the divergence of the two methods. This is contrary to what we expected to see, as we believed weight initialisation may very well impact the convergence speed due to larger noisy learning signals early on for larger magnitude weight initialisations. For the MountainCar problem, weight initialisation does seem to have impact on the variance within the learning process. This makes sense to us, as larger magnitude weight initialisations essentially entail the agent starts with more radically different value functions, which in turn may lead to larger differences in the learning process between different runs. For the the CartPole problem we can see that the full-gradient method initially does better but is quickly matched by the semi-gradient method. In the end, the semi-gradient method converges to higher values and the overall convergence seems more stable than the full-gradient method where the curve seems more teetering. This could be due to the circular dependence in the weight update of the full-gradient method, as it could cause model to optimise for the wrong objective, after which it recalibrates to the actual rewards and improves again.

5.2 Deep-Q-Network function approximation

The DQN results on the CartPole problem show that the full-gradient method performs better than the semi-gradient method. To us this result was interesting, as we expected the semi-gradient method to outperform the full-gradient method, both in convergence speed and ultimately in performance, due to the circular dependence discussed in section 2.2. We suspect this result may be attributable to the greater bias that is introduced in semi-gradient learning by making use of an update step that does not actually contain a full gradient of the function we're optimising; the TD-error. However, superiority in performance of the full-gradient method over the semi-gradient approach for the DQN are in contrast to the results obtained with linear function approximation. Our suspicion was that this difference is due to the fact that the DQN is able to encode more complex function approximations than the linear method. This may enable it to overcome the noise introduced early on by the gradient from the target approximation, and as the full-gradient approach ultimately is the more theoretically sound stochastic gradient descent method (the semi-gradient update step is not actually a gradient of anything), in the end it is able to perform better than its semi-gradient counterpart. However, in the MountainCar experiment, the semi-gradient approach seemed to outperform the full-gradient method. For weight initialisations with standard deviation 0.01 and 0.001, the full-gradient approach was, in many of the runs, unable to find the goal state even once. In investigating the results, we found that whether or not the DQN method converged to a sensible policy was highly dependent on whether or not the goal state was found early on in training. We now surmise that this may have had to do with our adjusting of the exploration-exploitation balance overtime by linearly decreasing the value of epsilon over the course of training. These findings are more in line with our expectations of the trade-off between full- and semi-gradient methods, as we expected the full-gradient approach to converge to an optimum more slowly than the semi-gradient approach. We think that in this particular experiment, the value of epsilon may have decreased too quickly for the full-gradient method to be able to find the goal state early on by exploring random actions, since we expected it to learn more slowly from selected actions than the semi-gradient approach. We would then also expect the semi-gradient approach to work better, as it would be faster to incorporate explored states into its approximation early on in training, and therefore is bound to explore more states. Since the MountainCar experiment has such high variance we do think we would need to run more experiments to support this hypothesis, as we will explain in section 5.3. For the DQN method, we see that weight initialisation again has an impact on the variance in the learning process over different runs. However, again opposite to in the linear method, here a smaller standard deviation seems to result in a more noisy training process. This is likely due to the fact that a DQN contains a series of weights intertwined with nonlinearities. For a weight initialisation that is too small, a neural network will not be able obtain gradients for the earlier layers that are useful to the learning process [12], as the signal-to-noise ratio is more likely to be too small to learn reliably.

In short, for linear function approximation methods, semi-gradient TD learning performs better than full-gradient TD learning, possibly due noise that is stripped from the gradient update early on by ignoring the gradient of the target approximation. In DQN function approximation, for the CartPole task the full-gradient approach actually outperforms the semi-gradient method in terms of performance, which we think may be attributable to the higher complexity value function approximations the DQN is able to make. However, because on the MountainCar environment the semi-gradient approach performed better, we would need to perform more experiments to investigate this suspicion. Weight initialisation does not seem to have a drastic impact on performance between full- and semi-gradient, but does seem to influence the amount of variance between different runs of the same model.

5.3 Limitations and future work

Now, we would like to briefly discuss some shortcomings of the current research project, and suggest a number of future extensions of the current project. First off, we would have liked to have done a hyperparameter grid search for the Deep-Q-Network as we acknowledge the importance of hyperparameter tuning in RL research, but as discussed, this was computationally unfeasible for the current project. We feel this shouldn't drastically impact the conclusions drawn from this research, as the main goal was to compare full- and semi-gradient methods, and we feel the best way to compare these methods is to use the same set of hyperparameters, regardless of whether they are the optimal hyperparameters for this problem or not.

For the linear approximation experiments ran on the MountainCar environment, we found out a bit late that a gamma value below 1 actually enables the full-gradient approach to learn a sensible policy. We would also have liked to run more experiments comparing full- and semi-gradient updates with these gamma values, as we think investigating the impact of the value of gamma on full-gradient learning may yield some interesting results. Intuitively, a lower value of gamma means a reduction in the contribution of the gradient of the target approximation to the full-gradient update. However, a lower value of gamma also means rewards further into the future are given less weight compared to rewards nearer in the future. If full-gradient methods require the gamma value to be lower than 1, would this also mean that they are intrinsically less suited for environments that require planning longer into the future? We think it would be very interesting to compare full-gradient methods with different gamma values in environments contrasting in deferment or delay in reward.

For the Deep-Q-Network approximation experiments, the experiments on the CartPole environment provided us with useful, and we think reliable, insights into the impact of weight initialisation on variance in the learning process. On the other hand, the experiments on the MountainCar environment caused us a lot of trouble. We had to make modifications to the environment's reward scheme and the DQNs replay memory for the model to be able to converge, as explained in appendix C. However, results shown in the previous section highlight that these modifications did not solve all problems with convergence. This can be seen in the huge spread in performance over different runs. As mentioned in the previous section, we think this may have had to do with the fact that we decrease epsilon over the duration of the training process. In future work we might explore a learning scheme where epsilon is only decreased once the goal state has been reached, in order to keep exploring further into the learning process. The high variance in results makes us wary to draw any hard conclusions on the trade-off between semi- and full-gradient methods from the MountainCar experiment's results. However, we do think that part of this variance is inherent to the MountainCar environment, where the agent has to perform quite an extensive series of actions to end up in a very specific goal state. This makes it hard to end up in the goal state purely by exploration. This variance in experimental results, arguably inherent to the MountainCar environment, makes it hard to draw reliable conclusions about the comparison we are actually interested in; the trade-off between full- and semi-gradient methods.

All in all, we ended up spending a lot more time on this project than we expected, mainly because we were interested in investigating a lot of the results obtained early on by further experiments. The results we ended up with in the end were quite surprising to us, as we did not expect the full-gradient method to perform well at all, but it held up in most experiments quite nicely. The results show that although semi-gradient TD methods are widely preferred over full-gradient methods, the full-gradient approach should not be written off so soon! We would be very interested in the impact of delayed rewards on the full-gradient methods in a possible future research project.

6 Appendix

A Radial Basis Functions details

To create features for the linear approximation method we used Radial Basis Functions. These are basis functions defined over a local area and transform the state space to a real value which is based on the input and some fixed point which is useful in the environments, as the state space is continuous but bounded over a finite area. We use Gaussian Radial Basis Functions, which are defined as:

$$\phi(r) = \exp(-\xi \cdot r) \tag{6}$$

Where ξ is the shape parameter of the underlying Gaussian, and r is the radial distance to the Basis Function center c; $r = c - x$. We project the observation space for either problem onto 16 RBFs, uniformly sampled over an interval between 0 and 1. Before projecting onto the feature space, the observations are normalized to fall between 0 and 1, to match the range over which the RBFs are defined.

B Environment properties

B.1 MountainCar

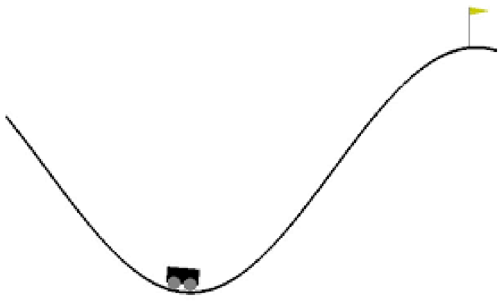
The MountainCar environment is shown in figure 6a. In the MountainCar environment a state consists of two observations; the car's position in the environment, ranging from -1.2 to 0.6, and the car's velocity ranging from -0.7 to 0.7. The car starts at a random position between -0.6 to -0.4 with 0 velocity. In each state, the agent can perform 1 of three actions; push left, no push and push right. For each action the agent performs that brings it to a non-goal state, it receives a reward of -1. The goal is to reach the state 0.6, which can only be achieved by oscillating between the far and near slope of the mountain.

B.2 CartPole

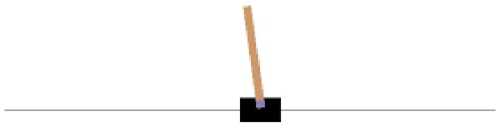
The CartPole environment is shown in figure 6b. Here a state consists of four observations; the position of the cart, the velocity of the cart, the angle of the pole and the velocity of the tip of the pole. At the start of an episode, all observations are assigned a random value between -0.5 and 0.5. At each state the agent can perform two actions; push the cart left, and push the cart right. Every action taken obtains a reward of 1. The episode is terminated when the pole angle is more than 12 degrees off center, the cart position is more than 2.4 off-center or the episode length is greater than 500. The goal is to master balancing the pole, in order to obtain the highest number of actions performed per episode.

C Deep-Q-Network MountainCar environment adaptation details

For the MountainCar environment, when using the Deep-Q-Network as described in section 3.1.2, we noticed that the network did not learn anything significant, and even after a long training time, the episodes ended only when the maximum number of steps was exceeded by the agent. We investigated and noticed that during all this training, not once did the agent reach the goal state. In the original specification of this environment by OpenAiGym only negative rewards can be incurred, and the goal, intuitively, is to stop obtaining negative rewards. This means that every time the agent reaches the maximum number of steps, this goal is reached, even though the agent may be in a state that is not the goal state. The agent may then learn to move to this non-goal state in a lower number of steps, in the same way as it would learn to move to the true goal state. However, when the agent then reaches this non-goal state before the maximum number of steps is reached, it will not stop incurring a negative reward. It will wander about some



(a)



(b)

Figure 6: Visualisations of the MountainCar environment (left) and the CartPole environment (right) obtained from the OpenAiGym Python implementation.

more, until the maximum number of steps is reached, after which it may consider its final state the goal state again. This would continue as training progresses and ultimately, the agent won't learn anything since, as far as it knows, every state could be considered the goal state. Only when reaching the actual goal state before the number of steps in an episode runs out, will the agent actually obtain a useful learning signal. To create a better distinction between the actual goal state, and a state reached when running out of steps, we introduced a positive reward of 100 for the goal state, and a negative reward of -10 of reaching a state when the maximum number of steps is reached. This lead to some improvement in performance, but training still did not converge for most random seeds. We suspected this may be due to the fact that for every time the agent reaches the goal state, only a single action with positive reward where the goal is actually reached is entered into the replay memory, whereas the number of . We therefore decided to duplicate the action that reached the goal state 100 times in replay memory, so that it gets sampled more often, and the agent thus more often receives a useful learning signal. The 50 actions leading up to the goal state were also duplicated 5 times into replay memory. After these changes for the MountainCar environment, learning, at least for the full-gradient method, drastically improved. These adaptations were used to obtain the results shown in section 4.

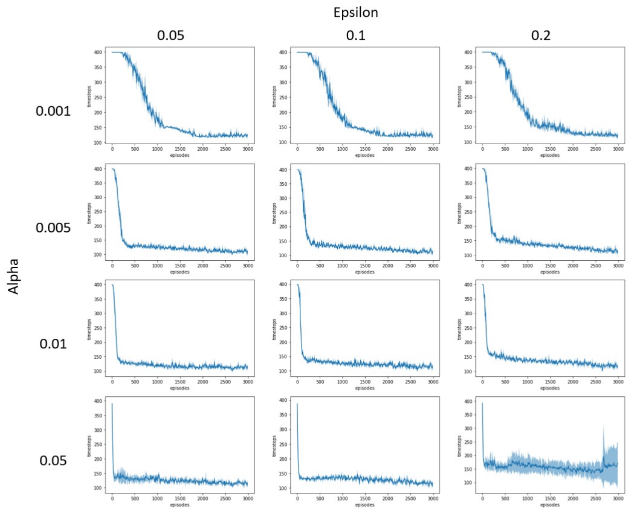
D Hyperparameter search results

As explained in section 3.3, we performed a hyperparameter search for the linear function approximation method, consisting of a search over different values of the learning rate alpha and the exploration factor epsilon, used in the epsilon-greedy action selection scheme. We experimented with a range of different epsilon values, $\{0.05, 0.1, 0.2\}$, and alpha values, $\{0.001, 0.005, 0.01, 0.05\}$. These values were selected after consulting examples found at OpenAiGyms website [3]. The search was performed using $\gamma = 0.99$, and the weights were initialised as zero-vectors. In figure 7a we show the results of the grid search over these hyperparameters on the MountainCar environment using the semi-gradient linear approximation method. We run each experiment for three different seeds, up to 3000 episodes. The results shown in the figures are the averages over these runs, along with their standard deviations. We can immediately see that a learning rate alpha of 0.001 causes slower convergence, whereas a learning rate that is too big (i.e. 0.05) causes a lot more variance. From these results we chose the learning rate alpha 0.005. An epsilon value of 0.2 causes a lot of variance, and surprisingly it seems like an epsilon value of 0.05 also causes more variance than an epsilon value of 0.1. seems optimal. We therefore chose hyperparameters alpha=0.005, epsilon=0.1 for for the linear approximation method in the MountainCar environment. The same hyperparameters were grid-searched over for the CartPole environment. Results from these experiments are shown in figure 7b. Here we can see that overall, results are a lot noisier. A learning rate of 0.05 leads to the highest variance, whereas a learning rate that is too low (0.001, 0.005) seems to result is a lower performance in convergence. We therefore pick a learning rate of 0.01. The optimal epsilon value to go along with this learning rate seems to be 0.1. For CartPole, we thus use alpha=0.01, epsilon=0.1.

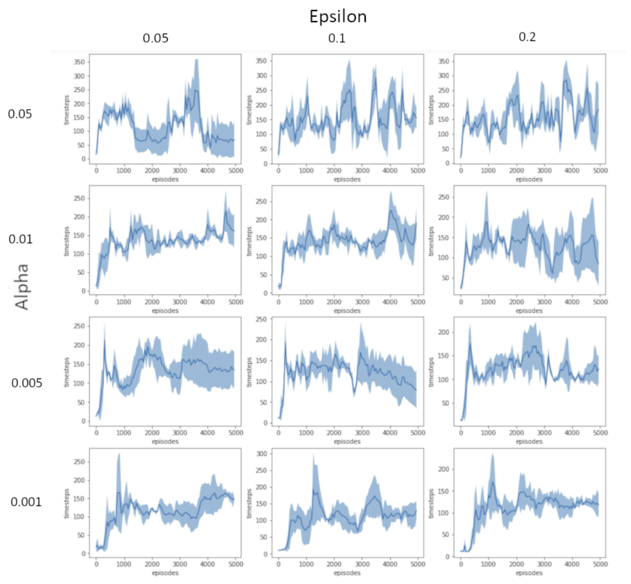
References

[1] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., ... & Osband, I. (2017). Deep q-learning from demonstrations. arXiv preprint arXiv:1704.03732.

[3] `https://github.com/openai/gym/tree/master/examples`



(a) Grid-search of hyperparameters alpha, epsilon on the MountainCar environment.



(b) Grid-search of hyperparameters alpha, epsilon on the CartPole environment.

Figure 7: Hyperparameter search results for the MountainCar and CartPole environments.

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

[3] Tesauro, G. (1992). Practical issues in temporal difference learning. In Advances in neural information processing systems (pp. 259-266).

[4] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[5] Barnald, E. (1993). Temporal-difference methods and markov model. IEEE Trans. Systems, Man and Cybernetics, 23, 357-365.

[6] Sutton, R. S., Szepesvári, C., & Maei, H. R. (2008). A convergent O (n) algorithm for off-policy temporal-difference learning with linear function approximation. Advances in neural information processing systems, 21(21), 1609-1616.

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[9] Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In Machine Learning Proceedings 1995 (pp. 30-37). Morgan Kaufmann.

[10] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Neural networks: Tricks of the trade (pp. 437-478). Springer, Berlin, Heidelberg.

[11] An, P. C. E. (1991). An improved multi-dimensional CMAC neural network: receptive field function and placement.

[12] Saxe, A. M., McClelland, J. L., & Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120.

[13] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017). Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560.

[14] Ten Kaate, P., Knigge, D., Wessels, D. & Le, D. (2020) Reproducibility project for Reinforcement Learning at the UvA - `Source Code` `https://github.com/paulltk/RL`.