Performance of Temporal Difference Reinforcement Learning Methods on a Grid World

Miles Epstein
miles.epstein@comcast.net

Brendan Woodward
brendanwoodward19@gmail.com

Abstract

Comparison of Reinforcement Learning algorithms is a well explored area of research and typically involves a limited number of algorithms. Such has been done on a simple grid world in the Hasselt (2010) study on Double Q-Learning, comparing it with Q-Learning. We extend on this and other research by comparing 11 different Temporal Difference learning algorithms on the grid world from Hasselt (2010). The algorithms tested are Q-Learning, SARSA, and R-Learning, along with variations of each of these algorithms. We find that algorithms that tend to underestimate the values of actions, such as Double Q-learning and Maxmin Q-learning, perform better in this particular grid world problem, with Maxmin Q-learning with large hyperparameter N performing best. We provide a detailed analysis of the performance of each algorithm explaining the reasoning for the results achieved.

1 Introduction

Reinforcement learning (RL) is a field of machine learning in which an *agent* begins in one *state* $s \in S$ within an *environment*. The agent repeatedly takes an *action* $a \in A$ based on its *policy* π. This results in a change of state and a *reward* R given to the agent. In general, the goal of reinforcement learning is for the agent to *learn* a policy for choosing actions that will maximize the reward.

Within reinforcement learning, a class of methods is Temporal-Difference (TD) methods, which are value-based (i.e. the agent learns *values* of each state/action, but no model is made, making it *model-free*) and used for Markov Decision Processes (MDPs). An MDP can formally define an RL environment: states S; actions A; $P_a(s, s')$, the probability that, when in state s action a leads to state s'; $R_a(s, s')$, the reward given when taking action a to move from state s to s'. TD methods maintain a table (the Q table in Q-learning) estimating the value of taking an action in a given state; this table is updated as the agent takes actions and explores the environment. The exact method by which the algorithm performs these updates depends on the implementation (see Section 3.2).

Grid world is a commonly-used environment for RL in which the agent moves about a grid one square at a time, receiving rewards, until it reaches the goal square and the simulation terminates. In this paper, we compare and analyze many of the most popular value-based TD algorithms on the Grid World from Hasselt (2010), which is designed to lead agents that overestimate values to make sub-optimal choices (see Section 3.1). We wish to do this to better understand the strengths and weaknesses of these algorithms and their variations. We train 11 TD methods, only two of which have been trained on this environment in past published work. This analysis will prove useful in selecting TD methods for future research and work, more specifically when using environments that punish overestimation of values.

2 Related Work

2.1 "Double Q-learning", Hasselt (2010)

Van Hasselt developed Double Q-learning, an off-policy TD method that differs from standard Q-learning in that it maintains two Q-tables, each updated using values from the other during training.

Course project for UBC CPSC 440/540, April 2023.

Van Hasselt trained the algorithms on both roulette and a simple grid world and found that Q-learning tends to overestimate action values while Double Q-learning tends to underestimate action values in both environments. In these examples, this led to Double Q-learning learning a closer-to-optimal policy. Since this research was focused on displaying Double Q-learning, it differs from our project, as we will be comparing many more algorithms and looking at more than the over and under-estimation biases.

2.2 "Maxmin Q-learning: Controlling the Estimation Bias of Q-learning", Lan et al. (2020)

As standard Q-learning has an overestimation bias of action values and Hasselt (2010)'s Double Q-learning has an underestimation bias, Lan et al. developed a generalization of Q-learning to find a balance between these methods. Their algorithm, Maxmin Q-learning, has a hyperparameter N, for which $N = 1$ means the agent is simply Q-learning and larger N increases underestimation bias, bringing the algorithm closer to (and beyond) Double Q-learning. Because of this, there is always a choice of N that leads to (approximately) unbiased action estimation. Additionally, the authors found that an overestimation bias leads algorithms to perform better in some environments while an underestimation bias leads algorithms to perform better in other environments. For instance, Hasselt (2010)'s grid world is the latter. Rather than trying a small number of algorithms in different environments, we will be showing a larger number of algorithms in a single environment.

2.3 "Average reward reinforcement learning: foundations, algorithms, and empirical results", Mahadevan (1996)

In this paper, Mahadevan explores a variety of average reward RL algorithms and approaches, beginning with some simpler models like dynamic programming and analyzing gain-optimal and bias-optimal policies on MDPs. In addition, Mahadevan expands on the research done by Schwartz (1993) on R-learning. They do this through applying the algorithm to larger and more realistic problems, specifically, a grid world and a robot environment. The results indicate that R-learning is able to outperform Q-learning in these environments, but that R-learning typically has a slower convergence rate. Furthermore, sensitivity analysis was performed, which indicated that a higher exploration parameter tended to perform better and avoided limit cycles. In our paper, we hope to extend further onto this by comparing R-learning with more algorithms on our grid world, as well as comparing it with RQ-learning, which combines both R-learning and Q-learning.

3 Methods

3.1 The Environment: van Hasselt's Grid World

Our environment is the same one used in Hasselt (2010): a 3 by 3 grid world with no movement restrictions, the start state in the corner, and the reward state (goal) in the opposite corner. The reward for reaching the goal is +5 and moving to any other state gives a reward of either -12 or +10, randomly selected with equal probability. The optimal strategy in this environment is to make it to move directly toward the goal (taking 5 steps). The optimal reward per step is +0.2, and the "correct" maximum Q-value possible from the start state is 0.36 (Hasselt 2010).

3.2 The Temporal Difference Algorithms

Temporal Difference methods are a class of tabular Reinforcement Learning methods. They combine the approaches of Dynamic Programming Planning and Monte Carlo RL methods. This approach has been fundamental in RL research and created some of the most popular types of algorithms studied today. Within our study, we used the following Temporal Difference methods:

3.2.1 Q-Learning

Q-learning is one of the most well-known TD algorithms and uses updates of the Q-value estimates to learn the optimal action-value function. Q-learning is an *off-policy* method, meaning that each update can use data collected at any point of training regardless of how the agent explores the environment or when the data was collected. It finds an optimal policy through maximizing the expected value of the total rewards over all steps starting from the current state. (Sutton and Barton 2020)

3.2.2 Double Q-Learning

Double Q-learning is a variant of the original Q-learning algorithm that aims to fix Q-learning's tendency to overestimate action values. To accomplish this, Double Q-learning uses two sets of Q-values to estimate the action-value function, which allows for a more unbiased approximation -

and thus reducing the overestimating problem. However, this algorithm can underestimate in some situations, having the opposite issues as original Q-learning. (Hasselt 2010)

3.2.3 Maxmin Q-Learning

The Maxmin Q-learning variant of Q-learning uses both the maximum and minimum Q-values to make the update for the next state. This change also helps combat the overestimation issue seen in Q-learning by also considering the worst-case situation. To help balance over- and under-estimation it uses a parameter N referring to the number of Q tables to use. (Lan et al. 2020)

3.2.4 Weighted Q-Learning

Weighted Q-learning is a another variant on Q-learning that assigns a weight to each action according to its estimated value. From this, it will find a policy by select actions with respect to the weighted sum of the Q-values. Using this approach for selecting actions, Weighted Q-learning has an improved convergence rate and stability. (D'Eramo, Restelli, and Nuara 2016)

3.2.5 Speedy Q-Learning

The Speedy Q-learning variant uses a momentum term when updating the Q-values to accelerate the learning process. This decreases the fluctuations in updates and smooths out the Q-value estimates, leading to faster convergence and typically better overall performance. (Ghavamzadeh et al. 2011)

3.2.6 Q-λ

Q-λ is a variant of Q-learning that uses eligibility traces and temporal differences to update the optimal action-value function. Eligibility traces help represent how responsible each state-action pair is for a given reward at a step, adding decay for past values to make older states less influential. This implementation allows for improved convergence rates and greater stability. (Sutton and Barton 2020)

3.2.7 SARSA

SARSA is another very popular RL algorithm that is discussed alongside as Q-learning as one of the staples within RL. This is an *on-policy* method, meaning that it can only use data collected while acting from the most recent version of the policy. SARSA selects actions based on the current policy and then will update the Q-values in accordance. (Sutton and Barton 2020)

3.2.8 SARSA-λ

SARSA-λ, like Q-λ, utilizes eligibility traces and temporal difference updates to estimate the action-value function. This provides improved convergence rates and stability. (Sutton and Barton 2020)

3.2.9 Expected SARSA

Expected SARSA, unlike traditional SARSA, is an off-policy method. It uses the current policy as well as the expected Q-values to update the Q-values. This allows it to handle stochastic policies and exploration. (Sutton and Barton 2020)

3.2.10 R-Learning

R-learning is another on-policy RL method that learns the expected reward function as opposed to the action-value function. Consequently, it selects actions based on their expected rewards, ignoring the Q-value estimates. This allows it to handle delayed rewards and sparse feedback. (Schwartz 1993)

3.2.11 RQ-Learning

RQ-learning combines both R-learning and Q-learning. This method estimates the optimal action-value function and expected reward function at the same time. This is a further improvement on the convergence and stability of R-learning with the introduction of Q-value estimates. (Tateo et al. 2017)

3.3 Our Approach

We train each of the methods in Section 3.2 on Hasselt (2010)'s grid world. We begin with a reproduction of Hasselt's work, available as a MushroomRL (D'Eramo et al. 2020) tutorial here. The grid world is as described in Section 3.1 and is available here.

The tutorial script trains each of Q-learning (3.2.1), Double Q-learning (3.2.2), weighted Q-learning (3.2.4), Speedy Q-learning (3.2.5), and SARSA (3.2.7) on the above environment for 10000 steps

(one step per fit) with an exponentially decaying learning rate (exponent of 0.8), an epsilon-greedy policy with an exponentially decaying epsilon (exponent of 0.5), and 10000 experiments for each algorithm. We build on the script, training Maxmin Q-learning (3.2.3; $N \in [1, 2, 3, 4, 6, 8, 10, 12]$), Q-$\lambda$ (3.2.6, $\lambda \in [0.1, 0.2, ...0.9]$), SARSA-$\lambda$ (3.2.8; $\lambda \in [0.1, 0.2, ...0.9]$), expected SARSA (3.2.9), R-learning (3.2.10; $\beta \in [0.1, 0.2, ...0.9]$), and RQ-learning (3.2.11; $\beta \in [0.1, 0.2, ...0.9]$) under the same conditions. All algorithms used are available here.

First, we identified hyperparameter choices that led to the greatest rewards per step (or, if all choices for hyperparameters led to similar performance, the hyperparameter that led to the most accurate action value). These were: $\lambda = 0.1$ for Q-λ and SARSA-λ, $\beta = 0.9$ for R-learning, $\beta = 0.3$ for RQ-learning, and $N = 12$ for Maxmin Q-learning.

Then, to compare methods, we plot average rewards per step and the maximum action (Q) value available from the starting state at each step for each algorithm (using the optimal choices of hyperparameters above). This allows us to compare performance: methods that reach greater rewards per step (closer to the maximum of +0.2) are desired; the "correct" maximum action value from the starting state is 0.36 (Hasselt 2010), we can see which algorithms over-estimate or under-estimate this value. We compare performance between algorithms and for different hyper-parameter values for each algorithm. Code used to perform training and analysis is linked in Appendix A.

4 Results

4.1 Performance of All Algorithms

First, we observed the rewards per time step over time of each algorithm, which were as follows:

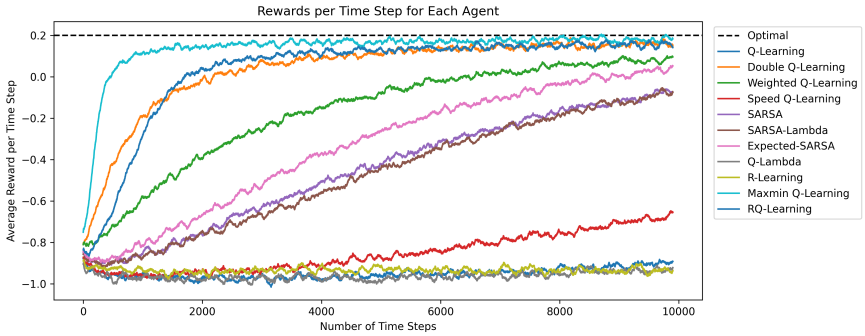

Figure 1: Average rewards per step over all steps for each algorithm. Optimal reward per step is 0.2.

The optimal reward per step in the van Hasselt Grid World is +0.2. Most algorithms show signs of convergence or progress toward this value over 10000 steps, although no algorithm reached this point. We will consider the top algorithms to be those which averaged a positive reward after the fewest steps. These algorithms include: Double Q-learning, Maxmin Q-learning, RQ-learning, and Weighted Q-learning. As with the Hasselt (2010) paper, it is no surprise that Double Q-learning performs well here. Similarly, since Maxmin Q-learning also is a variant of Q-learning that specifically attempts to fix over-estimation, it also exceeds here and was the fastest of the positive algorithms to converge. Weighted Q-learning also performed well, as it averages over all possible action values, limiting the overestimation associated with taking the maximum action value. RQ-learning is the last algorithm with positive average rewards after 10000 steps. This is surprising, as it combines both R-learning and Q-learning which were some of the worst performing algorithms on their own. Yet, taking advantage of an expected reward and action-value function allowed it to counterbalance the estimates made by each of the individual components. The best performing methods are all variants of Q-learning, many of which were designed to help in the overestimation problem along with improved convergence and stability.

Although none of the SARSA algorithms reached positive rewards after 10000 steps, they fared better than the remaining algorithms and Expected SARSA was very close to Weighted Q-learning. The other algorithms (Q-learning, Speedy Q-learning, Q-lambda, and R-learning) all performed relatively poorly, with R-learning showing no signs of convergence toward positive rewards.

Next, we compare the maximum Q-value from the starting state learned by each algorithm.

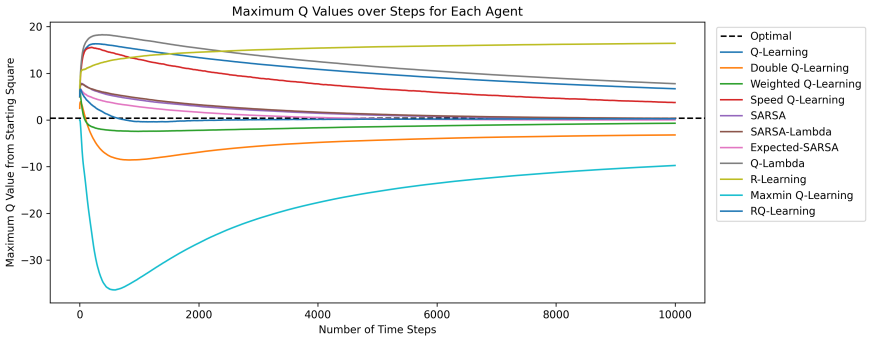

Figure 2: Maximum Q-value for action taken from starting square, for each algorithm over all steps. "Correct" maximum Q-value is 0.36.

R-learning performed the worst, with Q-values converging toward a positive value far from the "correct" Q-value. This may explain its lack of convergence toward the the optimal rewards: the agent incorrectly learned that moving to a non-goal square has a large positive reward, and thus learned to not move toward the goal square. This is consistent with the average reward per time step of -1. All other algorithms appear to be converging toward the correct Q-value, albeit at different rates.

The algorithms closest to the correct value were all of the SARSA algorithms. With moderate performance on average rewards, this was an unexpected result as we did not identify any characteristics of SARSA that would lead to this result. In addition, RQ-learning and Weighted Q-learning slightly underestimated the value, then Maxmin Q-learning and Double Q-learning further below these algorithms. Both Maxmin Q-learning and Double Q-learning are attempts to directly solve the overestimation problem, and as a result they tend to underestimate the Q-values.

Furthermore, **the agents that learn lower Q-values have greater rewards per step**, even if the learned Q-values are not necessarily accurate as quickly. These algorithms are RQ-learning, Maxmin Q-learning, Double Q-learning, and Weighted Q-learning. On our grid world, the optimal set of actions is to move directly toward the goal. However, because of the random nature of rewards, agents may wrongly learn other actions. Underestimating values associated with moving to a non-goal square will make the agent less likely to prefer those squares and thus more likely to make the correct action, even though values are underestimated. This advantage gained from underestimation will not necessarily apply in any environment.

4.2 Maxmin Q-Learning

Additonally, we can compare the rewards per time step over time for Maxmin Q-learning with different choices of N. This is worth investigating because Maxmin Q-learning performed among the best of all algorithms and has a hyperparameter that can be adjusted to underestimate action values, which we believe improves performance in this grid world.

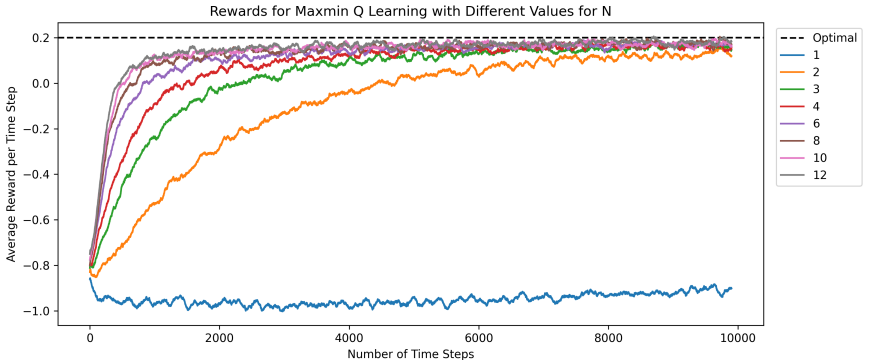

Figure 3: Average rewards per step over all steps for different choices of N in Maxmin Q-learning. Optimal reward per step is 0.2

When $N = 1$, Maxmin Q-learning is simply Q-learning, so the rewards per step match those of Q-learning above. Then, increasing N leads to a greater reward per step. In fact, a reward per step near optimal is achieved in approximately 1000 steps for large N, significantly faster convergence than even Double Q-learning. Finally, we compare the maximum Q-value from the starting state learned by Maxmin Q-learning with different values of N:

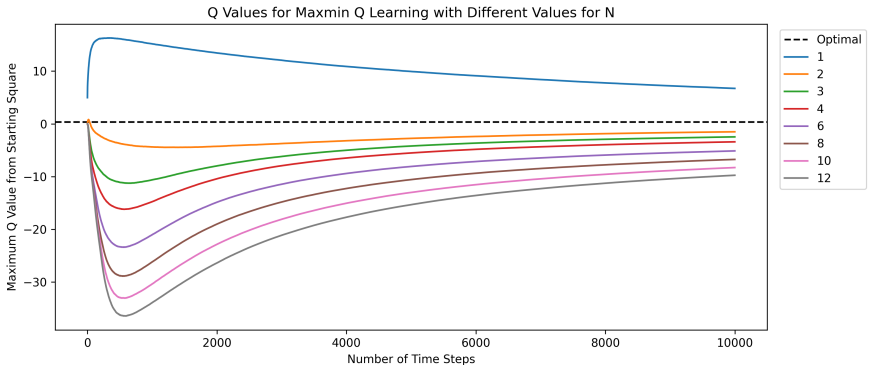

Figure 4: Maximum Q-value for action taken from starting square, for different choices of N in Maxmin Q-learning. "Correct" maximum Q-value is 0.36.

As expected, greater N leads to greater underestimation of action values. Again, algorithms with greater underestimation have greater rewards per step, which is reasonable given the above discussion. In fact, this greater rewards are achieved despite the learned Q-values getting further from the "correct" value with larger N. In a way, we are overfitting: with greater N, we force the agent to learn the correct path without learning the underlying reasons for that path (action values) as accurately.

5 Discussion and Future Work

5.1 Discussion of Overall Results

In short, because of the way our grid world is designed, algorithms that underestimate action values obtain greater rewards. Because of this, Double Q-learning and Maxmin Q-learning (with large N), both designed to underestimate action values, obtain near-optimal rewards the fastest.

5.2 Strengths and Weaknesses

A strength of this research is that a wide variety of methods are compared, allowing us to observe properties of each method and how they compare to each other. A limitation is that algorithms are only trained in one environment, limiting the generalizability of our results to all problems. Still, the results can reasonably be generalized to environments in which underestimating action values is optimal.

5.3 Direction for Future Work

For future research projects, exploring the performance of non-TD methods in this environment could help compare the performance of those types of algorithms with the best TD methods to see what solving method is typically better in this environment. Extending the approach to add deep models would also be an interesting analysis to see how much better a deep model does in comparison to some of the algorithms here, which use no neural networks. As the SARSA algorithms managed to get the most "correct" max Q-values, more exploration could be done as to why SARSA did so well in this regard and why it didn't lead to better performance in average rewards. Additionally, we could generalize our results by running the algorithms used on different grid worlds or entirely new environments and observing the relative performance of algorithms in those environments, which may be different from our results. Additionally, simply placing the agents at different starting points and seeing if they can find the optimal policy would also check for the generalization of these methods.

References

D'Eramo, Carlo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters (Jan. 2020). "MushroomRL: Simplifying Reinforcement Learning Research."

D'Eramo, Carlo, Marcello Restelli, and Alessandro Nuara (20–22 Jun 2016). "Estimating Maximum Expected Value through Gaussian Approximation." *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1032–1040. URL: https://proceedings.mlr.press/v48/deramo16.html.

Ghavamzadeh, Mohammad, Hilbert Kappen, Mohammad Azar, and Rémi Munos (2011). "Speedy Q-Learning." *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger. Vol. 24. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/ab1a4d0dd4d48a2ba1077c4494791306-Paper.pdf.

Hasselt, Hado van (2010). "Double Q-learning." *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

Lan, Qingfeng, Yangchen Pan, Alona Fyshe, and Martha White (Feb. 2020). "Maxmin Q-learning: Controlling the Estimation Bias of Q-learning."

Mahadevan, Sridhar (1996). "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results." *Machine Learning*. Ed. by Leslie Pack Kaelbling. Vol. 22. URL: https://doi.org/10.1007/BF00114727.

Schwartz, Anton (1993). "A Reinforcement Learning Method for Maximizing Undiscounted Rewards." *Machine Learning, Proceedings of the Tenth International Conference*. URL: 10.1016/B978-1-55860-307-3.50045-9.

Sutton, Richard and Andrew Barton (2020). *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: The MIT Press.

Tateo, Davide, Carlo D'Eramo, Alessandro Nuara, Marcello Restelli, and Andrea Bonarini (Nov. 2017). "Exploiting structure and uncertainty of Bellman updates in Markov decision processes," pp. 1–8. DOI: 10.1109/SSCI.2017.8280923.

A Appendix: Code

GitHub repository for our code: https://github.com/milesepstein13/grid_world