ASSIGNMENT 4: N-STEP TEMPORAL DIFFERENCE LEARNING

Maksym Lytovka
s3705609

Nataliia Bagan
s3648885

1 INTRODUCTION

This is a report for Assignment 4 of the Reinforcement Learning course. We investigate n-step Temporal Difference (TD) learning in the context of an undiscounted task on $12 \times 12 (= S)$ grid environment with cliffs (Figure 1), similar to the one on p.132 of Sutton & Barto (2018). The episode begins in one of the 2 start positions (blue) with equal probability (environment with deterministic transitions, but random starts) and ends either when the agent reaches the goal cell (green) or when 100 timesteps have passed. Every move (among action set $A = \{\text{up, right, down, left}\}$) brings rewards of -1, except cliff cells (red), which result in a reward of -100 and a respawn in one of the starting points. In sections 2-4 we consider Q-learning, SARSA, and Expected SARSA n-step TD agents. We investigate which values of the learning rate α and a number of steps forward n perform the best in the considered environment. In section 5 we consider the addition of a downward wind to the environment: at each action, with 50% probability, the agent moves to the cell below the intended one. In section 6 we compare algorithms with their best hyperparameters between each other.

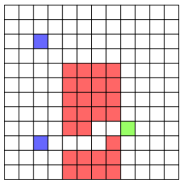

Figure 0: Cliff Walking environment

2 Q-LEARNING

2.1 METHODOLOGY

Temporal Difference learning is usually applied to the cases of the off-policy learning, where we have the policy we are using to make the moves (behavior policy), and the policy to evaluate the value of the next state (target policy). That second policy is the one that makes all the difference by making an imaginary next move and defining updates of the values based on it. We start with the simplest candidate from the TD family of models: Q-learning, where the target policy is a greedy policy. As for the behaviour policy, we use ϵ-greedy in all the experiments. Moreover, as we consider the TD version of the Q-learning, we also have another hyperparameter: the number of steps we push the update back (denoted as n)[1]. Thus, the implementation of a single experiment looks as follows:

1. Initialize action values to 0: $\forall s \in S, a \in A : Q(s, a) = 0$
2. Select an action based on ϵ-greedy policy with $\epsilon = 0.1$:

$$\pi_{\epsilon-greedy}(a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, & \text{if } a = \arg\max_{b \in A} Q(b) \\ \frac{\epsilon}{|A|}, & \text{otherwise} \end{cases} \tag{1}$$

3. Update corresponding action value n steps back basing on the Q-learning rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G - Q(s_t, a_t)) \tag{2}$$

, where

$$G = (\sum_{i=1}^{n} \gamma^{n-i} R_{t-n+2}) + \gamma^n \max_a Q(s_{t+1}, a) \tag{3}$$

So, after every step, when we arrive at the state, we "hallucinate" another step forward with the greedy policy ($\max_a Q(s_{t+1}, a)$ term). Then, we update the value of the state n steps back using

[1] For more detailed information about the algorithms refer to Sutton & Barto (2018)

the equation 6. To get the new state estimate (G), we sum over different previous rewards and corresponding discount factors, and sum it up with the "hallucinated" term. The γ constant equals 1 in all our experiments in the present report because in this task we estimate the whole return as a pure undiscounted sum of immediate rewards. Therefore, it may be disregarded in the formula.
To analyze the performance, we run $100 repetitions$ of each agent with every value of considered hyperparameters: α and n. Each repetition consists of 1000 episodes, each bounded by 100 steps. We use $\epsilon = 0.1$. We then plot 2 graphs: one comparing the overall performances of each pair of parameters based on Area Under the (learning) Curve (AUC); and the other one showing learning curves of each n value, along with its best alpha (based on AUC computed previously).

2.2 RESULTS

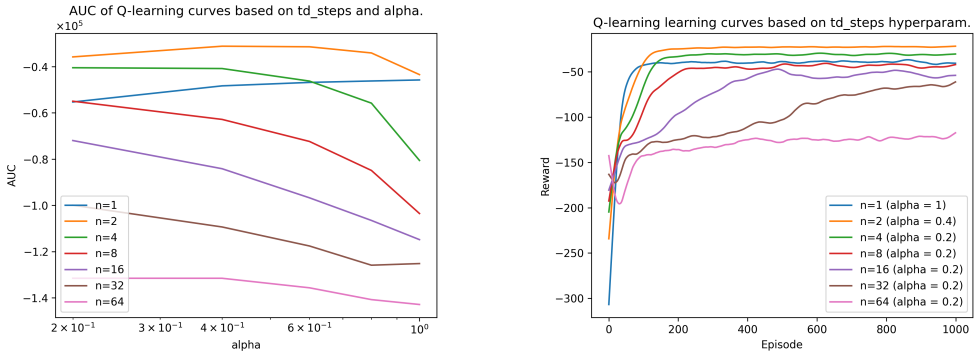

Figure 1: AUC of the learning curves for varying n and alpha (left) and learning curves for varying n (with best alphas based on AUC) of Q-learning TD agent. Averaged over 100 runs, each consisting of 1000 episodes. The learning curves are smoothed with a smoothing window of 31. Controlled n are (1,2,4,8,16,32,64) and alpha are (0.2,0.4,0.6,0.8,1). $n = 2$ has the highest AUC and learning curve, with $n = 1, 4$ being slightly worse. Further graphs are going downwards as n rises.

From the figures above, we can clearly see that n=2 is the best parameter: it both has a higher AUC for all of the considered learning rates and has a higher learning curve. Overall, we can see that in the first episode, curves are sorted as values of n (higher return at higher n), in the next ≈50 episodes they become sorted in the exactly opposite order (higher return at lower n), and finally for all other episodes, intermediate values (n=2,4) lead. Let us approach this from a theoretical perspective.
In the first episode, the bigger the n, the longer the update formula is, and thus all the updated values gain more information from the immediate rewards. At the beginning, it is an advantage, because the initial values jump from the very negative ones faster, but on the further initial episodes more precision is needed. Algorithms with bigger n bootstrap using the values further away, which are poorly estimated, partially because they were experienced fewer times.
In all of the following episodes, however, values of the terminate state propagate back faster, and looking at the states further away becomes beneficial, as they are precise enough. Therefore, algorithms looking somewhat further (i.e. n=2 and n=4), outperform 1-step TD(0), because they consider states closer to the terminate one. However, we see that for large n, the far values are not updated enough times and are imprecise. And because they are used often, the overall learning is slow. Besides that, algorithms with bigger `td_steps` value update their values less in general. For instance, if we take n=64, the agent only starts updating its values on the 64th timestep, thus resulting in a maximum of 37 updates.
All the previous reasoning applies to n-step TD algorithms in general, including the ones considered in the next sections. It will not be included there, but it is a valid discussion and its confirmation can be seen in Figures 2 and 3. Regarding AUC, we see the same trend of intermediate `td_step` values being the optimal ones. We also see that for the best value $n = 2$, intermediate alphas are the best, while for $n = 1$ higher values are better, and for $n \geq 4$ lower values. So we can argue that the derivative of the curve approximating the broken lines increases as n increases. This could be shown with higher resolution of alpha values (an extension, suggested in 8.1). Finally, we can notice that optimal alpha values decrease when n increases. This will be explained in detail in section 4.2.

3 SARSA

3.1 METHODOLOGY

The second algorithm we consider is SARSA. As discussed above (2.1), in the TD-like algorithms, there are two policies, and the second one is the one of interest. In the SARSA algorithms, the second policy is the same as the first one (i.e. behavior policy = target policy), and that is the main trick. So the imaginary step becomes the one we would do if we had to make a step during the exploration and not planning "in our head". We may see it easily in step 3 of the algorithm:

1. Initialize action values to 0: $\forall s \in S, a \in A : Q(s, a) = 0$
2. Select an action based on ϵ-greedy policy, with $\epsilon = 0.1$ (formula 1 from previous section).
3. Update the corresponding action value based on the SARSA rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G - Q(s_t, a_t)) \tag{4}$$

, where

$$G = (\sum_{i=1}^{n} \gamma^{n-i} R_{t-n+2}) + \gamma^n Q(s_{t+1}, a_{t+1}) \tag{5}$$

If you pay attention to the highlighted part, you will notice, that that is the only thing that changed from the Q-learning approach. Now we use not the simple greedy policy, but we "bootstrap" Sutton & Barto (2018) and use estimates from the same policy we used for making the steps.
The experiment is conducted using the same methodology as in the previous section.

3.2 RESULTS

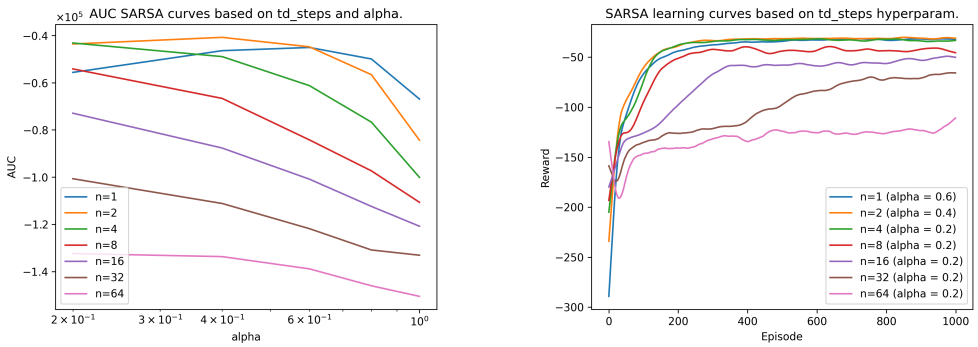

Figure 2: AUC of the learning curves for varying n and alpha (left) and learning curves for varying n (with best alphas based on AUC) of TD SARSA agent. Averaged over 100 runs, each consisting of 1000 episodes. The learning curves are smoothed with a smoothing window of 31. Controlled n are (1,2,4,8,16,32,64) and alpha are (0.2,0.4,0.6,0.8,1). n=1 and n=2 have the best intersecting AUC lines, but n=2 has the maximum value overall. Learning curves are best fro $n = 2$ and $n = 4$, with $n = 1$ being slightly worse, and further n significantly worse.

The results are more mixed than of the previous agent, meaning that there is no clear distinction of which n value is the best. This might be explained by the fact that SARSA is an on-policy method, opposite to Q-learning, which maximizes over actions, resulting in more explicit results (possibly influenced by maximisation bias). However, we can still see that $n = 2$ performs best, with $n = 1$ and 4 having slightly worse performances, and $n \geq 8$ being significantly worse, as in the previous section. Even though broken lines intersect in Figure 2(left), the overall best value belongs to $n = 2$, although for some alphas it is worse than $n = 1$.
In Figure 2(right), we can also see that its learning curve is located the highest on average, while larger n, especially $n \geq 16$ have very slow increase trend and still do not reach the asymptote of other curves. Moreover, we can again see decreasing trend in optimal alpha values for larger n (reasoning about in the next section). All in all, while same reasoning regarding TD algorithms applies, SARSA shows more similarity of the graph lines, partially because it is an on-policy method.

4 EXPECTED SARSA

4.1 METHODOLOGY

Finally, we look at the last algorithm in the present work: Expected SARSA. It makes one step further from the simple SARSA by not just taking the step we would take and getting the Q value from that, but by evaluating the expected value of Q (such as we evaluate the expected reward). It is nice to see it in the formula of the update step:

1. Initialize action values to 0: $\forall s \in S, a \in A : Q(s, a) = 0$
2. Select an action based on ϵ-greedy policy, with $\epsilon = 0.1$ (equation 1 from section 2.1).
3. Update the corresponding action value based on the Expected SARSA rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G - Q(s_t, a_t)) \tag{6}$$

, where

$$G = (\sum_{i=1}^{n} \gamma^{n-i} R_{t-n+2}) + \gamma^n \sum_a p(a|s_{t+1}) Q(s_{t+1}, a) \tag{7}$$

And again, the difference with the previous algorithm lies only in the highlighted part. The new Q-value we update with is the average outcome over all actions possible in the state(so the sum of the Q-values, multiplied by the probability of getting that Q-value). Such an approach allows the agent to consider all possible scenarios even more thoughtfully, than in 3.1. Imagine the case of having four actions with the same probabilities of occurrence, where one of them is extremely rewarding and all others are quite poor. Expected SARSA will "smooth out" the Q-value, meanwhile, plain SARSA will follow the original policy we are using for making steps (which is usually greedy or ϵ-greedy) and will be biased towards that highly rewarding action. We conduct the same experiment and analysis that we have done in 3.1.

4.2 RESULTS

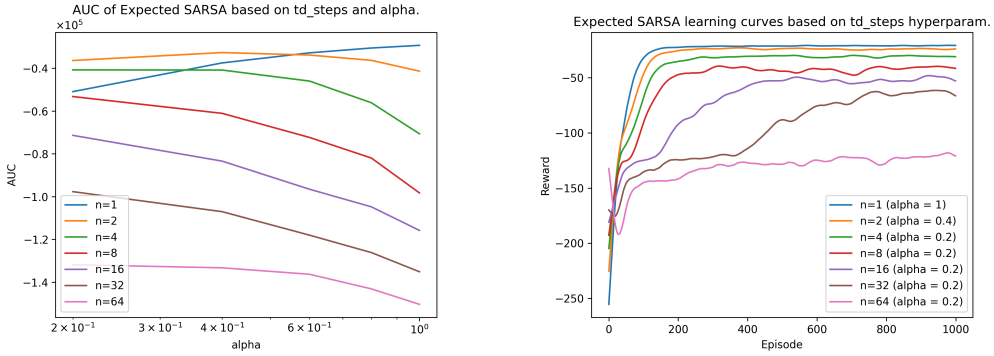

Figure 3: AUC of the learning curves for varying n and alpha (left) and learning curves for varying n (with best alphas based on AUC) of Expected SARSA TD agent. Averaged over 100 runs, each consisting of 1000 episodes; smoothing window is 31. Controlled are: n (1,2,4,8,16,32,64) and alpha (0.2,0.4,0.6,0.8,1). $n = 1$ has highest AUC value for specific alpha and highest learning curve. As n decreases, both AUC broken lines and learning curves become lower, indicating poor performance.

In the present experiment (Figure 3) we observe the $n = 1$ being the best number of steps for the TD algorithm. Furthermore, from Figure 3(left) we notice, that curves of all n values follow the same trend, despite the $n = 1$, which follows the opposite with the increase in α. Such behaviour signalizes, that the perfect number of steps is between 1 and 2 (based on the hypothesis about derivatives of the approximate curves in the earlier sections). Therefore, the same theory about preferable intermediate values applies here, except we can consider $n = 1$ being intermediate here. Moreover, we notice the tendency of the alpha value to be smaller, when the number of TD steps rises. Such a phenomenon of decreasing the learning rate with the increase of a backup trace length

may be explained by the fact that if we update the state far behind the previous one, we are less sure about the values we get, as there were many choices of actions done (the probability of selecting longer trace is lower). That way, we want to set the learning rate lower, as more information is needed to trust the values. The same reasoning applies to the previous agent algorithms.

5 WINDY ENVIRONMENT

Now, we add the possibility of the wind into the environment: after each action, there is a 50% chance that the agent is moved to the square below. All other methodology stays the same. The results are depicted in the Figure 4.
The most important conclusion we derive from the plots is that the TD with a long trace length (number of steps back) performs poorly in stochastic environments. The small TD value is preferred. Such a result is explained by the following. In a stochastic environment returns rely on a chance heavily. So, imagine that you are an agent and you move right, but due to the wind, you are also blown down and you find a really big reward in that state. Hereby, you found it occasionally, but if you backpropagate that reward many steps back, the agent will tend to come back to that location from the very beginning of his trace. However, the next time it arrives at the cell, the wind doesn't blow, and it receives no return. Such unexpected behaviour is a pitfall, which raises confusion in the agent, who tries to plan a lot of steps ahead. That is why we see, that $n = 1$ performs the best now, even though it was not always the case in the deterministic environment.
Another factor, that influences the results, is the size of the environment. In the present task, we have 144 states. For such a small world, the n of 64 is exceedingly high (so the agent has to make 64 steps before some values are updated). Moreover, during the initial 64 steps, the agent will be moving randomly, having a high chance to occasionally fall off the cliff and backpropagate unreasonably bad results. We may see that happening from the dips of the curves ($n = 64$) in the top plots in Figure 4. These two facts combined result in a horrible performance by the agent, which may be seen in the graphs as well. Elaborating on that we may suppose, that the TD value in the environments with high penetrations should be less or equal to the number of steps from the starting point to the highly draining (in terms of final reward) state.

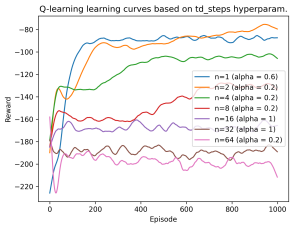

Figure 4.1: Q-learning

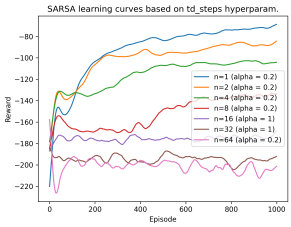

Figure 4.2: SARSA

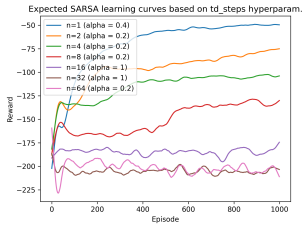

Figure 4.3: Expected SARSA

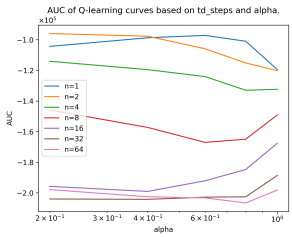

Figure 4.4: Q-learning

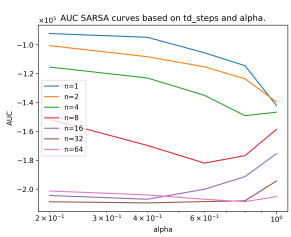

Figure 4.5: SARSA

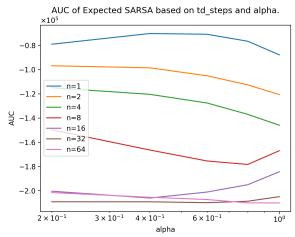

Figure 4.6: Expected SARSA

Figure 4: The results of the different algorithms in the stochastic environment with a 50% chance of wind. Top row shows learning curves of each agent, bottom row depicts AUC broken lines, as in the previous sections. The values obtained are averaged over 100 repetitions. The curves in the top plots are smoothed with a smoothing window of 31.

Another observation is that the fine-tuned Expected SARSA showed the best results in terms of the final reward. That may be explained by the fact, that this algorithm utilizes expected reward (as we know the transition function, we can calculate it). Such an approach is the best, as it weighs over all possible results and, thereby, doesn't fall into the maximization bias. Also, we verify again our theory about preference for the small amount of steps in a windy environment, as the agent with $n = 1$ significantly outperforms the agent with $n = 2$ and all the other values of n.

6 Comparison

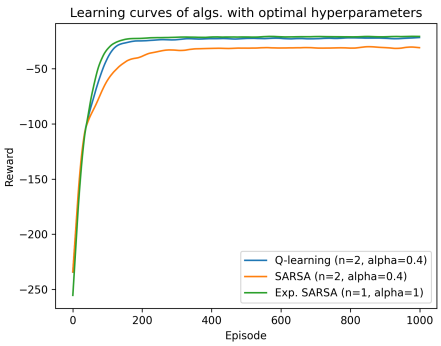

Figure 5.1: Initial environment.

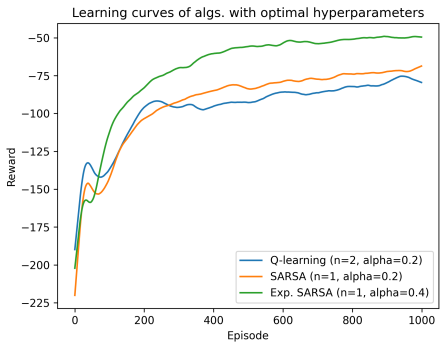

Figure 5.2: Windy environment.

Figure 5: Best learning curves of Q-learning, SARSA, and Expected SARSA agents based on the AUC metric in the different environments. The values obtained are averaged over 100 repetitions. The curves are smoothed with a smoothing window of 31. Expected SARSA has the highest graphs.

In the current section, we present the comparison of the fine-tuned algorithms in two environments (Figure 5). The first thing we notice from the graph is the difference in the shapes of the curves of the 2 environments. That happens due to the presence of the wind. Therefore, it takes longer for the agent to learn the most optimal trail. Secondly, we see that Expected SARSA has the best performance, which is expected as it combines the best characteristics of the other 2 algorithms. Lastly, and most importantly, we see that the predominant number of steps is either 1, or 2. That gives the main insights about the research: in small or stochastic environments, the small number of TD steps is preferred for the fastest possible learning procedure. Therefore, one of the possible further research on the topic may include environments with more states.

7 Conclusions

The main idea of this research was to determine the optimal number of TD steps for different algorithms in deterministic and stochastic environments. The results showed that fewer steps lead to faster learning, with the best performance observed at $n = 1$ and $n = 2$. In stable environments, the size of the environment relative to the number of steps appeared to influence the outcomes significantly. In the windy world, the added stochasticity introduced undesired uncertainty when updating states far before the previous one (higher n values). Also, in 5 we suggested the idea, that the number of TD steps should be lower or equal to the number of steps to the nearest cliff (a state with a high negative value) in all types of environments. Overall, we developed a number of theories, which got partially confirmed by the obtained plots: intermediate values of n perform best; derivative of the AUC curve seems to decrease at least at the smaller n; optimal alpha values decrease as n increases. With this in mind, we propose further research involving a larger state space with a relatively small number of TD steps to identify the optimal ratio of TD steps to state space size. In our research, we stuck between $n = 1$ and $n = 2$, which may not be fully representative. Also, future research could explore the link between the distance from the start to the nearest cliff and the optimal number of TD steps 8.6. Many more improvements, along with elaborations on study design are noted in the Appendix.

REFERENCES

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.