Assignment 4 - Dynamic Programming & Reinforcement Learning

Nathan Vaartjes[2596846]

Vrije Universiteit Amsterdam, Amsterdam 1081 HV, The Netherlands

1 Methods

1.1 Greedy

At the beginning, all the Q(x,a) values are 0, therefore the agent essentially makes random moves before the first reward is found (NB: we made sure that the agent would choose action A or B with equal probability when Q-values were equal. Otherwise, naive argmax would always pick action 1 and the agent would always stumble upon Q(5,A) by chance).

Because action B in state 1 is much easier to find from the origin (p=0.5) than action A in state 5 ($p=0.5^5$), the agent always find the reward of 0.2 in state 1/action B (s1aB) first, and updates Q(1,B), while Q(1,A) remains 0. From then on, the agent always chooses action B greedily and loops in state 1, and never takes any other action anymore. Thus, it never learns the reward of 1 in s5aA. This effect can be seen in fig. 1a

1.2 Epsilon-greedy

The epsilon-greedy agent, however, eventually learns the reward of s5aA (state 5, action A), and quickly adapts Q-values of all states to move towards state 5.

The value of Q(1,A), is higher than Q(1,B), suggesting that moving towards state 5 to collect 1 is worth it, even with the discount. Indeed, $0.9^5 * 1 > 0.2$. The Q-values can be seen in fig. 1b. Q-values can be seen in table 1.

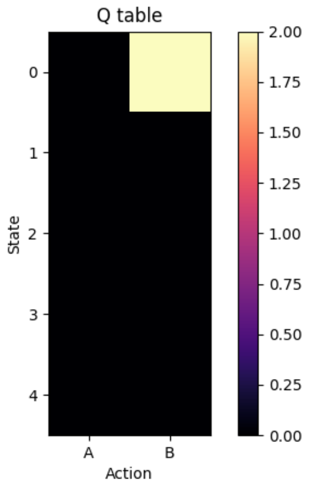
State	A	B
1	0.0	0.2
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
5	0.0	0.0

(a) greedy

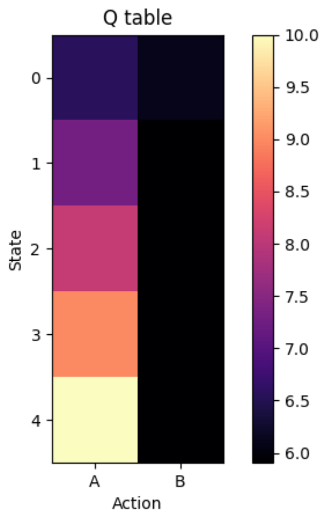
State	A	B
1	6.56	6.1
2	7.29	5.9
3	8.1	5.9
4	9	5.9
5	10	5.9

(b) ϵ-greedy

Table 1: Q-values of greedy and ϵ-greedy Q-learning



(a) Greedy Policy. All values are 0, except Q(1,B), which is 2.0

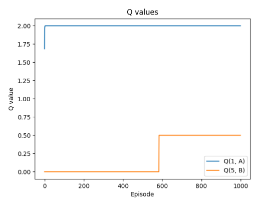


(b) Epsilon-greedy policy. Agent learns to go to state 5 and do action A. Note that all actions B are not 0 but around 6.

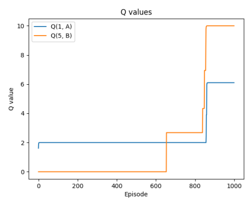
1.3 Effect of LR and ϵ

Epsilon It is very easy for the agent to discover the reward of 0.2 in s1aB. Reward 1 in s5aA is much harder, and only the epsilon-greedy approach can find it at all. However, the agent must explore enough to finally discover the reward of 1, and thus needs a sufficiently high ϵ. The probability of taking s5aA when starting in state 0 is $(\epsilon * 0.5)^5$, because it needs to take a suboptimal action 5 times in a row. Indeed, as can be seen in fig 2, the timestep at which the agent learns that s5aA is more valuable than s5aB happens much later when epsilon is higher. In case of $\epsilon = 0.1$, the agent often does not find the reward at all. What can also be derived form these plots, is that the agent must take the action s5aA at least 4 times before it learns to orient its behaviour towards maximizing the amount of times doing that action. The higher epsilon, the longer it takes for those 4 updates to be done.

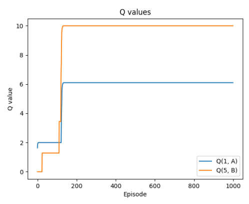
Learning rate The learning rate is equally as important, because in the rare chance that if performs action A in state 5, it must update its Q-values greatly. Only when the value of s5aA surpasses s5aB, will the agent learn to move to state 5 as much as possible. If the update is not strong enough, it still does not learn that action A in state 5 is superior. A low LR thus makes the agent iterate much more slowly until convergence, because the update steps are smaller. As can be seen in fig. 3, the agent must take s5aA many times before learning that



(a) $\epsilon = 0.1$



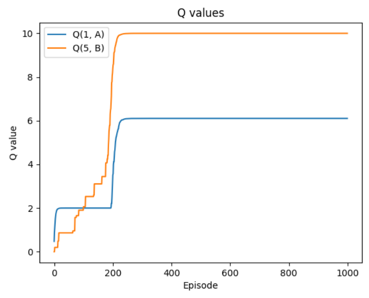
(b) $\epsilon = 0.2$



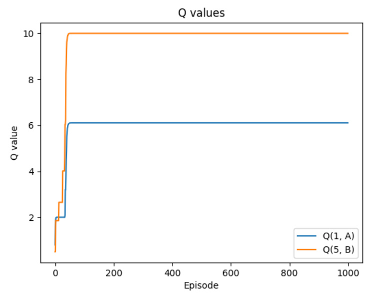
(c) $\epsilon = 0.5$

Fig. 2: Different values for ϵ. A low ϵ means a lower chance of finding s5aA, as can be seen in the amount of time it takes for Q(5,A) to rise.

the action is more valuable, after which the value Q(1,A) is also updated to reflect the value of going to state 5 form state 1.



(a) LR = 0.1



(b) LR = 0.5

Fig. 3: Different values for Learning Rate. We see that with a low LR, the Q-value increases are much smaller, and therefore we need more of them before the agent learns this is the optimal state/action pair.

1.4 10 states

For a chain of 10 states, the chance of finding s10aA is even smaller. If we were to take actions randomly, we would find it with a chance of 0.5^{10}. For an ϵ-greedy agent, this would be $(0.5 * \epsilon)^{10}$ (given action A is suboptimal) which is infinitely small in case of a small ϵ. With an ϵ of 0.5, the agent does not find the right Q-value for s10aA even in 5000 iterations of 50 steps each. This can be seen in fig. 4. Note that even with a state space of 10, it is still worth it to go to state 10 to do action A. This reflected in the Q-values, as Q(1,A) is 4.35, while Q(1,B) is 3.49.

In the formula above, ϵ is the most important factor to increase the chance of finding s10aA. We therefore increased epsilon to 1, and found that the agent was

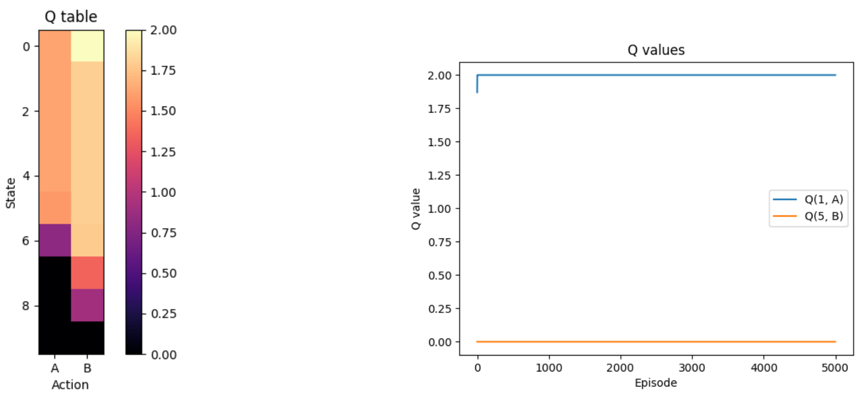


Fig. 4: Non-reaching, $\epsilon = 0.5$

successful in finding the right Q-value for s10aA. With ϵ=1, there is a $(0.5)^{10}$ chance of finding s10aA for every trajectory starting in state 0, which is about 3%. As Q-learning is an off-policy algorithm, the high ϵ does not make it unstable in estimating the Q-values. A example of a converging agent with ϵ=1 can be seen in fig. 6.

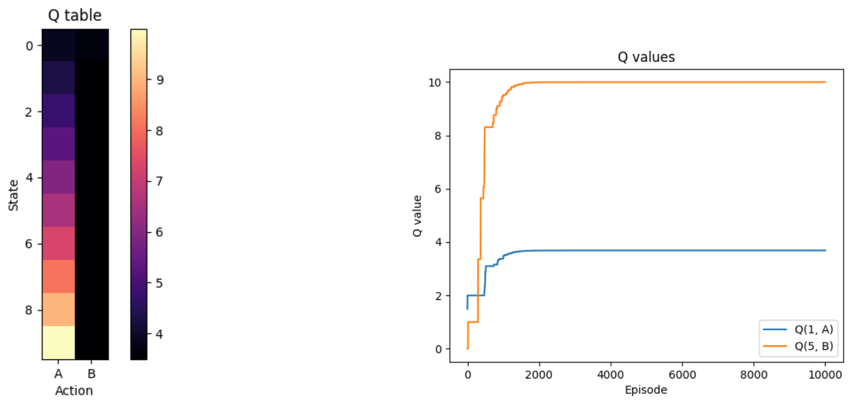


Fig. 5: Reaching, $\epsilon = 1$

2 Deep Q-learning

We implemented a neural network with a single hidden layer of 16 nodes and a ReLU activation. The input was the state/action index considered ([1,2] for action B of state 1, for ex.) and the output was the predicted Q value of that state-action pair. The output was non-activated, as the NN had to predict a real value.

For every episode, we let the agent run according to an epsilon-greedy policy for 100 steps, accumulating $Q_\theta(x, a)$ prediction values and (r(x,a) + $\gamma *$ $\max Q_\theta(x', a')$) target values.

After the episode, we selected randomly 32 or 64 tuples, and used those to update the NN's parameters via MSE loss and gradient descent.

We experimented with the parameters epsilon, batch size, learning rate (LR), and iteration size.

Iteration size We found that more than 2000 iterations of 100 steps were needed to reach convergence of the deep Q-learning algorithm. This was a surprisingly large number.

Epsilon Epsilon gave surprising results, as a too high value often resulted in no convergence or an unstable convergence process. This was in contrary to Q-learning, which was fine with an epsilon of 1. A too low value also resulted in the state 10 never being reached, as described in section 1. An equilibrium was reached with an epsilon of around 0.4 We interpret this is due to the fact that a too large epsilon does not give the chance to the algorithm to update the most promising actions consistently, as it will always sample random actions to update. A lower epsilon ensures the agent will perform the most valuable actions more frequently, and therefore update those more, going faster to convergence.

LR and batch size Via trial and error, we found an LR of 1e-4 and a batch size of 32 to be best.

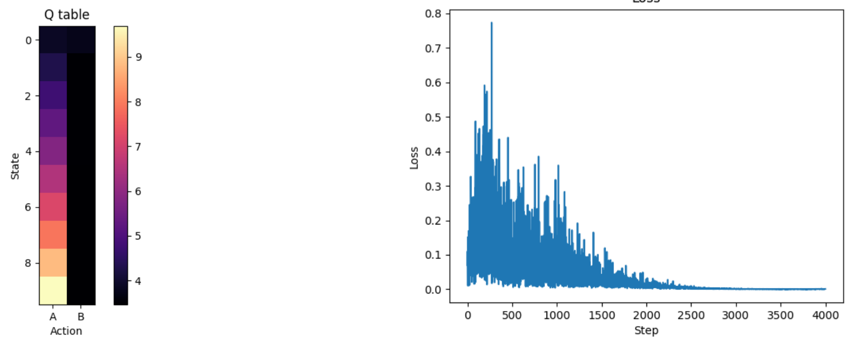


Fig. 6: Deep Q-learning Q-values and loss through iterations

Of course, this problem is not very well suited for deep Q-learning, as the state space is very small. As such, it is not possible to test the resulting model on some unseen part of the problem, because every state is seen. The model is also most certainly overfitting the problem, given such a small state space. Nevertheless, the model is able to converge to the same values as the Q-learning method described above, given the right hyperparameters.