Exercise 4 - Q-Learning

Theresa Eimer

1.1: The update rule

The general update function is:
$Q(i,u) = Q(i,u) + \alpha \cdot (r(i,u) + \gamma \cdot \max_u Q(j,u) - Q(i,u))$
[source: Sutton/Barto, Reinforcement Learning: An Introduction, second edition draft, p. 107]
This version is geared towards stopping the algorithm if a goal state is reached, because else the Q-value for G would continue to grow smaller towards $-\infty$ (as in this case $R = 0$ and $\gamma \cdot Q(j,u) = \gamma \cdot Q(i,u) = 0.5 \cdot Q(i,u)$, so the function degenerates to $Q(i,u) = Q(i,u) - \alpha \cdot 0.5 \cdot Q(i,u)$ in our case of $\gamma = 0.5$).
If stopping isn't an option for some reason, an indicator function for goal states would be an alternative, e.g.:
$Q(i,u) = Q(i,u) + I(i) \cdot \alpha \cdot (r(i,u) + \gamma \cdot \max_u Q(j,u) - Q(i,u))$
where $I(i) = 0$ if i is a terminal state and $I(i) = 1$ else. This stops the function from updating once the goal is reached. It doesn't really make sense to take any other action (like inverting the sign and letting Q(G, u) go towards infinity), because Q-Learning converges, so even if the value of G is never updates, the Q-values of the other states will adapt so that the proportions and thus also thee policy will over time go towards the optimal policy.

1.2 Calculating the function for one episode

As the initialization isn't important, let's assume $Q(i,u) = 0$ to start with. As $\alpha = 0$, it will be omitted from the following calculations.We're still working with $\gamma = 0.5$. To differentiate the states easily, we use double digit indexing, e.g. 11 for the left upper corner. The allowed actions are down, up, left and right and will be shortened to their starting letter (except up). Given this, we start the episode:
We start in 11 and move to 21 one step down with a reward of -1. Thus:
$Q(11,d) = 0 + 1 \cdot (-1 + 0.5 \cdot \max_u Q(21,u) - 0) = -1 + 0.5 \cdot 0 - 0 = -1$
The next steps are:
$Q(21,r) = 0 + 1 \cdot (-1 + 0.5 \cdot \max_u Q(22,u) - 0) = -1 + 0.5 \cdot 0 - 0 = -1$
$Q(22,up) = 0 + 1 \cdot (-1 + 0.5 \cdot \max_u Q(22,u) - 0) = -1 + 0.5 \cdot 0 - 0 = -1$
$Q(22,r) = 0 + 1 \cdot (-1 + 0.5 \cdot \max_u Q(23,u) - 0) = -1 + 0.5 \cdot 0 - 0 = -1$
$Q(23,up) = 0 + 1 \cdot (-1 + 0.5 \cdot \max_u Q(G,u) - 0) = -1 + 0.5 \cdot 0 - 0 = -1$
These are the only changed values, the rest remains at 0 (specifically also the Q-values of G). The updates here are very easy to compute, as everything was initialized with zero and isn't reused afterwards, thus all updated Q-values become the reward per step.

2.1 DQN implementation performance The network comprised of two convolutional layers (32 filters each, filter size 2x2, padded to size) with a max pool (2x2) before two fully connected layers with 128 units each and a linear output layer. There was a 50% dropout after each fully connected layer. Training was done with Adam and a learning rate of 0.0005. Training with 30000 steps worked well in some cases and was done mostly for time saving (I ran it on my laptop and it couldn't handle much more, running it on GPU would have solved that, I know), as often convergence would only occur much later and the loss would either grow very big initially or just oscillate a lot (the same goes for the same network with 32 units and only 3000 iterations, actually). A million steps would probably have mitigated that problem, but in the cases the training did work, it produced good results. The agent was able to find the goal in the following test run about as well as in the last task. I can only suspect it also generalizes better, but I didn't test it on an altered course. Below is a sample loss curve from one of the best training iterations with the smaller network.