Advanced Concepts of Machine Learning: Reinforcement Learning

Kevin Trebing (i6192338)

November 26, 2018

1 Learning algorithm: Q-Learning

I chose Q-Learning since this seemed to be an easy to implement option that should be able to learn the task well. Since Q-Learning uses a Q-Table for the state-action pairs it is easy to calculate a Value-function later on. The value-function is the best possible value in a state. This can be calculated later by looking at the Q-Table and returning the highest value of a state.

1.1 How easy it was to implement

At first I implemented something which I thought was Q-Learning, but later realized was more similar to Temporal Difference learning. This first implementation did not learn well at first, but after some tweaking of parameters and a long learning time it eventually learned something. After realizing I did not implement proper Q-Learning I adjusted the code strictly to the slides which resulted quite fast in a learning agent. As an enhancement I added a reward back-propagation for faster convergence. In my previous implementation it did not result in a faster learning agent, but with Q-Learning it did.

1.2 How easy it was to learn something useful

As previously already mentioned it was a bit tedious at first, since I implemented a wrong algorithm, but after Q-Learning was implemented properly the agent was already able to learn. This is probably due to the fact that my previous implementation also was able to learn something already.

2 Dealing with continuous space

In order to apply a Q-Learning algorithm the state space needs to be able to be looked up in a table. In order to do that I divided the continuous state space into bins. The features were divided into 50 same-sized bins. At first I chose a bin size of 20, but for a better visualisation I chose 50 later on.

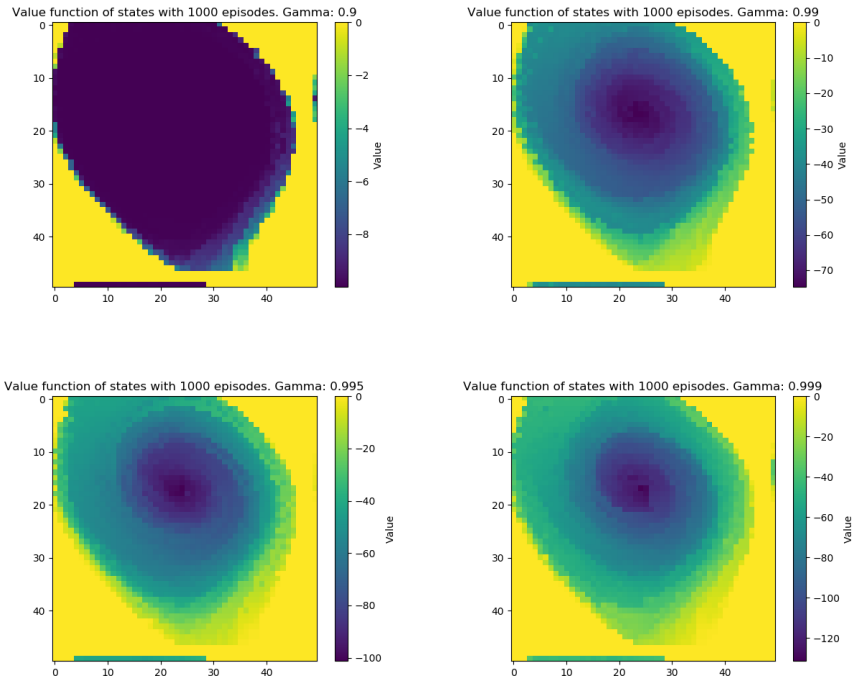

Figure 1: An increased gamma value changes the state values by a large amount. It also makes the plot finer grained. Looking at the plots with gamma bigger than 0.9 one can see spirals from the initial state. This may be due to the "swinging" that car has to make in order to gain speed.

3 Visualisation

The visualisation was constructed using the two features of the states. I plotted the highest value of the state with a heatmap to show the reward-surface. The agent tries to escape the blue area and trying to reach a yellow area. To show how the q-Values change I combined the intermediate q-Values into a gif. I created for three different gifs for gamma 0.9, 0.99 and 0.999.

3.1 Parameters to tune

One important parameter concerning visualisation is the number of bins. While it is possible to learn the task with few bins, it is not very appealing to have a highly pixelated plot. Therefore I set the number of bins to 50. This results in $50 * 50 * 3 = 7500$ states. The other important parameter is gamma. Since the reward function of the mountaincar task can be considered as sparse, since there are no positive rewards, the agent needs to plan sufficiently ahead. Therefore, gamma needs to be chosen in a way that rewards that are further away from the initial state are also captured in the q-value of the initial state. In Figure 1 we can see the value-function for different gamma values.