Reinforcement Learning
Exercise 7 - Solution

Jonathan Schnitzler - st166934
Eric Choquet - st160996

June 23, 2024

1 Linear function approximation

a) Tabular linear function approximation The method of just writing down all values for each state $V(s)$ can be achieved via the identity as a feature vector, also known as one-hot coding, e.g.

$$x(s_i) = \begin{pmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{pmatrix}^T . \tag{1}$$

This results in the same number as features as states in the state space. The tabular value is now just the weight

$$V(s) = w_i \tag{2}$$

and the possible non-linear function f is the identity. The state-action value $Q(s, a)$ could be achieved in a similar way, for $x(s, a) = e_j$ and an according enumeration. The linear function approximation

$$\hat{v}(s, w) = \sum_{i=1}^{d} w_i f(x_i)$$

is a generalization of the tabular methods.

b) Update rule for Sarsa

1. Tabular case

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

which is equivalent, shown in task a) to

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}_t)]$$

since there $w_{t,i} = Q(s, a)$

2. Function approximation

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}_t)]\nabla\hat{q}(S_t, A_t, \mathbf{w})$$

3. Linear function approximation

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[R_{t+1} + \gamma\mathbf{w}^T\mathbf{x}(S_{t+1}, A_{t+1}) - \mathbf{w}^T\mathbf{x}(S_t, A_t)]\mathbf{x}(S_t, A_t)$$

2 Mountain Car

a) Value Function Q-learning uses the update rule

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3}$$

The value function $V(s)$ is defined as the maximum over all possible actions a in state s of the state-action value function $Q(s, a)$. For the Mountain Car problem, we first discretize the state space as suggested in the exercise into 20 by 20 bins of position and velocity, i.e.

$$x_i = \frac{1.8}{20} \cdot i - 1.2, \quad i = 0, \ldots, 19 \tag{4}$$

$$\dot{x}_i = \frac{0.14}{20} \cdot i - 0.07, \quad i = 0, \ldots, 19 \tag{5}$$

The continuous state space is then rounded to the closest state of this state-aggregation. With three possible actions the state-action value function $Q(s, a)$ is a 20 by 20 by 3 matrix.

For the tabular version of linear function approximation, the weight vector $\mathbf{w}$ has the same size and is flatted to 1200 dimensions. The results of the value function during training are depicted in Figure 1 till Figure 3.

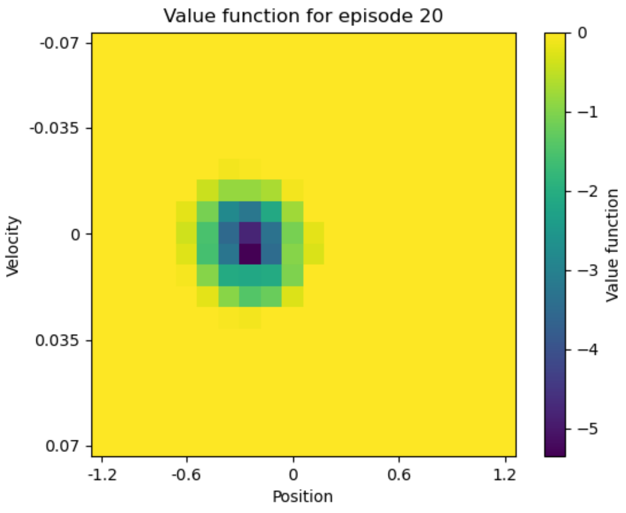

Figure 1: Value function after 20 episodes

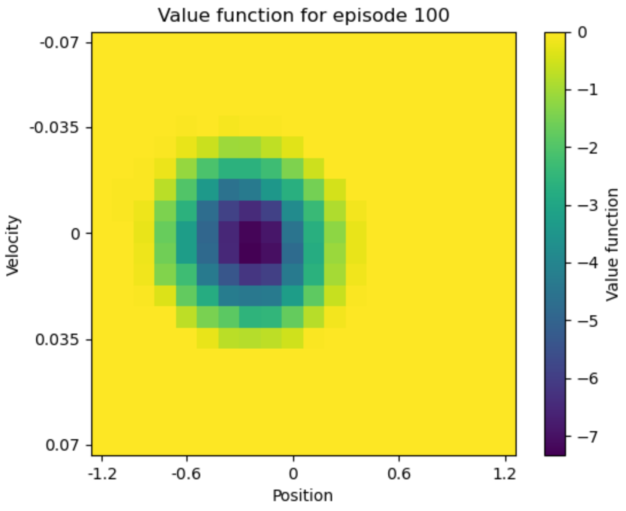

Figure 2: Value function after 100 episodes

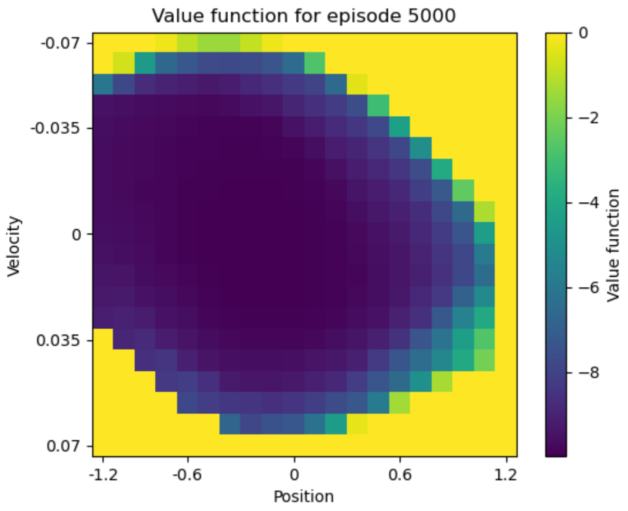

Figure 3: Value function after 5000 episodes

b) Average investigation In order to investigate the average performance of the agent, we run the algorithm 10 times and plot the average length and successes per episode. The results are depicted in Figure 4.

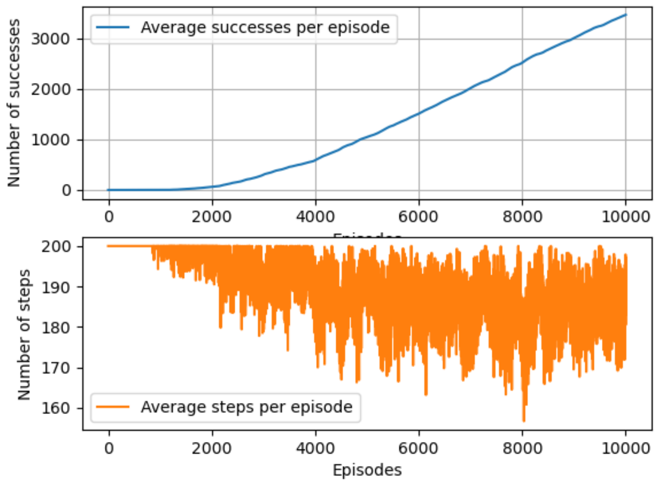

Figure 4: Averaged evaluation of 10.000 episodes over 10 runs

One can observe, that around episode 2000, the success rate is increasing significantly. As a logical consequence, the average length of the episodes is decreasing. The agent is learning to solve the task. Since the success rate is not yet 100%, the agent is not yet perfect. The average length of the episodes could probably also be decreased further.

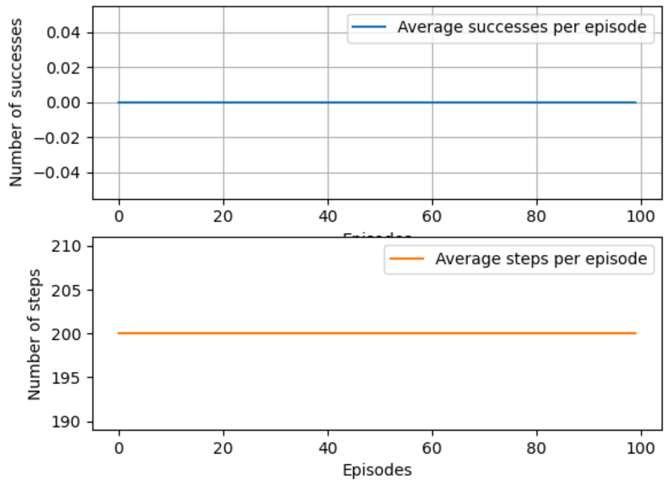

Figure 5: Single evaluation over 100.000 episodes

c) linear function approximation with RBF Instead of the simple identity feature vector x, we can use radial basis functions (RBF) to approximate the value function. The RBF is defined as

$$\phi(x) = \exp\left(-\frac{1}{2\sigma^2}||x-\mu||^2\right) \tag{6}$$

where μ is the center of the RBF, which is the current position of the mountain car and σ the width, which is chosen to $\sigma = 0.1$.

The learning curves for the RBF approximation are depicted in Figure 6. The program is not optimized and therefore would take more than 16-fold time to run, than the single optimizer, even when flooring the value of the RBF below 0.01 to 0. Alternative approaches could be implemented by avoiding the back transformation to the discretized state-aggregation.

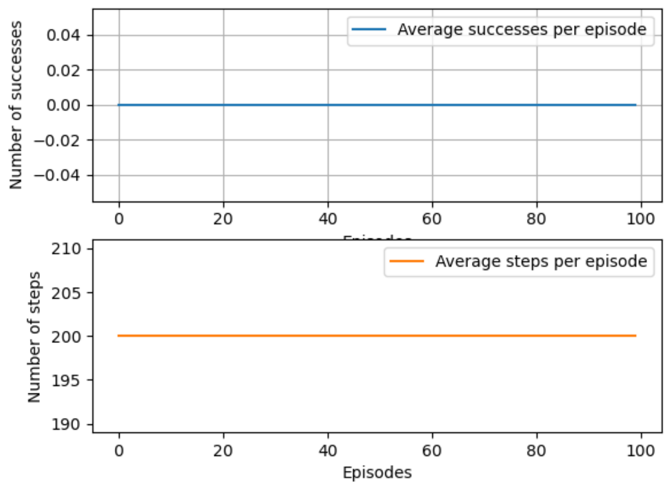

Figure 6: Learning curves for RBF approximation