Applications of Reinforcement Learning

Exercise 1: Learning rule for V-learning

In class we ‘derived’ the continuous-space SARSA-learning algorithm starting from a heuristic error function

\[ E = (Q(s, a) - r - \gamma Q(s', a'))^2, \]

using a radial basis function expression for \( Q(s, a) \).

Use an analogous heuristic approach in order to derive a continuous space V-learning algorithm. To do so, start with an error function

\[ E = (V(s) - r - \gamma V(s'))^2. \]

Computer-aided exercise: Gridworld

Study the points discussed in the previous exercises in computer simulations, using the class Gridworld provided in the file gridworld.py. The class implements an agent that lives on an NxN grid, whose state can be characterized by two integer coordinates \((x, y) \in (0, ..., N - 1) \times (0, ..., N - 1)\). At each position, he can choose among 4 actions: 'Up', 'Down', 'Left' or 'Right'. The agent’s task is to go to a previously unknown target position on the grid where he receives a reward. Moreover, he receives a smaller negative reward ('punishment') when bumping into walls. The agent learns according to the SARSA algorithm.

Size effects.

1. Create 2 different gridworlds of size 5x5 and 10x10

\[
\text{>> import gridworld}
\text{>> gw1 = gridworld.Gridworld(5)}
\text{>> gw2 = gridworld.Gridworld(10)}
\]

and let the agents learn for 20 trials:

\[
\text{>> gw1.run(N_trials=20)}
\]

Look at a few trials using the method visualize_trial() for both networks. What is the qualitative difference and how would you explain it?

2. Look at the navigation map of the agents, i.e., the action with the highest Q-value as a function of position.
How is the behavior of the agent reflected in the navigation maps? Compare the structure of the maps close to the target position and at larger distances.

3. Quantify this difference by plotting the latencies (i.e., the time it takes the agent to reach the target position) as a function of trial number. This curve is often called the *learning curve*. To get smoother curves, let the agent take 20 runs on the same problem and plot the latency curve averaged over all runs:

```python
  g1.run(N_trials=20,N_runs=20)
g1.learning_curve()
```

Compare the latencies for the 2 gridworlds. What are the latencies that you would expect if the agent has discovered the optimal strategy and how do they compare to the latencies in the computer simulations?

**Eligibility traces**

Create 2 different gridworlds of size 20x20, one of which learns with an eligibility trace and the other one without:

```python
  g1 = gridworld.Gridworld(20)
g2 = gridworld.Gridworld(20, lambda_eligibility=0.9)
```

Let the agents learn for 20 trials and a few runs (this might take a minute or so - why?). Compare the development of the latencies for the two agents. What do you observe and why?