Lecturer: Henning Sprekeler

Assistant: Denis Alevi
(`denis.alevi@bccn-berlin.de`)

Reinforcement Learning II

December 13, 2019

The solutions for these exercises (comprising source code, discussion and interpretation as an IPython Notebook) should be handed in before **December 20, 2019 at 10:15 am** through the Moodle interface.

1 Good vs. bad Döner

Consider a hungry reinforcement learning agent standing in a street with a bad and a good döner shop, which are at different ends of the street. The goal of this exercise is to study the effects of the discount factor on the behavior of the agent.

1. Write a program that implements an agent in an environment of 50 linearly arranged states (corresponding to different positions along the street). In each state, the agent can choose among 2 actions: up or down, which take it to the respective neighboring states. In every trial, it starts in a random position. If the agent walks up in the top-most state, it receives a reward of 100 (bad döner). If it walks down in the bottom-most state, it gets a reward of 1000 (good döner). For every step that does not lead to a reward, the agent gets a negative reward of -10 (still no döner). If either of the döners is received, the trial ends and the agent starts a new trial at a new random position. Initialize the Q-values with small random numbers and let the agent learn according to the SARSA algorithm:

   $$\Delta Q(s_t, a_t) = \eta(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)),$$

   with $\eta = 0.1$. The agent should follow an ϵ-greedy policy with $\epsilon = 0.1$.
   Hint: Terminal states in this setting require special attention. To ensure that the TD-error can go to zero in the last iteration, you can introduce two additional states at the borders of the environment whose Q-values are always 0.

2. Start with $\gamma = 1$. Let the agent learn for 50,000 trials to make sure that the Q-values have converged. Generate a plot that shows the duration of the trials as a function of trial number to check whether the agent is learning. For this, plot only the first 500 trials. You will observe that the agent's behavior improves a lot faster than the Q-values converge - explain why! After learning for 50,000 trials, make a figure with two curves showing the Q-values for walking up and walking down and another figure showing the preferred action as a function of the state (up $\rightarrow$ 1, down $\rightarrow$ -1). How does the agent behave after learning and why?

3. Create the same plots as in the last exercise for discount factors $\gamma = 0.99, 0.95, 0.9, 0.8$. How do the Q-values change? How does the final behavior of agent change? Provide an explanation.

2 Gridworld

In this exercise you will have to implement several methods in the class Gridworld for reinforcement learning in 2-dimensions. Most of the class is already complete, but you must now apply your knowledge of 1-dimensional SARSA to a 2-dimensional problem.

1. Download the gridworld.py file from Moodle. Review the existing code to get a sense of how the class functions. Provide an implementation for the empty methods `_run_trial`, `_update_Q`, and `_choose_action`. The comments in the file offer hints as to how you might do that. Do not modify gridworld.py! Create new functions in your Notebook and overwrite the methods of the imported class, e.g. `gridworld.Gridworld._run_trial = my_run_trial`. You will also need *avconv* on your computer to generate the trial animations. Install *libav-tools* using *apt-get* on Linux, *brew* on Mac OS. For Windows there are downloadable binary installers.

2. Create a gridworld of size 5x5 and let the agent learn for 20 trials:.

   ```
   >> import gridworld
   >> g1 = gridworld.Gridworld(5)
   >> g1.run(N_trials=20)
   ```

   Visualize a few trials like so (**Warning**: *Due to a quirk in the implementation, one should not generate further plots in a Notebook cell after visualizing a trial*):

   ```
   >> anim = g1.visualize_trial()
   >> gridworld.display_animation(anim)
   ```

   Repeat this procedure for a gridworld of size 10x10. What is the qualitative difference and how would you explain it?

3. Look at the navigation map of the agents, i.e., the action with the highest Q-value as a function of position using the method `navigation_map()`. It is also informative to view the Q maps individually with method `plot_Q()`.

   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.

4. 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:

   ```
   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?

5. Create 2 different gridworlds of size 20x20, one of which learns with an eligibility trace and the other one without. To do so, set constructor parameter `lambda_eligibility=0.9` to enable the eligibility trace.

   Let the agents learn for 50-200 trials and 20 runs. Explain why the runtime is rather high. Compare the development of the latencies for the two agents. What do you observe and why?