Lecturer: Henning Sprekeler

Assistant: Mathias Schmerling

Reinforcement Learning I

June 07, 2018

The solutions for these exercises (comprising source code, discussion and interpretation as an IPython Notebook) should be handed in before **June 14, 2018 at 10:15 am** through the Moodle interface (in emergency cases send them to `mathias.schmerling@bccn-berlin.de`).

Exercise 1: The 10-armed bandit

In this exercise we will test multiple policies that are supposed to learn the optimal strategy for a 10-armed bandit. A 10-armed bandit is a machine consisting of 10 levers and a reward distribution device, e.g., a chocolate dispenser. Each lever has a fixed probability of generating a reward of 1 (piece of chocolate) when operated. At the beginning, the "agent" does not know anything about the machine, but while activating different levers, she will learn more and more about the reward characteristics. While one aim is to get a good estimate of the reward probabilities by playing around ("exploration"), the agent concurrently intends to maximize the overall chocolate output ("exploitation"). The tricky question is: What is a good strategy that balances exploration and exploitation? Which levers should be activated given the reward history?
All strategies in this exercise rely on a common idea: The agent assigns a so-called *Q-value* $Q(a_i) = Q_i$ to each of the possible actions a_i (press lever i). Ideally, the Q-values mirror the expected reward associated with each action. After pressing a lever, the action values are updated depending on whether a reward was received ($R = 1$) or not ($R = 0$):

$$\Delta Q_i = \eta \left[R - Q_i \right] \tag{1}$$

The parameter η represents the learning rate.

1. Generate a set of lever reward probabilities $p \in [0, .9]$ of length 10 and **keep it fixed for the entire exercise sheet**.

2. Write a function `generate_reward` that will behave as an N-armed bandit (given lever numbers corresponding to indices into `p`, it returns rewards of 1 with a probability p_i, otherwise zeros). Confirm that your bandit works: Activate each lever 10000 times and compare the mean reward for each lever with its reward probability – they should be nearly identical.

3. Devise some method of storing and updating the Q_i value associated with the action of pressing lever i (take $\eta = 0.01$). Q should be initialized with zeros, since the agent is pessimistic and initially associates zero value with each action.

$$Q_i^{t+1} = Q_i^t + \Delta Q_i$$

4. We will first implement the ϵ*-greedy* policy: The agent will with probability $1-\epsilon$ press the lever which has the maximum associated Q-value (if there are several with that value, press them all), otherwise press a random lever. The agent starts by pressing all levers (since they all have the same Q-value initially), it updates the Q-values and then applies the described strategy iteratively. Write a function (or class method) `eps_greedy` that implements this strategy. The function/method should accept as input the current Q and should return the lever index (or indices) that are to be pressed, as well as the largest Q-value.

5. First we'll test the pure-greedy algorithm by setting $epsilon = 0$ and running 1000 iterations of the algorithm: 1) Choose the next lever(s) (`eps_greedy`). 2) Activate the chosen levers (`generate_reward`). 3) Update the Q-values according to the received rewards. Plot the expected reward (the reward probability of the current lever) against the trial number (each lever press constitutes one trial). Repeat the test 20 times and draw the resulting curves into a single plot. Does the agent consistently identify the levers with the largest reward probabilities?

6. Run 100 repetitions ("lives") of the ϵ*-greedy* strategy (with four different ϵ-values 0.0, 0.01, 0.1 and 0.5) at 1000 iterations each and store the expected reward in each trial. Plot the average of the expected reward, and the average largest Q-value across lives against the trial number. Include a line in the plot that runs parallel to the x-axis and represents the maximum expected reward (the maximum reward probability). Why do the expected reward and largest Q-value curves look the way they do for the four cases?

7. Finally, we program a *SoftMax* policy: The agent presses the levers with probabilities that depend on the Q-values:

$$P_i = \exp(\beta Q_i) / \sum_j \exp(\beta Q_j) \tag{2}$$

   Write a function (or class method) `softmax` that implements this strategy. The function/method should accept as input the current Q and output the lever index (it will always be a single lever index!) that is to be pressed, as well as the maximum Q-value. Test this function/method by running 100 lives with 5000 iterations each and store the expected reward and maximum Q-value in each trial. Draw the same curves as in task 6 for different values of $\beta = 1, 5, 15, 50$. Do the results of *SoftMax* show any improvements compared to the ϵ*-greedy* policies? How does the parameter β relate to the exploration/exploitation-balance? And once again explain why for each β the two curves (expected reward & largest Q-value) look the way they do.

8. Modify the 10-armed bandit so that operating the last lever always gives a reward of -5000 (the agent is punished and has to return 5000 pieces of chocolate). Test both the *epsilon-greedy* and *SoftMax* policies on this modified bandit (use $\epsilon = 0.1$, $\beta = 15$) by running 500 lives with 1000 iterations each and plot the same curve as in task 6. Do you observe a difference? Give an intuition for the result! (After this task ditch the punishment and return to the original bandit.)

9. Implement a *modified SoftMax* policy, where the parameter β increases linearly with the number of iterations i starting at a value of 1:

$$\beta(i) = 1 + i/b \tag{3}$$

Rerun the simulations with 5 different β-slopes: $b = 0.1, 0.4, 1.6, 6.4, 25.6$. Compare the results with the simple *SoftMax* ($\beta = 5$) by running 100 lives with 2000 iterations and plotting the same curves as in task 6. In a different figure, plot the cumulative sum of true rewards (the sum of rewards until the current trial, averaged over lives) against trial number. What do you observe? Furthermore, plot the cumulative reward at the end of the lifetime (2000 iterations) against the β-slope. Which of the five slopes generates the maximum overall reward? Explain the idea behind using a varying β, in particular regarding the exploration/exploitation-dilemma? Can you think of a similar modification to the ϵ-*greedy* policy?