Reinforcement Learning (SS18) - Exercise 5

Daniel Hennes

01.06.2018 (due 06.06.2018)

1. Recall the *Random walk* example presented in the lecture. From the results shown in the left graph (estimated value) it appears that the first episode results in a change in only $V(\mathrm{A})$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed (assuming $\alpha = 0.1$)?

Now consider the *Frozen Lake* environment:

Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend. If you intend to move down, you might slip to the right or left (intended and adjacent actions are executed all with equal probability 1/3). The surface is described using a grid like the following:

```
SFFF
FHFH
FFFH
HFFG

S : starting point, safe
F : frozen surface, safe
H : hole, fall to your doom
G : goal, where the frisbee is located
```

The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal (i.e. on the transition to the goal), and zero otherwise.

You may use the OpenAI Gym FrozenLake-v0 environment, but do not need to.

```
# https://github.com/openai/gym
# https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py
```

2. Implement *Sarsa* and obtain and plot the state–value function, action–value function, and policy for the *Frozen Lake* environment. Report the average episode length as training continues.
   (*Note: use sensible values for γ, α and ϵ.*)

3. Implement *Q–learning* and obtain and plot the optimal state–value function, action–value function, and policy for *Frozen Lake*. What can you say about on–line performance during training in comparison to the performance of the optimal policy?

4. Explorer how your results change if you switch to the non–slippery version (i.e. deterministic environment):

   ```
   from gym.envs.toy_text.frozen_lake import FrozenLakeEnv
   env = FrozenLakeEnv(is_slippery=False)
   ```

5. How would you need to change your code in order to implement off–policy *Expected Sarsa* with the *greedy* target policy?

6. (extra) Rerun your code for the larger *Frozen Lake* environment:

```
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
```

```
from gym.envs.toy_text.frozen_lake import FrozenLakeEnv
env = FrozenLakeEnv(map_name="8x8")
```