Exercise 4

Jari Mattila - 35260T
ELEC-E8125 - Reinforcement Learning

October 18, 2021

Task 1

Implement Q-learning using function approximation, at every timestep using the latest state transition to perform a TD(0) update. Also implement ϵ-greedy action selection. Test the implementation on the Cartpole environment. Test two different features for state representations:

(a) handcrafted feature vector $\phi(s) = [s, |s|]^T$

Training plots of all methods (Task 1 a), b), Task 2 and Task 4).

(b) radial basis function representations (use the featurizer inside the Agent class).

Training plots of all methods (Task 1 a), b), Task 2 and Task 4).

Source files: qlearning.py, xx.py

Question 1

Would it be possible to learn accurately Q-values for the Cartpole problem using linear features (by passing the state directly to a linear regressor)? Why/why not?

Task 2

Modify your Task 1 implementation to perform minibatch updates and use experience replay (**while keeping the original code for Task 1 submission**) [1, p. 440]. Run the experiments with Cartpole with both feature representations.

Training plots of all methods (Task 1 a), b), Task 2 and Task 4).

Source files: qlearning.py, xx.py

Question 2

Figure 2 shows the training performance of all four methods from Task 1, together with grid-based learning from Exercise 3 evaluated using GLIE with a = 50.

Question 2.1

How does the experience replay affect the learning performance?

Question 2.2

Discuss the benefits and cons of using hand-crafted features. As an example, you can refer to the given hand-crafted feature and more complex features like $\phi(s) = [s_x, s_{\dot{x}}, \cos(s_\theta), \sin(s_\theta), s_{\dot{\theta}}]^T$

Question 2.3

Do grid based methods look sample-efficient compared to any of the function approximation methods? Why/why not?

Task 3

Create a 2D plot of policy (best action in terms of state) learned with RBF with experience replay in terms of x and θ for $\dot{x} = 0$ and $\dot{\theta} = 0$.

Policy plot from Task 3.

Source files: qlearning.py, xx.py

Task 4

Replace the RBFs in your Task 2 implementation with a neural network (**while keeping the original code for Task 2 submission**). A basic DQN implementation can be found in dqn_agent.py. Evaluate the method's performance in CartPole and LunarLander environments.

Training plots of all methods (Task 1 a), b), Task 2 and Task 4).

Source files: qlearning.py, xx.py

Question 3.1

Can Q-learning be used directly in environments with continuous action spaces?

Question 3.2

Which steps of the algorithm would be difficult to compute in case of a continuous action space? If any, what could be done to solve them?

Final

To embed code snippets in the report, you can use the `pycode` environment.

```
for episode_number in range(train_episodes):
    reward_sum, timesteps = 0, 0
    done = False
    # Reset the environment and observe the initial state
    observation = env.reset()

    # Loop until the episode is over
    while not done:
        # Get action from the agent
        action, action_probabilities = agent.get_action(
           observation)
        previous_observation = observation

        # Perform the action on the environment, get new state and
            reward
        observation, reward, done, info = env.step(action)

        # Store action's outcome (so that the agent can improve
           its policy)
        agent.store_outcome(previous_observation,
           action_probabilities, action, reward)

        # Draw the frame, if desired
        if render:
            env.render()
```

If you add a figure, you can refer to it using Figure. 1.

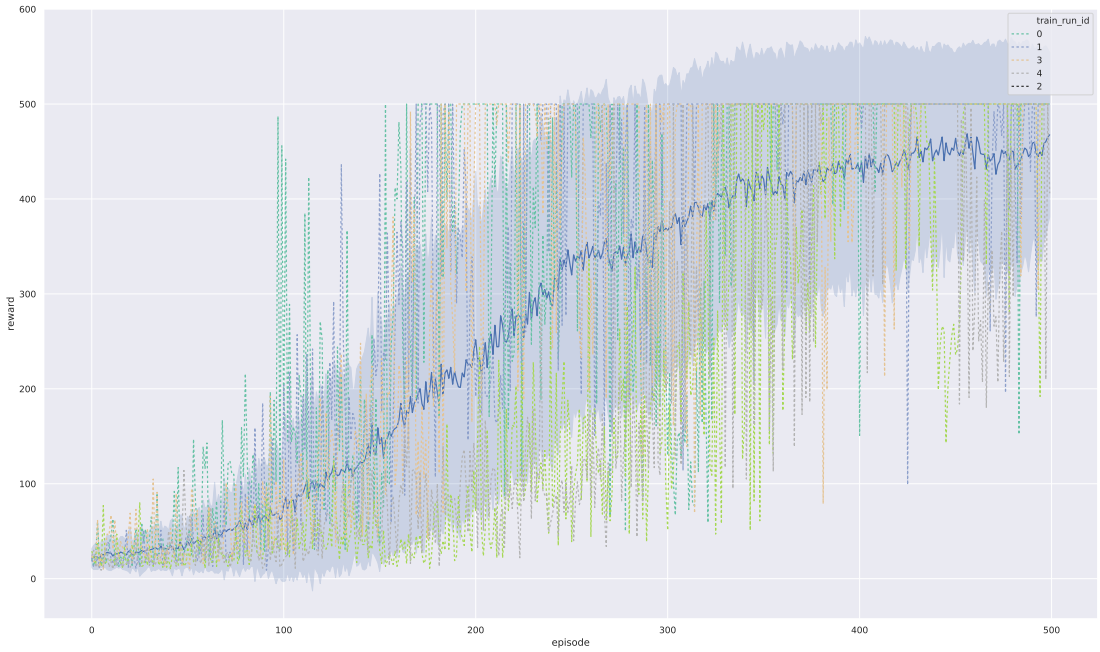

Figure 1: This is a sample figure.