Exercise 4

Jari Mattila - 35260T
ELEC-E8125 - Reinforcement Learning

October 23, 2021

Task 1

The training performance plots are presented below for Task 1a using handcrafted feature vector $\phi(s) = [s, |s|]^T$ in Figure 1 and Task 1b using radial basis function representations in Figure 2.

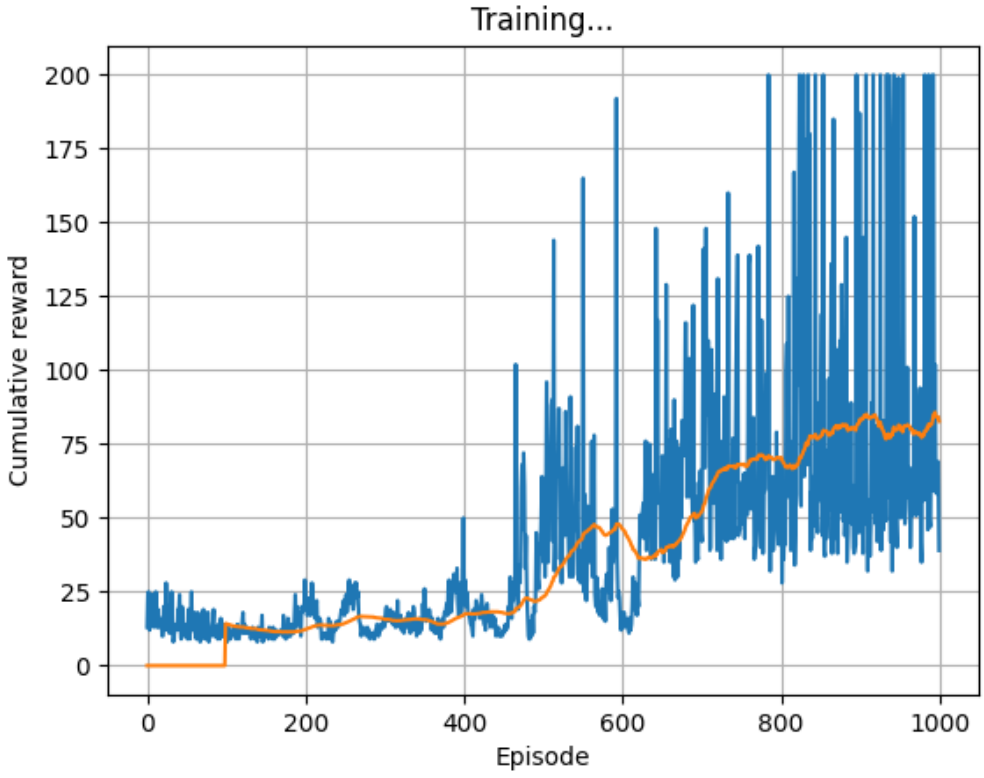

Figure 1: Training performance using handcrafted feature vector $\phi(s) = [s, |s|]^T$.

Source files: main_task1.py, rbf_agent_task1a.py, rbf_agent_task1b.py

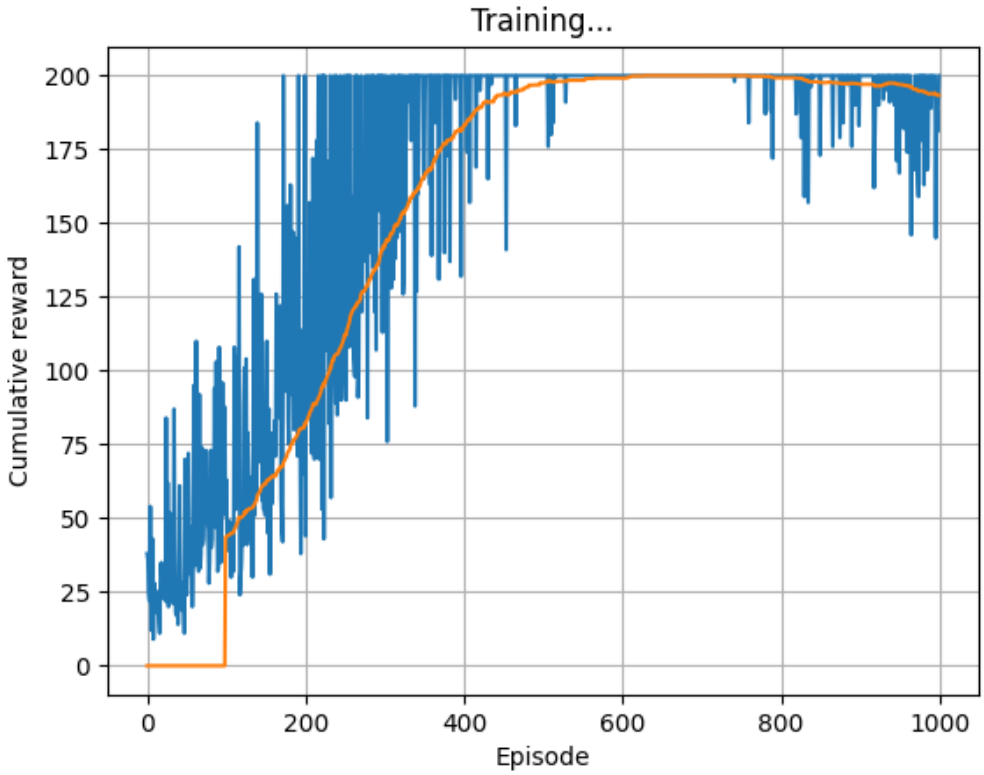

Figure 2: Training performance using radial basis function representations.

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?

Learning Q-values for the Cartpole problem accurately using linear features can be difficult because the linear model cannot model dependencies between the features.

For example, in the Cartpole problem the high angular velocity can be either good or bad depending on the angle - this cannot be taken into account by the linear model (for details see Section 9.5 of Sutton's book).

and use [**?**].

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: main_task1.py, rbf_agent_task1a.py, rbf_agent_task1b.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?