Exercise 4

Breno Aberle - 876438
ELEC-E8125 - Reinforcement Learning

October 18, 2020

1 Task 1

For task 1 we needed implement first of all the *featurize* fucntion. Each one of the features determines a specific state representation. The handcrafted features are implemented in the following way given the instructions in the task description:

```
return np.concatenate((state, abs(state)), axis=1)
```

While the radial basis function representations are implemented in this way:

```
return self.featurizer.transform(self.scaler.transform(state))
```

Moreover, we needed to modify the $single-update()$ function to perform a TD(0) update. First, we featurized all current and next states.

```
featurized_state = self.featurize(state)
featurized_next_state = self.featurize(next_state)
```

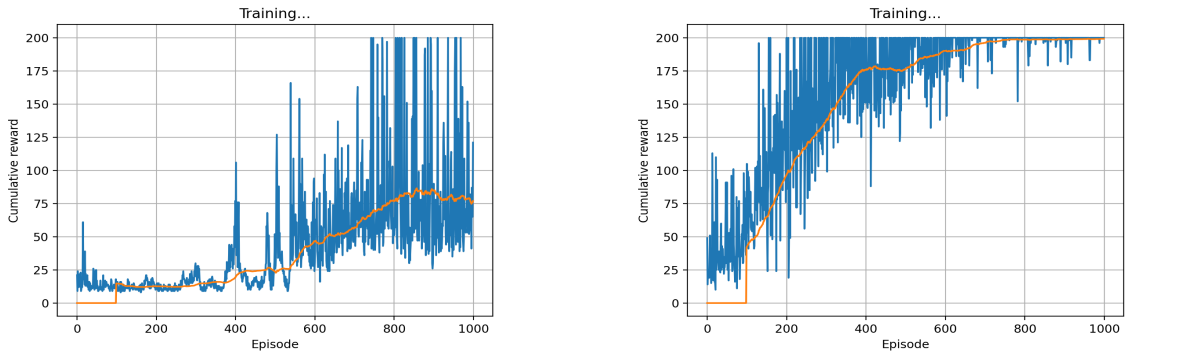
Then we calculated $Q(s', a)$ for the next state. Each action has its own function approximator. For each function we predict the Q value of the featurized next state and assign the max value to $Q(s', a)$.

```
predictions = []
for q_func in self.q_functions:
    predictions.append(q_func.predict(featurized_next_state))
next_qs = np.max(predictions)  # chose highest predicted value
```

Eventually, we calculate the targets. However, we need to differiante between terminal and non-terminal states in the update.

```
if done: # terminal state
    target = [reward + self.gamma * 0]
else: # not terminal state
    target = [reward + self.gamma * next_qs]
```

In figure 1, the training performance plots for both feature representations are shown. We can see that Q-Learning performs way better with the radial basis function representation. It already converges after approximately 600 episodes. While for the handcrafted features the reward still didn't exceed 100 on average after 1000 episodes.



(a) Handcrafted feature vector.

(b) Radial basis function representations.

Figure 1: Training performance of the Q-Learning algorithm.

2 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)?

In this exercise it is asked if the q-values can be learned accurately with a linear function approximation for the specific case of the cartpole.

Linear function approximation is great for simple reinforcement learning problems. Because the algorithm is easy to understand and efficient. However, it needs to be evaluated if a linear function approximator is appropriate for the problem? Linear function approximators have good convergence guarantee [1]. However, it was shown that for the well-known Atari game Reinforcement Learning problem a linear function is simply not effective at learning Q(s,a) values for these problems, because the problems are inherently non-linear. Within the Cartpole problem we are operating in a high-dimensional data environment and therefore the complexity of the model matters. Hence, the linear function approximation is just too simple. For example, neural networks as the function approximation could handle such a complex problem. In the OpenAI gym environment we are given position, velocity, angle and angle velocity. The ideal value function would be symmetric around the parameter theta equals 0. A linear approximation can't represent that properly. RBF features fit very well to that problem and this is why RBF features performed that well in Task 1.

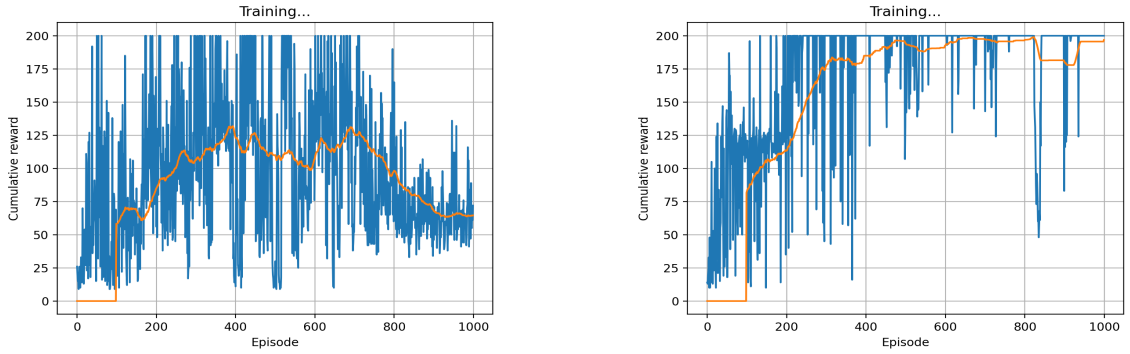
3 Task 2

In this task we use know minibatch updates and use experience replay. The transitions are stored and batch update done via the following functions calls:

```
agent.store_transition(state, action, next_state, reward, done)
agent.update_estimator()
```

Furthermore we need to complete the $update-estimator()$ function. Since we are working now with batches we needed to create states, action, next-states, rewards and done batches. Afterwards like done in Task 1 we need to calculate Q(s', a) for each sample of the batch and calculating the updated target values.

In figure 2 the performance plots for this task are visualized. We can see that similar results are achieved as in task 1. However, for handcrafted feature vectore the max reward average increased from around 90 to 130. With radial basis function representation it performs well in both tasks. However, in task 2 the convergence is achieved earlier after around 300 episodes due to experience replay and batch updates.



(a) Handcrafted feature vector.

(b) Radial basis function representations.

Figure 2: Training performance plots for Q-Learning with minibatch updates and experience replay.

4 Question 2

5 Question 2.1

How does the experience replay affect the learning performance?

With experience replay convergence is achieved earlier since you can access information from previous episodes which speeds up convergence. However, experience play doesn't necessarily improve the overall performance.

6 Question 2.2

Discuss the benefits and cons of using hand-crafted features

One great benefit of hand-crafted features is that less data for learning is needed than for example with rbf approximation function. Furthermore, depending what hand-crafted

features are chosen the computation is faster. Of course this depends on the complexity but in that case the complexity can be chosen. Another advantage is the flexibility. You are able to tailor the features better to the current problem. Since the features are made by yourself they are more straightforward and easier to understand what is happening. You have the opportunity to explicitly appreciate features which largely contribute to the particular task. So as in the example given with sin and cos of theta, we can put special emphasis of the attribute angle. Since the angle of the pole is an important feature in balancing the sin and cos can help in a way for the cartpole problem.

However, hand-crafted features also have disadvantages. First of all, it is difficult to assess if the hand-crafted features are suitable for that problem. With rbf function approximator you can get in general in many cases good and solid results. It may be challenging to come up with such reasonable and consistent features on some hard tasks. It is difficult to decide what is important and how to construct it. In the specific feature representation which is given in the question the cosine and sine would not help in a problem without angles. You always need to assess the problem and identify what attributes are involved and important.

7 Question 2.3

Do grid based methods look sample-efficient compared to any of the function approximation methods? Why/why not? Hint: An algorithm is said to be sample-efficient when it requires less samples (data) to reach an optimal performance.

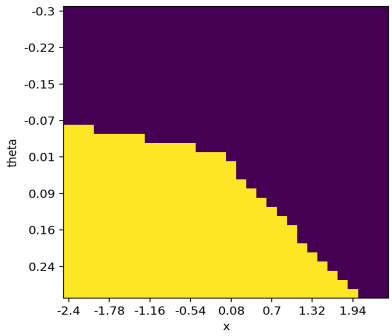
Given figure 2 in the task description of this week's exercise session, we can see that grid based methods don't look sample-efficient compared to any of the function approximation methods. It is not clear to see but 'handcrafted' seems to converge to optimal performance after 700 episodes. The curve of 'grid-based' looks really flat, so it is difficult to identify when convergence is reached. It seems that is increasing steadily in very low steps, so convergence is definitely much later than the other function approximations. 'RBF' converges after around 650 episodes. 'RBF, experience replay' converges after around 300 episodes and 'handcrafted, experience replay' after 100 episodes. To conclude, grid based doesn't look sample-efficient compared to any of the function approximation methods.

8 Task 3

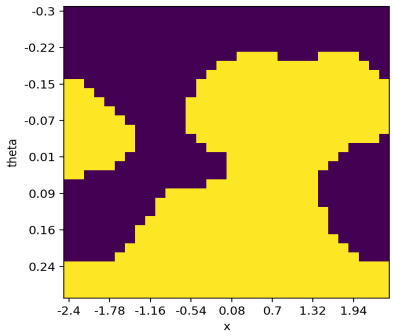
The 2D plot of policy learned with RBF with experience replay in terms of x and theta for velocity = 0 and angle velocity = 0 can be seen on the right of figure 3. Compared to the heatmap with handcrafted feature vectors it is definitely better since it adopts a more complex structure to adapt better to the problem. However, there is a lot of variance in the plots, so that's why it is difficult to evaluate it.

9 Task 4

Replace the RBFs in your Task 2 implementation with a neural network. Evaluate the method's performance in CartPole and LunarLander environments.



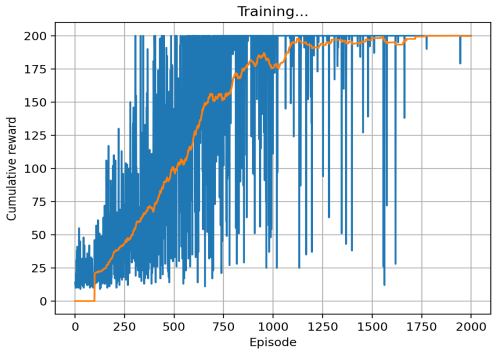
(a) Handcrafted feature vector.



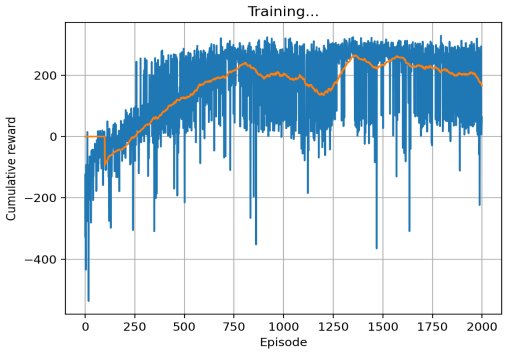
(b) Radial basis function representations.

Figure 3: Heatmap for Q-Learning with minibatch updates and experience replay over position and theta, with velocity and angle velocity equals zero.

In this task 4 we use a neural network instead of RBF like before. The computation for the cartpole environment took around 23 mins and for Lunar Lander around 43 mins runtime on my computer. This is because the environment of cartpole is less complex. In figure 4 you can see the performance plots for both environments.



(a) Cartpole environment.



(b) Lunar Lander environment.

Figure 4: Performance plots for DQN algorithm.

10 Question 3.1

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

Q-learning is not really suitable for continuous action space because we normally work with q grids and update values. Therefore Q-learning works much better in discrete state space.

11 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?

To do these beforementioned q value updates we need to calculate q-max. And this is one of the biggest challenge in applying Q-learning to continuous-action RL problems. How do we maximize the q value in the continuous action space. This is also adressed in [2]. There are several papers dealing with Q-Learning in continuous state and action spaces. However, they try to overcome these problems by applying advanced methods like neural networks.

There are some solutions to use local approximators such as for example Radial Basis Function networks like we have seen in the previous tasks. On the other hand, the dimensionality of your state space maybe is too high to use local approximators. Which is a major drawback.

We saw in task 4 that neural networks can help since they more complex. The next topics of the lecutres are policy gradient and actor-critic. They could be maybe also be able solve that difficulty.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[2] M. Ryu, Y. Chow, R. Anderson, C. Tjandraatmadja, and C. Boutilier, "Caql: Continuous action q-learning," 2020.