Reinforcement Learning - Week 4
Function Approximation

Antonio Chiappetta

October 2019

1 Function approximation with non-linear features

1.1 Task 1

The implementation of Q-learning using function approximation and TD(0) for updates at each timestep produced the following results. In the first case the state was represented by an **handcrafted feature vector** in the form $\phi(s) = (s \ |s|)^T$, while in the second case it was represented through a **radial basis function**.

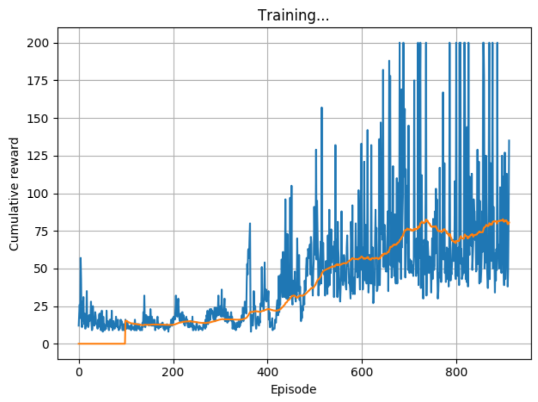

Figure 1: Q-learning with function approximation and handcrafted feature vector

1.2 Question 1

A linear regressor learning Q-values from the state would fail in this case. Using such an approximator prohibits the representation of interactions between features, that can be relevant for the cartpole environment. In fact, a high angular

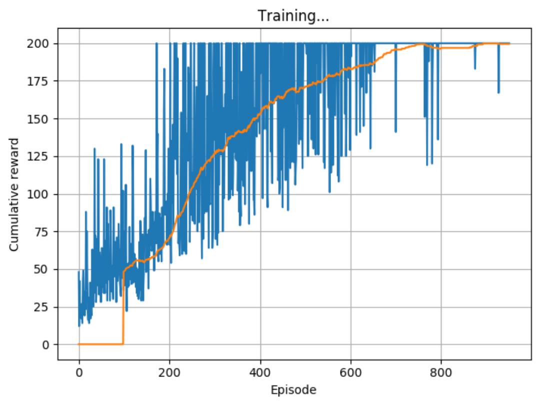

Figure 2: Q-learning with function approximation and RBF representations

velocity may be either good or bad depending on the angular position. In cases with such interactions additional features for conjunction of feature values are needed when using linear function approximation methods. [1]

1.3 Task 2

The following images show the results of using **mini-batch updates** and **experience replay** with both the configurations used in the previous task.

With this modifications both configurations reach almost the same final result as before, but show a quicker convergence.

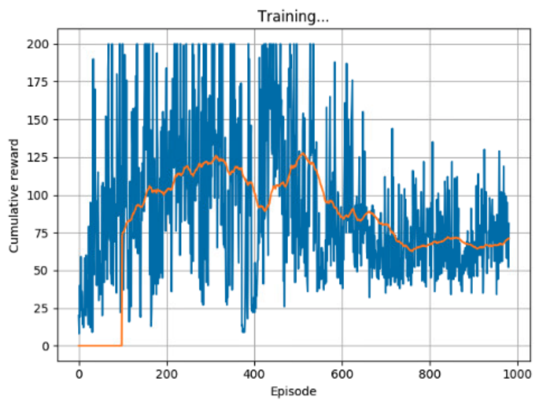

Figure 3: Q-learning with function approximation and handcrafted feature vector, using mini-batch updates and experience replay

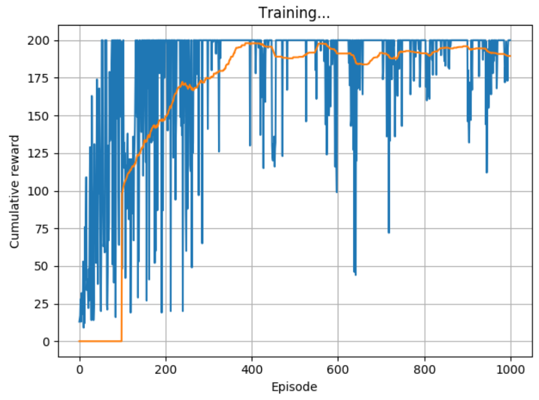

Figure 4: Q-learning with function approximation and RBF representations, using mini-batch updates and experience replay

1.4 Question 2

1.4.1 Sample efficient method

RBF with experience replay is the most sample efficient, as we can see from the figure where it approaches the goal better and in less episodes compared to all the others. It needs less samples than normal RBF because it uses the ones stored in the replay memory.

1.4.2 Improve handcrafted features

The efficiency of handcrafted features could be improved by correctly modeling the interactions between variables that constitute the state.

1.4.3 Grid-based methods sample efficiency

If we compare grid-based methods to function approximation methods they do not look sample efficient. The main reason is that they need a lot of episodes to learn a behaviour that is not even as efficient as the ones generated with function approximation.

1.4.4 Hyperparameters and design choices affecting sample efficiency

Function approximation methods generate a performance that highly depends, among the others, on:

- Feature selection: independent variables and their interactions;
- Method itself: can be linear or not linear, the choice can depend on interactions and input dimensionality;

- Performance measure to maximize;
- Hyper-parameters of the learning method.

1.5 Task 3

In the image below we can see how the policy learned with RBF using experience replay performs in terms of different values of position x and velocity θ of the cartpole.

To show the policy in a clear way, I discretized position values on the horizontal axis and velocity values on the vertical axis, mapping their values on 32 intervals. In the graph, value $a = 0$ corresponds to the action *go left*, while value $a = 1$ corresponds to the action *go right*.

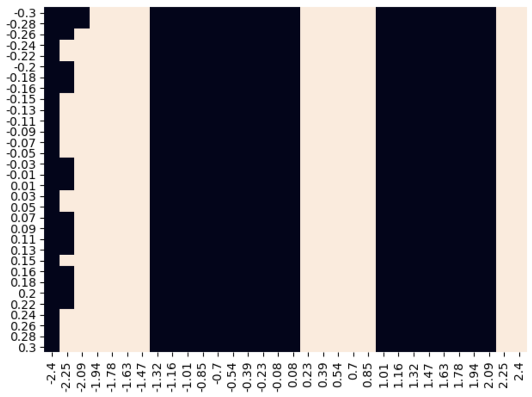

Figure 5: Plot of policy learned with RBF using experience replay in terms of x and θ

2 A (not so-) deep Q-network

2.1 Task 4

The following images show the training plots of the Cartpole and LunarLander environments generated by the **DQN** implementation.

2.2 Question 3

2.2.1 Q-learning with continuous action spaces

Q-learning cannot be applied directly with a continuous action space, the simplest approach to use it is **discretization**. If the dimensionality is high, *independence assumptions* could reduce the total number of discretized actions. [1]

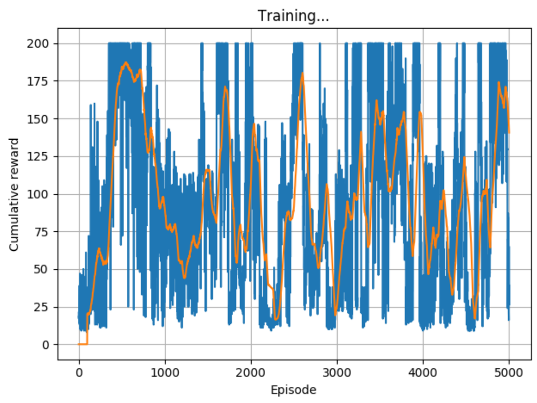

Figure 6: Training plot generated by DQN for the Cartpole environment

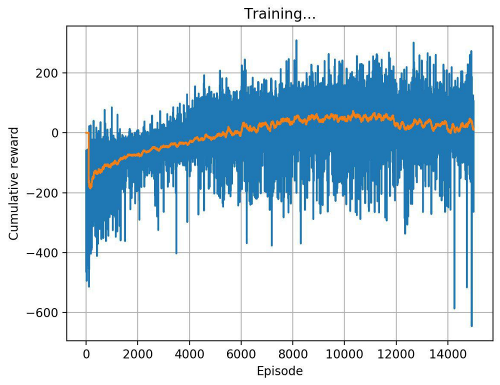

Figure 7: Training plot generated by DQN for the LunarLander environment

2.2.2 Improve algorithm for a continuous action space

The definition of the target state-action value needs to compute the maximum state-action value over a set of actions. If the action space is continuous, maximizing this function can be very time and resource consuming. Other approach could work better, such as **policy gradient** and **actor-critic** methods. [1]

References

[1] R. S. Sutton and G. Barto. *Reinforcement Learning*. The MIT Press, 2015.