CMPE 58Y - Robot Learning
Homework 2: Q-learning with function approximation

TA: Alper Ahmetoğlu
ahmetoglu.alper@gmail.com

April 16, 2021

1 Introduction

In this homework, you will implement Q-learning with function approximation for the cart pole task [2] in OpenAI Gym environment.

2 Function Approximation

As you might have noticed in the first homework, the number of entries in the table grow very quickly when you discretize continuous-valued variables. Discretization is also a quite cumbersome work since we might not know anything about the environment beforehand (i.e. how to discretize, what is important, and so on). Alternatively, we can use a function to represent $Q(s, a)$ instead discretizing continuous-valued variables. By this way, we do not have to fill a possibly very large table but only learn a few parameters.

As the cart pole task is quite easy, a linear transformation should suffice. We observe a four-dimensional state and select one of the two actions. Thus, we will have a `[4, 2]` sized matrix A, and `[2]` sized vector b as our parameter set. To compute the Q value for action a given state s:

```
# s @ A == np.matmul(s, A)
out = s @ A + b
```

Here, `out` will be a two-dimensional vector, one Q value for each action. So, for the first action, the Q value will be `out[0]`. To update A and b so to have a more accuracte Q value, we need some sort of direction, supervision. Remember the update rule for the Q-table learning:

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)) \tag{1}$$

Motivated from this update rule, we will use the following function as our loss function (also known as: objective function, error function) and update our parameters so to decrease this loss:

$$L = \frac{1}{2}\Big(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s, a)\Big)^2 \tag{2}$$

This is also known as temporal difference learning [1]. Since this loss function is differentiable with respect to our parameter set, we can use gradient-based learning. For this, we need to first find the gradients $\partial L/\partial \mathtt{A}$ and $\partial L/\partial \mathtt{b}$:

$$\frac{\partial L}{\partial \mathtt{A}} = \frac{\partial L}{\partial Q(s_t, a_t)} \frac{\partial Q(s_t, a_t)}{\partial \mathtt{A}} \tag{3}$$

$$\frac{\partial L}{\partial \mathtt{b}} = \frac{\partial L}{\partial Q(s_t, a_t)} \frac{\partial Q(s_t, a_t)}{\partial \mathtt{b}} \tag{4}$$

After you correctly calculate these gradients, you can update your parameters using stochastic gradient descent:

$$\mathtt{A} = \mathtt{A} \ - \eta * \frac{\partial L}{\partial \mathtt{A}} \tag{5}$$

$$\mathtt{b} = \mathtt{b} \ - \eta * \frac{\partial L}{\partial \mathtt{b}} \tag{6}$$

where η is the learning rate. As in the previous homework, the convergence of the algorithm depends on your hyperparameter settings. You can check `A` and `b` and how they change over time to understand what might be wrong.

One important thing is that while you set your target $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$, you should set it to r_t if s_{t+1} is a terminal state (i.e., the episode has terminated because the agent has failed or succeeded). That is, if `done` variable is true, then the episode has ended, and there will be no more opportunity for the agent to continue and collect future rewards. Therefore, including the discounted reward for the future (i.e. $\gamma \max_{a'} Q(s_{t+1}, a')$) will be a wrong target.

Plot the reward over episodes and submit your code (a jupyter notebook is also fine) to `ahmetoglu.alper@gmail.com`. Feel free to ask any questions. Cheating will be penalized by -200 points.

Deadline: 23.04.2021 23:59 GMT+3

Extended deadline: 30.04.2021 23:59 GMT+3 (-20 points)

References

[1] https://en.wikipedia.org/wiki/Temporal_difference_learning.

[2] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.