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

March 6, 2020

1 Introduction

In this homework you will implement Q-learning with **function approximation** for the cart pole task [3] in OpenAI Gym environment. As in previous homework, do not care about `done` variable. Terminate the episode after 500 iterations. You can consider the task is solved if you consistently get +400 reward.

2 Function Approximation

Instead of using a large table which is not feasible for continuous-valued variables, we can use a function. As you might have noticed in the first homework, you have to discretize states to keep a table. However, it is cumbersome in general since you might not know anything about the environment, how to discretize and so on. What we do here instead is using a function approximator which will directly give us action values. After all, all we need is to select the best action.

As this task is quite easy, a linear transformation should suffice. You will observe a four-dimensional state. You will have a `[4, 2]` sized matrix A, and `[2]` sized vector b as your parameter set. The computation is:

```
out = np.matmul(observation, A) + b
```

which will correspond to $Q(s, a)$. Here, out will be two-dimensional, one Q value for each action. To update A and b, you need some sort of direction, supervision. Remember the update rule for 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 with respect to this loss function:

$$L = \frac{1}{2}\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - \texttt{out}[a]\right)^2 \tag{2}$$

This is also known as **temporal difference learning** [2]. Since this loss function is differentiable with respect to our parameter set, we can use gradient-based learning. You need to find the gradients, $\partial L/\partial \mathtt{A}$, $\partial L/\partial \mathtt{b}$.

$$\frac{\partial L}{\partial \mathtt{A}} = \frac{\partial L}{\partial \mathtt{out}} \frac{\partial \mathtt{out}}{\partial \mathtt{A}} \tag{3}$$

$$\frac{\partial L}{\partial \mathtt{b}} = \frac{\partial L}{\partial \mathtt{out}} \frac{\partial \mathtt{out}}{\partial \mathtt{b}} \tag{4}$$

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. One of the most important thing is to clip your parameters into a range, `[-lim, lim]`, to stabilize learning.

3 Deliverables

Plot the reward over episodes. Submit your code (a jupyter notebook is also fine) to `ahmetoglu.alper@gmail.com`. For any questions regarding the description, environment installation, hyperparameters and so on, you can come to my office BM-31 (COLORS-LAB). Cheating will be penalized by -200 points.

Deadline: Friday, 13 March, 11:59 P.M. (You will be graded out of 100)
Late deadline: Friday, 20 March, 11:59 P.M. (You will be graded out of 80)
Bonus (10 points): Implement the stochastic gradient descent with momentum [1] and submit it on Friday, 13 March, 11:59 P.M.

References

[1] Stochastic Gradient Descent. https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Momentum.

[2] Temporal Difference Learning. https://en.wikipedia.org/wiki/Temporal_difference_learning.

[3] 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.