CSC 411: Assignment #4

Xiangyu Kong **Yun Lu**
kongxi16 **luyun5**

March 29, 2018

Problem 1

In class "Environment", the "grid" is represented by an 1-D array of 9 elements. Attribute "turn" represents whose turn it is. The valid values for "turn" is either 1 (x) or 2 (o). Attribute "done" represents whether the game is done.
The code and output of a game is shown in the listing below.

Listing 1: Code and Output of a game

```
for i in range(9):
        env.step(i)
        env.render()
return
x..
...
...
=====
xo.
...
...
=====
xox
...
...
=====
xox
o..
...
=====
xox
ox.
...
=====
xox
oxo
...
=====
xox
oxo
x..
=====
xox
oxo
x..
=====
xox
oxo
x..
=====
```

Problem 2

1. The Policy class is implemented in the listing below. The network consist of one hidden layer and the activations used are ReLU activation.

   Listing 2: policy

   ```
   class Policy(nn.Module):
       def __init__(self, input_size=27, hidden_size=64, output_size=9):
           super(Policy, self).__init__()
           self.network = nn.Sequential(
               nn.Linear(input_size, hidden_size),
               nn.ReLU(),
               nn.Linear(hidden_size, output_size),
               nn.ReLU())
       def forward(self, x):
           return self.network(x)
   ```

2. The 27-dimensional vector should be viewed as a 3×9 matrices with each column as a one-hot vector. The column number represents the position on the grid (1st column represent Environment.grid[0]), and the vector itself represents the state of the current position. The state correspond to the map in Environment.render() ({0: '.', 1: 'x', 2: 'o'}).

3. The 9 values Policy returns are the probabilities that it chooses the corresponding next move (e.g. the first element represents the probability that the next move is chosen to be placed at the first element in Environment.grid).

   The select_action samples the action according to the probabilities (stochastic) instead of choosing the maximum probability (deterministic). Then the policy is stochastic.

Problem 3

1. the implementation for compute_returns is shown in the listing below.

   Listing 3: compute return

   ```
   def compute_returns(rewards, gamma = 1.0):
           res = []
           for i in range(len(rewards)):
                   curr_return = 0
                   cur_rewards = rewards[i:]
                   for j in range(len(cur_rewards)):
                           curr_return += cur_rewards[j] * (gamma ** j)
                   res.append(curr_return)
           return res
   ```

2. The backward pass cannot be computed during the episode because the reward is not fully recorded and thus computing the gradient may produce a biased return.

Problem 4

1. See tictactoe.py for implementations.

2. The rewards are shown in the listing below. The win and lose rewards are +10 and −10 respectively. The rewards for valid move and invalid move are +5 and −5 respectively because we want to encourage the policy to make a valid move instead of one that is invalid. Also, their absolute reward values are lower than that of the win or lose rewards. This is because although we encourage the policy to make valid move, we want it to make moves that will lead to winning results more. Finally, we reward tie as 1. This is because getting a tied result is better than losing, so we encourage the policy to get a tied score just a little bit.

Listing 4: Rewards

```
Environment.STATUS_VALID_MOVE: 5,
Environment.STATUS_INVALID_MOVE: -5,
Environment.STATUS_WIN: 10,
Environment.STATUS_TIE: 1,
Environment.STATUS_LOSE: -10
```

Problem 5

1. The training curve is shown in Fig.1. The hyperparameters are as follows: lr = 0.001, gamma = 1.0, max_iter = 50000

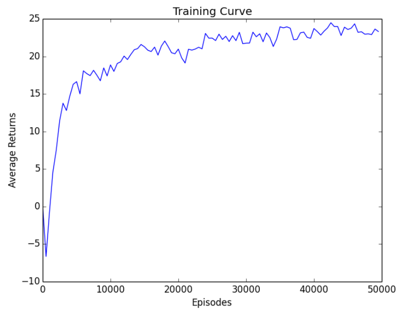

Figure 1: caption

2.
3.
4.

Problem 6

Problem 7

Problem 8