CSC411: Assignment 4

Due on Monday, April 2, 2018

Yian Wu, Zhou Quan

March 31, 2018

Part 1

Create a new environment, and play a game of Tic-Tac-Toe against yourself by calling the step(), and render() methods. Display the text output in your report.

```
Catherine's turns:
...
.x.
...
====
Stella's turns:
o..
.x.
...
====
Catherine's turns:
o..
xx.
...
====
Stella's turns:
o..
xxo
...
====
Catherine's turns:
o.x
xxo
...
====
Stella's turns:
o.x
xxo
o..
====
Catherine's turns:
o.x
xxo
ox.
====
Stella's turns:
oox
xxo
ox.
====
PEACE!
```

Part 2

(a)

```python
class Policy(nn.Module):
    """
    The Tic-Tac-Toe Policy
    """
    def __init__(self, input_size=27, hidden_size=64, output_size=9):
        super(Policy, self).__init__()
        self.hidden = torch.nn.Linear(input_size, hidden_size)
        self.predict = torch.nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.hidden(x))
        soft_layer = nn.Softmax(dim=1)
        x = soft_layer(self.predict(x))
        return x
```

(b)

The first 9 cell indicates whether an entry has been filled or not.
The next 9 cell indicates whether an entry has an ”x” on it or not.
The last 9 cell indicates whether an entry has a ”o” on it or not.

(c)

The 9-dimensional vector means the probability that this position will be occuppied. This policy is stochastic.

Part 3

Part 3(a): computing returns (10 pts)

```
def compute_returns(rewards, gamma=1.0):
    """
    Compute returns for each time step, given the rewards
      @param rewards: list of floats, where rewards[t] is the reward
                      obtained at time step t
      @param gamma: the discount factor
      @returns list of floats representing the episode's returns
          G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...

    >>> compute_returns([0,0,0,1], 1.0)
    [1.0, 1.0, 1.0, 1.0]
    >>> compute_returns([0,0,0,1], 0.9)
    [0.7290000000000001, 0.81, 0.9, 1.0]
    >>> compute_returns([0,-0.5,5,0.5,-10], 0.9)
    [-2.5965000000000003, -2.8850000000000002, -2.6500000000000004, -8.5, -10.0]
    """

    l = len(rewards)
    last_idx = len(rewards)-1
    Gt = np.zeros((1, l))

    Gt[0][last_idx] = rewards[last_idx]
    for i in range(last_idx-1, -1, -1):
        Gt[0][i] = rewards[i] + gamma * Gt[0][i+1]

    return Gt.tolist()
```

Part 3(b) When using Policy Gradient, we cannot compute the backward pass to update weights until the entire episode is complete. Explain why this is the case: why can we not update weights in the middle of an episode?

We can not compute the backward pass to update weights until the entire episode is complete. We can not update weights in the middle of an episode because when the rewards are not fully calculated, a biased return might be produced as the results.

Part 4

Part 4(a) (5 pts)

```python
def get_reward(status):
    """Returns a numeric given an environment status."""
    return {
        Environment.STATUS_VALID_MOVE: 1,
        Environment.STATUS_INVALID_MOVE: -10,
        Environment.STATUS_WIN: 20,
        Environment.STATUS_TIE: 0,
        Environment.STATUS_LOSE: -20
    }[status]
```

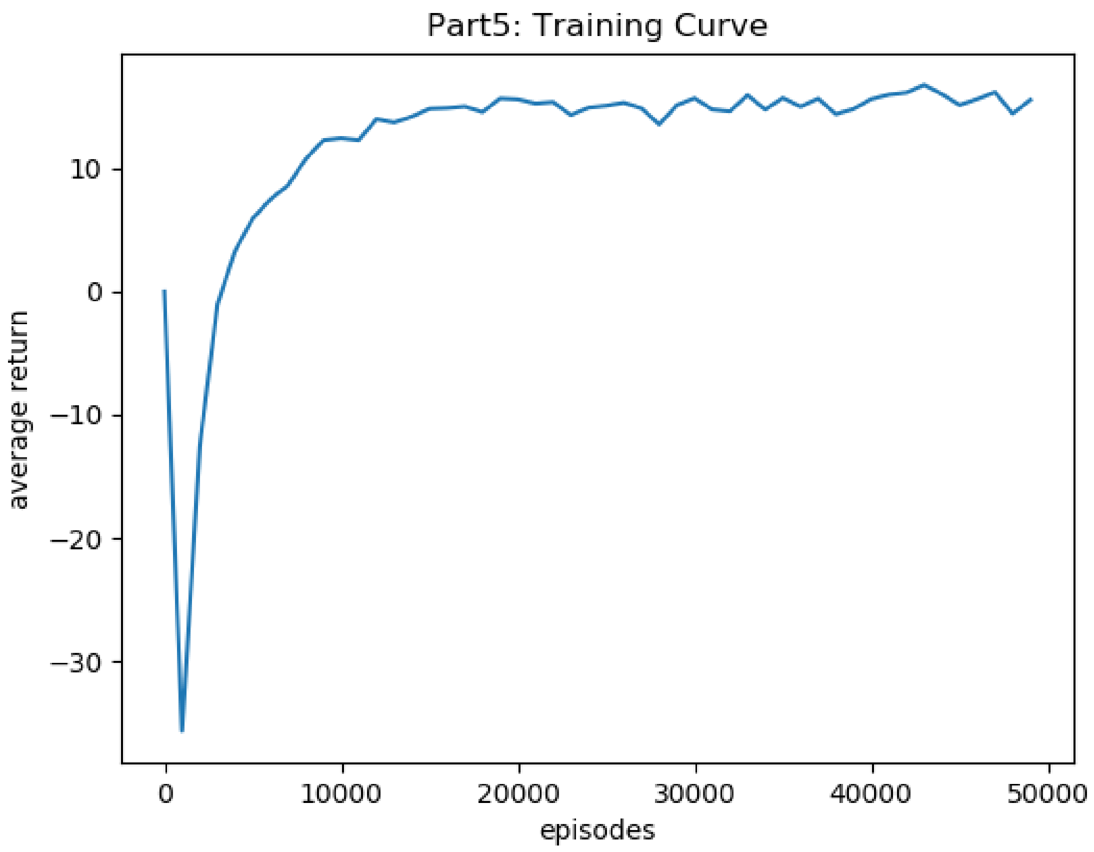
Part 4(b) (5 pts) Explain the choices that you made in 4(a). Are there positive rewards? Negative rewards? Zero rewards? Why? Explain how you chose the magnitude of the positive and negative rewards.

I give positive rewards for status: VALID MOVE and WIN. We prefer WIN status over VALID MOVE status because WIN status also includes VALID MOVE. Therefore, I set a larger magnitude for WIN status. INVALID MOVE are LOSE status will receive negative rewards. I set their magnitudes same to each other. Zero rewards are only given to status TIE. It is not as bad as LOSE or INVALID MOVE but we definitely do not want to encourage that.

Part 5

Part 5(a) (5 pts) Plot a training curve: your x-axis should be episodes, and your y-axis should be the average return. Clearly indicate if you have changed any hyperparameters and why.

I have trained for 50000 episodes. I have changed some hyperparameters for a better curve. I changed lr from 0.001 to 0.0005 and gamma to 0.9.



Part 5(b) (10 pts) One hyperparameter that you should optimize is the number of hidden units in your policy. Tune this hyperparameter by trying at least 3 values. Report on your findings

I have tried three values for the number of hidden unit in your policy. The results are shown below:
With hidden size 32: win count is 421, lose count is 66.
With hidden size 64: win count is 423, lose count is 59.
With hidden size 128: win count is 449, lose count is 41.
Therefore, the best hidden layer size is 128, which gives the best performance. From the results, a guess is that larger hidden layer size gives better performance.

Part 5(c) (5 pts) One of the first things that your policy should learn is to stop playing invalid moves. At around what episode did your agent learn that? State how you got the answer.

At around 22000th episode, my agent start to understand not to make invalid moves.
Episode #22000 has 0 invalid moves.
The invalid moves' count starts decreasing. I printed out the results and concluded this.

Part 5(d) (10 pts) Use your learned policy to play 100 games against random. How many did your agent win / lose / tie? Display five games that your trained agent plays against the random policy. Explain any strategies that you think your agent has learned.

WIN count: 72.

LOSE count: 22.

TIE count: 6.

```
---- Displaying game #62 ------
..x
...
..o
====
..x
..x
.oo
====
..x
oxx
.oo
====
..x
oxx
xoo
====
---- Displaying game #73 ------
.o.
x..
...
====
.oo
x.x
...
====
.oo
xxx
...
====
---- Displaying game #74 ------
.xo
...
```

```
...
====
xxo
.o.
...
====
xxo
.o.
...
====
xxo
oo.
..x
====
xxo
ooo
x.x
====
---- Displaying game #79 ------
...
x..
.o.
====
..o
x.x
.o.
====
..o
xxx
.o.
====
---- Displaying game #94 ------
.x.
.o.
...
====
.xx
.oo
...
====
xxx
.oo
...
====
```

The strategies that I think my agent has learned are:

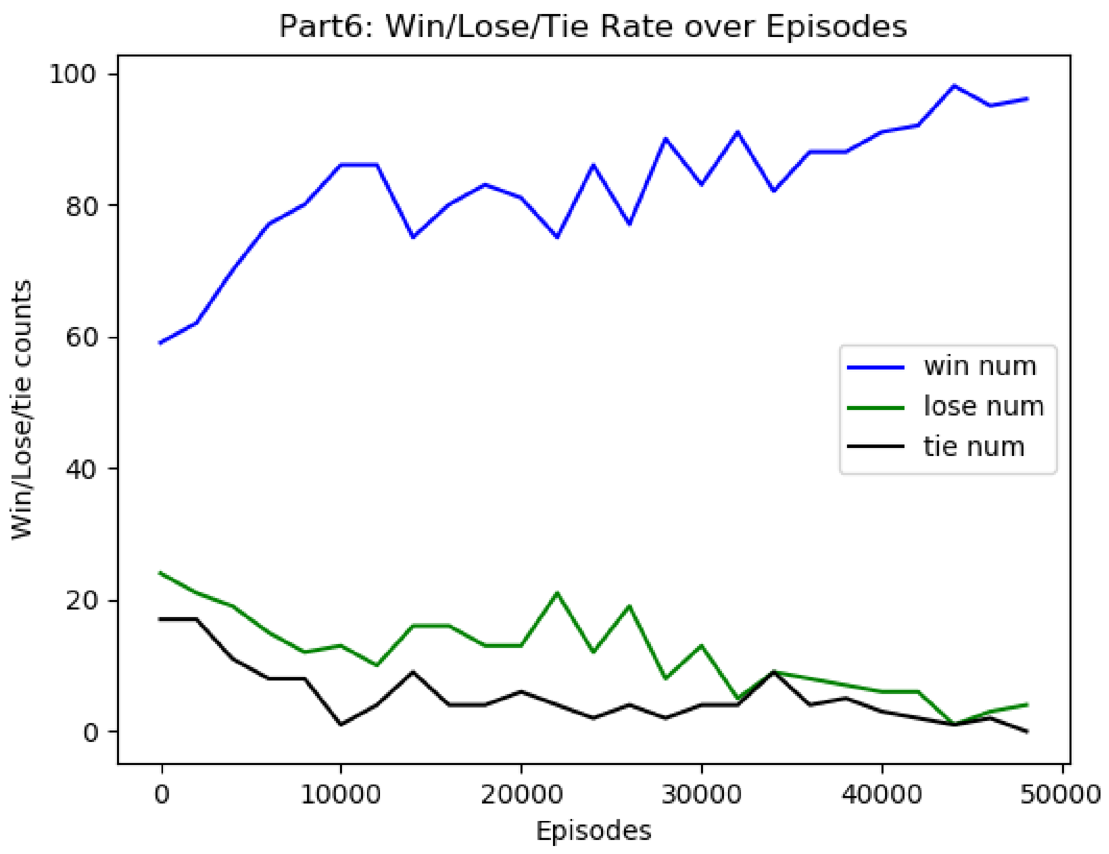
1. trying to take over the spot the opposite side of the other player's move

1. trying to take over the spot in the middle of a line to give itself more space to do next step and less chance for the other player to win.

Part 6

Part 6: Win Rate over Episodes (10 pts) Use the model checkpoints saved throughout training to explore how the win / lose / tie rate changed throughout training. In your report, include graphs that illustrate this, as well as your conclusions in English.

The performance is shown in graph below. We can clearly see that WIN rate is increasing while the LOSE and TIE rates are decreasing.



Part 7

First Move Distribution over Episodes (10 pts)

Part 8

Limitations (5 pts) Your learned policy should do fairly well against a random policy, but may not win consistently. What are some of the mistakes that your agent made?

When there's only two spots available on a line (which means it is impossible to win on that line already), the agent still will step onto those spot and fill in that two spots.
The agent sometimes focus on trying to prevent the other player to win more than completing steps to win itself. In real world, two things are both important to the winning.
The agent is not great at notice spots cross a line. like following situation:
. . o
. . . <= x should go here to prevent "o" win but my agent might not do so
. . o