CSC411: Project 4

Due on Sunday, April 2, 2018

Ying Yang

April 1, 2018

Part 1

Environment

The grid is represented by horizontal, vertical and diagonal indices. The coordinates are the indices from 0 - 8. The win-set is represented by a fix matrix in which the first row represents 3 sets of horizontal rows for which a player wins the game, the second row represent the vertical column, and the third row represents the diagonal of the tictactoe.
The attributes turn represents whose turn to play the game (player1 or player 2). The attribute done means whether the game is over (when a player wins, or a tie occurs).
Play a game of TicTacToe by calling the step(), and render() methods.

Listing 1: Compute network function

```
env.step(0)
(array([1, 0, 0, 0, 0, 0, 0, 0, 0]), 'valid', False)
env.render()
x..
...
...
====
env.step(2)
(array([1, 0, 2, 0, 0, 0, 0, 0, 0]), 'valid', False)
env.render()
x.o
...
...
====
env.step(4)
(array([1, 0, 2, 0, 1, 0, 0, 0, 0]), 'valid', False)
env.render()
x.o
.x.
...
====
env.step(8)
(array([1, 0, 2, 0, 1, 0, 0, 0, 2]), 'valid', False)
env.render()
x.o
.x.
..o
====
env.step(6)
(array([1, 0, 2, 0, 1, 0, 1, 0, 2]), 'valid', False)
env.render()
x.o
.x.
x.o
====
env.step(1)
(array([1, 2, 2, 0, 1, 0, 1, 0, 2]), 'valid', False)
env.render()
xoo
.x.
```

```
x.o
====
env.step(3)
(array([1, 2, 2, 1, 1, 0, 1, 0, 2]), 'win', True)
env.render()
xoo
xx.
x.o
====
```

Part 2

Complete the implementation so that policy is a neural network with one hidden layer

Part 2 (a)

Listing 2: Policy implementation

```
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__()
# TODO
self.linear_f1 = nn.Linear(input_size, hidden_size)
self.linear_f2 = nn.Linear(hidden_size, output_size)

def forward(self, x):
# TODO
h = F.relu(self.linear_f1(x))
out = F.softmax(self.linear_f2(h))
return out
```

Part 2 (b)

Listing 3: Policy

```
policy = Policy()
state = np.array([1,0,1,2,1,0,1,0,1])
state = torch.from_numpy(state).long().unsqueeze(0)
state = torch.zeros(3,9).scatter_(0, state, 1).view(1, 27)
print(state)
```

Listing 4: output

```
               Columns 0 to 12
               0     1     0     0     0     1     0     1     0     1     0     1
0


               Columns 13 to 25
               1     0     1     0     1     0     0     0     1     0     0     0
0


               Columns 26 to 26
               0
               [torch.FloatTensor of size 1x27]
```

State what each of the 27 dimensions mean

Part 2 (c)

Explain what the value in each dimension means. The value in each dimension means Is this policy stochastic or deterministic?
The policy is stochastic.

Part 3

Part 3a

Implement the compute_returns function

Listing 5: output

```
l = len(rewards)
rewards = np.array(rewards)
gammas = np.array([gamma ** (i) for i in range(l)])

G = []
for i in range(l):
    G.append(sum(rewards[i:] * gammas[:l - i]))
return G
```

Part 3b

Explain why can we not update weights in the middle of an episode

When we are in the middle of the episode, we haven't yet finished computing the final result for reward. As a result, if we compute backward pass before we got the reward result, we could get inaccurate result, since if we compute the gradient based on inaccurate number. Therefore, we should update the weights after the episode.

Part 4

Part 4(a)

Listing 6: modified function

```
def get_reward(status):
"""Returns a numeric given an environment status."""
return {
Environment.STATUS_VALID_MOVE: 1,
Environment.STATUS_INVALID_MOVE: -25,
Environment.STATUS_WIN: 50,
Environment.STATUS_TIE: 0,
Environment.STATUS_LOSE: -50
}[status]
```

Part 4(b)

Explain the choices that you made in 4(a)

The environments status were given weight based on the principle of rewarding biggest on win and penalize biggest on lose. In the same way, reward on status that contribute to valid moves and penalize on status that contribute to invalid moves. As a result, valid move is considered as a good move, and was given a positive number: 1. The invalid move is bad, so it should be penalized, and it was given: -25. The win status should be rewarded big, so it was given a big positive number: 50. The tie status is given 0 since it doesn't contribute to win or lose. Finally, the lose status was given -50 to get penalized.

Part 5

Part 5a

Plot the training curve of the tictactoe model.
Hidden units = 64 units

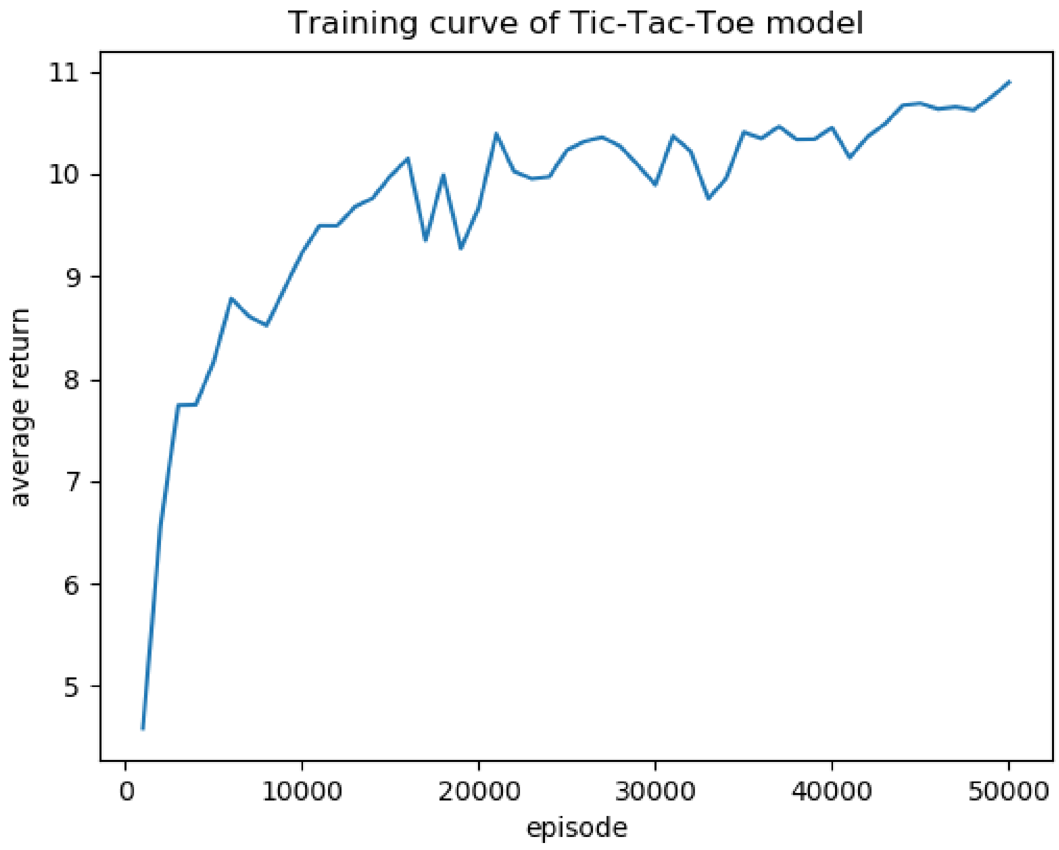

Figure 1: training curve of the tictactoe model, with 64 hidden units

Part 5b

Plot the training curve of the tictactoe model, by tuning the hyperparameter hidden units with 3 different values.
1. Hidden units = 128 units

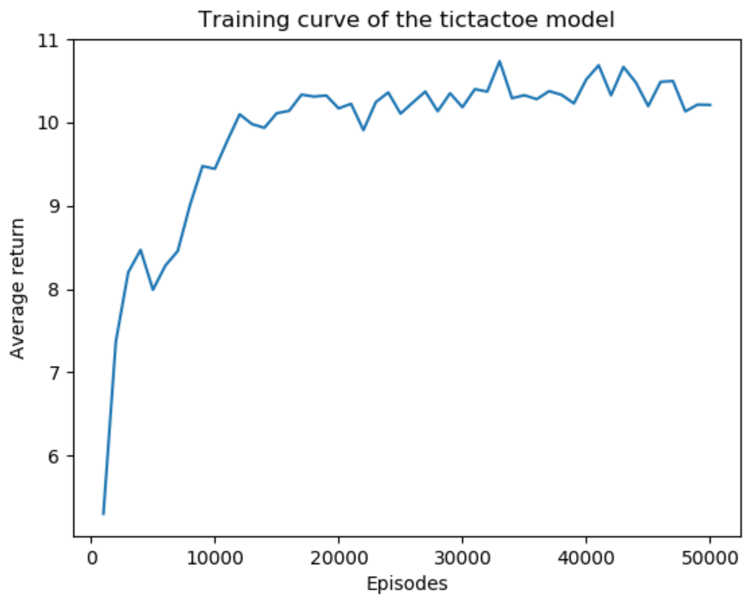

Figure 2: training curve of the tictactoe model, with 128 hidden units

2. Hidden units = 16 units

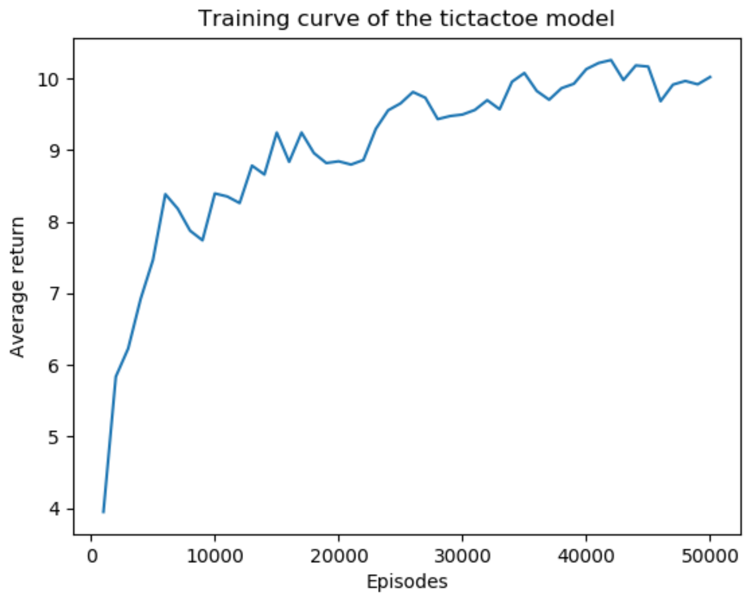

Figure 3: training curve of the tictactoe model, with 16 hidden units

3. Hidden units = 32 units

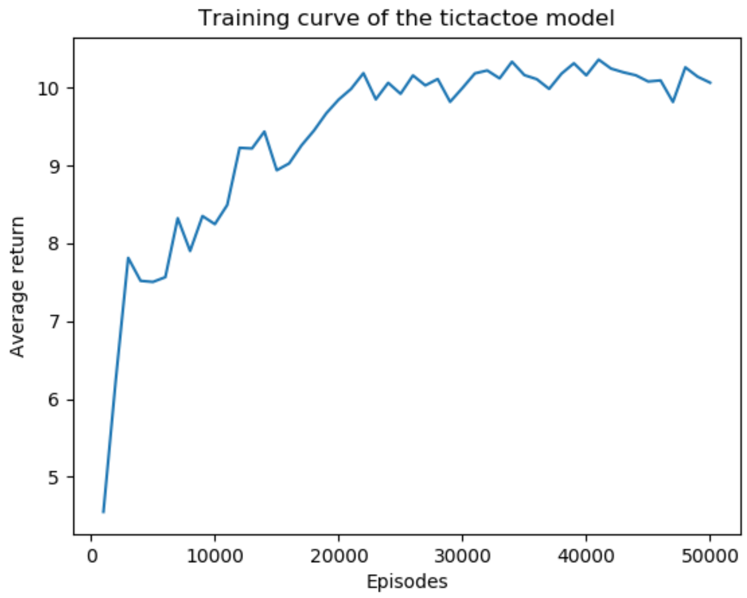

Figure 4: training curve of the tictactoe model, with 32 hidden units

After experimenting with 3 different number of hidden units : 16, 32, 128, it was possible to observe that the hyperparameter setting to 128 gave the best results.

Part 5c

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.
The agent learned to stop playing invalid moves at around 4000th iteration. This can be observed by counting the number of invalid moves for 100 random games. By inspecting the result, starting at 3000th iteration, the number of invalid moves dramatically decreased from 126 invalid moves to only 27 invalid moves.

Part 5d

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.

Part 6

Part 6a

Use the model checkpoints saved throughout training to explore how the win / lose / tie rate changed throughout training

Graph of win/lose/tie rate changed throughout the training

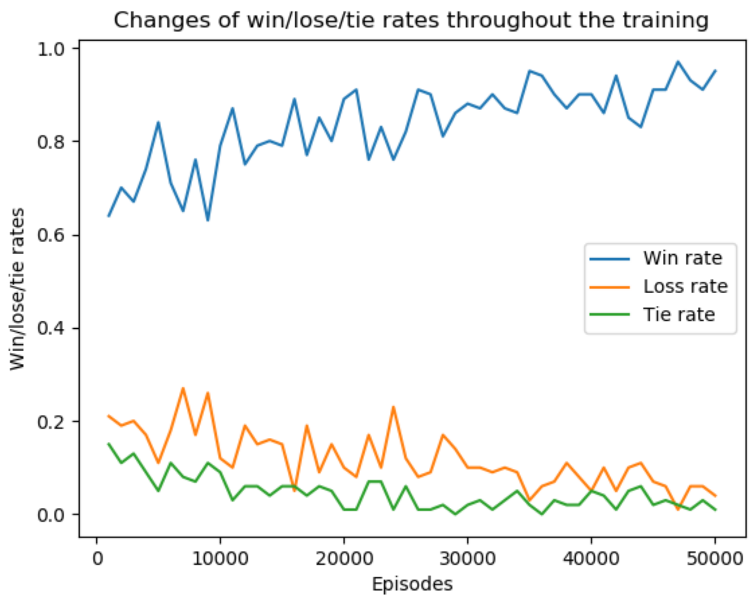

Figure 5: win/lose/tie rate changed throughout the training

Part 7

What has your model learned?
Does the distribution make sense?

Part 8

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?