CSC411: Project #4

Due on Monday, April 2, 2018

Hunter Richards, Yu-Chien Chen

Thursday, March 29, 2018

Problem 1

For this project on Tic-Tac-Toe the grid is represented as a one dimensional array where each element is either a 0, 1, or 2, which represents an empty slot, an X, or an O, respectively. The attribute `turn` represents which player's turn it is, 1 for X and 2 for O. The attribute `done` represents whether the game is finished or not (i.e. the game has resulted in a player winning or a tie) where `done = False` when the game is still in progress and `done = True` when the game has concluded.

The following output shows the progression of a game of Tic-Tac-Toe:

```
.x.
...
...
====
.x.
...
..o
====
.x.
...
x.o
====
.x.
.o.
x.o
====
xx.
.o.
x.o
====
xx.
oo.
x.o
====
xxx
oo.
x.o
====

X wins!
```

All code used in this assignment is implemented in Python 3.6

Problem 2

a) The `Policy` class was modified to have one hidden layer and a softmax over the output:

```
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.features = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.Linear(hidden_size, output_size),
            nn.Softmax(dim=1))

    def forward(self, x):
        return self.features(x)
```

b) The board is 3×3 and each position can be empty, filled with an X, or filled with an O. Hence, the dimension of the state vector is $3 \times 3 \times 3 = 27$. Each of the elements in the 27-dimensional state vector represent a single state that a position on the board can occupy.

For example:

If v is the state vector, and $v_0 = 1$, then the top left square of the board is empty and $v_9 = v_{18} = 0$.

c) Each value in each dimension represents how confident the model is in choosing that corresponding square for each move (or in other words, the probability that the corresponding square is a good move). Therefore, this policy is stochastic.

Problem 3

a) The function below takes a list of rewards r_t and computes the returns, $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$

```
def compute_returns(rewards, gamma=0.9):
    returns = list()
    for i in range(len(rewards)):
        returns.append(sum(r*gamma**t for t,r in enumerate(rewards[i:])))

    return returns
```

b) We cannot compute the backward pass to update weights in the middle of an episode since the average cost J needs to be calculated in order to update the weights. However, J is calculated by summing the product of all probability and expected rewards:

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

where $d^{\pi_\theta}(s)$ is the probability of the agent being in state s if we follow policy π_θ for a long time and $V^{\pi_\theta}(s)$ is the expected total reward if started from state s. We want states and actions that lead to high rewards to have high probability and they are found when we look at all possible actions.

Problem 4

a) The modified get_reward function:

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

b) The first decisions made were to assign values to `STATUS_WIN` and `STATUS_LOSE`. The return should be very positive for a win compared to the return of other moves. Conversely, the return for a loss should be very negative. Likewise the agent should never choose `STATUS_INVALID_MOVE` and its return should also be assigned a very large negative number.

This leaves `STATUS_VALID_MOVE` and `STATUS_TIE`. A tie is a neutral state, there is no gain and no loss. Therefore a tie has a return of 0. We are not given any other information about the game state so we must assume that making a valid move has an equal chance of resulting in a loss, a win, or a neutral state such as a tie or a continuation of the game. From this, the return of a valid move has an expected value of 0 and the return of `STATUS_VALID_MOVE` is assigned as such.

A magnitude of 1 was assigned to positive and negative values because an invalid move or loss is as "bad" as a win is "good". Moreover, all other values are 0 so a magnitude of 1 is simple and sufficient for establishing the relative scale between the return values.

Problem 5

a) The discount rate γ was changed from 1 to 0.9 to motivate the agent to make high return decisions early. Tic-tac-toe has a finite number of states, so the calculated reward will still be stable even for $\gamma = 1$, and there is only one state that gives a positive return (i.e. a win), but the change did still result in a slightly better average return during training. The following graph displays the learning curve for the policy:

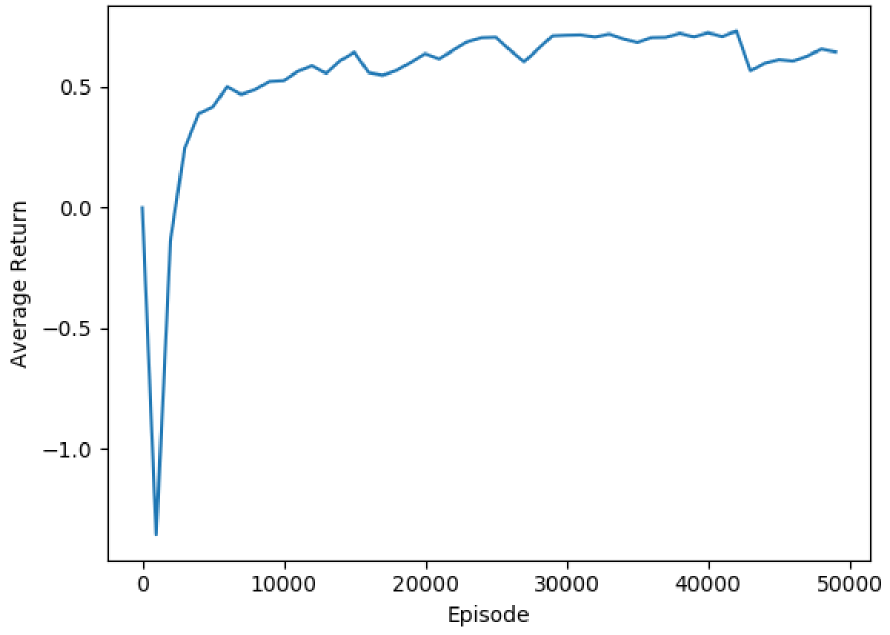
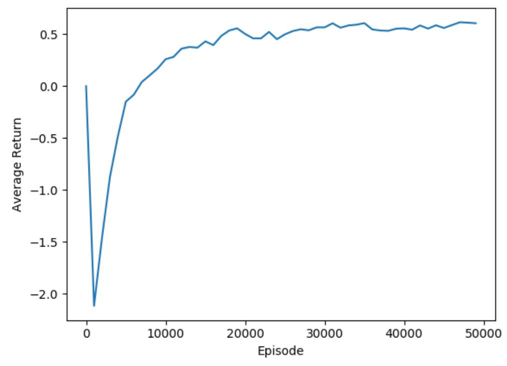
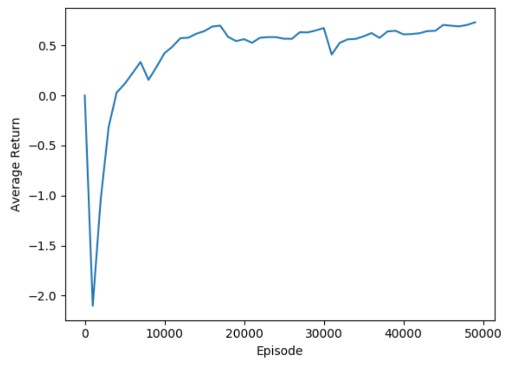


Figure 1: `hidden_size = 64`

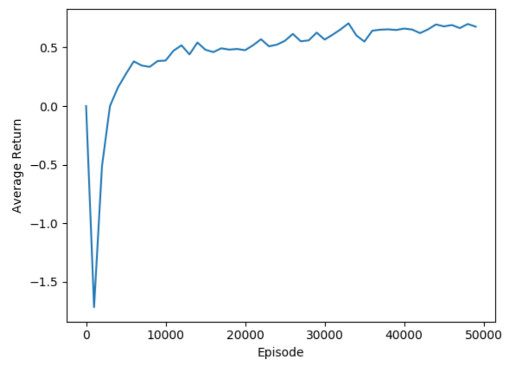
b) Several values for `hidden_size` were investigated. The following plots were produced:



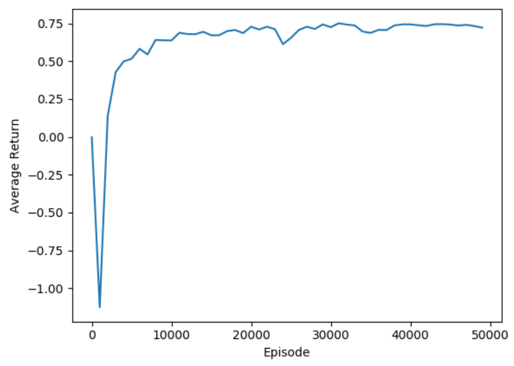
(a) `hidden_size = 8`



(b) `hidden_size = 16`



(c) `hidden_size = 32`



(d) `hidden_size = 96`

Figure 2

As `hidden_size` increases the policy plateaus at earlier episodes. Going beyond 96, however, causes training to become very slow or even stall. Thus, `hidden_size = 96` was chosen as a balance between getting higher returns sooner and training time.

c) There is no point where the number of invalid moves becomes 0 and remains 0 thereafter. Rather, the number of invalid moves steadily decreases and then stays low (0-5 invalid moves per 1000 episodes). There are occasionally points where the number of invalid moves will spike, but this is rare. At roughly 20000 episodes into training, the model has learned not to make invalid moves. This is done by checking `status == env.STATUS_INVALID_MOVE` after every move, updating a running sum, and checking to see if this sum in the range 0-5 for every 1000 episodes.

d) Against an opponent playing randomly the learned policy won 95 games, tied 4 games, and lost 1 game out of 100 games total. The following is output from 5 of the games played:

```
Game 0      Game 20     Game 40     Game 60     Game 80
..x         ..x         ..x         ..x         ..x
...         ...         ...         o..         ...
.o.         o..         .o.         ...         ..o
====        ====        ====        ====        ====
..x         x.x         ..x         ..x         ..x
.xo         ...         .x.         o..         ...
.o.         o.o         .oo         x.o         xoo
====        ====        ====        ====        ====
..x         xxx         ..x         ..x         ..x
.xo         ...         .x.         ox.         .x.
xo.         o.o         xoo         x.o         xoo
====        ====        ====        ====        ====
```

From the output it appears that the policy will always try to win diagonally, starting from the top right corner, then the bottom left, then the centre. While going for the corners is a common strategy in Tic-Tac-Toe[1], it makes the policy very predictable to always choose the same order of positions. If the policy was not always allowed to go first it would likely be much less successful.

[1]https://en.wikipedia.org/wiki/Tic-tac-toe#Strategy

Problem 6

One can see from the graph below that the win rate increases and both the tie and loss rate decreases over time:

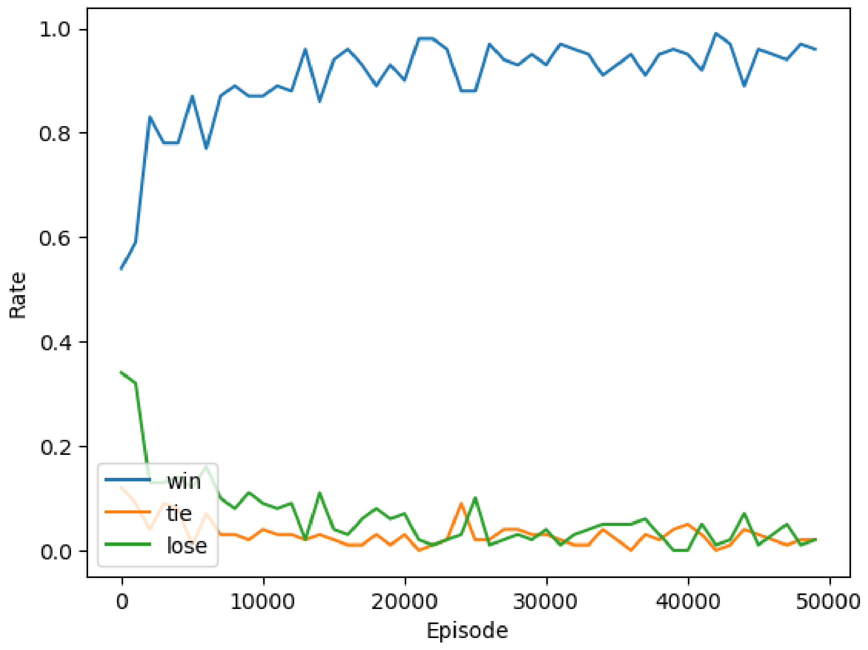


Figure 3: Win/ Loss/ Tie rate over episodes

It can also be observed that the curve is quite noisy. This is perhaps due to the stochastic nature of the policy model. Stochasticity, such as in mini batch gradient descent, often causes a lot of fluctuation in the model's output because of the injected randomness.

Problem 7

The learned distribution over the first move for our final trained model (episode 47000) is:

```
[7.67e-06 1.70e-09 9.99e-01 1.00e-08 8.95e-07 1.089e-11 2.42e-04 2.26e-12 2.03e-10]
```

Index 2 has a value of 9.99e-01, the highest value of all 9 elements in the distribution. This indicates that the model has learned to choose the top right corner as its first move to increase its chances of winning. This makes sense because it aligns with the observations from part 5(d) where it was mentioned that choosing corners is a common strategy for Tic-tac-toe.

Below is a snippet of the output that demonstrates the model's choice of the best first move converges to index 2 (third element in the list) during training:

```
Episode 0
 [0.148 0.122 0.106 0.125 0.092 0.102 0.075 0.143 0.081]

Episode 1000
 [0.033 0.195 0.412 0.114 0.073 0.078 0.060 0.025 0.005]

Episode 2000
 [0.054 0.328 0.473 0.043 0.028 0.003 0.060 0.003 0.003]

Episode 3000
 [0.017 0.097 0.094 0.020 0.279 0.001 0.488 0.001 0.001]

Episode 4000
 [5.12e-04 1.44e-03 8.71e-03 1.76e-04 3.79e-03 1.25e-05 9.85e-01 1.23e-05 4.46e-06]
...
Episode 9000
 [5.54e-05 5.77e-06 4.40e-04 2.48e-06 3.33e-06 1.06e-07 9.99e-01 1.02e-07 1.43e-08]

Episode 10000
 [6.81e-05 8.65e-05 5.32e-01 2.61e-04 7.12e-06 2.48e-06 4.67e-01 1.45e-06 3.40e-07]

Episode 11000
 [5.92e-05 4.56e-04 2.57e-01 5.27e-04 7.45e-06 3.20e-06 7.41e-01 1.30e-06 3.00e-07]
...
Episode 15000
 [7.87e-07 8.91e-06 9.94e-01 1.37e-05 8.04e-06 1.31e-07 5.68e-03 1.22e-07 4.00e-09]

Episode 16000
 [2.08e-07 1.23e-05 9.98e-01 4.41e-06 1.73e-06 3.32e-08 1.15e-03 4.46e-08 1.55e-10]

Episode 17000
 [1.04e-08 2.12e-07 9.99e-01 1.80e-07 5.06e-08 2.32e-09 1.27e-05 1.95e-09 4.83e-12]

Episode 18000
 [9.73e-09 1.56e-07 9.99e-01 9.35e-08 4.92e-08 4.45e-10 1.54e-06 8.03e-10 1.65e-12]
```

Problem 8

The policy does well against a random opponent, but it can still lose. A critical mistake that the policy made was that it cannot recognize when the opponent is about to win. For example, the opponent may be about to get three in a row, but the policy does nothing to block the opponent's streak. This is especially the case when the opponent disrupts the policy's diagonal play. This forces the policy to take extra moves to win and it loses the advantage of going first. The following outputs show how the policy knows that it should go for the corners to win, but does not understand how its opponent can also win:

```
    Episode 30000    Episode 30000    Episode 30000    Episode 47000
       Game 38          Game 59          Game 83           Game 5
        ..x               ..x              o.x               ..x
        ..o               ...              ...               o..
        ...               o..              ...               ...
        ====              ====             ====              ====
        ..x               oxx              o.x               ..x
        .xo               ...              .x.               oo.
        o..               o..              o..               x..
        ====              ====             ====              ====
        oxx               oxx              oxx               x.x
        .xo               o..              ox.               ooo
        o..               o.x              o..               x..
        ====              ====             ====              ====
        oxx
        oxo
        o.x
        ====
```