CSC411: Project 4

Due on Sunday, April 2, 2018

Ying Yang

March 29, 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

Part 4

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: -5,
            Environment.STATUS_WIN         : 10,
            Environment.STATUS_TIE         : 4,
            Environment.STATUS_LOSE        : -1
        }[status]
```

Part 5

Write vectorized code that performs gradient descent with momentum, and use it to train your network. Plot the learning curves

Learning curve with momentum:l

How new learning curves compare with gradient descent without momentum: Without the momentum, the performance of correct guesses increased slower than if momentum were used. For example, as we can observe in the learning curve, the performance of training set only increased to 92% after 800 iterations, wheareas with momentum term set to 0.99, the performance correcnesses increased to the same percentage by only using 400 iterations, because it moved faster when gradient consistenly points to one direction.

Code added in the gradient descent:

In the train_neural_network method, method signature was modified to take two extra terms: type and momemtum term. The rest of code remains unchanged as in part 4.

Listing 7: Gradient descent with momentum

```
train_neural_network(image_prefix, type="None", momentum_term=0):

v_w = 0 # for momentum
v_b = 0
if type == "momentum":
    v_w = momentum_term * v_w + alpha * df_w(train_set.T, W, b, train_label.T)
    W -= v_w
    v_b = momentum_term * v_b + alpha * df_b(train_set.T, W, b, train_label.T)
    b -= v_b

if __name__ == "__main__":
     momentum_term = 0.99
     train_neural_network("part5", "momentum", momentum_term)
```

Part 6

Part 6a

Plot the contour of the cost function

Part 6b

Part 6c

Part 6d

Part 7

Part 8