CS446: Machine Learning **Spring 2018**

Machine Problem 12

Handed Out: January. 16, 2018 *Due: April. 24, 2018 (11:59AM Central Time)*

Note: The assignment will be autograded. It is important that you do not use additional libraries, or change the provided functions input and output. Start early! Training takes about 3 days on CPU.

Part 1: Setup

- Remove connect to a EWS machine.

```
ssh (netid)@remlnx.ews.illinois.edu
```

- Load python module, this will also load pip and virtualenv

```
module load python/3.4.3
```

- Reuse the virtual environment from mp0.

```
source ~/cs446sp_2018/bin/activate
```

- Copy mp12 into your svn directory, and change directory to mp12.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp12 .
cd mp12
```

- Install the requirements through pip.

```
pip install -r requirements.txt
```

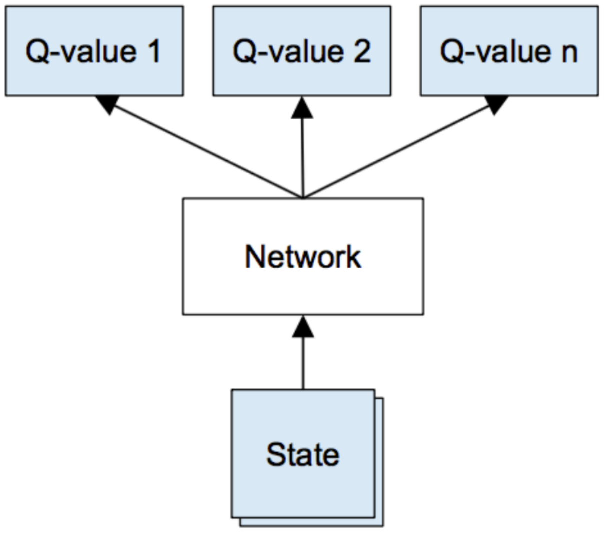
- Prevent svn from checking in the checkpoint directory.

```
svn propset svn:ignore saved_networks_q_learning .
```

Part 2: Exercise

In this part, you will train an AI agent to play the Pong game using Q-learning. On the low level, the game works as follows: we receive the last 4 image frames which constitute the state of the game and we get to decide if we want to move the paddle to the left, to the right or not to move it (3 possible actions). After every single choice, the game simulator executes the action and gives us a reward: either a +1 reward if the ball went past the opponent, a

Figure 1: Q-Network



-1 reward if we missed the ball and 0 otherwise. Our goal is to move the paddle so that we get lots of reward.

Q-learning

This exercise requires you to use Q-Learning to train a convolutional neural networks for playing the Pong game. We consider the deep Q-network in the figure below. Follow the instructions in the starter code to fill in the missing functions. Include a learning curve plot showing the performance of your implementation. The x-axis should correspond to the number of episodes and the y-axis should show the reward after every episode. Your agent should be performing well after 4-5m steps.

Function Description

- *get_action_index*: During the observation phase, this function returns a randomly chosen action. Beyond the observation phase, the action with the highest Q-value is returned with probability $(1 - epsilon)$ and a random action is chosen with probability *epsilon*. *epsilon* is a temperature parameter that controls the trade-off between exploration and exploitation.

- *scale_down_epsilon*: During the observation phase, *epsilon* is set to 1. Beyond the observation phase, *epsilon* is scaled down with $\frac{(INITIAL\_EPSILON - FINAL\_EPSILON)}{EXPLORE}$ as long as *epsilon* is larger than $FINAL\_EPSILON$.

- *run_selected_action*: Feed the selected action into the game simulator to obtain the next frame, the reward and a boolean $Terminal$ indicating whether the game terminated. This function returns the next state, reward and the boolean $Terminal$. The next state is produced by concatenating the four most recent frames, in order to capture the motion of the ball and paddle.

- *compute_target_q*: This function computes the target Q-value for all samples in the batch. Distinguish two cases depending on whether the next state is a terminal state.

Relevant Files: *q_learning.py*

Part 3: Writing Tests

In *test.py* we have provided basic test-cases. Feel free to write more. To test the code, run:

```
nose2
```

Part 4: Submit

Submitting the code is equivalent to committing the code. This can be done with the following command:

```
svn commit -m "Some meaningful comment here."
```

Lastly, double check on your browser that you can see your code at

```
https://subversion.ews.illinois.edu/svn/sp18-cs446/(netid)/mp12/
```

Figure 2: Pong Game