CS440 MP4 Report

Andong Jing	ajing2	- 3 credit	In charge of 1.3.2
Siping Meng	smeng10	- 3 credit	In charge of 1.1/1.2/1.3.1
Siyu Tao	siyutao2	- 3 credit	In charge of 2.1/2.3.1/2.3.2

1. Q-Learning

1.1 Single-Player Pong

1. Report and justify your choices for α, γ, exploration function, and any subordinate parameters. How many games does your agent need to simulate before it learns a good policy?

For single-player Pong, I chose learn rate constant C = 200, γ = 0.9 and exploration function = max(0.1,1-math.log2(epoch*10)/20), where epoch is the number of epochs it is currently running. For example, if currently the agent is in the 10000th training, and the exploration function result would be max(0.1,log2(10000*10)/20) = max(0.1,0.1695) = 0.1695. α is related to α(N)=C/(C+N(s,a)), which is indicated in the website.

The reason why I will choose these parameters is that learning rate determines to what extent newly acquired information overrides old information, if α == 1 then the agent consider only the most recent information, and α == 0 will let the agent learn nothing. According to experiment, a large N will let the agent learn very slow (it would reach 3 average bounces after 20k games, but an agent with N == 200 will learn 3 average bounces within 10k games). Meanwhile, a small N will let the agent converge very fast (converges to around 4 average bounces after 20k games, but an agent with N == 200 will learn 9 – 10 average bounces within 80k games).

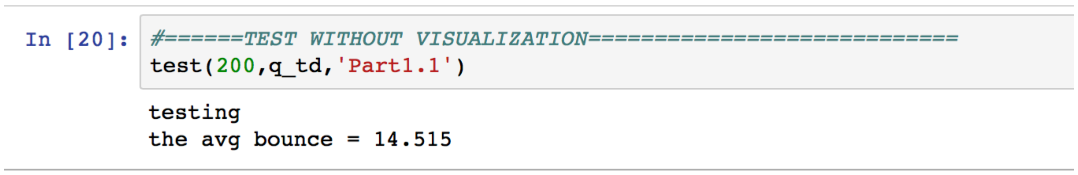
γ determines the importance of future rewards, so I would like to let the agent to be long sighted so I set the discount factor to be 0.9.

For the exploration function, I want the agent explore a lot at the very first to try a lot of possibilities and then exponentially decay after the agent has played several epochs (around 30k games to reach minimum exploration value).

Basically, our TD learning agent and SARSA learning agent would converge after 80000 training which is better than 100k training games stated in the website. And the average in current 1000 round is about 10 bounces.

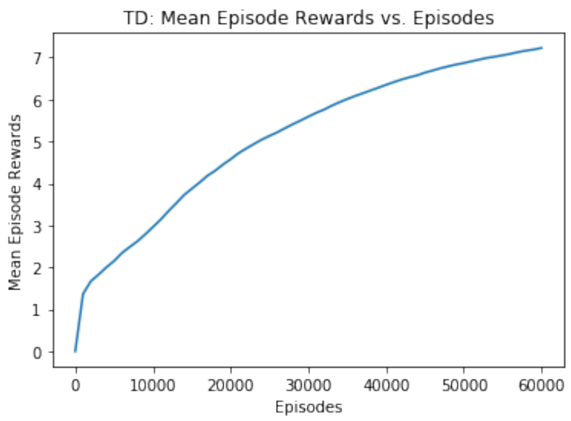
2. Use α, γ, and exploration parameters that you believe to be the best. After training has converged, run your algorithm on 200 test games and report the average number of times per game that the ball bounces off your paddle before the ball escapes past the paddle.

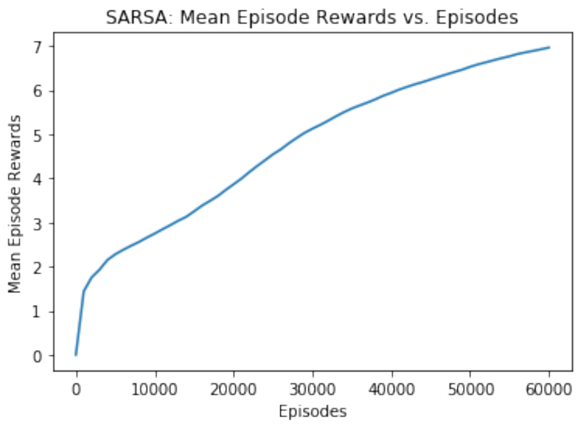
The average number of ball bounce the paddle is 14.515 for the 200 game testing.


```
In [20]: #======TEST WITHOUT VISUALIZATION============================
         test(200,q_td,'Part1.1')

         testing
         the avg bounce = 14.515
```

3. Include "Mean Episode Rewards vs. Episodes" plot for both Q-Learning and SARSA agents and compare these two agents.





For the two learning methods run over 60k games, we can see that SARSA learning is faster than TD learning at the beginning but has a slower slope after the first 10k games. On the other hand, TD learning has a smoother curve on the graph than SARSA. However, both of the two learning methods perform the same ability to bounce after 100k training epochs.

1.2 Environment Changed

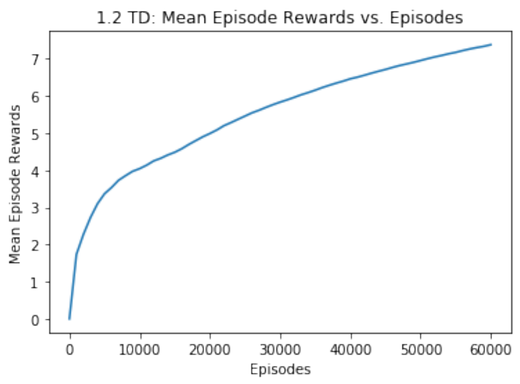
1. Describe the changes you made to your MDP (state space, actions, and reward model), if any, and include any negative side-effects you encountered after doing this.

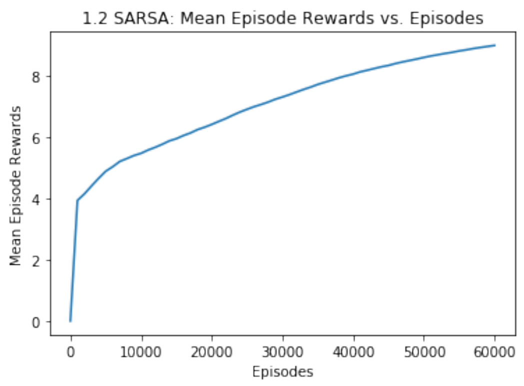
The only thing I changed in this part is the reward model. That is the condition for when the ball will bounce off the paddle or pass the paddle and get reward. I kept everything else the same as in Part1.1 since we are still in that pong playground. I cannot see any negative side-effects I encountered after doing this because after 100k games of training, the agent will master this side's game.

2. Describe your method of training agent A and tell us why it works.

I used a for loop to indicate how many epochs I need to train for this agent and then a while True loop to keep running the game until the reward is -1. During the infinite while loop, I have a function named **proceed_one_step** to proceed one step after this state. That is, change the ball's location to its next location by incrementing x and y coordinate by its velocities. I also using exploration function to choose the action this agent will perform at this state. After I pass all the parameters into this function, it will return a next_state state object which contains all the information I need including the reward. Then, according to these information I update the Q table and count table. I will also check the reward to see if the agent successfully bounce the ball or not.

3. Include two "Mean Episode Rewards vs. Episodes" plots and compare these two agents.



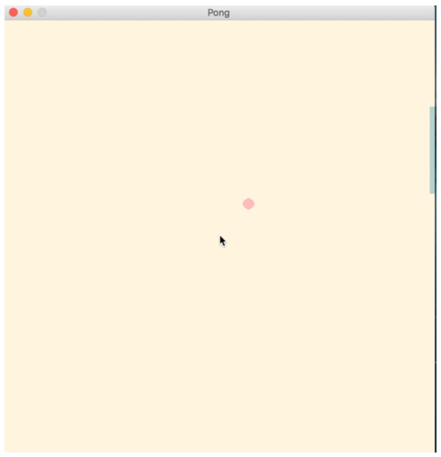


I got these two plots after 60k training for each agent. As we can clearly see both of the agents use less episodes to master the game than what they did in Part1.1. Moreover, SARSA agent has a perfect performance at the very beginning of the training than TD agent.

1.3 Part1 Extra Credit – GUI

1. Create a graphical representation of your Pong game. Create and include an animation of your agent playing the game.

We include this animation in the zip file named "GUI for PONG.mov".



2 If you created either a GUI or a console-based implementation, allow for a human to play against the AI. Are you able to defeat it? What are its strengths and weaknesses? Describe your discoveries.

Yes, we can defeat it occasionally, but we lose for most of times (intimately, A.I would win eight out of ten games). By our observation, compare to the fact that human player might miss some easy incoming balls by mistakes, our A.I. makes very few mistakes for “easy balls”. A.I.’s performance is more stable than human player. However, human player does better in some “difficult balls”, especially when the ball has a large vertical speed. What’s more, human players can gradually notice and learn the weakness of the A.I. while our A.I. doesn’t really have this function.
However, I think our A.I. is really good overall and it definitely plays better than me!

2. Deep Learning (Pong)

2.1 Cloning the Behavior of an Expert Player

1. What is the benefit of using a deep network policy instead of a Q-table (from part 1)? (Hint: think about memory usage and/or what happens when your agent sees a new state that the agent has never seen before).

Since our agent now uses supervised learning techniques as opposed to a lookup Q-table, we are now able to deal with continuous state spaces. When the agent sees a new state, the network based on training data will give an output with comparatively high confidence. In contrast, Q-table does not give an educated guess under such situation, but rather a random guess. The memory usage will also be disastrous if the Q-table has explored the environment fully (knows every state). The Q-table is enormous in such situation. The convergence of a deep network is also faster, given that Q-table needs to know enough states to be good. In situations where exhaustive exploration of the environment is impossible, a deep network policy is greatly preferred over a Q-table.

2. Implement the forward and backwards functions of a neural network and give a brief explanation of implementation and architecture (number of layers and number of units per layer).

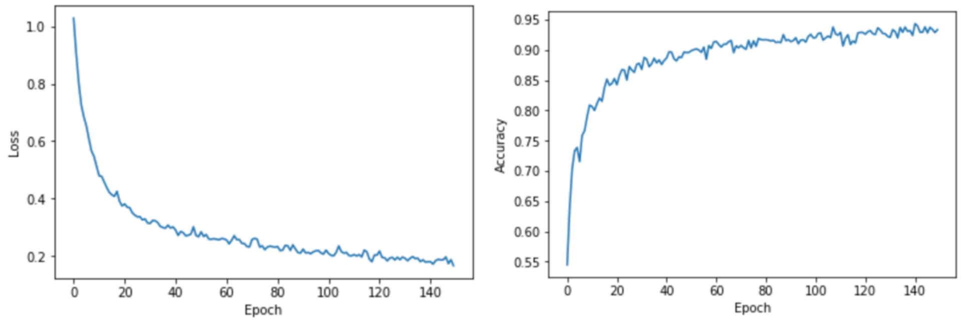
The team followed the recommendation of Ryley and used a five-layer architecture (including the 5-element input-layer and 1-element output-layer). The three hidden layers all have 256 units per layer. (256,256,256)

3. Train your neural network using minibatch gradient descent. Report the confusion matrix and misclassification error. You should be able to get an accuracy of at least 85% and probably 95% if you train long enough. Report you network settings including the number of layers, number of units per layer, and learning rate.

```
[[2984  154  196]
 [  41 2239   63]
 [  81  129 4113]]
Classification Accuracy: 0.9336
```

The confusion matrix is shown above. The classification accuracy is 93.36% after 150 training epochs. The team followed the recommendation of Ryley and used a five-layer architecture (including the 5-element input-layer and 1-element output-layer). The three hidden layers all have 256 units per layer. (256,256,256) The learning rate is 0.001. The batch size is 100.

4. Plot loss and accuracy as a function of the number of training epochs. You do not need to do a train-validation split on the data, but in practice this would be helpful.



The plots are shown above. The agent achieves an acceptable accuracy after 20 training epochs. (85%)

5. Report the number of bounces your agent gets. It should be around 8 bounces.

We simulated 5000 games, and the agent achieves an average of 8.151 bounces.

2.3 Extra Credit

1. Implement batch normalization for part 2.1. Including the forward and backwards operations. Report the number of bounces with and without batch normalization.

The number of bounces, surprisingly decreases to 7.9194 bounces. (5000 simulations) However, the classification accuracy gets higher, which is 95.34% on the original expert policy dataset.

3. Part 1 agent versus part 2. Let the agents from part 1 and part2 (2.1 or 2.2) compete. Report the results over 1000 games. It does not matter which agent wins.

Part 1 agent (trained for 60k games) won 605 out of the 1000 games versus part 2 agent.