Alex Nguyen

Program 3: Learning to Play Pong
ECS 170 Artificial Intelligence

Problem Representation:

1. In the hardware ram (array) method, we create multiple convolutional layers that are used to train the agent using pixels to represent the entire board including the paddle and the ball. It then stores a feature map as the state, which already is a smaller input than the game display. The game display (image) way uses the whole board image and sets that as the state. The problem is when the image has RGB colors it will create weights for the three RGB colors and potentially cause overfitting while using the game display method. This makes it longer than the convolutional method. Therefore, the agent will learn easier from the ram method.

2. Neural networks in Q-Learning are used to find the relationship between the input and output layers. The inputs will be the variables: batch_size, gamma, num_frames, replay_buffer, env. This will input the image of the game as a state. The output will be the prediction of the deep learner, which are the neurons representing actions we can take. The variable env is just the environment we are working in. The batch_size is the size of the batch of games that is played to train. The variable gamma is the discount rate. The variable num_frames is the number of trials we play the game for. The variable relay_buffer is used to store the reward, state, action, next_state, and done. In this neural network it includes three convolutional layers and two linear layers.

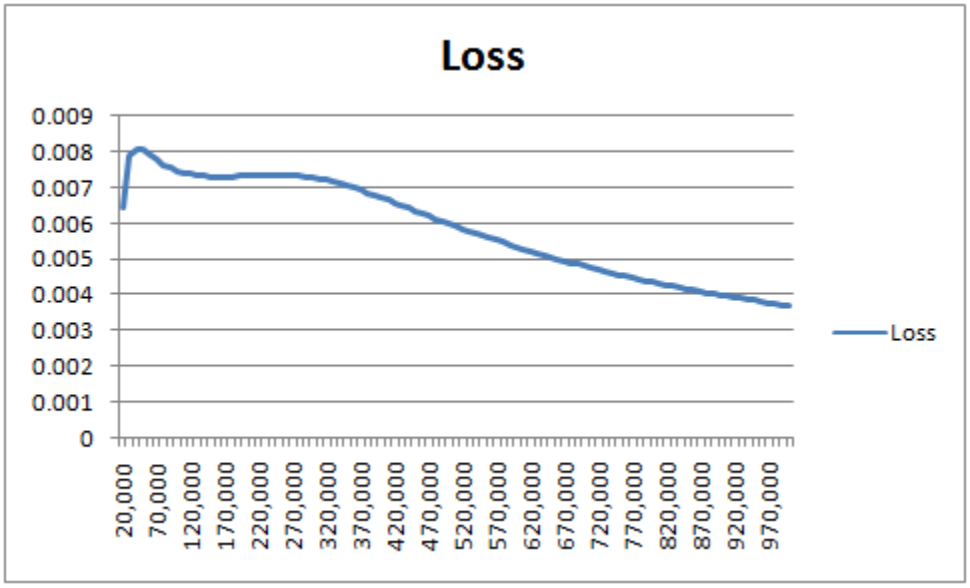
3. The purpose of the act function is to calculate whether we want to explore or exploit. Epsilon is initially set to a value close to 1 and decays over time, so the more we play the more epsilon decreases. If we have a value between 0 to 1 and it's bigger than epsilon that means we are exploiting the game. This means we are finding the max Q value and updating it. If the value is not greater than epsilon that means we are exploring, which is why the action is set to random.

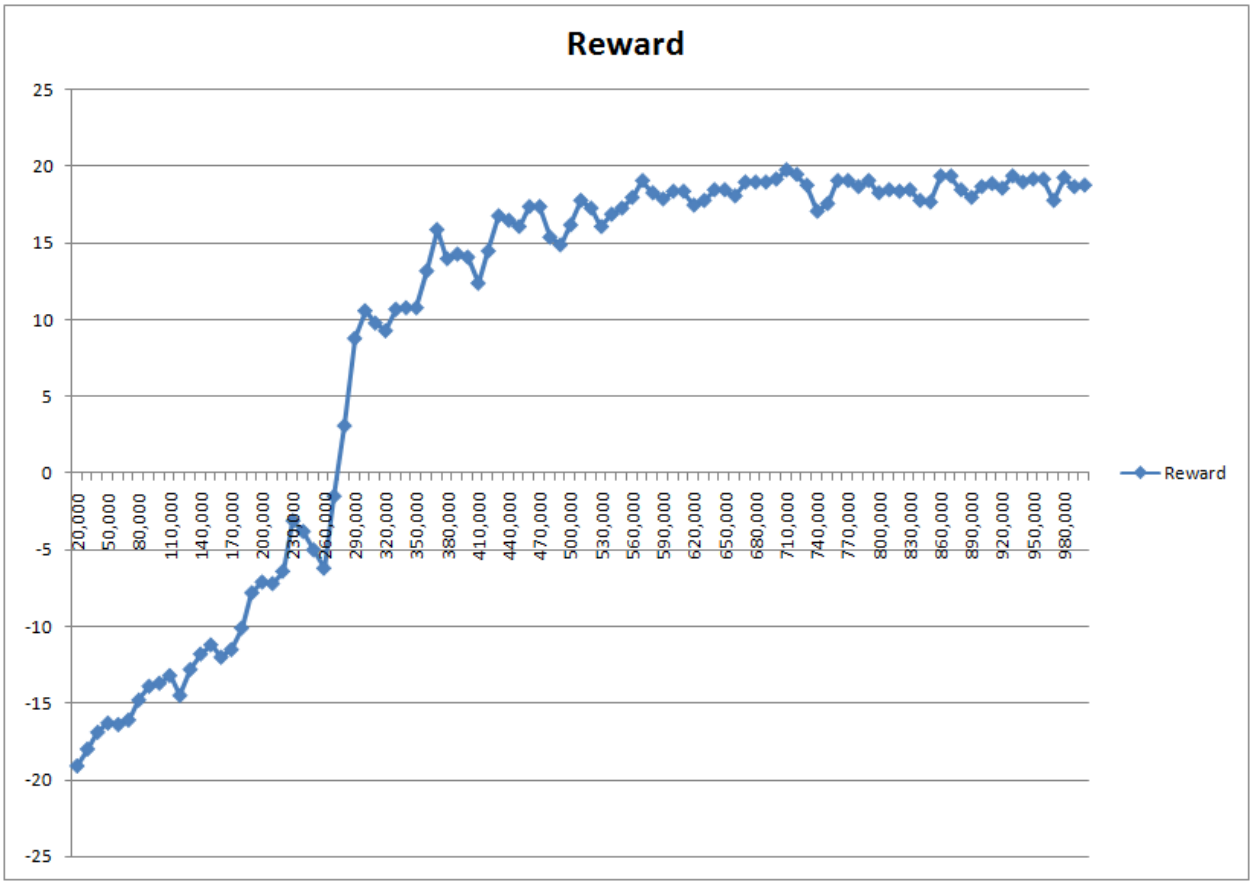
Making the Q-Learner Learn:

1. The loss model helps us compare the Q value at a certain state and action to the reward of executing the action. This ensures that the Q value is compared to the general observation space making it more meaningful data. This is why when the parameters have convergence, the model will create a value to be used as the input to the hidden layer.

Alex Nguyen

Learning to Play Pong





Alex Nguyen

My code did not create a model.pth file, but I have the correct output for the Frame and the Loss as seen below:

```
use_gui = "-g" in sys.argv

from Wrapper.layers import *
from Wrapper.wrappers import make_atari, wrap_deepmind, wrap_pytorch
import math, random
import gym
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Last-10 average reward: 18.500000
#Frame: 840000, Loss: 0.004202
Last-10 average reward: 17.800000
#Frame: 850000, Loss: 0.004167
Last-10 average reward: 17.700000
#Frame: 860000, Loss: 0.004132
Last-10 average reward: 19.400000
#Frame: 870000, Loss: 0.004098
Last-10 average reward: 19.400000
#Frame: 880000, Loss: 0.004066
Last-10 average reward: 18.500000
#Frame: 890000, Loss: 0.004032
Last-10 average reward: 18.000000
#Frame: 900000, Loss: 0.004001
Last-10 average reward: 18.700000
#Frame: 910000, Loss: 0.003969
Last-10 average reward: 18.900000
#Frame: 920000, Loss: 0.003938
Last-10 average reward: 18.600000
#Frame: 930000, Loss: 0.003907
Last-10 average reward: 19.400000
#Frame: 940000, Loss: 0.003876
Last-10 average reward: 19.000000
#Frame: 950000, Loss: 0.003846
Last-10 average reward: 19.200000
#Frame: 960000, Loss: 0.003817
Last-10 average reward: 19.200000
#Frame: 970000, Loss: 0.003788
Last-10 average reward: 17.800000
#Frame: 980000, Loss: 0.003760
Last-10 average reward: 19.300000
#Frame: 990000, Loss: 0.003732
Last-10 average reward: 18.700000
#Frame: 1000000, Loss: 0.003705
Last-10 average reward: 18.800000
PS C:\Users\Alex Nguyen\Desktop\ecs170\Project3\Starter Code>
```