ECS170 Programming Assignment 3 Written

Russell Chien

March 2024

Link to Notebook

`https://drive.google.com/file/d/1d91oarQ2mjlRSXJ0-hsJUUTdCg6Hez0i/view?usp=sharing`

Level 1: Presentation

1. For the Q-Learner, we must represent the game as a set of states, actions, and rewards. OpenAI offers two versions of game environments: the game display (as 84x84 pixel image) or the console's RAM (a 128-byte array). Which do you think would be easier for the agent to learn and why?
The console's RAM, a 128 byte array, would be easier for the agent to learn because it has a much smaller state space. Processing the game display, a 84x84 pixel image, is computationally expensive, leading to very long training times. Additionally, the array more accurately represents the game state and the agent can learn the relationships between actions and outcomes, while the image would require the model to extract the features that represent the game state using a CNN.
2. Using the starter code as reference (the cell labeled dqn.py) describe the network's input and output layers. What do these layers represent?
The initial input layer takes the number of input channels from self.input_shape[0] which would be the problem's state space, i.e. 128-byte array of the console's RAM. The output layer (self.fc) outputs a vector with a size of self.num_actions which represents the Q-values for each possible action.
3. What is the purpose of the following lines:

```
if random.random() > epsilon:
    ...
else:
    action = random.randrange(self.env.action_space.n)
```

These lines are used for the epsilon greedy strategy. Epsilon is first initialized to a large value and decreases over time. When the condition is true, the agent exploits its current knowledge to find the next best possible move. When the condition is false, the agent explores and chooses a random action from its possible action space.

4. Given a state, write code to compute the q value and choose an action to perform. (see lines 50-53 in the function act of the section labeled dqn.py).

```
def act(self, state, epsilon):
        if random.random() > epsilon:
            state = Variable(torch.FloatTensor(np.float32(state)).unsqueeze(0), requires_gra

            with torch.no_grad():
                # pass state through the network to get Q-values for all actions
                q_values = self.forward(state)
            # select action with the highest Q-value
            action = q_values.max(1)[1].item()
        else:
            action = random.randrange(self.env.action_space.n)
        return action
```

Level 2: Learning

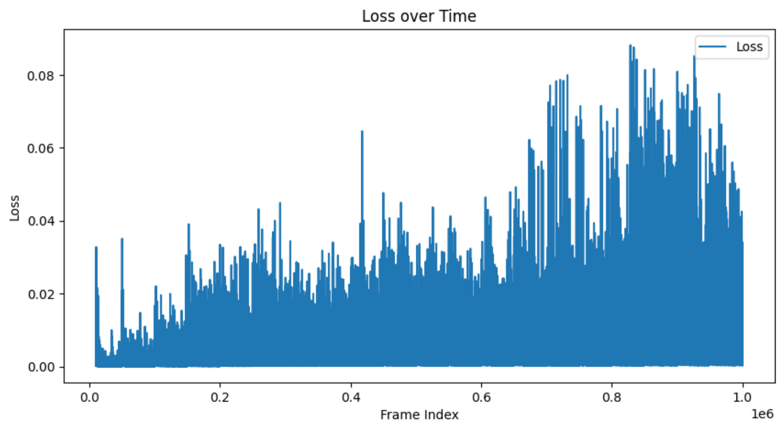
1. Explain the objective function of Deep Q Learning Network below. What does each variable mean? Why does this loss function help our model learn? This is described in more detail in the Mitchell reinforcement learning text starting at page 383. We've provided the loss function below. This should be summed over the batch to get one value per batch.

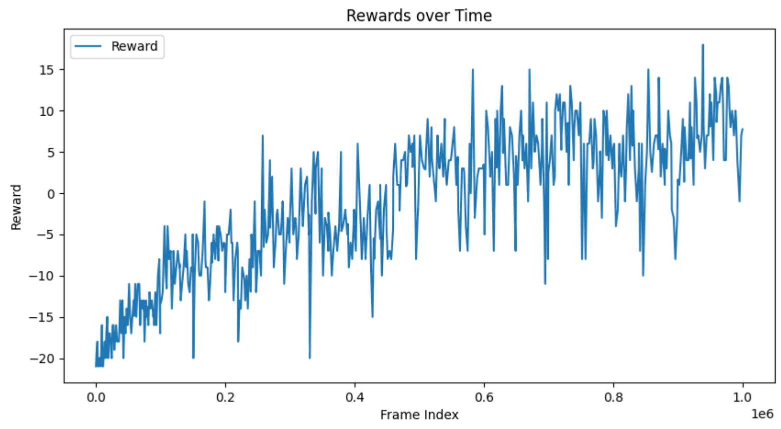
$$Loss_i(\theta_i) = (y_i - Q(s, a; \theta_i))^2 \tag{1}$$

θ_i represents the parameters of the neural network, y_i is the reward received when taking action a in state s, and $Q(s, a; \theta_i)$ is the predicted Q-value. The loss function helps the model learn by reducing the error and improving its parameters to converge towards the optimal policy.

Level 4: Training

2. The training cell currently records the loss and rewards in losses and all rewards respectively. Plot these rewards and losses over time and include the figures in your report. You may either save their output and using a plotting software of your choice, or use a python module (such as matplotlib) to assist in the creation of such plots. You do not need to provide the raw data in your submission.





Bonus Level: Explanation

1. We discussed the idea of overfitting in class. Looking at how your model behaves after it is trained, do you think that your model is overfit? How do you know? What does/would it mean for a model to overfit in the context of this project?
A model is overfit when it performs better on the training data versus the testing data. In the context of reinforcement learning, a model that is overfit would likely be unable to generalize - it would perform exceptionally well in certain scenarios that were seen in the training data, but would fail in unseen or underrepresented scenarios.
In the context of this project, the model may be somewhat overfit since it does not win every single point, which may suggest that it is overfit to more common game states and fails to handle less common game states, causing it to lose a point occassionally.