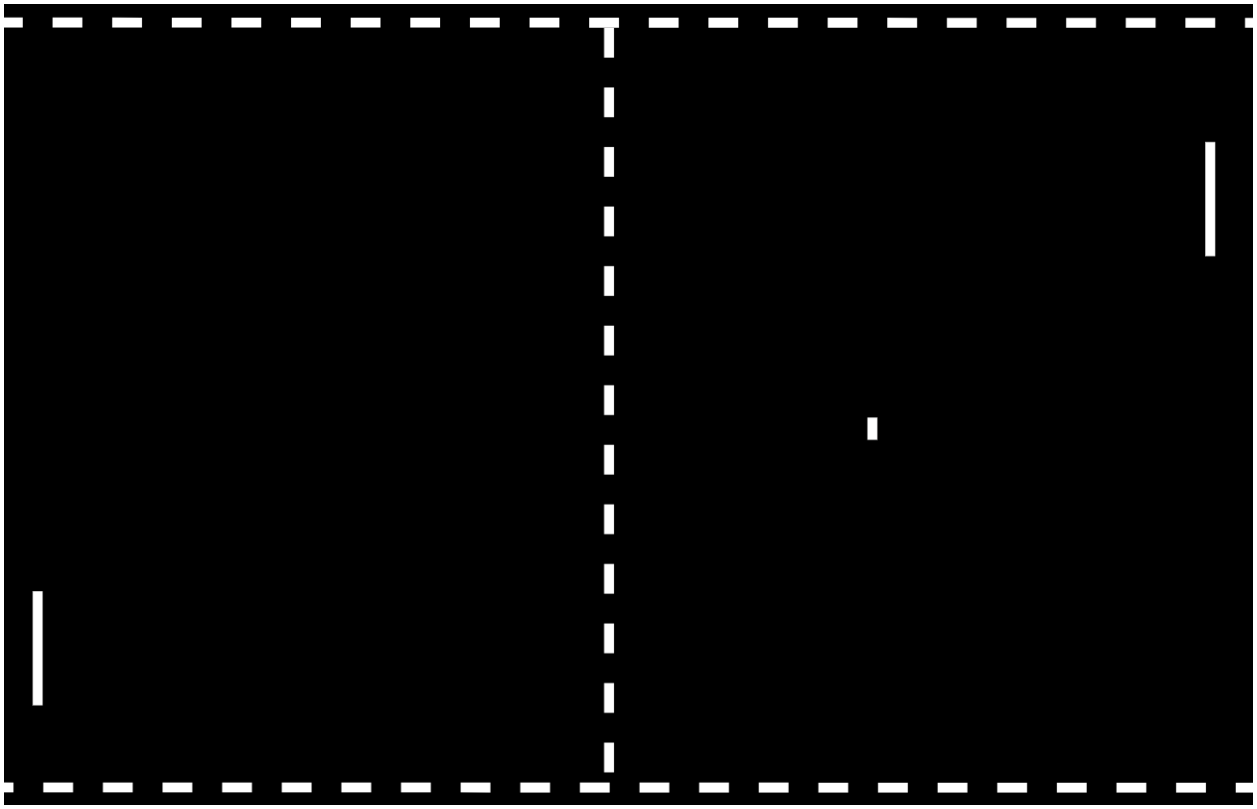
Programming Assignment 3:

Deep Q-Learning For Atari Pong



-Press Start-

0. Setup

1. If you are using Google Colab (link), everything should be already set up and ready to go. If you are running this on a different machine (CSIF or your own machine with a CUDA compatible GPU), then you may need to pip install the imported modules.

Level 1: Representation (+8 pts)

Before jumping into the code, let's think about the problem first. In your writeup, please answer the following:

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?
2. Using the starter code as refernce (the cell labeled dqn.py) describe the network's input and output layers. What do these layers represent?
3. What is the purpose of the following lines:

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

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).

Level 2: Learning (+8 pts)

1. (Written) 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.

$$\text{Loss}_i(\theta_i) = (y_i - Q(s,a;\theta_i))^2$$

2. (Programming) Implement the aforementioned loss function(see line 73 of DQN.py)

Level 3:

Buffer
(+8 pts)

The replay memory/buffer in Deep Q Networks is used to store many (state, action, reward, next state) entries. In typical Q-learning, these entries are used sequentially to update the Q-table. With experience replay, we instead store these entries and later use them to update our policy by randomly sampling from the buffer to get an entry and using that entry to update our policy. This is necessary as optimization is assuming independent and identically distributed samples. If we do not use experience replay then the agent will see many similar samples as sequential samples in the game are very similar. This will encourage the model to converge to a local minima.

1. (programming) Implement the “randomly sampling” function of replay memory/buffer. This should sample a batch from the replay buffer. (see line 90, function sample, of dqn.py)

Level 4:
Training
(+10 pts)

1. (programming) To help combat the difficulty in training time, we have provided a network that is partially trained. You will need to load this model to be able to train it further. It is good convention when training neural networks to save your model occasionally in case your code crashes or your server gets shut off for whatever reason. Adjust run dqn pong.py to be able to load in a model and occasionally save a model to disk. There are built-in functions to do so for Pytorch - please use torch.save(model.state dict(), filename). Do not manually extract or load in the model weights. Use .pth files.
2. (written) 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.
3. (programming) Train the model by running run dqn pong.py. To achieve good performance, you will need to train the model for approximately 750,000-1,000,000 more frames which should take between 2-4 hours on the Google servers. You may optimize different values for hyper-parameters such as γ and the size of the replay buffer, but you do not have to and this may take a long time.

Bonus Level:
Explanation
(4.5 pts)

There are two parts to the bonus level to get extra credit. One is a more programming intensive portion about explanation, and the other is a conceptual question about your model's behavior.

(1.5pts)

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?
2. (written) In about two paragraphs to one page (not including any pictures), discuss the above questions. I recommend you to write one paragraph about what overfitting is and what overfitting might look like in the context of RL, and another paragraph about how it looks like your model either is or is not overfit.

(3pts) A core challenge with DL is understanding or explaining how they make decisions. This generally area known as XAI is an active research area but here your aim is to gain some level of insight into how your DL is making decisions

1. (programming) For 1000 randomly picked frames, collect the 512 features (1000x512 features) as well as side information you think may be relevant. For example, the position of the agent paddle may be relevant.
2. (programming) Perform dimensionality reduction on the features (e.g. PCA, CCA, MDS, ISOMAP, LLE) to project the features to a 2D or 3D space and plot this embedding. Color code the embedding according to your collected side information.
3. (written) Analyze the embedding and conclude something about what your model has learned. These do not need to be positive results, you may conclude that your model does not take into account your side information. Include both your plot and your conclusions in your report to receive bonus points.