# Assignment 3: Playing Pong with Deep Q-Network

ECS 271 Machine Learning

May 25, 2020

## 1 Environment

In this assignment, you will utilize deep Q-Learning to play the Atari game: `PongNoFrameskip-v4` from OpenAI gym. You should have read the "A Brief Survey of Deep Reinforcement Learning" article on Canvas as well as the Mitchell chapter on reinforcement learning.
The typical RL and DRL covered in class and given in the starter code is insufficient to learn Pong so you will explore variations. In Part 1, you need to answer some questions and implement some key additions for DQN. In Part 2, you will visualize and analyze the trained DQN in Part 1. The starter code has been provided (based on `PyTorch` and `gym` environments).

If you have available GPU resources, you may work locally. Alternatively, you can use the Google Cloud platform (refer to the distributed Google Cloud Tutorial). If you are using Google Cloud, here are some suggestions to avoid some pitfalls:

a. When launching Deep Learning VM, I suggest you to choose the `TensorFlow 1.13` framework instead of `PyTorch 1.1` because `gym` is not installed by default in the latter one.

b. After launching the `TensorFlow 1.13` virtual machine, open its command line and do the following installation:

```
sudo apt install cmake libz-dev
pip install torch torchvision
pip isntall gym[atari]
```

c. I suggest you to run the codes in `screen` so that you don't have to keep the command line window open all the same. If you are not familiar with this, you might refer to this site.

## 2 Part 1: Extend the Deep Q-learner (60 points)

1. (**written**) Explain what is a replay memory/buffer in DQN. Why it is necessary?

2. (**coding**) Implement the "randomly sampling" function of replay memory/buffer (see line 89 of `dqn.py` for TODO and hints). Submit necessary code as sampling.zip.

3. (**coding**) Given a state, write code to calculate the Q value and the corresponding chosen action based on neural network (see lines 50 $\sim$ 55 in `dqn.py`). Submit necessary code as action.zip.

4. (**written**) Given a state, what is the goal of line 48 and line 57 of `dqn.py`? Aren't we calculating the Q value and the corresponding chosen state from the neural network?

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

5. (**coding**) Implement the "Temporal Difference Loss": the objective function of DQN (see lines $69 \sim 73$ of `dqn.py`), which is described in the Mitchell Q-learning text on the website:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a; \theta_{i-1}) \middle| s, a\right] \tag{1}$$

$$\mathcal{L}_i(\theta_i) = \mathbb{E}\Big[(y_i - Q(s, a; \theta_i))^2\Big] \tag{2}$$

   Submit necessary code as TD.zip.

6. (**written**) After you make these changes, train DQN by running `run_dqn_pong.py`. To achieve relatively good performance, you need to train more than 1000000 frames. It takes $\sim 10$ hours on a Google Cloud Virtual Machine with 2 vCPUs and 1 NVIDIA Tesla K80 GPU.

   Explain what parameter tuning you performed such as:

   a. You might tune the hyper-parameters like $\gamma$ and initial size and the size of replay buffer to gain a better performance.

   b. Modify `run_dqn_pong.py` to track and plot how loss and reward changes during the training process. Attach these figures in your report. Your grade will be based on the ranking of final reward.

# 3 Part 2: Visualize and Analyze Trained DQN (40 points)

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.
Given a trained DQN, we can evaluate the DQN on the input frames and activations of the last layer of the neural network. For example, if we evaluate the trained DQN on $X$ frames and dimension of the last hidden layer has $Y$ nodes, we are able to collect data of form $X \times Y$. Here you will use one or more of the dimension reduction techniques covered in class (PCA, CCA, MDS, ISOMAP or LLE) To visualize the (manifold or otherwise) structure of those frames/activations by mapping them to a 2-D space.

This embedding could be useful to analyze DQN and somewhat explain why DQN can achieve good performance. Here is an example: If we record the chosen actions of all "exploitation" frames, we will be able to see whether similar outputs of neural network lead to the same chosen action. The side information (*e.g.* screenshot of game board, reward, chosen action, state of the board, order of the frames, etc) that could be recorded when evaluating trained DQN on frames are very fruitful and could explain different aspects of the trained Q-learner.

a. Evaluate your trained DQN in Part 1 on (*e.g* randomly picked 1000) frames and collect the features as well as various side information.

b. Perform reduction on the features.

c. Analyze the embedding based on two kinds of your collected side information.