Program 3: Learning to Play Pong

ECS 170 Artificial Intelligence

Overview

In the previous programming assignments you have been manually creating intelligent agents by codifying human intelligence regarding the environments your agent will be exploring. Modern approaches to AI step away from methods requiring domain experts. This allows models to not only learn better policies, such as beating humans at go, but also learn policies for domains in which encoding human knowledge is near impossible, such as the pancake flipping example from lecture. For this assignment, you will be training a model to learn to play Pong without human knowledge of the game. You will be using the OpenAI gym to run and interact with the Pong Atari game and the deep Q-learning algorithm as covered in class. The provided starter code is insufficient to learn to play Pong, so for Part 1, you will make modifications to improve its ability to learn and train the provided pre-trained model to acceptable performance..

The model will use its Q network to decide what moves to take when playing the game. Unlike what has been covered in class, this network will instead push each decision it made to an experience buffer, detailed in section 3. This replay buffer will be polled from to make updates to the Q network using the loss function described in section 2. The network is contained within `dqn.py`. You can train your network using `run_dqn_pong.py`. You can play a single game with your network using `test_dqn_pong.py`. The environment will keep playing games of pong until either player gets 21 points.

Be aware that training this model will take several hours on high-end GPU servers. This is unavoidable, so start early! You will not be provided additional time if you are not able to train your model and will not receive credit if your trained model does not perform sufficiently.

Environment and Setup

If you have high-end available GPU resources, you may work locally. Otherwise, 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:

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

```
sudo apt install cmake libz-dev
sudo pip install torch torchvision
sudo pip install gym[atari]
sudo apt-get install python-opengl
```

- I suggest you run your commands utilizing either `nohup` or `screen`. These will enable you to exit the terminal with your script still running.

1 Problem Representation

1. (written) For the Q learner we must represent the game as a set of **states**, **actions**, and **rewards**. OpenAI offers two versions of game environments: one which offers the state as the game display (image) and one that offers the state as the hardware ram (array). Which do you think would be easier for the agent to learn from and why?
2. (written) **Use the starter code to answer this question.** Describe the purpose of the neural network in Q-Learning. Neural networks learn a complicated function mapping their inputs to their outputs. What will the inputs and outputs for this neural network be? What do these variables represent? The neural network in the starter code is `class QLearner(nn.Module)`.

3. (written) What is the purpose of lines 48 and 57 of dqn.py (listed below)? Doesn't the q learner tell us what action to perform?

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

4. (programming) Given a state, write code to compute the q value and choose an action to perform. (see lines 50-55 in function `act` of dqn.py).

2 Making the Q-Learner Learn

1. (written) Explain the objective function of Deep Q Learning Network for one-state lookahead 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.

$$Loss_i(\Theta_i) = (y_i - Q(s, a; \Theta_i))^2$$

Hint: Θ is convention for "model parameters" and y is convention for "model output". The subscript $_i$ refers to "the i-th iteration". If you are stuck, research "mean squared error".

2. (programming) Implement the aforementioned loss function (see lines 73-76 of dqn.py).

3 Extend the Deep Q-Learner

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)

4 Learning to Play Pong

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. (programming) `run_dqn_pong.py` currently records the loss and rewards in `losses` and `all_rewards` respectively. Modify `run_dqn_pong.py` to save these to memory for use in question 4. You do not need to attach these to your submission (they will be very large).

3. (programming) Train the model by running `run_dqn_pong.py`. To achieve good performance, you will need to train the model for approximately 500,000 more frames which should take between 3-6 hours on the Google servers. You may want to optimize different values for hyper-parameters such as γ and the size of the replay buffer.

4. (written) Plot how the loss and reward change during the training process. Include these figures in your report.

5 Bonus Points - Explanation

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.

6 Requirements

- Report.pdf - pdf report for the above questions marked 'written'; no handwritten responses
- model.pth - your best performing model saved to a pth file. This does not have to be your *most trained* model, but is instead your best performing model. **Make sure test_dqn_pong.py can load your saved model.**
- run_dqn_pong.py - with your relevant changes
- dqn.py - with your relevant changes
- any additional programming files you used for bonus points questions