Pong AI User Manual

Yash Bhutwala, Matt McNally, Kenny Rader, John Simmons

Games and toy problems often provide interesting environments to explore different aspects of artificial intelligence and machine learning. Through the years as this field developed, more and more approaches to learning have been created. Naturally, some work better than others. We can use video games as a great testing ground for these different approaches in order to determine definitively which is the best and most efficient way for a machine to learn.

Our experiment is to compare the two reinforcement learning methods of Deep Q-Learning Networks (DQN) and Policy Gradient (PG) learning. We can test both of these approaches by creating an agent that utilizes each algorithm, have them each train on a set number of trials, and then compare the results of a testing period in which they play another set number of trials. We will use the Atari Pong environment provided with OpenAI Gym to facilitate our experiment, where each trial is one full game. Following this experiment, our results should indicate clearly which method of learning is superior.

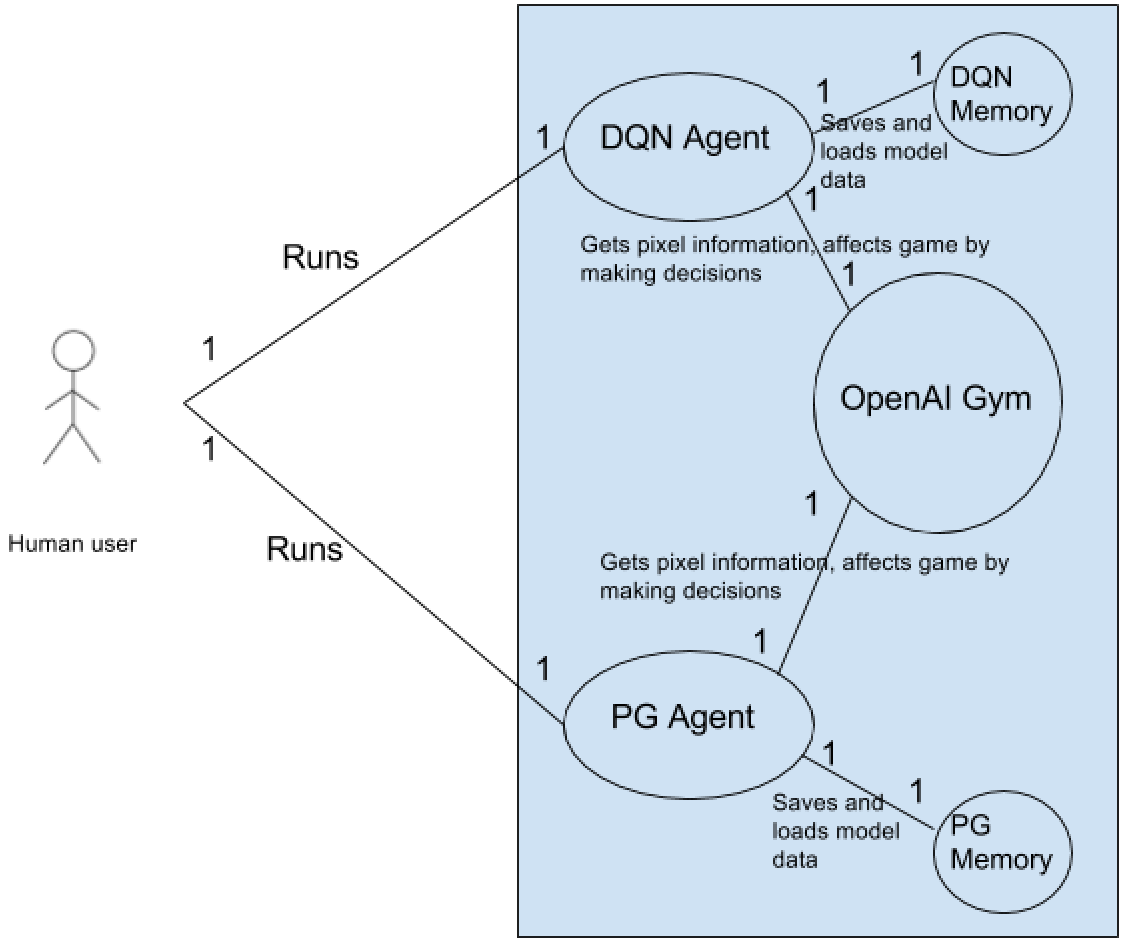
Introduction / Background / Motivation

Given a set a pixels from a game of Pong, a mechanism for measuring wins and losses, and a hard-coded opponent agent, we should have all that we need to test which of our two approaches are the best for having a machine learn to play the game. OpenAI Gym provides an easy to use framework for setting up our environment. It contains functions that return all of the necessary information we need: the pixels from the screen, a reward mechanism for telling our agent whether or not their move was fruitful, and an opponent agent to play against. All we need to do is supply our testing agents.

The first approach we plan to test is the Deep Q-Learning Network (DQN) agent. DQN works similarly to vanilla Q-learning: choose the best action based on the highest Q-value available and then subsequently back-propagate the errors to adjust the Q-values of the moves chosen for the next iteration. There is a key difference, however. DQN runs on a convolutional neural network, which is a special kind of network that has overlapping nodes in convolutional layers to read in visual input (like our pixels from the game). These pixels are transformed into meaningful data that is processed through the fully-connected layers of the network, leading to output that helps the agent choose the action with the best Q-value. This more complicated network architecture is essential for the agent to get meaningful information about its environment and use that information to inform its decisions.

The second approach we will look at is Policy Gradient (PG) learning. PG gets its information similarly, taking in images from the game and preprocessing them through an algorithm that strips the images of everything except the vital information needed. It then uses a standard neural network to compute a probability of moving in one direction. The agent samples from a probability distribution to determine its next move, and makes a decision based on the result. After a set amount of time, say the end of a trial, the results of the agent's movements are passed through a back-propagation algorithm to compute a gradient for the weights of its network connections. After a set number of trials, these gradients are summed and the weights are shifted in the direction of the resulting gradient. This process is repeated indefinitely until the agent settles on an equilibrium weight for each of its connections. This algorithm was developed as an improvement upon existing, similar algorithms, such as its testing opponent DQN. Our test will determine if it truly is an improvement.

Below is a UML Use Case diagram to show how our system is to be used.


Human user
Runs
Runs
1
1
1
1
DQN Agent
1
1
DQN Memory
Saves and loads model data
1
1
Gets pixel information, affects game by making decisions
OpenAI Gym
1
1
Gets pixel information, affects game by making decisions
PG Agent
1
1
Saves and loads model data
PG Memory

Instructions for running the programs:

The system can be run by a human user through the terminal. To run the DQN agent, the user must navigate to the ./dqn directory and use the command:

```
python main.py --env_name=Pong-v0 --is_train=True --display=True
```

This will start the agent, using the Gym environment Pong-v0, with training mode turned on, and the game rendering. To increase the speed of training, the user can set `--display=False`. This will prevent the game from rendering. If the user does not wish to train the agent, they can similarly set `--is_train=False`. If a different environment is desired, the user can change `--env_name` to something else, but all of our tests were done in the Pong environment, so this is not recommended.

To run the PG agent, the user must navigate to the ./pg directory and use the command:

```
python3 yashPong.py
```

This will start the agent, using the Gym environment Pong-v0, with training mode turned on, and the game rendering by default. To change any of these parameters, the code within the file yashPong.py can be changed to fit the user's needs. The values that determine rendering and training are simply Boolean statements that can be set to True or False.

The terminal output for each of these agents is slightly different, but in general, information about the number of games played and the associated rewards acquired for each iteration will be displayed to the user. This is critical in determining which of these approaches is the best to implement in a Atari-playing agent.