Machine Learning CSE574

David Jegan Abishek Dominique
Person no 50290785
davidjeg@buffalo.edu

Abstract

In this report we addresses how we could teach an model to play a game using reinforcement learning DQN (Deep Q-Network) where the agent learns continuously from its history of interaction with the environment.

1 Introduction

Gym, is a library that can simulate large numbers of reinforcement learning environments, including Atari games and **Stable Baselines** helps to improve the RL environments provided by OpenAI.

Observation learns from the environment objects, which is represented as raw pixel data on the screen. At each step, the state, action, the next state and the corresponding rewards are given to the neural network for training.

State : The agent takes action based on the state it is currently in. The agent moves across the grid.

Action : For a given state, an agent can perform all possible actions . For instance, in the edges the agent cannot move to certain states.

Environment : The world in which the game runs is called the environment, and the model continuously learns from the environment.

Agent : The agent finds the optimal shortest path from the goal to initial position using its history of interaction with the environment.

2 Atari Space Invaders

Environment : The observable RGB image screen as the environment with shape (210,160,3)

Agent: The linearily moving space craft must hit the aliens

State: Agent takes an action in the environment in a state

Action: Agent can do an action which consists of three actions - Up,Down,Shoot,Nothing

Reward: The craft's actions are reward motivated gets a reward if it hits the alien, no reward at rest, and gets negative reward if it is hit by the alien.

2.1 Step

Environment

* Gym package is used to render a Space invaders atari environment. We use the gym.make('SpaceInvaders-v0') command
* Vectors of observation space is retrieved using the DummyVecEnv package
* Logs are gathered from the environment for plotting

DQN

* The DQN gets the environment initialized, and the policy.
* The images are analyzed as inputs by our DQN model, so the corresponding policy is Cnnpolicy

Evaluation

* The mean reward is calculated for the model using the predefined number of steps.
* The model is then learns from the previous step, and now mean reward is calculated and is increased.

2.2 Tuning Hyperparameters

Default: The mean reward before training has increased after training the model. The increase would be drastic if the number of steps is in order of millions, but we have run on scale of thousands for this experiment.

Default	Value
Reward mean before training	203.1
Reward mean after training	281.5

Gamma: The mean reward has increased by increasing the gamma value. Therefore the gamma value is left to the default value of 0.99 since it has the highest mean reward

Gamma	Reward mean before training	Reward mean after training
0.99	203.1	281.5
0.75	189.4	271.9
0.50	162.3	266.2

Number of steps: The mean reward has increased by increasing the number of steps. Therefore the number of step value is left to the 10000 since it has a good computational feasibility in the local system and returns highest mean reward .

No of steps	Reward mean before training	Reward mean after training
10000	203.1	281.5
1000	167.9	259.2
100	79.2	102.8

2.3 Results

The result plotted for episodes and reward mean shows that our model has increased performance after training.

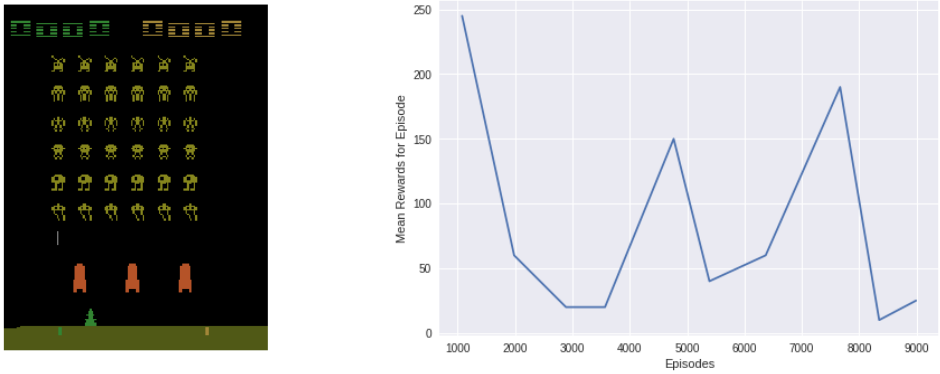

Figure 1: Ping pong game(left) and results(right)

3 Cartpole

Game : A pole attached by an un-actuated joint to a cart, moves in track. We apply force of +1 or -1 to the cart and pendulum starts upright. The goal is to prevent it from falling over and The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

Environment : The observable RGB image screen as the environment

Agent: The linearily moving pendulum which should be upright

State: Agent takes an action in the environment in a state

Action: Agent can do an action which consists of three actions - Left, Right, Nothing

Reward: The Pendulum's actions are reward motivated gets a reward if it is upright for every timestep ,and gets negative reward if it falls down.

3.1 Step

Environment

* Gym package is used to render a Cartpole environment. We use the gym.make('CartPole-v1') command
* Vectors of observation space is retrieved using the DummyVecEnv package
* Logs are gathered from the environment for plotting

DQN

* The DQN gets the environment initialized, and the policy.
* The feature vectors are analyzed as inputs by our DQN model, so the corresponding policy is Mlppolicy

Evaluation

* The mean reward is calculated for the model using the predefined number of steps.
* The model is then learns from the previous step, and now mean reward is calculated and is increased.

3.2 Tuning Hyperparameters

Default: The mean reward before training has increased after training the model. The increase would be drastic if the number of steps is in order of millions, but we have run on scale of thousand for this experiment.

Default	Value
Reward mean before training	12.1
Reward mean after training	144.9

Gamma: The mean reward has increased by increasing the gamma value. Therefore the gamma value is left to the default value of 0.99 since it has the highest mean reward

Gamma	Reward mean before training	Reward mean after training
0.99	12.1	144.9
0.75	11.8	129.9
0.50	10.7	101.8

Number of steps: The mean reward has increased by increasing the number of steps. Therefore the number of step value is left to the 10000 since it has a good computational feasibility in the local system and returns highest mean reward .

No of steps	Reward mean before training	Reward mean after training
100000	12.1	144.9
10000	8.1	97.1
1000	5.8	49.2

3.3 Results

The result plotted for episodes and reward mean shows that our model has increased performance after training.

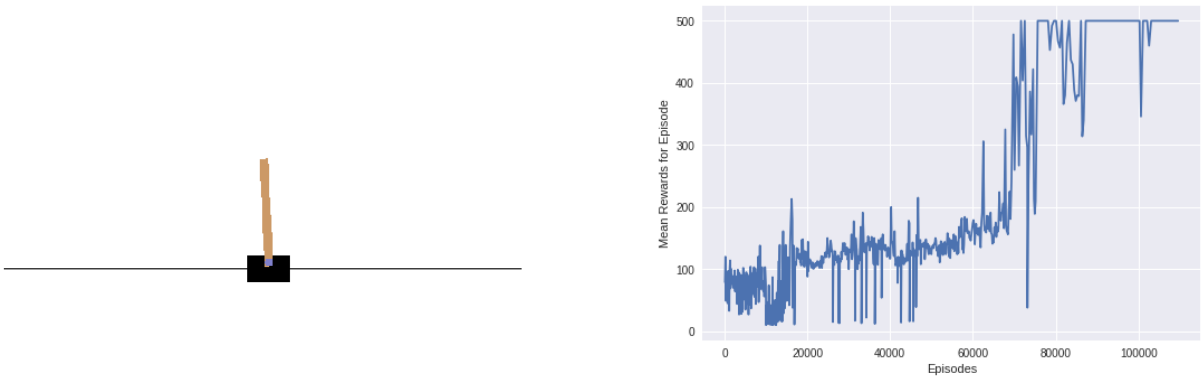

Figure 2: Cartpole game(left) and results(right)

References

1. https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df

2. https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26
3. https://becominghuman.ai/lets-build-an-atari-ai-part-0-intro-to-rl-9b2c5336e0ec
4. https://medium.freecodecamp.org/how-to-build-an-ai-game-bot-using-openai-gym-and-universe-f2eb9bfbb40a
5. https://hub.packtpub.com/create-your-first-openai-gym-environment-tutorial/
6. https://medium.com/@dhruvp/how-to-write-a-neural-network-to-play-pong-from-scratch-956b57d4f6e0
7. https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2
8. https://stable-baselines.readthedocs.io/en/master/guide/examples.html?highlight=monitor
9. https://stable-baselines.readthedocs.io/en/master/guide/examples.html