Training a MSPACMAN AI with different techniques

Li-Yuan, Chiang 108062131
Chia-En, Lu 108062135
Zih-Jyun, LIN 108062172
Hao-Chen, Jiang 108062203

Abstract—This report is a summary on our attempt to train a MSpacman AI. The goal is to practice DRL and learn how to design a game AI that has a decent performance. We define new rewards to pursue a higher score for the model.

I. Introduction

Ever since artificial intelligence was introduced to the world, people have always been trying to test the limits of it, a good way to evaluate a model is to compare results of an action performed by the AI. Playing games is a good way to evaluate a model since a game defines a strict policy on how rewards are defined. OpenAI gym provides us with a easy to use environment for us to evaluate our model. In this project, we try to achieve the highest score possible by designing an AI model for the Atari2600 game Ms.Pacman.

II. Debrief on our environment

A. Stella

Stella is a multi-platform Atari 2600 VCS emulator which emulates the Atari 2600 Video Computer System introduced in 1977.

B. The Arcade Learning Environment

The Arcade Learning Environment (ALE) is a simple framework that is designed to allow other program designers to develop AI agents for Atari 2600 games. It is built upon the Atari emulator Stella which provides a wide range of games for programmers to utilize.

C. GYM

Gym is a toolkit for developing and comparing reinforcement learning algorithms, it supports a wide range of games ranging from cartpool to Ms.Pacman. It has an open source interface to reinforcement learning tasks.

III. Game Choice

MS.Pacman is a 1982 maze arcade game. It was chosen since an AI can have a wide range of scores, indicating the effectiveness of the agent. The player gain points by eating balls and try to kill ghost by eating boosters. An average human player is able to score from 5,000 to 15,000 points. A good AI shown on the rating on OpenAI Gym has an average score around 2000 3000 points.

IV. Game Version

Gym provides us with two versions of the game for training our agent, MsSPacman-v0 and MsPacman-ram-v0.

A. MsPacman-v0

In this environment, the observation is an RGM image of the screen, the observation is presented in the form of an array of shape (210, 160, 3). We use this array and perform some optimizations for training, and then feed it into our model in order to train our agent.

B. MsPacman-ram-v0

Different from the MsPacman-v0 environment, the observation is the RAM of the Atari machine, which consists of 128 bytes. Each byte is a number ranging from 0 to 255, after doing some digging in the source code, it lead us to a paper [1] which identified some digits of the RAM, we identified some bits as some useful information, such as the position of pacman and different ghosts.

```
[  0 112 114 115  48   3  88  88  88  88  88   0  80  80  80  50  98   0
   0   3   0   0   1   0   0   1   6   6 198   4 227   0  45   1   0 198
 198   0   0  21   0   0  52   0   0 120   0 100 130   0   0 112   1 222
   0   0   3   0   6  80 255 255   0 255 255  80 255 255  80 255 255  80
 255 255  80 191 191  80 191 191  80 191 191  80 255 255  80 255 255  80
 255 255  80 255 255   0 255 255  80 255 255  20 223  43 217 123 217 123
 217 123 217 123 217 123 217 221   0  63   1   0   0   0   0   2  66 240
 146 215]

[  0 112 114 115  48   3  88  88  88  88  88   0  80  80  80  50  98   0
   0   3   0   0   1   0   0   1   6   6 198   4 229   0  45   1   0 198
 198   0   0  21   0  16  52   0   0 120   0 100 130   0   0 134   1 222
   0   1   3   0   6  80 255 255   0 255 255  80 255 255  80 255 255  80
 255 255  80 191 191  80 191 191  80 191 191  80 255 255  80 255 255  80
 255 255  80 255 255   0 255 255  80 255 255  20 223  43 217 123 217 123
 217 123 217 123 217 123 217 221   0  63   1   0   0   0   0   2  66 240
 146 215]

[  0 112 114 115  48   3  88  88  88  88  88   0  80  80  80  50  98   0
   0   3   0   0   1   0   0   1   6   6 198   4 233   0  45   1   0 198
 198   0   0  22   0   0  52   0   0 120   0 100 130   0   0 156   1 222
   0   2   3   0   6  80 255 255   0 255 255  80 255 255  80 255 255  80
 255 255  80 191 191  80 191 191  80 191 191  80 255 255  80 255 255  80
 255 255  80 255 255   0 255 255  80 255 255  20 223  43 217 123 217 123
 217 123 217 123 217 123 217 221   0  63   1   0   0   0   0   2  66 240
 146 215]
```

Fig. 1. Image of the RAM output

C. Our Final Choice

After various tests, we could not optimize the training result using MsPacman-ram-v0 since there are so many uncertain bytes that could effect the training result.

V. Training Method

VI. Q-learning

We tried to implement Q-learning into an AI agent, however, the resources that the agent has to utilize in order to save all the data in the Q-table would make it impossible to run.

VII. DQN

Deep Reinforcement Learning is a combination of Deep Neural Network and Q-learning, a kind of Reinforcement Learning technique. Before, we solve the RL problem by using Q-learning. Q-learning uses Q table to record information, in this case of the Atari Mspacman, the resource required to save all information in the Q table would be too large for any normal training environment, so what we need is DRL. In deep reinforcement learning, we represent the various components of agents, such as polices or values with deep neural network. And a deep version of Q-learning is called Deep Q-Learning(DQN). We applied DQN as the model to train an AI playing the Atari video game: Pacman. We define the agent as the pacman, observation as the map,reward as the score we get. In this project, our goal is to maximize the score we can get during the game.

DQN rather than using a lookup table to store the Q value, it train a Deep Neural Network from the state or action of this environment for AI to learn how valuable those are relative to our target. DQN is a combination of convolutional layers and full connection layers. The convolutional layers detect the features of each input pixels in this game. And the full connection layers maps the set of every features presenting in the observation to output layer with each node for each action combinations from the agent. And these output values would be the best prediction according to the reward got after agent take this action.

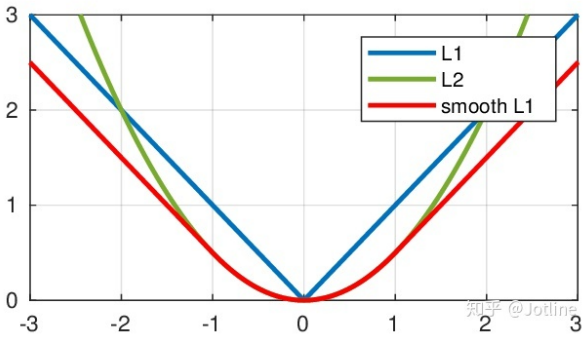
Following function Q(s,a) represent the total value of taking action "a" in state "s", and the result is the sum of the future rewards "r", which is adjusted by "gamma", used to let agent to determine how far the future should be taken in consideration.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\, a_t = a,\, \pi\right],$$

The parameters of the neural network are optimized by using Adam to minimize the smoothL1 loss function :

$$loss(x,y) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} .5 * (y_i - f(x_i))^2, & if\ |y_i - f(x_i)| < 1 \\ |y_i - f(x_i)| - 0.5, & otherwise \end{cases}$$

The advantage to use smothL1 loss function is that when the difference between the predicted value and the ground truth is small, the gradient will not be too large(The loss function is smoother than L1 Loss). And when the difference is large, the gradient value can be small enough (more stable and less prone to exploding gradient).



Since the training model is extremely large, the model needs to be pruned, we applied truncation gradient to reduce the number of final weights of the model.

$$f(w_i) = T_1(w_i - \eta \nabla_1 L(w_i, z_i), \eta g_i, \theta),$$

$$T_1(v_j, \alpha, \theta) = \begin{cases} \max(0, v_j - \alpha) & \text{if } v_j \in [0, \theta] \\ \min(0, v_j + \alpha) & \text{if } v_j \in [-\theta, 0] \\ v_j & \text{otherwise} \end{cases}.$$

Where gi R, and gi >=0, if i/K is not an integer, we let gi = 0; if i/K is an integer, we let gi = Kg for a gravity parameter g>0. According to the above formula, it is obvious that gi and determine the degree of sparsity of W. If both of them are large, the sparsity will be stronger. In particular, when gi= , we can use only one parameter g to control sparsity. We can avoid the exploding gradient problem by using this method.

There is an important issue is that DQN is sample inefficient and expensive. In the beginning, when we didn't use any optimization technique, the model need to run 5,6 hours in only 1500 episodes. So, applying some optimization technique is necessary, we will show the technique we use to make it run faster and perform more efficiently.

VIII. Optimizations

A. Frame Skip

In the Arcade Learning Environment, the game outputs sixty frames per second, and the usual setting will let agents submit its action at every single frame. However, it is conceivable that not every frame will provide a new situation which leads to a distinct action. Even human being will not make a decision through every 1/60 second. Therefore, a frame-reducing mechanism must be made to remove redundant frames. We have found a

AAAI Workshop paper showing that reasonable frame skip can be critical to the performance. Frame skip is the number of frames an action is repeated before a new action is selected. Since the action selection is computationally intensive for Atari game, setting a frame skip can significantly decrease the time it takes to simulate an episode, at the little cost of missing opportunities. In our network, we set 15 as the size of the frames skip, which means that we only need to select 4 actions in each second. The setting greatly reduces our tedious training time. With the reduced time, we have more time and more chances to tune other hyperparameters, resulting in better performance.

B. Stacked Frame

While playing the Atari game, we usually need to determine our action by viewing the situation in a period of time. Viewing the movement and current location of each character is critical, especially when the mazes are close to the Pac-Man. Suppose there's a maze at the row right above the Pac-Man. By a little period of time, we can know the maze will turn left or turn right, and we can choose to turn right or turn left to avoid it. The Deep Q-Network is also similar to human. Rather than abstractly viewing the screen for a moment, it takes multiple frames output by ALE as one input. Then, the main network will select the action base on those frames. Stacking frames allows the network to have broader view when selecting the actions. With this method, agents will can select better action, leading to better survival rate and better performance.

C. Experience replay

We know that, when training a Deep neural network, it's crucial to have data which are not mutually related, or it will be biased. Therefore, we also added experience replay technique into our network. Experience replay is a replay memory technique used in reinforcement learning where we store the agent's experiences at each time-step in a data-set, and pooled over many episodes into a replay memory. Then, we randomly sample them with a minibatch of experience, and use it to learn off-policy with Deep Q-network. This method has solved the problem of unstable training which is trained from autocorrelated data set, and making it more like a supervised learning problem.

IX. Difficulties That We Encountered

A. Starting stage of the Agent

To make the agent explore different sector of the map, we set the random value as 0.95 while initiating the model. The random value will gradually decay from 0.95 to 0.05 until the first policy update to the target network. After that, the exploration will be left to the model. This kind of setting can maximize the exploration at the most beginning, with the more sector being explored, the network gains more control on the agent. Originally, we didn't set a point to get rid of the random attribute. This would cause the model becomes unstable, and the initial agent will keep fluttering. That's the reason why we set a point to stop the random factor.

B. The MsPacman-RAM-v0 environment

At the beginning, we spent a great amount of time to train in another MsPacMan environment, which is the RAM based "MsPacman-ram-v0". We thought that the data stored in the RAM might have more deterministic information. With those 128 bytes data, we may get better result comparing to other models trained by frames inputs. However, regardless of how we improve the model, we keep failing while training with the RAM environment. Then, we try to use the "MsPacman-v0", it's easy to outperform the original one. We presume that it's because the data in RAM might not be critical enough to train a network. Furthermore, training the image with CNN allows the agent to understand the situation of all or even a sector of the current game, selecting a more reasonable action.

C. Defining a new reward system

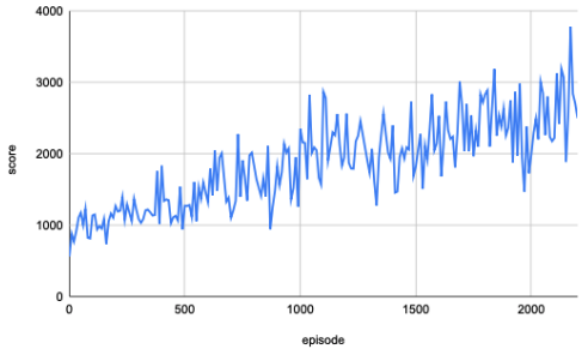
Distinct from the original environments given by gym, we have set a new reward mechanism. After many times of tuning, we added a deducting point situation while the Pac-Man has died. However, weighing different rewards correctly is super difficult.

D. Long Training Times

For each agent that we designed, we have to spend long training times for it to finish training. Often times we have to divide the training into several stages in order for the model to properly be trained due to the limits of google colab.

X. Result

After several different tries we settled with a model with a decent performance. It reached a high score of over 5000 points, with the average score being almost at 4000 points. This is a great performance AI when competing with other individuals, beating them in the average score.



XI. Conclusion

According to this final project, we have a deeper understanding about how to create an AI for playing Games. We not only surprised by the fact that we can make a very powerful AI by the DQN without mastering the game by ourselves, but also comprehend the difficulties of training an agent to play the game. The period after launching the training process and waiting the result is nervous because we have try a lot of time but the result end up very weak. Nevertheless after our agent finally have some progress and eating three ghost in a row, the happiness is beyond description. Hopefully one day, we can try to create an AI on a more complex game and beat every human player.

XII. work distribution

1. Li-Yuan, Chiang(30%): Model design, Model tuning, Research finding.
2. Chia-En, Lu(25%): Source code tracing, RAM environment research, reward design.
3. Zih-Jyun, LIN(25%) : Model design, Model optimization
4. Hao-Chen, Jiang(20%) : Model design, Report organization

References

[1] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari, 2020.