CISC 474 Final Report: Using Deep Q-Learning Approach to Solve 2048 Game

Gavin Liu #20114836

Enzuo Mou #20126862

Jiahao Cai #20099269

Introduction：

The game of 2048 is an exciting tile-shifting game in which you move tiles on a 4x4 board around to combine them, aiming for ever-larger tile values. In the beginning, there will be two tiles randomly placed on the board, two with the number of "2". All tiles are moved to one edge by shifting them up, down, left, or right. Once we have made a move, a new tile will be added to the board. The game then continues until there are no more moves possible. In general, the goal of the game is to reach a tile with a value of 2048. However, the game doesn't stop here, and we

can continue playing as far as it is possible to go, aiming for the largest possible tile. In 2014, 19-year-old Gabriele Cirulli used a weekend to make "2048 ". The next day, "2048" was discovered and forwarded to other websites, and it became popular all of a sudden. Within a few days, people began to discuss how this game made them sleepless, and the number of visitors to the game page also increased at a rapid speed. The challenge of solving this game is that it's a game with a random component. It's impossible to correctly predict not only where each new tile will be placed, but whether it will be a "2" or a "4". Thus, it is impossible to have an algorithm that will always solve the puzzle correctly. At each stage, we can play the probability game and determine what is likely to be the most effective move. However, with the state of the art in reinforcement learning, we decided to use a Deep-Q network to create an agent that can achieve a score of 2048 or higher.

Algorithm and Implementation:

Reinforcement Learning is an aspect of Machine Learning, in which an agent learns to behave in an environment according to the rewards that it gets and certain actions that it performs. Q-learning is one of the most popular algorithms used in reinforcement learning. In Q-learning, When we perform action 'a' in state's', we define a function Q*(s, a) to represent the discounted future reward, and continue optimally from there. However, since Q*(s, a) is unknown, we must use a non-optimal function to estimate it. The whole idea behind Q-learning is that we can improve our approximation of the optimal Q-function using the Bellman equation iteratively. However, if the size of possible states is too large, then the memory required to save and update will not be enough and the time that is required to explore each state to create the required Q-table would be unrealistic. To reduce the size of possible states, a neural network is used to approximate Q-function directly from the environment states because the neural network can model all functions theoretically. And this neural network is called Deep-Q Network. It applies multiple convolution layers for feature extractions. As we could see in the graph at the beginning of the report, the game 2048 does care about the relative position of each grid since only adjacent grids can merge into one. The CNN can preserve and find the features in the graphs. Then the fully connected layers are used to approximate the function.

Loss Function:

$$L(\Theta) = (r + \gamma \max_{a'} Q(s', a'; \Theta)) - Q(s, a; \Theta))^2$$

Activation: RELU

Optimizer: RMSPRop

Problem Formulation:

This project aims to reach the maximum possible cell value in game 2048, and the maximum theoretical value is 131,072. Here are the problems involved in reaching the maximum value:

1. Implement the game 2048
2. Create a suitable environment
3. Build Deep-Q Network
4. Create training method

- State representation:

It is a 4 X 4 matrix initialized with all zeros. Values on the grid can all be represented by a two-dimensional-list like grid [3][3].

- Actions:
    - UP, DOWN, LEFT, RIGHT

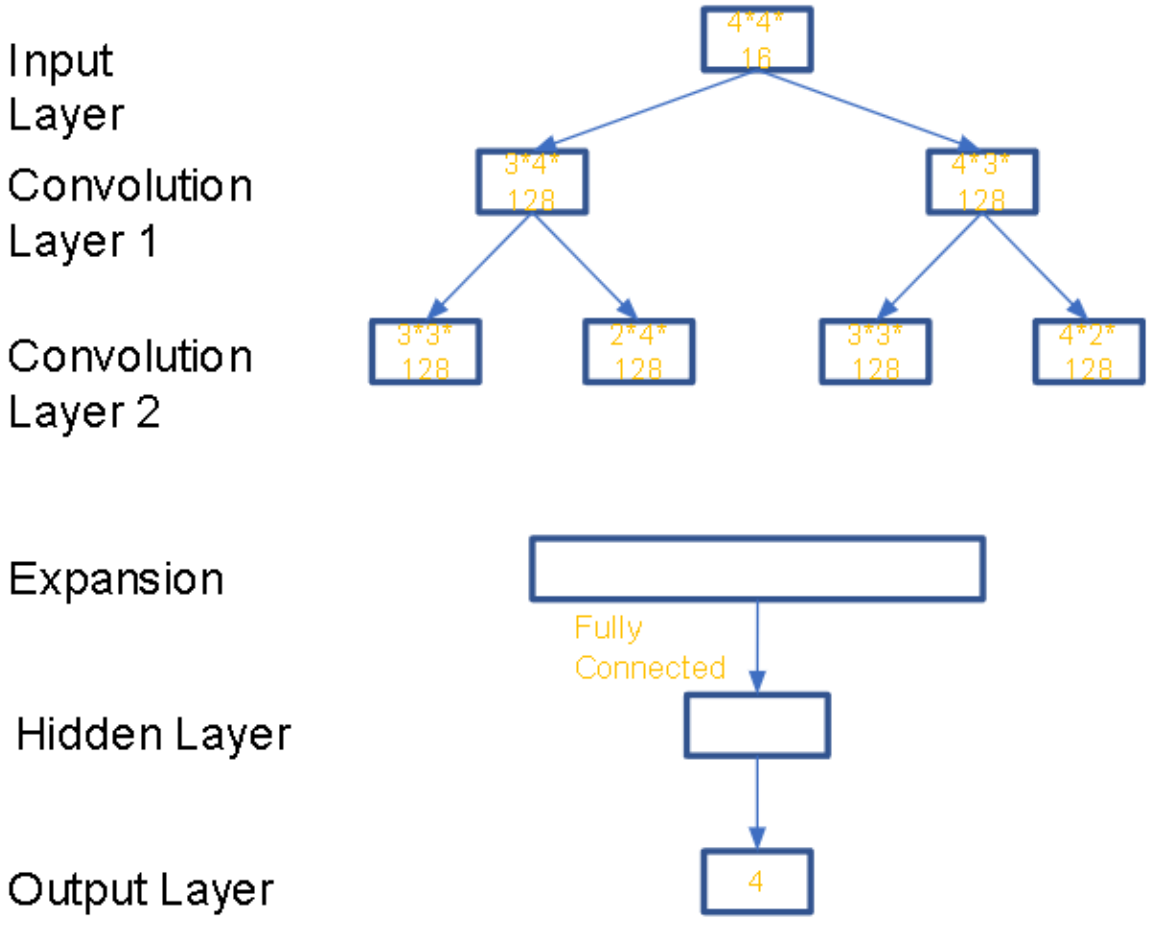
Each time choose one of the up, down, left, and right directions to slide, all the number tiles will move closer to the sliding direction, and the system will also randomly give a number

in an empty tile. The tiles of the same number will add up when they move closer or collide. The number square given by the system is either 2 or 4. The player has to find a way to make up the number "2048".

- Reward:

The reward can be calculated by adding all the values on the grid. When two tiles are combined, the reward will increase. The agent's goal: optimizing their scores at the same time moving towards the target state.

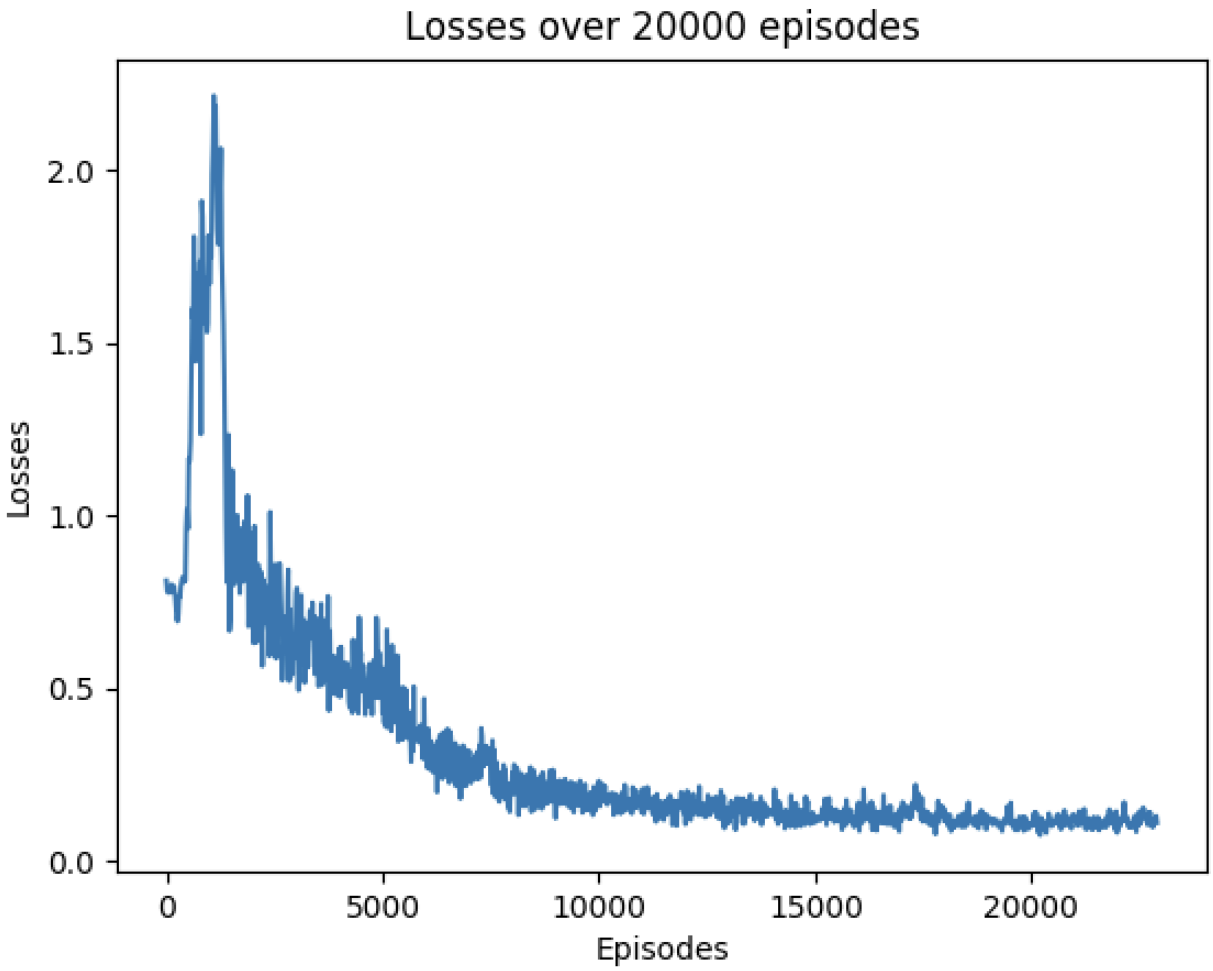
Network Structure:



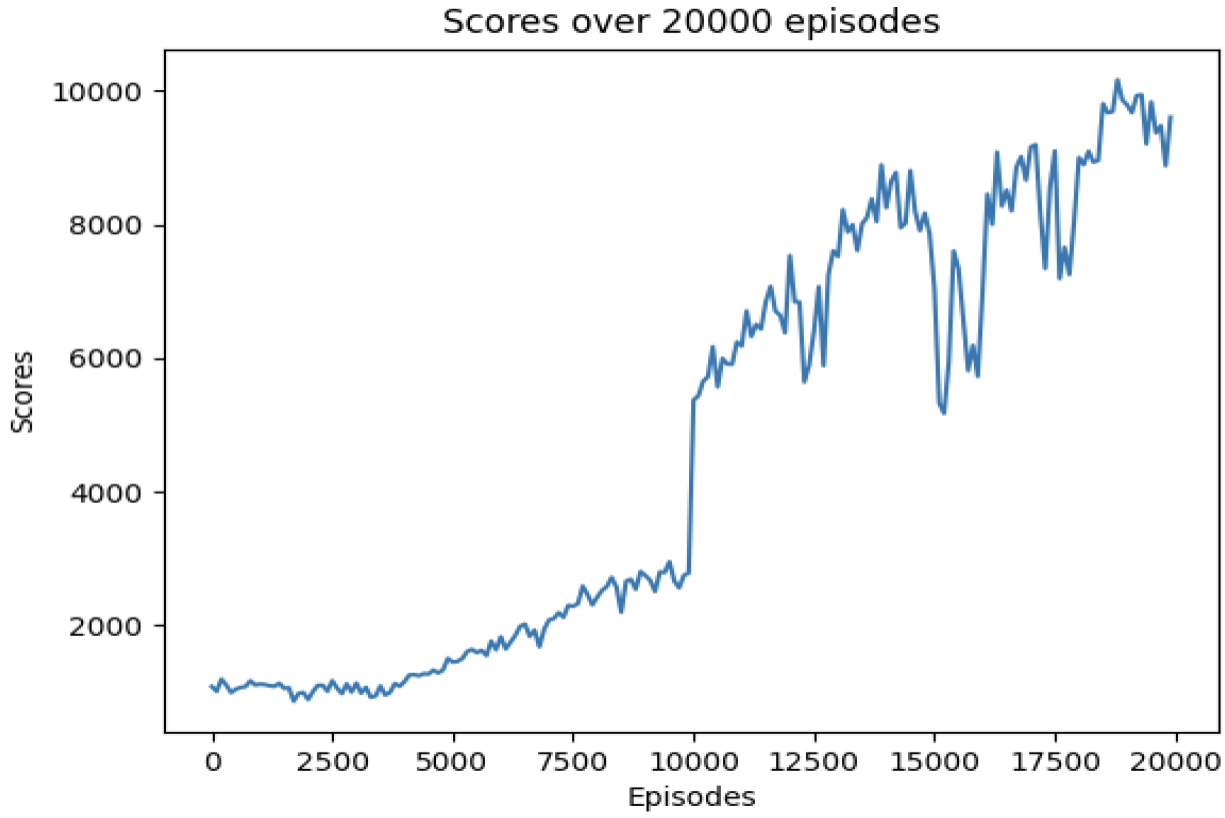
There are two convolutional layers and one ANN for calculating the output. The first input is 4 x 4 with 16 channels. Then it will be input into a convolutional layer with 128 channels. Put the result after this layer to the next convolutional layer and expand each grid in the convolution into a 1d matrix. Calculating the weights and bias gives us and finally choose between the four actions to take.

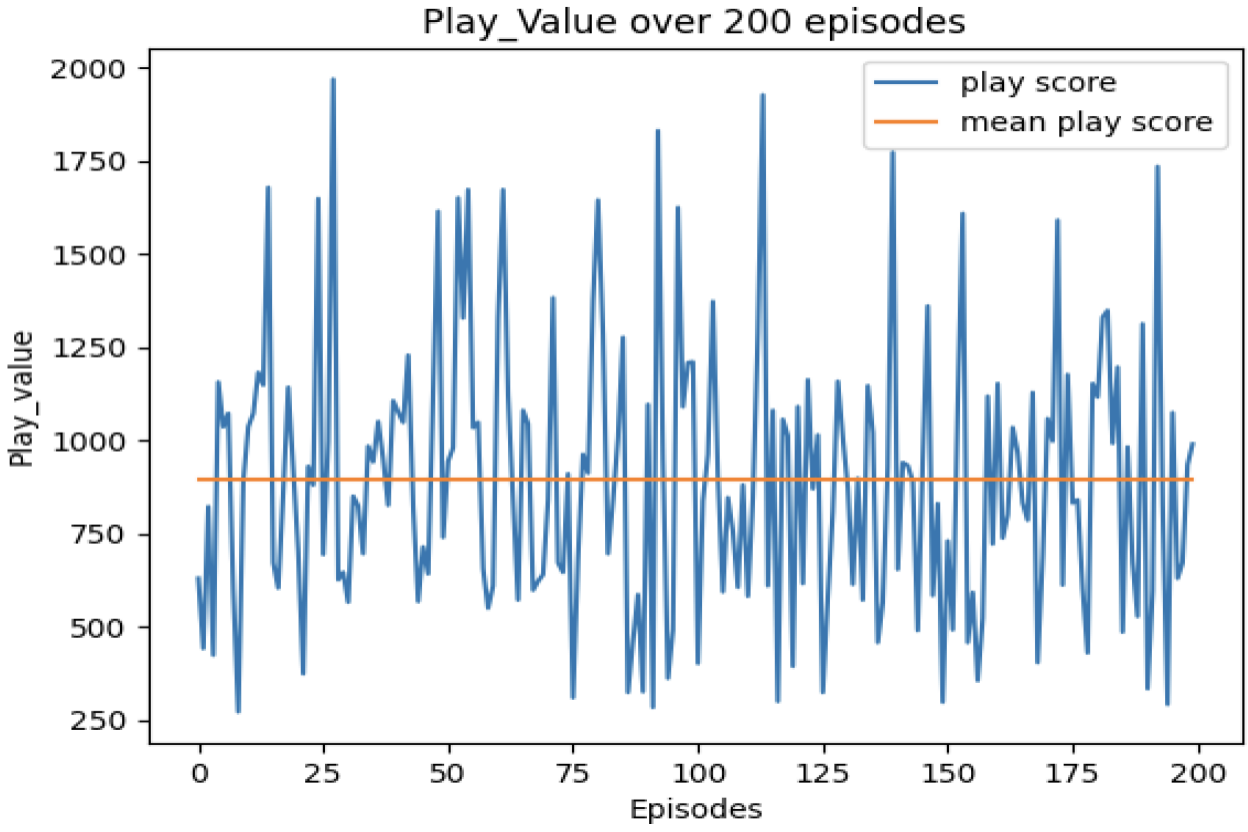
Result:

After training for 20000 episodes, we store the loss value for each episode and plot this diagram. This diagram shows the relation between loss (y-axis) and episodes (x-axis). From the diagram, we can see that the network has been converged.



After that, we also capture the scores over training for each episode. The figure below shows the relationship between scores (y-axis) and episode (x-axis). We can see that there is a leap at the 10000th episode, and the network reached the optimize policy after the 17500th episode.


Scores over 20000 episodes
Scores
Episodes
10000
8000
6000
4000
2000
0
2500
5000
7500
10000
12500
15000
17500
20000


Play_Value over 200 episodes
Play_value
Episodes
play score
mean play score
2000
1750
1500
1250
1000
750
500
250
0
25
50
75
100
125
150
175
200

The diagram above shows the relationship between the score we get using trained weight and the episode. We simulate the real game and let the network play by using the weights we trained. We can see that the highest score is around 2000 and the minimum value is about 250. We also calculate the mean value for the total episode.

Conclusion:

Deep Q Network can deal with the time complexity and the space complexity while training, and approximate the Q-table, which increases the efficiency. This weight Q-Value was trained with limited time and machine environment. With more training data and exploration in the environment, it will bring a better training weight and we will get a better policy and stable result.

Discussion:

Gavin, Enzuo, and Jiahao work together on the implementation of the code. Then, Jiahao Wrote the problem Introduction and problem formulation, Enzuo wrote the Algorithm and implementation part, Gavin wrote the Results part.

Reference:

Learning 2048 with Deep Reinforcement Learning. (n.d.). Retrieved December 6, 2021, from https://cs.uwaterloo.ca/~mli/zalevine-dqn-2048.pdf.

freeCodeCamp.org. (2020, April 7). *A brief introduction to reinforcement learning*. freeCodeCamp.org. Retrieved December 6, 2021, from https://www.freecodecamp.org/news/a-brief-introduction-to-reinforcement-learning-7799af5840db/.

Deep Q-learning: An introduction to deep reinforcement learning. Analytics Vidhya. (2020, April 27). Retrieved December 6, 2021, from https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

Deep reinforcement learning for 2048 - MIT. (n.d.). Retrieved December 6, 2021, from https://www.mit.edu/~amarj/files/2048.pdf.

YangRui2015. (n.d.). *Yangrui2015/2048_env: 2048 environment for Reinforcement Learning and DQN algorithm*. GitHub. Retrieved December 6, 2021, from https://github.com/YangRui2015/2048_env.

SergioIommi. (n.d.). *Sergioiommi/DQN-2048: Deep Reinforcement Learning to play 2048 (with keras)*. GitHub. Retrieved December 6, 2021, from https://github.com/SergioIommi/DQN-2048.

FelipeMarcelino. (n.d.). *Felipemarcelino/2048-gym: This projects aims to use reinforcement learning algorithms to play the game 2048.* GitHub. Retrieved December 6, 2021, from https://github.com/FelipeMarcelino/2048-Gym.