CISC 474 Project Report: A Deep Reinforcement-Learning Approach to Tackle the Game 2048

Tao Ma - 20060593

Yuntian Shan - 20057524

Ricky Zhang - 20053254

Hanyi Li - 20057296

Introduction

2048 is a single-player game played on a 4x4 grid, the objective of this game is to slide numbered tiles on a grid to combine them to create a tile with the number 2048 but you still can keep playing the game, creating tiles with larger numbers. It has enjoyed a great popularity for a long time over the Internet, and it was played by more than 23 million people[1]. During the first three weeks after release, the aggregated time that people devoted to this game is over 3,000 years. While the game is attractive for human players, it also caught the attention of many developers to implement AI methods to tackle this game. Reinforcement Learning interests in how software agents ought to take actions in an environment such that the benefits they get can be maximized is a proper

way to tackle this game. After learning the state of the art in Reinforcement Learning, we chose to use Deep-Q Network in this project to create an intelligent agent to play 2048 such that they can reach the score 2048 or even higher scores in general.

Problem Formulation

The ultimate problem to be solved in this project is to reach the maximum possible cell value in game 2048, and the highest value theoretically is 131,072. To reach the maximum goal, the subsequent problems are:

1. Correctly simulate the logic of the game
2. Define reasonable state representation, state value, and the reward
3. Create efficient Deep-Q Network
4. Generate proper training

Background

Reinforcement learning is an area of machine learning that has been studied by psychologists over the past 60 years, it concerns teaching agents to perform the necessary actions in some environment. Q-learning is one of the most popular reinforcement learning algorithms, the aim of Q-learning is to learn an optimal policy

by learning a state-action value function. The optimal policy is learned by starting from a random policy and performancing update rules to eventually converge to the optimal policy. As Q-learning can be generally applied to any MDP, for some practical points, it is not always a feasible solution.The most common encountered problem is that the limited memory is not affordable for the large number of possible states. The Q-table is estimated to be $C(1,16) \times 10^{15} \times 4$. The solution for reducing the size of the Q-table is to use a neural network model to learn an approximation of the Q-function directly from the environment state since the neural network is able to model all functions theoretically. In this case, the Q-function is also a function of the weights of the neural network, and the loss function is then defined as:

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

The resulting model is named Deep-Q Network, which is used in our project. This network apply multiple convolution layers in a parallel fashion for feature extraction. Fully connected layers are being put after the feature extraction part to approximate the function.

Design

- Environment State Representation

  4 by 4 matrix initialized with zeros. Value accessed from the specific matrix indexing is regarded as the game value on the 2048 game. For example, grid[1][2] = 1024 gives us the information that game value 1024 are on the second row and the third column grid. An empty tile will b erepresented as having a 0 on that tile.

- Actions

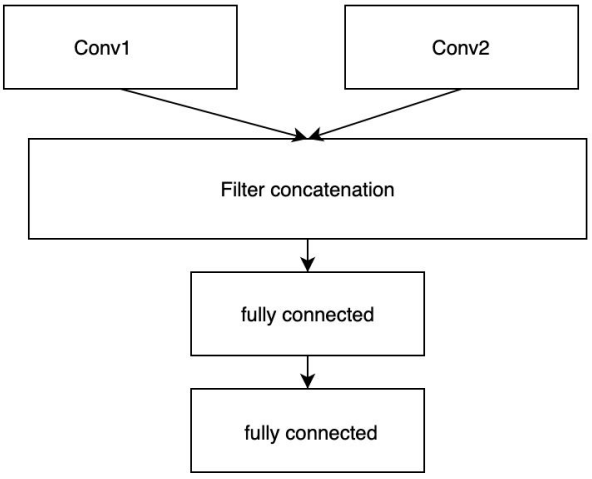
  UP, DOWN, LEFT, RIGHT

  For all the action, If two tiles of the same number are crushed together while moving, they will merge into a tile with the total value of the two tiles that collided. Each action represents the direction that can be applied on the gameboard to move the tiles. Every action would cause the gameboard to have some “empty” tiles and a new tile will randomly appear in an “empty” spot on the board with a value of either 2 or 4.

- Reward

  The reward can be calculated by adding up all the values on the grid. The reward increases when two tiles combine together, by the value of the new tile. The goal of the agent can be seen as having too parts: optimizing their score while moving toward the goal state(a state where there is one 2048 exist).

- Network Summary(Q-table estimation)



The graph of the network is shown above. The network applies convolution layers in parallel fashion and then concatenates them together. The reason for doing this is to capture more features at the same level. “Wider” instead of

"Deeper". Conventionally, when thinking about improving the performance of the feature extraction, deeper architecture would be preferred. However, that would cause huge computation complexity and sometimes ignore the features on the same level. Not to mention the overfitting problem that deep architecture can bring to us. Therefore, our own network remains as shallow but wide to capture more features. After the feature extraction layers, two fully connected layers are used to try to approximate the q-table function. The activation functions used in this network are relu everywhere due to the fact that it has advantage on its simplicitysimplicity (derivative of relu is as simple as the input value itself).

- Training Process

  The training process uses a back-propagation algorithm for updating weight.

Results

After training for 20000 episodes, the captured loss and score over episodes, shown in Figure 1 and 2, demonstrates that the network is approaching an optimized Q table.

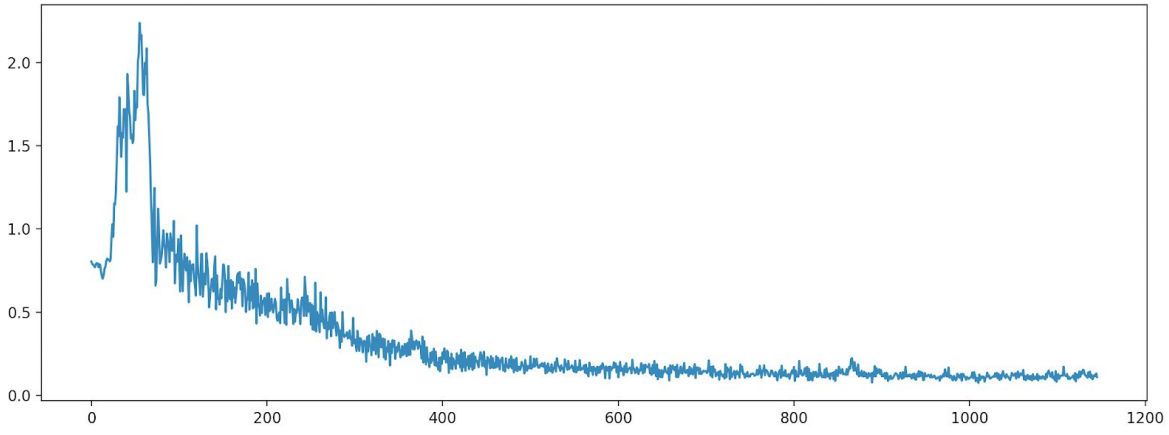


Figure 1 - Loss over training for 20000 episodes

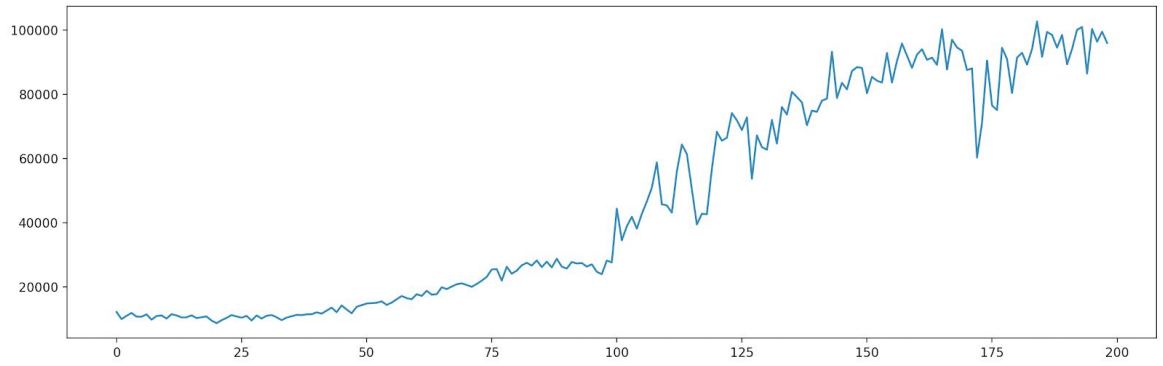


Figure 2 - Total score vs episodes (averaged per 100 episodes)

Then, the game is played for 200 rounds, with random policies, and policies generated from the learned network. Comparing the results in Figure 3, Deep Q Learning method is proven to be effective as it gives a consistently higher final board value (the total value on the game board after the game ends).

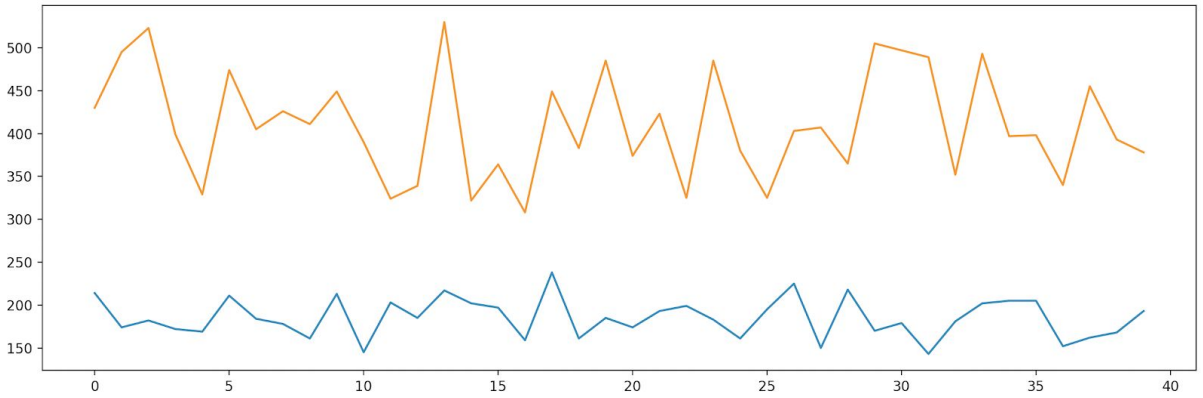


Figure 3 - Final board value: random policy vs trained policy, averaged per 5 games

Conclusion

This project tends to use reinforcement learning methods to solve the 2048 game. Moreover, in order to deal with the time complexity and the space complexity while training and exploring, this project applies a neural network to approximate the Q-table therefore largely increasing the efficiency. The result showed that our

algorithm has outperformed random baseline policy. There are great potential for the network to be improved. The current weight Q-Value approximations are trained with limitation of time and machine. Leaving the network architecture unchanged with more training data and exploration in the environment will promise a better result.

References

Learning 2048 with Deep Reinforcement Learning (2020). Retrieved 8 December 2020, from https://cs.uwaterloo.ca/~mli/zalevine-dqn-2048.pdf

SergioIommi/DQN-2048. (2020). Retrieved 8 December 2020, from https://github.com/SergioIommi/DQN-2048

a 2048 deep AI that does not suck. (2020). Retrieved 8 December 2020, from https://tjwei.github.io/2048-NN/

FelipeMarcelino/2048-Gym. (2020). Retrieved 8 December 2020, from https://github.com/FelipeMarcelino/2048-Gym

navjindervirdee/2048-deep-reinforcement-learning. (2020). Retrieved 8 December 2020, from https://github.com/navjindervirdee/2048-deep-reinforcement-learning

Raj, B. (2020, July 31). A Simple Guide to the Versions of the Inception Network. Retrieved December 08, 2020, from https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Want to Stay Anonymous? Don't Make a Hit Computer Game. (2014, March 18). Retrieved December 08, 2020, from https://blogs.wsj.com/digits/2014/03/18/want-to-stay-anonymous-dont-make-a-hit-computer-game/

(n.d.). Retrieved December 08, 2020, from https://abcnews.go.com/Technology/2048-mobile-game-eat-time/story?id=23037239

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1):1–127.