Learning 2048 by Reinforcement Learning

Chun Wing Wong[*], Asbin Rai[**], Wentao Xu[**], and Xiaowen Man[**]

[*]Dept. of Physics & Astronomy
[**]Dept. of Information Studies
University College London, WC1E 6BT

May 5, 2020

Abstract

2048 is a very popular decision base game on mobile devices. The game has very simple concept and very clear targets to achieve, which makes it an interesting case for an AI to learn through reinforcement learning. We have, therefore, decided to investigate this and multiple simulations are made. Unfortunately, we are not very convinced that our method has succeeded in training a competent AI to play the game.

1 Introduction

2048 has become a very popular game in the past 10 years. It has a very simple concept which involves moving the tiles around trying to achieve higher scores. As a group of people who enjoyed the game before, we have decided to attempt training an AI to play this game through pure reinforcement learning. Multiple methods have been attempted but the results prove that 2048 perhaps is a game too complicated, despite the simplicity in the game concept, to be learnt through pure reinforcement learning. According to some online resources, it is shown that deep reinforcement learning with a 3 layer neural network is required to achieve decent results [1].

In the following, we will first present some background information such as a detailed explanation of the game mechanics. Then, we will present the method we used during the training process in the following section. Results will be presented after methods followed by a discussion section which will explain why we think our method has failed.

2 Background

2.1 Game Rules

The game is played on a 4x4 grid. Initially, there will be two tiles on the board with values "2" or "4". The players will make moves "Left", "Right", "Up" and "Down". If the adjacent tile in the same direction of the move is the same, the two tiles will merge and form a new tile with values of the sum of the original value. Moves are not allowed if no tiles changes position after that invalid move. After each valid move, the a new tile of either "2" or "4" will be generated and randomly assigned to one of the empty tile. The goal of this game is to obtain a tile of value "2048" after a series of moves. When there are no longer valid moves available to the player, the game is terminated and the player looses the game.

2.2 Reinforcement Learning

Reinforcement learning is learning what actions to take in a certain situation that will maximise our rewards. The learner is not told what actions to take, but instead must discover which actions yield the most reward by trying them [2]. In our project, we will have 4 actions available for our agent, they are "Left", "Right", "Up" and "Down". We will need a representation to present the current state of the board. This will be

denoted as ϕ in the rest of the report. We have chosen the simplest approximation — linear approximation — where the q-value of the action is related to the state linearly as shown below:

$$q_a = \phi^T w_a$$

In order to play the game better, we have to update the weight after each episode. In this project, multiple schemes such as Monte-carlo has been considered but we have eventually decided to stick with the q-learning scheme.

Finally, after calculating the q-value, we choose to use epsilon-greedy policy to decide which action the agent takes. Furthermore, in the second simulation, we have decided to adopt the decaying epsilon-greedy policy, where we allow the agent to explore in the beginning but slowly decreasing the freedom of exploration as we reach the later stages of the learning.

3 Method

3.1 Game Mechanics in Simulations

Originally, we have considered simplifying the game mechanics by having a specific starting state and have a very fixed way of spawning new tiles to reduce randomness. However, we believe that this will defeat the whole point of the game, which mainly involves making moves that will limit the damage of randomness could do to our game. Thus, we have decided to follow the exact game mechanics with no reduction and simplification in the rules. This means that we will always start with a random board with 2 tiles with values on each of the tiles "2" or "4". Then, after each valid move, a new tile with values "2" or "4" will be generated on a random empty tile. We loose when no valid moves are allowed.

3.2 Representation and Q-value Calculation

In our experiment, we have tried 2 types of representation, which are the simple representation and the relationship representation. They will be explained as follows. To calculate the q-value, we will use the linear approximation approach for simplicity.

3.2.1 Simple Representation

This is the simplest type of representation of our game, which is just simply a vector of 16 elements with each element representing the values of the individual tiles. For example, if we look at figure 1, the state will simply be:

$$\phi = (0, 0, 2, 4, 0, 0, 4, 8, 0, 2, 16, 32, 0, 2, 2, 16)^T$$

To calculate the q-value, we will use our state vector acting on the weight vector representing each action, which can be written as:

$$q_a = \phi^T w_a$$

where q_a is the q-value of that action and w_a is the weight of that action.

3.2.2 Relationship Representation

Reading Barto's "Reinforcement Learning" book, I have learnt that, to improve my learning, my state vector needs to have information involving interaction between the tiles. In addition to that, I have also decided to use the book's convention where the action dependency will be put in the state instead of the weight vector. This means our Linear Approximation equation can be re-written as:

$$q_a = \phi(a)^T w$$

When we play the game, whenever we consider a horizontal move, we always compare the tiles horizontally. This means the interaction considered will be horizontal. We will represent the interaction by the difference between 2 tiles. Our state vector for move left will then be written as:

$$\phi(L) = (\text{values of all 16 tiles}, \text{horizontal differences between each tile of each row})^T$$

To distinguish move "L" and "R" the horizontal differences in $\phi(R)$ will have opposite sign to that in $\phi(L)$. $\phi(U)$ and $\phi(D)$ can be calculated with similar method except we will consider vertical differences instead.

Figure 1: An example game board. This image is obtained from Wikipedia.

3.3 Actions

As explained before, there are 4 available actions for the agent to choose from. They are "Left", "Right", "Up" and "Down". Our agent will choose the action by calculating the q-value of each action. Since there are lots of possible traces in this game, we choose the action using the epsilon-greedy policy to allow exploration by the agent. The epsilon-greedy policy is:

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, & a = argmax_{a'}(q(s,a')) \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases} \tag{1}$$

where $|A|$ is the number of available actions the agent has. In our case, $|A|$ will be most likely to be 4. As mentioned above, there are a huge possibility of traces in this game. To ensure we don't waste time exploring invalid moves, we have decided to restrict our agent to only choose from valid moves. This means if "Left" is not a valid move, we will only have 3 actions available for the agent to choose from and $|A| = 3$ in this case instead assuming the other 3 moves are still valid.

From multiple sources, I have also seen that instead of using a pure epsilon-greedy policy, people tend to use a decaying epsilon-greedy policy. This allows exploration at the beginning but it decreases the freedom of exploration as we get to the later episodes of the learning process. We have implemented this by a rather naive step-wise decaying scheme.

3.4 Reward Schemes

Multiple reward schemes has been attempted and they both have their merit and drawbacks. We started off with a simple reward scheme where it will reward the value of the greatest value on the board after a move. For example, if after a move we obtain a board state shown in figure 1, we will get 32 points.

Another reward scheme we have attempted will be the merge-reward scheme, where the agent will be rewarded whenever we merge two tiles into one. The reward will be the value of the new tile. For example, if the initial board is that shown in figure 1 and we move right, the two "2"s in the bottom role will merge and give a new tile "4". This will give us 4 points.

3.5 Learning

We have mainly used the q-learning equation when updating the weight vector, which is:

$$\begin{aligned} \Delta w_a &= \alpha(R_{t+1} + \gamma \text{max}_a(S_{t+1}(a), w_{t+1}) - q(S_t; w_t)\nabla_w q(S_t; w_t)) \\ &= \alpha(R_{t+1} + \gamma \text{max}_a(S_{t+1}(a), w_{t+1}) - q(S_t; w_t)S_t)) \end{aligned} \tag{2}$$

where we have taken $\alpha = 0.0000001$, $\gamma = 0.9$.

We used the decaying-epsilon-greedy policy as explained above with $\epsilon = 0.1$ initially. Finally, we have decided to play the game with at least 1000 episodes.

4 Results

As mentioned above, we tried 2 types of representation and have chosen to do each learning session in increments of 1,000 episodes starting from 1,000 episodes to 10,000 episodes. This was to track the progress of the algorithms and to see if there were any learning trends. The results for them will be shown below.

4.1 Simple Representation

The results from 10,000 episodes of the simple representation is shown below.

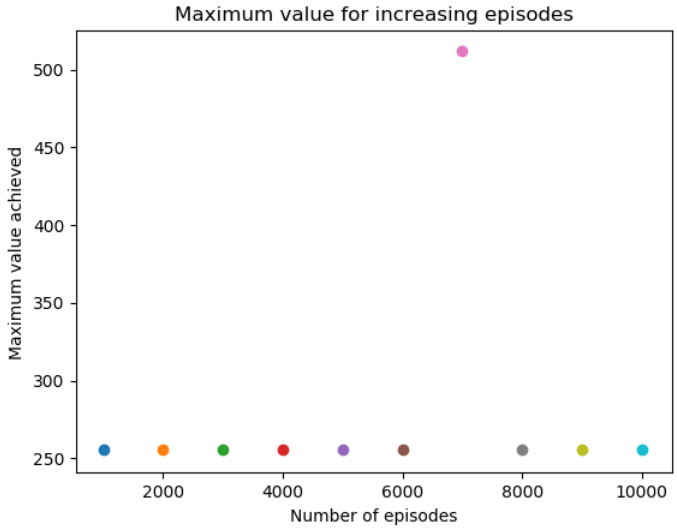

Figure 2: Results from simple representation with increasing episodes

From the results shown in figure 2, it can clearly be seen that this representation is not very effective at making our agent learn how to play 2048. However, the agent was able to reach the 256 tile quickly and consistently, which could mean that the agent was either able to visit the earlier states more often, therefore getting more accurate q-values for certain states and actions, or simply chose lucky actions. Furthermore, reaching the 512 tile only once out of 10 increasing episodes shows that the tile was reached with a high level of luck rather than knowledge of the action. We have two main reasons to believe why this representation was not capable of teaching the agent 2048. The first one being that the representation did not have enough information on how the tiles relate to each other, thus not allowing the agent to choose actions based on the certain factors of the board. The second one being that there are too many combination of states and that the agent was not able to visit all the states in the amount of episodes provided.

4.2 Relationship Representation

The results from 10,000 episodes of the relationship representation is shown below.

The results from figure 2 show that this representation is evidently more effective in learning 2048 compared to the previous representation, given the fact that it was able to reach the 512 tile with less training episodes and more consistently. However, it can also be seen that the maximum value achieved by this representation is the same as the previous representation, demonstrating that this representation was also not able to effectively teach the agent 2048. We believe that adding additional information such as interaction between tiles in the state vector has improved both the learning rate and consistency of the agent. However, the problem mentioned in the previous experiment still remains. This representation is still not able to account for the fact that there will be an impossibly large amount of unique states for the agents to visit, implying that the agent may not be able to visit every state in a feasible amount of time. On the other hand, it could be argued that there was not enough training time for the agent to completely learn 2048. Perhaps the agent would've able to learn 2048 if a larger amount of episodes were used to train the agent?

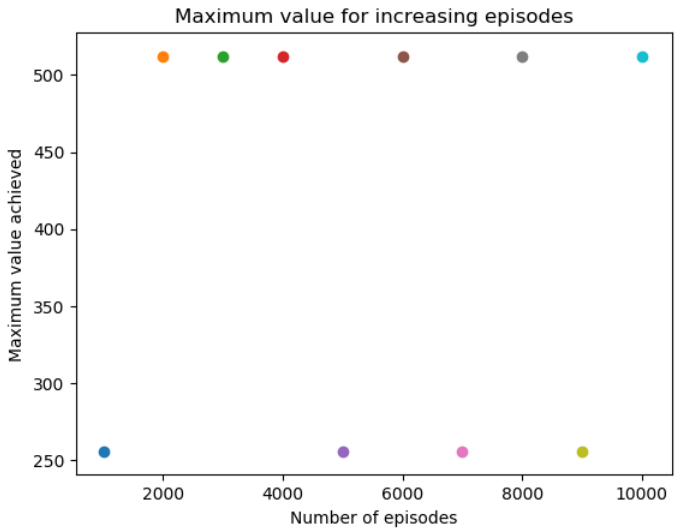

Figure 3: Results from relationship representation with increasing episodes

5 Discussion

In this report, we have presented our attempt to learn playing the game 2048. We have started with the simple representation which is simply a vector with all the values of tiles on the board. We used the a linear value approximation and the weight is updated by the q-learning scheme. Unfortunately, after 1000 episode learning process, there's no significant improvement in the outcome of the learning. Increasing the number of episodes turns out to have very little impact and our agent is still barely better than a random agent. We have come up with a few possible explaination, which are:

1. Our representation does not provide enough information for our agent to make good decisions. When we play the game, we don't make moves only based on the values of each individual tiles, we make moves based on the relationships between adjacent tiles. Therefore, perhaps we have to present this peice of information to our agent.

2. The number of states available is so large that by running 1000 episodes or even 10000 episodes, we still have not experienced most available states. Thus, our learning is still not very complete.

3. Randomness of the game means that even if we choose the best move possible that reduces the impact of randomness, it is still possible that a game ruining tile will spawn. This will give a wrong message to the AI that it is a bad move. This might impact our learning.

Point 2 and 3 are very much the restriction of the game and our current computation power, so we realistically can only improve through point 1. To do so, we have added extra terms in our state vector which encodes the differences between adjacent tiles, which we called the relationship representation. This will hopefully provide more information for our agent to make better moves.

Declaration

This document represents the group report submission from the named authors for the project assignment of module: Foundations of Machine Learning and Data Science (INST0060), 2018-19. In submitting this document, the authors certify that all submissions for this project are a fair representation of their own work and satisfy the UCL regulations on plagiarism.

References

[1] Antoine Dedieu and Jonathan Amar. Deep reinforcement learning for 2048. In *Conference on Neural Information Processing Systems (NIPS), 31st, Long Beach, CA, USA*, 2017.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.