AI vs 2048

William Scott - MT18026

Suraj Pandey - MT18025

Abstract - This document is a report for the project on 2048 in which we implemented 5 different searching and learning algorithms which can play the game 2048 trying to make the maximum score.

I. Introduction

The game 2048 is one of the most famous puzzle games of this century. It is a 4x4 grid game, where a new 2 or 4 tiles is introduced randomly in a random free grid, and the play has swipe options in 4 directions, leading to merge the similar blocks which are next to each other. In this game, we consider a win if we make a block of 2048. Our goal is to design AI Models which will help to maximize the score. The algorithms we are using are Minimax, Expectimax, RL, NN with RL and NN with Minimax. After the implementation, we will compare and contrast the working and performance of all the algorithms and pick the best one.

II. Problem Statement

In this game, a win is considered when a the tile reaches 2048. But the game can be continued even after reaching reaching that tile. Our optimization problem in game is the score. For score optimization, we need to design the algorithms in such a way that the action that it predicts will be the best possible action at that particular state.

III. Literature Review

AI people always love a challenge. Ever since 2048 was introduced, many AI works were done to build an AI which can win this game. While there were many success stories, Interestingly, till today the study continues on how to improve the score. Going through different implementations, we noticed that 2048 is exploited by all kinds of searching and learning (both supervised and unsupervised) algorithms which work exceptionally good.

As 2048 is a strategic game, searching algorithms tends to perform a little better on this as we design the search strategy to exactly match the problem statement. Learning algorithms on the other hand tend to just learn the pattern in which the moves are taken, so even though they can still predict the moves, the final score is not up to the mark. Neural Network is a supervised learning and Reinforcement Learning is a unsupervised learning. With neural network the training should be done on some data which is made by some already intelligently played actions. In reinforcement learning the learning is based on the reward. Each has its own drawbacks and advantages, which we will explain in our methodologies.

IV. Framework Used

Gym-2048 - only for RL

V. Approach

A. Minimax

In minimax, the max is the maximum value of all the swipes and min is the minimum value of placing a 2 or 4 on all the free slots. In this

algorithm, max is called by min and min is called by max, this kind of approach makes sure that the chances of winning of max is high and the chances of winning of min is less. Swipe is max and random placement is min in our case. "*max - min - max*"
In minimax, we have a concept of depth. This basically tells the algorithm how deep it should check to predict the next state. The higher the depth, the higher the score. But due to the computational limitations, we explored only depths at maximum of 4
In minimax there are many different combinations of heuristics that we tried applying. Upon trial and error we chose the following heuristics as best for minimax.

Alpha beta pruning is a technique using which we calculate the most improbable states and avoid taking those actions.

Best Heuristics:
- Snake pattern
  - Number of zeroes on the board
  - Number of merges

B. Expectimax
We implemented this algorithm as "*max - expectation - min - max*". Here, max is the maximum value of all the swipes, expectation is the average value of the probability of placing 2 or 4, and min is the minimum of placing 2 or 4 on the free slots available. Though this algorithm gave decent results. By removing the min from the algorithm, the score seemed to increase.
In the snake pattern heuristics, we basically assign weights to the whole board with the snake increasing values of powers of 4 in a snake pattern.

Best Heuristics:
- Monotonicity
- Greater values on edges
- Snake Pattern Weights (tried with and without)

$$eval(s) = \sum_{i=0}^{3} \sum_{j=0}^{3} weight[i][j] \times board[i][j]$$

Pseudocode:

```
def value(s)
  if s is a max node return maxValue(s)
  if s is an exp node return expValue(s)
  if s is a terminal node return evaluation(s)

def maxValue(s)
  values = [value(s') for s'in successors(s)]
  return max(values)

def expValue(s)
  values = [value(s') for s'in successors(s)]
  weights = [probability(s, s') for s'in
    successors(s)]
  return expectation(values, weights)
```

C. Q-Learning
Q-Learning is a RL based algorithm which basically learns with the desired action by the reward we give to each of the action it takes. For this to run the environment a little better, we used gym-2048 environment which takes the steps and gives back reward and new state and so on. But in that environment the reward its giving to each of the state is 0 other than the final state. So this is not much helpful for the RL to learn effectively.

Mainly in q-learning we need to maintain the q-table with respective to the number of states, but

in games like this, a particular configuration of the board is considered as a state. And as the game tiles grows in number, the number of possible states increases exponentially.

Hyperparameters:

1. Learning Factor
2. Discount Factor
3. Exploration rate
4. Decay rate

These all parameters will contribute such that RL will work effectively.

Learning rate: is rate at which algorithm will learn, less the learning rate , more powerful will be learning.

Discount factor: is factor or weightage given for future state reward. More the discount factor more will be contribution of future states's rewards.

Exploration rate: is rate by which the agent can explore randomly the path without taking care of Q-table. Less the exploration rate , more good will be learning.

Decay rate: is rate at which the exploration rate will decreases as time increases.

So basically, to reduce the number of states to a fixed number of states, we considered there factors to makeup the q-table.

1.Monotonicity in rows and columns

- We considered a monotonicity in each row and column which are 4 for row and 4 for column, resulting in 2^8 combinations

2. Number of Empty tiles

- The total number of empty cells possible are 15

3. Number of merges

- The total number of merges possible are 8 + 1 when there is no merge possible.

Now, the q-table is of 2^8 x 15 x 9 size. The number of states this holds seem a lot but the in reality we reduced a exponential table to a fixed size.

Q-table Update Rule

$Q(s_t,a_t) = Q(s_t,a_t) + learning\_rate\ (reward + discount\_factor * max(Q(s_{t+1},a_{t+1})) - Q(s_t,a_t))$

We were able to successfully develop this algorithm, but failed to achieve the score. Though we were able to reduce the number of states to a fixed number, the problem arises when the q-table has to be updated multiple times. The q-table has almost 30,000 states and each state have 4 actions. To generate an effective q-table, we need to run for millions of episodes which is not feasible. And without running for those many episodes the results will not be just good.

D. Neural Networks with Q-Learning

This is the additional algorithm we did, by applying the Q-Learned (State, Action, Reward) values to the Neural Network to learn. Our main motivation for doing this algorithm is with the idea that we can make the neural networks learn the correct actions that the RL is taking, and we implemented this logic by having a threshold limit on the reward that the (State, Action, Reward) has. We take a dump of (State, Action, Reward) for each action move for each episode and with this help we generate our dataset in

which the state is x and action is y. When we feed this to the neural network, we basically get the action that we need to take in the end.

Neural Network Architecture:

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 100)	1700
dense_5 (Dense)	(None, 100)	10100
dense_6 (Dense)	(None, 10)	1010
dense_7 (Dense)	(None, 4)	44

Total params: 12,854
Trainable params: 12,854
Non-trainable params: 0

Layers:

- Input Layer
- Hidden Layers (100, 100, 10)
  - Relu Activation Function
- Output Layer
  - Softmax Activation (4)

Activation Functions:

Relu:

max(0, x)

This activation function is a ramp function which only considers the positive part of the values.

Softmax equation

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{yj}}$$

Softmax is a probabilistic activation function which considers the output values of other neurons before assigning values. Due to this, the sum of all the neurons will be 1.

Few times for some unknown states, the neural network bluntly takes a invalid move, so just to avoid from getting struck in a particular state, we included a exploration_rate where the model will be taking a random step.

Training Accuracy: 57%

Number of Episodes: 100

E. Neural Networks with Minimax

In this, the architecture is same as the above mentioned Q-Learning NN architecture, but in this we train the dataset from minimax dump files rather than from the Q-Learning.

This seems to be a better idea because our RL does not seem to work better as the number of training episodes were not enough for it to take good decisions.

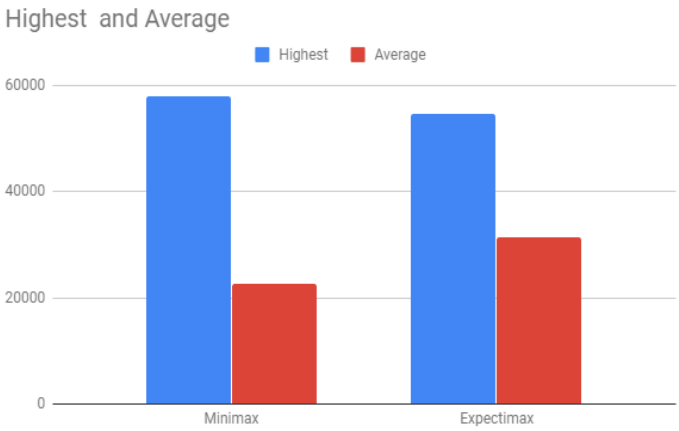
Training Accuracy: 88%

V. RESULTS

A. Minimax and Expectimax

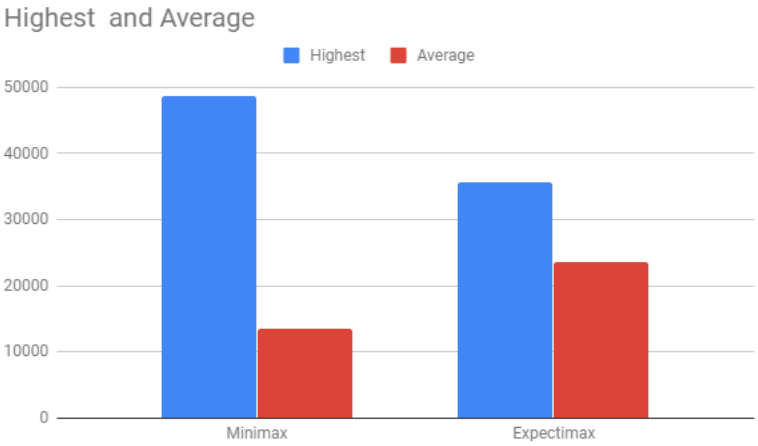
- The results with minimax and expectimax are stunning, in fact, our implementation outperformed the scores, from our references.
- As the increase in depth demands huge computational power, we only computed for the depths 2 and 3. But, with increase in the depth of the algorithm, the score will increase drastically.
- With minimax and expectimax on depth 2 & 3, we were able to achieve a max block of 4096. We ran the algorithm for 100 iterations to formulate the following statistics.
- Computation of expectimax is a bit slow compared to minimax, as at each depth it will be traversing to 3 levels.

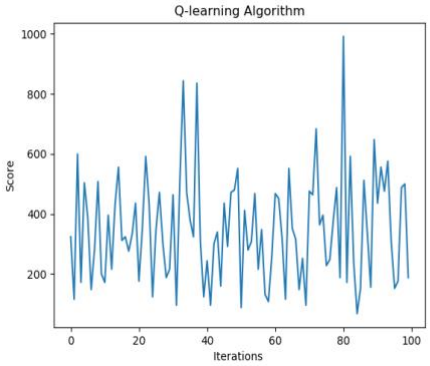
Minimax vs Expectimax - depth 2:

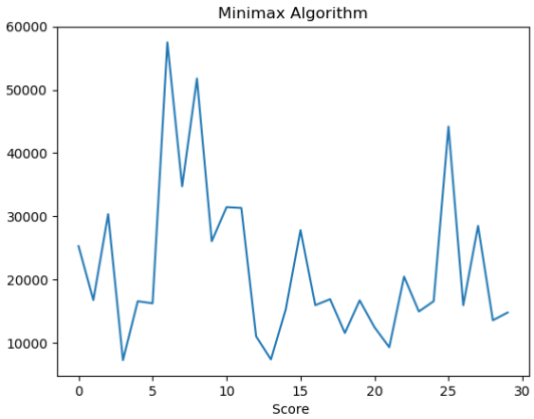


	Minimax		Expectimax	
Depth	2	3	2	3
High Score	48628	58012	35608	54660
Average	13502	22728	23605	31402
256	100%	100%	100%	100%
512	93%	100%	84%	84%
1024	72%	87%	49%	71%
2048	17%	46%	9%	34%
4096	1%	3%	-	2%
8192	-	-	-	-

Minimax vs Expectimax - depth 3:







B. Learning:

Due to inability to train for high number of episodes, the q-learning didn't perform well

	Q-Learning
128	100%
256	72%
512	0%
	NN Q-Learning
128	100%
256	88%
512	1%
	NN Q-Learning
128	100%
256	93%
512	10%

VI. TAKE AWAY FROM RESULTS

The Minimax and Expectimax is working well with achieving the 2048 with accuracy 17% and 9%.

But since the Q-learning take trillions of states , so we have to transform it from less numbered states such that to apply the Q-learning.

The Q-learning will give atmost 256 with less accuracy of that also (10%).

The Deep Q-learning will give accuracy of 80% for 256 and sometimes 512.

So, It is also not as powerful as Minimax and Expectimax.

We also applied Neural Network in Minimax and it gave accuracy of 88% training accuracy.

VII. CONCLUSION

2048 is an intriguing and addictive puzzle game. The AI for this game is still in its infancy, and while much progress has already been made, there is still plenty of room for improvement. The game has a large element of luck involved, so a good AI must minimise the risk, but not so much as to rule out greatness.

And Minimax and Expectimax will work perfectly to get the 2048. And Q-learning and Neural Network will fail to achieve such accuracy.

VIII. REFERENCES

[1] AI 2048 Paper by Standford University

[2] Investigation into AI strategies IEEE