Minesweeper - Techniques of artificial intelligence

Project report

Anthony Zhou, Hamza EL Miri & Julien Baudru

March 31, 2022

Contents

1 Introduction

Following its integration by standard on Windows 3.1 systems in 1992, The Minesweeper quickly became one of the most popular puzzle games of the century. The goal of this single-player game is simple, the player must discover as many tiles in the game as possible while trying to never click on a mine. The player has clues about the potential positions of the mines, indeed each discovered tile contains a number indicating how many mines are in its neighborhood.
In these pages, we will present the different points we needed to create an artificial intelligence capable of playing the game described above in the most successful way possible.

2 Models

In this section, the different solutions found to solve the given problem will be presented and compared, these solutions are a solver (an algorithmic logical approach), a neural network and reinforcement learning.

2.1 Solver

Before implementing any machine learning technique, we first designed a solver for the minesweeper game. The interest of proceeding in this way is to be able to compare the results obtained by the different models tested during this section with a more classical algorithmic solution.

2.1.1 Algorithm

The way we designed this solver is as follows: TODO

2.1.2 Testing

2.2 Convolutional neural network (CNN)

To solve the given problem, we have chosen a convolutional neural network (CNN). We chose this type of architecture because they are usually used in image processing and are therefore effective when it is a question of detecting features of the matrices of values that are given to them as input. More precisely, the chosen architecture is close to the famous *AlexNet* network, the different layers of our network are detailed on figure 1.

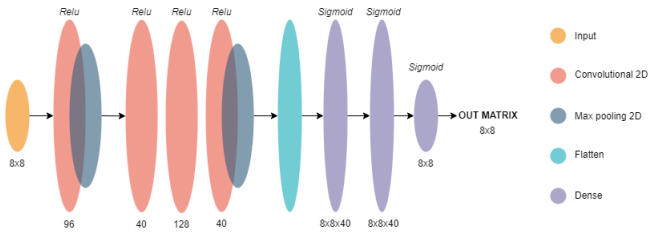
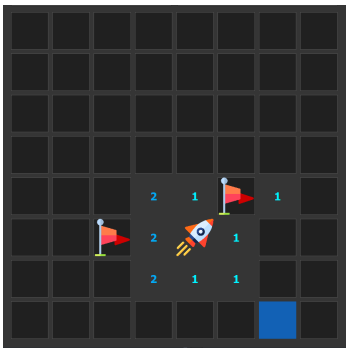


Figure 1: Architecture of our model based on the AlexNet network

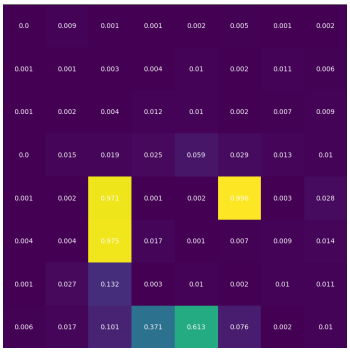
This CNN then receives game boards as input (details in section 2.2.1) and returns a matrix giving probability of finding a mine at each position (1 if mine, 0 otherwise). Then, for the positions in the surroundings of the revealed tiles, an algorithm is in charge, on the one hand, of finding the minimal probability (violet tiles on 2b), i.e. the position of the next move (blue tile on 2a), and on the other hand, of finding the maximal probability (yellow tiles 2b) and of placing a flag there, i.e. the most probable position of a mine at this moment of the game.

2.2.1 Training

There are several possible options for creating the set of inputs required for the training of the CNN. The method we have chosen is to generate many winning games and for each sequence of boards

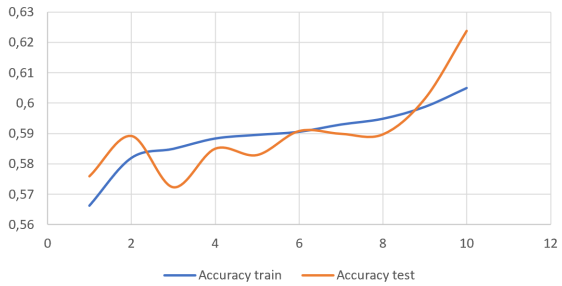


(a) Board game

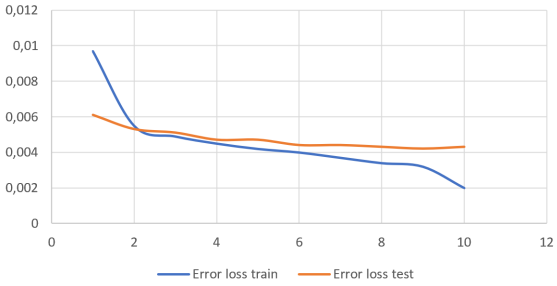


(b) Board game probabilities

of these games, we have kept only one board every 15 moves, this allows us to obtain varied game situations, namely beginning of games, middle of games and end of games. Initially, we only generated a set of beginning of game, which prevented the model to progress further in the game. These boards, or matrices, are constituted of the values of the revealed tiles (going from 0 to 8) and of the unknown tiles for which we have fixed the value to -1. Moreover, during the training, for each of these generated boards, the corresponding solutions are also provided to the CNN, which take the form of matrices in which the positions of the mines in the perimeters of the revealed tiles are marked by a 1 and otherwise a 0.
Thus, we managed to reach an accuracy rate of 62.35% and an error loss of 0.0424%, to do this we trained our model on 5,000,000 boards and this for 10 epochs, figure 3a shows the evolution of the model accuracy as a function of epochs and figure 3b shows the evolution of the error loss as a function of epochs.



(a) Accuracy evolution per epochs



(b) Error loss evolution per epochs

2.2.2 Testing

Then, in order to evaluate the real performance of our model, we tested it on a set of 1,000 games that it has never been confronted with, in other words a validation set. Under these conditions, the model reaches a win rate of 73.6%. Figure 8 shows the evolution of the percentage of games won by our model as a function of the number of boards on which it trained during 10 epochs. We notice that this value seems to follow a logarithmic function.

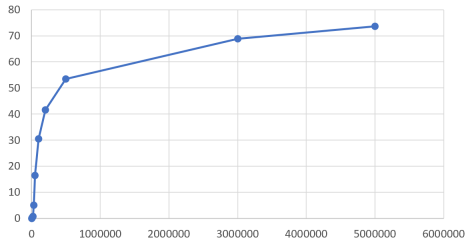


Figure 4: Win rate evolution (%) per number of boards

2.3 Reinforcement learning (RL)

For our reinforcement learning model, we use a modified version of Q-learning. After learning the Q values, the agent would use the values obtained to play minesweeper. The standard Q-learning algorithm is as follows:

$$Q(s_t, a_t) \leftarrow Q(s, a) + \alpha * (r_t + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

where s_t is the state at time t, a_t is the action at time t, α is the learning rate, r_t is the reward at time t and γ is the discount factor. This structure allows the agent to learn not just about the direct reward of a particular action, but whether a particular action is more likely to lead to reward in the long-term. While this is crucial for many games like chess, we are not as concerned with the endgame result in minesweeper. Instead, we are more interested in the immediate reward, whether a particular move will uncover a mine on a specific board configuration. For that, we simply remove the red part in equation 1 and we get:

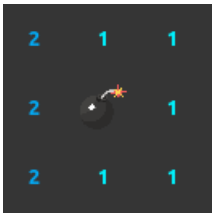
$$Q(s_t, a_t) \leftarrow Q(s, a) + \alpha * (r_t) \quad (2)$$

We exclusively used the 8x8 board with 10 mines for our training and testing.

2.3.1 Q-table

We will use a cluster of 3x3 around a tile to see if it gives enough information for the RL agent to make the right decision, examples of cluster can be seen in figure 5.

(a) 3x3 cluster with a normal tile



(b) 3x3 cluster with a mine

Figure 5: examples of 3x3 cluster

The different clusters will be the states of our Q-table and each time a new state is discovered, it will be added in the Q-table. 2 actions will be available for our agent: *click* or *ignore* the tile. A reward will be given depending on both the state and the action (cfr table 1).

States	*Click*	*Ignore*
Tile is **not** a mine	+1	-1
Tile is a mine	-1	+1

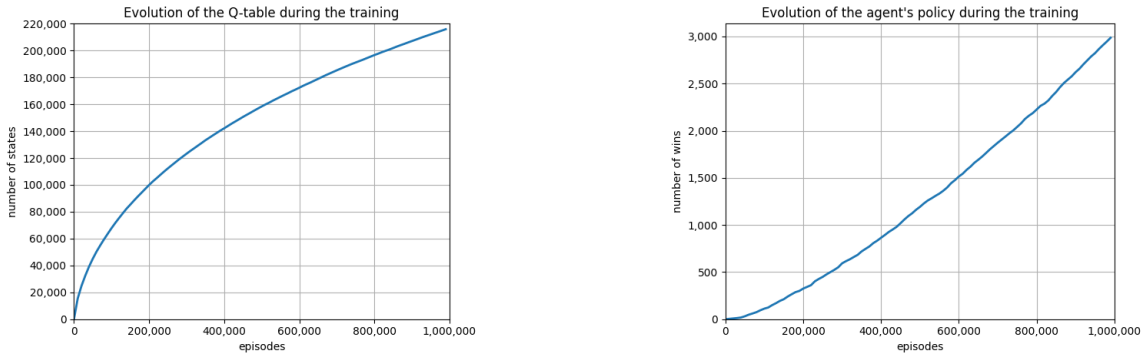
Table 1: Reward matrix

For every move, the agent looks at the Q-table to see if it has already recorded the state and select the action with the highest reward, called greedy action.

2.3.2 Training

We train our agent using the epsilon greedy algorithm to balance exploration and exploitation. We give our algorithm 1,000,000 episodes to converge and we evaluate our agent every 10,000 episodes. We trained the agent with $\alpha = 0.1, \varepsilon_{max} = 0.9, \varepsilon_{min} = 0.1$ and $\varepsilon_{decay} = 0.99$.

We can see that the numbers of states and wins keep increasing meaning the agent learns.
We now make a comparison on the number of states between an agent trained over 500,000 episodes,



(a) Evolution of number of states in the Q-table

(b) Evolution of number of wins of the agent

Figure 6: RL agent after training over 1,000,000 episodes

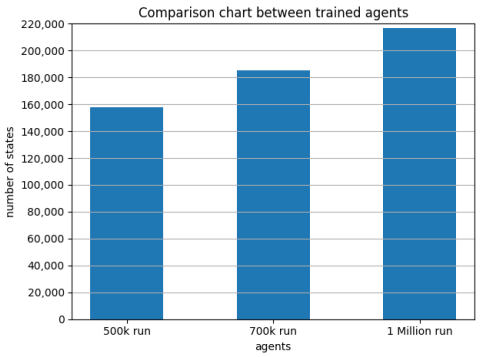


Figure 7: Comparison graph between an agent trained over 500,000 episodes, an agent trained over 700,000 episodes and the agent trained over 1,000,000 episodes

an agent trained over 700,000 episodes and the agent trained over 1,000,000 episodes in the figure 7.

We can see that augmenting the number of episodes increases the number of states discovered in our Q-table.

2.3.3 Testing

We now evaluate the trained agent on 1,000 games. Note that undiscovered states during the testing phase are ignored. We reach a win rate of 0.8%.

2.3.4 Further improvement

We can see that our reinforcement learning performs poorly to enhance it we could possibly increase the number of episodes for the training phase or increase the size of the cluster. Another idea, as it is just pattern matching, we could change to deep Q-learning and use a convolutional neural network instead of the Q-table as it is more fit for pattern matching.

3 Conclusions

Finally, as a conclusion, we notice that for the same size of board, i.e. 8x8, the algorithmic solver is the most performant with 75.2% of win rate, followed closely by the CNN with 73.6%. Then comes the model using reinforcement learning with 0.8% of win rate. It is important to note that the CNN and RL models can still be improved either by providing more training boards or by extending the duration of their training.

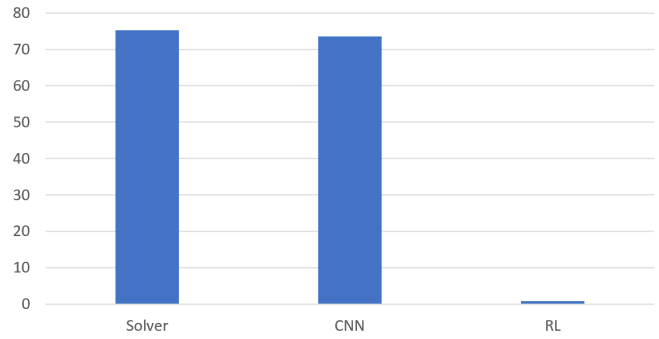


Figure 8: Comparison of the different models' win rates (%)