# Learning how to play Minesweeper using Double deep Q-learning

1st Adam Khalif
*Electrical Engineering Department*
*Chalmers University of Technology*
Gothenburg, Sweden
adamkh@student.chalmers.se

2nd Felix Gimbringer
*Electrical Engineering Department*
*Chalmers University of Technology*
Gothenburg, Sweden
felixgi@student.chalmers.se

***Abstract*—In this paper, a Double Deep Q-Network (QQDN) is used to solve the game of Minesweeper, modeled as an MPD. It utilizes two networks, one online and one offline, to create the optimal policy to use when deciding that actions to take. The game environment is simplified to only using a grid size of 3x3 and 1 bomb placed on the board, either in a fixed or random position. The agent is forced to never pick a cell that contains a bomb on the first action. The results show that an agent playing on a board with a fixed bomb placement will reach a win rate of 100% and when playing on a board with random bomb placement reach a win rate of just over 90%.**

***Index Terms*—Minesweeper, RL, DDQN, MPD**

## I. Introduction

Solving the game of Minesweeper by using Double deep Q-learning is a reinforcement learning (RL) method that is model-free, which means that the agent will learn to play the game in an optimal way by using the experience of playing the game itself. This is accomplished by training and using two separate neural networks to decide the best move in every state of the game.

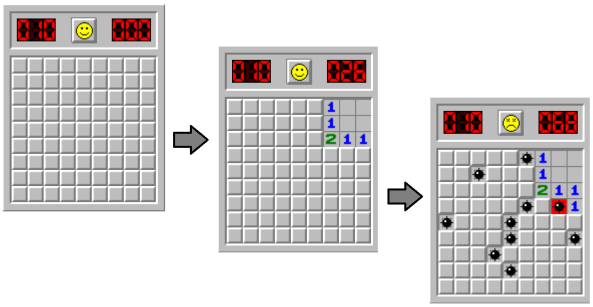

Fig. 1. A game of Minesweeper where the player loses

Minesweeper is a single player computer game that originates from the 1960s. The objective of the game is to clear a board with hidden mines. The mines are placed at random locations at the start of the game. When the player selects a cell on the board that does not contain a mine it will show a number which gives information about how many mines there are in the neighbouring cells. On the other hand, the player loses the game if a cell with a bomb behind it is chosen. The game is won if all cells are cleared except those with bombs behind them.

## II. Background theory and related work

The game of Minesweeper can be modeled as an Markov decision process (MDP). This is deeply investigated in Minesweeper, Preslav Nakov and Zile Wei (2003) [1]. The authors uses reinforcement learning methods and proves that an exact optimal policy can be found. This differs from the scope of this paper where the optimal policy is approximated an not found exactly. But it is helpful to know that the problem can be solved with reinforcement learning methods and that it can be regarded as an MDP.

The definition of an MDP is a 4-tuple: $(S, A, P, R)$.

- S - the set of states. The number of states are defined by the number of cells on the board. Every state can take 10 different discrete values. Either a number between 0 and 8, but it can also be unknown or a bomb.
- A - the possible actions. The number of actions are also determined by the size of the board. At the start of the game can any action be taken. But a cell can only be opened once and this means that the action space will shrink during the game.
- P - the transition probabilities. Given a state and an action a, this is the probability to end up in another state $s'$.
- R - the reward. This is given when taking an action $a$ that leads to a transition to a state $s$.

An usually important parameter when solving an MDP using Q-learing is the discount factor $\gamma$. But this variable does not affect the result very much when solving the Minesweeper problem. This is being explored in Evolution strategies and reinforcement learning for a minesweeper agent, by Hansen, Havtorn, Johnsen, and Kristensen (2017) [2]. The authors solves the problem using Deep Q-learning with different values for $\gamma$ and finds out that a near optimal policy can be found regardless of this variable.

## III. Method

The fundamental part if the Q-learning algorithm is the Bellman equation that calculates the Q-values. The Q-values are the values for all possible action in every state. Given that the Q-values are known, the optimal policy is the one where the agent picks the action with the highest Q-value. The idea behind Deep Q-learning (DQN) is to use neural networks in order to approximate these values. However, research have shown that conventional DQN sometimes tends to overestimate them. The approach used in this paper, Double deep Q-learning (DDQN), have been able to outperform regular DQN [3]

### A. Double deep Q-learning

A Double Deep Q-Network (**DDQN**) consists of two deep neural networks. The online one with learnable parameters $\theta$ and the offline one or "target-network" with parameters $\bar{\theta}$. The online network is optimized to minimize the loss function

$$L(\theta) = (Y - Q(\theta))^2$$

The Q-target, $Y$ is computed as

$$Y = R + \gamma Q\Big(s', \underset{a}{argmax} Q\big(s', a; \theta\big); \bar{\theta}\Big)$$

This can be interpreted as using the offline network to pick the best action, while the target network approximates the Q-values. This is where the method differ from conventional DQN. In that case is only the offline network used when calculating $Y$.

The actions that the agent will take are chosen by using an $\epsilon$-greedy approach. This means that not always the best possible action is chosen. Sometimes the agent will instead pick a random action. How often this happens depends on how large $\epsilon$ is. The training starts with a large $\epsilon$ which will make the agent explore a lot of possible combinations of actions and states. When reducing $\epsilon$ over time, the agent will be more likely to pick a good action.

### B. Implementation

The Minesweeper game is implemented as an MDP by creating an OpenAI gym environment. The state space is defined as the board of the game with all the information that is available for the agent. When the agent takes an action, e.i. chooses a cell, he will be transferred to a new state with new information available. If the agent picks a cell with a bomb the game is over which means that a terminal state has been reached. The agent can not pick a cell that has already been opened. This means that the possible actions to take will decrease during the game. The game is won if all cells that does not contain a bomb has been opened. This is also a terminal state. Rewards are given to the agent after every action. $+1$ for winning, $-1$ for loosing and $+0.3$ for picking a cell without a bomb.

The deep neural networks in the DDQN consists of three fully connected layers as illustrated in Figure 2. The number of neurons in the input layer is the number cells on the board and in the output layer the number of actions. The network training is optimized using the Adam optimization method.

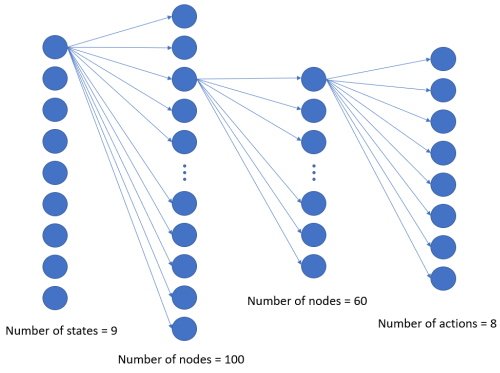


Fig. 2. Network architecture in the DDQN

## IV. Results

In the simple environment with a $3x3$ board and $1$ bomb and forcing the agent never to loose on its first action, the win rate when playing with a random policy can be calculated as the probability that the agent opens a cell that contains a bomb on it's first action. This is the same probability as of opening all cells except the one that contains the bomb:

$$w_r = \frac{7}{8} * \frac{6}{7} * \frac{5}{6} * \frac{4}{5} * \frac{3}{4} * \frac{2}{3} * \frac{1}{2} = \frac{1}{8} = 12.5\%$$

$12.5\%$ is the expected win rate when playing with a random policy. This will be the base line for other agents to compare themselves against. An agent playing with a random policy means that $\epsilon = 1$ and is not decaying over time.

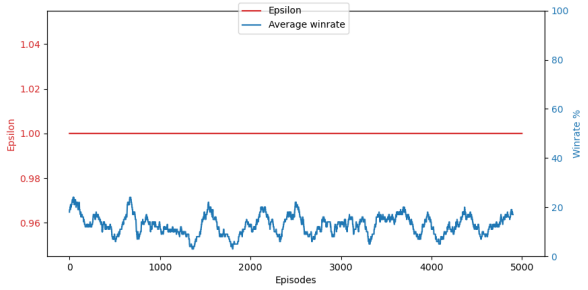


Fig. 3. Random agent playing

As one can see in Figure 3 the agent does not improve (as expected) over time and the average win rate is around $12\%$.

The first agent to be tested against the base line is an agent that plays the game on a board with a fixed bomb placement. This agent is expected to quickly learn the optimal policy when decaying $\epsilon$ over time. When $\epsilon = 0$ the agent is expected to have achieved a win rate of $100\%$. In Figure 4 the win rate reaches $100\%$ and thus is able to find the optimal policy.

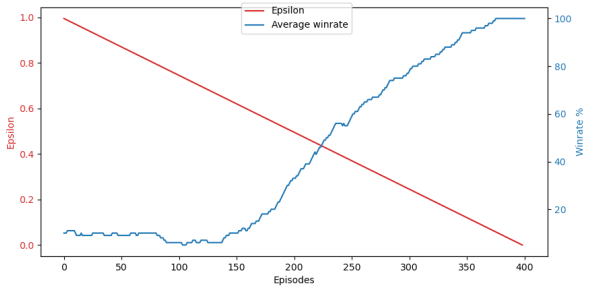


Fig. 4. Agent playing on a fixed board

A more realistic environment is of course a random bomb placement, as in the real game of Minesweeper. This means that the agent will have to learn a policy that takes into account the values of the open cells instead of just remembering what cell contains the bomb as in the case when the agent was playing on a board with a fixed bomb placement. This is much more difficult and therefor the agent will have to train over many more episodes.

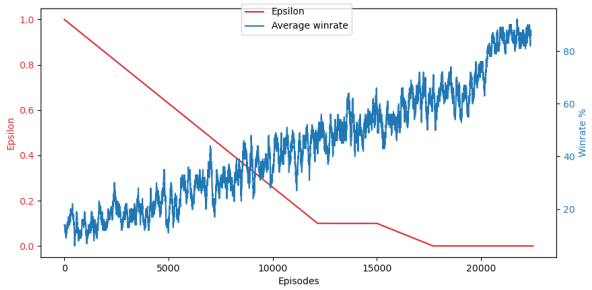


Fig. 5. Random bomb placement

As can be seen in Figure 5 the agent trains over around $25000$ episodes and learns rather slow compared to the agent in Figure 4. Here the value of $\epsilon$ is also decayed in two stages. First from $1$ to $0.1$ during $12000$ episodes and then evaluated with $\epsilon = 0.1$ for $3000$ episodes. Then $\epsilon$ is set to decay again, but now from $0.1$ to $0$ over $5000$ episodes, i.e. the decay is slower than during the first $15000$ episodes. When $\epsilon = 0$ the policy is evaluated over $10000$ episodes. during this period the win rate increases more rapidly and the policy found is close to being optimal. The final win rate after $25000$ episodes is close to $90\%$.

## V. Conclusions

As can be seen in Figure 5, the agent preforms on the hardest environment (random bomb placement) with a win rate just over $90\%$. This is considered to be sufficient enough to confidently say that the agent was able to find a policy that was close to being optimal. Further more, it is hard to say whether or not a fully optimal policy in this environment (win rate of $100\%$) could have been found since the Q-values are being approximated in the DDQN. The training loop was run many times and the win rate never converged to anything higher than the result presented in Figure 5.

Although the scope of this paper was a small board size (3x3) and only one bomb being placed in a cell it was still a rather hard problem to solve. It took many episodes until the network converged to a high enough win rate to say that the policy found was sufficiently good. To be able to solve even more complex environments, i.e. board sizes larger than 3x3 or more bombs placed in cells, many more training episodes would have had to been run. This can be investigated in future work. Also one could experiment with the usage of transfer learning. This would mean that some layers of the network could be kept frozen when training the model to speed up training when training on larger board sizes with more bombs.

## References

[1] Preslav Nakov and Zile Wei. Minesweeper. 04 2003.
[2] Jacob Hansen, Jacob Havtorn, Mathias Johnsen, and Andreas Kristensen. Evolution strategies and reinforcement learning for a minesweeper agent. 09 2017.
[3] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. 2015.