GDAŃSK UNIVERSITY OF TECHNOLOGY

TASK Quarterly 27 (4) 2023

Neural Networks in solving Minesweeper

K. Lubarski[a], B. Samujło, K. Wszeborowski

Department of Computer Communications, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology
Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland

https://doi.org/10.34808/FTWJ-Q764

Abstract

The purpose of this article is to present the operation of certain neural networks in solving the Minesweeper game and to assess whether it is possible to represent the decisions made by these neural networks in an understandable way using logical rules. Existing solutions such as CSP (Constraint Satisfaction Problem) were utilized to design an algorithm that analytically solves the Minesweeper game. The results obtained were then used to train Multi-Layer Perceptron (MLP), Encoding Neural Network (ENN), and Convolutional Neural Network (CNN) models. The CNN emerged as the best-performing network. Based on the tests conducted by this network, a decision tree was constructed that represents the network's logic for these specific tests with approximately 90% accuracy. Ultimately, none of the tested neural networks were able to match the analytical approach. However, based on the decision trees obtained for the functioning networks (mainly CNN), it was inferred that, in theory, with a sufficiently large number of tests, it should be possible to closely replicate the network's operation using logical rules (nested conditional statements)

Keywords:

artificial neural networks, minesweeper, decision tree

[a]E-mail: s188569@student.pg.edu.pl

1. Introduction

Minesweeper is a simple, single-player logic game that involves uncovering an entire board containing hidden mines. The game is played in turns. In each turn, a tile is selected to uncover. If the chosen tile does not contain a mine, a portion of the board is revealed; however, if a tile with a mine is selected, the game is lost automatically. The board consists of tiles that are gradually uncovered. The numbers on these tiles indicate how many mines are adjacent to that specific tile. The game itself has several difficulty levels, determined by the size of the board and the number of mines it contains[1].

1.1. Motivations

The main goal was to develop an analytical/logical method that, through appropriately arranged systems of equations, would determine the rules for selecting the safest move for a given situation on the board or identifying where the mines are located. The next step involved using the results from this approach to train an artificial neural network and achieve the best possible outcomes. The ultimate objective was to define the rules of gameplay based on the outputs provided by the neural network for specific board states.

1.2. Problems

The main challenges that might be encountered in achieving the objectives were:

1. The selection of an appropriate analytical method and the use of its results as a training dataset for the artificial neural network.
2. The selection of the most suitable type of neural network for playing Minesweeper.
3. The interpretation of the network's output results and their conversion into gameplay rules.

2. State of the art

2.1. Analytical methods

1. **Single-point strategy**

   This is a popular approach that appears in many studies related to Minesweeper. In brief, this method involves identifying one cell adjacent to a number as flagged (containing a mine), usually based on randomness or an approximate choice. Then, any adjacent number with not uncovered tiles next to it is selected, and a simple formula is applied[2]:

$$Mine = Number - AdjacentFlags \tag{1}$$

   If the result of this operation is zero, the covered tiles around the selected number are considered safe. Conversely, if the number equals the count of adjacent flags, the remaining tiles are identified as mines[3]. In other cases, other tiles are selected as flagged (Figure 1). There are extensions to this strategy, such as remembering certain tiles as less safe or making better flagging choices. However, the main issue remains the element of randomness[3].

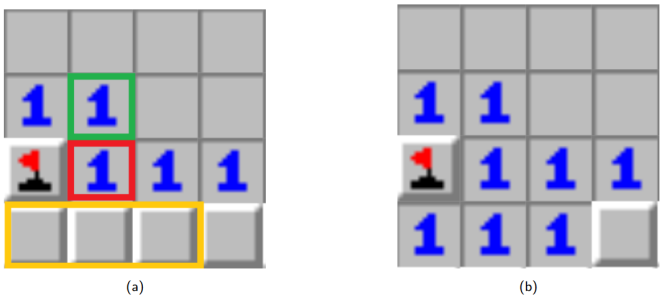

Figure 1: This figure presents the principle of operation of the single-point method.

2. **Constraint Satisfaction Problem (CSP))**

   This method is relatively simple yet effective in its operation. It involves marking all unknown tiles adjacent to a number and then setting up a system of equations. The unknown tiles around a given number are equated to its value[4][5][6][3]. The system is then solved using any method, such as Gaussian elimination. As a result, it is determined whether a given tile contains a mine or not, or if an unequivocal result is unavailable. This method has zero randomness; however, it is not 100% effective, as there are times when a definitive solution cannot be obtained[4][6].

Figure 2: A fragment of a Minesweeper game board.

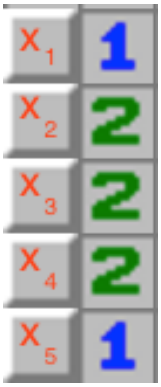

Figure 3: A fragment of a Minesweeper game board with highlighted auxiliary values.

Figure 2 shows a small fragment of the game board, presented in a column format for a clearer illustration of the CSP algorithm's method. Next, in the Figure 3 the same board as in Figure 2 is shown, but with the unknown tiles marked with x's — our unknowns.

$$x_1 + x_2 = 1 \tag{2}$$
$$x_1 + x_2 + x_3 = 2 \tag{3}$$
$$x_2 + x_3 + x_4 = 2 \tag{4}$$
$$x_3 + x_4 + x_5 = 2 \tag{5}$$
$$x_4 + x_5 = 1 \tag{6}$$

Following this, the system of equations presented above is constructed based on the contents of the board and our x's. This system must be solved using any method, with the constraint that the values in the system of equations must be integers. After obtaining the resulting system, the status of the previously unknown tiles can be assessed. If a given x is zero, the tile is safe; if the unknown equals 1, the tile contains a mine. Any other values usually indicate that the content of the tile cannot be determined with 100% certainty. The final state of the board is then obtained as shown in Figure 4.

Figure 4: The resulting game board.

2.2. Heuristic methods

1. **Supervised Learning**

   This approach is the first to require the use of datasets generated based on analytical methods to achieve measurable results. In this solution, a given board state is the input state, and the action is to uncover a specific tile[7]. Based on what the given action has provided (information derived from the analytical method), i.e. uncovering numerous new tiles, hitting a mine, or uncovering a single tile, the result is evaluated accordingly[7]. The model needs to be trained on a large amount of data

2. **Q-learning**

   This approach is similar to the previous method but with some differences. Notably, the model independently makes decisions during a specified number of games. The results are then evaluated accordingly. Following this, an attempt is made to solve Minesweeper using the model. Decisions are then made based on the knowledge acquired during training[7]. Unfortunately, the number of possible combinations increases as the board size grows, which requires very lengthy training of the model to improve its performance[8].

   A potential issue with both approaches mentioned above could be the number of game states the model needs to remember and the way they are evaluated (we want to do this as accurately as possibly). However, we can make certain assumptions and evaluate some boards and configurations as less important or irrelevant[1], which ultimately makes it easier to assess effectively. This could help improve the results[1].

3. **Filter-based method**

   This approach involves selecting a specific tile and assessing the probability of it containing a mine based on a constructed sample of filters. Such a filter assumes a sample neighborhood and evaluates each tile based on how safe its selection is. A set of filters (Figure 5) is then applied to the given portion of the board in a specific and predetermined order (which is not without significance)[9]. Ultimately, a rating for each tile is obtained. After evaluating each tile in this manner, a decision is made about which tiles to uncover.

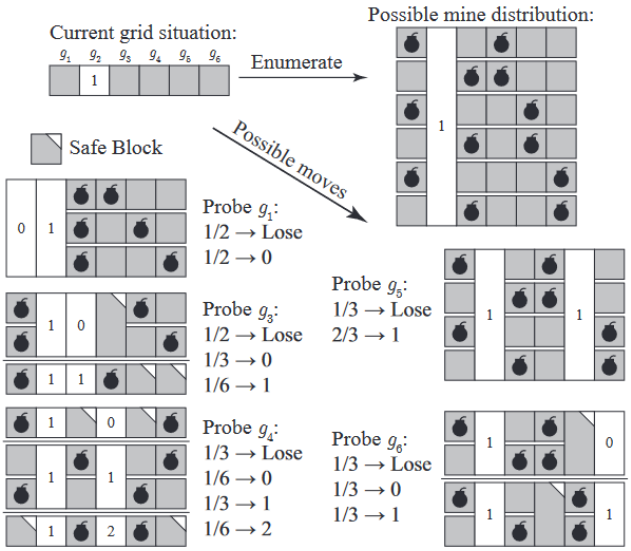

Figure 5: This figure presents example filters and their principle of operation.

2.3. Neural Networks

1. **Feedforward Neural Networks (MLP)**

   This is the first and simplest type of neural network in the group. Such a network consists of multiple interconnected layers, with two layers being visible: the input and output layers (which can be of any size). Between these, there are so-called hidden layers that process data in their own way. To achieve effectiveness, the network needs to be trained on sufficiently large datasets and through a sufficient number of iterations. The goal of this network is to produce results as close as possible to the actual output based on the given input[2].

2. **Convolutional Neural Networks (CNN)**

   This network is similar to the heuristic filter-based approach. Here, a set of filters of a defined size and parameters is also defined, which are then applied to the input data in a specified order. The difference is that the filter is purely mathematical (figure 6)[2]. This network is similar to the heuristic filter-based approach. Here, a set of filters of a defined size and parameters is also defined, which are then applied to the input data in a specified order. The difference is that the filter is purely mathematical (Figure 6)[2].

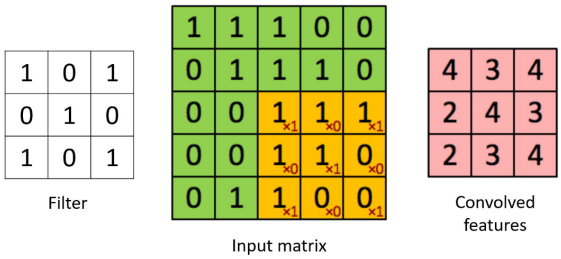

Figure 6: This figure presents the principle of operation of filters in a Convolutional Neural Network (CNN).

One noteworthy aspect is the approach to evaluating the risk of tiles on the Minesweeper board when using neural networks. The first approach involves providing the entire board as input and obtaining the risk assessment for each covered tile as output. The second method involves defining a "sliding window." The risk is evaluated only within this window (applied to a portion of the board), which is then moved across different parts of the board. The final approach is a slight variation of the aforementioned "sliding window" method. It focuses on determining the risk for only one central tile, with tiles outside this central region being considered unknown (Figure 7)[2].

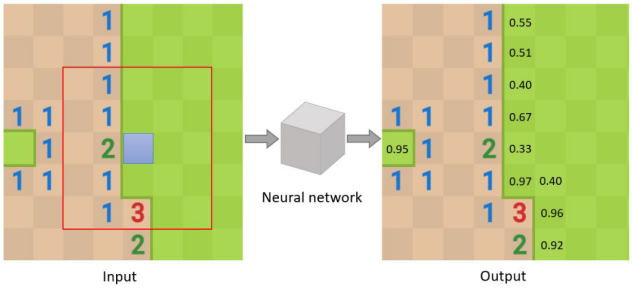

Figure 7: This figure presents the principle of operation of the "sliding window".

It is worth noting that there are other neural network models as well as alternative heuristic approaches to solving the Minesweeper game. However, these do not enjoy widespread popularity and are not extensively discussed in the available scientific literature.

3. Our Implementation

At the outset, it is worth mentioning that Python was chosen as the programming language due to its compatibility and the availability of built-in solutions. All the code written for the project can be found in the repository on GitHub.

3.1. Analytical

1. **CSP**

   In our implementation, we decided to use the CSP approach described in section 2. We chose this analytical approach due to its simplicity of implementation and the effectiveness of the method. To solve the resulting systems of equations, we used a modified Gaussian algorithm (with values required to be integers and within a finite range for correct interpretation of the results). If a safe move could not be determined, the algorithm would choose one of the available uncertain tiles.

2. **Improved Approach**

   When the algorithm could not make a decision that would not result in a loss and needed to make a choice to continue the game, the basic approach relied on randomly selecting an unknown tile. We decided to seek a solution that would minimize the chance of hitting a mine. To assess the risk of unresolved tiles in the basic method, we constructed a formula that takes into account the revealed tiles and previously found mines.

   This algorithm, along with the CSP method, helped us in creating datasets for training neural networks. The algorithm that determines the probability of a specific tile being a mine calculates it by analyzing the neighboring tiles. For each surrounding digit, the partial probability assigned to it is computed by dividing the value of that digit by the number of covered neighboring tiles. The final result is obtained by summing the partial probabilities, which is then subjected to normalization.

Table 1: CSP Table

Game/Results	Wins	Loses	No decision
7x7 7 mines	12,7%	0%	87,3%
10x10 10 mines	33,9%	0%	66,1%
20x20 20 mines	92,2%	0%	7,8%

Table 2: Enhanced CSP Table

Game/Results	Wins	Loses	No decision
7x7 7 mines	73,2%	26,8%	0%
10x10 10 mines	89,1%	10,9%	0%
20x20 20 mines	99,9%	0,1%	0%

The differences in results between the two approaches arise from the standard CSP method's inability to determine the safety of every tile on the board. This issue occurs when the matrix equation solution does not specify a single safe tile (only identifying mines or failing to find any solution). The enhanced approach, which includes additional risk calculations, addresses this problem by estimating approximate probabilities for tiles whose safety cannot be determined by the standard CSP method. Unfortunately, this introduces the potential for algorithm failure; however, the rate of such failures is significantly lower compared to the number of prematurely ended games with the standard CSP approach.

3.2. Neural Network

1. **Multi-Layer Perceptron (MLP) Network**

   In our case, we decided to use a pre-built solution available in the PyTorch library. Our network consists of 2 mandatory layers — input (game board / board segment) and output (risk of individual cells / specific tile) — as well as a predefined number of hidden layers that process the data. We chose to use 4 hidden layers containing between 10 and 100 neurons each. The activation function selected was RELU. The output is passed through a sigmoid function to ensure that the result is within the range of 0.0 to 1.0.

2. **Convolutional Neural Network (CNN)**

   In this case, we also utilized an architecture available in the PyTorch library. Our network consists of 3 filters with a base size of 5x5 and padding set to 2. The number of neurons in the layers ranges from 10 to 64. Each layer is activated using the RELU function, and, similar to the previous solution, the final result is passed through a sigmoid function. We also apply one-hot encoding to the game boards. This involves encoding certain key board parameters, which cannot be numerically defined by default or are computationally demanding, as numbers to facilitate the network's evaluation of the game states.

3. **Encoding Neural Network (ENN)**

   This type of network is an extended approach compared to the MLP networks. The model consists of layers with typically varying numbers of neurons, arranged in a shape reminiscent of a sideways hourglass. In this arrangement, the number of neurons gradually decreases from the input layer

to the central layer, and then increases again after passing the midpoint. This shape is intended to compress data in the first part and then decompress it in the second part[10]. As a result, this type of network can achieve better performance compared to the feed-forward approach. Similarly to the two previously described solutions, this network also uses the RELU activation function and a sigmoid function at the output.

4. **Network Parameters**
   Here is the translation and a possible format for presenting the detailed parameters of the neural networks used in the implementation (Tables 3, 4):

Table 3: Parameters for different Neural Networks.

Par/NN	MLP	CNN	ENN
Epochs	300	300	300
Criterion	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam
Learning rate	0.00005	0.00005	0.00005
Weight decay	0.000025	0.000025	0.000025
Step size	100	100	100
Gamma	0.9	0.9	0.9
Input layer size	board size	board size	board size
Output layer size	board size	board size	board size
Hidden layers	50, 100, 50, 25	25 5×5, 25 5×5, 64 5×5	0.8*input, 0.6*input, 0.4*input, 0.2*input, 0.4*input, 0.6*input, 0.8*input

Table 4: Parameters for different Neural Networks using "sliding window" method.

Par/NN	MLP 5x5	CNN 5x5
Epochs	300	300
Criterion	MSE	MSE
Optimizer	Adam	Adam
Learning rate	0.00005	0.00005
Weight decay	0.000025	0.000025
Step size	100	100
Gamma	0.9	0.9
Input layer size	25	25
Output layer size	1	1
Hidden layers sizes	50, 100, 50, 25	25 5×5, 25 5×5, 64 5×5

3.3. Decision Tree

Since Minesweeper is a game that can be solved using specific logical rules, it was decided that it would be worthwhile to attempt to create decision trees from the solutions. To construct the tree, the results obtained from the methods described earlier, which solve the board based on the "sliding window" approach, were used. An accurate model was ultimately created that correctly assessed the specific properties of the board and, based on those properties, proposed a specific path along with the associated risk of hitting a mine. This approach enables the description of Minesweeper rules and the operation of a specific type of network in a manner that is understandable to humans.

Figure 8 shows a fragment of the decision tree. Each node consists of several key elements. The most important of these are the coordinates of the cell in the "window" and the value of the number beneath it, used for classifying the central cell.

The feature importance for each node can be derived from how much the feature (cell) contributes to reducing impurity in the tree. Higher feature importance indicates that the corresponding cell plays a more significant role in determining the classification of the central cell, influencing the decision-making process at that particular node, as presented in Figure 9.

The next elements are the error rate, the number of samples defining that part of the tree, and the distribution of samples among the predefined classes. The class assigned to the central cell is derived from the class with the highest number of samples in that node. By traversing the tree from the root towards the leaves, one answers the given condition and moves to the appropriate nodes. Ultimately, the result is the class of the central cell.

In this way, the obtained decision tree can be represented as an algorithm consisting of a sequence of nested conditional instructions. For example, the decision tree shown in Figure 8, trained with 1000 games and results returned by the neural network (CNN), can be interpreted as follows: the first node (root) checks whether the cell at position $x = 2$ and $y = 0$ (where x is the horizontal axis and y is the vertical axis in the "sliding window") is less than 0.5. The data received by the tree during training consists of numerical values representing the game boards.

Where '_' denotes a cell outside the board, numbers correspond to cells with digits, and '?' represents an undiscovered cell (Figure 10). In other words, the first node checks if the mentioned cell is outside the board (i.e., cell_2_0 = −1.0) or a digit equal to zero (i.e., cell_2_0 = 0). If so, it returns true; otherwise, it returns false. By traversing all branches of the tree, the logic representing the network's behavior for these specific tests is obtained, based on which a tree is created with a maximum depth

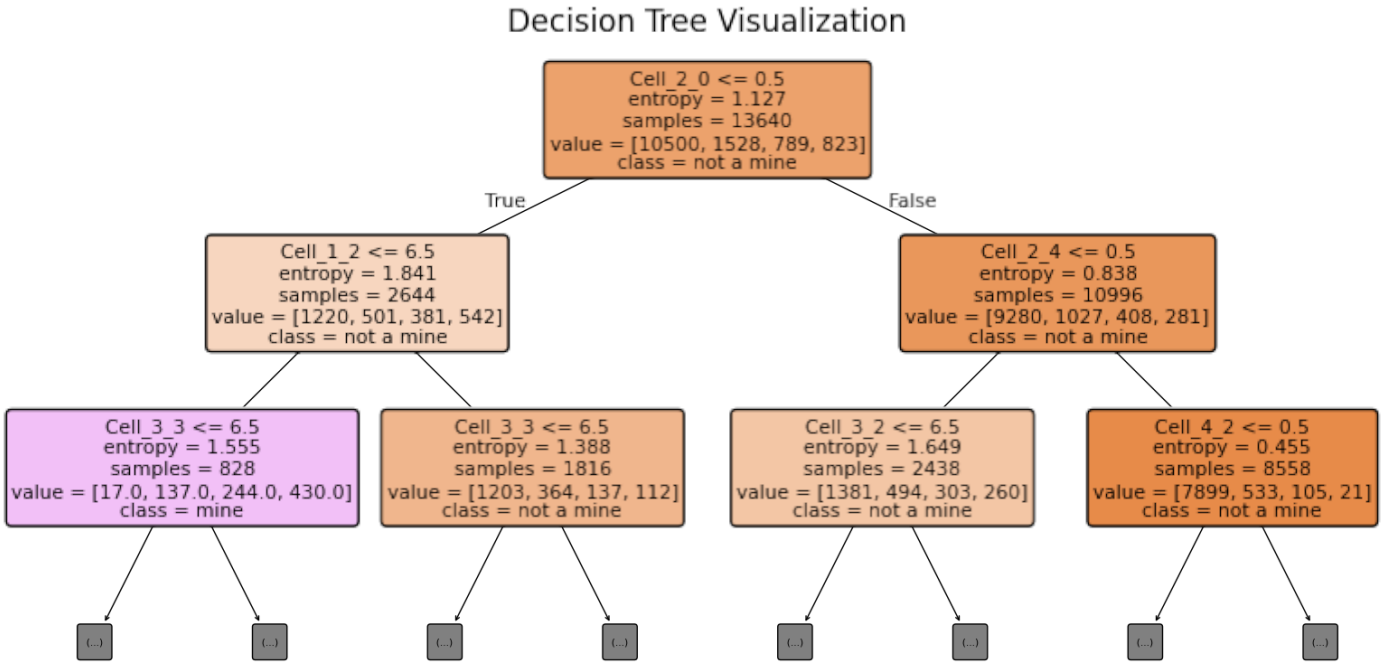

Figure 8: This figure shows a fragment of the decision tree with parameters for specific tests.

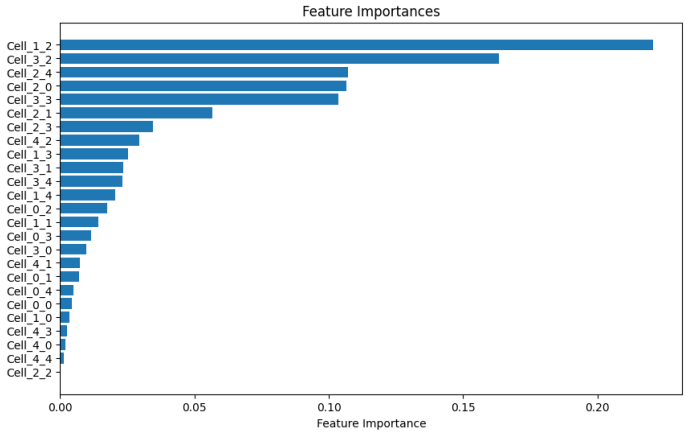

Figure 9: This figure shows the importance of each feature when making decisions for the previously generated tree (features here are the cells within the "sliding window").

```
mapping = {'_': -1.0, '0': 0.0, '1': 1.0, '2': 2.0, '3': 3.0,
'4': 4.0, '5': 5.0, '6': 6.0, '7': 7.0, '8': 8.0, '?': 9.0}
```

Figure 10: This figure illustrates the mapping of cells in the window/board to numerical data for the neural network/decision tree (the mapping is the same for both trees and networks).

of 8 (see Figure 8, which shows only a fragment of the tree).

It is worth noting that the presented tree and the feature importances for this tree represent only specific operations for the particular tests conducted on the neural network. This means they do not accurately depict the entire network. However, during the training of the trees, it was observed that with an increasing number of generated test data, the results became more similar — some nodes and general tree structures (tests performed, mainly near the root) had similar values even if there were still differences. Similarly, the importance values of the cells often showed that the closest cells had the highest importance, occasionally swapping places depending on the constructed tree, as shown in Figures 11 and 12.

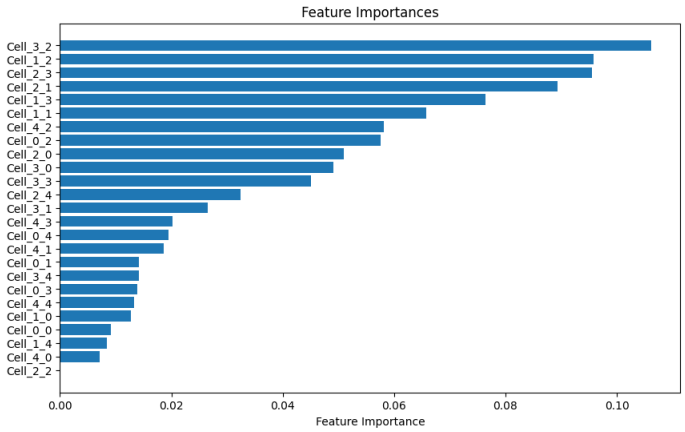

Figure 11: Feature importances for the first tree created from 1000 games using the CNN network, on a 10x10 board with 10 minesFeature importances for the second tree created from 1000 different games than those in Figure 12, using the CNN network, on a 10x10 board with 10 mines.

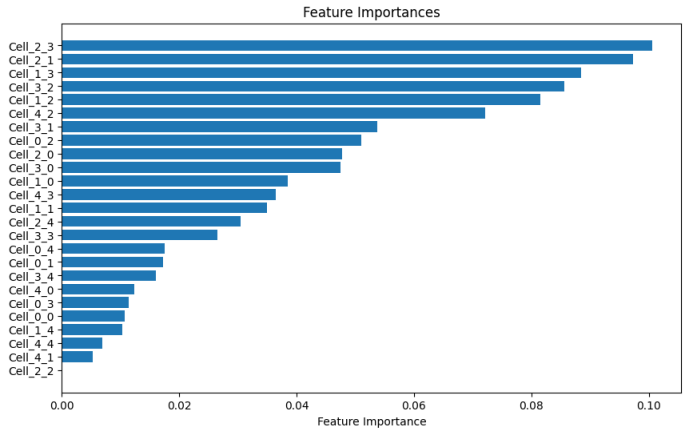

Figure 12: Feature importances for the second tree created from 1000 different games than those in Figure 11, using the CNN network, on a 10x10 board with 10 mines.

Even though it is not a true and faithful representation of the entire neural network's operation, it suggests that a decision tree, given sufficiently large test data, should theoretically approach a faithful and human-understandable representation of the entire neural network's logic (though it will never achieve complete accuracy).

4. Results

Neural networks were trained on data obtained from analytical solutions along with described risk prediction enhancements. The training dataset consisted of 1,000 games for most solutions. Due to hardware and time constraints, we decided to train the convolutional network model on just 50 games — using a significantly smaller dataset allowed us to further develop other parts of the project, and the results presented by this model did not deviate significantly from the other solutions. The presented results were obtained after conducting 1,000 games. The board was randomly generated eachtime based on a pre-determined seed to ensure that the results obtained by each method were as comparable as possible. For one of the three approaches, we used a holistic board-solving method. In the other cases, we also employed a method using a "sliding window," which in our case was 5x5 in size.

Results of individual approaches:

1. CSP (analytical solution, entire board)

   For a 7x7 board with 5 mines, we achieved 93.0% wins.

   For a 10x10 board with 10 mines, we achieved 89.1% wins.

   For a 20x20 board with 20 mines, we achieved 99.9% wins.

2. MLP Network (entire board)

   For a 7x7 board with 5 mines, we achieved 0.4% wins.

   For a 10x10 board with 10 mines, we achieved 0% wins.

   For a 20x20 board with 20 mines, we achieved 0% wins.

3. Convolutional Network (entire board)

   For a 7x7 board with 5 mines, we achieved 3.5% wins.

   For a 10x10 board with 10 mines, we achieved 0% wins.

   For a 20x20 board with 20 mines, we achieved 0% wins.

4. Encoding Network (entire board)

   For a 7x7 board with 5 mines, we achieved 0.4% wins.

   For a 10x10 board with 10 mines, we achieved 0% wins.

   For a 20x20 board with 20 mines, we achieved 0% wins.

It is worth noting that for the MLP and ENN networks using the full board method, as board sizes increased, they struggled with selecting subsequent moves. They frequently chose already uncovered cells as the next move, resulting in no change to the board (the move was ignored), and thus, the same cell was selected repeatedly. This issue appears to often stem from the neural networks not recognizing a new state as a distinct board — a problem inherent to MLP and ENN networks. Consequently, for simplicity, these situations were considered losses in the results presented below (Tables 5, 7, 9).

Another key issue is that the MLP network in the 5x5 variant mostly returned results that did not reflect the actual situation on the board at all — only values of 0 were returned. The results obtained with this variant are not

representative of this method and are due to the randomness of board selection, since when the entire board returned equal values, the same predefined field was always uncovered (Tables 6, 8, 10).

To better illustrate the results, Tables are presented below showing all the results obtained. The first three rows present the percentage of wins achieved after a given number of moves — if the game did not end after a specified number of moves, it was recorded as a win.

Table 5: Results for standard method (without "sliding window") on a 7*x*7 board with 5 mines.

Game/NN	CSP	MLP	CNN	ENN
1 move	-	82,3%	71,0%	83,1%
3 moves	-	1,5%	38,7%	4,4%
5 moves	-	0,4%	16,5%	0,4%
Full game	93,0%	0,4%	3,5%	0,4%

Table 6: Results for "sliding window" method (5*x*5) on a 7*x*7 board with 5 mines.

Game/NN	MLP/5x5	CNN/5x5
1 move	84,5%	88,6%
3 moves	56,0%	63,8%
5 moves	31,3%	49,2%
Full game	8,3%	21,0%

Table 7: Results for standard method (without "sliding window") on a 10*x*10 board with 10 mines.

Game/NN	CSP	MLP	CNN	ENN
1 move	-	88,7%	67,4%	86,1%
3 moves	-	0,1%	37,2%	0,3%
5 moves	-	0%	18,1%	0%
Full game	89,1%	0%	0%	0%

Table 8: Results for "sliding window" method (5*x*5) on a 10*x*10 board with 10 mines.

Game/NN	MLP/5x5	CNN/5x5
1 move	89,7%	93,8%
3 moves	64,2%	78,7%
5 moves	42,9%	64,5%
Full game	1,3%	6,5%

5. Summary

After comparing all the obtained results, it can be conclusively stated that the analytical method proved to be the best for all the tested cases. It is also worth noting that the waiting time for the results of this method was

Table 9: Results for standard method (without "sliding window") on a 20*x*20 board with 20 mines.

Game/NN	CSP	MLP	CNN	ENN
1 move	-	94,8%	59,3%	95,6%
3 moves	-	0%	20,6%	0%
5 moves	-	0%	5,6%	0%
Full game	99,9%	0%	0%	0%

Table 10: Results for "sliding window" method (5*x*5) on a 20*x*20 board with 20 mines.

Game/NN	MLP/5x5	CNN/5x5
1 move	69,6%	96,4%
3 moves	36,5%	90,4%
5 moves	17,3%	81,7%
Full game	0,7%	26,1%

significantly lower than for artificial neural networks. The results of these methods are discussed below.

For both the feed-forward learning method and the data compression/decompression method, the results were relatively similar — they performed best on smaller boards. For larger boards, with just 3 moves, most of them were unable to continue the game effectively. The convolutional network performed better, as it never got stuck and always made a move. However, training this model required considerably more time and computational power to achieve acceptable results.

Another issue with artificial neural networks is the inability to describe their operation in an understandable way. To select optimal parameters, one must rely on trial and error based on how the model performs in a series of games. It is difficult to rely on existing scientific work as results are very sensitive to minor changes, so there may be a different optimal approach for each case[7]. To help understand the operation of neural networks, constructing decision trees based on the obtained results can be useful. In our case, the best approach was the aforementioned manual adjustment of parameters through a series of tests. A similar approach has been encountered in other scientific works.

We concluded that despite numerous attempts to train the best neural network model, we were unable to create one that could realistically compete with the much simpler analytical solution based on solving systems of equations. Nevertheless, it was shown that with a proper amount of data and an appropriate maximum depth value, it is possible to construct a decision tree that can at least to some extent represent how a neural network operates in a way that is comprehensible to a human user.

References

[1] Z. W. Preslav Nakov, "Minesweeper, #minesweeper," *Berkeley EECS*, 2003. doi: 10.1109/TKDE.2003.180.

[2] M. H. Sajjad, "Neural network learner for minesweeper," *Loughborough University*, 2022. doi: 10.1109/TKDE.2022.180.

[3] B. David, "Algorithmic approaches to playing," *Harvard University's DASH repository*, 2015. doi: 10.1109/TKDE.2015.180.

[4] R. Massaioli, "Solving minesweeper with matrices," *Programming by Robert Massaioli*, 2013. doi: 10.1109/TKDE.2013.180.

[5] B. Y. C. Ken Bayer, Josh Snyder, "An interactive constraint-based approach to minesweeper." American Association for Artificial Intelligence, 2006. doi: 10.1109/TKDE.2006.180.

[6] C. Studholme, "Minesweeper as a constraint satisfaction problem." American Association for Artificial Intelligence, 2000. doi: 10.1109/TKDE.2000.180.

[7] R. K. N. Yash Pratyush Sinha, Pranshu Malviya, "Algorithmic approaches to playing," *International Institute of Information Technology Bhubaneswa*, 2021. doi: 10.1109/TKDE.2021.180.

[8] B. Smulders, "Optimizing minesweeper and its hexagonal variant with deep reinforcement learning." `https://research.tue.nl/en/studentTheses/bcceee8c-dea7-4309-8c4d-4d03cee6a2d4`, 2023. doi: 10.1109/TKDE.2023.180.

[9] S. C. C. Z. Z. G. Jinzheng Tu, Tianhong Li, "Exploring efficient strategies for minesweeper," *The AAAI-17 Workshop on What's Next for AI in Games? WS-17-15*, 2017. doi: 10.1109/TKDE.2017.180.

[10] J. C. B. Jozef Fekiač, Ivan Zelinka, "A review of methods for encoding neural network topologies in evolutionary computation." `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=814e438666496db4126e23404b2baf707218d7f2`, 2011. doi: 10.1109/TKDE.2011.180.