Using Convolutional Neural Networks To Assist in Minesweeper

Moez Bajwa (101196537) and Steven Kong (101189675)  
COMP3106A  
Fall 2024

Statement of Contributions

1. Each member made a significant contribution to the assignment

2. Each member made an approximately equal contribution to the assignment

3. Both members worked together on all parts of the assignment. Steven did a bit more of work on the Minesweeper logic and display side and Moez did a bit more work on the model.

Introduction

Minesweeper is a computer game first released by Windows in 1990 where a player attempts to reveal every square on the grid except for the mines. If a player is to click on a mine, they automatically lose. As such, players can flag where mines are to mark them, and the goal is to flag each mine so that every other square can be revealed. To understand where a mine is, a square may have a number on it once the square is revealed. This number indicates the number of mines immediately surrounding the square. This leads to the game being logic-based with players being able to determine where mines are and where is safe depending on the state of the board. However, it is very common for the board to reach a state where there is no move that can have a guaranteed outcome resulting in the player having to make a guess. Our goal was to create a Minesweeper AI meant for assisting the player in making the move with the best probability to be correct.

AI has been used to assist and help in playing Minesweeper for a long time. Often, these agents use reinforcement learning which has its own set of advantages and disadvantages. While it can assess each move using reward, it would be difficult to train since the only time it would receive negative reward is if it is wrong in thinking a space is not a mine, resulting in losing the game. An expert level Minesweeper board, which is a grid of size 30 by 16, has an extremely large number of states, considering there’s 480 squares and each of these squares can be covered, uncovered, or flagged. This means that a reinforcement learning agent would need an enormous amount of data to be trained on this. The environment is partially observable, since the agent only has access to the information on the squares that are uncovered, making it more difficult to learn where mines are. Additionally, it is likely that at any given state, there are several “optimal” moves as several squares may be able to be solved from a given state, however it does not matter what order these actions are applied in. Because of these reasons outlined above, we decided to take another approach to this problem by using a convolutional neural network. We chose this as CNNs are good at processing grids as they are often used for image recognition, they would be better at identifying patterns in smaller subsections of the grid because of the convolution layers and would be easier to train. Because of these reasons, others have approached Minesweeper with convolutional neural networks before.

After researching convolutional neural networks in Minesweeper, we were able to find three examples of people using CNNs for expert level (30 by 16 board and 99 mines) Minesweeper. First, ryanbaldini on GitHub used a combination of CNN and reinforcement learning to have the model learn the rules of Minesweeper and automatically play it. This approach featured 5 3x3 convolutional layers with 64 filters each using ReLU activation and a 1x1 convolutional layer with 1 filter and sigmoid activation which resulted in winning games at a 42% rate over an unspecified number of games. Secondly, CodingLikeMad on YouTube also used a convolutional neural network where they used 3 5x5 convolutional layers and another layer with a 1x1 kernel. This resulted in a 16.5% win rate while the agent automatically played the game however they did not share any code or dive deeper into the neural network architecture. Lastly, Johnny The Data Scientist on YouTube also used convolutional neural networks to automatically play Minesweeper. Though they did not share any metrics regarding performance, their approach used 3 4x4 convolutional layers and a 1x1 convolutional layer. This CNN was used in pair with a rule-based system to determine the best move for the agent to make. In comparison to our approach, each of these examples of previous work differ in multiple way which will be covered in the methods section of this report.

For this project, the main objective was to create an agent that sufficiently assists a player such that they can make the optimal move when no moves can be logically deduced. To do this, several sub-objectives are made. First, it was an objective that a square is automatically marked as a mine or a safe square in the heatmap if it can be guaranteed that the square is a mine or a safe square, respectively. Secondly, another objective was that the heatmap provides a visualization of the probability that each square is a mine. This is done so that players can know what move has the highest probability if they are not able to further deduce that information logically. Achieving these two subobjectives would result in completion of the main objective of assisting a player to play Minesweeper.

Methods

As previously mentioned, our approach to this project was using a convolutional neural network to assist players in Minesweeper. Our neural network has 6 2d convolutional layers, all with a ReLU activation function. Each layer had a 3x3 kernel size as this seemed most logical for Minesweeper interpretation. This was tested with larger kernel sizes, but this did not result in significant results. In each layer, different numbers of filters is used so that we could attempt to extract as many patterns from the data as we can considering that the board is passed into the model in many different states. The smaller filters are to detect less complex patterns while the larger filter amounts try to detect more common Minesweeper patterns. It also had 2 batch normalization layers as it is something that we have used in prior work to speed up training but made little to no difference to model validation. We also used 2 dropout layers which help prevent overfitting as it was an issue earlier in the development of the model, hence the 30% dropout rate between blocks of the neural network. In Minesweeper, a lot of the data is repetitive which can lead to certain weights becoming overfitted. Considering our overfitting concerns, we also implemented early stopping when training the model to stop training if the binary cross-entropy loss doesn’t improve after 5 consecutive epochs. We used 100 epochs for training the model, however with early stopping, it usually stopped around the 20th epoch where it stopping prevents overfitting and saves time. We used the Adam optimizer with a learning rate of 0.01 and a batch size of 64. Initially, we had used a batch size of 128 and a learning rate of 0.001 however we noticed slow convergence. After changes to the learning rate and batch size, it sped up training. We changed the batch size and learning rate as a higher batch size requires a lower learning rate so it is common that when one is changed, so is the other. Our main metrics used for performance evaluation were accuracy, precision, and recall. Precision gives us an idea of how many mine predictions were correct of all the places that were believed to be mines and is represented by true positives divided by all positives. Secondly, recall (or sensitivity) just tells us the percentage of mine predictions that the model got correct. Lastly, accuracy is just the number of correct predictions divided by total predictions. These were found to be the best metrics as Minesweeper boards will naturally have more non-mines than mines, and since the output of the model is a binary matrix, it could guess half the time and get many of the non-mines correct. When analyzing the performance of our model, what really matters is where it believes mines are, so we picked these metrics since they focus on the positives that the model returns

Along with the convolutional neural network to provide probabilities of each square being a mine, a rule-based system was implemented to determine when a square is guaranteed to be a mine or safe. We chose to implement a rule-based system to inform users what moves were guaranteed to be safe as the heatmap only visualized probabilities that squares were safe based on the model. This rule-based system was implemented by checking each square when it was updated after a players move. The first rule implemented was to determine when squares are guaranteed to be safe. This was implemented by adding a square location to a Python set if the number on the square was equal to the number of squares flagged in its neighbours. If this were true, the rest of the neighbours of the square that were covered could be guaranteed to be safe as shown in the image below.

A close-up of a grid

Description automatically generated

1: The squares with an 'X' on them are guaranteed to be safe because of the square directly to the right with ‘2’ in it and 2 mines being flagged by the player in its perimeter. This is reflected in the heatmap showing the squares cleared

The second rule implemented was the opposite of the previous rule, where a square is guaranteed to be a mine. This was determined if the number on a square was equal to the number of its neighbours that are flagged plus the number of its neighbours are covered. This would guarantee that the rest of the covered squares in a squares neighbours are mines since all of the safe squares around it are already uncovered.

A close-up of a grid

Description automatically generated

2: The squares with an 'X' on them are guaranteed to be mines because of the square directly to the right with ‘4’ in it and 4 squares being covered and 0 flagged mines in the perimeter of that square. This is reflected in the heatmap showing the squares as being flagged

To train our model we generated our own data using our functions to create boards in several different states. After several different amounts of data, we found that it was best to create a dataset of 300,000 boards as this is when our model reached sufficient convergence.

Proposal

1. Title: Applying Neural Networks to Minesweeper for Mine Detection

Students: Moez Bajwa (ID: 101196537) and Steven Kong (ID: 101189675)

Course Code: COMP3106A

Semester: Fall 2024

1. Minesweeper is a popular game where to win, players need to reveal spaces on a grid while avoiding spaces with mines. To be successful, players need to apply various problem-solving techniques and strategies. Utilizing probabilistic decision making and reasoning is crucial to avoid game ending mines. Applying these sorts of methods to win the game becomes more challenging as you increment the difficulty. As the difficulty increases, so does the grid size and the number of mines on the board, making Minesweeper a challenging problem to tackle. Our solution would aim to solve minesweeper boards to a decent degree of consistency.
2. We aim to take a supervised learning approach to solve this problem. Our main objective is to create an agent which can win a game of minesweeper more than the average human player, which is approximately 20% of the time on an expert-level board (30x16 board size and 99 mines).
3. Our implementation of the minesweeper agent would be a supervised binary classification task, implementing a convolutional neural network (CNN) to find possible mine positions on a game board. CNNs are widely known and used commonly in image tasks due to their ability to capture and recognize spatial relationships, something crucial for finding patterns in matrix shaped inputs. In this context, providing our model the game grid at any state, it will be able to provide the probability of each space on the board being a mine or clear.   
     
   Apart from this, we will be implementing a minesweeper board generation module. This module will likely end up using a variation of DFS to explore nodes to uncover based off an initial starting node on the grid, clearing out nodes which have no adjacent mines. This is a smaller part of the project, part of data preprocessing/generation, however worth mentioning as DFS is something we did cover in earlier portions of the course.
4. The data required for this project is rather simple. We will need different boards of minesweeper in various states; partially covered, fully uncovered, etc. Each of these boards will be labeled with a grid of the same size, where spaces with 1’s are the mines on that grid, and spaces with 0 are the empty spaces. There is no dataset available for this sort of data, however minesweeper is a game which is rather easy to develop states for. We plan to create a large dataset in the range of 5000 to 20,000 different minesweeper boards. This will be done via development of a minesweeper board generation module which will create the dataset which the model will be trained on.
5. For analyzing our results, our metric that we will use is win percentage. Depending on the size of the board, an ideal win percentage varies. With larger boards, this win percentage will be lower as there are more mines and it is more likely to have to guess what move to make next, rather than being able to know if a square is or isn’t a mine. For the sake of analyzing the results, we will use what most Minesweeper websites consider to be the expert difficulty. This would be a 30-by-16 board with 99 mines. After researching what the expected win percentage of a random board of this size is, a player should be able to have a win percentage of around 20%. As such, we will consider it to be a success if our agent is able to win at least 20% of games over a large sample size.
6. As for novelty of our project, using AI to solve Minesweeper boards is not a novel idea in and of itself. However, most of these projects are done using reinforcement learning. Our approach will be due to using supervised learning, where we will utilize convolutional neural networks to solve the boards. The neural network will calculate the probability that each square is a mine. If a square is guaranteed to be a mine, it will have a probability of 1 and will be flagged and if a square is guaranteed to be safe, it will have a probability of 0 and the square will be revealed. If no uncovered square is guaranteed to be safe, the square that is least likely to be a mine will be revealed.
7. Weekly Schedule:

Week 1 (Nov 3-Nov 9)- Generating the boards

Week 2 (Nov 10-Nov 16)- implementing the logic of the game (clearing board and flagging mines)

Week 3 (Nov 17-Nov 23)- implementing neural network to solve the board

Week 4 (Nov 24-Nov 30)- finish implementing neural network to solve the board and testing the agent by running it on many randomly generated boards

Week 5 (Dec 1-Dec 6)- writing report

1. We are available to present at any time between 9am-5pm on December 6th or 1pm-2pm on December 4th.
2. Our project will not require any GPU resources through the School of Computer Science