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 safe 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 agent 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.

For peer-reviewed applications of AI in Minesweeper, *Training a Minesweeper Solver* (Gardea, Koontz, & Silva, 2015) looked into multiple approaches for models. In this article, they attempted to use Q-learning, the constraint satisfaction problem, local classification, and global probability regression. Q-learning, regression, and classification performed well on a 4x4 board with 3 mines, however significantly dropped when approaching larger boards. Q-learning had the best performance on a 4x4 board with a 70% win rate, however Q-learning, classification, and regression all had a win percentage of less than 10% on a 4x4 board and less than 5% on a 5x5 board. The constraint satisfaction problem however performed well on larger boards, with it performing around 80% on an 8x8 board and around 70% on a 32x32 board. Each of classification, regression, and Q-learning had significantly more training than the constraint satisfaction problem. For example, when moving from a 4x4 to 5x5 board for Q-learning, the state space increased by a factor of , meaning it would take much longer to train for a large board, and both regression and classification trained the model in exponential time. Despite taking less boards to train, the constraint satisfaction problem it takes significantly more time since all computation happens while the model is playing the game. To put this into perspective, other methods could play thousands of games of Minesweeper while the constraint satisfaction process played one, which is a significant disadvantage.

In *MINESWEEPER, #*MINESWEEPER (Nakov & Wei, 2003) they also researched the applicability of reinforcement learning in Minesweeper and found that the largest problem for solving Minesweeper boards with it is the large state space. They said we must have better state space compression if we want to apply reinforcement learning to larger boards. They compared the performance of an optimal and greedy policy where they found that when an open move was free, the optimal policy performed consistently better on boards of varying number of mines. However, it only performs slightly better when no open move is free and performs worse when there are more than 9 mines on a 4x4 board.

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 (ryanbaldini, 2020) 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 (CodingLikeMad, 2021) 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 (Scientist, 2022) 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 ways, which will be outlined in the Methods section.

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 sigmoid activation function. This was chosen since the model doesn’t really care what number is on the square, it just cares if a spot is a mine, covered, or if has mines surrounding it. For that reason, sigmoid was used as it doesn’t care much about how many mines are surrounding it, rather that there are mines surrounding it. We also tested ReLU activation functions between layers, but this resulted in little to no difference in performance. 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, where a darker square indicates being less likely to be a mine and a bright square being more likely to be a mine. 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’s 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 generated1: 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. Also, the probabilities of squares being a mine can be seen with bright squares being more likely and dark squares being less likely

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 A close-up of a grid

Description automatically generatedsquare’s neighbours are mines since all safe squares around it are already uncovered.

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 randomly generated our own data using our functions to create boards in several different states. These states were of different complexities, with some being barely solved and others being with many revealed squares and possible moves. This was done so that the board was trained on as many different scenarios. After trying several different amounts of data, we found that it was best to create a dataset of 100,000 boards as this is when our model reached sufficient convergence. Data was randomly generated with random portions being uncovered as to simulate random states within an actual minesweeper game.

**Results**

Over the course of the work, as previously discussed, a few different performance measures or validation strategies were implemented to assess how the model was performing. The CNN’s performance was judged by splitting the data into a 80/20 split, specifically 80% of the dataset was used for training and 20% was used for validation. Additional boards were generated for final evaluation. A positive mine detection was classified with a threshold of 0.5.

**Quantitative Results**

A graph with blue and orange lines

Description automatically generatedThe quantitative results of the CNN will be discussed by going in depth and looking the results of our performance measures. This will be the ideal way to see how the model performed as well as look into why it was performing that way. We will also be able to use this information when looking into future improvements to the model and seeing what could be done better form all aspects of the process.

Figure 5.1: Accuracy on Epochs

Looking at accuracy first, we can see that the train and validation set performed decently well, peaking at around 0.84. In the context of the task and data, this is rather expected, which is why accuracy was not the main performance measure we used. In the game of minesweeper, naturally, there are always more non-mines than mines. This paired with the fact that the model is designed to output a binary matrix which detects the minority (mine) it is always more likely that there will be more 0’s outputted than 1’s on the matrix. This is not optimal, as these 0’s are taken into consideration in accuracy, when comparing to the label matrix. Accuracy is computed as the following, where T = True, F = False, P = Positives, N = Negatives:

Essentially, this is the percentage of correct guesses (0’s and 1’s) over all possible spots on the grid. Since the model by nature should output more 0’s than 1’s we can see that this validation measure will always be skewed to be higher than it is. This can lead us to the next performance measure we looked at to evaluate the model.

A graph with a line and a line

Description automatically generated

Figure 5.1: Precision on Epochs

The second quantitative performance measure we looked at was Precision. This can be computed using the following formula:

Precision is a good performance measure for this task, as it looks at the number of mines the model predicted correctly out of all of the spots it thought were mines. This measure gives us an understanding of how the model is doing properly, as it looks at the minority (mines) and precisely tells us how many of its predictions were correct. We can see the model performed decently well with an average precision of around 0.72. This put into words tells us that, on average, 71% of the models’ predictions were correct. Although in the same range, his number fluctuates more when looking at the validation set, however this can be attributed to common differences in the training and validation set. This measure although good, does not give us a good idea of how the model performs from the perspective of the minesweeper board. It does not tell us how the model did with respect to the whole board. This is the reason the next measure was used.

A graph with a line and a line

Description automatically generated

Figure 5.3: Recall on Epochs

Recall is a good performance measure for this task as it essentially provides a metric of the proportion of actual positives that were identified as positives by the model. This basically telling us how many mines the model guessed correctly out of all the mines on the grid. Recall can be computed as the following:

We can see on average, the model had a recall of about 0.35, telling us that on average it got 35% of all the miens on the board. This may not seem impressive at first glance, however when we look into the data and how minesweeper works we can break this number down further. When the data was fed into the model, the degree to which boards were “uncovered” were randomly generated at the time of the dataset creation. The probability a square would be “uncovered”, therefore uncovering neighbor squares was 0.3. The reason this was done was to ensure the model would be able to identify mine locations without having access to most of the board, just like a real game situation. This means the model was mostly given heavily covered grids to train on. This actually makes the 0.35 recall rather impressive, as the model is not expected to guess mines on completely covered parts of the board (most of the data had large uncovered portions). Looking at the recall as a function of how uncovered the board was, we can say that the model did a rather good job of identifying mines near uncovered spots of the board, something which mirrors a real life scenario.

We decided not to use an F1 score to evaluate the model, however given this is a combination of recall and precision, the two most important metrics in this instance, in the future this would definitely be something used. Another idea could be making a custom metric, a function of recall which takes into consideration how uncovered the board is, giving us a more “accurate” recall value.

**Qualitative Results**

A few different example screenshots taken during gameplay can be used to get some information into how the model performs.

A black and red checkered pattern

Description automatically generated

Above we can see an example scenario. The heatmap output, as discussed earlier shows the mine probabilities. The brighter red a square is, the more likely that is the location of a mine. We can see that most mine locations surrounding the uncovered area can be identified. In this scenario we can see most of the board is covered. The varying amounts of red are essentially random guesses as the agent is not sure if there are mines there or not. This is likely because the model was trained on a one channel input, not specifically provided with a mask of covered grid locations. This will be covered in more depth in the future work directions section.

A close-up of a grid

Description automatically generated

This is reinforced by another example where a large majority of the grid is covered. Absolute mine locations can be distinguished, but apart from that, the model does not perform well on large, covered portions.

A collage of a grid

Description automatically generated

In another example here, we are able to see the opposite. More of the board is uncovered therefore the agent performs much better. Many mine locations are identified with high confidence, along with locations slightly further away from direct mines. However, again, the model does not perform well with large, covered portions of the grid far away from uncovered spots. A grid like this would be the most similar to something the model was trained on, based upon how much of the board was uncovered. This is a good visualization of the recall metric which was discussed in the quantitative results portion of this report.

**Discussion (Limits, Future Work and Implications)**

Looking at the work, there are many areas which could use improvement, which, with more time could easily be ironed out and help the model perform much better. In this section of the report, we will go over them and see how in the future we could possibly fix these.  
  
 One of the main issues with the models is the fact that it tends to make predictions outside of where mines could be detected with the current state of the board. Although it is possible for the network to make these predictions given a game state, it would require a deeper convolutional neural network and possibly an entirely different approach. In this case, the CNN is not set up to do this, therefore the predictions on heavily covered areas of the board resembling a checkerboard shape would be rather erroneous. One way of approaching this problem could be fixing the input of the model. Our data had only 1 channel, meaning that the game state at a given time was encoded numerically to a format we decided on and that was what the CNN was trained on. Adding an additional channel to this could be beneficial. Instead of each spot on the grid being a single value (data shape being (1, 30, 16, 1)) it could be 2 values, the second one specifying if the spot is covered or not by giving a 1 or 0. This type of mask would make the new data shape (2,30,16,1) and might help the model identify and not put effort towards heavily covered parts of the grid, reducing false positives and ultimately increasing performance.

Another issue that can be looked at is the fact that the model is not able to make deep connections apart form nodes in the near vicinity of uncovered nodes. A goal of the project would be to have the model be able to make predictions further from the currently exposed area. To have the model do this, looking into deeper CNN’s and different architectures would be beneficial. As of right now the CNN is a rather shallow and simple CNN with normalization and dropouts to avoid overfitting. A problem in the real world which could mirror minesweeper in a sort of way could be image segmentation, where CNN’s classify portions of an image of being something specific. For example, providing the CNN a frame from a surgical video, different parts of anatomy could be classified and colored in the frame. This could be compared to minesweeper, where we are essentially doing this, but with the minesweeper board. A commonly used architecture for image segmentation are U-nets, which tend to be good for per-pixel classifications, or probability maps of sorts. Implementing a more complex U-net architecture could be a change worth investigating to improve model performance.

**Conclusion**

In conclusion, this work was a success in terms of performance. Looking at the results, it is evident the agent is decent at detecting mines in the near vicinity of the exposed area. This was the goal of the project, to assist the user in finding mines to a decently high degree of accuracy, which it does. In future work, integration of better data preprocessing and looking into more complex architectures such as U-nets which are used for similar image segmentation tasks would be quite beneficial. These next steps would allow the model to potentially make more complex connections between revealed areas of the board and non-revealed areas allowing for deeper and more complex connections, giving further mine locations. Apart from this the project was a success and a tremendous learning experience.

# References

CodingLikeMad. (2021, February 15). *AI Neural Network Beats Minesweeper[Tech Explanation Only]*. Retrieved from YouTube: https://www.youtube.com/watch?v=hyOVwwp4qu4&t=0s

Gardea, L., Koontz, G., & Silva, R. (2015). *Training a Minesweeper Solver.* Retrieved from https://luisgardea.com/assets/images/minesweeper.pdf

Nakov, P., & Wei, Z. (2003, May 14). *MINESWEEPER, #MINESWEEPER.* Retrieved from https://www.researchgate.net/profile/Preslav-Nakov/publication/228613592\_Minesweeper\_Minesweeper/links/00b7d523c1308589ef000000/Minesweeper-Minesweeper.pdf

ryanbaldini. (2020, April 13). *MineSweeperNeuralNet*. Retrieved from GitHub: https://github.com/ryanbaldini/MineSweeperNeuralNet

Scientist, J. T. (2022, May 30). *convolutional neural network (cnn) for minesweeper - cnn architecture*. Retrieved from YouTube: https://www.youtube.com/watch?v=Nb-pVqqEi2c&t=399s

Ronneberger, O. (n.d.). *NET: Convolutional Networks for Biomedical Image Segmentation*. U-Net: Convolutional Networks for Biomedical Image Segmentation.Retrieved from https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/