Mine Sweeper and Neural Networks

Tristan Lotivio - tcl3c@mtmail.mtsu.edu
CSCI 4850-001
Middle Tennessee State University
Course: Neural Networks - Spring 2020

Jeffrey Mitchell - jcm9e@mtmail.mtsu.edu
CSCI 4850-001
Middle Tennessee State University
Course: Neural Networks - Spring 2020

Darien Kidwell - dak3t@mtmail.mtsu.edu
CSCI 4850-001
Middle Tennessee State University
Course: Neural Networks - Spring 2020

Robert Bird - rjb5n@mtmail.mtsu.edu
CSCI 4850-001
Middle Tennessee State University
Course: Neural Networks - Spring 2020

Daniel Cox - djc6w@mtmail.mtsu.edu
CSCI 4850-001
Middle Tennessee State University
Course: Neural Networks - Spring 2020

Muhammad Abed - mja3u@mtmail.mtsu.edu
CSCI 4850-001
Middle Tennessee State University
Course: Neural Networks - Spring 2020

***Abstract*— Mine-Sweeper is NP-Complete. This means that it is not possible for an algorithm to complete the game consistently, this also creates a difficult task for the net to solve. However, because the game is so popular, we have found two other solutions that we used as inspiration for this project: the Baldini Solution and the Stanford Solution. These solutions provided a foundation for us to build off of with our own net. Our neural network takes singular board states and develops probability hot spots based on where the net believes there are mines. We found that due to the quality of NP-Completeness, we were not able to reach our desired accuracy, however due to the sense that these predictions are centered around probability, the data shows that it is significantly more accurate than the statistics display. One thing we learned during this project is that neural nets can be used to solve many different problems that would not be considered possible to solve using iterative algorithms. While it took numerous trials, large data sets, and many modifications, we were eventually able to develop a neural net that predicted a final board state, from a partial, with remarkable, comparative, accuracy.**

I. Introduction

The first thing that needs to be pointed out is that Mine-Sweeper is NP-Complete [7]. It should be known that this problem is optimally solved with machine learning, due to the fact that it is an NP-Complete problem [3]. The generality of Mine-Sweeper, as a game, allows for a multitude of possible approaches that can be taken in an attempt to create an optimal solution for the completion of the game. These are a couple of the main reasons why we chose to attempt to tackle this problem. These two points about the Mine-Sweeper task will be explained with a bit more detail in the next two subsections. Some of the approaches that we will be attempting are through: Reinforcement Learning and the use of Recurrent and Convolutional Neural Networks.

A. NP-Completeness?

Minesweeper as a concept is incredibly interesting to study, due to the game's quality of being NP-Complete. This is because that fact eliminates the ability to be consistently solved in an iterative algorithmic fashion: there has essentially been no pattern to spot any form of consistency in the generation of mines that would allow us to know, for certain, if a mine is in a specific square. It is the responsibility of the programmer, and in this case our group, to explore the options that we could utilize, through a Neural Network, in order to write code that can quickly and efficiently, learn, play, and win, with some form of consistency, at our simple implementation Mine-Sweeper.

B. *Generality*

The intriguing part about a game such as Mine-Sweeper is the simplicity of the structure and mechanics of the game. Like any other board game, it is simple in format with a generally low amount of possible variations of pieces, and a low level of complexity in user interaction, though it remains complex enough that it requires some level of machine learning to perform well in the game [3 and 6]. Because of this, we have both flexibility in the way that we choose to approach this task, as well as support due to Mine-Sweeper being popular enough such that this same experiment has been conducted before with good enough documentation for our group to use to spring board off of.

II. Background

As previously mentioned, the Mine-Sweeper Consistency problem has been a popular goal to push for in the past. With this in mind, we had two different previously attempted experiments that we have used as a form of foundation for our research. These two experiments led to the Stanford Solution and to the Baldini Solution. The results of these two attempts provided us with various strategies that we could adapt and build off of in hopes to find differing, and in hope, more consistent results.

A. *Stanford Solution*

The Stanford Solution is in reference to a paper from 2015, conducted by Undergraduate Students at Stanford who attempted two methods for their solution [2]. The first was a supervised learning process in which it was trained by learning through games that were won, and tested in a way such that the space with the lowest possibility of a mine being present was chosen, else a random space was chosen, until the game was lost. The second method to be tested was utilizing Q-Learning, which has two phases, the first being for initializing the game, and the second, to play the rest of the way through. An array which learned the game as it played essentially began at a corner square, then decided the lowest possibility of mines that could be chosen, chose that spot, then updated the memory structure based on that decision. Mines were marked as -1, and all other spaces were marked as one from the initial loading of the board. The reason for these values being assigned was to determine, in general, the location of mines.

B. *Baldini Solution*

The Baldini Solution is stored in a GitHub repository that was posted in mid-2017 [4]. This repository details the general idea of how a Convolutional Neural Network could be utilized in order to solve the problem. Within Baldini's code, we can observe a similar input and output to what we used in our code, as well as a set of 5 3x3 convolutional layers and 1 1x1 dense layer. These layers utilize ReLu and Sigmoid activation functions, respectively. Another interesting difference was the design choice to convolute the structure of the input array by spreading the board state through an 11x16x16 board array. The board array manages the information in a way that all different values present, in the flattened array, are abstracted to different indices in the array. This gives the net more exposure to specific and isolated patterns in the array, which allows for more targeted and enhanced predictions of the arbitrary board state. This provides the net with a method to process and create parameters for the whole board with comparative ease, since the board is also 16x16 with 31 more mines to handle in comparison to our own.

III. Methods

A. *Project Structure*

The general structure that we took to build our program follows a development path such that our group could explore the Mine-Sweeper problem from a different perspective. In terms of structure, the first task was to decide the IDE, which we elected to use JupyterLab, as it was the most readily available tool that the whole group could cooperatively use along with GitHub. With this in mind, the decision was also made to simplify the board size possibilities, to a single 9x9 board size with 9 mines, which allows for our group to focus on developing a novel learning process that handles a single simple implementation. We also from this, use a simple console graphically based version of minesweeper to run and test our net. Handling this data, is a file structure

B. *Net Development*

Our net essentially takes singular board states, and develops probability hotspots based off of where the net believes the mines to be. This is returned through our .csv file structure, and handled in the other .ipynb.

C. *Training*

It seems that currently the most success has been seen in the Convolution Net as that was the only type of model that we could manage to properly format in order to train well on our data, if at all. At this moment, we are achieving roughly a .5 consistency as training continues. This has been through strategies previously used, including: testing various kernel sizes and layer densities to reach differing amounts of parameters, testing different learning rates, and even shifting some of our initial data set's values. This collection of differentiation was followed by extensive training over a large data set with significant numbers of epochs and a continuously decreasing batch size as the number of eligible patterns may decrease for each iteration of the model.

IV. Results

Something that we found particularly interesting was the consistency, or lack thereof, concerning the data (Fig. 2 and Fig. 3) in comparison with the calculated accuracy (Fig. 4). Throughout the training it was noticed that in some cases, the accuracy would not change, however the predicted data would. While the accuracy may not be as high as we had hoped or expected it to be, the net is continuously predicting close values to the actual test in spaces where it is not completely correct. These values represent a probability of a mine residing in that space. For example, in many spaces where there should be a 1 (meaning there is, with complete certainty, a mine) there is say a .7 or a .9, pointing towards a high possibility of a mine being in this space. This is because of the NP-Completeness of the Mine-Sweeper Consistency problem.Currently it is not considered possible to consistently predict, with complete certainty, where a mine would be; even for a professional, when playing the game.

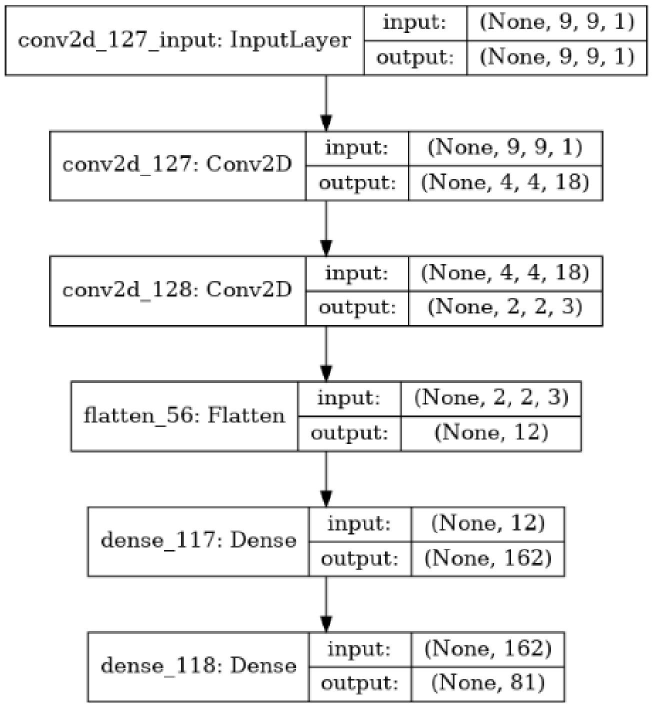

Fig. 1. This figure represents the model structure which we developed

$$\begin{bmatrix}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.2 & 1.0 & 0.1 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.2 & 0.3 & 0.3 \\
0.0 & 0.0 & 0.2 & 0.1 & 0.1 & 0.1 & 0.8 & 0.1 & 0.0 \\
0.0 & 0.0 & 0.1 & 0.9 & 0.2 & 0.1 & 0.1 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.0 \\
0.1 & 0.2 & 0.2 & 0.0 & 0.1 & 0.8 & 0.2 & 0.1 & 0.1 \\
0.3 & 0.9 & 0.2 & 0.2 & 0.4 & 0.8 & 0.3 & 1.0 & 0.3 \\
1.0 & 0.2 & 0.9 & 0.3 & 0.3 & 0.4 & 0.5 & 0.1 & 0.2 \\
0.2 & 0.3 & 0.0 & 0.1 & 0.1 & 0.9 & 0.1 & 0.0 & 0.0
\end{bmatrix}$$

Fig. 2. This is the grid space probability matrix that the net predicted

$$\begin{bmatrix}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 1.0 & 0.1 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.2 & 0.2 & 0.1 \\
0.0 & 0.0 & 0.1 & 0.1 & 0.1 & 0.1 & 1.0 & 0.1 & 0.0 \\
0.0 & 0.0 & 0.1 & 1.0 & 0.1 & 0.1 & 0.1 & 0.1 & 0.0 \\
0.0 & 0.0 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.0 & 0.0 \\
0.1 & 0.1 & 0.1 & 0.0 & 0.2 & 1.0 & 0.3 & 0.1 & 0.1 \\
0.2 & 1.0 & 0.2 & 0.1 & 0.2 & 1.0 & 0.3 & 1.0 & 0.1 \\
1.0 & 0.3 & 1.0 & 0.1 & 0.2 & 0.2 & 0.3 & 0.1 & 0.1 \\
0.1 & 0.2 & 0.1 & 0.1 & 0.1 & 1.0 & 0.1 & 0.0 & 0.0
\end{bmatrix}$$

Fig. 3. This is the grid space probability matrix of the actual grid to compare the prediction to for validation

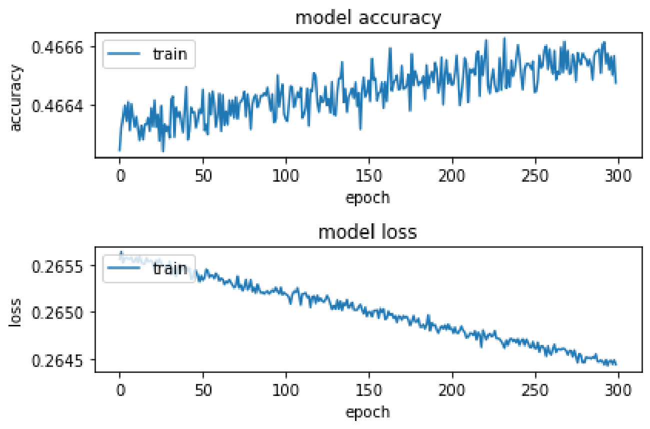

Fig. 4. This figure displays the val loss and accuracy statistics if the latest training

A. *expected results*

Initially, our expectations were to have the net complete, and win, the game approximately 8/10 times that it plays through. We had this expectation due to the fact that the net could select a space with a mine in it to simply begin the game, ending the attempt immediately. Because of this factor, we decided to set our expectations to what we assumed to be lower than what we otherwise would have stated.

B. *Learned Experiences*

A lot of the concepts that the group learned were multi-faceted, in all in thanks to the interestingly new global normal throughout the year. Not only were members in our group first exposed to the basics of GitHub and Git commands, but we we were also exposed to them in less than forgiving circumstances, along with a significant portion of course materials. Due in large part to these circumstances, the group made best of our circumstances and buckled down to learn these circumstances, and on average has become generally proficient in the tools that were grouped for group success. On a less tangential note, the group also made great strides in the domain of class specific objectives as well. The ability to apply and learn from the tools that we learned in the form of Optimizers and Loss functions that we learned and used in class, within our project was very eye opening to see. The difference between actually doing the encoding oneself, and receiving semi completed code is very different, and felt heavily in this case. This not only forced us to learn the techniques necessary to develop our model, but also to develop it real-time, and truly understand all aspects of the project, as well as require that same level of comprehension from all team members. Lastly, the ability to be able to experience a microcosm of what the world of Machine-Learning problems was very eye-opening to the uncertainty of many situations. Like humans, our net would create probability hotspots based off of what the net assumed the whole board state to be, and equally like humans the net allows us to visualize that same uncertainty (fig 2.). We can see that in training, the accuracy, can sometimes still be under 50 percent (fig. 4), which low, when visualized still bares an remarkably close resemblance, to the trained grid (fig. 2, fig. 3). It is in this that we can also see that to some degree we are successful, and that with further optimizations, our success can be expanded further.

V. Discussion

One of the most important things that we learned when analyzing the problem, was the high level of flexibility that is made available to an AI programmer, when it comes to approaching a problem such as this. While we would have been most definitely blessed to have come away from this project with more definitive results in any way shape or form, it does not change the fact that our group was able to spend a good portion of our time, while developing the base structure information passing system of the project duration, simply debating the structure of the learning algorithm that we would use, which changed often between variations of Q-Learning, Supervised Learning, and Convolutional Learning. Even without persuasion from the works of the past projects found in Background, all are viable options, given enough time and focus. Therefore, not only could this problem be solved in ways that we have yet to consider, but is also indicative of other problems in this field that may have a higher perceived level of structural complexity. So long as said arbitrary problem, regardless of complexity, can be tabulated, metricized, or categorized in some way by means of a computer, then it can be solved by means of a normal algorithm or a complex of a machine learning process.

References

[1] Prateek BajajCheck and Prateek Bajaj. 2019. Reinforcement learning. (December 2019). Retrieved April 29, 2020 from https://www.geeksforgeeks.org/what-is-reinforcement-learning/

[2] Luis Gardea, Griffin Koontz, and Ryan Silva. Training a Minesweeper Solver,

[3] Anon. ICS 161: Design and Analysis of Algorithms Lecture notes for March 12, 1996. Retrieved April 29, 2020 from https://www.ics.uci.edu/ eppstein/161/960312.html

[4] Anon. Retrieved April 29, 2020 from https://xlinux.nist.gov/dads/HTML/npcomplete.html

[5] Ryanbaldini. ryanbaldini/MineSweeperNeuralNet. Retrieved April 29, 2020 from https://github.com/ryanbaldini/MineSweeperNeuralNet

[6] Lilian Weng. 2018. A (Long) Peek into Reinforcement Learning. (February 2018). Retrieved April 29, 2020 from https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

[7] Richard Kaye. 2000. Minesweeper is NP-complete. The Mathematical Intelligencer 22, 2 (2000), 915.