Marc Vucovich

April 30, 2019

AI Practicum Report

For my final practicum I created a Neural Network that is able to play the game Tic-Tac-Toe. In order to implement this Neural Network, I needed to find a data set that I could use to train my network how to play the game optimally. To do this I created a minimax algorithm and stored each game state as well as the board states in a csv file. I then created a feed forward network because I need to take a given input and return an output. According to my research, and Analytics India, a feed forward is the best and easies type of Neural Network to implement this problem[1].

In order to create my Neural Network, I went through a lot of trial and error approaches. When I first started, I tried to create a class that had functions that would run through the steps of neural network. This approach had a lot of bugs and had an unreliable prediction. I eventually resorted to using TensorFlow and Keras in order to train my Neural Network. In my project folder, I have five different python files and nine classes. Many of my classes were near repeats of each other as three of my files are similar but each had a separate job at hand. When I first started training my neural net, I was using a data set I found online. However, this dataset was limited, and this led to my Neural Net having a poor accuracy score. That is when I had decided to implement a minimax algorithm to simulate 10,000 games and create my own data set. In my TicTacToeGame.py I have three classes. The first class is called Board. This class initializes an array to store the current board state, an array that stores all of the past moves, and a variable to declare a winner. I then have a function that prints out the array in the terminal in a 3x3 shape to resemble a tic tac toe board. This classes main task is implementing the moves made and storing the game state. In TicTacToe.py I also have a class called human which I first used to ask the user for a move in the game. After I tested the game by playing it myself, I changed the job of the Human class from asking a user for input to a random agent so that I would be able to simulate many games. The third class in this file is my minimax class. I used a minimax algorithm to play against the random agent because it would always make the optimal move [2]. After I tested the program for bugs, I ran a simulation of 10,000 games. In each game, whenever my minimax Algorithm was called to make a move, I would read the array containing the board state into a csv file and the optimal move from the minimax algorithm into another csv file. By the end of the simulation, I had about 35,000 board states and their corresponding optimal moves.

With the data ready to be processed, I created my Neural Network. In the file FinalNetwork.py, I read in my data sets into two separate 2D NumPy arrays. My first array contained all of the board states from my simulation, and the second array contained the location for the optimal move. Since my Neural Network is a Multi-Class network, I needed to modify the optimal move array from containing the number of the optimal location to an array of zeros and a one in the optimal location. It was then time to create the network. I imported keras.models and used a TensorFlow backend to run the network. For my network, I used two hidden layers to process the data. I messed around with the number of layers, but I saw diminishing returns on the accuracy when there were more than two hidden layers. I found in my research that the rectified linear unit, or relu, activation function was the most popular and used activation function in the hidden layers [3]. With that in mind I used a relu activation function in the hidden layers to keep the values in a reasonable range. I also found during my research that a multi-class network requires a SoftMax activation function for the output layer [4]. This is due to the fact that the output is not binary like a sigmoid function. I implemented this so that I could find which location given the board state is most likely to be the optimal move. After I created the layers, I compiled the network. While compiling the network, I once again make use of the information I found while researching multi-class Neural networks, and I used a categorical cross-entropy loss function and an optimizer called adam because they are best suited for predicting the correct label [5]. With the model compiled, I ran my simulation data through the network and saved the model. I then load the model in my Prediction.py file. This file takes in the board state and makes a prediction for the optimal move.

Finally, I have two more files, PlayNNSimulation.py and PlayNeuralNetwork.py, that are nearly identical. The both have the same first two classes, Board and Human, from TicTacToeGame.py except the simulation one uses a random agent to play against the Neural Network, and PlayNeuralNet.py allows a person to play against the Network. The third class in both files is the NerualNet class. This class is called when it is player 2’s turn. It then takes the board state at the time and sends it to the prediction file. The prediction file then returns an array with the probabilities of each location on the board being the optimal location. I then check to see if the max from the array is a legal move because the network is only about 94% accurate. If it is a legal move, I mark the board. If not, I find the next highest probability for a legal move and mark that spot. This process continues till the game ends.

My Neural Network performs very well against a random agent as well as a real person. It is not 100% successful but based on my data I would consider my implementation a success. Graphs demonstrating my success as well as explanations can be seen below.

A screenshot of a cell phone

Description automatically generated

The graph above shows the amount of wins, draws and losses my Neural Network had out of 100 simulated games. To create this graph, I ran my program against a random agent 100 times and store the result of the game into a NumPy array. It is apparent based on my distribution that my Neural Network has a very high win rate. Given that I have my Neural Network set up as Player 2, I believe this is a great accomplishment. There are a few losses, but when I looked back over the simulation, the losses seemed to arise on a specific board state. With more time, I would be able to add more data into my data set so that I could teach it the right move it should have made.

A screenshot of a cell phone

Description automatically generated

As for the Neural Network itself, the accuracy through each epoch during its training can be seen above. We can see that the accuracy of the model improves very quickly and then starts to slowly but still continuously increase. The final accuracy for this model came out to be 94.86%. This is a drastic improvement from the 58% accuracy I was getting with the limited data set I was using before I created my own. Overall, I would consider the models very high accuracy rate to be a successful implementation.

A picture containing screenshot

Description automatically generated

Along with the success rate, my Neural Network’s loss function looks near perfect. By analyzing the two graphs above, we can see that the predictions become more accurate the more we run through the data set. This can be shown in the loss function because the amount between the model’s prediction and the actually output rapidly decreases and the level out. This shows a successful changing of the weights through my feed forward network.

In conclusion, I believe my implantation of a minimax agent that gave data to my neural network was a success. My Network has a high winning percentage against a random agent, and the accuracy shown above is in the top ten percentile. Although the program does have a few losses, this could be corrected with even more data. With more time, I would create more data so that my Network could become even more accurate. I believe the set-up of my network itself, the two hidden layers, the SoftMax activation function for the output layer and categorical cross entropy loss function, is a great setup for this type of problem. As this was my first neural network, I am very pleased with the results. My Neural Network:

* Has a 94.86% accuracy rate predicting the optimal move given a board state
* Has a 77%-win rate against a random agent
* Has an 18% draw rate against a random agent
* Taught me a lot about patience

Overall, the project taught me a lot, and I plan on using this knowledge to start creating more Neural Networks.

Works Cited

[1] Kishan Maladkar. 2018. 6 Types of Artificial Neural Networks Currently Being Used in ML.

(November 2018). Retrieved May 2, 2019 from <https://www.analyticsindiamag.com/6->

types-of-artificial-neural-networks-currently-being-used-in-todays-technology/

[2] Vamsi Sangam. Artificial Intelligence - MiniMax Algorithm. Retrieved May 2, 2019 from

http://theoryofprogramming.com/2017/12/12/minimax-algorithm/

[3] 1395550283894582. 2017. Activation Functions in Neural Networks. (September 2017).

Retrieved May 2, 2019 from <https://towardsdatascience.com/activation-functions-neural->

networks-1cbd9f8d91d6

[4] Franky. 2018. Multi-Label Classification and Class Activation Map on Fashion-MNIST.

(July 2018). Retrieved May 2, 2019 from <https://towardsdatascience.com/multi-label->

classification-and-class-activation-map-on-fashion-mnist-1454f09f5925

[5] Anon. Loss Functions¶. Retrieved May 2, 2019 from <https://ml->

cheatsheet.readthedocs.io/en/latest/loss\_functions.html