COMS4030A: Adaptive Computation and Machine Learning Term Project, An Investigation into AI Agents for Tic-Tac-Toe

Jason Chalom 711985

***Abstract*—This project was an attempt to make a neural network based AIs which could play Tic-Tac-Toe better than random agents on any sized game board, however this project was a failure. This was due to underestimating the time constraints and how to generate the right data set to use for this task. Another bad estimation was assuming the generalisation over board sizes was possible and would produce adequate results - this was not the case.**

***Keywords*—Machine Learning, Artificial Intelligence, Tic-Tac-Toe, ANN**

I. Introduction

People have always had a fascination with games and the problems these domains present. Games can become computationally complex and adaptive algorithms can play an important role in making good AI agents. There are many approaches to playing games and different games will lend themselves to different methods and algorithms. [1]

A. Problem Statement

A comparison of non-tree search method based AI agents to play Tic-Tac-Toe in different board sizes.

II. Background

A. The Game Tic-Tac-Toe

Tic-Tac-Toe also known as noughts and crosses is a deterministic, zero-sum game where the object of the game is to produce a complete line of X's or O's in any direction on the board (which is usually 3x3). It is a very simple game which always has a set of optimal moves. The first player always has an unfair advantage whilst the second player has the ability to force a draw. The first player should never lose. [1]

B. Artificial Neural Networks

Artificial Neural Networks (ANN) are computational simulations inspired by the biology behind the operation of a brain. These are computational networks of nodes where each node is a mathematical representation of a neuron. These networks tend to have a structure where each layer produces the next layer in a forward fashion however there are many different types and implementations of neural networks. [2]

III. Methodology

A. General Assumptions

I have assumed that a randomly generated data-set of one million game states is sufficient to train any of the algorithms to their maximum potential. I have also assumed that all drawn diagrams were drawn using http://draw.io/ and charts were made with Microsoft Excel 2013. All results are averages of many experimental runs being performed.

B. Testing Method

I built a game supervisor which handles and maintains the game state for both agents. This allowed the results to be consistent between different AI agents per run. I ran multiple game simulations between all the AI implementations and the random agent. This acts as the base-line result. The performance metrics used looked at the win/loss ratio of each AI agent. Two board sizes were tested, namely a 3x3 and 5x5. These were averages taken over running 100's of games.

C. Implementation

All programming was done in Python. I used numpy, and the standard Python libraries in my implementations of Tic-Tac-Toe and the AI agents I tested. The agents which made use of algorithms which needed training were trained separately and their data was saved and recalled using the numpy save and load features when those AIs were tested. Initially I generated an array of a million random game actions where each row is a game state with an associated action and the final result for player 1 for that entire game. I generated a data-set for each board size. All algorithms which need data made use of these data-sets. I split this data set into training and testing data with a 80 to 20 split in favour of the training data. Validation was done with fresh data.

D. Artificial Neural Network

The artificial neural networks had a very similar structure. They both have three layers where each layer has the same number of nodes as there are elements in their respective boards. Every layer uses the sigmoid as their activation function and bias values for the layers (excluding output) are used. The networks were trained used stochastic gradient descent. The final weights generated after 24 epochs were saved and when the respective network was applied the weights were multiplied together with a given input matrix to get the result - which is the next move for the agent to make.

TABLE I. RESULTS OF AGENTS

Agent	Win (%)	Draw (%)	Loss (%)
3x3	1	40	59
5x5	1	34	65

E. Using Machine Learning to Deal with Board Sizes

I attempted to make an AI which can play any sized board. The first method tried was to apply principle components analysis (PCA) to standardise the number of inputs and outputs. The outputs would then be mapped into proper coordinates to make a move. This failed because the data does not lend itself to PCA. The results from generating the principle components meant that most the data from the game state had been lost and could not be recovered. This was due to how each player was represented by the same numerical value (for each respectively) on the board.
My second attempt was to use an auto-encoder to achieve the same goal. This attempt also failed because I could not get the neural network to produce accurate enough results to meaningfully use.

IV. RESULTS

The results of the neural network AIs were terrible. Running an average of 100 games against the random agent for both neural network agents showed them both producing results far worse than random.

A. Discussion and Comparison of Results

The 3x3 agent performed better than the 5x5 agent due to the state space of the board being smaller. The two most likely reasons why both the agents failed to perform with any kind of positive result is because the data set generated was not good. There was a faulty assumption that any move that leads to a match win is a positive move. Another fault was that the network structures of the agents were far too simple for the complexity of the problem being approximated. The attempts to produce an AI which could handle multiple board sizes also failed due to the data-set as well as the methods being employed being unable to handle the actual game states because the numbers do not translate well into other domains.

B. Problems Faced

I underestimated the amount of time I would need to both develop and train the agents. I also did not allot time for any unforeseen problems arising. The data-set I generated was terrible. It not not properly reflect the good move set since the moves were randomly generated. This means that a bad move could still have a win outcome as the other random agent may still make mistakes.

V. CONCLUSION

This project was a failure. I should have used a better data-set and not attempted to generalise the board size. The agents produced were not good at playing the game, rather they were worse than random. A more complex network structure and a better data-set would have produced far better results.

GLOSSARY OF TERMS

AI - Artificial Intelligence

ACML - Adaptive Computation and Machine Learning

HPC - High Performance Computing

PCA - Principle Components Analysis

REFERENCES

[1] Russell, Stuart and Norvig, Peter, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2009.

[2] Mitchell, T.M. *Machine Learning*, McGraw-Hill, 1997.