What are the effects of using genetic algorithms to optimize a neural network to play Connect Four?

Jonathan Clarke – URN: 6542571

A dissertation submitted in partial fulfilment of the requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

August 2021

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Alireza Tamaddoni Nezhad

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Jonathan Clarke
August 2021

TABLE OF CONTENT

1. Introduction

1.1 Background Problem

In this day and age of artificial intelligence revolutionising the worldwide industry, neural networks have seen a growing popularity and are probably the most widely used predictive modelling techniques. These have been used in a large variety of research and business problems from facial recognition to video and song recommendations or even optimizing banking decisions.

However, there are a few problems that arises when using neural networks. First of all, they are very complex and require either a thorough understanding of the topic to actually implement one. This can be bypassed by using some existing libraries such as TensorFlow or PyTorch, but by doing so, we have less flexibility to build a program that fits our exact needs. Secondly, the amount of data needed in order to train a neural network to learn a generalized solution to a problem is usually very large. Depending on the dataset, a network could take days/weeks to properly train. Most importantly, the biggest fall back of neural networks is their black box nature. It is not often the case that a human person can understand the reasoning behind a trained neural network. Due to this, we cannot interpret or explain the predictions of a neural network, which can be critical in some problems.

1.2 Motivation

This is where genetic algorithms come in to play. The concept of genetic algorithms is inspired by the evolution process and natural selection; by using some measure of fitness, the algorithm will only choose the fittest individuals to repopulate the next generation. This process is often used in machine learning to optimize a given set of parameters on an AI model to achieve a specified goal. Due to the complex nature of neural networks, genetic algorithms are a great tool to help us optimize them in order to find the best combination of parameters in the network. Through the evolutional process of genetic algorithms, we can also observe and understand the process that the network goes through by examining key changes between populations.

I always enjoyed seeing how A.I could be applied to video games by creating a model that learns to play the game. When I started studying computer science a few years ago for my IB, I learned about genetic algorithms which piqued my interest because these are often used to optimize a A.I model to play games.

My aim in this project is to use a genetic algorithm to optimize a neural network to learn how to play Connect Four. I chose the game of Connect Four as there are very few rules which makes it easy to play and understand. Even though this may be a simple game, with a standard board of 7x6, there are 417750499600 possible outcomes[i]. I believe this will result in enough possibilities to allow my neural network to train properly.

Doing this project would allow me to explore and experience this interest first-hand; therefore allowing me to evaluate how a neural network model is optimized and to observe if their previously mentioned fallbacks are solved by using a genetic algorithm.

1.3 Aims and Objectives

The goal is to compare and observe two different models: a neural network trained with a history of played games, a neural network train using genetic algorithms. The aims and objectives are the following:

- Research the different applications of genetic algorithms and how they are used
- Research and understand already existing solutions to the problems
- Use existing solutions to creates our models
- Compare the differences in the training stages of both models
- Compare how both models perform when playing against each other
- Compare how both models perform when playing against random agent

1.4 Success Criteria

In order to assess and evaluate how we meet our goals set above; we need to establish some expectations as to how genetic algorithms will affect the neural network model. These will come in the form of success criteria's which we will use to evaluate how the experiment went.

- A working implementation of the Connect Four game
- A neural network model trained using data from the history of games played by two random agents
- A neural network model optimized using a genetic algorithm playing against the unoptimized neural network
- A neural network model optimized using a genetic algorithm and data from the history of games played by two unoptimized neural networks
- A higher interpretability and understanding of the GA trained model over the non-GA trained model
- A shorter training time for the GA trained model over the non-GA trained model
- A higher win rate from the GA trained model over the non-GA trained model

1.5 Structure of Report

Section 1 : Introduction – This section introduces the background problem and motivation for the topic tackled in the report. It also briefly explains the aims and objectives as well as success criteria that should be met by the end of the paper.

Section 2 : Literature Review – This section will cover research about the important topics used in the experiment. This includes an explanation of the Connect Four game and how it's a solved game, how neural networks can be applied for the game, the applications of genetic algorithms in a more general sense, and finally it will talk about some existing solutions to the problem at hand.

Section 3 : System Design and Implementation – This section explains the system implementation that is used to conduct the experiment. It explains all the key algorithms and functions in the code as well as the choices behind them.

Section 4 : Results and Evaluation – This section reflects on the results obtained from the experiment. It explains in a bit more depth how the experiment was initiated and what these results mean in relation to the success criteria

Section 5 : Conclusion – This section will recall what was achieved in this report as a whole and from this information it will answer the research question. Finally I will also suggest what future work could be done from this point as well as my reflection on what I have learned and could do differently.

2. Literature Review

2.1 The solved gamed of Connect Four

Connect Four is a connection board game that was first published in 1974, where two players take turns dropping a coloured disc in one of the columns of the board. The goal is to form a line of 4 discs of the same colour either horizontally, vertically, or diagonally, if one of the players achieves this, they win the game. A draw occurs when all columns on the board are filled up and neither player has a line of 4 discs. There exist many variations of the Connect Four game, I will be looking at the most commonly used one which uses a 6 x 7 board.

The game was solved in 1988 by two computer scientists independently, James Dow Allen[ii] and Louis Victor Allis[iii]. They established that the first player can always force a win by playing in the middle column first. If the first player plays in the columns adjacent to the middle first, this allows the second player to force a draw. And if the first player plays in one of the four outer columns first, then the second player can force a win. Of course this implies that both players are able to play perfectly.

Since then, the computational power that we possess has drastically increased, which led to Connect Four being solved through brute-force methods. This was first done by John Tromp[iv], by compiling a database of every 8-ply positions of the game and their outcome. Using this database, every state of the game can be evaluated in a matter of seconds. However, the most popular algorithm used to strongly solve this game is the minimax algorithm which is also used for solving other games such as Chess, Go, Isola, Checkers, and many other two player games, zero-sum games. It works by traversing the game tree backwards and picking moves that will maximize its chance of winning while minimizing the chance of losing (i.e. the opponent winning). This algorithm is often optimized with alpha-beta pruning, which seeks to reduce the number of nodes that are being evaluated by the minimax algorithm when traversing the game tree.

2.2 Neural networks applied to Connect Four

I wanted to apply this neural network project on a game as it is an insightful way to observe the capabilities and limits of such a complex system. The game was narrowed to Connect Four as it is more complicated than trivial games like Tic-Tac-Toe but yet has a smaller search space than games like Go or Chess. Connect Four is a solved game, which means that the use of neural networks may seem excessive. Nonetheless, this game allows us to work with simple data and small search spaces. The process of gathering the data is much simpler, and therefore we can concentrate more on building the model.

An experiment was carried out by Luke Kim, Hormazd Godrej and Chi Trung Nguyen[v] in 2019, in which they implemented the algorithm from AlphaZero to learn Connect Four. The implementation design consisted of a neural network which takes the board positions as input and gives a probabilistic policy as well as the value, representing who has the advantage, of the input position as output. The experiment used two programs, one with supervised initialization and one without. It was found that having supervised data significantly helped the training time. However, the margin in win rate was not so noticeable.

Another useful contribution to my research is the work of Marius Bourcan[vi]. The idea behind his work is to let two random players play against each other for many games. Each move in each game will be recorded and then used as the input data for the training phase of the neural network. The neural network is able to play the game and make moves based on the prediction of the probability that a certain move will make the player win. The size of network is pretty small because this is a relatively simple problem to solve. The number of input neurons should be representative of the game board which means it will be typically 42 neurons (6 rows * 7 columns). The number of output neurons is only 3, and they represent the probability of the player winning, probability of a draw and the probability of the opponent winning. By calculating the probability of winning for each possible move, the model decides the move it will make based on the highest chance of winning. As for the hidden layers, Marius Bourcan found that 2 dense layers of size 42 each worked well. The neural network he developed achieved a 72% win rate as the red player (first player) and 46% win rate as the yellow player (second player).

2.3 Applications of genetic algorithms

Many different papers and experiments review the results of using a genetic algorithm to train a neural network. This concept has been widely used in various different fields: image classification[vii], permeability estimation[viii] or building design[ix]. The performance of a neural network highly depends on its architecture (how its parameters are set). If no one in a business has the expertise to understand how the problem can be converted into a neural network, then genetic algorithms are proven extremely useful to design optimal neural network architectures. In fact, the genetic algorithm is sometimes able to create an architecture which outperforms (or on par with) the already existing architectures for a given problem, whether that be in terms of accuracy of the model, or simply resources needed for the model. In addition, compared to other optimization methodologies, genetic algorithm proved itself to be considerably faster to optimize a neural network.

It is also important to note that genetic algorithms can be used to optimize a range of different problem solutions, not only neural networks. Some genetic algorithms have been used to optimize problem solutions in drug design[x], information retrieval[xi], facial recognition[xii] or bank lending decisions[xiii]. These types of problems often require a search on an extremely large domain space. By using genetic algorithms to generate new “partial solutions”, the algorithm does not need to search for every possibility but slowly builds towards the optimal solution by only selecting the solutions that fit best. In addition, compared to neural networks, genetics algorithms are robust as the optimal result will not be affected by small changes in the parameters. However, these should still be tuned with respect to the problem and are not the same for every optimization problem.

In each case, the measure of fitness is usually completely different, as it is linked to the problem at hand. The fitness function is defined by the user (along with parameters such as number of individuals per generations, or mutation rate) and the algorithm selects the individuals that best fit the fitness function. After the selection process is done, individuals are grouped in pairs based on their measure of fitness. Each pairs’ genomes are then combined to form new individuals, which gives birth to a new generation. Before the new generation is tested for fitness again, we usually decide to mutate some properties in their chromosomes. This allows the algorithm to randomly search for slightly different possibilities that might give rise to a high fitness. Once this process is complete, we can iteratively repeat until we achieve a specified level of fitness, or generations.

2.4 EXISTING SOLUTIONS

In the thesis of Robur Box[xiv], an evolutionary neural network learns to play Connect Four. The setup of his experiment is similar to the experiment I set out to complete in this paper but the implementation of it will be a different one. The neural network used has three different input layer versions: 42 neurons which simply takes the state of the board, 85 neurons which has 42 neurons for the board state and an additional 43 pre-processed information neurons which indicate whether four-in-a-row or three-in-a-row are present on the board, and 129 neurons which has the 85 previously mentioned neurons with additional pre-processed information neurons that indicate the presence of two-in-a-row on the board. It also makes used of the sigmoid function to introduce non-linearity. Robur's thesis also sets out to optimize the parameters of the evolutionary/genetic algorithm, he explains this program is based on nine parameters settings. These are: tournaments, selection, selected portion, population size, mutation, mutation rate, mutation probability, crossovers, and the size of the network. The results of his experiment have shown that the overall performance of genetic algorithms is equal or better than the Temporal Difference algorithm from Haan he was comparing with; showing that genetic algorithms can learn to play Connect Four.

Another similar experiment is tested in the thesis of Dylan Anthony Kordsmeier[xv]. In this case, instead of using a neural network, a simpler weight-based system is used. This consists of a two-dimensional array of weights, where the row index indicates how many red pieces are in a sequence and the column index indicates how many yellow pieces are in a sequence. At each turn, the algorithm needs to look at all possible sequence of four pieces, for every possible move that can be made. The two-dimensional array of weights is then used to determine the value of placing a piece in a board location. Hence, the computer then picks the move which has the most value. For the optimization phase, instead of having one population of weights playing both for the red and yellow side, the experiment was designed to have two population of weights, one for playing as red and the other for playing as yellow. Dylan's conclusion was that having two systems play against each other only lead them to compliment the strategy that the opponent deemed fit. The algorithm needs more variety to learn the different subtleties of the game. He also added that additional weights could be added to look several moves ahead which would lead to better strategy making.

3. SYSTEM DESIGN AND IMPLEMENTATION

Instead of creating my own code from start, I chose to use already existing codes found on the web. Having decided early on that I was going to use Python due its large and accessible library of machine learning code, I searched for existing codes on GitHub. The advantage of this is that I will spend less time trying to replicate code that is less efficient than ones I can find, and spend more time understanding them and how to apply them to my project.

I decided to base my experiment on the works of Marius Borcan and Ahmed Gad[xvi] [xvii]because they are very well written and documented. Together, they included almost everything I needed to fulfil the experiment's success criteria. However, before finding these source codes I came across other similar works which I tried to use but only ended up in errors. I believe this was occurring because of some deprecated functions which no longer worked on my Python version.

3.1 The Connect Four game and Neural Network model

According to the evaluation criteria established earlier (section 1.4 above), the first step to the experiment was to have a working implementation of the Connect Four game as well as a neural network trained by two random agents playing the game. This was achieved by replicating the project of Marius Borcan, Keras Neural Network learns how to play a game of Connect Four. The python project consists of 4 classes and a main script.

The Game class contains the playing boards data and functions used to change and evaluate the state of the game, such as checking whether or not a player has won by aligning 4 pieces in a row. It also offers all the available moves that can be made at the current state of the game so that we do not waste time checking if a move is legal or not. The *getBoardHistory* function returns the state of the game for every move made, this will be used as the data input for building the neural network.

The Player Class is made so that a player can have two different strategies. The first strategy is called random and simply plays by picking moves at random until the game ends. This strategy will be used to build a large amount of data from two random players facing each other many times and recording the board history for each game. The second strategy will be one using a Keras neural network model. By evaluating the state of the game after each possible move the model picks the move which returns the highest chance of winning according to its weights.

The Game Controller Class is used to control the flow of the game and collect the history of moves made by the players. The *simulateManyGames* function will be used to quickly get results from a batch of games between two players. The *playGame* function simply consists of a loop that lets the players make a move one after another from the list of available moves until the game is in end state (if either player wins or there is a draw).

The Model Class is initialized by creating a Keras neural network with two hidden dense layers of 42 neurons each. The *numberOfInputs* value passed as a parameter should be 42, as this represents the inputs returned from the *getTrainingHistory* function of the Game Controller Class. The *numberOfOutputs* value should be 3, this represents the three predictable outcomes of the game: red player wins, yellow player wins, and a draw. The *train* function takes the board history data from a batch of games as input and the result of this batch of games as output. These input and output values are split into a training set (80%) and a testing set (20%) in order to train the neural network. We also make use of the *predict* function which based on the state of a game, the model will predict the probability of winning.

The Main class will be used to create instances of players and games to allow our model to train and then play against a player with random strategy. This model will also be saved and then loaded in the second part of the project to be optimized by a genetic algorithm. Hence, the results of these games will serve as a benchmark and be compared against results obtained from the GA optimized model.

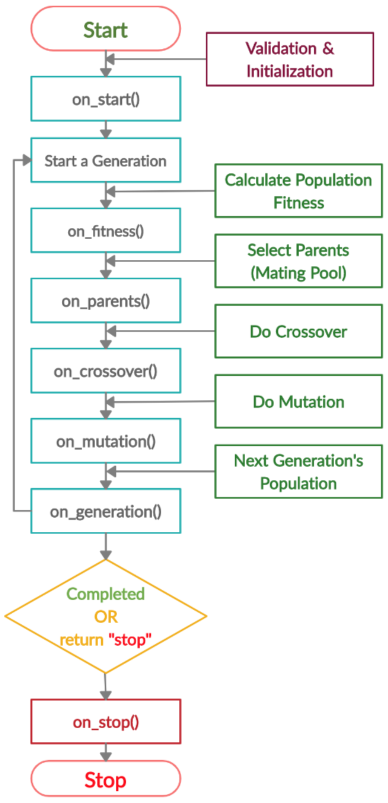
3.2 The PyGAD algorithm

In the next part, two different attempts at optimizing the neural network through a genetic algorithm have been made. The first relies on the PyGAD library made by Ahmed Gad, an open-source library used for building genetic algorithms and optimizing machine learning algorithms. It allows different types of problems to be optimized easily by customizing the fitness function and adapting it to the problem at hand. The fitness function is a critical part in the GA process, there is no way to calculate it and changes for every problem. This fitness function should be designed so that it is a maximization function, meaning that a solution/chromosome with a high fitness value is favoured compared to a low fitness value. Essentially this means that we should give the model a higher fitness value when it wins games and a lower one when it loses games. This function accepts two parameters, a 1D vector representing a solution (in our case, it consists of all the trainable weights in the model), and the index number of the solution among the whole population of solutions.

The first step in the fitness function is to reassemble the list of weights and biases from the 1D vector solution which holds all the values. These values are then used to update the neural network model which is then used to play two games against the benchmark model: one game playing as the red player and another playing as the yellow player. A score is given for both matches based on the result of the model; the fitness value is calculated by averaging these two scores. A win will yield a score of 100, a draw will yield a score of 23, and a loss will yield a variable score based on the length of the game. These values have been giving me the best results, it aims to reward successful solution rather than to punish bad solutions.

Some other important variables are to be considered before starting the optimization. These hyperparameter specify how the genetic algorithm will run: *num_generations* indicates how many times will the genetical algorithm loop, *num_parents_mating* sets how many solution will "reproduce" to create new solutions that will replace the unfit ones, *sol_per_pop* indicates the number of solution in each generation*, init_range_low* and *init_range_high* determines the genes initial values which should be between -1 and 1 as these correspond to our model weights, *crossover_type* indicates how two parent chromosome will be combined to form an offspring, *mutation_type* indicates how the offsprings gene will change to maintain diversity, and *mutation_percent_genes* indicates how many genes in the whole chromosome will be affected.

The next step is the genetic evolution of the model, this is handled automatically by PyGAD when calling the run function of the PyGAD instance. The algorithm goes through the following cycle:


Start
Validation & Initialization
on_start()
Start a Generation
Calculate Population Fitness
on_fitness()
Select Parents (Mating Pool)
on_parents()
Do Crossover
on_crossover()
Do Mutation
on_mutation()
Next Generation's Population
on_generation()
Completed OR return "stop"
on_stop()
Stop

At the end of each generation, the *callback_gen* function is called which prints the current generation number and the fitness achieved by the best solution.

When the genetic process is finished, we call the *plot_result* function which prints a graph of the fitness value over the generations.

3.3 The Matrix Multiplication algorithm

The second implementation follows the article written by Ahmed Gad, using Genetic Algorithm with Python. This method will not rely on creating and training a Keras neural network model to updates its weights. Instead it will directly perform matrix multiplication between the input matrix and the model's weight matrix to obtain the outcome value much faster. Since we want to work with matrices for the neural network but with a 1D vector for genetic algorithm, we need a function that will return the corresponding matrix for a vector of genes (*mat_to_vector*) and a function that will return the corresponding vector for a matrix of weights (*vector_to_mat*). I will not explain these in detail as this is done perfectly well in the article but the conversion of a vector to a matrix is possible because we already know the shape of the matrix we want to build.

The first step is to gather the data that will allow our "solutions" to be evaluated when making a series of prediction. This is done by initiating a game instance of Connect Four with both players using the Keras neural network model loaded from the previous part of the experiment. After using the *simulatedManyGames* function, a large amount of data has been recorded about the game history, which are then stored in the *data_inputs* and *data_outputs* variables for later use.

We then initiate the first population of solutions to be optimized. This is simply done by creating a matrix of weights which are randomly generated for each solution. Once our solutions are initialized, we can enter the generational process/loop. Firstly, the 1D solutions are converted in a matrix, and their fitness are calculated by using functions from the ANN class. The fitness function uses the *predict_outputs* function to get a value of accuracy. This is where matrix multiplication is used between the model weights and the data input in order to get predictions on the outcome of the games. At the end of this function, the fitness of a solution is given as a percentage of all correctly guessed outcomes of the data inputs.

Once the fitness is calculated, we will then use the GA class to perform changes on the population to improve their overall fitness. The *select_mating_pool* function will return only the fittest parents whose chromosomes will be used for the crossover stage. In our case, we have 8 solution per generation, and we chose to keep 4 parents per generation to create 4 offspring each new generation. The *crossover* function is made to create new offspring based on the genes of its parents. In this function a new offspring will have the first half of its genes taken from the first parent, and the second half of its genes taken from the second parent. Finally the *mutation* function is there to introduces some variety that would be impossible by just taking genes from previous generations. This function will randomly change a small percentage of genes from our set of newly created offspring.

Once we reach the last generation, we then print the accuracy of the best solution and create a graph of the evolution of fitness over the number of generations.

3.4 Summary of the experiment

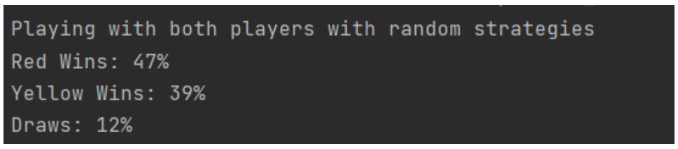
Using the three Python scripts explained above, the experiment will first consist of creating a neural network model trained using data from the history of games played by two random agents; this will act as a benchmark, of which the results and performance will be compared to the genetically optimized models. I will then attempt to optimize two neural network models: one using a genetic algorithm that measures fitness based on games played against the unoptimized neural network, and the other using a genetic algorithm that measures fitness based on outcome predictions from the history of games played by two unoptimized neural networks.

4. Results and Evaluation

In this section I will explain how I ran the experiment including hyperparameter choices, how long the training/optimizing of the models took, what results we achieved and how they relate to the success criteria.

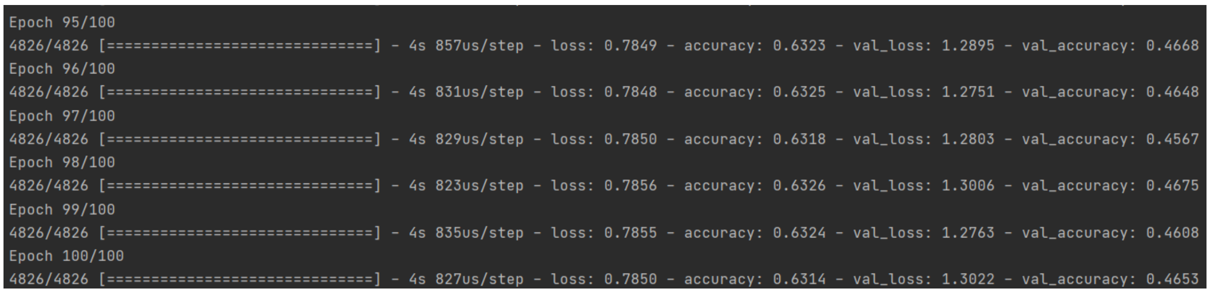
4.1 The Simple Neural Network Model

As explained previously, the benchmark neural network is made up of four layers: an input layer of size 42, two hidden dense layers with relu activation functions also of size 42 and an output layer of size 3 with SoftMax activation function. 10000 Games were simulated between the two random players and the history of these games was recorded making up to 482600 training samples which were fed as input for training our model. The overall results of these games can be seen in the screenshot below. It is normal to see a higher win rate from the red player since this is a turn-based game and they get to go first. We can also observe that a few of these games ended in a draw, which should give our model enough information about all three outcomes of the game.



```
Playing with both players with random strategies
Red Wins: 47%
Yellow Wins: 39%
Draws: 12%
```

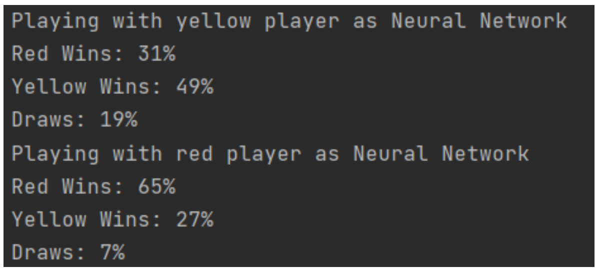
Getting this many data was quick and efficient because moves made from the random players do not require a lot of computation; thanks to this, acquiring unique data that I could use for training was done very quickly. During training, the batch size used was 50 and the number of epochs was 100. However it seems that after around 90 epochs, both the loss and validation loss stop getting smaller which means the model is not learning anything useful from the data and could be starting to overfit it.



```
Epoch 95/100
4826/4826 [==============================] - 4s 857us/step - loss: 0.7849 - accuracy: 0.6323 - val_loss: 1.2895 - val_accuracy: 0.4668
Epoch 96/100
4826/4826 [==============================] - 4s 831us/step - loss: 0.7848 - accuracy: 0.6325 - val_loss: 1.2751 - val_accuracy: 0.4648
Epoch 97/100
4826/4826 [==============================] - 4s 829us/step - loss: 0.7850 - accuracy: 0.6318 - val_loss: 1.2803 - val_accuracy: 0.4567
Epoch 98/100
4826/4826 [==============================] - 4s 823us/step - loss: 0.7856 - accuracy: 0.6326 - val_loss: 1.3006 - val_accuracy: 0.4675
Epoch 99/100
4826/4826 [==============================] - 4s 835us/step - loss: 0.7855 - accuracy: 0.6324 - val_loss: 1.2763 - val_accuracy: 0.4608
Epoch 100/100
4826/4826 [==============================] - 4s 827us/step - loss: 0.7850 - accuracy: 0.6314 - val_loss: 1.3022 - val_accuracy: 0.4653
```

We can see that the validation accuracy is a lot lower than the accuracy, this means that the model does not perform very well on unseen data. I believe that an accuracy of 63% should be good enough as a benchmark; there is obviously room for improvement which will hopefully be achieved with the use of our genetic algorithms later on.

In order to properly test our newly created model, I made it play 200 games against the random player: 100 games as the yellow player and 100 games as the red player. We can see the overall results of these games below.

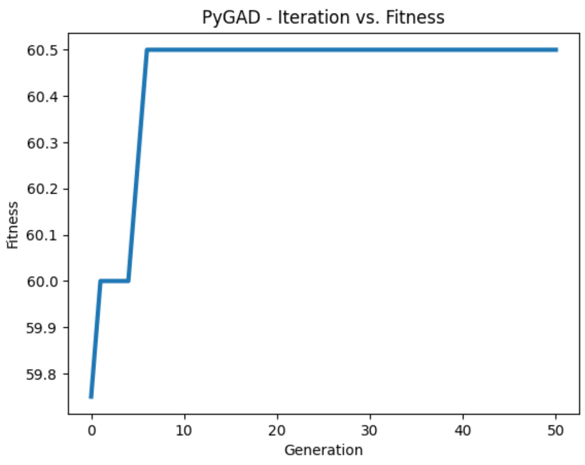


```
Playing with yellow player as Neural Network
Red Wins: 31%
Yellow Wins: 49%
Draws: 19%
Playing with red player as Neural Network
Red Wins: 65%
Yellow Wins: 27%
Draws: 7%
```

Playing as yellow, the neural network was still able to win almost 50% of games, this shows that despite the advantage given to red, the model is able to pick moves that will lead it to win before its opponent. When playing as red, we can see that win rate increases and the draw rate decreases, which implies that the model is making better use of its advantage.

4.2 The First GA optimized model

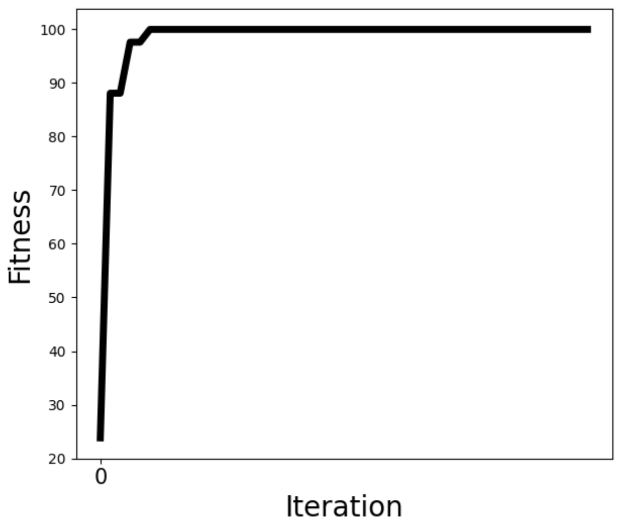
In order to have a fair comparison across our different models result, each neural network will have the same structure: 42 input neurons, 2 hidden layers of 42 neurons and 3 output neurons. The population of models were optimized over 50 generations. The graph below shows the fitness of the best solution over the generations.



We can see that the fitness value starts at 59.8 for the first generation. This means that when the fitness of a solution is being evaluated, the model is able to win one game and lose the other one (after about 39 moves). In the next few generations the fitness of the model goes up to 60, which shows that the second game is still being lost but later in the game (after 40 moves). Finally the model's fitness increases to 60.5 before the 10th generation. After this improvement the model loses the second game after 41 moves; considering only 42 moves can be played in a game, this shows how close the genetically optimized model is to winning both games against the unoptimized model. As you may notice, continuing the genetic algorithm after the 10th generation did not result in any further improvements in the games played. However, sometimes during this genetic process, it may take many generations before an observable change occurs, which is why I wanted to continue optimization for 50 generations, but in this case it did not happen.

4.3 THE SECOND GA OPTIMIZED MODEL

Moving on to the second genetic algorithm, the data that was used to evaluate the fitness in the *predict_outputs* function was generated by simulating 1000 games between two instances of the unoptimized neural network player. This resulted in 42000 data samples that we use to calculate the model's fitness based on the number of correct predictions on this data. The population of models were optimized over 500 generations. The graph below shows the fitness of the best solution over the generations.



The fitness value starts off small at 23.8 but escalates rapidly in the first generational change to 88. This will allow for the next offspring to already have a very good gene pool, which will therefore lead to the whole population having a high fitness within the early generations. The first time an individual received a fitness value of 100 was in the fourth generation. As we can see from the console, every individual in the population already achieves a fitness value over 88 at this point.

```
Generation :  4
Fitness
[ 97.61904762  97.61904762  92.85714286  88.0952381  100.
  92.85714286  97.61904762  97.61904762]
```

It was only after 14 generations that every individual reached the maximum fitness of 100.

```
Generation :  14
Fitness
[100. 100. 100. 100. 100. 100. 100. 100.]
```

After this point, improvements can still be made to the gene pool of each solution however this will not be noticed because the models are correctly predicting the result of every data sample already. At the end of the genetical algorithm, the weight values of each individual were almost identical to a few digits after the decimal point. This means that there seem to be only one set of weight values that is optimal to predict the result of a Connect Four game. The optimization of the models through the genetical process has led all possible solutions to this set of weight value, rather than another.

4.4 Summary of the evaluation

Overall, the results obtained from this experiment are mostly positive and give us a good idea of how to answer the research question. A working implementation of the Connect Four game is present, although it could be improved with UI and screen that shows the board state. The simple neural network trained from two random agents was able to train and play the game successfully. The first model to be optimized by genetic algorithm could have performed better, I believe that the fitness function could do with some changes in both efficiency and the way it attributes the fitness value, which is a bit simplified and relies too much on the outcome of one game. The second model however was an improvement, I did not expect it to have such a high accuracy from the early generations. It seemed that evaluating the model from a large number of predictions is the way to go in terms of optimization.

5. Conclusion

5.1 Project Evaluation

My personal goal for this project was to deepen my understanding and interest in the area of machine learning that is inspired by nature. I found it extremely challenging but at the same time rewarding to conduct the experiment but also research all the information that I needed to do so.

In order to answer my research question I had set out to attain as many aims and objectives as possible that were set in the beginning of the report in an effort to have a good base of information from which to base my evaluation. The success criteria, on the other hand, allows me to evaluate exactly what are the effects of the experiment. Two aims and objectives were eventually not reached: “Compare how both models perform when playing against each other”, “Compare how both models perform when playing against random agent”. However, having completed all of the other aims/objectives, I still had enough results and evidence to answer the research question.

The success criterion: “A higher interpretability and understanding of the GA trained model over the non-GA trained model”, was not reached as the experiment did not give us any insight on how the model operates on a neuron level. Another success criterion was not proven as anticipated: “A shorter training time for the GA trained model over the non-GA trained model”. For both the GA optimized models, the process of building the solution generation after generation was actually much longer than gathering data from series of games and using that to train the neural network. For both genetic algorithms, the evolutionary process of 50 generations took over 10 hours.

That being said, from the implementation explained in section 3 and the evaluation of results in section 4, we can observe that every other aims/objectives and success criteria was met during this project. From this information, the question I set out to answer in this paper: *What are the effects of using genetic algorithms to optimize a neural network to play Connect Four?* Can therefore be answered. By optimizing a neural network to play Connect Four using a genetic algorithm, we can expect the neural network model to have a better chance at winning than if it wasn’t optimized, however this very well depends on the optimization techniques used in the algorithms, such as how the fitness function is defined or how is the model represented as a chromosome with a set of genes. We can also note that this optimization comes at the cost of longer training time for the weights of the model.

5.2 Future Work

Considering that all the aims/objectives and success criteria were not all met throughout this project, further development could be made to attain all the aims, objectives and success criteria to the experiment in order to gather an exhaustive set of information to fulfil these. This may include a graphical representation of the neural network model at different stages of the optimization process to allow us to observe how the weights of the model are updated, and which of these turn out to be useful. Another important step that the experiment could benefit from is a more thorough comparison between the various models. This is achievable by using the trained weights from each model to create a Connect Four player, and simulating games between the players to get more statistical data about which model performs better against another. On top of this, a graphical display of the game board would allow us to easily see the decisions and strategy behind the model's moves.

This project was focusing on the game of Connect Four to observe the applications of genetic algorithms to neural networks. But as explained in section 2, this optimization can be used in many areas. The evolutionary process that are genetical algorithms are useful to rapidly form a solution in a search space that is wide and hard to search exhaustively. I believe that any problem possessing such a large search space could benefit from an application of genetic algorithms to find the fittest parameters to the solution.

6. Statement of Ethics

Even though this project does not have any heavy implications legally or ethically, it is important to discuss how the project considers some ethical, legal, social, and professional principles that are often encountered in the computer science department.

The first ethical principle to consider is that the project does no harm. This means that it should avoid exposing the public to additional risk that may harm them. It also ensures, when conducting research on sensitive subjects, that we respect the legality, confidentiality, and privacy of the public, taking special precautions if necessary. The Computer Misuse Act (CMA) was introduced to protect data from unauthorised access. To complete this project, research was done by accessing various resources online however each of them was accessed in accordance with the ethics enforced by the CMA. Every idea or code in my project that was not my work has been referenced and talked about in the report. In addition any research paper, article or code library that referenced or used was never accessed without the required permissions.

In the case that a project demands the involvement of a human person, an informed consent should be requested from the participant. He/she should be informed prior to the involvement on the purpose of it, what they should be expected to do, and any consequences that may come out of it. The key here is that the participants fully understand the conditions and are willing to be involved. This project does not involve any human participants in any way, as the only thing being tested is our machine learning model. Therefore, this principle is not being infringed.

Another important principle is based on the confidentiality of data, which is about protecting personal data against malicious, unauthorized, or accidental access. Confidentiality of data ensures that information is private and remains so unless enough clearance of authority is exerted. The Data Protection Act (DPA) was developed to control how personal information is processed by organisations. It states that everyone responsible for using personal data should follow strict rules called 'data protection principles' to make sure information is :used fairly, lawfully and transparently; used for specified, explicit purposes; used in a way that is adequate, relevant and limited to only what is necessary; accurate and, where necessary, kept up to date; kept for no longer than is necessary; handled in a way that ensures appropriate security, including protection against unlawful or unauthorised processing, access, loss, destruction or damage. Similarly to the previous principle, my project does not process or use in any way personal data, meaning that this principle too is being respected.

The final principle I will consider for this project is the contribution to society and human well-being. This principle, rather than focusing solely on legal matters, concerns the quality of life. It ensures an obligation of professionals to use their skills for the benefit of human society, its members and the environment surrounding it. This principle aims to minimize the negative effects of computing systems including threats to health, safety, personal security, and privacy. The outcome of this project does not in any way shape or form, present a negative effect to any of these. On the contrary, one of the background aims of this project is to improve the positive effects of computing systems. Also, I very well doubt that any of the results from this experiment can be mis-used to harm society.

7. References

[i] Weisstein, Eric. (2003). A090224 Number of possible positions for n discs on a standard 7 X 6 board of Connect-Four. [online] Available at: https://oeis.org/A090224 [Accessed 9 Nov. 2020].

[ii] James Dow Allen (2010). The complete book of Connect 4 : history, strategy, puzzles. New York: Sterling ; Lewes.

[iii] Allis, V. (1988). A Knowledge-Based Approach of Connect Four. ICGA Journal, 11(4), pp.165–165.

[iv] Tromp, J. (1995). John’s Connect Four Playground. [online] Available at: http://tromp.github.io/c4/c4.html [Accessed 8 Aug. 2021].

[v] Kim, L., Godrej, H. and Nguyen, C. (2019). Applying Machine Learning to Connect Four. [online] . Available at: http://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26646701.pdf [Accessed 9 Aug. 2021].

[vi] Borcan, M. (2020). Artificial Neural Network Learns To Play Connect Four. [online] Available at: https://programmerbackpack.com/artificial-neural-network-learns-to-play-connect-four/ [Accessed 9 Aug. 2021].

[vii] Sun, Y., Xue, B., Zhang, M., Yen, G.G. and Lv, J. (2020). Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification. IEEE Transactions on Cybernetics, [online] 50(9), pp.3840–3854. Available at: https://ieeexplore.ieee.org/abstract/document/9075201.

[viii] Saemi, M., Ahmadi, M. and Varjani, A.Y. (2007). Design of neural networks using genetic algorithm for the permeability estimation of the reservoir. Journal of Petroleum Science and Engineering, [online] 59(1), pp.97–105. Available at: https://www.sciencedirect.com/science/article/pii/S0920410507000472 [Accessed 9 Nov. 2020].

[ix] Magnier, L. and Haghighat, F. (2010). Multiobjective optimization of building design using TRNSYS simulations, genetic algorithm, and Artificial Neural Network. Building and Environment, 45(3), pp.739–746.

[x] Douguet, D., Thoreau, E. and Grassy, G. (2000). Journal of Computer-Aided Molecular Design, 14(5), pp.449–466.

[xi] Sivaram, M., Batri, K., Mohammed, A.S. and Porkodi, V. (2019). Exploiting the Local Optima in Genetic Algorithm using Tabu Search. Indian Journal of Science and Technology, 12(01), pp.1–13.

[xii] Zhi, H. and Liu, S. (2019). Face recognition based on genetic algorithm. Journal of Visual Communication and Image Representation, [online] 58, pp.495–502. Available at: https://www.sciencedirect.com/science/article/pii/S1047320318303389 [Accessed 9 Nov. 2020].

[xiii] Metawa, N., Hassan, M.K. and Elhoseny, M. (2017). Genetic algorithm based model for optimizing bank lending decisions. Expert Systems with Applications, [online] 80, pp.75–82. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0957417417301677 [Accessed 30 Mar. 2020].

[xiv]Box, R. (2015). Learning to play Connect-Four using Evolutionary Neural Networks. [online] . Available at: https://fse.studenttheses.ub.rug.nl/12645/1/AI-BA-2015-RoburBox.pdf [Accessed 9 Aug. 2021].

[xv] Kordsmeier, D. (2015). Using Genetic Learning in Weight-Based Game AI. [online] . Available at: https://core.ac.uk/download/pdf/72841392.pdf.

[xvi] Gad, A. (2020). pygad Module — PyGAD 2.16.0 documentation. [online] Available at: https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#initialize-population [Accessed 9 Aug. 2021].

[xvii] Gad, A. (2019). Artificial Neural Networks Optimization using Genetic Algorithm with Python. [online] Medium. Available at: https://towardsdatascience.com/artificial-neural-networks-optimization-using-genetic-algorithm-with-python-1fe8ed17733e.