Learning to play Connect-Four using Evolutionary Neural Networks
(Bachelorproject)

Robur Box, s2180952, Roburbox@gmail.com,
Marco Wiering [*]

May 18, 2015

Abstract

This thesis describes the machine learning technique Evolutionary Neural Networks applied to learning to play connect-four. It introduces the concept of Evolution and Genetic Algorithms, and explains the process of optimizing neural networks. Genetic Algorithms can be used in many different ways, so the design decisions and parameter optimization plays a large role. A total of 941 hours of (single core) processing time was dedicated to evolving neural networks with various different evolution settings. These settings include, amongst others; population size, mutation rate, crossovers, etc. The thesis concludes that Evolutionary Neural Networks are suited to learn to play connect-four, compared to earlier research of Neural-Fitted Temporal Difference Learning.

1 Introduction

In this thesis the following research question will be answered:
Do evolutionary neural networks lend themselves to learning to play connect-four?
In order to answer this question, we must look at what evolutionary neural networks are, and how they can be implemented. In this paper we describe, after having established the best evolution setup (population size, mutation rate, etc), an analysis of the reliability and repeatability of the best setups. This allows for an informed conclusion, answering the research question.

[*]University of Groningen, Department of Artificial Intelligence

But first let's start by looking at what genetic algorithms are.

1.1 Genetic Algorithms

In the field of Artificial Intelligence, Genetic Algorithms (GAs) are a heuristic search procedure based on the concepts of natural/biological evolution (Booker, Goldberg, and Holland (1989)). GAs consist of a cycle of steps in which a population of chromosomes undergoes improvements. Each chromosome in the population is a representation of a solution to the problem. At first the "solutions" are expected to be very poor but they improve over the generations. The longer a GA is able to evolve its gene pool (population of chromosomes), the more it will move to its optimum. Each generation recombines and/or mutates the better portion of the population in the hope of creating even better chromosomes. The process that determines which chromosomes will make offspring is referred to as natural selection. In order to be selected the individual has to survive until it can mate. This creates selection pressure on all individuals (chromosomes) in the population, resulting in a competitive environment. Separating the best and procreating them to form new and sometimes better chromosomes is what GAs derive their results from.

1.2 Reinforcement Learning

Reinforcement learning (Sutton and Barto (1998)) relies on obtaining feedback on the outcome of actions, and subsequently adjusting the decision making rules for future actions. When using evolutionary algorithms the feedback takes the form of performance of an individual. This performance is measured through the use of tournaments where individuals are measured against each other or another agent. Feedback is received in the form of a reward, and the accumulation of rewards is what represents the total fitness of an individual. The reward structure of the game is as follows; the agent receives one point for winning, half a point for a draw, and no points for losing. The agent will need to play several games in order to accumulate enough feedback in order for the evolutionary algorithm to make accurate assessments of the agent.

1.3 The Game

Connect-four is a two player zero-sum game, played on a board consisting of seven columns and six rows. Each player takes turns making his/her/its move until a terminal state is achieved. One move consists of choosing a non-full column in which a stone (or piece) is dropped. After at least seven moves and maximally 42 moves, one of two terminal conditions will be met. The winning condition consists of four connected stones of the same color. Four connected pieces may be positioned horizontally, vertically, or diagonally, and can only consist of one color. The draw condition is met when all columns are filled up and no connect-four is attained.

1.4 Neural Networks

Now that we know what game will be played, let's meet the player: the Neural Network. An elegant description of neural networks was made by Kevin Gurney:
"A neural network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron." Gurney (1997).
The neural network derives its power and usefulness from its ability to give a simple and clear response to a highly dimensional input. The network is able to do this through the use of layers of neurons connected in a specific way, and with specific weights between the neurons. When a stimulus (input) is offered, the network will pass this stimulus through its connections, altering the information using the weights. But before the network can do this, it needs to be given the correct weights and connections. The process which is able to optimize the neural network's response versus the desired response is what makes neural networks a machine learning algorithm. This optimization can be done using several different methods but in this paper we focus on evolutionary algorithms to do this.

1.5 The Background of this Research

Over the years the reinforcement learning technique known as Temporal Difference learning (TD) has been studied in depth (Sutton and Barto (1998) chapter 6). Success has been achieved using machine learning and neural networks, for example Wiering (2010), and van den Dries and Wiering (2012).
In a bachelor's thesis written by de Haan (2013) TD learning was applied to connect-four using neural networks. Some similar implementations will be made in this thesis. These include, for example, the neural network used to interpret the board state. Other than that the majority of this thesis is very different. Since the implementations around the machine learning (e.g. the neural network) are essentially the same, the machine learning algorithm can very well be compared. This way the research question can be answered with reference to this earlier work.

2 Methods

2.1 The Neural Network

The neural network takes the role of function approximator, and is situated between the game (connect-four) and the evolutionary algorithm. It is a fully connected feed-forward network, identical to de Haan (2013)'s implementation. This network type can be found in Gurney (1997) page 66. The network takes the board state as input, and depending on the network type additional higher level information is offered. The network then computes the output value, which represents the network's "opinion" on the board state. The higher the number, the better the board state. The network is used by trying all possible moves and subsequently choosing the action the network deems best. The input layer of the network can consist of three versions; 42 nodes, 85 nodes, and 129 nodes. The 42 node version is purely the board state as is (6x7 board cells). The 85 node version is based on the implementation by de Haan (2013) which has the same 42 input neurons as before, with the addition of 43 pre-processed information neurons. The 43rd neuron contains information of whether any four-in-a-row is present on the board. The remaining neurons represent the presence of a three-in-a-row on certain columns and rows, and how the three-in-a-row is directed (horizontal, vertical, etc). Finally the last option consists of 129 neurons, which contains the 85 neurons mentioned previously with the addition of information about two-in-a-rows, with their location and direction. The last two nodes (128 and 129) represent the sum of possible winning moves of both players. The neural network makes use of a sigmoid function to introduce non-linearity to an otherwise linear combination of inputs.

2.2 Evolving the Neural Network

Now that it has been established how the neural network looks and operates, the evolutionary part of the neural network will be explained. The weights of the network are arranged in an array (the chromosome), starting with the input-to-hidden weights, ending with the hidden-to-output weights. Adding multiple of these chromosomes together forms the population, which is then evolved using selection, mutation, crossovers, etc. The length of a chromosome is dependent on the number of input units and hidden units, and can vary between 44 weights and 6550 weights in extreme cases.

2.3 The Evolution Parameters

Evolutionary algorithms naturally have many parameters which need to be optimized. This section will explain the methods by which the evolutionary algorithm is implemented and how each part is examined for optimization. The program is based on nine parameter settings, which are subject to optimization. Additionally, a few internal settings are used to represent some of the design decisions. The external parameters will now be explained.

2.3.1 Tournaments

A tournament is a process of ranking the individuals of a population according to their performance. Tournaments usually consist of several smaller competitions between smaller populations. The role of the tournament is to organize the population into rankings with as few competitions as possible.

Three tournament types were implemented from existing sources. First off the ***Random Pairing Tournament***, based on Jacobs (2008) paper. In this tournament an individual from the population is paired with another individual, and the winner gets more points than the loser. The second tournament is the ***Round Robin*** (or *all-play-all*) tournament based on the implementation by Lucas, Samothrakis, Runarsson, and Robles (2012). This tournament pairs up every individual with every other individual. Because of its "brute force" nature, the Round Robin tournament is the slowest and most accurate tournament.

The third tournament is the ***Random Agent Tournament***. In this tournament the individuals are paired up against semi-random moving agents. The random agents will make the winning move and block a winning move if possible, otherwise they move randomly. This tournament is unique in that the opponent is fixed and does not change over the generations.

2.3.2 Selection

The process of selection is closely intertwined with the tournaments, and can be seen as one process. Selection determines which individuals in the population should procreate and live to the next generation, based on the results of the tournament. The number of individuals chosen is determined with the "Selection proportion" parameter in section 2.3.3. Two distinct selection procedures were used in the implementation:

Fitness Proportionate Selection, also known as *Roulette Wheel Selection* (Goldberg (2012) pages 11 and 237).

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.1}$$

Formula 2.1 states that the selection probability (p_i) is equal to the fitness of the individual (f_i) relative to the fitness of the population size (N). This

method is well suited when tournament scores differ significantly; or when certain individuals score significantly better than the rest.
Truncated Selection dismisses the numerical fitness scores and replaces them with the ranking of the individuals. Each individual is given a ranking based on its fitness score (e.g. 1, 3, 4, ... 20) and the top ranking individuals are selected (according to section 2.3.3). Even if individuals vary in fitness scores, their position in the ranking is always fixed. This method is preferred over fitness proportionate selection when the tournament scores do not diverge much. However this method does not give the bottom portion of the population any opportunity, even if the top portion is only slightly better.

2.3.3 Selected proportion

The selected proportion parameter represents the percentage of the population that is selected by the selection procedure. In truncation selection, for example, this parameter represents the truncation threshold. The individuals which are not selected are removed from the population and replaced by the offspring of the selected individuals.

2.3.4 Population Size

The population consists of a predetermined number of individuals, ranging from 2-200. Some individuals are newly added, and some have been removed, but the population size is always consistent. Having a large population size is rarely detrimental to the quality of evolution, though it can slow down the simulation. This is especially true when using tournaments like Round Robin. The risk of having a small population size is limiting the diversity of the number of solutions found by the algorithm.

2.3.5 Mutation

Mutation is the process of slightly changing the chromosome in the hope of creating a better one. The chance of creating an improvement reduces over time, as the population becomes more fit. In this implementation we separated mutation into two parameters: mutation rate and mutation probability:

2.3.6 Mutation rate

Every mutation changes the value of the chromosome point by a certain value. This implementation uses Gaussian Mutation, which favors small changes over large changes. The mutation rate parameter represents the standard deviation of the Gaussian distribution from which the random variable is derived. The idea of using Gaussian Mutation instead of a normal linear random number generator is that the non mutated version of the chromosome is already quite good because it survived this far. This method searches near this value rather than a distant value. The implementation is given by formula 2.2, where x is the input chromosome, x' is the mutated result, and m_r is the value of the mutation rate parameter, where i is the i^{th} neural network weight in the chromosome.

$$x'_i = x_i + N(0, m_r) \tag{2.2}$$

2.3.7 Mutation Probability

The mutation rate parameter described above will mutate along the entire chromosome. The problem with this is that sometimes the mutations bring too much change, and it may be more favorable to have less mutations, and thus the mutation probability was introduced. This parameter represents the chance of any point on the chromosome to receive a mutation. Since the parameter can be set to 100% chance to mutate, some of the trials are run without this parameter; the results will show whether this parameter was useful or not.

2.3.8 Crossovers

By the process of crossing over (recombination) the newly generated chromosome receives parts of two chromosomes. Recombination was implemented through the N-point method (Goldberg (2012) pages 119-120). In this method, N points along the new chromosome are considered, and at each point this chromosome switches to the other parent for the receiving of information. The version of the N-point crossovers method creates N evenly spaced points along the chromosome. The parameter took on values in the range of 0 to 20, where 0 means no crossovers were made.

2.3.9 Input size

The input layer of the neural network can be set to different sizes; each corresponding with the 42, 85, or 129 node input layer version. See section 2.1 for more information.

2.3.10 Hidden size

The hidden layer consists of a variable number of neurons, ranging from 1 to 50.

2.4 Internal design decisions

The before mentioned internal parameters are hard-coded and thus consistent for all trials. First off all randomly moving agents will always do the winning or blocking move. The knowledge and insight required to make this move is very basic and natural, any human player would know this right from the start.
Secondly any simulated game starts with a random first mover. This is to remove the effects of the first mover advantage or disadvantage. Finally, none of the contestants will be paired up with themselves.

2.5 The Parameter Search Procedure

The aim of the experiment is to find the best set of parameters, and is split up into separate trials. In each trial one parameter set is tested. In early testing, up to 1000 epochs (generations) were used for each trial. It was noted that bad parameters give bad results from the start. The good parameters consistently show steep initial curves in the first 50 or so epochs. For these two reasons 50 epochs are used per trial.

2.5.1 Single Parameter Variation

Before deciding to use Random Search, this method was used to set the values of the parameters. Each parameter was determined by looping through a predetermined range of values. This results in evenly scattered "measurements" across the nine dimensions (parameters). This would then be plotted in order to see how the fitness landscape looks for every individual parameter. The problem with this method was that it proved hard to find the correct range and incrementation size.

2.5.2 Random Search

An improvement upon this method was the random search method. This procedure takes random values for each parameter within a predetermined range. This results in an evenly scattered set of samples for each parameter. The advantage of this method is that the search is always in a complete state, but a more detailed (more sampled) result can always be made by continuing the search.

2.6 The opponent of the game

Thus far we have covered the game playing ability of the learning agent, and how it is improved through the use of GAs. In each epoch, which starts with tournaments and ends with selection, the progress of the GA is measured with the testing procedure. This procedure matches all the individuals of the population versus the advanced randomly-moving agent. This agent makes random moves until it encounters a winning move, by either player, in which case it always makes the winning or blocking move. This agent was also used by de Haan (2013), and has proven to be quite difficult to beat. The behavior of this advanced randomly-moving agent is similar to an experienced player with no strategy. Playing equal versus this opponent (50% winrate) means one needs to learn to make the winning and blocking move. Beating this opponent requires the addition of learning strategical gameplay (>50% winrate).

2.6.1 Complexity of the simulation

While running the simulation a large amount of calculations are made. To give a perspective into the complexity of the simulation, a few details were computed:

- The average chromosome represents a neural network of 85 input units, 25 hidden units, and 1 output unit, resulting in 85*25+25+25 = 2175 weights. The largest chromosome consists of 6550 weights, and the smallest 44 weights.
- The average population consists of 100 of these chromosomes.
- An average epoch (generation) using the Random Agent Tournament consists of 50

(matches) * 100 = 5000 games of connect-four played.

- The average simulation also performs 50 test-games versus the Random Agent per chromosome in earlier trials. In order to save processing time, in the final trials only the top 5 chromosomes of the tournament are tested. This testing is independent of the tournament, and is purely to show the progress of the evolution.
- Every trial consists of 50 epochs. The average number of connect-four tournament games played per trial is 250,000. The total number of testing games is 250.
- A total of 814 trials was run for a total of 203.5 million connect-four games, and over 200,000 testing games. Figures 1 through 8 in the results section compile the results from these 814 trials.
- A trial can consist of at least 5000 tournament games, and 1.9 million at most.
- From the 814 trials the best ten were separated for further prospecting. Since these ten results were only given 50 epochs to perform their evolution in, they should now be given the opportunity to fully evolve. These ten parameter sets vary broadly in the amount of processing time required. In reality some take much longer per epoch; in the order of 20 times longer.
- A total of 175 hours of (single core) processor time was dedicated to this random search method.
- A further 766 hours of (single core) processor time was dedicated to further testing with the top 10 parameters and the number 1 parameter in the results section.

3 Results

3.1 Random-Search Results

The random search method in section 2.5.2 of this paper has resulted in hundreds of trials, of which the full results are shown in figures 1 through 8. The y-axis of the plots are labelled "Performance score" which refers to the tail-end (last 5 generations) average score of the top five individuals. Using the tail-end eliminates most distortions by fluctuations, if present, and the top five individuals are a more accurate representation of what the population has to offer. The red line in the plots represents the data using a self-made smoothing function. This function takes the average value of all data points in the neighborhood.

3.1.1 Observations of the Evolution Parameters

This section gives a general reflection of how the evolution parameters perform. In results section 3.2 a more detailed description will be given with only the best parameters sets.
First off, in figure 1 (*Tournament and Selection*) we can see that the Random Agent Tournament has the overall best performance, followed by the Round Robin Tournament. Additionally, Fitness Selection yields higher scores than Truncated Selection, with the exception of the Random Pairing Tournament where they score equally well. From figure 7 we can see that the *NN Input* parameter clearly favors the more complex 129 input neurons over the 85 and 42 variants. This indicates that the additional information provided by the extra neurons has proven useful. As for the other seven parameters, none of them show clear cut tendencies or preferences. This shows that the individual parameters do not have any optimal range of values for themselves. What matters is the combination of all the parameters combined, there is no "sweet spot" for an individual parameter.
That being said, some of the parameters do show small tendencies towards higher or lower values:

- The *Selection Proportion* (figure 2) favors high values over low values.
- The *Population Size* (figure 3) performs slightly better with higher values over smaller values. There is a slight drop in performance below roughly 50-80 population size.
- The *Mutation Rate* (figure 4) favors lower values over high values.
- The *Mutation Probability* (figure 5) favors low values over high values on the broad scale.

- The *Crossovers* (figure 6) shows no tendencies.
- The *NN Hidden* (figure 8) favors low values over high values.

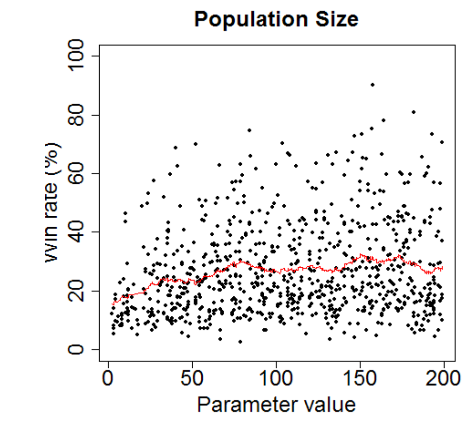


Figure 3: All 814 trials, Population Size

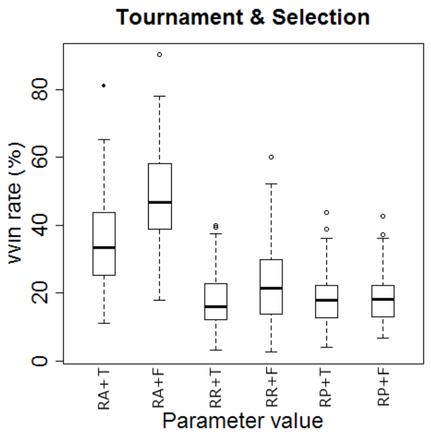


Figure 1: All 814 trials, Tournament & Selection
RA* = *Random Agent Tournament
RR* = *Round Robin
RP* = *Random Pairing
T* = *Truncated Selection
F* = *Fitness Selection

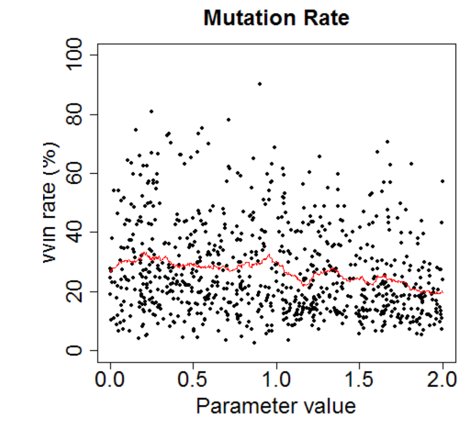


Figure 4: All 814 trials, Mutation Rate

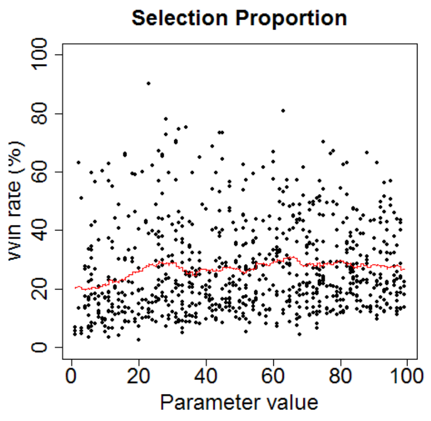


Figure 2: All 814 trials, Selection Proportion

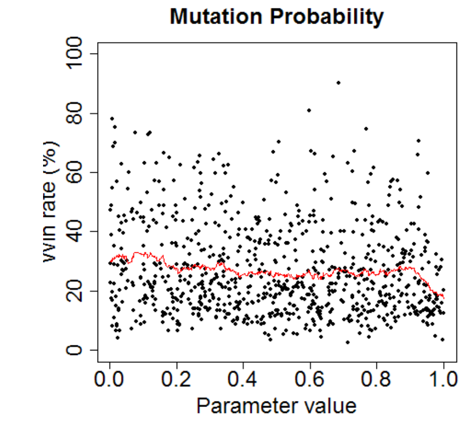


Figure 5: All 814 trials, Mutation Probability

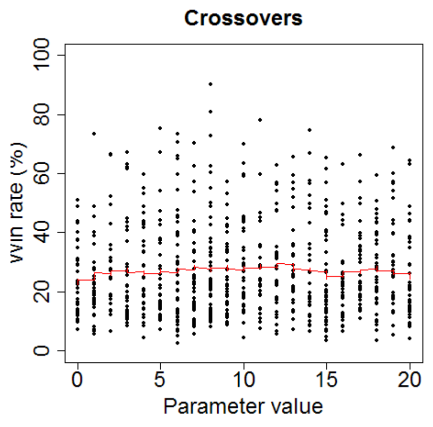


Figure 6: All 814 trials, Crossovers

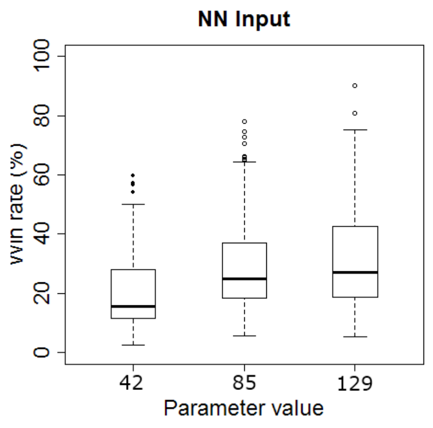


Figure 7: All 814 trials, NN Input

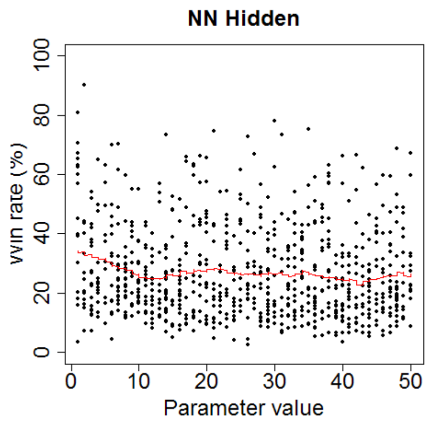


Figure 8: All 814 trials, NN Hidden

3.2 The top 10 parameters

The ultimate goal is to find the best parameter setup. The top ten parameters are separated from the rest and studied in more detail. The results are shown in figure 9.

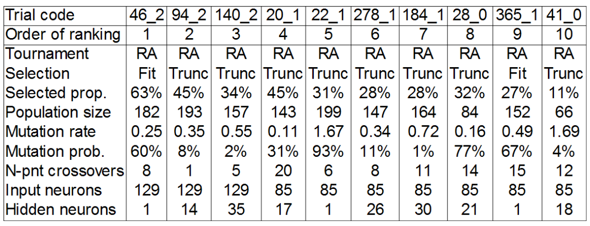
The tournament is unanimously the ***Random Agent Tournament***, with mostly a ***Truncated Selection***. The reason for this is likely that the Random Agent tournament is the only objective tournament type, and it remains exactly the same throughout the evolution. The other tournaments are based on a relative comparison, which changes as the individuals evolve. Since the testing phase consists of matches versus Random Agents, the results are to be expected.

The ***Selected Proportion*** appears to be mostly in the range of 30%, with a population size of around 150. This shows large population sizes are more prominent than smaller ones.

Some of the ***Mutation Probability*** have taken very low values, effectively not using mutations. These trials would then mostly rely on recombination. Of the trials which did use mutation, a mutation rate of roughly 0.5 was used. One prominent outlier; trial 22_1, has a 93% mutation probability and a very high (1.67) mutation rate.

Figure 9 shows that ***Recombination*** (crossovers) is deemed useful, as the best scoring parameters sets have values between 1 and 20 (not 0). A similar observation can be made about the ***Hidden*** layer where 3 of the 10 trials (including the best one) used 1 hidden neuron.

Finally, as was also noted in section 3.1, the 42 neuron networks do not make it to the top 10. The remaining 85 and 129 neuron networks are both represented in the top 10, but the 129 networks are slightly better.



Trial code	46_2	94_2	140_2	20_1	22_1	278_1	184_1	28_0	365_1	41_0
Order of ranking	1	2	3	4	5	6	7	8	9	10
Tournament	RA	RA	RA	RA	RA	RA	RA	RA	RA	RA
Selection	Fit	Trunc	Trunc	Trunc	Trunc	Trunc	Trunc	Trunc	Fit	Trunc
Selected prop.	63%	45%	34%	45%	31%	28%	28%	32%	27%	11%
Population size	182	193	157	143	199	147	164	84	152	66
Mutation rate	0.25	0.35	0.55	0.11	1.67	0.34	0.72	0.16	0.49	1.69
Mutation prob.	60%	8%	2%	31%	93%	11%	1%	77%	67%	4%
N-pnt crossovers	8	1	5	20	6	8	11	14	15	12
Input neurons	129	129	129	85	85	85	85	85	85	85
Hidden neurons	1	14	35	17	1	26	30	21	1	18

Figure 9: Table of top 10 parameters

3.2.1 Evolution in the long run

Now that some of the best parameters are identified it is time to see how they perform in the long run. Each setup was allowed to evolve for a duration of 5000 epochs, the results of which are shown in figure 10.

The learning curve is very steep at the start, reflecting the decision to have 50 epochs of run-time in the *random search*. After this short period of rapid learning, the curve starts to slow down. Some curves still fluctuate at the end, but are mostly stable in the long run, with the exception of a few curves that seem to keep gradually improving over time. The overall observation is that a thousand or so epochs results in most improvement, after which relatively little improvement is made.
It was noted by de Haan (2013) that in the Temporal Difference implementation the learning curve has a peak somewhere, and drops off after that. This phenomena is not observed in these findings. The final result achieved by the GA is a fluctuating but otherwise stable end result.

3.2.2 The reliability of the GA

In order to establish how reliable these results are, a final test is performed. The best trial (46_2) was repeated multiple times to see how reliably it achieves its final scores, and how consistent its learning curve is. Looking at figure 11 we can conclude approximately the same score is achieved each run with very similar curves. Especially in the last few epochs they all stabilize close together. Especially in the first hundred or so generations there seems to be much discrepancy. This is likely due to the randomness of the initialization. The effects of this will be analysed in further detail in the next section.

3.2.3 Random initialization search

Looking at figure 11 one may wonder whether the top scores can be achieved through a lucky initialization. In order to explore this option, a series of randomly initialized neural networks were tested. These networks are of the same structure as described in Methods section 2.1. Eight networks were initially constructed with either 42, 85, or 129 input units, and either 1, 32, or 64 hidden units. One of the networks was based on the findings in table 9, which consists of 129 input units and only 1 hidden unit. The weights are not optimized or changed after initialization, and are tested against the Random Agent (used in the Random Agent Tournament, and also used to test final scores). A total of 10,000 networks were initialized per network type (80,000 in total), of which only a few

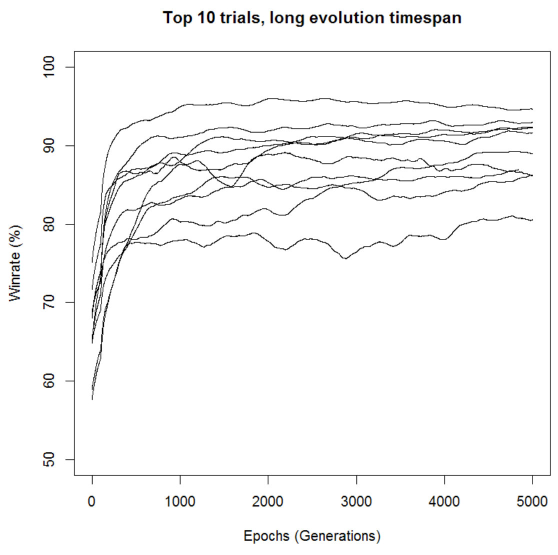


Figure 10: The top 10 trials with long runtime

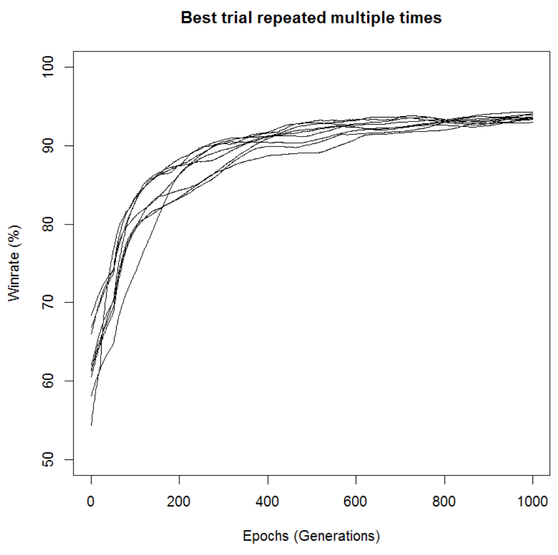


Figure 11: Best trial repeated 10 times

were able to achieve more than 60% winrate. The eight variations of networks resulted in similar frequency plots, the best one is shown in figure 12. The average initialization achieved 10% win rate over all seven networks. Taking note of the fact that the GA was able to get over 90% win rate, these results do not show the same potential. It is **very unlikely** for a random initialization to get above 90% win rate using this method.

4 Conclusion

In this paper we set out to answer the research question:
Do evolutionary neural networks lend themselves to learning to play connect-four?
The results of this paper have shown that the overall performance of GAs is equal to or better than the TD algorithm, showing that GAs can learn to play connect-four on a good level. The GA has achieved over 95% winrate versus the advanced random-moving agent. This indicates that the GA has not only learnt to execute winning and blocking moves, it has also learnt some form of strategy to beat the agent consistently.
This success did not come without effort, as a lot of work was spent trying to optimize the parameters of evolution (mutation rate, population size, etc.). Additionally, looking at the difference in win rate between the very top parameter sets and the rest, it is safe to say that the optimization is very important. Ultimately, getting the genetic algorithm to work will take some effort, possibly more than other alternatives.
Summarizing; GAs are good at learning to play connect-four after correctly optimizing the evolution parameters.

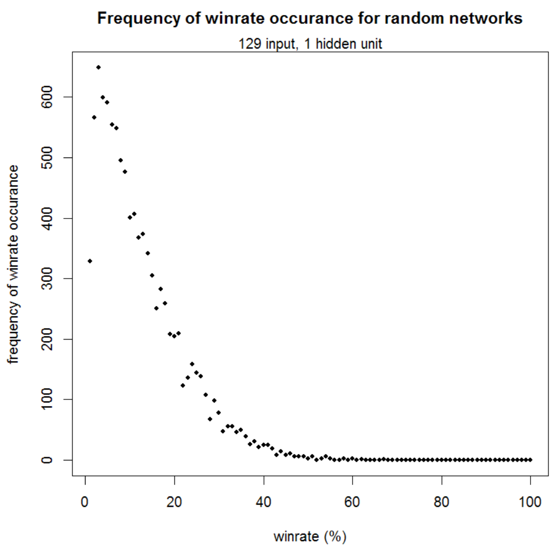


Figure 12: Randomly initialized networks without evolution

5 Discussion

In this paper the implementation of a genetic algorithm was described and analysed. Some variations in the evolution setup were individually tested, which resulted in several 80%+ win-rate chromosomes, and even a few 90%+ win-rate chromosomes. Further testing with longer runtimes and more generations revealed that even the best setups have strong variations in performance between them. This indicates that extending the duration of the random search procedure will result in more, and better, results. This thesis has also shown that the results achieved by the GA can be repeated successfully with little to no variations in the end results. This indicates that correct use of GAs can result in robust learning curves, and they are not dependent on their initialization for success.
The cheap alternative to the GA is the random network. The process of initializing and testing thousands of networks in rapid succession could lead to good findings. However the results indicate that the probability of achieving good results, comparable to the GA, is highly unlikely.

5.1 GA vs TD

This research has extended on the work done by de Haan (2013), with his thesis on Neural-Fitted Temporal Difference Learning on Connect-Four. Some of the methods used in this paper are applicable to that research, so that results can be compared. The similarities are in, for example, the implementation of the neural network. The difference was made with the machine learning technique.
How do the different neural network input layer

versions compare? Both the TD method and the GA method have evaluated themselves versus the advanced randomly moving agent. This agent will finish a winning move or perform a blocking move when possible, and otherwise it will make a random move. The two methods will be compared using this advanced randomly moving agent.
This paper revealed a win rate of 96% using the (more complicated) 129 neuron network, where the Temporal Difference variant has reported roughly 92% win rate versus the same opponent using the same 129 neuron network. Additionally the TD method achieved almost 90% win rate using the 85 network, and the GA has achieved almost 93% win rate using the same network. TD also used the Boltzmann exploration method, which yielded nearly 90% winrate. This shows the two methods have similar results, with the GA scoring slightly higher.
One should note that the GA has done over 800 random search trials in a relatively short period of time. Looking at figure 10 of the results section shows how varied the top 10 appears. Only the best one was considered in the comparison with the TD method. One can foresee that by running the GA's random search a couple thousand more times, the results will be better.

5.2 What is the best GA setup?

The Random Agent tournament has made the best results when compared to the Round Robin and Random Pairing tournaments. The method by which this tournament rates its individuals is an exact replica of how the performance score is measured (part 2.6 of the methods section). The other two tournaments were based on games between members of the population. In that case the fitness function is not the same as the winrate testing method, resulting in less success.
Furthermore, both mutations and recombination proved useful evolutionary operators. Some of the top results did, however, vary in how much they used them. For example, some trials used very low mutation rates and others used very high mutation rates.
Surprisingly, some of the top scoring trials used only one hidden neuron, while others used up to 35 hidden neurons.
The overall conclusion is that the evolution parameters can vary quite substantially without making much difference in their performance. It seems like the reason why some parameter sets perform better than another is through a specific combination of parameters. This suggests high complexity of the search space for good parameters combinations. With the exception of the tournament type there is no clear cut pattern or clustering of good results. For example, the difference between using high or low mutation rates is very small. It seems that the best way to optimize this GA is to look for combinations of multiple parameters that work well together.

5.3 Future work

Some future work can be made with alternative methods for mutation, crossover, tournaments, etc. In the paper by (Fullmer and Miikkulainen (1992)) the marker based genetic encoding is proposed. This method is "loosely based on the marker structure of biological DNA."
Furthermore, different types of neural networks can be tested. For example, networks that are not fully connected, or even cyclical networks. Another option would be to try other types of evolutionary neural networks. An example being the NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen (2002)). The authors proclaim that this method "outperform the best fixed-topology methods."
Finally, the evolutionary algorithm proposed in this paper could be applied to another game, like for instance checkers.

References

L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier systems and genetic algorithms. *Elsevier Science Publishers*, pages 235–282, 1989.

H. de Haan. Neural-fitted temporal difference learning to learn to play connect four. Bachelor thesis, AI dept University of Groningen, the Netherlands, 2013.

B. Fullmer and R. Miikkulainen. Classifier systems and genetic algorithms. *MIT Press; Toward a Practice of Autonomous Systems: Proceedings of* *the First European Conference on Artificial Life*, pages 255–262, 1992.

David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Indianapolis, Indiana, 13th edition, 2012.

Kevin Gurney. *An introduction to Neural Networks.* Boca Raton, Florida, 1997.

B. Jacobs. Learning othello using cooperative and competitive neuroevolution. diploma thesis, Utrecht University, the Netherlands, February 2008.

S.M. Lucas, S. Samothrakis, T.P. Runarsson, and D. Robles. Coevolving game-playing agents: Measuring performance and intransitivities. *IEEE*, pages 3–4, February 2012.

K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *MIT Press; Evolutionary computation*, 10:99–127, 2002.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

S. van den Dries and M.A. Wiering. Neural-fitted td-leaf learning for playing othello with structured neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 23:1701–1713, 2012.

M.A. Wiering. Self-play and using an expert to learn to play backgammon with temporal difference learning. *Scientific Research*, 2:57–68, 2010.