# A Neuroevolutionary Approach to Draughts

Student Name: T. P. Nguyen

Supervisor Name: S. Dantchev

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

***Abstract —***

**Background**

Presently, competitive Draughts AI players are currently designed to play at a fixed ability. While it has produced very competitive and intelligent players, they require some human intervention to improve their performance. This is due to their dependency on pre-defined move databases. Optimal moves are pre-calculated, and recalled when necessary. Through neuroevolution, this issue could possibly be solved by creating a player that can grow in ability over time, by learning to play itself.

**Aims**

The purpose of this project is to explore an neuroevolutionary approach to tackle the game of English Draughts. First, we study previous historical successes in the field, and look at the components that helped build their systems. Then, we look at how they can be adapted to suit the objective at hand. The project will establish whether neuroevolution is a viable approach to the problem.

**Method**

The method starts with designing a feed-forward neural network that evaluates the advantage a particular player, given a state of a checkerboard. This is then used in a algorithm that evaluates future moves to predict the best move at a given position. This, alongside a set of weights for the neural network, creates a player that can evaluate potential moves. Finally, the player is then used on an existing Draughts framework that will provide the player with the ability to play Draughts. Genetic algorithms are used to adjust the weights of the neural network, with the intention of making the system learn to evaluate checkerboards more precisely.

**Results**

The system has shown to perform considerably without any training, and has shown to improve over time. However, the system is not necessarily the most efficent of methods.

**Conclusion**

The solution is very appropiate for games that do not involve the use of training data. When using genetic algorithms with feed-forward neural networks, it The robustness of the system is quite volatile, and is best paired with mutation methods that are not heavily reliant on entropy.

***Keywords —*** Artificial Intelligence, Neural Networks, Genetic Algorithms, Draughts, Machine Learning, Monte Carlo Tree Search, Neuroevolution

## I INTRODUCTION

The intention of this project is to explore the effectiveness of genetic algorithms to improve the neurons of a neural network. Neural networks can be used to evaluate the performance of two players in a zero-sum game of Draughts. We attempt to manipulate the neurons through the use of genetic algorithms to increase the accuracy of the evaluation. This would potentially allow us to create an effective Draughts playing agent that, when provided with the option to consider a set of moves, would have the ability to play and learn without human input.

## *Draughts*

English Draughts (or Checkers) is a popular 2-player board-game played on an 8x8 chess board. Players choose a colour and begin with 12 pawns each of their respective colours, and they are placed on the dark-coloured squares of the board. Beginning with the black pawns, each player takes a turn to move a pawn diagonally in one square. In the event that a pawn reaches the opposite side of the board from where the piece originated, it is promoted to a king. Kings have the ability to traverse backwards in the same diagonal motion as pawns. Assuming that there is space upon landing, A piece (pawn or king) also has the option to capture their opponents piece by moving two consecutive diagonal squares, where the opponents piece is situated immediately opposing the players piece. Pieces can be captured consecutively in a single turn if the moves afford the scenario. A player wins by capturing all of their opponents pieces. A player loses by having all of their pieces captured. A draw occurs when both players agree to draw after a three-fold repetition (where both players take three moves to simulate the same position), or a player has pieces on the board but cannot move any of them. A draw can be forced when both players fail to capture a piece after a set amount of moves.

## *Neural Networks*

Neural Networks are non-linear statistical data-modelling tools, linking inputs and outputs adaptively in a learning process similar to how the human brain operates. Networks consist of units, described as neurons, and are joined by a set of rules and weights. The units are defined with characteristics, and appear in layers. The first layer is defined as the input layer, and the last layer being the output. Layers between the two aforementioned are described as hidden layers. Data is analysed through their propagation through the layers of neurons, where each neuron influences the content of the data as they pass through. Learning takes place in the form of the manipulation of the weights connecting the units in the layers. This allows it to model complex relationships between inputs and output, and it can also find patterns in the data.

## *Genetic Algorithms*

As subset of evolutionary algorithms in general, Genetic algorithms (GAs) are a group of search techniques used to find exact or approximate solutions to optimisation and search problems. The methodology is inspired by Charles Darwin's evolutionism theory; individuals are created by the crossover of the genetic information (genome) of their parents. Genomes are metaphors of genetic information; it typically refers to a data structure, most commonly an 1D array. Genetic Algorithms generally consist of a population of agents, a tournament selection process, algorithmic crossover mechanisms against the genomes of the agents, and the introduction of probabilistic mutation on genomes. The combination of evolutionary algorithms to train neural networks is described as Neuroevolution.

## *Aims*

Whilst the use of evolutionary algorithms and neural networks have been explored to create draughts players, my intention is to explore the effectiveness of genetic algorithms and neural networks. The intention is to determine the possibility of developing a performing draughts playing agent by the use of Genetic Algorithms and Neural Networks.

### Minimum Objective

The minimum objective is to implement a feed-forward neural network, alongside a Draughts game interface for the network to evaluate on. This will need to be paired with some decision making program that will determine the move to choose at a given state of the game. Finally, a simple genetic algorithm would also need to be implemented that interacts with the neural network.

### Intermediate Objective

The intermediate objective involves the implementation of an interface to play against the end result, and to create a monte-carlo decision making algorithm, which will be used by the agents to evaluate better quality moves. Also, as the genetic algorithm includes a tournament mechanism which will play many games simultaneously, this will need to be implemented to run in parallel.

### Advanced Objective

The advanced objective consists of producing a system that indicates that it is learning over time. The later sections of this paper evaluates the quality of the learning rate and other results.

To template the document, the later sections of the paper will discuss related and relevant work, the proposed solution, results and an evaluation of the project. Tha paper is then wrapped up with a conclusionary statement, discussing the implications of the project and potential future work.

## II RELATED WORK

Historically, Draughts has been used as a testing ground for artificial intelligence and computional performance since the early introduction of computers. This is due to the relatively vast potential move space, making it difficult to brute-force, until recently. Schaeffer proposed a method to solve the game in 1996(Schaeffer & Lake 1996), and eventually proved a method to weakly-solve the game itself later on in 2007(Schaeffer et al. 2007). However, the premise of the project is not necessarily to solve the game itself, but to rather see whether genetic algorithms are still relevant.

Research in artificial intelligence alone has been active since, targeting games with a higher move space, notably Chess and, more recently, Go. This section will summarise specific and notable techniques relevant to the discussion of the task at hand.

### *A Arthur Samuel*

In 1959, An early design of machine learning to play Draughts was devised by Arthur Samuel(Samuel 1959). His algorithm is based on the perpetual improvement of heuristics he devised, which act as an evaluator for a given state of the checkerboard. The evaluation function is calculated using the linear combination of his heuristics, which included various parameters of relevancy of the game. This includes the number of pawns, number of pawns positioned along the central diagonal, and so on.

His evolutionary strategy consisted of agents that have different coefficents for the heuristics, and new agents are formed using a successor's coefficents with subtle mutations.

Samuel also described an early concept of Mini-Max, with Alpha Beta Pruning. The weights were then trained by the algorithm playing itself in a form of genetic programming. A later investigation by a checkers magazine evaluated Samuel's player to below Class B (Schaeffer 1997, Fogel 2000) where Class B being categorised with an ELO range of [1600-1799].

However, the work described was naturally also dependent on a set of heuristics Samuel devised. This could handicap the agent's ability to play, as they are constrained on the quality of Samuel's heuristics.

### *B Blondie24*

The idea of evolving neural networks to play the game of Draughts is based on the success Chellapilla and Fogel had in evolving their own Checkers neural networks (Chellapilla & Fogel 1999). Their method used Samuel's system as a basis and modified it's evaluation method, using feed-forward neural networks instead of a linear combination. Their resulting player, Blondie24, was then taught to play without priori knowledge, in a similar vein, but with a higher level of competency.

Blondie45 uses an evolutionary algorithm, using fifteen agents $i = 1, \ldots, 15$, each of which consisted of two properties. The first property were weights of the neural network (which are initially random), with the second property being a self-adaptive parameter vector, $\tau_i$, which determined how random the mutations be when the agent $i$ evolved.

In each generation, every agent plays each other in a round robin style tournament, and successors are chosen based on their performance from it. The best agents are chosen to mutate themselves to create new agents. Mutation is performed using the formula:

$$\tau_i(j) = \tau_i(j) \cdot e^{\mu}$$

Where $j = 1, ..., N_w$ represented each weight of the neural network $w$, and $\mu$ representing a random decimal in the range of $[0 - 1]$.

However, crossover mechanisms were not considered (and is thus not classified as using genetic algorithms). Blondie24's neural network structure consists of a $\{32, 40, 10, 1\}$ set, such that the input layer consisted of 32 nodes, and a single output node. The output node consists of a classifier from [-1, 1] inclusive, representing a particular advantage for a given side of the board. One fallback that can be seen with this particular method is that spatial awareness is not considered as it takes an immediate input of the positions on the board. This makes it inherently more difficult for the neural network to generate heuristics based on spatial awareness. Relative distance between pieces are not encapsulated in a method that can be immediately interpreted by the neural network.

Blondie24's decision making algorithm primarily revolved around a mini-max algorithm with a ply cap. This algorithm, with the state of the checkerboard as an input, expands a tree of potential moves. Each node represents a state of a checkerboard. Child nodes represent a state of the checkerboard for a given move. Child nodes can only be produced from the parent node; i.e a move from a parent node will produce a child node.

We do not wish to repeat this work, but it is important to understand that we wish to abstract and modularise parts of the best features of this system to modify and extend them in the solution described in the paper.

## *C Post Blondie24*

Blondie24 was influential in the field of neuroevolution during its time, leading to various studies on how it evolves. A notorious example is the Al-Khateeb and Kendall series, where they explore in detail the different features that comprise Blondie24. One of their insightful studies with the Look-ahead Depth (Al-Khateeb & Kendall 2012) conlcuded that having a higher ply-depth increases the overall performace of the player, and that agents trained at a higher ply performs significantly worse when playing at a lower ply.

The tournament approaches (Al-Khateeb & Kendall 2009), the value of piece differences (Al-Khateeb & Kendall 2010).

Quetzalcoatl Toledo-Marin et al. (2016) performed studies on attacking strategies using the game of checkers as a reference (which however can be applied to any other zero-sum game), where they mathematically prove that for an offensive player, maximising the offensive improves their probability to win.(Toledo-Marin et al. 2016)

Cobbe, et al. improves Blondie24 by using an evaluation function that consists of Support Vector Machines, which achieves locally optimal performance in approximately a third as many generations than its typical evolution. (Cobbe et al. n.d.)

## *D Neural Networks*

Lai's Giraffe (Lai 2015) program used a deep reinforcement learning technique to play Chess to an advanced level in a relatively quick time-period. Giraffe's decision function involved using Temporal Difference Learning (TD-Leaf), an improvement on MiniMax, by Baxter et al. (Baxter et al. 1999) However, Giraffe has been trained on previously played games from grandmaster tournaments as a reference, and was not entirely self-taught.

## *E Return of Genetic Algorithms*

In 2017 Lingxi Xie et al. took it a step further and discussed using genetic algorithms to compute optimal deep learning structures automatically, using an image recognition dataset as the benchmark for competitiveness against the networks.(Xie & Yuille 2017)

Since then, there has been a non-trivial number of papers within Checkers, Genetic Algorithms and Convolutional Neural Networks. Sergei Perez (UC Irvine) describes the use of genetic algorithms to evolve weights for a convolutional neural network as a more performant alternative to back-propagation, in nearly all situations, other than when the number of generations generated for neural networks are small.[21]

Interestingly, Clune et al. on behalf of Uber Labs returned to the use of genetic algorithms as a basis to train deep convolutional networks with success. (Such et al. 2017) Their GA-NN system is used to play popular Atari games on the OpenAI framework, and is found to perform as well as other contemporary evolution strategies and reinforcement learning algorithms.

This is based on their system's ability to perform safe mutations (Lehman et al. 2017), a technique that revolves around measuring the sensitivity of the network to changes with some of the weights. This allows them to evolve deep neural networks, having over 100 layers and over four million weights and biases, allowing random but safe and exploratory steps. Their safe mutation exploits the knowledge of the neural network structure, by retaining sampled experiences of the network (inputs that feed into the network and its outputs). This allows the weights to be manipulated by comparing its significance with the output, by comparing it with the sampled experiences.

However, they also do not consider the use of crossover techniques in their paper, but they do describe that it would very well be possible to produce a safe crossover method that can exploit the structure of the neural network, inspired by the research they've produced on safe mutations.

Unfortunately, it is unrealistic to apply the same techniques described here into the proposed system. The machine used as the base (consisting of hundreds to thousands of cores), but a more simplistic and adapted model can be produced using similar ideas.

## III SOLUTION

This section describes the build of the system in order of implementation. Implementing the checkerboard interface is not included in the discussion as it is not one of our objectives. However, the sections below are considered the interface being built prior.

### *A Neural Network*

To evaluate the board, a feed-forward multilayer perceptron style neural network will be used. The network would contain 4 layers; the input layer consists of 91 nodes, with the output node having 1. Hidden layers will have 40 and 10 nodes respectively. Figure 3 shows the network in the form of a diagram.

Common knowledge infers that a king is worth more than a pawn, but it is disputed about its precise value advantage. For the sake of simplicity, a king's piece value is to be weighted at $1.5x$ a pawn value; a value of a pawn will be 1.

The input array takes in the checkerboard. The intention is to weigh the Black pawns with a value of 1, and white pawns as -1. To create the input layer, we treat the checkerboard into a 1D array, with the indexes displayed in figure **??**. The array is used to calculate all possible sub-squares of the checkerboard, ranging from a 3x3 kernel to a 8x8. Each sub-square is summed up to create an input node. There are consequently 91 combinations of the sub-squares, thus forming the input layer.

Weights of a neuron are multiplied by their input value, summed with their bias, and are passed through an activation function $O = f((Input * weight) + Bias)$ to become an input for another neuron. Activation functions, when visualised, usually have a sigmoid curve, but may also take the form of other shapes. Common characteristics of activation functions include values to be monotonically increasing, continuous, differentiable and bounded.

Our initial choice for the activation function is tanh, shown in figure 1. There exists inherent issues related to the properties of their derivatives, discussed by Nair and Hinton (Nair & Hinton 2010), described as the missing gradient problem. However, since this issue is related to the properties of a gradient based learning method and not through a stochastic learning method (of which

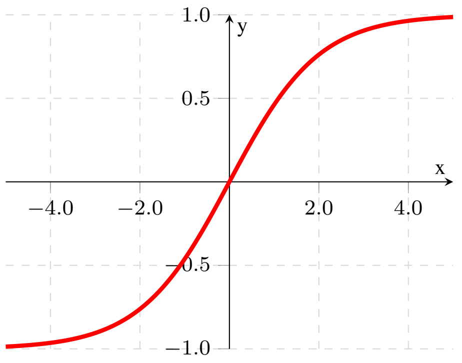


Figure 1: Graph of tanh function $f(x) = \frac{2}{1+e^{-x}} - 1$ (An example of an activation function with a sigmoid curve).

genetic algorithms are), this is not a concern for the project. Sigmoidal function also facilitates the ability to simulate a zero sum formula (for a range of -1 to 1) due to its central symmetric properties.

Results are cached such that inputs that have been already evaluated are saved such that they can be recalled immediately. This has shown varying levels of improvement dependent on the ply depth. As the ply increases, the size of the state space increases, and therefore the chances of recalling the same set of inputs naturally decrease. This is also used for our safe mutations algorithm, described later on in

## *B Decision Making Algorithm*

The mini-max method would have a search depth of 4ply (where the agent will search four moves ahead.) This should allow the agents to form a basic strategy where they can plan their moves in advance.

To choose the best move given a current position, an initial decision making process for an agent revolves around the use of minimax algorithm. This was the case for Chellapilla and Fogel's agent. (Chellapilla & Fogel 1999).

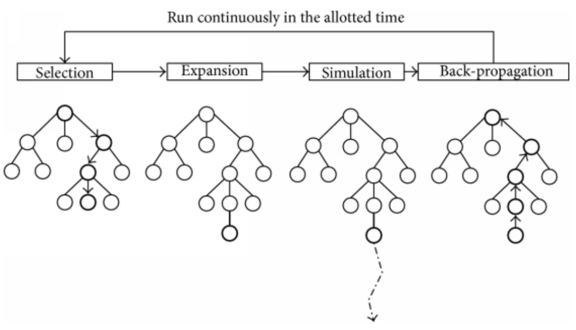


Figure 2: A diagram describing the MCTS process.

Once initial development on the mini-max algorithm is complete, The project would migrate to a hybrid technique that combines mini-max and a more contemporary algorithmic paradigm, Monte-Carlo Tree Search (MCTS). A basic MCTS differs from mini-max where future moves are randomly played to the end of the game. It acts as a sampling of possible outcomes, and does not depend on an evaluation function at all. The random simulation of games are skewed such that more reasonable moves are chosen. A survey by Browne et al. found that MCTS can dramatically outperform mini-max based search engines in games where evaluation functions are otherwise difficult to obtain (Browne et al. 2012). MCTS are played in rounds that consist of four operations:

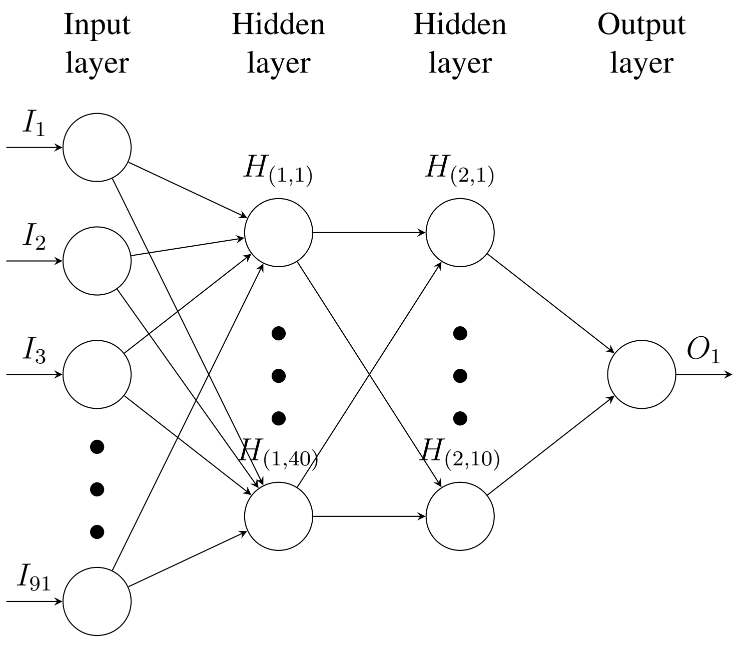
- Selection: A selection strategy is played recursively until some position is reached.
- Play-Out: A simulated game is played. Again, a basic form of MCTS would play until an end-game is reached.
- Expansion: A node is added to the tree.
- Back-propagation: The result of the game is back-propagated in the tree of moves.

These rounds are repeated until a certain period of time is met or a certain number of rounds are computed. Once finished, a node is chosen based on its total probability of reaching an winning outcome.

MCTS-EPT (or MCTS with early playout termination) is introduced by Lorentz (Lorentz 2016). MCTS-EPT modifies the MCTS random play-out approach; Instead of allowing the random moves play to the end of the game, the number of moves traversed are capped and an evaluation function can be used from that caped position instead. The termination ply would be capped at 6ply. This could potentially improve the amount of foresight for a given set of moves, without the need to depend on random generations of moves to the end, and the need to evaluate more moves (as typically needed even with alpha-beta pruning on a mini-max algorithm).

The minimax algorithm would be retained to evaluate the performance differences between the two decision making algorithms.

Figure 3: The chosen neural network model. The preprocessed values of the checkerboard are used as input. An output is produced after propagation that ranges from [-1,1].



### *C Genetic Algorithms*

The genetic algorithm (GA) is the premise of the system's learning strategy. GA's are used to train the neural network. We discuss the various algorithms that form the collection of GA strategies below. For the system, a population size of 15 will be chosen.

### C.1 Population Generation

In genetic algorithms, the population serves as a base that allows agents in the pool to play each other. Every generation has its own population, with all having the same number of agents. The initial population will consist of randomly generated weights and biases of the neural network, with values from [-1,1].

For a population size of 15, the next generation is created using the best five agents from the current generation. This is discussed in *C.2 Tournament Selection*. They will continue to play in the next generation; this strategy is described as elitism. These agents are also used as a base to create 10 new agents from.

The next eight players are generated through the use of crossover strategies, explained in *C.4 Crossover Strategy*. The weights of the $1^{st}$ and $2^{nd}$ place agents are used as input to the crossover strategy and will generate 4 offsprings. Two are reciprocal crossover representations from each other, and the other two being directly mutated from the parents themselves. Another four children will be created using the same strategy, with the 2nd and 3rd agent's weights. The remaining two will be direct mutations of the 4th and 5th place agents.

## C.2 Tournament Selection

To sort the quality of the players in a population, a tournament selection process is deduced. This allows us to choose the best players who will continue to play in the next generation.

Currently, each agent in the population would play 5 games as Black, against randomly selected opponents. Each game lasts a maximum of 100 moves from both players. If a winner is not deduced at this stage, a draw is called. Draws are also called when there is a three-fold move repetition from both players. A win is worth 2 points, a draw being none and a loss being $-1$ points. Both the agent and its opponent receives a score from the game. Scores are tallied up at the end of the tournament. Players are sorted by the number of points they scored. The best players would have the highest number of points.

## C.3 Coefficient Mutation

To create variation between agents in preparation for the next generation, we create statistical anomalies through the use of mutations. This is used as one of the learning mechanisms that help change the decision factors of the neural network.

Weight and biases of an agent's neural network would increment by a random value that is created using the following formula, where $WeightP$ is the current weight, $K$ represents the number of weights and biases in the neural network, and $m$ representing a random floating point in the range of [-1,1]. Equation 1 describes the mutation formula.

$$WeightN = WeightP + \frac{m}{\sqrt{2 * \sqrt{K}}} \tag{1}$$

The weights, like the activation function (in 1), will have a soft maximum of [-1, 1]. This would consequently mean that the mutation is not controlled, and dependent on the number of weights in the system; The more weights in the network implies a less significant mutation.

We also choose to experiment with soft mutations, inspired by Lehman et al. (Lehman et al. 2017) We use a subset of precalculated neural network calculations for inputs and outputs and use this as a premise to guide our mutation. The method is as follows:

1. With the current set of weights $w$, choose a subset of precalculated input and output tuples $\phi_w = (I, \lambda_{I,w})$ where $I$ represents the input values of the network, and $\lambda_{I,w}$ as the output.

2. Generate a perbutation of the weights calculated using Equation 1. Call this $y$. Use $y$ as a basis of a neural network.

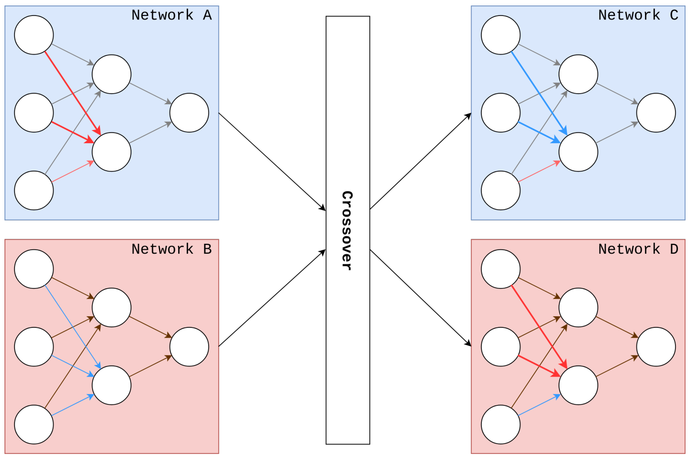


Figure 4: A diagram visualising the crossover process.

3. Using the inputs $I$ in $\phi_w$, calculate a new set of input and output tuples $\phi_y = (I, \lambda_{I,y})$.
4. Calculate the divergence $\delta = \lambda_{I,y} / \lambda_{I,w}$.
5. Calculate the quality of $y$ based on the number of inequalities $\delta \geq 1$.
6. Repeat steps 2-5 some $n$ times, keeping note of the the best $y$.

The intuition behind this is that if $\phi_w$ is a subset of moves that led to a winning game, then the manipulation of the weights is left open as long as it can replicate the performance of winning games. One of the concerns about this approach, also mentioned during the similar works, is the efficency of the approach.

### C.4 Crossover Strategy

Another learning mechanism provided from the use of genetic algorithms is the use of crossovers. This combines the traits that build two parent agents to create children. In our scenario we use the weights and biases of the parent's neural networks.

Two offsprings would be created from a pair of parents, with each offspring being the reciprocal crossover of each other. A random node from the dominant parent's hidden nodes are chosen. A dominant parent is the agent who gained more points in the tournament round than the other (who would be the submissive parent). If a dominant parent cannot be deduced it would be randomly chosen.

Once a random node is chosen, The weights and biases that feed into the chosen node are swapped with the submissive parents values at the same positions. As values in the weights can range from -1 to 1, dominant input weights are values that are greater than 0.75 or less than 0.75. Figure 5 shows this in more detail. In the diagram, Network A and B are parents, creating reciprocal offspring networks C and D. A is the dominant network, and A's node with the red inputs is chosen. The weights and biases of the red inputs are swapped with the values at the same position in network B. Network C is formed with Network B's dominant weights, with the rest being composed from A. Network D is the reciprocal, having Network A's dominant weights and the rest from B.

This crossover process would create a subtle modification to a neural network, by swapping a small subset of weights and biases at a time. Having a more dramatic crossover could potentially change the structure of the network flow, reducing its effectiveness. A visualisation of the crossover process can be found in Figure 5.

### *D Testing*

Initial runs will operate on a 1-ply load to determine the stability of the system on a linux (Ubuntu) machine containing a 4-core Intel i5 6600u processor with 12GB's of memory. Development and debugging will also occcur on this machine.

Once testing has proven to be stable, the system would run on Durham's MIRA distributed system (Debian) with a 6-ply move depth. This machine utilises 4 Intel Xeon E7-8860 processors (Each CPU contains 16 physical cores running in 2.2Ghz, with hyperthreading). This comes to a total of 64 Cores, and 128 threads. MIRA also comes with 256GB of DDR3 memory.

To keep simulations running on MIRA, MOSH is used to maintain a consistent connection to MIRA. The end champion is then transferred to the initial machine in order to be played against by human input. Statistical evaluations and calculations are also calculated on MIRA to reduce computational time.

At the end of a given generation, we measure growth of performance by using the generation's champion. Presently we will use the mean of means approach. When a new champion is generated, it is played against the previous 5 champions from earlier generations. 6 games are played for each previous champion, with 3 being as Black, and 3 being White. A mean score is calculated from those 6 games. The overall performance of the current champion is the mean of the 5 sets of games. A positive improvement is when the mean of means are greater than 0.

Point Score for the champion games are measured by 1,0,-1 where a Win counts as 1 point and -1 for a loss. The weights are scaled differently to the regular tournament in order to accurately portray the difference between previous champions.

### Evaluation Method

At the end of the generation run, the end player (being the champion of the last generation) will be used to compete against human players on various online multi-player checkers websites in order to determine an accurate ELO rating of the system.

The end player is also used to create comparison games. This player is tested against an agent who is choosing random moves, an agent who is using pure monte-carlo tree searches, and an agent from half the generations generations prior to its current state. These statistics will be used as evidence in order to determine whether the agent is learning, and can also be seriously considered as a viable alternative to other machine learning training approaches.

### *E Issues*

- shitty running times - finding a computer that would run the simulations - appropiately saving files - testing neural network was difficult - testing move decisions were difficult - optimising the learning rate

Interestingly, the machine used for training did not perform necessarily as effeciently. The system is built such that the tournaments are run in ”embarassingly parallel”, but the machine does not scale accordingly. Most of the computation is spent on calculating floating point matrix operations (to evaluate states of the board.)

Later research has shown that the use of hyperthreading does not necessarily show to scale the performance, even though more threads exist. (Leng et al. 2002) This is due to the shared floating point arithmetic registers in the CPU, where two threads utilise one physical cores property.

## IV RESULTS

### *A Learning Rate*

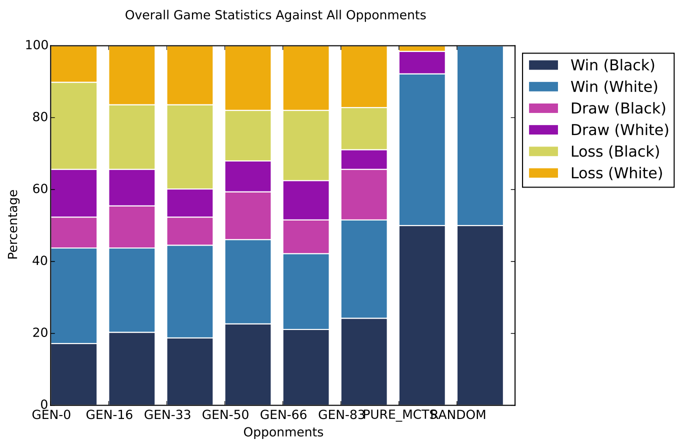


Figure 5: A diagram visualising the crossover process.

With a ply depth of 3, we can immediately see the impact of the system's performance against other agents that have

### *B Other Observations*

#### B.1 Number of Moves

#### B.2 CPU Times

#### B.3 Influence of Inheritance Methods

### *C Speed*

As predicted, calculating safe mutations is a very computationally expensive endeavour. Efforts to subsidise the performance of

## V EVALUATION

- The quality of the neuroevolutionary algorithm boils down to the quality of the fitness function, which in our case is the tournament; hard to choose the best method - Difficult to evaluate moves - Naturally slower learning rate than typical ventures as there is an immediate comparision to make against -

### *A Strengths*

### *B Limitations*

### *C Approach*

- should have started with times

## VI CONCLUSIONS

- importance of the work - for applications that do not have a immediate - applications - extensions - further work - experiment with different

## References

Al-Khateeb, B. & Kendall, G. (2009), 'Introducing a round robin tournament into blondie24', *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games* **44**(0), 112–116.

Al-Khateeb, B. & Kendall, G. (2010), The importance of a piece difference feature to Blondie24, *in* '2010 UK Workshop on Computational Intelligence (UKCI)', pp. 1–6.

Al-Khateeb, B. & Kendall, G. (2012), 'Effect of look-ahead depth in evolutionary checkers', *Journal of Computer Science and Technology* **27**(5), 996–1006.

Baxter, J., Tridgell, A. & Weaver, L. (1999), 'TDLeaf(lambda) : Combining Temporal Difference Learning with Game-Tree Search', *arXiv:cs/9901001* . arXiv: cs/9901001.
**URL:** *http://arxiv.org/abs/cs/9901001*

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012), 'A Survey of Monte Carlo Tree Search Methods', *IEEE Transactions on Computational Intelligence and AI in Games* **4**(1), 1–43.

Chellapilla, K. & Fogel, D. (1999), 'Evolving Neural Networks to Play Checkers without Expert Knowledge', *IEEE Transactions on Neural Networks* **10**(6), 1382–1391.
**URL:** *http://ieeexplore.ieee.org/abstract/document/809083*

Cobbe, K., Lee, P. & Gomez-Emilsson, A. (n.d.), Accelerating Checkers AI Evolution, Technical report, Stanford University.
**URL:** *http://cs229.stanford.edu/proj2011/CobbeLeeGomez-AcceleratingCheckersAIEvolution.pdf*

Fogel, D. (2000), 'Evolving a Checkers Player Without Relying on Human Experience', *Intelligence* **11**(2), 20–27.
**URL:** *http://doi.acm.org/10.1145/337897.337996*

Lai, M. (2015), 'Giraffe: Using Deep Reinforcement Learning to Play Chess', *arXiv:1509.01549 [cs]* . arXiv: 1509.01549.
**URL:** *http://arxiv.org/abs/1509.01549*

Lehman, J., Chen, J., Clune, J. & Stanley, K. (2017), 'Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients', *arXiv:1712.06563 [cs]* . arXiv: 1712.06563.
**URL:** *http://arxiv.org/abs/1712.06563*

Leng, T., Ali, R., Hsieh, J., Mashayekhi, V. & Rooholamini, R. (2002), 'An empirical study of hyper-threading in high performance computing clusters', *Linux HPC Revolution* **45**.

Lorentz, R. (2016), 'Using evaluation functions in Monte-Carlo Tree Search', *Theoretical Computer Science* **644**, 106–113.
**URL:** *http://linkinghub.elsevier.com/retrieve/pii/S0304397516302717*

Nair, V. & Hinton, G. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of the 27th international conference on machine learning (ICML-10)', pp. 807–814.

Samuel, A. (1959), 'Some studies in machine learning using the game of checkers', *IBM Journal of Research and Development* **3**, 210–229.
**URL:** *https://ieeexplore.ieee.org/document/5392560/*

Schaeffer, J. (1997), *One Jump Ahead*, Springer-Verlag.

Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., Lu, P. & Sutphen, S. (2007), 'Checkers Is Solved', *Science* **317**(5844), 1518–1522.
**URL:** *http://science.sciencemag.org.ezphost.dur.ac.uk/content/317/5844/1518*

Schaeffer, J. & Lake, R. (1996), 'Solving the Game of Checkers', *Games of No Chance* **29**, 119–133.
**URL:** *http://library.msri.org/books/Book29/files/schaeffer.pdf*

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. & Clune, J. (2017), 'Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning', *arXiv:1712.06567 [cs]* . arXiv: 1712.06567.
**URL:** *http://arxiv.org/abs/1712.06567*

Toledo-Marin, J. Q., Diaz-Mendez, R. & Mussot, M. d. C. (2016), 'Is a good offensive always the best defense?', *arXiv:1608.07223 [cs]* . arXiv: 1608.07223.
**URL:** *http://arxiv.org/abs/1608.07223*

Xie, L. & Yuille, A. (2017), 'Genetic CNN', *arXiv:1703.01513 [cs]* . arXiv: 1703.01513.
**URL:** *http://arxiv.org/abs/1703.01513*