Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

# The Ability of the NEAT Algorithm to Learn an Imperfect Information Game (Liar's Dice)

A Dissertation Submitted in Partial Fulfilment
of the Requirements for the
Degree of Bachelor of Arts (Honours)

Dara Crowley
The University of Dublin, Trinity College

Department of Economics

April 2025

Bachelor of Arts in Economics
Trinity College Dublin, The University of Dublin
April 2025

## Declaration Page

I confirm that:

(1) This dissertation has not been submitted as an exercise for a degree at this or any other University,

(2) This dissertation is entirely my own work

(3) I agree that the Library may lend or copy the dissertation upon request.

**Signed:** **Dara Crowley**

**Date:** **2/4/25**

## Acknowledgements

First and foremost, I would like to thank my supervisor Niamh Wylie whose guidance and support helped me find my way through the most challenging moments this capstone presented. I also give my thanks to Dr. Tara Mitchell whose advice and clarifying explanations of the key game theory concepts this dissertation deals with.

Finally, to my family and friends whose abundant support and patience made all the difference as I droned on for months about network topologies, training times, Nash equilibriums and fitness scores.

# Contents

## List of Figures

## List of Tables

## Abstract

This dissertation examines the ability of NEAT neural networks trained through self-play to learn an imperfect information game (Liar's Dice). Games of imperfect information are a key topic in game theory whose analysis allows for a better understanding of real-world strategic situations. Two NEAT models with contrasting philosophies are developed to evaluate the NEAT genetic algorithm's training evolution. The first is provided with the minimum information and guidance, unlike the second, whose design builds many core game principles. Both models are tested throughout their training against a pure strategy to evaluate their growth. In addition, networks from different stages of training are compared against each other. From the statistical analysis of these test results, it is concluded that the NEAT algorithm demonstrates an evident ability to learn and develop strategies for Liar's Dice. The more guided model is the stronger of the two but displays less relative growth. The results also point to a weakness of the self-play methodology as networks are not compared against all strategies developed during training, and they struggle to improve iteratively.

# 1 Introduction

This paper examines the ability of neural networks trained through self-play to learn imperfect information games. Imperfect information games are a particular game type studied in game theory. Game theory is an important economic field as it provides a theoretical framework from which real-world strategic interactions can be analysed, evaluated and solved. A crucial element of game theory analysis centres on the information available to individuals. The balance of information is generally categorised into two groups: (im)perfect and (in)complete information. Imperfect information implies there is some element of the game setup that players are not privy to. Incomplete information implies that players are uninformed of their opposition's type or strategy. This dissertation focuses on Liar's Dice, which is generally categorised as an imperfect information game as players are not informed of their opposition's dice. Examples of similar game styles are card games such as poker, bluff and bridge. There is also an incomplete information component, as players are not aware of their opponent's strategy. However, literature in the area generally refers to games of a similar structure as Liar's Dice as imperfect information games. To evaluate the neural network's ability to learn this game style, two models will be developed and compared at various game difficulty levels through testing. The first model is provided with minimal guidance and information, and the second takes the opposite approach. The study's hypothesis is that both models will demonstrate learning capabilities, with a decreased learning rate for more difficult game setups. The second model is expected to have stronger performance. The reasoning for this hypothesis is based on the mechanics of the neural network evolution process, which allows for efficient learning but has limited computational complexity.

## Liar's Dice

Liar's Dice has many rulesets. This paper focuses on the variation seen in Ahle's project, which received increased popularity through its use in the film The Pirates of the Caribbean: Dead Man's Chest (Dybdahl Ahle, 2022). The game can be played with any number of players, but there will be a limit to two for this capstone. Each player has a set of dice, also referred to as a hand, which they roll privately at the beginning of the game. Players are not informed of their opposition's hand, which is why it's a game of imperfect information.

Moves consist of bets that make a claim about the make-up of the total set of dice in the game. For example, a bet of 2x 3s claims at least two dice with a value of 3 between the players. Players take turns making bets to increase quantity or value. Should a player not believe their opponent's bet, they can call. When a call is made, players reveal their dice. If the bet is satisfied, the caller loses; if the bet is not satisfied, the caller wins. An example game state is included in the appendix Figure A1.

## Game Theory

Game theory for imperfect information games like Liar's Dice provides insight into the characteristics of successful strategies. A strategy can be pure or mixed. When faced with the same game information, a pure strategy takes the same action. When faced with the same game state, a mixed strategy can take a range of actions with different likelihoods. The advantage of a mixed strategy in imperfect information games is that it better disguises the player's private information. For example, in poker, if a player makes large bets every time they have a strong hand, the rest of the players will quickly uncover this strategy and fold whenever a big bet is made. This decreases the pure strategy player's earning potential. A strategy that varies responses is less exploitable.

## Neural Networks and NEAT

Neural networks are a form of computational model designed to simulate how neurons function. They take in information through input nodes, which are fed through a series of hidden layers comprised of nodes that perform various calculations. The resulting calculations of the final hidden layer are then fed to the output nodes, which is the model's result. The advantage of neural networks over other models is their ability to evaluate complex nonlinear relationships. Neural networks have been implemented with substantial success across a vast array of problems such as stock market prediction, cancer research, and facial recognition. Neural networks improve through training, which is a process of optimising network calculations through feedback, which comes in various forms, some of which are described in the literature review.

NEAT stands for NeuroEvolution of Augmenting Topologies. This is a particular algorithm for training models called a genetic algorithm. Genetic algorithms create populations of various neural networks which attempt to solve a given task. Each network receives a score or fitness based on their performance in solving the task. The genetic algorithm improves populations over time by generating the next population based on the best networks of the previous generation. The NEAT algorithm's ability to evolve the network's topologies distinguishes it. A network's topology is how its hidden nodes are structured and connected. The algorithm's ability to evolve this topology allows for efficient improvement or learning.

## Project Design

This dissertation differs from the available research as no other study has applied the NEAT algorithm to Liar's Dice. This analysis offers a unique insight into how genetic algorithms such as NEAT handle imperfect information games. Many neural network approaches have been taken to develop strong AI players for various imperfect information games. However, many of these methods require advanced network evaluation approaches and significant computational resources. This project will provide insight into the effectiveness of a simpler network and training process in tackling a game of similar complexity to those analysed in the research.

The methodology for this project has three main stages: network development and training, testing, and statistical analysis. Network development and training require the configuration of two models, as described in the opening paragraph. Both models will be trained through self-play, a form of network evaluation where performance is determined through the population of networks competing against each other. Throughout the training process, models will be tested against consistent opposition. This opposition will play with a pure probabilistic strategy based on dice probabilities. In addition, the models will be tested individually by comparing different generations of the same model against each other. These tests are designed to provide insight into the models' learning progress. Finally, statistical analysis of the testing data and other training metrics will be the basis of the dissertation's results and conclusions.

## Key Findings

The statistical analysis found that both models demonstrated improvement against the pure strategy through training for all but the highest difficulty level. The second model, which received more information and guidance through its implementation, was the stronger of the two models but showed less improvement over the course of its training. The inter-generational testing found little evidence of improvement against earlier generations. These results indicated a potential weakness of the self-play training philosophy due to dynamics described by game theory.

## Dissertation Structure

The remainder of this paper is structured as follows:

- **Section 2** reviews the relevant literature.
- **Section 3** outlines the dissertation's methodology, including game structure, neural network development and training, and pure strategy implementation.
- **Section 4** details the testing process and empirical strategy.
- **Section 5** presents and analyses the results of the statistical models.
- **Section 6** Concludes the paper and discusses areas for future research.

# 2 Literature Review

The literature surrounding this capstone provides various perspectives on machine learning and specifically neural network application to Liar’s Dice and imperfect information games generally. The available literature can be divided into three sections: Liar's Dice, Neural Networks in Game Theory, and NEAT. The first three sources of the Liar’s Dice section form the foundational resources that inspired this dissertation’s processes and goals. The neural network studies detail the range of approaches taken in developing game strategies. The NEAT sources offer a guide to NEAT implementation and an overview of its core design philosophy.

## Liar's Dice Literature

Liar's Dice by Self-Play (Dybdahl Ahle, 2022)**:**

Dybdahl Ahle's project trains a neural network based on Monte Carlo counterfactual regret minimisation. The difference between Ahle’s implementation and the implementation in this project is in the training methodology. As stated Ahle improves models through optimising CFR whereas the models developed in this dissertation improve through the fitness function. Ahle utilises a very accurate game theory derived testing method which evaluates a models exploitability, however, it is computationally limited to the most simple game formats.

Dynamic Programming for Liar's Dice (Johnson, 2007):

Johnson develops two game strategies: a pure probabilistic strategy where decisions are made based on dice probabilities and a matrix mixed strategy, which selects from a range of actions given the current game state. The models are tested against human opponents and each other to evaluate their strengths. The mixed strategy performs best against humans, suggesting it better achieves game theory optimal play. The testing against the probabilistic strategy inspired the similar testing method of this dissertation.

Endgame Strategies and Simulation Results for the Liar's Dice Game (Bursci and Nagy, 2014):

Bursci and Nagy investigate endgame positions to determine optimal strategies. They also employ evolutionary simulation, in which heuristic strategies improve over time by iterating on the best-performing strategies. This process of iteration shares similarities with the NEAT algorithm evolution but differs significantly in strategy scope.

The Mathematics of Games (Taylor, 2014):

Taylor's textbook covers a variety of probability-based games in which he analyses Liar's Dice. Taylor outlines how dice probabilities can be calculated to inform decision-making within the game. These calculations are used as foundation blocks for developing the probability strategy used in this dissertation. He highlights the challenges associated with evaluating games with bluffing mechanics.

The Tactics of Liar's Dice (Freeman, 1989):

Similar to Taylor's work, Freeman looks at dice probabilities for a slightly more advanced version of Liar's Dice, sometimes referred to as Poker Dice. He similarly identifies the challenge of integrating these tactics into a fully developed strategy.

Several papers explore a similar variation of the game where only one dice is used, and players must determine if their opponent is lying about the value of that dice (Ferguson and Ferguson, 1991; Ferguson, 1970; Sum and Chan, 2003). Ferguson's analysis adds an extra level of nuance, whereas, instead of using a die, the players receive a random number between 0 and 1. While the games analysed in these papers are quite different from those in this dissertation, many of their probabilistic methodologies are similar and provide further support as resources for this dissertation.

## Neural Networks in Game Theory Literature

A rich supply of papers implementing neural networks for game theory problems exists. These papers use advanced methods to develop incomplete/imperfect information game strategies. Many of these approaches require substantial computational power and expertise to develop effectively. This dissertation investigates the capabilities of a genetic algorithm trained through self-play in learning a game which differs from much of the available literature that tries to develop optimal solution strategies through complex methodologies outlined below.

### Reinforcement Learning (RL)

RL is a method of training neural networks. The philosophy behind this approach is to improve networks over time by rewarding better play (Sutton and Barto, 2015). Alushi's paper analyses how RL can be used in dice games and the advantages/disadvantages of different approaches (Alushi, 2022). Often, RL is extended to deep reinforcement learning, where the neural networks have significantly more layers. These additional layers allow for better performance in complex tasks at the cost of training time.

### Q-Learning (QL)

QL develops a strategy by mapping the best response for each game state and calculating the expected reward for each action (Watkins and Dayan, 1992). This method has a high computational cost, as it must map every possible game state and action. This quickly becomes infeasible for complex games. A popular solution to this issue is implementing deep neural networks to estimate rewards.

### Counterfactual Regret Minimisation (CFR)

CFR is highly effective at developing game strategies that satisfy Nash equilibrium requirements (Zinkevich, Johanson, Bowling, and Piccione, 2007). CFR effectively finds Nash equilibrium as regret minimisation reduces strategy exploitability, which is key. From a high level, CFR is similar to QL. Both develop strategies by evaluating all options and selecting the best of them. The key difference between the methodologies is how they evaluate actions. CFR uses "regret" to represent the difference in outcome probability between the actions taken and those that could have been taken. This approach has the same weakness as QL, which requires mapping of the entire game.

Monte Carlo CFR avoids mapping all states by determining regret as it goes through stochastic self-play (Lanctot, Waugh, Zinkevich, and Bowling, 2009).

Single Deep CFR removes the need for a secondary neural network similar to that required by deep-QL by embedding it into the self-play approach. Steinberger's approach demonstrated impressive results, outperforming the method it was derived from (Steinberger, 2019).

Opponent Modelling (OM)

OM is a very intuitive approach to developing strategies in games. It utilises pattern recognition to map its opponent's playing style and develop a strategy to best counter such playing style (Li and Miikkulainen, 2018). The advantage of this approach over that of CFR is its success in exploiting weak opposition. Where CFR minimises its exploitability, OM maximises its exploitation of the opposition.

## Neuroevolution of Augmenting Topologies (NEAT) Literature

Stanley and Miikkulainen developed NEAT. Their paper on the algorithm demonstrates its effectiveness and advantage over fixed-topology networks (O. Stanley and Miikkulainen, 2002). The NEAT algorithm allows for the evolution of the network topology, providing an expanded range of potential development during the learning process. In contrast to reinforcement learning the networks evolve through the genetic algorithm, which reproduces the next generation from the previous generation's best networks. The uniqueness of the NEAT algorithm gives novelty to this dissertation's analysis as its characteristics differ significantly from the approaches taken in the literature above.

The NEAT documentation and code depository extrapolate on the algorithm's abilities, outlining the relevant features and how they can be implemented (NEAT-Python Documentation, n.d.). These files specify the mutation, speciation, and reproduction processes, all of which can be adjusted.

Various publicly available code projects guided this dissertation's implementation of the NEAT algorithm. Ruscica's project instructed the general coding process (Ruscica, 2024). B2Studio's monopoly AI inspired the training methodology (b2developer, 2022). Fitzpatrick's poker bot provided a reference for the networks' configuration (Fitzpatrick, 2024).

# 3 Methodology

## 3.1 Overview

As outlined in the introduction, this capstone aims to investigate the ability of NEAT neural networks to develop a strategy for an imperfect information game. The networks implemented in this project utilise self-play training in Liar's Dice at various levels of game complexity. Both models will be tested incrementally throughout the training process to evaluate their level of improvement. The difference between the models relates to how their outputs impact game actions. The first network's outputs will directly imply actions, and the second's outputs will determine the probability of taking an action. The models are tested by playing against a pure probability strategy.

<u>Liar's Dice Code</u>

Before models can be trained the game environment needs to be developed from scratch. The game environment code handles all game actions, information, and visualisation. There are two actions in Liar's Dice: bet and call. Betting mechanics code logic ensures bets are only accepted if the bet quantity or value increases in alignment with the ruleset. In addition, no bets that exceed the limits can be accepted, i.e., the quantity is greater than the number of dice in the game, or the value is higher than 6. Illegal bets result in instant loss. Call mechanics are more straightforward; if a player calls, the code checks whether the current bet is true, determining the winner. The only case where a call is an illegal action is when there is no bet.

The game code keeps track of betting history, dice, and winning players. Betting history is stored in an array that appends bets as they are placed. Lists store the dice hands, which are generated randomly at the beginning of each game. Winning players are determined by checking call success and which player made the call. All game information resets at the beginning of each round.

The gameplay can also be visualised through the pygame library, allowing easier debugging. Visualisation is disabled during training to increase efficiency.

## NEAT Algorithm

Neural networks are a form of model used to solve non-linear problems. A network is composed of several layers made up of nodes. The first layer is the input layer, which provides the network with information. In this application, the input layer contains information about the current game state, such as the player's dice, the current bet, and previous bets. The following layers are the hidden layers. The nodes in these layers take information from the layer before them, aggregate it, and perform calculations before passing it to the next layer. The final layer is the output layer, which contains the result of the network's calculations. Networks improve by adjusting key elements of the calculations performed in each layer to fit the desired output better.

Genetic algorithms allow for unsupervised training of neural networks. Instead of detailing every desired output for each possible input set to train a network, the genetic algorithm simplifies feedback through a network’s score. For example, in the case of Liar's Dice, one implementation could award networks with more wins a higher score (fitness). This approach avoids evaluating the desired response for each game position. However, as networks aren’t assessed for every possible situation, a population of networks is required to expand the range of explored solutions. The networks' training iteratively improves by generating populations modelled on the best performers of the previous generation.

The NEAT algorithm, briefly described in the literature review, is an evolutionary genetic algorithm that allows new network topologies to develop. The following diagram demonstrates how networks structures can evolve through the addition/removal of hidden nodes (H) and connections signified by arrows.

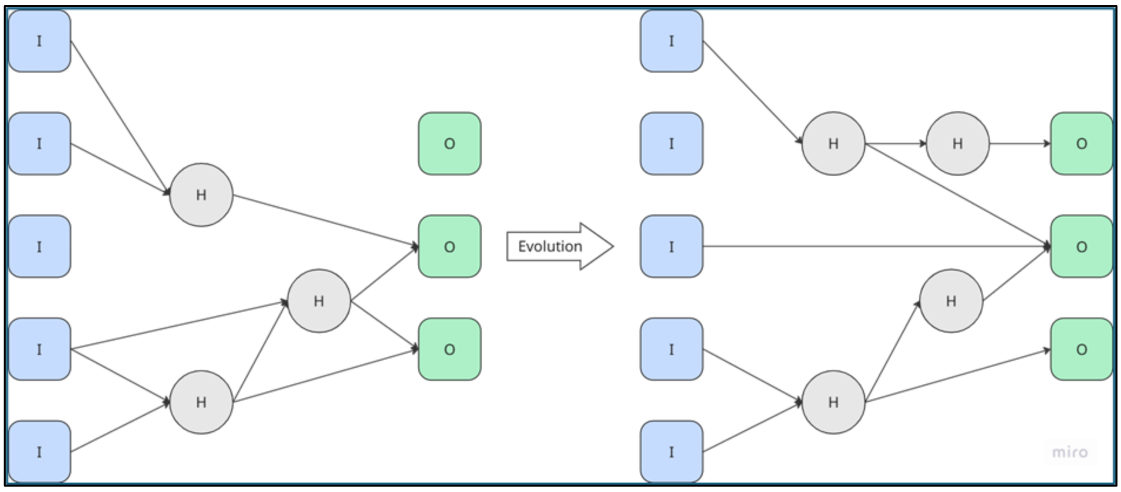

Figure 1 – NEAT Topologies

The ability to evolve the network topology allows for greater diversity in network development. As required, networks may add/remove components based on where complex "thought" is required. The rate of change of the networks can be customised to fit the use case best. Various parameters determine this mutation rate, such as the probability of adding/removing connections, nodes, or changing bias, activation, and aggregation functions. Populations with very high mutation rates will explore possible network structures more freely, which can improve the likelihood of finding an effective network. The risk with higher mutation is that potentially good networks may not get enough time to evolve due to too much change between generations.

A large population means greater network diversity, increasing the number of approaches evaluated and, in turn, increasing the probability of generating an effective network. This project's implementation uses a population size of 512. A large population such as this allows for less aggressive mutation parameters, as there is less need for broad evolutionary exploration.

## 3.2 Model A

The philosophy for the first neural network is to influence its evolution and learning as little as possible. The reasoning behind this is to develop a model to assess the NEAT algorithm's raw capabilities in complex games effectively. The NEAT implementation process consists of three main stages: Configuration, Game Implementation, and Training Methodology.

However, each stage affects the other and must be considered as a whole when working through the process.

## Configuration

Configuration decides how the network will be structured and its evolution style. The structural considerations relate to what information the network will require, what outputs are required of it, and a general assessment of the problem's complexity. Firstly, to align with the design philosophy, only the most essential and obvious game information is provided to the network: the player's dice, the current bet, and the previous bet. The aforementioned information set requires five input nodes for the most simple version of the game (1 dice per player). In addition to these five nodes, a random number between -1 and 1 is provided, as well as a constant 1. These two additional nodes provide the network with a source of randomness and a constant. The randomness component is essential as it allows the potential for the network to develop a mixed strategy. This is due to a mixed strategy requiring varied actions when given the same game inputs. The constant input is an additional calculation resource which can improve network performance.

The next consideration relates to the network's desired outputs. As two actions are possible in the game, all the network's outputs will directly determine the player's actions. One node will determine whether the player calls, and the other two nodes will determine the bet quantity and value. All outputs are restricted between 0 and 1 to allow for easy game implementation.

Finally, the initial number of hidden layers available to the network requires consideration. More layers allow the network to develop more complex thoughts. The downside to a high initial depth is that it requires significantly longer training as there are far more possible networks. For this problem, based on other similar implementations, the networks will be initialised with two hidden layers (b2developer, 2022, Fitzpatrick, 2024). The following diagram visualises the complete network structure for a game where players have 1 die each, additional dice require additional input nodes.

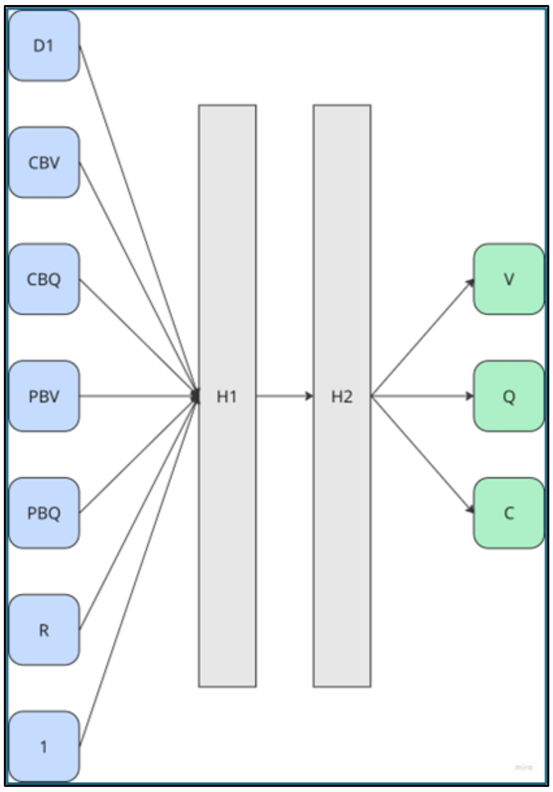

Figure 2 – Model A Configuration

Inputs: D1 = Player's Die, CBV = Current Bet Value, CBQ = Current Bet Quantity, PBV = Previous Bet Value, PBQ = Previous Bet Quantity, R = Random Number Between -1 and 1, 1 = Constant Integer.

H1, H2 represent the hidden layers

Outputs: V = Bet Value Output, Q = Bet Quantity Output, C = Bet Call Output.

The network's evolution style also requires configuration. As mentioned in the description of the NEAT algorithm, the more aggressive the evolutionary mutations, the more likely the model will explore a wide range of solutions, decreasing its probability of stagnation. The less aggressive configuration will have the opposite result with the upside of allowing more time for optimising networks to reach their potential. Similar NEAT projects and the NEAT algorithm documentation informed the network's evolutionary configuration. The resulting configuration takes a moderate to conservative approach. The population size mitigates this slight conservatism, which allows for sufficient network diversity.

## Game Implementation

The network must now be connected to the game code to inform it of the current game state and take game actions. As outlined above, the game code tracks the bet history and the player's dice hands. This information passes through the networks via their activation function. The resulting output of the networks then triggers actions in the game code. First, the call output is checked; a call is made if it exceeds 0.5. If the call is not greater than 0.5, the value and quantity outputs are converted to usable integers. As both lie between 0 and 1, they must be scaled up and evenly distributed across their feasible ranges. For example, bet quantity can be between 1 and the number of dice in the game; for games with 10 dice, the output is multiplied by 9, rounded to the nearest integer and added to 1. With this conversion complete, a bet is made in the game code with the corresponding quantity and value outputs.

Networks play against each other, with actions taken sequentially until one player calls, upon which a winner is determined.

## Training Methodology

As described in the NEAT algorithm section, the network population improves over time by using the networks with the highest fitness score from the previous generation as templates for the next generation. Networks are assigned fitness scores based on performance. Ideally, networks would be evaluated under the same game conditions; however, as networks play each other with differing strategies, each game will differ slightly. An alternative approach inspired by B2Studios' project is to develop a tournament structure where networks' fitness is determined based on where they finish in the tournament (b2developer, 2022).

Each generation's network population first plays 1000 games against a pure probabilistic strategy outlined in section 4.4 the top 50% of networks are then fed into the tournament structure. This initial seeding round provides a consistent network performance measure across the training process. The top 50% of networks from this seeding round are paired off and play 1000 games against each other, with the network with the most wins advancing to the next round. This process repeats until one network is the champion. The 1st generation's fitness is based on what round in the tournament the networks finish. The previous generation's champion is preserved in the following generation, where it competes in a new tournament. The fitness of the second generation onwards is determined by the network's relative position to the previous generations' champion. For example, suppose a network makes it to a round further than the previous champion; in that case, their fitness is calculated to be the previous champion's score increased by the number of rounds further they made it. This approach ensures networks only receive a higher fitness if they improve upon the previous generation's best network. The algorithm runs for 200 generations, outputting the champion of the final tournament as the best network. This tournament training process is carried out for each level of game complexity. Each training of 200 generations simulates 151.8 million games.

## 3.3 Model B

The second network philosophy is the opposite of the first. This network receives significantly more information and guidance. Unlike the first networks, where outputs directly determined actions, the outputs for model B denote probabilities with which actions should be taken. As before, there are three stages of implementation, with the third stage of training identical between the models. Configuration and Game implementation require slight adjustments to align with the design philosophy.

### Configuration

The primary difference between the models regarding the information provided is that model A receives raw game information, whereas model B receives additional processed information. Model B receives a count of its most common dice and most common dice above the current bet value. Secondly, the network receives dice probability information. These probability inputs consist of the probability that the opponent has the required dice for the current bet and the probability that the opponent has the dice for the subsequent bet increase. The following formula calculates these inputs;

$$p(r) = \sum_{r}^{d} \binom{d}{r} * \left(\tfrac{1}{6}\right)^{r} * \left(\tfrac{5}{6}\right)^{d-r}$$

where d represents the number of dice in the opponent's hand, and r represents the required number of dice.

The input values for the game state depicted below are given at the top of the diagram to give an example of how these inputs work. MC denotes the count of the player's most common dice, in this case, 3. MCA denotes the count of the player's most common dice above the current bet value, in this case, 1, as the player only has one of each dice (4, 6) above 3. PC denotes the probability that the opponent has the required dice to cover the current bet. As the current bet is 2x ⚂, the probability corresponds to the likelihood that the opponent will have at least one ⚂. Finally, PN denotes the highest probability of the next bet. In this case, the next bet with the highest probability of the opponent covering the required dice is 3x ⚀'s, as the player covers the bet by themselves.

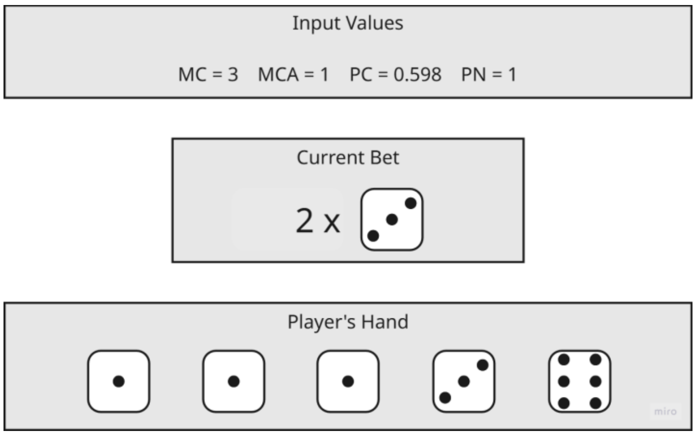

Figure 3 – Model B Game State

As mentioned in the overview, the output of this network determines the probability of the actions chosen. The betting action is split into bets where the value is raised and bets where the quantity is raised. The network for this desired output implementation requires two nodes. The first determines the probability of a call being made. The second determines the probability of an increased value bet. The remaining probability (the difference between 1 and the sum of both output nodes) determines the increased quantity bet probability. The full network configuration is displayed in the diagram below.

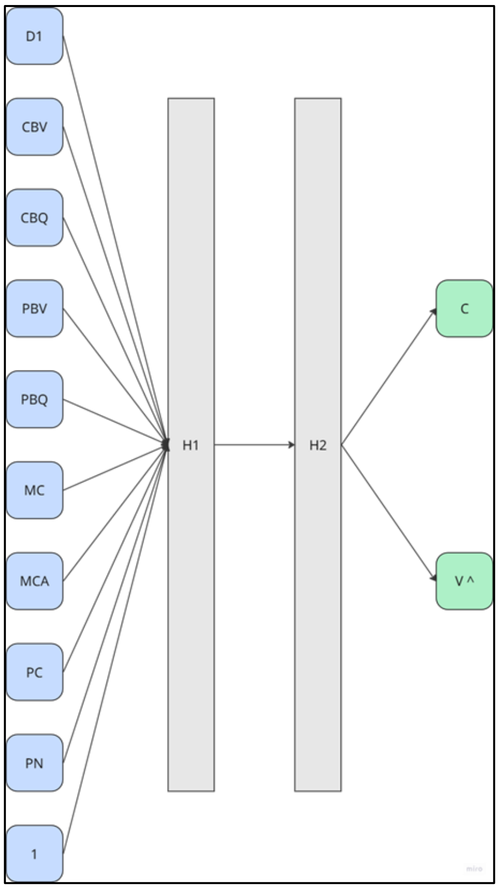

Figure 4 – Model B Configuration

Inputs: D1 = Player’s Die, CBV = Current Bet Value, CBQ = Current Bet Quantity, PBV = Previous Bet Value, PBQ = Previous Bet Quantity, MC = Count of Most Common Dice, MCA = Count of Most Common Dice above CBV, PC = Probability opponent covers Current Bet, PN = Probability opponent covers Next Bet, 1 = Constant Integer.

H1, H2 represent the hidden layers.

Outputs: V^ = Increased Value Bet Probability, C = Call Probability.

## Game implementation

Formulae calculate the required probability inputs as the network configuration outlines. The most common count information is derived from the player's hands.

The network outputs convert to game actions through a decider variable. The decider variable is randomly chosen from a uniform distribution between 0 and 1. A call is made if the value is less than or equal to the network's C output value. Should the value be greater than output one but less than or equal to the V^ output value, a bet is placed where the value increases to equal the player's most common die above the current value. Finally, should the decider variable be larger than both outputs a bet of increased quantity is made with the player's most common value.

This game implementation explicitly provides the network strategy with a mixed strategy, as actions within the same game state are taken with varying probabilities. This direct use of randomness removes the requirement for a random variable in the network's inputs.

Training methodology

As stated, the training methodology for both networks are identical. They have a seeding round followed by a tournament structure where round progression and win percentage determine fitness scores.

## 3.4 Probability Strategy

A pure probability strategy is a consistent competitor against which to test the models as they progress through and complete training. All actions for this strategy are taken based on dice probabilities. This strategy can achieve high win percentages; its weakness is that it is obvious and highly exploitable. Unlike mixed strategies, where players may choose different actions for the same game state, a pure strategy takes the same action when provided with the same game position.

To derive a probabilistic strategy, similar to those in Johnson's paper and Taylor's textbook, one must begin with endgame positions and work backwards (Johnson, 2007, Taylor, 2014). This method allows for backpropagation as earlier game positions incorporate the outcomes of endgame calculations. In the most straightforward game case where both players have one die, the maximum bet is 2x ⚅. This bet will always be called as there are no other possible actions, so its win probability can be easily calculated. Win probabilities are determined by the likelihood of the opposition's hand containing the die required to cover the current bet. If a player bets 2x ⚅, their win probability is zero if they do not hold a six or 1/6 if they hold a six. These results inform the win probabilities for betting 2x ⚄. Should a player bet 2x ⚄, their opposition can call or bet 2x ⚅ as betting 2x ⚅ win probabilities are worse than calling a bet of 2x ⚄, which at the lowest is 5/6, one can infer the opposition will call, resulting in a win probability for a bet of 2x ⚄ of 0 if they do not hold a five or 1/6 if they hold a five. This process can more clearly be seen in the diagram below. Where the grey boxes represent the current bet, green boxes show the possible actions when faced with the current bet and P(W) signifies the win probability associated with each action.

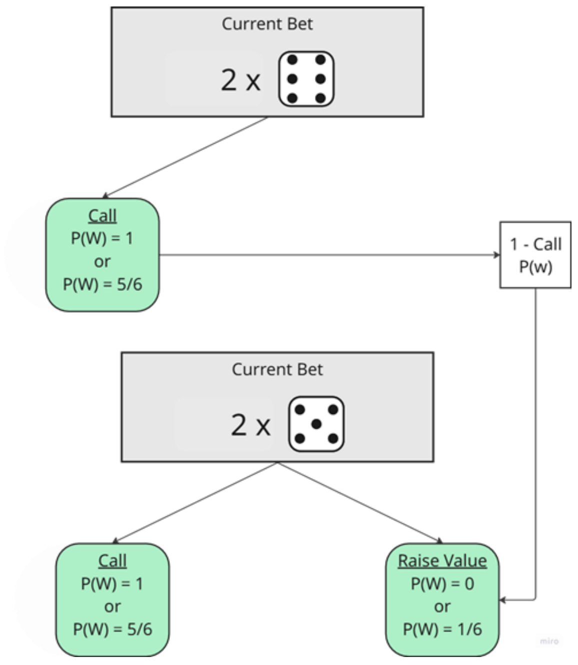

Figure 5 – 2 Dice Game Probability Tree

The methodology does not change significantly for games with more dice. The diagram below outlines the decision tree for a game with 10 dice.

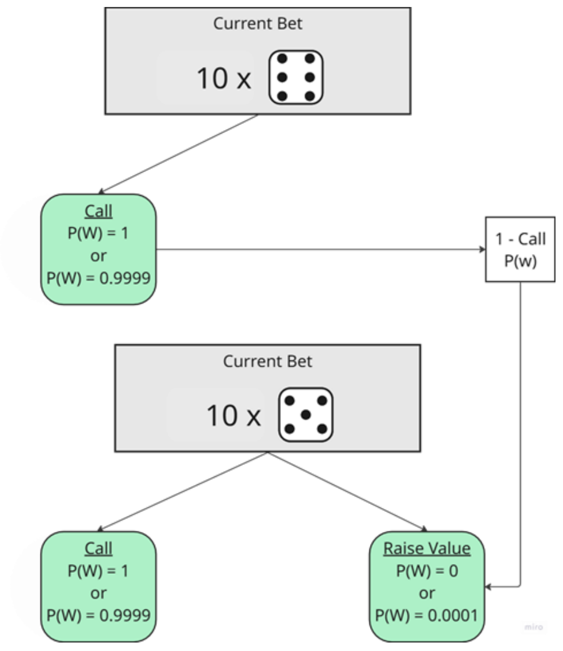

Figure 5 – 10 Dice Game Probability Tree

The player playing with the pure probability strategy takes the action associated with the highest win probability. As mentioned previously, the weakness of this strategy is that it can be easily countered. An opponent can work backwards through the decision tree to infer information about the pure strategy player's dice. Additionally, this approach does not factor in any potential information their opponent's bets provide. For example, an opponent making a high quantity bet might suggest that they hold a significant amount of dice corresponding to the bet value. The probability strategy does not take advantage of this information transfer, resulting in a suboptimal strategy. However, despite its shortcomings, the probability strategy is still highly effective as it makes no fundamental errors, and its weaknesses require time to identify and capitalise on.

# 4 Testing and Empirical Strategy

As this capstone aims to investigate how the NEAT algorithm learns and performs in incomplete information games at varying levels of complexity, several tests and analytical models are required to evaluate its performance. Two distinct tests inform evaluations of both models' improvement. The first test compares the best network from every 10th generation's population against the probabilistic strategy across 10,000 games. The second test is inter-generational as it competes the same selection of networks against each other. Both tests provide data to assess how the networks' strategies evolve throughout training. In addition to the data provided by the tests above, significant topology evolutions between generations are recorded throughout the training process. Such a change in topology can be seen between generation 10 and 50 for Model B in the 4v4 dice game displayed in the diagram below. This tracking could provide insight into how new network structures affect performance.

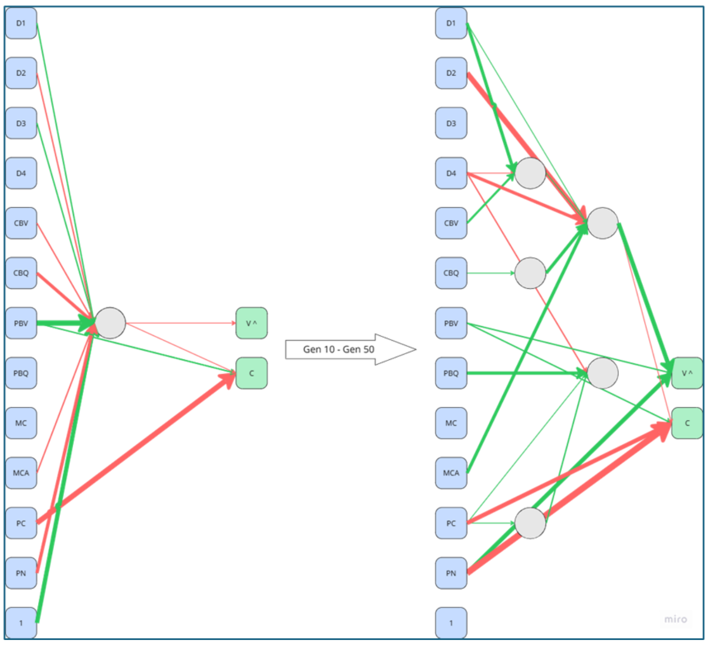

Figure 7 – Model B 4v4 Evolution

Two logistic regression models utilise the data above to extract insights. Logistic regression models evaluate how various factors affect the likelihood of a binary outcome. For example, a logistic regression could predict the probability of rain based on cloud coverage. The regression in this example would take data of days with and without rain and their corresponding cloud coverage. Through maximum likelihood estimation, the model determines the coefficient to best fit the data. The logistic model can be adapted to incorporate non-linearity by including polynomial turns and interaction terms. Logistic regression, as applied in this dissertation, requires four assumptions: a binary response variable, independent observations, no multicollinearity, and a large sample size.

## 4.1 Regression 1 – Model A vs Model B

The first regression evaluates training metrics and probability strategy tests to determine what factors best predict performance. The main goal of this regression is to investigate any potential differences between the models' training. The logistic regression equation takes the form:

$$log\left(\frac{p(Y)}{1-p(Y)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 (X_1 * X_2) + \beta_5 (X_1^2 * X_2) + \beta_6 X_3$$

Where:

- Y is the model’s test wins against the probability strategy.
- $X_1$ is the generation in which the test occurred.
- $X_1^2$ is the generation variable squared to capture potential non-linearity, such as a decreasing rate of performance improvement.
- $X_2$ is a binary dummy variable indicating the model (Model 1 = 0, Model 2 = 1).
- $X_1 * X_2$ is an interaction variable that aims to capture differences in model performance at different generations. It is generated from the product of $X_1$ and $X_2$.
- $X_1^2 * X_2$ is an interaction variable similar to that above but incorporates the squared generation.
- $X_3$ is a dummy variable denoting topological change in the last 10 generations.

Interpreting the coefficients:

- $\beta_1$ represents the effect the network’s generation has on the likelihood of winning.
- $\beta_2$ similarly represents the generation effect; however, nonlinearly.
- $\beta_3$ captures the difference between model performance. A positive coefficient could indicate that Model 2 is more likely to win against the probability strategy.
- $\beta_4$ indicates differences in model performance in terms of generation.
- $\beta_5$ operates similarly but differs by capturing non-linearity over generations.

- $\beta_6$ represents how topological evolutions affect network performance.

The model assumptions are easily checked to be satisfied. The outcome variable is binary 1/0 for win/loss. All outcomes are independent, as the previous game's outcome does not affect the next. Multicollinearity tables demonstrate the assumption is satisfied (see Tables A1-A5 in the appendix). Finally, the sample size requirement is also satisfied, as there are 200,000 game results per model.

## 4.2 Regression 2 – Inter-Generation Comparison

The second regression is applied to each model individually. The regression aims to capture whether the model improves during training compared to itself. For this, it utilises the data generated by the inter-generation test. The second logistic equation takes the form:

$$log\left(\frac{p(Y)}{1-p(Y)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Where:

- Y is the number of wins for the network A against network B
- $X_1$ is the generation difference between A and B (A-B)
- $X_2$ is an interaction variable calculated from the product of the generation difference and the generation average. This term aims to capture the effect of generation difference based on the overall age of the networks

Interpreting the coefficients

- $\beta_1$ represents the effect increased training has on network performance
- $\beta_2$ similarly represents the generation effect. However, this coefficient should evaluate whether generation difference has a more significant effect on win likelihood for younger or older networks.

# 5 Results

## 5.1 Logistic Regression 1 – Model A v Model B

Table 1: 1v1 Logistic Regression Model Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen | $0.009^{***}$ |
| | (0.0003) |
| $\text{Gen}^2$ | $-0.00002^{***}$ |
| | (0.00000) |
| Model | $1.749^{***}$ |
| | (0.023) |
| Interaction | $-0.009^{***}$ |
| | (0.0005) |
| $\text{Interaction}^2$ | $0.00003^{***}$ |
| | (0.00000) |
| Topology | $-0.133^{***}$ |
| | (0.008) |
| Constant | $-0.823^{***}$ |
| | (0.016) |
| McFadden $R^2$ | 0.804 |
| RMSE | 0.070 |
| Observations | 40 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 2: 2v2 Logistic Regression Model Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen | $0.005^{***}$ |
| | (0.0004) |
| $\text{Gen}^2$ | $-0.00002^{***}$ |
| | (0.00000) |
| Model | $2.574^{***}$ |
| | (0.026) |
| Interaction | $-0.001$ |
| | (0.001) |
| $\text{Interaction}^2$ | 0.00000 |
| | (0.00000) |
| Topology | $-0.163^{***}$ |
| | (0.009) |
| Constant | $-0.931^{***}$ |
| | (0.016) |
| McFadden $R^2$ | 0.97 |
| RMSE | 0.022 |
| Observations | 40 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 3: 3v3 Logistic Regression Model Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen | $0.003^{***}$ |
| | (0.0003) |
| Gen_Squared | $-0.00000$ |
| | (0.00000) |
| Model | $3.222^{***}$ |
| | (0.032) |
| Interaction | 0.001 |
| | (0.001) |
| Interaction_Squared | $-0.00002^{***}$ |
| | (0.00000) |
| Topology | $-0.164^{***}$ |
| | (0.010) |
| Constant | $-0.837^{***}$ |
| | (0.016) |
| McFadden RSquared | 0.975 |
| RMSE | 0.020 |
| Observations | 40 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 4: 4v4 Logistic Regression Model Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen | $0.003^{***}$ |
| | (0.0003) |
| Gen_Squared | $-0.00001^{***}$ |
| | (0.00000) |
| Model | $3.889^{***}$ |
| | (0.041) |
| Interaction | $-0.003^{***}$ |
| | (0.001) |
| Interaction_Squared | $0.00001^{**}$ |
| | (0.00000) |
| Topology | $-0.153^{***}$ |
| | (0.011) |
| Constant | $-0.702^{***}$ |
| | (0.016) |
| McFadden RSquared | 0.98 |
| RMSE | 0.017 |
| Observations | 40 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 5: 5v5 Logistic Regression Model Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen | 0.0005 |
| | (0.0003) |
| Gen_Squared | −0.00000 |
| | (0.00000) |
| Model | 4.281*** |
| | (0.053) |
| Interaction | −0.0001 |
| | (0.001) |
| Interaction_Squared | 0.00000 |
| | (0.00001) |
| Topology | −0.264*** |
| | (0.010) |
| Constant | −0.422*** |
| | (0.015) |
| McFadden $R^2$ | 0.962 |
| RMSE | 0.027 |
| Observations | 40 |
| *Note:* | *p<0.1; **p<0.05; ***p<0.01 |

Across game difficulties many coefficients are statistically significant at the 1% level. This high statistical significance level indicates that variables are significant in predicting win likelihood. The generation coefficient positivity confirms as expected that win likelihood increases with training. However, its statistical significance disappears for the highest difficulty game indicating that training becomes less significant for model performance. This sudden change could be due to the exponential increase in game states relative to dice increases (see Figure A2 in the appendix). The squared generation coefficient indicates this relationship is non-linear, which from the negative sign can be interpreted as a decreasing rate of improvement through training. The variable with the largest coefficient for all game difficulties is the model dummy variable.  This implies a sizeable difference in performance between the models with model B being the stronger of the two. There are many potential reasons for this difference. Firstly, model B is provided with significantly more information about the game state including probability values. Secondly, model B's game implementation ensures all game actions it takes are both legal and probabilistically optimal. In comparison to Model A's implementation which allows illegal actions. This openness while allowing for a less supported assessment of the NEAT algorithm could hamper performance especially early in training when the population has little game understanding. However, a potential side effect of this less restrictive implementation is greater performance improvement during training. The interaction variable negative coefficient indicates that Model A win likelihood improves more than Model B's during training. Similar to the generation coefficients the statistical significance here deteriorates as would be expected as it represents the interaction between model type and generation. While Model B is clearly significantly stronger than A, a more rigid implementation could result in less room to evolve. Another interesting implication of the regression results is the significance of the Topology variable. The negative coefficient implies that recent topology changes negatively affect win likelihood. A potential interpretation of this is that younger topologies may offer improved performance in the long run but take time to effectively optimise.

In terms of the models' accuracy and robustness the RMSE across models is low indicating a strong fit with the data. The pseudo $R^2$ value is very high indicating the model captures the majority of the variance in the data.

## 5.2 Logistic Regression 2 – Inter-Generational Competition

Table 6: 1v1 Inter-Generational (A) Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | $0.006^{***}$ |
| | (0.0002) |
| Interaction | $-0.0001^{***}$ |
| | (0.00000) |
| Constant | $-0.208^{***}$ |
| | (0.005) |
| McFadden $R^2$ | 0.076 |
| RMSE | 0.096 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 7: 1v1 Inter-Generational (B) Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | 0.0001 |
| | (0.0002) |
| Interaction | −0.00000 |
| | (0.00000) |
| Constant | 0.002 |
| | (0.005) |
| McFadden $R^2$ | 0 |
| RMSE | 0.006 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 8: 2v2 Inter-Generational (A)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | $-0.002^{***}$ |
| | (0.0002) |
| Interaction | 0.00000 |
| | (0.00000) |
| Constant | 0.006 |
| | (0.005) |
| McFadden $R^2$ | 0.194 |
| RMSE | 0.048 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 9: 2v2 Inter-Generational (B)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | $0.0005^{**}$ |
| | (0.0002) |
| Interaction | $-0.00000$ |
| | (0.00000) |
| Constant | 0.004 |
| | (0.005) |
| McFadden $R^2$ | 0.013 |
| RMSE | 0.009 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 10: 3v3 Inter-Generational (A)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | $0.002^{***}$ |
| | (0.0002) |
| Interaction | $-0.00002^{***}$ |
| | (0.00000) |
| Constant | $0.017^{***}$ |
| | (0.005) |
| McFadden $R^2$ | 0.033 |
| RMSE | 0.020 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 11: 3v3 Inter-Generational (B)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | 0.0001 |
| | (0.0002) |
| Interaction | $-0.00000$ |
| | (0.00000) |
| Constant | 0.002 |
| | (0.005) |
| McFadden $R^2$ | 0 |
| RMSE | 0.006 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 12: 4v4 Inter-Generational (A)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | 0.002$^{***}$ |
| | (0.0002) |
| Interaction | −0.00002$^{***}$ |
| | (0.00000) |
| Constant | 0.008 |
| | (0.005) |
| McFadden $R^2$ | 0.059 |
| RMSE | 0.041 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 13: 4v4 Inter-Generational (B)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | −0.0002 |
| | (0.0002) |
| Interaction | 0.00000 |
| | (0.00000) |
| Constant | 0.002 |
| | (0.005) |
| McFadden $R^2$ | 0.001 |
| RMSE | 0.005 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 14: 5v5 Inter-Generational (A)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | 0.002$^{***}$ |
| | (0.0002) |
| Interaction | −0.00002$^{***}$ |
| | (0.00000) |
| Constant | −0.025$^{***}$ |
| | (0.005) |
| McFadden $R^2$ | 0.074 |
| RMSE | 0.034 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

Table 15: 5v5 Inter-Generational (B)
Logistic Regression Summary

| | *Dependent variable:* |
|---|---|
| | Wins (out of 10,000) |
| Gen Diff | −0.0002 |
| | (0.0002) |
| Interaction | 0.00000 |
| | (0.00000) |
| Constant | 0.004 |
| | (0.005) |
| McFadden $R^2$ | 0.001 |
| RMSE | 0.005 |
| Observations | 190 |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

It is clear from the McFadden $R^2$ values of the second regression results that inter-generational improvement cannot be definitively determined for either NEAT model. When comparing the regressions summary tables there is a clear contrast in the statistical significance of the coefficients. The majority of Model A's coefficients across game difficulties are statistically significant. This implies there is some correlation between generational difference and win likelihood for Model A, however the overall regressions strength diminishes the inferential ability of these results. The stark difference in coefficient statistical significance between Model A and B suggests as seen in the interaction terms of regression 1 that Model A shows better learning ability despite its relative weakness. The limited analytical power of the regression results indicate that connection between more training time and win likelihood is not particularly strong. This suggests that generations don't improve over all their predecessors. While this appears to contradict the results of the first regression there are key differences between the tests that offer some guidance to interpret the results. The 1st regression compares networks against a consistent opponent whereas the opponents vary in the second test. This points to a potential game theoretic explanation of the contrasting results. As the training tournament structure rewards networks for developing strategies that are successful against the previous generation's strategies, this process may not result in iterative improvements. The best response strategy against the previous generation may be vulnerable to strategies from much earlier in the process. The second regression's results point to a key flaw in self-play training for game theory optimal performance. To properly assess a strategy's strength, it must be compared against all other possible strategies, this intuition is derived from the fact that Nash equilibrium strategies must be each other's best response. As networks aren't tested against all strategies developed during training their performance cannot be effectively evaluated and returned through fitness score.

## Summary

The 1st regression demonstrates that NEAT networks can learn and improve over time by testing generations against a consistent opponent for the less complex game classifications. The difference between the models suggests increased guidance through game implementation successfully improves performance. The cost of this implementation approach is a restriction in strategy diversity resulting in limited development. The 2nd regression crucially highlights the weakness of the training methodology. As generations are not privy to lifetime strategies they cannot effectively develop increasingly less exploitable strategies. This issue presents itself through the insignificance of more training on intergenerational competition.

# 6 Conclusion

As this dissertation's introduction outlines, this project focuses on the NEAT algorithm's ability to learn the imperfect information game Liar's Dice. Two separate algorithm implementation approaches were used to perform this investigation. Each model was trained in five game difficulty levels culminating in a total of 1.518 billion games played. The hypothesis associated with the papers goal was that NEAT could show learning ability with decreasing improvement for higher game difficulty levels. The results indicate that the hypothesis was generally accurate. Both models demonstrated increased performance against a consistent opponent through training for 4 out of 5 difficulty levels. Models for the most advanced games struggle to improve performance. The inter-generational test provided important nuance to analysing the networks' improvement. The lack of correlation between increased training and win percentage against younger generations points to weaknesses with the self-play training methodology and the need for additional testing mechanisms to evaluate improvement more concretely.

The regression results, in combination, highlight the key dynamics and challenges in developing AI to play game theoretic style games, specifically imperfect information games. As described in the introduction, games with similar structures to Liar's Dice, such as poker, and bridge, require opponents to disguise their private information, in this case, dice, through mixed strategies. Ineffective masking allows opponents to develop strategies to exploit one's strategy. Nash equilibrium strategies ensure consistent performance across all opposing strategies. The difference in approach between counterfactual regret minimisation (CFR) and opponent modelling represents opposing sides of this balance. CFR develops a strategy with the best performance against its strongest opposing strategy. This results in consistently high performance but does not capitalise effectively on weak opposition. Opponent modelling exploits opponents by developing custom strategies. This requires time to learn the opponent's strategy and requires new strategies for every opponent, as opposed to CFR, which is effective against all. The results of the inter-generational testing suggest that the training approach used in this dissertation shares similarities with opponent modelling. Networks' fitness scores increase by developing effective strategies against the previous generation, not all generations. As with opponent modelling, this results in effective strategies against specific opponents, but they may be vulnerable to alternative strategies from earlier in training. This evolution dynamic makes it challenging to measure definitive improvement from training.

These findings add to the available literature by providing insight into the strengths and weaknesses of genetic algorithms such as NEAT as tools to explore imperfect information game strategies. Where previous research focused on solving games this paper focuses on the progress models make during the learning process. Additionally, the results highlighted potential issues with assessing fitness through self-play. This training weakness emphasises the benefit of incorporating game theory principles into the training process albeit with a higher computational cost as seen in Ahle’s testing process (Dybdahl Ahle, 2022).

## Limitations

As detailed above, the self-play tournament training strategy struggles to evaluate strategy development properly. A more effective measure of strategy strength is strategy exploitability. Testing models against the strongest opposing strategy provides a universal metric for models' vulnerability, which could better compare the generation's strengths. The challenge with this approach is the substantial computing requirements to develop opposing strategies effectively. In Ahle’s paper, only strategies for a game with a maximum of 3 dice could be tested (Dybdahl Ahle, 2022). This would exclude all but 1 of the difficulty levels modelled in this capstone.

A second limitation of this project is the number of NEAT models developed. More expansive model configurations and implementations would provide a more comprehensive view of the NEAT algorithm's abilities.

## Areas for Future Research

Implementing NEAT using exploitability to determine fitness values would provide a fascinating insight into a potential CFR alternative. Utilising the opponent modelling approach seen in Li and Miikulainen’s paper could potentially reduce the computational barrier to exploitability evaluation (Li and Miikkulainen, 2018).

Developing NEAT networks without self-play by rewarding networks that better mirror strong strategies such as the probabilistic or CRF models. This approach would resemble the NEAT XOR model in the documentation, which proved highly effective.

Adjusting the seeding round approach to test networks against the best networks of all previous generations could support iterative improvement. This would require significantly more training time and memory storage but could provide sufficient context to the fitness values to ensure networks improve on all previous generations.

# Bibliography

Alushi, K. (2022, October 27). Exploration of Reinforcement Learning Methods when Training a Dice Game Playing Agent. *BSc diss.* Hochschule für Angewandte Wissenschaften Hamburg. https://reposit.haw-hamburg.de/bitstream/20.500.12738/16102/1/BA_Exploration_Reinforcement_Learning_Methods.pdf

b2developer. (2022, January 12). *MonopolyNEAT*. Retrieved from Github: https://github.com/b2developer/MonopolyNEAT.

Bursci, P. a. (2014). Endgame Strategies and Simulation Results for the Liar's Dice Game. *STUDIA UNIV. BABES¸–BOLYAI, INFORMATICA*, 47-58. https://www.cs.ubbcluj.ro/~studia-i/contents/2014-macs/04Burcsi.pdf

Dybdahl Ahle, T. (2022, January 9). *Liar's Dice by Self-PLay*. Retrieved from Towards Data Science: https://towardsdatascience.com/lairs-dice-by-self-play-3bbed6addde0/

Ferguson, C. P., & Ferguson, T. S. (1991). Models for the Game of Liar's Dice. In T. E. Raghavan, T. S. Ferguson, T. Parthasarathy, & O. J. Vrieze, *Stochastic Games And Related Topics* (pp. 15-28). Dodrecht: Springer. https://doi.org/10.1007/978-94-011-3760-7_3

Ferguson, T. S. (1970, April). On a Class of Infinite Games Related to Liar Dice. *The Annals of Mathematical Statistics 41, no. 2*, pp. 353-362. https://doi.org/10.1214/aoms/1177697075

Fitzpatrick, J. (2024, June 14). *poker-bot*. Retrieved from Github: https://github.com/Jason-Fitzpatrick1/poker-bot

Freeman, G. H. (1989). The Tactics of Liar Dice. *Journal of the Royal Statistical Society. Series C (Applied Statistics).*, 38(3), 507-516. https://www.jstor.org/stable/2347737

Johnson, T. S. (2007). An evaluation of how Dynamic Programming and Game Theory are applied to Liar's Dice. *BSc diss.* Rhodes University. https://www.cs.ru.ac.za/research/groups/vrsig/pastprojects/040learning/paper02.pdf

Lanctot, M., Waugh, K., Zinkevich, M., & Bowling, M. (2009). Monte Carlo sampling for regret minimization in extensive games. *Advances in neural information processing systems, 22*. https://dl.acm.org/doi/10.5555/2984093.2984215

Li, X., & Miikkulainen, R. (2018). Opponent modeling and exploitation in poker using evolved recurrent neural networks. *Proceedings of the Genetic and Evolutionary Computation Conference*, (pp. 186-196). https://dl.acm.org/doi/10.1145/3205455.3205589

*NEAT-Python Documentation*. (n.d.). Retrieved November 17, 2024, from https://neat-python.readthedocs.io/en/latest/neat_overview.html

O. Stanley, K., & Miikkulainen, R. (2002). Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, 99-127. https://doi.org/10.1162/106365602320169811

Ruscica, T. (2024, October 17). *NETA-Pong-Python*. Retrieved from Github: https://github.com/techwithtim/NEAT-Pong-Python

Steinberger, E. (2019, October 4). Single Deep Counterfactual Regret Minimization. arXiv preprint arXiv:1901.07621. https://doi.org/10.48550/arXiv.1901.07621

Sum, J., & Chan, J. (2003). On a liar dice game - bluff. *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693) Machine learning and cybernetics Machine Learning and Cybernetics, 2003 International Conference on.* (pp. Vol.4 2179-2184). Piscataway. https://ieeexplore.ieee.org/document/1259867

Sutton, R. S., & Barto, A. G. (2015). *Reinforcement Learning: An Introduction.* The MIT Press.

Taylor, D. G. (2014). *The Mathematics of Games: An Introduction to Probability (1st ed.).* New York: Chapman and Hall/CRC.

Watkins, C., & Dayan, P. (1992). Q-Learning. *Machine Learning*, Vol. 8, 279-292. https://doi.org/10.1007/BF00992698

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2007). Regret Minimization in Games with Incomplete Information. *Advances in Neural Information Processing Systems 20.* https://proceedings.neurips.cc/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf

# Appendix

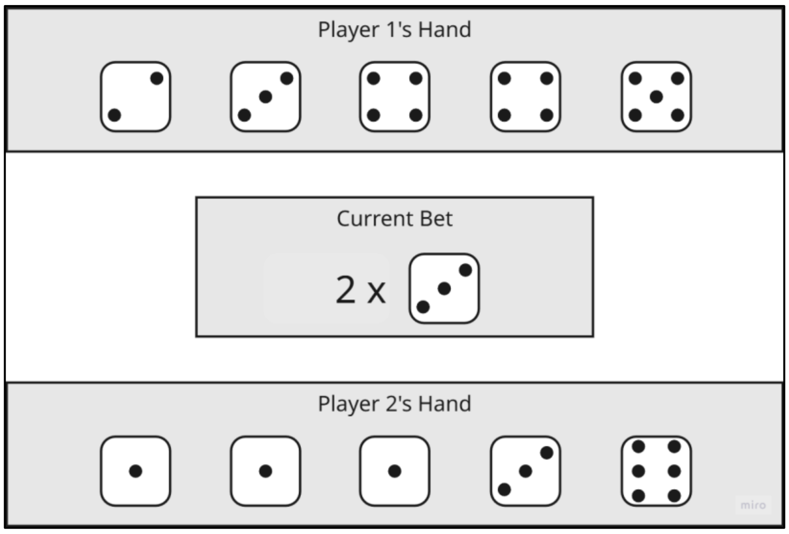

Figure A1 – Example Game State

| *1v1* | *Gen* | *Gen^2* | *Model* | *Inter* | *Inter^2* | *Topology* |
|---|---|---|---|---|---|---|
| Gen | 1 | | | | | |
| Gen^2 | 0.971348 | 1 | | | | |
| Model | 0 | 0 | 1 | | | |
| Inter | 0.433727 | 0.4213 | 0.789786 | 1 | | |
| Inter^2 | 0.532694 | 0.548407 | 0.631269 | 0.960654 | 1 | |
| Topology | 0.070088 | 0.032882 | 0 | 0.04777 | 0.052066 | 1 |

Table A1

| *2v2* | *Gen* | *Gen^2* | *Model* | *Inter* | *Inter^2* | *Topology* |
|---|---|---|---|---|---|---|
| Gen | 1 | | | | | |
| Gen^2 | 0.971348 | 1 | | | | |
| Model | 0 | 0 | 1 | | | |
| Inter | 0.433727 | 0.4213 | 0.789786 | 1 | | |
| Inter^2 | 0.532694 | 0.548407 | 0.631269 | 0.960654 | 1 | |
| Topology | -0.13016 | -0.15237 | 0.11547 | 0.039084 | -0.01448 | 1 |

Table A2

| *3v3* | *Gen* | *Gen^2* | *Model* | *Inter* | *Inter^2* | *Topology* |
|---|---|---|---|---|---|---|
| Gen | 1 | | | | | |
| Gen^2 | 0.971348 | 1 | | | | |
| Model | 0 | 0 | 1 | | | |
| Inter | 0.433727 | 0.4213 | 0.789786 | 1 | | |
| Inter^2 | 0.532694 | 0.548407 | 0.631269 | 0.960654 | 1 | |
| Topology | -0.09011 | -0.12643 | 0 | 0.004343 | -0.03784 | 1 |

Table A3

| *4v4* | *Gen* | *Gen^2* | *Model* | *Inter* | *Inter^2* | *Topology* |
|---|---|---|---|---|---|---|
| Gen | 1 | | | | | |
| Gen^2 | 0.971348 | 1 | | | | |
| Model | 0 | 0 | 1 | | | |
| Inter | 0.433727 | 0.4213 | 0.789786 | 1 | | |
| Inter^2 | 0.532694 | 0.548407 | 0.631269 | 0.960654 | 1 | |
| Topology | -0.05419 | -0.04462 | -3.5E-18 | -0.0188 | -0.02639 | 1 |

Table A4

| *5v5* | *Gen* | *Gen^2* | *Model* | *Inter* | *Inter^2* | *Topology* |
|---|---|---|---|---|---|---|
| Gen | 1 | | | | | |
| Gen^2 | 0.971348 | 1 | | | | |
| Model | 0 | 0 | 1 | | | |
| Inter | 0.433727 | 0.4213 | 0.789786 | 1 | | |
| Inter^2 | 0.532694 | 0.548407 | 0.631269 | 0.960654 | 1 | |
| Topology | -0.01001 | -0.01343 | 0 | -0.01303 | -0.00533 | 1 |

Table A5

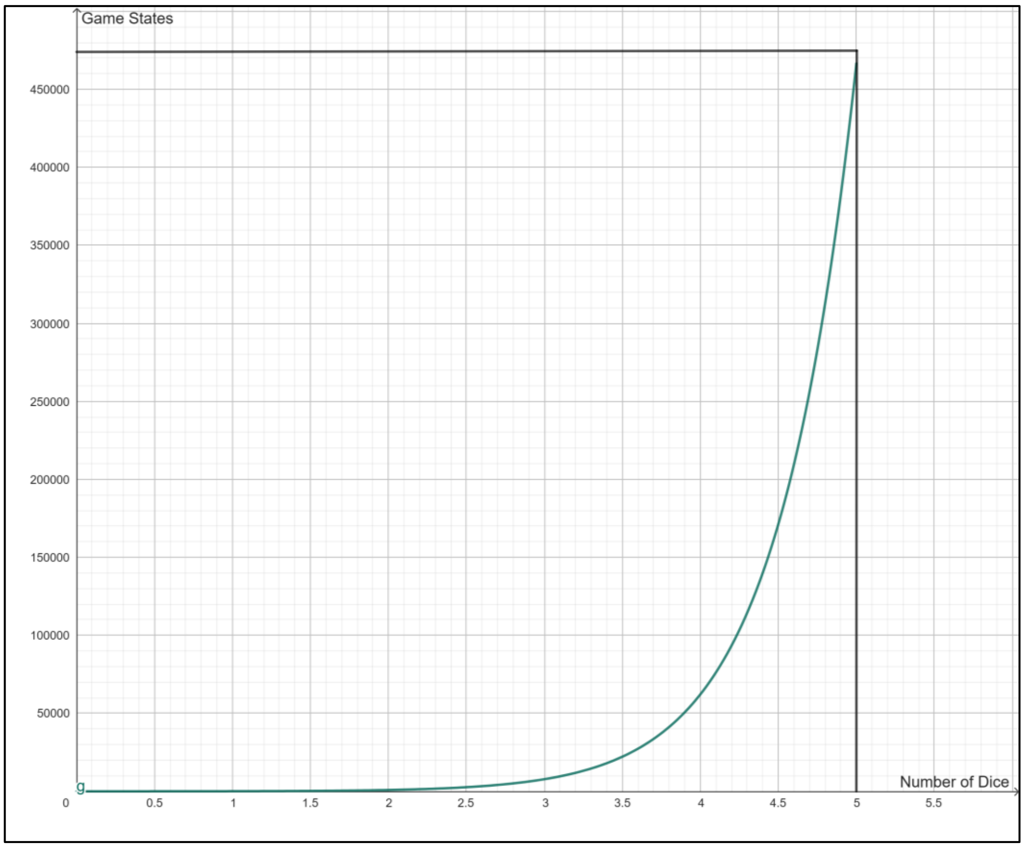

Figure A2 – Exponential Game States