Enhancing the Capabilities of Neuroevolution Ensembles via Gating Networks

Turner Burchard turnerburchard@gmail.com
Department of Computer Science
Montana State University
Bozeman, MT

Abstract

We propose a technique for combining NeuroEvolution of Augmenting Topologies (NEAT) networks into an ensemble through the use of a gating network. Using this model, a gating NEAT network can decide which network's output to use from the ensemble, based on a problem's inputs. By applying this method to various problems, improvements in prediction power can be clearly observed, particularly in difficult reinforcement learning. We confirm the technique's validity on the XOR problem, apply it to a modified pole balancing task, and finally display its learning power by training to play the game of Go. Though performance gains were not extreme in comparison to a single NEAT network, the ensemble method has clear potential for solving difficult learning problems when applied carefully.

Keywords: NEAT, Neural Networks, Gating Networks, Reinforcement Learning

1. Introduction

NEAT (Neuroevolution of Augmented Topologies) is a modern technique which has recently gained popularity for developing efficient neural networks. Unlike more traditional methods for neural network evolution, such as gradient descent or evolutionary algorithms, it functions by starting with a minimal network, consisting only of an input layer directly connected to an output layer. As the evolution process continues, mutation and recombination work to slowly build up a network, ostensibly consisting only of the necessary connections and optimal weights. NEAT additionally employs a unique form of evolution, with specially designed operators to ensure the diversity of the population, while still allowing for productive recombination and mutation. This technique has been successfully applied to a wide variety of problems, especially in the field of reinforcement learning, but it still has its downsides. Since NEAT begins with an essentially empty network, different individuals in a population can grow to have wholly differing topologies, and each learner may only be optimal for a subset of the data. This creates several issues. Namely, there tends to be high variation in NEAT solutions from one run to another, thanks to highly diverse populations. Secondly, a population may have high overall fitness, but the individual with the highest fitness may be consistently unsuccessful at classifying a subset of the data.

Ensemble methods have been used extensively in the machine learning community to mitigate these issues. By combining the predictions of multiple individuals, ensembles can improve overall accuracy and reduce the variance in solutions. In the context of NEAT, ensembles offer a promising solution to address the challenges of high variation and suboptimal performance. However, ensembles can be problematic, as it is difficult to combine their predictions in a way which takes all outputs into account without losing the ability to accurately classify in a variable environment.

In an attempt to solve these issues, we propose a form of NEAT ensemble which takes into account the predictions of all individuals, specifically through the use of a gating network. This approach allows us to create a population of specialists which can each accurately predict a specific range of problems, with a managing network to decide which specialist to ask. Our work contributes to the growing body of research on NEAT and highlights the potential of ensemble methods for improving the performance of neural network optimization techniques. By combining the strengths of NEAT and ensemble methods, we are able to create a powerful approach that can address the limitations of traditional NEAT and achieve state-of-the-art performance on benchmarks. We expect to find that the ensemble will have significantly increased computational complexities, but that its downsides will be outweighed by the heightened ability to find accurate predictions in a variety of scenarios.

2. Background

2.1 NEAT

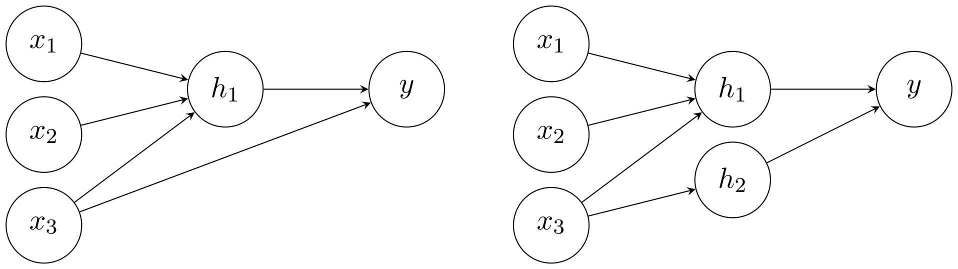

Figure 1: Example of a NEAT network structural mutation by adding a new node within an existing connection between input node x_3 and output node y.

Since the concept of perceptrons was first created, deciding on the topology of a network of neurons has been one of the hardest problems to solve (Rosenblatt, 1958) The best way to decide on specific details, such as the number of layers and nodes in each layer, has generally been tuning through trial-and-error. Since the 1990s, many have solved this issue with Topology and Weight Evolving Artificial Neural Networks (TWEANNs), which are designed to evolve both the weights of a network, and its topology simultaneously. Even basic NeuroEvolution (NE) strategies have been shown to be faster and more efficient than other solutions at solving continuous and high dimensional problems. They also excel in non-Markovian spaces, thanks to their ability to form basic representations of memory in recurrent connections (Gomez and Miikkulainen, 1999).

One of the most modern versions of this method is known as NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002), which presents novel solutions to a number of the constantly prevalent issues with most TWEANNs. NEAT uses mutation to alter both weights and structure, with new connections being assigned random weights, and new nodes using an input weight of one to reduce impact to the network.

Weights are mutated in the same way that other NE systems use, randomly choosing connections. Structural mutation occurs by either adding a new connection between two previously unconnected nodes (Figure 2.1), or by taking an existing connection and adding a new node within it. As a result of this, the genomes gradually grow larger.

The network is represented as a list of connections, which refer to two connected nodes, an in-node and an out-node, as well as the weight, whether it is activated, and the historical marker. Crossover is often an issue between networks with competing structures, so NEAT uses historical markers to help align structures during crossover, hugely increasing the algorithm's ability to maintain genetic information without complex calculations. This works by incrementing a global innovation number which is assigned to new genes, which is inherited throughout evolution. This allows the algorithm to know which genes match up with which, so in crossover innovation numbers are lined up. Genes that do not match are always included from the more fit parent. Its population is separated through speciation to protect innovation, which calculates genetic distance between individuals, and groups them into species with similar genetic material. This speciation distance δ is calculated using fitness sharing, as a linear combination of the number of excess E and disjoint D genes, as well as the average weight differences of matching genes $\overline{W}$, including disabled genes:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}. \tag{1}$$

In each generation, the population is separated into non-overlapping species, based on a compatibility threshold δ_t. To ensure no one species takes over the population, the fitness f_i' for each individual i is calculated based on its distance δ from each other individual j in the population:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{2}$$

with sh set to 0 when $\delta(i,j)$ is above the threshold δ_t, else 1. Thus, species are clustered according to compatibility, eliminating their lowest performing members in each generation, then replaced by their offspring in the following generation.

NEAT is generally considered to be one of the most important innovations in evolutionary neural networks, and has been extended and modified in hundreds of ways (Papavasileiou et al., 2021). It has been shown to be adept at solving almost any type of learning problem, and is currently considered to be the most optimal solution for several difficult problems. In particular, NEAT algorithms have some of the best combined efficiency and performance on continuous-space problems, such as the pole-balancing problem, or regression data with difficult-to-determine non-linear solutions.

2.2 Ensemble Predictions

Using a consensus of predictors rather than a single one has long been a tactic of improving the accuracy and variance of predictions. This can mitigate the biases or limitations present in individuals, and as such has seen use with many applications and predictors (Mohammed and Kora, 2023). Random forests and other similar ensemble methods are known to be some of the best models in terms of efficiency and performance across many types of data (Breiman, 2002).

However, across ensemble learning, NE is rarely studied. There are some notable exceptions, however, such as Faber et al. (2021). In this paper, the authors use an ensemble of NE learners to detect multivariate time series anomalies. It is noted that the NE ensemble approach did work well, but would take significantly more fine-tuning to see desired performance on the dataset. For this reason, we plan to utilize more standard performance metrics in hopes of displaying the base performance of an ensemble on a more general scale, while also displaying its applications on a more intractable problem.

There are several possibilities for training a group of networks as an ensemble rather than individuals. The first is by a simple voting procedure, either weighted or unweighted. Simple voting has been shown to work best in situations with a small number of discrete outputs, such as classification. However, as this experiment will not focus on these problems, this can be safely ignored. Applying weights to the vote allows for voting in continuous spaces. This can be problematic, though. In the case that half the population wants to take one action, and the other half wants to take another, the result is an action which none of the voters want. Or in the case of extreme outliers, one vote can hold far greater importance than the rest. Another option is to include confidence and pick the voter with the greatest confidence. This can be suitable in certain cases, but is unlikely to work well in continuous spaces. Bagging, or Bootstrap Aggregating, is another popular ensemble technique that involves training multiple models on different subsets of the training data. These models are combined using a voting or averaging mechanism to make a prediction (Breiman, 1996). Stacking, also known as stacked generalization, is a meta-ensemble technique that involves training multiple base models and then combining them using a higher-level model. The higher-level model learns to combine the predictions of the base models to make a final prediction (Wolpert, 1992).

The last option which is commonly used in the literature is a *gating network*, which trains a separate network to learn which learners to use based on the input. Gating networks have been employed in the literature in a variety of different ways, with no real consensus on methodology. Some papers have shown significant improvements by adding a gating network to other approaches (Ebrahimpour et al., 2013), and substantial research has been done to make it work efficiently with neural networks (Brown, 2004). Despite this, the concept of gating networks in conjunction with NEAT has not been studied extensively.

3. Experimental Design and Results

In this paper, we propose using a gating network to program an ensemble of NEAT learners. The ensemble consists of multiple NEAT learners that are trained on the same problem, but with different random start seeds. The gating network is another NEAT network, trained as a population. By feeding the inputs to the gating network, the output of the gating units can be used to select which NEAT learner to use for a specific problem.

Formally, let f_i be the i-th NEAT learner in the ensemble, and g be the gating unit. Given an input x, the output of the gating unit $g(x)$ is a scalar value in the range [0,1] that represents the probability that f_i should be used to generate the output. The probability that f_i is selected is given by the softmax function:

$$P(f_i|x) = \frac{e^{g(x)}}{\sum_{j=1}^{N} e^{g(x)}}$$

The result of this process when repeated for each f_i is a vector of probabilities $\mathbf{p} = [p_1, p_2, ..., p_n]^T$, where $p \in [0, 1]$ and $\sum_{i=1}^{n} p_i = 1$. Then, a random selection is made based on these probabilities, similar to the process in fitness proportionate selection.

We use cooperative coevolution to train the ensemble of NEAT learners with the gating network, to maximize the overall performance by evolving both simultaneously. During the cooperative coevolution process, the gating units and individual NEAT learners are evolved together to maximize the overall performance of the ensemble. The gating units evolve to select the most appropriate individual NEAT learner for each problem, while the individual NEAT learners evolve to improve their individual performance. This approach allows the ensemble to take advantage of the strengths of each individual NEAT learner while avoiding the potential weaknesses that come with relying on a single learner.

During the training process, we jointly optimize the weights and biases of the gating network and the individual NEAT learners to maximize the overall performance of the ensemble. Specifically, we use a fitness function that evaluates the performance of the entire ensemble on a set of training examples. The fitness function is given by:

$$fitness = \sum_{x \in D} \sum_{i=1}^{N} p_i(x) \cdot f_i(x),$$

where D is the set of training examples, N is the number of individual NEAT learners, $p_i(x)$ is the probability produced by the gating network for input vector x and individual NEAT learner i, and $f_i(x)$ is the fitness of individual NEAT learner i on input vector x. The fitness of each learner is defined by the specific problem to which the method is applied.

Four problems were chosen for testing: XOR, regression, double pole balancing, and Go. XOR is used first, as a confirmation that the approach works and reliably outputs results. Regression then allows for detailed analysis of the system, especially displaying its capabilities in choosing an expert for a specific problem. Double pole balancing is a slightly more complex version of the very common pole balancing problem, which allows for general comparison against other techniques. Finally, the classic game of Go is used to display the capabilities on a more difficult problem.

For comparison, two models were used in each experiment: a single NEAT network and the proposed NEAT ensemble. NEAT was implemented using the neat-python package, which uses the parameters and implementation details from Stanley and Miikkulainen (2002). In testing, the default parameters were able to accurately replicate the original NEAT results, so they were maintained for this experiment. The tuning that was done involved finding the correct number of networks to use in the ensemble. Ensemble sizes of 2 through 9 were tested on the double pole balancing problem, resulting in the finding that training time was reduced from 2 to 5 populations, but did not decrease significantly after a plateau is reached at an ensemble size of 5.

For each experiment, a total of 30 runs were completed. Due to the heightened computational complexity of running an ensemble of NEAT networks, it was not possible to run the experiments for a significantly larger number of times. Still, 30 runs were enough to obtain a sufficiently robust picture of the models' behaviors and performances, and enough to perform statistical analysis.

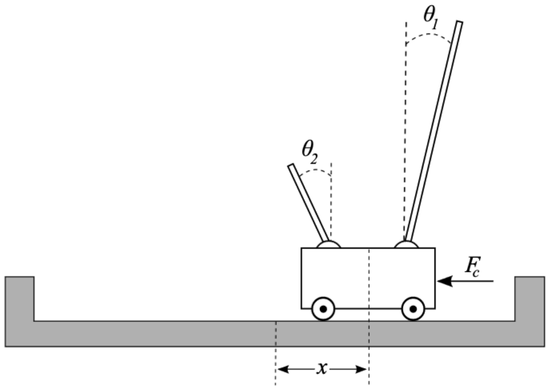

Figure 2: Double pole balancing problem

3.1 XOR

To begin with, the proposed method was tested on the basic XOR problem. In this problem, there are two binary inputs, with a single binary output. The fitness of an individual is a maximum of 4, which is achieved if the network successfully XORs all four inputs. Over the 30 runs, both methods correctly solved the XOR problem 100% of the time. As expected, the single NEAT network evolved to XOR the inputs in an average of 35 generations ($\sigma = 7.2$), which reflects the results found from the NEAT paper. Similarly, the proposed ensemble was able to solve it, taking an average of 32 generations ($\sigma = 5.5$). Though these results are not significantly different, and the XOR problem is not particularly useful for comparison, the reduced standard deviation of the ensemble is an interesting result. This is the first display of the usefulness of the ensemble, as it can more consistently produce a result, which can be useful in certain applications.

3.2 Double Pole Balancing

The double pole balancing problem, as described in Figure 2, is a well-known benchmark in the literature, and almost every technique has been applied to it. Two poles are connected to a moving cart by a hinge, and the neural network must apply force F_c to the cart to keep the poles balanced for as long as possible without going beyond the boundaries of the track. The system state is defined by the cart position (x) and velocity ($\dot{x}$), the first pole's position (θ_1) and angular velocity ($\dot{\theta_1}$), and the second pole's position (θ_2) and angular velocity ($\dot{\theta_2}$). Control is possible because the poles have different lengths and therefore respond differently to control inputs. By including the velocity as inputs to the neural network, the problem becomes Markovian, and has been applied to many different standard systems.

NE has long been known to be an effective solution for pole balancing in comparison to other techniques, displaying decreased training times and higher fitness than other common reinforcement learning methods (Moriarty and R., 1997). These capabilities have been thought to be effective due to NE's ability to learn behaviors and patterns, generating increasingly complex structures in the network as it learns. In the original NEAT paper, various forms of the pole balancing problem were the primary focuses of displaying the

Method	Evaluations	Generations	No. Nets
EP	307,200	150	2048
EP Ensemble	279,500	150	2048
NEAT	3,600	24	150
NEAT Ensemble	3,420	23	150

Table 1: Double Pole Balancing Problem Results

technique's capabilities. Their results found that NEAT performs 25 times faster than Cellular Encoding and 5 times faster than ESP.

The double pole balancing problem was set up using the parameters described by Stanley and Miikkulainen (2002), and implemented in the neat-python package. Fitness is not determined by ability to solve the problem, as all four methods were able to maintain the success criteria of keeping both poles balanced (between -36 and 36 degrees from vertical) for 100,000 time steps. As displayed in Table 1, the inclusion of the ensemble method did not significantly improve results. This result is likely because the ensemble method does not create significantly improved ability to solve a problem quickly or efficiently, but is instead useful in gaining performance. As the double pole balancing problem can be solved by even a simple method, albeit slowly, the ensemble was not able to display its capabilities in this problem.

3.3 Go

The final test for the proposed method was the game of Go, which is a board game that originated in China and is played on a 19x19 board. The game has a large branching factor and a high degree of complexity, making it challenging to solve using traditional search-based methods. It is commonly studied in the literature, but even the most advanced method with supercomputer capabilities struggle to consistently perform against human players, despite its simplicity. There are two players, black and white, which alternate turns placing stones on an empty intersection. Stones are not moved, but a group of stones can be surrounded by the opposing player, "capturing" the stones and removing them from the board. The winner is determined at the end of the game, when there are no legal moves or both players have passed, by counting the total area controlled by each player.

Despite the basic rules, there are great complexities and theories surrounding the game, especially as it has been played around the world for thousands of years, making it arguably the oldest board game which is still commonly played. Go has long been recognized as a challenging problem for artificial intelligence due to the vast number of possible moves and board configurations, which exceeds the number of atoms in the universe. However, recent advances in machine learning, such as deep neural networks and reinforcement learning, have shown promising results in creating Go-playing algorithms that can compete with top human players (Silver et al., 2016). These advances have not only led to improvements in the performance of game-playing agents but also shed light on broader research questions in the fields of decision making, planning, and cognitive psychology. As such, Go has become an increasingly popular domain for testing and benchmarking machine learning algorithms, and has led to important contributions to the field of AI.

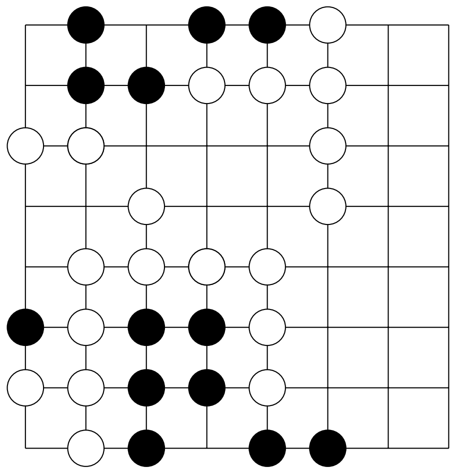

Figure 3: An example 7x7 board position generated by the model. In this position, white is clearly winning, but the game has not finished.

To assess the model's capabilities, the Gym API developed by OpenAI was employed.. In this, the game is represented by an object with a three-dimensional $3 \times n \times n$ array representing the board. The three layers represent the black and white pieces respectively and invalid moves for the next action (*ko*), with additional channels indicating whose turn it is, whether the previous move was a pass, and whether the game is over. To ease the computational complexity, the experiment used a board size of 7×7, which is a standard variation of the game.

Each network is fed the two layers from the game representing the positions of black and white pieces, with a similar output layer to decide where to place the stone. This results in a total of $8 \times 8 \times 2 = 128$ inputs, and $8 \times 8 = 64$ outputs (as the edges are played on, not the spaces). The output layer consists of 64 scores ranging from 0 to 1 which represent the model's estimated quality of a play. To ensure no illegal moves are made, the outputs of any illegal position (i.e. *ko* and positions which already contain a stone) are artificially set to zero. To speed up training, the scores are then squared and normalized to one again. Formally, let $\mathbf{y}$ be the output vector of the neural network with n nodes and $\mathbf{y}'$ be the altered vector:

$$\mathbf{y}' = \frac{\mathbf{y}^2}{\sum\limits_{i=1}^{n} \mathbf{y}_i^2}$$

To train the populations, we used the board positions and moves from around 8,000 games, which were sourced from a variety of internet Go sources. Based on the sources and characteristics of the data, we are confident that the data is all derived from games played by humans, and many of them are from professional games. The 9x9 board format is not very common at the professional level, so it was difficult to source more games than this, or to get them all from a single source. However, we are confident that the data quality is high enough for training, and that many games is plenty for training, as it totals to over 400,000

Model	Average Difference	Standard Deviation
NEAT v. NEAT	-0.2	14.2
Ensemble v. Ensemble	0.1	10.9
Ensemble v. NEAT	8.7	4.5
NEAT v. Beginner Bot	4.6	10.2
Ensemble v. Beginner Bot	11.1	9.9
NEAT v. Intermediate Bot	-1.3	8.4
Ensemble v. Intermediate Bot	3.2	8.2

Table 2: Average territory differences and standard deviations after 30 trials of Go games.

board positions. We used the training methodology from Clark and Storkey (2015), with 8% of the games used for training, 4% for validation, and the remaining 88% for training. The validation set was used to test parameters, and monitor the progress of training.

We were able to train the models on this dataset for a total of 50 epochs, which was limited by time constraints. Due to the size of the dataset, though, we believe this was a sufficient training time, and is more than some other similar experiments. To ensure that the networks could play we, then trained the networks for an additional 50 epochs with a form of competitive evolutions, by having the networks repeatedly play against each other. In each training step, two networks are randomly chosen to play against each other (75 games per iteration with $n = 150$). Once the game concludes, the two players are assigned a fitness equal to the difference between the two players' territories, which typically ranges from 0 points for a tied game to as much as 30.

The test set was then used to validate the performance of the models. We found the NEAT networks maintained a performance of 33.7% accuracy on the testing sets, while the ensembles were able to reach 38.2% accuracy. These scores do not appear very high, but they are very competitive to other techniques with similar complexity. With the ensembles achieving significantly higher performance than the single NEAT networks, these scores confirm the ability of the NEAT ensemble in complex, difficult to predict environments. As the dataset used had multiple sources, it is difficult to directly compare our technique against other models, but the best performance yet achieved on a 19x19 board is 40-45% (Clark and Storkey, 2015).

After training, several tests were performed. First, to compare the methods, the ensembles were tested against the single NEAT networks to see if the ensemble is able to learn to play Go with higher performance. The resulting scores after 30 games are displayed in Table 2. As would be expected, the models playing against themselves generally came out about even, with about half the games being won by each player. However, the result of the ensembles versus NEATs confirms the ability for the ensembles to learn a complex game such as Go. The ensemble models won around 97% of the games against the normal NEAT, which is a clear demonstration of its abilities.

Additionally, the OpenAI Gym Go package comes with a prebuilt Go bot, which has difficulty levels which roughly correspond to beginner, intermediate, and advanced players (using the *kyu* and *dan* system). Both the single NEAT network and the ensemble were able to generally beat the beginner player, and the ensemble did it with significantly more territory. Against the intermediate bot, scores were much closer to even, but the ensemble did manage to achieve a win rate over 50%. The Ensemble's territory scores were significantly better than NEAT against the intermediate bot, but only by a slim margin. Unfortunately, against the advanced bot, neither model was ever able to win.

Though the model was able to learn to play the game with proficiency, it was not able to discover and utilize common strategies used by human players. For example, the concept of *eyes* is important in Go, which refers to the idea of creating empty spaces that are surrounded by a player's stones, and which cannot be captured by the opponent. However, the model did not learn to recognize this concept, instead focusing on capturing as many stones as possible, which can be seen in the inefficient position in Figure 3. One notable aspect of this position is the presence of clear geometric structures, especially lines of stones, which are not usually apparent in human-played games. Additionally, the model was not able to display a capability to plan ahead and anticipate the consequences of its moves, which is a crucial skill for successful Go players. This limitation is likely due to the fact that the model was trained solely through self-play, without any guidance or feedback from human players. While this approach has its benefits, such as the ability to generate novel strategies and tactics, it also has limitations in terms of developing human-like playing styles. Future research could explore hybrid approaches that combine self-play with human feedback or imitation learning, in order to address some of these limitations and create models that can play at a high level while also utilizing human-like strategies. Despite that, it was able to consistently play at an intermediate level, with very little complexity and training time in comparison to more robust methods.

4. Conclusions

Though a technique such as this could be useful in environments where performance matters much more than computational complexity, the applications of such complicated ensemble methods tend to be limited. This ensemble took more than 6 times as long to train as a single NEAT population would, as a result of needing 5 populations for prediction and another population for the gating network. One reason that ensembles may become more useful in the near future, though, is the ease of using modern parallelization techniques to improve performance. The Python package used in this experiment comes with the ability to parallelize computation, which successfully improved the training times. Still, there is an obvious gap between computational resources and the small gains found in this experiment, with several hundred percent increases in computation versus single digit percentage performance gains.

After training, however, this method is almost certainly better than a single NEAT network. While the model may seem overly computationally complex, it is not problematic because the feed forward step for each network only takes less than a second to complete. This is because the number of neurons and connections in each network is not very large, typically in the order of tens to hundreds, thanks to NEAT's ability to form efficient, small networks. Thus, the feed forward step is not prohibitively computationally intensive. Moreover, after the NEAT ensemble is trained, the gating network can quickly select the best network for a particular input, instead of evaluating the fitness of all networks in the ensemble, which significantly reduces the computation time. Still, the training time remains large, because in training every network in every ensemble needs to be evaluated, in order to get an accurate fitness for each gating network in the population. So, there are clear applications for this technique in places where large resources can be committed to training time, and fast, accurate predictions are needed in problems.

There is still a massive amount of research to be done in this topic. In future experiments, further tuning of hyperparameters would be preferred. By doing detailed testing across a wide variety of hyperparameters, much better and more conclusive analysis could be performed. However, the inability to do this can also be considered a fair component of the comparison. In most modern applications, users do not have the background knowledge nor time to accurately choose hyperparameters. Considering this, it is unreasonable to assume hyperparameters will be tuned in the real-world, which contributed to the decision to use default values in this experiment. One of NEAT's major downsides is its large range of complex hyperparameters, most of which cannot be reasonably tuned for most use cases. There is some work on automatically finding good values for hyperparameters in NEAT Radaideh et al. (2021), the inclusion of which could drastically improve results.

Testing NEAT and ensembles on more complex problems would likely be the obvious next step. In this experiment, the ensembles were not able to display their full capabilities, because none of the problems fully took advantage of the ensemble's capabilities. In an environment where slight differences in performance matter greatly, such as safety-critical environments, and performance is difficult to achieve, ensembles could be a more useful option. Most of our researched problems rewarded fast training more than anything else, which was improved by the ensembles, but it could still be better showcased. It would further be a useful experiment to try this method on as many problems as possible, to see what it is best suited for, rather than a few randomly chosen problems.

Ideally, the Go problem in particular would have better forms of training and testing. However, this comprised the large majority of actual training and testing time, so it was difficult to experiment with trying different possible implementations, though it would almost certainly have a large positive impact on performance. For instance, ensuring that each NEAT network in the ensemble is able to specialize in a certain task, rather than simply training them all on the same data. For instance, in the Go game, each network could be trained on a different portion of the game, such as beginning, middle, and end. Many previous works have attempted to do this, especially in data classification (Brown, 2004). Other methods of improvement could involve changing the fitness equations used in this experiment, altering the training procedures, and increased optimization. It would not be particularly difficult to incorporate the method of Monte Carlo Tree Search into our model, which is currently considered the state-of-the-art for advanced Go bots, though it would likely increase the computational complexity (Silver et al., 2016).

Despite the large amount of future work which could be done, this experiment did successfully show that ensembles can be applied to NEAT using a gating network method. In all the experiments done, the number of generations needed for training was able to be reduced (albeit insignificantly), and performance generally increased. As many of these techniques were somewhat crude and not using the most modern strategies, this is clearly an area of artificial intelligence research with great future potential.

References

L Breiman. Random forests. *Machine Learning*, 45:5–32, 2002.

Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

Gavin Brown. *Diversity in neural network ensembles.* PhD thesis, Citeseer, 2004.

Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play go. In *International conference on machine learning*, pages 1766–1774. PMLR, 2015.

Reza Ebrahimpour, Seyed Ali Asghar Abbaszadeh Arani, and Saeed Masoudnia. Improving combination method of ncl experts using gating network. *Neural Computing and Applications*, 22:95–101, 2013.

Kamil Faber, Marcin Pietron, and Dominik Zurek. Ensemble neuroevolution-based approach for multivariate time series anomaly detection. *Entropy*, 23(11), 2021.

Faustino J. Gomez and Risto Miikkulainen. Solving non-markovian control tasks with neuroevolution. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1356–1361, 1999.

Ammar Mohammed and Rania Kora. A comprehensive review on ensemble deep learning: Opportunities and challenges. *Journal of King Saud University - Computer and Information Sciences*, 35(2):757–774, 2023.

D.E. Moriarty and Miikkulainen R. Forming neural networks through efficient and adaptive coevolution. *Evolutionary computation*, 5(4):373–399, 1997.

Evgenia Papavasileiou, Jan Cornelis, and Bart Jansen. A systematic literature review of the successors of "neuroevolution of augmenting topologies". *Evolutionary Computation*, 29(1):1–73, 2021.

Majdi I Radaideh, Katelin Du, Paul Seurin, Devin Seyler, Xubo Gu, Haijia Wang, and Koroush Shirvan. Neorl: Neuroevolution optimization with reinforcement learning. *arXiv preprint arXiv:2112.07057*, 2021.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.