Evolving Ensembles of Neural Networks using Ensemble-Based Fitness Functions

A Major Qualifying Project
Submitted to the faculty of
Worcester Polytechnic Institute
in partial fulfilment of the requirements for the

Degree in Bachelors of Science
in
Computer Science

Degree in Bachelors of Science
in
Data Science

By:
Aidan Horn, Gregory Conrad, Gabriel Deml

Abstract

NeuroEvolution of Augmenting Topologies (NEAT) is a technique that employs genetic evolution to optimize neural networks to solve a particular machine-learning task. The team sought to build upon the capabilities of NEAT to evolve ensembles of neural networks that solve both classification and reinforcement learning tasks. To do so, the team developed a novel fitness function for evaluating a neural network by the performance of ensembles containing it, termed *Constituent Ensemble Evaluation*. To test this approach, the team developed an ensembling algorithm that maximizes genetic diversity among ensemble members and compared this method to other established algorithms. As ensembles benefit from diversity, the team also attempted to determine an optimal configuration of NEAT parameters to yield a more diverse population of neural networks. It was hypothesized that neural networks evolved using constituent ensemble evaluation and ensembled using a diversity heuristic would outperform ensembles generated using individual fitness. For simple reinforcement learning tasks, constituent ensemble evaluation did not improve the capability for NEAT to evolve useful ensembles, regardless of the ensembling method used. For classification tasks, the team claims that evolving individuals based on their constituent ensemble performance is not sufficient alone; however, when constituent ensemble evaluation is incrementally introduced into the fitness function, ensembles yield better overall performance. This approach yielded an average test accuracy of 0.601 ± 0.1 on the UCI Heart Disease data set, which outperforms an alternative method for evolving ensembles, Orthogonal Evolution of Teams.

1 Acknowledgements

We want to thank Associate Professor of Computer Science, Joseph E. Beck, for advising this project. Professor Beck's guidance and insightful ideas helped the project find and investigate many new areas of interest.

Further, we would like to acknowledge WPI Academic & Research Computing (ARC), where we ran thousands of compute-hours worth of experimentation. Without ARC, and help from ARC/IT staff, this project would not have been feasible.

Contents

2 Introduction

This paper explores the application of genetic algorithms for evolving ensembles of neural networks using an ensemble-based fitness function. This section will present these topics at an introductory level while a further justification of the methods employed in this paper will be explored during the literature review.

2.1 Genetic Algorithms

Genetic algorithms describe a field of optimization and search techniques analogous to the principles of natural selection and genetics, wherein only the fittest members of a population may go on to reproduce their genes in each successive generation. Each individual in the population corresponds to a point in the search space of a given problem such that this point may be evaluated as a potential solution. Within the scope of this project, an individual is represented by its "genome" (a list of the neural network's graph edges/weights), which can be expressed as a "phenotype" (a feed-forward neural network).

Popular techniques such as Neural Evolution for Augmented Topologies (NEAT) seek to approach the optimal solution to a particular task by evaluating each genome based on the outputs of the phenotype, selecting the fittest candidates, and then selectively mutating the genomes in some way. For example, mutation can be described as adding or deleting perceptrons and modifying connection weights. In genetic algorithms, the fittest individuals are more likely to be selected for reproduction, passing their genetic material to the next generation, while the weaker ones are discarded. By iteratively applying the genetic operators of selection, crossover, and mutation to the population, genetic algorithms can search for the optimal or near-optimal solutions to a wide range of optimization problems, including function optimization, feature selection, and parameter tuning. Successive rounds of selection and mutation is referred to as evolution. Evolution, as described above, can simultaneously optimize both the weights and structure (the network topology) compared to an approach like back-propagation which solely optimizes weights. This technique has been used to solve computationally intractable problems. However, as the genetic algorithm relies on an arbitrary fitness function to perform selection, the choice of fitness criteria will necessarily determine the overall effectiveness of the approach.

2.2 Fitness Functions

Fitness functions are a crucial component of genetic algorithms. The fitness function evaluates the quality of each candidate solution or individual in a population, assigning a numerical value representing its fitness or suitability for the problem being solved. In reinforcement learning, where the model is rewarded based on the current state of the agent, the fitness function may simply be the sum of the reward earned by the model. For classification problems, where the goal is high accuracy, the fitness function may be the accuracy of the model on the training data. The design of an effective fitness function is crucial for the success of genetic algorithms, as it determines the direction and speed of the search process and can lead to different outcomes and trade-offs. One key challenge is overfitting, which occurs when a model has become to closely tied to the training data and fails to perform similarly well on testing data.

2.3 Regularization

The process of tackling overfitting is called regularization. Regularization encourages the model to reach an optimal level of complexity, such that it is able to appropriately learn the overall patterns in the training data without completely mimicking it, termed *generalization*. Explicit regularization refers to techniques which specifically constrain the model to avoid overfitting, such as penalizing the fitness of a genome by its size. In the context of this paper, explicit regularization is attempted through the use of an ensemble-based fitness function.

2.4 Ensembles of Neural Networks

Ensemble learning is a powerful technique in machine learning that combines multiple models to improve the accuracy and robustness of predictions. By averaging or combining the predictions of multiple models, ensembling can reduce the variance in the predictions, which can improve the generalization performance of the model and help avoid overfitting. This paper proposes an ensemble-based fitness function which rewards models that are complimentary to other members in the population, such that the final population contains models that have evolved to work well in an ensemble.

3 Literature Review

Artificial intelligence has experienced a significant increase in interest in utilizing evolutionary algorithms to optimize neural networks. Kenneth Stanley and Risto Miikkulainen's 2002 seminal paper, *Evolving Neural Networks through Augmenting Topologies*, proposed NeuroEvolution of Augmenting Topologies (NEAT) as a method for evolving neural networks by employing genetic algorithms to optimize the networks' topologies—the weights and connections of the neural network.

NEAT relies on the concept of a genome, which symbolizes the structure and characteristics of a neural network. The attributes of a given neural network are encoded as a linear representation specifying incoming and outgoing connections, as seen in Figure 1. This linear representation allows for two key operations, crossovers and connection parameter mutations [7]. Connection parameter mutations refer to changes made to an individual neural network, such as adding, removing, and changing connections between nodes [2]. Crossover is a powerful technique for introducing diversity into the population by allowing genomes to recombine and produce offspring that have inherited aspects of their parents topologies [2]. A point on both parents' linear genome is picked randomly, and designated a 'crossover point'. Information to the right of that point are swapped between the two parent chromosomes. This results in two offspring, each carrying some genetic information from both parents. Due to the stochastic nature of crossover and mutation, there is always a chance that good solutions fail to persist [3]. NEAT approaches this problem by implementing "steady-state" selection [9]. Instead of creating an entirely new population each generation, a percentage of the fittest solutions are allowed to persist, and the unfit members are replaced by offspring of the fittest.

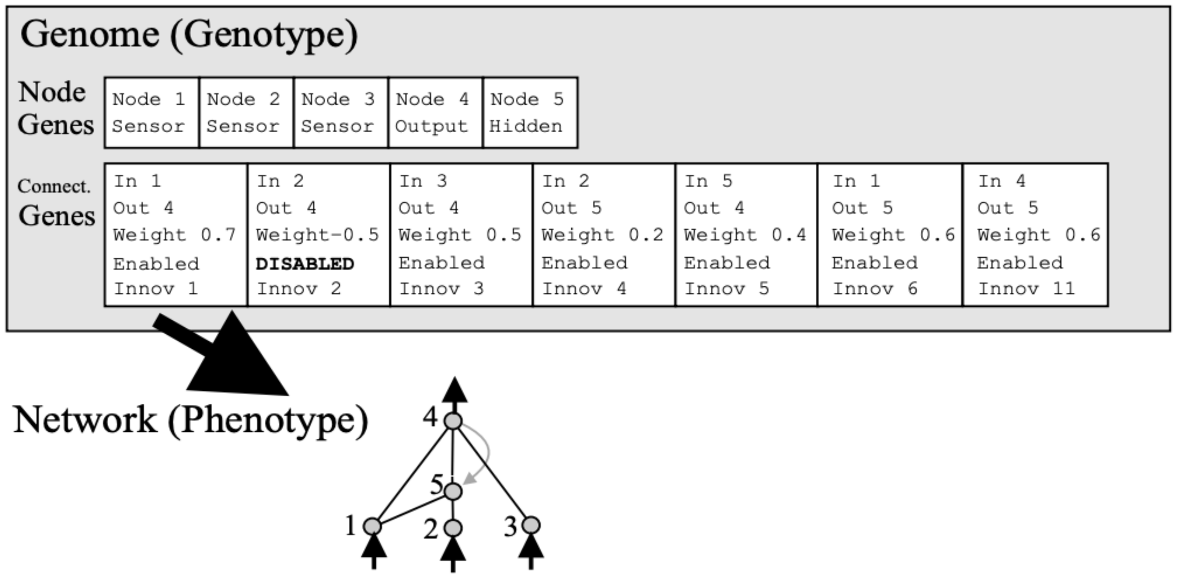

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

Figure 1: A genotype to phenotype mapping example. A genotype is depicted that produces the shown phenotype. There are 3 input nodes, one hidden, and one output node, and seven connection definitions, one of which is recurrent. The second gene is disabled, so the connection that it specifies (between nodes 2 and 4) is not expressed in the phenotype

Adapted from *Evolving Neural Networks through Augmenting Topologies*, Stanley and Miikkulainen, 2002

.

Prior to NEAT, previous approaches were faced with the *Competing Conventions Problem* [5], a difficulty with crossover between two permutations of genomes representing the same solution which produces offspring with redundant connections in place of potentially useful ones. NEAT presents a solution to this problem with a crucial technique, tracking "innovations" in neural network topology. The authors implemented a historical marking system to track each network's evolutionary history. When crossover occurs, individual genes are paired according to their matching innovation number, of which one is selected randomly to persist. Genes which do not match are allowed to persist if they belong to the fittest of the two genomes. Genes that do not match are either disjoint or excess, depending on whether they occur within or outside the range of the other parent's innovation numbers. This system allows NEAT to build on previous successes and avoid replicating failed attempts. NEAT implements additional techniques for preserving innovations while maintaining diversity within the population. The authors propose limiting competition within the population to smaller niches, defined as "species", which are constructed by separating genomes by their species distance. The species distance is defined by

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W}$$

where E is the number of excess genes, D is the number of disjoint genes, and $\bar{W}$ is the average weight differences of matching genes. By default, this equation is parameterized by $c_1 = 1.0$, $c_2 = 1.0$, $c_3 = 0.5$, and $N = 1$. Using this approach, each species is allowed to evolve separately, and are each permitted a certain number of offspring based off of an adjusted fitness function which takes the existing species size into account. The combination of innovation and speciation allow for NEAT to preserve useful topologies while maintaining a manageable population size, yielding a stable and computationally viable algorithm for evolving neural networks.

The NEAT algorithm has been highly influential in evolutionary neural network optimization and has been applied to various problems, including image and speech recognition, game playing, and robotics. It has also inspired the development of other algorithms based on evolutionary computation principles, such as HyperNEAT and CoDeepNEAT.

Neuro-evolutionary algorithms have demonstrated utility in approaching machine learning tasks, though there are many ways to improve this technique. One problem facing many machine learning algorithms is overfitting the training data. Yao and Liu [11] point out that previous approaches to neuro-evolution tend to choose a single network from the population, which yields the minimum error rate after each generation. This approach has the problem of being extremely dependent on the initial random state of the population initialization, making it difficult to know if a solution is generalized or able to adapt to unseen data. To avoid this, they achieve implicit regularization of the networks by maintaining smaller networks with diverse structures which are less capable of overfitting. This is in contrast to explicit diversity regularization, explored by Optiz and Shavlik [4] which includes in the fitness function a diversity term measuring the average difference of a given networks weights with that of the average population network weights.

Ensemble based methods are also used to achieve regularization. Both Yao and Liu and Optiz and Savlik generate a final ensemble from their populations using a rank based linear combination of individuals, ranked by fitness. Their approach is similar to bagging (Bootstrap Aggregating), which generates multiple models and aggregates their predictions [1] using a ranking function which attempts to maximize the variance in the final prediction. Rank-based ensembling has been shown to improve the performance of various classification algorithms, however, Zhou et al. [12] improve on the bagging approach through the use of a genetic algorithm to evolve a weighted selection criteria for selecting a subset of the available neural networks, rather than using the entire population. They conclude that using many (between 50% and 75%) of the available networks tends to yield the best bias-variance trade-off and improve generalization.

Soule et al. [6] [8], discuss the limitations of existing evolutionary algorithms for evolving teams or ensembles comprising cooperating team members. The paper examines two current methods for training ensembles: island and team. The island approach evaluates each genome individually, only assembling them at the end. In contrast, the team approach creates groups of individuals and enforces selection based on the teams' ensemble performance. The island and team approaches have subtle yet significant weaknesses restricting their performance. Island approaches generate highly fit team members but with correlated errors, resulting in sub-optimal team performance. On the other hand, team approaches can produce team members with inversely correlated errors, leading to relatively good team performance; however, the members themselves are relatively poor, limiting the teams' overall performance. Soule et al. present a novel class of evolutionary algorithms called Orthogonal Evolution of Teams (OET), which overcomes these limitations. OET algorithms generate highly successful individual solutions with members specializing in distinct sub-domains of the problem space. This specialization leads to robust solutions covering the problem domain with minimal gaps or errors. Results show that OET algorithms merge the benefits of both island and team approaches, outperforming standard evolutionary approaches significantly. The results of this paper inspired the team's attempt to approach neuro-evolution for ensembles by implementing individual selection as in the island approach but using an ensemble based fitness metric, as in the team's approach.

In conclusion, the field of artificial intelligence has seen significant advancements in the optimization of neural networks through evolutionary algorithms and the use of ensemble-based classifiers. NEAT and OET have demonstrated the potential of evolutionary approaches to optimize neural network topologies and create robust teams of classifiers, respectively. These methodologies have been applied to various problems, showcasing their versatility and effectiveness.

4 Methodology

One of the key factors determining an ensemble's success is the diversity of the individual models. The more diverse the models are, the more likely they are to capture different aspects of the problem and avoid making the same mistakes. In neural network ensembles, diversity can be achieved by varying the individual networks' initialization and architecture. The team's approach seeks to use the capabilities of NEAT to evolve diverse neural networks, which are selected based on group performance. Instead of evaluating a neural network's fitness by its individual fitness for a given learning task, the team instead evaluates a neural network by its performance in groups. The team hypothesizes that this approach will result in a population of neural networks better suited to ensembling. Additionally, the team will test different approaches to ensembling with selection algorithms, which dictate how to construct varying size ensembles. Within this project's scope, the team will experiment with the parameters of the NEAT algorithm, the choice of the fitness function, and the final ensembling heuristic, with the intent of finding an optimal configuration across a variety of classification and reinforcement learning tasks.

4.1 Ensembling Algorithms

To start the project, the team first created a set of algorithms that create ensembles given a population of individual neural networks. These algorithms could then later be used to analyze and evaluate the results of a particular configuration given to an experimental trial.

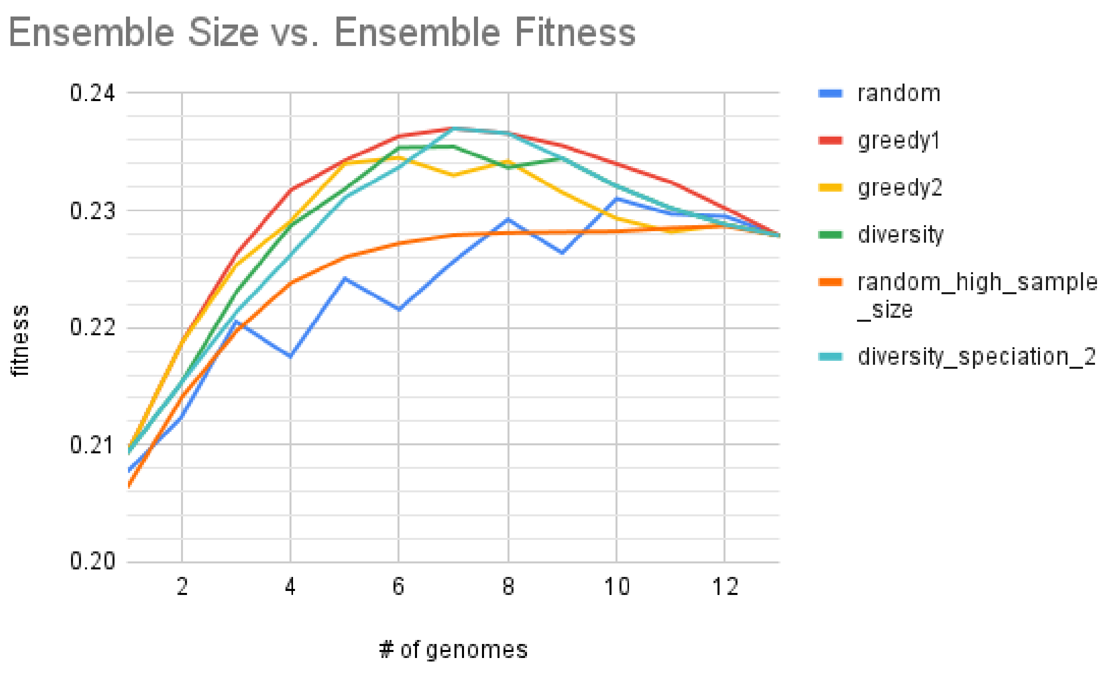


Figure 2: Initial Algorithm Results From Training Data

Each algorithm has its own unique strengths and weaknesses, ranging from its effective ensemble size(s) (in relation to the entire population size) to its run-time complexity. Straight-forward implementations of the following algorithms can be found in the source code repository's algorithms.py.

4.1.1 Random (control)

The first ensembling algorithm to discuss is the random algorithm. The random ensembling algorithm is important because it serves as the control to compare all other algorithms against.

The random algorithm picks random ensemble(s) at all given ensemble sizes and evaluates their fitnesses. The time complexity for random is the lowest of all algorithms at $O(n)$, since the random algorithm needs to pick at most k ensembles to test (a constant, more on this later) for n different ensemble sizes where n is the population size. Thus, the team test $O(kn)$ ensembles, which boils down to $O(n)$.

As mentioned above, the random algorithm utilizes a constant k, depicting the limit of the number of ensembles to test at a given ensemble size n. In other words, k is the limit on the sample size to test for each given ensemble size. To help illustrate the need for k, take a look at the *random* (colored blue) and *random_high_sample_size* (colored orange) series in Figure 2.

As illustrated in the graph above, it makes much more sense to have a higher k; with only one sample per ensemble size, the results are sporadic and inconsistent (as pictured by *random*). With a higher sample size ($k = 100$ for *random_high_sample_size*), the random distribution is sampled a sufficient number of times to achieve a more accurate mean fitness. Since the actual implementation of random relies upon some internal ensembling algorithms, here is some easier-to-digest pseudo-code instead.

```
procedure random()
  accuracies = []
  for n = 1 to population_size
    ensembles = generate at most k random ensembles
    accuracies.append(average fitness of ensembles)
  return accuracies
```

4.1.2 Brute Force

The brute force algorithm does exactly what it sounds like: it brute-force evaluates all possible ensemble combinations to determine the best possible ensemble for all given ensemble sizes for a given population (and consequently, the theoretical best possible ensemble for a given population, assuming the entire data set is known in advance). This ability is extremely powerful, but it also comes with a cost.

The team chose not to implement the brute force algorithm for good reason. To implement brute force, one must sample every possible ensemble for all ensemble sizes from 1 to the population size; thus, the time complexity is:

$$\sum_{i=1}^{n} \binom{n}{i} = O(2^n)$$

While this is feasible for smaller populations, it becomes infeasible very quickly. For example, for a population size of 15, one would only need to evaluate 32767 possible ensembles. However, when jumping to a population size of 32, one must evaluate over 4 billion ensembles. To address the issue of the extraordinarily large sample space, the team proposes a greedy algorithm that sufficiently approximates the best possible ensemble at a given ensemble size within a somewhat reasonable time complexity. The team names this algorithm "Greedy 1."

4.1.3 Greedy 1

Greedy 1 was theorized with the help of the project advisor, Professor Joseph Beck. Originally, thoughts of creating an ensembling algorithm based on dynamic programming (DP) were discussed, but those thoughts were quickly shut down because it is theoretically impossible to apply DP to the problem of ensembling (because there is no deterministic recursive step). However, Greedy 1 does draw parallels to a true DP solution and approximates the best-performing ensembles fairly well, given a deterministic data set.

Greedy 1 works with the following philosophy: when picking the next genome for the ensemble, pick the genome that yields the highest ensemble fitness. This heuristic was implemented with the following Python:

```python
def greedy_1_selection_accuracies(genomes, eval_func):
    # Some variables needed for the greedy algorithm
    # genomes_left is the genomes left to choose from
    # genomes_picked is the current best predicted k-wise ensemble
    genomes_left = {*genomes}
    genomes_picked = []
    accuracies = []

    # Remove the genome that improves
    # ensemble the most after each round
    while genomes_left:

        # Initialize this round's variables
        best_accuracy = float("-inf")
        best_genome = None

        # Find the genome that best
        # improves the current ensemble (genomes_picked)
        for genome in genomes_left:
            ensemble = [*genomes_picked, genome]
            ensemble_accuracy = eval_func(ensemble)
            if ensemble_accuracy > best_accuracy:
                best_accuracy = ensemble_accuracy
                best_genome = genome

        # Some housekeeping to finish off the round
        genomes_left.remove(best_genome)
        genomes_picked.append(best_genome)
        accuracies.append(best_accuracy)
    return accuracies
```

The actual number of ensembles evaluated is $n + (n - 1) + (n - 2) + \ldots + 2 + 1$, which evaluates to $O(n^2)$. While this is sufficient for smaller population sizes, this can be infeasible for larger populations, especially considering ensemble evaluation is relatively expensive.

Due to the lack of scalability in Greedy 1; another greedy algorithm was devised that produces similar results but consumes an order of magnitude less time; this algorithm is dubbed "Greedy 2."

4.1.4 Greedy 2

Greedy 2 shares many commonalities with Greedy 1, but makes an important tradeoff: it evaluates all genome fitnesses ahead of time. Instead of picking new genomes based on how they impact the ensemble, Greedy 2's philosophy is to pick the next best genome based on the genome's individual fitness. Consequently, one needs to evaluate the fitness of each genome once and then sort the genomes based on their individual fitness in descending order. Afterward, one must take the first k genomes to create an ensemble of size k.

Here is an example implementation of Greedy 2 in Python:

```
def greedy_2_selection_accuracies(genomes, eval_func):
    genomes_in_order = list(genomes)
    genomes_in_order.sort(reverse=True, key=lambda g: g.fitness)
    return __accuracies_for_genomes_in_order(genomes_in_order, eval_func)
```

It may not be readily obvious what *__accuracies_for_genomes_in_order* does[1], but it is not as important; the crux of the algorithm is in the simple `sort` call!

Due to the sort step, Greedy 2 is technically $O(n * log(n))$; however, Greedy 2 approximates $O(n)$ in practice since the genome evaluation step is what takes up the bulk of the time; the sort step is largely irrelevant.

4.1.5 Diversity-Based Round Robin

Diversity-Based Round Robin (DBRR) is similar to Greedy 2 in operation but also considers diversity (with a preference for diversity over fitness). The key observation is that in NEAT, genomes are divided up each generation based on their genetic diversity; this in turn, allows selecting genomes for ensembles based on their genetic differences. The philosophy behind DBRR is that after a certain threshold point, picking for diversity (i.e., genetic differences) will outperform picking for goodness (i.e., genome fitness).

For a concrete example of DBRR's philosophy, take a school group project. At first, picking the best few students in the class will likely produce the best resulting project (Greedy 2); however, if adding more and more students to the project, students with diverse backgrounds can likely bring new ideas to the table and continue to improve the project further.

DBRR works by first "speciating" a population's genomes into different species based on their genetic diversity along with a genetic diversity threshold (called the "speciation threshold"). Next, each species (a list of genomes with similar genetic structure) is sorted in descending order based on fitness. Finally, DBRR constructs ensembles by picking genomes from each (sorted) species round robin style. This structure may sound familiar; in fact, Greedy 2 acts the same as DBRR if the speciation threshold was set to infinity.

```
def diversity_rr_selection_accuracies(genomes, eval_func,
speciation_threshold=3.0):

    # Step 1: Divide genomes based on speciation threshold
    species = speciate(genomes, speciation_threshold)

    # Step 2: Sort genomes in each species
    # in descending order by their fitness
    for s in species:
        s.sort(reverse=True, key=lambda g: g.fitness)

    # Step 3: Pick the genomes from
    # each species round-robin style
    species = [deque(s) for s in species]
    genomes_in_order = []
```

[1] *__accuracies_for_genomes_in_order* creates the accuracies for a list of genomes in their ensemble order; e.g., the genomes for an ensemble of size 1 would be *genomes_in_order*$[0:1]$, and the genomes for an ensemble of size k would be *genomes_in_order*$[0:k]$.

```
    # For each round-robin round while there are
    # still species left to choose from
    while species:

        # Pick best genome for each species
        for s in species:
            genomes_in_order.append(s.popleft())

        # Remove empty species
        species = [s for s in species if s]

    # Step 4: Calculate the accuracies based on the picked genomes in order
    return __accuracies_for_genomes_in_order(genomes_in_order, eval_func)
```

In the proposed DBRR implementation, the speciation step takes $O(n)$ by utilizing one pass through the population. The Greedy 2 step takes $O(n/s * s * log(s)) = O(n * log(s))$, where s is the species size for a uniformly speciated population. However, do keep in mind that it is unlikely in practice for speciation to be uniform, and s may approach n (or even equal n exactly for bad/high speciation thresholds!). However, as discussed in the previous section on Greedy 2, this step really resembles O(n) in practice. Finally, the ensembling and evaluation steps both make one pass through and are consequently $O(n)$. Summing these steps up, DBRR is $\Theta(n * log(s))$, where $1 <= s <= n$. Regardless, DBRR resembles $O(n)$ in practice because the sort step is far from the bottleneck (sorting numbers is much faster than genome evaluation).

4.2 Machine Learning Tasks

Before beginning experimentation, the team needed to choose a set of machine learning tasks with some objective function to optimize. The team focused on two subdomains of machine learning, classification and reinforcement learning. Within each subdomain, the team selected well-studied tasks so that each experimentation result could be compared with a baseline.

4.2.1 Classification Data Sets

Since ensembling is traditionally used for classification tasks, the team decided to test their novel heuristics and algorithms with well-known classification data sets to start.

MNIST To test the team's team-based fitness heuristics and ensembling algorithms, the team initially chose MNIST as the classification data set. MNIST is a collection of 60,000 handwritten digits between 0 and 9. Each input image is comprised of a 28x28 matrix, totaling 784 inputs. There are ten possible outputs representing the probability for each digit.

The team tried using standard NEAT to classify MNIST data; however, such a task proved unsuccessful. Even after running for 24 hours, the team still had a best accuracy of 9.8%, which is as good as a random guess. After viewing the created genomes, the team determined that MNIST was too complex a task for genetic algorithms evolved with NEAT. In order to make accurate predictions on MNIST, a model needs to be of adequate size and establish numerous connections between inputs and outputs. Although this is theoretically achievable, the team's attempt at evolving a suitable model using NEAT could not grow big enough, and it would have taken an impractical amount of time to do so. As a result of the inherent complexity of MNIST, the team ultimately decided not to include it in any further experimentation.

UCI Heart Disease Following the challenges with MNIST, the team selected the University of California Irvine's Heart Disease data set. This is a multivariate, multi-class classification problem with thirteen features, three hundred instances, and imbalanced data with five possible target classes. It is an appropriate challenge for ensemble learning as an individual classifier (logistic regression) will struggle to fully capture the class relationships.

Unlike MNIST's 784 inputs, the UCI Heart Disease data set only has 13 attributes and tries to predict a single number.

Here are the 13 attributes:

1. age (Age of the patient in years)
2. sex (Male/Female)
3. cp chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])
4. trestbps resting blood pressure (resting blood pressure (in mm Hg on admission to the hospital))
5. chol (serum cholesterol in mg/dl)
6. fbs (if fasting blood sugar $>$ 120 mg/dl)
7. restecg (resting electrocardiographic results) – Values: [normal, stt abnormality, lv hypertrophy]
8. thalach: maximum heart rate achieved
9. exang: exercise-induced angina (True/ False)
10. oldpeak: ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
12. ca: number of major vessels (0-3) colored by fluoroscopy
13. that: [normal; fixed defect; reversible defect]

Consequently, the UCI Heart Disease data set is a much easier task for a model. The team split the data into a train/test split of 80/20, consistent with that of OET (so the team can compare their own ensemble performances with OET's baseline).

4.2.2 Reinforcement Learning

After some initial success with classification, the team decided to experiment with reinforcement learning, selecting several tasks of varying complexity.

Acrobot Acrobot was the most interesting of all the reinforced learning environments. The environment is a double pendulum, an example can be seen in Figure 3. The model can move the first linkage. The goal of the model is to get the double pendulum as high as possible. The action space is to apply negative torque, no torque, or positive torque. The observation space is the Cosine of theta1, Sine of theta1, Cosine of theta2, Sine of theta2, Angular velocity of theta1, and Angular velocity of theta2.

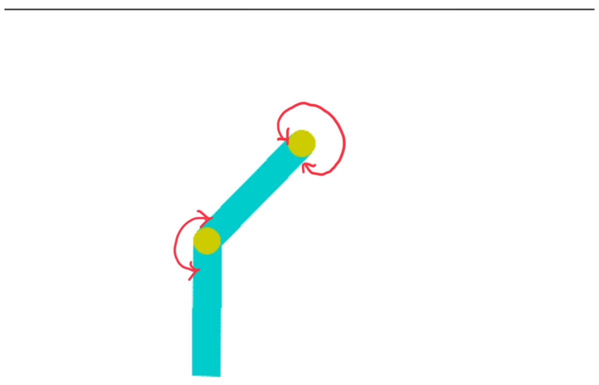


Figure 3: Double Pendulum

What made Acrobot interesting was the reward function. Originally, the reward was calculated by the linkage tip being above a certain height. With this reward function, the model never learned anything. The linkage would keep flailing around and never seemed to get better. What was happening was the reward function was never giving a good score to any ensemble because the tip never got above the threshold to get points. This is because the model would have to guess a reasonably good solution randomly. In theory, it would be possible for a model to guess a good solution, but not probable. After some experimentation, the team realized that reward functions must be a gradient. There needed to be a continuous space of solution rewards. This is because then, even if a model is just slightly better, it will get a better reward. NEAT does not know what to do if all models have the same reward.

As a solution, a reward function was implemented which calculated the average height of the tip, then subtracted the number of generations it took for the model to reach the target height. This was done to better determine which models were capable of swinging the double pendulum higher than others and at achieve the task faster. This technique saw immediate success in helping NEAT differentiate individuals in the population.

Bipedal Walker Bipedal Walker is a much more complex reinforcement learning environment. The model's objective is to control a bipedal robot and make it walk as far as possible while avoiding obstacles. The observation space is 24 dimensional, consisting of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with the ground, and 10 lidar rangefinder measurements. The action space is continuous, with the model controlling the 4 motors in the hips and knees of the walker.

The complexity of this environment makes it a challenging problem for reinforcement learning. The observation space is much larger than in simpler environments like Cart Pole or Hill Climbing, and the action space is continuous, making it difficult for the model to find the optimal solution. None of the models tested were able to converge on this task. One explanation is that the task was to complex to be sufficiently optimized within the time frame allowable. Nonetheless, no further results will be presented for Bipedal Walker.

4.3 Tools and Technology Used

In order to complete the project, the team utilized a wide variety of tools and resources, ranging from ML ops to the actual servers running experiments.

4.3.1 Weights and Biases

Initially, the team struggled with manually configuring the team's experiments. Weights and Biases (WandB) is a powerful tool that the team discovered later during this project, which could have saved us a lot of time if found earlier. It is primarily used for data collection, organizing experiments, and providing graphical representations of results, including analyzing hyperparameter importance and correlations.

WandB excels in collecting data, logging everything from hyperparameters to the genomes produced, and ensuring that all aspects of an experiment are recorded for future reference. This allows for easy review and analysis of past experiments.

Organizing experiments is streamlined with WandB, as it allows for grouping sets of runs, calculating standard deviations for specific experiments, and conducting hyperparameter sweeps. WandB automates the setup and execution of experiments, helping to optimize hyperparameters and visualize their correlation with performance, even if the correlation is negative.

The Parameter Importance Panel in WandB enhances hyperparameter tuning by identifying the most important hyperparameters in terms of predicting model performance. It calculates both importance and correlations for hyperparameters, providing a deeper understanding of their impact on the model. Importance measures the degree to which each hyperparameter influences the chosen metric, while correlations capture linear relationships between individual hyperparameters and metric values. These two measures combined give valuable insights into the most critical hyperparameters.

4.3.2 Slurm

Slurm (Simple Linux Utility for Resource Management) is an open-source job scheduler and resource management system used for managing and scheduling workloads on Linux clusters. Slurm on Turing, WPI's cloud computing resource, posed several challenges during the course of the project.

One of the issues the team encountered was a limitation on the number of concurrent jobs that could be run per user. Each user was restricted to running only 75 jobs simultaneously, which presented constraints on the team's ability to perform parallel processing for large-scale experiments.

Another major limitation of Turing is the inability to install packages directly. Instead, it was only possible to load in preexisting modules for any necessary functionality. While Turing generally provided a wide array of modules, there were a few instances where the required modules were unavailable. This forced us to find alternative solutions or run certain tasks locally. For instance, the team had to render the videos of the OpenAI gym environments on the team's local machines, as the appropriate module was not available on Turing.

Besides the issue with unavailable modules, the team also faced some other challenges while using Slurm. The learning curve was steep, and it took time to understand the system and its intricacies. However, once the team became familiar with the system, using Slurm was not as problematic as initially thought. Despite the initial difficulties, the benefits of Slurm started to become more apparent as the team adapted to the platform.

4.3.3 Anaconda

This project used Anaconda as the team could not install packages on Turing and an Anaconda environment can be standardized across machines/users. Anaconda allowed users to install Python packages in the user-space and not system-wide; which is beneficial since the team did not have administrative privileges.

4.3.4 PyTorch

Since PyTorch-NEAT used PyTorch, the team ended up using PyTorch a lot. PyTorch is designed for efficient matrix multiplication, which constituted the majority of calculations in this project. It was also GPU accelerated. It was thought that GPU acceleration was going to be important at the beginning, but realized that it was more efficient to use CPU computing. This is because of two factors. Firstly Turing only has so many GPUs, which are in high demand. This makes it so the team would not have been able to run many experiments in parallel. Secondly, the overhead of moving the model from the CPU to the GPU was large enough that it was more efficient to run the model on the CPU

4.3.5 OpenAI Gym

OpenAI Gym is an open-source package which provides premade environments for reinforcement learning tasks. The team recommends utilizing this package for further reinforcement learning projects.

4.4 Frameworks

Once the team decided that the work would be based on NEAT, the team started looking at NEAT implementations that the team could use as a starting point. There is no point in reinventing the wheel. Further research found a few options Deep-Neuroevolution, Python-NEAT, and PyTorch-NEAT. Deep-Neuroevolution never ran, because some dependencies did not exist anymore. This caused the team to not look into Deep-Neuroevolution anymore. Python-NEAT is what the team originally thought the team's work would be based on. Once the team began to code, the team ran into some problems with it. It was not GPU accelerated, which was thought to be important at the beginning of the project, even though it was not in the end. One con was no experiment wrapper. It also was a pain to work with. So, in the end, the team went with PyTorch-NEAT. This gave us GPU acceleration and experiment wrappers for free. The downsides were it does not evolve the activation functions and does not have back propagation support. Neither of which were integral to the project.

4.5 Heuristics

A heuristic is an approach to problem-solving that employs a practical method not guaranteed to be optimal nor perfect, but is sufficient for immediate goals. They serve as rules-of-thumb for an algorithm to follow in order to approach a solution; however, due to the stochastic nature of the NEAT algorithm and evolution as a whole, there is no guarantee that an optimal solution will emerge. The team aims to study the effect of two categories of heuristics: fitness heuristics and ensemble heuristics. It is the aim of the team to determine which choice of fitness and ensemble heuristics are most suited toward evolving optimal ensembles.

4.5.1 Fitness Heuristic

Fitness heuristics are used during evolution only to determine a given genome's fitness which in turn determines which genomes are selected to persist and reproduce. As a control, each genome is evaluated using a fitness function corresponding to a specific machine learning task. For example, in a classification task, a genome is evaluated by testing the accuracy or cross-entropy loss of its phenotype (feed forward neural network). For a reinforcement learning task, genomes are evaluated by the amount of reward earned during a task. In addition to the control, the team proposes an alternative fitness heuristic *Constituent Ensemble Evaluation* which evaluates the fitness of a genome by how well ensembles containing that genome perform. The team hypothesizes that such a fitness heuristic will aid in final ensemble performance, in line with the teams/islands approach taken by OET.

Constituent Ensemble Evaluation In NEAT, the genome which represents a given neural network (phenotype) in the population is evaluated each generation using a fitness function. However, in the team's experimental design, a different fitness heuristic was used, named Constituent Ensemble Evaluation. In lieu of evaluating an individual genome based on its individual fitness, this heuristic instead iterates through possible teams, or *combinations*, of a given size that contain the genome and grade the genome based on its overall team (ensemble) performance. There are a number of considerations with this method, including run time, ensembling method, and fitness metric.

Run Time Considerations - Random Ensemble Sampling For small populations and ensemble sizes, constituent ensemble evaluation on all possible combinations for a given genome is computationally light. However, the total number of combinations grows factorially with a linear increase in the population size, denoted by the equation:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

where n is the population size and r is the group size. As the constituent ensemble evaluation is computed for each generation, the time to complete a single experimental trial is thus greatly impacted by the choice to fully evaluate all possible constituent ensembles. While this observation only impacts the run-time of only a particular run, the concept is important because hundreds of trials must be conducted. As such, the constituent ensemble evaluation is computed over a sample of all total combinations for a given ensemble size. This ensemble sample can then be optimized for a given learning task using hyperparameter search.

Ensembling Method - Soft Voting Once an ensemble has been formed, it must then be decided how the ensemble will come to its final vote. In soft voting, every individual of an ensemble provides a probability value that a specific data point belongs to a particular target class. This is achieved by summing the final activations of each neural network, then converting the summed activations to a probability vector, in this case using the Softmax function:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

This then produces the final probability vector to make class predictions. The choice of fitness metric will determine how this probability vector is then converted into a genome fitness score.

Fitness Metrics At each generation, each individual in the population is evaluated using a fitness function. This fitness function uses a given metric and converts the resulting numeric value into a fitness score which NEAT can then use to optimize the population through natural selection (fitter members have a higher chance of reproduction and mutation). During experimentation, the choice of fitness metric was varied to determine which method helped train better ensembles.

Genome Fitness Metric - Reward, Accuracy, and Loss In the traditional NEAT algorithm, populations evolving to improve on a reinforcement learning task use the reward metric. The reward of a reinforcement learning task differs by the type of environment. For example, the Acrobot environment normally returns a value of -1 for each time step that the tip of the double pendulum has not reached the target height. For classification problems, there is a choice between using the accuracy of predictions made or the cross-entropy loss. Accuracy simply measures the number of correct predictions divided by the total number of predictions. Since high accuracy is the objective of classification problems, this is a straightforward criteria to judge an individual genome. Alternatively, cross-entropy loss is a commonly used metric that measures the dissimilarity between the predicted and actual outputs. Specifically, it measures the average number of bits required to represent the true distribution of the classes, given the predicted distribution. In other words, the advantage of using cross-entropy loss as a fitness function is that it selects individuals with confident predictions.

Constituent Ensemble Fitness Metric - ACER, ACEA, and ACEL To build upon the standard individual genome fitness metrics, three constituent ensemble fitness metrics were used in the course of the team's experiments: Average Constituent Ensemble Reward (ACER), Average Constituent Ensemble Accuracy (ACEA), and Average Constituent Ensemble Loss (ACEL). For experiments with reinforcement learning, an agent seeks to maximize the reward. The ACER of an individual genome represents the average reward across the sample of ensembles containing the individual. Likewise, for classification tasks, the ACEA and ACEL represent the average accuracy and cross-entropy loss across the sample of ensembles containing the individual.

Using both Genome and Constituent Ensemble Fitness Metrics - Warm-up One concern during initial testing was that during early generations, constituent ensemble evaluation would limit useful genomes from persisting due to the chance that they would be grouped with non-complimentary genomes and thus have lower fitness. To combat this, an alternative fitness heuristic was implemented which would evaluate both the genome fitness and the constituent ensemble fitness, then combine them in a weighted fitness function. This function would initially weigh the genome fitness as 90% of the overall fitness, and the constituent ensemble fitness as 10%. These percentages are inversely proportional with time, such that the final weights result in genome fitness as 10% and constituent ensemble fitness as 90% of the overall fitness. The team named this method *constituent ensemble with warm-up* because the technique "warms-up" the evolution process using "island" fitness before switching to "team" fitness.

4.5.2 Ensemble Heuristic

After a population has been trained for a certain number of generations using the fitness heuristic, ensembles are generated using one of the ensemble heuristics as described in subsection 4.1, and are then evaluated against test data (Classification). Note that for the remainder of the paper, "ensembling algorithm" will be used interchangeably with "ensembling heuristic." Some ensembling heuristics like greedy search use forward stepwise selection to create optimal ensembles. A novel approach implemented in this paper uses the species difference coefficient proposed in NEAT to select the best candidate from diverse species, ensuring that each species is represented. The hypothesis being that diversity is an appropriate regularization term for creating ensembles that avoid overfitting by increasing the variance in the final ensemble.

4.6 Experimental Design

To reiterate, the purpose of this experimentation is threefold. First, it is the aim of this paper to determine an optimal configuration of the evolutionary NEAT parameters as follows:

1. Add Connection Mutation Rate (Chance that a genome will add a connection between two nodes)

2. Add Node Mutation Rate (Chance that a genome will add a new node to the network)

3. Connection Mutation Rate (Chance that a connection will be reassigned to a different node)

4. Connection Perturbation Rate (Chance that a connection weight will be increased or decreased by a small amount)

5. Crossover Re-enable Connection Gene Rate (Chance that a connection that has been disabled in both parent genomes is reactivated during breeding)

6. Percentage to Save (Proportion of fittest individuals to persist to the next generation)

7. Speciation Threshold (Degree to which a genome must differ from the existing species to be categorized as a new species)

8. Use Bias (Whether to include a bias in the activation function)

Second, it is the aim of this paper to optimize the hyperparameters associated with NEAT which affect the population as a whole. These include custom fitness heuristics specific to constituent ensemble evaluation. They are as follows:

1. Population Size (The initial size of the population)

2. Number of Generations

3. Use Genome Fitness (Whether to include individual fitness in the fitness function)

4. Genome Fitness Metric (For Classification, whether to use Accuracy or Cross-Entropy Loss for genome fitness)

5. Use Fitness Coefficient (Whether to gradually increase the weight of ensemble fitness and decrease the weight of genome fitness in the combined fitness function, "Warm-up")

6. Ensemble Fitness Metric (For Classification, whether to use ACEA or ACEL for genome fitness)

7. Generational Ensemble Fraction (The fraction of the total population to set as the ensemble size for constituent ensemble evaluation)

8. Candidate Limit (The fraction of the total number of combinations (ensembles) to evaluate during constituent ensemble evaluation)

Third, this papers seeks to determine the best choice of ensemble heuristic to create the final ensemble. These are:

1. Random

2. Greedy1

3. Greedy2

4. Diversity

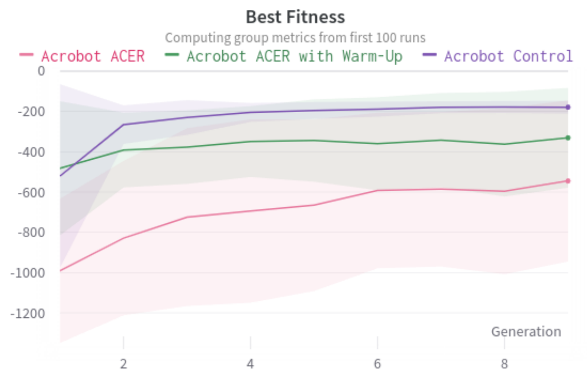
Realistically, finding the best possible combination is not an achievable goal. Evaluating every possible combination would be extremely computationally expensive. Instead, the search space is limited by making some assumptions about the likely performance of certain combinations using preliminary experimentation. As more is understood about the subtle behavioral changes in the algorithm from certain choices, certain parameters can be fixed as defaults for later searches. This paper utilizes the Weights and Biases (WandB) platform, which enables efficient configuration and tracking of parameter optimization trials. There is a variety of tools available to assist with this search, referred to as a sweep in WandB. The first is random search. Given a set of parameters and possible ranges, WandB will initialize multiple sweeps with different parameter configurations. This is a surprisingly effective method for determining the importance of certain parameters when combined with WandB's built-in sensitivity analysis and correlation tools. After an initial round of random sweeps, the performance of certain trials are graphically examined with respect to their parameters. This allows further narrowing down of the optimal values for certain parameters. Bayesian optimization [10], which updates the probability of selecting certain configurations based on previous performance will then be used as a comparison to the graphical results. This is an essential tool, especially when considering the highly stochastic and non-deterministic nature of the NEAT algorithm, meaning that two runs with identical configurations may have different results. Thus, the Bayesian approach ensures that configurations are evaluated multiple times and that the likelihood of finding useful parameters is less dependent on random chance. Finally, the estimated best parameters will be chosen and evaluated using multiple trials using different splits of the training and testing data.

5 Reinforcement Learning Results

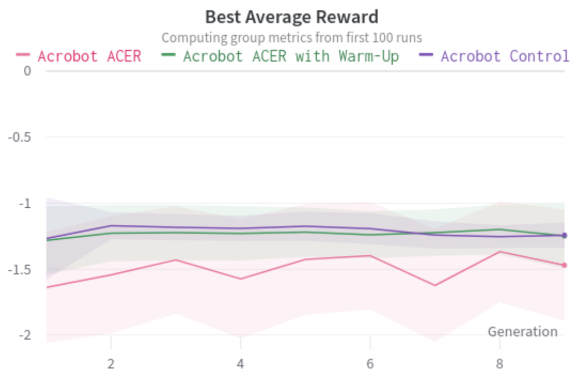
The following section describes the performance of over 200 trials using various fitness heuristics on the Acrobot reinforcement learning task. The results are organized into two sections: training performance and ensemble testing performance.

5.1 Training Performance

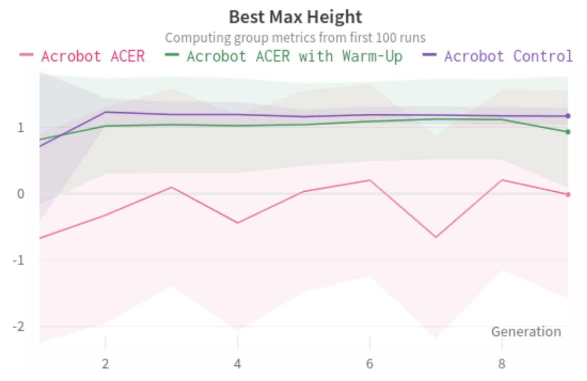
Figure 4 depicts four key metrics collected during training using the NEAT algorithm and varying fitness heuristics. Some key definitions: *Reward* is the sum of the height achieved at each frame by the tip of the double pendulum. The pendulum starts at a height of *-2* and attempts to reach a height of *1* or above. *Step Completed* is set as the frame which the tip of the double pendulum reaches the target height. *Fitness* is calculated as the reward minus the step completed. When using constituent ensemble evaluation (ACER), this is calculated as the average fitness across the sample of ensembles containing the genome being evaluated. *Best Fitness* (a) describes the maximum of the population fitness at each generation. *Best Average Reward* (b) describes the maximum of the average reward obtained at each generation. *Best Max Height* (c) represents the maximum height achieved at each generation. *Best Step Completed* (d) represents the minimum step completed at each generation. When aggregated over multiple trials, these metrics describe the capability of each fitness heuristic to adequately improve the population over time.



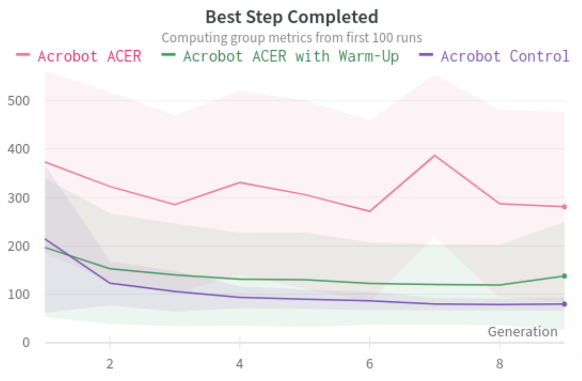
(a) Best Fitness of Acrobot with Standard Deviation



(b) Best Average Reward of Acrobot with Standard Deviation



(c) Best Max Height of Acrobot with Standard Deviation)



(d) Best Step Completed of Acrobot with Standard Deviation

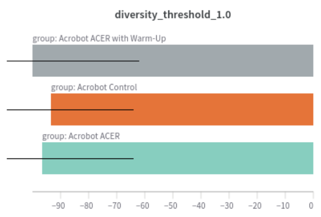
Figure 4: These figures display the mean and standard deviation of four metrics collected during training using the NEAT algorithm.

Based on the observations in Figure 4 it is clear that ensemble regularization does not yield better results than the baseline even with warm-up. The baseline NEAT algorithm is able to get the double pendulum to the desired height in fewer steps than baseline NEAT. The shrinking standard deviation bounds for the control case indicate that most populations converge to a solution, while the constant width bounds for ACER and warm-up indicate that many populations fail to converge to a solution. The team theorizes that the Acrobot task does not benefit from ensemble regularization as the task is simple enough that the solution space is easily traversed through stochastic search alone.

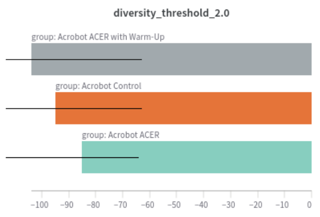
5.2 Ensemble Testing Performance

Following training, the final populations were then passed to various ensemble algorithms to determine which combination of fitness heuristic and ensemble heuristic yielded the best ensembles. The final ensembles were tasked with playing the Acrobot environment as done in training. The objective of the game is to swing the Acrobot to reach a target height of *1* in as few frames as possible. During this test, the ensembles earned a reward of *-1* for each frame spent below the target height. Thus, an ensemble is useful if it maximizes the total reward.

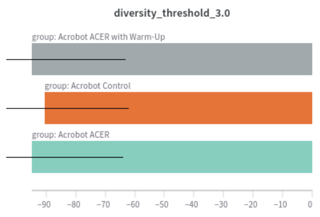
Figure 5 describes the average performance of ensembles on the Acrobot task. The ensembles were generated using the algorithms described in subsection 4.1. For each algorithm, the results are grouped by choice of fitness heuristic (Contol, ACER, and ACER with Warm-Up).



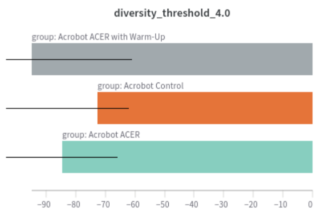
(a) Diversity Threshold 1 Ensemble Heuristic with Standard Deviation



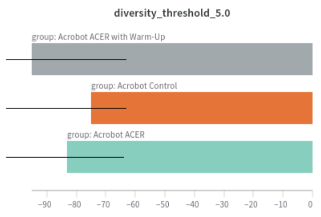
(b) Diversity Threshold 2 Ensemble Heuristic with Standard Deviation



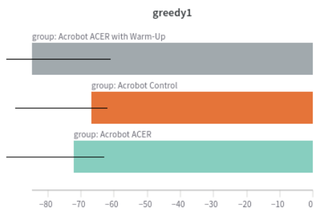
(c) Diversity Threshold 3 Ensemble Heuristic with Standard Deviation



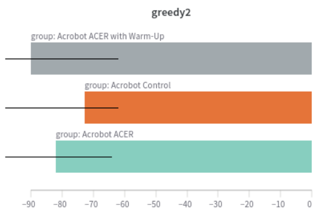
(d) Diversity Threshold 4 Ensemble Heuristic with Standard Deviation



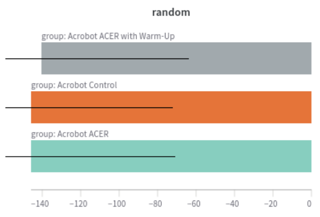
(e) Diversity Threshold 5 Ensemble Heuristic with Standard Deviation



(f) Diversity Greedy 1 Ensemble Heuristic with Standard Deviation



(g) Diversity Greedy 2 Ensemble Heuristic with Standard Deviation



(h) Diversity Random Ensemble Heuristic with Standard Deviation

Figure 5: These figures display the mean reward for the respective ensemble heuristic for the baseline NEAT, ACER, and ACER with warm-up algorithms. The black line on each bar indicates the minimum and maximum reward. The rewards are all negative in this figure, with 0 being on the right since it is the best theoretical reward possible. The closer the model gets to zero the better it did on the task.

Figure 5 indicates that generally, ensembles constructed from populations selected using individual fitness perform better than ACER and ACER with Warm-Up. The best overall performance is achieved by the Greedy1 algorithm which is the most exhaustive approach. The team concludes that constituent ensemble evaluation is not an appropriate method for evolving ensembles of neural networks for the Acrobot task.

6 Classification Results

6.1 Best Overall Fitness Heuristic

An experiment was conducted by initializing 150 NEAT trials with randomly varying parameters to evolve ensembles for classification on the UCI Heart Disease data set. The key metric used in this section is the mean best ensemble accuracy. Each generation, DBRR, defined in subsubsection 4.1.5, creates ensembles from the population. The mean best ensemble accuracy is computed by taking the mean of the test accuracy returned by the best ensemble created by DBRR across all trials. The primary goal in this experiment was to determine which fitness heuristic, control, constituent ensemble, or warm-up, was most beneficial in evolving ensembles. The results of this experiment is documented in Figure 6. It was observed that, while constituent ensemble evaluation performed worse than the control, the warm-up approach, which combines individual and constituent ensemble fitness, outperformed both.

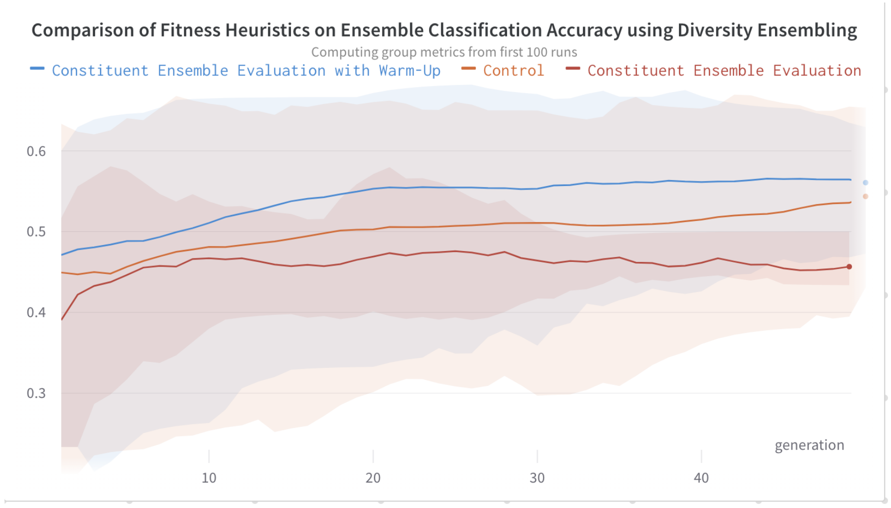


Figure 6: This figure summarizes the ensemble performance of 150 trials over time. The trials are grouped by the chosen fitness heuristic. The graph indicates that while constituent ensemble evaluation alone performs worse than the control, combining both approaches (warm-up) performs slightly better than either on average.

It should be noted that the standard deviation of the mean best ensemble accuracy (depicted as the shaded areas on the graph) are large in magnitude. This is due to the stochastic nature of NEAT, wherein some populations are better or worse depending on their initial state.

Some upward trend in the graphs is observed, indicating that learning is taking place, though slowly. One explanation is that, over time, the population begins to grow to a point where the diversity ensembling algorithm has too high a degree of freedom to over-fit the data; this is associated with *high variance*. On the other hand, this explanation would imply that, since the population continues to grow, the effect of overfitting should magnify, resulting in decreasing performance over time. This is seen in the constituent ensemble group, but not in the warm-up group, implying that constituent ensemble evaluation may work to reduce the variance, or increase the bias, which results in a more optimal bias-variance trade-off.

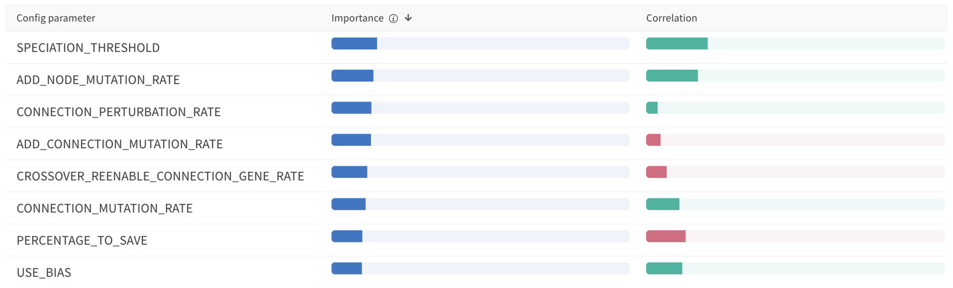
6.2 Discussion of NEAT Parameter Importance on Ensemble Classification Accuracy

The purpose of this section is to determine which, if any, NEAT parameters, defined in item 4.6 are significantly important in improving the accuracy of ensembles. The results are organized by ensemble heuristic (random, greedy 1, greedy 2, DBRR), and further by fitness heuristic (control, constituent ensemble, and constituent ensemble with warm-up). The results are displayed as figures which display the relative importance and correlation of the NEAT parameters within their respective groups. Correlation is the linear correlation between the hyperparameter and the chosen metric. While it can reveal that a certain parameter is associated with a change in accuracy, it does not necessarily show causation. Relative importance is the degree to which each parameter was useful in predicting the chosen metric. It is useful as it accounts for interactions between parameters, rather than solely the interaction between the parameter and the metric.

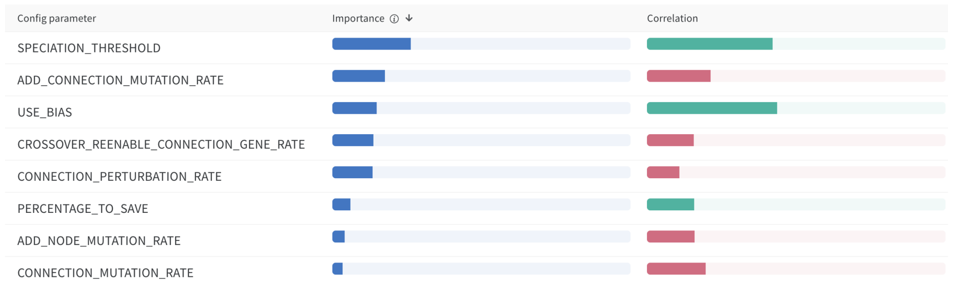
The results of parameter importance and correlation to ensemble performance are mixed. In some cases, there are straightforward observations across multiple ensemble heuristics and fitness heuristics, such as the *speciation threshold* having a negative correlation to ensemble accuracy. The likely explanation is that lowering the *speciation threshold* allows more diverse networks to persist, which improves the final ensemble performance. A less explicable observation is the negative correlation between *connection perturbation rate* and non-random (greedy 1, greedy 2, diversity) ensemble accuracy. It may be that too high a degree of connection perturbation tends to undo any beneficial connections in the previous generation, weakening the population as a whole. Overall, parameter importance was inconsistent within ensemble heuristics and fitness heuristics. This is especially evident in the diversity heuristic in subsubsection 6.2.4, where the only somewhat significant parameter was *use bias* for the control. Further, the analysis of warm-up showed very low importance and correlation across the board. While these results are displayed to demonstrate the fulfillment of the experiment, there are no specific recommendations that can be made that are supported empirically.

6.2.1 Ensemble Heuristic - Random

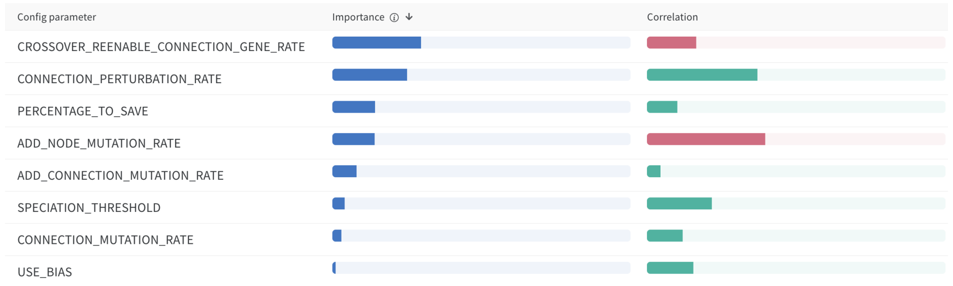
Figure 7 indicates the importance of each NEAT parameter and its correlation to the final classification ensemble performance using the Random heuristic in the control, constituent ensemble, and constituent ensemble with warm-up groups. Green indicates a positive correlation and red is negative.



(a) Control



(b) Constituent Ensemble

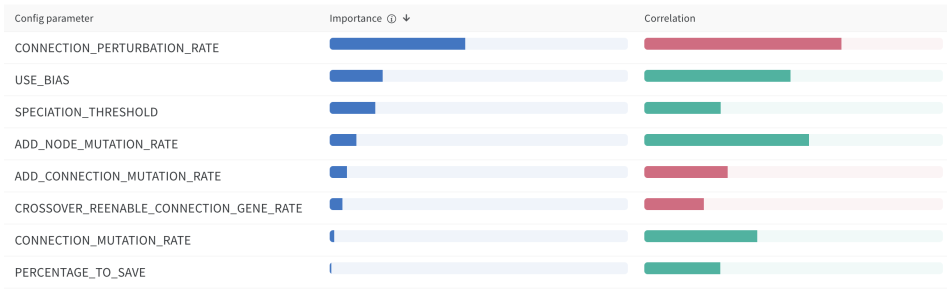


(c) Constituent Ensemble with Warm-up

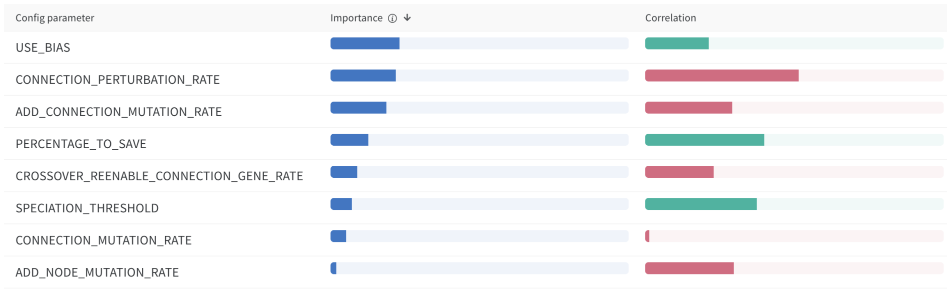
Figure 7: NEAT Parameter Importance w.r.t Random Ensemble Accuracy. A random heuristic is akin to estimating the mean fitness of the population. No one parameter stands out as having high importance in improving the mean fitness of the population. It is uncertain whether this represents the true significance of these parameters, or that there were insufficient trials to adequately determine their importance.

6.2.2 Ensemble Heuristic - Greedy 1

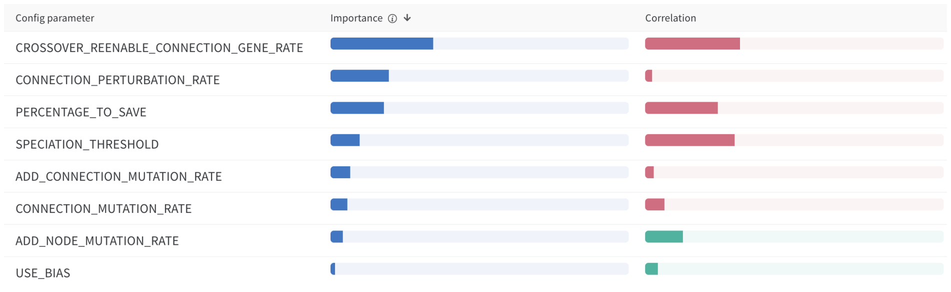
Figure 8 indicates the importance of each NEAT parameter and its correlation to the final classification ensemble performance using the Greedy1 heuristic in the control, constituent ensemble, and constituent ensemble with warm-up groups. Green indicates positive correlation and red is negative.



(a) Control



(b) Constituent Ensemble



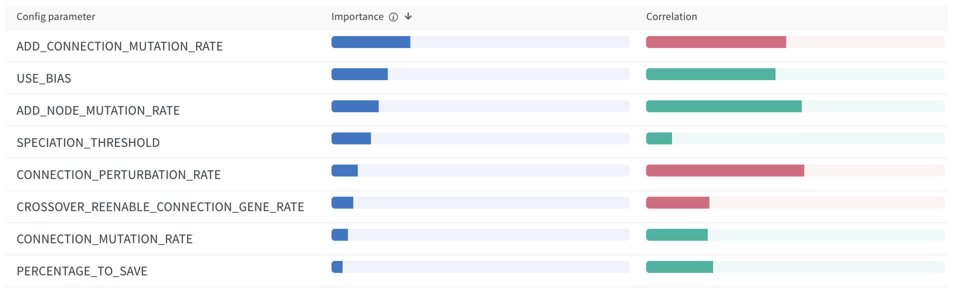
(c) Constituent Ensemble with Warm-up

Figure 8: NEAT Parameter Importance w.r.t Greedy1 Ensemble Accuracy. Connection perturbation rate is an important parameter across all trials, but has low linear correlation for (c), possibly indicating a nonlinear correlation with an optimal value somewhere near the middle of the values tested

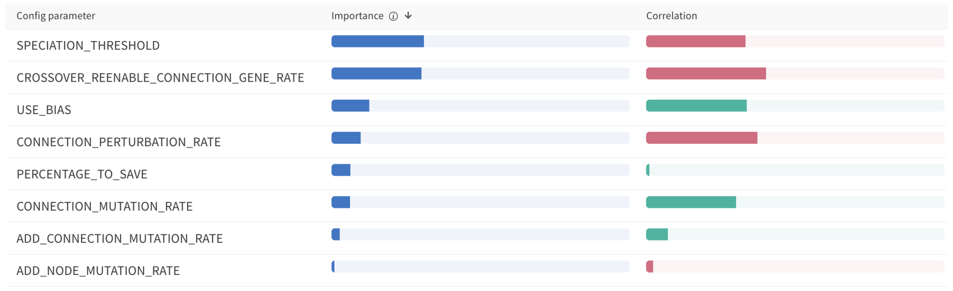
Figure Figure 8 presents an interesting result. The importance and correlation of the parameters in control (a) and constituent ensemble (b) are similar, favoring a bias term and a low connection perturbation rate. These are reflected in constituent ensemble with warm-up (c), but to a lesser degree. Most striking is that while (a) and (b) seem to benefit from a high speciation threshold, the opposite is true for (c).

6.2.3 Ensemble Heuristic - Greedy2

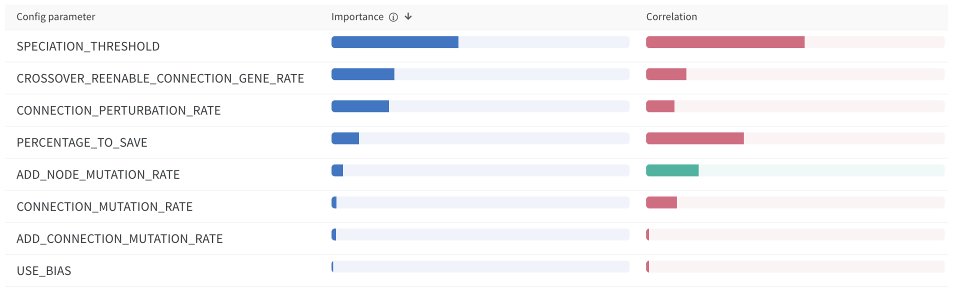
Figure 9 indicates the importance of each NEAT parameter and its correlation to the final classification ensemble performance using the Greedy2 heuristic in the control, constituent ensemble, and constituent ensemble with warm-up groups.



(a) Control



(b) Constituent Ensemble



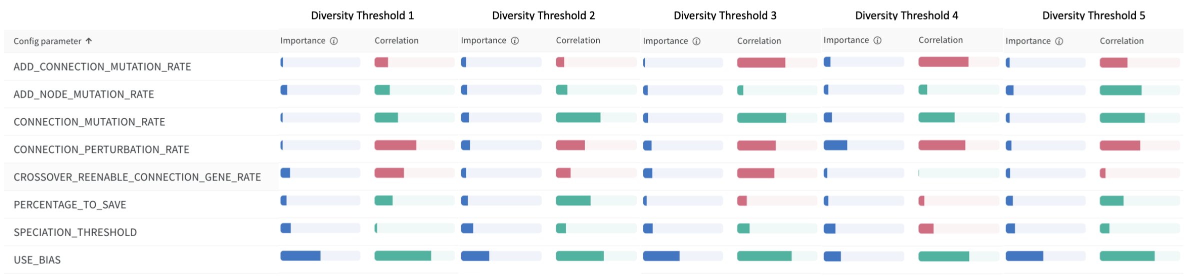
(c) Constituent Ensemble with Warm-up

Figure 9: NEAT Parameter Importance w.r.t Greedy2 Ensemble Accuracy. Speciation threshold again emerges as a significant parameter for constituent ensemble evaluation. Its negative correlation indicates that by protecting diverse genomes, the resulting population is more fit for ensembles

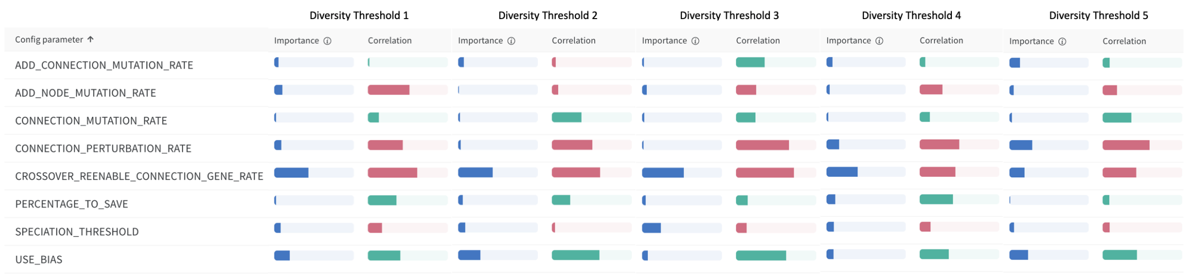
The results in Figure 9 appear similar to those of Figure 8. This indicates that the Greedy 1 heuristic and Greed 2 heuristic are affected similarly by the same parameter configurations.

6.2.4 Ensemble Heuristic - Diversity

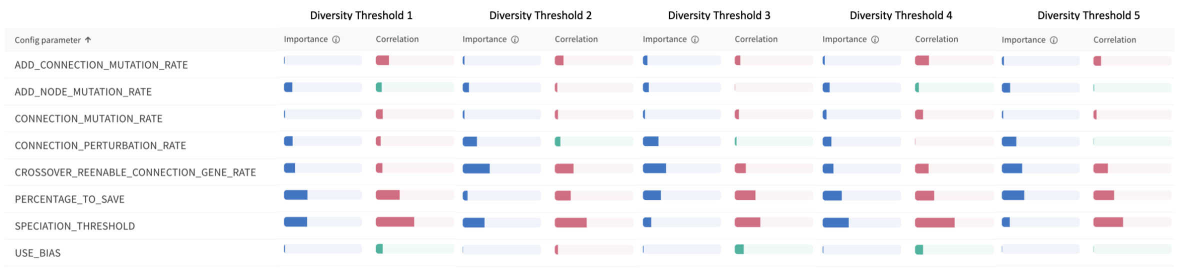
Figure 10 indicates the importance of each NEAT parameter and its correlation to the final classification ensemble accuracy using the Diversity heuristic in the control, constituent ensemble, and constituent ensemble with warm-up groups. The diversity heuristic was evaluated using a speciation threshold ranging from 1 to 5.



(a) Control



(b) Constituent Ensemble



(c) Constituent Ensemble with Warm-up

Figure 10: NEAT Parameter Importance w.r.t Diversity Ensemble Accuracy. These results fail to detect whether any parameter is significantly important for determining ensemble accuracy using the diversity heuristic

These results are difficult to draw any strong conclusions from. Especially for (c), no significant parameter emerges. This may indicate that the diversity heuristic is less dependent on the choice of hyperparameter than previous heuristics.

6.3 Hyperparameter Choice on Constituent Ensemble Evaluation

The purpose of this section is to examine the effect of hyperparameter choice on the ensemble test accuracy of genomes evolved using constituent ensemble evaluation. The hyperparameters are described in item 4.6. In total, 150 trials were conducted by randomly selecting hyperparameter configurations. The key metric used in this section is the mean best ensemble accuracy. Each generation, DBRR, defined in subsubsection 4.1.5, creates ensembles from the population. The mean best ensemble accuracy is computed by taking the mean of the accuracy returned by the best ensemble created by DBRR across all trials. This metric determines whether runs using a certain hyperparameter choice create better ensembles than others, and whether those hyperparameter choices improve the population over time.

As documented in figure Figure 11, the effect of hyperparameter choice has a demonstrable impact on the performance of constituent ensemble evaluation to produce accurate ensembles using the diversity heuristic.

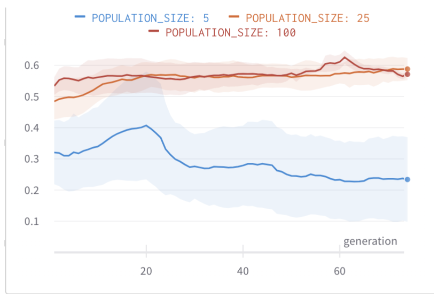
Population Size A low population size is not sufficient to increase the population diversity such that the diversity ensemble heuristic yields high accuracy ensembles. Higher performance is seen with larger population sizes of 25 and 100. Being that they are nearly equivalent, size 25 is recommended to reduce the compute time of the algorithm.

Number of Generations The number of generations has a positive impact on the overall performance of the final ensemble. One interpretation is that the hypothesis is correct that constituent ensemble evaluation is a viable approach to training ensembles of classifiers using a diversity ensemble heuristic. However, it remains to be seen whether continuing evolution beyond 200 generations would continue to yield better performance or simply plateau.

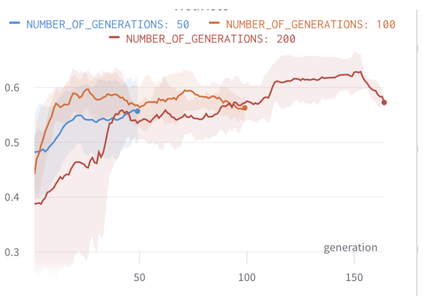
Use Genome Fitness The use of genome fitness in addition to constituent ensemble evaluation yields much better performance than solely using constituent ensemble evaluation. One interpretation of this result is that constituent ensemble evaluation helps select for complimentary networks but only after the population achieves a sufficiently high level of diversity. This would indicate the constituent ensemble evaluation is a better suited as a fine tuning method to avoid overfitting.

Genome Fitness Metric For classification tasks, choosing to reward accuracy of loss of the individual genomes rewards higher overall performance, though cross-entropy loss appears more consistent (lower standard deviation).

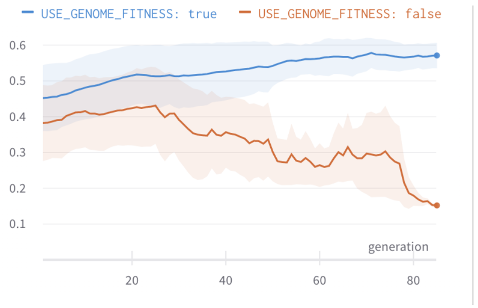
Ensemble Fitness Metric Conversely, the choosing ACEL over ACEA yields higher performance. One interpretation is that ACEL encourages ensembles with a higher degree of consensus, which results in higher generalization.



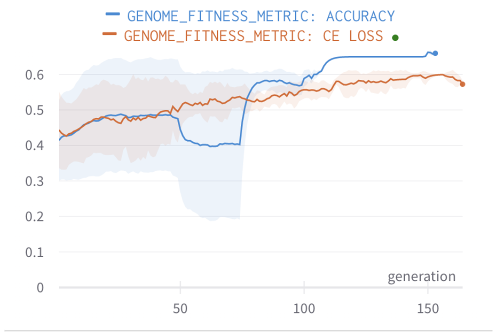
(a) Population Size



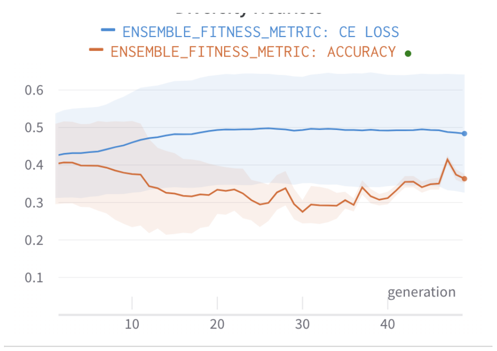
(b) Number of Generations



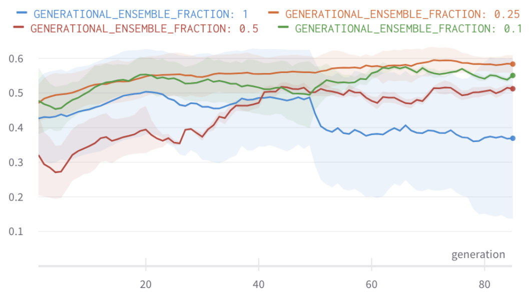
(c) Use Genome Fitness (Warm-Up)



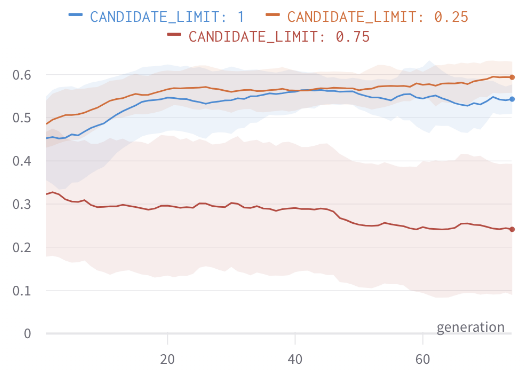
(d) Genome Fitness Metric



(e) Ensemble Fitness Metric



(f) Generational Ensemble Fraction



(g) Candidate Limit

Figure 11: These figures display the mean best ensemble accuracy over time with respect to hyperparameter choice. The figures also include the standard deviation of the mean best ensemble accuracy. The standard deviation indicates how closely any given run is from the mean. A large standard deviation indicates a wide margin for runs to perform better or worse than the mean, while a small standard deviation indicates most runs perform similarly.

Generational Ensemble Fraction These results show that a low generational ensemble fraction (0.1 - 0.25) has the highest performance. Additionally, a large generational ensemble fraction (1) is correlated with decreasing performance over time. One interpretation is that by increasing the proportion of the population ensembled, the constituent ensemble fitness of each genome approaches the average population fitness, which limits the selection algorithm's ability to identify useful genomes.

Candidate Limit Surprisingly, either using a small fraction (25%) or all available (100%) possible constituent ensembles yields similar high performance while using most (75%) yields low performance. This may be due to selection bias as trials using a higher fraction would crash due to high run-time unless they had small populations. As such, using a smaller population and higher candidate limit may be equivalent to using a larger population and lower candidate limit.

6.4 Ensembling Algorithm Performance

Lastly, the team ran several hundred trials using the UCI Heart Disease data set to compare ensembles created with the varying ensembling algorithms (i.e., Random, Greedy 1, Greedy 2, and Diversity Based Round Robin), the results of which are viewable in Figure 12. More specifically, the team was looking to see how DBRR compared with the other algorithms in line with the team's hypothesis that creating ensembles of diverse networks would perform better than those just based on individual fitness. Thus, the teams chose a low speciation threshold to increase population diversity and tuned other such parameters to favor DBRR based on the results of the sensitivity analysis in the previous section. These hand-picked parameters were hold constant between each trial to reduce any possible systematic error that could be caused by varying other parameters.

6.4.1 Random

As the team hypothesized, the random (control) algorithm generates ensembles whose fitness' steadily increases as the number of individuals in said ensembles increase, as seen by the *random* series in Figure 12. This finding mimics the real world, in which adding further guesses without taking into account the accuracy of said guesses is more likely to produce a result closer to the mean/actual value. In other words, having a larger sample size is more likely to help predict the true value/mean of a given distribution.

6.4.2 Greedy 1 Vs. Greedy 2

While Greedy 1 performs near-optimally when considering only training data like in Figure 2, it statistically does not outperform Greedy 2 when its ensembles are tested with testing data like in Figure 12, especially when taking into account the perceived error of both algorithms over several hundred trials.

Overfitting The team suggests that Greedy 1's lack of ensemble fitness over Greedy 2's is due to overfitting and a lack of diversity when casting votes. Because Greedy 1 selects the next best network based on how it improves ensemble performance on

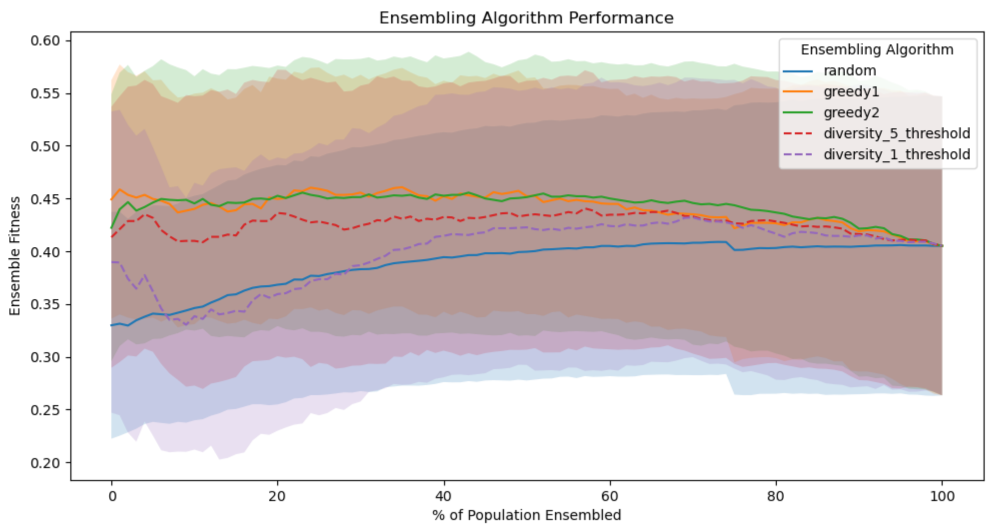


Figure 12: **Determining Optimal Ensemble Algorithm and Size**. Each algorithm was evaluated with an increasing ensemble size. The algorithms converge to the same value, representing the average accuracy of the ensemble containing 100% of the population

given training data, it doesn't consider variance in the real world (and consequently testing data). Instead, Greedy 1 hand-picks ensembles that work well only with the given training data and lack diversity amongst the ensembles' member networks. As such, Greedy 1 has a tendency to over-fit, though limiting the ensemble size did improve performance.

Ensemble Size The team also noticed (in several different configurations) that ensemble fitness either increases or stays consistent until around 50% of the population is ensembled when using Greedy 1 or Greedy 2, as seen in Figure 2 and Figure 12. The team hypothesizes that after this 50% threshold, too many inferior individuals are included in the voting process, resulting in poorer fitness. This observation differs from Random, because Random cannot *just* pick the top 50% of a population; instead, Random works with what it gets, and when sampling a random distribution, you will tend to get more accurate results with the more samples taken. (But Greedy 1/Greedy 2 can selectively see which samples are "worse samples," and exclude them.)

Performance Considerations Seeing as Greedy 2 takes an order of magnitude less time to compute, and Greedy 1 does not provide any statistically relevant gains, the team recommends the use of Greedy 2 over Greedy 1. Greedy 2 is a quick and cheap way to generate ensembles, without the need for any complicated optimizations.

6.4.3 Diversity

While the team hypothesized that picking individuals for ensembles based on their genetic diversity (over just the individuals' fitness alone) would significantly improve the resulting ensemble's fitness, experimental results cannot support this hypothesis.

For a quick reminder, a higher speciation threshold results in fewer species and thus less diversity within ensembles, whereas a lower speciation threshold results in more species and consequently higher diversity within ensembles.

Low Speciation Threshold Low speciation thresholds, such as 1.0 as used in `diversity_1_threshold` in Figure 12, did not work as well as anticipated. In fact, `diversity_1_threshold` was the worst performer of all (non-control) tested algorithms. To explain this finding, the team suggests that at such low speciation thresholds, far too many species are generated, and resulting ensembles do not have enough well-performing individuals in the network to help solve tasks (one cannot just have diversity, one needs diversity and some intelligence). Interestingly, however, picking for diversity alone still does outperform random on average.

High Speciation Threshold High speciation thresholds, such as 5.0 as used in `diversity_5_threshold` in Figure 12, have proven more effective at generating fit ensembles than lower thresholds. The team hypothesizes this is because at higher speciation thresholds, the Diversity algorithm more closely approximates Greedy 2, which itself is a strong performer.

Due to this observation, the team accepts it is unlikely that picking for diversity improves ensembles more than simply picking individuals by their fitness alone, at least for the machine learning tasks the team selected (including the UCI Heart Disease data set).

6.4.4 Algorithmic Recommendations

Given the above findings, the team recommends the use of Greedy 2 at around a 50% population size to create ensembles in most configurations. These suggestions could change easily given differing machine learning tasks but are at least consistent with the team's findings.

6.5 Summary of Results

UCI Heart Disease			
Fitness Heuristic	Ensemble Heuristic	Median Ensemble Size	Average Accuracy
Control	Random	123	0.475 (0.09)
Control	Greedy1	13	0.567 (0.06)
Control	Greedy2	18	0.557 (0.06)
Control	DBRR	40	0.545 (0.19)
CE	Random	22	0.453 (0.12)
CE	Greedy1	6	0.531 (0.11)
CE	Greedy2	6	0.520 (0.13)
CE	DBRR	8	0.501 (0.12)
CE w. Warm-up	Random	25	0.490 (0.11)
CE w. Warm-up	Greedy1	10	0.588 (0.09)
CE w. Warm-up	Greedy2	12	**0.601** (0.09)
CE w. Warm-up	DBRR	31	0.554 (0.10)
Comparison Results from Soule et al. [8]			
Island	Island	5	0.542 (0.03)
Team	Team	5	0.566 (0.03)
Island	Team (OET1)	5	0.564 (0.03)
Team	Island (OET2)	5	**0.568** (0.02)

Table 1: This table indicates the average accuracy of ensembles evolved using the various fitness and ensemble heuristics tested in this paper. It also provides a comparison to results from Soule et al. and their OET methods. The results indicate that *constituent ensemble evaluation (CE) with Warm-Up* using the *Greedy 2* heuristic outperforms OET2

7 Conclusion

This project aimed to evolve ensembles of neural networks using neuro-evolution for augmenting topologies (NEAT). Using a novel fitness heuristic, termed *constituent ensemble evaluation*, individuals in the population are evolved based on the performance of a small sample of ensembles of which they were a part.

In the case of reinforcement learning, there is no conclusive evidence as to which approach yields better performance. The team theorizes that the chosen task, Acrobot, was easily optimized by both algorithms. Incidentally, while it is not the focus of this paper, the modification of the fitness function to use the sum of the height achieved minus the step completed lead to much faster optimization using NEAT than the default reward metric. Nevertheless, while NEAT is generally capable of solving simple reinforcement learning tasks, constituent ensemble evaluation does not improve its capability.

For classification tasks, the team claims that constituent ensemble evaluation is not sufficient alone, but when incrementally introduced into the fitness function (warm-up), yields better overall results than solely using individual fitness. Moreover, this approach can be further improved by evaluating ensembles using the cross-entropy loss rather than accuracy and by lowering the speciation threshold to encourage more diversity in the population. Otherwise, the team recommends that default NEAT parameters are sufficient. Additionally, this paper tested a variety of ensembling algorithms. It was hypothesized that prioritizing species diversity of candidates over candidate fitness might yield better performing ensembles, though this was not demonstrated to be the case. Overall, the straightforward approach of selecting candidates by training fitness (Greedy 2) was generally the best approach. The combination of *Constituent Ensemble Evaluation with Warm-Up* and *Greedy 2* selection even outperform a similar approach to evolving ensembles, *Orthogonal Evolution of Teams.*

7.1 Future Works

7.1.1 Limiting Population Explosion

By far, the largest challenge in working with NEAT was the high potential for a population size to grow rapidly, which slowed down experimentation severely. This is the unfortunate trade of with encouraging diversity in NEAT; as the population becomes more diverse, the number of species allocated to preserve this diversity grows. This was especially prevalent when the speciation threshold was low, as it drastically increased the likelihood that new species would be created, increasing the total population size. For future studies, this problem must be corrected, possibly through the inclusion of a species budget or stricter elimination criteria to encourage consistent purging of more members of a species.

7.1.2 Alteration of Diversity Metric

As NEAT already implemented a diversity metric, the team choose to use this metric to explore the diversity heuristic. However, there are numerous ways to quantify diversity of neural networks, specifically by using the diversity in predictions rather than diversity in structure. Based on the tendency for trials to converge to an overall fitness proportional to the presence of the majority class in the test data, it appears that over time, diverse model structures will begin to represent similar solutions. Using methods such as boosting, which increase the reward for correct predictions on instances that are usually missed, may help encourage better ensembles.

7.1.3 Evolution of Neural Network Layers

Due to the challenges the team faced while attempting to classify the MNIST data set, the team suggests that perhaps larger scale topology evolution could make sense. Instead of evolving individual nodes and edges, perhaps a concept similar to NEAT could be employed but instead on the layers of a neural network to evolve things such as the number of layers, size of each layer, type of each layer (including drop off), etc., and then train each generation through traditional back propagation and let the fittest models' topologies seed the next generation.

While the idea is promising, one must consider that evolving a topology in this manner could easily lead to overfitting, so some measures should be taken to prevent overly optimized topologies for a given input.

References

[1] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[2] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing.* Springer, 2015.

[3] Timmy Manning, Roy Sleator, and Paul Walsh. Naturally selecting solutions. *Bioengineered*, 4, 12 2012.

[4] David Opitz and Jude Shavlik. Generating accurate and diverse members of a neural-network ensemble. In D. Touretzky, M.C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

[5] J.D. Schaffer, D. Whitley, and L.J. Eshelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, 1992.

[6] Terence Soule and Pavankumarreddy Komireddy. *Orthogonal Evolution of Teams: A Class of Algorithms for Evolving Teams with Inversely Correlated Errors*, pages 79–95. Springer US, Boston, MA, 2007.

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

[8] Russell Thomason and Terence Soule. Novel ways of improving cooperation and performance in ensemble classifiers. pages 1708–1715, 07 2007.

[9] L Darrell Whitley et al. *The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best.* Colorado State University, Department of Computer Science, 1989.

[10] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimizationb. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[11] Xin Yao and Yong Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(3):417–425, 1998.

[12] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1):239–263, 2002.