Jackson Dean
Evolutionary Robotics
Final report: NEAT vs. HyperNEAT

I. Introduction

This project investigated two evolutionary algorithms, NEAT and HyperNEAT, and compared their performance on the task of evolving a neural network controller for forward locomotion of a simple quadruped. The primary question that I wanted to answer was: given a fixed computational budget, will NEAT or HyperNEAT evolve neural network controllers that are better able to control a quadruped to move further in one direction over a fixed duration?

This is an interesting question because both NEAT and HyperNEAT have been shown to effectively evolve controllers for locomotion (Mehmet Erkan, 2018; Clune et al., 2009). It would be tempting to assume that, because HyperNEAT is a more advanced and more recent algorithm, it will perform better than NEAT. In fact, Stanley, Ambrosio, and Gauci make a similar comparison and find that to be the case in the original HyperNEAT paper (2009). However, this project hoped to investigate a simple task and a limited computational budget, which HyperNEAT might be "overkill" for.

For the four deliverables, my strategy was to implement versions of both algorithms and verify that they worked by looking at best fitness over time and comparing the behavior of an evolved robot to random behavior. After verifying the two algorithms, the next step was to optimize them both to allow for a fair comparison. Finally, I conducted the A/B test to determine which algorithm evolved the furthest forward locomotion after exhausting a fixed computational budget.

II. Methods

For the first deliverable, I modified the codebase to allow for evolving the structure of the neural network using NEAT. I used the procedure described in Stanley and Miikulainen (2002). The most significant hurdle was allowing for both the topology and the weights of the NNDF file to change between generations. The NEAT algorithm also has some details that needed to be added, such as speciation to maintain genetic diversity, and crossover between neural networks. I took advantage of OOP principles to write this code, so that I could reuse as much as possible for HyperNEAT.

My second deliverable was an implementation of HyperNEAT according to Stanley, Ambrosio, and Gauci (2009). I was able to reuse the code from NEAT for evolving controllers to evolve the Compositional Pattern Producing Networks (CPPNs) of HyperNEAT. The main difference with the CPPN compared to the ANN controller is the variety of activation functions that a given node in the CPPN can use. Inputs are interpreted as coordinates of controller connection midpoints, and the CPPN's outputs are weights of the neural network controller, instead of directly controlling motors as in NEAT. I experimented with different types of substrates and found the "sandwich" substrate to be the most effective.

The third deliverable was a robust system of data recording and visualization. Since I was using more complex evolutionary algorithms, I wanted to have the ability to analyze the evolutionary process more thoroughly. I added the ability to record and visualize a number of metrics including: population diversity, speciation, number of genotype connections, number of genotype nodes, number of generations to converge, and fitness of the best individual. See the appendix for these additional graphs. I also created tools for visualizing and exploring the weights produced by HyperNEAT. Finally, I added functionality to record and show the footprint graph of the quadruped.

For the fourth deliverable, I prepared my A/B test. Because there are many hyperparameters in these two evolutionary algorithms, I ran several preliminary experiments before deciding on settings to use for the final experiment. I wanted to ensure that the set of parameters that I used for my A/B test did not bias the results towards either algorithm.

Both evolutionary algorithms were run with a population size of 20 for 100 generations. Each algorithm had a target of 3 species. Simulations ran for 800 time steps. The weight of the torso was increased to 4 units, while all other limbs remained at 1 unit. There were 4 touch sensor neurons (one on each foot), and 12 motor neurons allowing for 12 degrees of freedom. There were two degrees of freedom at each connection between the torso and upper leg, and one degree of freedom between each upper leg and lower leg. Motors had a maximum force of 50 units. Weights within the controller ranged from -5 to 5 and weights within the CPPN ranged from -1 to 1. CPPN activation functions included: hyperbolic tangent, identity, sine, gaussian, and sigmoid. Controller networks used only hyperbolic tangent. Fitness was measured as distance from the origin in the negative x direction averaged with distance in the y direction.

III. Results

Evolution occurred in both the NEAT and HyperNEAT cases. Figure 1. shows the best fitness from the population at each generation, averaged across 30 experimental trials. We see an improvement from generation 0 (random individuals) to generation 100, demonstrating that evolution is occurring.

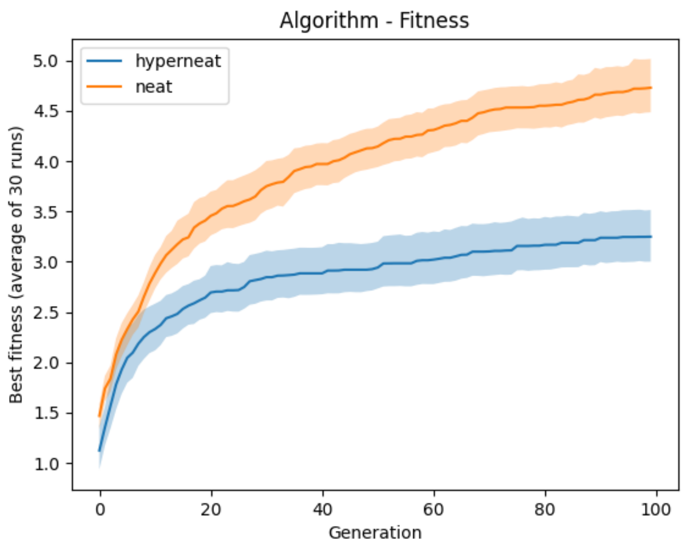


Figure 1. Fitness of the most fit individual each generation averaged across 30 trials with bootstrapped confidence intervals.

In this experiment, NEAT was shown to produce more fit individuals than HyperNEAT given these parameters and computational budget. We cannot make conclusions about what would happen given a larger population size or more generations.

Both algorithms evolved a type of pronking gait, however NEAT was able to evolve a much more periodic and reliable gait. This is demonstrated by the footprint graphs below in figure 2. We can see that, although there is symmetry and periodicity in the HyperNEAT gait, the feet drag at times, especially the front and left legs, unlike NEAT. Both algorithms show a more viable gait than the random network.

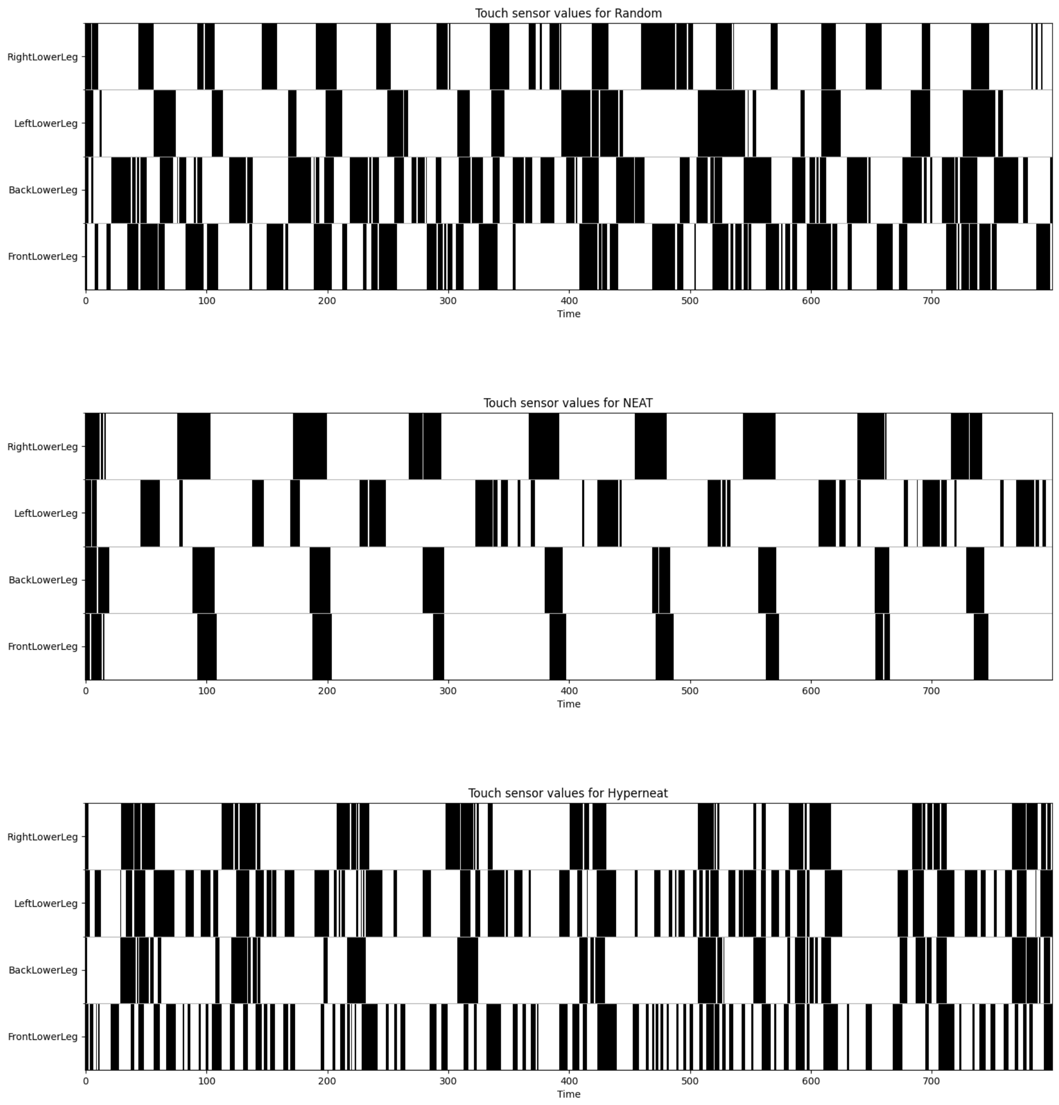


Figure 2. Footprint diagrams for a random controller (top) the best robot evolved with NEAT (middle) and HyperNEAT (bottom). NEAT evolved more symmetrical, periodic behavior.

IV. Conclusion

The results from this A/B test were surprising to me. I expected HyperNEAT to be able to evolve better controllers than was shown. I suspect that, with a small population and few generations, it was more effective to evolve the neural networks directly via NEAT rather than adding an extra layer of abstraction via the CPPN.

It was surprisingly easy to implement HyperNEAT once I had NEAT working. I expected a larger difference in implementation between the two algorithms, but since the core evolutionary process is the same, I was able to reuse the bulk of the code. It was surprisingly difficult, on the other hand, to find good hyperparameters for an A/B test. Since the goal was to compare the two algorithms, I wanted to keep as many parameters equal between the two algorithms as possible, but found this to be an unfair comparison in some cases. For example, changing the maximum possible weight of the genotype network had very different effects on behavior depending on the algorithm, since, in NEAT, those weights get applied directly to sensor values but, in HyperNEAT, they are used to calculate the weights of the controller network, which are then applied to sensor values. HyperNEAT also has parameters that do not exist in NEAT. For hyperparameters that could not be equal between algorithms, I decided to run separate experiments to find values that resulted in the highest fitness within variants before comparing between variants. I realize that this could introduce bias into the experiment, but in my opinion, resulted in a more meaningful comparison between the algorithms.

For future experiments, I would like to investigate these algorithms with a much larger computational budget and on more sophisticated tasks. In order to achieve this, it would be helpful to allow Pyrosim to evaluate neural network controllers on the GPU. One benefit of HyperNEAT that I was not able to explore is its scalability to large neural networks. I would be interested to see how these two algorithms compare if we were to allow HyperNEAT a much larger phenotype network size, which may be impractical without GPU parallelism. HyperNEAT also has the added benefit of having genetic representations isomorphic with the layout of the robot. Therefore, it would be interesting to investigate different substrate configurations in more depth. While it would be easy to scale up these experiments, writing efficient code that can run on a large population of large networks for many generations would be a challenge.

References

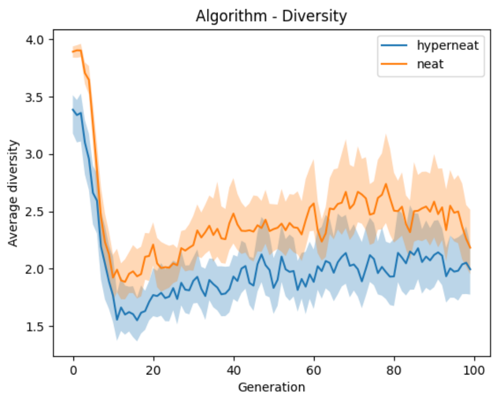
J. Clune, B. E. Beckmann, C. Ofria and R. T. Pennock, "Evolving coordinated quadruped gaits with the HyperNEAT generative encoding," *2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 2764-2771, doi: 10.1109/CEC.2009.4983289.

K. O. Stanley, D. B. D'Ambrosio and J. Gauci, "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks," in *Artificial Life*, vol. 15, no. 2, pp. 185-212, April 2009, doi: 10.1162/artl.2009.15.2.15202.

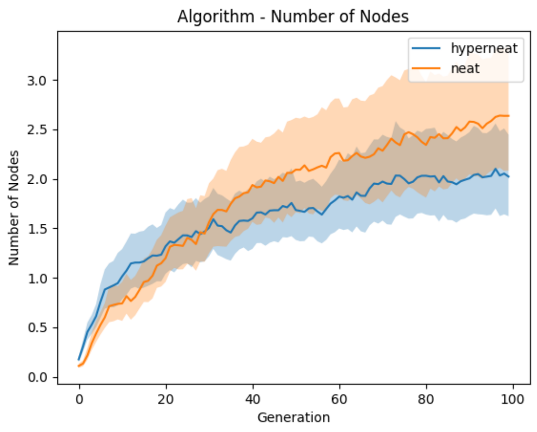
K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," in *Evolutionary Computation*, vol. 10, no. 2, pp. 99-127, June 2002, doi: 10.1162/106365602320169811.

Yuksel, Mehmet Erkan. “Agent-Based Evacuation Modeling with Multiple Exits Using Neuroevolution of Augmenting Topologies.” Advanced Engineering Informatics, vol. 35, 2018, pp. 30–55., https://doi.org/10.1016/j.aei.2017.11.003.

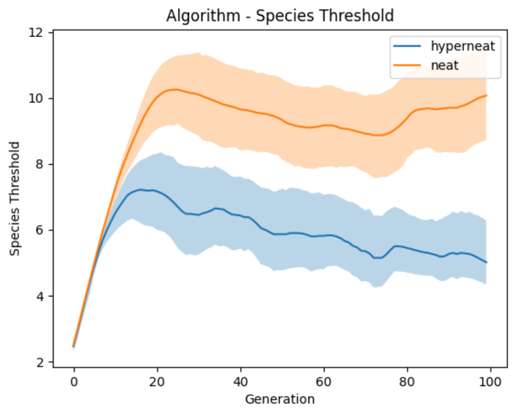
Appendix



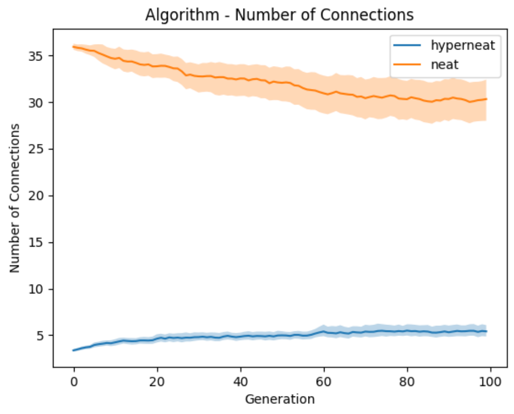
Population genetic diversity averaged over 30 runs. Measured as the average genetic difference between individuals.



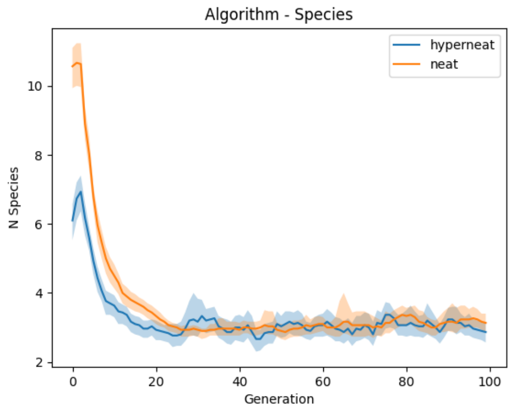
Average number of genotype hidden nodes per individual in the population, averaged over 30 runs.



Average threshold for speciation over 30 runs. A higher threshold means it is more likely that two genetically different individuals will be assigned the same species.



Average number of genotype connections per individual in the population, averaged over 30 runs. HyperNEAT has much less because there are fewer input (x1, y1, x2, y2) nodes and output (weight) nodes than in NEAT (4 input, 12 output), resulting in fewer possible connections.



Average number of species in the population over 30 runs. Both algorithms quickly converge to the target of 3.