NEAT Algorithm

***N**euro**E**volution of **A**ugmenting **T**opologies*

What is **NEAT**?

- **NEAT** is a combination of the concepts of Genetic algorithms and Neural Networks (NN).
- It attempts to “grow” an NN from a simple start through random changes

Link to the NEAT Paper

Why did I pick **NEAT**?

- We only touched upon Genetic Algorithms during class, and I wanted to see how they worked. When I found out about **NEAT** I wanted to play around with it to see how effective it was.

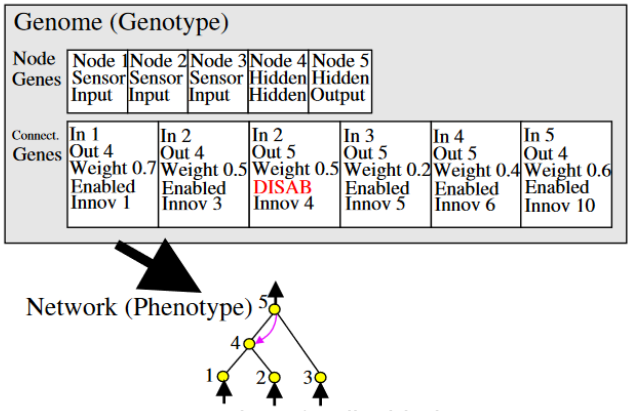
What was my experience with **NEAT?**

- **NEAT** was *incredibly* sensitive to the fitness equation, even more than the Min-Max was. A tremendous amount of time was spent just tuning it until it aligned with my intentions.
- Designing the inputs to the NN required a few iterations before I found one that trained quickly enough
- Simple tasks (e.g. eating food) were easy and quick to train. But advanced tasks greatly increased the training time with each additional level of complexity.
- Overall, designing an implementation for a competitive snake game was slow and difficult.

What implementation of **NEAT** did I go with**?**

- These results were made with NEAT-Python, which is built on the PyTorch framework.

How Does NEAT Work?



Demonstration of a disabled gene
Image taken from the NEAT paper

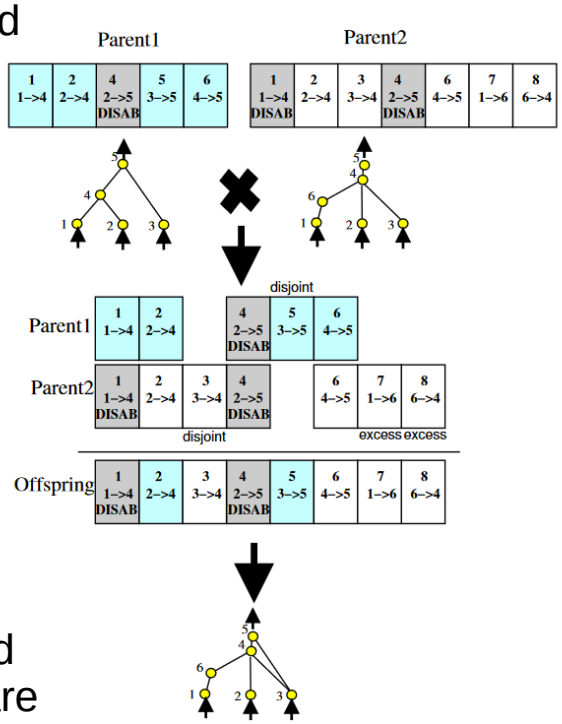
The **NEAT** algorithm builds a structure referred to as a “genome,” which has **node genes** and **connection genes.**

The **node genes** represent the individual neurons, and the **connection genes** contain the connecting edges and activation functions for the **node genes.** They also contain additional information that **NEAT** uses to work.

A fitness evaluation is checked for each generation, and more fit models are increased in population, less fit are reduced, and Unchanging “stagnant” models are eliminated.

Then the settings such as weights, connections, or even the activation functions are “mutated” randomly by **NEAT**.

Besides the mutations the structure can also change by mimicking sexual reproduction. Two genomes can have their genes combined in order to synthesize a new structure.



Combining two genomes
Image taken from` the NEAT paper

In the past, the differing topologies that occurred from random changes made it difficult to compare the fitness of two different neural nets or “mate” them together, since this would only work if they Were similar enough in structure. **NEAT** handles this by giving each new connection gene an “Innovation” number stored in the node gene.

This can be used to track ancestry of genes, which implies similar structures that can be used for comparison and combination. It also permits us to break genomes down into “species,” which lets a new mutation (which initially lowers fitness) have a chance to flourish without having to compete with higher fitness species already dominating the environment.

How Long To Learn Features?

With **NEAT** we lack direct control of designing the NN. This means how long a **NEAT** structure takes to learn is a little bit random.
Here's some rough guidelines of how many generations it took for our implementation to learn certain topics.
Note: Each generation is 800 individuals

300 Generations (**0.83 Hours**) – Navigates to food directly, but dies heading on to the next food, usually by doubling back on itself.

500 Generations (**1.3 Hours**) – Can eat food without killing itself, but if food doesn't appear instantly it simply runs into walls or players.

1200 generations (**3 Hours**) – Learns walls are fatal, most of the time.

8000 generations (**22 Hours**) – Learns to spin in place when no food so that it doesn't die when waiting for food to appear

On average, it took 10-15 seconds to process a generation.

GPU Acceleration for **NEAT** wasn't well developed until as recently as April 2nd, 2024, when **TensorNeat** was published. **TensorNeat** uses the JAX framework

Link to **TensorNEAT** Paper