NEAT AIRO BROS 3. MACHINE LEARNING AGENT PLAYS NES GAMES.

ABSTRACT

Neural Evolution of Augmenting Topologies (*NEAT*) is a genetic algorithm for neural networks with a high focus in retaining innovations during reproduction between two genomes.

Super Mario Bros. 3 is a side-scrolling action platformer developed for the Nintendo Entertainment System. Most levels in the game require the player to navigate Mario from the left side of the screen to the right side. Along the way, the player must face various enemies and bottomless pits.

The goal of this project was to create and train a NEAT network to learn to play a single level in Super Mario Bros. 3.

An emulator (BizHawk) was used to run the game, while the network was written as a Lua script for the emulator.

The neural network showed decent, although not satisfactory results. That aside, for tasks with this much temporal and contextual complexity, we believe that there exist more suited neural networks out there.

INTRODUCTION

An **artificial neural network** (hereinafter: neural network) is a mathematical system of connected nodes called **neurons** used to solve artificial intelligence problems.
A **genetic algorithm** is an evolutionary optimization algorithm, where the optimal solution is found by maximizing (or minimizing) a heuristic function called the **fitness** among multiple generations of individuals.

Neural evolution is a family of genetic algorithms where individuals are encoded as neural networks.
We hypothesize that a neural evolution could be utilized to complete a single level of Super Mario Bros. 3. Indeed, similar feats have been accomplished before [1].

NEAT

NEAT is a genetic algorithm which encodes its individuals using neural networks. The idea is for the network to start out as simple as possible, and to evolve its *topology* through trial and error.
An individual, or **genotype** in NEAT is a list of **genes**, connecting **neurons** (one-way). The **phenotype** of the individual is the actual neural network. An individual then provides a solution to the problem whose **fitness** is calculated is compared against other individuals. Once enough individuals in a **generation** provide a solution, the **reproduction phase** begins.

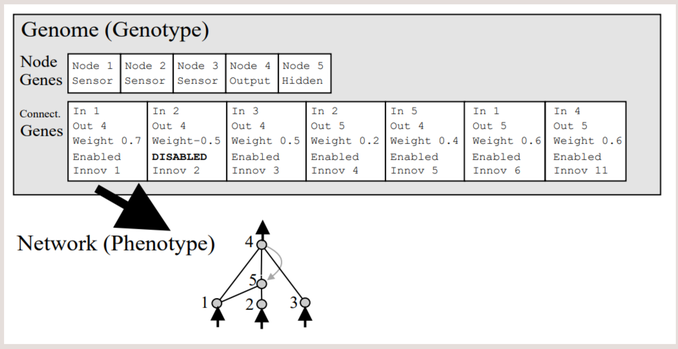

Image 1. Genotype and phenotype in NEAT.

Individuals with a higher fitness have a higher chance of reproducing. Reproduction includes **crossing-over**, **mutation** as well as culling weak genotypes for the next generation.

Between generations, the network of an individual becomes increasingly complex. To keep track of the evolution, an **innovation number** is introduced. The innovation number works like a unique identifier for genes, incremented each time a new gene is created through mutation.

This is important because genetic algorithms perform crossing over between corresponding genes. A neural network is not linear - naively crossing adjacent genes in two different networks may not produce good results: certain *innovations* may disappear.

With innovation numbers, we match up genes with the exact same innovation number – it is guaranteed that two same genes from different individual have the same innovation number.

If a gene with some innovation number is not present in both individuals during breeding, it is called **disjoint** or **excess**, depending on whether its innovation number is greater than all innovation numbers of the other individual. These genes are retained from the more fit individual. All other genes are called **matching** genes, and they are picked randomly between the two individuals.

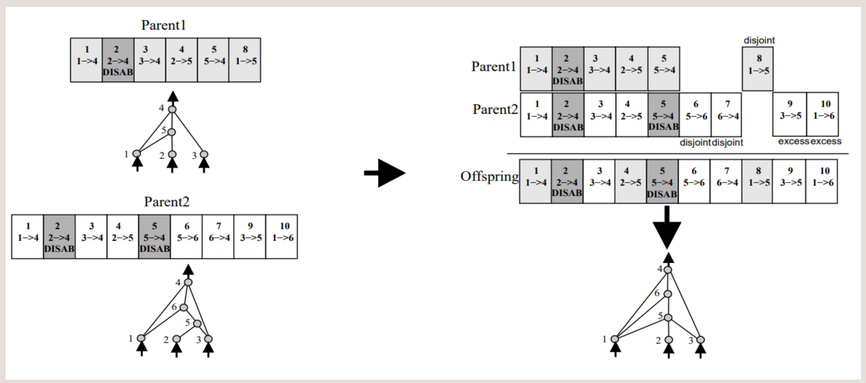

Image 2. Crossing over in NEAT.

After breeding, the child undergoes mutation.

There are several ways to mutate a neural network:

1. ***Weight mutation*** – randomly modifying the weight of a single gene.
2. ***Node mutation*** – add a new gene between two disconnected neurons.
3. ***Gene mutation*** – add a node inside an existing gene.
4. ***Activation mutation*** – enable or disable a gene.

Note that both the weight and the structure of the network are eligible for mutation. This is what allows for topology augmentation.

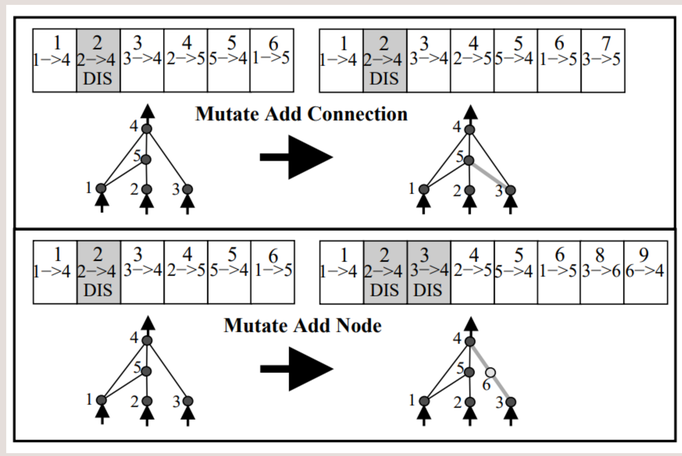

Image 3. Node mutation (top) and gene mutation (bottom).

An individual may produce desired mutations yet never have the chance to spread its genes because of a dissatisfactory mutation with an individual that's too disjoint from it. NEAT solves this by introducing **speciation**. The idea is to have individuals in a population compete within their niche, called a **species**. Innovation numbers are used for this task.

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

The compatibility distance is a sum of disjoint, excess and matching genes. The coefficients are computed empirically. *N* is the number of genes of the larger individual. We use this formula along with a *representative* to create a species. To prevent a single individual from dominating the species, its fitness is scaled.

RESULTS & DISCUSSION

We have implemented a NEAT agent in the Lua scripting language with a task to complete a single level of Super Mario Bros. 3 for the NES. The script runs in the BizHawk emulator.
Input neurons are a grid of NxN cells from the player's current position. The grid can take the values 0 (empty space), 1 (solid block) or -1 (enemy). A bias node is also added. **Output neurons** are buttons. The **fitness function** is player_x_position – elapsed_time/2. **tanh** is the activation function.

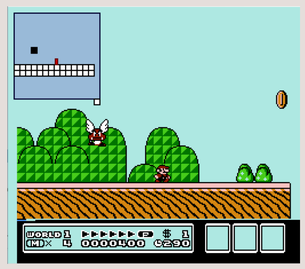

Image 4. Input nodes in the top-left corner.

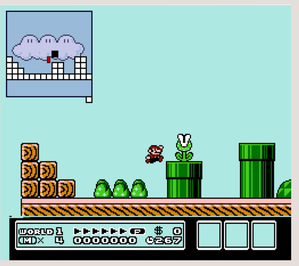

Image 5. This is as far as the agent made it.

We were unable to train the agent so as to complete a full level of the game. The training procedure lasted for ~20 hours. A total of 421 generations have evolved. The best performing generation on average was between 100 and 200. After that, little improvement was found. In fact, the agent showed weaker performance from that point on.

Fine-tuning the various parameters was a difficult task (it took 1-2 hours of training before an evaluation could be made).
Over-fitting was another problem, which is why the agent alternates between 2 spots in the level. Our agent was most likely stuck in a **local optimum.**

We have performed a smaller test on a different level, with similar results. Ultimately, complex tasks like playing video games showcase a lot of non-linearity, which NEAT does not handle well. A deep neural network is more suited for these problems.

REFERENCES

1. MarI/O - Machine Learning for Video Games - YouTube
2. stanley.ec02.pdf (utexas.edu)