Using NEAT Algorithms to Play Retro Games*

Department of Electrical and Computer Engineering
North Carolina A&T State University
Greensboro, NC, USA
dlcummings@aggies.ncat.edu,
tsallen8@aggies.ncat.edu

Department of Electrical and Computer Engineering
North Carolina A&T State University
Greensboro, NC, USA
mjsmith11@aggies.ncat.edu,
dwebb2@aggies.ncat.edu,

***Abstract* - With an increased interest in Artificial Intelligence (AI) and Machine Learning (ML), delving into possible applications and scenarios in which these fields are applicable is commonplace. Our project uses an algorithm (Neuroevolution of Augmenting Topologies or NEAT) to investigate its ability to learn and play retro games (Galaga on the GameBoy). An emulator (PyBoy) is used to emulate the game of choice and a model is constructed through Python. The result is a model capable of playing the game on its own. This result shouldn't differ from other compatible games: the only thing that needs changing is the fitness function of the game.**

I. INTRODUCTION

Genetic Algorithms are search algorithms based on how nature evolves.[2] It creates a set population with random features and bases its survival on how these features perform in a black-boxed fitness function. The better a species does, then the more it populates the population. On the contrary, the worse a species does, the more it dies out. The randomness of the system is what makes this work. Species mutate, crossover, and repopulate solely on how well it does and the probability for these random changes to occur.

But what if we take this same concept and use it to determine the performance of neural networks? This is the idea of the NeuroEvolution of Augmenting Topologies, or NEAT for short. The main concept is the same: We take a population and test its performance on a black box

The difference is what the features are. Instead of a set of characters used to determine an individual's fitness, we use a neural network instead. The performance of the neural network, still based on a black-boxed fitness function, determines the survival of a species.

II. PROBLEM FORMULATION

A way to show the capabilities of AI is through the use of video games. Games in themselves have a complexity that makes them fun; knowing what to do at the right time, taking quick calculation of the scenario in front of the player, and translating it to button presses on the controller. We can use this complexity to showcase how well NEAT can overcome these challenges and perform the calculations and actions needed to play these games well.

For this project, we use two well-known retro games:

- Galaga: Galaga is an arcade shooter made by Namco and Midway you control a spaceship tasked with shooting down a squad of aliens. The aliens have a setup phase and an attack phase. In the setup phase, the aliens will fly onto the screen where they will idle. In the attack phase, the aliens will fly down to attack you, and then return to their idle area. You are tasked with shooting down enemy aliens, moving out of the way of incoming aliens, and progressing through as many levels as possible.

* This research work is done as a class project for Genetic Algorithms

- Super Mario Land: Super Mario Land is a 2D-platformer on the Game Boy. The goal of the game is to progress through the different levels by defeating enemies and gaining power-ups. The game gets harder as it progresses, with more enemies to defeat or more obstacles to go across.

III. METHODOLOGY

We used certain modules to complete this project:

- PyBoy: A Python Game Boy Emulator. You can use python to control the inputs in the game.[4]
- Neat-python: A python library that handles the implementation of NEAT. You can create a calculate_fitness function, as well as a configuration file that handles certain parameters of the algorithm.[5]

We created a neural network template that takes in the game screen and outputs the buttons that would be pressed in that frame:

Galaga Model:

- Input Layer: 32x32 (1024) pixel game area
- Output Layer: 3 neuron button inputs (left, right, shoot)

Super Mario Land:

- Input Layer 16x20 (320) pixel game area
- Output Layer: 4 neuron button inputs(left, right, jump, dash)

The hidden network is created through the NEAT algorithm.

We also determined a fitness function that will determine a neural network's effectiveness as it is trained:

- Galaga: Time, accuracy and score
- Super Mario Land: distance and jump count

IV. LITERATURE REVIEW

NEAT, which stands for Neural-Evolution of Augmenting Topologies, is a system that evolves from Topology and Weight Evolving Artificial Neural Networks (TWEANNs)[2]. In the paper "Efficient Evolution of Neural Network Topologies", Kenneth O. Stanley addresses three challenges of TWEANNs: a proper representation for networks to properly crossover, a way to protect innovations that need some time to properly optimize, and a way to minimize topologies without having to incorporate it when designing a fitness function. NEAT does this with four design improvements.

Genetic encodings - The genome of a network consists of a list of connections that specify how it's connected. Each gene specifies an in-node, an out-node, the connection weight, whether or not a connection is incorporated in the network, and an innovation number. When a new connection or node is created, a new gene is created on this list, disabling previous genes in the case of adding nodes.

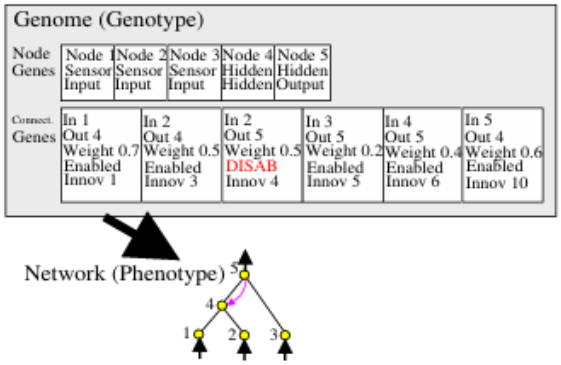

Fig. 1. Example of genome in population

Historical markings - The innovation number in a genome is a global number, meaning that whenever a new gene is created, regardless of the network that did it. This allows for useful comparison of two networks: you can know the differences and similarities between two networks by comparing their innovation numbers. This makes crossover very simple: a child get all of the similar qualities of the parents, but gets the excess of the better parent (the child gets random excess genes from both parents if they are equal in strength)

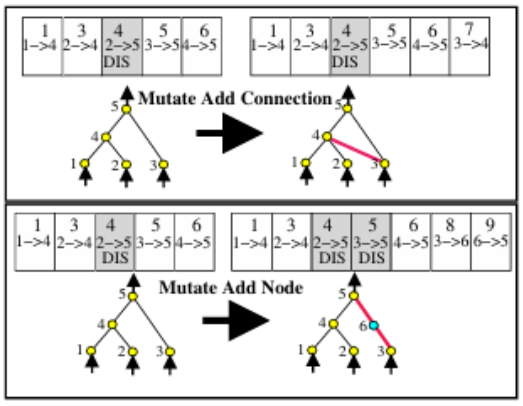

Fig. 2. Instance of mutation with a genome. Genes are added to a list of genes as they mutate

Speciation - Networks under NEAT compete mostly with other similar networks, rather than with the population entirely. This allows for individuals to optimize themselves before competing with other optimized innovations. Historical markings make this trivial as well: we can calculate a compatibility distance using the amount of access and disjoint genes, as well as the average weight differences between two networks. Networks of the same species also share fitnesses as a whole. The larger a species' fitness, the more of that species you will see in the entire population. Likewise, the larger the fitness of one network in a species, the move of that network will populate in that species.

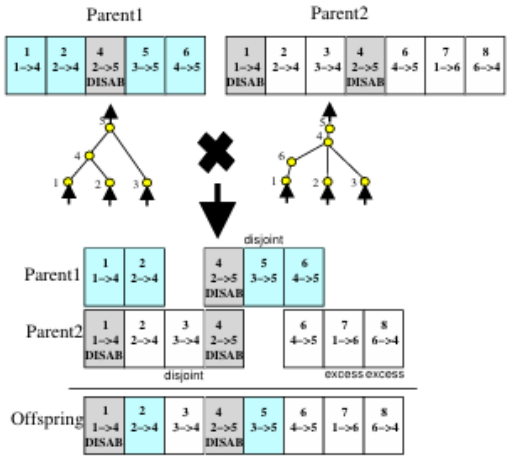

Fig. 3. Crossover between parents. Shows use of historical markings to make crossover easier

Minimizing Dimensionality - NEAT has an initial population of just inputs and outputs. There are no hidden nodes in these initial networks, but as the algorithm progresses, so will the hidden layer. This way, we can create solutions that have minimal structures. This is possible because of previous improvements. Because speciation protects innovation, NEAT can start with no structure, as opposed to TWEANNs, which need that initial random structure.

X. VIDEO GAME NEAT RESULTS

For Galaga, our resulting model showed promise in its performance. After 300 generations and a population size of 100, the robot proved capable of understanding what to do in certain scenarios.

For incoming enemies, the model learned to dodge the enemies in its path in order to stay alive. The model also learned that an optimal answer for most of the game is to stand in the corner and shoot at enemies it can reach. However the progress of the algorithm stagnated. This is possible due to the exploitation of our fitness function. By allowing models to score more points via how long it takes to survive, it then created models that excelled at surviving, but not scoring. This is especially true when looking at the gameplay of the strongest model. It can survive forever as long as it can handle all of the enemies on the left side and latches on the left corner. The enemies never reach that side of the corner and the player never leaves. This can be fixed by tweaking the fitness function of the algorithm. Examining what changes can be made to improve this performance make a great future project.

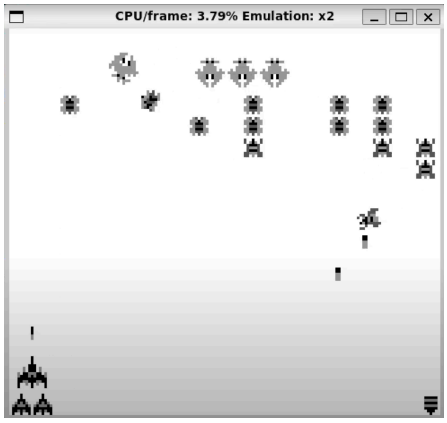

Fig. 4. Instance of NEAT-created Galaga bot gameplay. It shoots at enemies in its corner, but stays there instead of shooting other enemies

In our design for a NEAT implementation to play Super Mario Land, we aimed to create a model that could learn to play the first level as the minimum viable product. A 2D platformer with the main objective being to go from left to right, our main variable for our fitness function was the distance traveled left. Due to the GameBoy's hardware limitations, this value had to be calculated through continuous register polling throughout the model's run rather than a single register read upon the model's death.

With our initial attempt using a population size of 100 and running for 1000 generations, the conjunction of the simple fitness function and the relative simplicity of gameplay corresponded to model evolution far more rapid than Galaga initially. However, the algorithm seemed to suffer from terrible stagnation issues, with no improvements seen over hundreds of generations. To combat this, we tweaked the population size and number of generations in search of an optimal algorithm. Decreasing the population size from 100 to 50 or even 20 yielded significantly faster evolution in the early stages as a lower number of models corresponded to a shorter generation length thus a higher rate of genetic operations. The significant downside to this approach became evident over the NEAT algorithm's runtime; unlike Galaga, where patterns in the game area remain fairly constant, Super Mario Land constantly presents unique game area images to an ever-growing model as it traverses the level. This results in a seeming logarithmic decrease in numeric distance between model improvements the further the model progresses through the level. Combined with the low variation inherent to a small population size such as 20, the algorithm could stagnate for 10,000 generations or more.

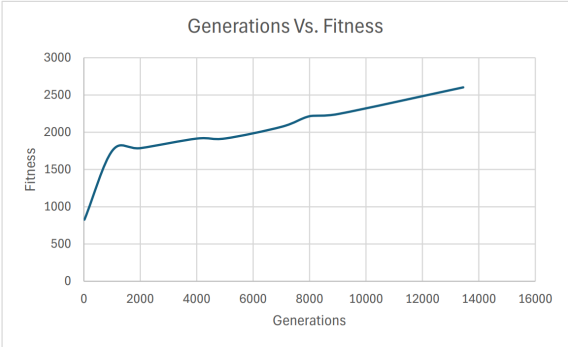

Fig. 4. Logarithmic decrease of fitness to generation efficiency over time

Realizing the need for more variation, the population size was tweaked until we settled upon a size of 400, providing a balance of fair variation and modest generation runtime. Albeit displaying the logarithmic efficiency decrease issue, this model performed very well, learning to jump over enemies after 21 generations, reaching the level halfway point by the 1,500th generation, and finally completing the level by the 13,000th generation.

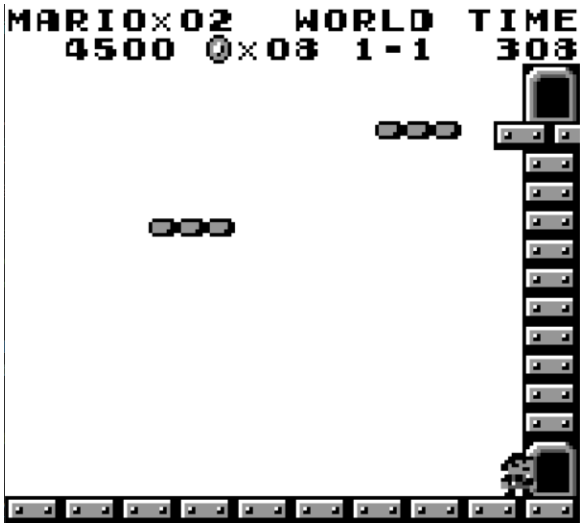

Fig. 5. Model at end of level (level completion)

Although improvements could likely be made to our NEAT implementation, we were able to successfully "teach" neural network models to play videogames through genetic processes.

VI. REFERENCES

1. *Stanley, Kenneth O, & Miikkulainen, R. (2002). Efficient evolution of neural network topologies. Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600). https://doi.org/10.1109/cec.2002.1004508*
2. *Goldberg, D. E. (2013). Genetic algorithms in search, optimization, and machine learning. Pearson.*
3. *SethBling. (2015, June 13). Mari/O - Machine Learning for Video games. YouTube. https://www.youtube.com/watch?v=qv6UVOQ0F44*
4. *PyBoy Github: Baekalfen/PyBoy: Game Boy emulator written in Python (github.com)*
5. *McIntyre, A., Kallada, M., Miguel, C. G., Feher de Silva, C., & Netto, M. L. neat-python [Computer software] CodeReclaimers/neat-python: Python implementation of the NEAT neuroevolution algorithm (github.com)*