Genetic Algorithm Learning and Game Adaptation (GALAGA)

Tate Sturm

December 2024

Abstract

Arcade games have long been a staple in household entertainment. One such game, Galaga for the NES, has been a lifelong favorite. This project aims to develop an algorithm capable of both learning and mastering Galaga using a genetic neural network and the neuroevolution of augmented topologies (NEAT) framework. A genetic neural network imitates the natural process of evolution through mutations in both its structure and weight between nodes, and NEAT looks to improve the evolutionary process with speciation and preservation of essential structures.

1 The Problem

1.1 Motivation

When selecting a topic for this final paper, the initial thought of using genetic algorithms to play classic arcade cabinet or old-school console games came to mind. Genetic algorithms, especially those that can best humans, have always been of peculiar interest. Previously in 2015, a Youtube creator by the name SethBling uploaded a video displaying their approach to applying machine learning to video games [7]. Specifically, they utilized a recently developed framework, NEAT, to teach a genetic neural network to learn to play Super Mario Bros for the NES. After finding this video again in 2024, the idea to adapt their code to play another arcade classic of particular fondness, Galaga, came to mind.

1.2 Proposal

To approach this problem, two programming components must be written or adapted; one for reading game data directly from memory to act as inputs for the

genetic algorithm, and the other being the genetic algorithm itself. An arcade emulator with Lua scripting, BizHawk in particular, will be used to run a Galaga ROM to act as a training ground and house the algorithm as it learns. Collecting input information will involve decoding where Galaga places its vital information like lives left, score, enemy ship coordinates, and more, in memory and reading it as the game runs and reacts to the network's outputs. This information must then be sorted and refined into simplified data snapshots in order to shorten the overall training process and reduce noise. Finally, the genetic algorithm itself will be largely based off of the NeuroEvolution of Augmenting Topologies framework, allowing for even faster learning.

1.3 NEAT

As written in Stanley and Miikkulainen's paper, NEAT is described as an evolving neural network that employs three strategies to outperform others: principled crossover between different topologies, speciation, and incremental growth from a minimal initial structure [10]. By starting from the essential structure of a neural network and allowing mutations to only add simple changes, dimensionality can be reduced, helping reduce vestigial framework and improve performance. Additionally, a more meaningful breeding system that prioritizes newer genes from the more fit parent can be implemented by applying historical genetic markers to each mutation. Lastly, to protect differing core structures in competing high-fitness networks, speciation can be introduced to prevent breeding between two networks that have vast structural differences [2]. These core ideas can then be applied to the presented problem of playing Galaga. By representing a game state as an NxM matrix occupied by enemies, projectiles, or the player, and letting neurons read elements within the matrix and output signals resulting in button presses, an evolving neural network is created.

1.4 Expectations

Similar to the solution given to Super Mario Bros, the neural network is likely to start off at a slow and uneventful pace; learning simple mechanics like firing its weapon and moving around the screen. Due to the constant need to improve in order to raise its fitness to be selected for breeding, these simple outputs must give way to more complicated tasks. Common principles such as self-preservation through dodging enemies and intentional aim to save time are the next step in adaptation. Once these core mechanisms are in place, a final tier of possibly unexpected results may arise. Two prominent strategies in Galaga, fighter capture and entrance pattern anticipation, are distinct possibilities for final improvements. Given enough time, a genetic neural network can be expected to not only play at the level of the average user, but exceed it with niche and possibly undiscovered strategies.

2 Previous Work

2.1 Neural Networks

Neural networks were first theorized in 1943 by Warren S. McCulloch and Walter Pitts who proposed a mathematical representation of the brain using nodes and weighted edges between them. By showing that neural activity is binary, they were able to formulate a mathematical model where any net of nodes forms a graph that can be treated as a complicated logical statement [5]. Over 40 years later, the next breakthrough in neural networks appeared with the idea of backward propagation, allowing for the training of neural networks using computers. Backward propagation brought forth the introduction of hidden variables that could be extracted and refined by the algorithm to improve results output by the network. Although mathematically intensive, back propagation can be summarized as the adjustment of edge weights using gradient descent and its associated activation function, leading to a minimization of error over time [6]. While this approach allowed for the initial training of neural networks, the activation function it relies on, sigmoid, fails when used on deep neural networks with many layers of hidden variables. In recent years, the introduction of the 'Rectified Linear Unit' activation function, or ReLU, its successor 'Leaky Rectified Linear Unit', or LReLU, [1] and general improvements to graphics processing units have fostered new and modern takes on neural networks.

2.2 Genetic Neural Networks and NEAT

Unlike their mathematics-based predecessors, genetic neural networks opt to train and evolve using a genetic algorithm. Rather than minimizing an error function, nodes and connections are randomly mutated and selected for breeding based off of a fitness measure. This crossbreeding of the most fit networks in a generation often leads to more fit children, thus propagating an increasing fitness score until a goal is reached. Genetic algorithms also provide the ability to train a neural network no matter how its neurons are connected, allowing for the training of 'general' neural networks which utilize a mix of connection types [4]. However, basic genetic neural networks often face challenges when it comes to computational expense, scalability, and premature convergence. In 2002, a framework for genetic neural networks known as NEAT was conceptualized to help solve these challenges. Neuroevolution of Augmenting Topologies, or NEAT, looks to address these previous pitfalls through the introduction of three concepts: speciation, minimal initial structure, and interbreeding of different topologies [10].

2.3 NEAT Applications

Since its introduction in 2002, the NEAT framework has led to a wide variety of specialized successors. Some of these successors, often denoted by their NEAT suffix, look to both expand the capabilities of the framework into a real-

world and real-time setting or apply NEAT to a specialized setting like content generation.

2.3.1 cgNEAT

One such application of NEAT that has been previously explored is the idea of continual content generation in online video games to promote prolonged engagement. In 2009, Kenneth O. Stanley, Ratan K. Guh, and Erin Jonathan Hastings created a game Galactic Arms Race with the intention of testing and implementing content generated by their new NEAT framework, cgNEAT. In Galactic Arms Race, players are able to outfit their ships with unique weapons that vary in size, attack pattern, color, and damage. Initially, a few pre-generated weapons were created with the plan to allow cgNEAT to create new options based off of their respective popularity with players. The weapon features that were found to be most popular were then given a higher chance of appearing in the spawning pool, theoretically fitting the content to what players wanted most. After two months of data collection on a population of over 1000 players, a pattern in the evolution of weapons popular in player versus environment and weapons popular in player versus player began to appear. In single player and play versus environment scenarios ascetically pleasing weapon features, like color and visual appeal, took precedence over other components. In contrast, player versus player matches often led to a few key weapon types, such as the shield, goop, vortex, and fork. Shield weapons generated a field of particles around a player's ship, while goop left long-lasting particle blobs to act as area denial. Since initial weapons given to players simply shot in a straight line, these specialized features demonstrate cgNEAT's ability to generate useful and desired content without any human input from the developers' end [3].

2.3.2 rtNEAT

Another implementation of NEAT came in the form of real-time adaptation of video games' characters to prevent the exploitation of scripted artificial intelligence. Like cgNEAT, a custom game named Neuroevolving Robotic Operatives was created to facilitate a new framework real time NEAT, or rtNEAT, and test its ability to adapt agents in real time in response to large agent versus agent battles. To start off, players have the ability to pre-train their agents with their own definitions of fitness with different values for enemies killed, bases destroyed, accuracy, or any sort of objective or capability of the agent. Once they were sufficiently trained, one player's agents would be matched against another player's in a random arena, containing varying walls and objectives. When the battle begins, rtNEAT takes over with its starting net being what was pre-trained. Since it is happening in real time, fitness scores are evaluated continuously and used to compare one agent's performance to another and to gather the overall average fitness. With this knowledge in mind, the algorithm looks to replace its lowest fitness agent with a new one bred from those determined to be performing the best. Unlike NEAT, agents could not be individually separated into species,

rather the entire collection of agents alive at one time would be listed as one species. After a tournament to test players' agents was held, different strategies could be seen evolving both through the tournament as a whole and in individual matches: agents would learn to take a longer path to the objective after losing too many to a particular hallway with heavy enemy presence, or would learn to trap avoidant agents who avoided enemies into corners they could not escape. This mid-match adaptation showed rtNEAT's ability to adapt and provide positive results in real time scenarios. Future applications of rtNEAT look to expand into adaptive characters in player versus environment video games to adapt to individual users as they play, or adaptive difficulty when it comes to training emergency response professionals [8].

2.4 Future Applications of NEAT

While NEAT has found applications in small to medium-scale environments such as online video games with modest populations, large-scale networks still pose a challenge when it comes to minimizing computational demand. One approach, HyperNEAT, looks to address these scalability concerns.

2.4.1 HyperNEAT

One of the largest problems with large-scale neural networks comes from their intrinsic complexity. Thousands of neurons with thousands of connections can lead to long computation times for even a single input. One approach HyperNEAT takes is to indirectly encode connections via compositional pattern-producing networks (CPPN). CPPNs encode patterns found in neural networks rather than each individual neuron and weight, vastly reducing the complexity of large-scale networks. Since neural networks are often represented as points in a 3D space, HyperNEAT expands to a hyper-cube representation creating an additional dimension to assign additional spatial relationships between neurons. The combination of both a hyper-cube representation and CPPN greatly reduces the size and complexity of large neural networks while maintaining their learned functions [9].

3 Solution

3.1 Data Collection

Although human players are able to visualize and understand almost every element of Galaga, a small neural network will become easily overloaded and unable to make meaningful adaptations if it is simply fed every bit of information available. Due to this, information must be hand-selected and then later refined based off of its importance. For GALAGA, three main pieces of information were selected as inputs for the network: enemy ship positions and types, player lives left, and current stage. In order to find this information, the memory of Galaga for the NES has to be mapped to find these three pieces of data are

stored. Once found, the algorithm will read these pieces of information, refine them, and ultimately determine an action or actions to take in response.

3.2 Data Refinement

Once the data has been collected from memory, it must be refined to create standardized and predictable inputs for the neural network. Raw positions and types of ships on screen may provide all available information, however, the algorithm will struggle to make meaningful adaptations when given seemingly random numbers with a large range between them. To solve this, the actual positions of ships must be discarded in place for a grid of generalized coordinates. By treating each 16x16 pixel section of the playable screen, an array of size 12x14 can be created whose values correspond to the sprite type(s) currently occupying it. Additionally, to further simplify the learning process, the grid can be centered around the player ships position.

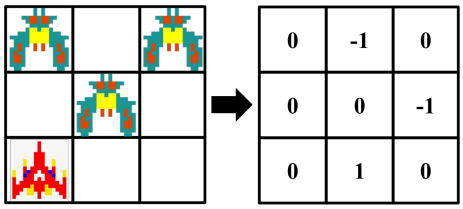

Figure 1: Example of ship coordinates being refined into single values with an offset applied

3.3 Genetic Neural Networks

The genetic neural network behind our solution will implement the NEAT framework as previously described. Initially, a pool of random simple genomes will be generated to represent the first generation. Each genome houses a variety of species, each of which represents a small change to individual parts of the core network. Species that fail to improve fitness after a few iterations, known as stale species, are culled in place for those that show improvement. The highest scoring of these species within genomes are then selected for breeding of the next generation. In NEAT, breeding mainly focuses on cross-mating within genomes with occasional children between children in order to maintain a core network that has been shown to work. These children are then given random mutations and have their fitness tested.

Algorithm 1 Genetic Neural Network using NEAT

```
Initialize simple random population
while true do
    Evaluate fitness of genomes
    if fitness greater than previous fitness then
        Add to breeding pool
    end if
    Apply genetic markers to preserve essential structure(s)
    Reproduce using crossover
    Mutate children randomly
end while
```

3.4 Fitness Variables

The last part to our solution is how the fitness of each individual network will be calculated. Although quite a few factors could be used in this calculation, only four will be used to determine a network's success: score, enemy ship kills, lives remaining, and time passed. Although score alone could be used as fitness, lowering its value and instead adding a flat amount per ship kill allows the network to learn ship eliminations are important without the added trope of Galaga's complicated scoring over-rewarding singular kills in specific scenarios. Additionally, the amount of lives remaining rewards networks who adapt a sense of self preservation, promoting networks that learn to live long even without scoring at the same time. Finally, to promote adaptations that lead to faster times than previous networks, time past is subtracted from the fitness score calculated by the previous three variables. Overall, the following fitness equation is presented:

$$(1) fitness = (score * 0.25) + (kills * 100) + (lives * 100) - (frames * 0.1) \quad (1)$$

4 Experiments and Results

Testing of the algorithm largely meant running it on the emulated version of Galaga for extended periods of time. Each trial run was given around eight hours of runtime, generally leading to 20-30 generations being created, each containing 350 species. To prevent time waste on species who were unable to score points, a timeout counter was added which culls a species prematurely if it does not gain score within a certain period of time.

4.1 First Trial

Initially, the data refinement did not contain offsetting the data to center it around the player ship. With this in mind, the algorithm was run for about eight hours, leading to 26 generations. As shown in Figure 2, the algorithm was able to quickly learn that shooting was the only way to increase score, causing it to be a core feature of all future generations. Additionally, the first level of Galaga was beaten by generation 5, albeit with rudimentary strategy and no sense of self-preservation. Lastly, fitness plateaued after generation 8, likely due to the complexity of the data it was given, leading to changes in future trials.

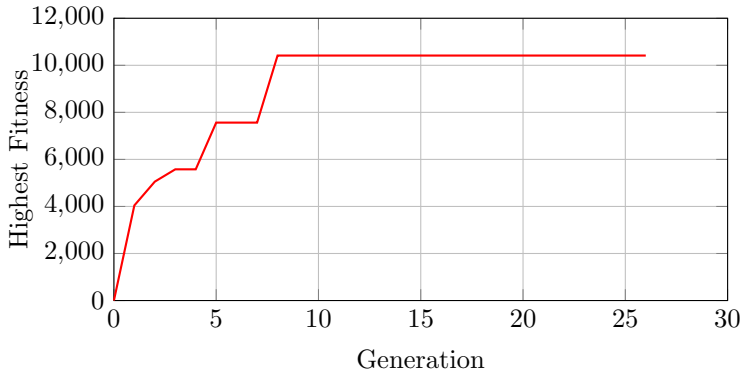

Figure 2: Initial trial fittest networks of each generation

4.2 Second Trial

With the stagnation of learning in the previous trial, data was instead refined to center it around the player ship. This change allowed the neural network to learn how to use objects' relation to the player ship, rather than their relation to overall position on the screen. For example, the network would only have to learn to dodge if an enemy was coming straight towards it once, rather than learning to dodge many times over, once for each possible position the ship could be in. After running a new trial with these changes, the network not only learned faster, but exceeded the previous trial's highest fitness. However, stagnation appeared around a similar fitness score, leading to questions about how fitness should be calculated.

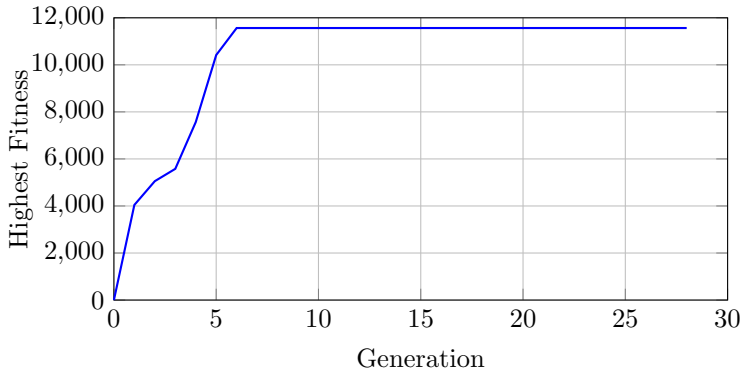

Figure 3: Second trial fittest networks of each generation

4.3 Third Trial

Before the third and final trial was run, a few changes were made to how fitness was calculated. For the first two trials, fitness was calculated simply by score. In order to prevent premature convergence, this equation had to be changed into its final form as described in equation (1). This led to overall lower fitness scores, however their percentage growth in relation to one another stayed largely the same. Due to the third trial being the final, it was run for over 24 hours to allow for more results. In this time frame, it achieved fewer generations because of the high fitness it achieved early on. This initial high score led to multiple levels being completed, increasing training time for the average species. This trend of completing additional levels continued until generation 18, where it hit its final plateau and was stopped before it could adapt beyond it.

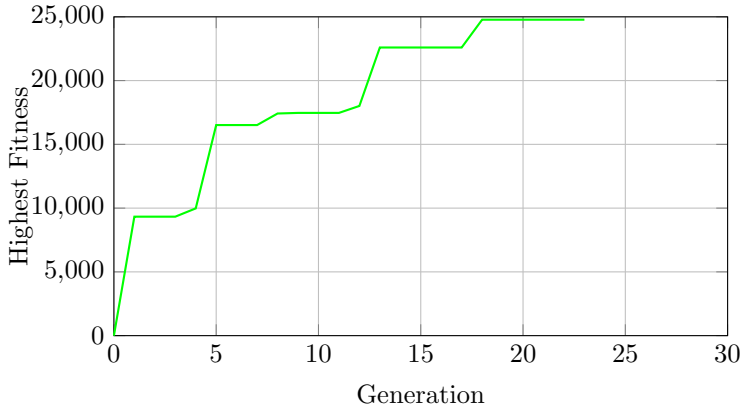

Figure 4: Third trial fittest networks of each generation

5 Analysis

When all three trials are plotted on the same chart, the changes in both how inputs are formatted and fitness is calculated show increases in learning rate and average relative fitness gained per generation. While the first two trials hit a local maximum early on, the third iteration was stopped prematurely and could have easily continued to adapt well beyond its ending fitness. While adding the offset on input data to follow the player ship around allowed for a slight increase in fitness and learning rate, the final change to how fitness was calculated improved the genetic neural network's learning rate significantly. Both of these changes display how formatting input data for algorithms can greatly contribute to their ability to achieve results.

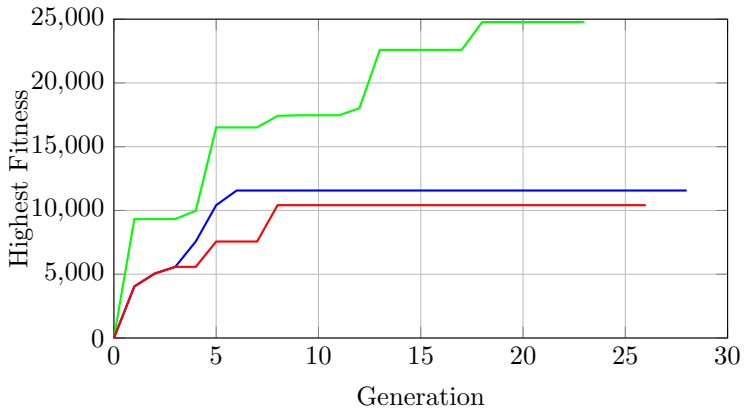

Figure 5: All three trials fitness trends combined

5.1 Learning Successes

As expressed in the expectations, the algorithm was predicted to learn basics like dodging and intentional aim. Although some species learned to dodge in specific scenarios, the ability to evade all ships and projectiles never emerged to its fullest extent possible. This is likely due to the limited time the algorithm was allowed to run for; had it been able to run for longer it likely would have been able to merge these singular evolutions of dodging into one individual network. Additionally, the algorithm learned quickly that shooting was the only way to gain score, leading every genome to contain some form of it. However, the lack of any penalty for missed shots led to a near-constant state of firing. In the more advanced tactics, the algorithm did learn early on, around generation 13 of trial three, to allow itself to be captured to effectively double its rate of fire. The jump in fitness can easily be seen in Figure 5, where every future increase in fitness incorporated this double ship strategy. While it did learn to gain an additional ship, it did not adapt to maintain it, leading to frequent losses shortly after obtaining it.

5.2 Learning Failures

The largest failures in learning came from failure to properly manipulate data and issues with calculating fitness. When the input data was not offset to follow the player fighter around, the network was expected to learn the same maneuver for each of the twelve possible positions the ship could be in. By applying the offset for future trials learning potential was greatly increased. Another failure came in how fitness was initially determined. By only accounting for score, networks were not rewarded for survival but instead playing to Galaga's convoluted scoring system. In certain scenarios, certain ship kills could be worth over sixteen times that of a regular elimination, causing the evolutions to favor self-destructive tendencies that rewarded a large amount of score, but destroyed the player fighter in the process. By reducing the value of score and instead adding a flat value per ship destroyed, species were rewarded for scoring kills while maintaining the option for small bumps in efficiency for playing to Galaga's innate scoring. One of the most prominent learning failures came in the network's inability to learn the entrance patterns of ships. Although it was able to complete multiple levels in a single species, it never learned to destroy ships as they entered to eliminate the possibility of enemy attacks instead of simply dodging them. This failure is likely attributed to the network being reactive rather than proactive. Since there was no memory shared between species, a network could not know where enemies were coming from until they already entered the screen. This inability to predict meant the network would only move to destroy ships after they had already spawned.

6 Future Changes

Much of the changes for future trials would come in the form of further data refinement and altering input data. While a majority of objects on screen were captured for the neural network, situations like approaching the edge of the screen or knowledge of having one or two ships at once were forgotten. To fix this, a value must be placed where the edge of the screen would be, with offset accounted for, to allow the network to learn when it can no longer go left or right to evade objects. Additionally, a second value or change in initial value for the player ship must be added when two allied fighters are on screen, allowing it to learn to preserve both and not just the initial one. A final change for future trials would come in how fitness is calculated. Due to the lack of punishment for missed shots, all genomes maintained a careless approach when determining when and what to shoot at. By introducing a punishment for low accuracy, the network could eventually learn to be more precise and possibly excel in other areas.

7 Conclusion

While genetic neural networks using the NEAT framework have been shown to conquer simpler games like Pong or single levels of Super Mario Bros, Galaga stood as a greater challenge. Through three separate trials, improved methods of data manipulation and fitness calculation were developed to improve the capabilities of the algorithm to conquer Galaga. Although successes were achieved in basic tactics of self-preservation and scoring kills, all but one advanced strategy failed to evolve. This failure can likely be attributed to limitations within genetic neural networks and failure to provide all necessary data inputs for advanced learning. Overall, genetic neural networks have the potential to solve complex problems given properly formatted inputs and enough time to evolve.

References

[1] Ethern Alpaydin. *Introduction to Machine Learning, Fourth Edition.* MIT Press, 2020.

[2] Yong Wee Foo, Cindy Goh, and Yun Li. Speciation and diversity balance for genetic algorithms and application to structural neural network learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.

[3] Erin Jonathan Hastings, Ratan K. Guha, and Kenneth O. Stanley. Automatic content generation in the galactic arms race video game. *IEE Transactions on Computational Intelligence and AI in Games*, 2009.

[4] Richa Mahajan and Gaganpreet Kaur. Neural networks using genetic algorithms. *International Journal of Computer Applications*, 2013.

[5] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Methamtical Biophysics*, 1943.

[6] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1968.

[7] SethBling. Mari/o - machine learning for video games, 2015.

[8] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 2005.

[9] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *University of Central Florida, STARS*, 2009.

[10] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *The MIT Press Journals*, 2002.