MAR I/O

[1]Anshul Hudda, [2]Kartik Goel, [3]Pawan Kumar Rai, [4]Shubham Jadon, [5]Ms. Priyanka Singh
[1,2,3,4]Student (B.Tech 8th sem), [5]Professor
Department of Information Technology
Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi, India

Abstract: - A large number of algorithms to train agents of the game have been developed in recent years. Most of them use artificial intelligence techniques that need a training stage. In this context, this paper aims to develop an agent that can be trained without the need of a training stage and instead uses NEAT to evolve that agent capable of playing Super Mario. NEAT is used to find the neural network architecture that can perfectly play the game and allows agents to improve by playing the game. The algorithm starts with the idea of basic rules of the game like all the conditions about how an agent can die and what different keys it can use as input to agent and aim of the game like to reach the finish line and all the pixels representing agent, enemy and blocks are fed to the algorithm. NEAT uses all this data to find optimal values of weight and bias for the Neural Network by maximizing the Fitness function which tells it how good it is doing at a certain time during training. To simulate the Mario game OpenAI gym-retro is used that makes it possible for NEAT to get updated states of agent and its environment. Coupling the minimal training strategy, openAI, a representative fitness and NEAT, the algorithm was able to find a neural network that allowed it to finish the game and with more training achieved the almost perfect behavior in the game. This paper reviewed how genetic algorithms like NEAT can be used to train AI and make it learn in complex environments. But there are some issues with this approach, like the need to define a good fitness function as with good fitness function AI can learn well but with poor fitness function AI may never learn at all.

Keywords: - artificial intelligence; autonomous agents; neuroevolution; super mario bros

1. INTRODUCTION

The most important parameter to make any AI agent or machine learning modal to learn is to have Training data on which model can be trained and Test data on which it can be tested . Getting such data introduces a new level of complexity to develop any good agent that is to make good or unbiased Training or Test data which is very time consuming and even very hard to make in certain cases. That is where Neuroevolution technique comes into play as It makes it possible for Artificial Neural Networks(ANN) to learn/evolve in unsupervised learning problems(where there is no input and output data to train or test agent). With the help of Neuroevolution technique ANN doesn't need to depend on correct output value, which helps it to calculate cost function which in turn makes it possible to optimize the network settings though Gradient descent algorithm [7]. Neuroevolution technique takes its inspiration from how a human species as whole develops and continues to survive because of evolution which allows humans to adapt to various dietary or survival needs [2]. So, to see this technique in action, the game Super Mario shows itself as a promising virtual testing environment to optimize agents (Mario character). It is a popular game in which the aim of the player is to reach the end of a certain level without touching any enemy or falling off a block which kills the player. The player's moment depends on four keys. In this type of problem, the whole learning process happens with trial and error. The Training data is developed when the agent interacts with its environment. In the beginning of the game there is no information about what actions should be taken or whether certain actions taken are good or bad which make agents' communication with the environment random. In this context, the use of Super Mario as an environment for Neuroevolution algorithms tests is an excellent study, since the obstacle-transposes many challenges which find applications in many fields like Smart navigation for Robots, game development etc. Because of the challenge of finding path to the end of level without dying makes using Neuroevolution as the best choice as it can help us find best configuration of an ANN without depending on a set of correct actions within the Super Mario, also technique of neuroevolution requires only a value that translates agent performance at the end of its lifetime, and through the choice of best performance of various agents and combinations of various setting the technique finds the ANN that can direct agent from start to finish. In this work we use a Neuroevolution algorithm called Neuroevolution of Augmenting Topologies(NEAT) [6] in the Super Mario environment, using a minimal strategy NEAT makes a simple agent to play the game. The choice of NEAT is decided by the fact that it starts from very simple configuration of agent very it only knows how to move and complicates this configuration over generations to include enemies, and other states of game and making agent more better with newer generations making it possible for agent to learn to navigate its way from start to finish line without the need of any training data [8].

2. STRUCTURE OF ANN

It’s actually quite similar to the human brain. Typically, an ANN is composed of interconnected processing units called nodes or neurons. The connection between the neurons is called synaptic weights, just like the synapses in your brain that receive signals from your environment. The neurons can

be classified as input, hidden or output, all of which varies depending on the task at hand. For instance, if you want to classify an image, the inputs can be the features of your dataset and the many outputs are classes to predict. To make it simpler, think of the baby when the baby is born, the baby sees oranges and apples (the inputs) but they don't know how to name them yet, just as how our naive neural network doesn't know what names to put on images. After a couple rounds of practice, the baby's brain picks up the words, and in their mind, they know that's an apple. Similarly, our ANN post-training will ideally be able to classify oranges and apples with an ideally high accuracy rate. The architecture of ANN is characterized by the specific structure of nodes and connections that create a certain network. Usually, the topologies of ANNs are predetermined by the programmer, but the problem is that they may not be the most efficient model. That's why we have something known as back propagation, or more intuitively speaking "the back propagation of errors". Like how it sounds, the ANN compares the predicted output with the actual output. Then, it essentially uses this information to adjust the synaptic weight with the hope to minimize the cost/error function [4].

Genetic Evolution Algorithm: Genetics algorithm was inspired by Charles Darwin's theory of natural selection. This algorithm uses the process of natural selection where the fittest individuals are chosen for reproduction in order to produce offspring of the next generation [10].

Here's how it works in nature on a macro scale:

I. The process of natural selection starts with a population with individuals who have different characteristics. Let's take the example of a giraffe population that has both long-necked and short-necked giraffes. These giraffes have different phenotypes because they have different genomes (think codes for life).

II. The long-necked giraffes can more easily reach the leaves at the top and thus have a higher survival advantage than their short-necked peers. As a result, more of them will survive and pass their genes to the next generation. In other words, the long-necked giraffes are considered to be more fit.

III. The fittest giraffes would survive and through hundreds or maybe thousands of generations, all individuals in the giraffe population will have long necks. In short, nature favors the survival of fitness. If you used to think that giraffes have always had long necks and think that this is quite interesting, let's dive even

deeper into the molecular level of things and understand the root of evolution.

A genetic algorithm provides a potential solution to a problem (the phenotype) in a chromosome-like data structure called the genotype or genome.

Recapitulating this process of natural selection, the genetic algorithm creates an initial population of random genomes, which are materialized as phenotypes and evaluated on the basis of some fitness function. Basically, using genetic algorithms, you can determine the "survival of the fittest ANNs.

NEAT Algorithm: As the name implies, the architectures of the neural networks, i.e. topology, becomes better and better at completing a certain task in the course of evolution. The genome of an ANN contains two parts: node genes and connection genes. Each node gene specifies a single neuron. On the other hand, the connection genes represent the connection between the neurons, the weight of the connection, whether or not the connection is enabled, and a historical marker that provides information about the ancestral history of the gene[6].

Before NEAT, there were a handful of attempts at evolving topologies of networks that were somewhat successful, however, they identified a series of problems that would need to be overcome before the technology could actually do anything incredibly useful. What made NEAT and its paper so interesting is some of the solutions it proposed to these problems, solutions that still make this paper relevant today [1].

3. METHODOLOGY

Two components are required to this work: fitness function calculation and phenotype settings.

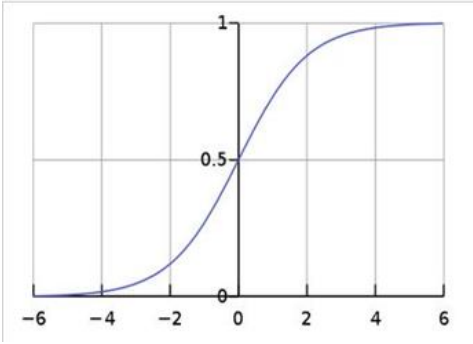
A. Fitness

To compute the fitness of the agent we use distance covered by the agent and the fitness function is denoted by sigmoid.

Here, x is the horizontal distance.

The domain of sigmoid function is real numbers, with return (response) value commonly monotonically increasing but could be decreasing. Sigmoid functions most often show a response in the range 0 to 1. Another commonly used range is from −1 to 1 [10] .

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} = 1 - S(-x).$$



B. Phenotype settings

Earlier, there were a handful of attempts at evolving topologies of networks that were somewhat successful; however, they identified a series of problems that would need to be overcome before the technology could actually do anything incredibly useful. What made NEAT and its paper so interesting is some of the solutions it proposed to these problems, solutions that still make this paper relevant today [1].

Encoding

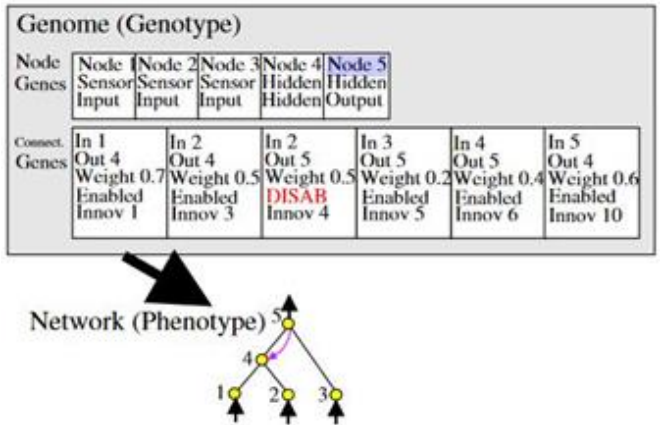
We have a genotype and a phenotype in biology. A genotype is the genetic representation of a creature and the phenotype is the physical representation of the creature. Evolutionary algorithms always mirror biology, neuroevolution being no different in this respect.

The idea of encoding comes from the question of how we wish to represent individuals' genetics in our algorithm. The way in which we encode our individuals lays out the path for how our algorithm will handle the key evolutionary processes: selection, mutation, and crossover (also known as recombination). Any encoding will fall into one of two categories, direct or indirect.

A direct encoding will specify every information about an individual. If it represents a neural network this means that each gene will directly be connected to some node, connection, or property of the network. This can be a binary encoding of 1s and 0s, a graph encoding (linking various nodes by weighted connections), or something even more complex. The point is that there will always be a direct link between genotype and phenotype that is very obvious and readable.

An indirect encoding is the exact opposite of direct. Instead of directly specifying what a structure may look like, indirect encodings tend to specify protocol or parameters of processes for creating an individual. That's why indirect encodings are much more compact. The other side is that setting the rules for an indirect encoding can result in a heavy bias within the search space, therefore, it is much harder to create an indirect encoding without complete knowledge about how the encoding will be used.

The NEAT algorithm chooses a direct encoding technique because of this. Their representation is a little more complex than a simple graph or binary encoding, however, it is still understandable. It has two lists of genes, a series of nodes and a series of connections. To see what this looks like visually, we used a picture here:

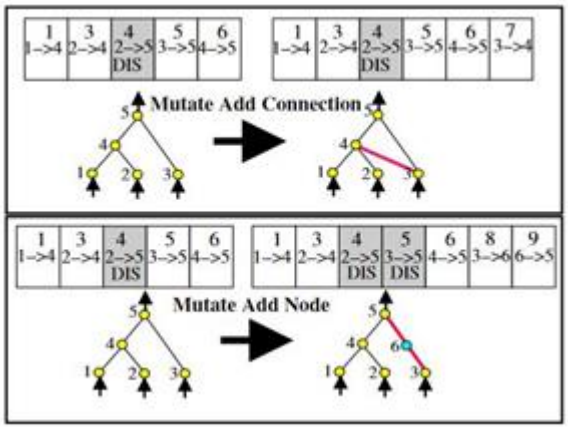


Input and output nodes are not evolved within the node gene list. Hidden nodes can be added or deleted. As for connection nodes, they specify where a connection comes into and out of, the weight of such connection, whether or not the connection is enabled, and an innovation number (something we'll discuss in the next section).

Mutation

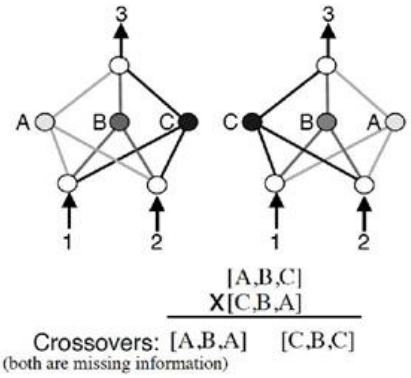
In NEAT, mutation can either mutate existing connections or can add new links to a network. If a new link is added between a start and end node, it is randomly assigned some weight [3].

If a new node is added in structure, it will be placed between two nodes that are already connected. The previous connection is disabled (though it is still present within the genome). The previous start node is connected to the new node with the weight of the old connection and the new node is connected with the previous end node with a weight of 1. This was found to assist mitigate issues with new structural additions.



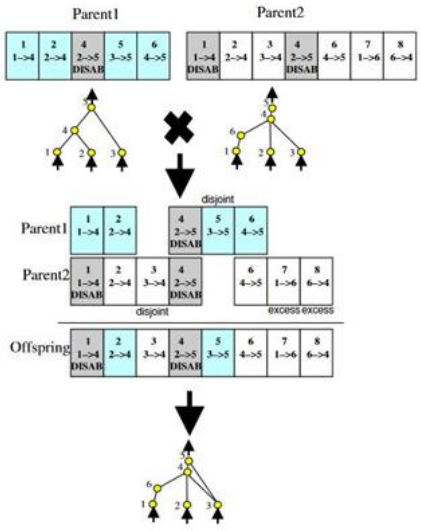
Competing Conventions

Another big problem in evolving the topologies of neural networks is something that the NEAT paper calls "competing conventions." The thought is that just blindly crossing over the genomes of two neural networks could end in networks that are horribly mutated and non-functional. If two networks are dependent on central nodes that both get relinked out of the network, we have an issue



More than that, genomes are often of various sizes. How can we align genomes that don't seem to be obviously compatible? In biology, this is nursed through an idea called homology. Homology is the alignment of chromosomes based on matching genes for a specific characteristic. Once that happens, crossover can happen with much less chance of error than if chromosomes were blindly mixed together.

NEAT solves this issue through the usage of historical markings (as shown in figure). By assigning new evolutions with a historical number, when it comes time to crossover two individuals, this can be done with much less chance of making individuals that are non-functional. Each gene can be aligned and potentially crossed-over. Each time a new node or new type of connection occurs, a historical marking is assigned, allowing easy alignment when it comes to breeding two of our individuals. Shown in fig.



Speciation

A very interesting idea put forth in NEAT was that the majority of new evolutions aren't good ones. In fact, adding a new connection or node before any optimization of weights has occurred often leads to a lower performing individual. This will put new structures at a disadvantage. How can we protect new structures and permit them to optimize before we remove them from the population entirely? NEAT suggests speciation.

Speciation simply splits up the population into several species supporting the similarity of topology and connections. If the competing convention problem still existed, this would be very hard to measure. However, since NEAT uses historical markings in its encoding, this becomes much easier to calculate. A function for deciding how to speciate is given within the paper, but the important part to note is that individuals in a population only have to compete with other individuals within that species. This allows for new structures to be created and optimized without fear that it will be eliminated before it can be truly explored.

More than that, NEAT moves things one step forward through something called explicit fitness sharing. That means that individuals share how well they are doing across the species, boosting up higher performing species, though still allowing other species to explore their structure optimization before being out evolved.

Minimal Structure

Major goal of the NEAT paper was to make a framework for evolving networks that allowed for minimal networks to be evolved. The authors didn't want to make an algorithm that first found good networks and then had to scale back the number of nodes and connections after the fact. Instead, the thought was to create an algorithm that started with the minimal amount of nodes and connections, evolving complexity as time goes on if and as long as it is found to be useful and necessary.

NEAT sets up their algorithm to evolve minimal networks by starting all networks with no hidden nodes. Each individual within the initial population is simply input nodes, output nodes, and a series of connection genes connected between them. By itself, this may not necessarily work, but when combined with the idea of speciation, this proves to be a strong idea in evolving minimal, yet high-performing networks [5] .

4. RESULT

All AI were trained using the pink box as inputs. . For this we ran several populations on level 1-1 with the same random seed. The random seed was to possess populations select, crossover, and mutate an equivalent. There is a tradeoff between accuracy and speed for training. With more inputs, Mario could be ready to learn tons more about the

environment, however, it's going to take significantly longer. We ran for a number of generations to see which populations learned faster. We then took the input dimensions of the best population and restarted all training with those dimensions that it will be eliminated before it can be truly explored.
More than that, NEAT moves things one step forward through something called explicit fitness sharing. That means that individuals share how well they are doing across the species, boosting up higher performing species, though still allowing other species to explore their structure optimization before being out evolved.

Minimal Structure

Major goal of the NEAT paper was to make a framework for evolving networks that allowed for minimal networks to be evolved. The authors didn't want to make an algorithm that first found good networks and then had to scale back the number of nodes and connections after the fact. Instead, the thought was to create an algorithm that started with the minimal amount of nodes and connections, evolving complexity as time goes on if and as long as it is found to be useful and necessary.

NEAT sets up their algorithm to evolve minimal networks by starting all networks with no hidden nodes. Each individual within the initial population is simply input nodes, output nodes, and a series of connection genes connected between them. By itself, this may not necessarily work, but when combined with the idea of speciation, this proves to be a strong idea in evolving minimal, yet high-performing networks [5] .

5. RESULT

All AI were trained using the pink box as inputs. . For this we ran several populations on level 1-1 with the same random seed. The random seed was to possess populations select, crossover, and mutate an equivalent. There is a tradeoff between accuracy and speed for training. With more inputs, Mario could be ready to learn tons more about the environment, however, it's going to take significantly longer. We ran for a number of generations to see which populations learned faster. We then took the input dimensions of the best population and restarted all training with those dimensions.

It's extremely precise and many players would never be able to actually do one. This AI, on the other hand, was able to figure out how to perform a wall jump and utilize it to get past a hole. One thing that AI's are being used to find exploits and vulnerabilities in systems. The AI has no idea that it should not be able to do something or that it should. It simply tries all possible ways to maximize fitness. Evaluating all of the results together it can be seen that our methodology turned the game into a problem simple to solve by the algorithm. Those results show that training agents using a very restricted training environment can lead to optimal behaviors.

6. CONCLUSION

In this work, the authors propose a minimal training strategy to generate agents capable of achieving optimal scores and passing different levels in the game Super Mario.

This implementation of mar I/O gives better results than a human after few generations of learning. This shows that this strategy can find optimal solutions in a short number of generations but some levels are seems impossible to beat. That's all there is to it ,with a few adjustments, this framework is applicable to any game for the NES, SNES, SEGA Genesis, and more.

REFERENCES

[1] R. Chuchro, "Game playing with deep q-learning using openai gym," Semantic Scholar, 2017.

[2] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, pp. 653–668, 2005.

[3] Gomez, F., and Miikkulainen, R. (1998). 2-D pole-balancing with recurrent evolutionary networks. In Proceedings of the International Conference on Artificial Neural Networks, 425–430. Berlin: Springer.

[4]] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[5] M. Ebeling-Rump and Z. Hervieux-Moore, "Applying q learning to flappy bird," Semantic Scholar, 2016.

[6] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127, 2002.

[7] S. Karakovskiy and J. Togelius. The mario ai benchmark and competitions. Computational Intelligence and AI in Games, IEEE Transactions on, 4(1):55–67, 2012.

[8] J. Togelius, S. Karakovskiy, and R. Baumgarten. The 2009 mario ai competition. In Evolutionary Computation (CEC), 2010

[9] A. McIntyre, M. Kallada, C. G. Miguel, and C. F. da Silva,"neat-python," https://github.com/CodeReclaimers/neat-python.

[10] D. Whitley. A genetic algorithm tutorial. Statistics and Computing, 4(2):65–85, 1994.

