Investigation and Implementation of an Evolutionary Artificial Neural Network

B.A Digital Games
Cologne Game Lab / TH Köln

by
Jorge Hurtado Grueso
2019

Table of Contents
Thesis

Documentation of Project

Investigation and Implementation of an Evolutionary Artificial Neural Network

By Jorge Hurtado Grueso, Student ID 11132045, r4z31996@gmail.com

written in the Summer Semester 2019 for BA Digital Games at Cologne Game Lab/ TH Köln supervised by Prof. Dr. Roland Klemke submitted on 28/06/2019

ABSTRACT

This paper discusses the combination of two different methods in machine learning, the hybridization between Evolutionary Algorithms (EA) and Artificial Neural Networks resulting in Evolutionary Artificial Neural Networks (EANN). The proposed system is implemented in a Car Simulator and the results are collected in this paper.

Keywords

Artificial Neural Network, Genetic algorithm, Optimization

Acknowledgements

I want to thank my thesis advisor, Roland Klemke, for his guidance and useful feedback. My grateful thanks are also extended to Mr. Jorge Casedas for his help and advice.

I would also like to thank my family for giving me the opportunity to study my passion and for giving me lots of support in every decision I took.

Overview

This thesis paper is divided in 6 chapters. In the Chapter 1 introduces the main topic of the project. Chapter 2 summarizes all the basic concepts of the thesis: Neural Networks and Genetic Algorithms. Chapter 3 cover the hybridization of the Artificial Neural Networks and Genetic Algorithms. In Chapter 4 we study the State-of-Art of Neuroevolution. Chapter 5 covers the implementation. Lastly, Chapter 6 recollect the conclusion.

1 INTRODUCTION:

Learning is the gain of knowledge by study, experiencing or being taught.

In 1959, Arthur Samuel describes Machine Learning as *"The field of study that gives computer the ability to learn without being explicitly programmed"*. As conclusion he said that programming computers to learn by themselves from experiences should eventually eliminate

the need for most of this detailed programming effort. [23]

Samuels definition sounds a little bit vague but in 1997 Tom Mitchell, gives a more precise definition:

Well posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. [29]

According to his definition, if we want our program to recognize for example, handwritten digits (task T), we can run it through a machine learning algorithm with human-labeled images of handwritten digits (experience E) and, if it has successfully "learned", it will then do better at recognizing handwritten digits (performance measure P).

1.1 Neuroevolution

Neuroevolution is the combination of Evolutionary algorithms with Artificial Neural Network. The objective of this combination is the optimization of the hyperparameters of the ANN. In this paper we will use a type of Evolutionary Algorithms (EA), more precisely Genetic algorithms, which are the most popular type of Evolutionary Algorithms.

Both, Genetic algorithms and Artificial Neural Network are inspired by biological processes, such us evolution and brain processes.

1.2 The Combination

In the late 80s, combining Artificial Neural Networks and Genetic algorithms became popular and started to appear some researches with the objective of choosing the correct number of parameters (Hyperparameters) before training the Neural Network.

2 CONCEPTS

2.1 NEURAL NETWORKS

Artificial Neural Networks as the name suggests are inspired by biological nervous systems. They are able to adapt and learn with the correct given training. Contradicting what people usually think, Artificial Neural Networks are not capable of thinking in the same way as human brains do. Nonetheless they are useful for searching solutions to different types of problems for which an algorithm or at least an efficient algorithm, has not

been found thanks to their adaptive learning properties. [2]

The first idea of an Artificial Neural Network goes back to 1943, to two researchers at the University of Chicago, Warren McCullough, a neurophysiologist and Walter Pitts, a mathematician. [30]

2.1.1 Artificial Neurons

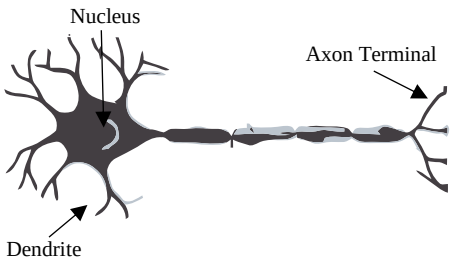


Figure 1 Model of a biological neuron

Each Neural Network is composed of different number of Artificial Neurons. The number of Neurons and structure depends on problem for which they have been proposed.

One of the most important and trivial artificial neurons that uses the linear threshold function was the Perceptron. It was developed by Frank Rosenblatt in 1957. The Perceptron was used in machines with adaptive capabilities.

In the biological neuron we found the dendrites which correspond to the inputs in the artificial neuron, while the axon of the biological neuron is represented as an output in the artificial neuron.

As we saw before each Neural Network is composed of the connection of different number of Artificial Neurons, more precisely the outputs from a neuron to the inputs from another neuron. We will study in-depth in *2.1.2 Network Topology*.

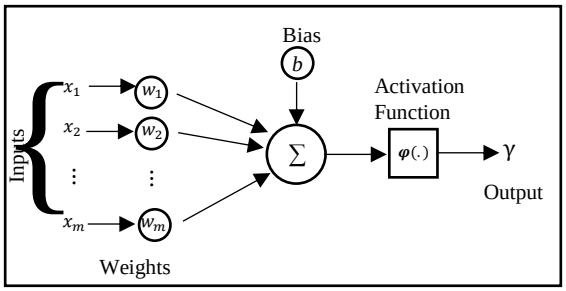


Figure 2 Model of an artificial neuron

The simple artificial neuron has *n* inputs represented as a vector $\vec{x} \in \mathbb{R}^n$, a bias $b \in \mathbb{Z}$, each input with a corresponding weight $\omega_i \in \mathbb{Z}$ and combined they produce a value. This output runs through an activation function producing an output $\mathcal{y}$.

As an example of an artificial neuron let´s consider a neuron *k. T*his neuron has a vector of inputs $\vec{x} = (x_{1k}, \cdots, x_{nk})$, weights $w_{1k}, \cdots, w_{nk}$ and a bias θ, which results in the next equation for our Artificial Neuron *Figure 2*

$$\xi_k = \sum_{i=1}^{n} w_{ik} x_{ik} + \theta_k$$

It is time to choose an activation function, for this example we consider the Sigmoid Function.

$$f\ (\xi_k) = f(x) = \frac{1}{1 + e^{-\xi_k}}$$

As a result, the output y_i of the neuron k is given by

$$y_i = f\ (\xi_k) = f\left(\sum_{i=1}^{n} w_{ik} x_{ik} + \theta_k\right)$$

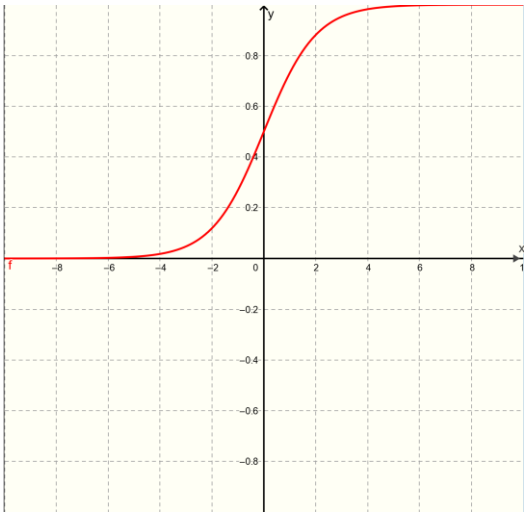
2.1.1.1 Activation Functions

Activation Functions are functions used to compute the value of the neuron and to decide if a neuron should be fired or not.

They can be either linear or non-linear depending on the functions they represent. [7]

Now, we study the most common Activation Functions used in Deep Learning.

1 Sigmoid Function:



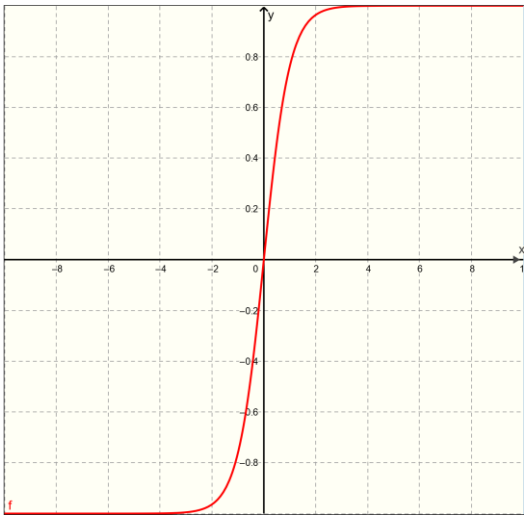
$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure 3 Graph of the Sigmoid function

The Sigmoid function is a non-linear Activation Function and is mainly used in feedforward neural networks.

Some advantages of the Sigmoid Function are that is easy to understand and works well with shallow networks.

2 Hyperbolic Tangent Function-Tanh



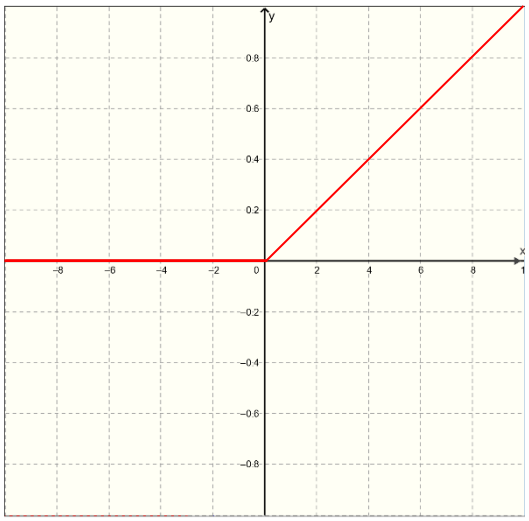
$f(x) = \tanh(x)$

Figure 4 Graph of Hyperbolic Tangent Function

The Hyperbolic Tangent Function, also known as Tanh function, is another non-linear Activation Function and used in Deep Learning. It is a smoother zero centered function with an output range lies between -1 to 1.

Because of its better training performance for multilayer Artificial Neural Networks it is usually preferred compared to the Sigmoid Function.

3 Rectified Linear Unit Function (ReLU)



$$f(x) = \max(0, x) = \begin{cases} x_i, if\ x_i \geq 0 \\ 0, if\ x_i < 0 \end{cases}$$

Figure 5 Graph of ReLU function

In 2010, Nair and Hinton proposed the ReLU Activation Function (Rectified Linear Unit Function). Until today ReLU has been one of the more used functions in Deep Learning, representing a nearly linear function. By preserving the linear models properties, ReLU got easier to optimize using gradient-descent methods.

4 Softmax Function

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j\ exp\ (x_j)}$$

Softmax Function

The Softmax Function, also used in neural computing, takes an input and normalizes it into a vector of values, this vector of values follows a probability distribution whose total sums up to 1. The range of the output values goes from 0 to 1, avoiding binary classification.

.

2.1.2 Network Topology

Until now, we have studied only simple artificial neurons which can solve simple tasks, but Artificial Neural Networks are entire networks of simple artificial neurons.

This structure of neurons can deal with complex associations between the

neuron's inputs and outputs, letting the Artificial Neural Network solve complex decisions.

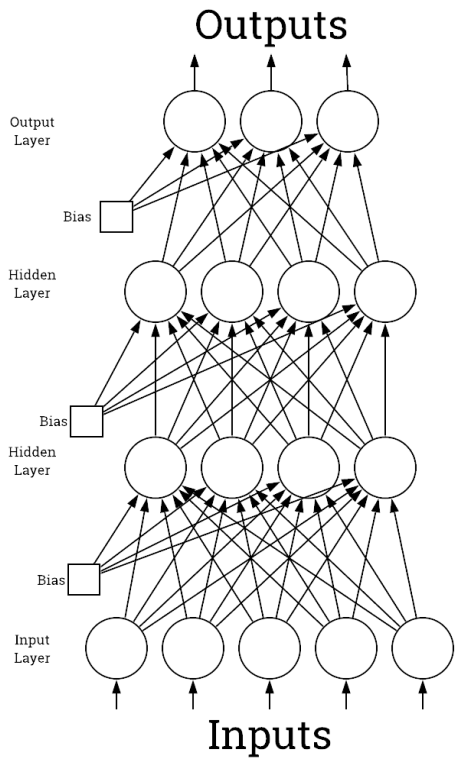


Figure 6 Example Neural Network structure

In the following part we will discuss the simple Deep Feedforward Neural Network. This model is called "feedforward" because the information of the neural networks flows from the inputs to the outputs through the network. The neurons of the network are structured in different layers, each layer output from the neurons is passed to the inputs of the next layers of neurons. This network is fully-connected which results in all neurons of one layer, receiving all outputs of the previous layers and its fully connected.

As Figure 6 shows, each artificial neuron is represented as a circle. Each neuron is connected to another with an arrow indicating that the output of one neuron is the input from the other. Also, we can see that there are different layers. The first layer of neurons is the Input Layer and the last layer is the Output Layer. In this Feedforward network, the flows of the inputs go from the Input Layer through all layers to the Output Layer. The layers between the Input and the Output layers are called "Hidden layers", as the name suggests only the first (Input Layer) and the last (Output Layer) layers are "visible". As we saw in the study before, every layer except the Input Layer receives the Bias, this Bias it is added to the equation of the Artificial Neuron .[15]

2.2 GENETIC ALGORITHM:

"One general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die." Charles Darwin, The Origin of Species

As Artificial Neural Networks, Genetic Algorithms are biologically inspired in the natural selection. The competition among individuals for resources results

in the fittest individuals dominating over the weaker individuals. The basic methods and techniques of Genetic Algorithms are designed to simulate this natural system to choose the fittest individual.

Using generations of individuals, we can simulate the natural selection and choose the fittest individuals over generations for solving a project or an objective. Each generation is formed by a population of strings analogous as the chromosome in our DNA and each individual represents a possible solution of the problem. [11]

These algorithms are computationally simple and result in a very effective search for improvement.

For a simple genetic algorithm to be effective and to give a good result, it should have the following operators:

1. Reproduction
2. Crossover
3. Mutation

2.2.1 Elements and Operators of Genetics Algorithms

Population Encoding In order to find a solution to the problem we need to find a way to represent our population in strings of characters, alphabet or numbers as chromosomes.

Chromosome A	10110010110010101110010 1
Chromosome B	11111110000011000001111 1

Figure 7 An example chromosome using binary encoding.

Depending of the problem different implementations of the Genetic Algorithm and chromone encoding should be used. Chromosomes have no practical restrictions or limits on how they can be encoded.

As Figure 7 shows an individual can be encoded as a string of bits, 0 or 1.

Fitness Function During the execution of the algorithm, the Fitness Function determines how the fitness of each individual increases or decrease. An increase of the fitness of the population determine a better chance to find a solution.

Selection As in Darwin's evolution theory this operator selects the best individuals (according to the fitness of each individual) and choose which ones should be parents in the new generation.

Crossover It selects genes from parent chromosomes and creates a set chromones for a new child. We can choose a crossover point randomly and

copy it before this point from the first parent chromosome and after from the other parent.

Individual 1	11011 | 00100110110
Individual 2	11011 | 11000011110
Child 1	11011 | 11000011110
Child 2	11011 | 00100110110

Figure 8 An example of chromosome crossover. ('|' represent the crossover point)

As Figure 8 shows, a chromosome crossover has been made choosing a crossover point, but there are other ways to make crossover. Depending on the chromosome encoding, the crossover can be more complicated or simpler, a specific crossover function can be made to improve the genetic algorithm.

Mutation After the crossover takes place, there is a chance of the new chromosome to mutate. This mutation changes one or more parameters of the new chromosome randomly, within the limits of the solution space. For our previous examples of bit strings this could mean flipping selected bits randomly.

The main purpose of this Mutation is to add a random operator with the intention of not letting the population get stuck in the maximal fitness individuals, exploring new possible solutions without disrupting good solutions too much.

Elitism As an alternative selection method that can be added, Elitism is a good one to prevent losing the best-found solution. The method is simpler, and the best individuals of the population are put in the new generation.

2.2.2 Flowchart

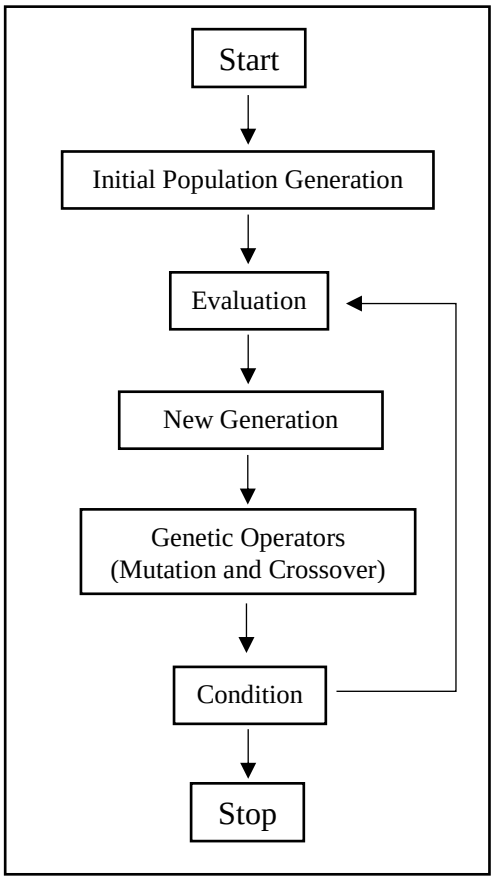


Figure 9 Flowchart of a genetic algorithm.

As seen in Figure 9, the basic process of the Genetic Algorithm is simple.

First, we generate a new population with random chromosomes, which is evaluated with a Fitness Function and ranked. After ranking the population, we generate a new one and which is filled with the previous one but after being applied the genetic operators (Crossover and Mutation). If the condition of the loop is satisfied, we stop the algorithm otherwise we continue until it is satisfied and return the best population.

3 EVOLUTIONARY ARTIFICIAL NEURAL NETWORKS

The three main introductions of Genetic Algorithms into Neural Networks are:

1. Connection Weights
2. Architecture of the ANN
3. Learning Rules

The connection weights evolution gives a global and adaptive approach to training. The architecture evolution avoids the selection of the architecture by the human intervention and lets the Artificial Neural Network adapt their topologies on their own. The learning rules evolution can be considered as the process of the Artificial Neural Network of "learning to learn" [30].

In this paper we shall consider the first and second introductions.

3.1 Training Fixed Neural Networks Structures

With the objective of optimizing the neural network performance and as a replacement of Gradient-Based algorithm, Genetic Algorithms can be used for this purpose.

Evolving the weights of the ANNs using Genetic Algorithms can find the optimal weight set, avoiding computing the gradient information, individual fitness can be defined by the error between the actual output and the expected output.

Another advantage of Genetic Algorithm over the Gradient-Based Algorithm is that Gradient-based methods can get stuck in local a minimum and GAs are more likely to find the global optimum requiring relatively little knowledge of the problem being solved.

3.2 Evolving the Neural Network Structure

The big problem with Neural Networks is which architecture should be used. Complex structures can perform well on different problems but a simpler one may

be unable to perform well with the desired task. Gruauetal (1996) claimed that evolving the particular structure saves the time that human waste to choose which topology should be used for the desired problem.

A good approach to search between different structure alternatives is the use of Genetic Algorithms, avoiding the exitance of redundant nodes and connections and solving the problems in a correct way.

Evolving this structure makes the ANNs fit into different task avoiding a fixed topology with a fully connected hidden layer, deciding which is the number of hidden nodes to perform well the problem. As Kenneth O. Stanley (2002) [18] presents in his method "NEAT" if the topology is evolved and grown incrementally, we can appreciate a significant learning speed result and an efficiency improve.

4 STATE OF THE ART

Nowadays, the Artificial Intelligence community has been attracted by Neuroevolution and recent researches conclude that Neuroevolution methods can be applied in a more widely way while in terms of computational complexity is more achievable. [16]. In this chapter we will study the State of the Art of Neuroevolution.

4.1 The evolution of Evolutionary Artificial Neural Networks

As mentioned before, the hybridization of Neural Networks and Evolutionary Algorithms dates back many years ago. [27]

In 1999, Xin Yao "Evolving Artificial Neural Networks" reviews different combinations between Artificial Neural Networks and Evolutionary Algorithms, discussing different search operators which have been used in Evolutionary Algorithms.

One of the main uses of Genetics Algorithms was to optimize the weights of a fixed Artificial Neural Network but in 2002, Stanley and Miikkulainen came up with the idea of a new algorithm NEAT - Neuroevolution of Augmenting Topologies. This investigation is collected in their paper "Evolving Neural Network through Augmenting Topologies", offering a new method which determines an appropriate topology of the Neural Network to solve the problem. NEAT was focused on the evolving of weights and the own topology of the network, with three different mutations. Firstly, modifying

the weight of the connections, secondly, adding a connection between nodes and thirdly inserting nodes while it splits an existing connection. NEAT uses direct encoding which every neuron and connection in the Artificial Neural Network is specified directly in the genotype.

This NEAT algorithm lead into different extensions over the past years, rtNEAT, HyperNeat, Phased pruning, odNEAT…

One of this extensions, HyperNEAT, based on the NEAT principles solves the problem with the evolution of large-scale neural networks, using an indirect encoding of the neural network. [28]

A recommended lecture is the following research paper, different methods of encoding is "A Review Of Methods For Encoding Neural Network Topologies In Evolutionary Computation" from Jozef Fekiač, Ivan Zelinka and Juan C. Burguillo. [17]

4.2 The comeback

Currently, researches focused on Evolutionary algorithms are emerging more often, helping Artificial Neural Networks to evolve their structure and weight connections through a training of trial and error.

The ascent of deep learning has helped with refocusing the attention on Neuroevolution through the past years.

A big reason why Neuroevolution is coming back is the recent increase of hardware processing power while the Artificial Neural Networks take advantage and benefit from the parallel processing. [19]

A big population of Artificial Neural networks can be processed and intrinsically in parallel, evaluating all the networks at the same time that a simple neural network would be evaluated. The amount of new applications that can take advantage of this kind of speed-up is growing.

Research labs with huge computing clusters can exploit this "Neuroevolution goldmine", prompting new researches and experiments of Neuroevolution to grow up out of labs mainly.

Reinforcement learning is another area in which Neuroevolution is having a comeback. A team of OpenAI "Evolution Strategies as a Scalable Alternative to Reinforcement Learning" [5] reported an experiment with Neuroevolution, in which an extension of this method matches the performance of reinforcement learning methods on modern agent-environment benchmarks,

offering benefits related to the code complexity. Also, they conclude that this variant of Neuroevolution have an important facility of scaling to large-scale settings.

The evolution of plastic Artificial Neural Networks has been also in the aim of the Neuroevolution researches. These plastic Neural networks are networks which evolve the connection weights over the lifetime of the neural network. Neuroevolution is not only choosing the right architecture and weights, also the rules of Learning "how and when the weights of the networks during the lifetime changes". [4]

4.3 Recent Research papers

In this section we summarize recent research papers to give an overview of the actual State of Art.

4.3.1 Large-Scale Evolution of Image Classifiers

In 2017 "Large-Scale Evolution of Image Classifiers" was published by Google, a paper in which the human participation to discovering neural networks was minimized. Employing simple evolutionary techniques and present novel mutation operators to discover the new models. The evolution of complex neural networks such as (CNN) convolutional neural networks was the objective of the algorithm and a large search space was enabled. This allowed a lot of possibilities inserting and removing layers to avoid the restriction of the possible values to chosen from.

During each evolution step the Evolutionary Algorithm is a tournament selection:

- Each computer (worker) selects two individuals random and compare the fitness, the worst is killed and removed from the population, the best one selected to be a parent.
- The parent produces a copy of itself and a mutation operator is applied.
- The new child (parent mutated) is trained and put its back to the population.

Each individual of the population is only a single-layer model without convolution, forcing the evolution to discover better architectures.

Each experiment was distributed over 250 parallel workers. This is a massively parallel lock free infrastructure.

As a conclusion the investigation shows that Neuroevolution can construct large accurate networks for two challenging and popular image classification benchmarks. Once the process is started it does not need experimenter participation and results in fully trained models; Neuroevolution is capable to do this from a trivial initial condition while it searches in a large space. [12]

4.3.2 Efficient Neural Architecture Search via Parameter Sharing

In 2018 ENAS was proposed, a new approach for automatic model design. Discovering artificial neural networks architectures [14] "by searching for an optimal subgraph within a large computational graph".

One of the main issues of Neural Architecture Search was the computational complexity and this paper conclude with a speed up of the Neural Architecture Search, more than 1000x in terms of GPU hours by sharing parameters among child models.

In comparison, NAS can take 3-4 days on 450 GPUs whiles Efficient Neural Architecture Search (ENAS) took 16 hours in a single GPU. This was possible thanks to force all child models to share their weights.

Efficient Neural Architecture Search consist in a controller trained with policy gradient. This controller was trained to search for an optimal subgraph that can maximize the expected reward. During this search, the selected model of the subgraph its also trained to minimize the canonical cross entropy loss.

4.5 Neuroevolution and Games

Georgios N. Yannakakis and Julian Togelius, expert in this field of studies show to us as different ways in which Neuroevolution can be applied in Videogames in their book "Artificial Intelligence and Games". [13]

Neuroevolution can be used to generate different content in games such as: level generation using music [9], in 2014 a small team formed by Amy K. Hoover, Georgios N. Yannakakis and Julian Togelius experiment generating levels for Super MarioBros(Nintendo, 1985) by an extension of the functional scaffolding for musical composition (FSMC) method, this extension translates the pitch of the music to the height at which a tile is placed and the duration of it to which number of times the same tile type is repeated in the same column. [3] This is not the unique example in which Neuroevolution has

been used to generate procedural content in video games, we could generate different content in games: visuals, audio, narrative, rules and mechanics… [21] [22]

Not only procedural content can be generated, we have different approaches to games of Neuroevolution [24]:

- Direct action selection: In some cases, Neural Networks and Genetic algorithms can be used to play videogames. Controlling the NPC though direct-action selection. The Neural Network receive as input the current state of the game or different data from it and choose an action through the outputs. As example we have the implementation of this thesis in which the neural network receives as inputs different sensors and outputs for steering and acceleration/braking.

- Selection between strategies: In some games instead of controlling direct actions of the player take more sense to choose between different strategies to be played for a short time. As an example, a group of three researches [25] experiment with the classic "Keepaway Soccer" game. They evolved Neural Networks to play this game using the NEAT algorithm, but they realized that it works better if they hybridize NEAT with other forms of reinforcement learning.

- Modelling opponent strategy: Neuroevolution can be used to predict the opponent strategy and act in order to play better. In 2008, Alan J. Lockett and Risto Miikkulainen [1] evolved Neural Network to predict the opponent strategy in Texas Hold'em Poker. This experiment results in a win rate increase of agents used this model.

- Modelling player experience: Several types of Neural Networks are usually used as an approximator function. That is the reason why Neural Networks are popular in supervised learning applications. As we saw before, Neural Networks can be trained with backpropagation methods but in some cases Neuroevolution provide a superior performance that other training methods. Preference Learning is one of these cases in which Neuroevolution gives a

superior performance. In Preference Learning, the task is learning to predict an order relation on a collection of objects. Preference Learning through Neuroevolution can model the experience of the player. In 2010, Pedersen, Togelius and N. Yannakakis trained Neural Networks to predict the player preference of different levels in a clone of Super Mario Bros. [8] Training this Neural Networks through Neuroevolution gives a 91% accuracy in the prediction of player preference. Later, this model was used by Shaker, Togelius and N. Yannakakis to evolve personalized levels for the same Super Mario Bros game. [25]

- State/action evaluation: One of the most widespread use of Neuroevolution in games is the evaluation of board positions for an agent in classic board games. Usually an algorithm like MinMax is used to search for future movements and thanks to Neuroevolution the fitness of the future board state is evaluated. In 2004, a group of researchers experiment with a Neural Network that learned to play chess by playing games against itself [10]. Plenty of work have been done on evolving evaluators for different classic games over past years. Othello also called Reversi, has been used by different researchers to experiment with Neuroevolution. Lucas and Runarsson used an evolvable Neural Network based on sampling combinations of board positions with a well performance results in a very good state evaluators learning. [20]. Also, Neuroevolution methods can work even in typical single player arcade games, using Max-tree instead of a Minmax-Tree. As example we have an evolution of state evaluators in Ms. Pac-Man [26] from Lucas. That paper inspired future studies as the work of Borg Cardona, Togelius and Nelson. [6]

5 THE IMPLEMENTATION

The main goal of this thesis consists of an efficient implementation of an Evolutionary Artificial Neural Network, the implementation is collected and

studied in this chapter. The chapter is divided in different sections, we have a total number of 4 sections.

In section 4.1 we discuss which is the objective of this implementation and how it should be done. Section 4.2 summaries all the elements of the implementation and how it works. Section 4.3 will be focused in the main problem that this implementation had. Section 4.4 is dedicated to study the results of the implementation.

5.1 Introduction

In this implementation the main objective was to hybridize Artificial Neural Networks with Genetic Algorithms and implement it in a simulator successfully.

The first thing that we needed for this implementation was a Neural Network Framework and a Genetic Algorithm. The quickest thing should had been, to choose one of the multiple frameworks that can be found in Internet, but with all the research that has been done during this investigation it was easy to build one from scratch and the best way to learn something is to do it by oneself.

No external frameworks were used, and it was done one from scratch in the programming language C#.

After solving the problem with the base of the framework, we needed to find which simulator we will use and after study multiple choices, a car simulator was selected.

The implementation was done using a game engine “Unity 3D”.

5.2 Elements

5.2.1 Car Simulator

The Car simulator is a simple simulator with basic functions.

- Accelerate / Decelerate
- Turn Left/ Turn Right

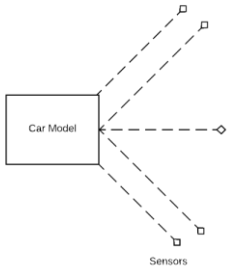


Figure 10 Illustration of the model that was used in the implementation

With this function, is enough to control the car. The car has 5 sensors in different angles to measure several distances to obstacles.

All the cars race in the same race circuit at the same time.

5.2.2 The hybridization

To evolve the Neural Network has been used Neuroevolution. As we studied before we could evolve the Neural Networks in different ways, for this implementation we evolve the weights of the connections between Neurons and the Bias of each layer.

The Network is composed by 4 Layers, the Input Layer has 6 Neurons, the Output Layer has 2 Neurons and the Hidden Layer has 6 and 4 Neurons.

As Input has been used 5 front sensors that each car has plus the own speed of the car. From the Outputs we obtain two values, the first one represents the acceleration/deceleration and the second one represents the steering.

Each Neural Network was encoded using a Weight Encoding technique, using real numbers. This let the Genetic algorithm work with an easier representation of the chromosomes.

5.2.3 Main Steps

Main steps of our implementation:

1. Create initial population of 20 cars with random artificial neural networks
2. Let all units play the game simultaneously by using their own neural networks
3. Once all cars have crash, calculate its fitness to measure its quality for each unit.
4. Apply a replacement strategy using genetic operators.
5. Go back to step 2

5.2.4 Fitness Function

As we want to evolve a population, we should choose the fittest cars. The population must be sorted before each new generation. This order has to be done by checking the quality of each unit.

Usually the fitness function is the way to measure the quality(fitness) of an object. While we have a fitness function to measure the fitness of each car, we can sort and select the fittest cars to reproduce and mutate for the next generation.

In this implementation, each car is rewarded equally to its travelled distance through the race circuit, to select the quickest car we use the average speed over the race. So, in that way, we

difference the cars that travelled the same distance.

To conclude, our fitness function is the sum of the total distance covered by a car plus the average speed.

5.2.5 Activation Function

In this implementation the Tanh Function was chosen because like the Sigmoid Function, it is sigmoidal ("s" shaped) but the outputs range between (-1,1) instead of (0,1). Strongly negative inputs will be mapped to negative outputs. Another advantage is that only zero-valued inputs will be mapped to near zero outputs. These properties will avoid the Neural Network to get "stuck" during the training.

5.2.6 Replacement Strategy

As summary, the best cars will survive, and their children will replace the worst cars in the new generation.

1. The population it is sorted by their fitness.
2. Best 2 cars are selected and pass directly to the new generation.
3. Create 12 child as a crossover of the best two and apply a mutation.
4. Create 2 cars as a crossover of two random previous cars and apply a mutation.
5. Create 4 cars with random genes.

5.3 Results

For this experiment the program was executed five times and each execution data was recollected in different groups (A, B, C, D, E). Each execution saved 10 generations of data, and the best fitness of each generation was collected. Figure 11 visualizes the fitness improves over generation in each group.

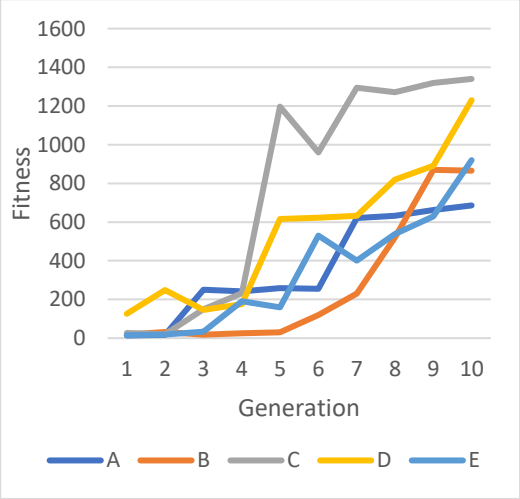


Figure 11 A plot visualizing the fitness improves over generation of 5 random executions.

As we see, each execution of our implementation improves over time and results in an accomplished task.

6 CONCLUSIONS

As we have seen before, Neural Networks and Genetic Algorithms are concepts which share a background of biology.

Artificial Neural Networks emphasize in the individual adaptive learning intelligence while Genetic Algorithms simulate the natural selection characteristics. This combination shows a more complete intelligence.

This hybridization of two different methods with similar background have shown that they perform very well and can solve different types of problems.

There is a big amount of possibilities of future work in Neuroevolution. Every day, the available computational resources grow and with it the search space that can be covered.

References

[1] A. Lockett and R. Miikkulainen: Evolving opponent models for Texas Hold 'Em. 2008

[2] Alan F. Murray: Applications of Neural Networks, pages 1-4, Springer, Boston, MA, USA, 1995

[3] Amy K. Hoover, Paul A. Szerlip, and Kenneth O. Stanley Functional Scaffolding for Composing Additional Musical Voices, 2014

[4] Andrea Soltoggio, Kenneth O. Stanley, Sebastian Risi: Born to Learn: the Inspiration, Progress, and Future of Evolved Plastic Artificial Neural Networks, 2017

[5] Andrej Karpathy,Tim Salimans, Jonathan Ho, Peter Chen, Ilya Sutskever, John Schulman, Greg Brockman, Szymon Sidor: Evolution Strategies as a Scalable Alternative to Reinforcement Learning, 2017

[6] Andrew Borg Cardona, Julian Togelius, Mark J. Nelson: Competitive coevolution in Ms. Pac-Man, 2013

[7] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachangan and Stephen Marshall: Activations Functions: Comparison of Trends in Practice and Research for Deep Learning, 2018

[8] Christopher Pedersen, Julian Togelius, Georgios N. Yannakakis: Modeling Player Experience for Content Creation, 2010

[9] Composing Video Game Levels with Music Metaphors through Functional Scaffolding: Amy K. Hoover, Georgios N. Yannakakis, Julian Togelius, 2015

[10] D. B. Fogel, T. J. Hays, S. Hahn, and J. Quon. A self-learning evolutionary chess program, 2004.

[11] David E. Goldberg: Genetic Algorithms in search Optimization & Machine Learning, The University of Alabama, 1989

[12] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, Alex Kurakin: Large-Scale Evolution of Image Classifiers, 2017

[13] Georgios N. Yannakakis, Julian Togelius: Artificial Intelligence and Games, 2018

[14] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean: Efficient Neural Architecture Search via Parameter Sharing, 2018

[15] Ian Goodfellow, Yoshua Bengio, Aaron C. Courville: Deep Learning, 2015

[16] James Kobielus: www.informationweek.com/big-data/ai-machine-learning/neuroevolution-will-push-ai-development-to-the-next-level/a/d-id/1331405 , 3/30/2018

[17] Jozef Fekiač , Ivan Zelinka , Juan C. Burguillo: A Review Of Methods For Encoding Neural Network Topologies In Evolutionary Computation, 2011

[18] Kenneth O. Stanley and Risto Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation, 2002.

[19] Kenneth O. Stanley: Neuroevolution: A different kind of deep learning, 2017

[20] Lucas and T. Runarsson: Preference learning for move prediction and evaluation function approximation in Othello. 2014.

[21] Noor Shaker, Georgios Yannakakis and Julian Togelius: Towards Automatic Personalized Content Generation for Platform Games, 2010

[22] Noor Shaker, Julian Togelius, and Mark J. Nelson, editors.Procedural Content Generationin Games. Springer, 2016

[23] Samuel, A. L. (1959), "Some Studies in Machine Learning Using the Game of Checkers" in IBM Journal of Research and Development (Volume:3, Issue: 3), p. 210

[24] Sebastian Risi, Julian Togelius: Neuroevolution in Games: State of the Art and Open Challenges, 2014

[25] Shimon Whiteson, Matthew E. Taylor, Peter Stone: Critical Factors in the Empirical Performance of Temporal Difference and Evolutionary Methods for Reinforcement Learning, 2004

[26] Simon Lucas: Evolving a neural network location evaluator to play Ms. Pac-Man, 2005

[27] Stanislav Frolov: Neuroevolution: A Primer On Evolving Artificial Neural Networks: https://www.inovex.de/blog/neuroevolution/ ,2018

[28] Stanley, Kenneth O.; D'Ambrosio, David B.; Gauci, Jason: A Hypercube-Based Indirect Encoding for Evolving Large-ScaleNeural Networks, 2009

[29] Tom Mitchell, McGraw Hill: Machine Learning, 1997.

[30] Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity 1943

[31] Xin Yao: Evolving Artificial Neural Networks ,1999

Documentation of Project

1 User documentation

The use of this prototype is very simple:

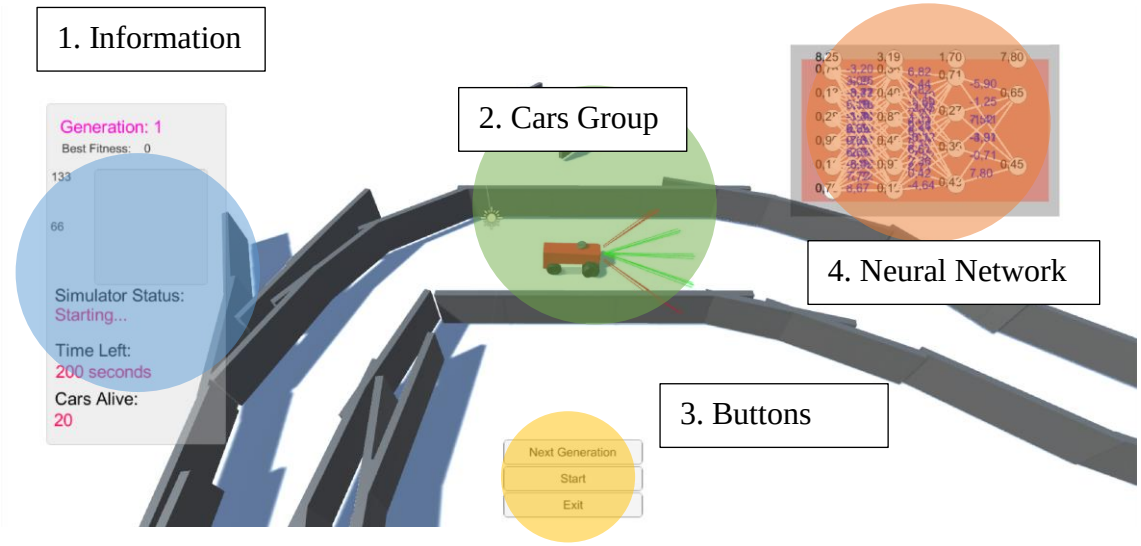


Fig 1: Screenshot of the implementation.

Information View: It shows the information of the current generation and the past generations.

Cars Group: The cars population of the current generation.

Buttons: Buttons to control the implementation.

Neural Network Display: It shows the current topology of the Network.

How to use?

First execute the application, after the execution press the “Start Button” in the lower zone of the screen. The simulation will start, the simulation will change automatically to a new generation if all the cars have crashed. If you want to change the generation manually press the button “New Generation”.

NOTE: For a correct performance, change the generation automatically only if the cars get stuck. The generation will change automatically after 200 seconds.

Car model

The implementation includes different types of cars, each car has a different color representing a type. The are colors are explained in Figure 2.

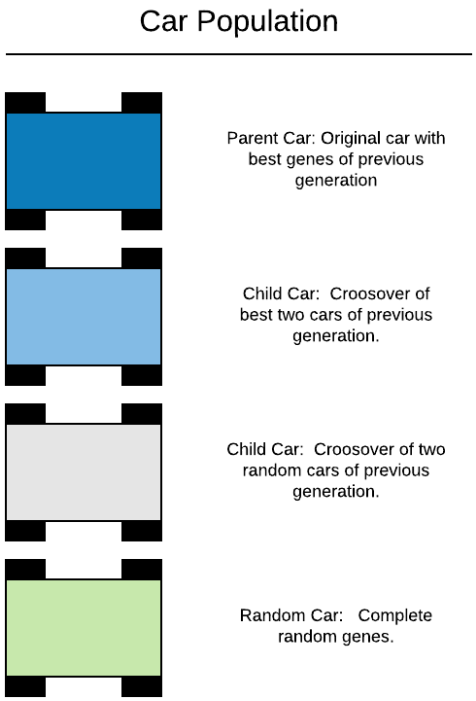


Fig 2: Illustration of the different types of cars.

2 Work Diary

The project was divided in two phases of development, each phase was studied for a correct division of time and work.

As this project consists of 70% prototype and 30% thesis, the big load of work and more effort was given to the prototype.

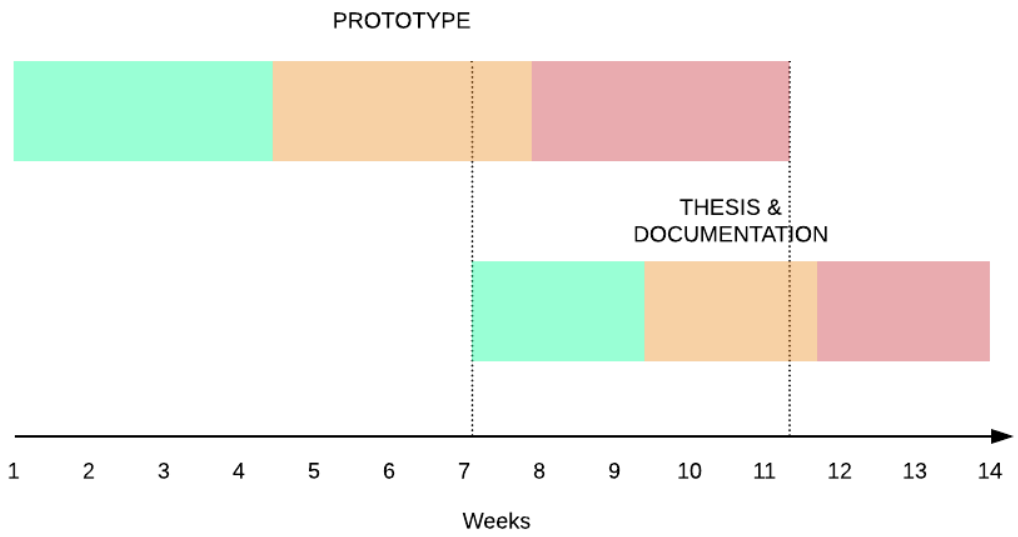


Fig 3: Division of weeks and expected dates.

For better performance and to avoid a waste of time, I divided the time in small objectives for each week. It was very important to have a good organization to accomplish every task.

	PROGRESS UPDATE	**PLANS FOR NEXT WEEK**
1 Week (28-31) March START DAY	This week was only to raise the division of work.	Investigate about Neural Networks and Genetic Algorithms. Search for different types of implementations.
2 Week (1-7) April	In this week, after reading the basics of Neural Networks, I started a project in Unity and I developed the structure of the scripts.	For next week I will work on a basic Neural Network Framework.

3 Week (8-14) April	The Neural Network is completed and its works, the Feedforward works (I tried different Activation Functions)	Read about genetic algorithms and decide if I try to implement the NEAT algorithm or if I only evolve the weights of connections.
4 Week (15-21) April	After having the conflict if I should also try to evolve the topology, I conclude that at first, I will only evolve the weights and if I have time I will work in the NEAT implementation. I coded a basic genetic algorithm that works.	I will implement the genetic algorithm with the neural network.
5 Week (22-28) April	The implementation it is done, and it should work.	I need to code the Car simulator and implement everything together.
6 Week (29-5) April/May	I have some problems with the encode of the Networks. I coded a basic car simulator.	I need to start developing the thesis structure and fix the encoding problem.
7 Week (6-12) May	I started collecting data and structuring the thesis. The car simulator works well but I realized that I have some bugs on the hybridization of neural networks and genetic algorithms.	Fix the bugs of the hybridization.
8 Week (13-19) May	I coded a graphic visualization of the networks. I fixed the bugs and the implementation works well.	I need to search for a good replacement strategy and decide which activation function I will use. I must search the topology of the neural network.
9 Week (20-26) May	After several tests I found a topology that performs well and a good replacement strategy. I Fixed some bugs. I wrote the 10% of the thesis.	I should work more on the thesis and the documentation. I will finish the prototype.
10 Week (27-2) May/June	The prototype is 100% done. I expected that the thesis would be easier, but I was wrong. I will focus completely on the thesis.	Implement NEAT algorithm.

11 Week (3-9) June	I realized that it will be impossible to add the NEAT implementation. 50% of thesis is done.	I need a feedback about the current state of the thesis.
12 Week (10-16) June	My supervisor gave me feedback, and I added more information with higher scientific quality and a state-of-the-art section about Neuroevolution to the thesis. Thesis 90% done and the documentation is 20%.	I need to finish the thesis and the documentation.
13 Week (17-23) June	NO WORK I couldn't work because I went to my bachelor graduation in Spain.	NO WORK
14 Week (24-28) June END DAY	I finished the thesis and documentation.	

3 What I have learned

The aim of this thesis was the extension of my knowledge in this field of studies, Neuroevolution is one of my favorite subfields of artificial intelligence.

I started the project with a very low level of knowledge and I have learned a lot, the current state-of-art of Neuroevolution, how neural networks works, genetic algorithms…

One thing that helped me a lot during the development is the avoid of using external frameworks. Coding all the base of the project from scratch gave me a high knowledge and it upgraded my programming skills with these algorithms.

4 Problems

The biggest problem that I had was trying to do so much work in a small amount of time. When I started the project, I wanted to implement an algorithm to evolve the structure of the neural network over generations, but it was impossible because of the lack of time that I had. Another problem was that I underestimate the difficulty of the thesis part, the language was part of this problem (English is not my native language and this difficult the writing and did it slower).

During the hybridization between neural networks and genetic algorithm appeared a lot of bugs and problems. One important was the encode problem of the network, I was not sure how to encode it and my small amount of knowledge in this field of studies made the development of the prototype in some points more difficult.

5 Solutions

After some hard weeks, I solved all the problems and overcame all the difficulties. Choosing the correct decisions was the key to finish the whole project in time.

The most realistic solution for my lack of time was to aim a lower objective, in this case, instead of evolving the network topology, I would only evolve the weight connections.

Another solution for the slow writing and language problem was to focus completely for the last weeks in the thesis and avoid upgrading the prototype, wasting time in something that was finished.

The solution for the lack of knowledge was very simple and it was solved reading research papers, websites and blog posts.

6 Contribution to my professional future

This project has opened me a big number of doors the world of machine learning, now I know the basics to start investigating more and learning by my own. Neural Networks is one of the most powerful used algorithms in the subfield of machine learning called deep learning, this knowledge will help me to continue learning.

Nowadays Machine Learning and Artificial Intelligence are one of the evolving and blooming fields where someone who is willing to become a Data Scientist or anything similar has a high scope in the job market. This will lead to be part of many innovations and achievements.

Statement of Originality

This is to certify that the content of this project, documentation and thesis is my own work. It has not been submitted for any other degree or other purposes. I certify that the intellectual content of my submission is the product of my own work and that all the assistance received in. Preparing it as well as all sources used have been properly acknowledged.

Signature: