Vedant Patil (UFID:39485964)
Shivani Patil (UFID:53549503)

COT 5405 ANALYSIS OF ALGORITHMS

GENETIC ALGORITHMS: A SURVEY

Abstract

The class of versatile optimization algorithms, genetic algorithms which are inspired by the process of natural evolution in biological organisms have found applications in a wide range of fields, including machine learning, bioinformatics, and many more. We will summarize the basic terminologies of genetic algorithms discussed in "An Introduction to Genetic Algorithms" [1] and "Genetic algorithms in search, optimization, and machine learning" [6], followed by summarizing neural network architecture and how genetic algorithms can be used in neural networks. The paper examines the use of genetic algorithms in the neural network, summarizing hyperparameter tuning, the Breeder genetic programming Algorithm with Occam's Razor, and the NeuroEvolution of Augmenting Topologies (NEAT) algorithm, followed by giving an overview of the future work of GA in neural networks. Overall, this survey paper provides a comprehensive overview of the current state of research on genetic algorithms in neural networks, highlighting both their potential that need to be addressed to effectively implement the solutions for a given task and current trends in the field.

1. Introduction

Genetic algorithms form a set of algorithms which are used for solving both bounded and unbounded optimization problems which take their inspiration from Darwin's theory of biological evolution which suggest that only the fittest survive and natural evolution is extended from them. A genetic algorithm identifies these fit individuals and uses these parents to produce the next generation of parents. This makes sure that the traits of only the fittest parents are passed to the offspring. GAs was discovered by John Holland and was further studied by him and his students in 1960s [1]. His goal was to formally study how can the process of natural adaption be applied to computer systems as it happens in nature. Each solution in GAs is represented as a chromosome, which is evaluated using a fitness function to measure its effectiveness. Chromosomes with the highest fitness scores are selected to reproduce and create the next generation of solutions. This process is repeated for many generations, with each new generation improving upon the previous one. During reproduction, crossover and mutation operations are used to create new solutions. Crossover combines two parent chromosomes to create a child chromosome with traits from both parents, while mutation randomly changes some of the values in a chromosome. By using selection,

crossover, and mutation, the genetic algorithm can search a vast solution space and converge on a high-quality solution. Genetic algorithms have been used to solve problems in many different fields, including engineering, finance, biology, and computer science. In this paper we discuss how genetic algorithms can be used to evolve Artificial Neural Networks.

2. Artificial Neural Networks

Artificial Neural Network (ANN) is inspired but the structure of the human brain and how the human brain work. This network is mainly made up of a network of artificial nodes, these nodes called neurons, which take in information, also exchange data among themselves. For training ANNs, several learning algorithms like supervised learning, unsupervised learning, and reinforcement learning. The error function in ANN gives information about errors between the predicted output of the network and the actual output. The weights (synaptic weight) of the connections between neurons are updated during the training of ANN, to reduce the error. [1], sending errors back through the network is used to train the ANN. [6]

3. Introduction to some Genetic Algorithm Terminologies:

To discuss about Genetic Algorithms, let's first have a look at different basic terminologies involved in developing the Algorithms. [2]

Chromosome: A Chromosome is representation of a solution to a given problem. Generally represented as a string of bits or numbers.

Gene: Each individual bit in a chromosome is called a gene. Chromosomes are made up of a sequence of genes.

Initial Population: It is a subset of all the possible solution to any given problem. Initial Population is like human population. Also called as collection of chromosomes, where each candidate is a potential solution to the given problem also referred to as a chromosome.

Fitness Function: A function that assigns a fitness value to each chromosome in the population. Fitness value represents the quality of the chromosome with respect to the desired solution.

Crossover: The process of combining two or more chromosomes to create a new chromosome. Typically done by combing a set of genes from one chromosome to another.

Mutation: Just like in biology, Mutation means changing the value of a gene in a chromosome with a view of improving its fitness value.

Selection: Process of selecting a set of chromosomes from the population to generate the next generation of chromosomes. The idea is to select the fittest chromosomes and only let their genes pass on to the next generation.

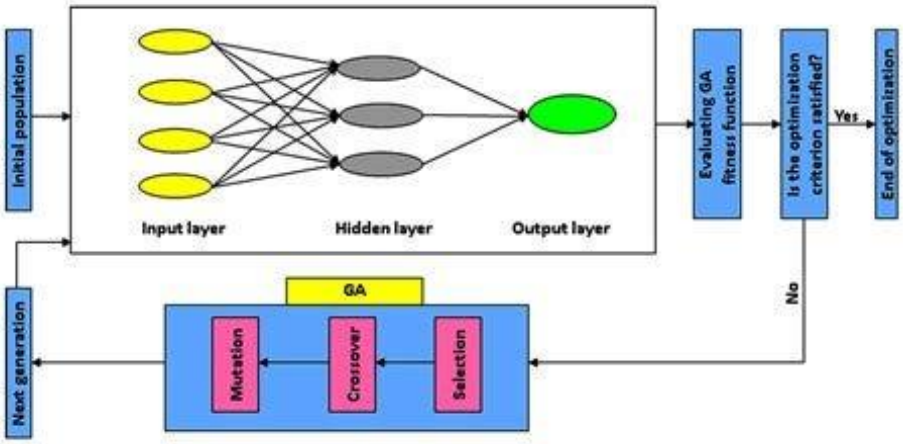
4. Neural Networks with Genetic Algorithms

Hyperparameter tuning in Neural Network is one the most difficult problem to solve in the deep learning space. GA serve as one of the effective means of tuning these hyperparameters. Constructing a multilayered Neural Network involves tuning each neuron to an optimum weight to make an accurate prediction of the problem. The problem is therefore of weight optimization where each set of weights can be represented as Chromosomes and using genetic algorithms, we can find the weights that best fit the network. Let's look at different algorithms for performing this.

4.1 Hyper-parameter tuning using Genetic Algorithm

The following algorithm can set a basic framework for developing any GA for tuning the hyperparameters [3].

- Initialize a population of N neural networks with random weights and biases.
- Tune the hyperparameters with Random Values
- Repeat the following algorithm
    1. Train the neural networks simultaneously or one after the other
    2. Calculate training cost for the population
    3. Calculate the fitness of each neural network based on its cost. The best neural network will have the lowest cost. Thus, fitness would be inversely proportional to the cost. Thus, lowest cost has the highest fitness and therefore more likely to reproduce.
    4. For the number of Neural Networks in the population do the following:
        a. Pick the best two neural networks based on their fitness
        b. Crossover the genes of the 2 neural networks. This child would have some properties of one parent and some properties of the other parent.
        c. Mutate the genes of the child to introduce some randomness in the Genetic Algorithm.
    5. Store the children created in the new population and use them as the initial population for the next iteration.



We run the above algorithm until the optimization criteria is met. At the end of the above algorithm, we would have a population with a Neural Network having optimum hyperparameters.

4.2 Breeder genetic programming Algorithm with Occam's Razor [4]

Let's look at a more complex method to tune these hyperparameters

1) *Network Configuration*

This algorithm discusses a novel approach derived from breeder genetic algorithm (BGA). It makes use of simple hill climbing approach which is less expensive as compared to gradient based method. To employ that we convert our weights into binary.

If we assume a neural network's neuron as a binary device with a threshold, which becomes active if the sum of weighted inputs exceeds the threshold. So given an input vector x, the net input of the i^{th} unit, I_i is computed as follows:

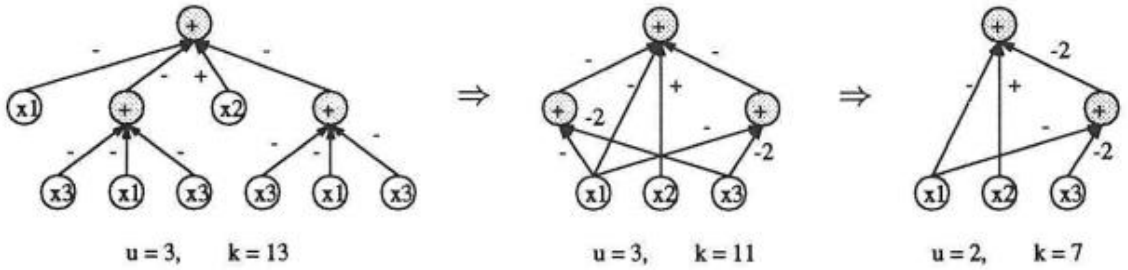
$$I_i = \sum_{j \in R(i)} w_{ij} x_j$$

Where w_{ij} is the connecting weight from j to i and R(i) is the receptive field of unit i. The activation function for neurons is given by

$$f_i(I_i) = \begin{cases} 1 & \text{if } I_i \geq \theta_i \\ 0 & \text{otherwise} \end{cases}$$

Where θ_i denotes the threshold of neuron i and f is the activation function.

This algorithm represents a neural network as a tree. The following figure shows now a tree can be converted into a neural network and its fine tuning.



2) BGP algorithm

The BGP evolutionary algorithm can be seen in the following figure where A denotes a population of M individuals A_i, where A_i is the i^{th} neural network. The initial Population A(0) is generated using random number of layers.

1. Generate initial population $\mathcal{A}(0)$ of M networks at random. Set current generation $g \leftarrow 0$.
2. Evaluate fitness values $F_i(g)$ of networks using the training set of N examples.
3. If the termination condition is satisfied, then stop the evolution. Otherwise, continue with step 4.
4. Select upper τM networks of gth population into the mating pool $\mathcal{B}(g)$.
5. Each network in $\mathcal{B}(g)$ undergoes a local hillclimbing, resulting in revised mating pool $\mathcal{B}(g)$.
6. Create $(g+1)$th population $\mathcal{A}(t+1)$ of size M, by applying genetic operators to randomly chosen parent networks in $\mathcal{B}(g)$.
7. Replace the worst fit network in $\mathcal{A}(t+1)$ by the best in $\mathcal{A}(t)$.
8. Set $g \leftarrow g+1$ and return to step 2.

The algorithm undergoes hill climbing in step 5, where weights are updated. An acceptable solution at the end is determined by the Variance function V(g), where if V(g) <= V_{min}, we terminate the function.

$$V(g) = \frac{1}{M}\sum_{i=1}^{M}\left(F_i(g) - \bar{F}(g)\right)^2 \leq V_{min}$$

Where F(g) is the average fitness in A(g).

3) *Genetic Breeding*

A chromosome is a unique configuration of network's weights. Using truncation selection, the algorithm chooses the individuals with highest fitness scores and performs the hill climbing on these individuals. Hill climbing applies the mutation operator repeatedly to these individuals randomly until no further improvements can be made to the fitness of the network. The mutation operation follows a DFS order of the tree-like structure of the network. DFS defines the order in which the mutations are applied to the tree starting from the topmost layer and then moving down to the subsequent layers below.

4) *Fitness function*

According to the principle of Occam's razor, simpler models should be given preference over complex models. Extending this principle, the fitness function in this algorithm is designed to construct a neural network with minimal complexity. This means the algorithm would prefer a smaller network over a larger one. However, this preference should not be at the cost of the performance of the network, otherwise the function cannot reduce the approximation error, and the network would not be powerful enough to solve the task. The following equation combines both the approximation error in the network and the complexity of the network and formulates the fitness of the network.

$$F(D \mid W, A) = \alpha C(W \mid A) + \beta E(D \mid W, A)$$

Here the first term refers to the complexity of the network and α is the tuning parameter for complexity. The complexity function is defined as follows

$$C(W \mid A) = \sum_{k=1}^{K} w_k^2$$

Where K is the number of adjustable weights. We therefore penalize large weights and hope to achieve smoother mapping by tuning α. The second term in the function is the Error fitting function which is defines as follows.

$$E(D \mid W, A) = \sum_{i=1}^{N} E(y_i \mid x_i, W, A) \quad = \sum_{j=1}^{m} (y_{ij} - o_j(x_i; W, A))^2$$

where, y_{ij} denotes j^{th} component of the i^{th} desired output vector y_i and $o_j(x_i; W, A)$ denotes j^{th} actual output of the network with weights W for the i^{th} training input vector x_i, and β is tuning parameter for approximation error function.

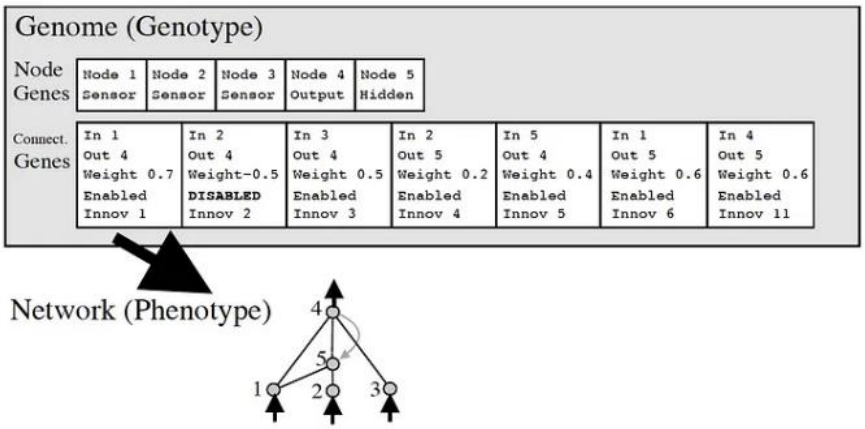
4.3 NeuroEvolution of Augmenting Topologies (NEAT) algorithm

According to research conducted by Ken Stanley in 2002 [9] , the NEAT algorithm is successful because of its distinct features - 1. Combines different ANN structures

through crossover 2. Protects innovation in network structures by grouping similar structures 3. Uses earlier grouped structures to gradually build the simplest structure to solve the task. [7]

1) *Encoding scheme*

NEAT incorporates a direct encoding technique. This technique involves the use of a list-like data structure that contains node and connection genes. This encoding scheme is a little more complex than binary encoding. Information on the connected nodes, weight, enable/disable state, and a historical marker for genealogy is contained in the connection genes. This neural network architecture is directly encoded into a GA chromosome. The following figure is a visual representation of how encoding looks like. [10]



2) *Historical Markers*

All the genes that have the same historical origin have the same structure in a neural network. Each new gene created through the mutation has a unique global innovation number. Genes keep their historical marker when copied to the offspring genome. The same structural components are identified in a different neural network, by comparing the genomes to find genes with the same historical marker. This solves problems related to viable offspring and topological innovations and can accelerate the evolution of neural networks and achieve better performance on the given task.

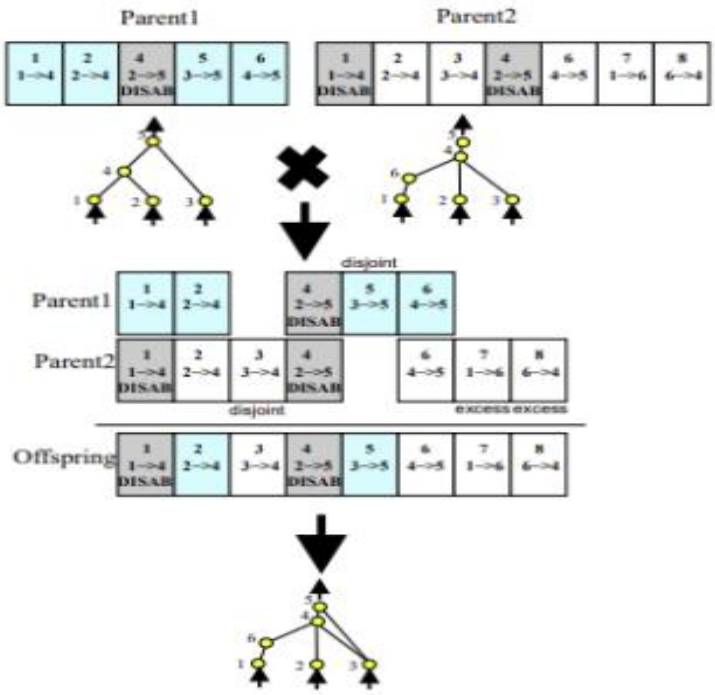
3) *Crossover*

Artificial Synapsis, where genes are aligned using their historical markers is used by the NEAT algorithm during the crossover. There are two types of genes. One, matching genes, which share the same ancestral origin, and others are disjoint/excess genes, which do not share an ancestral origin. Matching genes are

inherited randomly. But in the case of excess and disjoint genes, genes are inherited from a more fit parent. Structural similarities between parent are found using a historical marker, and that doesn't require any expensive topological analysis.

This helps to solve competing convention problems. Organisms that are compatible with a mate in such a way that they preserve the functional subunits. This reduces the probability of producing offspring that have damaged characteristics. Even though there is the possibility of the existence of competing conventions in the population, NEAT avoids the primary consequences of the competing conventions problem by avoiding the evaluation of damaged offspring. [8][10]

In the following example, the historical marking of parent 1 and parent 2 is the same. Offspring is produced by randomly choosing matching genes like 1->4 and disjoint genes are chosen from more fit parents. Here, it is parent 2.



4) *Speciation*

The algorithm groups genomes into species and allows them to compete with the population. The topological discrepancy between the two neural networks is measured by the number of excess/disjoint genes in their genomes.[8] The

following compatible function is used to determine the evolutionary history shared between two genomes.

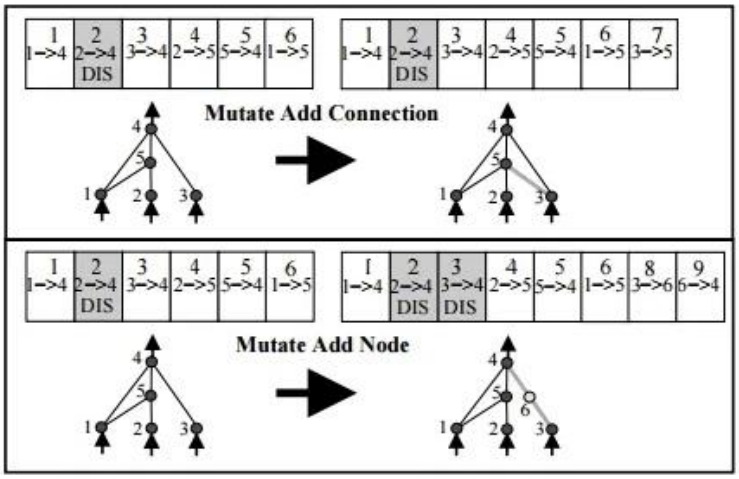
$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}$$

δ - compatibility distance, E - excess genes, D - disjoint genes, W – average weight differences of matching genes. This compatibility is a hard threshold, once the organism crosses it, a new species is created.

5) *Mutation*

- Adding a neuron to the network: This is done by selecting a random connection, then it is replaced by a new neuron and two new connections. To avoid a drastic change in fitness, the weights of the new connections are accordingly selected.

- Adding connection between 2 neurons: Here two neurons are selected at random and then a new connection is added between them. Before adding a new connection, there is a check to ensure that there is no existing connection between selected neurons. The following figure visually represents the above-explained operations:



- Removing a neuron from the network and 4. Removing a connection from the network is performed similarly.

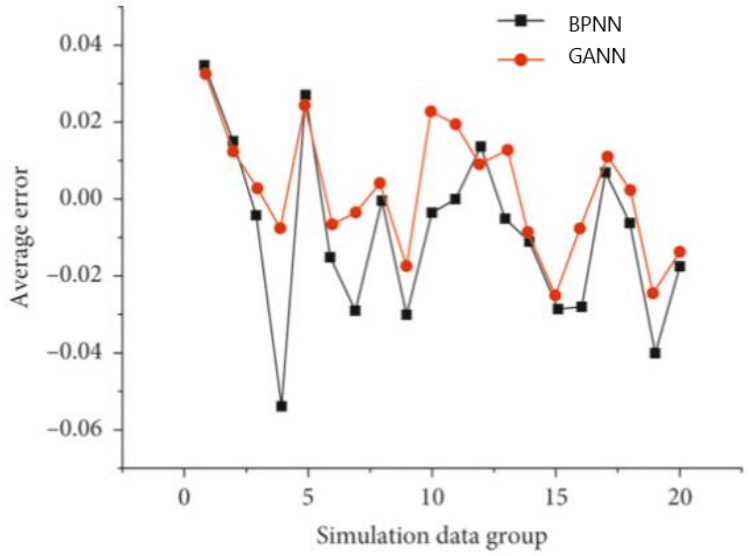
6) *Complexification*

Many genetic algorithms start with a random selection of organisms and topology, but NEAT was designed to start with a minimal network with minimal nodes and connections and then increase complexity only if it turns out to be useful [10]. The NEAT algorithm starts with a few input nodes, output nodes, and

connections between them, there are no hidden nodes at the start. Incorporating this idea with speciation proves to be an efficient way of creating a high-performing but minimal network.

5. Result & Analysis

Following graph [11] is visual representation of comparison of average output error of backpropagation neural network and Genetic algorithm based neural network. Algorithms were executed for 20 sets of data for simulated analysis.



6. Current Trends

Recent research developments in Neural Networks had shifted the current trends of the field towards more efficient optimization techniques. Genetic Algorithms are computationally expensive, especially when the network is large, or the training set is large. However, GAs coupled with hybrid algorithms are still used in numerous applications like evolving network topologies and parameter optimization [5]. Such algorithms use the strength of both algorithms. Genetic Algorithms are also used as augmenting mechanism for super optimization of algorithms in computer vision. Domains like Evolutionary Deep learning and Multi-Objective Optimization use Genetic Algorithms for their structure evolution and use innovative techniques like dominance ranking, diversity preservation etc. for optimization.

7. Conclusion and Future Scope

Genetic Algorithms are well suited for problems with large search spaces and unbounded computation time. They work on minimum assumptions and give good solutions quickly. Their performance in evolution of neural networks, multi-object optimization and genetic programming is very impactful. Future developments in the field of initial sampling methods can help genetic algorithms overcome the overhead spend on initial sampling bias [5]. Genetic Algorithms can further take inspiration from biology to incorporate genal development in some computational form. Recent developments in "Structural Genetic Algorithms" are expanding the visions of Genetic Algorithms and incorporating different biological phenomena in computations. Genetic Algorithms are naturally scalable with parallelism and with increasing computation powers and GPU advancements, the future for Genetic Algorithms seems very exciting as the domains of their applications just keeps on expanding. Fields like meta-learning, learning how to learn have seen groundbreaking developments in genetic algorithms due to which AI based genetic algorithms have become increasingly popular to produce Artificial Intelligence.

References:

[1] Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.
[2] Vijini M. (2017, July 6). Introduction to Genetic Algorithms, including example code. Towards Data Science. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
[3] Suryansh, J. (2018, March 26). Genetic Algorithms + Neural Networks = Best of Both Worlds. https://towardsdatascience.com/gas-and-nns-6a41f1e8146d
[4] Byoung-Tak Zhang, Heinz Miihlenbein (1993) "Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor".
[5] Aymeric V., Alissa M. Kleinnijenhuis and Doyne J. Farmer (January 14, 2021) "Qualities, challenges and
[6] Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addion wesley, 1989(102), 36.
[7] Kearney, W. T. (1996). Using genetic algorithms to evolve artificial neural networks. Colby College.
[8] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), 99-127.
[9] Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From architectures to learning. Evolutionary Intelligence, 1(1), 47-62.
[10] Goyal, A. (2019). NEAT: An awesome approach to neuroevolution. Towards Data Science. Retrieved from https://towardsdatascience.com/neat-an-awesome-approach-to-neuroevolution-3eca5cc7930f future of genetic algorithms: a literature review"
[11] Zhang, J., & Qu, S. (2021). Optimization of Backpropagation Neural Network under the Adaptive Genetic Algorithm. Complexity, 2021, 1718234. https://doi.org/10.1155/2021/1718234