Discovery of Neural Network Weight Update Equations Through Genetic Programming

Tarik Hasan Kurnaz

***Abstract*—This project explores the utilisation of Genetic Programming (GP) to evolve weight change equations for neural network (NN) architectures. The aim is to develop more efficient and effective weight change functions that can yield advanced and precise models across various domains. GP is a machine learning technique that can automatically generate functions by simulating the process of natural evolution. This enables the discovery of innovative algorithms that can outperform existing methods. Different NN architectures were developed such as a single-layered perceptron, a multi-layered perceptron with only one hidden layer and a full NN architecture capable of handling various numbers of hidden layers and nodes. The NN architectures were designed for a regression task, and therefore, the sigmoid activation function was utilised, enabling the usage of the back-propagation function to be used as a basis for comparison in evaluation. By exploring different GP configurations, including those initialised with back-propagation and entirely random equations, our study sheds light on their adaptability and potential. The equations discovered through GP were tested on various datasets. The findings emphasise GP's capacity to excel in diverse scenarios, especially when opportunities for dataset generalisation arise.**

***Index Terms*—Genetic Programming, Optimisation, Neural Network, Back-Propagation**

I. Introduction

In recent years, the field of machine learning (ML) has experienced remarkable advancements due to powerful computing systems and sophisticated algorithms [1]. However, there is still room for growth in some areas of ML where the current state-of-the-art algorithms predominantly rely on manually crafted designs rooted in human expertise, often having their foundations in mathematical principles. While this conventional approach has undoubtedly delivered substantial progress in the field, it may not always yield the optimal algorithm to address the specific demands of a given task [2], [3]. As tasks become increasingly complex and datasets more diverse, it is not always feasible for human experts to handcraft algorithms. Furthermore, mathematical principles, while powerful, can be inherently constrained by the assumptions made during their creation. This can result in algorithms that are sub-optimal for certain tasks and may not fully adapt to real-world data with its inherent variability and complexity.

Genetic programming is a method that utilises evolutionary principles to automatically generate code and improve solutions to complex problems [4]. In GP, computer programs are represented as trees. These trees consist of function sets, which contain mathematical and logical operations, and terminal sets, which include inputs and constant values. GP trees undergo modification via genetic operators: mutation and crossover [5]. Crossover involves two parents swapping sections of the GP tree, while mutation replaces a part of the tree with a randomly generated sub-tree. These actions diversify the search, preventing early convergence to local minima. Individual fitness is assessed through a fitness function, gauging their effectiveness in addressing the problem. Over time, the fittest programs survive and reproduce, passing on their genetic material to future generations. By leveraging GP, tailored algorithms can be created for specific problem domains and datasets, potentially surpassing the performance of handcrafted designs. GP offers the potential to automatically discover and optimise functions for specific cases, removing the need for manual creation.

Although previous work on evolving functions using GP has already been done [6]–[8], this proposal aims to explore the use of genetic programming (GP) as a means to discover functions to represent the weight changes of neural network (NN) architectures [9]. By doing so, this research may simply rediscover the commonly used weight change equation for these architectures. In effect, we can create more advanced and accurate weight change equations, thereby contributing to the advancement of the field. Overall, this project aims to tackle the challenge of manually designing machine learning algorithms by leveraging GP to automatically generate optimal solutions.

The incorporation of artificial intelligence (AI) into various applications has the potential to yield significant environmental benefits [10]. Specifically, the design of ML algorithms using GP offers a range of advantages that align with the principles of environmental sustainability. One primary advantage is the potential for designing more efficient algorithms. These algorithms promote energy efficiency and optimised resource utilisation by reducing computational demands. Furthermore, they reduce the time invested in the algorithm design process, enabling resource-efficient practices. Overall, these factors collectively support the development of machine learning algorithms that align with the principles of sustainable development, making environmentally conscious progress within the field of AI.

This project discovered several GP-generated functions that an NN can utilise the change its hidden layer weights. We examined three distinct configurations. The All Back-Propagation configuration, which initialises the GP with back-propagation equations for all initial individuals, consistently outperforms other GP-generated equations and, in some cases, even traditional back-propagation. The One Back-Propagation configuration, which initialises just one individual with a back-propagation, mirrors this performance and allows for a broader exploration of solution space. The Random Initialisation configuration, which begins with entirely randomly generated

This project was supervised by Andrew Lensen and Marcus Frean.

equations, initially outperforms back-propagation but struggles to maintain performance. By disabling output layer weight changes, GP equations achieve rapid error reduction but lack long-term consistency. The GP's ability to rapidly evolve equations that match dataset characteristics is notable, indicating that it can outperform traditional back-propagation when dataset generalisability is high. In summary, GP, especially with a back-propagation baseline, demonstrates adaptability and potential in various scenarios.

II. Literature Review

a) Evolutionary Optimization of Deep Learning Activation Functions [6]: This paper is a study describing a method for employing GP to develop specialised activation functions for deep neural networks. The authors demonstrate that by exploring a wide range of tree-based candidate functions using techniques like crossover, mutation, and exhaustive search, they were able to discover new activation functions that perform better than traditional ones such as ReLU. Each candidate function is tested for fitness by training a neural network using it as its activation function and seeing how well it performs on the CIFAR-10 and CIFAR-100 image categorisation tasks. The research results demonstrate how the evolved activation functions fully use the potential of meta-learning by performing particularly well for particular datasets and neural network designs. This research sheds light on how deep learning activation functions may be optimised using evolutionary algorithms and creates fresh possibilities for enhancing the functionality of neural networks.

b) Searching For Activation Functions [3]: Introducing a new activation function called Swish, defined as $f(x) = x * sigmoid(x)$, this paper uses automatic search methods instead of evolutionary algorithms. The authors combine exhaustive and reinforcement learning-based search techniques. The goal of reinforcement learning-based search is to maximise the performance of a target neural network on a given task by training a controller neural network to produce new activation functions. The authors use these techniques to find a number of novel activation functions that show promising performance, including the function named Swish [3]. Swish is a good activation function, in the opinion of the authors, because it generates improved outcomes and has traits like smoothness and non-monotonicity. Smoothness is important as it allows the function to have a continuous derivative. Non-monotonicity means that the Swish function is not strictly increasing or decreasing over its entire domain, enabling neural networks to model complex relationships and capture intricate patterns in data such as oscillations, peaks, and valleys, that are not suited to monotonic activation functions. The paper compares Swish to ReLU because ReLU is one of the most widely used activation functions in deep neural networks, and it is a simple and computationally efficient activation function that can help address some problems (vanishing gradient problem) that occur in deep neural networks [3]. The findings of this paper indicate that there is still room for improvement in even the most commonly used machine learning algorithms, and different search techniques might give us better algorithms than those we already possess.

c) Mechanism Discovery and Model Identification Using Genetic Feature Extraction and Statistical Testing [8]: Instead of attempting to find better equations like the previous papers, this paper uses GP as a key component to automatically identify functional forms that approximate the input-output relationship of the system being modelled. The paper proposes a Genetic Feature Extraction and Statistical Testing (GFEST) framework that makes use of GP to automatically recognise models that reflect the fundamental physical, chemical, and/or biological systems generating data. The authors incorporated a library of elementary functions guided by first-principles-based insights and an understanding of the application domain to improve model identification. This approach leads to simpler and easier-to-interpret models that also fit the data satisfactorily. With the aid of GP, this research enables the identification of functional forms that closely resemble the known input-output relationship of the system being modelled.

d) Evolution of Activation Functions for Deep Learning-Based Image Classification [7]: The activation functions (AFs) in neural networks are evolved using a co-evolutionary algorithm in this paper. Co-evolutionary algorithms are a type of evolutionary algorithm that involves evolving multiple populations simultaneously, where each population represents a different aspect of the problem being solved. The authors use Cartesian Genetic Programming, which represents an evolving individual as a two-dimensional grid of computational nodes. The co-evolutionary algorithm used in this study involves evolving AFs from three different layers: input-layer, hidden-layer, and output-layer AFs. Combining three individuals, one from each population, results in an AF architecture that can be evaluated. The authors propose a novel method for co-evolving AFs that outperforms other methods in finding good AFs and architectures. In conclusion, this paper offers insightful information about the significance of AFs in deep learning and presents a promising strategy for enhancing model performance through automated AF searches.

e) Optimizing Deep Neural Network Architecture with Enhanced Genetic Algorithm [11]: This paper introduces an innovative method that employs an enhanced genetic algorithm to optimize the architecture of deep neural networks, encompassing both hyperparameters and architecture optimization. Similarly, in another paper titled Automating Configuration of Convolutional Neural Network Hyperparameters Using Genetic Algorithm [12], genetic algorithms are utilised to optimize the architectures of convolutional neural networks. These studies demonstrate the efficacy of genetic algorithms in addressing the challenge of hyperparameter optimization and architectural design in neural networks. By leveraging the principles of GP, which extends upon genetic algorithms [13], we can further extend the applications to optimize hyperparameters and the components of machine learning algorithms.

A. Summary of Related Work

The analysis of the aforementioned papers offers convincing proof that improving machine learning tasks is still possible due to the superior performance of evolutionary algorithms. Motivated by these findings, our approach involves employing

the power of GP to discover and understand different solutions to change weights in Neural Networks. The research serves as a foundation for further exploration of GP as a valuable tool in creating advanced algorithms for ML tasks.

B. Tools and Methodology

Python is one of the most widely used programming languages for artificial intelligence and machine learning languages [14], providing all the libraries and frameworks necessary for GP. The DEAP Python package [15] serves as a foundation for the GP implementation in this project. The main Python libraries utilised are NumPy [16] and Pandas [17] to handle tables, load files, and create data frames easily. SymPy [18] was utilised to simplify our GP-generated equations. Matplotlib [19] aids in visualising GP runs and investigating equations generated. Given that the development process is linear and follows a step-by-step approach, the methodology being employed is the Waterfall methodology [20].

III. Design and Implementation

A. Overall Process of Our GP

This section aims to simplify the GP process that is vital to our project for individuals who may not have an extensive background in machine learning. To facilitate this, we developed a fundamental perceptron program, enabling the generation and categorization of datasets suitable for prediction using the same classifier. Our primary focus in this GP-driven project centres on the gradual improvement of weight adjustments within the perceptron and other network architectures.

A general flowchart of how GP or other evolutionary algorithms work is shown as Figure 1. We begin by forming an initial population, where each individual is distinct and assigned a fitness value. Afterwards, we select a handful of parents from the previous population and introduce crossover and mutation operations to create a fresh generation. This iterative cycle continues until the specified termination criteria are satisfied.

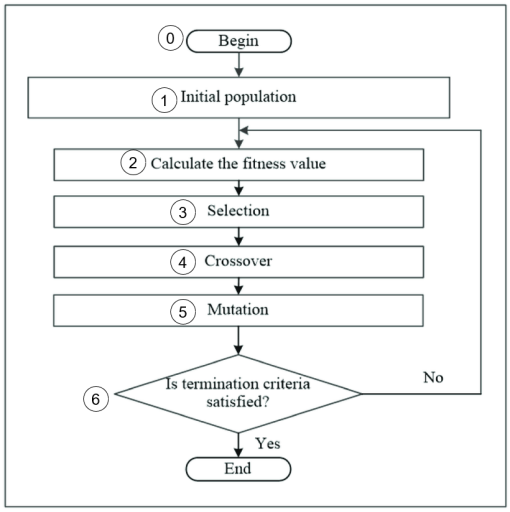

Fig. 1. Flowchart of the Standard Evolutionary Algorithm, adapted from [21]

Following the general evolutionary approach, our GP program to evolve a weight change equation of a perceptron architecture works as follows:

- Step 0: Define the parameters required for initialisation and set them up.
  - Mathematical operations are added to the function set as possible operations that could be performed to grow the tree from the inputs to the output. These operations are addition, subtraction, multiplication, division, and negation, collectively enabling the exploration of a wide array of mathematical functions. This limited set of operations aids GP in narrowing its search to commonly used equations within our network architectures.
  - The terminals, which are the tree's possible inputs, are added. The inputs, the actual outputs, and the desired outputs have all been used as terminals. GP can also utilise random integers selected from the range of -1 to 1 inclusive. By limiting the integers to the range of -1 to 1, we facilitate GP's utilization of numeric values in equations without the need for complex transformations, such as $input/input$ to obtain "1," thereby simplifying the search for effective equations.
  - We create a tree using the previously defined operations and terminals, employing the half-half method for tree generation. In this approach, half of the tree's nodes are exclusively drawn from the set of functions until a specified depth is attained, while the other half incorporates nodes from both the function and terminal sets. This utilization of the half-half method effectively combines the strengths of each approach, achieving a well-balanced result.
- Step 1: Create the initial population.
  - The genetic programming procedure commences with the creation of a substantial initial population. Individuals are represented as randomly constructed trees, which serve as the foundational entities for subsequent evolutionary optimization of weight change equations. Although it can be altered as needed, 300 was selected as it was a large enough number. Every member of this population is a unique tree that will be assigned a fitness score.
- Step 2: Calculate the fitness of each individual.
  - The GP-generated function is used to modify the weights of the network. Achieving a noticeable impact from this equation typically necessitates a substantial number of iterations, often referred to as epochs. Specifically, in our implementation, we use 100 epochs. Since our algorithm uses online learning, the function will be applied to each data point in the dataset individually, for 100 epochs.
  - The actual result for each person is computed using a forward pass method after applying the equation epoch-number of times.
  - The fitness level of each individual is assessed by calculating the root mean squared of errors between

the achieved final result and the target output values. The reason why this performance metric is chosen will be explained in the Evaluation section of the report.

- Steps 3, 4 and 5: Selection, crossover and mutation.
  - To form the next generation of individuals, a tournament selection method is employed. This process entails randomly selecting k number of individuals from the initial population, identifying the best-performing individual, and repeating this procedure until the next generation is formed. Within our program, this k number is 5 to allow GP to select well-fitted individuals.
  - Following the selection phase, individuals undergo crossover and mutation. A crossover probability of 0.6, encourages the creation of diverse offspring, facilitating the exploration of the solution space. A mutation probability of 0.3 serves to maintain a stable population established through crossover, minimizing excessive disruption, while enabling the discovery of potentially advantageous traits. These parameter choices strike a balance between exploration and preservation of effective genetic material. Elitism is also used, preserving the best-performing individual across populations and consistently enhancing overall results [22].
- Step 6: Check whether the termination criterion is satisfied.
  - Upon reaching the predefined number of generations, the evolutionary process halts returning the best individual. If the set number of generations has not been reached, the procedure iterates through steps 2-5 until the predefined limit.

B. Single Layer Perceptron Architecture

As a proof of concept, we started off with a simple perceptron architecture to see if GP could discover the weight changes required in such an architecture. A perceptron is a neural network with just one layer, making it an ideal choice for an initial demonstration that helps conceptualize our future ambitions. The perceptron utilises its inputs to estimate an output without any hidden layers. These inputs are called features and the feature inputs are simply multiplied by the corresponding weights, and the bias is added. Following this, the activation function is applied, and the resulting value is scaled by the learning rate, controlling how fast the algorithm learns. This whole process is called feed-forward and the equation is shown below in Equation 1:

$$output = activationfunction((\sum_{i=1}^{n} W_i * x_i) + b) \quad (1)$$

We wanted to evolve the weight change equation used in the perceptron to update the weights of the network. This equation is shown in Equation 2 and fundamental to the training of perceptrons. We aim to modify and identify the optimal weights to achieve accurate outputs based on input data. The equation is rather straightforward to evolve from scratch as it does not have many parameters. To maintain consistency and avoid introducing arbitrary constants in each run, we chose to set the learning rate to a fixed value of $\eta = 0.1$.

$$\Delta w = \eta(targetY - actualY)x_i \quad (2)$$

A perceptron can only predict data that is linearly separable. Therefore, we utilised a perceptron program to create the dataset, which involves generating many inputs with a certain number of features each, then initialising the weights and the bias randomly. The next step involves applying the feed-forward calculation to each input, resulting in the actual outputs of the set. These pairs of input and output values are then forwarded to the GP section of our code for further analysis. GP component undertakes the task of determining the optimal equation for weight updates. The objective is to minimise the error between the actual outputs and the adjusted outputs, achieved by applying the equation generated to change the weights.

Additional enhancements were introduced to include the bias update within the GP process, as the initial setup did not account for bias changes. This was achieved by assigning a fixed input feature set to one for representing the bias. This allowed us to simplify the equation as $\sum_{i=0}^{n}(W_i * x_i)$, with i starting from zero, and x_0 being set to one. Another improvement was ensuring that each individual within the GP process utilises the same set of weights. This change was motivated by the desire for consistency and the realization that random initialisation of weights might inadvertently result in weights that were already close to the correct values needed to map the input-output relationship. Such a scenario would compromise the accuracy of the weight change equation.

We also used the library SymPy [18] to simplify the equation derived from the top-performing individual at the end of each GP run. Additionally, we developed a supplementary program named Histogram to execute the GP program iteratively and visualize the most frequently occurring equation forms on a histogram graph. The simplification step became essential as GP would often discover identical equation representations. The histogram graph of the GP for perceptron weight change can be viewed in Figure 1 in the appendix [23].

In summary, we determined that GP was suited for the task of discovering effective equations from the ground up to use as weight change equations of a perceptron, as it discovered the weight change equation that is most commonly used as shown in the preliminary report [24].

C. Multi-layer Perceptron Architecture

Following the preliminary report, our aim was to explore the applicability of GP to larger network architectures and assess its ability to discover appropriate weight change equations. However, we encountered a performance bottleneck with GP due to its relatively slow execution. To address this issue, we attempted to parallelise the code. We first enhanced program performance by introducing parallelisation measures. Initially, we allowed the Histogram program to independently generate GP runs. Subsequently, we extended this parallelisation to

the GP level, where we aimed to accelerate the execution of each GP instance by enabling the creation and evaluation of individuals within a generation to occur concurrently, as opposed to the previous sequential approach. To further expedite the process, we leveraged the Rāpoi Cluster [25], enabling the simultaneous execution of multiple program instances to generate equations. These equations were then aggregated to construct a histogram graph for analysis.

Upon resolving the performance challenges, our aim was to extend the original code, initially designed for discovering weight change equations in a perceptron, to encompass a more complex multi-layer perceptron architecture. This expansion necessitated significant alterations in how input and output pairs were generated, as it now entailed the creation of data utilizing a multi-layer perceptron. The reason for this shift was that a single-layer perceptron can only generate linearly separable data, whereas we needed non-linearity to address scenarios where a multi-layered perceptron architecture was required to adjust its weights. In light of this, we introduced a new version of the code capable of generating input-output pairs using a multi-layer perceptron. Additionally, we made corresponding adjustments to the GP to accommodate a multi-layered architecture effectively.

To enable the multi-layer perceptron to learn and adapt, we also incorporated an activation function. In this context, the sigmoid function was chosen as the activation function for its suitability in introducing non-linearity to the network's transformations [26]. While there are other common activation functions available, the choice of the sigmoid function was driven by its prevalence in neural network literature and its ease of implementation [27]. The sigmoid function offers smooth, non-linear behaviour, making it capable of capturing complex relationships in the data and enhancing the capacity of the multi-layer perceptron to handle intricate tasks.

The equation provided, shown in Equation 3 represents the calculation for the weight update for a network using a back-propagation algorithm for networks with sigmoid activation functions. Table I shows the breakdown of the variables involved.

$$
\begin{aligned}
\epsilon_l &= y - a_l && \text{if } l = L \\
&= \sum_{m=1}^{n_{l+1}} w_{l+1}^{mk} \epsilon_{l+1}^{m} a_l') && \text{otherwise} \\
a_l' &= a_l(1 - a_l) \\
w_l^i &\leftarrow w_l^i + \eta \epsilon_l a_l^i \\
b_l^i &\leftarrow b_l^i + \eta \sum_{m=1}^{n_l} \epsilon_l^m
\end{aligned}
\tag{3}
$$

This equation is the back-propagation process in neural networks, used to adjust the weights between each layer. Adapting this equation to our GP implementation posed a challenge due to its varying components based on whether the weights to be adjusted are related to the output layer or the hidden layers. Calculating the output layer's error was straightforward, as there were no layers above it. However, handling the hidden layers proved to be complex to implement, as it required the exchange of information with subsequent layers.

TABLE I
BACK-PROPAGATION NOTATION

ϵ_l^m	error signal at neuron m in layer l
y	target output
a_l^i	output of neuron i in layer l
n_l	number of neurons in layer l
$\sum_{m=1}^{n_{l+1}}$	sum of neurons in layer l + 1
w_l^{mk}	weight connecting neuron m to k in layer l
a_l'	derivate of the sigmoid function at layer l
b_l^i	bias of neuron i in layer l
η	learning rate

In an effort to address this challenge, we implemented a temporary solution by providing the GP with the weight change equation specifically designed for output layer weights. This enabled the GP to focus exclusively on deducing the appropriate weight changes for the hidden layer. Additionally, we supplied the GP with a single parameter containing the output layer error, output layer weights, and output layer outputs, which we computed in advance. This approach eliminated the need for the GP to gather information from upper layers, streamlining its task considerably. Due to the inherent simplicity of the simplified task, our GP implementation consistently identified the simplified back-propagation equation. However, our objective was to enable the GP to discover the complete equation.

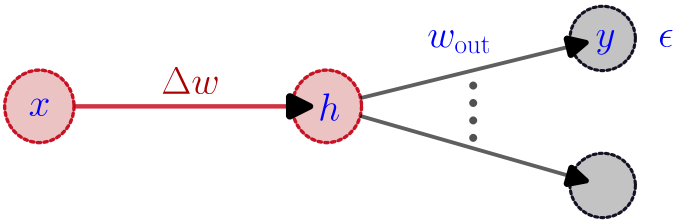

Fig. 2. Diagram Showing Which Information is Available to the Weight Change

Figure 2 illustrates how our GP currently implements the back-propagation equation. Instead of doing a sum of the information from the subsequent layer's nodes, it uses an implicit sum by iterating over the elements of each vector and doing an element-wise summation. Thus, the information available to Δw involves a single output in the subsequent layer, allowing a straightforward multiplication of scalar values of each vector. The blue-coloured variables represent scalar values that serve as terminal nodes to construct the red-coloured weight change. This process continues iteratively, with the weight being adjusted by the information from each subsequent output node until all output nodes are processed.

This approach enabled the GP to determine how to effectively utilise these components. Following the implementation of these code modifications, we observed that the GP was able to find equations where some segments aligned with the back-propagation equation. However, a complete match was not achieved.

D. Problems with Data Creation

To assess the effectiveness of the GP-generated equations, I developed an additional program to simulate the behaviour of the most frequently generated equations. This allowed us to observe how these equations impacted the weights of the multi-layer perceptron and their ability to minimise output errors. The results of these evaluations will be presented and analyzed in the subsequent evaluation section.

We questioned why GP struggled to discover the back-propagation equation or a closely related alternative. To investigate, we developed a program to explore how the outputs of a randomly initialised multi-layer perceptron evolved when a specific input feature was incrementally altered, while other features remained constant. The examination of this alternating feature revealed a relatively linear change in the output. This observation suggested that the hidden layer might not be necessary for mapping input-output pairs, indicating a need for improved data generation that would necessitate the functionality of the hidden layer. To address this, we explored alternative approaches to weight initialisation to induce a non-linear relationship between inputs and outputs in our generated pairs. We experimented with both He weight initialisation [28], typically used for RELU activation functions, and Xavier weight initialisation [29], suited for sigmoid functions. However, even with these weight initialisation methods, the relationship between inputs and outputs largely remained linear. During our exploration, we conducted further experimentation with the Xavier method, adjusting specific parameters. It became evident that increasing a constant within the formula employed for weight creation increased the likelihood of generating non-linear relationships between inputs and outputs.

In conjunction with the modified Xavier initialisation method, we considered increasing the number of input-output sets. Each set represented a distinct configuration of weights, and by employing various sets, we aimed to increase the likelihood of encountering input-output pairs with non-linear relationships. The expectation was that this would influence GP to make use of certain aspects of the back-propagation function. Despite this adjustment, the functions generated for weight updates remained sub-optimal. These functions frequently appeared overly simplistic or, at times, effectively amounted to a 'do-nothing' operation. Consequently, when these functions were implemented within the code to simulate their performance, they consistently failed to yield satisfactory results.

One of the reasons for the sub-optimal equations generated by the GP might be its practice of forming new weights for each individual evaluated by the fitness function. If a randomly generated individual received a set of weights that closely approximated the correct weights required for mapping the input-to-output relationship, it would have no "incentive" to modify its weights. Such occurrences might have been a significant contributor to the GP's difficulty in producing effective functions. To address this concern, we first implemented a measure to ensure that a consistent set of weights serves as the initial configuration for all individuals in each generation. This did not yield significant improvements.

Following the extensive efforts outlined above, we came to the realization that our time and resources might have been better spent by using an appropriate dataset readily available online, rather than investing significant effort into generating our own data. This decision to utilise an online dataset for GP training introduces a set of advantages and disadvantages. Online datasets cover a wide range of real-world scenarios, providing diverse data for training. However, drawbacks include potential misalignment with our experiment's requirements and varying dataset quality, posing challenges in finding an appropriate dataset for our experiment.

E. Neural Network Architecture

Several datasets suitable for our project were identified through research. To assess their linear separability, a small program was developed with the ability to load datasets and subsequently train both single-layer perceptrons and multi-layer perceptrons. This allowed for a direct comparison of their respective accuracies.

While developing this program and seeking suitable datasets, a multi-layered Neural Network (NN) architecture was introduced. This addition was prompted by the consideration that a larger network might yield improved performance on some datasets, especially since the existing architecture supported only a single hidden layer. The newly devised architecture was designed for code conciseness and flexibility, capable of accommodating multiple hidden layers, each with varying numbers of nodes. Following this, the new NN architecture was integrated into the program used to evaluate datasets with different network architectures, aiding in the assessment of their complexity. After evaluation of some datasets, the 2016 Miami Housing dataset [30] was selected as the primary dataset for training. This decision was based on its notable performance improvements when trained on architectures with hidden layers, making it a suitable choice for the project's objectives.

To align with the newly introduced NN architecture, the GP code was modified to effectively accommodate this framework. The adaptation process proved relatively straightforward due to the uniform applicability of the back-propagation rule to all hidden layer weights, irrespective of the specific hidden layer. Consequently, the same formula from the multi-layered perceptron architecture could be applied. However, despite having a dataset well-suited for a neural network, the results were disappointing. These outcomes resembled those observed in the previous multi-layered perceptron architecture, where we had been generating our own data. Frequently, the equations generated by the GP amounted to 'do-nothing' operations or overly simplistic functions.

To address this problem, the weight creation process was expanded to generate multiple sets of weights, rather than a single set. This approach was adopted because utilising multiple sets of weights reduces the likelihood of the initial weights being closely aligned with those required to map the inputs to the outputs of the dataset. While this approach did result in higher computational costs, we deemed it a necessary adjustment to improve our GP's performance.

An alternative approach was investigated by integrating a mechanism within the GP framework to allow it to modify output weights. The hypothesis around this was that given the current correct adjustment of output layer weights, as detailed in Equation 3, the hidden layer weights might not require substantial alterations. This was particularly evident when the number of epochs was set to a high value, as output weights underwent frequent modifications. Consequently, even if sub-optimal functions failed to adapt the hidden layer weights adequately, they could still yield good fitness values because the output weights consistently met expectations and played a significant role in the output result's value change. To address this, we extended the GP's capacity to modify the output layer weights dynamically using its generated equations rather than relying on a fixed solution. However, this adjustment proved to have its own challenges. The GP now exhibited a strong bias toward accurately changing the output layer weights, as this had a more direct impact on the output. As a result, it often neglected the equation required for the proper adaptation of hidden layer weights, leading to sub-optimal results.

Another idea considered was to refrain from making any changes to the output layer weights and instead allow the GP to focus only on modifying the hidden layer weights. This approach aimed to prevent the GP from relying on the correct adjustment of output layer weights or overly prioritizing changes to the output layer. Upon implementing the necessary modifications to the code and conducting subsequent runs, it became evident that this approach yielded the most promising results.

F. Details of the Final Architecture

The final version of the code represents a GP model designed to discover weight update functions of a feed-forward neural network architecture, capable of handling multiple hidden layers, each with varying numbers of nodes. Although it accommodates any number of input nodes, it is designed primarily for regression tasks with only one output node. The project's core goal is to evolve and optimize weight update functions for neural network architectures. In a regression task, the neural network's objective is to predict a continuous numerical output, which mirrors the project's aim of finding functions that optimise and fine-tune the weights to produce accurate predictions. This focus on regression enables the project to provide practical solutions for various applications requiring continuous data predictions, like finance and weather forecasting.

The activation function employed at each hidden and output node is a sigmoid function, although it can be easily modified within the code. To utilise this GP model, one can invoke its main method by providing the data inputs, data outputs, desired hidden nodes, the number of epochs for function application, and the learning rate as input parameters.

The GP will then generate several sets of random weights to serve as initial values, which will subsequently be modified. It is important to note that an individual equation developed by the GP applies to all of these weight sets, not just a single set. This requirement encourages the GP to devise an equation that performs well across multiple sets of weights, enhancing its consistency. The generated equation is exclusively applied to all hidden layers, and the weights of the output layer remain unaltered throughout the process. Moreover, the algorithm offers the flexibility to incorporate specific individuals into its initial population, such as the back-propagation equation, to further aid in the learning process. Upon completing the generational loop, the algorithm simplifies the best equation found and returns it along with its fitness value. Additionally, it provides an estimate of the error that would result if the back-propagation equation were used to rapidly evaluate the performance of the developed function.

IV. Evaluation

In this section, we assess how well our GP has worked in evolving weight change equations for neural networks. To achieve this, we rely on a set of performance metrics designed to offer a quantifiable and objective evaluation of our solution. These metrics not only serve as indicators of success but also serve as benchmarks for comparison against predefined criteria.

A. Performance Metric

To conduct an evaluation of our solution, we utilise root mean squared error (RMSE) as a metric. RMSE serves as a crucial metric to gauge the accuracy of our evolved weight change equations. It quantifies the discrepancy between predicted values and actual values. It is calculated by $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$, where n is the number of data points, y_i is the actual output and $\hat{y}_i$ is the predicted output. A lower RMSE value indicates superior predictive performance. We selected this metric for several reasons: it provides results in the same units as the target variable, offers mathematical conveniences for optimization, aligns with industry standards [31], places emphasis on outliers and our back-propagation equation is the gradient of this error [32]. The choice of RMSE was driven by our desire to gain a clear understanding of the disparities between actual and NN outputs, with a particular focus on penalising outliers by incorporating squared differences. By adopting this methodology, we achieve a precise evaluation of our project's objectives.

Moreover, it is important to highlight that in all the performance evaluations carried out, we have standardised both the input features and the output values in the dataset, ensuring that comparisons between different datasets and experiments are fair, consistent, and easily interpretable.

B. Scatter Plots of Errors over Generations with Different Configurations

In this section, we conduct a comparative analysis of GP runs under various configurations. All of the graphs in this section belong to the run that has generated the equation with the lowest error value across six runs, for each configuration. Each configuration has 100 individuals at each generation and differs based on the number of individuals randomly generated within the initial population of GP. We explore three

distinct configurations, where the number of individuals of each generation is 100,

- **Random Initialisation:** In this configuration, all individuals are initialised randomly.
- **One Back-Propagation:** Here, one of the individuals starts with the back-propagation equation as a foundation.
- **All Back-Propagation:** In this configuration, all individuals are initialised as back-propagations.

The combination of these three configurations covers a range of possibilities, from a strong back-propagation baseline to more diverse and random starting points. This comprehensive approach allows for a deeper understanding of how GP-generated equations perform and adapt under different conditions. In the scatter plots presented below, each blue dot represents an individual within the population. The green line is for the error value associated with the back-propagation equation, while the red line signifies the baseline scenario where the neural network's weights remain unchanged. For each equation, we utilise five sets of randomly initialised weights. The dataset employed for this evaluation is the 2016 Miami Housing dataset [30], as discussed earlier. Notably, we limit the dataset to 100 random instances due to computational constraints, as the full dataset comprises over 10,000 instances, which exceeds the available computational resources within the project's time constraints. The NN architecture was configured to have 13 input nodes, corresponding to the 13 numerical features in the dataset, as it automatically sets its number of input nodes to match the dataset. The network was designed with two hidden layers, each comprising six nodes, as the code automatically selects half the number of hidden nodes for each layer, for two layers.

a) All Back-Propagation Configuration : Figure 3 depicts the configuration in which all individuals within the GP are initialised as back-propagations. This is evident from the fact that all individuals in Generation 0 exhibit the same low error values on the corresponding green line. Figure 2 in the appendix [23], shows the same data in a box-plot format. GP applies mutation and crossover operations in each subsequent generation to introduce diversity. These operations lead to the discovery of functions that, at least for this dataset and sets of initial weights, appear better suited than the initial back-propagation. While there is some variation in error values from one generation to the next, the overarching trend is a steady decrease. In each new generation, the number of functions with fitness values superior to that of the back-propagation increases. Additionally, due to the incorporation of elitism, the best function generated in each generation consistently attains a fitness level equal to or greater than that of the previous generation.

We can see that there is some banding on the error values across generations. This stems from the fact that the GP tends to discover equations that, when simplified, represent the same mathematical expression. This redundancy within the GP-run equations, called intron [5], has no effect on the fitness of the individual. An example of this can be seen in Figure 3 of the appendix [23].

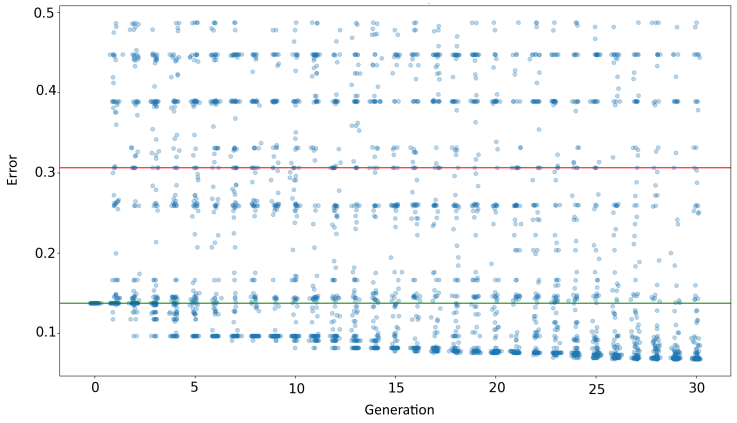

Fig. 3. Scatter Plot of Functions Generated by GP over 30 Generations with their Fitness Values for the All Back-Propagation Configuration

Figure 4 illustrates the run where the GP program starts with the same function, hence having the same depth at generation zero. The trend that the functions discovered are getting simpler initially as the initial function was relatively deep. However, as it progresses, it tends to find more complex functions. This shift is due to the GP's tendency to generate deeper trees, which can represent intricate functions. However, it's crucial to consider the risks of over-fitting and the complexities associated with training and interpreting these deeper trees.

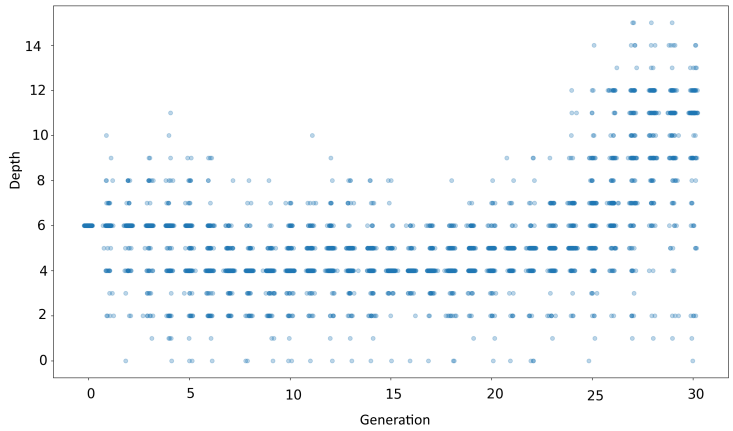

Fig. 4. Scatter Plot of Functions Generated by GP over 30 Generations with their Depths for the All Back-Propagation Configuration

b) One Back-Propagation Configuration: In this GP configuration, only one of the individuals in the initial population is the back-propagation equation, the rest are equations that are generated by the GP. Figure 5 illustrates each individual's error values in conjunction with their respective generations. Notably, Generation 0 exhibits a distinct outlier with a considerably superior fitness value compared to the others. This outlier represents the individual that is the back-propagation equation. This particular individual contributes to our run by providing influential segments that the GP can utilise to construct a new function, given its relatively low fitness, where fitness is inversely related to the error value. The inclusion of the back-propagation baseline provides valuable insight into

the GP's ability to evolve functions that refine the initial reference.

For this GP configuration, similar to the one where all individuals were initialised as back-propagation equations, the first generation has already generated several functions with fitness values outperforming that of the back-propagation equation. This observation holds true for the specific dataset and initial weight sets considered in this context. The comparison between evolved functions and the back-propagation as well as the baseline equation enables an assessment of the GP's capacity to produce weight update equations of higher quality or refinement. Over the course of generations, there has been a consistent decline in error values, indicating an improvement in the evolved functions. Each subsequent generation exhibits an increasing number of functions with fitness values surpassing that of the initial back-propagation reference. This trend underscores the GP's adaptability and effectiveness in generating superior weight update equations for the specific problem as it evolves. The graph depicting the depths of the functions generated in this configuration is provided in Figure 5 of the appendix [23] as the overall trend is similar to the in Figure 4, as the generation number increases, so does the complexity of the equations generated.

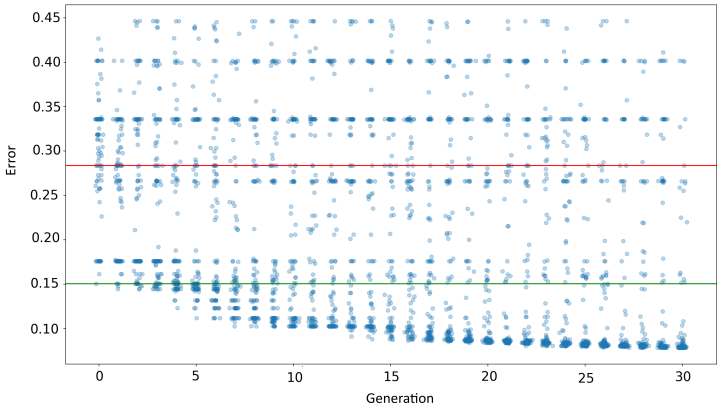

Fig. 5. Scatter Plot of Functions Generated by GP over 30 Generations with their Fitness Values for the One Back-Propagation Configuration

c) Random Initialisation Configuration: In this configuration, all individuals within the GP's initial population are randomly initialised, devoid of any prior knowledge or reference equations. This setting reflects a scenario where the GP starts with no predefined starting point. Through successive generations, mutation and crossover operations introduce diversity and enable the GP to explore an extensive solution space. The absence of constraints from a reference equation grants the GP the freedom to evolve weight update functions entirely from scratch. This configuration serves as a valuable benchmark for assessing the GP's capacity to adapt and generate weight change equations independently. It gauges the algorithm's ability to evolve functions that can outperform back-propagation function and establish an effective neural network training process.

Figure 6 depicts individual error values over time within this random initialisation configuration. Notably, none of the individuals within the initial population align with the back-propagation line. As our GP evolves through successive generations, a noteworthy trend emerges. It becomes evident that the GP is capable of generating equations that exhibit superior performance when compared to the back-propagation equation. In this specific run, the GP successfully identifies an equation that surpasses the performance of the back-propagation algorithm within just four generations. This outcome demonstrates the GP's ability to determine intricate relationships within the dataset and construct functions that perform better in comparison to the back-propagation algorithm. It provides a demonstration of the algorithm's ability to explore solution spaces and evolving functions that enhance NN training processes, all achieved without the reliance on prior guidance or reference equations.

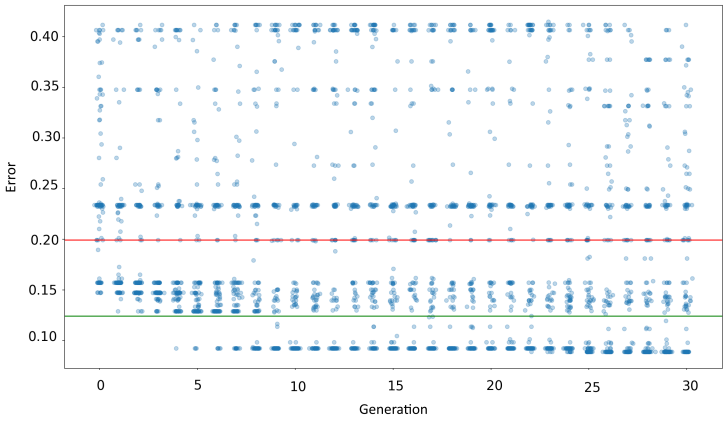

Fig. 6. Scatter Plot of Functions Generated by GP over 30 Generations with their Fitness Values for the Random Initialisation Configuration

It is important to acknowledge that the GP algorithm's performance can exhibit variability across different runs, largely influenced by the inherent randomness in the process. As depicted in Figure 8 in the appendix [23], there are instances where the GP algorithm failed to discover a superior equation to the back-propagation method within the initial 30 generations. This disparity in outcomes can be attributed to the stochastic nature of the GP's operations, including the random selection of dataset instances and the initial random generation of weight sets used for each run.

The observed run-to-run variability highlights a limitation of the GP algorithm: its sensitivity to the specific dataset instances and initial conditions. Unfortunately, mitigating this issue is complex since it's rooted in practical limitations. These limitations are tied to the finite nature of available computational resources, even when employing techniques like multiprocessing and the Rāpoi Cluster [25]. As a result, some level of variability remains unavoidable since running the GP across multiple datasets, data instances, and initial weight sets is not currently feasible. Nevertheless, it is essential to emphasize that, despite the variance in performance outcomes, the overall trend remains consistent. The error values of individuals tend to decrease steadily as the number of generations increases, reflecting the GP's continual refinement and optimization process.

TABLE II
BEST EQUATION OF EACH CONFIGURATION ALONGSIDE BACK-PROPAGATION AND BASELINE FOR COMPARISON

Equation Name	Mathematical Notation, following Figure 2
Back-Propagation (Equation 0)	$x * h * (1-h) * y * (1-y) * \epsilon * w_{out}$
Best Equation of All Back-Propagation (Equation 1)	$x * h * (1-h) * \epsilon * w_{out} * 14$
Best Equation of One Back-Propagation (Equation 2)	$x * h * (1-h) * \epsilon * w_{out} * (h+1) * (y+1) * (y+2) * [-h * \epsilon * w_{out} * x^3 * (\epsilon+1) + h + 2]$
Best Equation of Random Initialisation (Equation 3)	$\epsilon * (y - w_{out}) * (y - x)$
Baseline, No-Change Equation (Equation 4)	0

In cases where the GP algorithm does not immediately outperform the back-propagation method, extending the number of GP generations could yield more favourable results. This aligns with previous observations that suggest the GP, given sufficient time, can discover equations that optimise weight updates, enhancing performance on the dataset and initial weight sets.

C. Investigating the Best Equations

This section evaluates the top-performing GP-generated weight change equations derived from the runs associated with the previously discussed scatter plots. Table II shows the best equations in Figure 2 notation. The red segments within the equations correspond to the parts where the equation matches parts of the back-propagation, whereas the blue parts correspond to non-matching sections. The performance of the GP-generated weight change equations will be measured against the conventional back-propagation for each dataset. We will utilise RMSE as a metric to assess the accuracy of these equations.

In order to thoroughly assess the adaptability and effectiveness of these equations, we applied them to different datasets. Dataset selection plays an important role in evaluating the equations. Therefore, we chose datasets from different fields of real-world context, including housing prices, aircraft control, and bike-sharing systems. The multi-dataset evaluation seeks to demonstrate the practical applicability of the GP-generated weight change equations in diverse real-world scenarios. It investigates whether these equations can effectively contribute to the training of NN for a range of tasks, extending their utility beyond their initial dataset-specific design. Three distinct datasets that have no missing values were selected from various real-world scenarios for this analysis:

— Miami Housing 2016 [30]: This dataset served as the foundation for training the GP and evolving the weight change equations. The price of a house in this dataset can be determined using information such as the land area, distance to various parts of the city, age of the structure and other factors.

— F16 Aircraft Elevator Control Actions [33]: This is a test dataset concerning actions taken on the elevator component of an F16 aircraft. Employing the GP-evolved equations on a separate dataset serves as a test of their adaptability and capacity for generalisation.

— Capital Bikeshare System, Washington D.C., USA, 2011-2012 [34]: Another test dataset, centring around bike-sharing systems, providing comprehensive information on travel patterns, weather conditions, and other factors relevant to the total rental bike count.

a) Longer Runs on Miami Housing 2016: An evaluation was conducted that involved extending the number of training epochs to 1000 for the neural network using the training dataset, Miami Housing 2016, where GP-generated equations were employed as weight update mechanisms.

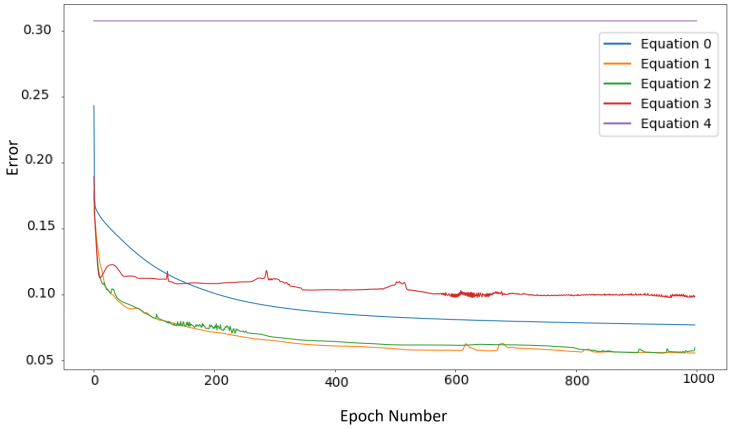

Fig. 7. Equations, as shown in Table II, ran on Miami Housing 2016 dataset [30]

The plots in Figure 7 provide a visual representation of the performance of these equations. The absence of error reduction in Equation 4, despite its exclusive application to hidden layer weights, stems from our choice of keeping the output weights unchanged. This measure is taken to isolate the impact of the equations, specifically designed for the hidden layers, allowing us to assess their performance independently. Equation 0, the back-propagation equation, is used as a reference point to assess how well the GP-generated equations perform. The back-propagation equation shows a high level of consistency when compared to the GP-evolved equations, which often show more variability and unpredictable behaviour.

For the GP-generated equations initialised with back-propagation as a reference in the initial generation, we observe significantly lower error values compared to pure back-propagation. Notably, the All Back-Propagation configured equation outperforms the other slightly in terms of lower error values. This indicates that the GP-generated equations that are trained on this dataset, often perform in ways that generalise

for the dataset. However, further evaluation of additional datasets will provide a more comprehensive understanding of their adaptability and real-world applicability.

In contrast, the equation generated under the Random Initialisation configuration, exhibits a strong initial performance, surpassing the back-propagation function. However, as the number of training epochs progresses, these equations reach a plateau, and their performance levels off. This behaviour can be attributed to the GP's limited evaluation period, where each individual equation is assessed for only 100 epochs within the GP. While Equation 3 may exhibit superior performance during this limited time frame, back-propagation eventually surpasses it in the long term. This dynamic underscores the importance of continuous assessment and refinement to ensure that GP-generated equations remain effective across various scenarios.

b) F16 Aircraft Elevator Control Actions: To evaluate how these equations perform on a different dataset, we conducted a similar analysis to the one above using the F16 Aircraft Elevator Control Actions [33] as a test dataset.

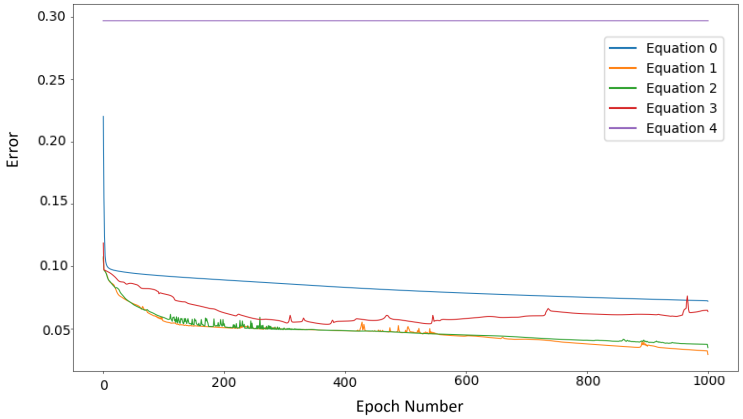

Fig. 8. Equations, as shown in Table II, ran on F16 Aircraft Elevator Control Actions dataset [33]

Figure 8 provides a visual representation of the performance of the GP-generated equations. We have maintained the same colour coding as in the previous section for consistency.

Similar to the Miami Housing 2016 dataset, the back-propagation equation follows a more consistent downward trend without significant fluctuations. In contrast, the GP-generated equations exhibit lower error rates, yet they display somewhat unstable behaviour, occasionally experiencing jumps. As observed in the previous dataset, the All Back-Propagation configuration demonstrates slightly superior performance, reinforcing its position as a robust reference. Notably, for the previous dataset, the equation from the Random Initialisation configuration was surpassed by the back-propagation equation after approximately 180 epochs. However, for this dataset, back-propagation struggled to surpass this configuration, except for a random error spike of the Random Initialisation equation.

These observations suggest that while these equations were not specifically evolved using this dataset, they can still yield improved results, hinting at their potential for generalisation. This implies that GP-generated equations hold promise for optimising the weight updates of various neural networks, extending beyond network designs tailored for specific datasets.

c) Capital Bikeshare System, Washington D.C. 2011-2012: Capital Bikeshare System [34] was used as the other test dataset to execute a comparable analysis following the same methodology.

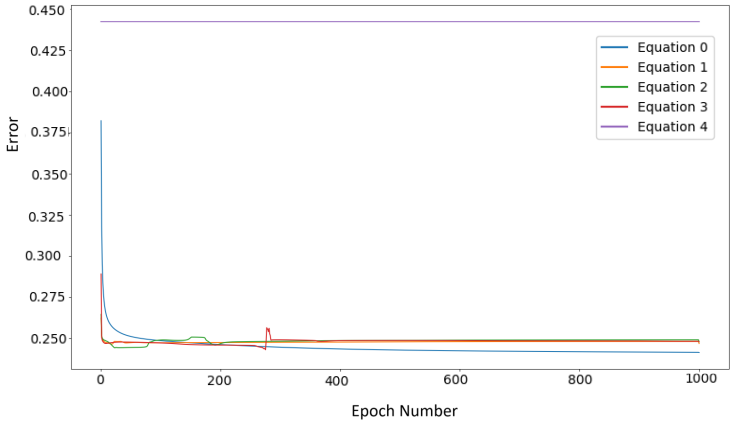

Fig. 9. Equations, as shown in Table II, ran on Capital Bikeshare System, Washington D.C. 2011-2012 dataset [34]

Figure 9 shows the performance of the GP-generated equations. Similar to the other two datasets, the back-propagation equation exhibits a relatively steady decrease in error, maintaining a more consistent trend. Contrarily, the GP-generated equations show less fluctuation this time and appear to produce error values quite close to each other. The back-propagation equation starts to outperform them around epoch number 100 and fully overtakes them around the 220th epoch.

The error values depicted on the y-axis emphasize the challenging nature of this dataset, as the RMSE error values exhibit a limited reduction, struggling to attain values significantly below 0.250 for any of the equations. This indicates the complex nature of the relationship between input and output in our neural network, reflecting the limitations imposed by our fixed neural network architecture. Specifically, the predetermined number of hidden layers and nodes within our algorithm remains unchanged. Modifying this architecture by varying the number of hidden nodes would introduce additional complexity, necessitating extensive testing to determine the optimal configuration for each dataset. However, such an undertaking exceeds the scope of our current project. Moreover, modifying the input-to-hidden node ratio for each dataset would render the error values incomparable across datasets, as the network structures would differ significantly.

The back-propagation equation, being part of a more traditional and straightforward training approach that is mathematically derived, is better suited to tackle complex datasets like this one. Back-propagation, as it is the gradient of the error function [32], can gradually refine the neural network's internal parameters to align with the dataset's complexities. This adaptability, especially in the presence of intricate and challenging data, contributes to the superiority of back-propagation over the GP-generated equations in this specific

scenario. The lack of fine-tuned adjustments could lead to the GP-generated equations falling short of achieving the same level of performance as back-propagation in challenging datasets, especially when the equations are not generated for that particular dataset.

D. Overall Evaluation

The evaluation of GP-generated weight change equations reveals the potential for them to excel on specific datasets. However, their performance exhibits variations between different datasets, which highlights the need for a more in-depth analysis of their adaptability and consistency.

Among the evaluated equations, the All Back-Propagation configuration's equation stands out as a notable performer. It consistently achieves lower error rates compared to other GP-generated equations. It is noteworthy that in two out of three datasets, including both a test and a training dataset, this configuration outperforms the back-propagation method. This performance underlines the potential of leveraging back-propagation as a foundation for GP-generated equations in certain scenarios, where it can outperform the traditional back-propagation method.

The performance of the One Back-Propagation configuration equation closely mirrored that of the All Back-Propagation configuration, with their error values consistently remaining within close range of one another. This observation suggests that the practice of initialising all individuals in the initial generation with back-propagation equations may not be necessary. In fact, it raises the possibility that allowing randomly generated initial individuals could contribute to a broader exploration of the solution space during the earlier generations. This, in turn, may lead to more diverse solutions in the later stages of the evolutionary process, further supplementing the search for optimal equations.

The Random Initialisation configuration equation displayed a distinct pattern in its performance. In the initial stages of the run, it frequently outperformed the back-propagation method, achieving lower error values within the first 100 to 200 epochs. However, as training of the network progressed with each epoch, the back-propagation equation's error values began to drop more consistently with its smoother convergence, a trend observed in the training dataset and one of the test datasets. Interestingly, in the second test dataset, the equation generated by the Random Initialisation configuration maintained its superior performance even after 1000 epochs, which was the maximum duration of our testing. The favourable aspect of the Random Initialisation configuration producing equations that align well with certain datasets underlines the adaptability of the GP algorithm in diverse scenarios. It is worth noting, though, that while the Random Initialisation configuration displayed strong potential for certain datasets, the other two configurations that relied on back-propagation as a foundation consistently generated equations that better suited all three datasets. This outcome suggests that while the Random Initialisation configuration can deliver competitive results, the GP may benefit from the foundational support provided by back-propagation, especially in ensuring consistent performance across different datasets.

One plausible explanation for why GP-generated equations seem to be better performing is because of our intentional prevention of changes in the output layer weights. This constraint forces the GP to provide equations that lead to a faster error reduction, showing that there is a trade-off between rapid error reduction and long-term consistency. This suggests there is more work that needs to be done to understand the types of datasets these GP-generated functions are better suited for. A possible setting of an NN can be to utilise our GP-generated equations for the first few 100 epochs and then swap to use the back-propagation equation. This should result in rapid error reduction in earlier epochs and then a steady convergence to achieve better results without the fluctuations of the GP-generated equations.

A notable observation in this context is that within the configurations initialised with back-propagation, the first generation already includes individuals with superior fitness compared to back-propagation. This observation underscores the GP's ability to rapidly evolve equations that closely match the dataset's unique characteristics. By using back-propagation as a starting point, the GP can craft equations that are well-suited to the dataset, outperforming the back-propagation method which uses the gradient of the loss function [32], which is our fitness value. In scenarios where the dataset exhibits generalisability, the GP has the potential to discover highly effective equations or independently rediscover the principles of back-propagation from the Random Initialisation configuration.

V. Conclusion

The primary objective of this project was to explore the potential of GP to evolve efficient weight change equations for NN architectures. By doing so, it seeks to potentially rediscover and enhance commonly used weight change equations, automating their design process. The NN architecture implemented is a feed-forward network architecture capable of handling multiple hidden layers, each with varying numbers of nodes, a sigmoid activation function is employed at each hidden and output node. This architecture is designed for regression tasks. Three different configurations were used to generate functions: the All Back-Propagation configuration, which initialises the GP with back-propagation equations for all initial individuals; the One Back-Propagation configuration, which initialises just one individual with a back-propagation; and the Random Initialisation configuration, which begins with entirely randomly generated equations. In the evaluation phase, six runs were executed for each configuration, where each individual was used to update the weights of the network for 100 epochs. Subsequently, the runs that generated the best equations from each configuration were compared using scatter plots depicting each individual's errors across generations. These best equations were later run on distinct datasets to visualise their performance over 1000 epochs where we utilise RMSE as the performance metric. One of the three datasets was utilised to train our GP.

Utilising at least one back-propagation in the initial generation of the GP resulted in discovering equations that are better suited for two out of three datasets when compared

to back-propagation. The equation generated by the Random Initialisation configuration was found to initially outperform back-propagation but struggled to maintain performance. This occurrence can be attributed to our intentional limitation on altering output layer weights within GP-generated equations, driving the GP to prioritise rapid error reduction, and revealing a trade-off between rapid error reduction and long-term consistency, posing a challenge for GP-based weight update equation optimisation.

Our findings indicate that the GP algorithm, especially when supported by a strong baseline like back-propagation, can quickly adjust and generate equations that closely fit the dataset's needs. This adaptability highlights the GP's potential to perform well in different situations, especially with datasets that offer greater potential for generalisation, emphasizing the requirement for additional exploration and experimentation to unlock the complete capabilities of this algorithm in diverse scenarios. While the GP exhibits promise in terms of adaptability, ensuring its stability and consistency represents crucial areas for improvement and future investigation.

Building on the results from this evaluation, there are several avenues for future work that could be investigated. One promising direction involves enhancing the generalisability of the GP-generated equations. This could be achieved by identifying additional datasets from various real-world domains or by utilising multiple datasets simultaneously, helping to mitigate the risk of over-fitting to one dataset. The current code structure allows for the utilisation of multiple datasets.

Another future direction is also exploring the impact of modifying the predetermined number of hidden layers and nodes as the current runs only covered two layers with each having half the input nodes, resulting in a relatively small network. This exploration can facilitate the GP in discovering more relevant weight update equations suitable for specific datasets.

Other future directions include expanding the function set used in GP to include more mathematical functions such as logarithm, sine, square root, etc., allowing for a wider search area. This bigger search area requires increasing the population size or the number of generations within GP. However, this increase means that GP will take a longer time to explore the possibilities, resulting in a computational cost increase. Another direction is investigating the usage of classification tasks within the NN architecture with appropriate datasets.

As we have already tried integrating the output layer weight changes within GP or hard-coding them to be correct, and it did not yield satisfactory results, we propose the usage of co-evolutionary techniques [35] to simultaneously evolve weight change equations for both hidden layers and the output layer. This approach can help create equations that work together in unison across the entire network.

An additional potential direction is investigating the feasibility and effectiveness of a hybrid approach that combines GP-generated equations for the initial training 100 epochs with subsequent epochs utilising back-propagation, which can potentially improve convergence and performance.

This project delved into GP's ability to evolve functions with a primary focus on discovering effective weight change equations for NN architectures. This research aimed to streamline the process of equation design, focusing on a feed-forward network architecture's weight changes, suited for regression tasks. Our GP was able to generate effective weight change equations that have yielded better results than the reference point, back-propagation, within our experiments. These findings show promise as this project can be used to tackle the challenge of manually designing machine learning algorithms by utilising GP to automatically generate optimal solutions. Several future directions have also been proposed to further increase the effectiveness of this methodology for the task of evolving weight change equations.

REFERENCES

[1] K. Sharifani and M. Amini, "Machine learning and deep learning: A review of methods and applications," *World Information Technology and Engineering Journal*, vol. 10, no. 07, pp. 3897–3904, 2023.

[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *CoRR*, vol. abs/1611.01578, 2016.

[3] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[4] R. Poli and J. Koza, *Genetic Programming*, pp. 143–185. Boston, MA: Springer US, 2014.

[5] L. Vanneschi and R. Poli, *Genetic Programming — Introduction, Applications, Theory and Open Issues*, pp. 709–739. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[6] G. Bingham, W. Macke, and R. Miikkulainen, "Evolutionary optimization of deep learning activation functions," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 289–296, 2020.

[7] R. Lapid and M. Sipper, "Evolution of activation functions for deep learning-based image classification," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2113–2121, 2022.

[8] A. Chakraborty, A. Sivaram, L. Samavedham, and V. Venkatasubramanian, "Mechanism discovery and model identification using genetic feature extraction and statistical testing," *Computers & Chemical Engineering*, vol. 140, p. 106900, 2020.

[9] K. S. Ganesh, "What's the role of weights and bias in a neural network?." https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f, Sep 2022. [Online; accessed 25-May-2023].

[10] A. sbai, "Ai and environmental sustainability." https://infomineo.com/ai-and-environmental-sustainability/, Jan 2023. [Online; accessed 22-May-2023].

[11] A. Shrestha and A. Mahmood, "Optimizing deep neural network architecture with enhanced genetic algorithm," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 1365–1370, 2019.

[12] F. Johnson, A. Valderrama, C. Valle, B. Crawford, R. Soto, and R. Ñanculef, "Automating configuration of convolutional neural network hyperparameters using genetic algorithm," *IEEE Access*, vol. 8, pp. 156139–156152, 2020.

[13] M. Fatehnia and G. Amirinia, "A review of genetic programming and artificial neural network applications in pile foundations," *International Journal of Geo-Engineering*, vol. 9, no. 1, 2018.

[14] A. Ryabtsev, "8 reasons why python is good for artificial intelligence and machine learning." https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/, Dec 2022. [Online; accessed 22-May-2023].

[15] DEAP, "Deap documentation - deap 1.3.3 documentation 2023." https://deap.readthedocs.io/en/master/index.html. [Online; accessed 23-May-2023].

[16] "Numpy." https://numpy.org/, 2023. [Online; accessed 01-Oct-2023].

[17] "pandas." https://pandas.pydata.org/, 2023. [Online; accessed 01-Oct-2023].

[18] "Sympy." https://docs.sympy.org/latest/index.html, May 2023. [Online; accessed 31-May-2023].

[19] "Matplotlib: Visualization with python." https://matplotlib.org/, journal=Matplotlib, 2012. [Online; accessed 30-May-2023].

[20] A. C. Team, "Waterfall methodology: Project management." https://business.adobe.com/blog/basics/waterfall, 2022. [Online; accessed 26-May-2023].
[21] M. A. Albadr, S. Tiun, M. Ayob, and F. AL-Dhief, "Genetic algorithm based on natural selection theory for optimization problems," *Symmetry*, vol. 12, no. 11, p. 1758, 2020.
[22] R. Poli, N. F. McPhee, and L. Vanneschi, "Elitism reduces bloat in genetic programming," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, (New York, NY, USA), p. 1343–1344, Association for Computing Machinery, 2008.
[23] T. H. Kurnaz, "Final report appendix." Appendix for Discovery of Neural Network Weight Update Equations Through Genetic Programming.
[24] T. H. Kurnaz, "Discovery of neural network weight update equations through genetic programming preliminary report." Preliminary Report of this project.
[25] "Rāpoi cluster documentation." https://vuw-research-computing.github.io/raapoi-docs/. [Online; accessed 02-Oct-2023].
[26] H. Pratiwi, A. P. Windarto, S. Susliansyah, R. R. Aria, S. Susilowati, L. K. Rahayu, Y. Fitriani, A. Merdekawati, and I. R. Rahadjeng, "Sigmoid activation function in selecting the best model of artificial neural networks," *Journal of Physics: Conference Series*, vol. 1471, no. 1, p. 012010, 2020.
[27] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, "A survey on modern trainable activation functions," *Neural Networks*, vol. 138, pp. 14–32, 2021.
[28] J. Brownlee, "Weight initialization for deep learning neural networks." https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/, Feb 2021. [Online; accessed 07-October-2023].
[29] "What is xavier initialization?." https://365datascience.com/tutorials/machine-learning-tutorials/what-is-xavier-initialization/#h_24242636975541686829817569, Jun 2023. [Online; accessed 07-October-2023].
[30] L. Grin, "Miamihousing2016." https://www.openml.org/search?type=data&sort=runs&id=44147, 2022. [Online; accessed 11-October-2023].
[31] T. O. Hodson, "Root-mean-square error (rmse) or mean absolute error (mae): when to use them or not," *Geoscientific Model Development*, vol. 15, no. 14, pp. 5481–5487, 2022.
[32] J. Brownlee, "Difference between backpropagation and stochastic gradient descent." https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/, Jan 2021. [Online; accessed 11-October-2023].
[33] L. Grin, "elevators." https://www.openml.org/search?type=data&sort=runs&id=44134, 2022. [Online; accessed 11-October-2023].
[34] L. Grin, "Bike sharing demand." https://www.openml.org/search?type=data&sort=runs&id=44142, 2022. [Online; accessed 11-October-2023].
[35] M. Aichour and E. Lutton, *Cooperative Co-evolution Inspired Operators for Classical GP Schemes*, pp. 169–178. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.