Evolving Architecture of Feed-Forward Neural Networks Using Genetic Algorithms

Bhargav Joshi[1], Gerry Dozier [2]

***Abstract*— This paper presents an approach for finding an optimum architecture of a simple feed-forward neural network to improve its accuracy. The architecture of a neural network can be considered as the structure of an artificial brain. Genetic algorithms (GA) have been used to find the best solution by replicating the evolution process that occur in nature. The application of the concept of genetic evolution to evolve this "artificial brain" results into an ideal architecture of hidden layers for a neural network that has improved accuracy.**

I. Introduction

Neural networks have been the core component in deep learning. The flexibility to develop complex and layered architecture of neural network has opened many opportunities to solve complex learning problems that involve a large number of patterns to be learned. Neural networks are mighty as nonlinear signal processors, but obtained results are often far from satisfactory [1], meaning that there are a few challenges related to neural network's architecture and training algorithms that need to be addressed to make neural network applications successful. The following are the critical challenges to neural network applications [1]:

1. Which neural network architecture should be used?
2. How vast should a neural network be?
3. Which learning algorithms are most suitable for a chosen architecture?

Researchers often use trial and error approach to find the suitable size of a neural network architecture. In this paper, an idea is proposed to use a genetic algorithm to perform the trial and error to find a good architecture that yields good accuracy.

The remaining of this paper is structured as follows: Section II explains multi-layer percepetron networks (MLP), genetic algorithms, flat crossover method, $\mu + 1$ replacement strategy, keras deep learning library, and the datasets used in the experiment. Section III provides the experiment setting, Section IV provides the architectures achieved using the genetic algorithm, and Section V concludes the paper and our achieved findings.

II. Methodology

A. Multi-Layer Percepetron Network (MLP)

Multi-layer percepetron (MLP) networks (also known as simple feedforward neural networks) consist of the layered architecture of percepetron neurons. They have many perceptrons that are organized into layers. Figure 1 shows a 5-3-4-1 MLP network which is interpreted as a network having a layer of 5 inputs and a layer of an output connected through two hidden layers of 3 neurons and 5 neurons respectively. Unlike the early age perceptron discussed above, percepetron neurons in MLP can have different activation functions such as sigmoid functions, linear functions, softmax functions, rectifier linear units[2][3]. Since MLPs are fully connected, each node in one layer connects with a certain weight w_{ij} to every node in the following layer.

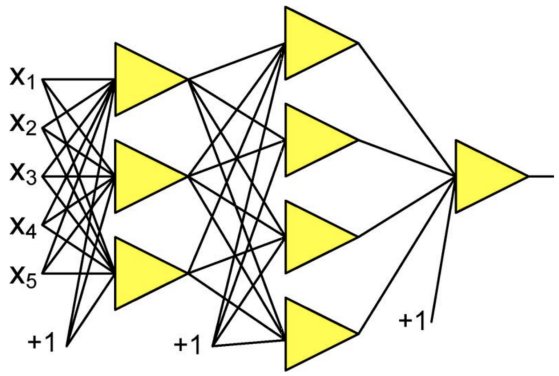

Fig. 1. MLP network with 8 neurons and two hidden layers [4]

The degree of error in an output node j at the nth data point can be denoted as,

$$e_j(n) = d_j(n) - y_j(n) \tag{1}$$

where, d_j is the desired output value and y_j is the value produced by the percepetron. The weights at each node are targeted be adjusted such that the total error in the entire output is minimized. This error function is computed as,

$$\mathscr{E}(n) = \frac{1}{2}\sum_j e_j^2(n) \tag{2}$$

The computation of new weights at each node to minimize the error function is performed through learning algorithms. Some of the examples of training algorithms include error-back propagation (EBP) [1],scaled conjugate gradient descent (SCG) [5], levenberg-marquardt (LM) algorithm [6].

This is why the number of hidden layers and the number of neurons in each hidden layer contributes to how a neural network solves a problem. Hence, it is important to find an ideal architecture for the neural network

B. Genetic Algorithms

A Genetic Algorithm (GA) is optimization technique to tackle mathematical problems. It is based on the concept of

[1]Department of Computer Science and Software Engineering Auburn University Auburn, USA

genetic reproduction and evolution observed in the nature. A GA is inspired by the mechanism of natural selection, a biological process in which stronger individuals are likely be the winners in a competing environment [7]. GA is not considered as a mathematically guided algorithm. It uses the biological concept of evolution to evolve the obtained optima from generation to generation rather than using stringent mathematical formulation where the obtained optima is an end product containing the best elements of previous generations where the attributes of a stronger individual tend to be carried forward into the following generation [7].

Briefly put, A Genetic Algorithm follows the similar process of reproducing offsprings from their parents' genes. Each offspring individual is expressed with its genotype and evaluated using its phenotype [8]. The process starts with (1) selection where the parents are selected, then (2) crossover where parents' genes are crossed over to create offspring's gene, then (3) mutation where the genes are slightly modified externally, and (4) computing fitness where the fitness of newly created offspring is calculated. After going through the whole process, A GA creates a new generation of chromosomes where some or all parents are replaced by some or all offspring. The replacement is determined through different evolutionary computational strategies among which the following were used in the experiment.

C. Modification to Flat Crossover (BLX-0.0) Technique

Blend Crossover ($BLX-\alpha$) with $\alpha = 0.0$, also known as the flat crossover, assigns a random value between the genome of the two selected parents to each genome of an offspring. For example: Given two parents where X and Y represent a floating-point number: (1) Parent 1: X (2) Parent 2:Y (3) Offspring: random(X,Y). For evolving hidden layers it is necessary that the offspring's chromosome contains the number of neurons in each hidden layer as integer numbers. Hence, in the experiment the results from crossover operation were converted into integer numbers.

D. $\mu + 1$ Replacement Strategy

In steady-state GA, two parents are chosen using any selection techniques and an offspring is created. The newly created offspring replaces the worst fit individual from the population. The steady-state genetic algorithm works in such a way that one or two members of the population are changed at a time . In the experiment, a modification to steady-state GA was implemented with $\mu + 1$ strategy to ensure that offspring does not replace the worst fit individual in case if the offspring is worse than the worst fit individual.

E. Keras For Neural Network

Keras is a high-level neural networks API, written in python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

Fig-2 shows the code block of how the evolving architecture could be implemented using Keras deep learning library. In Fig-2, the 'LayerList' variable contains a list of hidden layers with the number of neurons in each hidden layer. In the experiment, the 'LayerList' is the chromosome component of a GA.

```
# Define Model Architecture
LayerCount = 0
print("LayerList\n" + str(LayerList))
NeuralNetwork = tf.keras.Sequential()
NeuralNetwork.add(tf.keras.layers.Flatten(name='InputLayer'))
for i in range(len(LayerList)):
    if int(LayerList[i]) and int(LayerList[i]) > 10:
        NeuralNetwork.add(tf.keras.layers.Dense(int(LayerList[i]), activation=tf.nn.relu))
        LayerCount += 1
NeuralNetwork.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax, name='OutputLayer'))
print("Layers Added: "+ str(LayerCount))
```

Fig. 2. Implementation of MLP architecture using Keras for evolving hidden layers

F. MNIST Dataset

MNIST dataset is one of the example datasets from the keras deep learning library. The dataset contains 60,000 28×28 sized images of hand-written digits from 0 to 9. A neural network can learn from these samples and predict a digit value by looking at an image of a hand-written digit. It is an example of a large dataset.

G. NeuroPLC Dataset [9]

An effort has been made to secure the industrial automation systems running on programmable logic controllers (PLCs). A neural network is attached with a PLC's IO port. Based on the port values the neural network learns how the automation system operates and then it tries to predict the next step that the automation system is going to do. The evolving architecture was performed to find the optimum architecture that balances between the full accuracy and the binary accuracy.

III. Experiment

The experiment was performed on the MNIST dataset (a large size dataset) and the NeuroPLC [9] dataset (a small size dataset). A neural network which is configurable to have up to five hidden layers and up to 150 neurons in each layer, was given to the genetic algorithm with BLX-0.0 crossover and $\mu + 1$ replacement strategy to evolve the number of hidden layers as well as the number of hidden neurons in each layer. The experiment was performed with the population size of 5 and the mutation amount of 0.05. The experiment was performed five times on each dataset. The GA was able to produce a neural network architecture with improved accuracy on each run.

IV. Results

Table-I and Table-II show the best hidden layer architecture obtained from the GA from each experiment run on the MNIST and NeuroPLC datasets. In both tables, hidden layers are represented as [a, b, c] which means there are three hidden layers and each layer has a, b, and c number of neurons respectively. For an example, [139, 107, 73, 141] suggests that there should be four hidden layers and each layer should have 139, 107, 73, and 141 neurons respectively.

TABLE I

HIDDEN LAYERS GENERATED BY GA FOR THE NEUROPLC DATASET

Run	Hidden Layer	Accuracy
1	[19, 30]	0.87
2	[29, 38]	0.91
3	[47, 21, 37]	0.85
4	[20, 13, 29]	0.89
5	[22, 23]	0.85

TABLE II

HIDDEN LAYERS GENERATED BY GA FOR THE MNIST DATASET

Run	Hidden layers	Accuracy
1	[122, 109, 116]	0.9659
2	[139, 107, 73, 141]	0.9649
3	[96, 86]	0.9637
4	[108, 52]	0.9618
5	[114, 145, 87]	0.9635

Table-I shows that the GA evolved architectures with two or three hidden layers and no more than 50 neurons per layer whereas from the Table-II, the GA evolved architectures that are larger in size compared with what is in Table-I. This makes sense since the MNIST dataset is a larger dataset compared to NeuroPLC and it has more patterns to be learned.

V. CONCLUSION

From the results we can conclude that genetic algorithms can evolve the architecture of a neural network to improve its accuracy. This is a better approach to develop an architecture than trial-and-error approach. The crossover operations and an appropriate replacement strategy can generate better architectures from randomly generated architectures that are focused on improving the accuracy of the model.

VI. FUTURE WORK

REFERENCES

[1] B. M. Wilamowski, "How to not get frustrated with neural networks," *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 5–11, 2011.

[2] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, 2011, vol. 15. [Online]. Available: http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

[3] Wikipedia, "Multilayer perceptron - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Multilayer_perceptron

[4] D. Hunter, H. Yu, M. S. Pukish, III, J. Kolbusz, and B. M. Wilamowski, "Selection of Proper Neural Network Sizes and Architectures—A Comparative Study," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, pp. 228–240, may 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6152147/

[5] M. Fodslette Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525–533, 1993.

[6] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, jun 1963. [Online]. Available: http://epubs.siam.org/doi/10.1137/0111030

[7] K. F. Man, K. S. Tang, and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design]," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, Oct 1996.

[8] K. Casey, A. Garrett, J. Gay, L. Montgomery, and G. Dozier, "An evolutionary approach for achieving scalability with general regression neural networks," *Natural Computing*, vol. 8, no. 1, pp. 133–148, Mar 2009. [Online]. Available: https://doi.org/10.1007/s11047-007-9052-x

[9] B. Joshi, "Application of neural network on plc-based automation systems for better fault tolerance and error detection," *AUETD*, Aug 2019.