Predicting diabetes in patients using Classification Neural Networks

Garv Sudhir Nair, Michael Lu Han Xien, Anjali Radha Krishna, Liew Tze Chen, Liew Hsien Loong

Department of Computing and Information Systems, Sunway University
Subang Jaya, Malaysia

19073535@imail.sunway.edu.my

18081588@imail.sunway.edu.my

16009847@imail.sunway.edu.my

18032730@imail.sunway.edu.my

18030221@imail.sunway.edu.my

***Abstract* – This document contains a study on the use of neural networks on a dataset to predict the likelihood of diabetes occurring in a person. The neural network chosen for this project was the multilayer perceptron and was implemented using Python.**

***Keywords* – Neural Networks, Multi-layer Perceptron, Python, Hybrid Intelligence Systems**

I. Introduction

Machine learning and the concepts based within its domain have grown and adapted alongside technology for decades. As such, there came a point where technology could base itself off complex biological structures to further improve itself. One such example would be Artificial Neural Networks (ANNs) . They are artificially adaptive systems that are inspired by the human brain, in the way that they are capable of learning and adapting to a functional objective [1]. In this report, students were tasked with selecting a dataset, and constructing an appropriate neural network that can be applied upon it. We shall discuss the chosen dataset and its description, as well as our chosen neural network. We will also discuss the enhancements that can be made to the constructed neural network by making use of hybrid intelligence systems. Python was used to construct and apply the neural network on the selected dataset with the assistance of specialized libraries. The libraries used are as follows:

1. *pandas* – a library used for data analysis and manipulation
2. *matplotlib* – a visualization library used primarily for generating graphs, and figures
3. *numpy* – a library that allows for easier array manipulation
4. *sklearn* – A machine learning library that provides means for classification and prediction
5. *Dataset* – A dataset for diabetes prediction in patients.

A. Dataset Description and Purpose

The dataset we have chosen for this assignment is the *Pima Indians Diabetes Database* created by the National Institute of Diabetes and Digestive and Kidney Diseases in the United States of America.

This dataset consists of the following variables:

- Pregnancies – Number of pregnancies the patient has had
- Glucose – Plasma glucose concentration from 2 hours in oral glucose tolerance test
- BloodPressure – Diastolic blood pressure (mm Hg)
- SkinThickness – Thickness of the patient's triceps skin folds (mm)
- Insulin – 2 hours serum insulin (mu U/ml)
- BMI – Body Mass Index measured from weight(kg) / height(m)^2

- Age – Age of the patient in years
- DiabetesPedigreeFunction – Risk of diabetes based on family history
- Outcome – The patient's diabetic status
    - 1: has diabetes
    - 0: does not have diabetes

These studies are stored numerically and intended to be used to predict any individual's likelihood of having diabetes.

II. Neural Network

A. Architecture

An ANN is a set of deep learning algorithms that attempts to recognize the underlying relationships in a piece of data using a method inspired by biological neurons [1][2]. It is often used to solve problems in the fields of Artificial Intelligence, Machine Learning, and Deep Learning. Examples of such are clustering, pattern recognition, classification, regression, structured prediction, anomaly detection, dimension reduction, computer vision, machine translation, visualization, and decision making. In this report, the problem that we are attempting to solve comes under structured prediction, where the inputs and outputs of the dataset are known. The objective is to correctly predict the output of the dataset i.e., the purpose of applying the neural network to the chosen dataset is to accurately generate predictions on whether a patient is diabetic or not.

The class of neural network used in this paper and applied to the dataset is known as a Multi-Layer Perceptron (MLP). It is a feedforward neural network consisting of three node layers: an input layer, a hidden layer, and an output layer [3]. Data is fed through the input layer and then undergoes abstraction within the hidden layer. The predictions are then given out in the output layer. All input layers are associated with initial weights in a weighted sum and put through an activation function, much like a normal perceptron [4]. However, an MLP distinguishes itself by utilizing a supervised learning technique known as *backpropagation*, which assists it in training. Backpropagation allows the neural network to iteratively adjust the weights in its network to minimize its cost function [3][5]. We have chosen to use a multi-layer perceptron as they are well suited for classification prediction problems. As the goal is to generate a prediction on whether or not a patient has diabetes, it is ideal to make use of MLP as they are designed for accurate approximations.

B. Implementation

The Multi-layer Perceptron neural network was implemented with the help of a python machine learning library called *scikit-learn*. *Scikit-learn* is one of the top machine learning libraries that provides various machine learning algorithms for classification, regression, clustering, and more. As we are trying to implement a Multi-layer Perceptron for the purpose of classification, the *MLPClassification* object provided by the library will be used. The object implemented in the library allows users to modify the hidden layers, activation functions, weight optimization solvers, maximum iteration for training, and many more. These variables mentioned previously will be tested to find the optimized values for the neural network. During each test run, all random number generators will be seeded with the same value. Furthermore, they will also use the same set of training and testing data for all different configurations to keep the tests stable and fair. Different test runs will have a different set of training and testing data as well as seed values to permit some degree of randomness whilst holding on to the overall fairness of the tests as a whole. The dataset for all test runs was split into 80% and 20% for training and testing respectively. To determine how well a neural network performs, a *score* function provided by the library was used to determine the mean accuracy on the given test data.

Test Case 1: Activation function and solver combination – The first test is to find the optimal activation function and solver to be used in the Multi-layer Perceptron. From the *MLPClassifier* object, users can choose from a fixed set of choices of activation functions and solvers. Choices provided for activation functions are the identity function (*'identity')*, logistic sigmoid function (*'logistic'*), hyperbolic tan function (*'tanh'*), and rectified linear unit function (*'relu'*). Choices provided for solvers are the Limited-memory BFGS (*'lbfgs'*), stochastic gradient descent (*'sgd'*), and stochastic gradient-based optimizer (*'adam'*). Although more values for the activation functions and solvers can also be configured to further optimize the neural network, the values are kept to the default values provided by the library to simplify the optimization and testing process. With that, each activation function is tested with every solver for 20 test runs. During all test runs, the permitted maximum iteration for training was set to 50,000 to ensure that every configuration gets to converge on a solution. Two hidden layers were provided to every configuration with 20 neurons in each layer. The figures below show the mean score calculated for all different configurations after conducting 20 test runs each.

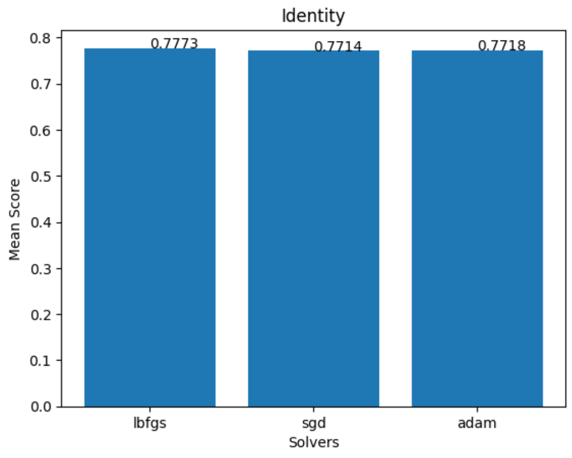


Figure 2.1 - Identity function with different solvers

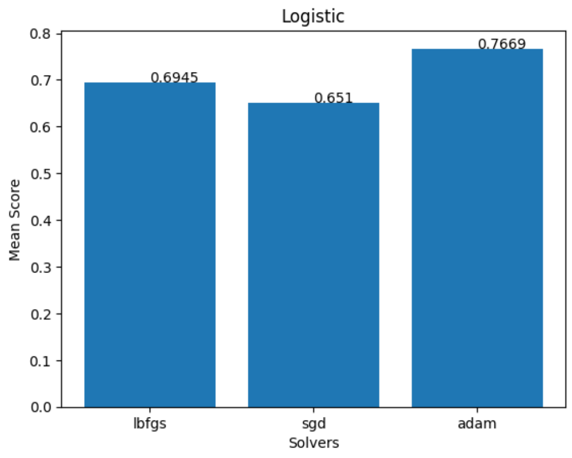


Figure 2.2 – Logistic sigmoid function with different solvers

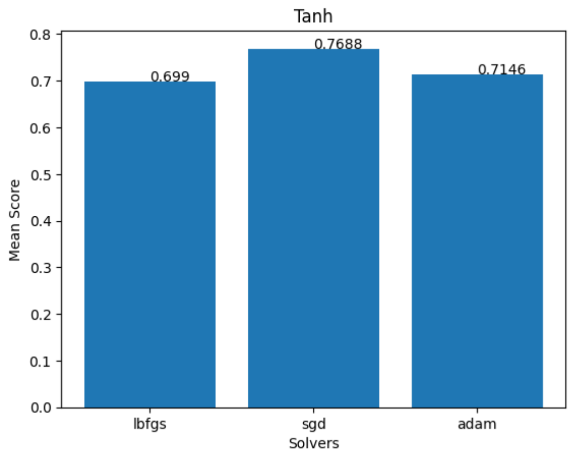


Figure 2.3 – Hyperbolic tan function with different solvers

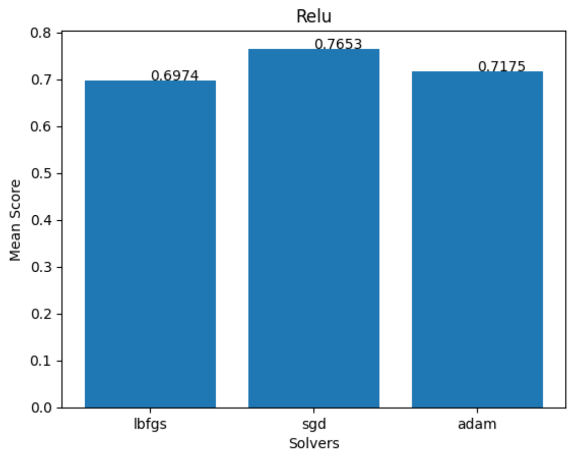


Figure 2.4 – Rectified linear unit function with different solvers

From the figures above (Figure 2.1 – 2.4), it can be observed that the identity function is the most stable and best performing activation function among all the various solver combinations. It performs at its best when utilized with the Limited-memory BFGS and is able to achieve a mean score of 77.73% accuracy. It performs worse with the stochastic gradient descent solver, achieving only a mean score of 77.14% accuracy, but still outperforms every other activation function's combination.

Test Case 2: Hidden Layer and Neuron – In this test, the number of hidden layers and neurons associated in each layer will be tested to find the optimal number of hidden layers and neurons. Like the previous test, the number of maximum iterations allowed for training was set to 50,000 and 20 rounds of tests were conducted for every combination. However, in this test case, the activation function and solver combination were set to the identity function and Limited-memory BFGS, as it was found to be the best performing combination from the previous test case. The maximum number of hidden layers permitted for testing was set to 6 layers, and each hidden layer can was set at a maximum of 60 neurons. To simplify the test and lower computational time, the number of neurons in every layer was kept the same, as exploring all different possible combinations of varying number of neurons in each layer would require more complexity and data generation. The results of the test can be seen in the figures below.

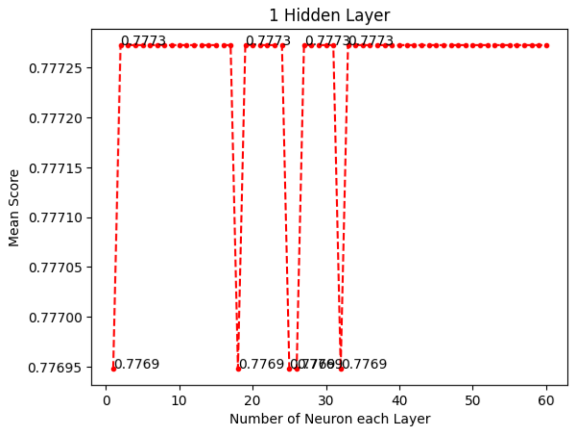


Figure 2.5 – Mean Score of 1 Hidden Layer and Varying Number of Neuron

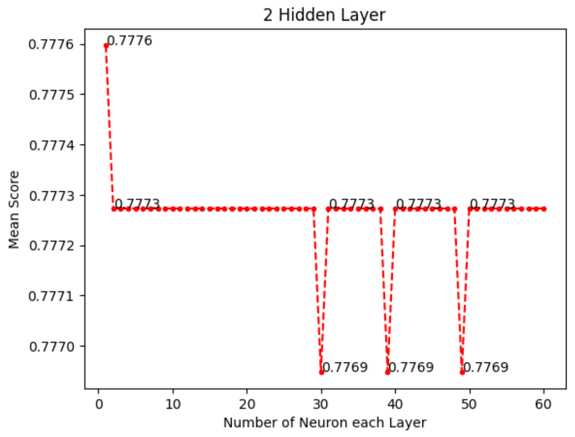


Figure 2.6 – Mean Score of 2 Hidden Layer and Varying Number of Neuron

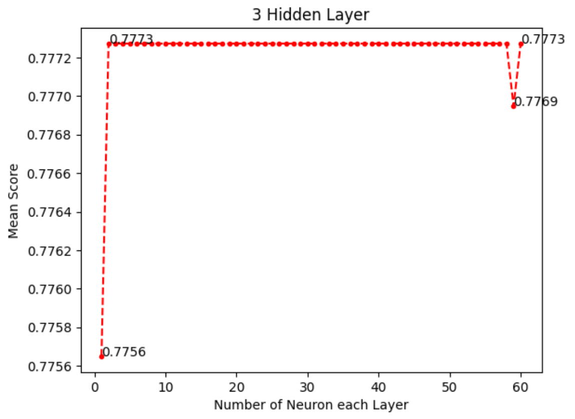


Figure 2.7 – Mean Score of 3 Hidden Layer and Varying Number of Neuron

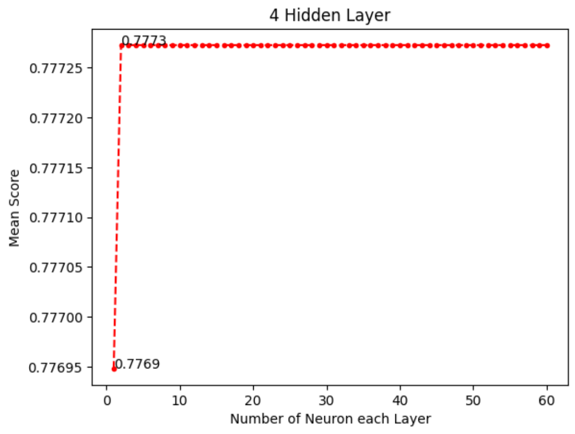


Figure 2.8 – Mean Score of 4 Hidden Layer and Varying Number of Neuron

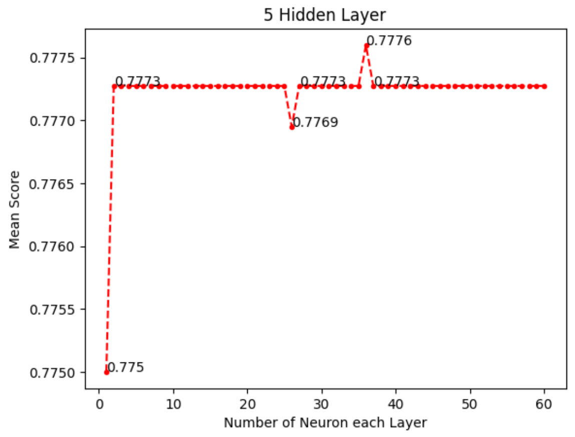


Figure 2.9 – Mean Score of 5 Hidden Layer and Varying Number of Neuron

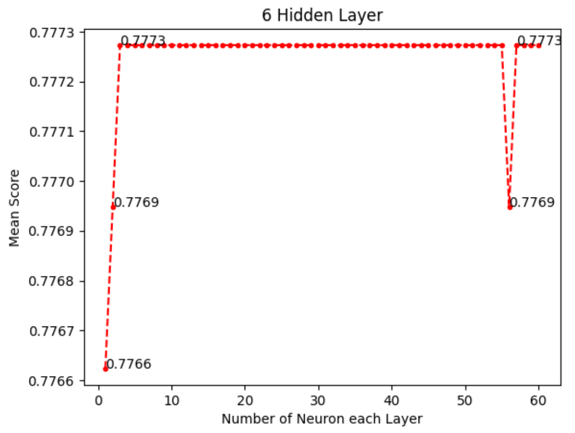


Figure 2.10 – Mean Score of 6 Hidden Layer and Varying Number of Neuron

From the figures above (Figure 2.5 – 2.10), it can be observed that at every different number of hidden layers, the neural network stabilizes at a mean score of 77.73% accuracy. However, most of the hidden layers' mean accuracy fluctuates based on the different number of neurons provided to them. Hidden layers 1 and 2 fluctuate the most and can be deemed to have volatile behavior. It can also be observed that at hidden layers 2 and 5, the mean accuracy reached a peak score of 77.76% accuracy. However, since this peak only occurred once in both layers and the accuracy is not too different from the stabilizing point, it can be assumed that the peak was caused by favorable values generated by randomness and is not an accurate representation of the neural network's performance. Other hidden layers provide the most stable accuracy reading, with the starting point showing a lower mean score compared to the stabilizing point, which quickly increases to the stabilizing point as the number of neurons increases. This shows that increasing the number of neurons does help improve the neural network's performance. As 4 hidden layers provide the most stable reading with no fluctuations occurring, it is deemed as the most optimal and stable number of hidden layers. As for the number of neurons to provide for the hidden layer, Figure 2.8 shows that with only 2 neurons in each layer, the neural network can reach the stabilizing point. However, the number of neurons chosen to provide each hidden layer was 6 neurons. The purpose of this was to provide some safeguarding and redundancy.

Test Case 3: Training Iteration – After finding the optimal hidden layers, activation function, and solver, a final test is conducted to find the optimal number of maximum iterations to provide to the neural network. In this test, the optimal activation function, solver, and hidden layers found in test cases 1 and 2 were used. However, the number of test runs was increased to 30 runs as the computational time was significantly faster than the previous test cases. The value range for maximum iterations was set from 1 to 100. The following figure 2.11 shows the results of the test.

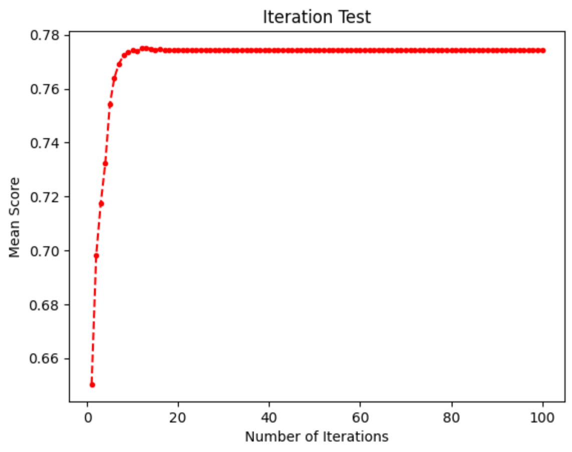


Figure 2.11 – Mean Score of Maximum Training Iterations

From Figure 2.11, it can be observed that as the maximum number of iterations for training increases, accuracy of the neural network increases as well. This is because the neural network needs time to optimize the weight values and converge on an optimum solution. It can be observed that after 20 iterations, the program stabilizes at a mean score of 77.73% accuracy and thus, it was decided that the optimal number of maximum iterations to provide for training was 20.

Final Test – After all variables have been tested and the optimum values for each variable have been found, a final test was conducted to observe how well the Multi-layer Perceptron neural network can perform. In this test, the neural network was configured with the identity function, Limited-memory BFGS solver, 4 hidden layers with 6 neurons in each layer, and 20 maximum iterations allowed for training. With that, 200 test runs were conducted and the results are shown in Figure 2.12 and Figure 2.13 in the next section.

III. Result and Discussion

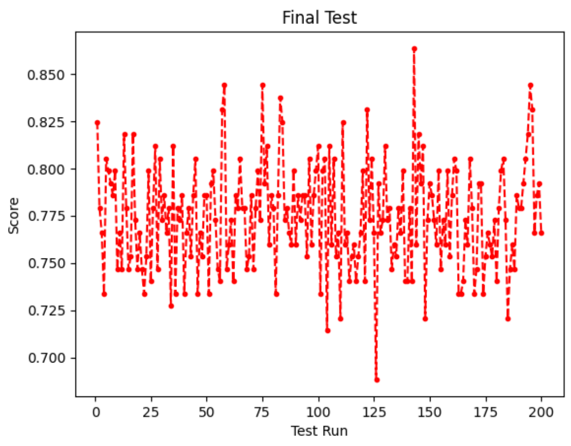
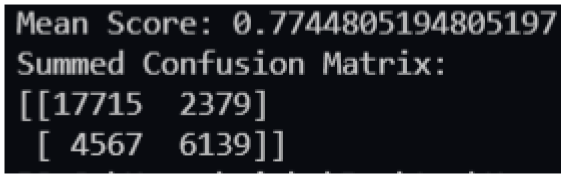


Figure 2.12 – Mean Accuracy of Each Test Run



```
Mean Score: 0.77448051948051 97
Summed Confusion Matrix:
[[17715  2379]
 [ 4567  6139]]
```

Figure 2.13 – Mean Score and Confusion Matrix of Final Test

From figure 2.13, it can be observed that after running 200 test runs, the Multi-layer Perceptron neural network only managed to achieve a mean score of 77.45% accuracy. This result is much lower than the expected mean score of 77.73% accuracy. The reason for the lower accuracy could be caused by the randomness of values generated. This behavior of randomness can be observed from Figure 2.12 where the score of the neural network can be observed to still be fluctuating in a volatile manner. From figure 2.13, it can be observed that the neural network is able to predict non-diabetic patients accurately but begins to struggle when trying to predict diabetic patients. The reason for this could be due to the bias between diabetic and non-diabetic patients in the dataset. This bias can be observed as out of the 768 entries of patients' data, only 268 are of diabetic patients. Which means only 35% of the dataset consists of diabetic patients. This sets a large bias towards non-diabetic patients which, in turn, could be the reason why the neural network struggles to classify diabetic patients.

IV. Hybrid Intelligence Systems

Hybrid Intelligence Systems are those systems that combine two or more methods and techniques from various subfields of artificial intelligence into a single functioning system. Examples of such systems would include,

- Neuro-Fuzzy Systems
- ANFIS
- Evolutionary Neural Networks
- and Fuzzy Evolutionary Systems

These hybrid intelligence systems in particular combine the features of two distinct systems in order to make up for each systems' weaknesses. As previous sections covered the use of a multi-layer perceptron neural network to predict whether or not a patient has diabetes, this section will cover the application of two possible hybrid intelligence systems that could be applied to our neural network.

A. Evolutionary Neural Network

The first hybrid intelligent system that could be used to enhance the neural network is an evolutionary neural network. Conventional neural networks such as the one implemented in this report utilize backpropagation algorithms to identify optimal weights for connections between neurons. However, backpropagation cannot always guarantee an optimal solution for every problem. The topology of the neural network is also often chosen randomly, or with restricted heuristics by the human observer, therefore there is no guarantee that the chosen topology is optimal for the problem. Many of these problems may be addressed with an evolving neural network.

Evolutionary neural networks are neural networks where the optimal weights and topologies are determined using a genetic algorithm. In such cases, multiple neural networks are generated with varying weights and topologies. The parameters of each neural network are then encoded as chromosomes in the genetic algorithm, where the genetic algorithm will attempt to optimize its parameters using crossovers and mutations. The networks that performed best (highest fitness) are then used to improve and optimize future networks.

To implement an evolutionary neural network, the existing parameters must first be converted into a manner usable by a genetic algorithm. Weights within the neural network can be encoded as a chromosome that can be used by the genetic algorithm. The fitness function of each chromosome can be determined by the performance of the network it represents [6]. In this implementation, the goal of the genetic algorithm can be used to optimize the weights such that the maximum fitness (accuracy of the network) is achieved. After encoding the chromosomes and determining the fitness function, the evolutionary neural network will then proceed to follow the genetic algorithm's method of reproduction and mutation to find the optimal weights.

An evolutionary algorithm shares many similarities to a traditional neural network in the way input/outputs are computed, and these segments do not require any modifications. Instead, the goal of the evolutionary neural network is to provide an alternative way to optimize the weights and topology of the neural network (genetic algorithm vs backpropagation algorithm). As evolutionary neural networks can freely modify both the weights and topologies of the network as opposed to the backpropagation's sole purpose of optimizing weights, it is assumed that an evolutionary neural network would be able to help us identify and form more meaningful connections between neurons and achieve a higher accuracy than a traditional neural network. Thus, an evolutionary neural network is believed to be able to help improve the Multi-layer Perceptron neural network system implemented in this report.

B. Neuro Fuzzy System

The second hybrid intelligent system that could be used to improve the performance and functionality of the neural network is the neuro-fuzzy system. A neuro-fuzzy system is one that determines its parameters such as fuzzy sets and fuzzy rules by processing data samples using a learning algorithm developed from neural network theory [6]. The neural network contains a low-level computational structure which makes it good at handling raw data, whereas the fuzzy logic or inference system is good at high-level reasoning of linguistic information. However, there are some problems that can be found in both neural networks and fuzzy logic. The neural network is opaque to the user which makes the user have difficulty understanding how the neural network performs decision making, whereas although fuzzy logic is transparent, but it cannot learn or adjust to a new environment. Therefore, the combination of both forms a neuro-fuzzy system, allowing the neural network to be more transparent and gain the capability of learning.

In the neuro-fuzzy system, there are multiple layers that can be found such as the input layer, input membership functions layer, fuzzy rules layer, output membership functions layer, and the defuzzification layer. The neuro-fuzzy system uses the data collected from the dataset to update the critical points of the membership functions. Then, it helps to identify the fuzzy rules and their importance. The importance of the rules depends on the connection of weight in the neural network. Finally, the rules are approved or disapproved using the data. In this case, the data that were used to update the critical points in the membership function are mentioned in the dataset description in Section I.

Moreover, the fuzzy sets can be thought of as weights, whereas the rules, input, and output variables can be described as neurons. It is necessary to define the membership functions that express the linguistic terms of the inference rules. There is no formal technique to define their functions in the fuzzy set theory. With an arbitrary set of parameters, any shape can be considered a membership function. As a result, these functions in terms of data optimization are crucial for fuzzy systems.

However, a major difference between Neuro Fuzzy System and Multi-layer Perceptron lies within the concept of classification and Fuzzy logic. Classification follows a Boolean logical method of computing results where something is either true or false [7]. Fuzzy logic on the other hand, computes based on the degree of truth. With this, Neuro Fuzzy System could help expand the capability of the Multi-layer Perceptron neural network. Instead of only classifying whether a patient is either diabetic or non-diabetic, Neuro Fuzzy System could help determine how diabetic a person is, what type of diabetes the patient has, and so on. With this, Neuro Fuzzy System is believed to not only be able to help improve the Multi-layer Perceptron neural network system implemented in this report, but also extends its capability of its predictions.

V. CONCLUSION

This paper described an approach to analyzing and predicting the likelihood of a patient developing diabetes using a Multi-layer Perceptron, a neural network that is well suited for accurate approximations. Once tested, an analysis was further carried out on determining how effective the model was in its performance and what improvements may be implemented to further enhance the results received through the neural network.

While the neural network was successful in pointing out which patients are non-diabetic, it came upon issues when trying to predict which patients are diabetic, with an accuracy score of 77.45%. With the result being lower than the expected value of 77.73%, we can conclude that there is room for improvement in our implementation. These improvements may come in the form of hybrid systems such as an Evolutionary Neural Network or a Neuro Fuzzy system.

The use of the first hybrid system comes with the added benefit of using a genetic algorithm to optimize the weights and topology while training the neural network. This method of optimizing the neural network would greatly improve its accuracy and prove to be a great enhancement made on the multi-layer perceptron that was created and tested in this paper. Alternatively, a neuro fuzzy system can also studied. While its compatibility with the multi-layer perceptron is limited as the way the information is classified may require more significant changes on how the data is perceived and presented, it does extend the capabilities of the existing MLP system through the use of fuzzy logic. With the added flexibility provided by fuzzy logic, the system would be capable of determining the type of diabetes a patient has and other such factors. Thus, one can conclude that both options are viable enhancements to the current neural network at hand.

REFERENCES

[1]E. Grossi and M. Buscema, "Introduction to artificial neural networks", *European Journal of Gastroenterology & Hepatology*, vol. 19, no. 12, pp. 1046-1054, 2007. Available: 10.1097/meg.0b013e3282f198a0 [Accessed 26 November 2021].

[2]G. Arminger and D. Enache, "Statistical Models and Artificial Neural Networks", Data Analysis and Information Systems, pp. 243-260, 1996. Available: 10.1007/978-3-642-80098-6_21 [Accessed 26 November 2021].

[3]B. White and F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", *The American Journal of Psychology*, vol. 76, no. 4, p. 705, 1963. Available: 10.2307/1419730 [Accessed 26 November 2021].

[4]T. Hastie, J. Friedman, and R. Tisbshirani, The elements of Statistical Learning: Data Mining, Inference, and prediction. New York: Springer, 2017.

[5]David E. Rumelhart; James L. McClelland, "Learning Internal Representations by Error Propagation," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations , MIT Press, 1987, pp.318-362.

[6]Ridha H, Njah M, Chtourou M. “Multilayer perceptron training using an evolutionary algorithm”. International Journal of Modelling Identification and Control 5(4), DOI:10.1504/IJMIC.2008.023515

[7]“What are Neuro-Fuzzy Systems?” http://fuzzy.cs.ovgu.de/nfdef.html (accessed Nov. 26, 2021).

[8]W. Chai, “What is Fuzzy Logic? - definition from whatis.com,” SearchEnterpriseAI, 08-Jun-2021. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/fuzzy-logic. [Accessed: 26-Nov-2021].