

# **HEALTH MONITORING SYSTEM USING ARTIFICIAL NEURAL NETWORKS**

A project report submitted in partial fulfilment of the requirement for the award  
of the degree of

**BACHELOR OF TECHNOLOGY**

In

**ELECTRONICS AND COMMUNICATION ENGINEERING**

Submitted

By

|                       |            |
|-----------------------|------------|
| G. SAI VIVEK          | 16011M2003 |
| P. DHATHRI PRAVALLIKA | 16011M2006 |
| N. NIRNAY REDDY       | 16011M2007 |
| CH. SUSHMA SWARAJ     | 16011M2014 |

Under the esteemed guidance  
of

**DR. A. RAJANI**

(ECE Department)



Jawaharlal Nehru Technological University Hyderabad  
JNTUH College of Engineering, Hyderabad Department of  
Electronics and Communication Engineering, Kukatpally,  
Hyderabad - 500085

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**  
**HYDERABAD**

**JNTUH College of Engineering, Hyderabad (Autonomous)**

**(Kukatpally, Hyderabad - 500085)**

**Department of Electronics and Communication Engineering**



**DECLARATION BY THE SUPERVISOR**

This is to certify that the project entitled, **“HEALTH MONITORING SYSTEM USING  
ARTIFICIAL NEURAL NETWORKS”** submitted by

|                       |            |
|-----------------------|------------|
| G. SAI VIVEK          | 16011M2003 |
| P. DHATHRI PRAVALLIKA | 16011M2006 |
| N. NIRNAY REDDY       | 16011M2007 |
| CH. SUSHMA SWARAJ     | 16011M2014 |

In partial fulfilment of the requirements for the award of major project in Electronics and Communication Engineering at Jawaharlal Nehru Technological University Hyderabad during the academic year 2019-2020, is an authentic work carried out by them under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree or Diploma.

**PROJECT SUPERVISOR**  
**DR. A. RAJANI**  
Department of ECE, JNTUHCEH

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**  
**HYDERABAD**

**JNTUH College of Engineering, Hyderabad (Autonomous)**

**(Kukatpally, Hyderabad - 500085)**

**Department of Electronics and Communication Engineering**



**CERTIFICATE BY THE HEAD OF THE DEPARTMENT**

This is to certify that the project entitled, **“HEALTH MONITORING SYSTEM USING  
ARTIFICIAL NEURAL NETWORKS”** submitted by

|                       |            |
|-----------------------|------------|
| G. SAI VIVEK          | 16011M2003 |
| P. DHATHRI PRAVALLIKA | 16011M2006 |
| N. NIRNAY REDDY       | 16011M2007 |
| CH. SUSHMA SWARAJ     | 16011M2014 |

In partial fulfilment of the requirements for the award of major project in Electronics and Communication Engineering at Jawaharlal Nehru Technological University Hyderabad during the academic year 2019-2020, is an authentic work carried out by them under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree or Diploma

HEAD OF DEPARTMENT  
DR. A. ANITHA SHEELA  
PROFESSOR AND HEAD,  
Department of ECE, JNTUHCEH

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**  
**HYDERABAD**

**JNTUH College of Engineering, Hyderabad (Autonomous)**

**(Kukatpally, Hyderabad - 500085)**

**Department of Electronics and Communication Engineering**



**DECLARATION BY THE CANDIDATES**

We hereby declare that the major project entitled “**HEALTH MONITORING SYSTEM USING ARTIFICIAL NEURAL NETWORKS**” is a bonafide record work done and submitted under the esteemed guidance of **DR.A.RAJANI**, Professor, Department of ECE, JNTU, Hyderabad, in partial fulfilment of the requirements for the major project in Electronics and Communication Engineering at the Jawaharlal Nehru Technological University Hyderabad during the academic year 2018-2019. This record is a bonafide work carried out by us and the results kept in the major project have not been reproduced or copied. The results in the major project have not been submitted in any other university or institution for the award of degree or diploma.

|                       |            |
|-----------------------|------------|
| G. SAI VIVEK          | 16011M2003 |
| P. DHATHRI PRAVALLIKA | 16011M2006 |
| N. NIRNAY REDDY       | 16011M2007 |
| CH. SUSHMA SWARAJ     | 16011M2014 |

## **ACKNOWLEDGEMENTS**

We would like to express our sincere deep appreciation and indebtedness particularly to our guide DR. A. RAJANI, Professor, Electronics and Communication Engineering Hyderabad, for her invaluable guidance, comments and suggestions throughout the project. We also thank our head of the department DR. K. ANITHA SHEELA, Professor & Head, Electronics & Communication Engineering, for her endless support, kind and understanding spirit.

The completion of this undertaking could not have been possible without the participation and assistance of so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. I also thank my relatives, friends and others who shared their support, morally, financially or physically. Above all, I thank the Great Almighty, the author of knowledge and wisdom, for his countless love.

|                       |            |
|-----------------------|------------|
| G. SAI VIVEK          | 16011M2003 |
| P. DHATHRI PRAVALLIKA | 16011M2006 |
| N. NIRNAY REDDY       | 16011M2007 |
| CH. SUSHMA SWARAJ     | 16011M2014 |

## ABSTRACT

Infections and diseases are gaining a hand over the human beings in this era. With the health care centres being over loaded with patients, continuous vigilance of the patients at the time of epidemic like Malaria, Dengue, Ebola is not possible. As each life is crucial and cannot be risked, measures need to be taken such that in absence of doctors or nurses the patient should feel safe. This can be achieved only if we can substitute with an alternative like robots. As the world is advancing rapidly the use of technology in the field of medical sciences is essential but with a proper understanding of the functionalities of the products, as such there are many machines which measure the parameters like Pulse, Blood Pressure and Blood glucose.

Artificial Neural Networks can play an essential role in the diagnosis and estimation of the patient's condition. When such information and data is provided to the doctors, it becomes easy for the doctors to monitor the patient and prevent the patient from entering into the critical stage. We are using ANN model with multiple hidden layers to create a model which can estimate the health condition of patients who are not in continuous vigilance of the hospital staff. Many diseases can be identified by the symptoms but in few situations change in the values of parameters like blood pressure, blood glucose and pulse come into account. With the help of this model we can analyse the health condition of the patient. The information regarding blood pressure, blood glucose and pulse can be used to take necessary counter measures. A dataset with a sample size of 99149 consisting of all the possible combinations of all the 5 parameters age, pulse, blood pressure, blood glucose and alert is created. From the dataset only 80% of the data is used for training the model and 20% is used for testing the model. Based on the results the accuracy of the model is analysed. Confusion matrix is used to know the estimations made by the model to the given input's (alert).

| TABLE OF CONTENTS              | PAGE.NO. |
|--------------------------------|----------|
| 1. INTRODUCTION                | 1        |
| 1.1 Introduction               |          |
| 1.2 Aim                        |          |
| 1.3 Objective                  |          |
| 1.4 Approach                   |          |
| 2. TOOLS AND METHODOLOGY       | 5        |
| 2.1 Python                     |          |
| 2.2 TensorFlow                 |          |
| 2.3 Deep Neural Network        |          |
| 2.4 The Datasets               |          |
| 3. ARCHITECTURE                | 9        |
| 3.1 Architecture               |          |
| 3.2 Activation Functions       |          |
| 3.3 Loss Function              |          |
| 3.4 Gradient Descent           |          |
| 3.5 Adam's Optimization        |          |
| 4. RESULTS                     | 18       |
| 5. CONCLUSION AND FUTURE SCOPE | 26       |
| REFERENCES                     |          |
| APPENDIX                       |          |

## LIST OF FIGURES

| S.NO | PAGE.NO | FIG.NO. | NAME OF THE FIGURE                   |
|------|---------|---------|--------------------------------------|
| 1    | 1       | 1.1     | Flowchart of the program             |
| 2.   | 3       | 1.2     | The mechanism of model               |
| 3.   | 7       | 2.1     | Structure of a biological neuron     |
| 4.   | 7       | 2.2     | Non-linear architecture              |
| 5.   | 8       | 2.3     | Dataset of the model                 |
| 6.   | 8       | 2.4     | Dataset of the model                 |
| 7.   | 9       | 3.1     | Feedforward Architecture             |
| 8.   | 10      | 3.2     | Backpropagation Architecture         |
| 9.   | 11      | 3.3     | Architecture of the Model            |
| 10.  | 13      | 3.4     | Graph of Activation function Relu    |
| 11.  | 14      | 3.5     | Graph of Activation function Sigmoid |



| <b>S.NO</b> | <b>PAGE.NO</b> | <b>FIG.NO.</b> | <b>NAME OF THE FIGURE</b>                                |
|-------------|----------------|----------------|--|
| 12.         | 15             | 3.6            | Schematic of Gradient Descent                            |
| 13.         | 19             | 4.1            | Graph of 1 hidden layer loss                             |
| 14.         | 20             | 4.2            | Graph of 2 hidden layer loss                             |
| 15.         | 21             | 4.3            | Graph of 3 hidden layers loss                            |
| 16.         | 22             | 4.4            | Estimation results of inputs                             |
| 17.         | 23             | 4.5            | Graphs for validation Accuracy                           |
| 18.         | 23             | 4.6            | Graphs for validation Loss                               |
| 19.         | 24             | 4.7            | Graphs related to Accuracy Vs<br>Number of hidden layers |
| 20.         | 24             | 4.8            | Graphs related to Accuracy Vs<br>Number of nodes         |
| 21.         | 25             | 4.9            | Description  |

## **LIST OF TABLES**

| <b>S.NO</b> | <b>PAGE.NO</b> | <b>FIG.NO.</b> | <b>NAME OF THE FIGURE</b>             |
|-------------|----------------|----------------|---------------------------------------|
| 1.          | 4              | 1.1            | Parameters and ranges for the dataset |
| 2.          | 18             | 4.1            | Result                                |

# 1. INTRODUCTION

## 1.1 INTRODUCTION

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modelling is the task of approximating a mapping function ( $f$ ) from input variables ( $X$ ) to discrete output variables ( $y$ ).

## 1.2 Aim

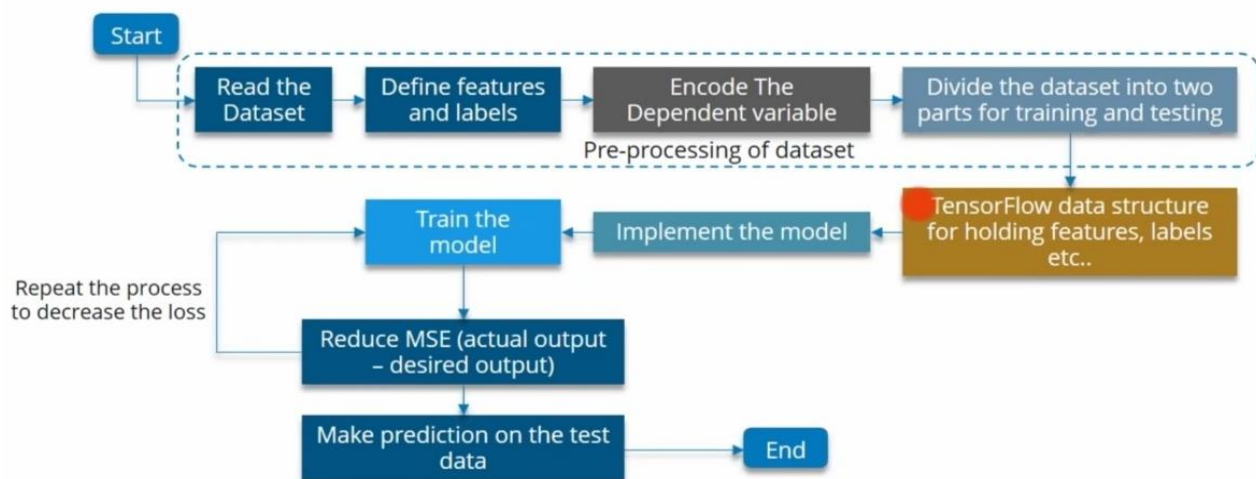
To classify the inputs, estimate the health condition of the patient and alert the hospital staff.

## 1.3 Objectives

The objective of the project is to design a model, train using the dataset and give manual inputs for estimation.

## 1.4 Approach

Dataset consisting of samples of data related to the 5 parameters Age, Pulse, Blood pressure (Systolic and Diastolic), Blood glucose and alert is created. The parameters are given as input to the model for training. After the training, manual inputs are given for estimating the health condition of the patient.



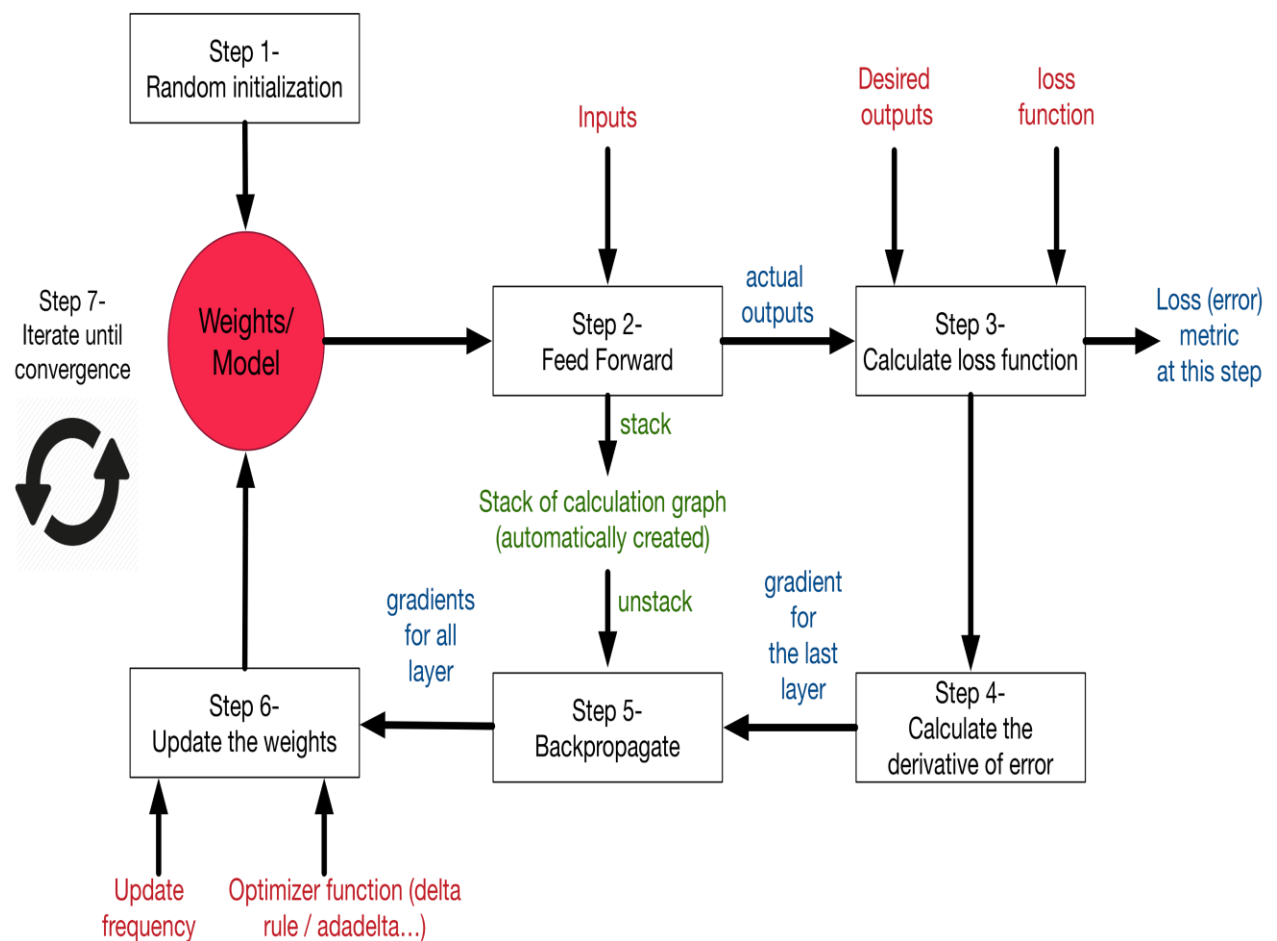
**Fig.1.1**

**(Fig.1.1 Block diagram of health monitoring system)**

The above Fig.1.4.1 shows the flowchart of Health monitoring system using ann. The neural network designed is configured for data classification through a learning process in which the input data contains the independent variables and the output data contains the dependent variable. The parameters considered in the neural network i.e Age, Pulse, Heart Beat and Blood Glucose are defined as variables. After defining the independent and dependent variables, the data set with a sample size of 99149 possible combinations of these variables is read. The input layer having 5 neurons, receives the data and passes it on to the hidden layer for processing and the results at output layer. Weights are assigned to the parameter variables in between these layers. The RELU activation function is defined in the input and hidden layer and the SIGMOID activation function is then defined in the output layer. After the features and labels of the program have been defined, the dependant variables of the network are encoded. [5]

**Pre-processing of the dataset:** Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data pre-processing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for analysis and converted into understandable format. Since the data used is comprised of attributes with varying scales, rescaling of all the parameter attributes is beneficial. After rescaling, data is binarized using a binary threshold. All the values above the threshold values are taken as 1 and all equal to or below the threshold values are taken as 0 and then standardized. After the pre-processing data, numpy and panda libraries are imported, the categorical data is handled here. The data set is divided into two parts for training and testing. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. The test dataset is to test the model's estimation on this subset. The Tensor Flow works as a library for dataflow programming and uses techniques for calculation of mathematical expressions efficiently. The neural network model is then implemented by training it and defining the number of epochs. Training a neural network basically refers to minimizing the errors. The number of epochs used are 100 to increase the accuracy and efficiency of the model. Once the looping process starts, the total error starts decreasing and by the end of the training there is least error. The training process involves taking inputs from the training dataset and perform some adjustments based on their weights by using Adam's Optimization algorithm. It is an adaptive

learning rate optimization algorithm that is designed specifically for training deep neural networks to fine-tune the weights of our neural network in such a way that the errors are minimized. Using cross validation we get a reliable estimate of model performance using only the training data. We iterate this process for an arbitrary number of 100 times. In every iteration, the whole training set is processed simultaneously. The weights of the neuron are optimized for the provided training data and the error is reduced. Finally, the network makes random estimations, which are compared with the actual output and the error is corrected [5].



**Fig.1.2**

**The Fig.1.2 Algorithm of backend mechanism of the model.**

| <b>S.No</b> | <b>PARAMETER</b>  | <b>OPTIMUM<br/>VALUE</b> | <b>RANGE</b>                 |
|-------------|---|--------------------------|------------------------------|
| 1.          | AGE   | NIL                      | 18-35 years                  |
| 2.          | PULSE   | 72 bpm                   | 49-78 bpm                    |
| 3.          | BLOOD PRESSURE (mm Hg)<br><br>SYSTOLIC<br><br>DIASTOLIC | 120 mmHg<br><br>80 mmHg  | 90-181mmHg<br><br>60-101mmHg |
| 4.          | BLOOD GLUCOSE (mg/dL)                                   | 72-99 mg/dL              | 60-180 mg/dL                 |

**Table 1.1**  
**Parameters and ranges for the dataset [9]**

## **2. TOOLS AND METHODOLOGY**

### **2.1 Python**

Python is a remarkably powerful dynamic, object-oriented programming language that is used in a wide variety of application domains. It offers strong support for integration with other languages and tools and comes with extensive standard libraries. To be precise, the following are some distinguishing features of Python:

Very clear, readable syntax.

Cross Compilation

Full modularity.

Exception-based error handling.

High level dynamic data types

Supports object oriented, imperative and functional programming styles.

Embeddable.

Scalable

Interpretable

With so much of freedom, Python helps the user to think problem centric rather than language centric as in other cases. These features make Python a best option for scientific computing [7].

### **2.2 TensorFlow**

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains [2].

### **2.3 Deep Neural Networks (DNN)**

A deep neural network is a neural network with a certain level of complexity, a neural network with more than two layers. Deep neural networks use sophisticated mathematical modelling to process data in complex ways.

## 2.4 Artificial Neural Network (ANN)

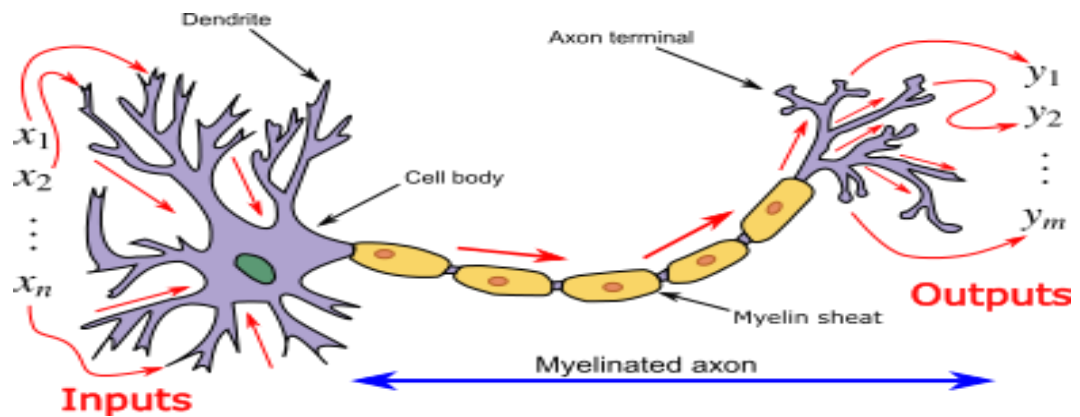
Artificial Neural Network (ANN) helps in estimating the health condition of patients in critical condition. Health is one of the global challenges for humanity. At present Information and communication are playing an important role in healthcare monitoring and wellness management. Whereas the frequently visiting doctor and knowing the condition become difficult maybe it is because of inadequate time to spend. To deal with this type of situations we need various types of parameters. The design for health monitoring system involves health condition of a human being is monitored based on several parameters. The basic parameters for knowing the condition are Heart beat or pulse, Blood Pressure which is Systolic and Diastolic, Blood glucose. The parameters are used for estimating the health condition of human being and this is obtained by creating a model. The optimum value for blood pressure is 120/80 (systolic is 120 and diastolic is 80) mm Hg, pulse is 72 bpm (beats per minute), blood glucose is 72 to 99 mg/dL. Since the optimum value is not true in all the situations, we have considered a range of values for the parameters. A dataset of 99149 samples with 5 parameters is created with age ranging from 18-35 years, pulse ranging from 49 to 78 bpm, blood pressure is divided into systolic and diastolic so the range for systolic is 90 to 181, the range for diastolic is 60 to 101, the range for blood glucose level is 60 to 180 mg/dL.

The Artificial Neural network is similar to biological neuron where the biological neurons are helpful in learning, storing and understanding the information given to a human being. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

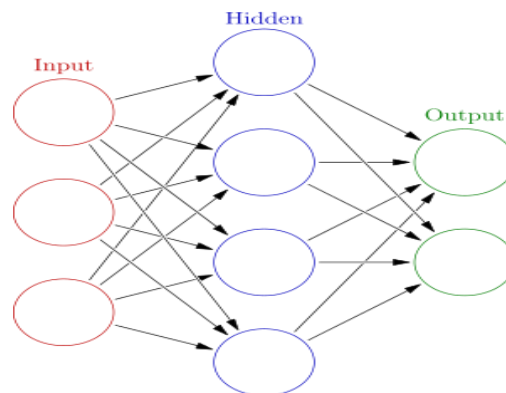
Once the dataset is given to the model it starts to understand and analyse the data present in it. After reading the data the parameters are forwarded to the input layer which is similar to dendrites of a neuron, this is further sent to the hidden layers which are similar to cell body and axon of a neuron. In the hidden layers training of the model with the inputs takes place and errors are rectified. Once the errors are rectified the results are sent to the output layer which are similar to



axon terminals. These results are helpful for taking necessary counter measures for the health condition of the patient [1].



**Fig.2.1**



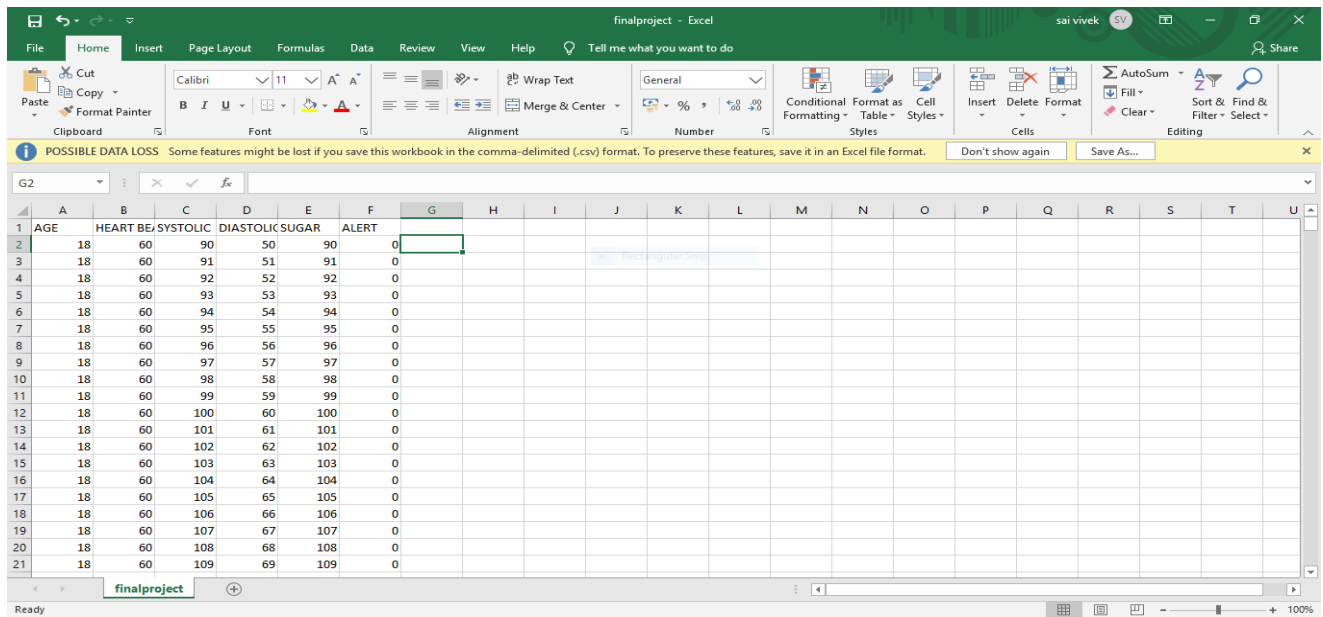
**Fig.2.2**

The neural network architecture is based on a simplified model of the brain which consists of n-number of neurons. The neurons act as the layers of the communication, the processing task being distributed over numerous neurons. Although a single neuron is able to perform simple data processing, the strength of a neural network is obtained as the result of the connectivity and collective behavior of these simple nodes. Neural networks store the strength of synaptic connections (weights of the network's branches) by which the required data are reproduced. Neural networks can be trained by adjusting the connection strengths in order to abstract the relations between the presented input/output data. We want to select a network architecture that is large enough to approximate the function of interest, but not too large that it takes an excessive amount of time to train. Another issue with large networks is that they require large amounts of data to train you cannot train a neural network on a hundred data samples and expect it to get 100%

accuracy on an unseen data set. We use multiple hidden layers as well as multiple nodes within the hidden layers, as these seem to result in the best performance. Increasing the number of layers of neural networks tends to improve overall test set accuracy.

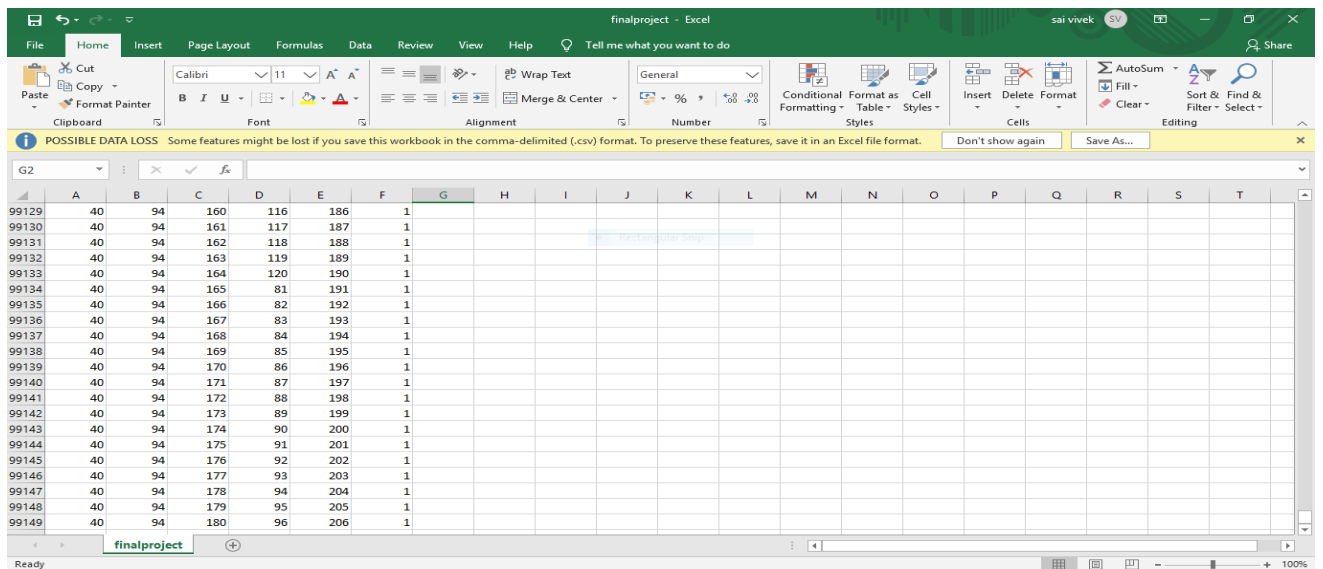
## 2.5 DATASET

Dataset is an excel sheet consisting of the combination of inputs in 5 parameters age, pulse, blood pressure (systolic and diastolic) and alert. The dataset consists of 99149 samples with various combinations. The dataset has been created keeping in notice of the optimum values.



|    | AGE | HEART BE/SYSTOLIC | DIASTOLIC SUGAR | ALERT |
|----|-----|-------------------|-----------------|-------|
| 1  | 18  | 60                | 90              | 50    |
| 2  | 18  | 60                | 91              | 51    |
| 3  | 18  | 60                | 92              | 52    |
| 4  | 18  | 60                | 93              | 53    |
| 5  | 18  | 60                | 94              | 54    |
| 6  | 18  | 60                | 95              | 55    |
| 7  | 18  | 60                | 96              | 56    |
| 8  | 18  | 60                | 97              | 57    |
| 9  | 18  | 60                | 98              | 58    |
| 10 | 18  | 60                | 99              | 59    |
| 11 | 18  | 60                | 100             | 60    |
| 12 | 18  | 60                | 101             | 61    |
| 13 | 18  | 60                | 102             | 62    |
| 14 | 18  | 60                | 103             | 63    |
| 15 | 18  | 60                | 104             | 64    |
| 16 | 18  | 60                | 105             | 65    |
| 17 | 18  | 60                | 106             | 66    |
| 18 | 18  | 60                | 107             | 67    |
| 19 | 18  | 60                | 108             | 68    |
| 20 | 18  | 60                | 109             | 69    |
| 21 | 18  | 60                | 109             | 109   |

Fig.2.3

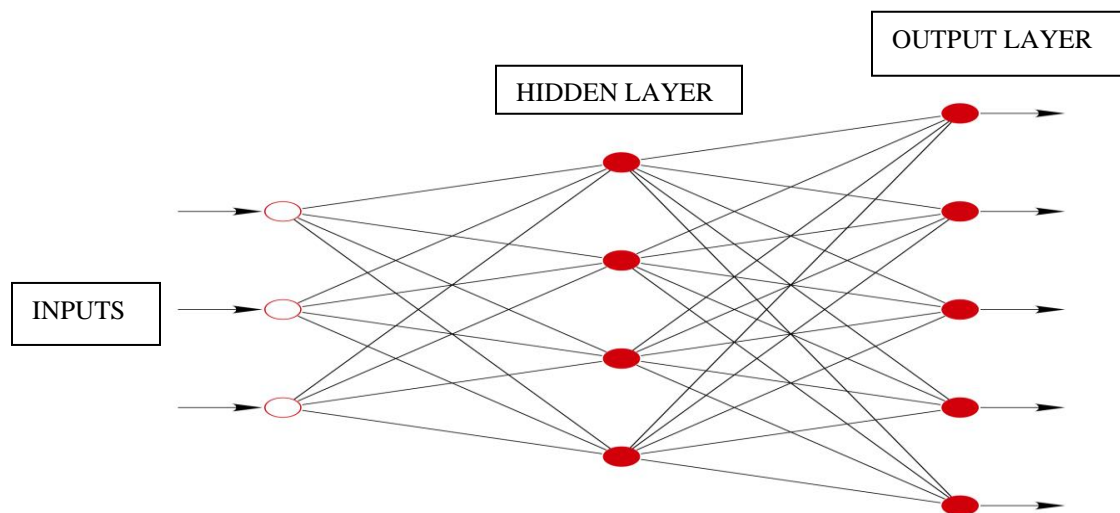


|       | AGE | HEART BE/SYSTOLIC | DIASTOLIC SUGAR | ALERT |
|-------|-----|-------------------|-----------------|-------|
| 99129 | 40  | 94                | 160             | 116   |
| 99130 | 40  | 94                | 161             | 117   |
| 99131 | 40  | 94                | 162             | 118   |
| 99132 | 40  | 94                | 163             | 119   |
| 99133 | 40  | 94                | 164             | 120   |
| 99134 | 40  | 94                | 165             | 81    |
| 99135 | 40  | 94                | 166             | 82    |
| 99136 | 40  | 94                | 167             | 83    |
| 99137 | 40  | 94                | 168             | 84    |
| 99138 | 40  | 94                | 169             | 85    |
| 99139 | 40  | 94                | 170             | 86    |
| 99140 | 40  | 94                | 171             | 87    |
| 99141 | 40  | 94                | 172             | 88    |
| 99142 | 40  | 94                | 173             | 89    |
| 99143 | 40  | 94                | 174             | 90    |
| 99144 | 40  | 94                | 175             | 91    |
| 99145 | 40  | 94                | 176             | 92    |
| 99146 | 40  | 94                | 177             | 93    |
| 99147 | 40  | 94                | 178             | 94    |
| 99148 | 40  | 94                | 179             | 95    |
| 99149 | 40  | 94                | 180             | 96    |

Fig.2.4

### 3.ARCHITECTURE

In designing an ANN architecture, we can start by selecting the number of neurons in the input and output layers. In the neural network two processes are applied of which one being feedforward and the other is backward propagation. A feedforward neural network is a biologically inspired classification algorithm. It consists of a (possibly large) number of simple neuron-like processing units, organized in layers. Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal, each connection may have a different strength or weight. Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called nodes.

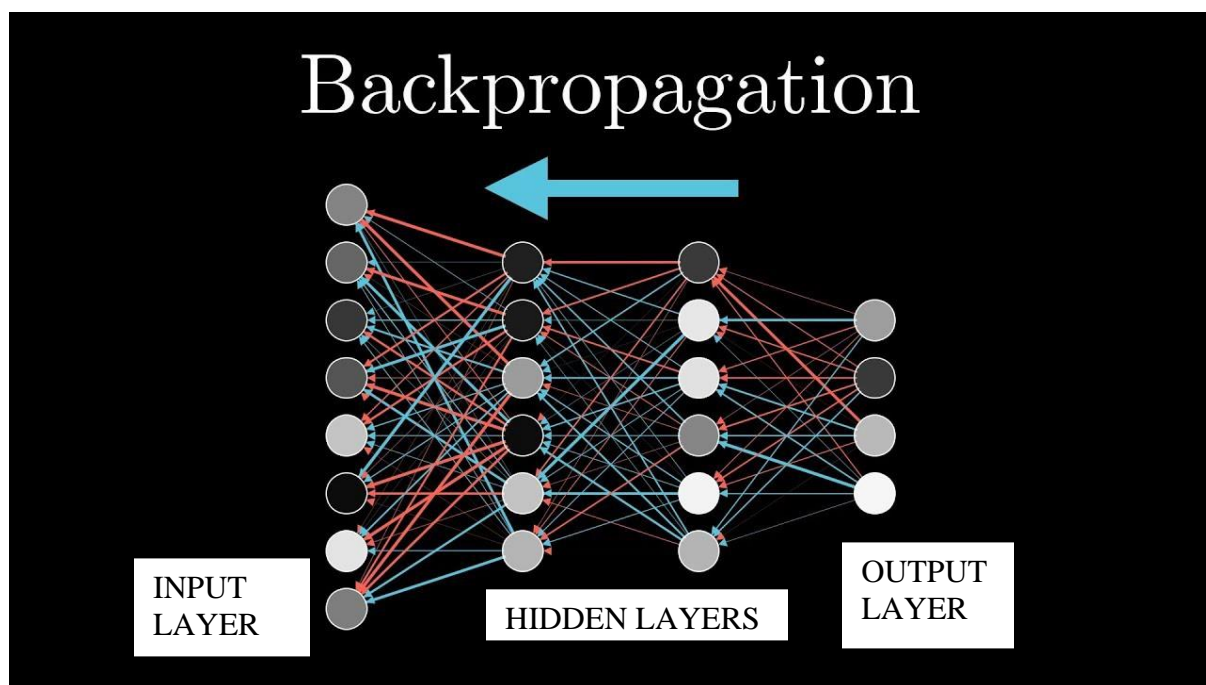


**Fig.3.1**

The 3 inputs are shown as circles and these do not belong to any layer of the network (although the inputs sometimes are considered as a virtual layer with layer number 0). Any layer that is not an output layer is a hidden layer. This network therefore has 1 hidden layer and 1 output layer.

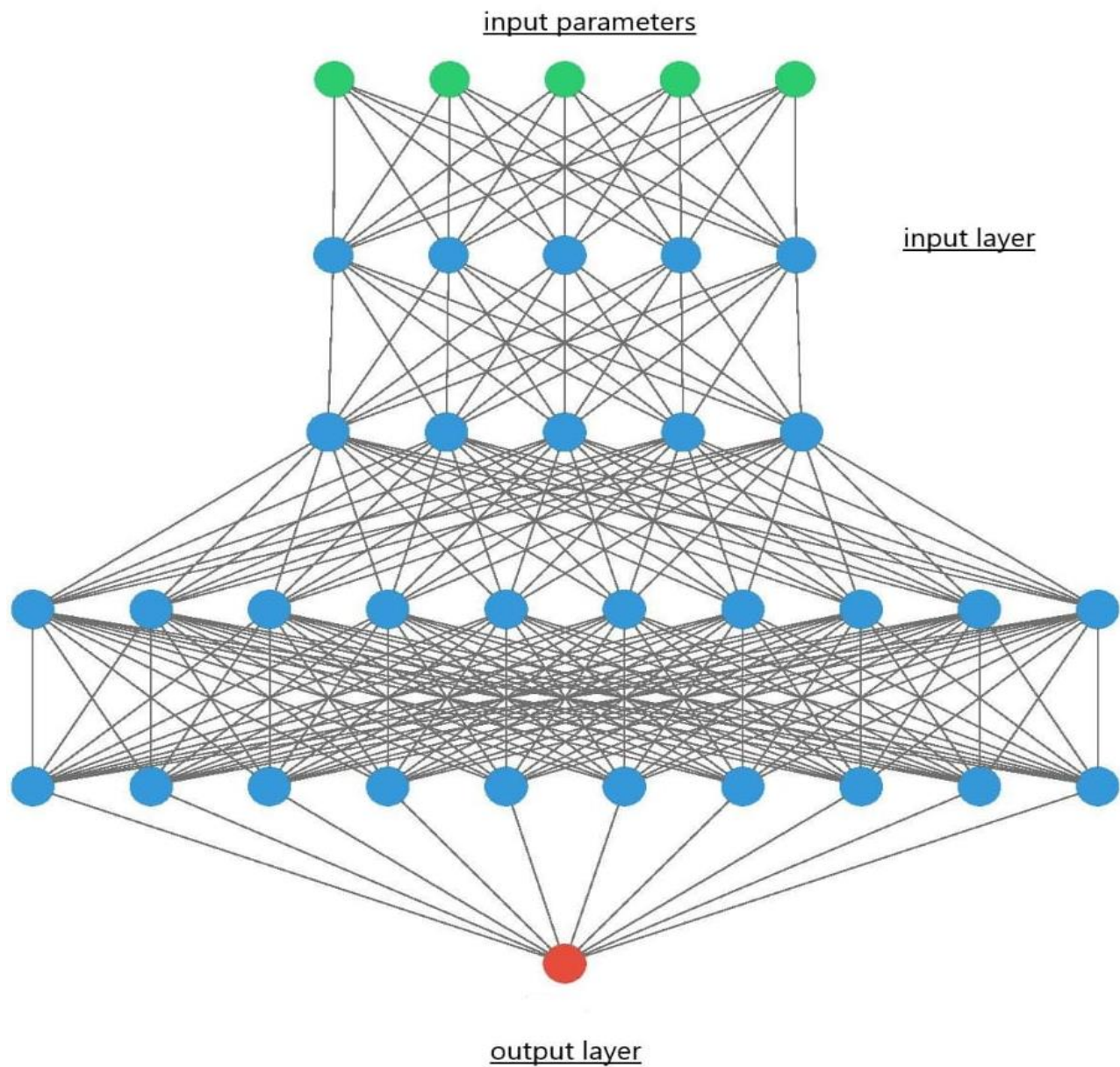
The Fig.3.1 also shows all the connections between the units in different layers. A layer only connects to the previous layer.

Back-propagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization [8].



**Fig.3.2**

The left most layer is the input layer and the right most layer is output layer. The layers in between the input and output layers are hidden layers. In backpropagation the inputs are processed and at the end of the hidden layer they are analysed with respect to inputs and the error is minimized by updating the weights and sent back to the first hidden layer for reprocessing.



**Fig.3.3**

This model uses 5 variables as inputs for each sample, thus there will be 5 input neurons considering the parameters age, pulse, blood pressure (systolic and diastolic), blood glucose and alert. In this we need to create an output neuron for output layer. To know what is the best number of hidden layers and hidden neurons to be used we should know the minimum number of connected lines to separate. Each value from the input layer is duplicated and sent to all the nodes in hidden layer. This is called a fully interconnected structure. The values of a hidden node are multiplied by weights, a set of predetermined numbers stored in the program. The weighted inputs are then added

to produce a single number. Before leaving the node, this number is passed through a nonlinear mathematical function called Relu. The image (Fig.3.1) shows three hidden layers in blue (2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> line). This architecture involves multi-layer perceptron structure. In this all the layers are fully connected i.e. every node in a layer (except the input and output layer) is connected to every node in the previous layer and the following layer.

The model involves target detection, the output layer only needs a single node, output of this node is threshold to provide a positive or negative indication of the target's presence or absence in the input data. Samples of 99419 are used for the model among which only 80% of the data is used for training and the remaining 20% of the data is for testing. In the data 5 parameters are directed to 5 input nodes. This network is a multilayer network and is fully connected. The input layer has 5 nodes. It has 3 hidden layers of which the first hidden layer has 5 nodes and the remaining two layers consist of 10 nodes each. Results are obtained in the output layer. The weights are selected automatically by the Adam's optimizer, the output can be configured to report a wide range of information. The output is 1 or 0 (yes/no) based on the inputs for the respective parameters.

## **3.2 ACTIVATION FUNCTIONS**

### **3.2.1 Activation function Rectified Linear Unit (ReLU)**

In the input layer and hidden layer's we have activation function as 'ReLU'. ReLu stands for Rectified Linear Unit. ReLu defined as the positive part of its argument and mathematically it is defined as  $R(z) = \max(0, z)$ . The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

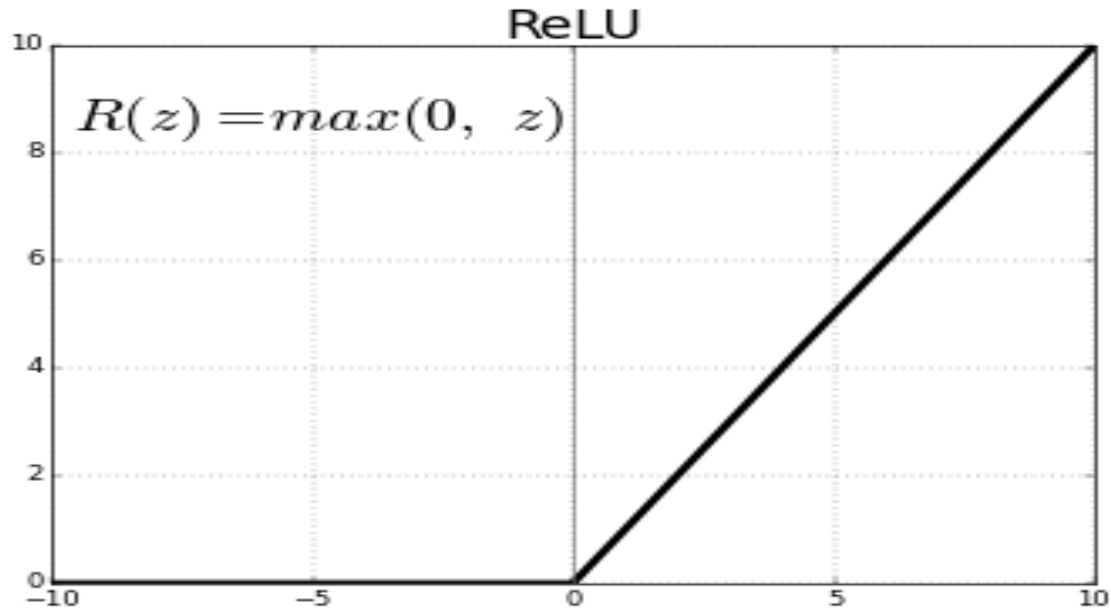


Fig.3.4

### 3.2.2 SIGMOID FUNCTION

This function is used in the output layer to convert data in the form of array to binary. With the help of sigmoid activation function, we are able to reduce the loss during the time of training because it eliminates the gradient problem in machine learning model while training.

$$f(x) = \sigma(x) = \frac{1}{(1+e^{-x})} \text{ -----Eq.1.}$$

The sigmoid function is used mostly used in classification type problem since we need to scale the data in some given specific range with a threshold. Sigmoid function is used in the cases where the output is binary (0 or 1) or there is only one node in output layer. The output is calculated using the formula,  $f(x) = \sigma(x) = 1 / (1 + e^{-x})$ . The output is binary because of the exponential function  $e^{-x}$ . Since the exponential component is in the denominator the result is in decimals.

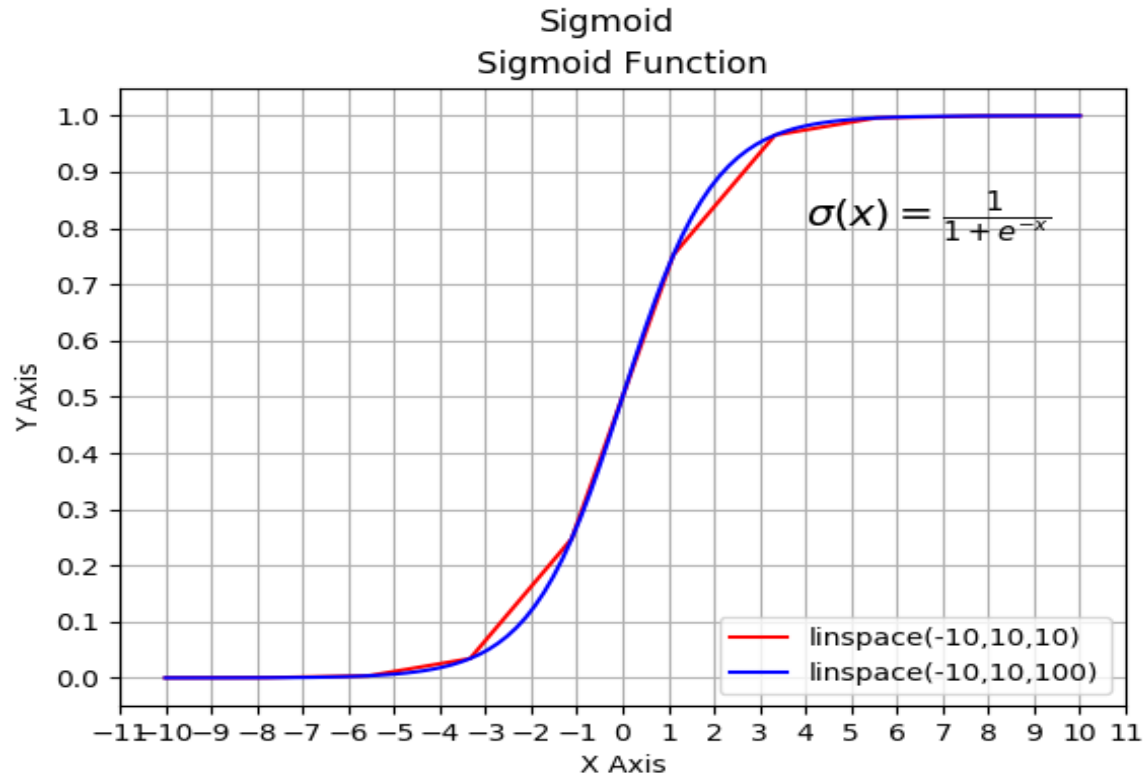


Fig.3.5

### 3.3 LOSS FUNCTION

In most learning networks, error is calculated as the difference between the actual output and the predicted output.

$$J(w) = y - \hat{Y} \text{ -----Eq.2.}$$

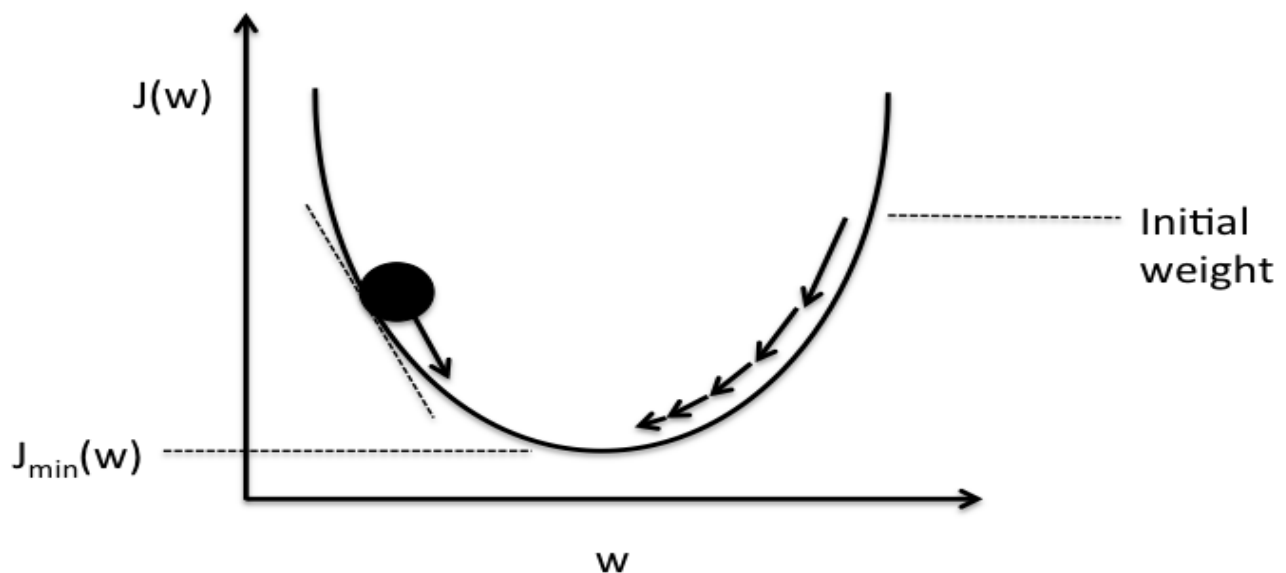
The function that is used to compute this error is known as Loss Function  $J(.)$ . Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model. One of the most widely used loss function is mean square error,



which calculates the square of difference between actual value and predicted value. Different loss functions are used to deal with different type of tasks, i.e. regression and classification. Then we use the gradient descent method to update the weights of the neural network such that the loss is minimized. There are 4 types of loss functions of which we are using only binary crossentropy.

$$\text{Cost function } J(w) = \frac{1}{m} \sum_{i=1}^m (y - y')^2 \text{-----Eq.3}$$

### 3.4 GRADIENT DESCENT



**Schematic of gradient descent.**

**Fig.3.6**

Gradient descent is an algorithm which calculates the loss at the end of the hidden layer and minimizes it. The minimized value is given to the input layer to update the weights and proceed to results. Gradient descent checks the values of the result in form of graph wherein it searches for the lowest value of error. Gradient descent results in global minimum.

Gradient descent is like a ball being pulled by gravity i.e. where ever the ball is released in sky it reaches the ground which is the minimum height.

### 3.5 Adam's Optimization

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. It is an adaptive learning rate optimization algorithm that is designed specifically for training deep neural networks to fine-tune the weights of our neural network in such a way that the errors are minimized. Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

Some advantages of Adam include:

Relatively low memory requirements (though higher than gradient descent and gradient descent with momentum).

Usually works well even with a little tuning of hyperparameters (except alpha).

#### **Back Propagation**

Calculate the error i.e. the difference between the actual output and the expected output. Depending on the error, adjust the weights by multiplying the error with the input and again with the gradient of the Sigmoid curve:

Weight += Error Input Output (1-Output)

Here,

Output (1-Output) is derivative of sigmoid curve.

Forward-propagation is the technique we will need to generate predictions during training that will need to be corrected, and it is the method we will need after the network is trained to make predictions on new data.

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Allows the information to go back from the cost backward through the network in order to compute the gradient. Therefore, loop over the nodes starting at the final node in reverse topological order to compute the derivative of the final node output with respect to each edge's node tail. Doing so will help us know who is responsible for the most error and change the parameters in that direction. The following derivatives' formulas will help us write the back-propagate functions: Since  $\mathbf{b}^l$  is always a vector, the sum would be across rows (since each column is an example).

## 4.RESULTS

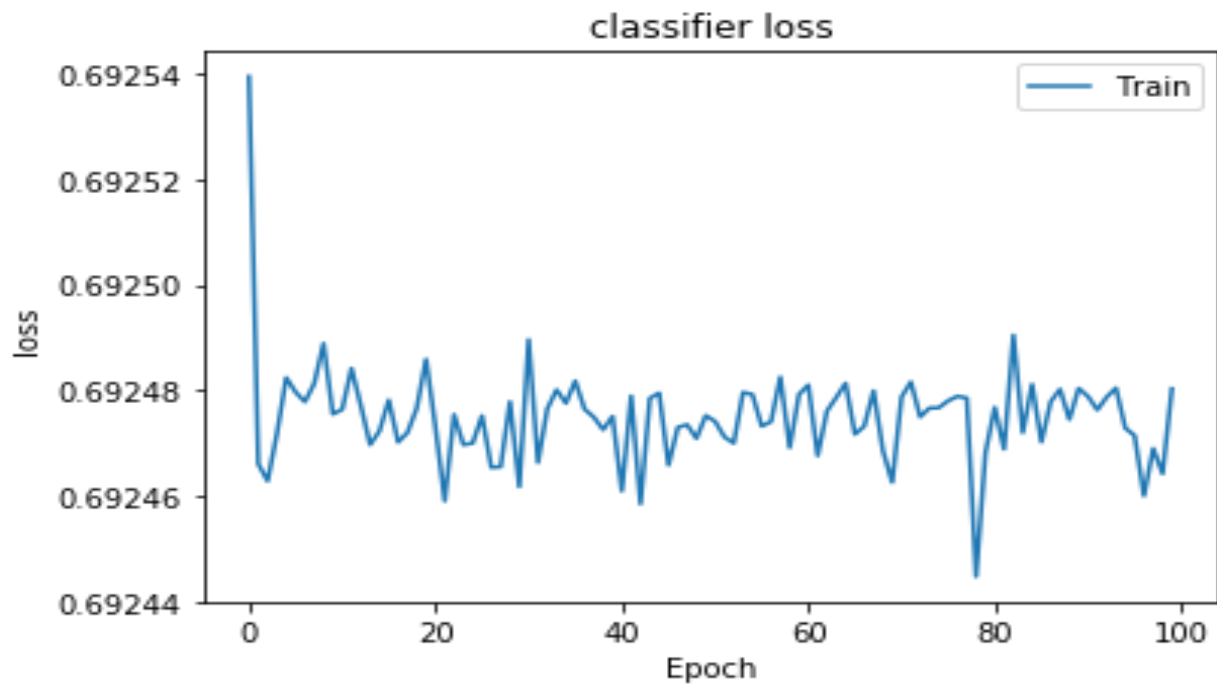
### 4.1 COMPARISION

The following are the different results based on the number of hidden layers. Upon increase of the hidden layers in the model the accuracy increases and the processing speed also increases. The predicted results to the actual results increased with the increase in the number of hidden layers. The accuracy for 1 hidden layer model, 2 hidden layer model and 3 hidden layer models are 51%, 73% and 100% respectively.

| <b>MODEL</b>                       | <b>TRAINING<br/>ACCURACY</b> | <b>VALIDATION<br/>ACCURACY</b> | <b>NUMBER OF<br/>TRAINING<br/>SAMPLES<br/>CLASSIFIED<br/>CORRECTLY</b> | <b>NUMBER OF<br/>VALIDATION<br/>SAMPLES<br/>CLASSIFIED<br/>CORRECTLY</b> |
|------------------------------------|------------------------------|--------------------------------|--|--|
| <b>ONE<br/>HIDDEN<br/>LAYER</b>    | <b>51%</b>                   | <b>90%</b>                     | <b>32,361/63454</b>  | <b>14,277/15864</b>  |
| <b>TWO<br/>HIDDEN<br/>LAYERS</b>   | <b>71%</b>                   | <b>93%</b>                     | <b>45,052/63454</b>  | <b>14,753/15864</b>  |
| <b>THREE<br/>HIDDEN<br/>LAYERS</b> | <b>100%</b>                  | <b>95.5%</b>                   | <b>63454/63454</b>   | <b>15150/15864</b>   |

Table 4.1

## USING SINGLE HIDDEN LAYER:



**Fig.4.1**

Epoch 1/100

79318/79318 [=====] - 2s 27us/step - loss: 0.6925 - accuracy:  
0.5186

Epoch 2/100

79318/79318 [=====] - 2s 23us/step - loss: 0.6925 - accuracy:  
0.5043

Epoch 3/100

79318/79318 [=====] - 2s 23us/step - loss: 0.6925 - accuracy:  
0.5186

Epoch 4/100

79318/79318 [=====] - 2s 24us/step - loss: 0.6925 - accuracy:  
0.5186

## USING TWO HIDDEN LAYERS:

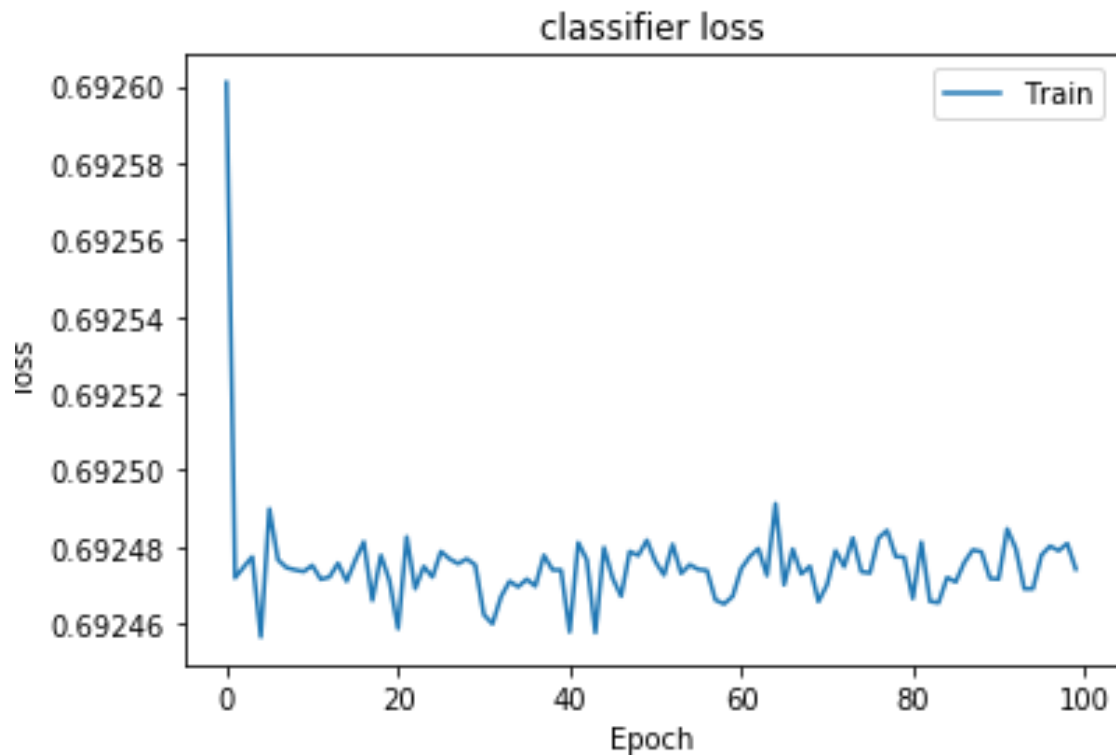


Fig.4.2

Epoch 1/100

79318/79318 [=====] - 2s 31us/step - loss: 0.2829 - accuracy:  
0.7171

Epoch 2/100

79318/79318 [=====] - 2s 26us/step - loss: 0.2814 - accuracy:  
0.7186

Epoch 3/100

79318/79318 [=====] - 2s 24us/step - loss: 0.2885 - accuracy:  
0.7115

Epoch 4/100

79318/79318 [=====] - 2s 24us/step - loss: 0.2502 - accuracy:  
0.74

### USING 3 HIDDEN LAYERS:

Epoch 1/100

69403/69403 [=====] - 3s 37us/step - loss: 0.4386 - accuracy:  
0.8249

Epoch 2/100

69403/69403 [=====] - 2s 27us/step - loss: 0.2761 - accuracy:  
0.9271

Epoch 3/100

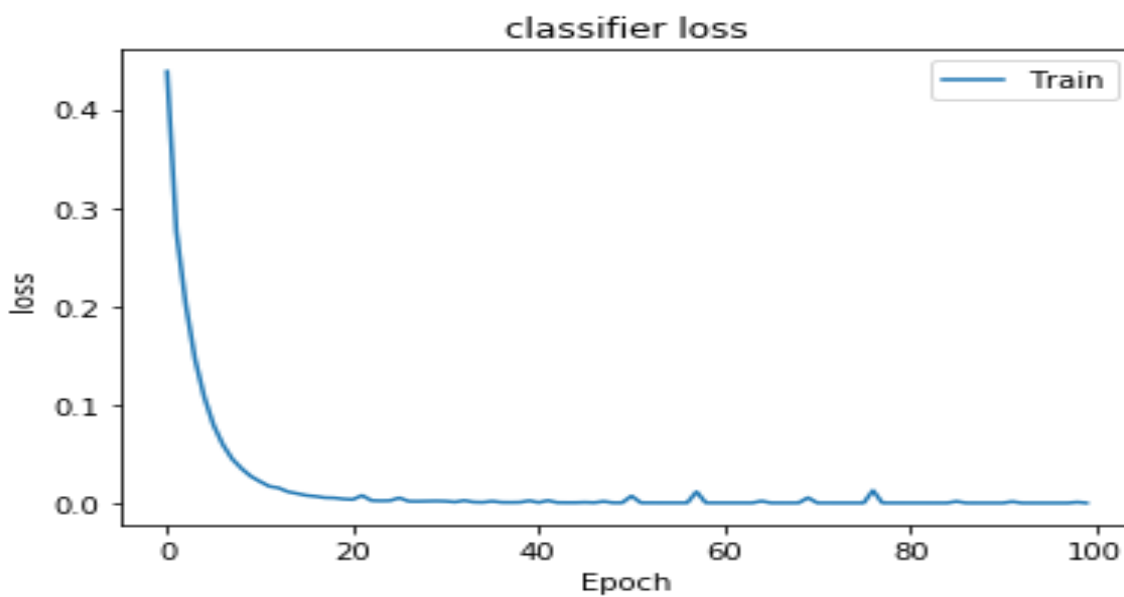
69403/69403 [=====] - 2s 27us/step - loss: 0.2023 - accuracy:  
0.9470

Epoch 4/100

69403/69403 [=====] - 2s 30us/step - loss: 0.0078 - accuracy:  
0.9979

Epoch 5/100

69403/69403 [=====] - 2s 27us/step - loss: 0.0026 - accuracy:  
1.0000



**Fig.4.3**

## ESTIMATIONS

```
In [48]: M pred=classifier.predict(np.array([[18,63,158,114,184]]))
if(pred > 0.5):
    pred =int(1)
else:
    pred =int(0)
pred
```

Out[48]: 1

```
In [49]: M pred=classifier.predict(np.array([[19,72,145,101,171]]))
if(pred > 0.5):
    pred =int(1)
else:
    pred =int(0)
pred
```

Out[49]: 1

```
In [57]: M pred=classifier.predict(np.array([[38,94,92,52,92]]))
if(pred > 0.5):
    pred =int(1)
else:
    pred =int(0)
pred
```

Out[57]: 0

```
In [51]: M pred=classifier.predict(np.array([[19,63,116,76,116]]))
if(pred > 0.5):
    pred =int(1)
else:
    pred =int(0)
pred
```

Out[51]: 0

```
In [35]: M pred=classifier.predict(np.array([[19,73,300,260,122]]))
if(pred > 0.5):
    pred =int(1)
else:
    pred =int(0)
pred
```

Out[35]: 1

**Fig.4.4**



## GRAPHS FOR VALIDATION ACCURACY AND LOSS

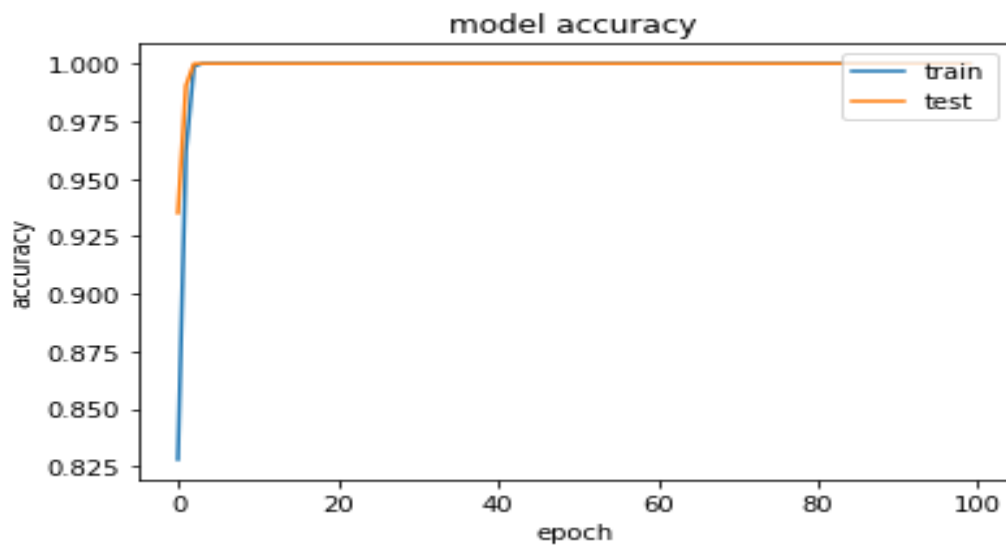


Fig.4.5

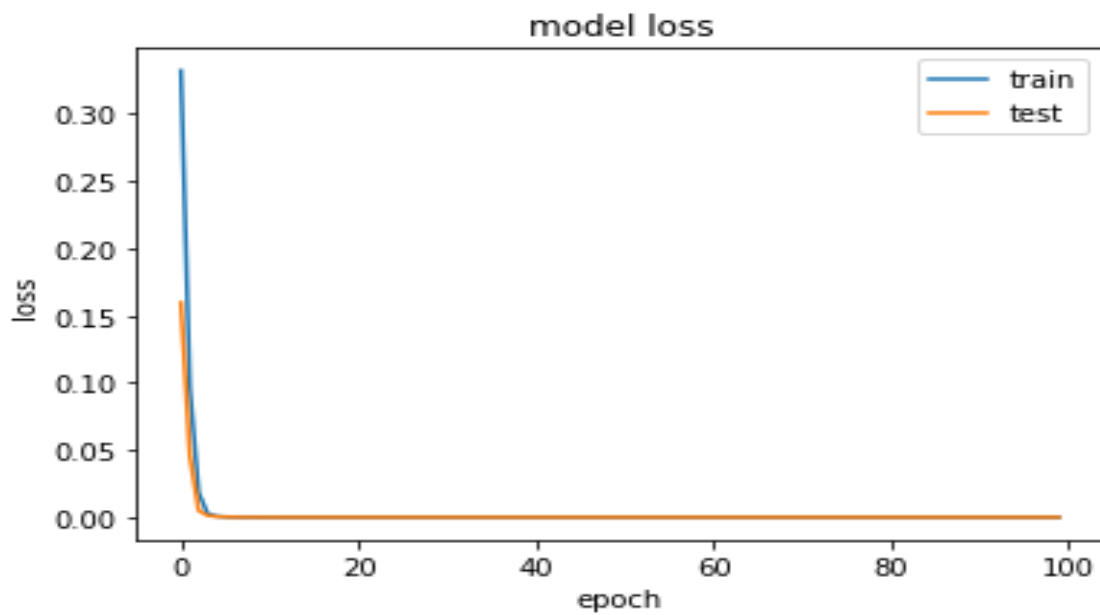
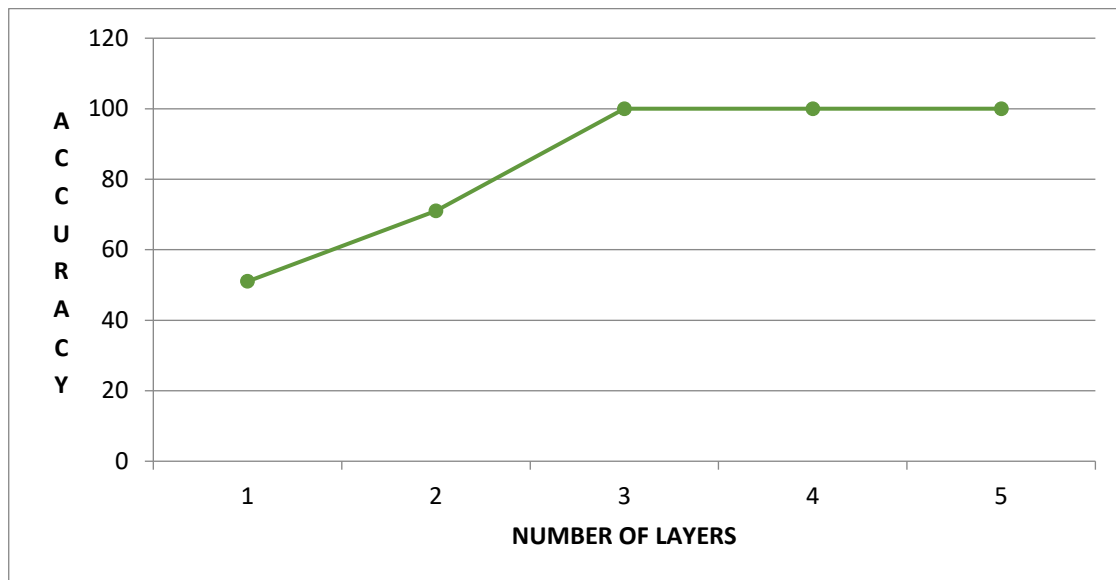


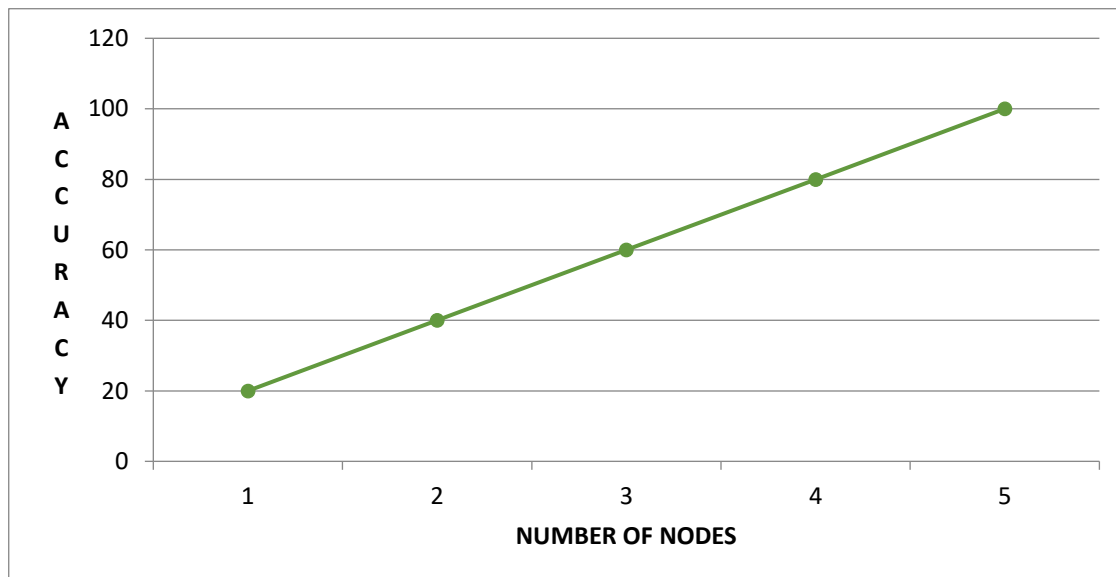
Fig.4.6

## GRAPHS FOR ACCURACY AND NUMBER OF HIDDEN LAYERS



**Fig. 4.7**

## GRAPHS FOR ACCURACY AND NODES



**Fig.4.8**

## 4.2 Description

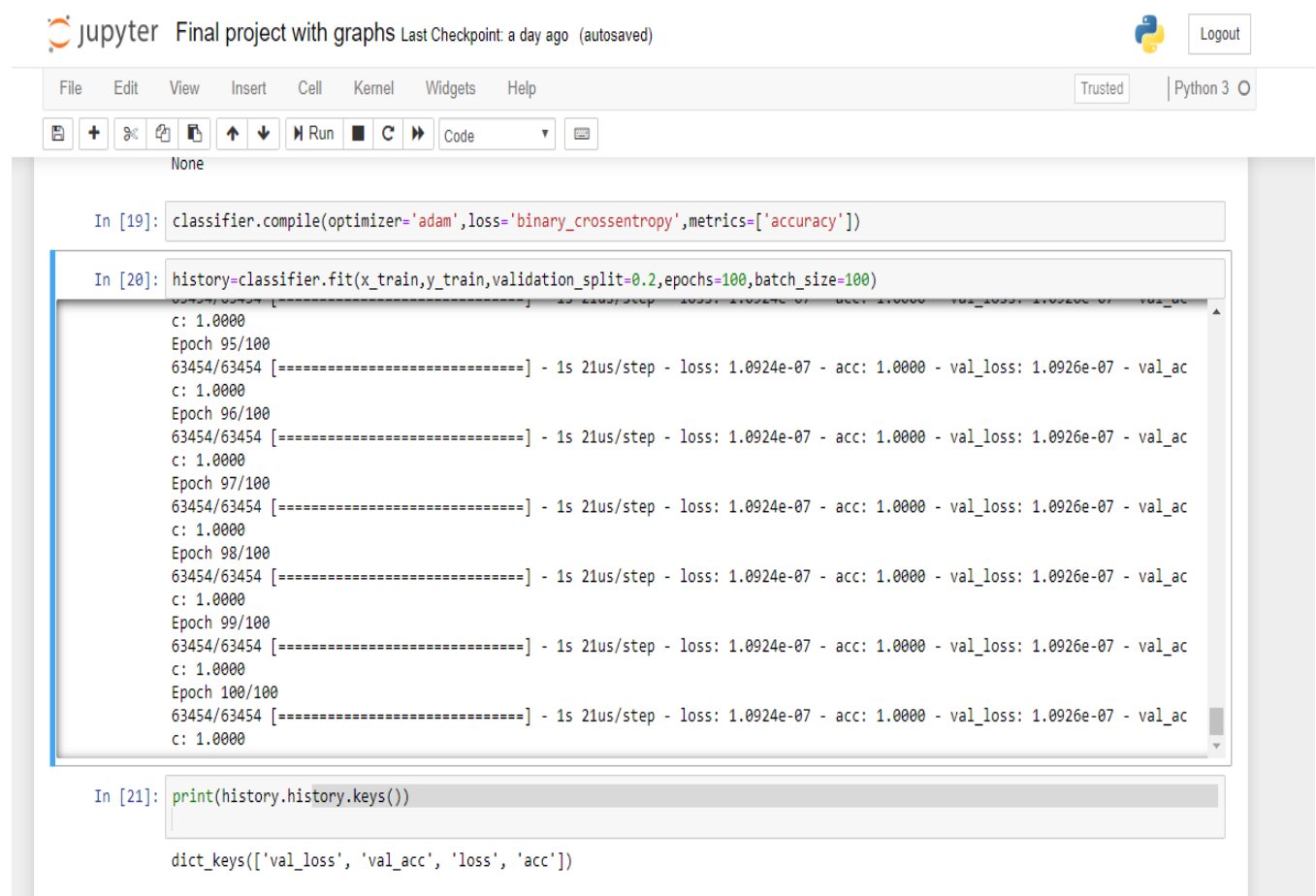
The below screenshots (Fig.4.10) represents the accuracy of classifier and clearly there are many factors to consider when judging a classifier result, like:

Variety of data it is trained on.

Quantity of data it is trained on.

Type of classifier used.

Tuning of the hyperparameters in the network.



The screenshot shows a Jupyter Notebook titled "Final project with graphs" with a "Last Checkpoint: a day ago (autosaved)" status. The interface includes a top bar with the Jupyter logo, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a right sidebar with "Trusted" and "Python 3" indicators. Below the menu bar is a toolbar with icons for file operations, running, and code execution. The notebook content consists of three code cells:

```
In [19]: classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
In [20]: history=classifier.fit(x_train,y_train,validation_split=0.2,epochs=100,batch_size=100)
```

The output of the second cell shows the training progress for 100 epochs. Each epoch displays the training accuracy (acc), validation accuracy (val\_acc), training loss (loss), and validation loss (val\_loss). The training accuracy is consistently 1.0000, while the validation accuracy is approximately 0.9926. The training loss is approximately 1.0924e-07, and the validation loss is approximately 1.0926e-07.

```
c: 1.0000
Epoch 95/100
63454/63454 [=====] - 1s 21us/step - loss: 1.0924e-07 - acc: 1.0000 - val_loss: 1.0926e-07 - val_ac
c: 1.0000
Epoch 96/100
63454/63454 [=====] - 1s 21us/step - loss: 1.0924e-07 - acc: 1.0000 - val_loss: 1.0926e-07 - val_ac
c: 1.0000
Epoch 97/100
63454/63454 [=====] - 1s 21us/step - loss: 1.0924e-07 - acc: 1.0000 - val_loss: 1.0926e-07 - val_ac
c: 1.0000
Epoch 98/100
63454/63454 [=====] - 1s 21us/step - loss: 1.0924e-07 - acc: 1.0000 - val_loss: 1.0926e-07 - val_ac
c: 1.0000
Epoch 99/100
63454/63454 [=====] - 1s 21us/step - loss: 1.0924e-07 - acc: 1.0000 - val_loss: 1.0926e-07 - val_ac
c: 1.0000
Epoch 100/100
63454/63454 [=====] - 1s 21us/step - loss: 1.0924e-07 - acc: 1.0000 - val_loss: 1.0926e-07 - val_ac
c: 1.0000
```

```
In [21]: print(history.history.keys())
```

The output of the third cell shows the keys of the history dictionary:

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

**Fig.4.9**

## **5.CONCLUSION**

Our study found artificial neural networks can be applied across all levels of health care organizational decision-making. Influenced by advancements in the field, decision-makers are taking advantage of hybrid models of neural networks in efforts to tailor solutions to a given problem. We found ANN-based solutions applied on the meso- and macro-level of decision-making suggesting the promise of its use in contexts involving complex, unstructured or limited information. The advance applications can help human beings in various aspects linking technology to wellness of the human beings. An appropriate model, proper dataset with variable parameters can help a model to estimate even diseases such as cancer well in advance. Loading the model into a hardware can be used as a portable device which can monitor the vitals of a particular human being at regular intervals of time. Further, more parameters can be used for a detailed study of the health condition of a human being.

## **FUTURESCOPE**

Artificial Neural Networks can be essential in many fields and the applications of ANN in the field of Medical Sciences do not end here as they can be designed in such a way that the patients can scan their body sitting at home. This reduces the duration for the regular scans as well as the doctor can give the required medication to the patient. So, the Health Monitoring System Using ANN can be integrated with the IOT devices and used in the hospitals to alert the hospital staff about the serious condition of the patient. The Health Monitoring System Using ANN is to be made mandatory in every hospital in order to save the patients in ICU's. With the upcoming technologies which tend to change the way humans interact with computers in a large scale few of the applications can be improved which use neural networks.

Creating software for government for storing citizens' data and sharing them with the hospital staff for future use.

Blood glucose samples to identify the stage of infection using artificial neural networks.

Using parent's genetics as data sets to find the features of the new generation.

Recognising human emotions and displaying their feelings using their heartbeats and skin temperature as datasets fed into neural network model. Estimating magnetic rays from sun to reduce the collision of cellular signals from clashing them.

Bio-Electronic implants inside the human bodies that measures the parameters second by second and perform particular task based on it.

## REFERENCES

- [1] “Make Your Own Neural Network: A Gentle Journey Through the Mathematics of Neural Networks, and Making Your Own Using the Python Computer Language” by Tariq Rashid
  
- [2] [www.tensorflow.org](http://www.tensorflow.org)
  
- [3] [www.numpy.com](http://www.numpy.com)
  
- [4] [www.matplotlib.com](http://www.matplotlib.com)
  
- [5] “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems” by Aurélien Géron
  
- [6] [https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning)
  
- [7] “Python Machine Learning: Unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics “by Sebastian Raschka
  
- [8] ”Nerual Networks a Comprehensive Foundations” by Simon Haykin
  
- [9] [www.webmd.com](http://www.webmd.com)

## APPENDIX

### CODE

```
# In[1]:

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.models import load_model
import numpy as np
import pandas as pd

# In[2]:

import graphviz

# In[3]:

dataset = pd.read_csv('C:/Users/sushma/Desktop/data/data sheet.csv')

# In[4]:

dataset

# In[6]:

dataset.head()

# In[7]:

x= dataset.iloc[:99149,:5].values

# In[8]:

type(x)

# In[9]:

x
```

```
# In[10]:
```

```
y= dataset.iloc[:99149,-1:].values
```

```
# In[11]:
```

```
y
```

```
# In[12]:
```

```
from sklearn.model_selection import train_test_split  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
# In[13]:
```

```
classifier=Sequential()
```

```
# In[14]:
```

```
classifier.add(Dense(units=3,kernel_initializer='uniform',activation='relu',input_dim=5))
```

```
# In[15]:
```

```
classifier.add(Dense(units=10,kernel_initializer='uniform',activation='relu'))
```

```
# In[16]:
```

```
classifier.add(Dense(units=10,kernel_initializer='uniform',activation='relu'))
```

```
# In[17]:
```

```
classifier.add(Dense(units=20,kernel_initializer='uniform',activation='relu'))
```

```
# In[18]:
```

```
classifier.add(Dense(units=1,kernel_initializer='uniform',activation='sigmoid'))
```

```
# In[19]:
```



```

classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

# In[20]:

hist=classifier.fit(x_train,y_train,epochs=100,batch_size=100)

# In[21]:

classifier.save('finalproject2.h5')

# In[22]:

y_pred=classifier.predict(x_test)

# In[23]:

l = len(y_pred)
l

# In[24]:

for i in range(l):
    if(y_pred[i] > 0.5):
        y_pred[i] =int(1)
    else:
        y_pred[i]=int(0)

# In[25]:

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm

# In[26]:

score=classifier.evaluate(x_test,y_test)
accuracy=score[1]

# In[27]:

```

# CODE EXPLANATION

## CONTENTS

IMPORT THE COLLECTED DATA SET

EXPLORE THE DATA

PREPROCESS THE DATA

BUILD THE MODEL [SET UP THE LAYERS, COMPILE THE MODEL]

TRAIN THE MODEL

EVALUATE THE ACCURACY

MAKE PREDICTIONS

This guide trains a neural network model to classify health condition of human, like pulse, blood pressure, sugar. It's okay if you don't understand all the details, this is a fast-paced overview of a complete TensorFlow program with the details explained as we go.

This guide uses [tf.keras](#), a high-level API to build and train models in TensorFlow.

**#TensorFlow and tf.keras**

```
import tensorflow as tf
```

```
from tensorflow import keras
```

**# Helper libraries**

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
print(tf.__version__)
```

**#Libraries**

```
import keras
```

```
from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.optimizers import SGD
from keras.models import load_model
import pandas as pd
```

## **IMPORT AND READ THE DATA**

### **Present directory name**

Code : pwd

Result : 'C:\\Users\\saivi'

Present directory name allows the user to know the location of the program with its resources.

### **To load the datasheet to program**

```
dataset = pd.read_csv('finalproject.csv')
```

pd.read allows the user to read the file which is in the format text.csv

### **dataset.head():**

This function allows the user to view the datasheet in short format (showing the first five rows of datasheet).

### **Assigning the rows to a variable:**

```
x= dataset.iloc[:99149,1:].values
```

This reads all the rows and all columns except the last. This is later assigned to variable x.

The dataset created is of sample size 99149 and 5 parameters. We will use 79318 samples to train the network and 19831 samples to evaluate how accurately the network learned to predict the output.

### **Training and Testing the data requires the following function:**

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Here the test size is used to partition the data as 80% for training and 20% for testing.

Random state: A function which enables the dataset to partition in same way as the rest of the inputs.

## BUILD THE MODEL

Building the neural network requires configuring the layers of the model, then compiling the model.

### Setup the layers

The basic building block of a neural network is the *layer*. Layers extract representations from the data fed into them. And, hopefully, these representations are more meaningful for the problem at hand.

Most of deep learning consists of chaining together simple layers. Most layers, like `tf.keras.layers.Dense`, have parameters that are learned during training.

```
classifier=Sequential()  
  
classifier.add(Dense(units=3,kernel_initializer='uniform',activation='relu',input_dim=5))  
  
classifier.add(Dense(units=10,kernel_initializer='uniform',activation='relu'))  
  
classifier.add(Dense(units=10,kernel_initializer='uniform',activation='relu'))  
  
classifier.add(Dense(units=20,kernel_initializer='uniform',activation='relu'))  
  
classifier.add(Dense(units=1,kernel_initializer='uniform',activation='sigmoid'))
```

### Sequential function: [Sequential()]

The sequential API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or output.

#### Sequential Models

```
from keras.models import Sequential  
from keras.layers import Dense
```

```
model = Sequential()  
model.add(Dense(2, input_dim=1))  
model.add(Dense(1))
```

In the given example, layers are added piecewise via the `Sequential` object.

The Sequential model API is great for developing deep learning models in most situations, but it also has some limitations. For example, it is not straightforward to define models that may have multiple different input sources, produce multiple output destinations, or models that re-use layers.

## INPUT LAYER

```
classifier.add(Dense(units=3, kernel_initializer='uniform', activation='relu', input_dim=5))
```

This layer uses the function `model.add(Dense(units, kernel_initializer, Activation function, input_dim))`

`model.add` is used to create a layer.

## Usage of initializers

Initializations define the way to set the initial random weights of Keras layers.

The keyword arguments used for passing initializers to layers will depend on the layer. Usually it is simply `kernel_initializer` and `bias_initializer`:

```
model.add(Dense(64, kernel_initializer='random_uniform', bias_initializer='zeros'))
```

## Available initializers

The following built-in initializers are available as part of the `keras.initializers` module:

[\[source\]](#)

### Initializer

```
keras.initializers.Initializer()
```

Initializer base class: all initializers inherit from this class.

### Zeros

```
keras.initializers.Zeros()
```

Initializer that generates tensors initialized to 0.

### Ones

```
keras.initializers.Ones()
```

Initializer that generates tensors initialized to 1.

Constant

```
keras.initializers.Constant(value=0)
```

Initializer that generates tensors initialized to a constant value.

**Arguments:**

**value:** float; the value of the generator tensors.

**RandomUniform**

```
keras.initializers.RandomUniform(minval=-0.05, maxval=0.05, seed=None)
```

Initializer that generates tensors with a uniform distribution.

**Arguments:**

**minval:** A python scalar or a scalar tensor. Lower bound of the range of random values to generate.

**maxval:** A python scalar or a scalar tensor. Upper bound of the range of random values to generate.

Defaults to 1 for float types.

**seed:** A Python integer. Used to seed the random generator.

## TRAIN THE MODEL

Training the neural network model requires the following steps:

1. Feed the training data to the model—in this example, the dataset is fed
2. The model learns to associate inputs to parameters.
3. We ask the model to make predictions about a test set—in this example, the dataset array. We verify that the predictions match the labels from the dataset array.

To start training, call the `model.fit` method—the model is "fit" to the training data:

Here `x_train` and `x_test` are the folders created by the model itself. The model is trained using the dataset which is divided into 79318 samples and the remaining 19831 samples are used for testing.

```
classifier.fit(x_train,y_train,epochs=100,batch_size=100)
```

```
Epoch 1/100 79318/79318 [=====] - 3s 40us/step - loss:
0.3639 - accuracy: 0.8353
```

```
Epoch 2/100 79318/79318 [=====] - 2s 27us/step - loss:
0.2612 - accuracy: 0.8920
```

```
Epoch 3/100 79318/79318 [=====] - 2s 27us/step - loss:
0.2525 - accuracy: 0.8959
```

```
Epoch 4/100 79318/79318 [=====] - 2s 27us/step - loss:
0.2463 - accuracy: 0.8989
```

```
Epoch 5/100 79318/79318 [=====] - 2s 27us/step - loss:
0.2382 - accuracy: 0.9023
```

```
Epoch 6/100 79318/79318 [=====] - 2s 27us/step - loss:
0.2251 - accuracy: 0.9076
```

```
Epoch 7/100 79318/79318 [=====] - 2s 27us/step - loss:
0.2052 - accuracy: 0.9162
```

```
Epoch 8/100 79318/79318 [=====] - 2s 27us/step - loss:
0.1465 - accuracy: 0.9408
```

```
Epoch 9/100 79318/79318 [=====] - 2s 27us/step - loss:
0.0522 - accuracy: 0.9812
```

```
Epoch 10/100 79318/79318 [=====] - 2s 27us/step - loss:
0.0271 - accuracy: 0.9912
```

The data set used in this program consists of 99,149 rows with 5 parameters assigned in their respective columns of which 79,318 rows are used for training the model.

As the model trains, the loss and accuracy metrics are displayed. This model reaches an accuracy of about 1.0 (or approximately 100%) on the training data.

## **SAVING THE PROGRAM**

After training the model the model needs to be saved for further usage. Once the model is trained it is saved using the extension .h5.

```
classifier.save('finalproject2.h5')
```

## **TESTING THE MODEL**

The model is tested by predicting the output using the following statements.

```
y_pred=classifier.predict(x_test)
```

The function classifier.predict is used to predict the dataset which is located in x\_pred.

To predict length of y user applies the following function

```
l = len(y_pred)
```

```
l
```

## **CODE FOR PREDICTION**

```
for i in range(l):
```

```
if(y_pred[i] > 0.5):
```

```
y_pred[i] =int(1)
```

```
else:
```

```
y_pred[i]=int(0)
```

## **LOAD THE MODEL**

```
classifier = load_model('finalproject2.h5')
```

## **CONFUSION MATRIX**

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions.



It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone.

|                   | <b>PREDICTED NO</b> | <b>PREDICTED YES</b> |
|-------------------|---------------------|----------------------|
| <b>ACTUAL YES</b> |                     |                      |
| <b>ACTUAL NO</b>  |                     |                      |

#### **How to calculate a confusion matrix:**

You need a test dataset or a validation dataset with expected outcome values. Make a prediction for each row in your test dataset. From the expected outcomes and predictions count:

The number of correct predictions for each class.

The number of incorrect predictions for each class, organized by the class that was predicted.

These numbers are then organized into a table, or a matrix as follows:

**Expected down the side:** Each row of the matrix corresponds to a predicted class.

**Predicted across the top:** Each column of the matrix corresponds to an actual class.

The counts of correct and incorrect classification are then filled into the table.

The total number of correct predictions for a class go into the expected row for that class value and the predicted column for that class value. In the same way, the total number of incorrect predictions for a class go into the expected row for that class value and the predicted column for that class value.

#### **CONFUSION MATRIX CODE**

```
from sklearn.metrics import confusion_matrix  
cm=confusion_matrix(y_test,y_pred)  
cm
```

## OUTPUT

```
array ([10204,      0],
       [ 0,      9626], dtype=int64)
```

## ACCURACY

```
score=classifier.evaluate(x_test,y_test)
accuracy=score[1]
```

## RESULT

```
19830/19830 [=====] - 1s 36us/step
```

## PREDICTING THE OUTPUT

```
pred=classifier.predict(np.array([[22,70,120,120,150]]))
if(pred > 0.5):
    pred =int(1)
else:
    pred =int(0)
pred
```

**Output: 1**