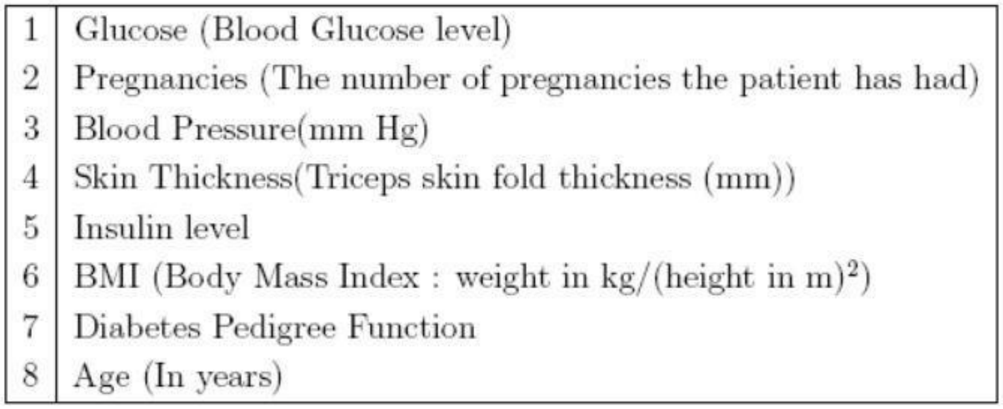
Binary Classification: Predicting Diabetes Using Neural Network

➢ Objective

The goal of this project is to perform **binary classification** using machine learning. Based on the diagnostic measurements provided in the dataset, the objective is to predict whether a patient has diabetes (**class 1**) or not (**class 0**). The project uses the **Pima Indians Diabetes Database (diabetes.csv)**. A **neural network model** is implemented for prediction, with two variations: one using **L2 regularization** and the other using **Dropout**, and their performances are compared.

➢ Dataset Description

- Source: Pima Indians Diabetes Database (diabetes.csv)
- Data-Info: The dataset comprises medical data from female patients over the age of 21
- Number of rows/records: 768
- Number of columns/attributes: Eight
- Target Variable: Outcome
- Key Features:



1	Glucose (Blood Glucose level)
2	Pregnancies (The number of pregnancies the patient has had)
3	Blood Pressure(mm Hg)
4	Skin Thickness(Triceps skin fold thickness (mm))
5	Insulin level
6	BMI (Body Mass Index : weight in kg/(height in m)^2)
7	Diabetes Pedigree Function
8	Age (In years)

➢ Python Environment and Libraries

The implementation was carried out in Google Colab. The project utilizes several Python libraries, including:

- Scikit-learn
- NumPy
- Pandas
- Matplotlib
- Seaborn
- Tensorflow
- Keras

➢ Data Processing

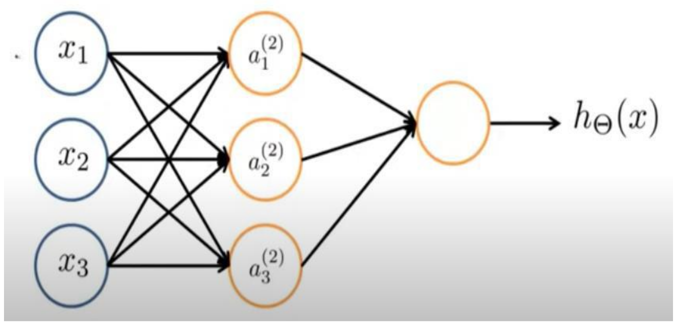
- Data Extraction: The original CSV file (diabetes.csv) was loaded into a Pandas DataFrame to extract and manage feature values. NumPy was then utilized for matrix-based operations during further processing and model preparation.
- Data Partitioning: Data is randomly split into training, validation, and test sets at a 60:20:20 ratio.
- Scaling: I normalized the data using a **mean-based scaling technique**, which centers each numeric feature around zero and scales it relative to the feature range. This brings all features to a **similar scale**, improving model training stability. The scaling formula is:

$$X_i^{\text{scaled}} = \frac{X_i - \text{mean}(X)}{\max(X) - \min(X)}$$

➢ Neural Network Implementation

- **Neural Network model creation**: I built a Neural Network model with a 100-(500-500)-1 architecture using Keras. This model is a feedforward multi-layer perceptron arranged sequentially.
    - The input layer has 100 neurons, which receive the feature values.
    - The network has two hidden layers, each with 500 neurons, to learn complex patterns in the data.
    - The output layer consists of a single neuron that produces a binary output using sigmoid function: either class 0 or class 1.

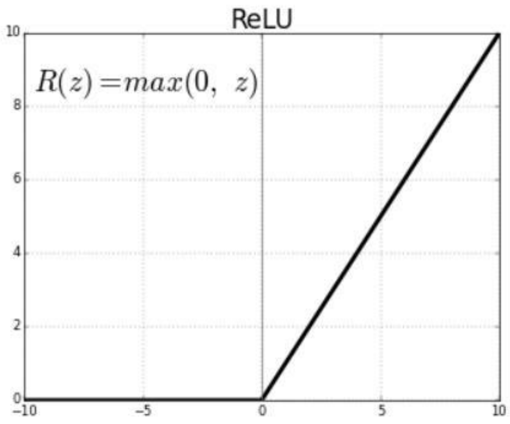
  To ensure reproducibility, the model weights are initialized using a fixed seed value of 6.



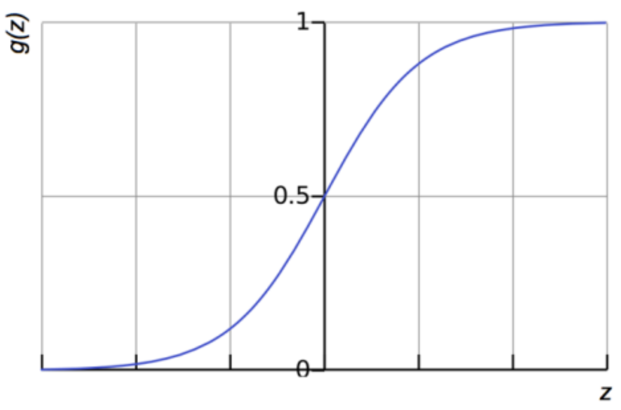
- **Forward propagation**:
  I inserted feature values into the input layer in the first phase. The data moves to the next layer based on the weight and input feature values. Each node's output is calculated using the linear function. The next node then transforms the input data using the ReLu function, and this value, together with the weight, is inserted into the linear function to transmit the signal to the next node. This procedure will be repeated till the final layer is reached. While, in the last layer a sigmoid activation function is used. In this model X0 is the initial bias, thetas are weights and g(theta) are the ReLu/sigmoid functions.

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)\\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)\\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)
\end{aligned}
$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$



ReLu function.



Sigmoid activation function.

- **Training the model**

In the training process I have used ***stochastic gradient descent*** optimizer function for optimizing the weights. Further I have used the ***binary cross entropy*** method to calculate the loss. The batch size is set at 20. While the epochs after multiple attempts is set at 200. After 200 iteration the neural network model is trained and the weights are converged at the local minima.

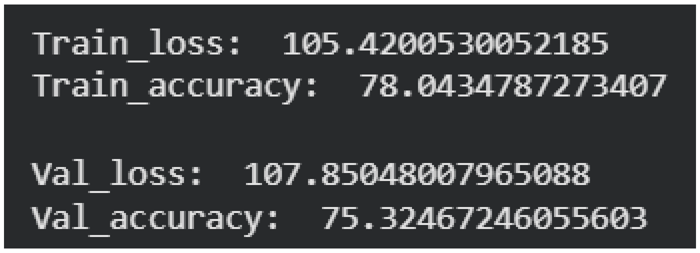
$$\boxed{Loss = L(y, \hat{p}) = -y\log(\hat{p}) - (1-y)\log(1-\hat{p})}$$

where $y \Rightarrow$ label,
$\hat{p} \Rightarrow$ estimated probability of belonging to the positive class

The Binary Cross-Entropy Loss function

➢ **Calculating Loss and accuracy for training data and validation data**

After training the model, the ***loss*** and ***accuracy*** of the training data obtained is around 105 and **78%** respectively. While the loss and accuracy of the validation data is around 107 and **77%** respectively.

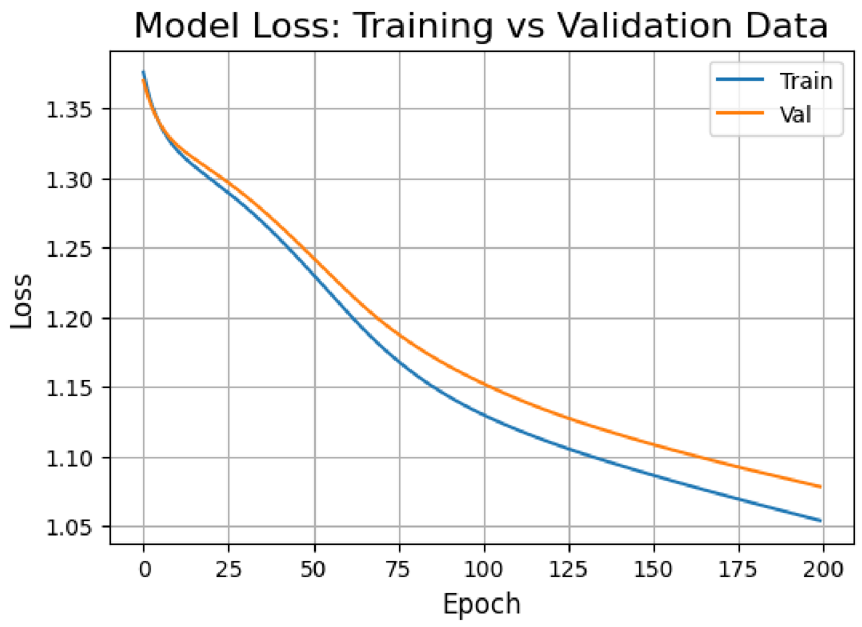


```
Train_loss:  105.4200530052185
Train_accuracy:  78.0434787273407

Val_loss:  107.85048007965088
Val_accuracy:  75.32467246055603
```

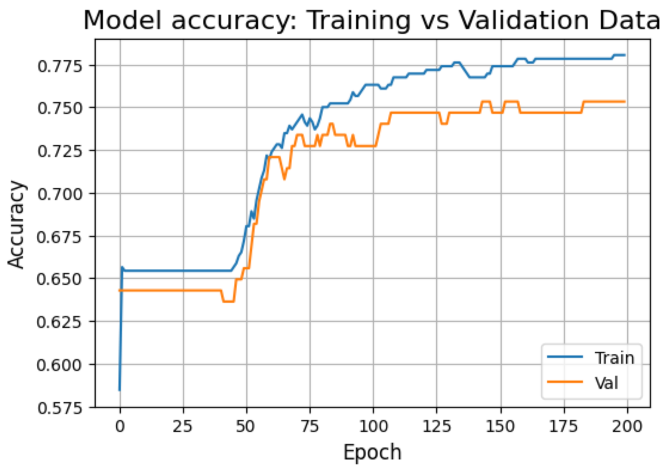
➢ **The Plot of loss Function**

Loss function *comparison* for training data and validation data set is below

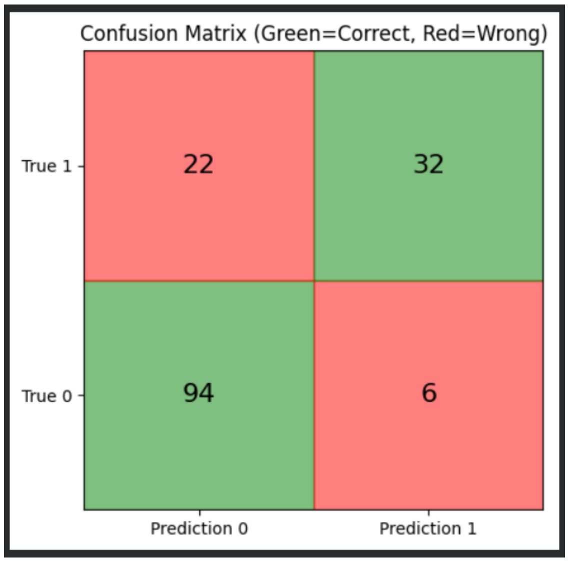


- **The Plot of Accuracy**

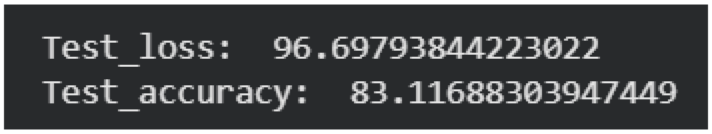
  Accuracy *comparison* for training data and validation data set is below



- **The Plot of Confusion Matrix**



- **Measuring the strength of model**
  The accuracy obtained for the test data is around **83%**.



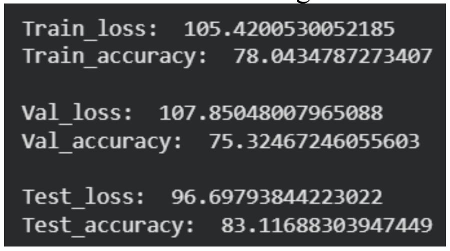
```
Test_loss:  96.69793844223022
Test_accuracy:  83.11688303947449
```

The neural network model is making a prediction with an accuracy of 83%. Therefore, this model can be effectively used to detect the cases of diabetes in the patients based on their input features/ health reports.

- **Comparison of Neural Network Implementation for different regularization method - L2 and Dropout**
  - **L2 regularization method:**
    First, I have used the L2 regularization method to train the model. After that accuracy and loss are calculated for training, validation, and test data sets. For L2 regularization I have set a flag value of zero '0', when the code encounter flag as zero, the L2 regularization model is initiated. L2 is called the Ridge Regression which adds "squared magnitude of the coefficient as penalty term to the loss function. The highlighted part below represents the L2 regularization. L2 regularization is used to avoid over-fitting of the curve. Here, I have used lambda as 'lamb = 0.001' in the training of the neural network model.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

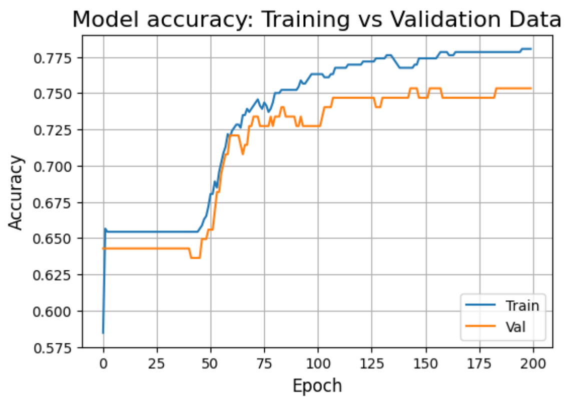
Loss function for L2 Regularization

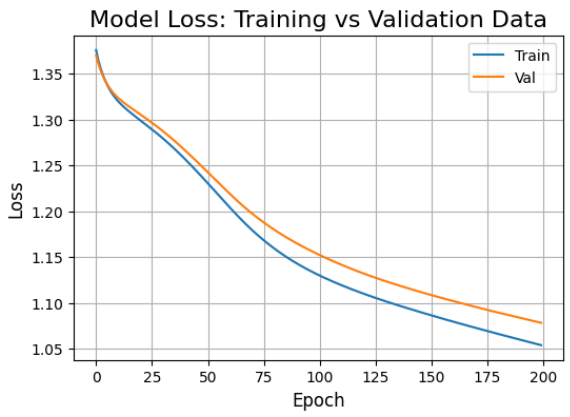


```
Train_loss:  105.4200530052185
Train_accuracy:  78.0434787273407

Val_loss:  107.85048007965088
Val_accuracy:  75.32467246055603

Test_loss:  96.69793844223022
Test_accuracy:  83.11688303947449
```



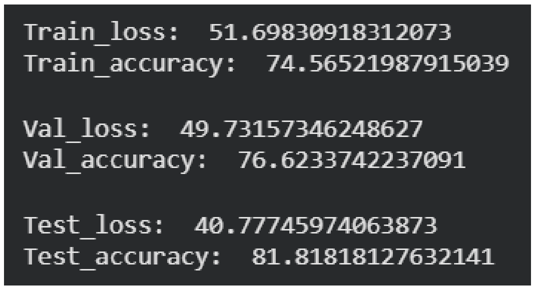


The loss and accuracy obtained for training, validation and test data is above.

- **Dropout Regularization method:**
  A dropout is a regularization approach that approximates simultaneous training of a large number of neural networks with various topologies. During training, certain layer outputs are disregarded or "dropped out" at random. This has the effect of having the layer seem and be regarded as if it had a different number of nodes and connections to the preceding layer. Each update to a layer during training, in effect, is conducted with a distinct "view" of the specified layer.

  Dropout has the effect of making the training process noisy, pushing nodes within a layer to take on more or less responsible for the inputs on a probabilistic basis. This approach implies that dropout may be used to break up situations in which network levels co-adapt to rectify mistakes made by previous layers, making the model more resilient.

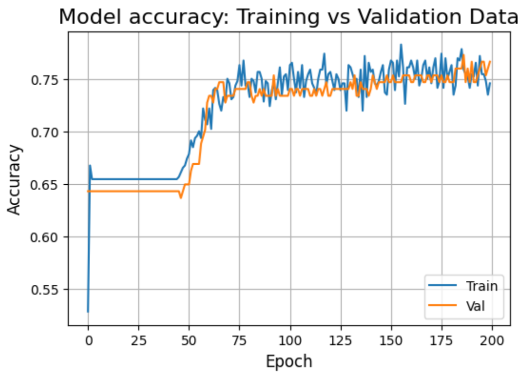
  I have set a flag value of '1' for Dropout regularization, thus when the code encounters a flag value of one, Dropout regularization will be triggered for model training.

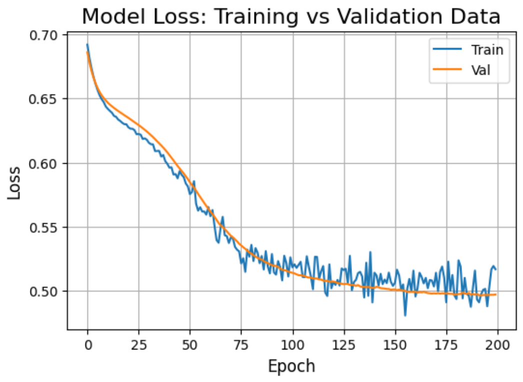


```
Train_loss:  51.69830918312073
Train_accuracy:  74.56521987915039

Val_loss:  49.73157346248627
Val_accuracy:  76.6233742237091

Test_loss:  40.77745974063873
Test_accuracy:  81.81818127632141
```





The loss and accuracy obtained for training, validation and test data is above.

- **Comparison of L2 and Dropout method:**

Metrics	**L2 Regularizer**	**Dropout Regularizer**
Loss (Training)	105	52
Accuracy (Training)	78	75
Loss (Validation)	108	50
Accuracy (Validation)	75	77
Loss (Test)	97	41
Accuracy (Test)	83	82

From the table, it can be seen that the loss value in the Dropout method is relatively low compared to the L2 method. While accuracy is very much similar for all data sets in both the cases.