# 18CSE388T - ARTIFICIAL NEURAL NETWORKS

# PROJECT FINAL REPORT


## Team Member:

- Aranya Banerjee(RA1811026010027)
- Ajay Bhadoria (RA1811026010064)


## The objective of Project:

To build a neural network based on attributes such as BMI, age, diastolic blood pressure etc. for classifying whether a patient has diabetes or not.

## Datasets:

- We will use the pima Indians dataset form UCL Machine Learning repository which describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.
- The original Dataset link:- https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

## Models Used:

➔ In this project, the model used is a *binary classification.*

## Details Of The Experiment:

The project includes TensorFlow, Keras, NumPy, pandas, and scikit-learn. The steps used in this project are :

### 1) Load data –

We first define the functions and classes we intend to use. We will use the NumPy library to load our dataset and we will use two classes(sequential, dense) from the Keras library to define our model. We have used the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years. As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input variables that describe each patient are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values. There are eight input variables and one output variable (the last column). These are as follows:

- **Input Variables (X):**

- Number of times pregnant
- Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- Diastolic blood pressure (mm Hg)
- Triceps skin fold thickness (mm)
- 2-Hour serum insulin (mu U/ml)
- Body mass index (weight in kg/(height in m)^2)
- Diabetes pedigree function
- Age (years)

- **Output Variables (y):**

- Class variable (0 or 1

## 2) Define keras model –

Models in Keras are defined as a sequence of layers. We have created a Sequential model and add layers one at a time. Fully connected layers are defined using the Dense class. We have specified the number of neurons or nodes in the layer as the first argument, and specify the activation function using the activation argument. We will use the rectified linear unit activation function referred to as ReLU on the first two layers and the Sigmoid function in the output layer.

## 3) Compile keras model –

We have compiled the Keras model using TensorFlow. We will use cross-entropy as the loss argument. This loss is for binary classification problems and is defined in Keras as “binary_crossentropy“. We have defined the optimizer as the efficient stochastic gradient descent algorithm “adam“. This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems.

## 4) Fit keras model –

We can train or fit our model on our loaded data by calling the fit() function on the model.Training occurs over epochs and each epoch is split into batches.

Epoch: One pass through all of the rows in the training dataset.
Batch: One or more samples considered by the model within an epoch before weights are updated.

## 5) Evaluate keras model –

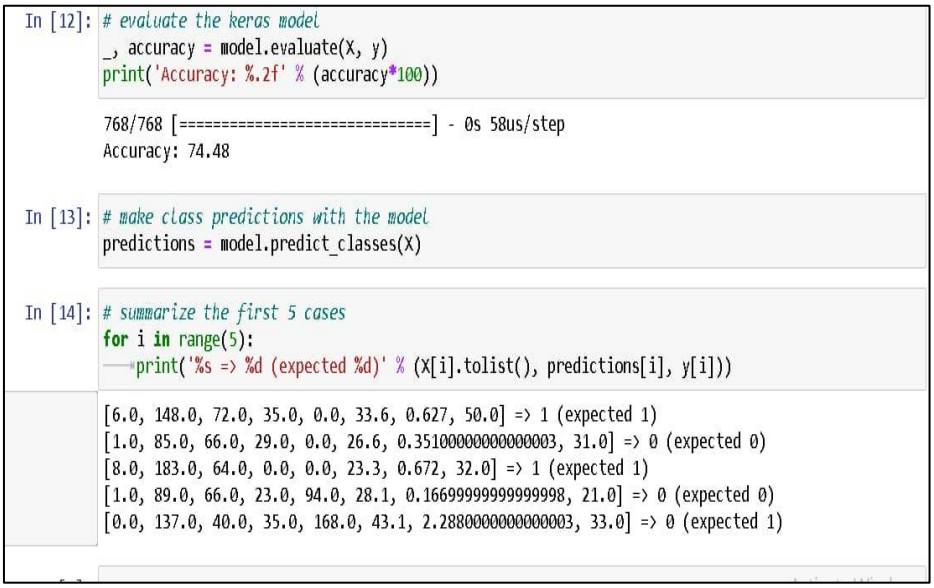
We have evaluated our model on your training dataset using the evaluate() function and have passed it the same input and output used to train the model. The evaluate() function will return a list with two values. The first will be the loss of the

model on the dataset and the second will be the accuracy of the model on the dataset. We are only interested in reporting the accuracy, and so have ignored the loss value.

## 6) Making predictions –

We have used the predict() function on the model, to make predictions. We are using a sigmoid activation function on the output layer, so the predictions will be a probability in the range between 0 and 1. We can easily convert them into a crisp binary prediction for this classification task by rounding them.

## Result –



```
In [12]: # evaluate the keras model
         _, accuracy = model.evaluate(X, y)
         print('Accuracy: %.2f' % (accuracy*100))

         768/768 [==============================] - 0s 58us/step
         Accuracy: 74.48

In [13]: # make class predictions with the model
         predictions = model.predict_classes(X)

In [14]: # summarize the first 5 cases
         for i in range(5):
             print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

         [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
         [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.35100000000000003, 31.0] => 0 (expected 0)
         [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
         [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.16699999999999998, 21.0] => 0 (expected 0)
         [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.2880000000000003, 33.0] => 0 (expected 1)
```

In [11]:
```
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
```

```
Epoch 1/150
768/768 [==============================] - 4s 5ms/step - loss: 13.8119 - accuracy: 0.4193
Epoch 2/150
768/768 [==============================] - 0s 196us/step - loss: 1.9510 - accuracy: 0.6120
Epoch 3/150
768/768 [==============================] - 0s 191us/step - loss: 1.2116 - accuracy: 0.6536
Epoch 4/150
768/768 [==============================] - 0s 156us/step - loss: 1.0456 - accuracy: 0.6445
Epoch 5/150
768/768 [==============================] - 0s 175us/step - loss: 0.8912 - accuracy: 0.6380
Epoch 6/150
768/768 [==============================] - 0s 165us/step - loss: 0.7656 - accuracy: 0.6628
Epoch 7/150
768/768 [==============================] - 0s 174us/step - loss: 0.7273 - accuracy: 0.6576
Epoch 8/150
768/768 [==============================] - 0s 213us/step - loss: 0.6989 - accuracy: 0.6576
Epoch 9/150
768/768 [==============================] - 0s 156us/step - loss: 0.6793 - accuracy: 0.6549
Epoch 10/150
768/768 [==============================] - 0s 173us/step - loss: 0.6836 - accuracy: 0.6510
```