In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as Tnr
import sklearn
#Read the training data
train_data = pd.read_csv('diabetes dataset.csv')
```

In [2]:
```
#Print the data
print(train_data.head())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

In [3]:
```
#Print the dimesnion of the data
train_data.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
#separating X train and Y train
X_train = train_data[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeF
Y_train = train_data[['Outcome']]
```

In [5]:
```
# importing train_test_split from sklearn
from sklearn.model_selection import train_test_split
# splitting the data
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size = 0.3, random_state = 0)
```

In [6]:
```
#print the shape of train and test data after splting
print (X_train.shape)
print (Y_train.shape)
print (X_test.shape)
print (Y_test.shape)
```

```
(537, 8)
(537, 1)
(231, 8)
(231, 1)
```

In [7]:
```
from keras.layers import Dense
from keras.models import Sequential
```

In [8]:
```
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

In [9]:
```
model.compile(optimizer=Tnr.keras.optimizers.Adam(learning_rate=0.001),
                loss=Tnr.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])
model.fit(X_train, Y_train, epochs=500, batch_size=32, verbose=1)
```

```
Epoch 1/500
17/17 [==============================] - 1s 2ms/step - loss: 0.7079 - accuracy: 0.4786
Epoch 2/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6478 - accuracy: 0.6387
Epoch 3/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6446 - accuracy: 0.6387
Epoch 4/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6372 - accuracy: 0.6387
Epoch 5/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6318 - accuracy: 0.6387
Epoch 6/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6273 - accuracy: 0.6425
Epoch 7/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6222 - accuracy: 0.6387
Epoch 8/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6181 - accuracy: 0.6369
Epoch 9/500
17/17 [==============================] - 0s 1ms/step - loss: 0.6135 - accuracy: 0.6443
Epoch 10/500
```

In [10]:
```
# Evaluate the model on the test set
test_loss, test_acc1 = model.evaluate(X_test, Y_test, verbose=0)
```

In [11]:
```
# Build the model with ReLU activation function
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [12]:
```
# Compile the model
model.compile(optimizer=Tnr.keras.optimizers.Adam(learning_rate=0.001),
                loss=Tnr.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

In [13]:
```
# Train the model
model.fit(X_train, Y_train, epochs=500, batch_size=32, verbose=1)
```

```
Epoch 1/500
17/17 [==============================] - 1s 1ms/step - loss: 14.5122 - accuracy: 0.3892
Epoch 2/500
17/17 [==============================] - 0s 1ms/step - loss: 3.6641 - accuracy: 0.5289
Epoch 3/500
17/17 [==============================] - 0s 1ms/step - loss: 1.9744 - accuracy: 0.4804
Epoch 4/500
17/17 [==============================] - 0s 1ms/step - loss: 1.2619 - accuracy: 0.5233
Epoch 5/500
17/17 [==============================] - 0s 1ms/step - loss: 1.0726 - accuracy: 0.5754
Epoch 6/500
17/17 [==============================] - 0s 1ms/step - loss: 0.9117 - accuracy: 0.6034
Epoch 7/500
17/17 [==============================] - 0s 1ms/step - loss: 0.8478 - accuracy: 0.5922
Epoch 8/500
17/17 [==============================] - 0s 1ms/step - loss: 0.9345 - accuracy: 0.5810
Epoch 9/500
17/17 [==============================] - 0s 1ms/step - loss: 0.8609 - accuracy: 0.6034
Epoch 10/500
```

In [14]:
```
# Evaluate the model on the test set
test_loss, test_acc2 = model.evaluate(X_test, Y_test, verbose=0)
print("Test accuracy with sigmoid activation:", test_acc1)
print("Test accuracy with ReLU activation:", test_acc2)
```

```
Test accuracy with sigmoid activation: 0.7142857313156128
Test accuracy with ReLU activation: 0.7186146974563599
```

In [ ]: