# Predicting Bank's Customer Churn Rate using Machine Learning

**Project**

**MOOC's on**

**Theoretical Machine Learning**

**Submitted By:**

**Adesh (RA 2011026010421)**

**Mukesh S (RA 2011026010423)**

**Bharathi Subrahmanian (RA 2011026010440)**

**Harshit (RA 2011026010447)**

## Introduction

Recently, a bank with millions of customers has noticed unusual number of customers leaving the bank. To make sure that this negative development will be stopped on time, the bank rest needs to identify the customers who have high likelihood to churn. Identifying these customers will help the bank also to avoid repeating the same mistakes with the rest of the customers and avoid them leaving the bank from the early stages. In our analysis, we aim to predict the churning rate of the customers from the bank. From the millions of customers, we have randomly selected 10K customers. We will use customer's characteristics to determine his/her probability of leaving the bank. To learn about bank's customers, we will make use of one of the Deep Learning techniques, the Artificial Neural Networks (ANN). Moreover, we will use popular Python libraries such as Tensorow, Keras and Machine Learning techniques such as Adam Optimizer to train the ANN model and predict the churn rates.

## Data

The data contains 13 variables characterizing 10L bank customers. These variables are Customer Id Surname, CreditScore, Geography Gender, Age, Tenure Balance, NumOf- Products, HasCrCard, IsActiveMember, EstimatedSalary and Exited. The bank has also observed these customers over the period of last 6 months to out whether they left the bank during this time or not. Customer Id and Surname are both string type of variables representing unique identifiers for the bank's customers. Geography is a string type of categorical variable, where the values have no sequential order, containing information about the location of the customer. Additionally, Gender is another string variable, with two possible values describing whether the customer is a male or female.

Furthermore, from the remaining variables, Age, Tenure, Balance, NumOfProducts, and EstimatedSalary are all numerical variables where higher values represent higher amounts. Additionally, the last 3 variables, HasCrCard, IsActiveMember, Exited, are binary variables taking values 0 and 1. More specifically, if HasCrCard takes value 1 for a specific customer, this suggests that the customer has a Credit Card in the bank. Moreover, if IsActiveMember takes value 1 for a customer, this suggests that this customer is and active user of the bank's services.

Aside from these deterministic variables, the bank has also observed its customers over the period of last 6 months and based on this experiment, to create a dummy variable which takes value 0 if the customer has left the bank during this observation period and takes value 1 stayed as a customer of the bank during this time. This variable is referred as Exited and will be used as dependent variable in our analysis.

## Data Processing

### Data Transformation: Encoding

Firstly, we perform transformation on two string variables in the data to a numerical variable. More specifically, we have transformed gender variable of stringType to a binary numerical variable taking values f0,1g. To do this we use LabelEncoder from Scikit learn library. Furthermore, we use OneHotEncoder from the same library, to transform Geography variable, with multiple string values with country names, to a categorical variable.

### Data Normalization

Artificial Neural Networks are sensitive to the scale of the data. Therefore, we will use StandardScaler from the Scikit learn library to scale the data. Given that the dependent variable is a dummy variable that can only take values 0 and 1, we only need to scale the features data, the set of all independent variables.

### Data Splitting

To make sure we rest train the model using only a part of the data and then use the trained model to predict the values for the customers whose data have not been used during the training. Therefore, we split the original data into train and test where 80% of all customers fall in the train set whereas the 20% of the customers fall in the test set.

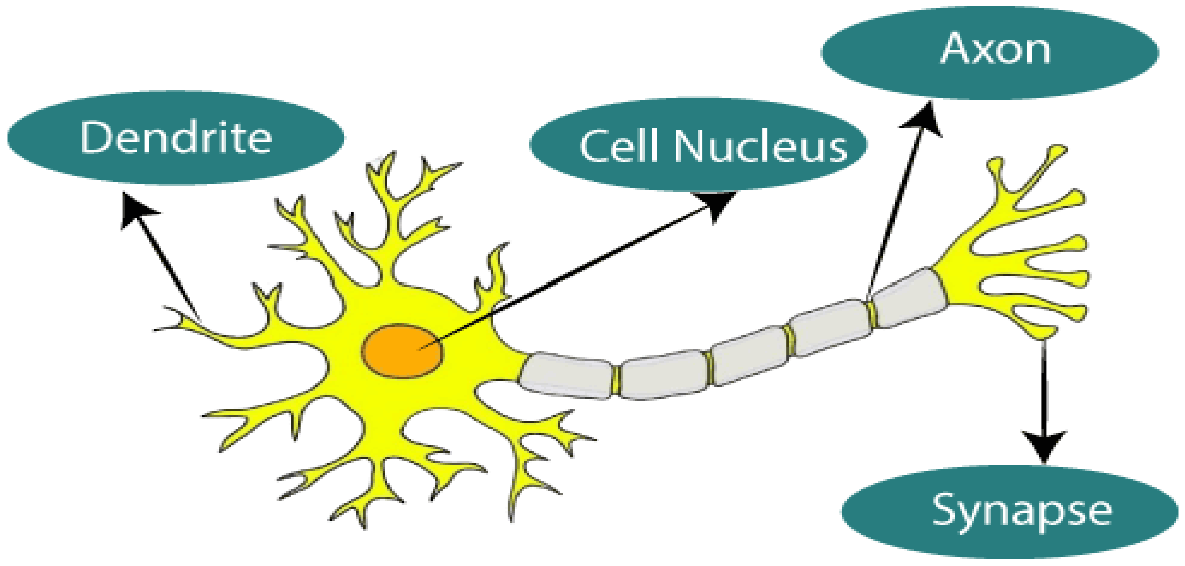
## Methodology

The goal of this analysis is to segment bank's customers into certain groups and identify the customers with the highest chance to leave the bank. The Exited binary variable will serve as a dependent variable in the process of identifying the leaving customers and predicting the churn probabilities of these customers. This will help to understand the reasons why customers leave and use this information on the existing customers to avoid them leaving the bank.

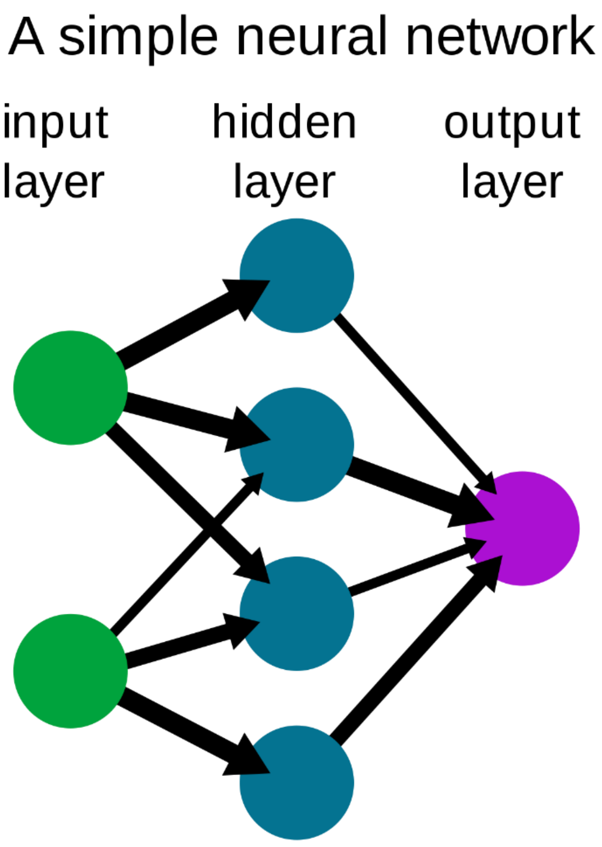
## ANN USAGE

During the last decade, Deep Learning has become one of the most popular subfields of the Machine Learning (3). It is a series of algorithms inspired by the structure and function of the brain called Artificial Neural Networks (ANN). Deep Learning allows quantitative models composed of multiple processing layers to study the data representation with multiple levels of abstraction. It has drastically refined the state-of-the-art in speech recognition, object detection, visual object recognition, and many other disciplines such as drug discovery and genomics.

The entire idea behind Deep Learning is to mimic how human brains work given that the human brain is one of the most powerful tools for learning. When comparing to the human brain, Neural Networks refer to systems of neurons, either organic or artificial in nature. Neuron is the basic building block of the Neural Networks. visualizes the architecture of a neuron. A neuron has a body, located in the left lower part of the figure, with so branches emerging from it that are referred as Dendrites (the receiver of the signal) and it has an Axon (the transmitters of the signals), which is that long tail that connects the body of the neuron to the other neurons, which are often referred as synapses in the Deep Learning terminology. Then the signals are passed to other receptors through these Synapses. Then the cell body sums all those input signals to generate output. The same idea is used in the ANN. Multiple input signals, referred as Input Layer Neurons, are transmitted to Hidden Layer Neurons, which on its turn are used to predict the output, Output Layer. In the sense, input signals are like the human senses, such as human sight, hearing, smelling, tasting, and touching, only in the case of ANN, those input signals can be various type of features characterizing an observation.



This process is visualized by the which shows an example of ANN with 3 inputs signals forming the Input Layer Neurons set, 3 Hidden Layers and 1 Output Layer. Neural networks adapt themselves to the changing input so that the network generates the best possible result without the need to redesign the output criteria. The functionality of neural networks is often compared to the one of the multiple linear regressions, where one uses multiple input features, also called independent variables, to predict the output variable, the dependent variable. In case of Neural Network, we also use input features, referred as Input Layer Neurons to get information and learn about the outcome variable, referred as Output Layer. The main difference between such regression and Neural Network is that, in the case the former the process runs in one iteration by minimizing the sum of the squared residuals (similar to cost function), whereas in case of Neural Network there is an intermediate step portrayed by the Hidden Layer Neurons which are used to get signals from the input layers and learn about the observations over and over again until the goal is achieved, the cost is minimized and no improvement is possible.
So, one can say that ANNs are much more sophisticated than multiple linear regression.

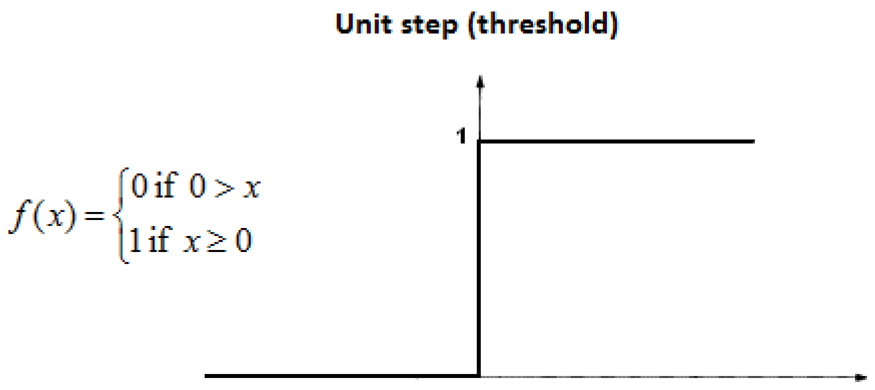


## Activation Function

As it was mentioned before, the cell body sums all the input signals to generate output. This summation process is mimiced in ANN as well where we use a function to join signals from different input neurons into one value. Each synapse gets assigned a weight, an importance value. These weights form the corner stone of how Neural Networks learn. These weights determine whether the signals get passed along or not, or to what extent each signal gets passed along. If we define the input value of signal i by xi and its importance weight by wi then the sum of these signals, can be defined by"
This function is called Activation Function and there are 4 common Activation Function used in Deep Learning: Threshold, Sigmoid, Rectifier, and Hyperbolic Tangent Function. Following figure visualizes those 4 functions.

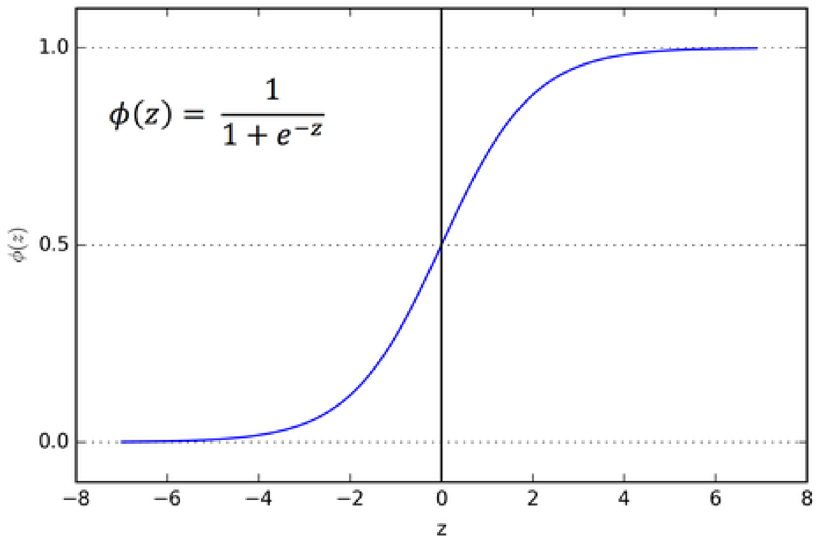
## **1.Threshold Function:**

In case of the Threshold function, the possible values of the function are either 0 or 1. If the summed value is smaller than o then the activation function will give as an output 0, otherwise it will give as an output 1. It's very simple and straight forward activation function with binary output.



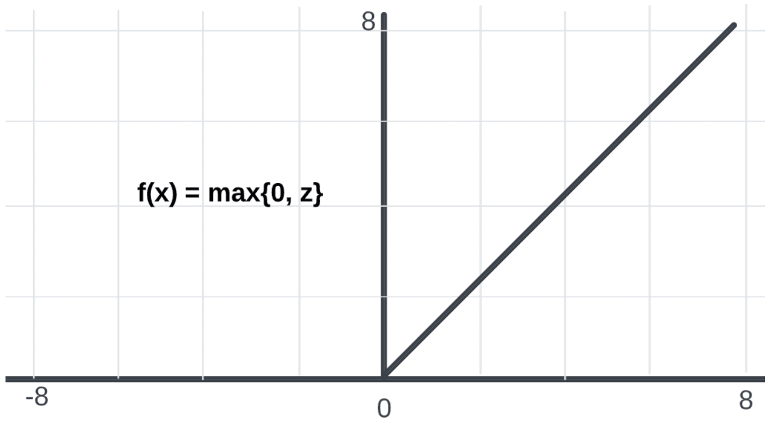
## **2.Sigmoid Function:**

Sigmoid function is based on similar function as the Logistic Regression does. The function is much smoother compared to the threshold function and is very useful when the output value is expected to be a probability.



## 3.Rectifier Function:

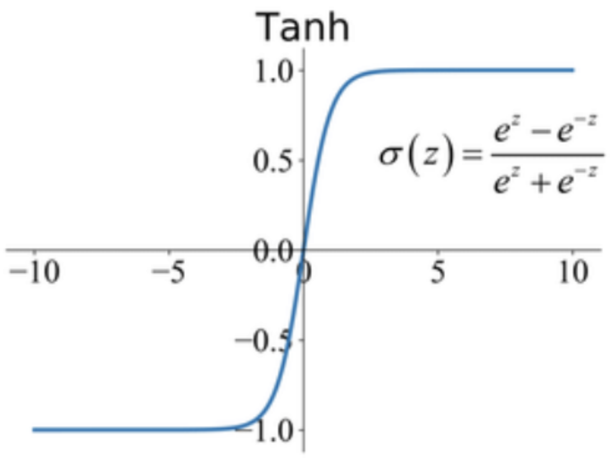
This function is one of the most popular activation functions in ANN which as an output the maximum value of the summed amount and 0 (1).



## 4.Hyperbolic Tangent Function:

This activation function is defined as the ratio between the hyperbolic sine and the cosine functions (the ratio of the half-difference and half-sum of two exponential functions in the points).

The combination of the Rectifier and Sigmoid activation functions is quite popular, and this exact combination will be used in this case study as well, given that our goal is to estimate the probability that the customer will leave the bank.

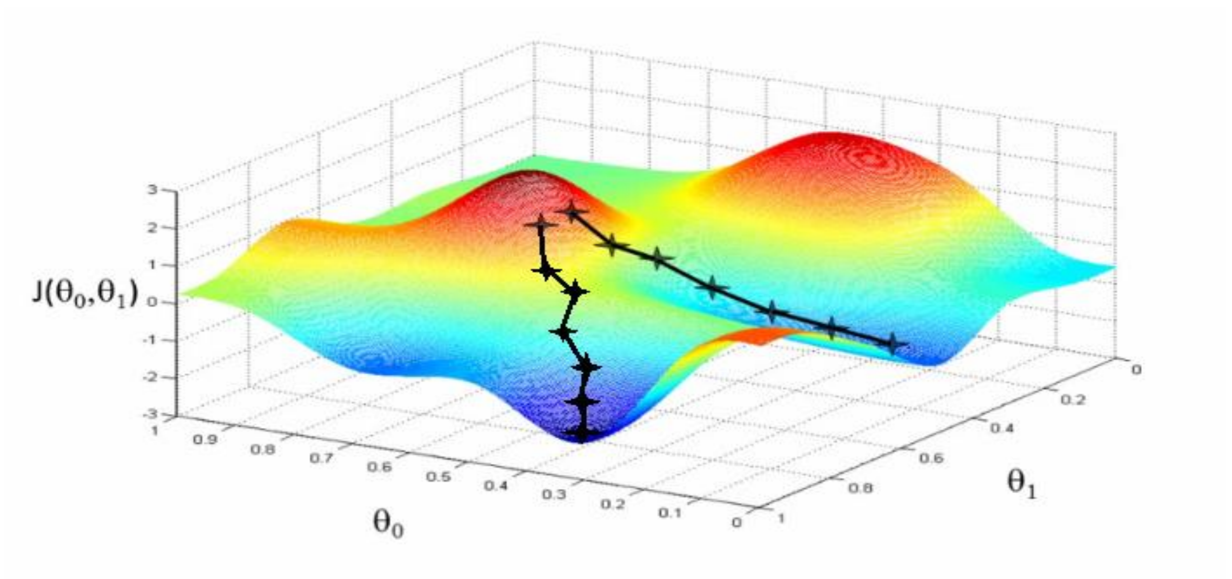


# Cost Function Optimizers

Our goal during the ANN training process is to minimize the amount of error we are making, the difference between the predicted value and the real value of the output variable (y). So, to learn about these different signals is to predict the output value of y, (yhat) then compute the cost associated with this iteration or epoch, determine which set of weights at this stage will minimize the cost function, and use this to update the weights. This process is repeated as much that the cost function is minimized as much as possible. To determine which weights minimizes the cost function, different statistical algorithms can be used such as Stochastic Gradient Decent (SGD) or Adam Optimizer.

## SGD:

SGD method, also known as Incremental Gradient Descent, is an iterative approach for solving optimization problems with a differential objective function (4). The SGD method is often referred as the stochastic approximation of the gradient descent which aims to the extreme or zero points of the stochastic model containing parameters that cannot be directly estimated. Demonstrates an example of an optimization problem where GD and SGD are used. From the two black patters in the graph, the straight pattern in the lower part corresponds to a standard Gradient Descent method where the parameters updates are done using all training points, whereas the black non-linear pattern in the upper side with a lot of actuations, corresponds to the SGD method where for the parameter updates only one training sample is used. With SGD reaching the global maximum is likely to happen faster, because one starts making updates and improving the direction of the pattern using a single training sample. Due to so many updates (actuations in the pattern in the direction of reaching global optimum point) SGD is likely to converge faster compared to the GD. Both SGD and GD suffer from the problem of reaching a local optimum instead of the global optimum. However, in case of SGD the likelihood of reaching a local optimum instead of global optimum is higher compared to the GD, because of its constant changes in moving direction. We observe that it is likely that the actuating black pattern, corresponding to the SGD, will end up in L instead of G while changing his moving direction so often. Whereas it is less likely that the straight pattern, corresponding to the GD, will end up in L, while both patterns have the same starting point.



## Adam Optimizer

A popular technique for enhancing SGD optimization procedure is the Adaptive Moment Estimation (Adam) introduced by (2). Adam is the extended version of the SGD with momentum method. The main difference of it compared to the SGD with momentum, which uses a single learning rate for all parameter updates, Adam algorithm defines different learning rates for different parameters. The algorithm calculates the individual adaptive learning rates for each parameter based on the estimates of the first two moments of the gradients. So, each parameter has a unique learning rate, which is being updated using the exponential decaying average of the first moments (the mean) and second moments (the variance) of the gradients. For all mentioned benefits of Adam optimizer compared to the GD and SGD, we will use Adam Optimizer in this analysis to minimize the loss function and update the weights.

## Data set

Link:

https://drive.google.com/file/d/1q-BZXpI_mSY8OoNzTpvbd7nfDx4khFYV/view?usp=share_link

## Source Code

```
#importing required libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score


#Data Pre-processing #
# Checking the tensorflow version
print(tf._version_)

# Loading the data
bank_data = pd.read_csv("Artificial_Neural_Network_Case_Study_data.csv")

# Taking  all rows and all columns in the data except the last column as X (feature
matrix)
#the row numbers and customer id's are not necessary for the modelling so we get
rid of and start with credit score
X = bank_data.iloc[:,3:-1].values
print("Independent variables are:", X)
#taking all rows but only the last column as Y(dependent variable)
y = bank_data.iloc[:, -1].values
print("Dependent variable is:", y)


# Transforming the gender variable, labels are chosen randomly
le = LabelEncoder()
X[:,2] = le.fit_transform(X[:,2])
print(X)

# Transforming the geography column variable, labels are chosen randomly, the ct
asks for argument [1] the index of the target vb
```

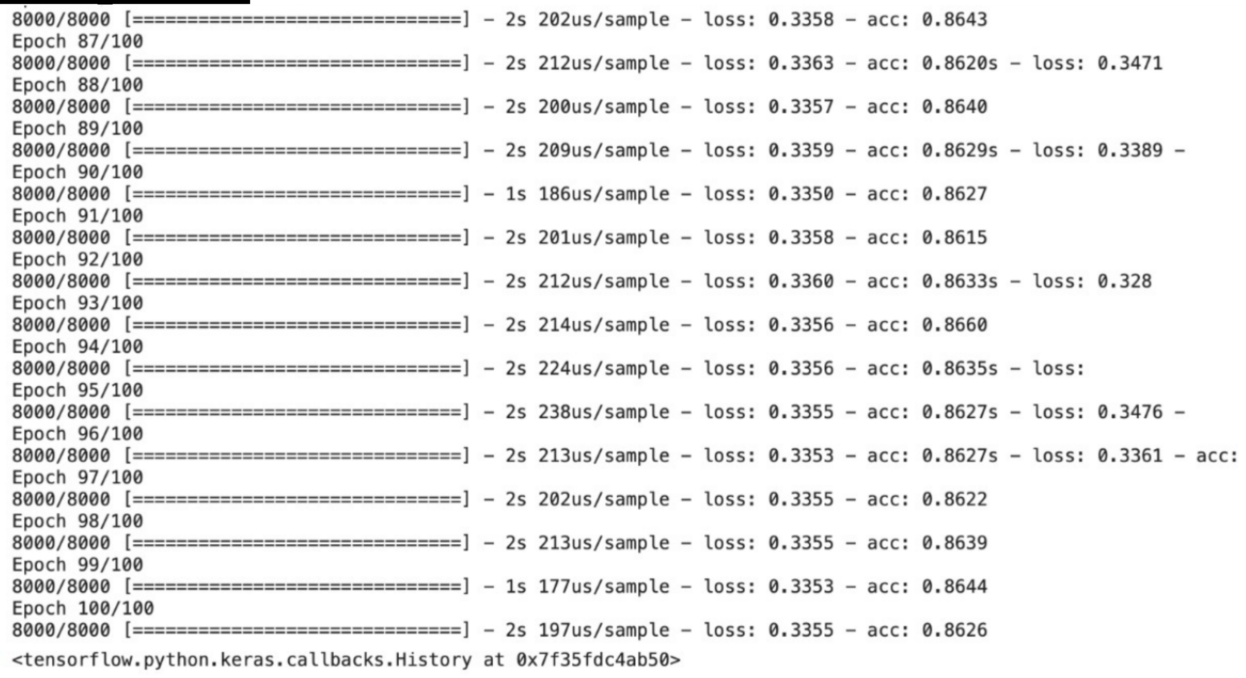
```
ct = ColumnTransformer(transformers = [('encoder', OneHotEncoder(),[1])],
remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
print(X)

# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
#printing the dimensions of each of those snapshots to see amount of rows and
columns i each of them
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)

# Data Scaling/normalization of the features that will go to the NN
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# Building the model #

# Initializing the ANN by calling the Sequential class fromm keras of Tensorflow
ann = tf.keras.models.Sequential()


# Adding "fully connected" INPUT layer to the Sequential ANN by calling Dense
class
# Number of Units = 6 and Activation Function = Rectifier
ann.add(tf.keras.layers.Dense(units = 6, activation = 'relu'))


# Adding "fully connected" SECOND layer to the Sequential AMM by calling
Dense class
# Number of Units = 6 and Activation Function = Rectifier
ann.add(tf.keras.layers.Dense(units = 6, activation = 'relu'))


# Adding "fully connected" OUTPUT layer to the Sequential ANN by calling Dense
class
# Number of Units = 1 and Activation Function = Sigmoid
```

```
ann.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

#Training the model #
# Compiling the ANN
# Type of Optimizer = Adam Optimizer, Loss Function =  crossentropy for binary
dependent variable, and Optimization is done w.r.t. accuracy
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])

# Training the ANN model on training set  (fit method always the same)
# batch_size = 32, the default value, number of epochs  = 100
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)

# Evaluating the Model #
# the goal is to use this ANN model to predict the probability of the customer
leaving the bank
# Predicting the churn probability for single observations

#Geography: French
#Credit Score:600
#Gender: Male
#Age: 40 years old
#Tenure: 3 years
#Balance: $60000
#Number of Products: 2
#with Credit Card
#Active member
#Estimated Salary: $50000

print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])))
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) >
0.5)
# this customer has 4% chance to leave the bank


#show the vector of predictions and real values
#probabilities
y_pred_prob = ann.predict(X_test)

#probabilities to binary
```

```
y_pred = (y_pred_prob > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)), 1))

#Confusion Matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix", confusion_matrix)
print("Accuracy Score", accuracy_score(y_test, y_pred))
```

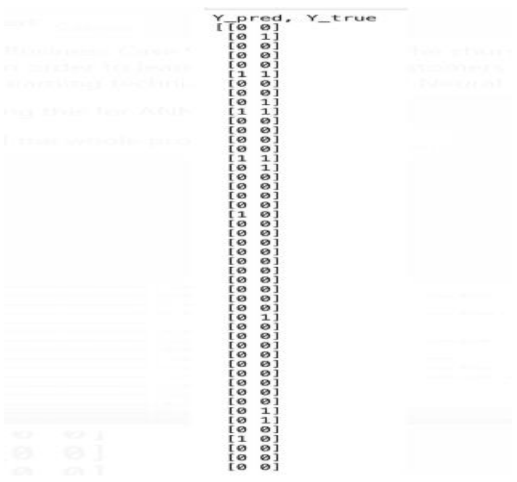
## Implementation:

## Output-1:



```
8000/8000 [==============================] - 2s 202us/sample - loss: 0.3358 - acc: 0.8643
Epoch 87/100
8000/8000 [==============================] - 2s 212us/sample - loss: 0.3363 - acc: 0.8620s - loss: 0.3471
Epoch 88/100
8000/8000 [==============================] - 2s 200us/sample - loss: 0.3357 - acc: 0.8640
Epoch 89/100
8000/8000 [==============================] - 2s 209us/sample - loss: 0.3359 - acc: 0.8629s - loss: 0.3389 -
Epoch 90/100
8000/8000 [==============================] - 1s 186us/sample - loss: 0.3350 - acc: 0.8627
Epoch 91/100
8000/8000 [==============================] - 2s 201us/sample - loss: 0.3358 - acc: 0.8615
Epoch 92/100
8000/8000 [==============================] - 2s 212us/sample - loss: 0.3360 - acc: 0.8633s - loss: 0.328
Epoch 93/100
8000/8000 [==============================] - 2s 214us/sample - loss: 0.3356 - acc: 0.8660
Epoch 94/100
8000/8000 [==============================] - 2s 224us/sample - loss: 0.3356 - acc: 0.8635s - loss:
Epoch 95/100
8000/8000 [==============================] - 2s 238us/sample - loss: 0.3355 - acc: 0.8627s - loss: 0.3476 -
Epoch 96/100
8000/8000 [==============================] - 2s 213us/sample - loss: 0.3353 - acc: 0.8627s - loss: 0.3361 - acc:
Epoch 97/100
8000/8000 [==============================] - 2s 202us/sample - loss: 0.3355 - acc: 0.8622
Epoch 98/100
8000/8000 [==============================] - 2s 213us/sample - loss: 0.3355 - acc: 0.8639
Epoch 99/100
8000/8000 [==============================] - 1s 177us/sample - loss: 0.3353 - acc: 0.8644
Epoch 100/100
8000/8000 [==============================] - 2s 197us/sample - loss: 0.3355 - acc: 0.8626
<tensorflow.python.keras.callbacks.History at 0x7f35fdc4ab50>
```

## Output -2:



```
Y_pred, Y_true
[[0  0]
 [0  1]
 [0  0]
 [0  0]
 [0  0]
 [1  1]
 [0  0]
 [0  0]
 [0  1]
 [1  1]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [1  1]
 [0  1]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [1  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  1]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  0]
 [0  1]
 [0  1]
 [0  0]
 [1  0]
 [0  0]
 [0  0]
 [0  0]
```

## <u>Analysis and Result</u>

For the entire analysis, from data preparation till evaluation of the results, we will use Python. Firstly, we initialize the ANN by calling the Sequential class from Keres library. Keras comes in combination with Tensorow library. Both will be used in this analysis. Secondly, then add fully connected Input Layer to the Sequential ANN by calling Dense class from Keras. Furthermore, we add fully connected Output Layer to the Sequential ANN by calling Dense class from Keras once again. For the first transition we use the Rectifier activation function. Furthermore, for the second layer, to get the output from the ANN, we use the Sigmoid activation function, because our goal is to predict the probability that the customer will leave the bank and as it was mentioned in the previous section Sigmoid type of functions give an output with value between 0 and 1, like in case of probabilities.
Additionally, we use the Adam Optimizer to determine the set of weights for each epoch and because the output variable is a dummy variable, we use the Binary Cross Entropy as a cost function. Given that the focus of this study is of more descriptive nature, we don't prune the model parameters such as the batch size or number of epochs, therefore, we use the default value 32 for the batch size and 100 for the number of epochs.

## Training and Testing  Model

We will train the model using the training dataset of the features, characteristics of the customers in the training set, (Xtrain), the and the training output variable, dummy variable identifying whether the customers in the training set has left the bank during the last 6 months, (Ytrain). Once we have trained the ANN model, we use this fitted model and the test data to predict the output value for the test users, for each customer in the test set the probability of leaving the bank (^ Ytest). Given that the original output data field is in the form of the dummy variable, we need to transform the output with the probability’s values to a binary variable.

Depending on how sever can be the consequences when prediction mistakes are made, one can decide the threshold k accordingly. In this analysis we will use threshold equal to 0.5 (k = 0.5) which means that if the output probability from the model is larger or equal to 0.5, that is the customer has larger than 0.5 chance to leave the bank, than the dummy variable ^ Ytest takes value 1, and 0 otherwise.

## Evaluation:

From the previous step we obtained a dummy variable for the test users based on the predicted probabilities (^ Ytest). Furthermore, we have in our data the true output values of these test users (Ytest). By comparing these two sets of values, predicted output values and the true output values, per user with a goal to find out the accuracy of the model, in this case the percentage of customers for which the model has made a good prediction. The model has showed accuracy of 0.86 which means that the model managed too correctly predict whether the customer will leave the bank or not in 86 out of 100 cases (2K test users in total).

### Confusion Matrix:

To find out more about the performance of the model and more specifically about the false positives and false negatives, we use the Scikit learn library to obtain the following confusion matrix corresponding to the model.

(1510 85)
(197 208)

What we observe is that for 1510 customers who left the bank, the model was able to correctly predict that. Moreover, for 208 customers the model correctly predicted that they have not left the bank. For 85 customers the model predicted that they have left the bank while they didn't (False Positives) and for 197 cases the model predicted that the customers have not left the bank while they did (False negatives).

## **Reference:**

Glorot, X., Bordes, A., Bengio, Y. (2011). Deep Sparse Rectifier Neural NetworksInternational Conference on Artificial Intelligence and Statistics. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 15(15), 315 - 323
Rogerson, R. J. (2015). Adam: A Method For Stochastic Optimization. 3rd International Conference on Learning Representations (ICLR2015), 36(1), 1–13
LeCun, Y., Bengio, Y., Hinton, G., (2015). Deep learning.Nature, 521, 436 – 444
Wiegerinck, W. and Komoda, A. and Heskes, T. (1999). Stochastic dynamics of learn-ing with momentum in neural networks. Journal of Physics A: Mathematical andGeneral, 27(13), 4425 – 4425