Predicting Bank Customer Behavior Using Artificial Neural Network

1st Tahmid
Dept. of CSE
East West University
Dhaka, Bangladesh
2017-1-65-009@std.ewubd.edu

1st Avia Anwar
Dept. of CSE
East West University
Dhaka, Bangladesh
2017-1-60-154@std.ewubd.edu

1st Nayma Alam
Dept. of CSE
East West University
Dhaka, Bangladesh
2018-1-60-180@std.ewubd.edu

***Abstract*—Neural networks are set of algorithms inspired by the functioning of human brian. An artificial neural network is configured for a specific application, such as pattern recognition or data classification,Image recognition, voice recognition through a learning process. Here we have implemented ANN using tensorflow library to predict if a customer of bank will stay or not in the bank using all the features provided in the dataset. The data used here is called Churn Modelling which contains 9 feature to measure if a customer will stay in the bank or not.**

***Index Terms*—Neural Network, Tensorflow**

I. Introduction

With the advancement in Machine Learning, Artificial Intelligence has taken a high road. Deep Learning is considered to be the most advanced technology built to solve complex problems that use massive data sets and artificail neural network is the most popular and important algorithm of this branch. These neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts.

Tensorflow: TensorFlow is a library based on Python that provides different types of functionality for implementing Deep Learning Models. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

NumPy: NumPy brings the computational power of languages like C and Fortran to Python, a language much easier to learn and use. With this power comes simplicity: a solution in NumPy is often clear and elegant.

Pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.It has functions for analyzing, cleaning, exploring, and manipulating data. Pandas allows us to analyze big data and make conclusions based on statistical theories.Pandas can clean messy data sets, and make them readable and relevant.

Scipy: SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

Scikit-learn: Scikit-learn is python library built on on NumPy, SciPy, and matplotlib. It is a Simple and efficient tools for predictive data analysis

II. Feed Forward Neural Network

A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle.As such, it is different from its descendant: recurrent neural networks. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

A. How a feed forward neural network works

How does a Feed Forward Neural Network work? A Feed Forward Neural Network is commonly seen in its simplest form as a single layer perceptron. In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1. The single layer perceptron is an important model of feed forward neural networks and is often used in classification tasks. Furthermore, single layer perceptrons can incorporate aspects of machine learning. Using a property known as the delta rule, the neural network can compare the outputs of its nodes with the intended values, thus allowing the network to adjust its weights through training in order to produce more accurate output values. This process of training and learning produces a form of a gradient descent. In multi-layered perceptrons, the process of updating weights is nearly analogous, however the process is defined more specifically as back-propagation. In such cases, each hidden layer within the network is adjusted according to the output values produced by the final layer.

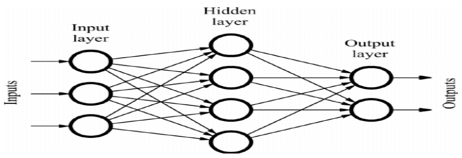


Fig. 1. Feed forward neural network

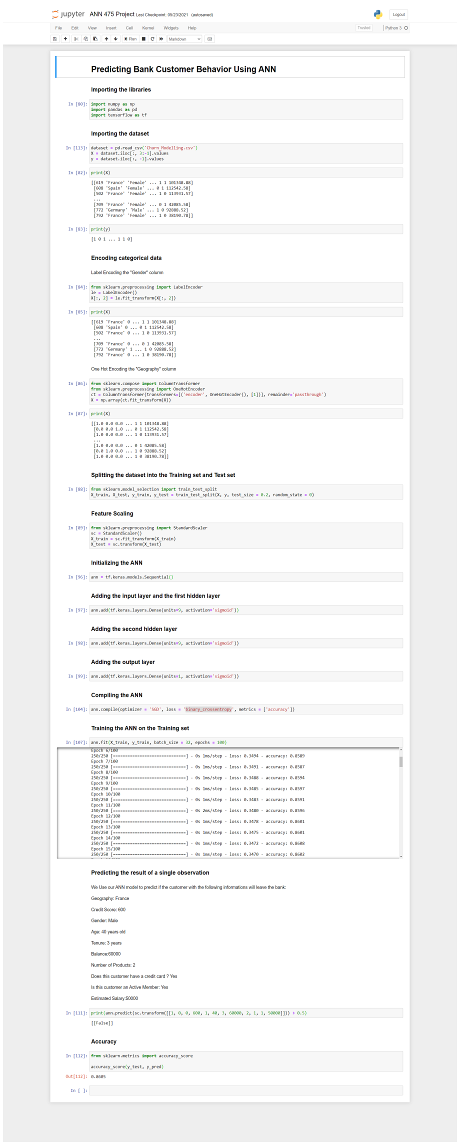
III. Implementation

Firstly, pre-process the data We import from the csv file (data set) into a data frame using pandas library and remove unnecessary features such as row number and column id. These features are irrelevant to predict if a customer will leave or not. We assign all the necessary features in the variable X and their corresponding results in variable Y. Now we realize that there are categorical data inside X. One is gender and other one is geography. We simply label the gender feature where male is 1 and female is 0. Now we encode the geography feature using one hot encoding, as ordinal relationship exists. Now we split the data set into training and testing sets in the ration of 80-20. As there are different kinds of features with different scale like credit score and number of products, we scale the features using standard scaling.

Now our data is ready for a neural network's training phase. We initialize a neural network using tensorflow. Then we add 9 neuron input layer along with 9 neuron hidden layer. Now we add a second hidden layer and finally a single neuron output layer. We used sigmoid as activation function for all the neurons. Now we compile the neural network, we used Stochastic Gradient descent as optimizer and binary crossentropy as loss function. Now the training begins with 32 batch size and 100 epoch.

IV. Code

```
import numpy as np
import pandas as pd
import tensorflow as tf

dataset = pd.read_csv('Churn_Modelling.csv')
print(dataset)
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values

print(X)

print(y)

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```



Predicting Bank Customer Behavior Using ANN

Importing the libraries

```
In [80]: import numpy as np
         import pandas as pd
         import tensorflow as tf
```

Importing the dataset

```
In [113]: dataset = pd.read_csv('Churn_Modelling.csv')
          X = dataset.iloc[:, 3:-1].values
          y = dataset.iloc[:, -1].values

In [82]: print(X)

         [[619 'France' 'Female' ... 1 1 101348.88]
          [608 'Spain' 'Female' ... 0 1 112542.58]
          [502 'France' 'Female' ... 1 0 113931.57]
          ...
          [709 'France' 'Female' ... 0 1 42085.58]
          [772 'Germany' 'Male' ... 1 0 92888.52]
          [792 'France' 'Female' ... 1 0 38190.78]]

In [83]: print(y)

         [1 0 1 ... 1 1 0]
```

Encoding categorical data

Label Encoding the "Gender" column

```
In [84]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         X[:, 2] = le.fit_transform(X[:, 2])

In [85]: print(X)

         [[619 'France' 0 ... 1 1 101348.88]
          [608 'Spain' 0 ... 0 1 112542.58]
          [502 'France' 0 ... 1 0 113931.57]
          ...
          [709 'France' 0 ... 0 1 42085.58]
          [772 'Germany' 1 ... 1 0 92888.52]
          [792 'France' 0 ... 1 0 38190.78]]
```

One Hot Encoding the "Geography" column

```
In [86]: from sklearn.compose import ColumnTransformer
         from sklearn.preprocessing import OneHotEncoder
         ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
         X = np.array(ct.fit_transform(X))

In [87]: print(X)

         [[1.0 0.0 0.0 ... 1 1 101348.88]
          [0.0 0.0 1.0 ... 0 1 112542.58]
          [1.0 0.0 0.0 ... 1 0 113931.57]
          ...
          [1.0 0.0 0.0 ... 0 1 42085.58]
          [0.0 1.0 0.0 ... 1 0 92888.52]
          [1.0 0.0 0.0 ... 1 0 38190.78]]
```

Splitting the dataset into the Training set and Test set

```
In [88]: 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)
```

Feature Scaling

```
In [89]: from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
```

Initializing the ANN

```
In [96]: ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
In [97]: ann.add(tf.keras.layers.Dense(units=9, activation='sigmoid'))
```

Adding the second hidden layer

```
In [98]: ann.add(tf.keras.layers.Dense(units=9, activation='sigmoid'))
```

Adding the output layer

```
In [99]: ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Compiling the ANN

```
In [104]: ann.compile(optimizer = 'SGD', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

```
In [107]: ann.fit(X_train, y_train, batch_size = 32, epochs = 100)

          Epoch 6/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3494 - accuracy: 0.8589
          Epoch 7/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3491 - accuracy: 0.8587
          Epoch 8/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3488 - accuracy: 0.8594
          Epoch 9/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3485 - accuracy: 0.8597
          Epoch 10/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3483 - accuracy: 0.8591
          Epoch 11/100
          250/250 [==============================] - 0s 2ms/step - loss: 0.3480 - accuracy: 0.8596
          Epoch 12/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3478 - accuracy: 0.8601
          Epoch 13/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3475 - accuracy: 0.8601
          Epoch 14/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3472 - accuracy: 0.8608
          Epoch 15/100
          250/250 [==============================] - 0s 1ms/step - loss: 0.3470 - accuracy: 0.8602
```

Predicting the result of a single observation

We Use our ANN model to predict if the customer with the following informations will leave the bank:

Geography: France

Credit Score: 600

Gender: Male

Age: 40 years old

Tenure: 3 years

Balance:60000

Number of Products: 2

Does this customer have a credit card ? Yes

Is this customer an Active Member: Yes

Estimated Salary:50000

```
In [111]: print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)

          [[False]]
```

Accuracy

```
In [112]: from sklearn.metrics import accuracy_score

          accuracy_score(y_test, y_pred)

Out[112]: 0.8605

In [ ]:
```

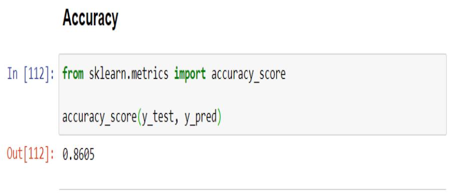
Fig. 2. ANN implemented using Jupyter Notebook

```
print(X)

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder',
                        OneHotEncoder(),[1])],
                        remainder='passthrough')
X = np.array(ct.fit_transform(X))

print(X)

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)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
ann = tf.keras.models.Sequential()

ann.add(tf.keras.layers.Dense(units=6,
                              activation='sigmoid'))

ann.add(tf.keras.layers.Dense(units=6,
                              activation='sigmoid'))

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

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

ann.fit(X_train, y_train,
        batch_size = 32, epochs = 100)

print(ann.predict(sc.transform
([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]]))
> 0.5)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

V. Result

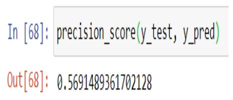
We have successfully implemented the neural network using tensorflow and trained it with our data set. We tried out an example with different parameters which gave us a result that indicated the customer will not leave the bank. We set the threshold value to 0.5.If predicted value is more than 0.5 we will consider it as 1 and it'll indicate that the customer will stay. The accuracy we got from calculating with test data set we got and 86% accurary and 57% precision. Here among all the data 1514 are true positive, 81 false positive, 298 are false negative and 107 are true negative.



Accuracy

```
In [112]: from sklearn.metrics import accuracy_score

          accuracy_score(y_test, y_pred)

Out[112]: 0.8605
```

Fig. 3. Accuracy

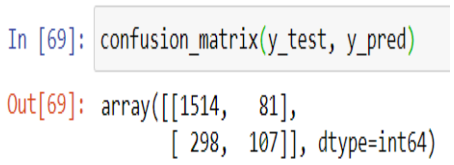


```
In [68]: precision_score(y_test, y_pred)

Out[68]: 0.5691489361702128
```

Fig. 4. Precision

VI. Conclusion

We have trained and tested our dataset and tried to predict if a bank customer will leave or not. We have used TensorFlow for this purpose and we have successfully implemented the neural network that if able to give correct prediction.



```
In [69]: confusion_matrix(y_test, y_pred)

Out[69]: array([[1514,   81],
                [ 298,  107]], dtype=int64)
```

Fig. 5. Confusion Marix

VII. Acknowledgement

The project has been done with the optimum contribution of three group members,Tahmid Absar, Avia Anwar and Nayma Alam East West University, Dhaka. This enormous thanks to our course instructor, Prof. Dr. Md.Golam Rabiul Alam, Associate Professor (Adjunct), Department of Computer Science Engineering, East West University, Dhaka. With his expert supervision and kind help, we have completed this project.

References

[1] keras-team / keras (Ed.). (n.d.). tf.keras.optimizers.SGD : TensorFlow Core v2.6.0. TensorFlow. Retrieved September 12, 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD.

[2] keras-team / keras (Ed.). (n.d.). tf.keras.losses.BinaryCrossentropy : TensorFlow Core v2.6.0. TensorFlow. Retrieved September 12, 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy.

[3] Russell, Stuart J. (Stuart Jonathan). Artificial Intelligence : a Modern Approach. Upper Saddle River, N.J. :Prentice Hall, 2010.

[4] https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

[5] https://en.wikipedia.org/wiki/Feedforward$_neural_network$