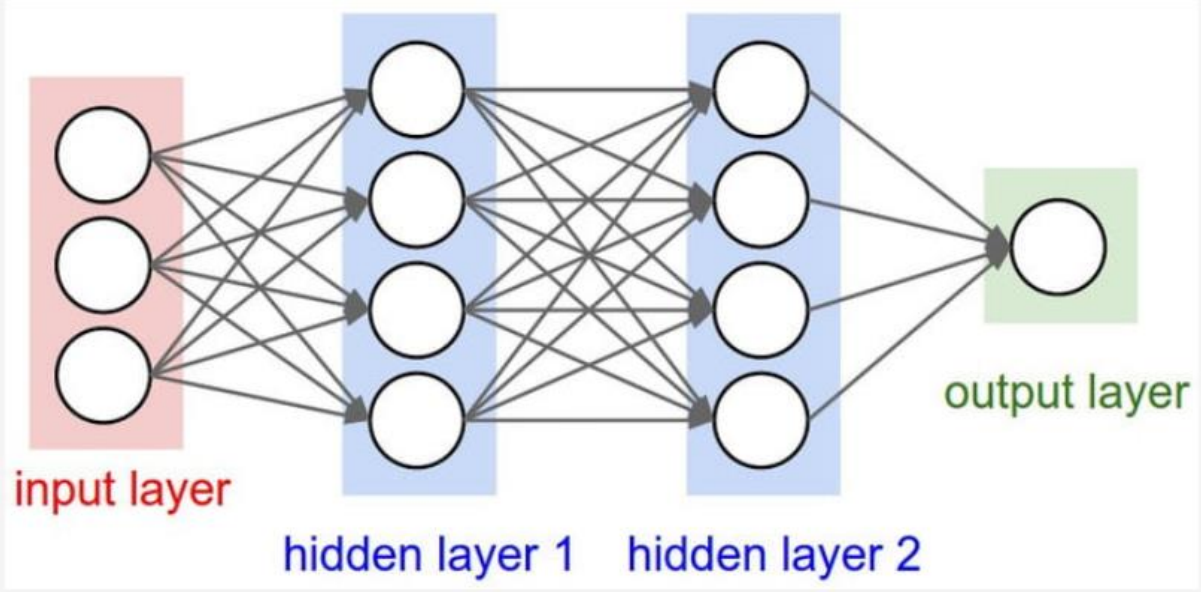
Bank Customer Churn Classification using Keras

Problem Statement

Every year bank loses customers due to various reasons. For a bank, the cost of acquiring a new customer is more than retaining an existing one. The bank would like to do an analysis on the data available, which will help in predicting the future customers who are likely to leave the bank.

Artificial Neural Networks: Keras

Inspired by the human brain, the neural networks are implemented in order to replicate the way in which humans learn. Artificial neural networks consists of input layers, hidden layers and output layers. The number of layers could be altered depending on the depth of processing required for the input in order to get a particular output.



Keras is a high level neural network API, which runs on top of TensorFlow, Theano or CNTK. Keras enables fast implementation in order to fetch the solution for research purpose with least possible delay. Keras provides priority to user experience by offering consistent and simple APIs which in turn minimises the user actions required. The standalone, fully-configurable modules that can be combined with few restrictions in sequence form a model. New modules and layers are easy to add in Keras, making Keras really useful for research purpose. Convolutional networks, recurrent networks and the combination of both are supported by Keras.

Data

The data in file *'Customer_churn.csv'* represents the data of the customers that are part of the bank and the customers that have exited the bank. The data consists of 14 different features namely 'Row Number', 'Customer ID', 'Surname', 'Credit Score', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'Number of Products', 'Has Credit Card', 'Is Active Member', 'Estimated Salary' and 'Exited'.

Row Number: Denotes the number of rows and also the number of entries in the data.

Customer ID: Every customer has a unique Customer ID. It is the primary key for each customer in this dataset. It can be used to get all the details for the particular customer.

Surname: Consists of the surname of the customer. This detail could be used for verification purpose.

Credit Score: Denotes the rating obtained by the analysis of the customer's credit files and factors including the Salary, Balance and previous paid credit card and mortgage dues if any.

Geography: Gives the information about the location of the customer and the bank.

Gender: Provides information on the gender of the customer.

Age: Provides information about the age of the customer.

Tenure: Gives us information about the total number of years the customer has been part of the bank.

Balance: Denotes the account balance for the customer.

Number of Products: Provides information on the number of services such as credit card, mortgage, FDs, mutual funds etc which are provided by the bank, that the customer has subscribed to.

Has Credit Card: Provides the status on whether the customer has a credit card or not.

Is Active Member: Provides information on how frequent the transactions take place for a particular customer.

Estimated Salary: Gives us the information of the annual income of each customer.

Exited: Denotes whether the customer has exited the bank or not.

Data Pre-processing

There are 14 different features listed in the dataset. In order to predict which customers will exit the bank in the future, choosing key features is very important. In order to achieve this, we must look at all the features and derive whether it matters in classifying a customer or not. The first three features namely Row Number, Customer ID and Surname are not important in classification of the customer churn process and hence we can eliminate them from the data that is used for further analysis. The features 'Credit Score', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'Number of Products', 'Has Credit Card', 'Is Active Member', 'Estimated Salary' and 'Exited' will be considered for analysis, training and predicting whether the customer is going to stay.

Data Analysis

The dataset is imported and stored. The features 'Credit Score', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'Number of Products', 'Has Credit Card', 'Is Active Member', 'Estimated Salary' will be part of one list, which will be used for training and testing the model. The feature 'Exited' is stored in separate list, which will help in determining how accurate the model can predict whether the customer is going to leave the bank.

The features 'Geography' and 'Gender' consisting of categorical data are encoded into integers using label encoding method, considering all the rest of the features have numerical value.

Geography: France->0

Germany->1

Gender: Male->0

Female->1

The label encoding is not sufficient for these features, as it assumes that the category with the higher categorical value is better. So we use one hot encoder to binarize the categories after label encoding and include them to train the model.

For Example, let us consider the Geography feature. The output obtained after one hot encoding is:

France	Germany	Other details
0	1	Other details
1	0	Other details

Split the dataset into training and testing data in order to train the model and then test it with the remaining data labelled as testing data. 80% of the data is considered as training data and 20% of the data from the dataset will be utilised for testing.

The training data is subjected to Feature Scaling process in order to scale all the features to be even, such that one feature does not dominate another. Normalising the range of all features is essential as one feature such as Bank Balance might range from to 0 to 6 digits value, whereas Gender values from 0 to 1. Normalising these features will enable them to contribute equally to the final result.

Building the Artificial Neural Network

Using Keras and TensorFlow backend, we will build an artificial neural network to build a model which will help in predicting the customer churn. The neural network will consist of an input layers, few hidden layers if required and an output layer. In order to decide the number of nodes for each layer, we can calculate the average of the total features utilised for input and output layers. The number of nodes would be equal to (11+1)/2=6 nodes.

In order to compute deep learning for the given model, we will implement two hidden layers in between the input and output layers.

Input Layer

The input layer consists of attributes such as activation, input_dim, units and kernel_initializer.

```
classifier.add(Dense(activation = 'relu', input_dim = 11, units=6, kernel_initializer='uniform'))
```

activation: We assign ReLu activation function for the input layer.

input_dim: Since we have 11 features, we assign the value as 11 in input layer.

units: We assign the value to 6, which are the number of nodes in the input layer.

kernel_initializer: Since we want the distribution of data to randomly initialize weights to the nodes, we set it as uniform.

Hidden Layers

Let us add two hidden layers in between the input and output layer for this model. The attribute input_dim is not mentioned in these layers, whose value can be inferred from the input layer. The hidden layers are defined in order to get more depth into the details of the given dataset. For the

given dataset, two hidden layers are sufficient. The accuracy increases with the increase in the hidden layers, as more features of the data could be studied in detail. The accuracy increased when the model was tested with two hidden layers when compared to one hidden layer. When the third hidden layer is introduced, the accuracy didn't change to a greater value and hence considering the model features, two hidden layers are sufficient to implement deep learning for this particular dataset.

```
classifier.add(Dense(activation = 'relu', units=6, kernel_initializer='uniform'))

classifier.add(Dense(activation = 'relu', units=6, kernel_initializer='uniform'))
```

Output Layer

The output layer consists of attributes such as activation, input_dim, units and kernel_initializer.

```
classifier.add(Dense(activation = 'sigmoid', units=1, kernel_initializer='uniform'))
```

activation: We assign Sigmoid instead of ReLu, since it generates probabilities for the data determining whether the customer leaves the bank or not.

input_dim: Since we have 11 features, we assign the value as 11 in input layer.

units: We assign the value to 6, which are the number of nodes in the input layer.

kernel_initializer: Since we want the distribution of data to randomly initialize weights to the nodes, we set it as uniform.

Compiling and Fitting the network

In order to tune individual weights on each neuron, we apply the Stochastic Gradient descent om the whole neural network. Stochastic gradient descent, is an incremental gradient descent, an iterative method for selecting the best elements using a differentiable object function.

The compiling process consists of attributes such as optimizer, loss and metrics.

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

optimizer: We assign it to adam, a very efficient variation of Stochastic gradient descent used to find the optimal set of weights.

loss: We assign binary_crossentropy, since our output variable is binary. It is a logarithmic loss function used within adam algorithm.

metrics: We assign it to accuracy, since the accuracy will be evaluated by the model.

In order to fit the neural network, the process consists of attributes such as X_train, Y_train, batch_size and epochs.

```
values = classifier.fit(X_train, y_train, batch_size=10, epochs=50)
```

X_train: The training data which needs to be fit into the model.

Y_train: The output data which is obtained from the model after fitting.

batch_size: Denotes the number of time back propagation for error values can be implemented, so the individual node weights can be corrected.

epochs: The number of iterations we want to process the data in order to tune the weights.

```
Epoch 1/50
8000/8000 [==============================] - 1s 119us/step - loss: 0.4815 - acc: 0.7956
Epoch 2/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4309 - acc: 0.7960
Epoch 3/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4249 - acc: 0.8015
Epoch 4/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4210 - acc: 0.8239
Epoch 5/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4186 - acc: 0.8295
Epoch 6/50
8000/8000 [==============================] - 1s 79us/step - loss: 0.4169 - acc: 0.8301
Epoch 7/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4143 - acc: 0.8329
Epoch 8/50
8000/8000 [==============================] - 1s 96us/step - loss: 0.4134 - acc: 0.8327
Epoch 9/50
8000/8000 [==============================] - 1s 90us/step - loss: 0.4125 - acc: 0.8339
Epoch 10/50
8000/8000 [==============================] - 1s 86us/step - loss: 0.4107 - acc: 0.8356
```

```
Epoch 11/50
8000/8000 [==============================] - 1s 85us/step - loss: 0.4091 - acc: 0.8335
Epoch 12/50
8000/8000 [==============================] - 1s 101us/step - loss: 0.4087 - acc: 0.8355
Epoch 13/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4069 - acc: 0.8336
Epoch 14/50
8000/8000 [==============================] - 1s 100us/step - loss: 0.4066 - acc: 0.8359
Epoch 15/50
8000/8000 [==============================] - 1s 96us/step - loss: 0.4061 - acc: 0.8344
Epoch 16/50
8000/8000 [==============================] - 1s 97us/step - loss: 0.4053 - acc: 0.8356
Epoch 17/50
8000/8000 [==============================] - 1s 95us/step - loss: 0.4044 - acc: 0.8355
Epoch 18/50
8000/8000 [==============================] - 1s 93us/step - loss: 0.4048 - acc: 0.8350
Epoch 19/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4040 - acc: 0.8349
Epoch 20/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4040 - acc: 0.8335
```

```
Epoch 21/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4029 - acc: 0.8345
Epoch 22/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4033 - acc: 0.8342
Epoch 23/50
8000/8000 [==============================] - 1s 92us/step - loss: 0.4028 - acc: 0.8350
Epoch 24/50
8000/8000 [==============================] - 1s 79us/step - loss: 0.4023 - acc: 0.8357
Epoch 25/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4023 - acc: 0.8349
Epoch 26/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4024 - acc: 0.8354:
Epoch 27/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4015 - acc: 0.8334
Epoch 28/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4023 - acc: 0.8344
Epoch 29/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4021 - acc: 0.8352
Epoch 30/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4014 - acc: 0.8345
```

```
Epoch 31/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4017 - acc: 0.8337
Epoch 32/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4014 - acc: 0.8344
Epoch 33/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4016 - acc: 0.8352
Epoch 34/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4007 - acc: 0.8356
Epoch 35/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4015 - acc: 0.8356
Epoch 36/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4010 - acc: 0.8337
Epoch 37/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4011 - acc: 0.8352
Epoch 38/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4010 - acc: 0.8361
Epoch 39/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4010 - acc: 0.8374
Epoch 40/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4010 - acc: 0.8352
```

```
Epoch 41/50
8000/8000 [==============================] - 1s 72us/step - loss: 0.4004 - acc: 0.8340
Epoch 42/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4009 - acc: 0.8364
Epoch 43/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4011 - acc: 0.8366
Epoch 44/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4004 - acc: 0.8367
Epoch 45/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4012 - acc: 0.8342
Epoch 46/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4007 - acc: 0.8356
Epoch 47/50
8000/8000 [==============================] - 1s 70us/step - loss: 0.4006 - acc: 0.8350
Epoch 48/50
8000/8000 [==============================] - 1s 71us/step - loss: 0.4007 - acc: 0.8347
Epoch 49/50
8000/8000 [==============================] - 1s 75us/step - loss: 0.4013 - acc: 0.8346
Epoch 50/50
8000/8000 [==============================] - 1s 72us/step - loss: 0.4010 - acc: 0.8356
```

Testing

The testing data is used to predict the accuracy of the trained classifier. We will understand the performance of the classifier by creating a confusion matrix. In order to use confusion matrix, the probabilities that a customer will leave the bank must be converted into Boolean values true or false. In this scenario, a cut-off value of 0.5 could be inserted to filter the customers who are likely to exit. The number of correct classifications are found from the confusion matrix and the testing data accuracy is calculated.

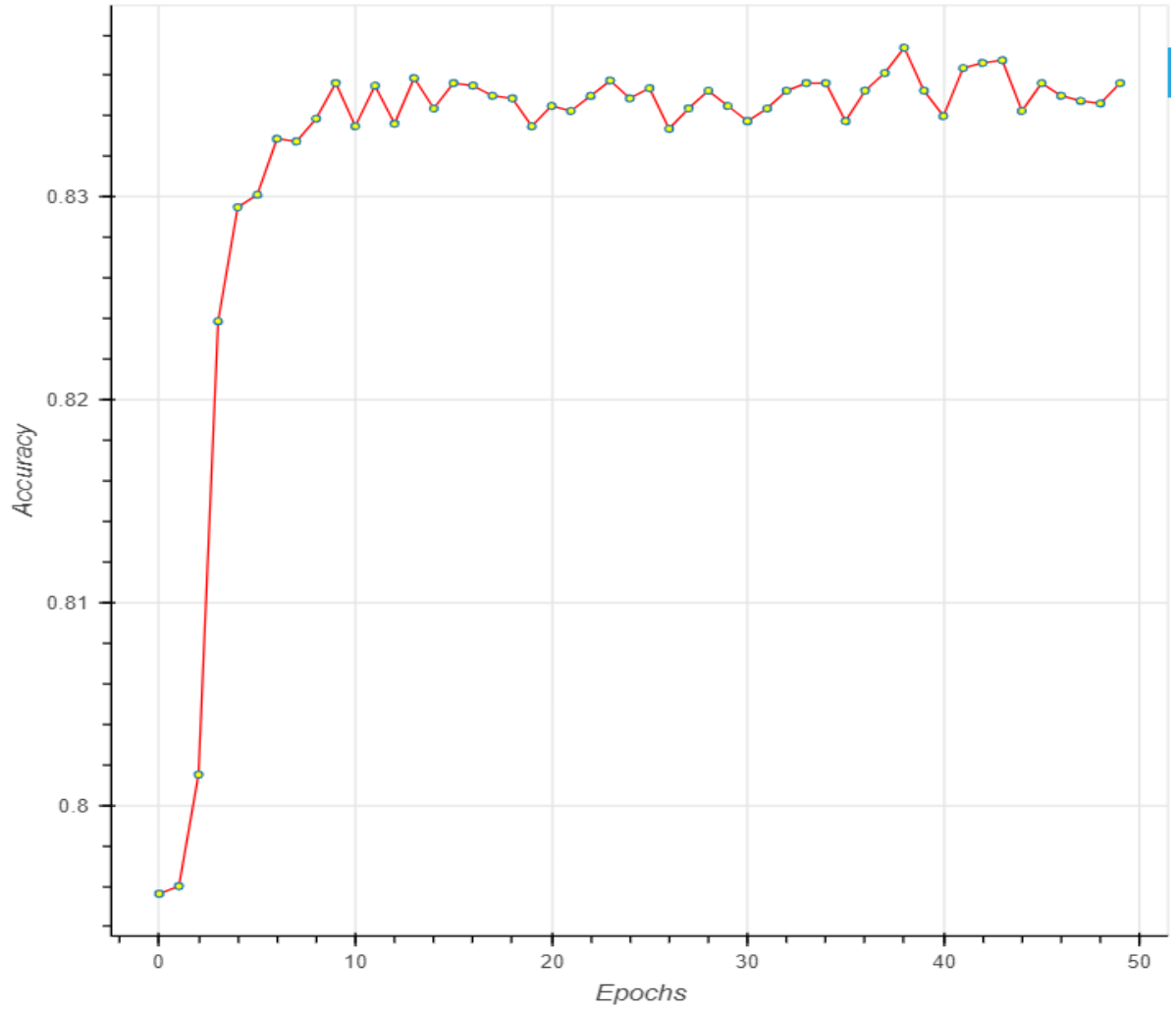
```
Testing accuracy:  84 %
```

Bokeh

Bokeh is an interactive visualisation library, used to create different type of visualisations for given data. All the accuracy values in each epoch of the training model are stored and a graph of Accuracy vs Epoch can be plotted using Bokeh.

X-axis: Epoch values ranging from 0 to 50

Y-axis: All the accuracy values recorded for each epoch



The analysis of this data also signifies that customers who are less active, have less subscribed to number of products that the bank has to offer are more likely to leave.