Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
```

Import Tensorflow Library

Installing Tensorflow

- pip install Tensorflow --> in anaconda prompt
- !pip install tensorflow --> in jupyter notebook

In [2]:
```
import tensorflow as tf
```

In [3]:
```
#tf.__version__
```

Out[3]:
```
'2.4.1'
```

Keras Library

Installing Keras

- pip install Keras --> in anaconda prompt
- !pip install --upgrade keras --> in jupyter notebook

In [ ]:
```
# from tensorflow import keras
```

In [8]:
```
import keras
```

Part 1 - Data Preprocessing

In [9]:
```
### Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
dataset
```

Out[9]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActi
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

10000 rows × 14 columns

In [10]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [11]:
```
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

In [12]:
```
X.shape, y.shape
```

Out[12]:
```
((10000, 10), (10000,))
```

In [13]:
```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
```

In [14]:
```
#One Hot Encoding the "Geography" column
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 [15]:
```
### Splitting the dataset into the Training set and Test set
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)
```

In [16]:
```
X_train.shape,X_test.shape,y_train.shape,y_train.shape
```

Out[16]:
```
((8000, 12), (2000, 12), (8000,), (8000,))
```

In [17]:
```
### Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Part 2 - Building the ANN

In [18]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

Initializing the ANN

In [19]:
```
ann = Sequential()

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

Adding the input layer and the first hidden layer

In [20]:
```
ann.add(Dense(input_dim = 12,units = 6, kernel_initializer = 'uniform', activation = 'relu'))
#ann.add(Dropout(rate = 0.1))

#ann.keras.layers.add(Dense(units=6, activation='relu'))
```

Adding the second hidden layer

In [21]:
```
ann.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
#ann.add(Dropout(rate = 0.1))
```

Adding the output layer

In [22]:
```
ann.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

Part 3 - Training the ANN

Compiling the ANN

In [23]:
```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

In [24]:
```
ann.fit(X_train, y_train,batch_size = 32, epochs = 100)
```

...

Part 4 - Predictions & Evaluating the model

Making the predictions

In [25]:
```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
#print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Evaluating the model

In [26]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

print("Test Accuracy:",accuracy_score(y_test, y_pred))
confusion_matrix(y_test, y_pred)
```

```
Test Accuracy: 0.862
```

Out[26]:
```
array([[1531,   64],
       [ 212,  193]], dtype=int64)
```

Cross validate the model

In [27]:
```
def build_cross_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 12))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = "adam", loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
```

In [28]:
```
from keras.wrappers.scikit_learn import KerasClassifier
classifier = KerasClassifier(build_fn = build_cross_classifier, batch_size = 10, epochs = 100)
```

In [29]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 5)
accuracies.mean()
```

...

Predicting the result of a single observation

Use our ANN model to redict if the customer with the followin 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

So, should we say goodbye to that customer ?

Solution

In [30]:
```
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)
```

```
[[False]]
```

Therefore, our ANN model predicts that this customer stays in the bank!

Important note 1: Notice that the values of the features were all input in a double pair of square brackets. That's because the "predict" method always expects a 2D array as the format of its inputs. And putting our values into a double pair of square brackets makes the input exactly a 2D array.

Important note 2: Notice also that the "France" country was not input as a string in the last column but as "1, 0, 0" in the first three columns. That's because of course the predict method expects the one-hot-encoded values of the state, and as we see in the first row of the matrix of features X, "France" was encoded as "1, 0, 0". And be careful to include these values in the first three columns, because the dummy variables are always created in the first columns.

Part 5 - Improving and Tuning the ANN

This can be done by using 3 options

- 1.Hyperparameter Tuning
- 2.Regularization (L1 & L2) to reduce overfitting if needed (for Regression problems)
- 3.Dropout

Hyperparameter tuning can be done for identifying no.of hidden layers & no.of neurons in each hidden layer

- layers = [[20],[10],[30],[40]] --> 1 hidden layer with different options of no.of neurons in that hidden layer
- layers = [[20],[40,20],[45,30,15]] --> Multiple hidden layers with different options of no.of neurons

Hyperparameter tuning can be done for identifying best activation function for hidden layers

- activations=['relu','sigmoid']

Hyperparameter tuning can be done for identifying best optimizers

- optimizer =['adam','rmsprop']

Hyperparameter tuning can be done for identifying best batchsize for building/training model

- batchsize =[10,16,32,64,128,256]

In [31]:
```
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 12))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
```

In [32]:
```
classifier = KerasClassifier(build_fn = build_classifier)
```

In [33]:
```
parameters = {'batch_size': [10,32], 'epochs': [50,100], 'optimizer': ['adam','rmsprop']}
```

In [34]:
```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(estimator = classifier,param_grid = parameters,scoring = 'accuracy', cv = 5)
```

In [ ]:
```
grid_result = grid.fit(X_train, y_train)
```

...

In [36]:
```
#best parameters
grid_result.best_params_
```

Out[36]:
```
{'batch_size': 32, 'epochs': 50, 'optimizer': 'adam'}
```

In [37]:
```
#best accuracy
grid_result.best_score_
```

Out[37]:
```
0.844125
```

In [38]:
```
y_pred = grid_result.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
C:\Users\rrr90\Anaconda3\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: UserWarning: `mod
el.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(mode
l.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-l
ayer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification
(e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

In [39]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score

print("Test Accuracy:",accuracy_score(y_test, y_pred))
confusion_matrix(y_test, y_pred)
```

```
Test Accuracy: 0.8405
```

Out[39]:
```
array([[1530,   65],
       [ 254,  151]], dtype=int64)
```