Artificial Neural Network

Importing the libraries

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

In [2]:
```python
tf.__version__
```

Out[2]:
```
'2.7.0'
```

Part 1 - Data Preprocessing

Importing the dataset

In [4]:
```python
dataset = pd.read_csv('Project_1_ANN_Banking.csv')
X = dataset.iloc[:, 3:-1].values #Printing starting from index 3(CreditScore) to EstimatedSalary excluding -1(Exited)--->Train Attributes
y = dataset.iloc[:, -1].values #Printing only last column Exited--->Target Attribute
```

In [5]:
```python
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 [6]:
```python
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]]
```

Encoding categorical data

In [9]:
```
Label Encoding the "Gender" column
```

In [10]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2]) #Encoding index 2 column only(Gender)
```

In [11]:
```python
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 [13]:
```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough') #[1]--->Column index 1
X = np.array(ct.fit_transform(X))
```

In [14]:
```python
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 [15]:
```python
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) #0.2--->20% data from train data set will be tested
```

Feature Scaling

In [16]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
#Scaling everything
```

Part 2 - Building the ANN

Initializing the ANN

In [17]:
```python
ann = tf.keras.models.Sequential() #Initializing ANN as a sequence of layers
```

Adding the input layer and the first hidden layer

In [18]:
```python
ann.add(tf.keras.layers.Dense(units=6, activation='relu')) #Automatically add input neorons,assuming 6 neurons in first hidden layer
```

Adding the second hidden layer

In [19]:
```python
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the output layer

In [21]:
```python
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid')) #For probability of binary outcome & prediction sigmoid preferable,Output layer 1
```

Part 3 - Training the ANN

Compiling the ANN

In [22]:
```python
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
#adam--->updating weights to reduce loss through iteration
#loss--->compute difference between prediction & real result
#binary_crossentrophy--->Always be for binary classification & for non-binary it will be catagorical_crossentrophy
```

Training the ANN on the Training set

In [23]:
```python
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
250/250 [==============================] - 4s 7ms/step - loss: 0.6375 - accuracy: 0.6315
Epoch 2/100
250/250 [==============================] - 2s 6ms/step - loss: 0.4646 - accuracy: 0.7959
Epoch 3/100
250/250 [==============================] - 2s 7ms/step - loss: 0.4301 - accuracy: 0.8083
Epoch 4/100
250/250 [==============================] - 2s 6ms/step - loss: 0.4148 - accuracy: 0.8161
Epoch 5/100
250/250 [==============================] - 2s 6ms/step - loss: 0.4001 - accuracy: 0.8248
Epoch 6/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3846 - accuracy: 0.8351
Epoch 7/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3722 - accuracy: 0.8426
Epoch 8/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3649 - accuracy: 0.8454
Epoch 9/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3615 - accuracy: 0.8468
Epoch 10/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3599 - accuracy: 0.8482
Epoch 11/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3582 - accuracy: 0.8489
Epoch 12/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3576 - accuracy: 0.8474
Epoch 13/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3565 - accuracy: 0.8495
Epoch 14/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3555 - accuracy: 0.8503
Epoch 15/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3556 - accuracy: 0.8511
Epoch 16/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3545 - accuracy: 0.8503
Epoch 17/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3534 - accuracy: 0.8514
Epoch 18/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3530 - accuracy: 0.8520
Epoch 19/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3516 - accuracy: 0.8526
Epoch 20/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3515 - accuracy: 0.8531
Epoch 21/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3506 - accuracy: 0.8543
Epoch 22/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3495 - accuracy: 0.8550
Epoch 23/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3489 - accuracy: 0.8556
Epoch 24/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3487 - accuracy: 0.8571
Epoch 25/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3478 - accuracy: 0.8566
Epoch 26/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3471 - accuracy: 0.8575
Epoch 27/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3470 - accuracy: 0.8580
Epoch 28/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3460 - accuracy: 0.8597
Epoch 29/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3457 - accuracy: 0.8586
Epoch 30/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3444 - accuracy: 0.8601
Epoch 31/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3442 - accuracy: 0.8596
Epoch 32/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3439 - accuracy: 0.8615
Epoch 33/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3430 - accuracy: 0.8583
Epoch 34/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3430 - accuracy: 0.8612
Epoch 35/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3429 - accuracy: 0.8583
Epoch 36/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3422 - accuracy: 0.8597
Epoch 37/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3416 - accuracy: 0.8593
Epoch 38/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3415 - accuracy: 0.8606
Epoch 39/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3407 - accuracy: 0.8612
Epoch 40/100
250/250 [==============================] - 2s 9ms/step - loss: 0.3409 - accuracy: 0.8615
Epoch 41/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3407 - accuracy: 0.8596
Epoch 42/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3403 - accuracy: 0.8627
Epoch 43/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3401 - accuracy: 0.8605
Epoch 44/100
250/250 [==============================] - 2s 9ms/step - loss: 0.3401 - accuracy: 0.8622
Epoch 45/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3402 - accuracy: 0.8594
Epoch 46/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3403 - accuracy: 0.8597
Epoch 47/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3390 - accuracy: 0.8609
Epoch 48/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3395 - accuracy: 0.8600
Epoch 49/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3391 - accuracy: 0.8595
Epoch 50/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3393 - accuracy: 0.8599
Epoch 51/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3390 - accuracy: 0.8600
Epoch 52/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3385 - accuracy: 0.8616
Epoch 53/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3390 - accuracy: 0.8615
Epoch 54/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3388 - accuracy: 0.8600
Epoch 55/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3387 - accuracy: 0.8593
Epoch 56/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3379 - accuracy: 0.8616
Epoch 57/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3382 - accuracy: 0.8627
Epoch 58/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3378 - accuracy: 0.8619
Epoch 59/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3380 - accuracy: 0.8618
Epoch 60/100
250/250 [==============================] - 1s 6ms/step - loss: 0.3376 - accuracy: 0.8627
Epoch 61/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3385 - accuracy: 0.8605
Epoch 62/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3376 - accuracy: 0.8625
Epoch 63/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3384 - accuracy: 0.8620
Epoch 64/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3377 - accuracy: 0.8624
Epoch 65/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3377 - accuracy: 0.8636
Epoch 66/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3375 - accuracy: 0.8625
Epoch 67/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3375 - accuracy: 0.8610
Epoch 68/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3374 - accuracy: 0.8635
Epoch 69/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3371 - accuracy: 0.8640
Epoch 70/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3368 - accuracy: 0.8644
Epoch 71/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3375 - accuracy: 0.8639
Epoch 72/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3369 - accuracy: 0.8626
Epoch 73/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3372 - accuracy: 0.8639
Epoch 74/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3368 - accuracy: 0.8643
Epoch 75/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3372 - accuracy: 0.8618
Epoch 76/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3365 - accuracy: 0.8618
Epoch 77/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3367 - accuracy: 0.8631
Epoch 78/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3360 - accuracy: 0.8636
Epoch 79/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3367 - accuracy: 0.8622
Epoch 80/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3366 - accuracy: 0.8630
Epoch 81/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3362 - accuracy: 0.8644
Epoch 82/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3363 - accuracy: 0.8633
Epoch 83/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3358 - accuracy: 0.8631
Epoch 84/100
250/250 [==============================] - 2s 6ms/step - loss: 0.3369 - accuracy: 0.8633
Epoch 85/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3359 - accuracy: 0.8656
Epoch 86/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3361 - accuracy: 0.8624
Epoch 87/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3363 - accuracy: 0.8646
Epoch 88/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3355 - accuracy: 0.8656
Epoch 89/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3361 - accuracy: 0.8625
Epoch 90/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3359 - accuracy: 0.8621
Epoch 91/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3351 - accuracy: 0.8641
Epoch 92/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3355 - accuracy: 0.8641
Epoch 93/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3353 - accuracy: 0.8620
Epoch 94/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3356 - accuracy: 0.8654
Epoch 95/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3354 - accuracy: 0.8640
Epoch 96/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3345 - accuracy: 0.8635
Epoch 97/100
250/250 [==============================] - 2s 8ms/step - loss: 0.3355 - accuracy: 0.8648
Epoch 98/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3347 - accuracy: 0.8640
Epoch 99/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3350 - accuracy: 0.8655
Epoch 100/100
250/250 [==============================] - 2s 7ms/step - loss: 0.3349 - accuracy: 0.8635
```

Out[23]:
```
<keras.callbacks.History at 0x26bac98edf0>
```

Part 4 - Making the predictions and evaluating the model

Predicting the result of a single observation

Homework

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

So, should we say goodbye to that customer?

Solution

In [25]:
```python
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5) #predicting desired train data
#False means value is close to zero--->means customer eill stay according to target dataset
```

```
[[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.

Predicting the Test set results

In [26]:
```python
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))
#Comparing between predicting result(0 column) vs real result(1 column)
```

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

Making the Confusion Matrix

In [27]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
#Confusion matrix--->Finding accuracy
#1516 correct prediction-Customer stays in the bank,205 leaves from the bank & 79 inaccurate prediction-Customer leaves from the bank,200 -Customer
```

```
[[1526   69]
 [ 205  200]]
```

Out[27]:
```
0.863
```

In [ ]: