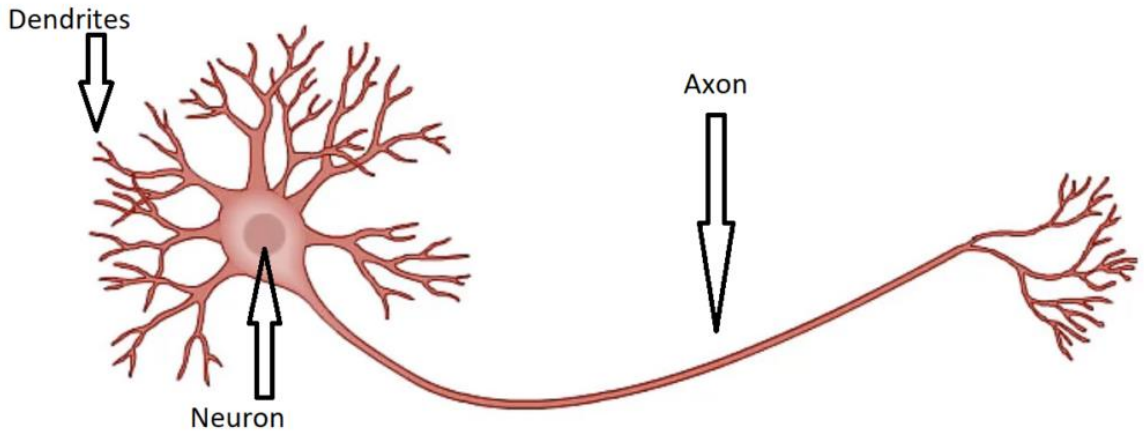
Artificial Neural Network

What is an Artificial Neural Network?

Artificial Neural Network is much similar to the human brain.

The human Brain consist of **neurons**. These neurons are connected with each other. In the human brain, neuron looks something like this…



As you can see in this image, There are **Neuron, Dendrites, and axon.**

Do you think?

When you touch the hot surface, how you suddenly remove your hand?. This is the procedure that happens inside you.

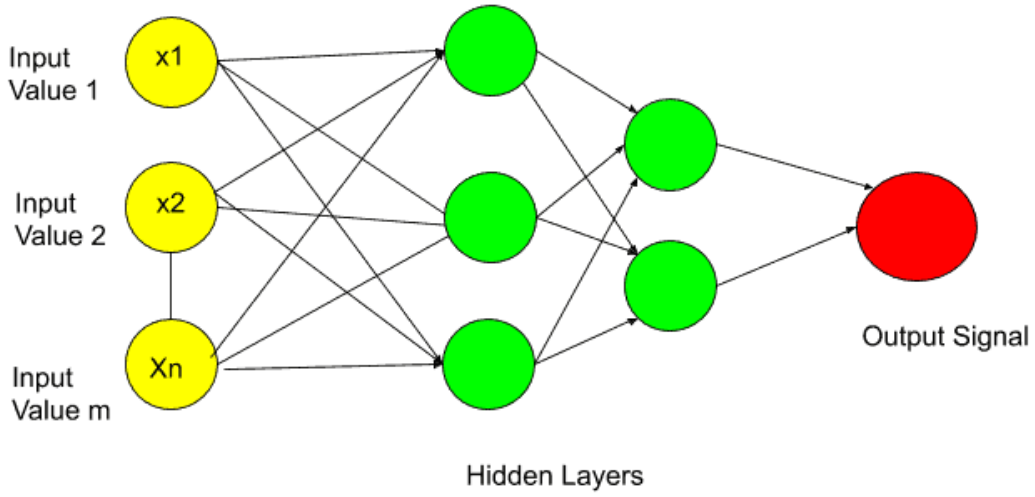
When you touch some hot surface. Then automatically your skin sends a signal to the neuron. And then the neuron takes a decision, **"Remove your hand"**.

So that's all about the **Human Brain.** In the same way, **Artificial Neural Network works.**

Artificial Neural Network has three layers-

1. Input Layer.
2. Hidden Layer.
3. Output Layer.

Let’s see in this image-

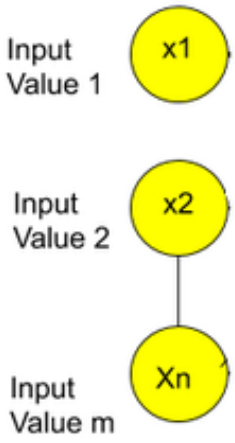


In this image, all the circles you are seeing are neurons. Artificial Neural Network is **fully connected** with these neurons.

Data is passed to the **input layer**. And then the input layer passed this data to the next layer, which is a **hidden layer**. The hidden layer performs certain operations. And pass the result to the **output layer**.

So, this is the basic rough working procedure of an Artificial Neural Network.

In these three layers, various computations are performed.Now Let’s understand each layer in detail. So the first layer is the **Input Layer**.



You may have a question in your mind that What signals are passed through the Input layer?.

So in terms of the human brain, these input signals are your senses. These senses are whatever you can see, hear, smells, or touch. For example, if you touch some hot surface, then suddenly a signal sent to your brain. And that signal is the Input signal in terms of the human brain.

But,

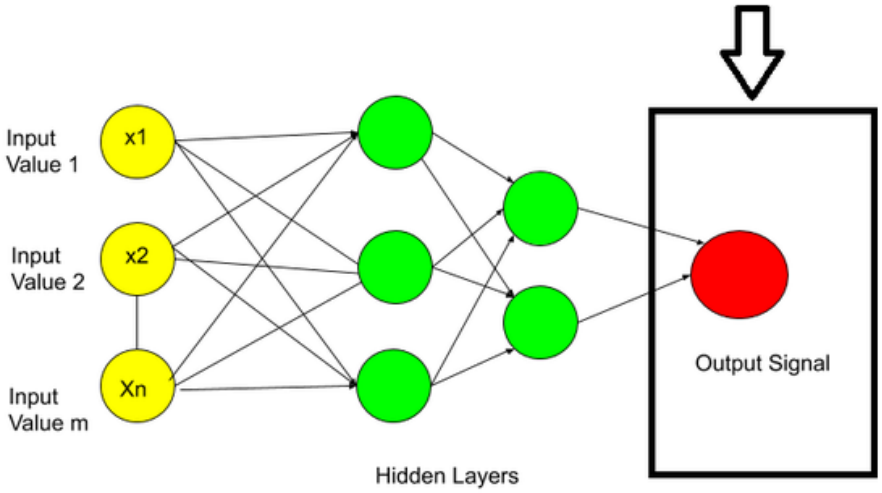
In terms of an artificial neural network, the input layer contains **independent variables**. So the independent **variable 1, independent variable 2, and independent variable n.**

The important thing you need to remember is that these **independent variables are for one observation**. In more simple words, suppose there are different independent variables like a **person's age, salary, and job role.** So take all these independent variables for one person or one row.

Another important point you need to know is that you need to perform some **standardization or normalization** on these independent variables. It depends upon the scenario. The main purpose of doing standardization or normalization is to make all values in the same range. I'll discuss this in the implementation part.

Now let's move on to the next layer and that is-

Output Layer-

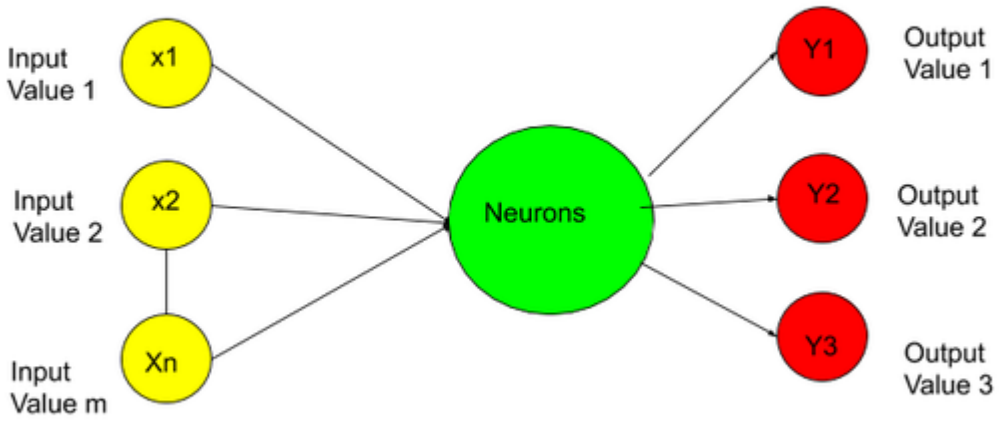


So, the next question is What can be the output value?

The answer is the output value can be-

1. Continous(Like price).
2. Binary(in Yes/no form).
3. Categorical variable.

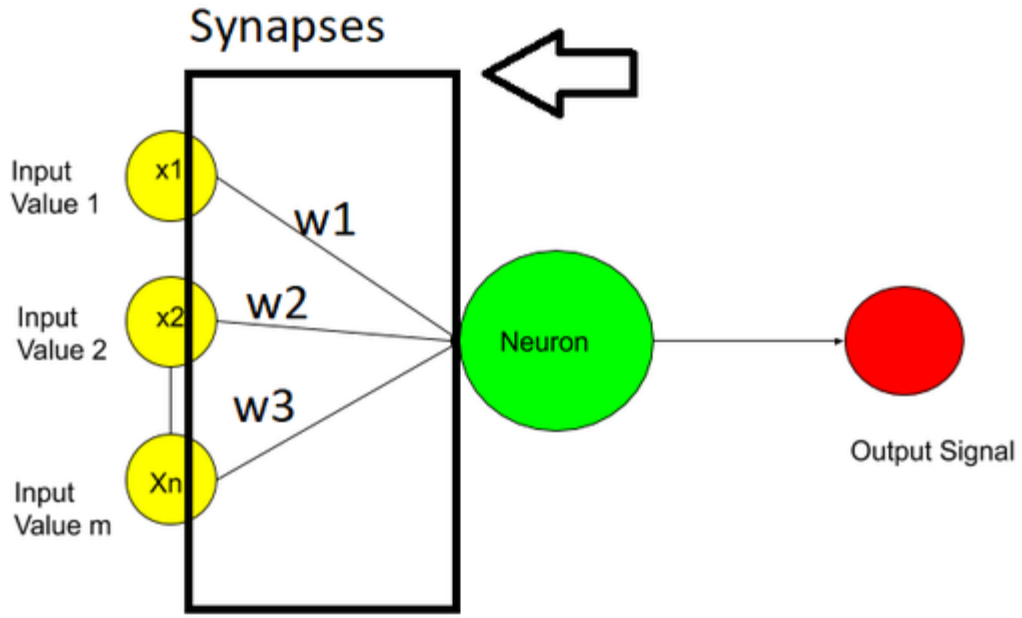
If the output value is categorical then the important thing is, in that case, your output value is not one. It may be more than one output value. As I have shown in the picture.



Next, discuss synapses.

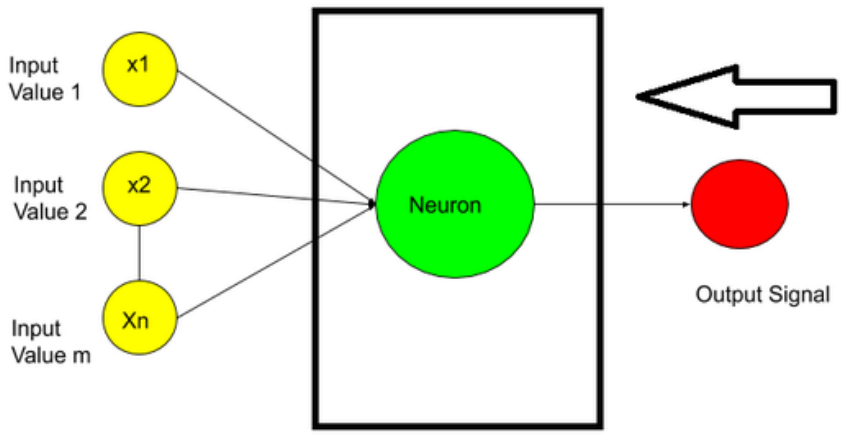
Synapses-

Synapses are nothing but the connecting lines between two layers.



In synapses, weights are assigned to each synapse. These weights are crucial for artificial neural networks work. Weights are how neural networks learn. By adjusting the weights neural network decides what signal is important and what signal is not important.

Hidden Layer or Neuron-



The next question is What Happens inside the neurons?

So Inside the neurons, the two main important steps happen-

1. Weighted Sum.

2. Activation Function.

The first step is the weighted sum, which means all of the weights assigned to the synapses are added with input values. Something like that-

[x1.w1+x2.w2+x3.w3+……..Xn.Wn]

After calculating the weighted sum, the **activation function** is applied to this weighted sum. And then the neuron decides whether to send this signal to the next layer or not.

I hope now you understood the basic work procedure of an Artificial Neural Network. Now let's move to the implementation **of Artificial Neural Network in Python**.

ANN Implementation in Python

For implementation, I am gonna use **Churn Modelling Dataset**. You can download the dataset from **Kaggle**. Artificial Neural Network can be used for **both classification and regression**. And here we are going to use **ANN for classification.**

This dataset has following features-

Index	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101349	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.9	1	0	1	112543	0
2	3	15619304	Onio	502	France	Female	42	8	159661	3	1	0	113932	1
3	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.6	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125511	1	1	1	79084.1	0
5	6	15574012	Chu	645	Spain	Male	44	8	113756	2	1	0	149757	1
6	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
7	8	15656148	Obinna	376	Germany	Female	29	4	115047	4	1	0	119347	1
8	9	15792365	He	501	France	Male	44	4	142051	2	0	1	74940.5	0
9	10	15592389	H?	684	France	Male	27	2	134604	1	1	1	71725.7	0
10	11	15767821	Bearce	528	France	Male	31	6	102017	2	0	0	80181.1	0
11	12	15737173	Andrews	497	Spain	Male	24	3	0	2	1	0	76390	0
12	13	15632264	Kay	476	France	Female	34	10	0	2	1	0	26261	0
13	14	15691483	Chin	549	France	Female	25	5	0	2	0	0	190858	0
14	15	15600882	Scott	635	Spain	Female	35	7	0	2	1	1	65951.6	0
15	16	15643966	Goforth	616	Germany	Male	45	3	143129	2	0	1	64327.3	0
16	17	15737452	Romeo	653	Germany	Male	58	1	132603	1	1	0	5097.67	1

This dataset has **Customer Id, Surname, Credit Score, Geography, Gender, Age, Tenure, Balance, Num of Products they(use from the bank such as credit card or loan, etc), Has Credit card or not (1 means yes 0 means no), Is Active Member (That means the customer is using the bank or not), estimated salary.**

So these all are independent variables of the Churn Modelling dataset. The last feature is the **dependent variable** and that is **customer exited or not from the bank in the future**(1 means the customer will exit the bank and 0 means the customer will stay in the bank.)

The bank uses these independent variables and analyzes the behavior of customers for 6 months whether they leave the bank or stay and made this dataset.

Now the bank has to create a predictive model based on this dataset for new customers. This predictive model has to predict for any new customer that he or she will stay in the bank or leave the bank. So that bank can offer something special for the customers whom the predictive model predicts will leave the bank.

I hope now you understood the problem statement. So the first step in the Implementation of an Artificial Neural Network in Python is **Data Preprocessing**.

1. Data Preprocessing

In data preprocessing the first step is-

1.1 Import the Libraries-

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

NumPy is an open-source Python library used to perform various **mathematical and scientific tasks.** NumPy is **used** for working with arrays. It also has functions for working in the domain of **linear algebra, Fourier transform, and matrices.**

Matplotlib is a plotting library, that is used for creating a **figure, plotting area in a figure, plot some lines in a plotting area, decorates the plot with labels, etc.**

Pandas is a tool used for **data wrangling and analysis.**

So in step 1, we imported all required libraries. Now the next step is-

1.2 Load the Dataset

```
dataset = pd.read_csv('Churn_Modelling_dataset.csv')
```

So, when you load the dataset after running this line of code, you will get your data something like this-

Index	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101349	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.9	1	0	1	112543	0
2	3	15619304	Onio	502	France	Female	42	8	159661	3	1	0	113932	1
3	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.6	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125511	1	1	1	79084.1	0
5	6	15574012	Chu	645	Spain	Male	44	8	113756	2	1	0	149757	1
6	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
7	8	15656148	Obinna	376	Germany	Female	29	4	115047	4	1	0	119347	1
8	9	15792365	He	501	France	Male	44	4	142051	2	0	1	74940.5	0
9	10	15592389	H?	684	France	Male	27	2	134604	1	1	1	71725.7	0
10	11	15767821	Bearce	528	France	Male	31	6	102017	2	0	0	80181.1	0
11	12	15737173	Andrews	497	Spain	Male	24	3	0	2	1	0	76390	0
12	13	15632264	Kay	476	France	Female	34	10	0	2	1	0	26261	0
13	14	15691483	Chin	549	France	Female	25	5	0	2	0	0	190858	0
14	15	15600882	Scott	635	Spain	Female	35	7	0	2	1	1	65951.6	0
15	16	15643966	Goforth	616	Germany	Male	45	3	143129	2	0	1	64327.3	0
16	17	15737452	Romeo	653	Germany	Male	58	1	132603	1	1	0	5097.67	1

As you can see in the dataset, there are 13 independent variables and 1 dependent variable. But the first three independent variables **Row Number, Customer Id, and Surname** are useless for our prediction. So we will eliminate these three independent variables in the next step. And we will also split the independent variables in X and a dependent variable in Y.

1.3 Split Dataset into X and Y

```
X = pd.DataFrame(dataset.iloc[:, 3:13].values)
y = dataset.iloc[:, 13].values
```

Why **dataset.iloc[:, 3:13].values**? because credit_score has an index value as 3. And we want features from credit_score to estimated_salary.

When you will run these lines, you will get two separate tables X and Y. Something like this-

Independent Variables (X)-

Index	0	1	2	3	4	5	6	7	8	9
0	619	France	Female	42	2	0	1	1	1	101349
1	608	Spain	Female	41	1	83807.9	1	0	1	112543
2	502	France	Female	42	8	159661	3	1	0	113932
3	699	France	Female	39	1	0	2	0	0	93826.6
4	850	Spain	Female	43	2	125511	1	1	1	79084.1
5	645	Spain	Male	44	8	113756	2	1	0	149757
6	822	France	Male	50	7	0	2	1	1	10062.8
7	376	Germany	Female	29	4	115047	4	1	0	119347
8	501	France	Male	44	4	142051	2	0	1	74940.5
9	684	France	Male	27	2	134604	1	1	1	71725.7
10	528	France	Male	31	6	102017	2	0	0	80181.1
11	497	Spain	Male	24	3	0	2	1	0	76390
12	476	France	Female	34	10	0	2	1	0	26261
13	549	France	Female	25	5	0	2	0	0	190858
14	635	Spain	Female	35	7	0	2	1	1	65951.6
15	616	Germany	Male	45	3	143129	2	0	1	64327.3

In this image, you can see that dataset is starting from **Credit_Score to the Estimated_Salary.**

Dependent Variable(Y)–

	0
0	1
1	0
2	1
3	0
4	0
5	1
6	0
7	1
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	1

Now we have divided our dataset into **X and Y.** So the next step is-

1.4 Encode Categorical Data–

Why encoding is required…?

Because as we can see, there are two categorical variables-**Geography and Gender.** So we have to encode these categorical variables into some labels such as 0 and 1 for gender. And some hot encoding for geography variable.

So first let's perform **label encoding for gender variable-**

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_2 = LabelEncoder()
X.loc[:, 2] = labelencoder_X_2.fit_transform(X.iloc[:, 2])
```

Why I used 2…?

Because Gender variable has index value 2.

So after performing label encoding on the Gender variable, the male and female are converted in 0 and 1 something like this-

Index	0	1	2	3	4	5	6	7	8	9
0	619	France	0	42	2	0	1	1	1	101349
1	608	Spain	0	41	1	83807.9	1	0	1	112543
2	502	France	0	42	8	159661	3	1	0	113932
3	699	France	0	39	1	0	2	0	0	93826.6
4	850	Spain	0	43	2	125511	1	1	1	79084.1
5	645	Spain	1	44	8	113756	2	1	0	149757
6	822	France	1	50	7	0	2	1	1	10062.8
7	376	Germany	0	29	4	115047	4	1	0	119347
8	501	France	1	44	4	142051	2	0	1	74940.5
9	684	France	1	27	2	134604	1	1	1	71725.7
10	528	France	1	31	6	102017	2	0	0	80181.1
11	497	Spain	1	24	3	0	2	1	0	76390
12	476	France	0	34	10	0	2	1	0	26261
13	549	France	0	25	5	0	2	0	0	190858
14	635	Spain	0	35	7	0	2	1	1	65951.6
15	616	Germany	1	45	3	143129	2	0	1	64327.3
16	653	Germany	1	58	1	132603	1	1	0	5097.67

O represents female and 1 represents the male. Now we have one more categorical variable and that is Geography. Now we will perform One hot encoding to convert France, Spain, and Germany into 0 and 1 form.

One Hot Encoding-

First, we need to apply label encoding similarly as we did in the gender variable. And then we will apply one-hot encoding. So, let's have a look-

```
labelencoder_X_1 = LabelEncoder()
X.loc[:, 1] = labelencoder_X_1.fit_transform(X.iloc[:, 1])
```

After applying label encoding, now it's time to apply One Hot Encoding-

```
onehotencoder = OneHotEncoder(categorical_features = [1])
labelencoder_X_1 = LabelEncoder()
X.loc[:, 1] = labelencoder_X_1.fit_transform(X.iloc[:, 1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
```

So, when you run this code, you will get output something like this-

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	619	0	42	2	0	1	1	1	101349
1	0	1	608	0	41	1	83807.9	1	0	1	112543
2	0	0	502	0	42	8	159661	3	1	0	113932
3	0	0	699	0	39	1	0	2	0	0	93826.6
4	0	1	850	0	43	2	125511	1	1	1	79084.1
5	0	1	645	1	44	8	113756	2	1	0	149757
6	0	0	822	1	50	7	0	2	1	1	10062.8
7	1	0	376	0	29	4	115047	4	1	0	119347
8	0	0	501	1	44	4	142051	2	0	1	74940.5
9	0	0	684	1	27	2	134604	1	1	1	71725.7
10	0	0	528	1	31	6	102017	2	0	0	80181.1
11	0	1	497	1	24	3	0	2	1	0	76390
12	0	0	476	0	34	10	0	2	1	0	26261
13	0	0	549	0	25	5	0	2	0	0	190858
14	0	1	635	0	35	7	0	2	1	1	65951.6
15	1	0	616	1	45	3	143129	2	0	1	64327.3
16	1	0	653	1	58	1	132603	1	1	0	5097.67

Confused after seeing the dataset…?

Here **[0 0] means- France.**

[0 1] means Spain, and

[1 0] means Germany

So, the **first two columns**, represents the **Geography variable**. I hope now you understood.

The next step is splitting the dataset into Training and Test set.

1.5 Split the X and Y Dataset into the Training set and Test set

For building a machine learning model, we need to train our model on the training set. And for checking the performance of our model, we use a Test set. That's why we have to split the X and Y datasets 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)
```

While splitting into training and test set, you have to remember that, 80%-90% of your data should be in the training tests. And that's why I write **test_size = 0.2**.

Now we have splitted our dataset into **X_train, X_test, y-train, and y_test.**

So the next step is **feature scaling.**

1.6 Perform Feature Scaling

As you can see in the dataset, all values are not in the same range especially the **Balance and Estimated_salary.** And that requires a lot of time for calculation. So to overcome this problem, we perform **feature scaling.**

One thing you need to make sure is always perform feature scaling in Deep Learning, no matter you have already values in 0 forms.

Feature scaling help us to **normalize the data within a particular range.**

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

After performing feature scaling, all values are normalized and looks something like this-

	0	1	2	3	4	5	6	7	8	9	10
0	-0.569844	1.74309	0.169582	-1.09169	-0.464608	0.00666099	-1.21572	0.809503	0.642595	-1.03227	1.10643
1	1.75487	-0.573694	-2.30456	0.916013	0.301026	-1.37744	-0.00631193	-0.921591	0.642595	0.968738	-0.748664
2	-0.569844	-0.573694	-1.1912	-1.09169	-0.943129	-1.03142	0.579935	-0.921591	0.642595	-1.03227	1.48533
3	-0.569844	1.74309	0.0355658	0.916013	0.109617	0.00666099	0.473128	-0.921591	0.642595	-1.03227	1.27653
4	-0.569844	1.74309	2.05611	-1.09169	1.73659	1.04474	0.810193	0.809503	0.642595	0.968738	0.558378
5	1.75487	-0.573694	1.29325	-1.09169	-0.177495	-1.03142	0.442535	0.809503	0.642595	-1.03227	1.63252
6	-0.569844	-0.573694	1.61283	0.916013	0.779547	-1.37744	0.304328	-0.921591	-1.55619	-1.03227	0.481496
7	-0.569844	1.74309	-0.541734	0.916013	0.205321	1.04474	-1.21572	0.809503	0.642595	0.968738	1.07382
8	-0.569844	1.74309	-0.149995	0.916013	3.55497	1.39076	0.80633	-0.921591	0.642595	0.968738	-1.0495
9	-0.569844	-0.573694	-0.29432	-1.09169	-0.656016	0.352686	1.48636	0.809503	0.642595	-1.03227	0.0153936
10	-0.569844	-0.573694	0.324216	-1.09169	-0.560312	1.04474	-0.017786	-0.921591	0.642595	0.968738	-1.17159
11	-0.569844	-0.573694	0.612865	0.916013	1.44948	0.352686	1.5191	-0.921591	0.642595	0.968738	1.16193
12	1.75487	-0.573694	-0.58297	-1.09169	-0.943129	-0.68539	0.874975	-0.921591	0.642595	-1.03227	-0.680001
13	-0.569844	1.74309	1.49943	0.916013	0.205321	1.04474	0.502554	-0.921591	0.642595	-1.03227	-1.41995
14	1.75487	-0.573694	-0.459262	0.916013	-0.0817912	-0.68539	0.380665	-0.921591	-1.55619	-1.03227	-1.09703
15	-0.569844	-0.573694	-0.222157	0.916013	0.492434	0.00666099	-1.21572	4.27169	-1.55619	-1.03227	0.302441
16	-0.569844	-0.573694	-1.25305	0.916013	0.301026	-1.37744	1.30115	-0.921591	0.642595	0.968738	-0.311041

Now, we are done with the data preprocessing steps. Now it's time to move to the second part and that is Building the Artificial Neural Network.

2. Build Artificial Neural Network

The first step is-

2.1 Import the Keras libraries and packages

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

2.2 Initialize the Artificial Neural Network

```
classifier = Sequential()
```

The Sequential class allows us to build ANN but as a sequence of layers. As I told you in the theory part that ANN is built with fully connected layers. After initializing the ANN, it's time to-

2.3 Add the input layer and the first hidden layer

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

Dense is the famous class in Tensorflow. Dense is used to add fully connected layer in ANN.

"add" is the method in the **Sequential Class.** output_dim represents the number of hidden neurons in the hidden layer. But there is no rule of thumb for this. That's why I used 6. You can use any other number and check.

The **activation function** in the hidden layer for a fully connected neural network should be the **Rectifier Activation function**. That's why I use **'relu'**.

Our Input layer has **11 neurons**. Why…?

Because we have 11 independent variable(including 2 column of Geography).

That's why **input_dim = 11.** Now we have built our first input layer and one hidden layer. In the next step, we will build the next hidden layer by just copying this code-

2.4 Add the second hidden layer

```
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
```

Here again, we are using 6 hidden neurons in the second hidden layer. Now we have added one input layer and two hidden layers. It's time to add our output layer.

2.5 Add the output layer

```
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

In output layer, we need 1 neuron. Why…?

Because as you can see in the dataset, we have a dependent variable in Binary form. That means we have to predict in 0 or 1 form. That's why only one neuron is required in the output layer.

And I write **output_dim = 1.**

The next thing is **Activation Function**. In output layer, there should be Sigmoid activation function. Why…?

Because Sigmoid activation function allows not only predict but also provides the probability of customer leave the bank or not.

Now we have finally done with the creation of our first Artificial Neural Network. In the next step, we will train our artificial neural network.

3. Train the ANN

The training part requires two steps- Compile the ANN, and Fit the ANN to the Training set. So let's start with the first step-

3.1 Compile the ANN

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

compile is a method of Tensorflow. **"adam'** is the optimizer that can perform the stochastic gradient descent. The optimizer updates the weights during training and reduces the loss. In order to understand the theory behind Gradient Descent, you can check this explanation-

One thing you need to make sure, when you are doing binary prediction similar to this one, always use loss function as **binary_crossentropy**.

For evaluating our ANN model, I am gonna use Accuracy metrics. And that's why **metrics = ['accuracy']**. Now we have compiled our ANN model. The next step is-

3.2 Fit the ANN to the Training set

```
classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
```

Instead of comparing our prediction with real results one by one, it's good to perform in a batch. That's why I write **batch_size = 10**.

The neural network has to train on a certain number of epochs to improve the accuracy over time. So I decided the **nb_epoch = 100**. So when you run this code, you can see the accuracy in each epoch.
In the first epoch the accuracy was-

```
Epoch 1/100
8000/8000 [==============================] - 5s 568us/step
- loss: 0.4848 - accuracy: 0.7958
```

That is 79%, but after running all 100 epoch, the accuracy increase and we get the final accuracy-

```
Epoch 100/100
8000/8000 [==============================] - 1s 187us/step
- loss: 0.3994 - accuracy: 0.8350
```

That is 83%. Quite good. Now we are done with the training part. The last but not least part is Predicting the test set results-

4. Predict the Test Set Results-

```
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

y_pred > 0.5 means if y-pred is in between 0 to 0.5, then this new y_pred will become **0(False)**. And if y_pred is larger than 0.5, then new y_pred will become **1(True).**
So after running this code, you will get y_pred something like this-

	0
0	False
1	False
2	False
3	False
4	False
5	True
6	False
7	False
8	False
9	True
10	False
11	False
12	False
13	False
14	False

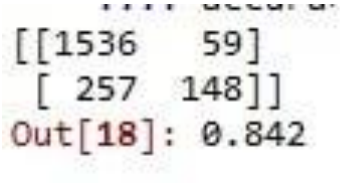
But can you explain by looking at these predicted values, how many values are predicted right, and how many values are predicted wrong?

For a small dataset, you can. But when we have a large dataset, it's quite impossible. And that's why we use a confusion matrix to clear our confusion.
So, the next step is-

5. Make the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test,y_pred)
```

And we got 84.2% accuracy.



That's not bad. Now I would recommend you to experiment with some values, and let me know how much accuracy are you getting?

References:

https://www.youtube.com/watch?v=ER2It2mIagI&t=144s

https://www.youtube.com/watch?v=vpOLiDyhNUA

Text Books:

Peter Harrington "Machine Learning in Action", Dream Tech Press

Han and Kamber,"Data Mining and its concepts", Morgan Kaufmann imprint Elsevier

Reference Books:

William W. Hsieh, "Machine Learning Methods in the Environmental Sciences", Cambridge