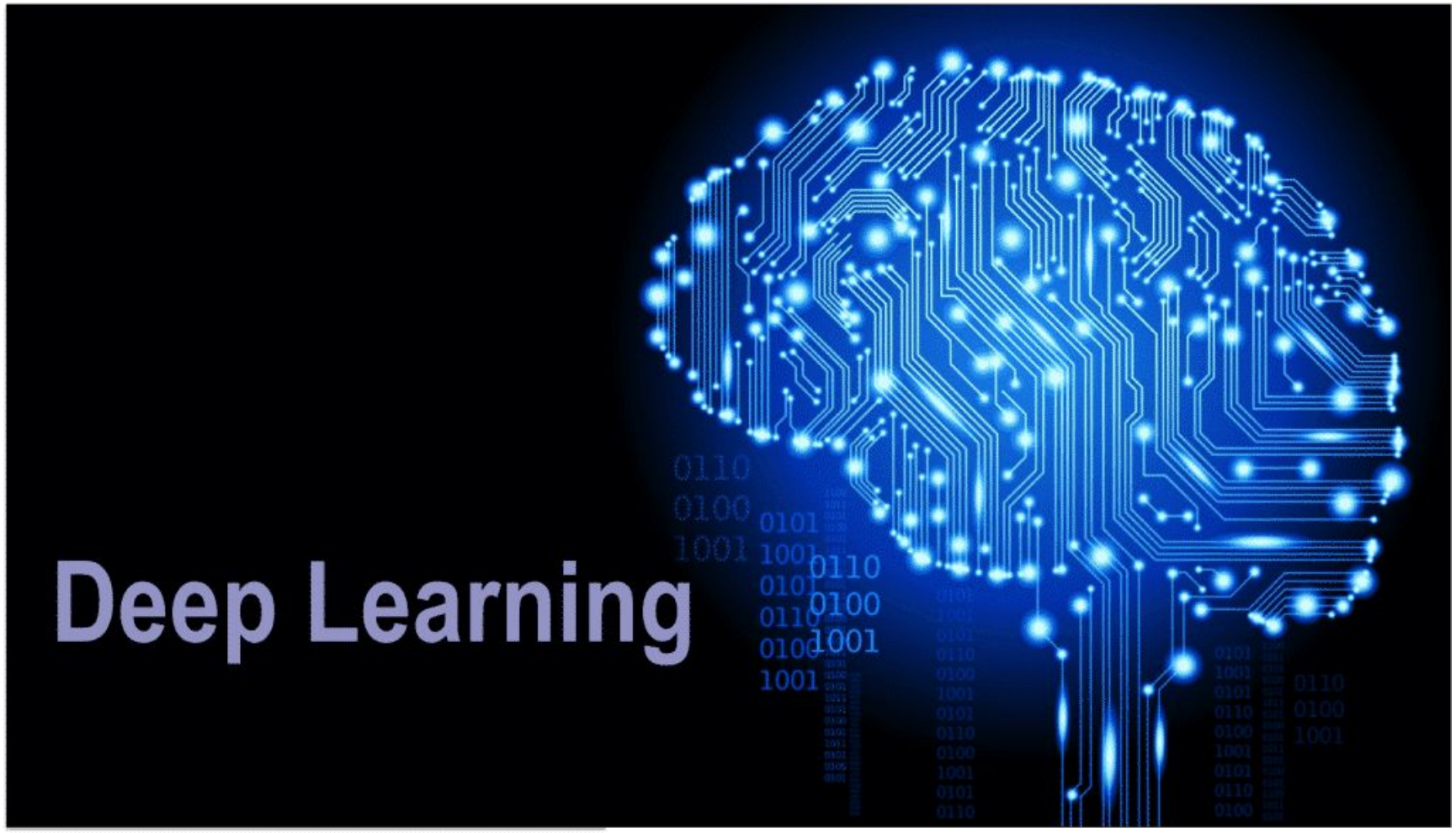

Deep Learning

Introduction To Deep Learning

Deep learning → Deep learning is a subset of machine learning. It focuses on algorithms that are loosely inspired by the structure and functionality of the human brain, specifically neural networks. Deep learning has demonstrated remarkable capabilities in various applications, ranging from image and speech recognition to natural language processing (NLP) and game playing.

At its core, deep learning involves training artificial neural networks on large datasets to perform tasks such as classification, regression, and clustering. Unlike traditional machine learning models, which often rely on manually crafted features, deep learning models automatically discover the optimal features through a hierarchical representation of the data.

Deep Learning Applications

We can find deep learning applications in various of areas:

- **Image Recognition** → From identifying objects in photos to recognizing faces, deep learning models like CNNs have achieved near-human accuracy in various tasks.
- **Speech Recognition** → Systems like Apple's Siri, Google Assistant, and Amazon Alexa rely on deep learning to understand and respond to human speech.
- **Natural Language Processing (NLP)** → Applications such as sentiment analysis, machine translation, and text summarization have seen significant improvements with deep learning, particularly through transformer models.
- **Healthcare** → From diagnosing diseases from medical images to personalized treatment plans, deep learning is making significant inroads in medical research and practice.
- **Autonomous Vehicles** → Deep learning algorithms are at the core of self-driving cars, helping them perceive and navigate the environment.

Artificial Neural Network - ANN

Artificial Neural Network (ANN) → An Artificial Neural Network (ANN) is a computational model inspired by the way biological neural networks in the human brain process information. It is designed to recognize patterns, make decisions, and solve complex problems by mimicking how neurons interact in the brain. ANN model considered as supervised learning model because it's need a labeled data in order to learn.

ANN build from 3 main layers:

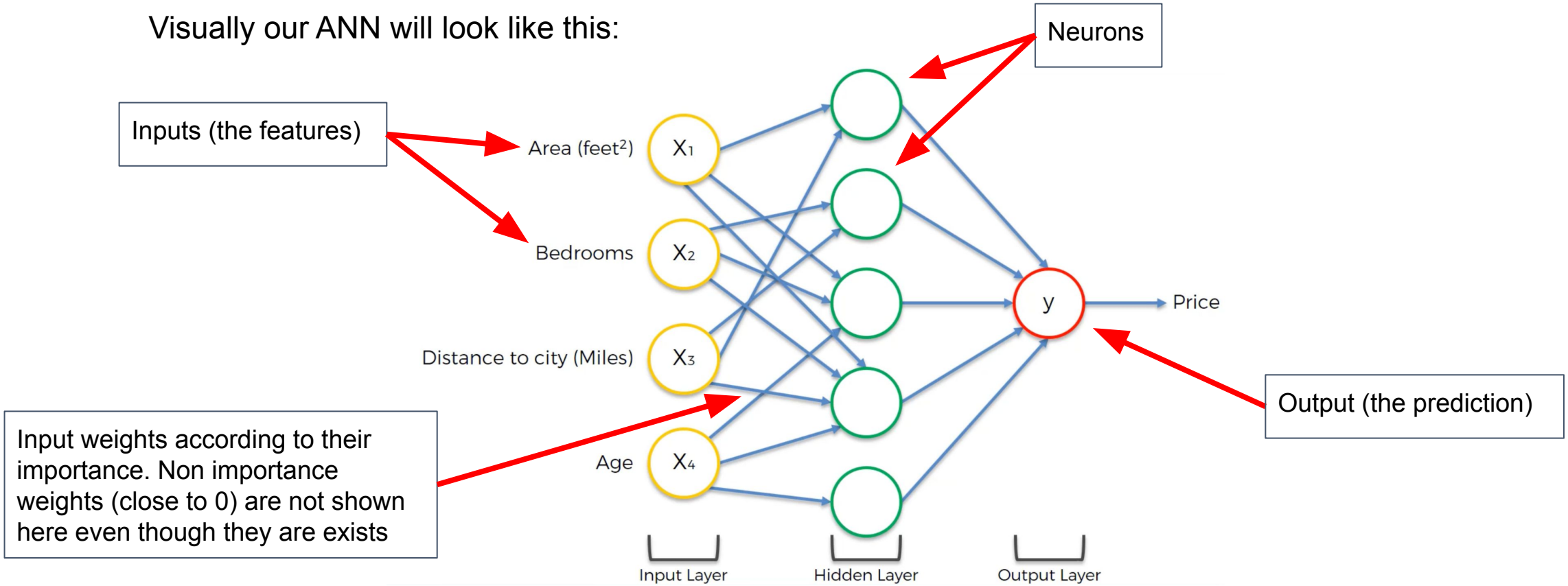
- **Input layer** → Contain the inputs (features) of the model that each neuron will get.
  The input layer is responsible for receiving the input data and passing it onto the next layer.
- **Hidden layer** → Contain the neurons themselves and the activation function.
  The hidden layers are where the actual processing happens. Each neuron in a hidden layer takes inputs from the previous layer, applies a weighted sum, adds a bias, and then applies a non-linear activation function to produce its output.
- **Output layer** → Contain the output (prediction) of the model that is an aggregated information from all neurons in the hidden layer.

Artificial Neural Network - ANN

In order to visualize a typical ANN let's consider the following example.

Let's say we apply ANN in order to predict house pricing (label) according to the following inputs (features) – Area, Bedrooms, Distance to city, Age.

Visually our ANN will look like this:



Artificial Neural Network - The Neuron

What is a neuron in human brain?

Neurons responsibility is to process and transmit information through electrical and chemical signals.

This information is then passed to the brain which decide the outcome for those signals.

The brain itself is hidden within the skull and we can think of it as a ‘black box’.

The brain doesn’t have any ability to understand by itself what is happening outside and it’s totally rely on the neurons to transfer this information to him.

What is a neuron in ANN?

A neuron, also known as a node or unit, is the fundamental building block of an artificial neural network (ANN). Neurons are inspired by biological neurons found in the human brain and perform the essential task of processing and transmitting information within the network.

Artificial Neural Network - The Neuron

A typical artificial neuron includes several key components:

- **Inputs** → The signals or data that the neuron receives from either an external source (input neurons) or from other neurons in the previous layer (hidden and output neurons).
- **Weights** → Each input to a neuron is multiplied by a weight. The weight signifies the importance or strength of the input signal. During the training process, those weights are adjusted to minimize error and improve the model's performance.
- **Activation Function** → An activation function is applied to the weighted sum to introduce non-linearity into the model. This nonlinearity enables the network to capture complex patterns in the data.

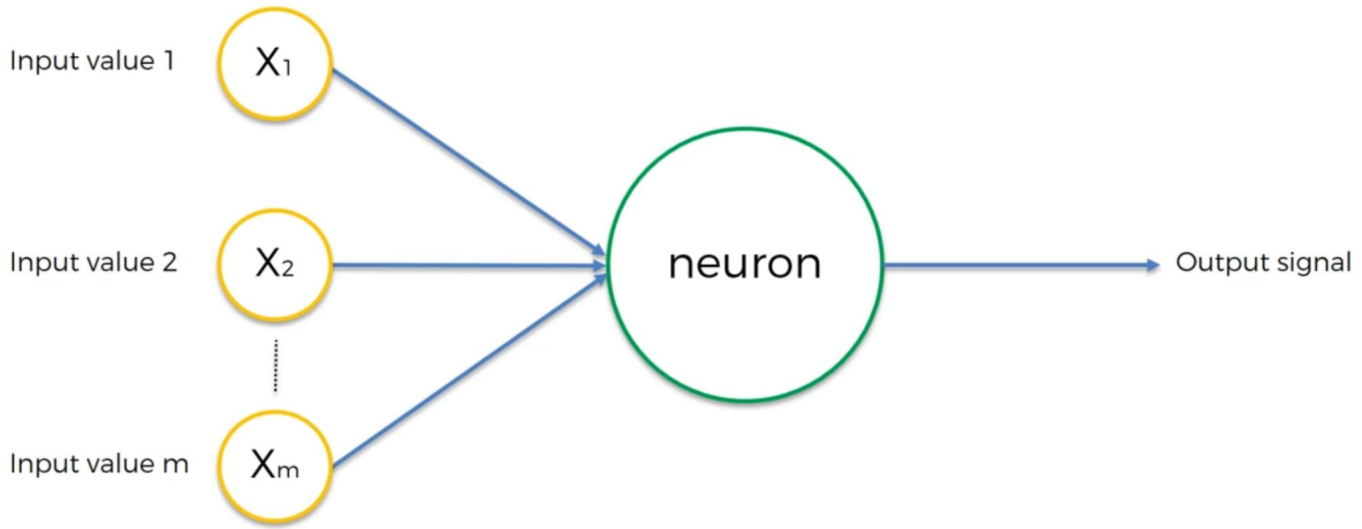
Each neuron can get multiple input values (those can consider as the dataset features).
The output of the neuron is the neuron prediction (Y-hat) according to the label we want to predict.

Note: In ANN every unit in every layer is considering a neuron and for each unit we can configure its corresponding inputs and activation function.

Artificial Neural Network - The Neuron

The neuron output value depend on the type of problem we try to solve:

- **Continuous problem** → Continuous output (like price)
- **Binary problem** → Binary output (like yes / no)
- **Categorial problem** → Categorical output (like belong to a specific category)



Artificial Neural Network - The Neuron

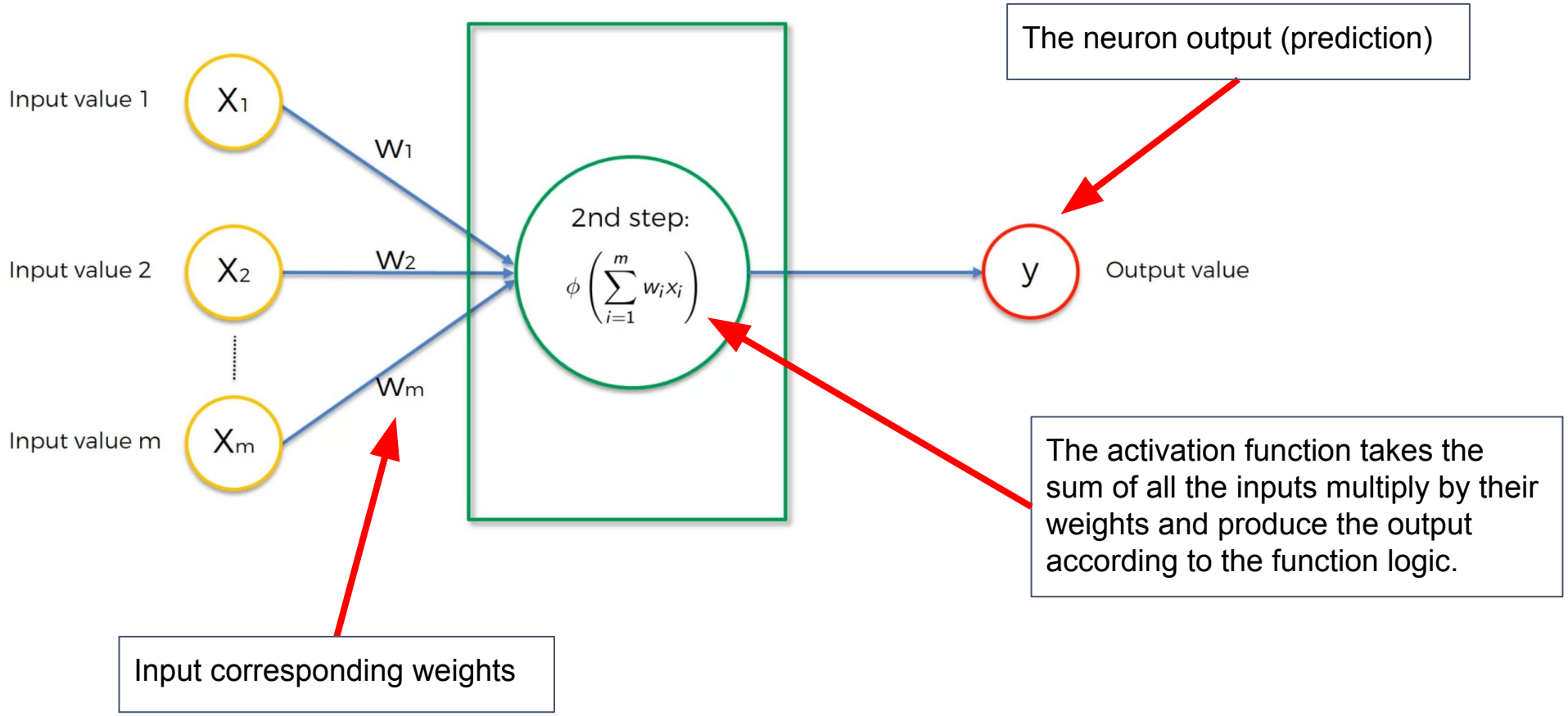
Each neuron perform the following steps for each iteration:

- **Step 1** → The neuron receives input signals from other neurons or from external sources. Each input signal is associated with a weight, which signifies the strength and direction (positive or negative) of the input.
- **Step 2** → Each input signal is multiplied by its corresponding weight. This weight determines the importance or influence of that input on the neuron's output.
- **Step 3** → The neuron sums all the weighted input signals to compute a combined input signal.
- **Step 4** → The combined input signal (summation) is passed through an activation function to produce the neuron's output.
- **Step 5** → The result of the activation function is the output of the neuron. This output can be passed on to other neurons in subsequent layers or can serve as the final output of the network

Note: Before starting an ANN model all the inputs must be normalized or standardized to improve the model performance.

Artificial Neural Network - The Neuron

The neuron work steps can be described in the following chart:



Artificial Neural Network - The Neuron

How the initial input weights been determined?

The initialization of input weights in an Artificial Neural Network (ANN) is a crucial step, as it can significantly affect the training process and the performance of the model.

There are couple of methods for initialization weights but the most common is using **random initialization**.

Here we can select between:

- **Uniform Distribution** → Weights are initialized randomly within a specific range, typically between -0.5 and 0.5 or between -1 and 1.
- **Normal Distribution** → Weights are initialized using a Gaussian (normal) distribution with mean 0 and a small standard deviation.

Note: ANN system involve using many neurons and each of them will have it's own learning process and output. Even if we will provide all neurons the same input values, because of the random weight initialization, the activation functions and other factors in the learning process the neurons will not provide the same output. This independence allows the network to learn a rich set of features and representational capacity.

Artificial Neural Network - Activation Function

Until now we learned that as part of the neuron output processing it trigger an activation function on the sum of the inputs and weights. Let's now understand what is this activation function and what it's purpose in the neuron learning process.

Activation function → An activation function is a mathematical function used in artificial neural networks (ANNs) to determine the output of a neuron. It introduces non-linearity into the network, allowing it to learn and represent complex relationships in data.

Without using an activation function in the output process the ANN will behave just like linear regression model based on weights (betas).

Artificial Neural Network - Activation Function

There are couple of activation functions that we can choose from:

- **Threshold function** → The Threshold function can produce binary output of 0 or 1 based on some threshold value. The most common threshold is the 0 threshold meaning if the sum of inputs and weights is equal or above 0 the output will be 1 and else the output will be 0.

- **Sigmoid function** → The Sigmoid function (or logistic function) produces a smooth, continuous output between 0 and 1, making it especially useful for interpreting results as probabilities. It transforms the input data using the logistic function, which squashes the input values into a range between 0 and 1. This is particularly useful for binary classification problems.

- **Rectified Linear Unit (ReLU)** → The ReLU function produces an output based on the input's value. If the input is positive or zero, the output is the input itself. If the input is negative, the output is zero.

Artificial Neural Network - Activation Function

There are couple of activation functions that we can choose from:

- **Hyperbolic Tangent (Tanh) function** → The Tanh function transforms the input into a value between -1 and 1. It is an S-shaped curve that maps negative input values strongly negative, zero inputs near zero, and positive inputs strongly positive.

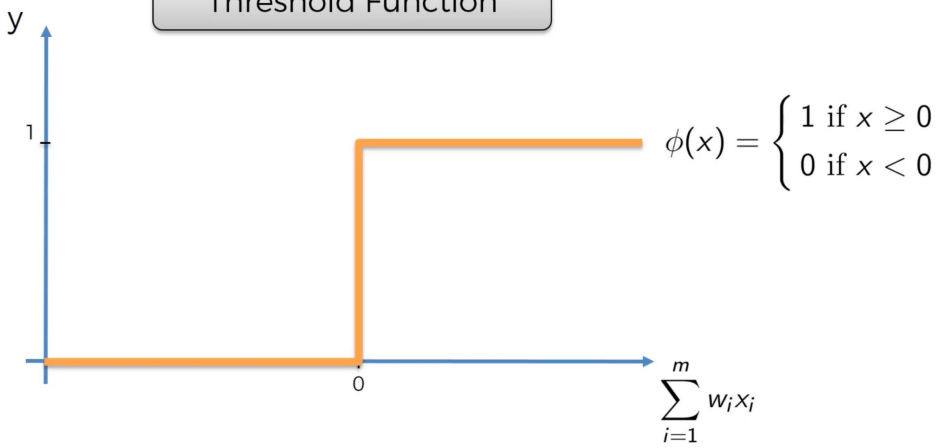
- **Softmax function** → The Softmax function is specifically designed to convert the raw output scores of a neural network into probabilities across multiple classes. It transforms the input vector, which consists of raw score values, into a probability distribution where the sum of all probabilities equals 1.
  The Softmax activation function is particularly suitable for multi-class classification problems.

- **Linear function** → The Linear function is used for regression tasks where the goal is to predict continuous values. Unlike other activation functions that apply a non-linear transformation to the input, the linear activation function leaves the input unchanged, allowing the neural network to predict a wide range of continuous values without any constraints. The linear activation function is particularly suitable for regression problems.

Artificial Neural Network - Activation Function

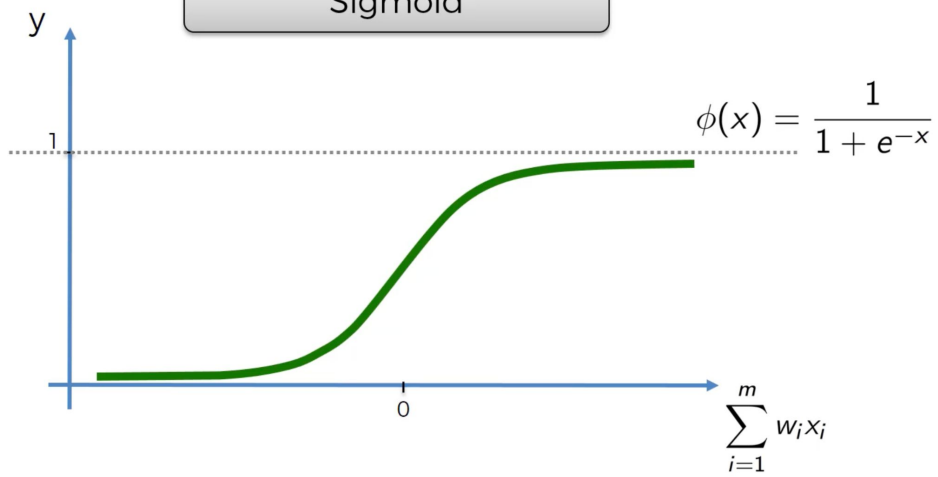
Let’s see how some of those activation functions behave with a visual representation:

Threshold Function



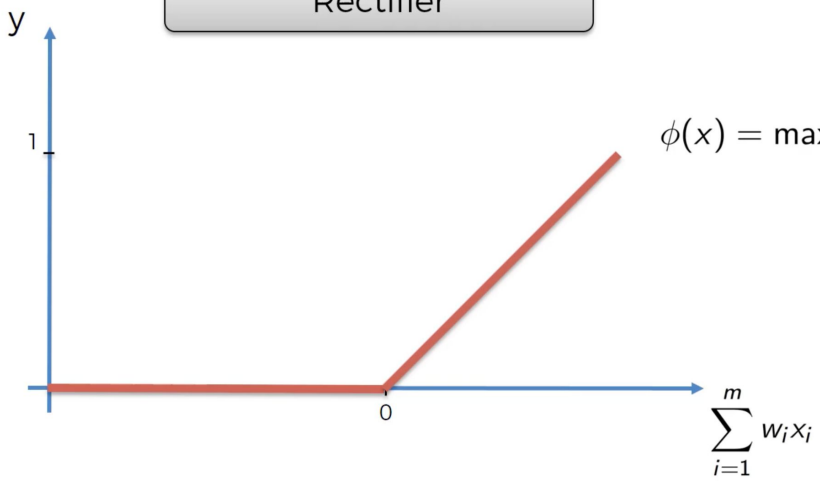
$$\phi(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

Sigmoid



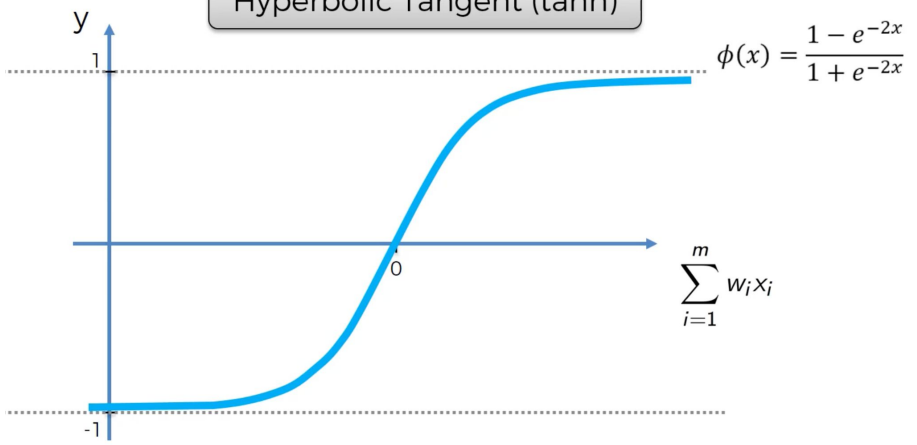
$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Rectifier



$$\phi(x) = \max(x, 0)$$

Hyperbolic Tangent (tanh)



$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Artificial Neural Network - Backpropagation

Until now we learned how the neuron produce its output and how it determine the first weights values for each input. What we still need to understand is how the neuron learn and adjust those weights until it finds the optimal weights for the final neuron output. This process of adjusting the weights and reprocessing the steps again called **Backpropagation**.

Backpropagation → Backpropagation is a fundamental algorithm used to train artificial neural networks, it's purpose is to adjust the weights of the network in order to minimize the error in its predictions.
Similar to linear regression models, the ANN has a cost function that it try to minimize.
ANN will use gradient descent to calculate the optimal weights for each input and adjust them accordingly. Then repeat the process of calculating the sum of inputs X weights and apply the activation function to produce the output. This process of adjusting the weights called Backpropagation.

Artificial Neural Network - Backpropagation

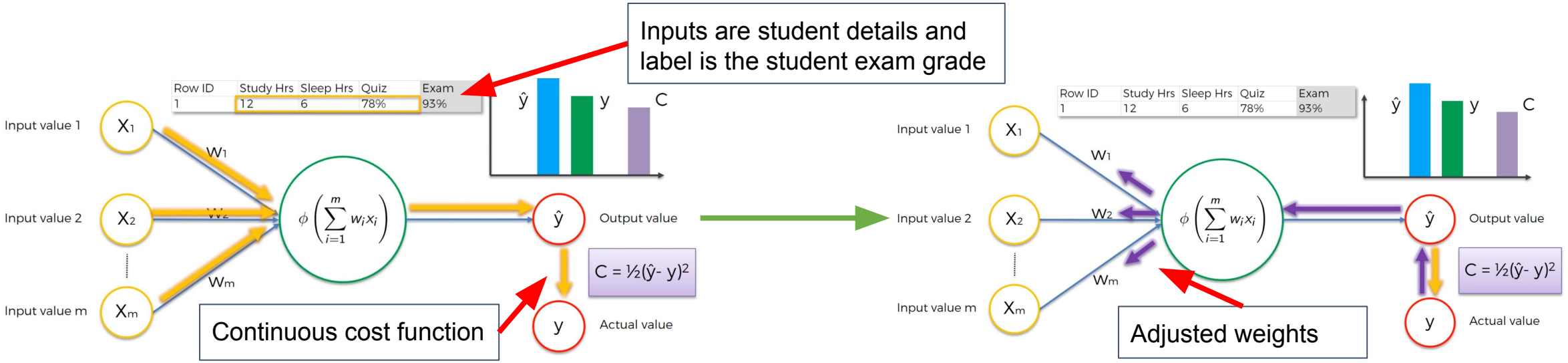
Each neuron will run its own Backpropagation process until finding it's optimal weights.

In addition, the cost function is determined by the type of output required.

For example → For continuous output the ANN will use MSE cost function (similar to linear regression).

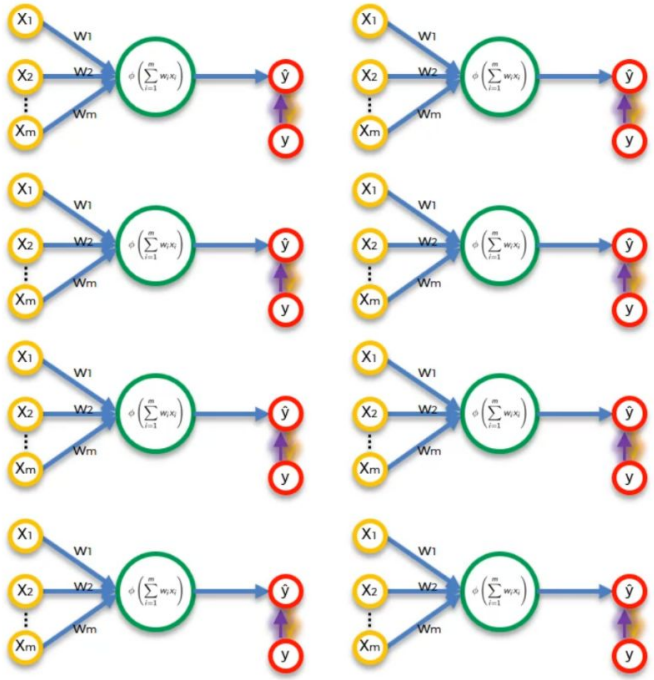
For classification problem the ANN will use Categorical Cross-Entropy which measures the discrepancy between the true labels and the predicted probability distributions.

We can use the following continuous problem example to help us visualize the propagation process:



Artificial Neural Network - Backpropagation

The backpropagation process is applied to each neuron in the ANN and involves using the entire training dataset iteratively until the network collectively minimizes the cost function. This iterative optimization process may use methods like batch, stochastic, or mini-batch gradient descent to update the weights of the network. Although each neuron has its own weights, those weights are updated in a coordinated manner to minimize the collective error of the network as a whole.



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

TensorFlow

Introduction To TensorFlow

TensorFlow is an open-source machine learning library developed by Google. It is widely used for a variety of machine learning applications, including neural networks, which makes it particularly popular in the realms of deep learning and artificial intelligence. TensorFlow supports dense and sparse data, dynamic computation graphs, and automatic differentiation. It is designed to interoperate with other libraries in the Python ecosystem such as NumPy.

Main features in the TensorFlow library:

- TensorFlow supports a wide range of machine learning and deep learning algorithms.
- TensorFlow integrates tightly with Keras, a high-level neural networks API. Keras makes it simple to build and train neural networks, supporting both convolutional and recurrent networks, as well as combinations of the two.
- TensorFlow supports multiple platforms, including mobile (Android, iOS), edge, and more.
- TensorFlow has a very wide community and Ecosystem.

Packages Installation And Import

In order to install TensorFlow package we should run the following command:

```
pip install tensorflow
```

In order the validate the the installation finished successfully, you can run on your Jupyter notebook:

```
import tensorflow as tf
```

We can make sure that our installation succeeded by running the following command and see the TensorFlow version that was installed.

```
In [66]: tf.__version__
Out[66]: '2.16.1'
```

ANN - Python Example

Now that we understand the basic principles of ANN we can move to an actual example.

For this example we will use the **‘Churn_Modelling.csv’** file which contain historical data about bank customers such as Gender, Age, Credit Score, Balance, Has Credit Card, etc…

The dataset also contain a boolean column called ‘Exited’, meaning if the customer eventually left the bank.

We want to use this historical data to train an ANN model in order to predict if future customers will leave the bank or not.

In addition, because this is a categorical problem (with 2 categories – exited or not exited) we will also want to evaluate our model according to categorical metrics such as ‘accuracy’ and ‘confusion matrix’.

ANN - Python Example

First let's extract the dataset and examine the data:

```
In [4]: df = pd.read_csv('/Users/ben.meir/Downloads/Churn_Modelling.csv')
        df = df.drop('RowNumber', axis=1)
        df.head()
```

Out[4]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Next, let's create our feature dataset and label dataset:

```
In [5]: X = df.drop(['CustomerId', 'Surname', 'Exited'], axis=1)
        y = df['Exited']
```

Next, let's do some basic data preparation handling the categorial columns - 'Gender' and 'Geography':

```
In [6]: X = pd.get_dummies(X, columns=['Gender'], drop_first=True)
        X = pd.get_dummies(X, columns=['Geography'], drop_first=False)
        X.head()
```

Out[6]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Gender_Male	Geography_France	Geography_Germany
0	619	42	2	0.00	1	1	1	101348.88	False	True	False
1	608	41	1	83807.86	1	0	1	112542.58	False	False	False
2	502	42	8	159660.80	3	1	0	113931.57	False	True	False
3	699	39	1	0.00	2	0	0	93826.63	False	True	False
4	850	43	2	125510.82	1	1	1	79084.10	False	False	False

The 'drop_first' parameter will drop one of the new dummy columns and keep the rest.
In Gender' we can handle with 1 column 'Gender_Male' but with 'Geography' we need all 3 columns.

ANN - Python Example

Now that we have our datasets ready we can move to splitting the data and standardization.

Those steps are the same as with regular supervised learning models.

```
In [7]: from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
        scaler = StandardScaler()

        scaled_X_train = scaler.fit_transform(X_train)
        scaled_X_test = scaler.transform(X_test)
```

Now we can start to build our ANN model, this model will contain 3 layers (input, hidden and output).

For each layer we will need to decide the following parameters:

- How many neurons the layer will contain?
- What is the layer activation function? (same activation function to all neurons in the layer)

How to select the number of neurons in a layer?

Unfortunately, there is no one-size-fits-all rule for selecting the number of neurons in a layer.

You can always select common choices like 32, 64, 128, etc… and perform iterative optimizations.

Remember that the more neurons you add the more complex your model is.

ANN - Python Example

Regarding the output layer, the number of neurons depend on the problem the model try to predict.

- **For Regression** → Use 1 output neuron.
- **For Binary classification** → Use 1 output neuron.
- **For Multi-class classification** → Use output neurons as the number of classes (for example, for classification problem with 3 class use 3 output neurons).

How to select the layer activation function?

Different activation functions have different properties and behaviors that can affect the model's training process and its ability to capture complex patterns in the data so it’s important to select the right activation problem.

ANN - Python Example

How to select the layer activation function?

- **Input layer** → No need to configure activation function since it's not in use. The purpose of the layer is just to transfer the raw data to the other layers in the system.
- **Hidden layer** → Here we can select from common activation functions like ReLU, Tanh or Sigmoid. We can also explore with different activation functions and evaluate the results.
- **Output layer** → Here we should choose the right output layer activation function based on the model problem.
    - **For Regression** → Use 'Linear' activation function.
    - **For Binary classification** → Use 'Sigmoid' activation function.
    - **For Multi-class classification** → Usd 'Softmax' activation function.

ANN - Python Example

Let's create the ANN model instance and add the relevant layers and layer configuration:

```
In [11]: ann = tf.keras.models.Sequential()

         ## First layer - Input layer
         ann.add(tf.keras.layers.Dense(units=10))

         ## Second layer - Hidden layer
         ann.add(tf.keras.layers.Dense(units=10, activation='relu'))

         ## Third layer - Output layer
         ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Create The ANN instance

For the hidden layer we will use the 'ReLU' activation function

For the output layer we will use the 'Sigmoid' activation function

The next phase is to determine the model parameters:

- **'optimizer'** → Determine the optimization algorithm for adjust the weights of the network in order to minimize the loss function. We can choose from different algorithms such as 'Gradient Descent', 'Stochastic Gradient Descent', 'Batch Gradient Descent' and more.
- **'loss'** → Determine the loss function that the model will try to minimize during the training process.
- **'metrics'** → `metrics` are used to evaluate the performance of your model during training and testing. Unlike loss functions, which are used to train the model by guiding the optimization process, metrics are used strictly for monitoring and reporting.

ANN - Python Example

Let's compile the ANN model with the selected parameters:

- **'optimizer'** → For the optimizer algorithm we will choose 'sgd' which mean Stochastic Gradient Descent.
- **'loss'** → For the loss function we will choose 'binary_crossentropy' which is right loss function for binary categorical problems.
- **'metrics'** → For metrics that we want to evaluate during the training process we will choose 'accuracy'.

```
In [17]: ann.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
```

The final phase is to execute the fit() method which will start the training process.

Here we can also select additional model parameters:

- **'batch_size'** → Specifies the number of rows that will be processed together in each batch during training. Batching allows the model to efficiently handle large datasets by splitting the dataset into smaller, more manageable subsets.
- **'epochs'** → Specifies the number of iterations the model will perform in order to optimize the weights values. More epochs meaning the model will perform more iterations on the entire dataset.

ANN - Python Example

Finally, let's call the fit() method with the selected parameters and execute the training process.

```
In [16]: ann.fit(scaled_X_train, y_train, batch_size=32, epochs=100)
Epoch 92/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 340us/step - accuracy: 0.8557 - loss: 0.3391
Epoch 93/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 308us/step - accuracy: 0.8602 - loss: 0.3331
Epoch 94/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 307us/step - accuracy: 0.8542 - loss: 0.3431
Epoch 95/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 307us/step - accuracy: 0.8517 - loss: 0.3459
Epoch 96/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 311us/step - accuracy: 0.8609 - loss: 0.3364
Epoch 97/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 317us/step - accuracy: 0.8538 - loss: 0.3384
Epoch 98/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 307us/step - accuracy: 0.8512 - loss: 0.3443
Epoch 99/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 309us/step - accuracy: 0.8608 - loss: 0.3318
Epoch 100/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 307us/step - accuracy: 0.8526 - loss: 0.3433

Out[16]: <keras.src.callbacks.history.History at 0x2997a0d60>
```

We selected batch_size of 32 meaning the data for each iteration will be splitted into batches of 32 rows.
We selected epochs of 100 meaning.

We want 100 different iteration on the entire dataset until finding the optimal weights for the model prediction.

TensorFlow provide us logging during the training process.
Each log will be printed in the end of each iteration (epoch) and will provide the corresponding metric value.
According to the metrics we selected in the 'metrics' parameter.

ANN - Python Example

Once the model finish it's training process we can get it's prediction on the test set.

In addition we can evaluate the model using relevant metrics.

```
In [18]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

         y_pred = (ann.predict(scaled_X_test) > 0.5)

         print(accuracy_score(y_test, y_pred))
         print(confusion_matrix(y_test, y_pred))

         94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 411us/step
         0.868
         [[2329   87]
          [ 309  275]]
```

The model predict the probability to belong to each class (exited / not exited) but the actual label values are boolean and not probability values.
In order to compare correctly the labels with the predictions we transform the prediction values into boolean using threshold.

We can also use the model to predict new unknown data:

```
In [21]: new_customer_details = [[600, 40, 3, 60000, 2, 1, 1, 50000, 1, 1, 0, 0]]

         scaled_customer_details = scaler.transform(new_customer_details)

         ann.predict(scaled_customer_details) > 0.5

         1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

Out[21]: array([[False]])
```

The model predict that the provided customer will not leave the bank (exit = Fales)