Day 14 of Training at Ansh Info Tech

Topics Covered

- **Introduction to AI, ML, and DL**
- **Artificial Neural Networks**
- **Perceptrons**
- **Activation Functions**
    - Threshold, Sigmoid, ReLU, Softmax, Hyperbolic Tangent Function
- **Neural Networks**
- **Back Propagation**
- **ANN Practice Model**

Summary

Introduction to AI, ML, and DL

Artificial Intelligence (AI) is the simulation of human intelligence in machines. Machine Learning (ML) is a subset of AI that involves training algorithms to learn patterns from data. Deep Learning (DL) is a subset of ML focused on neural networks with many layers.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the human brain. They consist of interconnected nodes (neurons) that process data in layers, allowing for complex pattern recognition.

Perceptrons

A perceptron is the simplest type of artificial neural network. It consists of a single neuron with adjustable weights and a bias, used for binary classification tasks.

Activation Functions

Activation functions introduce non-linearity into the neural network, enabling it to learn complex patterns. Common activation functions include:

- **Threshold**: Outputs 1 if input exceeds a certain threshold, otherwise 0.
- **Sigmoid**: Produces an S-shaped curve, outputting values between 0 and 1.
- **ReLU (Rectified Linear Unit)**: Outputs the input directly if it is positive, otherwise 0.
- **Softmax**: Converts outputs into a probability distribution.
- **Hyperbolic Tangent (Tanh)**: Produces values between -1 and 1.

Neural Networks

Neural networks are composed of multiple layers of neurons, including input, hidden, and output layers. They are designed to recognize patterns and make predictions based on data.

Back Propagation

Back propagation is the learning algorithm used for training neural networks. It adjusts the weights of the neurons by propagating the error backward from the output layer to the input layer, minimizing the error through optimization techniques like gradient descent.

ANN Practice Model

Implementing a practice model involves creating a simple ANN using a neural network framework or library. This model can be trained on a dataset to perform tasks such as classification or regression, allowing for hands-on experience with the concepts learned.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris


iris = load_iris()


iris.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
print(iris['DESCR'])
```

```
**Data Set Characteristics:**

        :Number of Instances: 150 (50 in each of three classes)
        :Number of Attributes: 4 numeric, predictive attributes and the class
        :Attribute Information:
            - sepal length in cm
            - sepal width in cm
            - petal length in cm
            - petal width in cm
            - class:
                    - Iris-Setosa
                    - Iris-Versicolour
                    - Iris-Virginica

        :Summary Statistics:

        ============== ==== ==== ======= ===== ====================
                        Min  Max   Mean    SD   Class Correlation
        ============== ==== ==== ======= ===== ====================
        sepal length:   4.3  7.9   5.84   0.83    0.7826
        sepal width:    2.0  4.4   3.05   0.43   -0.4194
        petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
        petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
        ============== ==== ==== ======= ===== ====================

        :Missing Attribute Values: None
        :Class Distribution: 33.3% for each of 3 classes.
        :Creator: R.A. Fisher
        :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
        :Date: July, 1988

    The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
    from Fisher's paper. Note that it's the same as in R, but not as in the UCI
    Machine Learning Repository, which has two wrong data points.

    This is perhaps the best known database to be found in the
    pattern recognition literature.  Fisher's paper is a classic in the field and
    is referenced frequently to this day.  (See Duda & Hart, for example.)  The
    data set contains 3 classes of 50 instances each, where each class refers to a
    type of iris plant.  One class is linearly separable from the other 2; the
    latter are NOT linearly separable from each other.

    .. topic:: References

       - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
         Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
         Mathematical Statistics" (John Wiley, NY, 1950).
       - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
         (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
       - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
         Structure and Classification Rule for Recognition in Partially Exposed
         Environments".  IEEE Transactions on Pattern Analysis and Machine
         Intelligence, Vol. PAMI-2, No. 1, 67-71.
       - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
         on Information Theory, May 1972, 431-433.
       - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
         conceptual clustering system finds 3 classes in the data.
       - Many, many more ...
```

```
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
df['target'] = iris['target']
```

```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df.sample(10)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
142	5.8	2.7	5.1	1.9	2
100	6.3	3.3	6.0	2.5	2
133	6.3	2.8	5.1	1.5	2
126	6.2	2.8	4.8	1.8	2
50	7.0	3.2	4.7	1.4	1
112	6.8	3.0	5.5	2.1	2
83	6.0	2.7	5.1	1.6	1
94	5.6	2.7	4.2	1.3	1
149	5.9	3.0	5.1	1.8	2
117	7.7	3.8	6.7	2.2	2

```
X = iris['data']
y = iris['target']
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
```

```
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
```

```
from tensorflow.keras.utils import to_categorical
```

```
y_cat = to_categorical(y)
```

```
y_cat
```

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
```

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

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y_cat, test_size = 0.33)
```

```
X_train.shape
```

```
(100, 4)
```

```
X_test.shape
```

```
(50, 4)
```

```
y_train.shape
```

```
(100, 3)
```

```
y_test.shape
```

```
(50, 3)
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
scaler.fit(X_train)
```

```
MinMaxScaler
MinMaxScaler()
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

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

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stop = EarlyStopping(patience = 10)
```

```
model = Sequential()

model.add(Dense(4, activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(3, activation = 'softmax'))

model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
```

```
model.fit(X_train, y_train, epochs = 500, validation_data = (X_test, y_test), callbacks = [early_stop])
```

```
Epoch 1/500
4/4 [==============================] - 1s 96ms/step - loss: 1.0770 - accuracy: 0.0600 - val_loss: 1.0693 - val_accuracy: 0.3400
Epoch 2/500
4/4 [==============================] - 0s 13ms/step - loss: 1.0703 - accuracy: 0.3300 - val_loss: 1.0638 - val_accuracy: 0.3200
Epoch 3/500
```

```
4/4 [==============================] - 0s 13ms/step - loss: 1.0640 - accuracy: 0.3300 - val_loss: 1.0582 - val_accuracy: 0.3200
Epoch 4/500
4/4 [==============================] - 0s 18ms/step - loss: 1.0579 - accuracy: 0.3300 - val_loss: 1.0524 - val_accuracy: 0.4000
Epoch 5/500
4/4 [==============================] - 0s 19ms/step - loss: 1.0518 - accuracy: 0.4900 - val_loss: 1.0467 - val_accuracy: 0.6800
Epoch 6/500
4/4 [==============================] - 0s 22ms/step - loss: 1.0459 - accuracy: 0.6400 - val_loss: 1.0415 - val_accuracy: 0.7000
Epoch 7/500
4/4 [==============================] - 0s 21ms/step - loss: 1.0409 - accuracy: 0.6400 - val_loss: 1.0368 - val_accuracy: 0.7000
Epoch 8/500
4/4 [==============================] - 0s 18ms/step - loss: 1.0363 - accuracy: 0.6400 - val_loss: 1.0321 - val_accuracy: 0.7000
Epoch 9/500
4/4 [==============================] - 0s 12ms/step - loss: 1.0314 - accuracy: 0.6400 - val_loss: 1.0270 - val_accuracy: 0.7000
Epoch 10/500
4/4 [==============================] - 0s 12ms/step - loss: 1.0265 - accuracy: 0.6400 - val_loss: 1.0217 - val_accuracy: 0.7200
Epoch 11/500
4/4 [==============================] - 0s 17ms/step - loss: 1.0207 - accuracy: 0.6400 - val_loss: 1.0159 - val_accuracy: 0.7200
Epoch 12/500
4/4 [==============================] - 0s 13ms/step - loss: 1.0150 - accuracy: 0.6500 - val_loss: 1.0100 - val_accuracy: 0.7600
Epoch 13/500
4/4 [==============================] - 0s 19ms/step - loss: 1.0093 - accuracy: 0.6500 - val_loss: 1.0040 - val_accuracy: 0.7600
Epoch 14/500
4/4 [==============================] - 0s 18ms/step - loss: 1.0030 - accuracy: 0.6600 - val_loss: 0.9982 - val_accuracy: 0.7600
Epoch 15/500
4/4 [==============================] - 0s 20ms/step - loss: 0.9975 - accuracy: 0.6700 - val_loss: 0.9917 - val_accuracy: 0.7600
Epoch 16/500
4/4 [==============================] - 0s 13ms/step - loss: 0.9911 - accuracy: 0.6700 - val_loss: 0.9849 - val_accuracy: 0.7800
Epoch 17/500
4/4 [==============================] - 0s 18ms/step - loss: 0.9849 - accuracy: 0.6700 - val_loss: 0.9779 - val_accuracy: 0.7600
Epoch 18/500
4/4 [==============================] - 0s 18ms/step - loss: 0.9783 - accuracy: 0.6700 - val_loss: 0.9710 - val_accuracy: 0.7800
Epoch 19/500
4/4 [==============================] - 0s 12ms/step - loss: 0.9717 - accuracy: 0.7000 - val_loss: 0.9638 - val_accuracy: 0.7800
Epoch 20/500
4/4 [==============================] - 0s 17ms/step - loss: 0.9649 - accuracy: 0.7000 - val_loss: 0.9572 - val_accuracy: 0.7800
Epoch 21/500
4/4 [==============================] - 0s 18ms/step - loss: 0.9586 - accuracy: 0.7200 - val_loss: 0.9504 - val_accuracy: 0.7800
Epoch 22/500
4/4 [==============================] - 0s 20ms/step - loss: 0.9518 - accuracy: 0.7000 - val_loss: 0.9437 - val_accuracy: 0.7800
Epoch 23/500
4/4 [==============================] - 0s 21ms/step - loss: 0.9454 - accuracy: 0.7600 - val_loss: 0.9372 - val_accuracy: 0.8200
Epoch 24/500
4/4 [==============================] - 0s 20ms/step - loss: 0.9391 - accuracy: 0.7800 - val_loss: 0.9301 - val_accuracy: 0.8200
Epoch 25/500
4/4 [==============================] - 0s 22ms/step - loss: 0.9321 - accuracy: 0.8000 - val_loss: 0.9238 - val_accuracy: 0.8600
Epoch 26/500
4/4 [==============================] - 0s 18ms/step - loss: 0.9256 - accuracy: 0.8300 - val_loss: 0.9175 - val_accuracy: 0.8600
Epoch 27/500
4/4 [==============================] - 0s 18ms/step - loss: 0.9193 - accuracy: 0.8600 - val_loss: 0.9112 - val_accuracy: 0.8600
Epoch 28/500
4/4 [==============================] - 0s 17ms/step - loss: 0.9131 - accuracy: 0.8600 - val_loss: 0.9043 - val_accuracy: 0.8600
Epoch 29/500
4/4 [==============================] - 0s 17ms/step - loss: 0.9062 - accuracy: 0.8600 - val_loss: 0.8970 - val_accuracy: 0.8600
```

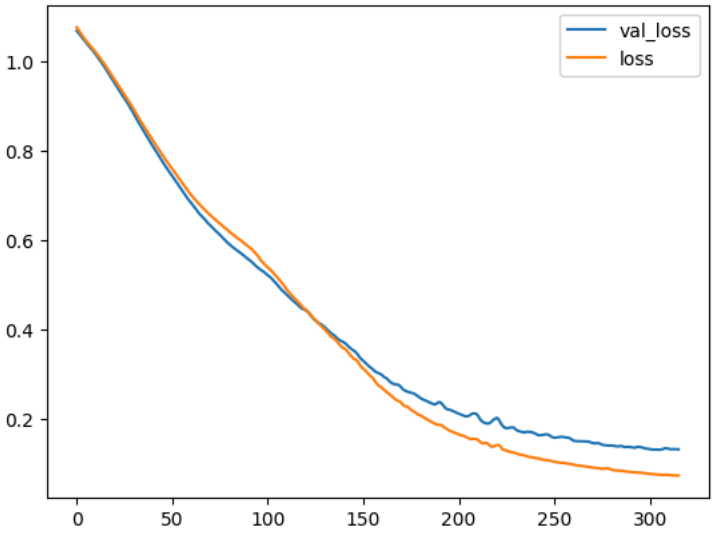
```
loss_df = pd.DataFrame(model.history.history)
loss_df
```

	loss	accuracy	val_loss	val_accuracy
0	1.076999	0.06	1.069339	0.34
1	1.070253	0.33	1.063819	0.32
2	1.064000	0.33	1.058177	0.32
3	1.057942	0.33	1.052365	0.40
4	1.051843	0.49	1.046720	0.68
...	...	...	...	...
311	0.073963	0.98	0.131747	0.96
312	0.073150	0.98	0.132248	0.96
313	0.073195	0.98	0.131888	0.96
314	0.072877	0.98	0.131986	0.96
315	0.072997	0.98	0.131660	0.96

316 rows × 4 columns

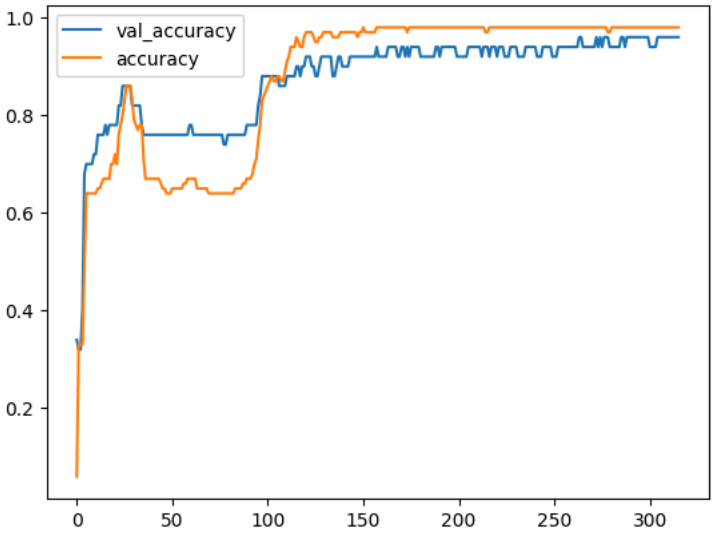
```
loss_df[['val_loss', 'loss']].plot()
```

<Axes: >




```
loss_df[['val_accuracy', 'accuracy']].plot()
```

<Axes: >




```
predictions = model.predict(X_test)
```

```
2/2 [==============================] - 0s 5ms/step
```

```
predictions = np.argmax(predictions, axis = 1)
```

```
actual_y = np.argmax(y_test, axis = 1)
```

```
from sklearn.metrics import classification_report
```

```
print(classification_report(actual_y, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      0.86      0.92        14
           2       0.90      1.00      0.95        19

    accuracy                           0.96        50
   macro avg       0.97      0.95      0.96        50
weighted avg       0.96      0.96      0.96        50
```

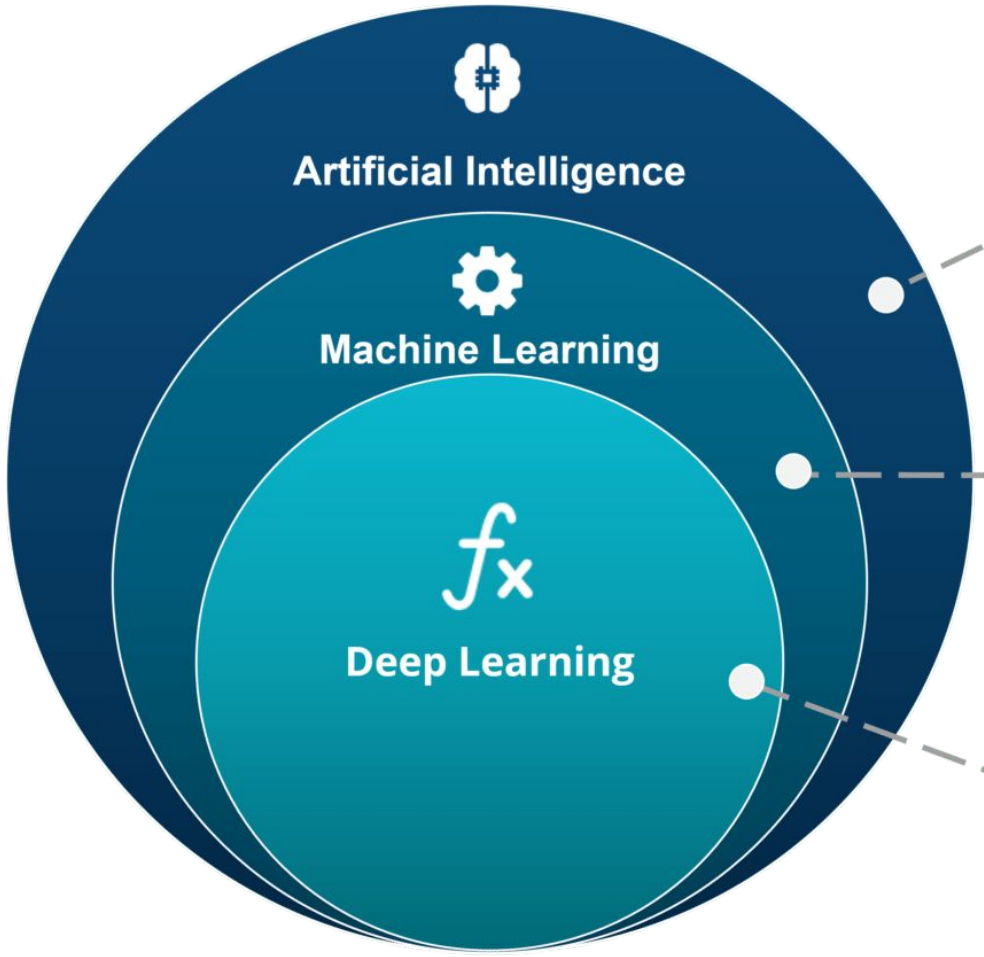
Start coding or generate with AI.

Start coding or generate with AI.

Deep Learning from Scratch

Theory + Practical

AI vs ML vs DL



ARTIFICIAL INTELLIGENCE

A technique which enables machines to mimic human behaviour

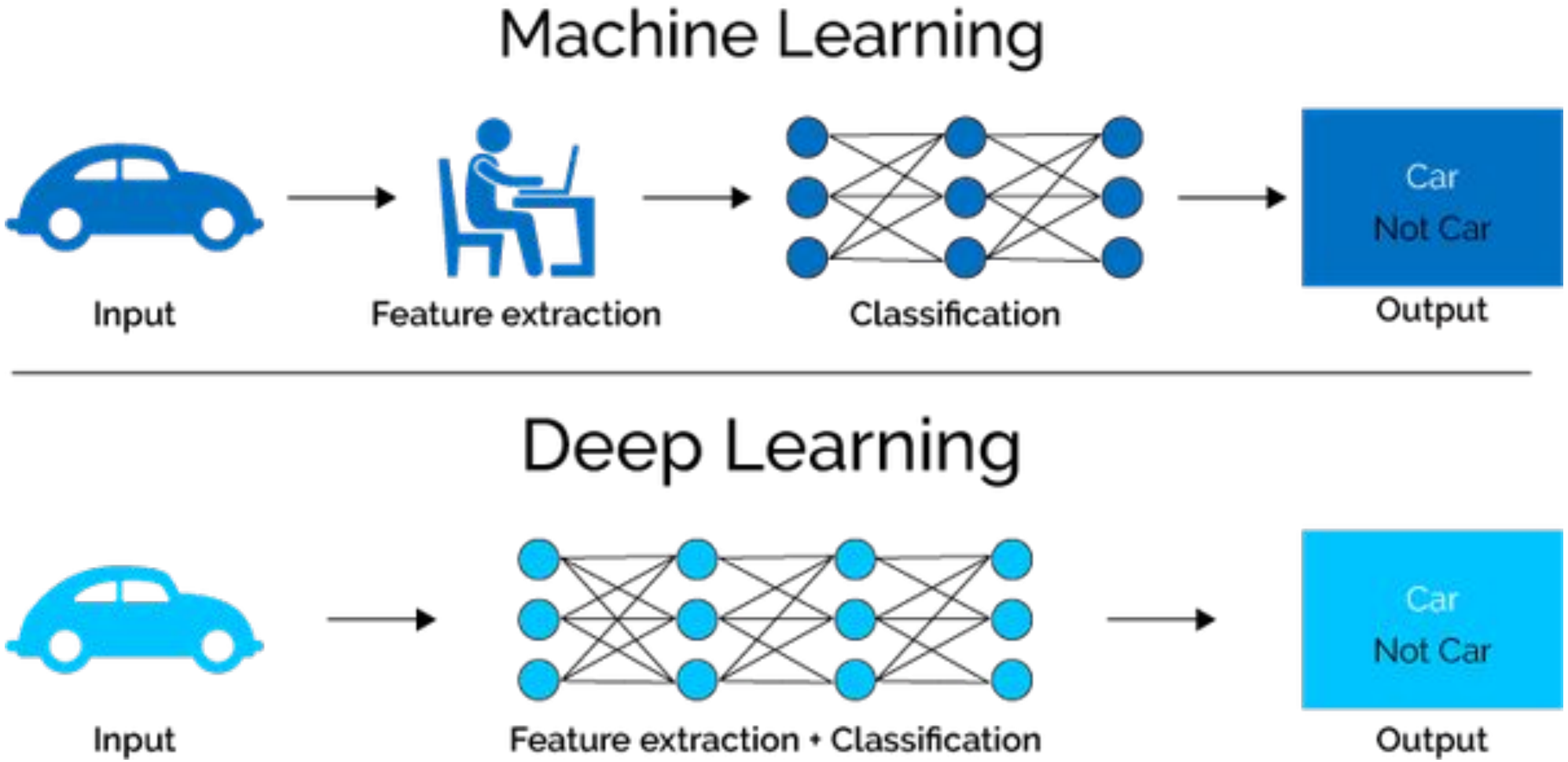
MACHINE LEARNING

Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING

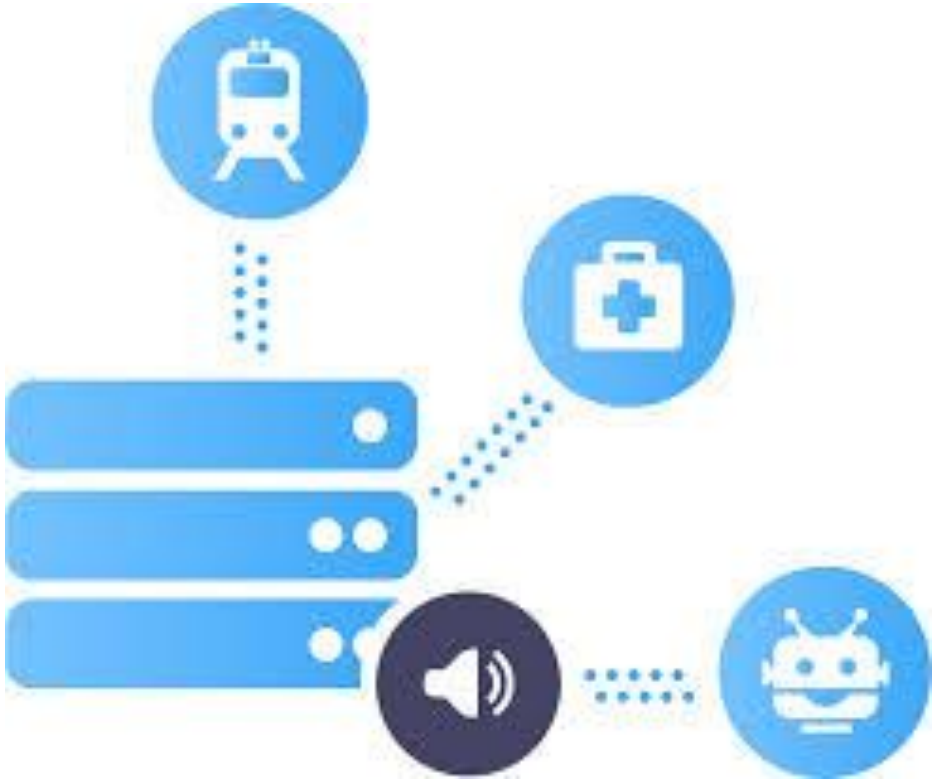
Subset of ML which make the computation of multi-layer neural network feasible

ML vs DL


Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

So, What do you think, what is Deep Learning

Deep learning is a machine learning technique that learns features and task directly from the data, where data may by images, text or sound!



So, What we learn in this course

- Artificial Neural Network
- Convolutional neural network
- Recurrent neural network
- Boltzmann machine Vs Deep BM
- Self organizing maps
- Autoencoder
- GAN (Generative Adversarial Network)
- Deep Q Learning
- ***Pre Train Model (CNN Architecture and many more…***

Keras

Vs

TensorFlow

Thank You!

Deep Learning from Scratch

Theory + Practical

Nidhi Chouhan

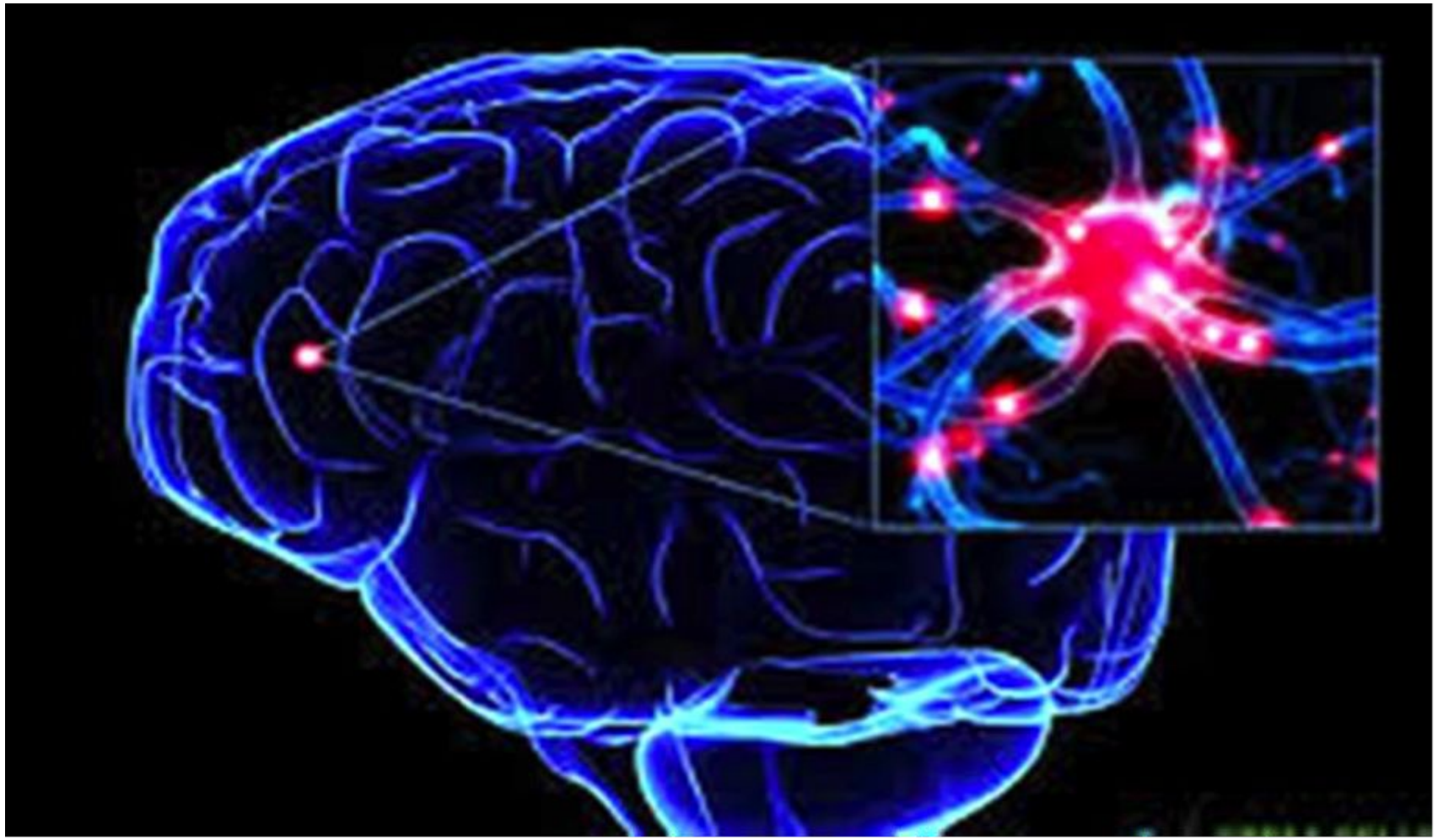
So, What we learn in this course

Supervised **Unsupervised** **Reinforcement Learning**

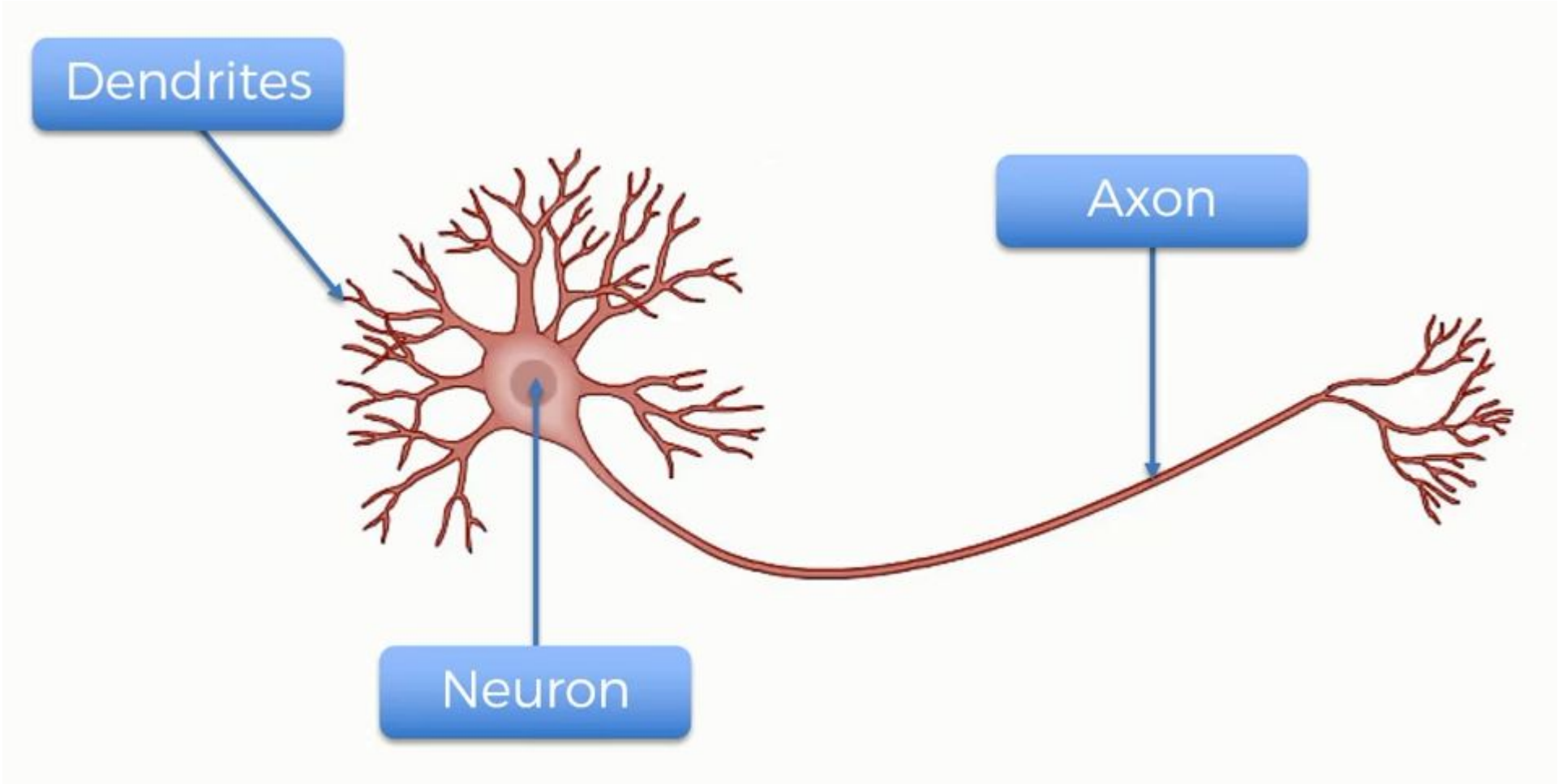
Artificial Neural Network

Regression and Classification

What is Neural



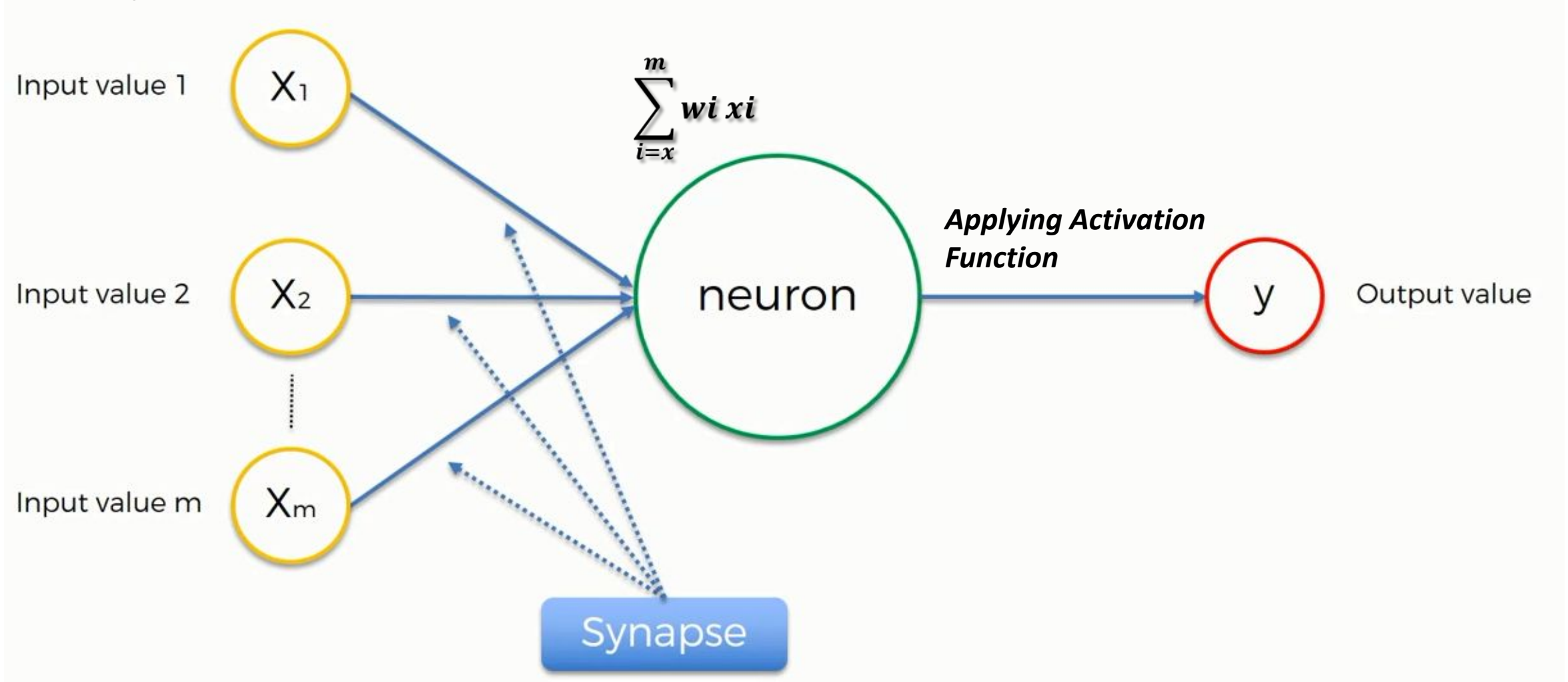
Neural Working



Perceptron in deep learning

Artificial Neural Network

Normalize/Standardize



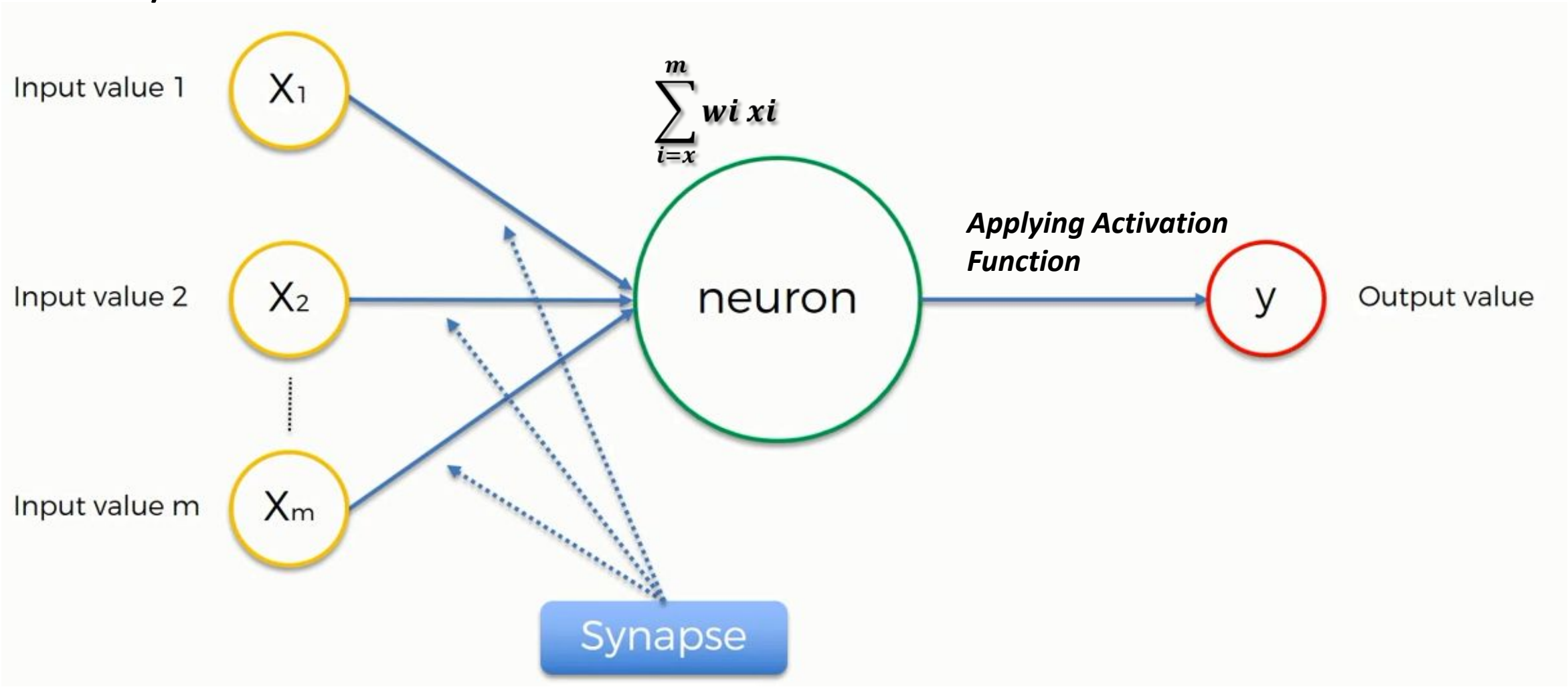
Deep Learning from Scratch

Theory + Practical

Nidhi chouhan

Artificial Neural Network

Normalize/Standardize



What is an Activation Function?

Activation functions are an extremely important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

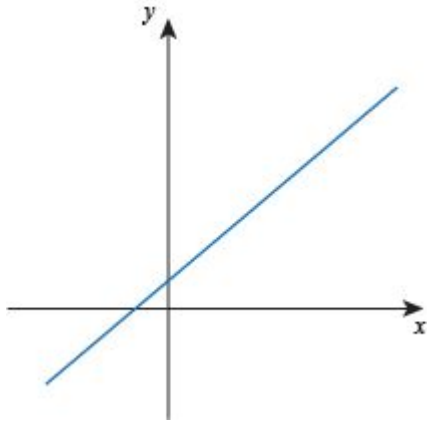
The activation function is the non linear transformation that we do over the input signal. This transformed output is then seen to the next layer of neurons as input.

- Linear **Activation Function**
- Non Linear **Activation Function**

What is an Activation Function?

Linear Function

The function is a line or linear. Therefore, the output of the functions will not be confined between any range



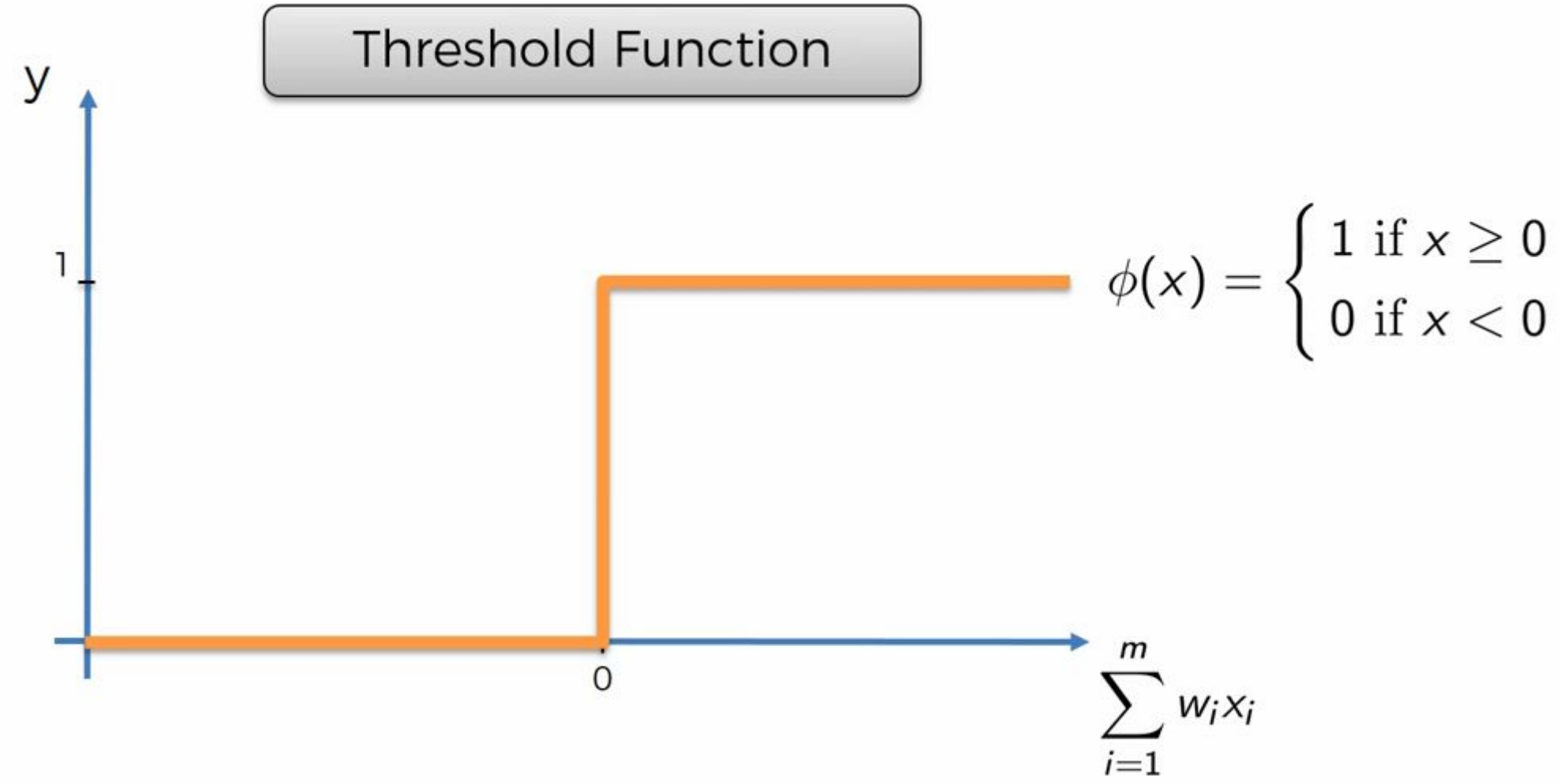
Non Linear Function

They make it easy for the model to generalize or adapt with variety of data and to differentiate between the output

The Nonlinear Activation Functions are mainly divided on the basis of their **range or curves**

1. Threshold
2. Sigmoid
3. Tanh
4. ReLU
5. Leaky ReLU
6. Softmax

Threshold Function?

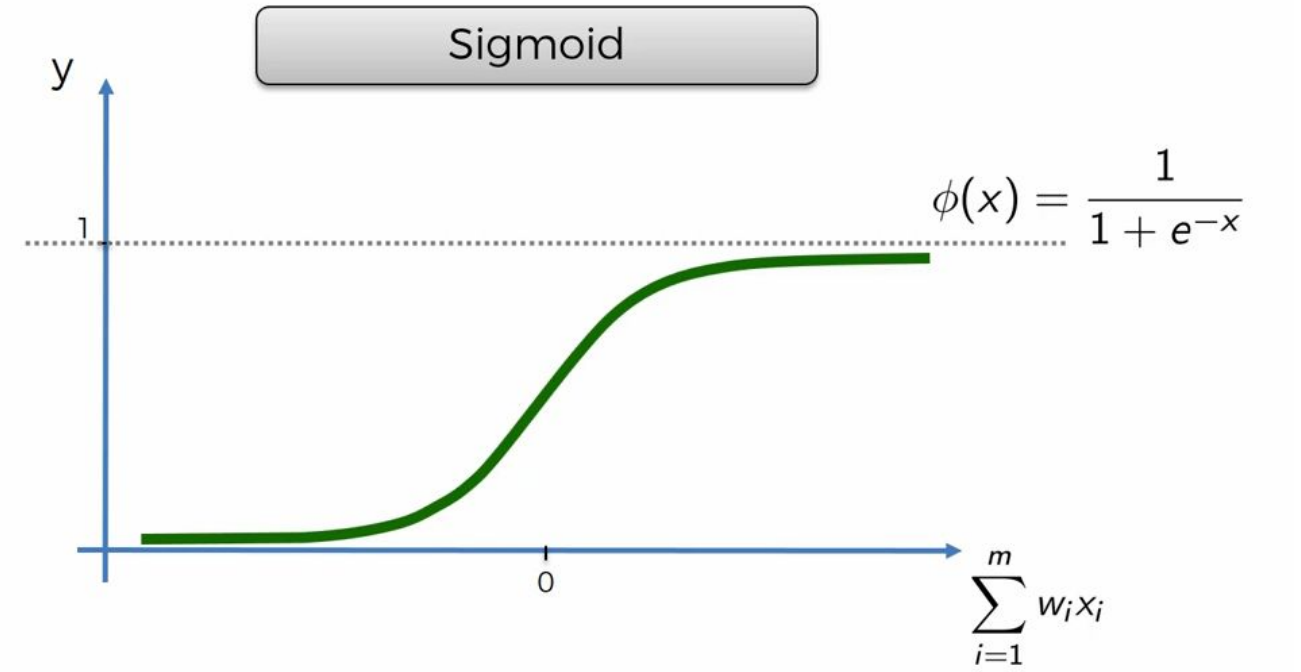


Sigmoid Function?

The Sigmoid Function curve looks like a S-shape

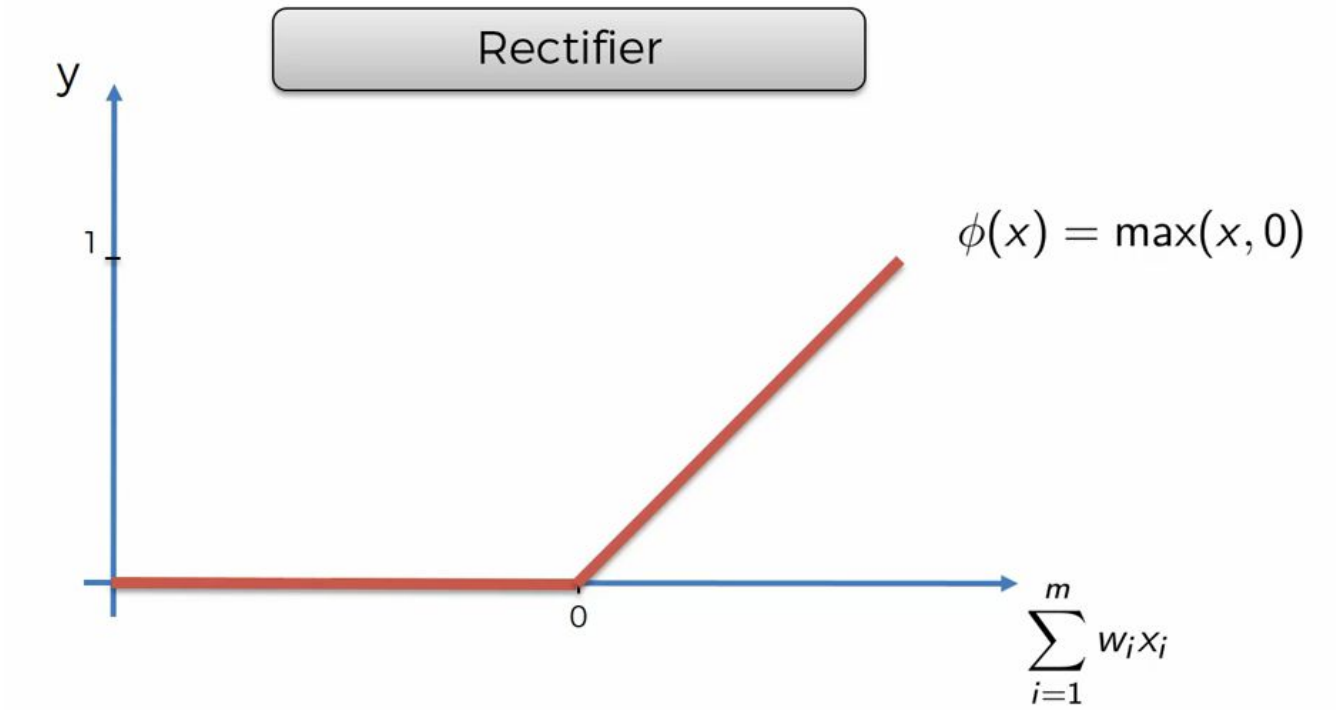
This function reduces extreme values or outliers in data without removing them.

It converts independent variables of near infinite range into simple probabilities between 0 and 1, and most of its output will be very close to 0 or 1.



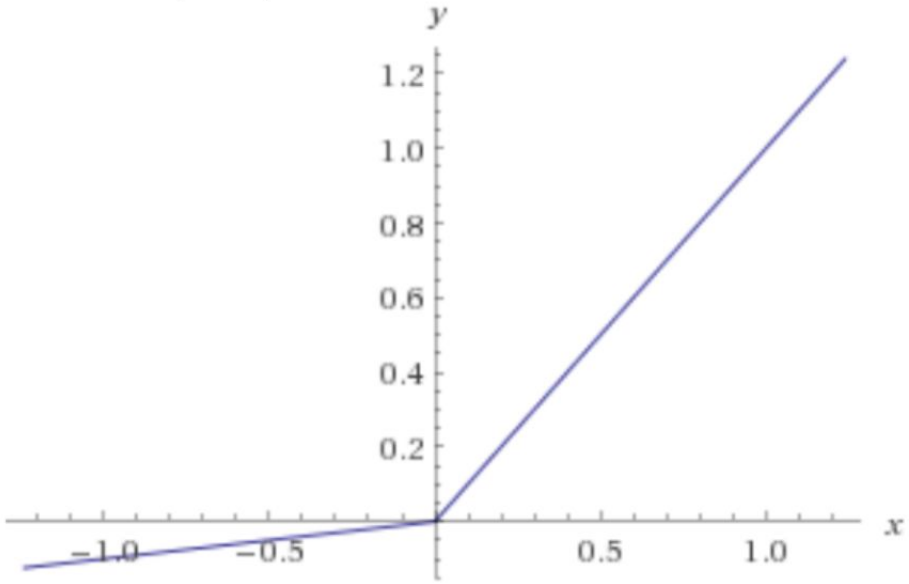
Rectifier (Relu) Function?

ReLU is the most widely used activation function while designing networks today. First things first, the ReLU function is non linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.



Leaky Relu Function?

Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for x<0, which made the neurons die for activations in that region. Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x.



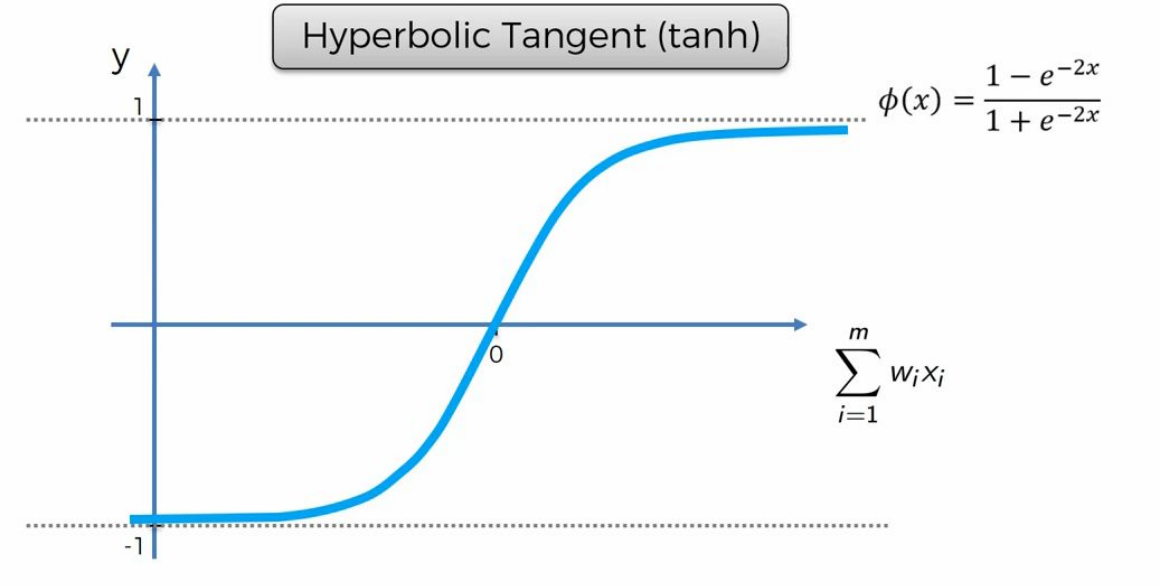
What we have done here is that we have simply replaced the horizontal line with a non-zero, non-horizontal line. Here a is a small value like 0.01 or so.

Hyperbolic tangent function?

Pronounced "*tanch*," tanh is a hyperbolic trigonometric function

The tangent represents a ratio between the opposite and adjacent sides of a right triangle, tanh represents the ratio of the hyperbolic sine to the hyperbolic cosine: tanh(*x*) = sinh(*x*) / cosh(*x*)

Unlike the Sigmoid function, the normalized range of tanh is –1 to 1 The advantage of tanh is that it can deal more easily with negative numbers

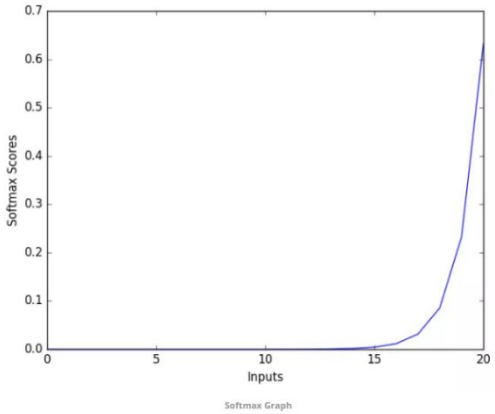


Softmax Function (for Multiple Classification)?

Softmax function calculates the probabilities distribution of the event over ‘n’ different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will be helpful for determining the target class for the given inputs.

The main advantage of using Softmax is the output probabilities range. The range will 0 to 1, and the sum of all the probabilities will be equal to one. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.

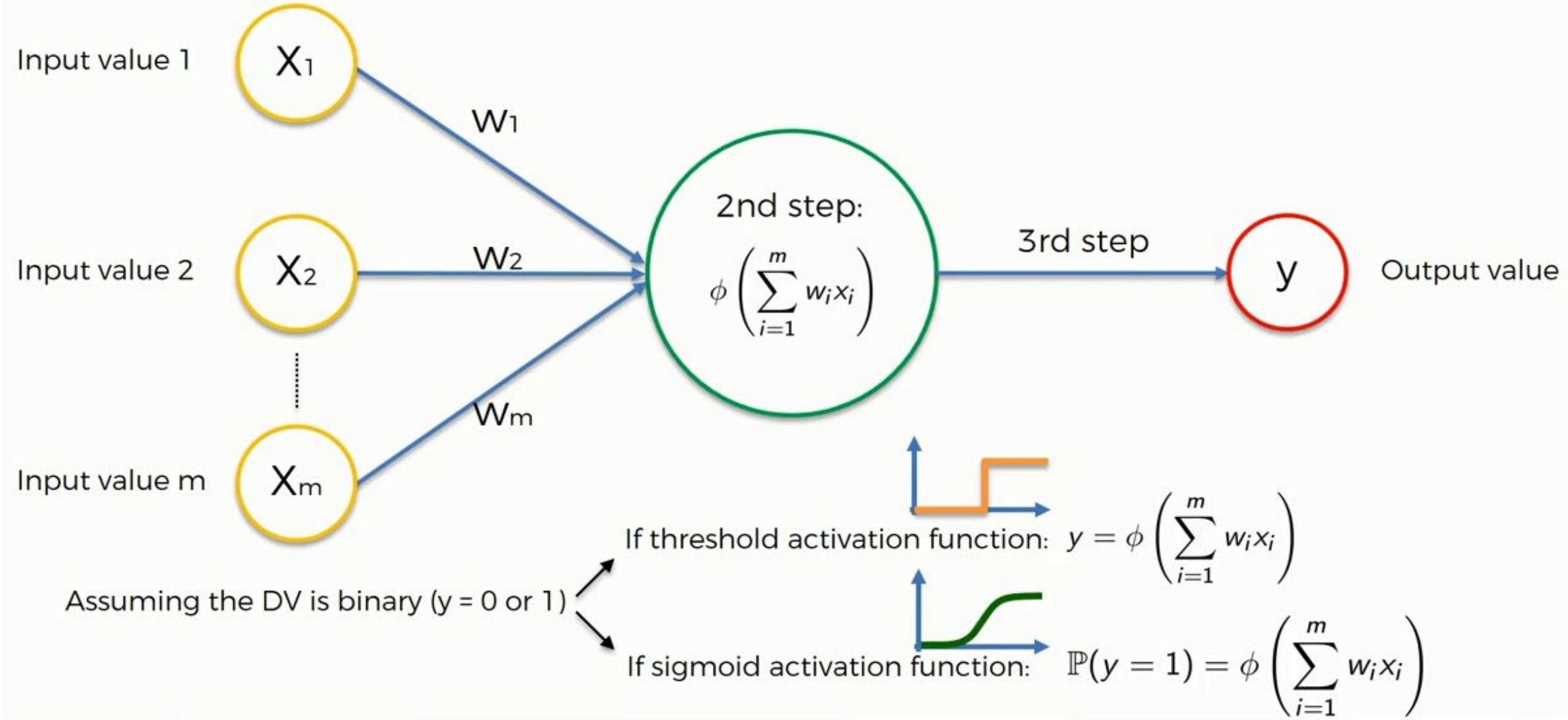
The formula computes the exponential (e-power) of the given input value and the sum of exponential values of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function.



Softmax Graph

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Activation Function Example

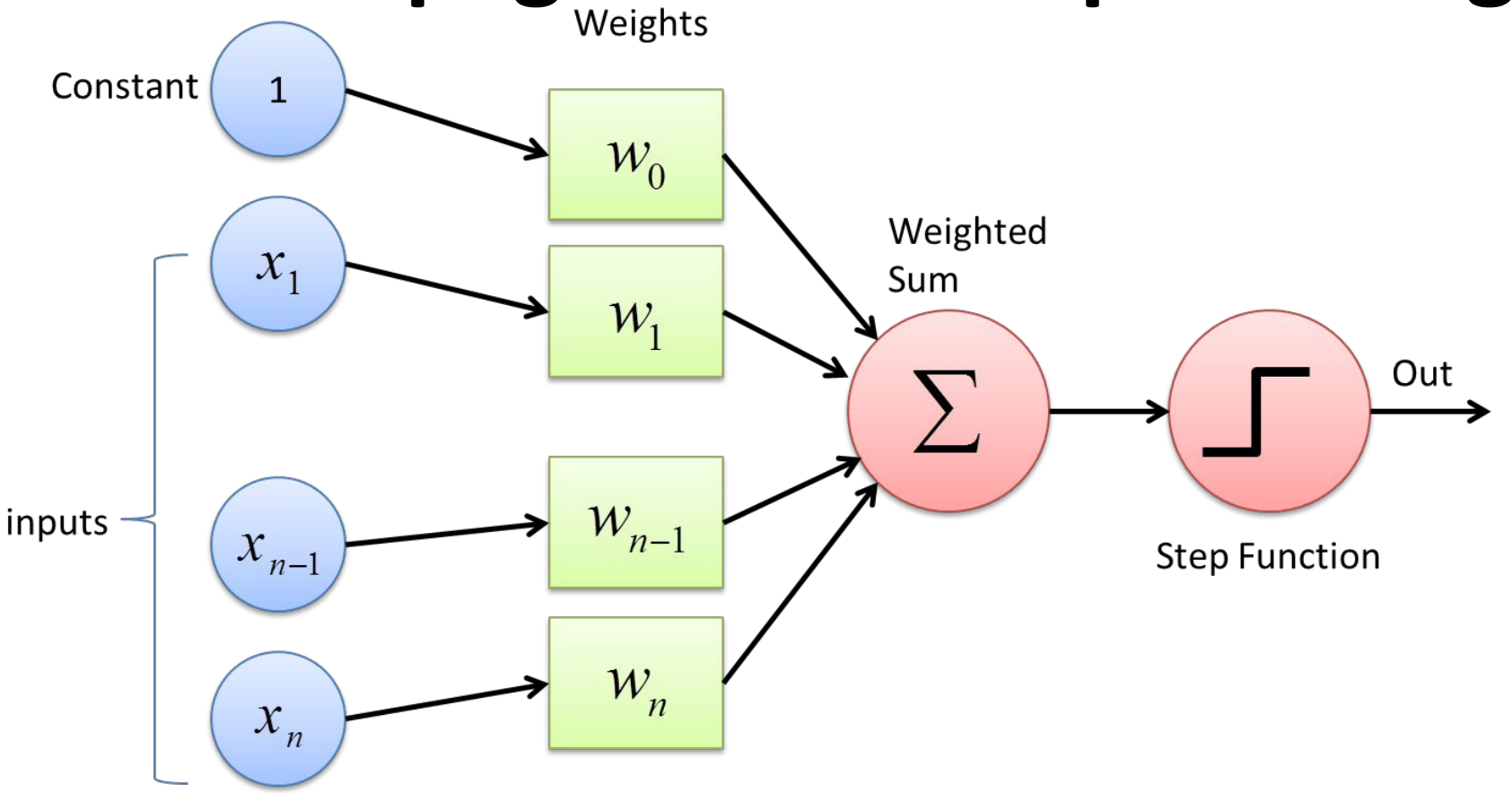


Thank You!

Deep Learning from Scratch

Theory + Practical

How Neural Network Work
and
Back Propagation in deep learning



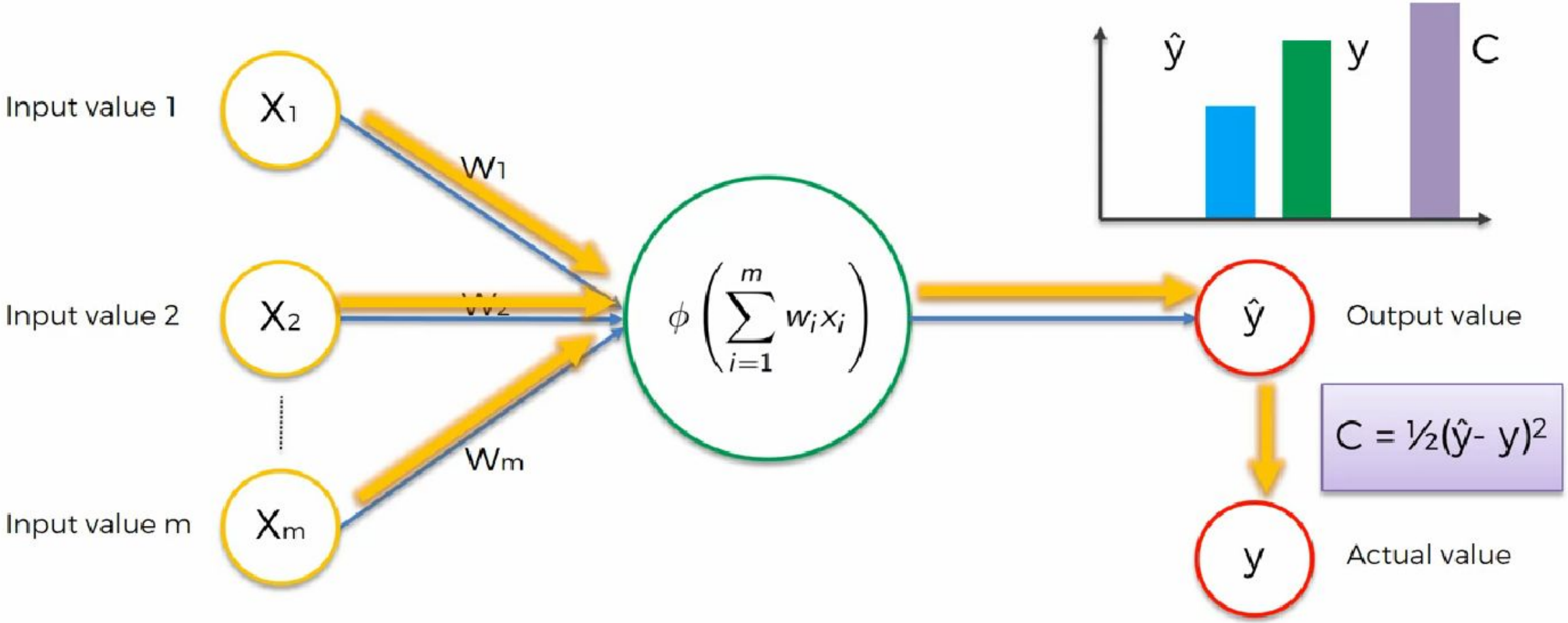
How Neural Network Work with many neurons

Back Propagation in deep learning

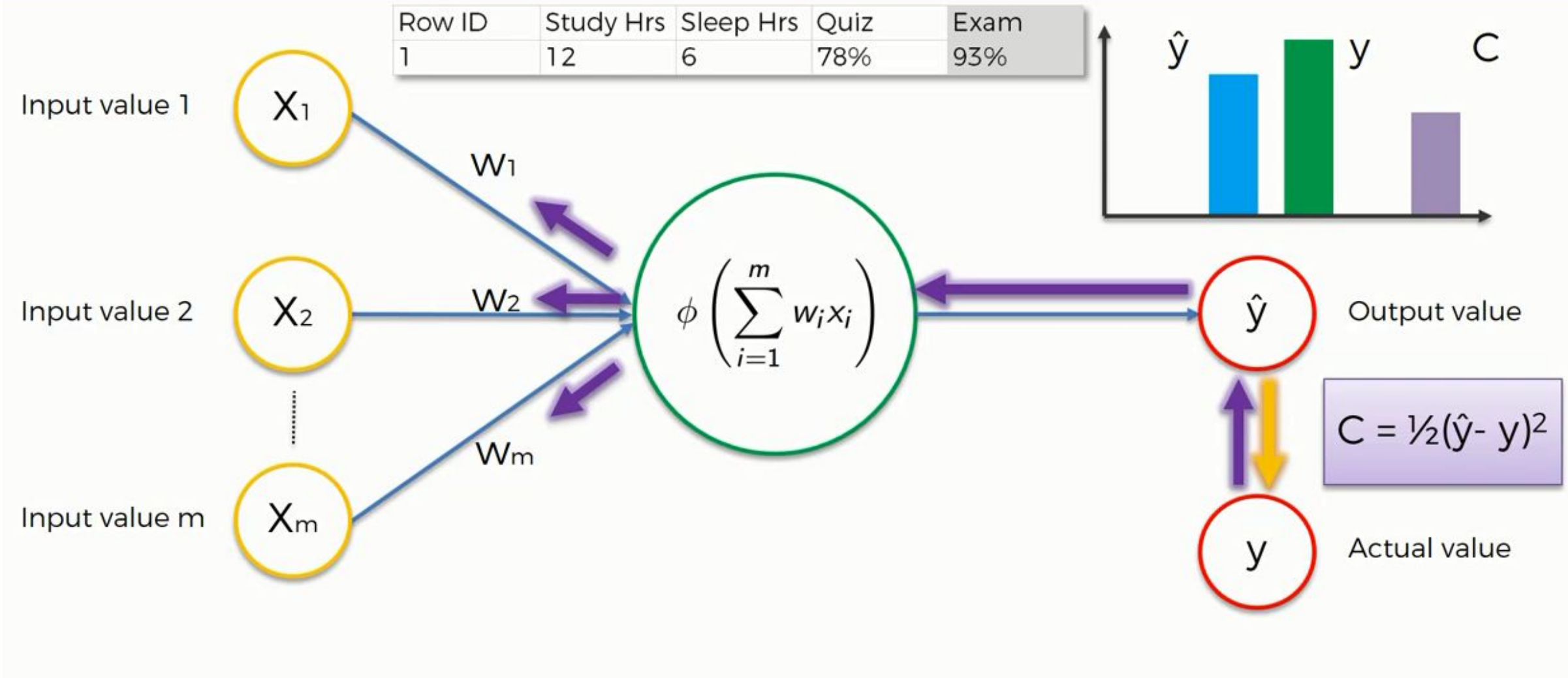
Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

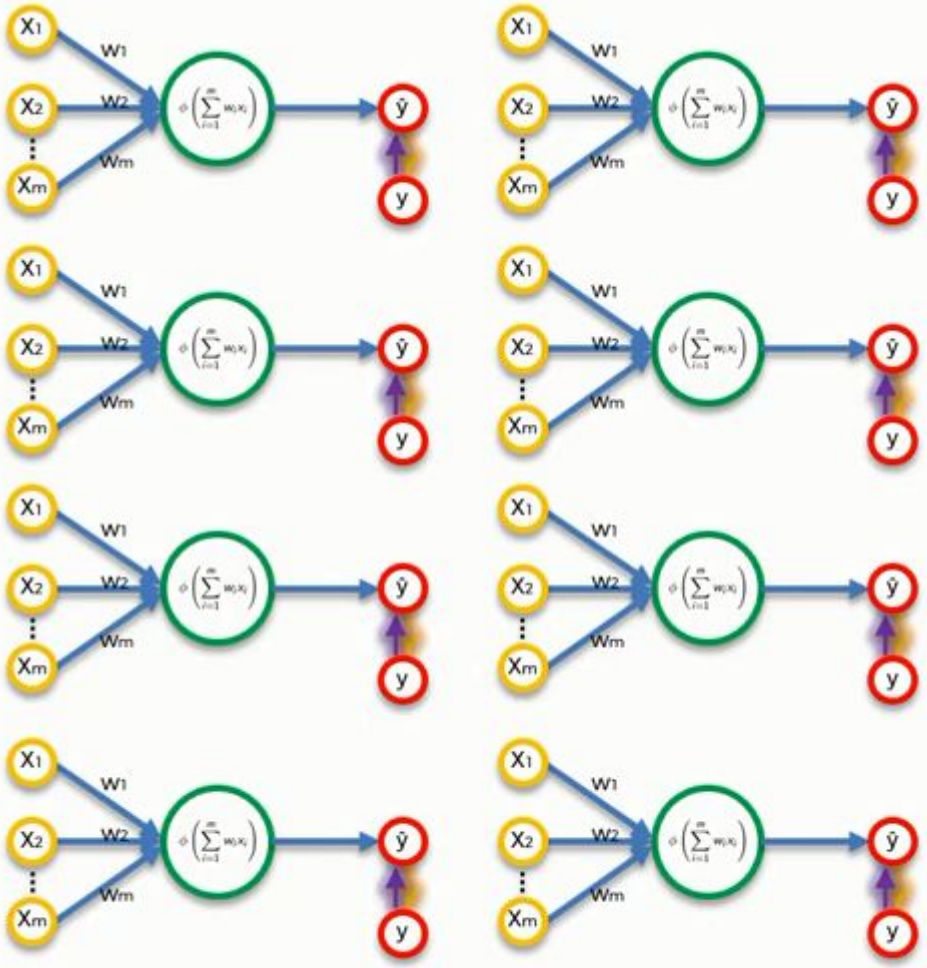
Back Propagation in deep learning



Back Propagation in deep learning

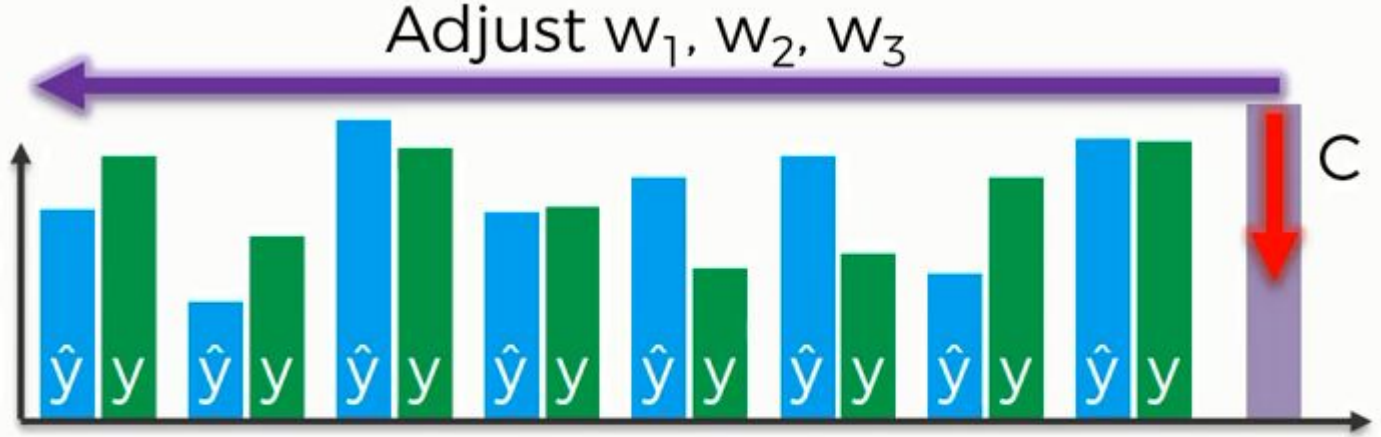


Back Propagation in deep learning (epoch)



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%

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



For further assistance
Visit Stack Exchange

https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications