UNIT - IV

Neural Networks (NN), Support Vector Machines (SVM), and K-nearest Neighbour: Fitting neural networks, Back propagation, Issues in training NN, SVM for classification, Reproducing Kernels, SVM for regression, K-nearest–Neighbour classifiers (Image Scene Classification)

Neural networks

- Neural networks, inspired by the structure and functioning of the human brain, are a class of machine learning models that excel at learning complex patterns and representations from data.
- They consist of interconnected nodes, known as neurons, organized into layers.
- Neural networks are a fundamental component of deep learning, a subfield of machine learning characterized by the use of deep architectures with multiple layers.

Key Concepts:

Neurons:

Neurons are the basic building blocks of a neural network. Each neuron processes input data and produces an output.

Layers:

Neural networks are organized into layers, including an input layer, one or more hidden layers, and an output layer. The input layer receives data, hidden layers process it, and the output layer produces the final result.

Weights and Biases:

Weights and biases are parameters that the neural network learns during training. They determine the strength of connections between neurons and affect the output.

Activation Functions:

Activation functions introduce non-linearities to the model, allowing it to learn complex relationships in the data. Common activation functions include sigmoid, tanh, and rectified linear unit (ReLU).

Feedforward and Backpropagation:
In the training phase, data is passed through the network in a feedforward manner to make predictions. The backpropagation algorithm is then used to adjust weights and biases based on the error, optimizing the model.

Loss Function:
The loss function measures the difference between the predicted output and the actual target. During training, the goal is to minimize this loss, guiding the network to make more accurate predictions.

Deep Learning:
Deep neural networks have multiple hidden layers, enabling them to learn hierarchical representations of data. This depth allows them to handle intricate features and patterns.

Types of Neural Networks:
Different architectures cater to various tasks, including:
Convolutional Neural Networks (CNNs) for image-related tasks.
Recurrent Neural Networks (RNNs) for sequential data.
Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks for improved handling of long-range dependencies.

Applications:

•**Image and Speech Recognition:** Neural networks have achieved significant success in tasks such as image classification, object detection, and speech recognition.

•**Natural Language Processing (NLP):** They are widely used in language-related tasks, including sentiment analysis, machine translation, and text generation.

•**Medical Diagnostics:** Neural networks contribute to medical image analysis, disease diagnosis, and predicting patient outcomes.

•**Autonomous Vehicles:** In the field of autonomous driving, neural networks play a crucial role in tasks like object detection, path planning, and decision-making.

•**Financial Forecasting:** Neural networks are applied to predict stock prices, analyze market trends, and model financial data.

Example: Virtual Personal Assistant's Speech Recognition

Imagine using a virtual personal assistant, such as Apple's Siri, Amazon's Alexa, or Google Assistant. These virtual assistants employ neural networks, particularly recurrent neural networks (RNNs) and convolutional neural networks (CNNs), for speech recognition.

1.Data Input:

You activate the virtual assistant by saying a command, such as "Hey Siri" or "Alexa."

2.Neural Network Processing:

The neural network within the virtual assistant's system processes the incoming audio data in real-time. The network has been trained on vast datasets containing various spoken phrases and words.

3.Feature Extraction:

The neural network extracts relevant features from the audio data, capturing the nuances of your voice, accent, and speech patterns. Recurrent neural networks are particularly effective in handling sequential data like spoken language.

4.Pattern Recognition:

The neural network recognizes patterns and converts the audio input into a sequence of words or commands. This involves complex computations to understand context, syntax, and semantics.

5.Command Execution:

Based on the recognized speech, the virtual assistant executes the corresponding command. For example, if you say, "What's the weather today?" the neural network interprets the query and triggers the appropriate response by fetching real-time weather information.

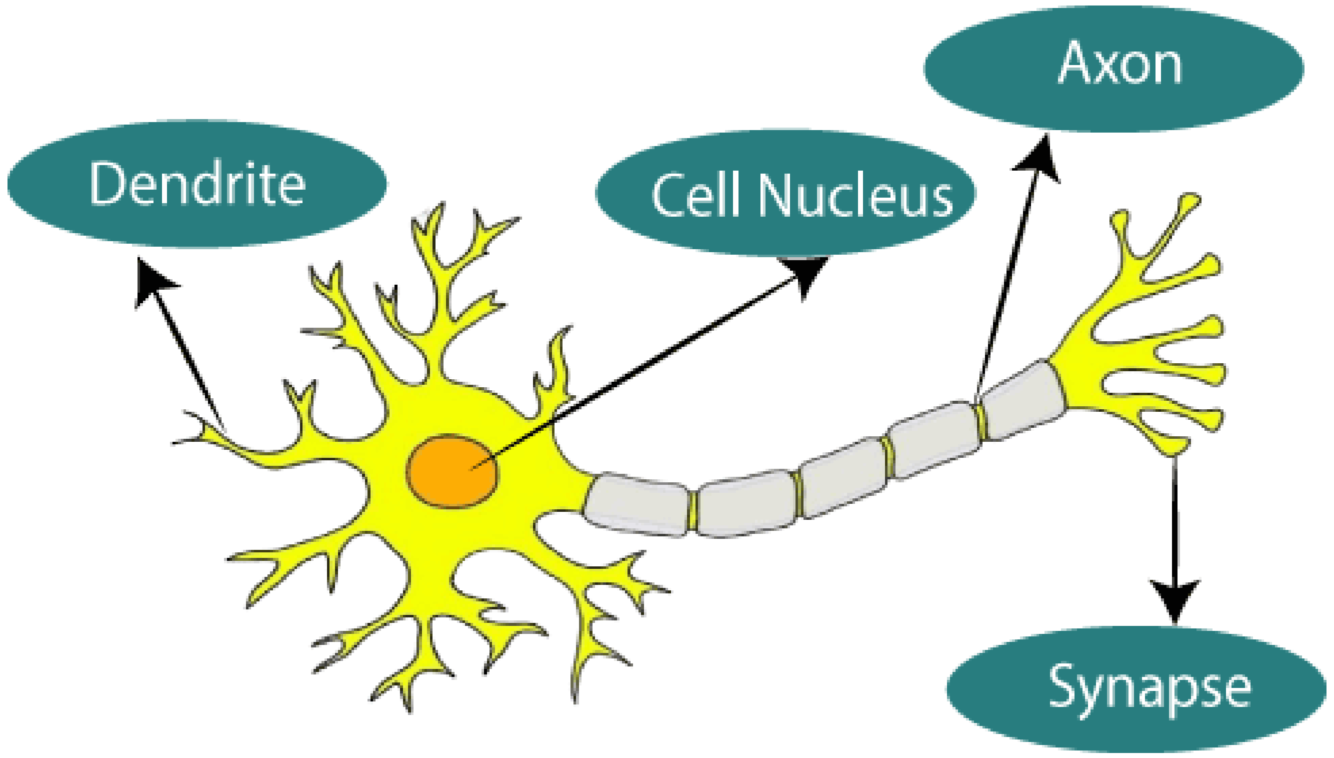
6.Continuous Learning:

Neural networks in virtual assistants are often designed for continuous learning. As users interact more, the neural network adapts to individual speech patterns, accents, and preferences, enhancing its performance over time.

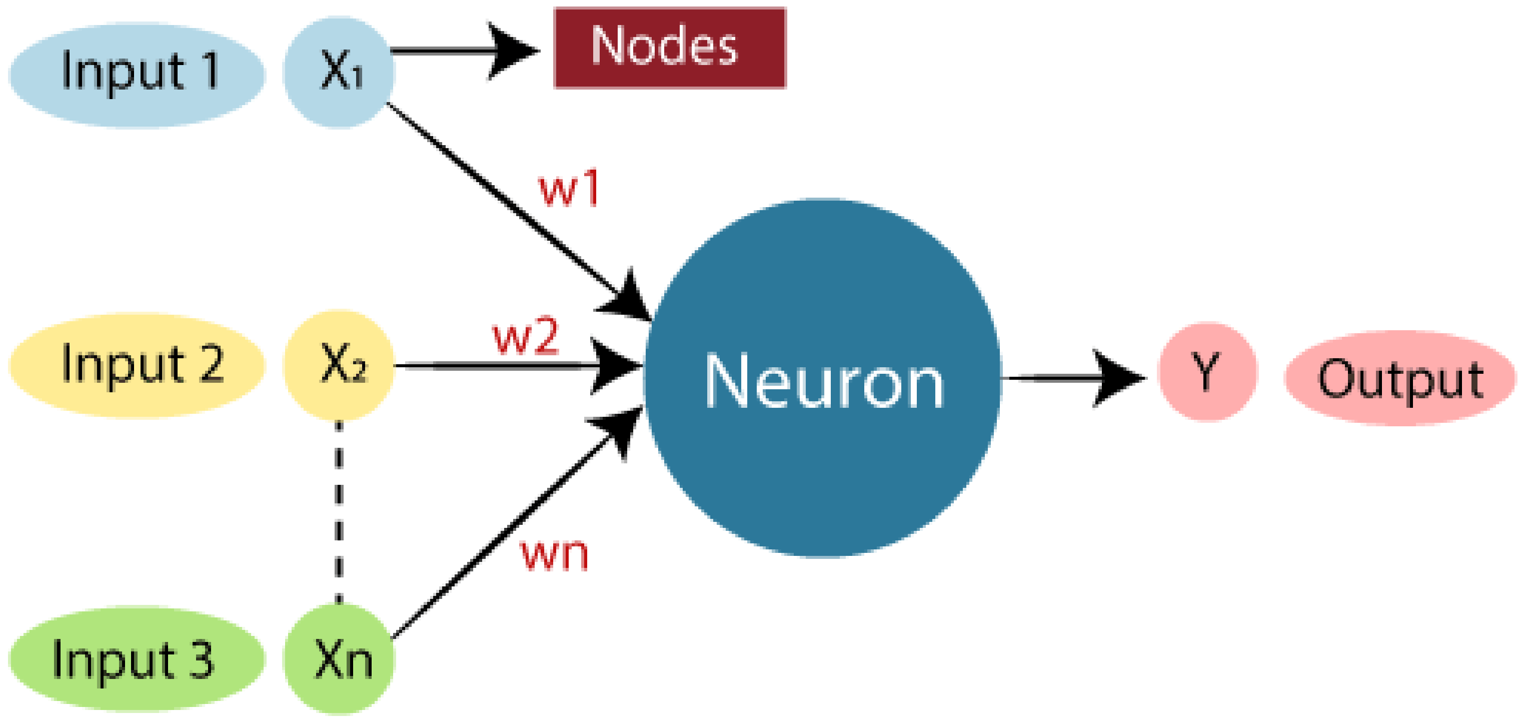
This example illustrates how neural networks in speech recognition applications have become integral to our daily lives. The technology enables natural and seamless interactions with devices, showcasing the power of neural networks in understanding and processing complex patterns in real-world scenarios.

What is Artificial Neural Network?

The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.
The given figure illustrates the typical diagram of Biological Neural Network.



The typical Artificial Neural Network looks something like the given figure.

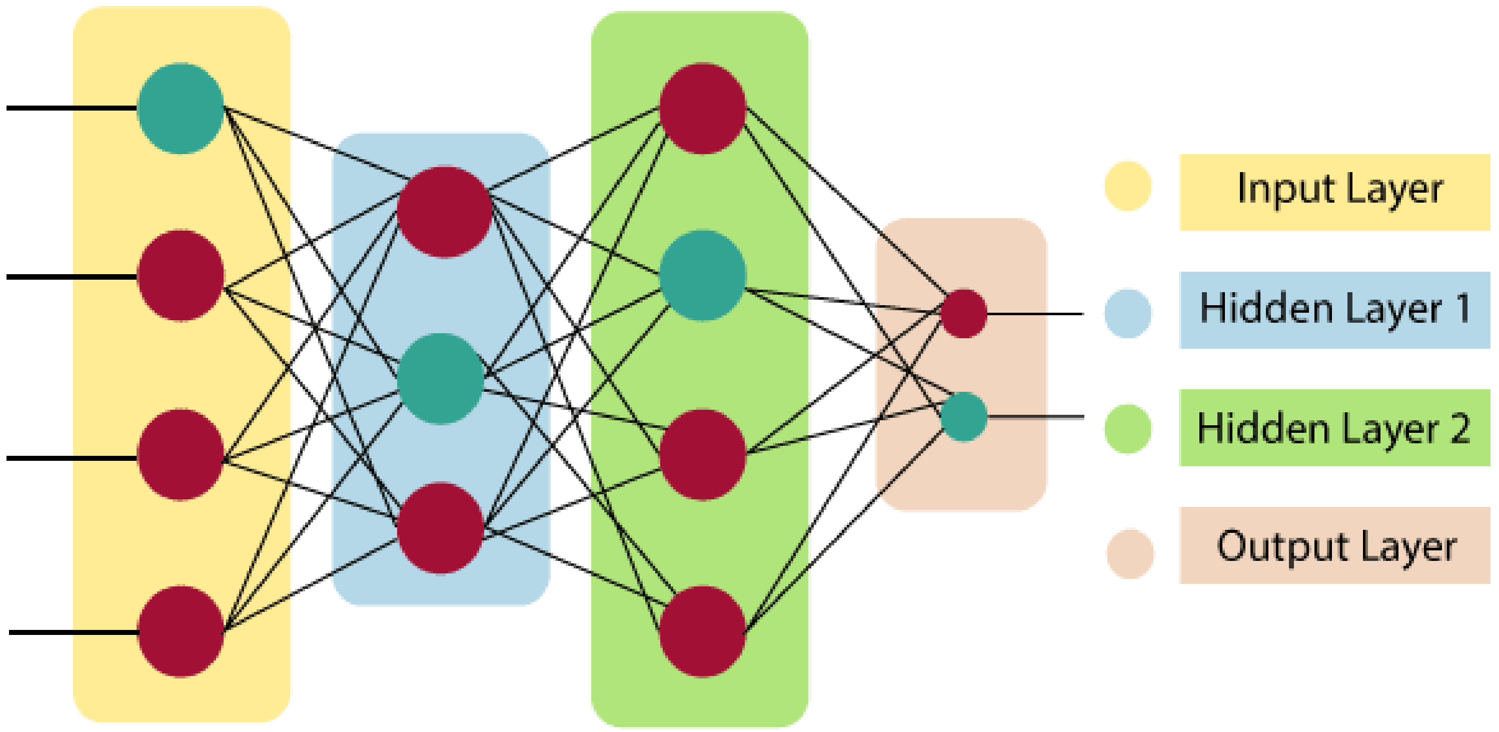


Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

The architecture of an artificial neural network:

- ❑ To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of.
- ❑ In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers.
- ❑ Lets us look at various types of layers available in an artificial neural network.
- ❑ Artificial Neural Network primarily consists of three layers:



Input Layer:
As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:
The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.
The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\boxed{\sum_{i=1}^{n} Wi * Xi + b}$$

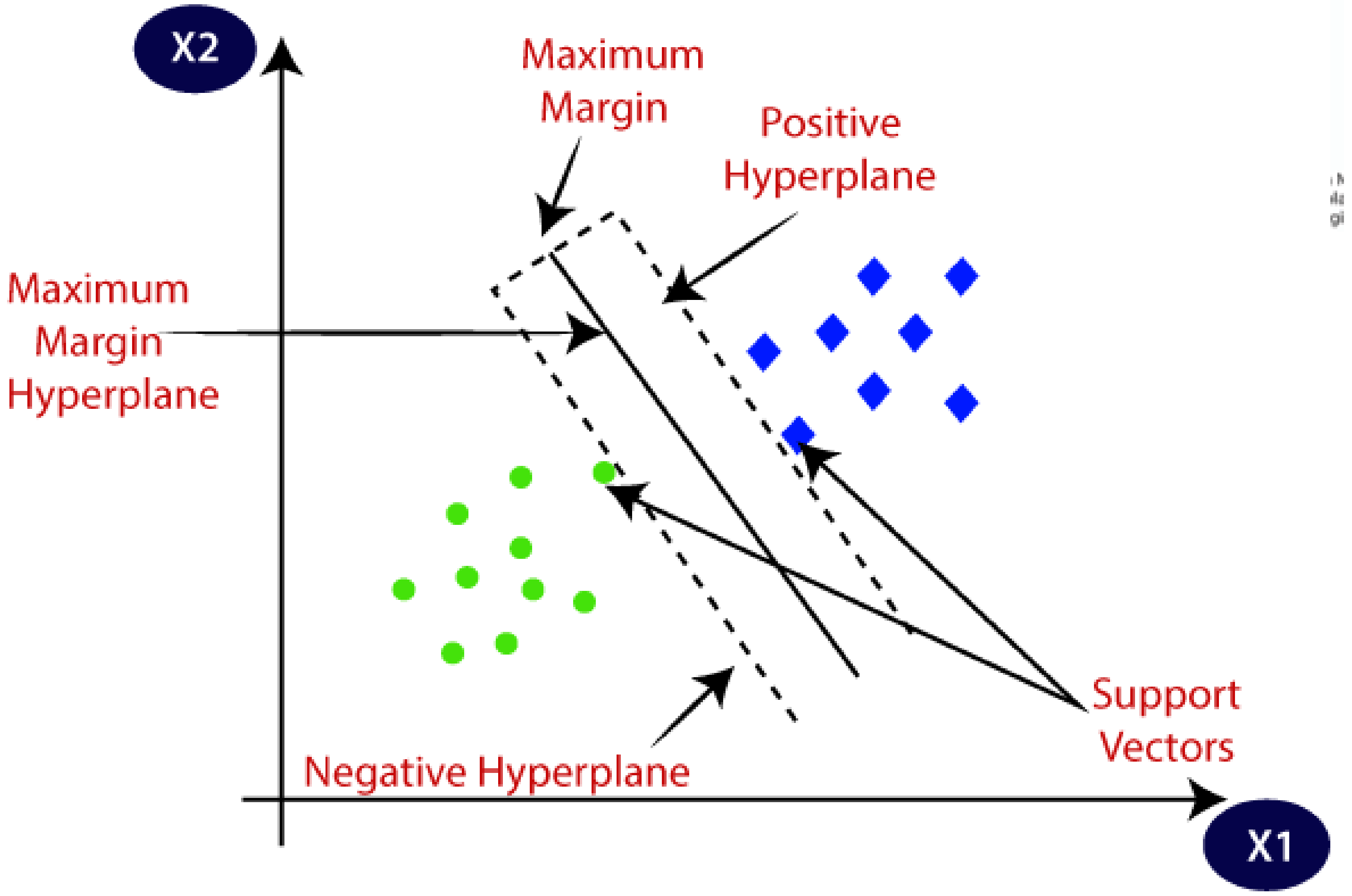
Support Vector Machine Algorithm:

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes

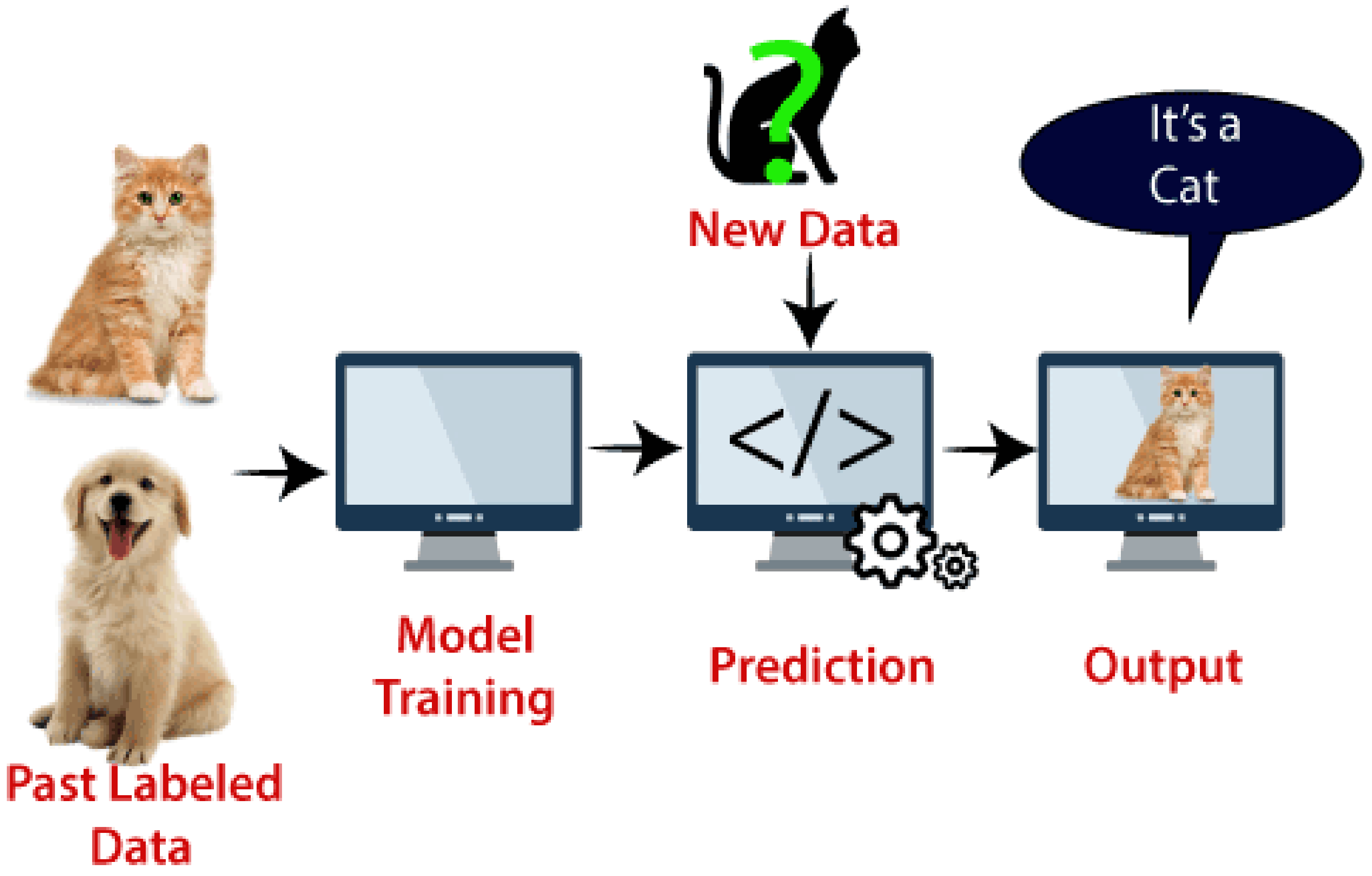
- so that we can easily put the new data point in the correct category in the future.This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane.
These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.
Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



- **Example:** SVM can be understood with the example that we have used in the KNN classifier.
- Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm.
- We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature.
- So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog.

On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



❖ SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM
SVM can be of two types:

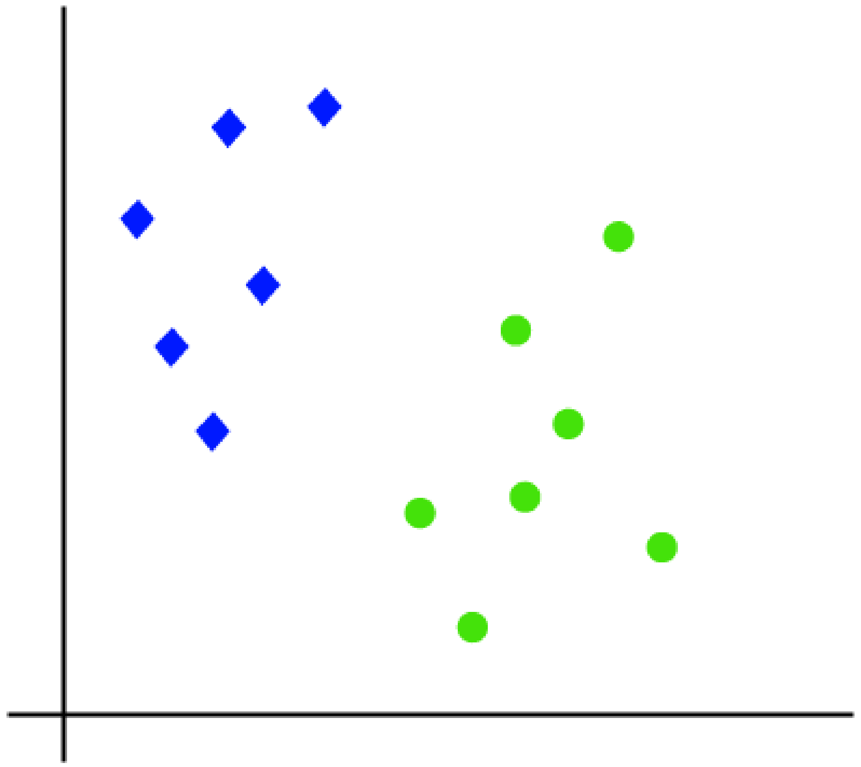
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

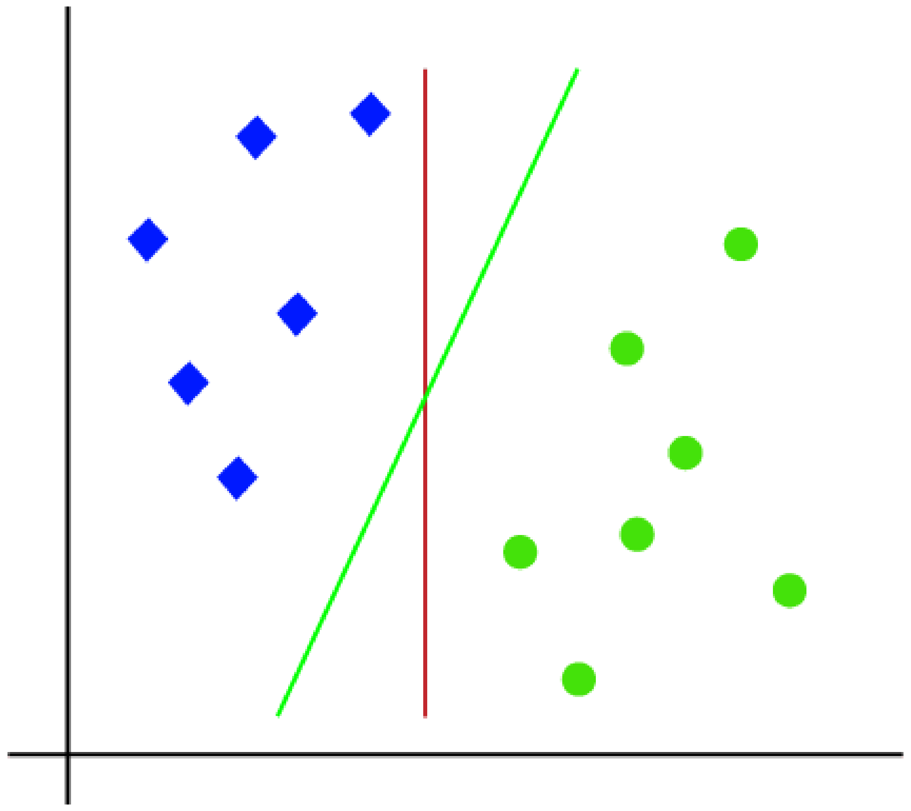
How does SVM works?

Linear SVM:

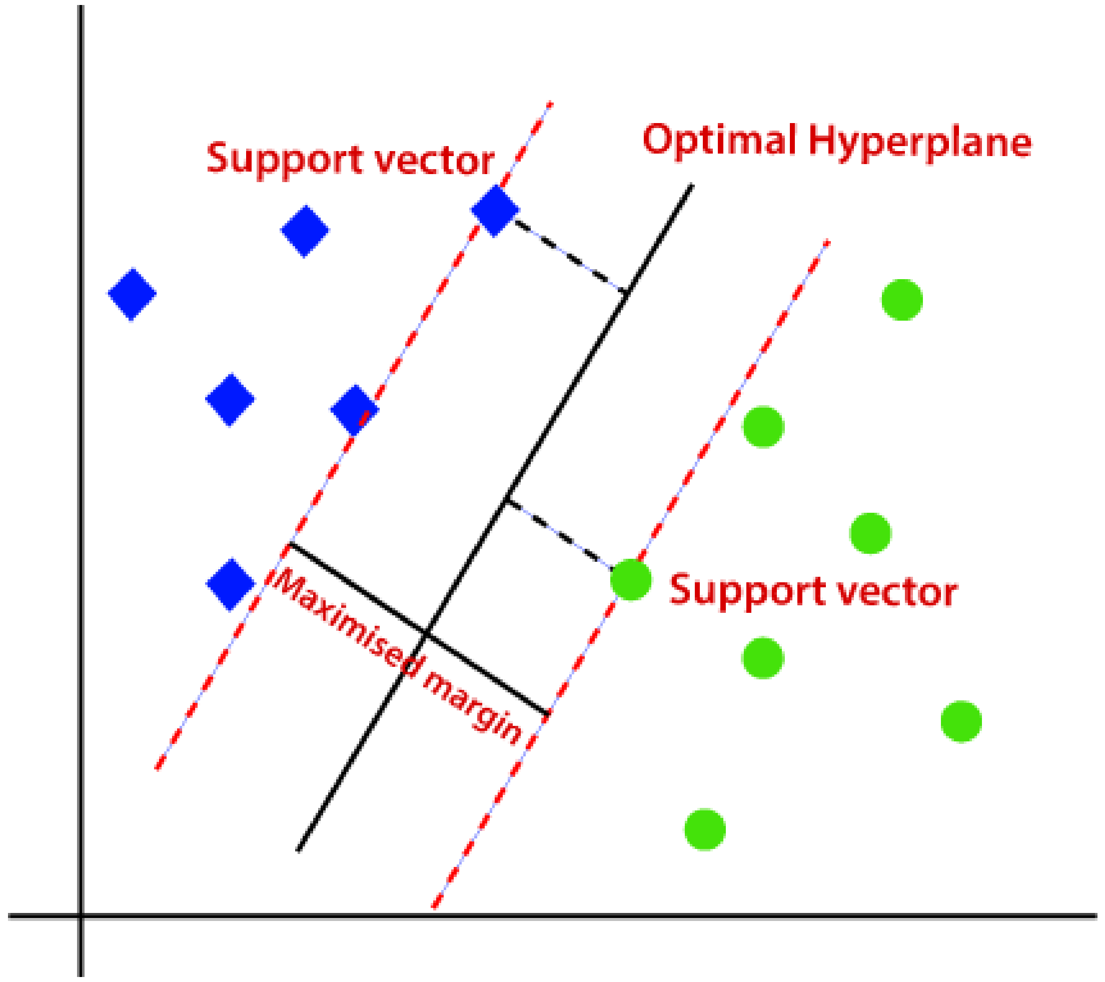
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

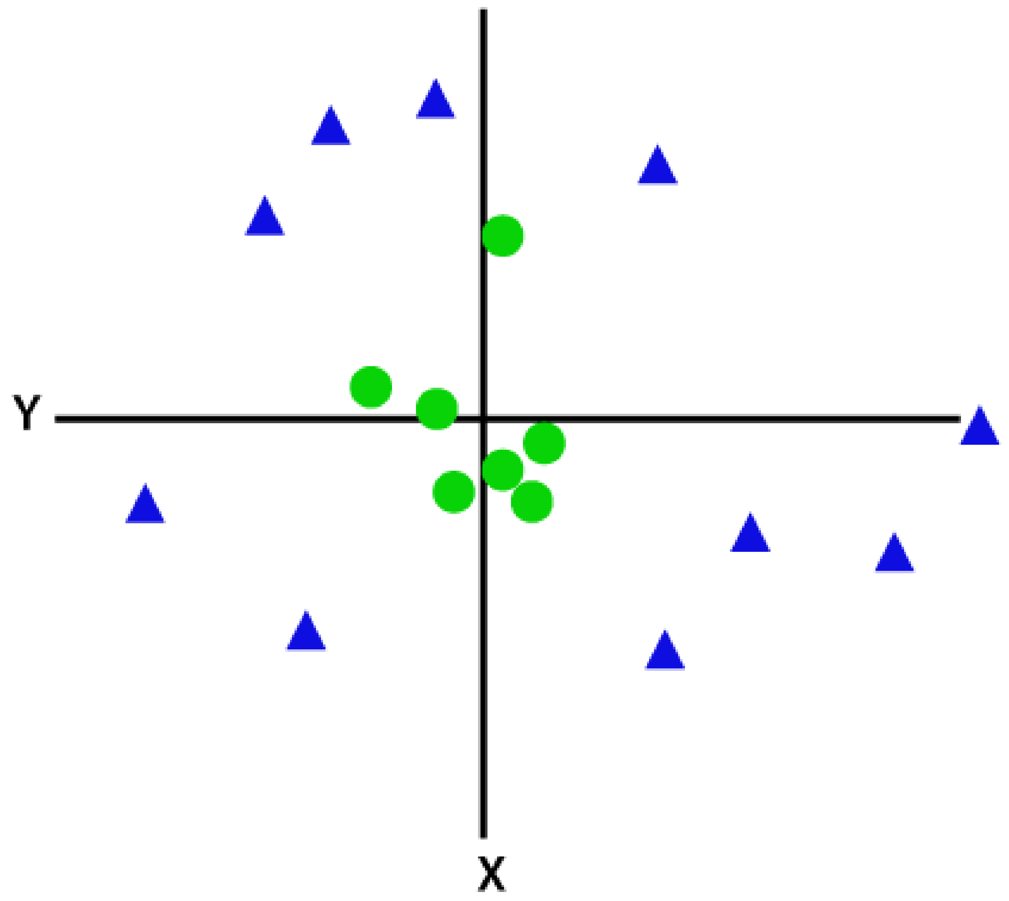


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM:

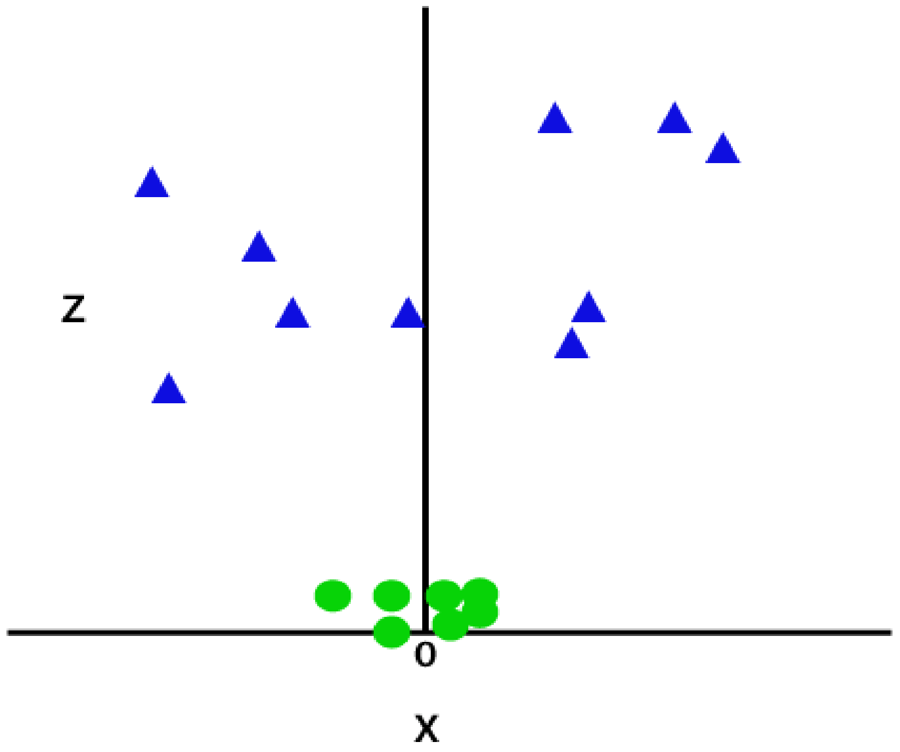
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



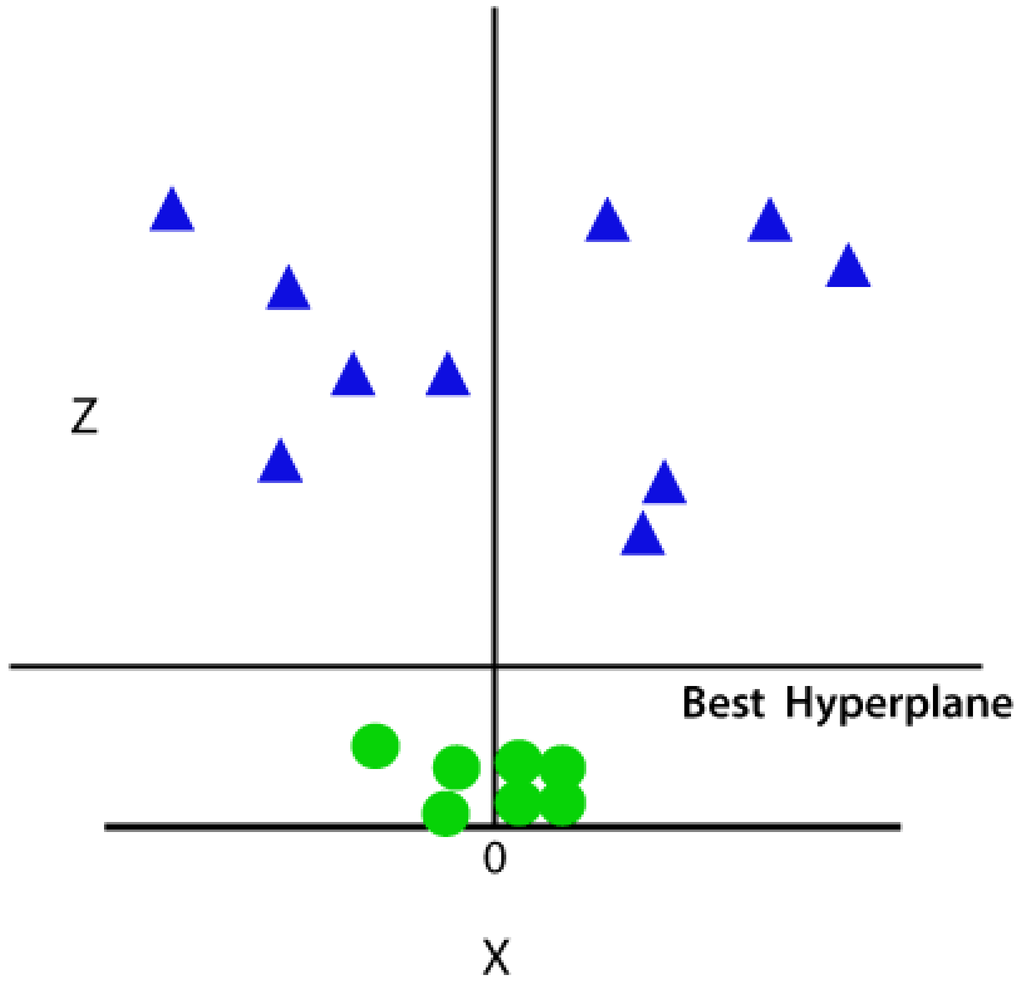
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z=x2 +y2$$

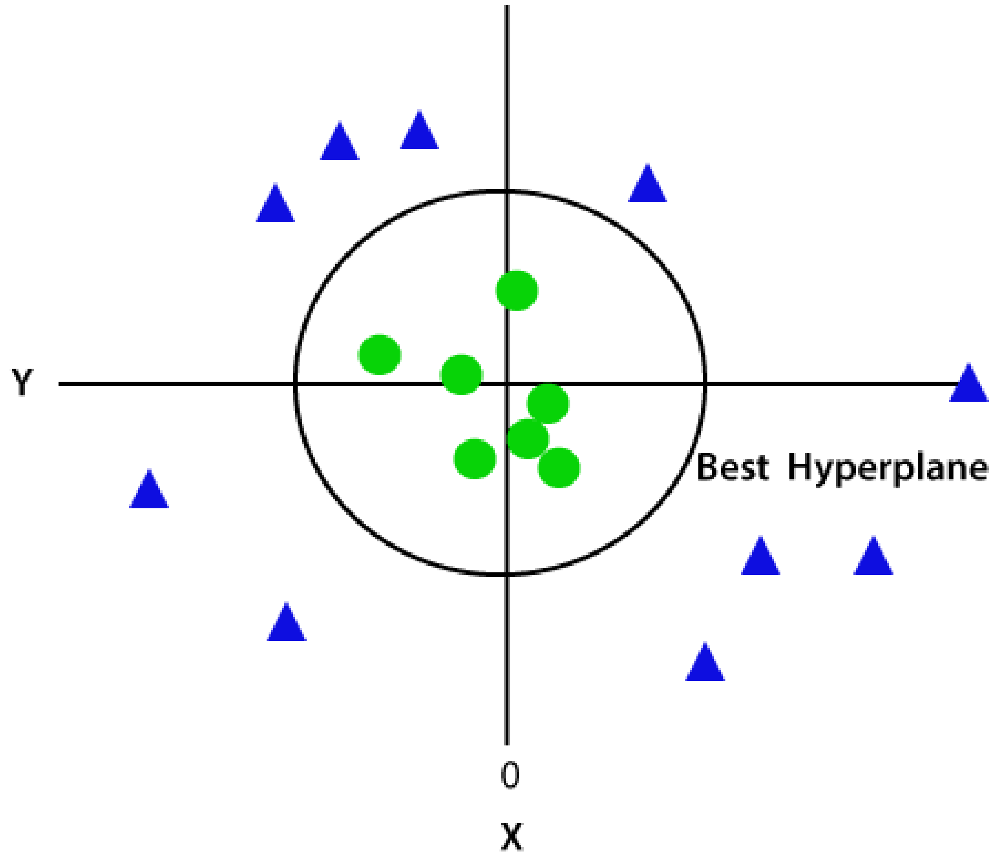
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

K-nearest Neighbour:

•K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

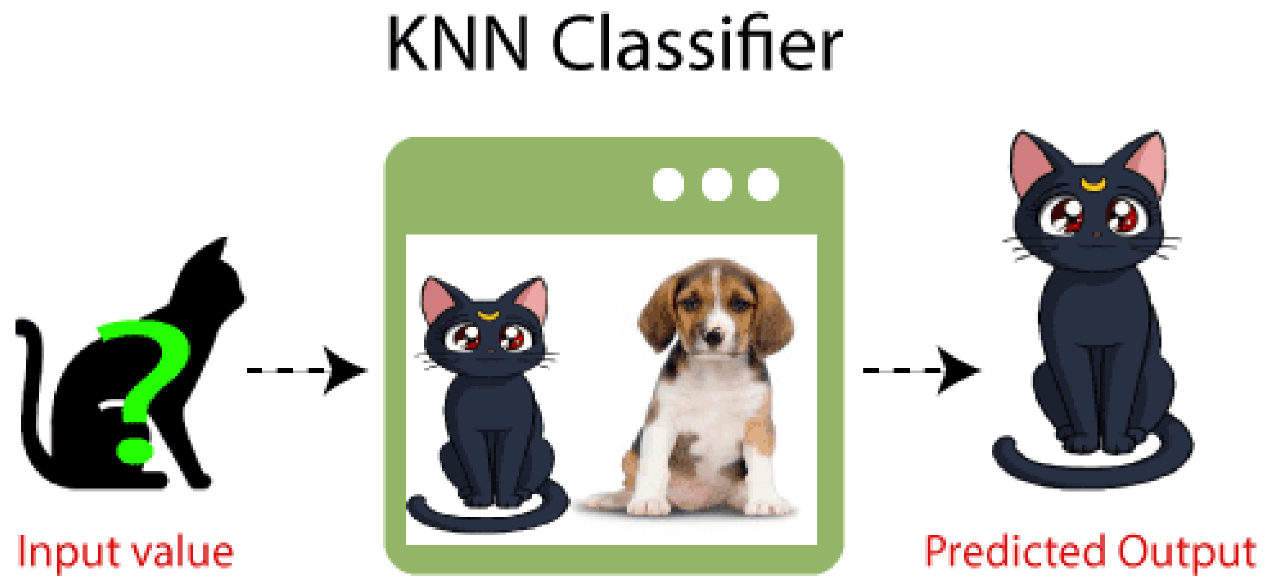
•K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

•K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

•K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

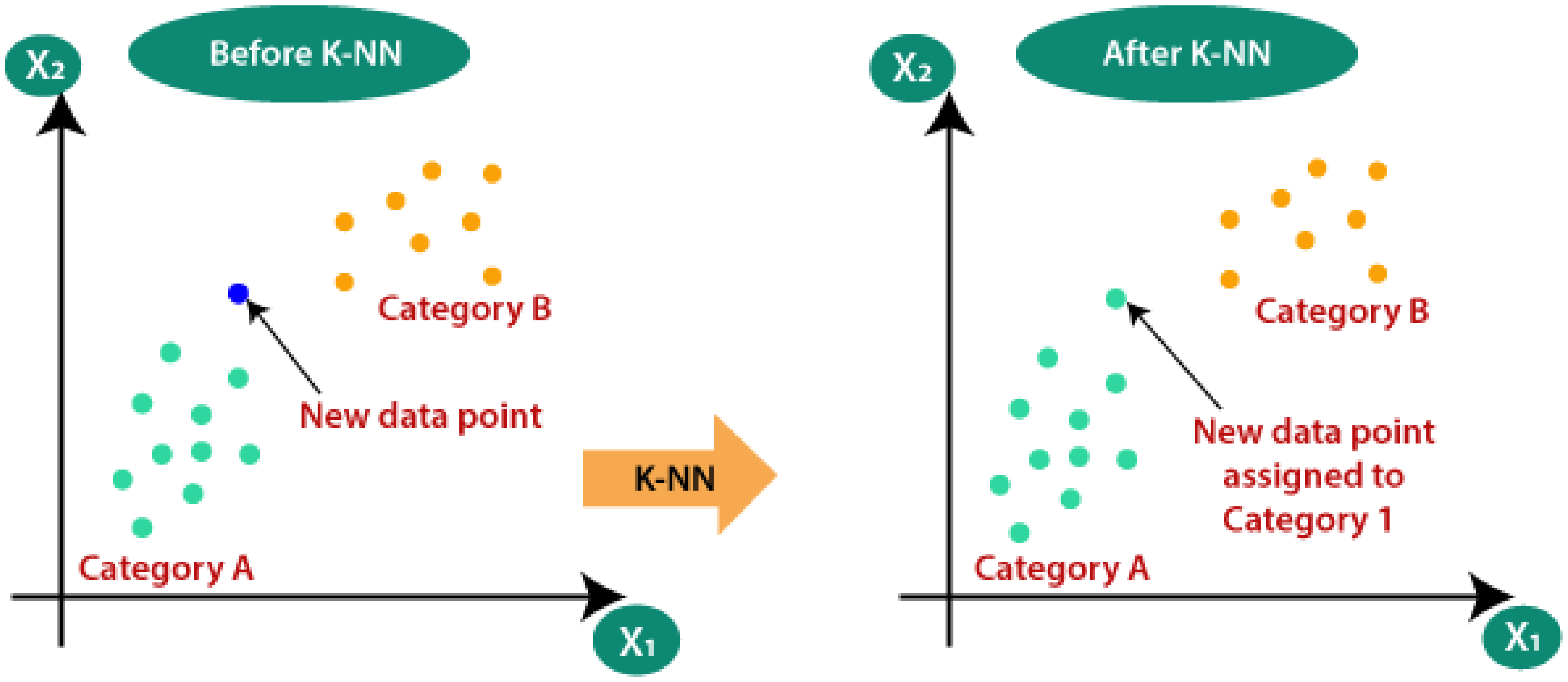
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

- **Example:**
- Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog.
- So for this identification, we can use the KNN algorithm, as it works on a similarity measure.
- Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



Fitting neural networks

- Fitting neural networks, also known as training or training a neural network, is the process of adjusting the weights and biases of the network to minimize the difference between the predicted outputs and the actual outputs for a given set of input data.
- This process involves several key steps:

1. **Define the Architecture:**

Specify the number of layers in the neural network.

Determine the number of neurons in each layer.

Choose an activation function for each neuron.

2. **Initialize Weights and Biases:**

Assign initial weights and biases to the connections between neurons. Common initialization methods include random initialization.

3. Forward propagation:
Pass input data through the network to compute the predicted output.
Apply activation functions at each neuron in the hidden layers.

4. Compute Loss:
Calculate the difference between the predicted output and the actual output using a loss function.

5. Backpropagation:
Propagate the error backward through the network.
Update the weights and biases using optimization algorithms like gradient descent to minimize the loss.

6. Optimization:
Choose an optimization algorithm that determines how the weights and biases are updated during backpropagation.

7. Repeat:
Iterate through steps 3 to 6 for multiple epochs (passes through the entire dataset) until the model converges and the loss is minimized.

8. Validation and Testing:

Evaluate the model on a separate dataset (validation set) to ensure it generalizes well to unseen data.

Test the model on a completely independent dataset to assess its performance.

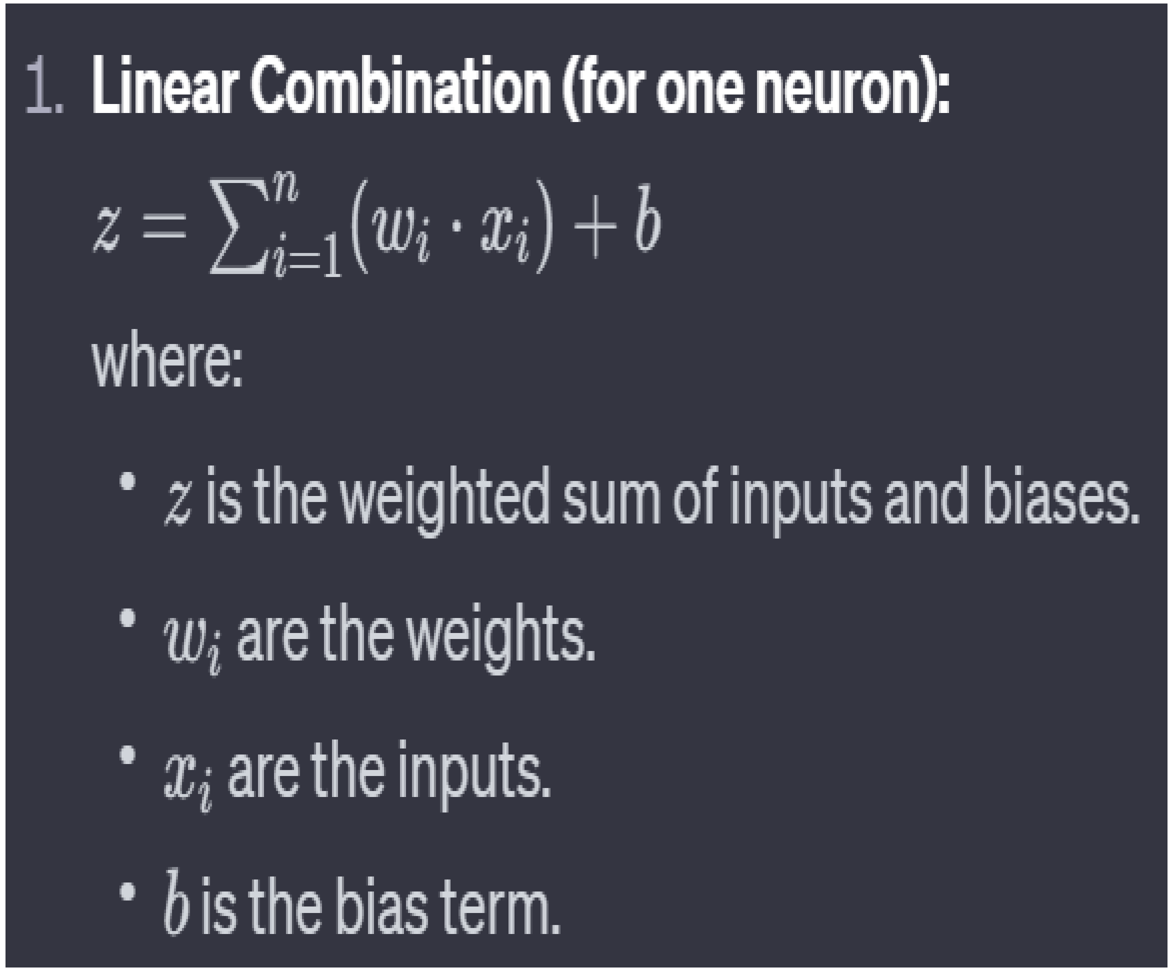
9. Hyperparameter Tuning:

Adjust hyperparameters such as learning rate, batch size, and the number of hidden layers/neurons to optimize the model's performance.

10. Regularization:

Apply regularization techniques, such as dropout or L2 regularization, to prevent overfitting.

The process of fitting neural networks involves various formulas and computations. Here are some key formulas and concepts involved in training a neural network:

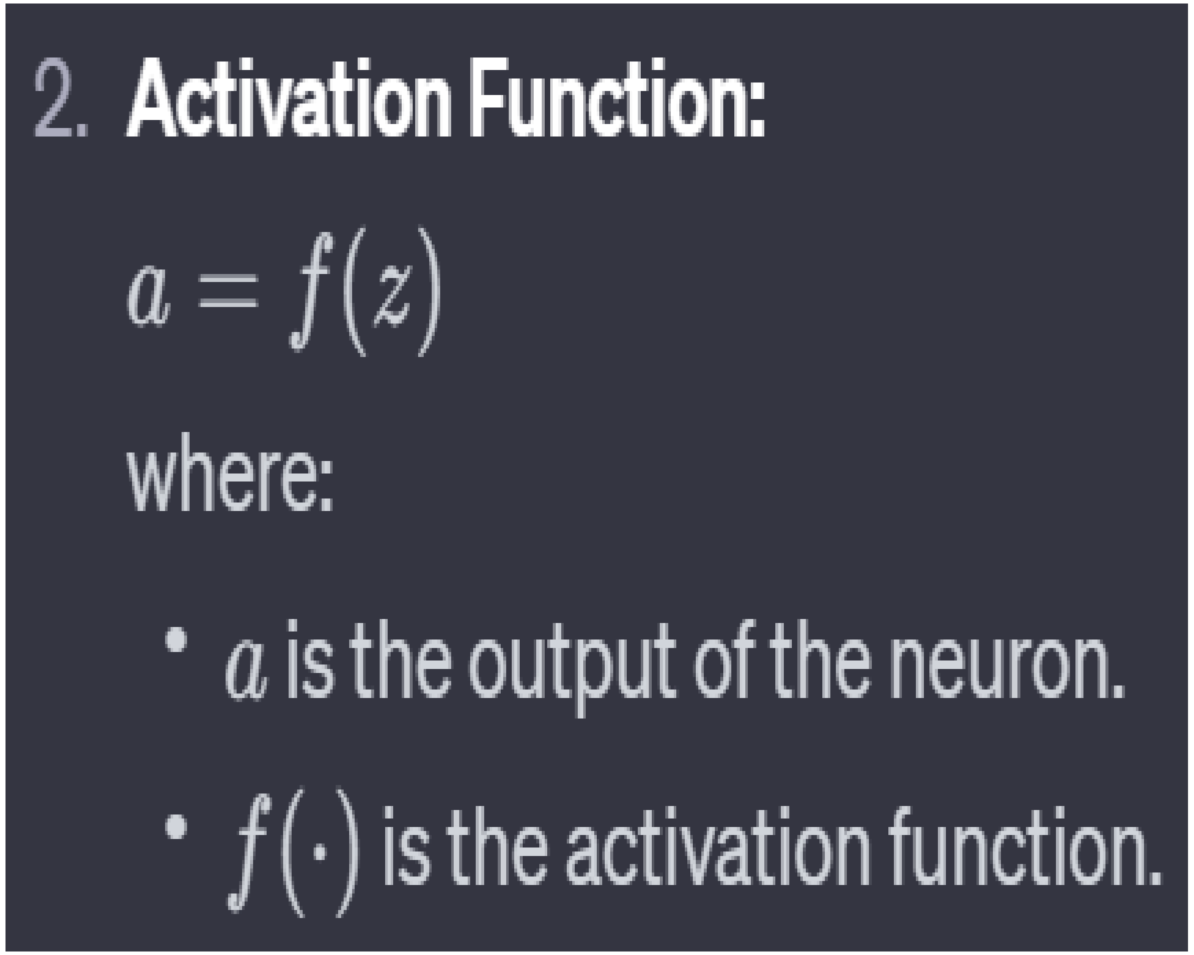


1. **Linear Combination (for one neuron):**

   $z = \sum_{i=1}^{n}(w_i \cdot x_i) + b$

   where:

   - z is the weighted sum of inputs and biases.
   - w_i are the weights.
   - x_i are the inputs.
   - b is the bias term.

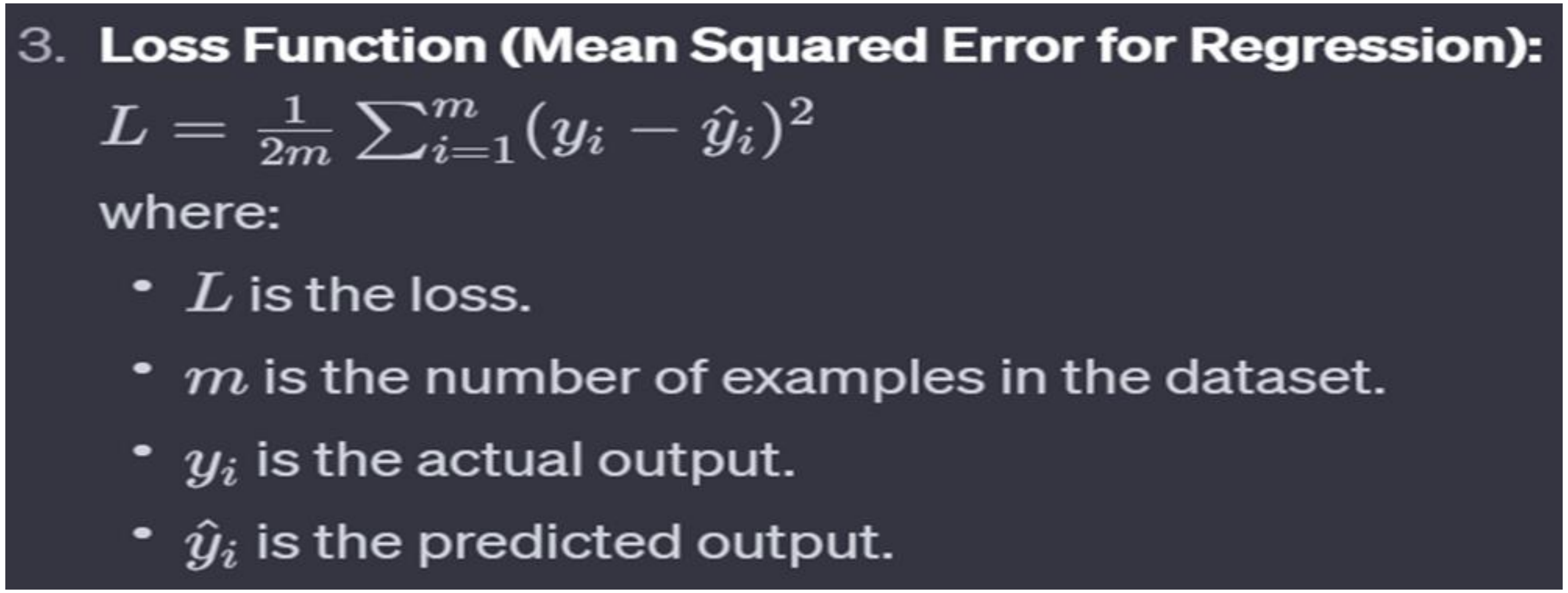


2. **Activation Function:**

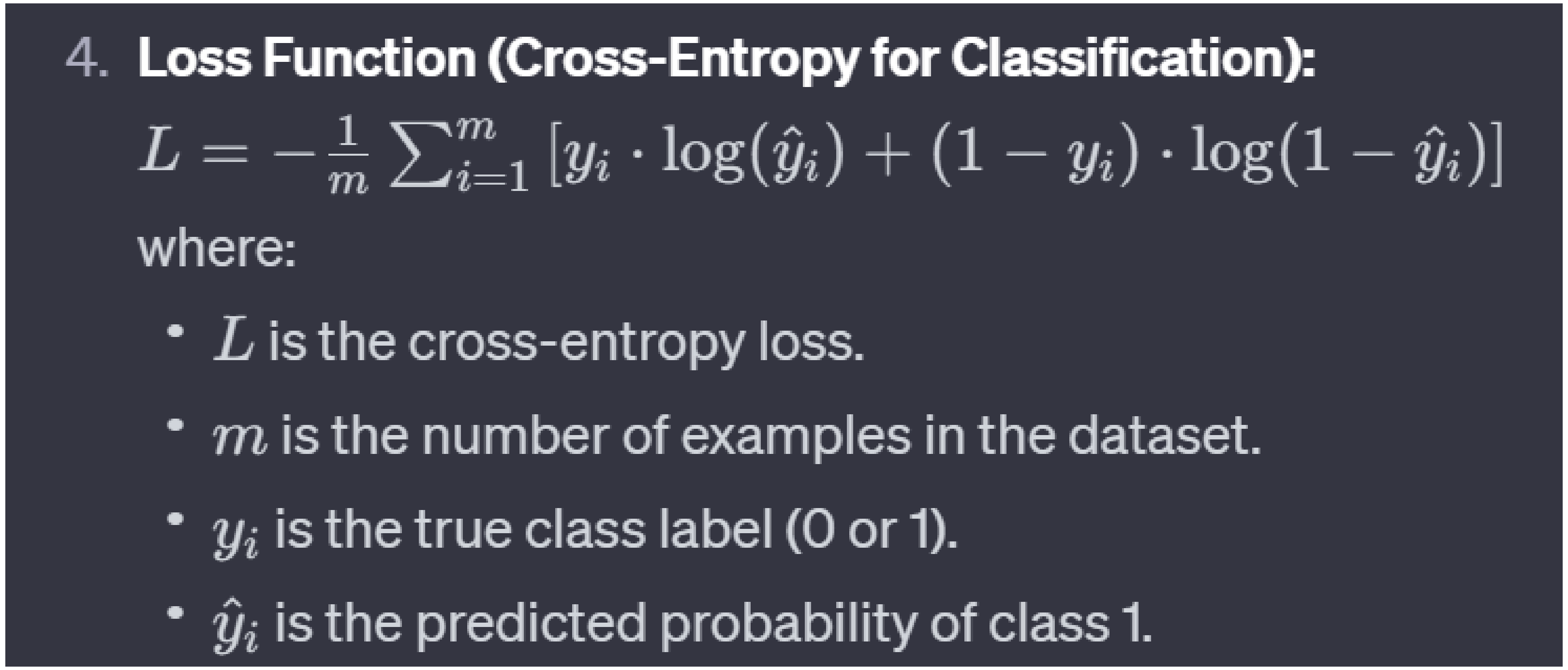
   $a = f(z)$

   where:

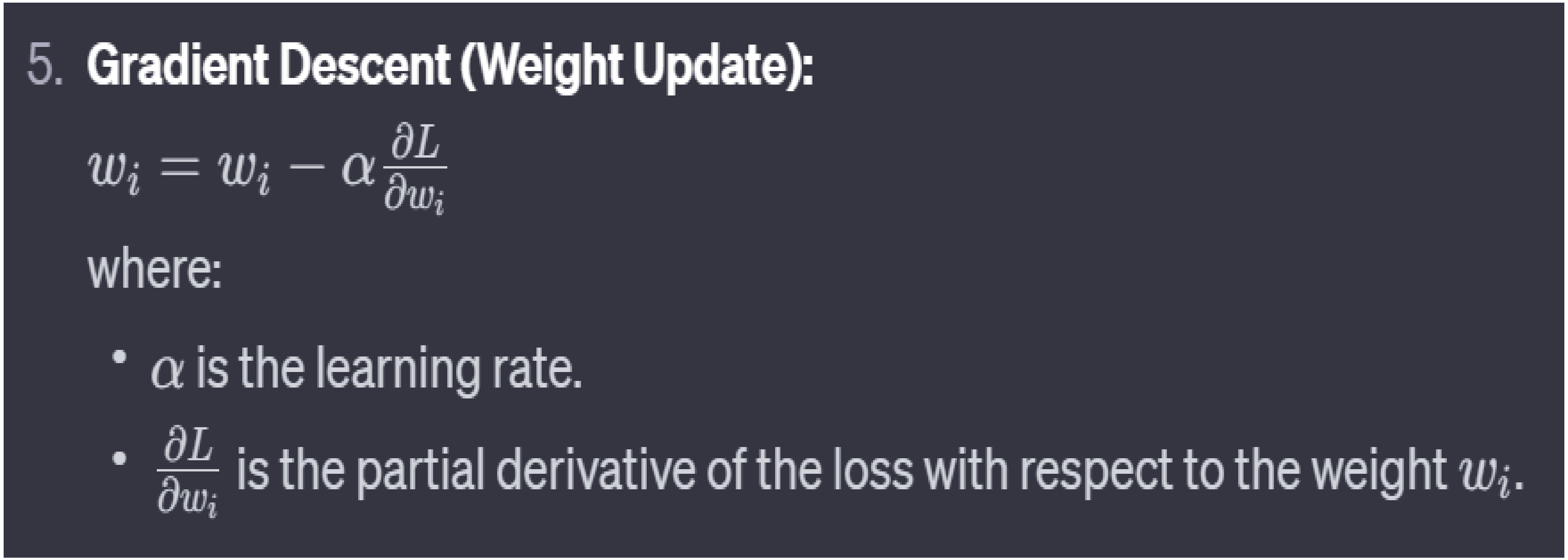
   - a is the output of the neuron.
   - $f(\cdot)$ is the activation function.



3. **Loss Function (Mean Squared Error for Regression):**
   $L = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$
   where:
   - L is the loss.
   - m is the number of examples in the dataset.
   - y_i is the actual output.
   - $\hat{y}_i$ is the predicted output.



4. **Loss Function (Cross-Entropy for Classification):**
   $L = -\frac{1}{m} \sum_{i=1}^{m} [y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$
   where:
   - L is the cross-entropy loss.
   - m is the number of examples in the dataset.
   - y_i is the true class label (0 or 1).
   - $\hat{y}_i$ is the predicted probability of class 1.

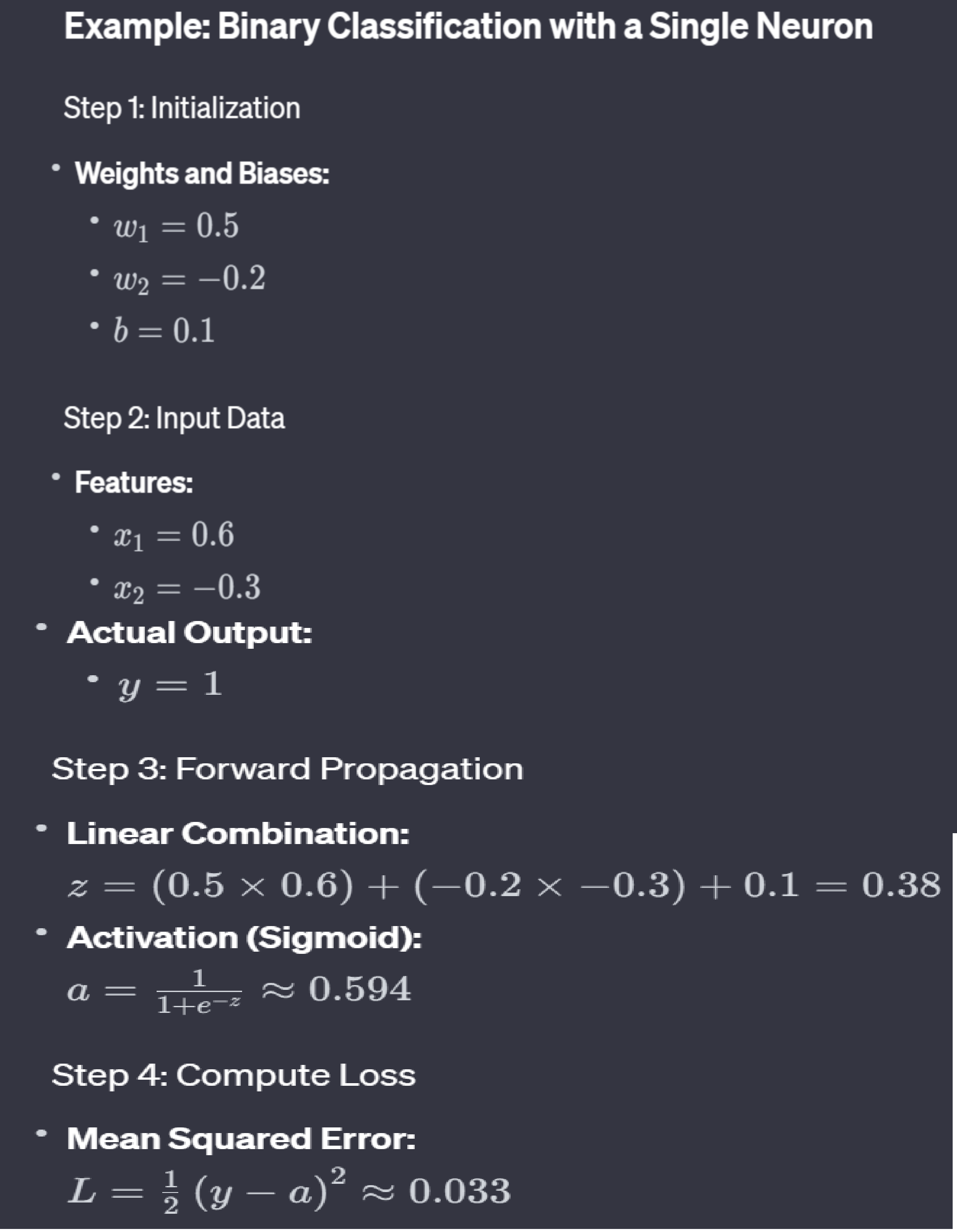


5. **Gradient Descent (Weight Update):**

   $$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

   where:

   - α is the learning rate.
   - $\frac{\partial L}{\partial w_i}$ is the partial derivative of the loss with respect to the weight w_i.



Example: Binary Classification with a Single Neuron

Step 1: Initialization

- **Weights and Biases:**
  - $w_1 = 0.5$
  - $w_2 = -0.2$
  - $b = 0.1$

Step 2: Input Data

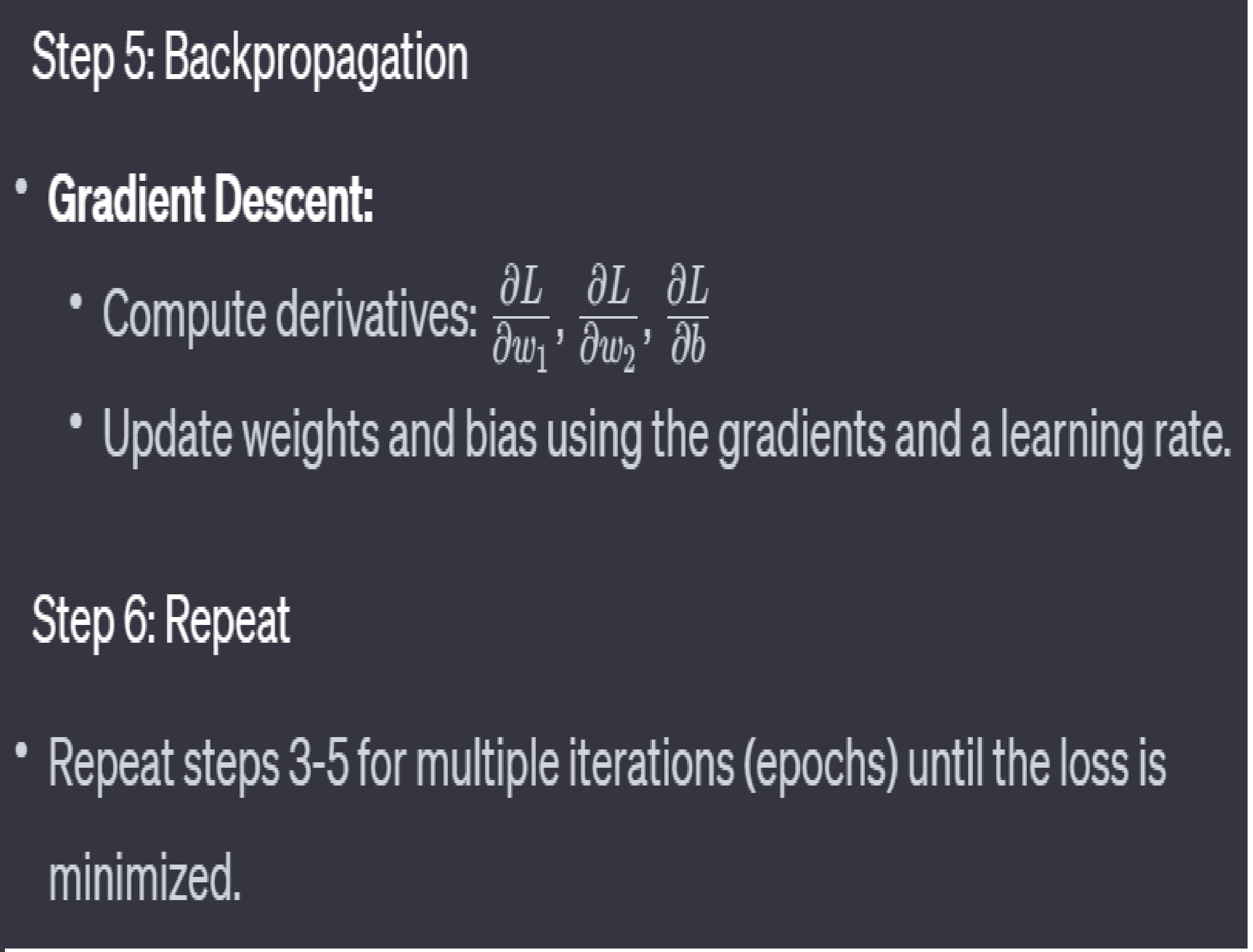
- **Features:**
  - $x_1 = 0.6$
  - $x_2 = -0.3$
- **Actual Output:**
  - $y = 1$

Step 3: Forward Propagation

- **Linear Combination:**
  $z = (0.5 \times 0.6) + (-0.2 \times -0.3) + 0.1 = 0.38$
- **Activation (Sigmoid):**
  $a = \frac{1}{1+e^{-z}} \approx 0.594$

Step 4: Compute Loss

- **Mean Squared Error:**
  $L = \frac{1}{2}(y-a)^2 \approx 0.033$



Step 5: Backpropagation

- **Gradient Descent:**
  - Compute derivatives: $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial b}$
  - Update weights and bias using the gradients and a learning rate.

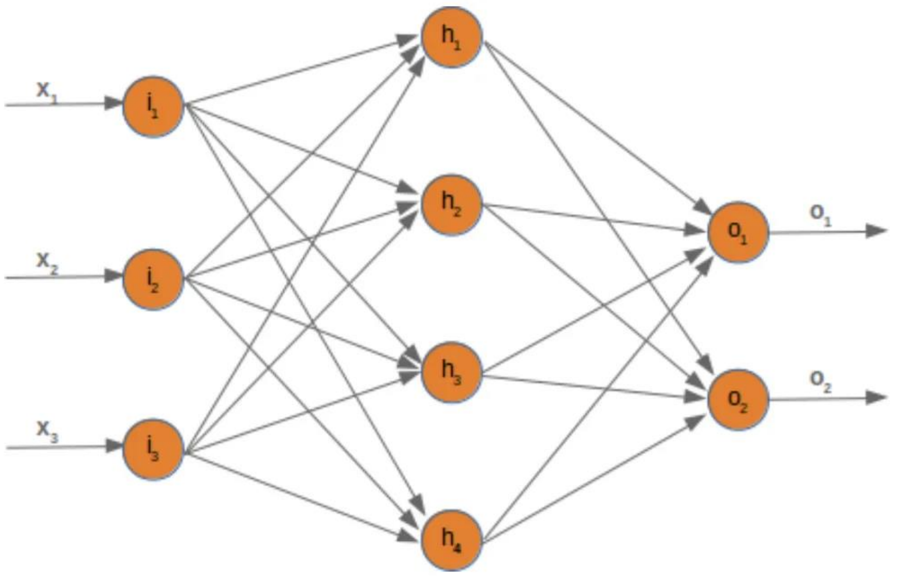
Step 6: Repeat

- Repeat steps 3-5 for multiple iterations (epochs) until the loss is minimized.

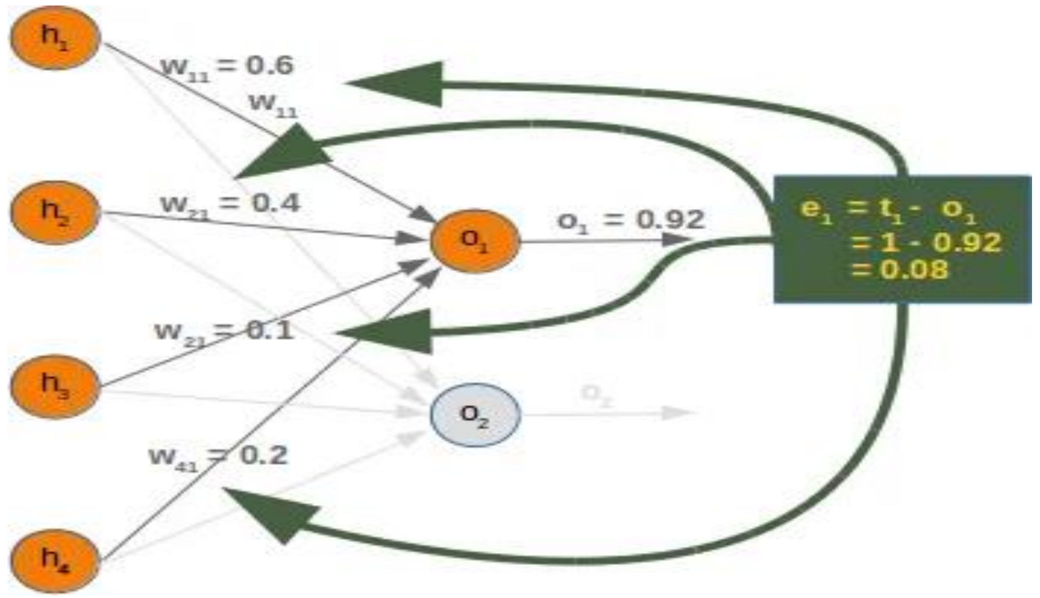
Back propagation

- Backpropagation is a popular method for training artificial neural networks, especially deep neural networks.
- Backpropagation is needed to calculate the gradient, which we need to adapt the weights of the weight matrices.
- The weights of the neurons (ie nodes) of the neural network are adjusted by calculating the gradient of the loss function.
- For this purpose a gradient descent optimization algorithm is used. It is also called backward propagation of errors.

Assuming we start with a simple (linear) neural network:



with the following example value associated with weights:



We have labels, i.e. target or desired values t for each output value o. The error is the difference between the target and the actual output:

$$e_i = t_i - o_i$$

We will later use a squared error function, because it has better characteristics for the algorithm.

$$e_i = \frac{1}{2}(t_i - o_i)^2$$

We will have a look at the output value o1, which is depending on the values w11, w21, w31 and w41. Let's assume the calculated value (o1) is 0.92 and the desired value (t1) is 1. In this case the error is

$$e_1 = t_1 - o_1 = 1 - 0.92 = 0.08$$

Depending on this error, we have to change the weights from the incoming values accordingly. We have four weights, so we could spread the error evenly. However, it makes more sense to do it proportionally, according to the weight values. This means that we can calculate the fraction of the error e1 in w11 as:

$$e_1 \cdot \frac{w_{11}}{\sum_{i=1}^{4} w_{i1}}$$

This means in our example:

$$0.08 \cdot \frac{0.6}{0.6 + 0.4 + 0.1 + 0.2} = 0.037$$

The total error in our weight matrix between the hidden and the output layer looks like this:

$$e_{who} = \begin{bmatrix} \frac{w_{11}}{\sum_{i=1}^{4} w_{i1}} & \frac{w_{12}}{\sum_{i=1}^{4} w_{i1}} \\ \frac{w_{21}}{\sum_{i=1}^{4} w_{i1}} & \frac{w_{22}}{\sum_{i=1}^{4} w_{i1}} \\ \frac{w_{31}}{\sum_{i=1}^{4} w_{i1}} & \frac{w_{32}}{\sum_{i=1}^{4} w_{i1}} \\ \frac{w_{41}}{\sum_{i=1}^{4} w_{i1}} & \frac{w_{22}}{\sum_{i=1}^{4} w_{i1}} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

The denominator in the left matrix is always the same (scaling factor). We can drop it so that the calculation gets simpler:

$$e_{who} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{22} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$