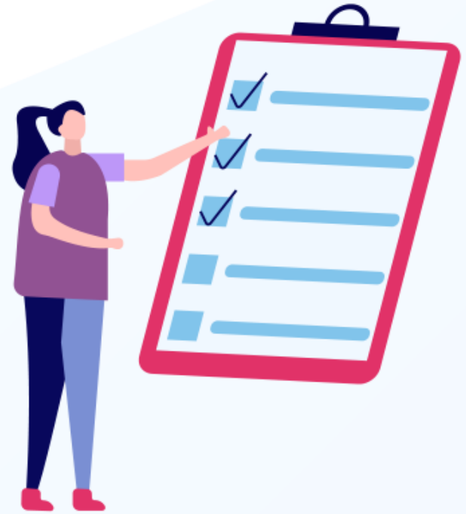
Deep Learning

Deep Neural Networks

Learning Objectives

By the end of this lesson, you will be able to:

- Describe the loss function
- Evaluate the implementation of loss function
- Analyze backward and forward propagation in DNNs
- Examine the use of TensorFlow to train DNNs



Business Scenario

A financial services company is looking to improve its risk assessment model for its loan application process. The company wants to implement a deep neural network (DNN) to analyze customer data and calculate the probability of loan defaults.

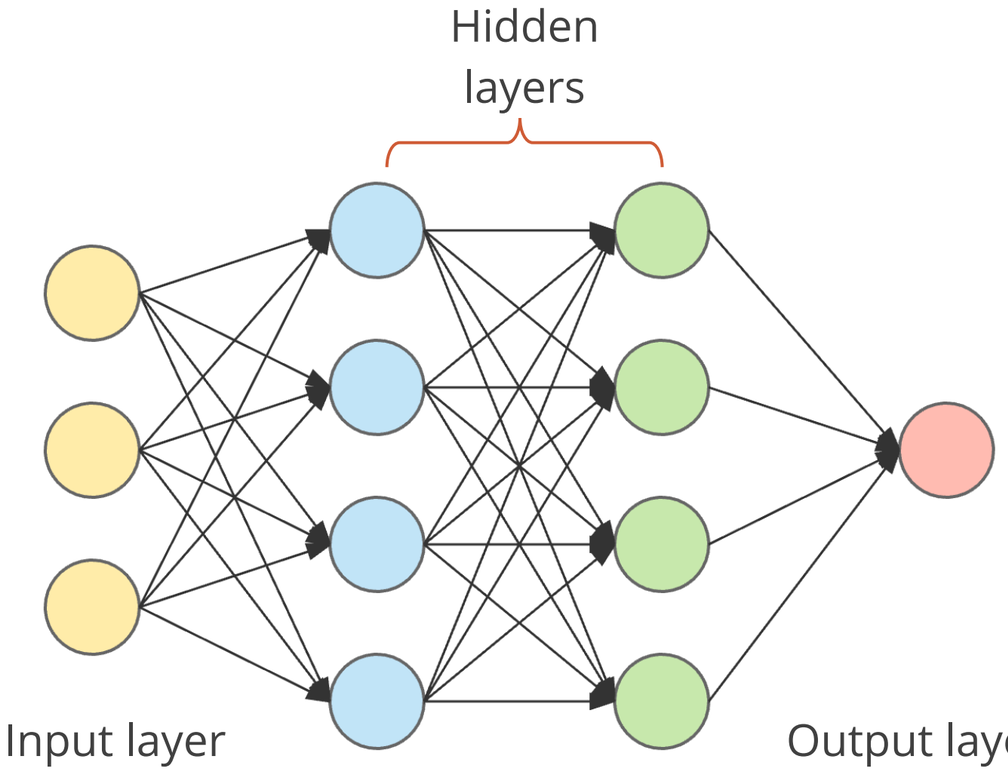
It has a large dataset with various features, including income, credit score, and employment history. The company plans to build the DNN using TensorFlow and normalize the data to ensure effective learning. It will use loss functions to estimate the model's error or loss and adjust the network accordingly. The company plans to use the TensorFlow Playground as a helpful tool to understand the neural network visually and make necessary adjustments.

By implementing a DNN, the company aims to improve the accuracy of its risk assessment model, reduce defaults, and ultimately provide better loan decisions for its customers.

Introduction to Deep Neural Network (DNN)

Deep Neural Network

It refers to a type of artificial neural network that consists of hidden layers between the input and output layers.

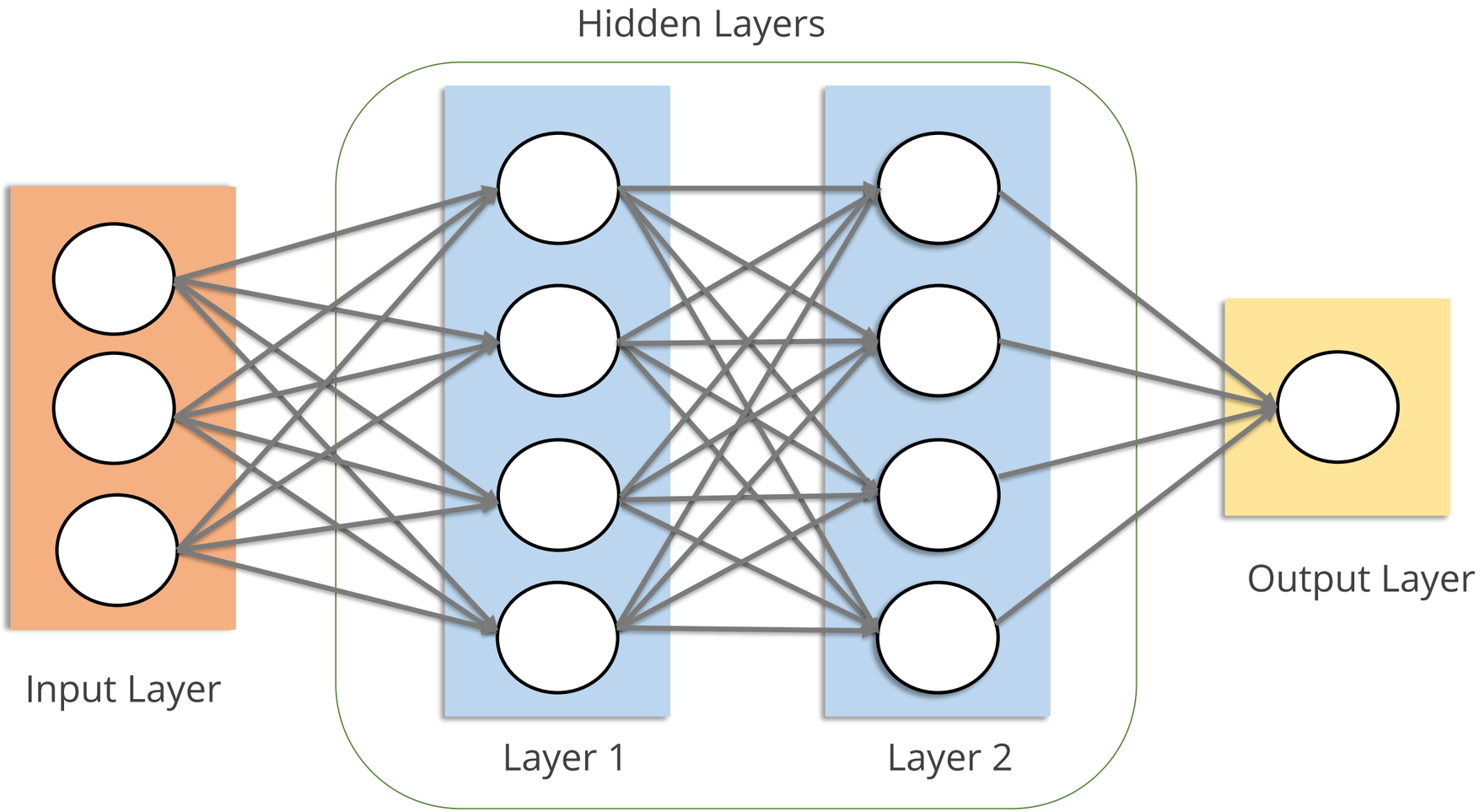


Nonlinearity is a key factor that distinguishes deep neural networks (DNNs) from traditional neural networks (NNs), enabling DNNs to outperform NNs in various tasks.

DNNs offer higher accuracy and have the ability to emulate the decision-making process of the human brain more effectively.

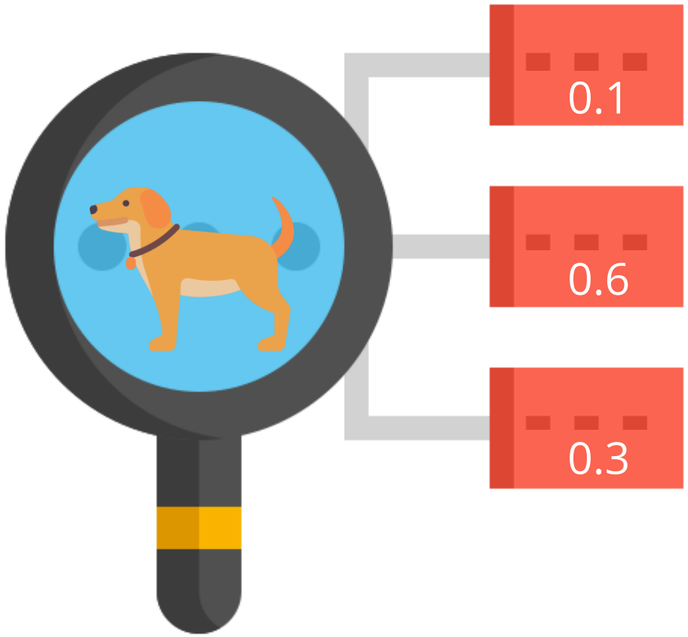
Deep Neural Network

DNNs are a powerful category of machine learning algorithms implemented by layers of neural networks along the depth and width of smaller architectures.



DNN: Example

Consider a DNN designed and trained to recognize dog breeds



It can analyze an image of a dog and predict its specific breed based on probability calculations.

The breed with the highest probability is usually chosen as the predicted breed.

DNN: Example

If an image or a sound is fed into a computer system, it would not be able to recognize such input without DNN.

Consider the sound of a trumpet played on a computer

DNN identifies the sound and sorts it into different categories.

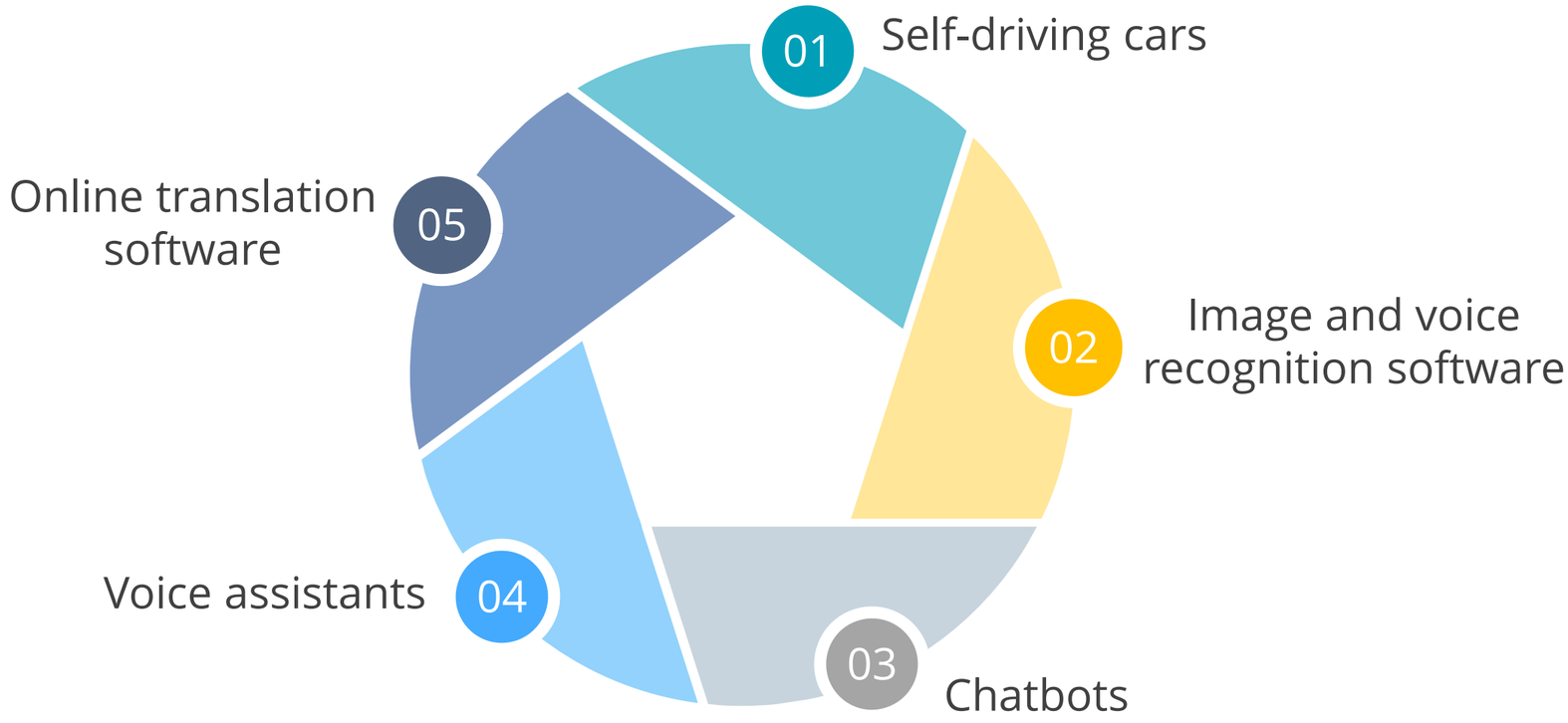
This is carried out by the DNN's many hidden layers.

Greater recognition of trumpet sounds by a DNN enhances accuracy and speed.

Benefits of DNN

DNN is one of the most efficient and accurate AI learning processes when given large amounts of data.

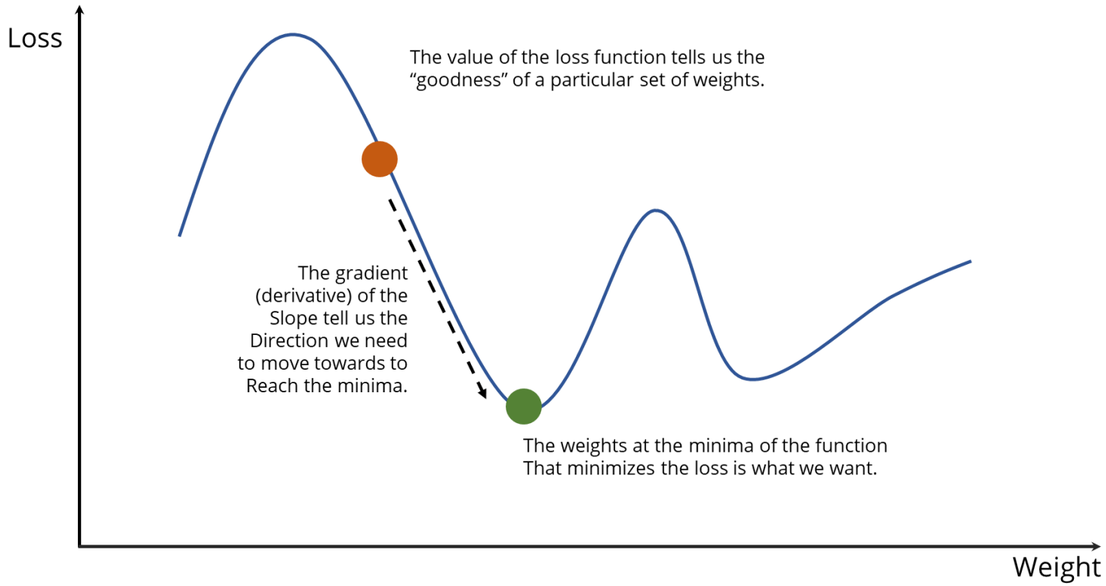
It has accelerated the development of technologies such as:



Loss Function in DNN and Types

Loss Function in DNN

It is used to measure the discrepancy between predicted and actual values, enabling the network to learn and improve its performance during training.



The model being developed needs to be continually assessed for potential errors as part of the optimization process.

Loss Function: Example

Assign labels to two images: the horse is 0, and the human is 1.

Classify all images of horses and humans in this manner

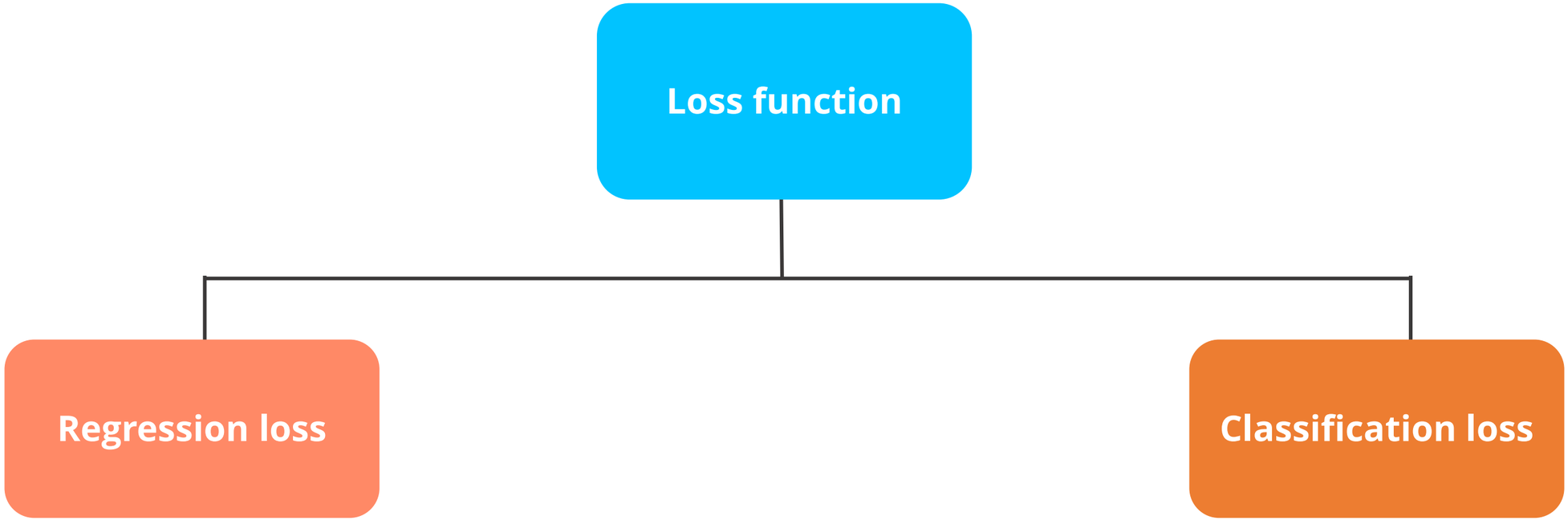
If the model receives an image of a horse and identifies the output as 0.25, the difference in the model's prediction and the label is:
0.25 - 0 = 0.25

This difference is known as the error in the model.

Every output undergoes this precise process, and the error is gathered from each individual output during the learning iterations.

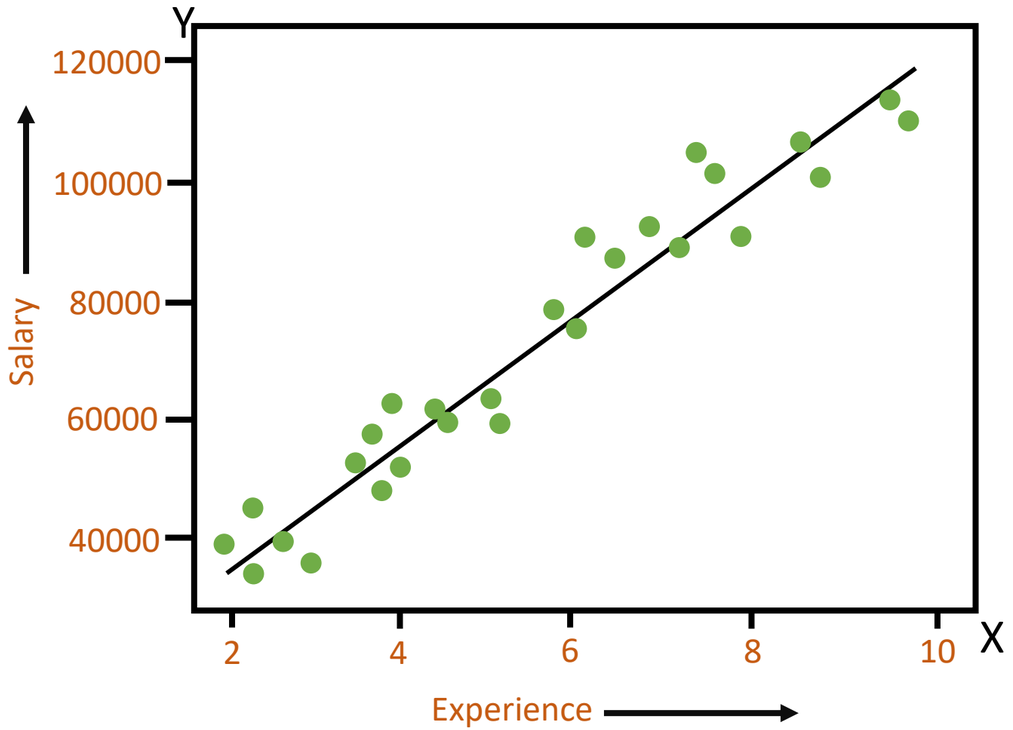
Loss Function and Its Major Categories

The losses of deep learning models can be evaluated very easily by using the loss function.



Regression Loss

Neural network regression predicts continuous values using input features and a suitable loss function.



The output here provides a value, such as the salary of an employee on the basis of the experience

Types of Regression Loss

The different types of regression loss are:

Mean Absolute Error

MAE measures the average absolute difference between the predicted and true values in regression tasks.

$$MAE = \frac{1}{n} \Sigma |y - \hat{y}|$$

y	→	The actual target values
$\hat{y}$	→	The predicted values
$y - \hat{y}$	→	The absolute difference between actual and predicted values

Mean Absolute Error

MAE is used as a metric to measure the average magnitude of errors between predicted and actual values in regression problems.

MAE = Sum of Mean Errors/N

Y (Actual Value)	Y' (Predicted Value)	|Y – Y'| (Actual – Predicted value)
10.2	9.4	0.8
7.1	6.9	0.2
17.2	18.4	1.2
9.5	11.3	1.8
11.5	11.1	0.4
Sum		4.4
MAE		4.4/5 = 0.88

Mean Absolute Error

MAE is the absolute difference between the actual and predicted values for a given number of training data points.

Syntax:

```
#Calculate the mean absolute error
import numpy as np
def mean_absolute_error(actual, predicted):
 absolute_errors = np.abs(actual - predicted)
 mean_absolute_error = np.mean(absolute_errors)
 return mean_absolute_error
```

Mean Squared Error

MSE measures the average squared difference between predicted and true values in regression tasks.

The equation below can be used when there is a specific input and output pair:

$$MSE = (output - input)*(output – input)$$

In the loss function MSE, the difference between input prediction and output label for a single sample is calculated.

Mean Squared Error

If multiple samples are passed at the same time, the mean of the squared errors over all the samples can be taken.

$$\text{MSE} = \frac{1}{n} \Sigma |y - \hat{y}|^2$$

y	→	The actual target values
$\hat{y}$	→	The predicted values
$(y - \hat{y})^2$	→	The square of the difference between actual and predicted values

MSE is only one of many loss functions that can be implemented.

Mean Squared Error

The following equation can be used when evaluating the mean squared error (MSE) metric, which measures the average squared difference between the predicted and actual values.

MSE = Sum of Squared Errors/N

Y (Actual Value)	Y' (Predicted Value)	$\|Y - Y'\|^2$ (Actual – Predicted value)^2
10.2	9.4	0.64
7.1	6.9	0.04
17.2	18.4	1.44
9.5	11.3	3.24
11.5	11.1	0.16
Sum		5.52
MSE		5.52/5 = 1.104

Mean Squared Error

MSE between the actual and predicted values in a regression task using a mathematical formula.

Syntax:

```
#Calculate the mean squared error
import numpy as np
def mean_squared_error(actual, predicted):
 square_errors = (actual - predicted) ** 2
 mean_square_error = np.mean(square_errors)
 return mean_square_error
```

MSE vs. MAE

In MSE, since each error is squared, it penalizes larger differences in prediction more heavily compared to MAE due to the squaring of errors in MSE.

Y (Actual Value)	Y' (Predicted Value)	$\|Y - Y'\|$ (Actual – Predicted value)	$\|Y - Y'\|^2$ (Actual – Predicted value)^2
10.2	9.4	0.8	0.64
7.1	6.9	0.2	0.04
17.2	18.4	1.2	1.44
9.5	11.3	1.8	3.24
11.5	11.1	0.4	0.16
Sum		4.4	5.52
		MAE = 4.4/5 = 0.88	MSE = 5.52/5 = 1.104

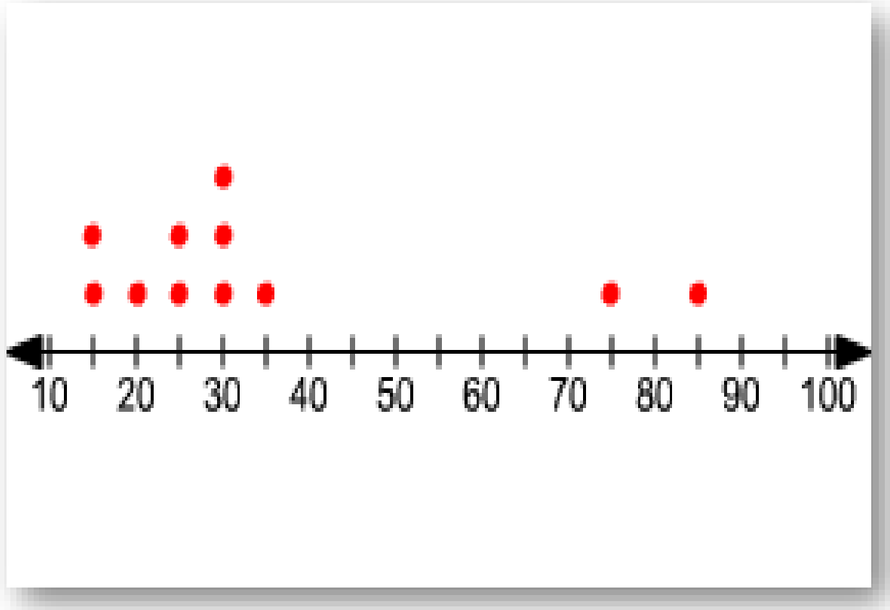
MSE vs. MAE

The effect of MSE on outliers is adverse. Since each error is squared in the MSE, the final MSE can increase significantly. For example:

Y (Actual Value)	Y' (Predicted Value)	|Y – Y'| (Actual – Predicted value)	|Y – Y'|^2 (Actual – Predicted value)^2
10.2	9.4	0.8	0.64
7.1	6.9	0.2	0.04
17.2	18.4	1.2	1.44
31.5	11.3	20.2	408.04
11.5	11.1	0.4	0.16
Sum		22.8	410.32
Loss Function		MAE = 22.8/5 = 4.56	MSE = 410.32/5 = 82.064

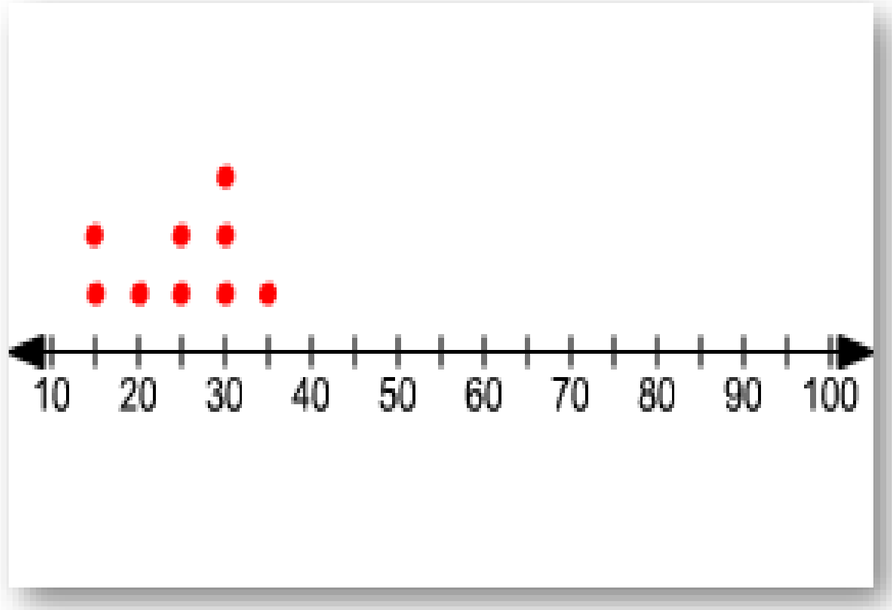
MSE vs. MAE

If the data has outliers, MAE will be a better option than MSE. For data without outliers, MSE is preferable.



MAE as a loss function

(Absolute difference evaluation metric)



MSE as a loss function

(Squared error evaluation metric)

MSE in Backpropagation

MSE is commonly used in backpropagation due to its empirical effectiveness in minimizing squared differences between predicted and actual values, making it applicable to various regression tasks.

MSE Equation

$$MSE\ (W) = \frac{1}{N} \sum_{i=1}^{N} (y - y')^2$$

Binary Classification Problem

It is a problem where a particular example is classified into one of two classes.

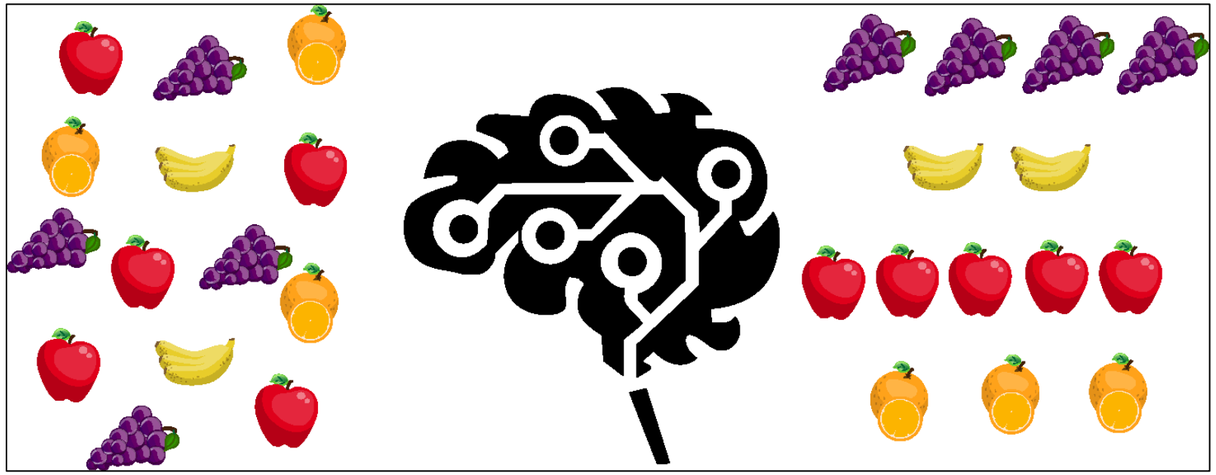
Example

A class of which the integer value is 1 and another class of which the value is 0

It assigns examples to predefined classes based solely on the likelihood of belonging to a class, rather than predicting the probability of belonging to a specific class.

Multi-Class Classification Problem

It is a problem where an example can be classified as being in one of more than two classes.



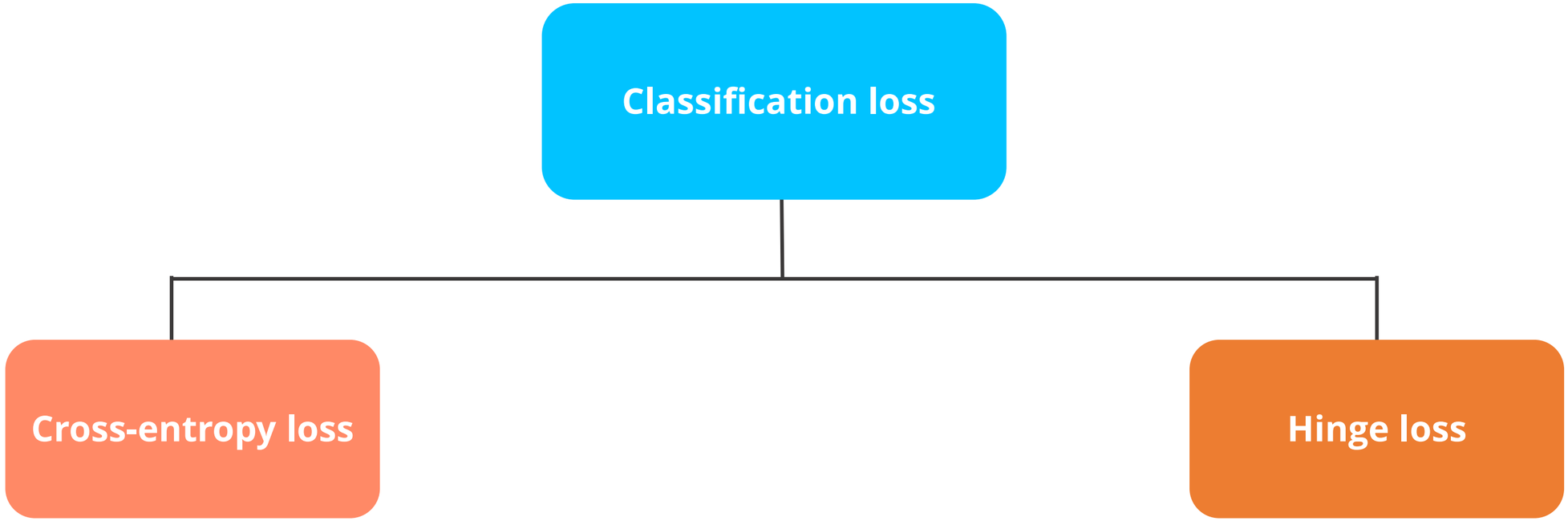
Example

Classifying images of fruits into categories such as apples, bananas, grapes, and oranges, where each category represents a different class label

It is set up to predict the probability of an example belonging to each class.

Types of Classification Loss

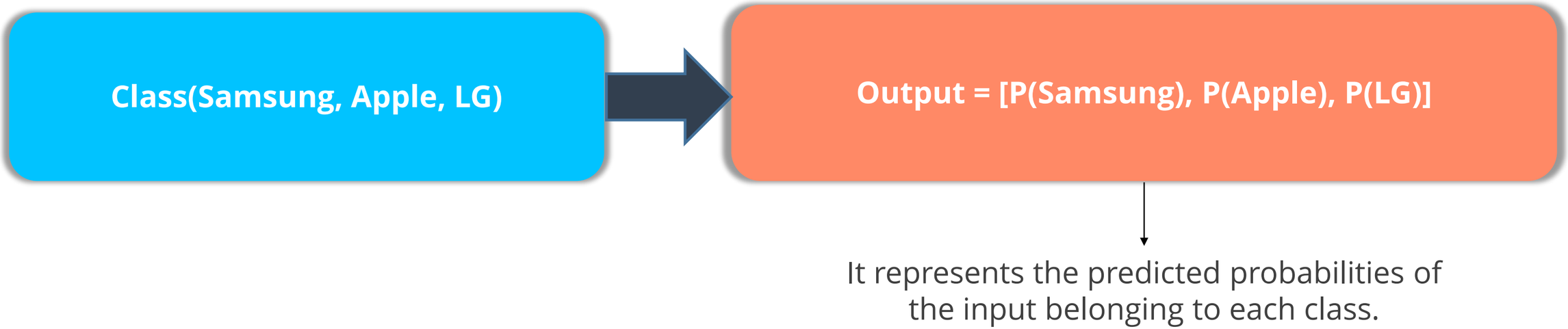
The different types of classification losses are:



Cross-Entropy Loss

It is a way to calculate the distance between two probability distributions.

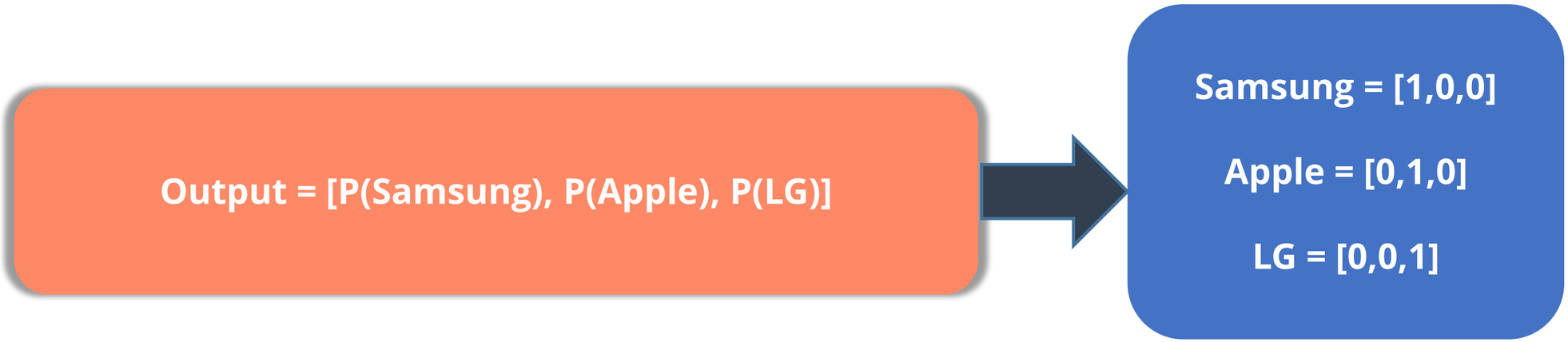
For example, consider a classification problem with three classes that is Samsung, Apple, and LG.



The class with the highest probability is the winner.

Cross-Entropy Loss

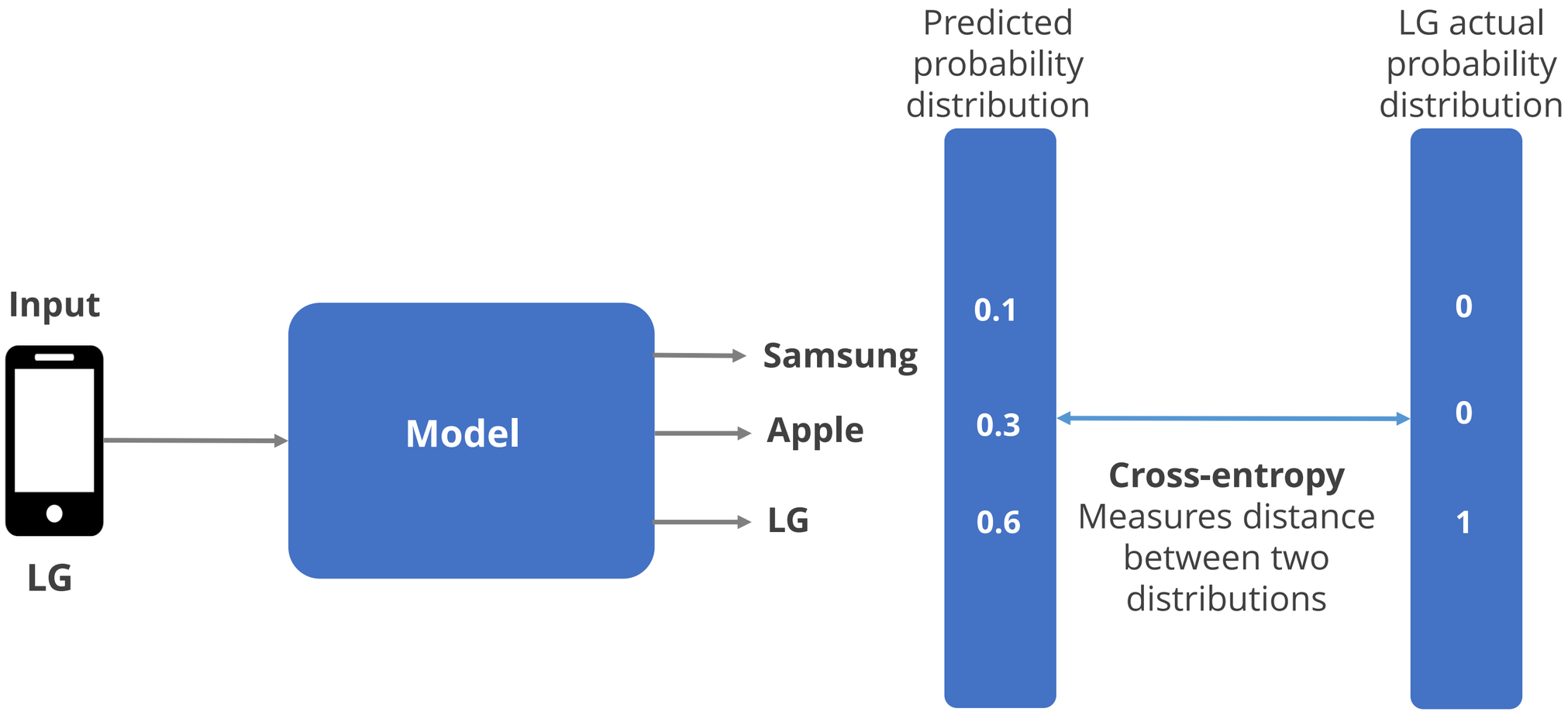
If the predicted probability distribution is not close to the actual value, the model adjusts its weight.



The actual probability distribution for each class

Cross-Entropy Loss

In this scenario, cross-entropy is used as a tool to calculate the difference between the predicted probability distribution and the actual one.



Intuition behind cross-entropy

Cross-Entropy Loss: Calculation

- The model gives the predicted probability distribution for N classes for a particular input data C.

$$P(C) = [y1', y2', y3', \ldots\ldots yN]$$

- The actual or target probability distribution of data C is:

$$A(C) = [y1', y2', y3', \ldots\ldots yN]$$

- The cross-entropy for data C is calculated as:

$$\text{Cross-Entropy}(A,P) = -(y1 * \log(y1') + y2 * \log(y2') + y3 * \log(y3') + \cdots . + yN * \log(yN'))$$

Cross-Entropy Loss: Calculation

The following formula measures the cross-entropy for a single observation or input of data from the example:

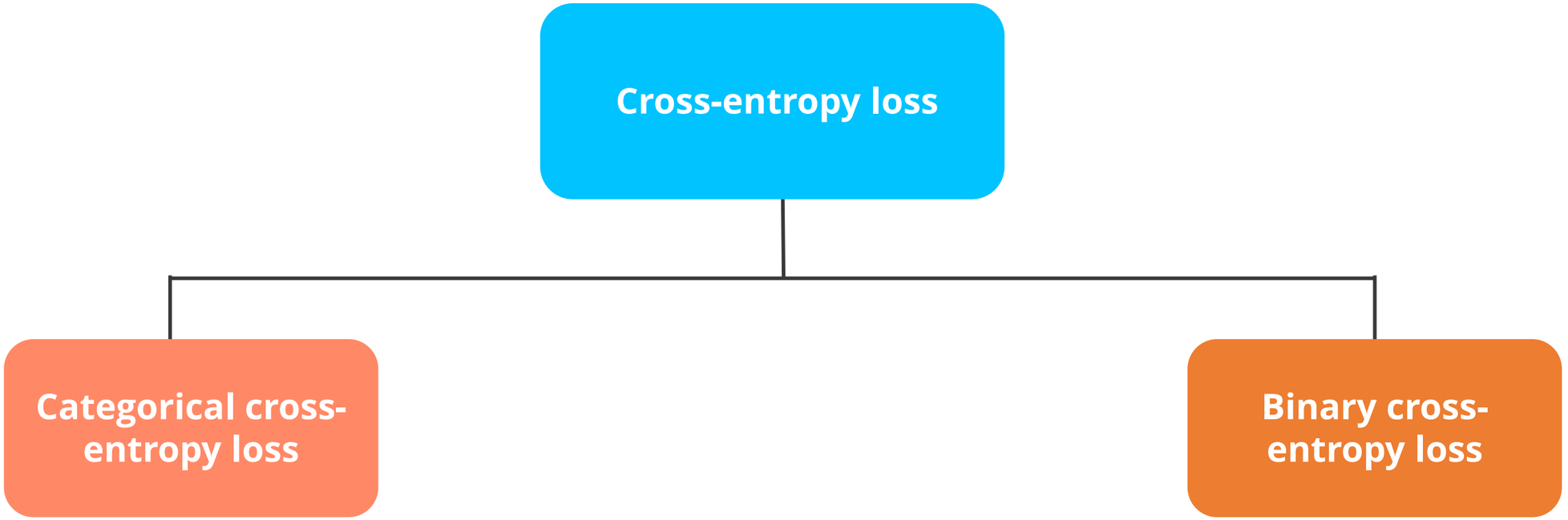
$$P(LG) = [0.6, 0.3, 0.1]$$

$$A(LG) = [1, 0, 0]$$

$$\text{Cross-Entropy}(A,P) = -(1 * \log(0.6) + 0 * \log(0.3) + 0 * \log(0.1)) = 0.51$$

Types of Cross-Entropy Loss

The different types of cross-entropy losses are:



Categorical Cross-Entropy Loss

It measures dissimilarity between predicted class probabilities and true class labels in multi-class classification.

Categorical cross-entropy = sum of cross-entropy for N data/N

Data	Actual probability distribution	Predicted probability distribution	Cross-entropy
Samsung	[1, 0, 0]	[0.6, 0.3, 0.1]	– (1*log(0.6) + 0*log(0.3)+0*log(0.1)) = 0.51
Samsung	[1, 0, 0]	[0.9, 0.1, 0]	– (1*log(0.9) + 0*log(0.1)+0*log(0.1)) = 0.1
Apple	[0, 1, 0]	[0.2, 0.7, 0.1]	– (0*log(0.2) + 1*log(0.7)+0*log(0.1)) = 0.35
LG	[0, 0, 1]	[0.3, 0.2, 0.5]	– (0*log(0.3) + 0*log(0.2)+1*log(0.5)) = 0.69
Apple	[0, 1, 0]	[0.6, 0.1, 0.3]	– (0*log(0.6) + 1*log(0.1)+0*log(0.3)) = 2.3
Samsung	[1, 0, 0]	[0.5, 0.2, 0.3]	– (1*log(0.5) + 0*log(0.2)+0*log(0.3)) = 0.69
LG	[0, 0, 1]	[0.1, 0.1, 0.8]	– (0*log(0.1) + 0*log(0.1)+1*log(0.8)) = 0.22
Loss function			(0.51 + 0.1 + 0.35 + 0.69 + 2.3 + 0.69 + 0.22) / 7 = 4.76

Binary Cross-Entropy Loss

- Binary cross-entropy assumes a binary value of 0 or 1 to denote the negative and positive classes, respectively, when there is only one output.
- The actual output is denoted by a single variable y, and then the cross-entropy for a particular data C can be simplified as follows:

Cross-Entropy (C) = – y*log(y') when y = 1

Cross-Entropy (C) = – (1- y)*log(1- y') when y = 0

- The error in binary classification for the complete model is given by binary cross-entropy, which is nothing but the mean of cross-entropy for N data.

Binary Cross-Entropy = Sum of Cross-Entropy for N data/N

Binary Cross-Entropy Loss: Calculation

The implementation of a function to determine the cross-entropy for a list with actual 0 and 1 values in comparison to the expected probability for class 1.

Syntax:

```
from math import log

# Calculate binary cross entropy
def binary_cross_entropy(actual, predicted):
 sum_score = 0.0
 for i in range(len(actual)):
 sum_score += actual[i] * log(1e-15 + predicted[i])
 mean_sum_score = 1.0 / len(actual) * sum_score
 return -mean_sum_score
```

Sparse Categorical Cross-Entropy Loss

It is a loss function used in machine learning for multi-class classification tasks where each sample belongs to only one class, represented by integer labels, and not one-hot encoded vectors.

Equation

$$\text{Loss} = -1/N \sum_{i=1}^{\text{Output size}} \log(Pi)$$

- N is the number of samples or instances in the dataset.
- P_i is the predicted probability of the correct class for the i^{th} sample.

Cross-Entropy Loss Over MSE/MAE

Overconfident wrong prediction occurs when MSE or MAE is used in classification, especially during the training phase.

Cross-Entropy Loss Over MSE/MAE

- Using MSE or MAE in classification can lead to overconfident wrong predictions during the training phase.
- The model may assign relatively small errors to wrong predictions, resulting in high certainty for incorrect classifications.
- This overconfidence can cause the model to make mistakes with a false sense of certainty.
- To mitigate this issue, appropriate loss functions like cross-entropy should be used, which directly optimize for classification, accuracy, and probability distribution.

Forward Propagation in DNN

Discussion

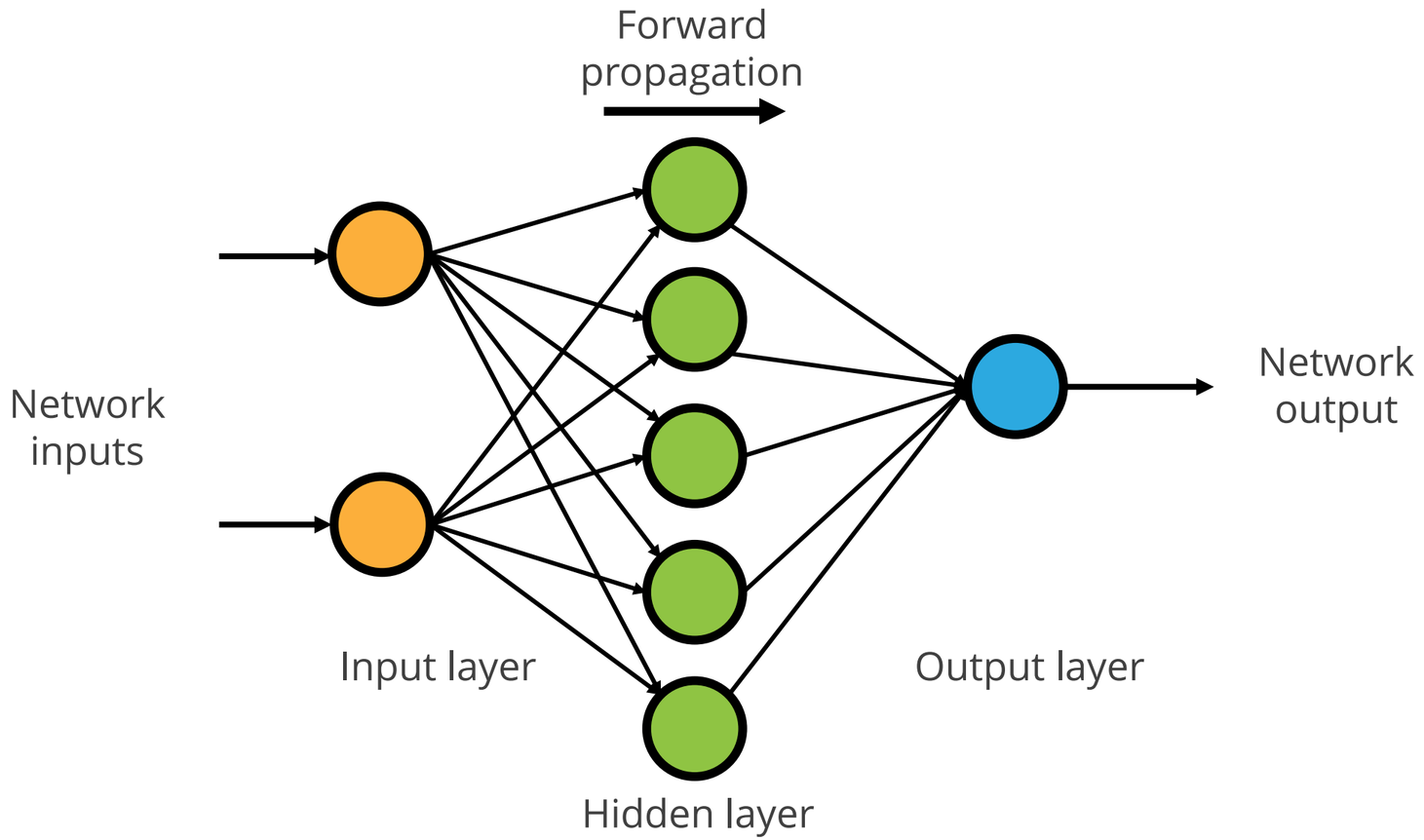
Discussion: Forward and Backward Propagation

Duration: 10 minutes

- What is forward propagation in a deep neural network?
- What is backward propagation in a deep neural network?

Forward Propagation

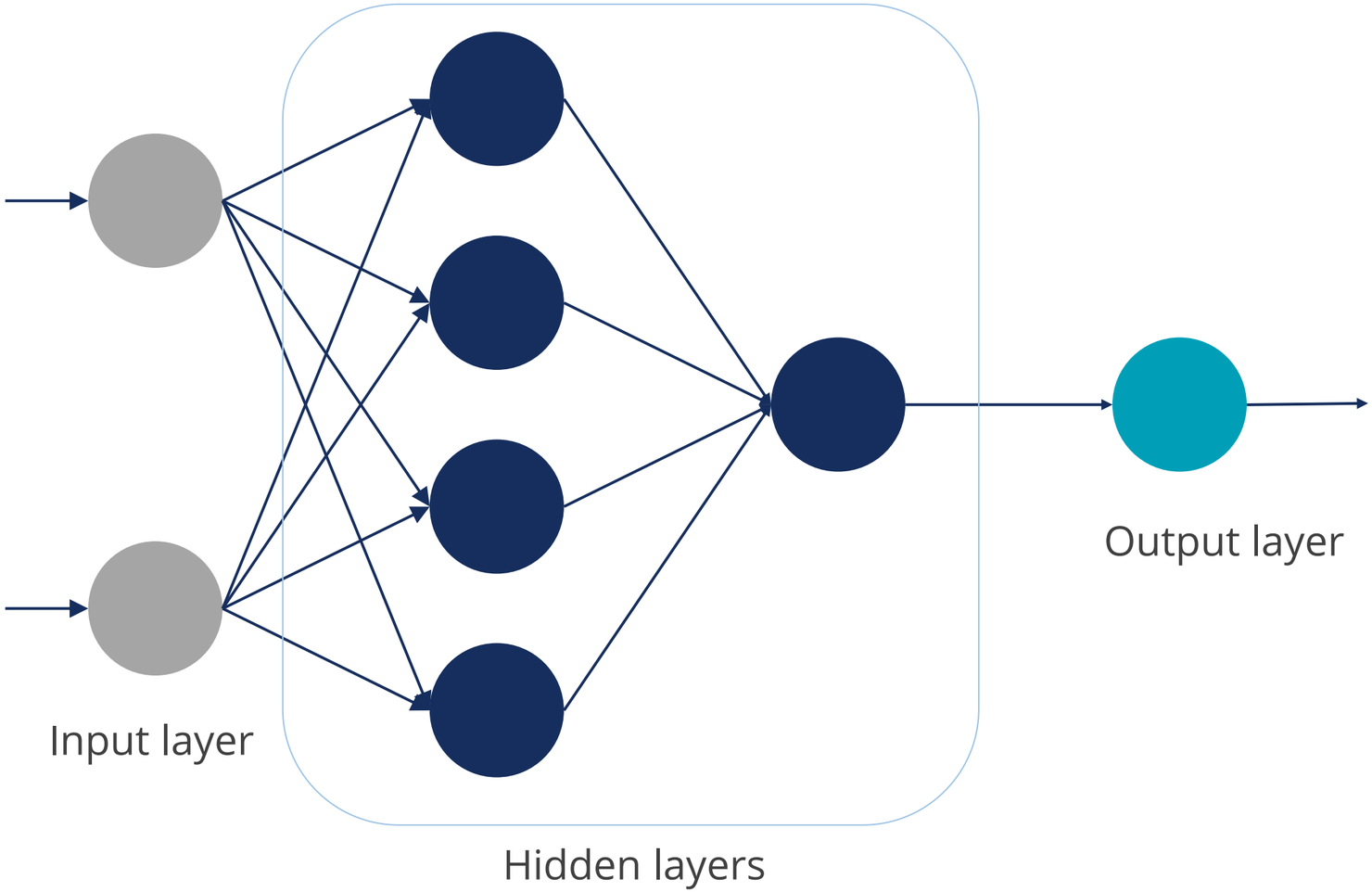
When a weighted total is supplied to an activation function and the result is the output for a specific node, it is passed as part of the input for the nodes in the following layer.



This process is known as forward propagation.

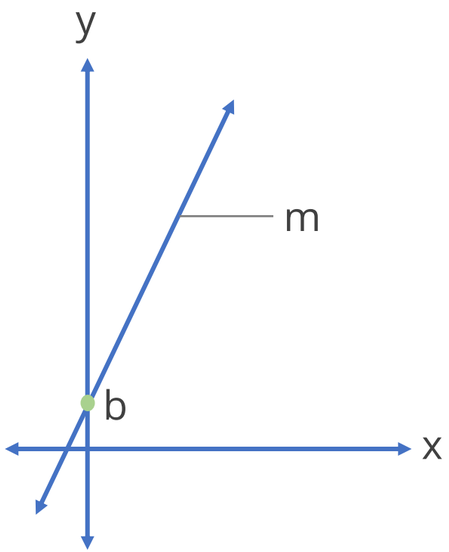
Forward Propagation

The process occurs for each layer in the network until the input reaches the output layer.



Working on Forward Propagation

The workings of forward propagation can be explained mathematically:



A line can be represented by the equation.

$$y = mx + b$$

y is the y coordinate of the point

m is the slope

x is the x coordinate

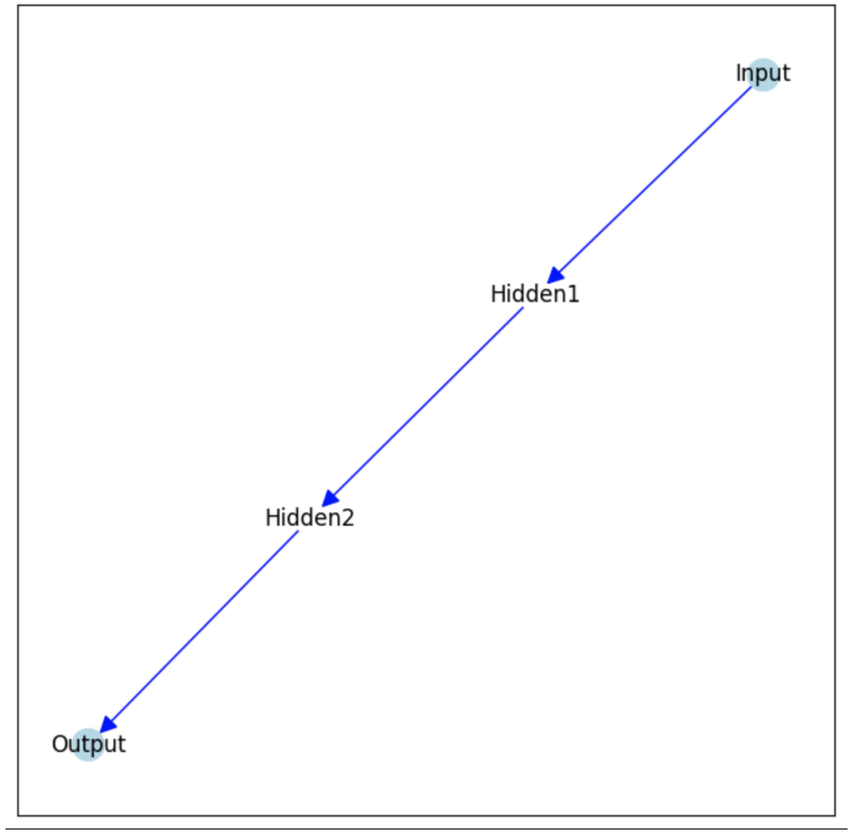
b is the y-intercept which is the point at which the line crosses the y-axis

Example: If it is assumed that x is the input and y is the output, then to initialize the parameter, it can be assumed that y is an x multiplication factor.

Backward Propagation in DNN

Backward Propagation

It refers to the practice of adjusting the weights of a neural network based on the error rate or loss collected in the previous epoch.



It is the essence of neural net training and helps ensure lower error rates.

Backpropagation can indeed overtrain or overfit the model, just like any other training or fitting method

Backward Propagation

The method of calculating the loss varies depending on the loss function used.

The loss function is the distance the model is from correctly classifying the provided input.

It is the difference between the model's prediction for a given input and what the actual provided input is.

Backward Propagation

The goal of the gradient descent is to reduce the size of the loss function.

d(loss) / d(weight)

This is done by calculating the derivative, or gradient, of the loss function with regard to the model's weights.

Backward Propagation: Uses

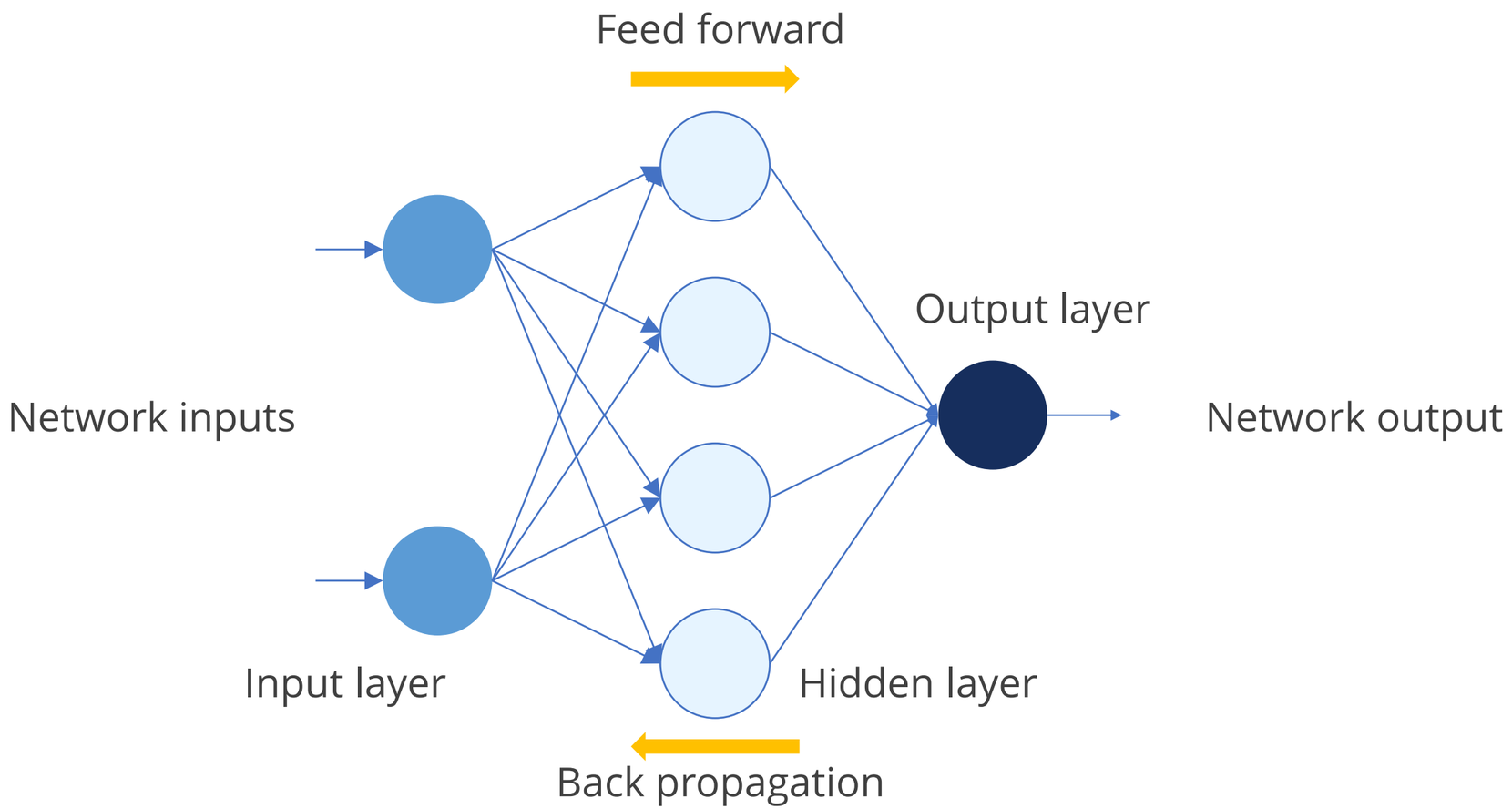
The gradient descent algorithm uses backward propagation to find the gradient of the loss function.

The loss is calculated for the output generated from the given input.

Subsequently, backward propagation is used to update the weights in order to minimize the loss function.

Backward Propagation: Uses

Gradient descent begins by looking at the activation outputs from the output nodes to readjust the weights.



Backward Propagation: Uses

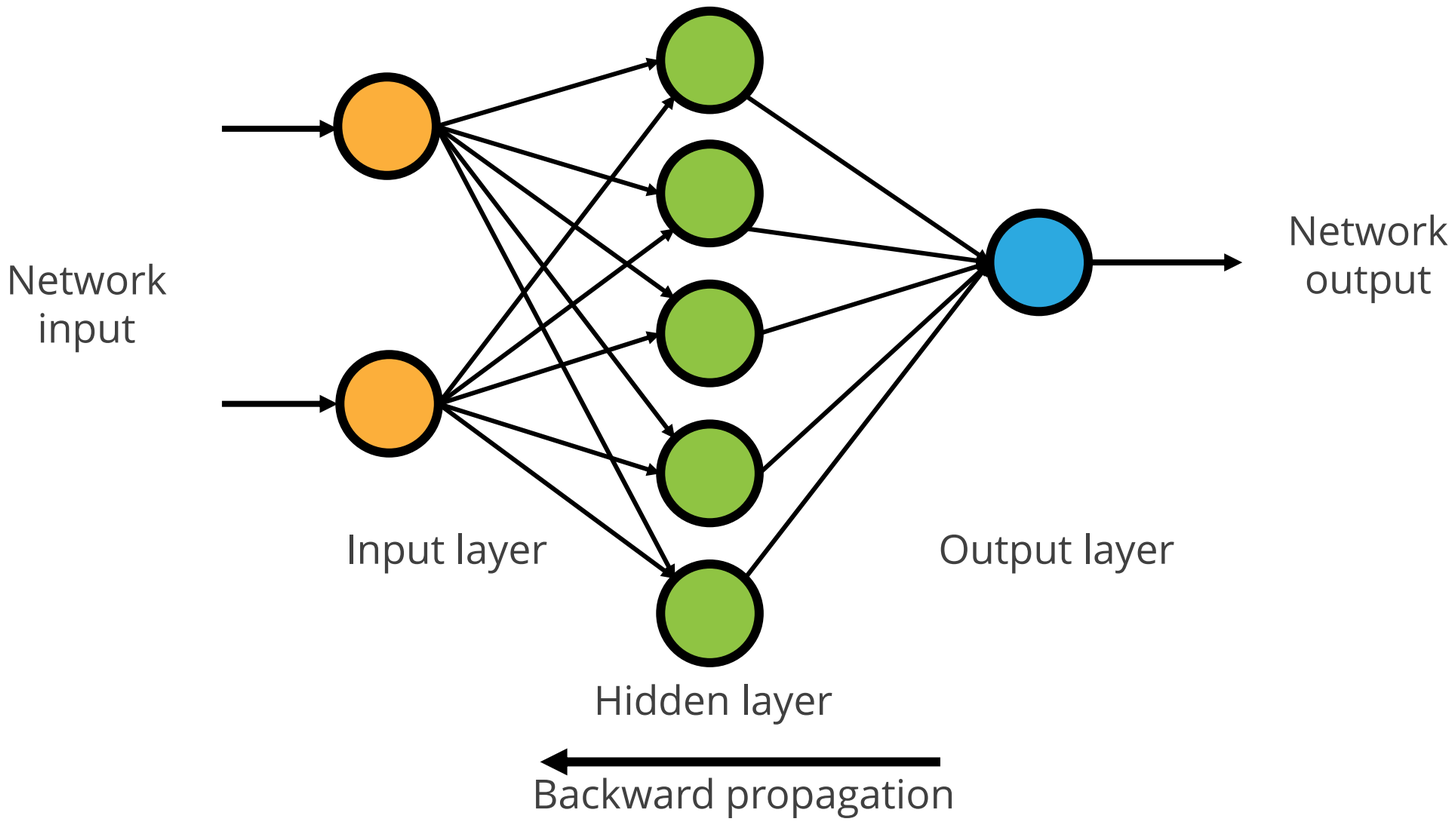
If the output nodes predict a value higher than the true value, the gradient descent algorithm recognizes the following:

The value of all output should fall.

These allow the algorithm to lower the loss for the input.

Backward Propagation: Uses

The algorithm travels backward through the network and adjusts the weights from right to left.



This way, it can slightly shift the values from the output nodes in the direction that they are required to go to assist in reducing the loss.

Discussion: Forward and Backward Propagation

Duration: 10 minutes

- What is forward propagation in a deep neural network?

Answer: It is the process of computing and passing input data through the network's layers to generate an output prediction.

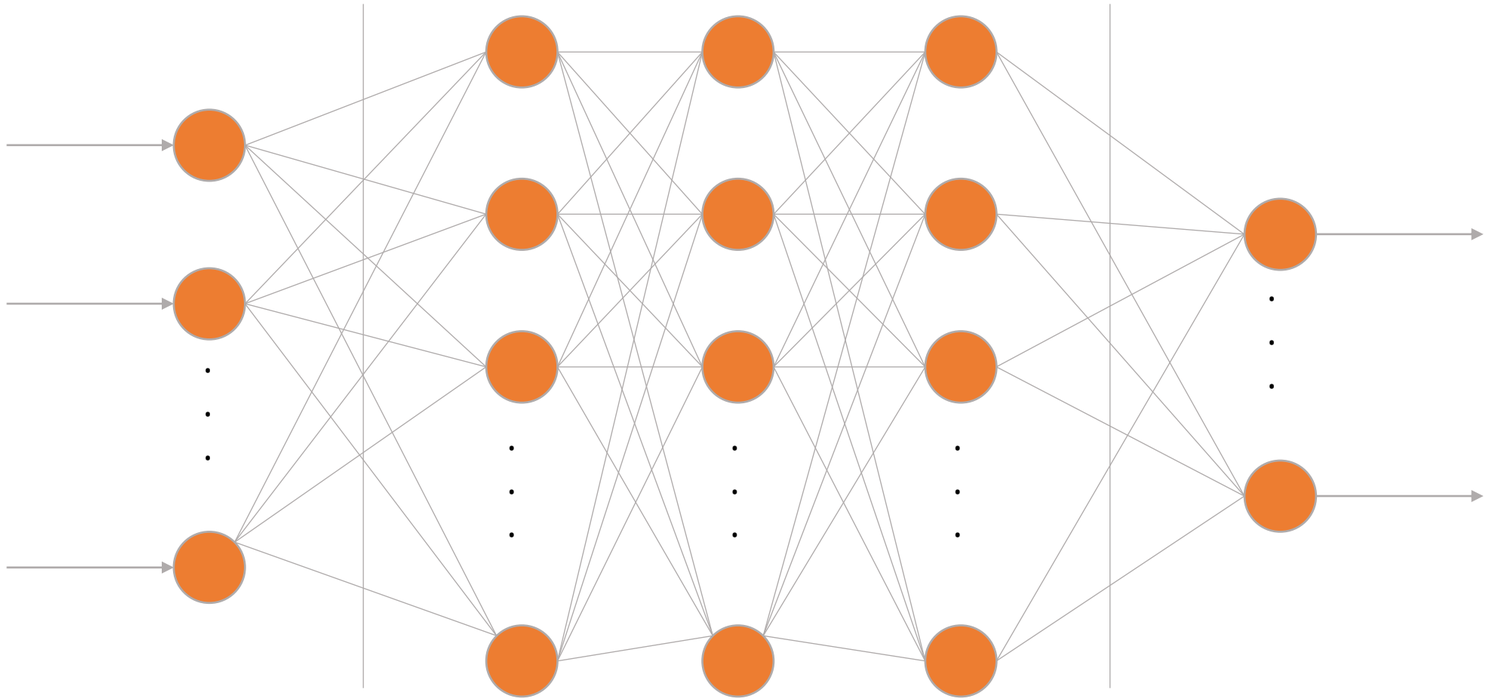
- What is backward propagation in a deep neural network?

Answer: It is the process of calculating and propagating gradients from the output layer back to the input layer to update the model's parameters during training

Regularization

Deep Neural Networks

When a neural network contains more than one hidden layer, it becomes a deep neural network.

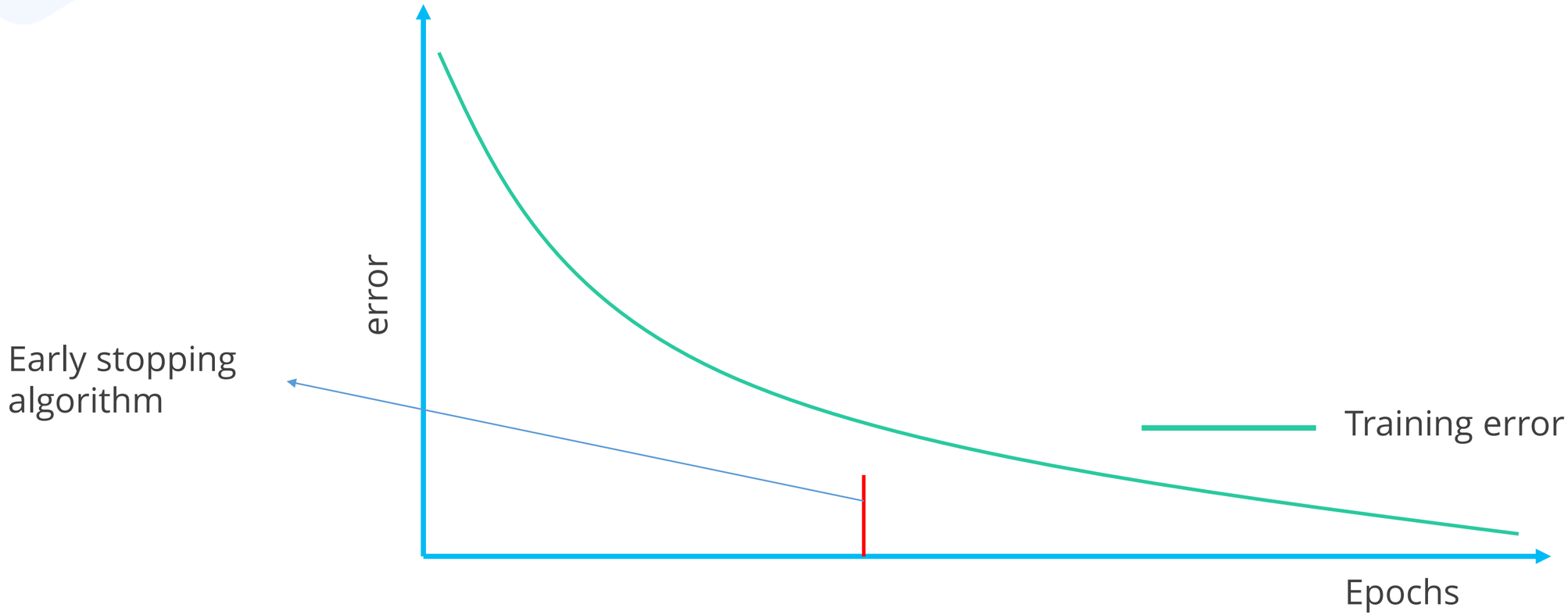


Input layer

Multiple hidden layers

Output layer

The Overfitting Problem



Epochs refer to the number of times the entire dataset is passed forward and backward through a neural network during training.

The learned hypothesis may **fit** the training data and the outliers (**noise**) very well but fails to **generalize** test data.

Dealing with the Overfitting Problem

L2 regularization

Dropout regularization

- Regularization penalizes big weights in addition to the overall cost function.
- The weight decay value determines how dominant regularization is during gradient computation.
- A big weight decay coefficient implies a big penalty for big weights.
- Here, C is regularized cost function, C_0 is the original cost function and λ is the weight decay coefficient.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

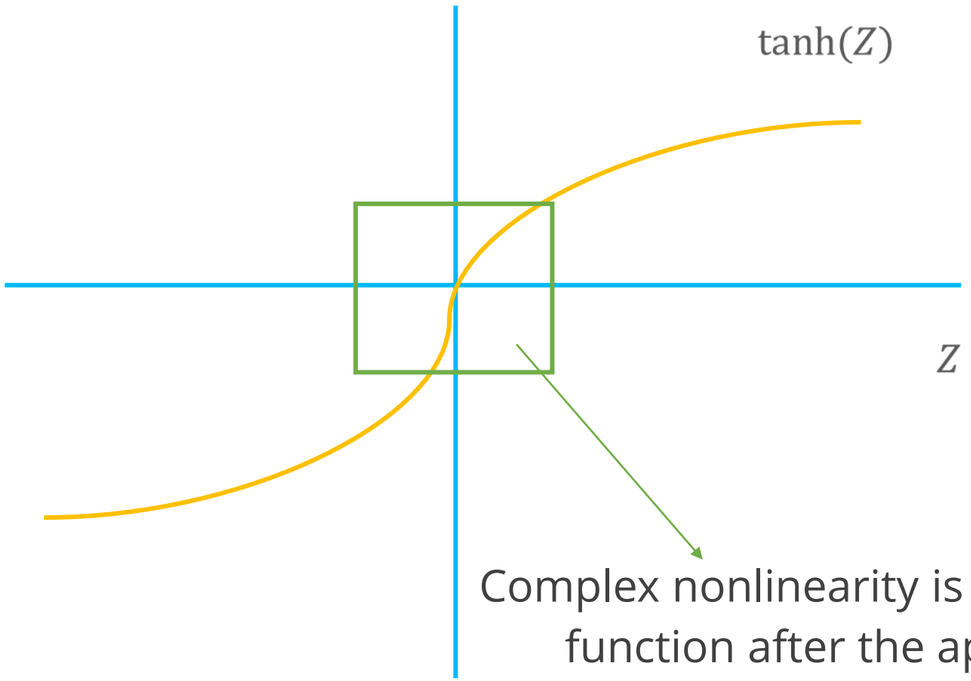
Regularization term Squared weights

Dealing with the Overfitting Problem

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

L2 regularization

Dropout regularization

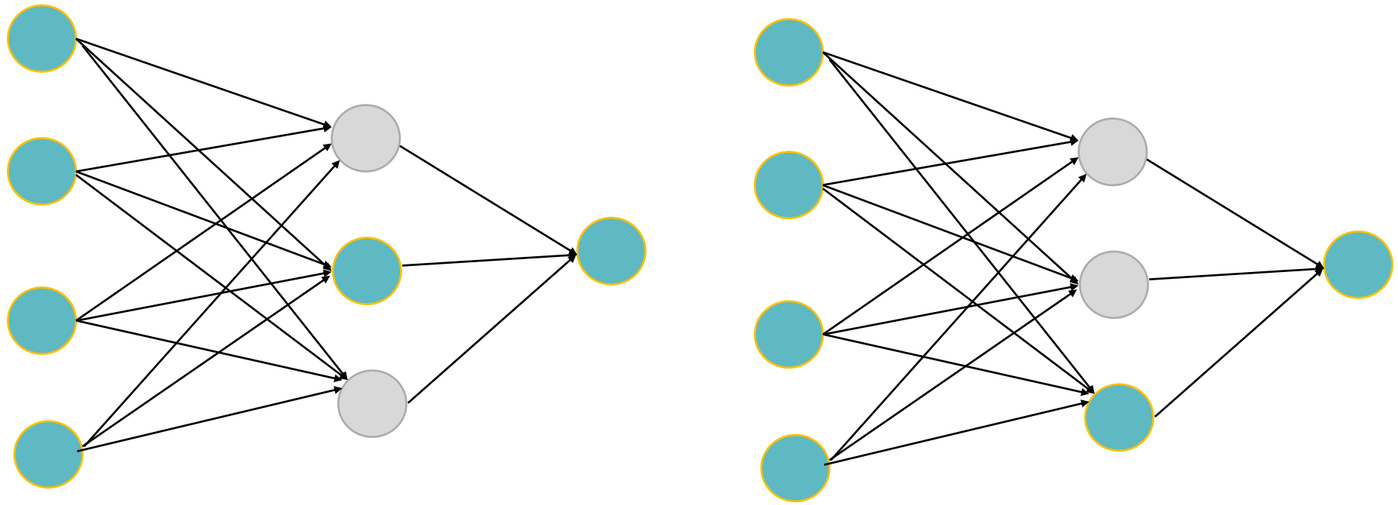


Complex nonlinearity is reduced to a linear function after the application of L2 regularization, thus reducing the complexity due to hidden layers.

Dealing with the Overfitting Problem

L2 regularization

Dropout regularization



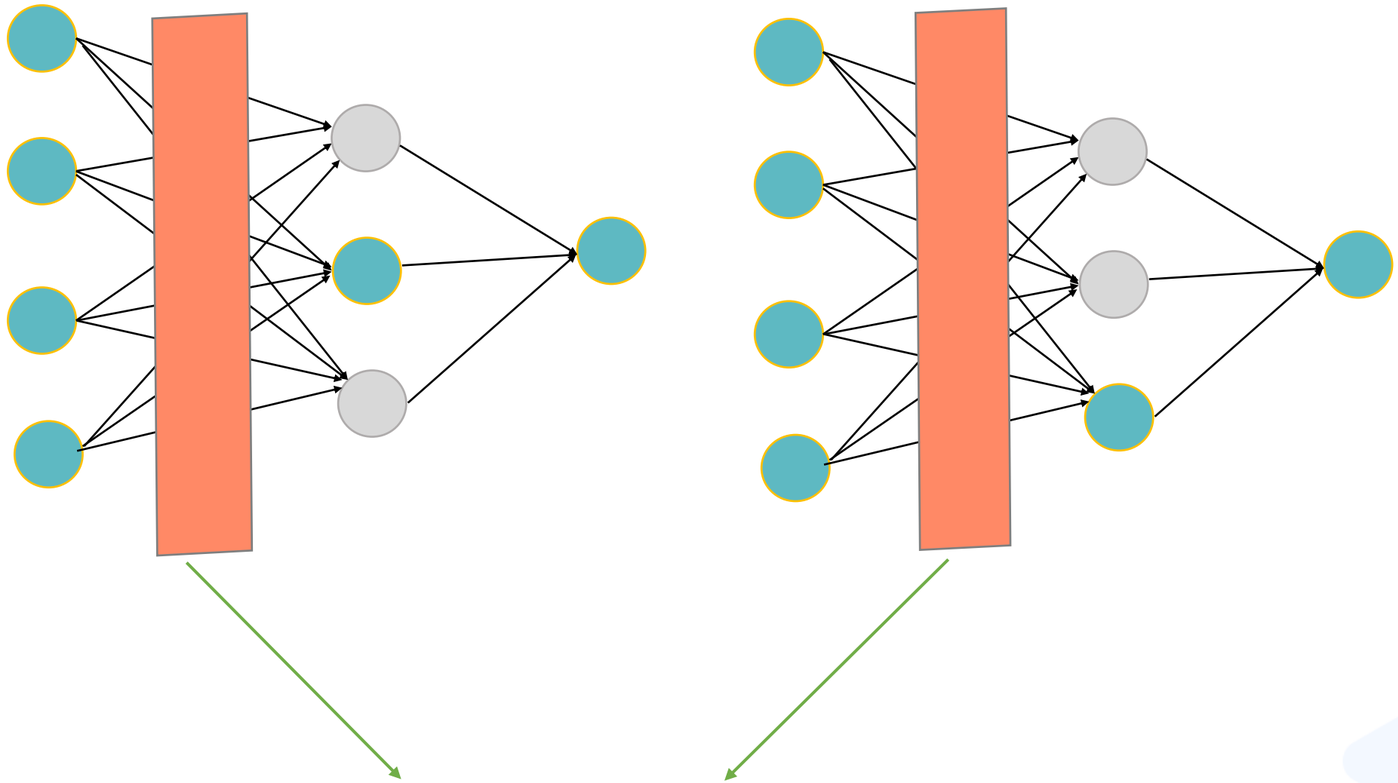
- Randomly drop units (along with their connections) during training.
- Each unit is retained with a fixed probability p, independent of other units.
- $0 < p < 1$
- The hyper-parameter p has to be chosen (tuned).
- In dropout regularization, during training, a fraction of the weights are randomly set to zero.

Dealing with the Overfitting Problem

The goal is to prevent overfitting and improve the generalization capability of the neural network by randomly deactivating units during training.

L2 regularization

Dropout regularization



Dropout layers

Dropout Experiment

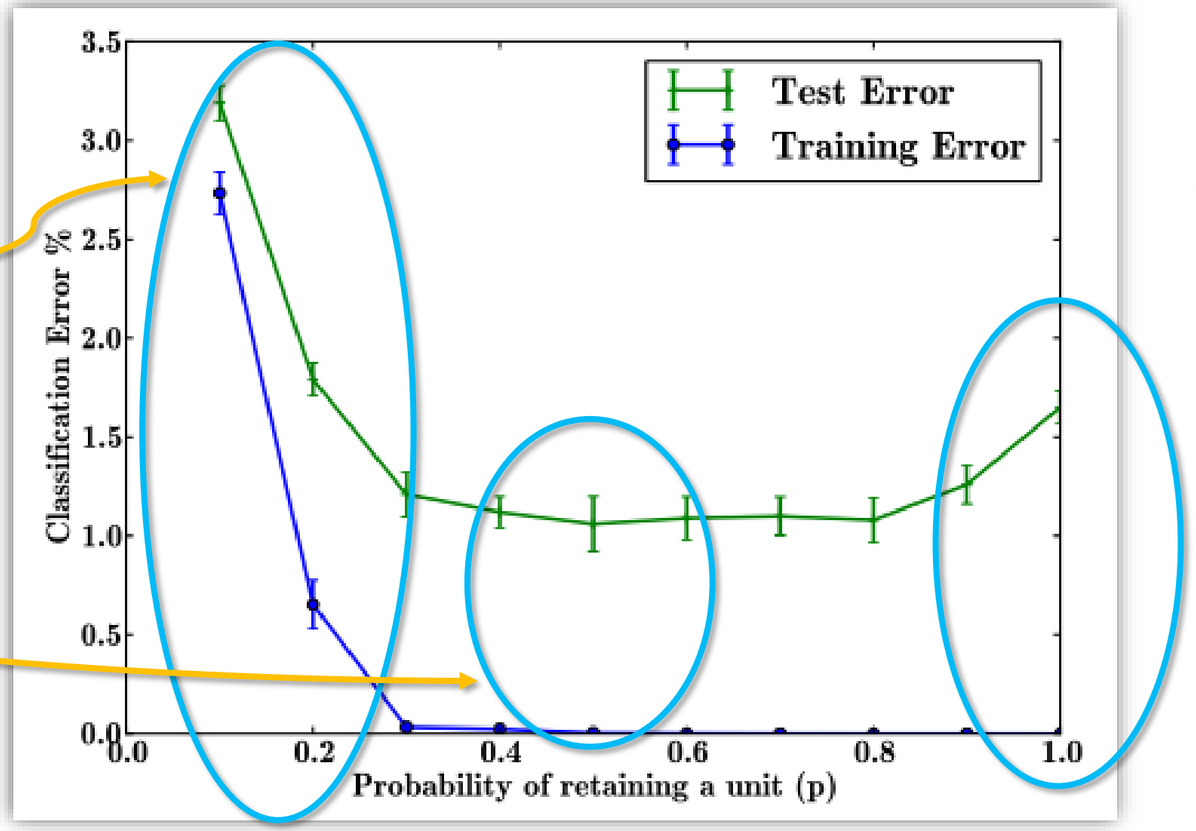
An architecture of 784-2048-2048-2048-10 is used on the MNIST dataset. The dropout rate p was changed from a small number (most units drop out) to 1.0 (no dropout).

High rate of dropout ($p < 0.3$)

- Underfitting
- Very few units are turned on during training

Best dropout rate ($p = 0.5$)

- Training error is low
- Test error is low



No dropout ($p = 1.0$)

- Training error is low
- Test error is high

Key Takeaways

- A deep neural network (DNN) is an artificial neural network that has multiple hidden layers between the input and output layers.
- A DNN provides better calculation of output probabilities and more accurate predictions.
- Loss functions in DNNs act as error functions, estimating the model's error or loss.
- Normalizing the data is an important step when building DNNs using TensorFlow as it ensures that the model can learn effectively from the input features.

Knowledge Check

Knowledge Check

1

What is the task of a loss function in a DNN?

A. To estimate the model's error or loss

B. To calculate the probability of an output

C. To pass input to multiple hidden layers

D. To adjust the weights of the neural network based on the error rate

Knowledge Check

1

What is the task of a loss function in a DNN?

A. To estimate the model's error or loss

B. To calculate the probability of an output

C. To pass input to multiple hidden layers

D. To adjust the weights of the neural network based on the error rate

The correct answer is **A**

The task of a loss function is to estimate the model's error or loss and change the weights in the hidden layers in the network to reduce the loss in the next assessment.

Knowledge Check 2

How does DNN work?

A. By passing input through one hidden layer

B. By passing input through multiple hidden layers

C. By using decision trees

D. By using linear regression

Knowledge Check 2

How does DNN work?

A. By passing input through one hidden layer

B. By passing input through multiple hidden layers

C. By using decision trees

D. By using linear regression

The correct answer is **B**

DNN works by passing input through multiple hidden layers that allow for better calculation of the probability of every single output.

Knowledge Check 3

What is the purpose of regularization in building deep neural networks?

A. To make the model more complex

B. To prevent overfitting

C. To speed up the training process

D. None of the above

Knowledge Check 3

What is the purpose of regularization in building deep neural networks?

A. To make the model more complex

B. To prevent overfitting

C. To speed up the training process

D. None of the above

The correct answer is **B**

The purpose of regularization in building deep neural networks is to prevent overfitting, which occurs when the model becomes too complex and fits the training data too well.

Lesson-End Project: MNIST Image Classification

Problem statement: The MNIST dataset is widely used for image classification. However, while validating the same, researchers found out that the classification model was overfitting, as it was not giving acceptable accuracy on the testing data.
Use the mnist_test.csv and mnist_train.csv for model optimization (using dropout layers). Also, you will have to use one-hot encoding for training and testing labels.

Objective:
Optimize a neural network-based classification model using dropout regularization such that the p-value is 0.70 for input and hidden layers

Access: Click on the **Lab** tab on the left side of the LMS panel. Copy the generated username and password. Click on the **Launch Lab** button.
On the new page, enter the username and password you copied earlier into the respective fields. Click **Login** to start your lab session. A full-fledged Jupyter lab opens, which you can use for your hands-on practice and projects.

LESSON-END PROJECT

Thank You