Deep Learning with Keras and TensorFlow

Deep Neural Networks

Learning Objectives

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

- Understand the architecture and functionality of deep neural network (DNN)
- Interpret and calculate the effects of different types of regression losses, such as MAE and MSE, on the performance of neural network models
- Evaluate different loss functions used in deep neural networks
- Demonstrate the process of forward and backward propagation in deep neural networks

Business Scenario

A financial services company aims to enhance its loan application process by improving the risk assessment model. The company plans to utilize a deep neural network (DNN) to analyze customer data and determine the likelihood of loan defaults.

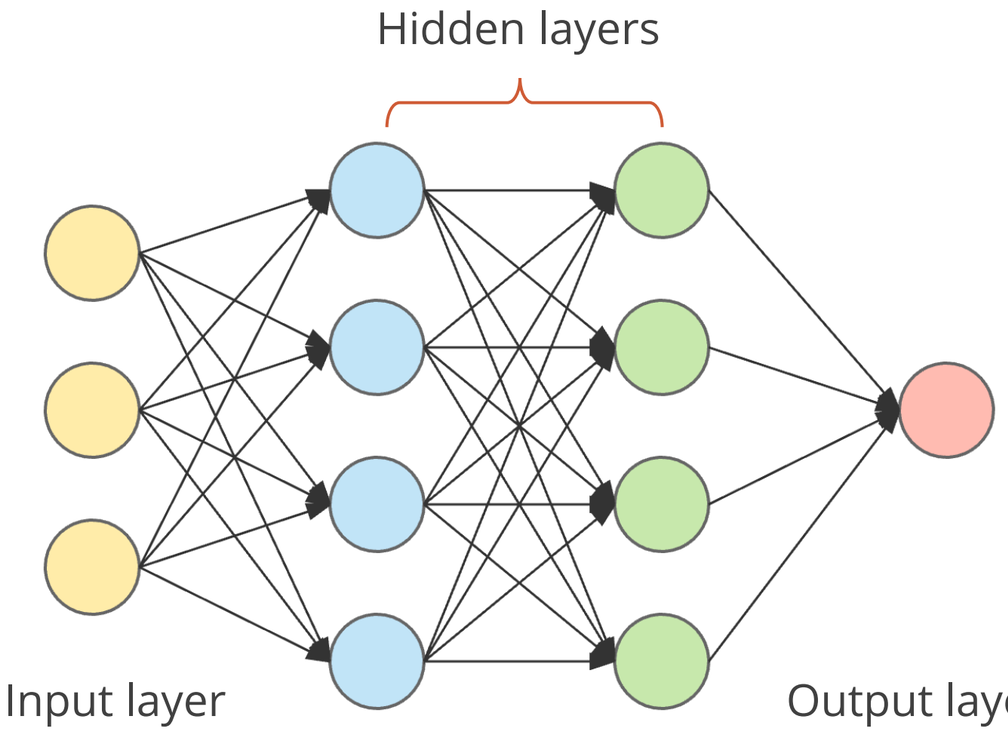
With a rich dataset containing various features such as income, credit score, and employment history, the company intends to construct a DNN and normalize the data to facilitate effective learning. Additionally, the company will employ loss functions to estimate the model's error and adjust the network accordingly.

Furthermore, it plans to use the TensorFlow Playground as a tool to visually comprehend the neural network and make necessary adjustments. Through the implementation of a DNN, the company seeks to enhance the accuracy of its risk assessment model, decrease defaults, and ultimately provide improved 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.

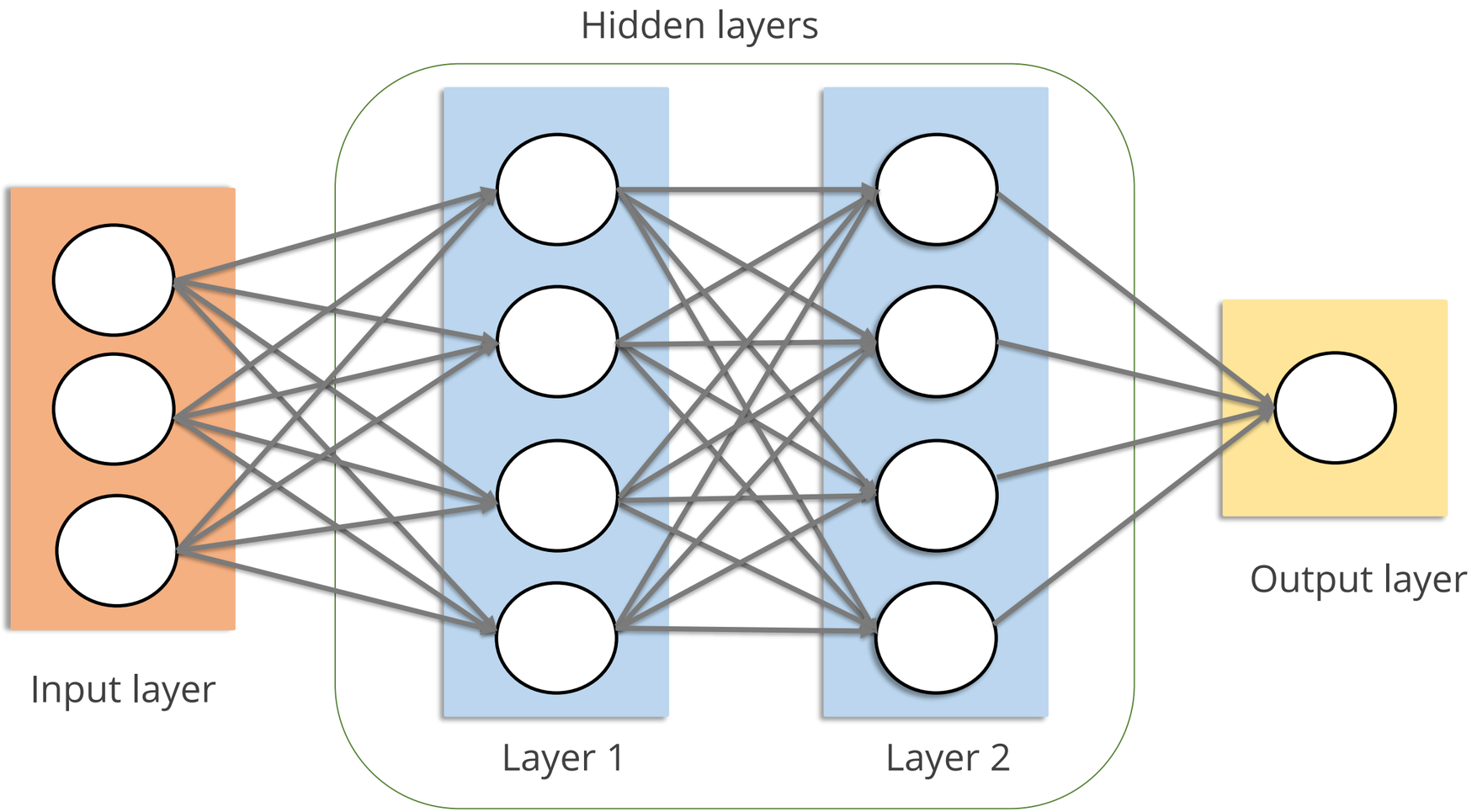


The **Depth** or number of hidden layers is a key factor that distinguishes deep neural networks (DNNs) from traditional neural networks (NNs), enabling DNNs to outperform NNs in various tasks.

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

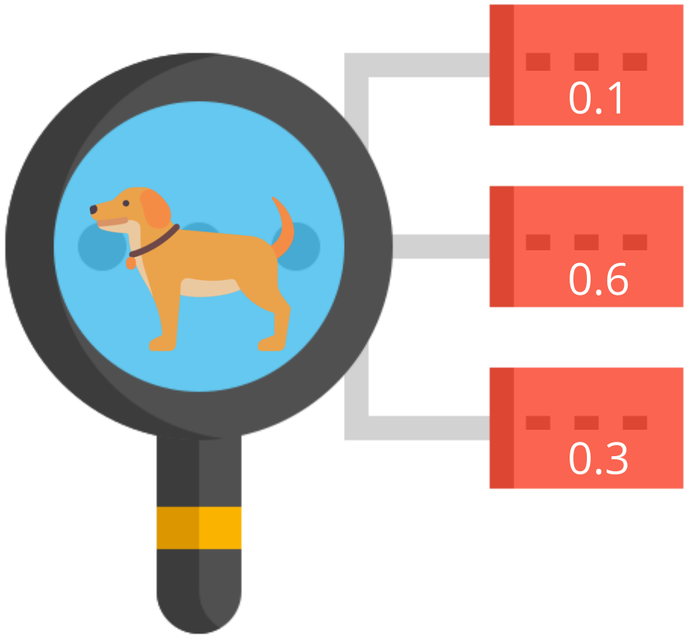
Deep Neural Network

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 to a computer

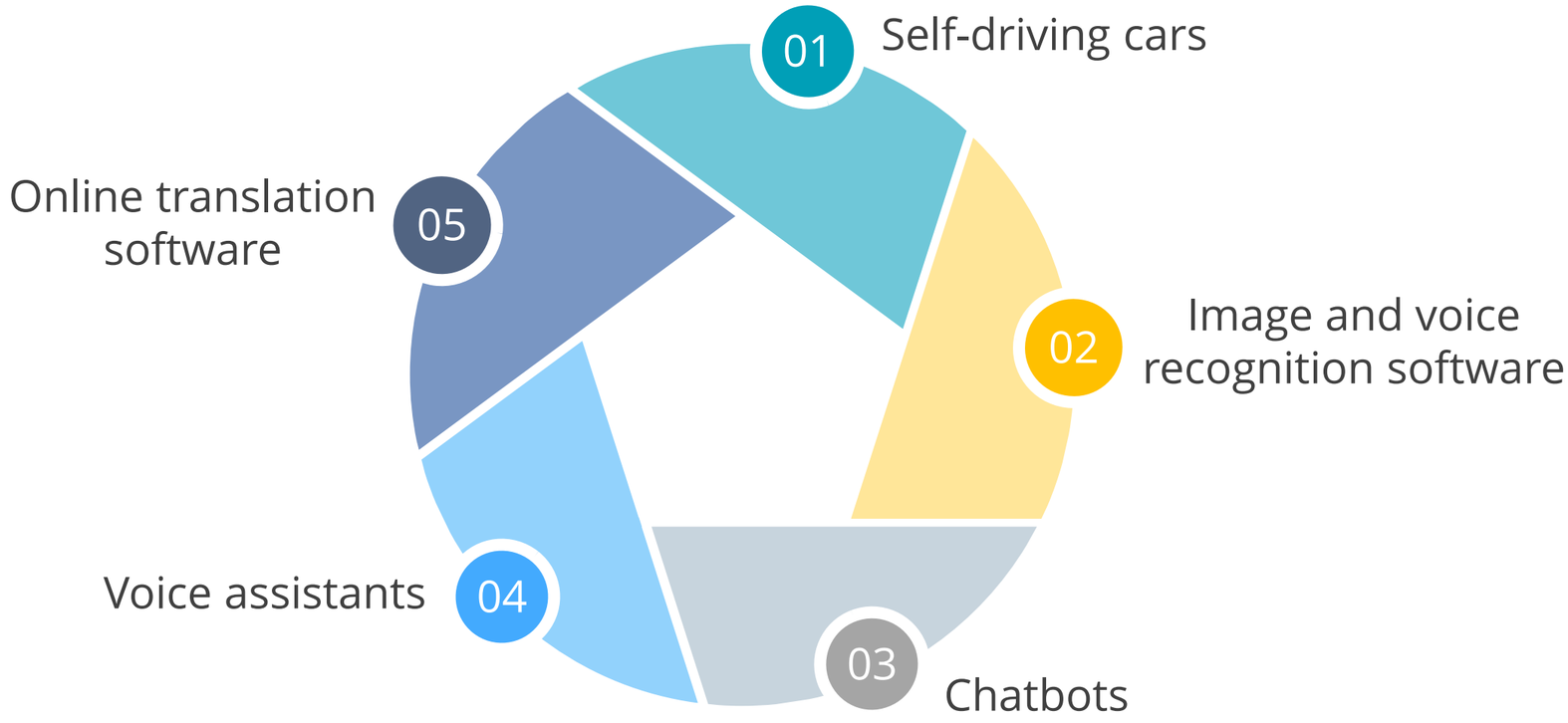
DNN identifies the sound and sorts it into different categories.

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

Benefits of DNN

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

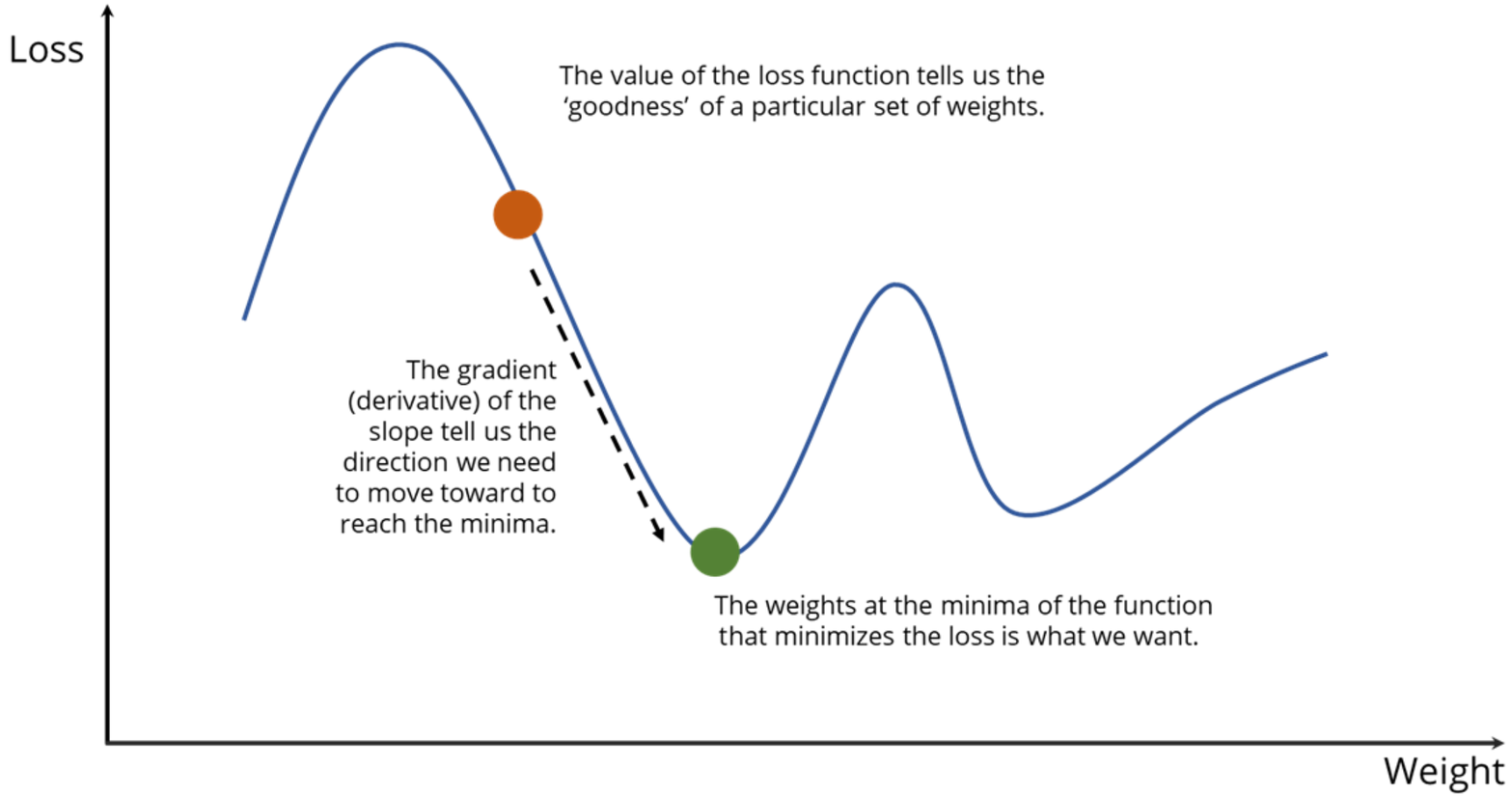
It has accelerated the development of technologies such as:



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: 0 to a horse and 1 to the human

Classify all images of horses and humans in this manner

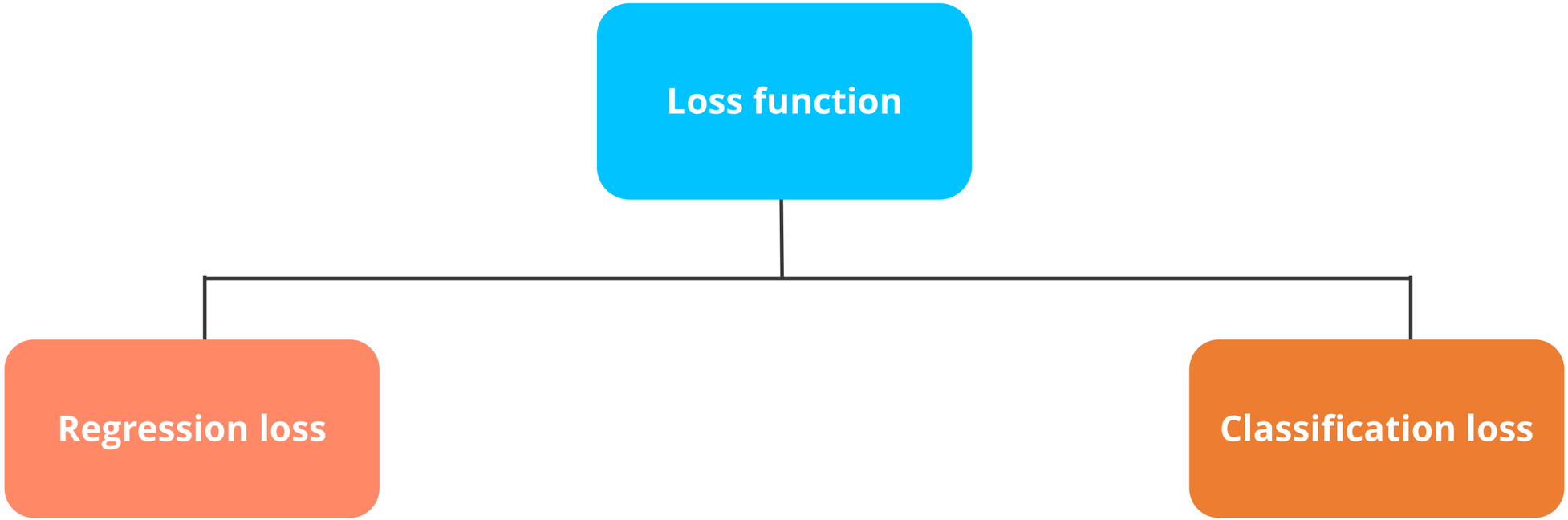
If the model receives an image of a horse and identifies the output as 0.25, the difference in the model's prediction and the label is:
0.25 - 0 = 0.25

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.

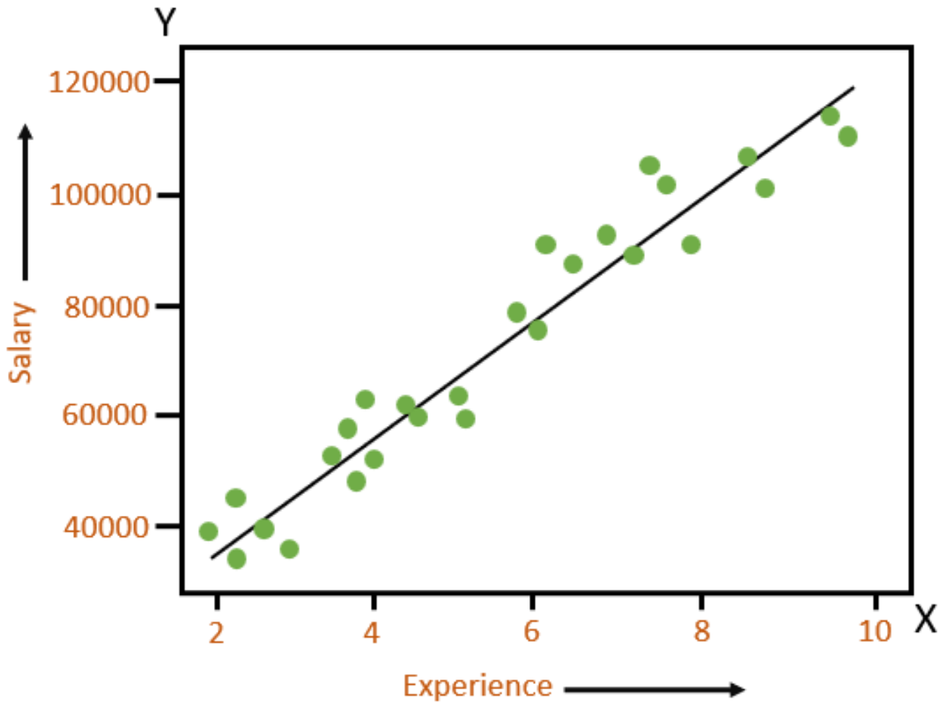
Loss Function and Its Major Categories

Loss functions are fundamental tools in evaluating deep learning models, and they can be broadly categorized into



Regression Loss

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



The output in such a model provides a value that could be, for example, the salary of an employee.

Types of Regression Loss

The two main types of regression losses are

Mean Absolute Error

MAE measures the average absolute difference between the predicted and actual 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
n	→	The total number of observations

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

This code snippet calculates the mean absolute error (MAE) between the actual and predicted values.

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.

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 a standard loss function 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

This code snippet calculates the MSE between the actual and predicted values.

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.

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

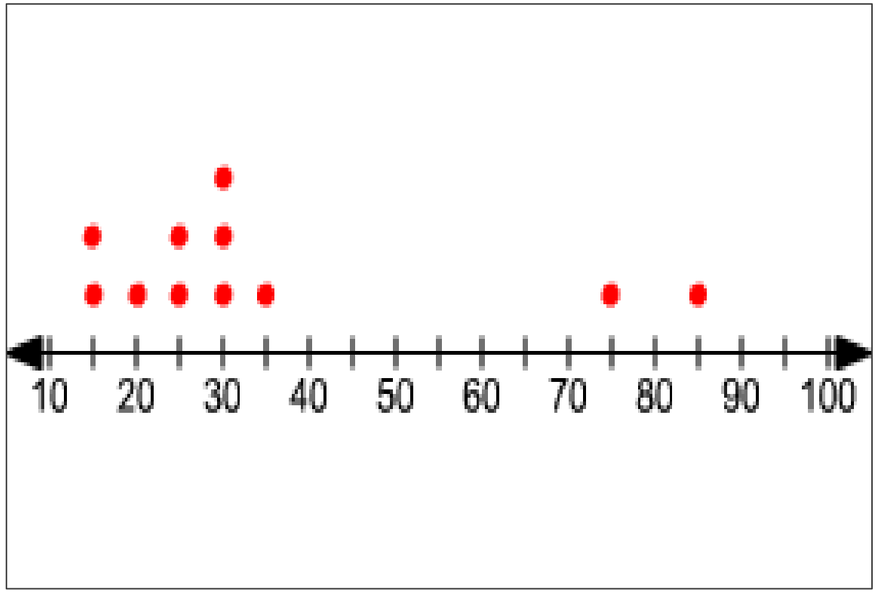
MSE vs. MAE

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

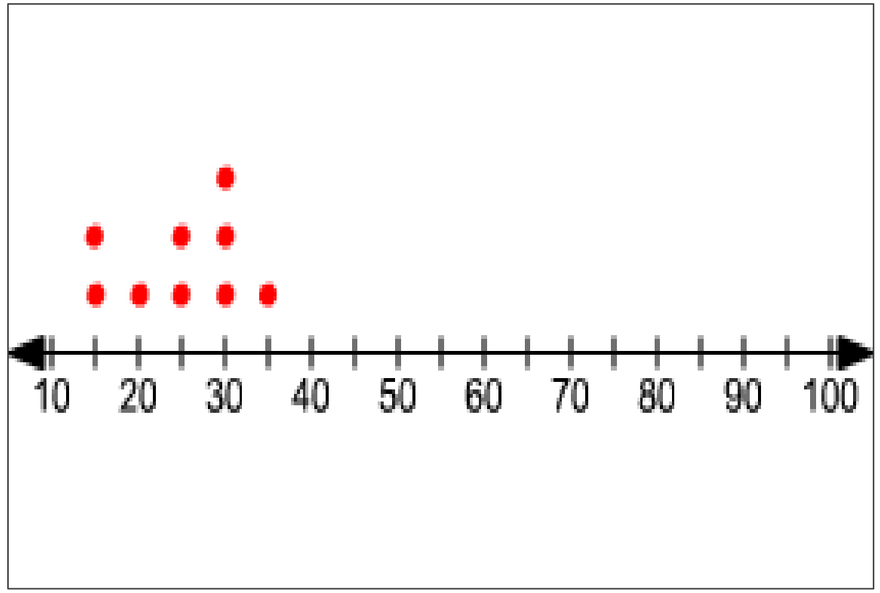
MSE vs. MAE

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.

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.

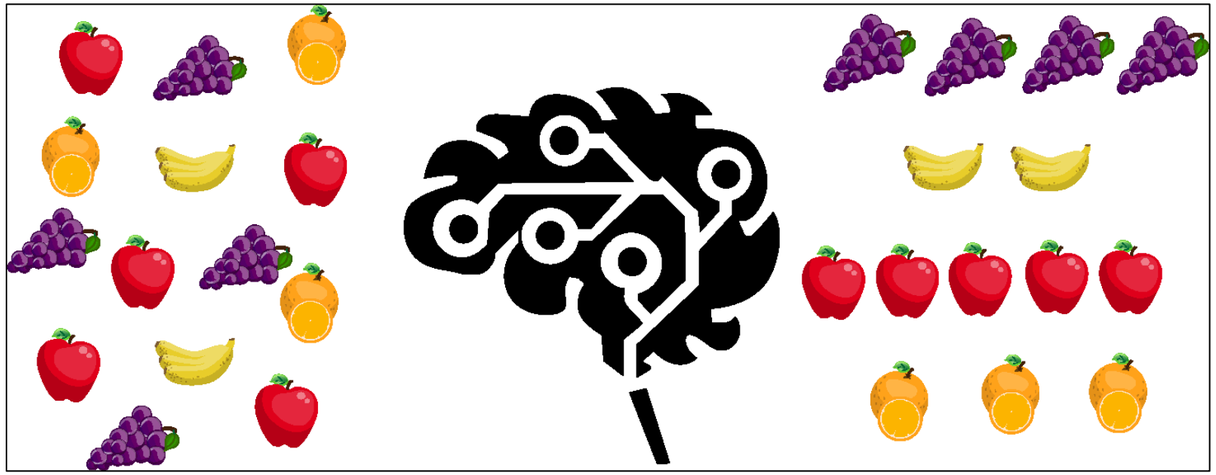
Example

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.



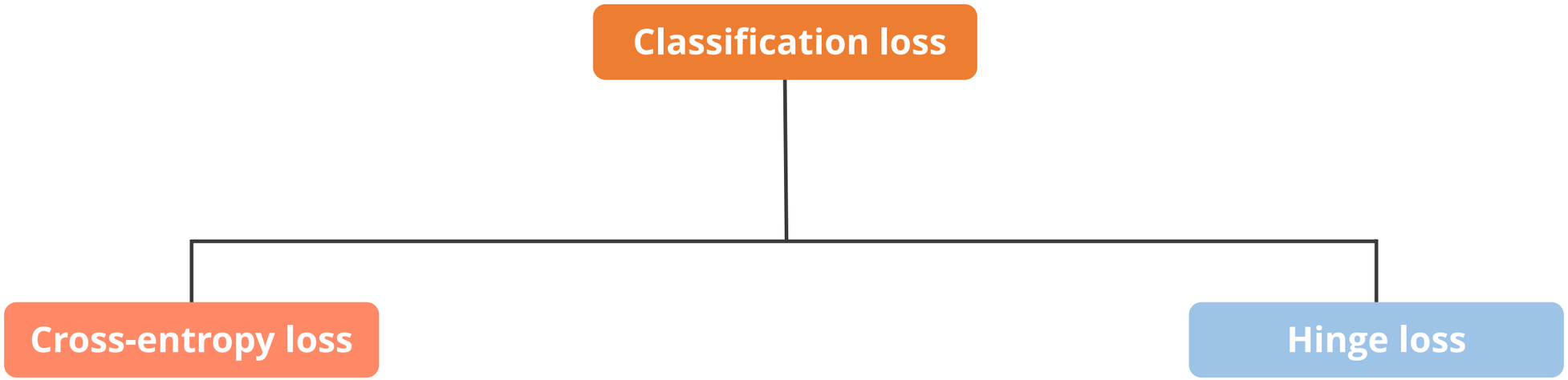
Example

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.

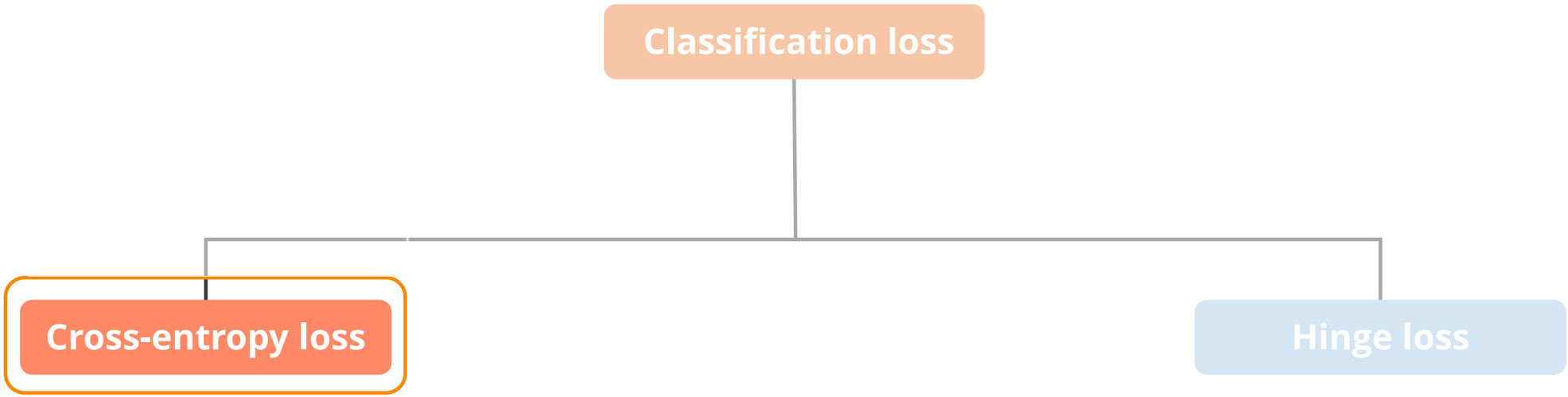
Types of Classification Loss

The different types of classification losses are:



Types of Classification Loss: Cross-entropy loss

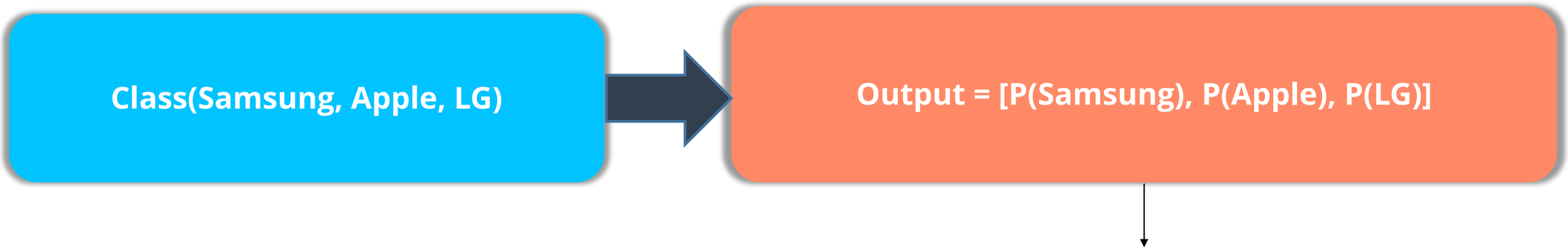
The different types of classification losses are:



Cross-Entropy Loss

Cross-entropy is a mathematical measure used to quantify the difference between two probability distributions

Consider a classification problem with three classes: Samsung, Apple, and LG.

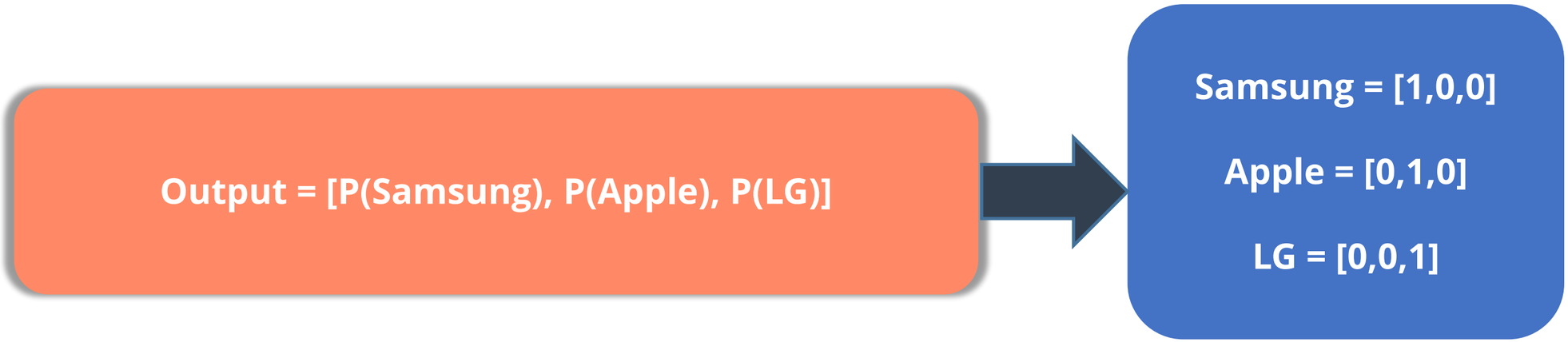


It represents the predicted probabilities of the input belonging to each class.

The class with the highest probability is the winner.

Cross-Entropy Loss

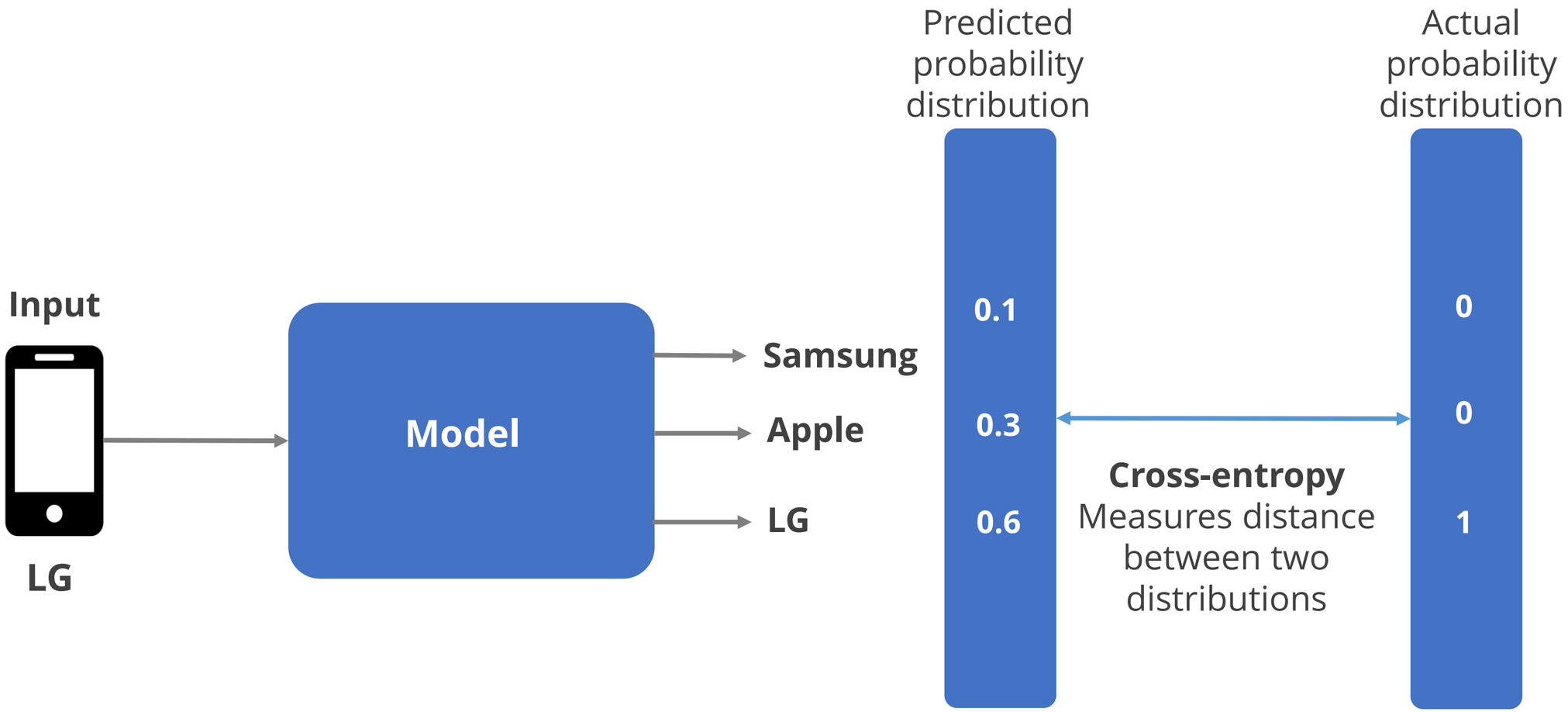
If the predicted probability distribution is not close to the actual value, the model adjusts its weight.



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

Binary cross-entropy loss

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 values 0 and 1 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])+ (1-actual[i]) *
log(1e-15 + (1 - 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

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

- 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

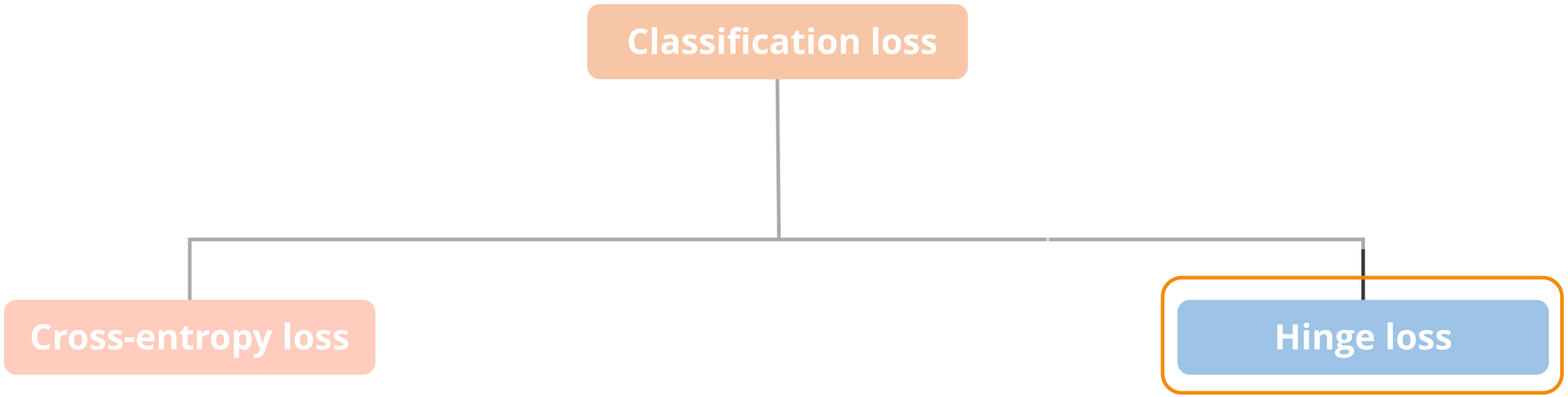
Overconfident wrong prediction occurs when MSE or MAE is used in classification, especially during the training phase.

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.

Types of Classification Loss: Hinge loss

The different types of classification losses are:



Hinge Loss

Hinge loss is a loss function utilized within machine learning to train classifiers. It is optimized to increase the margin between data points and the decision boundary.

- It is mainly used for maximum margin classifications.
- It ensures maximum margin by penalizing misclassified predictions and those correctly classified but too close to the decision boundary.
- It is a commonly used loss function in machine learning, particularly for training support vector machines (SVMs).

Hinge Loss

Hinge loss is defined for a binary classification problem, where the target label y is +1 or -1. The hinge loss for a single prediction is denoted by:

Equation

$$\text{Hinge Loss} = \max(0, 1 - y \cdot f(x))$$

Where:
y is the true label of the data point (+1 or −1)
$f(x)$ is the predicted score (decision function) for the data point x.

Hinge Loss: Target Label vs. Predicted Value

Target label(y)

This is the actual class label for the data point. In binary classification with hinge loss, (y) is +1 or -1.

Predicted value (f(x))

This is the output from the classifier. For linear classifiers like support vector machines (SVMs), this is often the raw output before applying a threshold to make a binary decision.

Predicted value range:

f(x) can take any real number.

For example:

1. If the classifier is confident that a sample belongs to the positive class (+1), it might output a large positive number (for example: f(x) = 2.5).
2. If the classifier is confident that a sample belongs to the negative class (-1), it might output a large negative number (for example: f(x) = −3.0). If the classifier is unsure, the output might be close to zero (for example: f(x) = 0.1).

Scenarios In Hinge Loss

In the context of hinge loss for binary classification, prediction scenarios can be categorized into three main cases based on how the predicted value (f(x)) compares to the true target (y). The first one is as follows:

The prediction is correct, which occurs when $f(x) \geq 1.0$.

In the first case, for example, when $y=1.2$, the output of $1-(f(x)\cdot y)$ will be $1-(1*1.2)=1-1.2=-0.2$.

The loss is then calculated as $\max(0,-0.2)=0$. Therefore, the loss is zero for all correct predictions, even if they are *too correct*. In the *too correct* situation, the classifier is extremely confident that the prediction is correct.

Scenarios In Hinge Loss

The second scenario is as follows, where the target y = 1:

The prediction is incorrect, which occurs when f(x) < 0 (because the sign swaps from positive to negative).

In the second case, for example when y = -0.5, the output of the loss equation will be 1 — (1 * -0.5) = 1 — (-0.5) = 1.5, and hence the loss will be max(0, 1.5) = 1.5. Wrong predictions are penalized significantly by the hinge loss function.

Scenarios In Hinge Loss

The third scenario is as follows, where the target y = 1:

The prediction is incorrect, but the results are close to correctness($0.0 \leq f(x) < 1.0$).

The output function will be 1 — (1 * 0.9) = 1–0.9 = 0.1. Loss will be max(0, 0.1) = 0.1. The prediction is approaching correctness, indicated by a small but non-zero loss.

Properties of Hinge Loss

Max-margin

Hinge loss encourages a large margin between classes. Penalizing correct predictions but close to the decision boundary helps create a margin separating the classes as widely as possible.

Non-differentiability

The hinge loss function is not differentiable at $y \cdot f(x) = 1$. However, this is handled in practice using sub-gradient methods.

Regularization

Hinge loss is often used with regularization terms (like L2 regularization) to prevent overfitting and to promote simpler models.

Sensitive to outliers

Hinge loss can be sensitive to outliers because it penalizes large margin violations heavily.

Squared Hinge Loss

It extends the traditional hinge loss used in machine learning, particularly in support vector machines (SVMs). It modifies the hinge loss by squaring the hinge loss term, which leads to smoother optimization and potentially better performance.

Equation

$$\text{Squared Hinge Loss} = \max(0, 1 - y \cdot f(x))^2$$

Where:
y is the true label of the data point (+1 or −1)
$f(x)$ is the predicted score (decision function) for the data point x

Properties of Squared Hinge Loss

Smooth penalty

The squaring of the hinge loss term results in a smoother penalty for misclassifications.

Stronger penalty for misclassifications

Incorrect predictions are penalized more severely than traditional hinge loss.

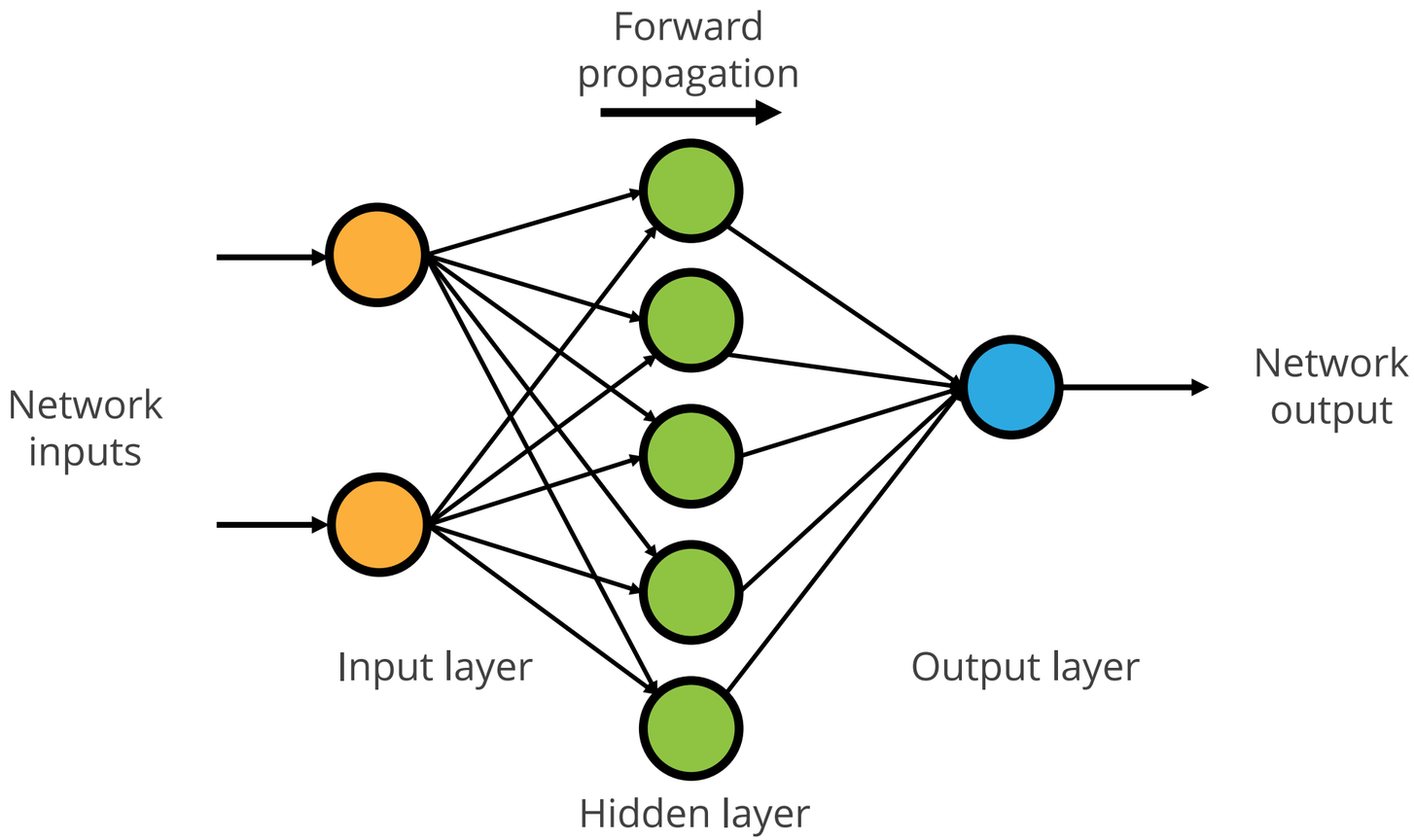
Enhanced gradient descent

The squared term improves the gradient properties, aiding in stable and efficient convergence during training.

Forward Propagation in DNN

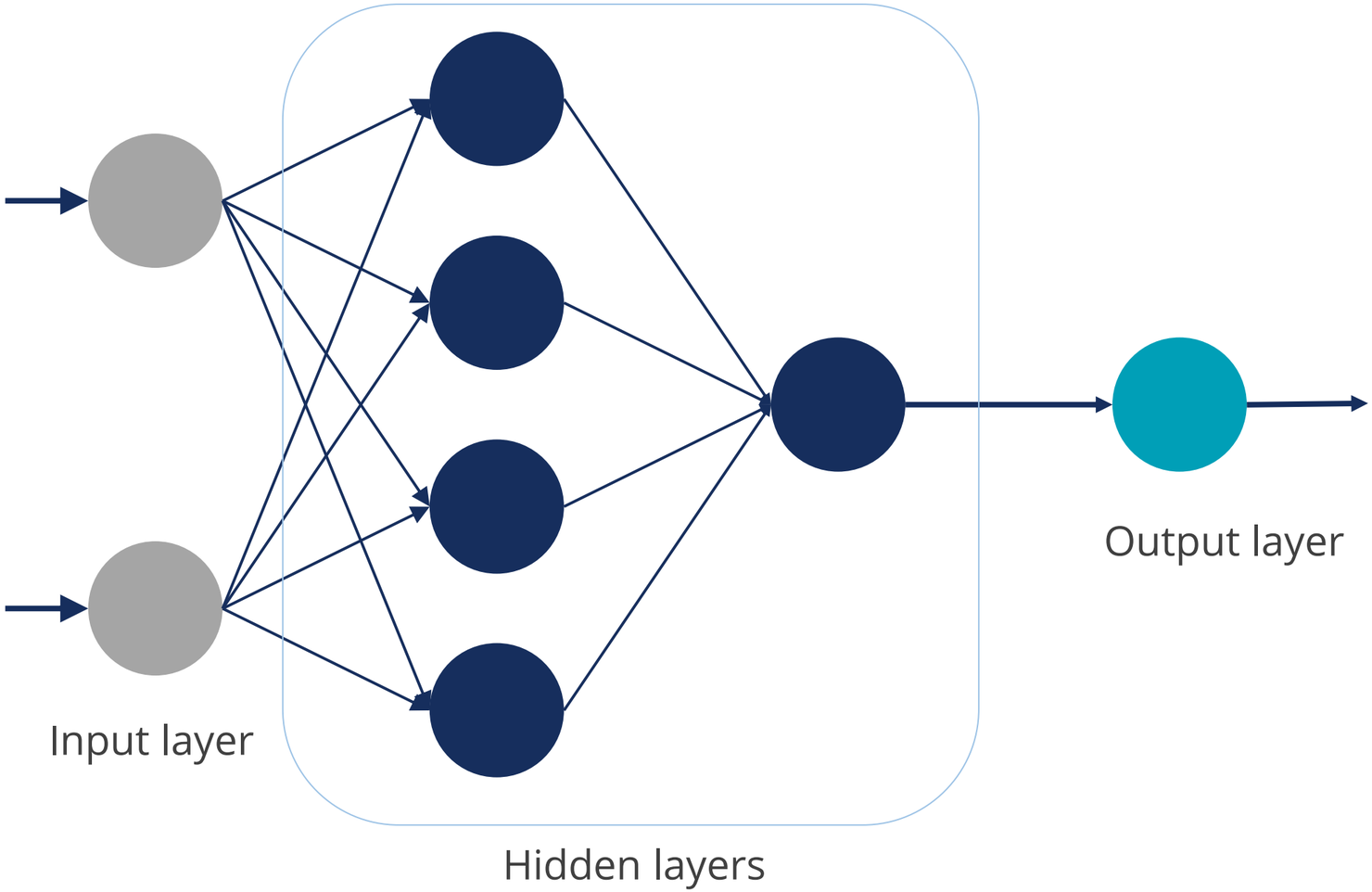
Forward Propagation

Forward propagation involves feeding input data through a neural network to produce an output. The data passes through hidden layers, where each layer processes it using an activation function before passing it to the next layer. This forward flow ensures data does not circulate back, preventing non-output states.



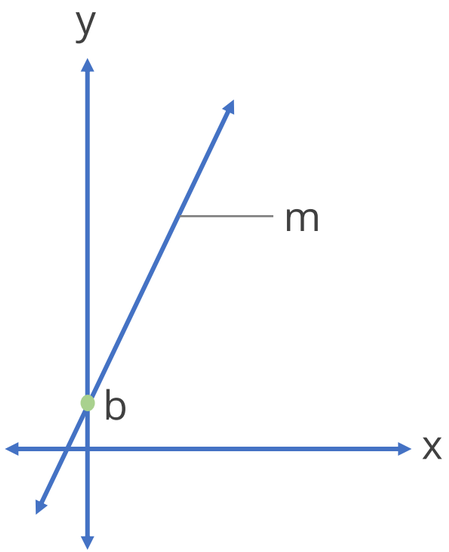
Forward Propagation

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



Working of 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.

Assisted Practice

Let’s understand the concept of forward propagation using Jupyter Notebooks.

- 4.04 Example: Working on Forward Propagation

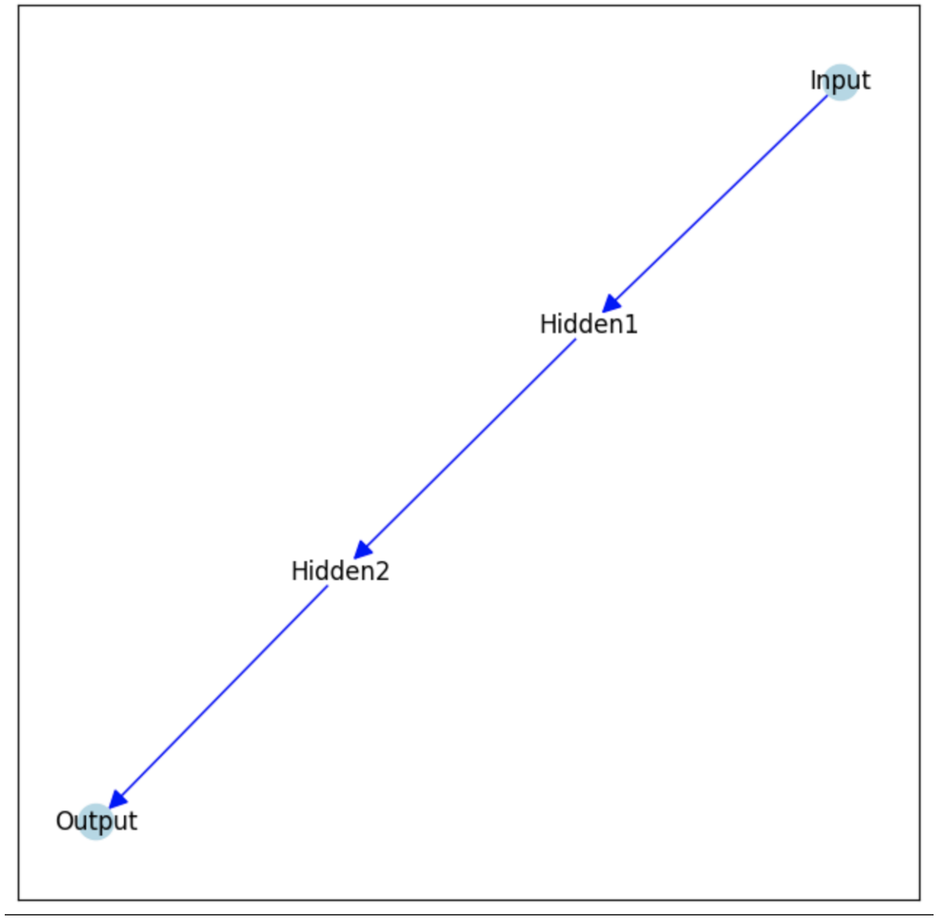
Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

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.

Loss Calculation in 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: Uses

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

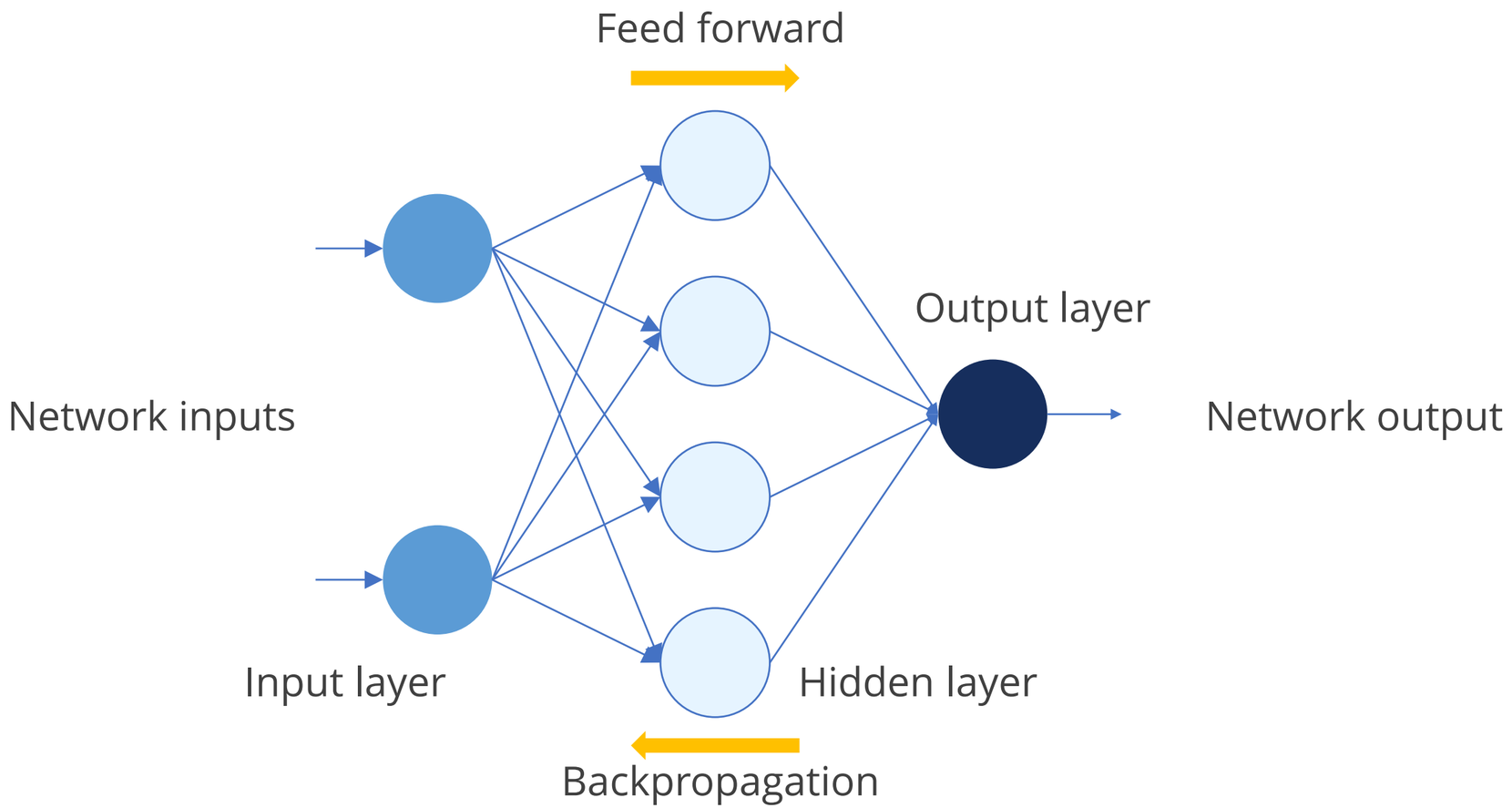
To minimize error, we need to optimize the weights by examining how the error changes with respect to the weights. This requires finding the derivative of the error with respect to the weight, known as the gradient.

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

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

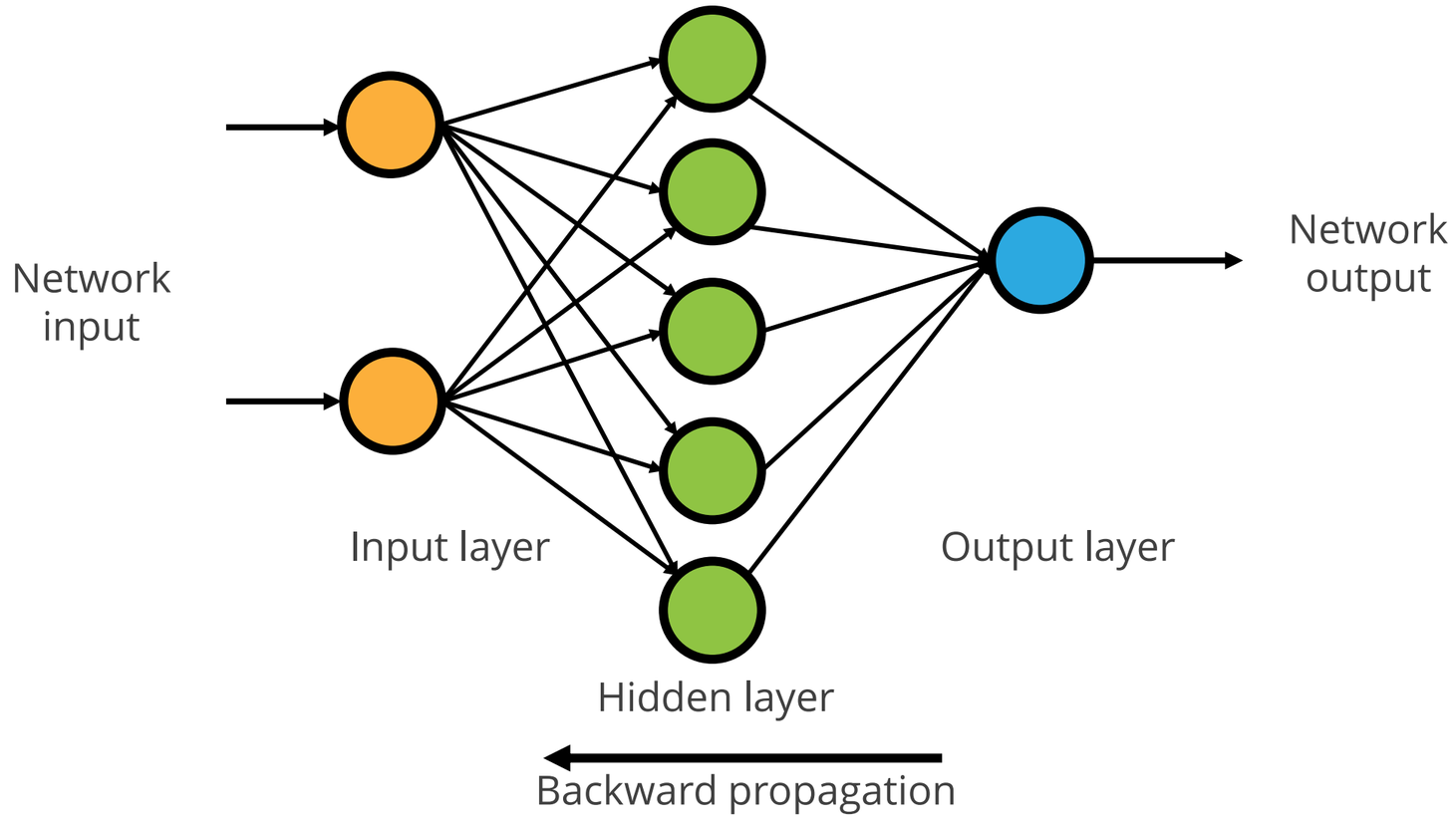
Backward Propagation: Uses

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 will lower the predicted values. This adjustment helps reduce the loss for the input.

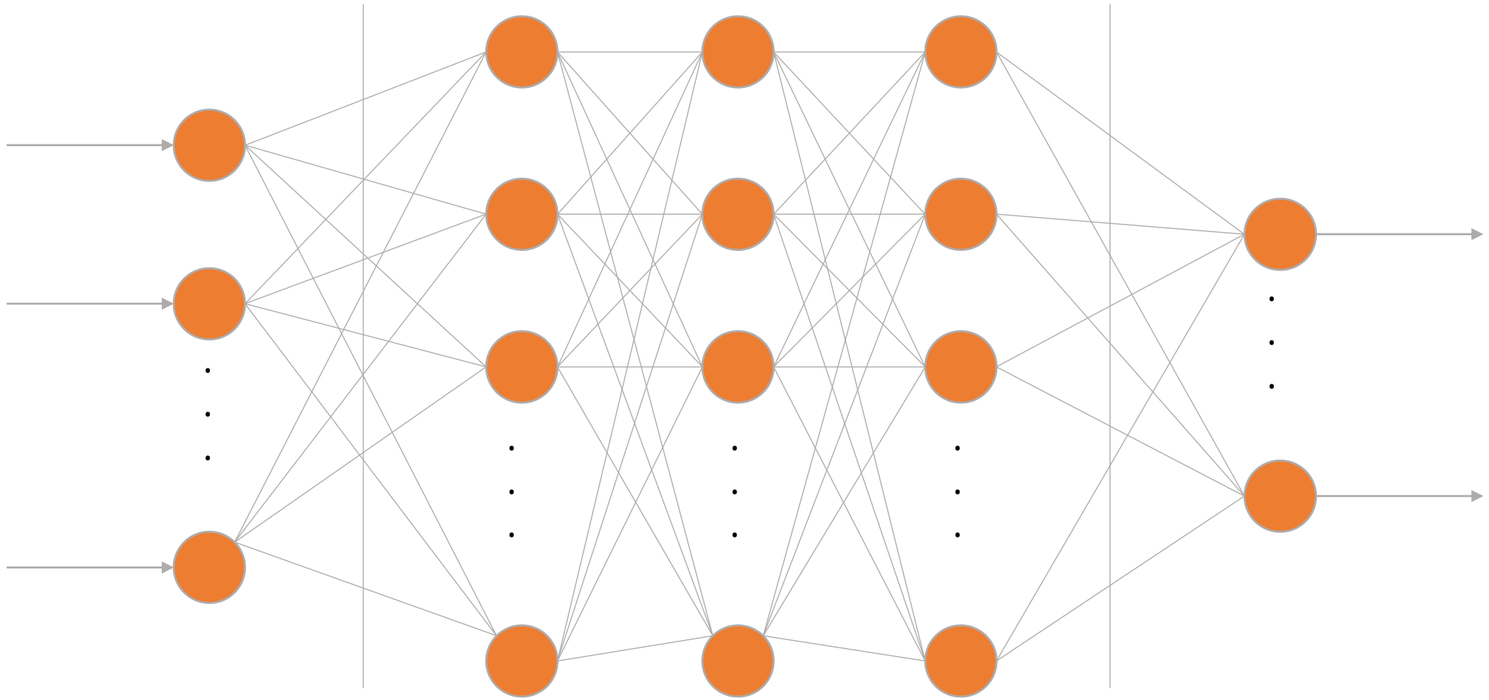


The algorithm travels backward through the network and adjusts the weights from right to left.

Regularization

Deep Neural Networks

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



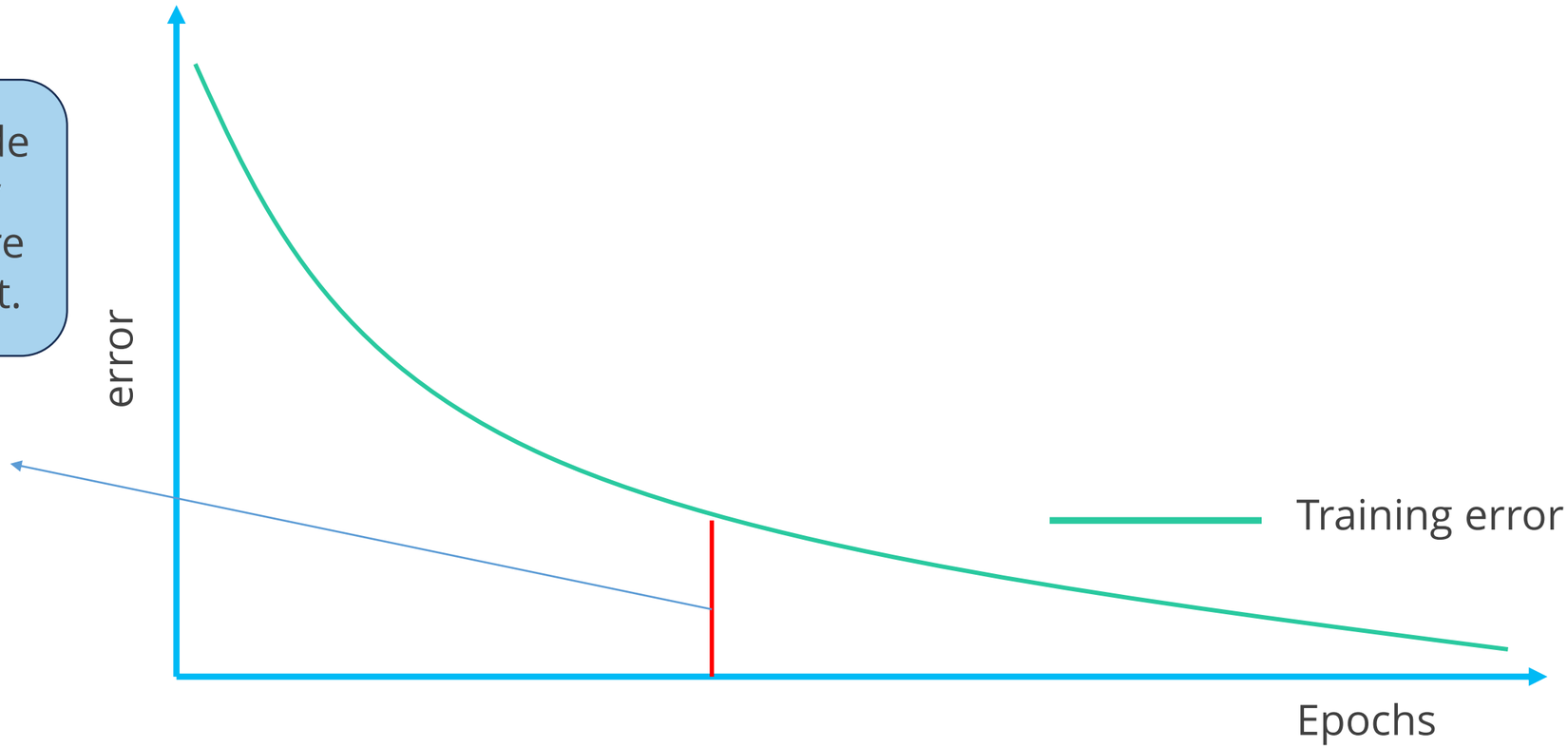
Input layer

Multiple hidden layers

Output layer

The Overfitting Problem

Early stopping rules provide guidance as to how many iterations can be run before the model begins to overfit.



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

The learned model may **fit** the training data, including its outliers (**noise**) too well but fails to **generalize** test data.

Regularization

Regularization is a method applied in machine learning and deep learning to avoid overfitting and enhance a model's ability to generalize. This technique introduces a penalty into the loss function during the training process

These are some of the commonly used regularization techniques:

L2 regularization

Dropout regularization

L2 Regularization

Weight decay, also known as L2 regularization, is a specific type of regularization that adds the squared magnitude of the weights to the cost function.

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

Regularization term

Squared weights

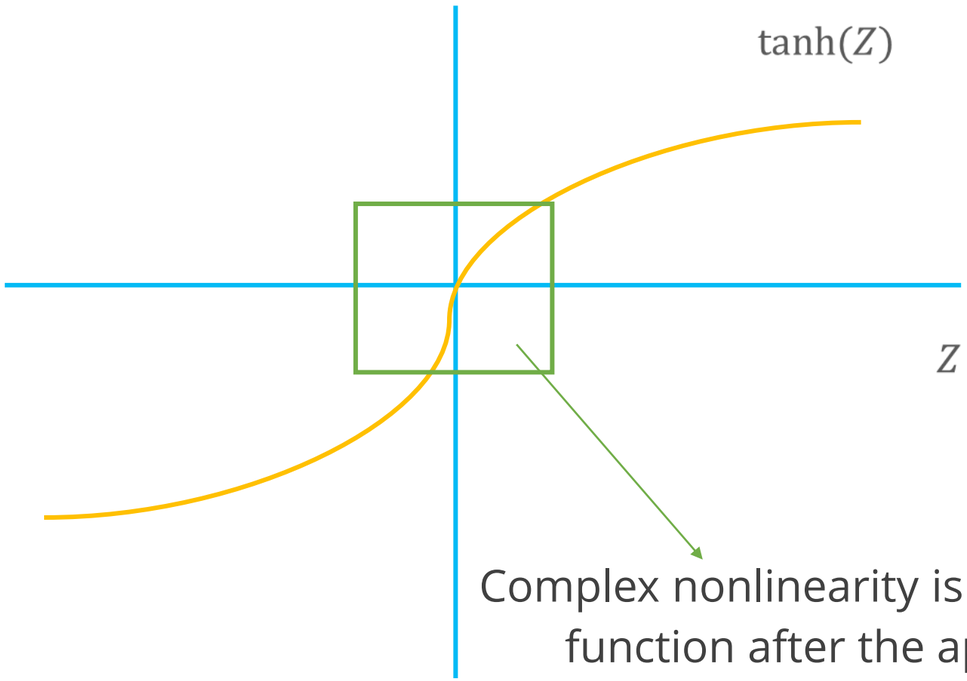
- Regularization penalizes large weights in addition to the overall cost function.
- The weight decay value determines how dominant regularization is during gradient computation.
- A large weight decay coefficient implies a large penalty for large weights.
- Here, C is regularized cost function, C_0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

L2 Regularization

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

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.

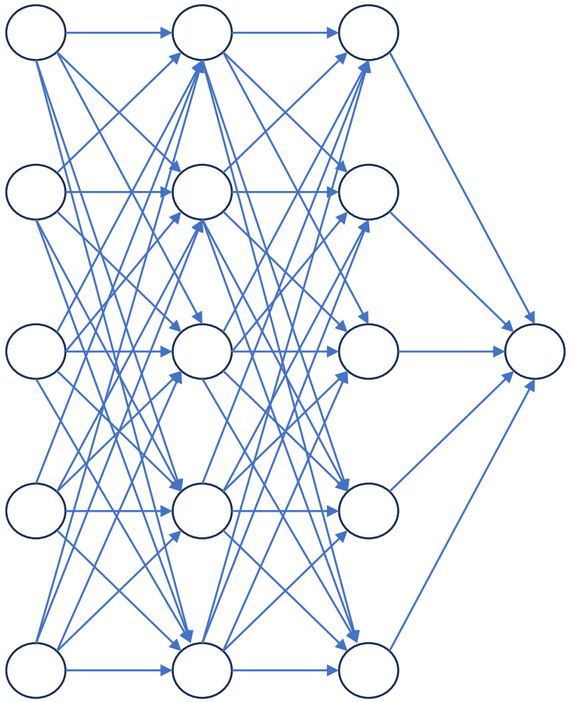
Dropout Regularization

Dropout is a technique used during training in neural networks to prevent overfitting by randomly dropping units (neurons) and their connections with a certain probability, p.

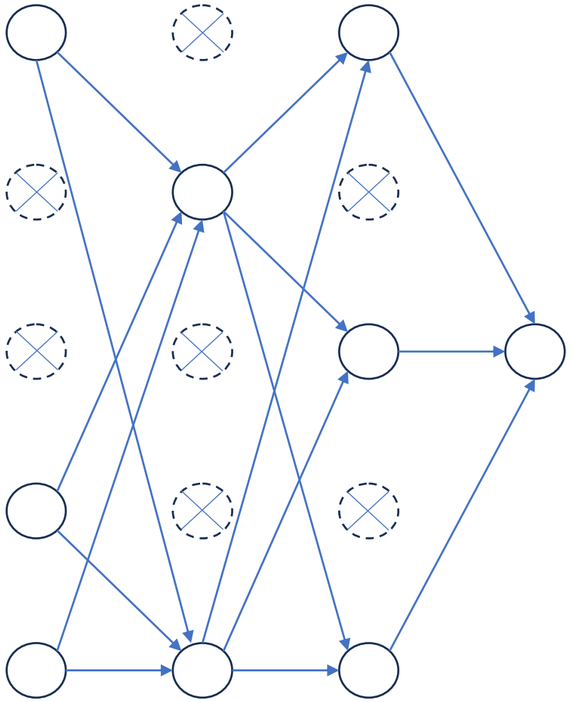
- Randomly drop units (along with their connections) during training.
- Each unit is retained with a fixed probability p, independent of other units.
- $0 < p < 1$
- The hyper-parameter p has to be chosen (tuned).
- In dropout regularization, during training, a fraction of the weights are randomly set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

Dropout Regularization

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



(a) Standard neural net



(b) After applying dropout

Activated neurons

Deactivated neurons

Image Reference: Alarfaj, F. K., Malik, I., Khan, H. U., Almusallam, N., Ramzan, M., & Ahmed, M. (2022). Credit card fraud detection using State-of-the-Art machine learning and deep learning algorithms. *IEEE Access*, *10*, 39700–39715. https://doi.org/10.1109/access.2022.3166891

Dropout Experiment

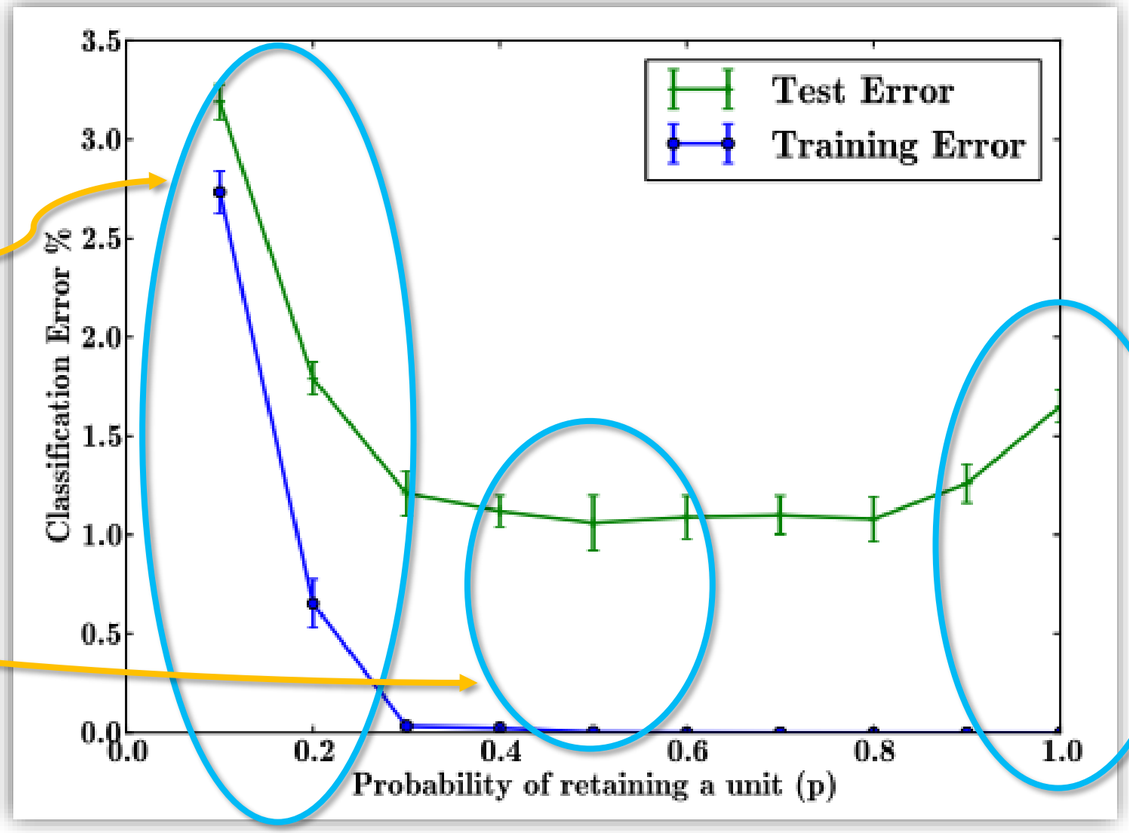
A Neural network with architecture of 784-2048-2048-2048-10 is used on the MNIST dataset. The dropout rate p was changed from a small number (most units drop out) to 1.0 (no dropout).

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.
- There are different types of Regularization techniques to reduce overfitting and improve model’s ability to generalize to unseen data

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.

Thank You