

INDEX

1. Introduction to Deep Learning

- 1.1 Overview of Machine Learning
- 1.2 Overview of Deep Learning
- 1.3 Machine Learning Vs Deep Learning
- 1.4 Application of Deep Learning
- 1.5 Biological neuron
- 1.6 Fundamentals of Neural Networks

2. Neural Network Basics

- 2.1 Activation Functions
- 2.2 Loss function and its types.
- 2.3 Why do we need loss functions
- 2.4 Different types of loss functions in deep learning & machine learning
- 2.5 Loss function Vs cost Functions.
- 2.6 Perceptrons and working process of perceptron
- 2.7 Artificial neural network and its type
- 2.8 Artificial neural network layers
- 2.2 Multi-layer Perceptron (MLP)
- 2.3 Training Neural Networks
- 2.4 Backpropagation Algorithm

3. Deep Neural Network Architectures

- 3.1 Feedforward Neural Networks
- 3.2 Convolutional Neural Networks (CNN)
- 3.3 Recurrent Neural Networks (RNN)
- 3.4 Long Short-Term Memory (LSTM)
- 3.5 Autoencoders and Variational Autoencoders

4. Optimization and Regularization

- 4.1 Gradient Descent and Its Variants

- 4.2 Adam Optimizer
- 4.3 Dropout and Batch Normalization
- 4.4 Weight Initialization
- 4.5 Vanishing Gradient Problem

5. Convolutional Neural Networks (CNN)

- 5.1 Image Representation and Preprocessing
- 5.2 Convolutional Layers and Pooling
- 5.3 Transfer Learning and Fine-tuning
- 5.4 Object Detection and Localization

6. Recurrent Neural Networks (RNN)

- 6.1 Sequence Modeling
- 6.2 Natural Language Processing (NLP)
- 6.3 Attention Mechanisms
- 6.4 Language Generation

7. Generative Models

- 7.1 Generative Adversarial Networks (GAN)
- 7.2 Variational Autoencoders
- 7.3 Image and Text Generation
- 7.4 Style Transfer

8. Advanced Topics

- 8.1 Reinforcement Learning
- 8.2 Unsupervised Learning
- 8.3 Ensemble Methods
- 8.4 Explainability and Interpretability

9. Applications of Deep Learning

- 9.1 Image Classification
- 9.2 Speech Recognition
- 9.3 Medical Imaging
- 9.4 Autonomous Vehicles

10. Practical Implementation

10.1 Deep Learning Frameworks

10.2 TensorFlow and PyTorch

10.3 Model Deployment and Scaling

11. Future Trends and Emerging Technologies

11.1 Current Research Areas

11.2 Ethical Considerations in Deep Learning

11.3 Future Directions and Challenges

12. Conclusion and Recap

12.1 Summary of Key Concepts

12.2 Practical Tips for Deep Learning Projects

1.1 Overview of Machine Learning

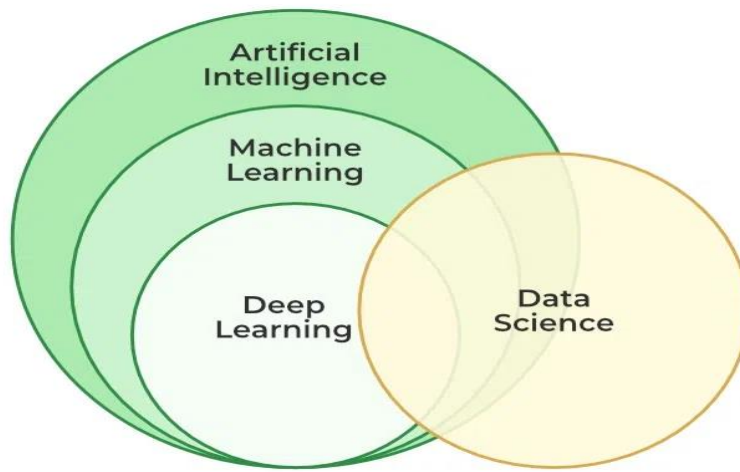
Machine learning is a subfield of artificial intelligence (AI) that focuses on developing algorithms and statistical models that enable computer systems to learn from data and make predictions or decisions without being explicitly programmed. Machine learning algorithms learn pattern and relationship from the examples, In other words, machine learning allows systems to automatically learn patterns, relationships, and insights from data, improving their performance over time.

Machine Learning Algorithms basically divides into three parts:

- Supervised algorithms
- Unsupervised algorithms
- Semi unsupervised algorithms

1.2 Overview of Deep Learning

Deep learning is a subset of machine learning, which is work based on artificial neural network. It is capable of learning complex patterns and relationships within data. An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data. In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the



final layer produces the output of the network. It has become increasingly popular in recent years due to the advances in processing power and the

availability of large datasets.

These neural networks are inspired by the structure and function of the human brain's biological neurons, and they are designed to learn from large amounts of data.

Training deep neural networks typically requires a large amount of data and computational resources. However, the availability of cloud computing and the development of specialized hardware, such as Graphics Processing Units (GPUs), has made it easier to train deep neural networks.

Deep Learning has achieved significant success in various fields, including image recognition, natural language processing, speech recognition, and recommendation systems. Some of the popular Deep Learning architectures include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs).

1.3 Deep Learning Vs Machine Learning

Machine Learning	Deep Learning
Apply statistical algorithms to learn the hidden patterns and relationships in the dataset.	Uses artificial neural network architecture to learn the hidden patterns and relationships in the dataset.
Can work on the smaller amount of dataset	Requires the larger volume of dataset compared to machine learning

suitable for the low-label task	Suitable for complex task like image processing, natural language processing, etc
Takes less time to train the model.	Takes more time to train the model
In image processing, relevant features extract manually to train the model	in image processing, relevant features extract automatically from the image, it is an end to end learning process
It can work on the CPU or requires less computing power as compared to deep learning.	It requires a high-performance computer with GPU

1.4 Deep Learning Application

The main applications of deep learning can be divided into computer vision, natural language processing (NLP), and reinforcement learning.

1. Computer vision

Computer vision is a field of study and technology that enables computers to interpret and make sense of visual information from the world, allowing them to understand and analyse images or videos like humans do.

some common solutions that can be developed using computer vision:

- Object Recognition and Tracking
- Facial Recognition
- Image Classification
- Gesture Recognition
- Medical Image Analysis
- Autonomous Vehicles
- Agricultural Monitoring
- Quality Control in Manufacturing

1. **Object detection and recognition:** Deep learning model can be used to identify and locate objects within images and videos, making it possible for machines to perform tasks such as self-driving cars, surveillance, and robotics.
2. **Image classification:** Deep learning models can be used to classify images into categories such as animals, plants, and buildings. This is used in applications such as medical imaging, quality control, and image retrieval.

3. **Image segmentation:** Deep learning models can be used for image segmentation into different regions, making it possible to identify specific features within images.

2. Natural Language Processing

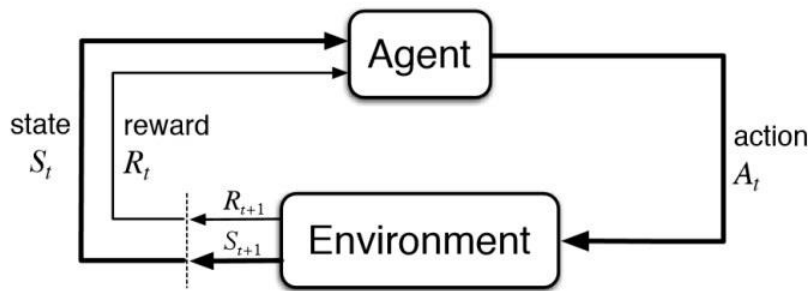
Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between computers and human languages. It involves the development of algorithms and computational models to enable machines to understand, interpret, and generate human language.

NLP used to solve a wide range of tasks and applications, including:

- Text Understanding
 - Sentiment Analysis
 - Question Answering
 - Language Generation
 - Speech Recognition
 - Text Summarization
 - Named Entity Recognition
1. **Automatic Text Generation:** Deep learning model can learn the corpus of text and new text like summaries, essays can be automatically generated using these trained models.
 2. **Language translation:** Deep learning models can translate text from one language to another, making it possible to communicate with people from different linguistic backgrounds.
 3. **Sentiment analysis:** Deep learning models can analyse the sentiment of a piece of text, making it possible to determine whether the text is positive, negative, or neutral. This is used in applications such as customer service, social media monitoring, and political analysis.
 4. **Speech recognition:** Deep learning models can recognize and transcribe spoken words, making it possible to perform tasks such as speech-to-text conversion, voice search, and voice-controlled devices.

3. Reinforcement learning:

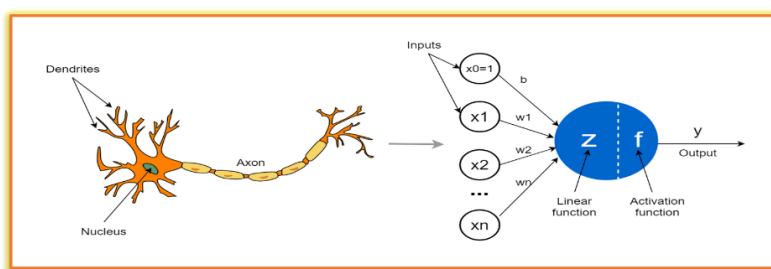
Reinforcement Learning is a part of machine learning. Here, agents are self-trained on reward and punishment mechanisms. It's about taking the best possible action or path to gain maximum rewards and minimum punishment through observations in a specific situation. It acts as a signal to positive and negative behaviours. Essentially an agent (or several) is built that can perceive and interpret the environment in which is placed, furthermore, it can take actions and interact with it.



- **Game playing:** Deep reinforcement learning models have been able to beat human experts at games such as Go, Chess, and Atari.
- **Robotics:** Deep reinforcement learning models can be used to train robots to perform complex tasks such as grasping objects, navigation, and manipulation.
- **Control systems:** Deep reinforcement learning models can be used to control complex systems such as power grids, traffic management, and supply chain optimization.

1.5 Biological neuron

A biological neuron is like a tiny worker in your brain that helps you think and process information. It's part of a team of many such workers that work together to make your brain function. These neurons are connected to each other, passing messages to help you see, hear, think, and do everything your brain does. They're like little messengers working together in your head, and deep learning inspired by this biological neural network architecture, so deep learning introduced a term called “artificial neural network”, and principle of artificial neural network same to the biological neural network.



1.6 Fundamentals of Neural Networks

The fundamentals of neural networks form the basis for understanding the principles behind deep learning. Here is a concise overview of the key concepts related to neural networks:

Neurons:

Definition: Neurons are the basic building blocks of neural networks. They receive input signals, perform a computation, and produce an output signal.

Function: Neurons process information by applying an activation function to the weighted sum of input signals.

Activation Function:

Definition: The activation function introduces non-linearity to the neural network, allowing it to learn complex patterns.

Function: It determines the output of a neuron, transforming the weighted sum of inputs into an output signal.

Weights and Bias:

Definition: Weights represent the strength of connections between neurons, and biases allow the model to adjust the output.

Function: Weights and biases are learned during the training process to optimize the network's performance.

Layers:

Definition: Neural networks consist of layers, including an input layer, hidden layers, and an output layer.

Function: Input layers receive external data, hidden layers' process information, and the output layer produces the final results.

Feedforward Neural Network:

Definition: In a feedforward neural network, information flows in one direction—from the input layer to the output layer—without cycles.

Function: It's the simplest form of neural network and is used for tasks like classification and regression.

Backpropagation:

Definition: Backpropagation is a supervised learning algorithm used to train neural networks by adjusting weights and biases.

Function: It minimizes the difference between predicted and actual outputs, updating parameters based on the gradient of the loss function.

Loss Function:

Definition: The loss function measures the difference between the predicted output and the actual target.

Function: During training, the goal is to minimize the loss, guiding the network to make accurate predictions.

Gradient Descent:

Definition: Gradient descent is an optimization algorithm used to minimize the loss function.

Function: It iteratively adjusts the model's parameters in the opposite direction of the gradient, moving towards the minimum loss.

2.1 Neural Network Basics

Activation function

activation functions are added non linearity, without an activation function, a Neural Network is just a linear regression model.

Activation functions play a crucial role in neural networks by introducing non-linearities to the model. They determine whether a neuron in the network should be activated or not, and Activation functions allow neural networks to learn and represent complex relationships in data.

the purpose of an activation function is to add non-linearity to the neural network.

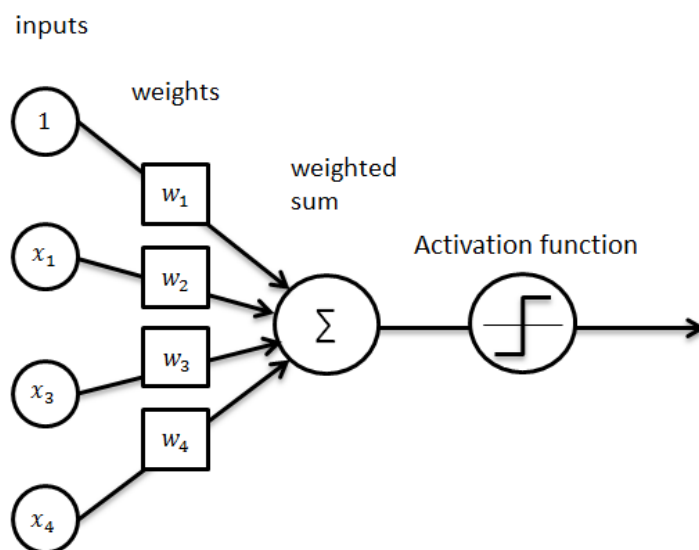


Image1.1

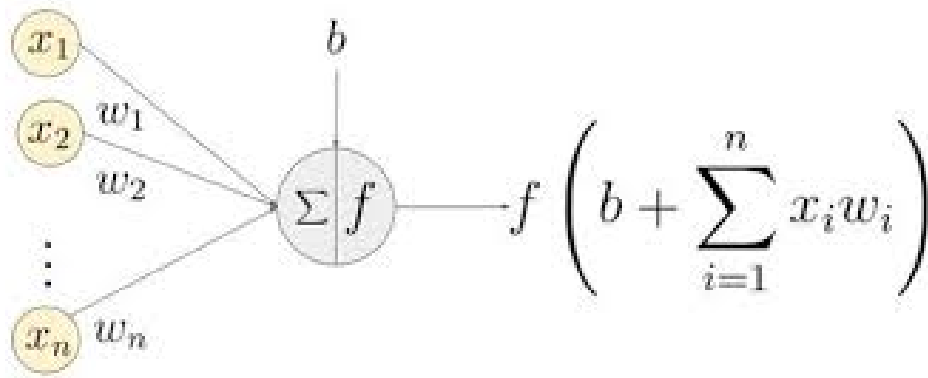


Image 1.2

you can see a function is “f” In above mentioned (Image 1.2), which is activation function, in which you can see how the input is get initialized with weights, and passed through the activation function. And produce a single value, called activation function output.

Activation functions in machine learning and deep learning are:

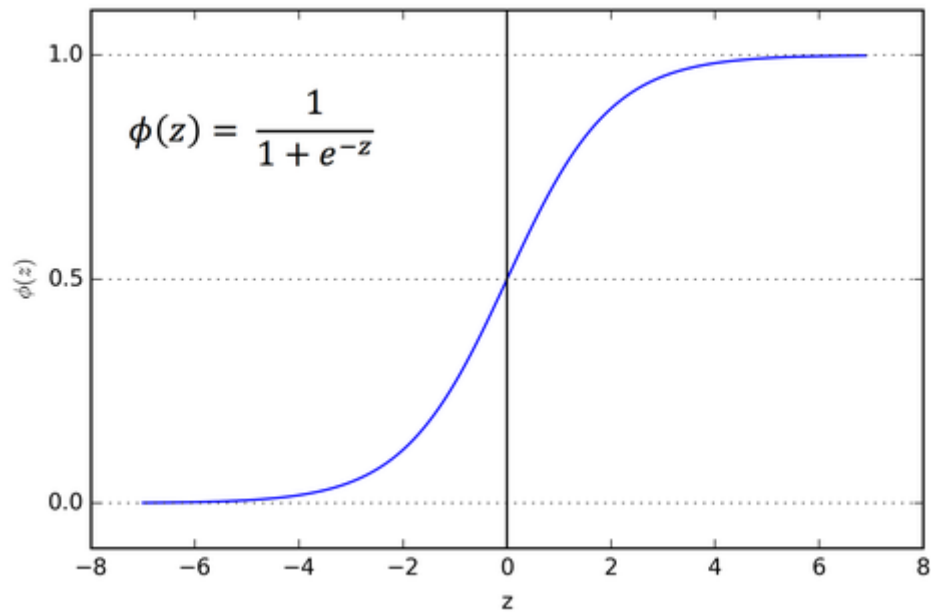
- Sigmoid
- Tanh
- Relu
- Leaky relu
- Parametric relu
- Exponential relu
- Softmax
- Linear

2.1.1 Sigmoid or logistic Activation function

This function takes any real value as input and outputs values in the range of **0 to 1**.

The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.

The Sigmoid Function curve looks like a S-shape



Sigmoid / Logistic

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

Here:

- $f(x)$ is the output of the sigmoid function.
- e is the base of the natural logarithm (approximately 2.71828).
- x is the input to the function.

Here X

$X_1, X_2, X_3, \dots, X_m$ are inputs feature of a task

$W_1, W_2, W_3, \dots, W_n$ are weights of corresponding inputs.

$$X = \sum X_i \cdot W_i + b \quad \text{where } b = \text{bias}$$

And now this X will be pass through the Activation Function

$$f(x) = \sigma(X) \quad \text{here we used binary Sigmoid activation function.}$$

Code to be implement sigmoid activation in python``

python

Copy code

```
import math

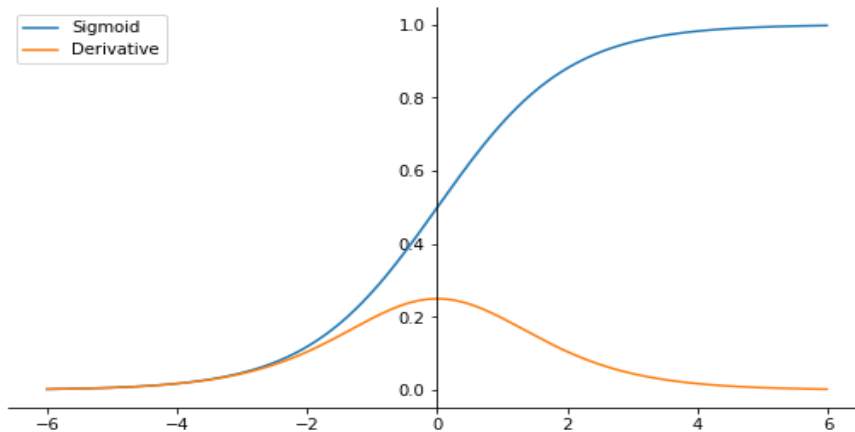
def sigmoid(z):
    return 1 / (1 + math.exp(-z))
```

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to **predict the probability** as an output.

The sigmoid activation function is commonly used in the **output layer** of a neural network for **binary classification problems**.

2.1.2 Tanh or hyperbolic tangent Activation Function

Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of **-1 to 1**. In Tanh, the larger the input (more positive), the closer the output value will be to **1.0**, whereas the smaller the input (more negative), the closer the output will be to **-1.0**.



Tanh

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

Output range: in between -1 to 1.

Derivative range: in between 0 to 1.

Advantages of using this activation function are:

- The output of the tanh activation function is **Zero centered**, hence we can easily map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers of a neural network as its values lie between **-1 to 1**, therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.

Disadvantages of using this activation function are:

- Time complexity is high
- It leads **vanishing gradient problem**, in case of deep neural network.

2.1.3 Relu

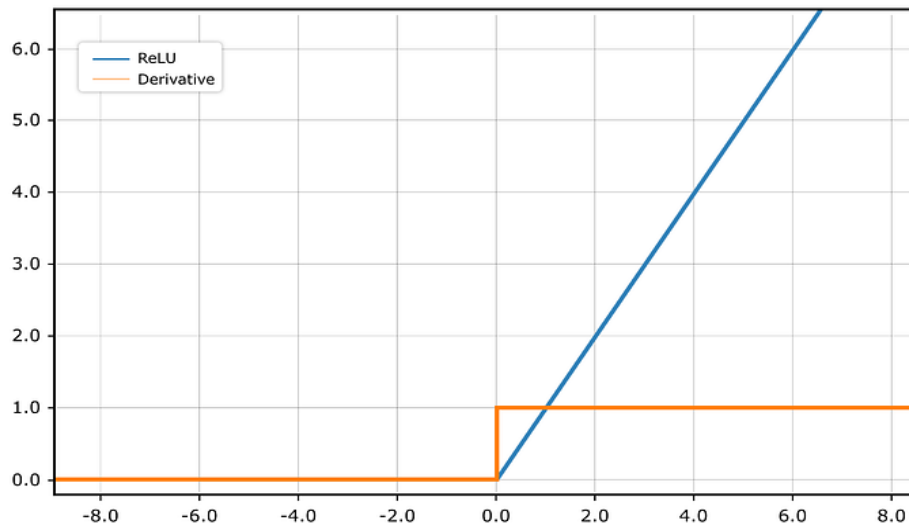
ReLU stands for Rectified Linear Unit, and it is an activation function commonly used in artificial neural networks and deep learning models. ReLU is considered as one of the biggest breakthroughs in deep learning because ReLU makes it possible to train a very deep neural network. ReLU is easy to optimize because it is so simple, computationally cheap, and similar to the linear activation function, but in fact, ReLU is a nonlinear activation function that allows complex patterns in the data to be learned. The only difference between a linear activation function and a ReLU is that ReLU pushes the negative value to **0**.

The mathematical representation of the ReLU activation function is quite simple. For a given input x , the output y is defined as:

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

In other words, if the input x is positive, the output is equal to x , and if x is negative, the output is 0. This piecewise linear function has become popular in deep learning due to its simplicity and effectiveness in training neural networks.

- **Equation :** $R(x) = \max(0, x)$. It gives an output x if x is positive and 0 otherwise.
- **Value Range :** $[0, \infty)$
- **Nature :** non-linear, which means we can easily back propagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :** ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.



Output range: 0 to infinite

Derivative range: either 0 and 1

Advantages:

- Simplicity
- Faster Convergence
- Non-linearity
- Prevent vanishing gradient problem.

Disadvantages:

- Dead neuron problem
- Output Not Centered Around Zero

Dead neuron problem

choosing an activation function for the hidden layer is not an easy task. The configuration of the hidden layer is an extremely active topic of research, and it just doesn't have any theory about how many neurons, how many layers, and what activation function to use given a dataset.

The "dead neuron" problem occurs when a ReLU neuron always outputs zero for any input, essentially becoming inactive. This can happen when the weights associated with the neuron are adjusted in such a way that the weighted sum of inputs is always negative, resulting in the ReLU function outputting zero. Once a neuron becomes "dead," it no longer contributes to the learning process because its gradient is zero, and it essentially stays inactive throughout training.

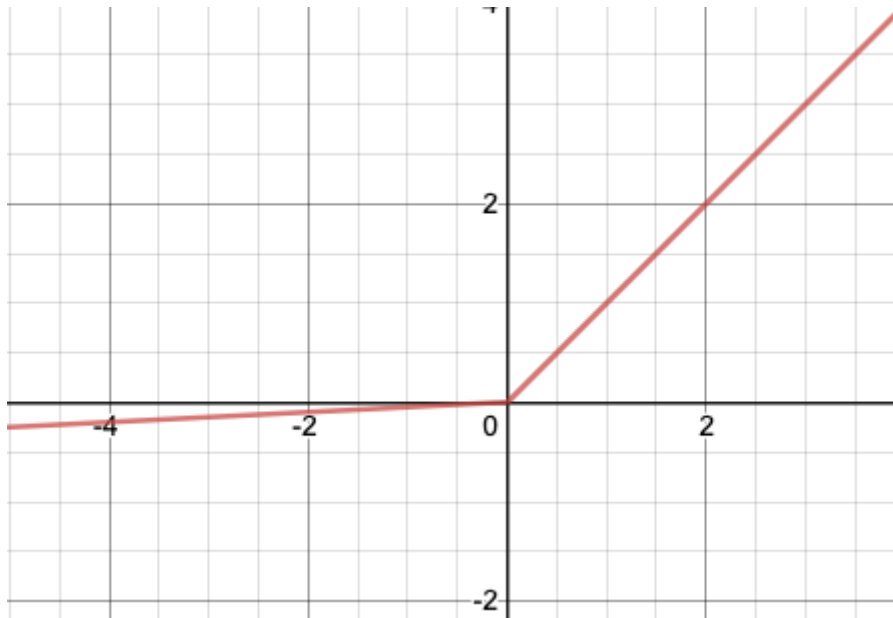
the weights of this neuron will never be updated again. Such a neuron can be considered as a dead neuron, which is considered a kind of permanent **“brain damage”** in biological terms.

Relu has three variants

- Leaky ReLU
- Parametric ReLU (PReLU)
- Exponential Linear Unit (ELU)

2.1.4 Leaky Relu

Leaky ReLU is a variant of ReLU, it introduced to prevent the dead neuron problem in neural network, Instead of pushing the negative value to 0, Leaky ReLU allows some leak at the negative region by multiplying x with a constant of 0.01



By doing this, even if the neuron has a big negative weight and bias, it's still possible to back propagate the gradient through the layer. The formula for Leaky ReLU is as follow:

Output range:

$$f(x) = \begin{cases} x, & x > 0 \\ 0.01x, & x \leq 0 \end{cases}$$

And its derivative range:

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0.01, & x \leq 0 \end{cases}$$

2.1.5 Parametric ReLU

Instead of using constant 0.01, parametric ReLU said try with different parameters,

The mathematical representation of Parametric ReLU is as follows:

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ \alpha_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

The above equation can also be represented as follows:

$$f(y_i) = \max(0, y_i) + \alpha_i \min(0, y_i)$$

if

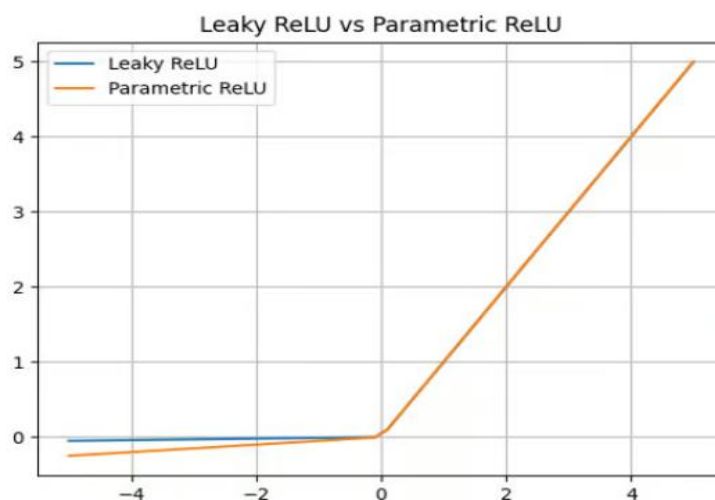
$\alpha = 0$ then behave same to ReLU

$\alpha = 0.01$ then behave same to the LeakyReLU

because α is hyper parameter

Parametric ReLU vs. Leaky ReLU

In this section, we compare Parametric ReLU with the performance of Leaky ReLU.



Here, we plot Leaky ReLU with $\alpha=0.01$ and have Parametric ReLU with $\alpha=0.05$. In practice, this parameter is learned by the neural network and changes accordingly.

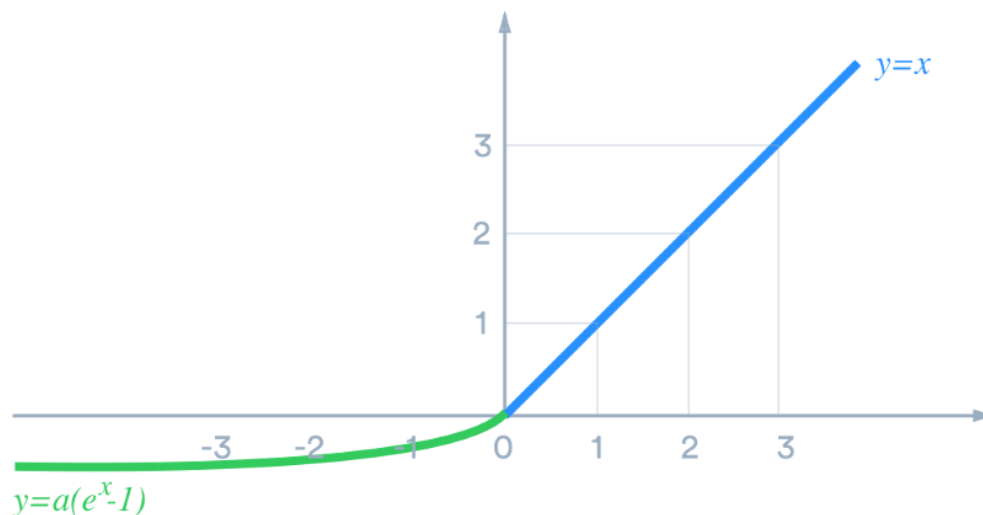
2.1.6 Exponential Linear Unit (ELU)

ELU is smooth and differentiable everywhere, including at the origin. This property can be beneficial during optimization processes, as it helps to avoid issues related to non-differentiability.

ELU helps mitigate the problem of "dead neurons" that can occur with ReLU activation.

This increased computational cost may impact the training time, especially in deep neural network.

As you can see smoothness in the curve of activation, this smoothness introduced by ELU.



The math is quite simple. The equation can be seen below to understand output:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

Equation to understand the derivative

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^x & \text{if } x \leq 0 \end{cases}$$

2.1.7 Softmax

Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. So it used at the last layer


of artificial neural network to perform multiclass classification, while sigmoid is use for binary classification, so whenever you will required to perform classification in between more than two classes, in that situation we will use softmax activation.


Mathematically, Softmax is defined as,


$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^n \exp(y_j)}$$

Lets keep a vector $Y = \{ 2.33, -1.46, 0.56 \}$ for an example and passed to the softmax.

Example :

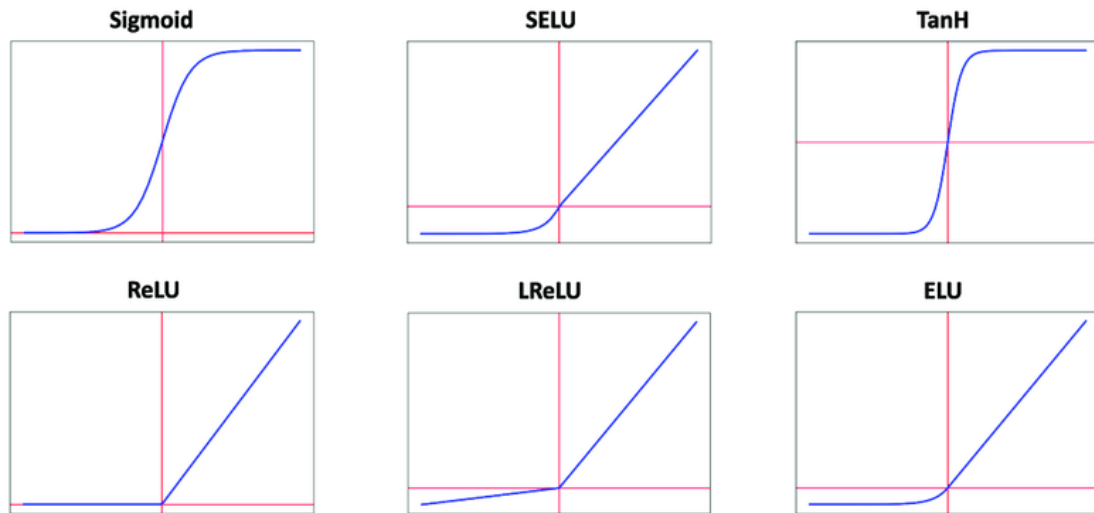

$$P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$


$$P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$


$$P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

In softmax, sum of all probabilities always equal to 1, so we got it 0.83 probability for class 1, which is highest probability. So according to the softmax activation function class 1 will be our truth prediction in this case.

Comparison graph with respective all the activation function

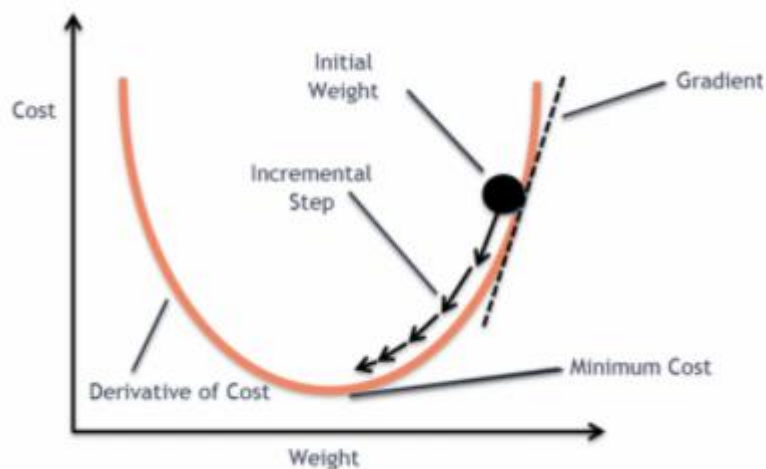


2.2 Loss Functions

A loss function is a function in machine learning that measures how far off a model's predictions are from the actual results it, the goal during training is to minimize this "loss" so that the model gets better at making accurate predictions. Loss function work as guider during the train of neural network, for the optimization algorithm. Loss function describes how far algorithm away from target point.

The primary goal of training a neural network is to minimize this loss function. After calculate the loss, according to the measured loss, optimizers optimize the model parameters (weights and biases) to minimize the value of the loss function. The goal is to find the set of parameters that results in the smallest possible loss, indicating that the model is making accurate predictions on the training data.

We will explore only those loss functions those are used to solve regression and classification



problems. Loss functions are in machine learning and deep learning are ,

Loss functions are used in regression:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Huber loss

Loss functions are used in classification:

- Binary Cross-Entropy Loss
- Log Loss (Cross-Entropy Loss)
- Categorical Cross-Entropy Loss
- Sparse Categorical Cross-Entropy Loss
- Hinge Loss (SVM loss)

2.3 Why Do We Need Loss Functions in Deep Learning?

There are two possible mathematical operations happening inside a neural network:

- Forward propagation
- Backpropagation with gradient descent

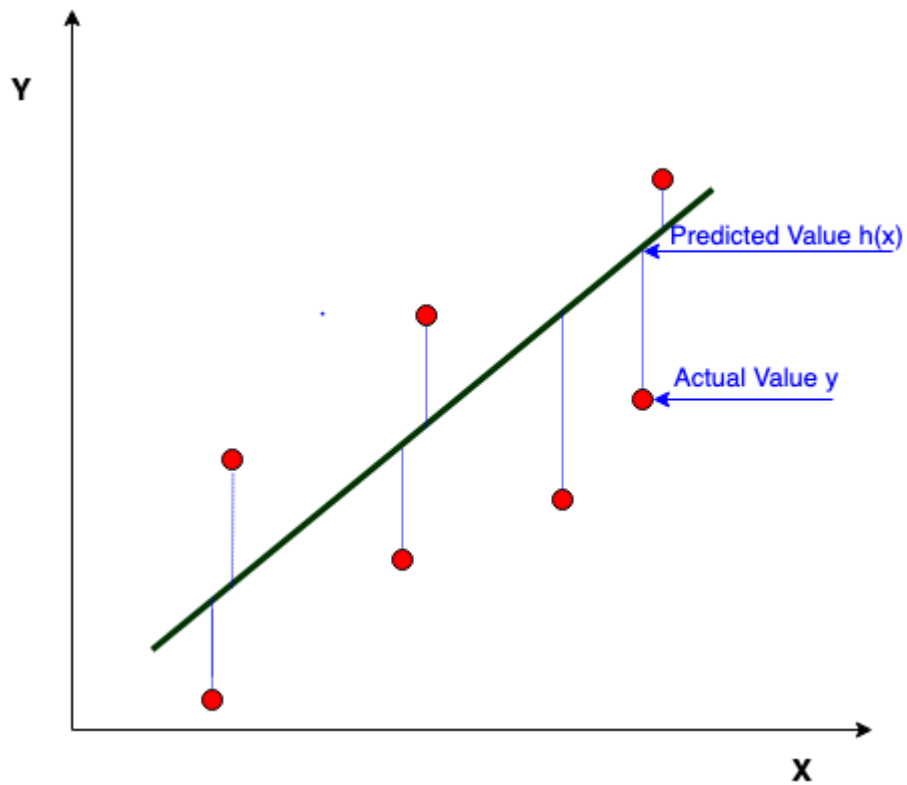
While forward propagation refers to the computational process of predicting an output for a given input vector x , backpropagation and gradient descent describe the process of improving the weights and biases of the network in order to make better predictions.

And to guiding the training process of neural network, loss function describes that how far algorithm away from the target point. So loss functions work as a guider to training of neural network.

2.2.1 Mean Squared Error (MSE)

Mean Square Error (MSE) is a commonly used metric to measure the average squared difference between the actual (observed) or Vector Y values and the predicted values \hat{Y} in a regression problem.

As you can see difference in between actual value and predicted value in above graph, this difference is known as Residual, and mean squared error measure the average squared difference between actual value and predicted value.



MSE formula can be written as:

$$\text{MSE} = \overset{\text{Mean}}{\frac{1}{n} \sum_{i=1}^n} \overset{\text{Error}}{(Y_i - \hat{Y}_i)} \overset{\text{Squared}}{^2}$$

where:

- n is the number of data points or samples.
- Y_i represents the actual or observed value for the i th data point.
- \hat{Y}_i represents the predicted value for the i th data point.

The diagram illustrates the calculation of error for a single data point. It features two boxes: a green box on the left containing the symbol Y_i and a red box on the right containing the symbol \hat{Y}_i . Above the green box is the text "real value" in green, and above the red box is the text "predicted value" in red. A horizontal line connects the two boxes, and a bracket underneath this line is labeled "error".

$$\text{real value } Y_i - \text{predicted value } \hat{Y}_i = \text{error}$$

Advantages

- Mathematical Simplicity
- Differentiable
- It has single global minima.
- It is a convex function so convergence is fast.
- It works well with optimization algorithms like gradient descent.

Disadvantages

- It is not robust to outliers

2.2.2 Mean Absolute Error

MAE is a statistical measure that assesses the average absolute difference between the actual historical data and the forecasted values predicted by a model. It calculates the average magnitude of errors by taking the mean of the absolute differences for each corresponding pair of actual and predicted data points.

MAE optimize the value of slope and intercept in sub region, because it use sub gradient concept. So it takes much more time in convergence comparatively with MSE. But it is robust to outliers.

What are the gradient descent , optimization algorithm , slope and intercept all these terms we will understand in upcoming pages.

Formula can be written as:

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Advantages:

Easy Interpretation: MAE is easy to understand and compute.

Robust to Outliers: It is less influenced by extreme values, making it suitable when dealing with data containing outliers.

Equal Treatment of Errors: MAE treats positive and negative errors equally. It gives the same weight to overestimations and underestimations.

Simple Computation: The computation of MAE is straightforward and involves taking the average of absolute errors, making it computationally efficient.

Disadvantages

Equal Emphasis on All Errors: MAE treats all errors, whether small or large, with equal importance. In cases where large errors are more concerning, MAE may not adequately highlight their significance.

Lack of Emphasis on Extreme Values: MAE does not emphasize or penalize large errors as much as metrics like Mean Squared Error (MSE) do. This can be a drawback in situations where extreme errors have a significant impact on the overall performance.

Convergence Time: its take much more time in convergence as comparatively to MSE.

2.2.3 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE and is the most popular error measure, also known as the *quadratic loss function*,

RMSE can be defined as square root of average squared difference between the actual (observed) or Vector Y values and the predicted values \hat{Y} in a regression problem.

Lets take an e.g. , Suppose you're predicting the daily temperature for a city, and your model predicts the temperature for three consecutive days as follows:

- Day 1 Prediction: 25°C
- Day 2 Prediction: 22°C
- Day 3 Prediction: 28°C

Now, let's say the actual temperatures for those days were:

- Day 1 Actual: 24°C
- Day 2 Actual: 20°C
- Day 3 Actual: 27°C

To calculate the absolute errors:

- Absolute Error Day 1: $|25 - 24| = 1^\circ\text{C}$
- Absolute Error Day 2: $|22 - 20| = 2^\circ\text{C}$
- Absolute Error Day 3: $|28 - 27| = 1^\circ\text{C}$

Now, calculating the Mean Absolute Error:

$$\text{MAE} = \frac{1}{3} \times (1 + 2 + 1) = \frac{4}{3} \approx 1.33$$

In this context, the MAE of approximately 1.33°C indicates the average absolute difference between the predicted temperatures and the actual temperatures over the three days.

Now, let's calculate Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{3} \sum_{i=1}^n ((1)^2 + (2)^2 + (1)^2)$$

$$\text{MSE} = \frac{1}{3} * 1 + 4 + 1$$

$$\text{MSE} = \frac{6}{3} = 2$$

Now, let's calculate Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{3} ((1)^2 + (2)^2 + (1)^2)}$$

$$\sqrt{\frac{1}{3} (1 + 4 + 1)} = \sqrt{\frac{1}{3} * 6} = \sqrt{\frac{6}{3}} = \sqrt{2} = 1.14$$

In this example, **MSE** is approximately **2.0°C**, **MAE** is approximately **1.33°C** , and **RMSE** is approximately **1.41°C**.

- **MAE** (Mean Absolute Error): Approximately 1.33°C
- **RMSE** (Root Mean Squared Error): Approximately 1.41°C
- **MSE** (Mean Squared Error):

Loss function Vs Cost function

Loss Function

The loss function calculates the error for a single training example and is used to update the model's parameters during the training process.

Example:

Mean Squared Error (MSE) is a typical loss function used in regression tasks. It calculates the average squared difference between the predicted and actual values.

Cost Function

It's the sum or average of the individual losses computed by the loss function for each training example.

Example:

The Mean Squared Error can also be considered a cost function when calculated over the entire training dataset.

Loss functions are used in classification:

- Log Loss (Cross-Entropy Loss)
- Binary Cross-Entropy Loss
- Categorical Cross-Entropy Loss (softmax loss function)
- Sparse Categorical Cross-Entropy Loss
- Hinge Loss (SVM loss)

OH

LE

2.2.4 Log Loss

Log loss, also known as logarithmic loss or cross-entropy loss, is a common evaluation metric for binary classification and multiclass classification problems. It measures the performance of a model by quantifying the difference between predicted probabilities and

actual values. Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification), and it penalizing inaccurate predictions with higher values. Lower log-loss indicates better model performance.

By default, the output of the logistic regression model is the probability of the sample being positive (indicated by 1). For instance, if a logistic regression model is trained to classify a company dataset, the predicted probability column indicates the likelihood of a person buying a jacket. In the given dataset, the log loss for the prediction that a person with ID6 will buy a jacket is 0.94.

In the same way, the probability that a person with ID5 will buy a jacket (i.e. belong to class 1) is 0.1 but the actual class for ID5 is 0, so the probability for the class is $(1-0.1)=0.9$.

0.9 is the correct probability for ID5

ID	Actual	Predicted Probabilities	Corrected Probabilities
ID6	1	0.94	0.94
ID1	1	0.9	0.9
ID7	1	0.78	0.78
ID8	0	0.56	0.44
ID2	0	0.51	0.49
ID3	1	0.47	0.47
ID4	1	0.32	0.32
ID5	0	0.1	0.9

We will find a log of corrected probabilities for each instance.

ID	Actual	Predicted Probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.02687
ID1	1	0.9	0.9	-0.04576
ID7	1	0.78	0.78	-0.10791
ID8	0	0.56	0.44	-0.35655
ID2	0	0.51	0.49	-0.3098
ID3	1	0.47	0.47	-0.3279
ID4	1	0.32	0.32	-0.49485
ID5	0	0.1	0.9	-0.04576

Negative Average

As you can see these log values are negative. To deal with the negative sign, we take the negative average of these values, to maintain a common convention that lower loss scores are better.

$$\log \text{ loss} = -1/N \sum_{i=1}^N (\log (P_i))$$

In short, there are three steps to find Log Loss:

- To find corrected probabilities.
- Take a log of corrected probabilities.
- Take the negative average of the values we get in the 2nd step.

If we summarize all the above steps, we can use the formula:-

$$-\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Here Y_i represents the actual class and $\log(p(y_i))$ is the probability of that class.

- $p(y_i)$ is the probability of 1.
- $1 - p(y_i)$ is the probability of 0.

2.2.5 Binary cross entropy

Binary cross entropy (also known as logarithmic loss or log loss) is a model metric that tracks incorrect labeling of the data class by a model, penalizing the model if deviations in probability occur into classifying the labels. Low log loss values equate to high accuracy values. Binary cross entropy is equal to $-1 \cdot \log(\text{likelihood})$.

Formula can be written as :

$$-\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Here Y_i represents the actual class and $\log(p(y_i))$ is the probability of that class.

- $p(y_i)$ is the probability of one
- $1 - p(y_i)$ is the probability of zero

2.2.6 Categorical cross entropy

Categorical cross-entropy loss function is a fundamental metric in deep learning used to quantify the dissimilarity between predicted and actual probability distributions in classification tasks with multiple classes. If you have two classes in your problem statement mean that you are performing binary classification so you used binary cross entropy, which we have explained in previously. But think what you will do when you will have to perform multiclass classification.

The categorical cross-entropy loss function is commonly used in neural networks with softmax activation in the output layer for multi-class classification tasks.

$$\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_i \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i)$$

Where:

- \mathbf{L} is the loss function.
- \mathbf{y} is the true probability distribution (the one-hot encoded ground truth).
- $\hat{\mathbf{y}}$ is the predicted probability distribution.
- \mathbf{y}_i and $\hat{\mathbf{y}}_i$ are the probabilities for the i th class in the true and predicted distributions, respectively.

Let's assume you have three classes in your dependent variable then formula will be

$$\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^3 \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i)$$

Where:

- 3 represents the total number of classes.

Cost function for the categorical crossentropy

$$- \frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

2.2.7 Sparse Categorical cross entropy

Sparse categorical crossentropy and categorical crossentropy are both loss functions used in multiclass classification tasks, particularly in the context of neural networks and deep learning.

Sparse categorical crossentropy is used when the targets are integers, rather than one-hot encoded vectors. In other words, instead of each target being a one-hot encoded vector, it is simply the index of the class.

sparse categorical crossentropy and categorical crossentropy serve the same purpose measuring the difference between the true and predicted probability distributions in multiclass classification tasks. The main difference lies in the format of the target labels sparse categorical crossentropy accepts integer labels directly, while categorical crossentropy expects one-hot encoded labels. Mathematically, they are equivalent when dealing with the same underlying data representation

Sparse categorical crossentropy loss function equation

$$\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_i \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i)$$

Where:

- \mathbf{L} is the loss function.
- \mathbf{y} is the true probability distribution (the one-hot encoded ground truth).
- $\hat{\mathbf{y}}$ is the predicted probability distribution.
- \mathbf{y}_i and $\hat{\mathbf{y}}_i$ are the probabilities for the i th class in the true and predicted distributions, respectively.

Make sure before apply sparse categorical crossentropy loss function on your dataset, then dependent variable or target variable must be integer index labelled, if you have one hot encoded to the target variable, then you should use categorical crossentropy.

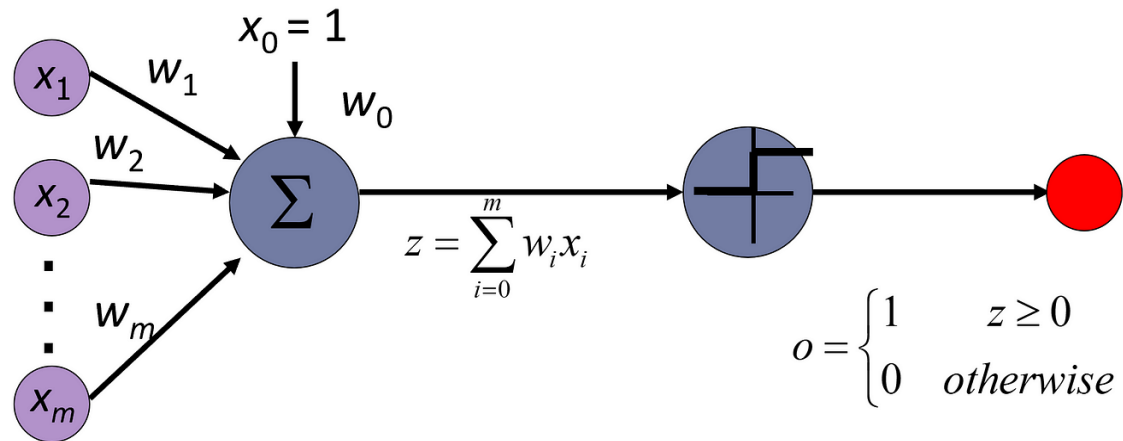
Sparse categorical crossentropy is quiet fast to the categorical crossentropy, because it calculates the loss only True classe rather than calculating for each class, categorical crossentropy calculates the loss for each class.

Perceptron

A perceptron is an artificial neuron, and it is the smallest unit of artificial neural network, Frank Rosenblatt first proposed in 1958 is a simple neuron which is used to classify its input into one or two categories. Perceptron is a linear classifier, and is used in supervised learning. Perceptron is mainly used to classify the data into two parts. Therefore, it is also known as Linear Binary Classifier.

by making or combining multiple perceptron in a network, it forms an artificial neural network.

The original Perceptron was designed to take a number of binary inputs, and produce one binary output (0 or 1).



perceptron is the simplest form of a neural network, the activation function traditionally used is a step function. The perceptron takes multiple input values, each multiplied by its corresponding weight, and then the sum of these weighted inputs is passed through an activation function. The step function is often used as the activation function in a perceptron to produce a binary output.

Working of perceptron

The perceptron is a simple type of artificial neuron or a basic building block of neural networks. Let's break down the mathematics of a perceptron step by step:

Input values: Let's denote the input values as x_1, x_2, \dots, x_m , where m is the number of inputs.

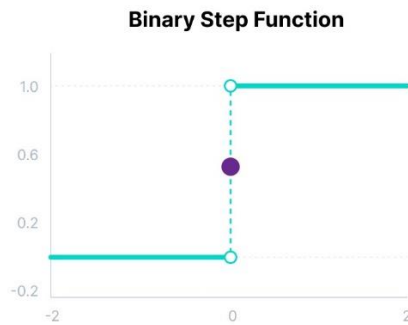
Weights: Assign a weight to each input. Let w_1, w_2, \dots, w_m be the corresponding weights.

Bias: Introduce a bias term denoted as b . The bias allows the perceptron to adjust its decision boundary.

Weighted sum: Calculate the weighted sum of the inputs plus the bias:

$$Z = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_m \cdot x_m + b$$

Activation function: Apply an activation function to the weighted sum. Traditionally, a step function is used for a perceptron. The step function outputs 1 if the weighted sum is greater than or equal to 0, and 0 otherwise.



Z will be pass through the binary step activation function, and it will return 0 or 1 based on the threshold, if ($z \geq \text{threshold}$) it will return 1 else it return 0.

After getting prediction of all the data item, then it will calculate the cost or error, using loss function (cost function)

Let take an example to describe the working process of peceptron, consider X_1, X_2, \dots, X_m are input feature for the perceptron, and W_1, W_2, W_3, W_n are corresponding weights. to perform binary classification,

X_1, X_2, X_3, X_m are inputs feature of a task

W_1, W_2, W_3, W_n are weighs of corresponding inputs.

$$Z = \sum X_1.W_1 + X_2.W_2 + \dots + X_m.W_n + b \quad \text{where } b = \text{bias}$$

And now this Z will be pass through the Activation Function

$$y^{\wedge} = \sigma(z) \quad \text{here we used binary step activation function.}$$

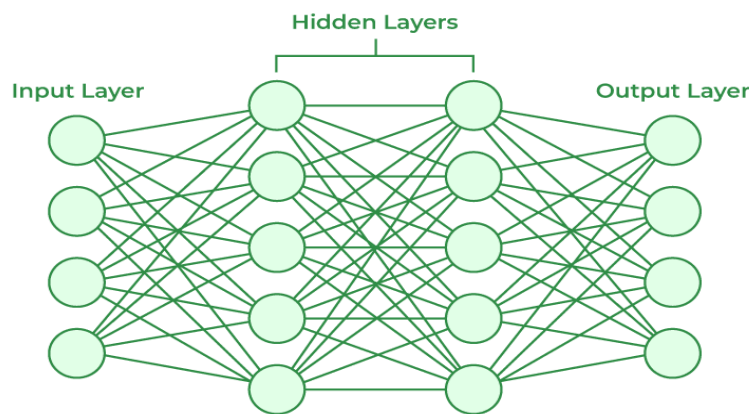
After all this $Y^{\wedge} = \sigma(z)$ will be passed through the loss function. To calculate the loss in between actual True positive and prediction probability.

Various neural networks architecture in deep learning

- Artificial neural network.
- Convolutional neural network.
- Region-based convolutional neural network.
- Generative adversarial neural network.
- Recurrent neural network

Artificial neural network

The artificial neural network is the mathematical model which is mainly inspired by the biological neuron system in the human brain. As the neural network in human mind, which process the signal, in a same way The artificial neural network is made up of a large number of processing components that are linked together by weighted paths to form networks. The result of every element is computed by applying a non-linear mathematical function of its weighted inputs. When these processing components are combined into networks then it forms artificial neural network, they can perform arbitrarily complicated non-linear functions like classification, regression, or optimization, Like the human brain, these artificial neural networks learn from experiences, and generalize from examples.



Layers in Artificial neural network

- Input layer
- Hidden layers
- Output layer

Input layer: In artificial neural networks (ANNs), the input layer is the initial layer of nodes where the raw input data is fed into the network. Each node in the input layer represents a feature or attribute of the input data. The number of nodes in the input layer corresponds to the dimensionality of the input data.

Hidden layers: in artificial neural networks (ANNs) are layers of nodes between the input and output layers that process input data through transformations to extract features and patterns, enabling the network to learn complex relationships in the data.

Output layer: in artificial neural networks (ANNs) is where the network's computation ends, providing the final predictions or outcomes based on the processed input data. Each node in this layer corresponds to a specific class or value that the network is trying to predict.

Artificial neural network training done in two phase:

- Forward propagation.
- Backward propagation

Forward Propagation: Forward propagation is a fundamental step in training a neural network. It's like passing a message from the input layer through all the hidden layers until it reaches the output layer.

Here's a simple explanation of forward propagation:

Start with Input: You begin by feeding your input data into the neural network. Each piece of input data represents one sample, such as an image or a sentence.

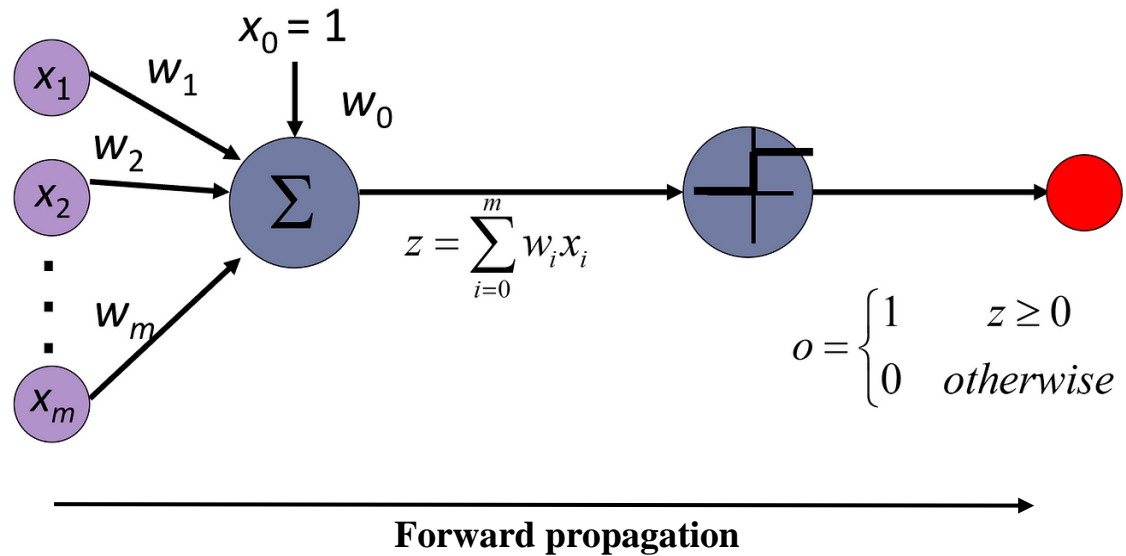
Compute Activations: As the data moves through the network, each neuron in each layer calculates its activation based on the input it receives and its internal parameters (weights and biases). Neurons in the first hidden layer receive the input data directly, while neurons in subsequent hidden layers receive the output from the previous layer.

Pass Through Layers: The calculated activations are passed from one layer to the next, with each layer performing a transformation on the data. This transformation involves multiplying the activations by the weights of the connections between neurons, adding biases, and applying an activation function (such as ReLU or sigmoid) to introduce nonlinearity.

Output Layer: Finally, the transformed data passes through the output layer, where it undergoes a final transformation to produce the network's output. This output could represent predictions for classification tasks, regression values, or any other desired output based on the problem being solved.

Prediction: The output produced by the output layer is the prediction made by the neural network for the given input data. This prediction can then be compared to the actual target values to compute the loss, which is used to update the network's parameters during training.

In the diagram below, we illustrate the concept of forward propagation in neural networks.



Backward propagation: is a fundamental algorithm used in training artificial neural networks. Backpropagation is an iterative optimization algorithm that allows a neural network to learn from its mistakes by efficiently adjusting its internal parameters, such as weights and biases, based on the discrepancy between its predictions and the actual target values.

The main aim of the backpropagation algorithm is to enable a neural network to learn from its mistakes and improve its performance over time. Specifically, its primary objectives are as follows

- Error Minimization
- Parameter Optimization
- Learning from Data

To minimize the error in between actual and predicted point, it utilizes the optimizer concepts, so Backpropagation and optimizers work together in the training process of neural networks, with each playing a crucial role in optimizing the network's parameters like (weight and biases) and to minimizing the loss function.

Optimizers in BackPropagation

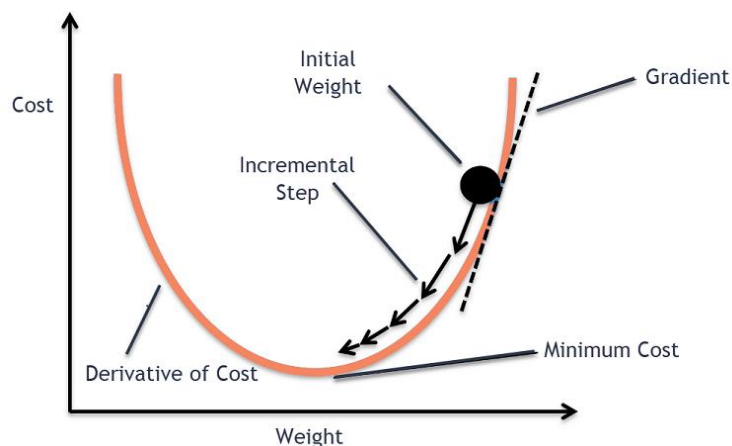
there are several types of optimizers used to update the parameters of a neural network during training. Each optimizer has its own characteristics and update rules that influence how efficiently the network learns. Some common types of optimizers in deep learning include:

- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Mini-Batch Stochastic Gradient Descent
- Stochastic with Momentum
- Adagrad (Adaptive Gradient descent)
- RMSprop
- Adadelta
- Adam (Adaptive Moment Estimation)
- Adamax

1. Gradient Descent

Gradient descent optimizer is a fundamental technique in deep learning used to minimize the loss function during the training of a neural network. The basic idea behind gradient descent is to adjust the parameters of the model iteratively in order to minimize the difference between the predicted outputs and the actual outputs (i.e., the loss).

Here's how it works:



Gradient descent finds out the optimal weights and bias for the ANN, where the cost is minimum, so gradient descent is a convergence or optimization algorithm that tries to optimize the value of ANN training parameters, as you can see in the above image where the gradient tries to push the initial weights towards the minimum cost. And gradient's main aim is to minimize the loss.

Calculate the Gradient: First, the gradient of the loss function with respect to each parameter in the neural network is computed. This gradient represents the direction and magnitude of the steepest increase of the loss function.

Update Parameters: The parameters of the neural network are then adjusted in the opposite direction of the gradient to minimize the loss. This adjustment is made by taking a small step (controlled by a parameter called the learning rate) in the direction that reduces the loss.

Repeat: Steps 1 and 2 are repeated iteratively for a fixed number of times (epochs) or until a certain convergence criterion is met.

Gradient Descent:

$$\theta = \theta - \eta \cdot \nabla J(\theta)$$

Where:

- θ represents the parameters (weights and biases) of the neural network.
- η (eta) denotes the learning rate, controlling the size of the steps taken towards the minimum.
- $\nabla J(\theta)$ is the gradient of the loss function J with respect to the parameters θ .

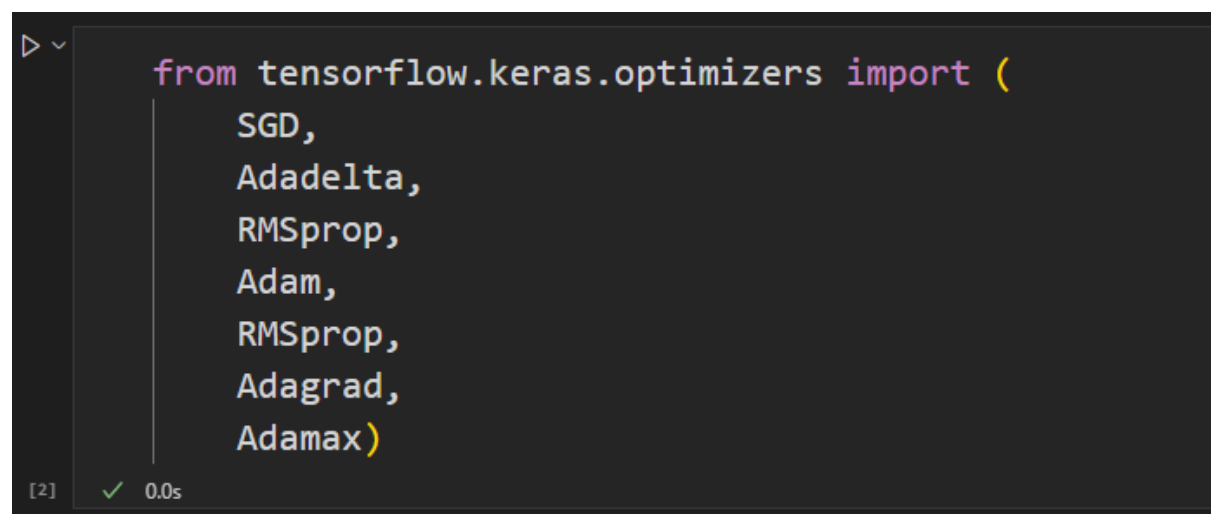
There are different variants of gradient descent optimizers, each with its own specific way of updating the parameters and handling the learning rate. Some common variants include:

Stochastic Gradient Descent (SGD): Updates the parameters using the gradient of the loss computed on a single training example at a time.

Mini-batch Gradient Descent: Updates the parameters using the gradient of the loss computed on a small subset (mini-batch) of the training data.

Adam (Adaptive Moment Estimation): An adaptive learning rate optimization algorithm that computes adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients.

You can import all the optimizers from `tensorflow.keras.optimizers`, as shown in below image. Where I imported all necessary optimizers.



```
from tensorflow.keras.optimizers import (  
    SGD,  
    Adadelta,  
    RMSprop,  
    Adam,  
    RMSprop,  
    Adagrad,  
    Adamax)
```

[2] ✓ 0.0s

Practical of ANN implementation on a real word dataset

an Artificial Neural Network (ANN) involves several practical steps, from data pre-processing to model evaluation. Here's a step-by-step guide to help you describe the practical process of ANN implementation:

First, let's understand the problem that we are going to solve

We're addressing the challenge faced by airlines in accurately measuring passenger satisfaction. By training an Artificial Neural Network (ANN) on a dataset provided by the airline, our goal is to predict passenger satisfaction levels. This predictive capability will empower airlines to enhance customer experience, improve services, and address areas of concern, ultimately leading to higher satisfaction levels and better overall performance.

This is dataset, which provided by the airline, where all the necessary details regarding passengers are available like flight distance, departure delay, arrival delay, so we are going to train our artificial neural network on this dataset,

Before the training of ANN on this dataset, first we will have to clean and transform this dataset in terms of ANN.

	A	B	C	D	E	G	H	I	J	K	L
1	Gender	Type of Travel	Class	Age	Flight Distance	Baggage handling	Cleanliness	Departure Delay in Minute	Arrival Delay in Minute	Customer Type	SATISFACTION
2	Male	Personal Travel	Eco Plus	13	460	4	5	25	18	Loyal Customer	neutral or dissatisfied
3	Male	Business travel	Business	25	235	3	1	1	6	dissloyal Customer	neutral or dissatisfied
4	Female	Business travel	Business	26	1142	4	5	0	0	Loyal Customer	satisfied
5	Female	Business travel	Business	25	562	3	2	11	9	Loyal Customer	neutral or dissatisfied
6	Male	Business travel	Business	61	214	4	3	0	0	Loyal Customer	satisfied
7	Female	Personal Travel	Eco	26	1180	4	1	0	0	Loyal Customer	neutral or dissatisfied
8	Male	Personal Travel	Eco	47	1276	4	2	9	23	Loyal Customer	neutral or dissatisfied
9	Female	Business travel	Business	52	2035	5	4	4	0	Loyal Customer	satisfied
10	Female	Business travel	Business	41	853	1	2	0	0	Loyal Customer	neutral or dissatisfied
11	Male	Business travel	Eco	20	1061	4	2	0	0	dissloyal Customer	neutral or dissatisfied
12	Female	Business travel	Eco	24	1182	5	2	0	0	dissloyal Customer	neutral or dissatisfied
13	Female	Personal Travel	Eco Plus	12	308	5	1	0	0	Loyal Customer	neutral or dissatisfied
14	Male	Business travel	Eco	53	834	3	1	28	8	Loyal Customer	neutral or dissatisfied
15	Male	Personal Travel	Eco	33	946	2	4	0	0	Loyal Customer	satisfied
16	Female	Personal Travel	Eco	26	453	2	2	43	35	Loyal Customer	neutral or dissatisfied
17	Male	Business travel	Eco	13	486	4	4	1	0	dissloyal Customer	neutral or dissatisfied
18	Female	Business travel	Business	26	2123	4	4	49	51	Loyal Customer	satisfied
19	Male	Business travel	Business	41	2075	5	5	0	10	Loyal Customer	satisfied
20	Female	Business travel	Business	45	2486	5	4	7	5	Loyal Customer	satisfied

I have .csv file in which this data is available so first we will have to read this dataset. So to read this file, we will use popular python library called **pandas**.

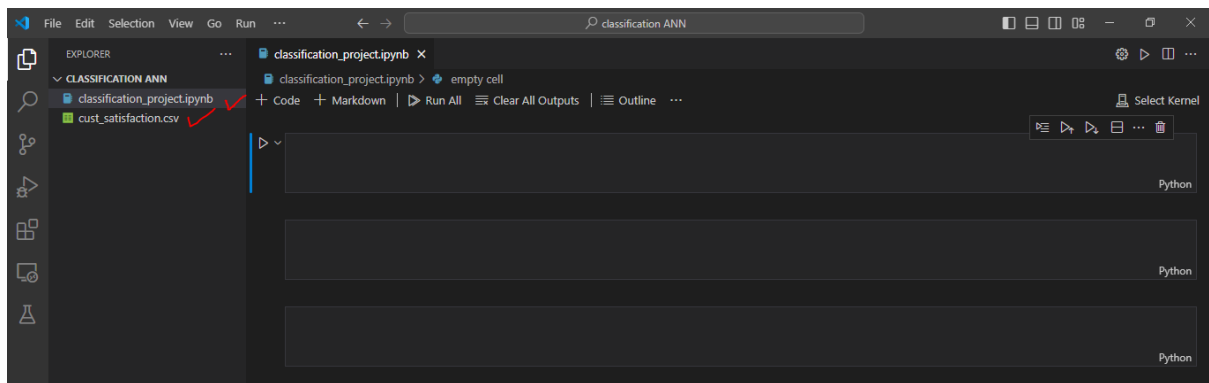
If you are first time using pandas then you will have to install it in python, so you can utilize the functionalities of pandas.

To install the pandas library execute the “**pip install pandas**” in the command prompt. After executing it command, pandas will be installed in your python.

And now we will create a new project folder, where we will keep all the files and dataset of this project.

You can see in below mentioned image, we have created a file “**classification_project.ipynb**”

And another is “**cust_satisfaction.csv**” that is a dataset file in .csv format, in this file we have airline dataset regarding passenger.



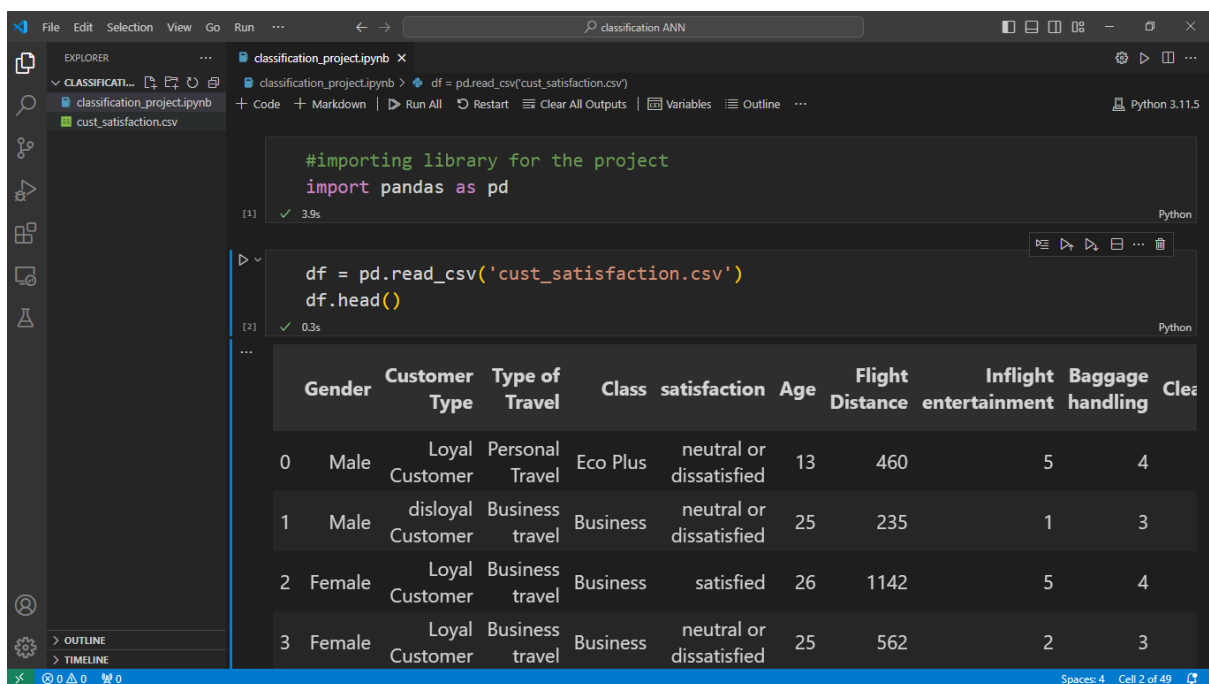
So now we have successfully installed pandas, and created “**classification_project.ipynb**” jupyter notebook, where we will our code of this project. And we have created this file in Microsoft VS Code (visual studio).

So let’s start the coding part, first we will import the **pandas** library, to read this “cust_satisfaction.csv” file, because pandas has ability to read data from various sources.

We have imported pandas library, and read the dataset, by using “**pd.read_csv(“file_name”)**” function of pandas. And pandas will return a dataframe that we holded in “**df**” variable. You can see in below mentioned images.

If you want to download this dataset click on below link:

https://drive.google.com/file/d/1xY4ltjOSIDJU5_bhFBw1_wxRDhHwugfg/view?usp=sharing



Let's understand the dataframe, we have executed "**df.info()**" to detailed the dataset, so according to this output we have 103903 passenger records. And 11 columns in our dataframes. And also we are able to see in this output that all columns are "non-null" means in this dataset has no missing value in any column. And also we can check out the data type of all the columns.

```
df.info()
```

```
[4] ✓ 0.0s
```

```
***
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Gender                                103904 non-null object
1   Customer Type                         103904 non-null object
2   Type of Travel                        103904 non-null object
3   Class                                103904 non-null object
4   satisfaction                           103904 non-null object
5   Age                                    103904 non-null int64
6   Flight Distance                       103904 non-null int64
7   Inflight entertainment                103904 non-null int64
8   Baggage handling                      103904 non-null int64
9   Cleanliness                           103904 non-null int64
10  Departure Delay in Minutes            103904 non-null int64
11  Arrival Delay in Minutes              103594 non-null float64
dtypes: float64(1), int64(6), object(5)
memory usage: 9.5+ MB
```

So we have profiled our dataset, and now we are writing the code to transform the dataset in terms of ANN, because Deep learning algorithms don't accept the alphabetical values, so will have to transform our dataset column values.

There are some columns which are containing categorical alphabetical value, in this below code cell we are transforming the columns value into numerical classes from categorical class, in simple word we are apply label encoding on our columns, this process is called Data Transformation.

```
df['Gender'] = df['Gender'].map({'Male':1,'Female':0})
df['Customer Type'] = df['Customer Type'].map({'Loyal Customer':1,'disloyal Customer':0})
df['Type of Travel'] = df['Type of Travel'].map({'Business travel':1,'Personal Travel':0})
df['Class'] = df['Class'].map({'Business':1,'Eco':2,'Eco Plus':3})
df['satisfaction'] = df['satisfaction'].map({'neutral or dissatisfied':0,'satisfied':1})
```

```
[3] Python
```


Below listed All columns are transformed

- Gender
- Customer Type
- Type of Travel
- Class
- Satisfaction

We used `map()` function to transform the column values, we have passed the python dictionary inside the `map()` function, where key is that value, which you want to encode, and value of dictionary representing that value by which you want to replace the key value.

In classification problems, we need to make sure that our data is balanced, meaning each class has a similar number of examples. However, sometimes we encounter imbalanced datasets where one class has many more examples than the others. This can cause our model to favor the majority class and perform poorly on the minority ones. To tackle this issue, we use techniques like adding more examples to the minority class, and other methods are allow to deal with imbalance dataset.

So we plot bar plot in below image, on our target column, and plotting this bar we can easily determine that our dataset is imbalance dataset. Where axis x has category of customer, there are two category of customers are in target column, first one is loyal customer and other hand disloyal customer, by analysing this plot we can easily understand that no of loyal customers is high than disloyal customer. So this is imbalance in our dataset. So before further move forward, we will have to deal with this imbalance of dataset.



And now we have worked with imbalance dataset, and transform our dataset into balance dataset. So as you in above bar chart, we had 20 thousand around disloyal customers, so these are all enough to train our model, for this problem statement. That's why we reduced the no. of loyal customers to make equivalent length for both class.

We have fetched sample of 20 thousand loyal customer data points, to make length equivalent and make the balanced in both class data points, so that our deep learning ANN model will not get confused.

As you can below bar chart image, where both class data point approximately equivalent to each other's. we used seaborn python library to make this bar chart using **catplot()** function, where we set the kind="bar" to make the bar plot.



Finally, we dealt with imbalanced dataset, our next step is to ensure the quality of our data by checking for missing values. Now, let's move on to checking for missing values in the dataset, missing values represent incomplete information within the dataset, which can significantly impact the reliability of our analysis. Therefore, it's essential to perform data quality assurance before proceeding with any further processing, especially when training artificial neural network (ANN) models.

In the image below, we're conducting a check for missing values in our dataset.

So to check the missing value pandas library has function called **isnull()**, which is used to check missing values in the dataframe. In the output of our code, there is only column which is "Arrival Delay in minute" having 94 missing values.

```
# checking missing value in our dataset.
balance_df.isnull().sum()

Gender      0
Customer Type      0
Type of Travel      0
Class      0
satisfaction      0
Age      0
Flight Distance      0
Inflight entertainment      0
Baggage handling      0
Cleanliness      0
Departure Delay in Minutes      0
Arrival Delay in Minutes      94
dtype: int64
```

Let's checkout to missing values record from dataframe, we have filtered the missing value records and display top 5 records as an output, in the below image you can see Arrival Delay in Minute columns has missing records.

We have checked the missing value records and also we have filtered all those data points, who has missing value. And now it's time to deal with missing values. Because for Artificial neural network training, we will have to provide missing value free dataset.

```
#To extract only those records, where the missing value is present.
null_df = balance_df.loc[balance_df.isnull()['Arrival Delay in Minutes']]
null_df.head()
```

	Gender	Customer Type	Type of Travel	Class	satisfaction	Age	Flight Distance	Inflight entertainment	Baggage handling	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes
213	0	1	1	2	1	38	109	5	4	5	31	NaN
1124	1	1	0	2	0	53	1012	4	4	4	38	NaN
1529	1	1	1	1	0	39	733	2	2	3	11	NaN
2108	0	1	0	2	0	24	417	5	2	5	1	NaN
2485	0	1	0	2	1	28	2370	3	4	3	3	NaN

To deal with missing values data science and machine learning, there are various ways of missing value imputation. Such as delete the NaN value records, and fill the missing value by its mean and median based on distribution of column.

A process of deleting NaN (missing) data points from the dataset can result in the loss of information. However, if the percentage of missing data points is relatively low, such as between 1 to 10 percent, and the dataset is large, dropping all missing value records can be a viable option. to remove all the missing data points we used dropna() function from pandas library, dropna() function has by default axis=0, it means that will drop all those rows who

has missing values. After applying this function, you can see in output, now we don't have any missing records.

```
balance_df.dropna().isnull().sum()
## null free data
```

```
[12]
```

Gender	0
Customer Type	0
Type of Travel	0
Class	0
satisfaction	0
Age	0
Flight Distance	0
Inflight entertainment	0
Baggage handling	0
Cleanliness	0
Departure Delay in Minutes	0
Arrival Delay in Minutes	0
dtype: int64	

Till now we have cleaned our raw dataset for the training purpose. After cleaning the dataset next step is the splitting dataset into training and testing part. Before splitting the dataset into training and testing, we will split into “features” (independent variable) and “Target” (dependent variables).

Independent variables (Features or input variables):

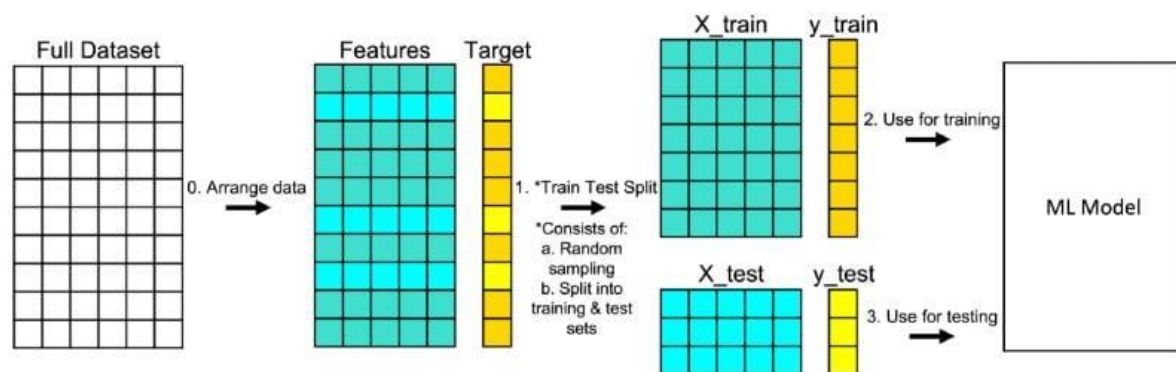
the independent variable, also known as the input variable, feature, or predictor, is the variable that is manipulated or controlled by the researcher or data scientist. It's the variable whose variation or behaviour you want to analyse in relation to other variables. In essence, it's the input to the model that you use to predict or explain the outcome of interest, which is usually referred to as the dependent variable.

For example, in a model predicting house prices, independent variables might include features like the size of the house, the number of bedrooms, the neighbourhood's crime rate, etc. These are factors that you believe could influence the price of a house.

Dependent variables (Target variable):

The dependent variable, also known as the target variable or response variable, is the variable in a statistical or machine learning model that you are trying to predict or explain. It's the variable whose variation or behavior you're interested in understanding in relation to changes in other variables, particularly the independent variables.

In simpler terms, the dependent variable is what you're trying to forecast, explain, or understand based on the values of the independent variables.



You can understand data splitting process in more detail, by interpreting the above image. First we will split dataset into two parts, so first one is features and second one is Target, so features has all columns and variable except only one target variable.

We have hold all features in **X** in variable, and Target variable in **Y** variable. After splitting data into **X** and **Y** variables, we will split it into Training set and Testing, so for training and testing split we have used **train_test_split()** function from **sklearn** library of python, which is used to perform machine learning related task.

Train_test_split() function takes two parameters, where we will specify our **X** and **Y** variables, and to define the ration of splitting, there is one more other parameter, which is **test_size**. So we have defined, **test_size=0.2**, which mean we are going to provide **80%** data for the training purpose and **20%** data for the testing purpose.

X_train and **Y_train** will be used for training purpose, and **X_test** and **Y_test** for the testing purpose.

So how you can be splitting your dataset, into training and testing part, you can follow below written code to split your dataset for splitting.

```
# splitting the dataset into independent and dependent variables
x = balance_df.drop('Customer Type',axis=1)
y = balance_df[['Customer Type']]

print("shape of x :", x.shape , " and shape of y :",y.shape,"\n")
# splitting the dataset into training and testing set
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)

print(" x_train shape :",x_train.shape,"\n","x_test shape :",x_test.shape,
      "\n","y_train shape :",y_train.shape,"\n","y_test shape :",y_test.shape)

[27] ✓ 0.0s

... shape of x : (35269, 11)  and shape of y : (35269, 1)

x_train shape : (28215, 11)
x_test shape : (7054, 11)
y_train shape : (28215, 1)
y_test shape : (7054, 1)
```

Standard scaling of the dataset:

Standard scaling, also known as standardization, is a pre-processing technique used in machine learning to transform the features of a dataset to have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean of each feature from the dataset and then dividing by the standard deviation.

The formula for standard scaling is:

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

σ = Standard Deviation

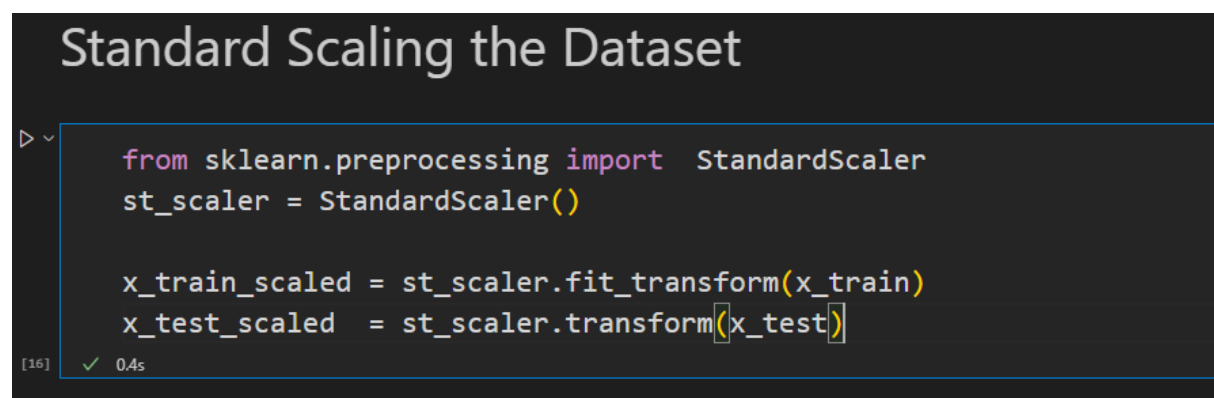
Where is:

- X is the original value of the feature.
- μ is the mean of the feature.
- σ is the standard deviation of the feature.

Ann uses gradient descent algorithms to optimize the weights and bias parameters, so for the smooth convergence, Ann used scaled dataset.

To standard scale a dataset in Python using libraries like scikit-learn, you can use the **StandardScaler** class:

In below mentioned code, we have scaled down our input features. Like **X_train** , **X_test**.



```
Standard Scaling the Dataset

from sklearn.preprocessing import StandardScaler
st_scaler = StandardScaler()

x_train_scaled = st_scaler.fit_transform(x_train)
x_test_scaled = st_scaler.transform(x_test)
```

[16] ✓ 0.4s

Till now we have cleaned, pre-process, transformed, and split our dataset for the ANN training.

Define ANN Architecture using Tensorflow

```
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.models import Sequential
### Defining ANN Model

Ann_Model = Sequential()
# Hidden layers
Ann_Model.add(Dense(units=68,activation='relu',input_dim=x_train_scaled.shape[1]))
Ann_Model.add(Dense(units=32,activation='relu'))
Ann_Model.add(Dense(units=24,activation='relu'))
Ann_Model.add(Dense(units=12,activation='relu'))
#output layers
Ann_Model.add(Dense(units=1,activation='sigmoid'))

#compiling the models
Ann_Model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
Ann_Model.summary()
```

So in above mentioned image, we have defined ANN architecture of our model, where we have defined input, hidden and output layers of ann architecture. So let's understand the briefly our Ann architecture designed.

Input Layer: The input layer is defined implicitly when specifying the `input_dim` parameter in the first hidden layer. The number of units in this layer is determined by the number of features in your input data (`x_train_scaled.shape[1]`).

Hidden Layers: Four hidden layers are added with **68, 32, 24, and 12** units respectively. Each hidden layer uses the ReLU activation function, which is commonly used in deep neural networks to introduce non-linearity.

Output Layer: The output layer consists of a single neuron with a **sigmoid activation** function. Since you're using the binary cross-entropy loss function, a sigmoid activation function is appropriate for binary classification tasks.

Loss Function: The loss function is set to **'binary_crossentropy'**, which is suitable for binary classification problems.

Optimizer: The optimizer is set to **'adam'**, which is an adaptive learning rate optimization algorithm commonly used for training neural networks.

Metrics: The model is configured to monitor **'accuracy'** as the evaluation metric during training.

So when you will execute above mentioned code, you will get below mentioned output due to `summary()` function, because we have written “`Ann_Model.summary()`” function.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 68)	816
dense_1 (Dense)	(None, 32)	2208
dense_2 (Dense)	(None, 24)	792
dense_3 (Dense)	(None, 12)	300
dense_4 (Dense)	(None, 1)	13
Total params: 4129 (16.13 KB)		
Trainable params: 4129 (16.13 KB)		
Non-trainable params: 0 (0.00 Byte)		

Let's understand the model summary,

First Dense layer:

"Ann_Model.add(Dense(units=68, activation='relu', input_dim=x_train_scaled.shape[1]))"

So this is first hidden layer, where we have defined hidden layer parameters

Units= 68 means there 68 neurons at this layer.

input_dim=11 means 11 neurons defined for input layers, and 11 is replacement of this code "x_train_scaled.shape[1]".

Activation function='relu'

Formula to calculate param

$$\text{Params} = (\text{Input features} + 1) * \text{No. of neurons at layers}$$

Let's understand how 816 comes in #param summary.

In our architecture input features = **11**

No. of neurons at layers = **68**

$$\text{Param} = (11+1) * 68 \rightarrow 12 * 68 == 816 \text{ Params}$$

In a same way param calculated for each row and "Total params" is sum of all param.

$$\text{Total params} \rightarrow 4129 = 816 + 2208 + 792 + 300 + 13$$

So, to calculate the memory required for the parameters, you can multiply the total number of parameters by the size of each parameter in bytes:

$$\text{Memory (in bytes)} = \text{Total params} \times \text{Size of each parameter in Memory (in bytes)}$$

Given that each parameter is represented as a **32-bit floating-point number (4 bytes)**, the size of each parameter is **4 bytes**.

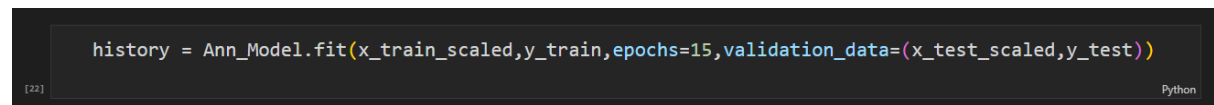
$$\text{Memory (in bytes)} = 4129 \times 4 = \mathbf{16516 \text{ bytes}}$$

To convert bytes to kilobytes (**KB**), you divide by **1024 bytes per kilobyte**:

$$\text{Memory (in KB)} = \frac{16516}{1024} \approx \mathbf{16.13 \text{ KB}}$$

We have understood the model Architecture, and also we have seen how model parameters can be count and how the model memory size measured.

Now we are ready to train our model, on the dataset.



```
history = Ann_Model.fit(x_train_scaled, y_train, epochs=15, validation_data=(x_test_scaled, y_test))
```

So we used **fit()** to fit our ANN model on the dataset, we have passed some parameters in fit function. Let's understand all of these

X train scaled: is our training input features that we have scaled.

Y train: training Target feature variable, or labelled for data point.

epochs: An epoch is one complete pass through the entire dataset during the training of a deep learning Ann model, if **epochs=1**, it means where it combines one time forward and backward propagation on each data points, but we have set **epochs=15**, it means same process of training will be repeat for 15 times, and model will get learn more from dataset.

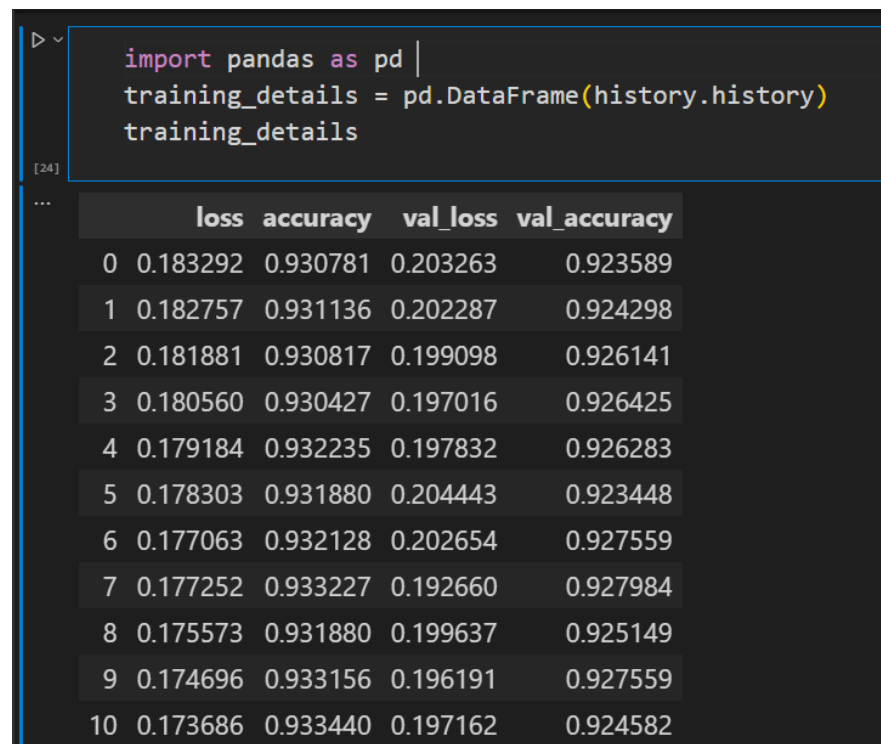
validation data: refers to a separate portion of the dataset that is used to evaluate the performance of a model during training. This data is distinct from the training data and is typically not used to update the model's parameters. Instead, it is used to monitor the model's performance and detect overfitting or under fitting.

During training, the model's performance on the validation data is periodically measured, often after each epoch. This allows practitioners to track how well the model generalizes to unseen data and to make adjustments to hyper parameters or the training process accordingly.

After execution of this cell of code, our model training will begin, and all training information at each epoch will be saved into “**history**” variable.

Once your model training starts, you will get all the training-related information in your console output, where you can find what the accuracy or loss at each epoch corresponding to both dataset like training data and validation data.

You can also analyse your training history, using your **history** where you have held your entire training history. Using this syntax:



```
import pandas as pd |
training_details = pd.DataFrame(history.history)
training_details
```

[24]

	loss	accuracy	val_loss	val_accuracy
0	0.183292	0.930781	0.203263	0.923589
1	0.182757	0.931136	0.202287	0.924298
2	0.181881	0.930817	0.199098	0.926141
3	0.180560	0.930427	0.197016	0.926425
4	0.179184	0.932235	0.197832	0.926283
5	0.178303	0.931880	0.204443	0.923448
6	0.177063	0.932128	0.202654	0.927559
7	0.177252	0.933227	0.192660	0.927984
8	0.175573	0.931880	0.199637	0.925149
9	0.174696	0.933156	0.196191	0.927559
10	0.173686	0.933440	0.197162	0.924582

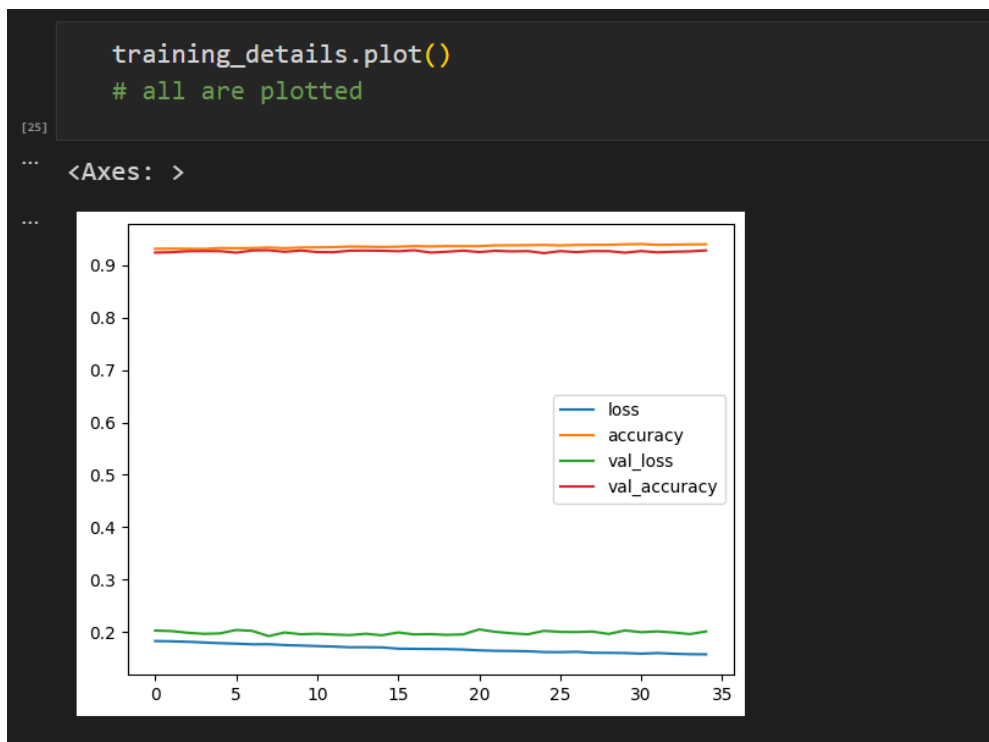
You can see, I have converted history into pandas dataframe, where I got four columns (loss, accuracy, val_loss, val_accuracy) in which all the corresponding information are kept. And index referring the **epoch**, Now you can analyse your model performance at each epoch.

Let's see the result, when I plot all these training information in a chart. And chart is mentioned in below, where you can find four lines for all these corresponding columns,

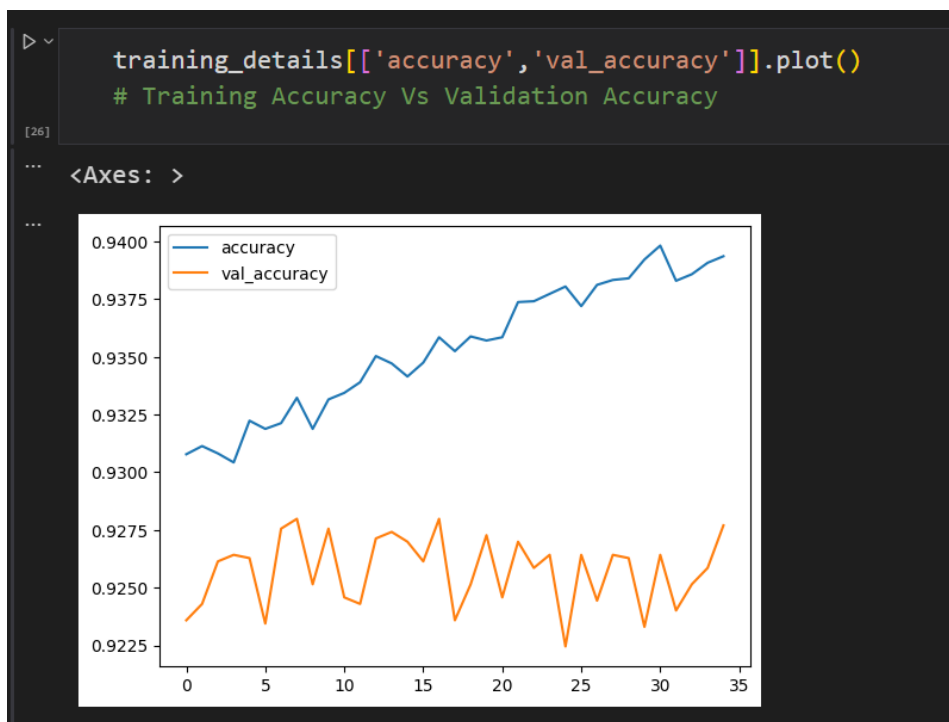
In the below chart you can easily determine the model performance by using this beautiful chart, where (red & orange) line representing the accuracy trend, and (blue & green) lines are representing loss trend on test data and validation data,

And in below chart you can see red and orange lines are on the top of the chart, which means we are getting high accuracy on training and validation data. While blue and green in bottom of the chart which mean our model have lower loss or error rate.

I have trained my model on **epochs=35** so **axis=x** is representing a range from 0 – 35 for epochs.



Let's plot another chart with **accuracy & val_accuracy**, separately so that we can analyse in more depth.



As you can see this chart, where val_accuracy is not more that **92%** but accuracy is always during the training is above to **93%** , val_accuracy is getting improvements after 32 epochs.

Get predictions on unseen data points

Prediction is making an educated guess about what will happen in the future based on available information and patterns observed in past data, so to get the prediction we are calling **predict()** function on our trained model, that have learnt complex relationship from our input data, and now we are going to get prediction on our unseen data points

“**x_test_scaled**” in this variable has our testing data points, which we have not provide to our model during the training for the training purpose. And **x_test_scaled** has only input feature variables information except target columns information, and our target column is in “**y_test**” for **x_test_scaled** data points, that is unseen data points, now we are getting the prediction on **x_test_scaled** unseen data point, using our trained model.

So we have called a **predict()** function to get prediction, and passed the **x_test_scaled** data points, that is unseen data points.

After getting the predicted label, we have stored in “**prediction_label**” and our actual label in “**y_test**”. Using this below code, we can get prediction by the model on our data points.

```
# Get Prediction Probability
y_pred = Ann_Model.predict(x_test_scaled)
prediction_label = (y_pred>0.5).astype('int').ravel()
prediction_label

[40] ✓ 0.7s

... 222/222 [=====] - 0s 2ms/step

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

Model Evaluation:

A confusion matrix is a table used to evaluate the performance of a classification model. It provides a summary of the predictions made by the model compared to the actual ground truth across different classes. The confusion matrix typically consists of four main components:

True Positives (TP): These are cases where the model correctly predicted the positive class. For example, if the model correctly identifies 100 pregnant women out of 150 who are actually pregnant, these would be counted as true positives.

True Negatives (TN): These are cases where the model correctly predicted the negative class. For e.g. These are cases where the model correctly predicts that a woman is not pregnant, and she indeed is not pregnant.

False Positives (FP): These are cases where the model incorrectly predicted the positive class when the actual class was negative. For e.g. if the model incorrectly identifies 50 non-pregnant women as pregnant, these would be counted as false positives. It is

also a (**Type I error**).

False Negatives (FN): These are cases where the model incorrectly predicted the negative class when the actual class was positive. E.g. the model fails to identify women who are actually pregnant. And it is also a (**Type II error**).

Confusion Matrix components are shown in image on it position.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Let's generate confusion Matrix on our result of the model, where first we will have to import confusion Matrix.

```
> ~  
#Importing confusion matrix and classification report  
from sklearn.metrics import confusion_matrix,classification_report  
[ ]
```

So we have imported confusion matrix from sklearn library of python, and inside the **confusion_matrix()**, we will pass actual ground truth and predicted labels.

So as you can see in below image, where I have plotted the confusion matrix, with **heatmap()** function of seaborn library.

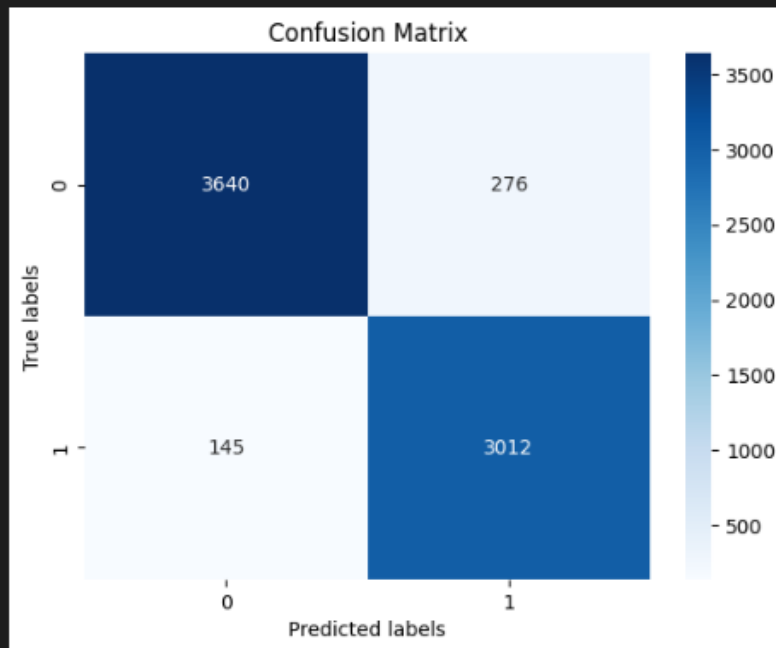
```
# confusion matrix
cm = confusion_matrix(prediction_label,y_test)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

[44]



0.3s



In the above mentioned image of confusion matrix, where TP = 3640, TN = 3012 , FP = 276, FN = 145, according to confusion matrix TP and TN should be high and FP and FN should be low.

According to our confusion matrix, our FP & FN is low, which means our model is trained so good. And to get better performance result of model, we can calculate Precision, Recall, accuracy, F-1 score. You can also calculate all these function manually using formulas in my way that I am calculated value of these functions.

$$\text{Accuracy Formula} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Accuracy} = \frac{3640+3012}{3640+3012+276+145}$$

$$\text{Accuracy} = \frac{6652}{7073} = \mathbf{0.94}$$

$$\text{Precision Formula For Positive} = \frac{TP}{TP+FP}$$

$$\text{Precision +V} = \frac{3640}{3640+276}$$

$$\text{Precision +V} = \frac{3640}{3916} = \mathbf{0.92}$$

$$\text{Precision Formula For Negative} = \frac{TN}{TN+FN}$$

$$\text{Precision -V} = \frac{3012}{3012+145}$$

$$\text{Precision -V} = \frac{3012}{3012+145}$$

$$\text{Precision -V} = \frac{3012}{3157} = \mathbf{0.95}$$

$$\text{Recall Formula} = \frac{TP}{TP+FN}$$

$$\text{Recall} = \frac{3640}{3640+145}$$

$$\text{Recall} = \frac{3640}{3785} = \mathbf{0.96}$$

$$\text{F1-score Formula} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{F1-score} = \frac{2 * 0.92 * 0.96}{0.92 + 0.96}$$

$$\text{F1-score} = \frac{1.7664}{1.88} = \mathbf{0.93}$$

Classification report:

A classification report is a summary of the performance of a classification model, providing a comprehensive evaluation of its predictive capabilities for each class in the dataset. It typically includes several key metrics such as precision, recall, F1-score, and support for each class.

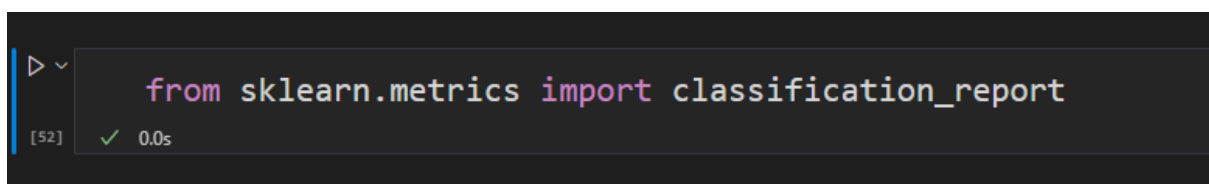
We have already seen that how we can calculate the precision, recall, f1-score from confusion matrix. But classification report you will be find one more other parameter called **Support**.

Support: The number of actual occurrences of each class in the dataset. Support helps to understand the distribution of instances across different classes.

For e.g. Let's say the dataset contains 1000 images in total, with 600 images of cats, 300 images of dogs, and 100 images of birds. In this case, the support values for each class would be:

- Support for cats: 600
- Support for dogs: 300
- Support for birds: 100

If you want to create your classification report for the model you can follow below code. And classification report can import from sklearn.



```
[52] ✓ 0.0s from sklearn.metrics import classification_report
```

And then pass your actual and predicted data point in `classification_report()`.

```
▷ ▾  
#classification report|  
print(classification_report(prediction_label,y_test))  
[48] ✓ 0.0s
```

	precision	recall	f1-score	support
0	0.96	0.93	0.95	3916
1	0.92	0.95	0.93	3157
accuracy			0.94	7073
macro avg	0.94	0.94	0.94	7073
weighted avg	0.94	0.94	0.94	7073

Save and Load Model:

After training your model, we will have to save our model, so that we can use this trained model in further process. To save the model call **save()** on the model, and specify the name of your model "**model_name.h5**". this is extension of tensorflow.

```
#you can save your model,  
Ann_Model.save("custom_satisfaction.h5")  
  
# to load your model  
Ann_Model = load_model('custom_satisfaction.h5')  
[19] ✓ 0.3s
```

With our ANN trained and predictions in hand, we've reached a pivotal milestone in our journey. But our work doesn't end here; it's time to safeguard our progress by saving our model. This ensures that all the effort and learning we've invested in training our ANN are preserved for future use. By saving our model, we create a valuable asset that can be easily accessed and deployed whenever needed.

Regularization Techniques in Deep Learning

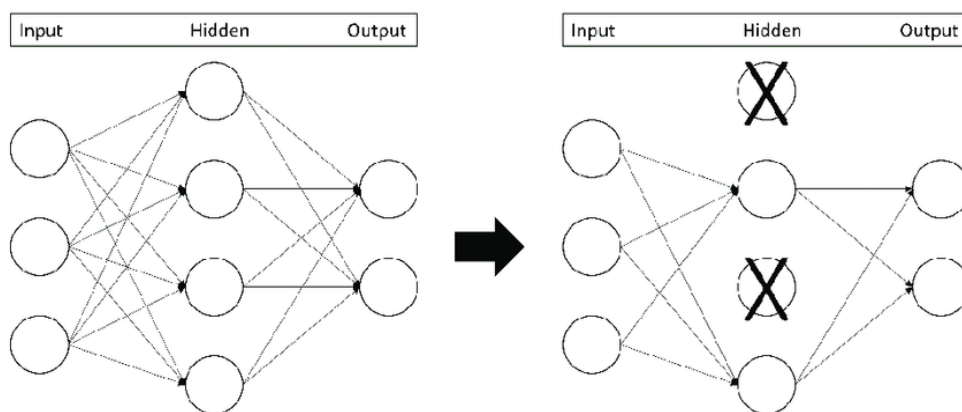
Regularization techniques in deep learning are methods used to prevent overfitting, where the model learns to fit the training data too closely, resulting in poor generalization to unseen data. Overfitting occurs when the model captures noise or random fluctuations in the training data instead of underlying patterns. Regularization techniques introduce additional constraints

or penalties on the model's parameters during training to encourage simpler models that generalize better to new data. Here are some common regularization techniques in deep learning:

- Dropout
- L1 and L2 Regularization
- Early Stopping

Dropout:

In Artificial Neural Networks (ANNs), a dropout layer is a regularization technique used during training to prevent overfitting. During each training iteration, a dropout layer randomly selects a subset of units (neurons) in the layer and temporarily removes them, no of neuron will be selected based on the P value of dropout layer, If the dropout probability P is 0.3 and the total number of units (neurons) is 100, then during each training iteration, on average, $P \times 100 = 0.3 \times 100 = 30$ neurons will be selected to be removed temporarily.



In above mentioned image we have choose p value = 0.5, which means 50% neurons will get selected to temporary remove. So total 4 units of neurons on this hidden layer, and we have set p value = 0.5 means 2 neurons will have got selected to remove.

```
import tensorflow as tf
# Define dropout probability
p = 0.5
# Create the ANN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(4, activation='relu', input_shape=(input_shape,)),
    tf.keras.layers.Dropout(p),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

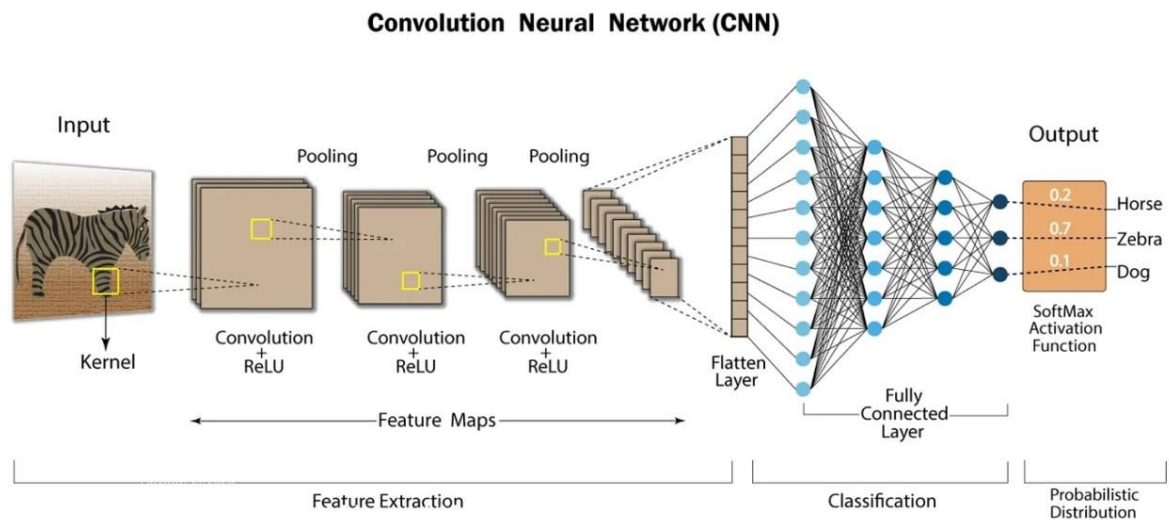
This is code where we show that how you can define the drop out layer in your sequential ann model. In the above mentioned code, we keep 4 neurons on the first hidden layer, and after that we define dropout layer with $P = 0.5$, it means 2 will have got selected.

L1 and L2 Regularization

Early Stopping

Convolutional Neural Network

CNN stands for Convolutional Neural Network, which is a type of deep learning algorithm mainly used for analysing visual imagery. CNNs are particularly powerful in tasks such as image recognition, object detection, and classification.



Here are some key components of CNNs:

Convolutional Layers: These layers apply convolution operations to the input, extracting features through filters or kernels. Each filter learns to detect specific patterns or features in the input data.

Pooling Layers: Pooling layers downsample the feature maps generated by the convolutional layers, reducing the spatial dimensions (width and height) while retaining important information. Common pooling operations include max pooling and average pooling.

Activation Function: After each convolutional and pooling operation, an activation function like ReLU (Rectified Linear Unit) is applied to introduce non-linearity into the network, allowing it to learn complex patterns.

Flattening: Before the fully connected layers, the feature maps are flattened into a single vector, which serves as the input to the dense layers.

Fully Connected Layers: Typically placed towards the end of the network, these layers connect every neuron in one layer to every neuron in the next layer, allowing the network to learn high-level features and make predictions.