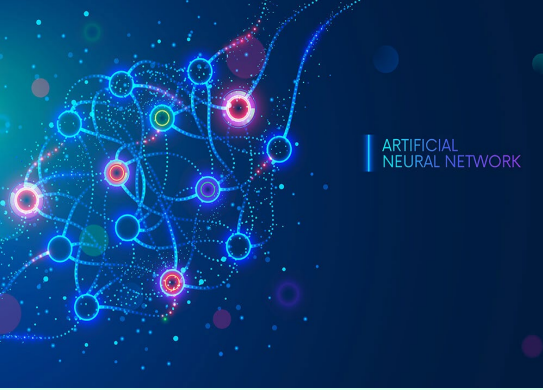




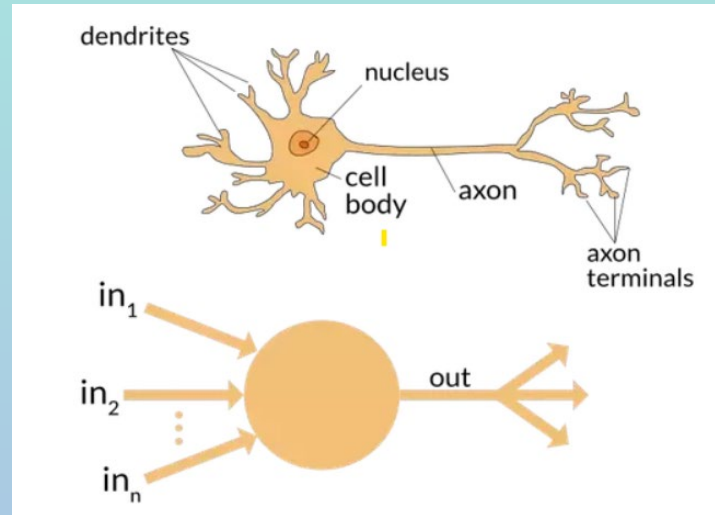
Artificial Intelligence (ACCA7026)

Artificial Neural Networks and Deep Learning

Dr.Seena Joseph

Artificial Neural Networks

- **Neural Network** or **Artificial Neural Network (ANN)**:
Computational model inspired by the function and structure of a biological neural network.



Artificial Neural Networks

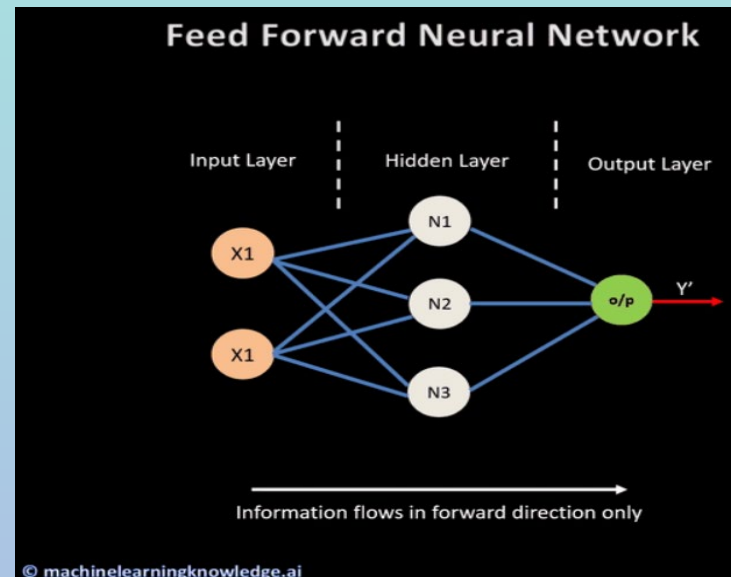
- *Learn it* from the data
 - Acquires knowledge and improves its performance by analysing and processing input data
- *Application* of NN
 - Image classification: Image recognition, object detection, facial recognition
 - Natural language processing: Language translation, text generation, chatbots
 - Speech recognition: voice assistance and transcription services
 - Recommendation system: streaming platforms, personalized content recommendation
 - Medical diagnosis
 - Financial forecasting
 - Autonomous Vehicles
 -

Artificial Neural Networks

- Types of Neural Networks
 - Feed forward Neural Network (FNN)
 - Convolutional Neural Network (CNN)
 - Recurrent Neural Network (RNN)
 - Long Short Term Memory (LSTM)
 - Auto encoders
 - Generative Adversarial Network (GAN)
 - ...

Artificial Neural Networks

- Feed Forward Neural Network
 - This is the most basic neural network architecture, the building block for most other neural networks.
 - Information flows in one direction, from the input layer to the output layer. The connections between nodes do not form a cycle.



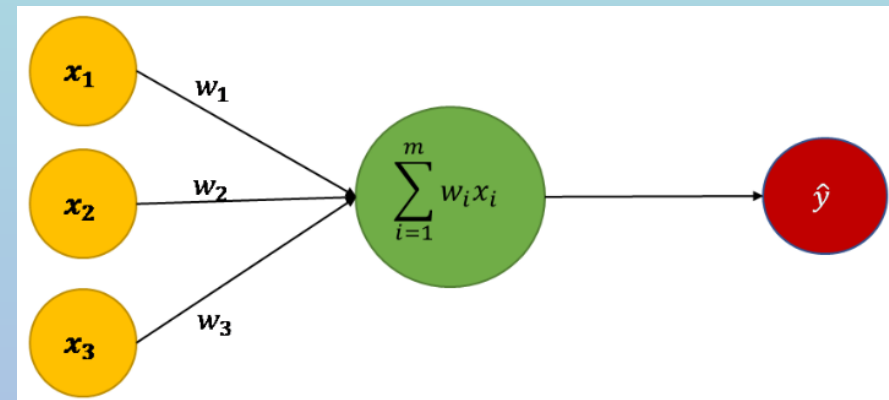
Building blocks of Artificial Neural networks

Neuron – Fundamental building block.

- Receive input signal -> multiply by weights->Weighted sum of inputs-> apply activation function-> Output

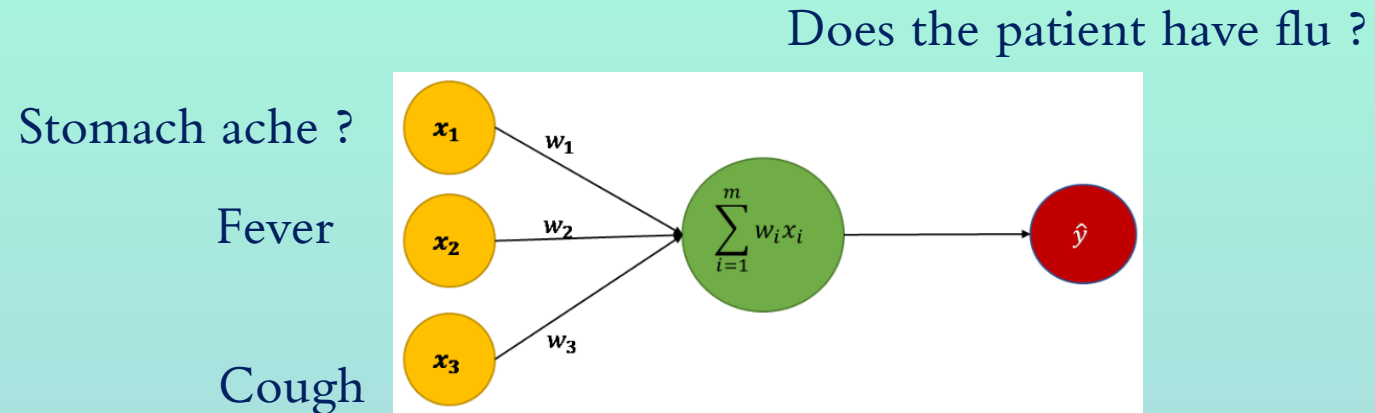
A neuron in a neural network works in three steps:

- Multiply the input signal with the corresponding weights
- Obtaining a weighted sum of the inputs
 - $Z = x_1 * W_1 + x_2 * W_2 + x_3 * W_3$
- Apply an activation function (introduce non-linearity) – this function determine whether the neuron should activate or remain dormant based on the calculated value.



Building blocks of Artificial Neural networks

An Example



- Three inputs : Stomach ache (X1), Fever (X2), and Cough (X3),
- output we're trying to obtain is: **Does the patient have the flu or not?**
- Let's assume that the inputs are in **1**'s and **0**'s, **1 - Yes**, and **0 – No**
- Assumption: stomach ache is not a common symptom of flu, so its weight should be set low and since fever and cough are the symptoms of flu, set the weight set to be high. And a threshold value **5**, determined by an activation function.

	Input	Value	Weight
X1	Stomach ache	YES(1)	-1
X2	Fever	NO(0)	3
X3	Cough	YES(1)	3

$$Z = 1 * -1 + 0 * 3 + 1 * 3 = 2$$

Since the output is less than zero, the activation function of the neuron predicts that the patient is not suffering from flu.

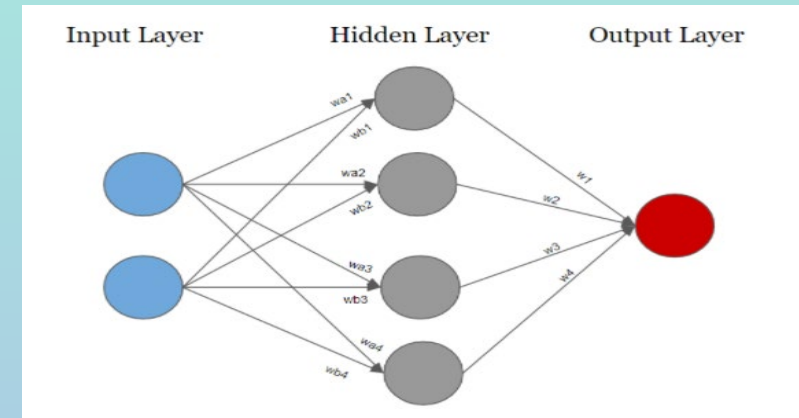
Building blocks of Neural networks

- **Layers**

- Input layer – Vector that describes the feature of the input data (passive)
- Hidden layer – Intermediate layer between the input and output layers
 - Processing and transforming the input data before producing the output
- Output layer – Produces final output

- **Weights and biases**

- Strength of the connection between the neuron
 - Numerical Value
 - Determines the impact of input on the output of the connected neuron
 - Weights are adjusted during training to optimise the network's performance
-
- Biases are additional parameters
 - Each neuron (Except the input layer neuron) has an associated bias
 - It allows neurons to activate even when the weighed sum of their inputs is not sufficient on its own.
 - Helps in producing a non—zero output when input features are zero



Building blocks of Neural networks

Biases

Analogy :

- Think of a neuron like a light.
- The inputs are the switches that control the brightness. If all switches are off, there is no input.
- Normally, the light would stay off. However, the **bias** is like a small built-in brightness that keeps the light slightly on even when all switches are off.

Building blocks of Neural networks

- **Activation function**

- Introduce nonlinearity, and determine the output of the neuron or layers based on the weighted sum of inputs and the bias term. It determines whether a neuron should be activated or not.
- Common types of activation function

- Binary Step Function – A threshold based classifier. Use a threshold value to determine

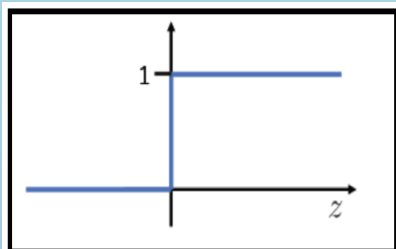
- $f(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$

- Sigmoid Function – Most commonly used in the **output layer** for **binary classification**.

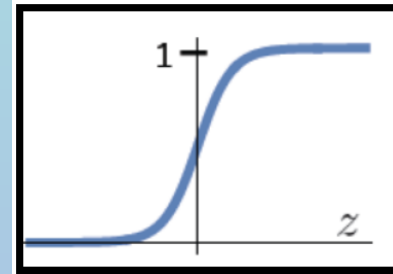
- It provides a smooth output between 0 and 1 [predict the probability]

- Output Range (0,1)

$$f = \frac{e^z}{1 + e^z}$$



Binary step function



Sigmoid function

Building blocks of Neural networks

- **Activation function**

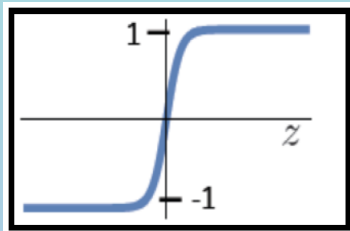
- Tanh (Tangent Hyperbolic Function) – Mathematically shifted version of Sigmoid

- Preferred over sigmoid in **hidden layers**
 - Output Range **$(-1,1)$**

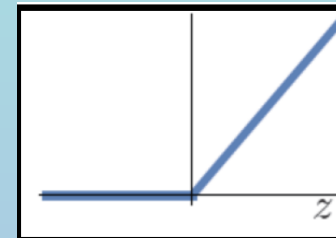
$$f = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- ReLU (Rectified Linear Unit) – Mainly used in hidden layers

- Output Range **$(0, \infty)$** $f = \max(0, Z)$



Tanh



ReLU

Building blocks of Neural networks

- **Activation function**

- **Softmax** - Most commonly used in the output layer for **multiclass classification**. Takes in a vector of raw outputs neural networks and returns probability score. $f(z) = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$
- Output Range **(0,1)**

ACTIVITY

- [Softmax Activation Function: Everything You Need to Know | Pinecone](#)

Hidden Layer - Artificial Neural Networks

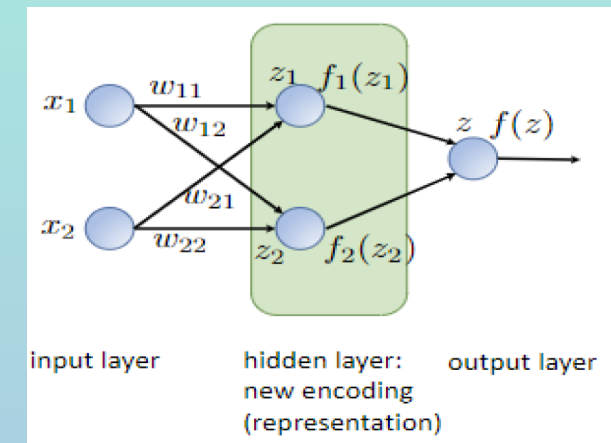
One Hidden layer with two neurons

- Each neuron in the hidden layer learn the weights associated with it.
- Two neurons are essentially independent of each other having different weights and each input is weighted differently.

$$Z_1 = X_1 * W_{11} + X_2 * W_{21}$$

$$Z_2 = X_1 * W_{12} + X_2 * W_{22}$$

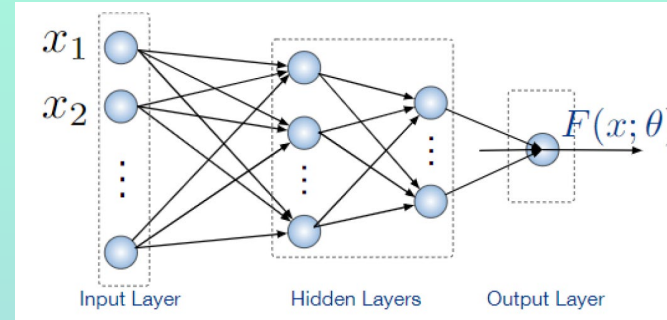
Each neuron captures different characteristics of the input



The output from the hidden layer will be high level representation of the inputs (x_1 and x_2) and that will be the input to the output layer

Hidden Layer - Artificial Neural Networks

Multiple hidden layers.

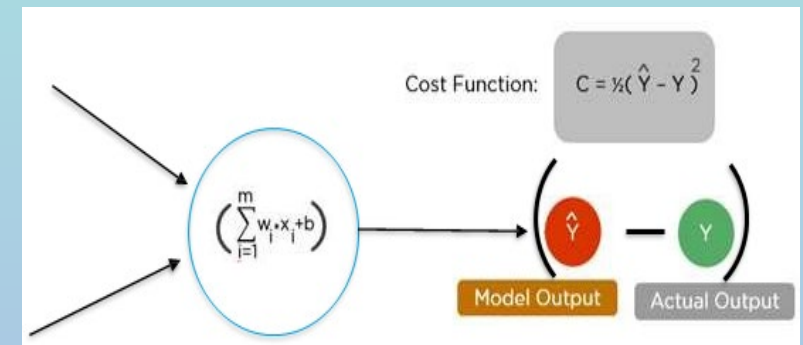


- In a fully-connected feedforward neural network, every neuron in the hidden layer is connected to **all the neurons** in the **next** layer, and every neuron in the hidden layer accepts input from **all the neurons** from the **previous** layer.
- Each of these neurons in the hidden layer will typically **act** as a **linear classifier** and can make simple decisions.
- It takes the weighted combination of inputs and applies activation functions to it.
- When these ensembled neurons are put together, they solve **complicated decisions** by breaking them down into simpler ones.

Forward Propagation

- In the forward propagation, the input data is propagated forward from the input layer through the hidden layer until it reaches the final/output layer where predictions are made.
- At every layer, data gets transformed in three steps in every neuron:
 - ❖ Sum weighted input at every neuron (by multiplying X by the hidden weight W) and the bias
 - ❖ Apply the activation function on the sum
 - ❖ Pass the result to all the neurons in the next layer
- The last layer is the output layer, which may have a sigmoid function (for binary classification) or a softmax function (if the network is a multi-class classifier). The output layer gives the predictions of the neural network.

After getting the predictions, we use an optimisation algorithm that helps us to *minimise* error (cost) function $E(x)$ which is simply a mathematical function dependent on the model's **learnable parameters** which are used in computing the target values (Y) from the set of *predictors* (X) used in the model.

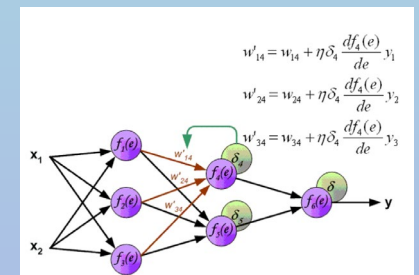


Forward Propagation

- Forward Propagation Steps
 - Calculate the weighted input to the hidden layer by multiplying x_1 (input) by the hidden weight W_{11}
 - Apply the activation function and pass the result to the next layer
 - In the next layer repeat the above sections by replacing the input (x_1) with the output from the previous layer.
 - This is repeated till the output layer.
- The final output from the neural network is called, $\hat{y}$ (Predicted)
- The next step is to calculate the loss / cost
- $$\text{Loss} = (y - \hat{y})^2$$
- The loss should be reduced to zero i.e. y & $\hat{y}$ becomes equal. The role of an optimiser is to reduce the loss/cost.

Back Propagation

- Back propagation is the process of learning that the neural network employs to re-calibrate the weights at every layer and every node to minimise the error in the output layer.
- During the first pass of forward propagation, the weights are random numbers.
- The output of the first iteration is not always accurate. The difference between actual value / class and predicted value / class is the error.
- All the nodes in all the preceding layers contribute to error and hence need to get their share of the error and correct their weights.
- This process of allocating a proportion of the error (error gradient) to all the nodes in the previous layer is called the back propagation.
- The goal of back propagation is to adjust weights in proportion to the error contribution and in iterative process identify the optimal combination of weights.
- At each layer, at each node, the gradient descent algorithm is applied to adjust the weights.

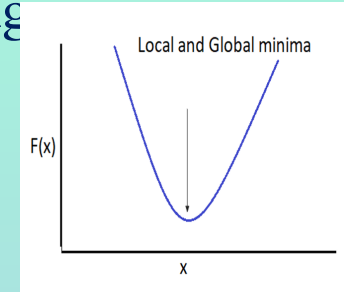


Optimisations

The goal of optimisation is to find a set of weights that minimises the loss function. It is a crucial element that fine tune the network's parameters during training

There are two types of optimisations:

- Convex optimisation
- Non-Convex optimisation

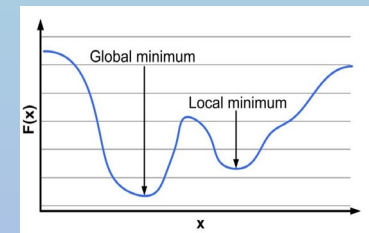


Convex Optimisation

Convex optimisation involves the function that has only one minima, corresponding to Global optimum (minima or maxima). There is no concept of local optima for convex optimisation problems.

Non-Convex Optimisation

Non-convex optimisation involves a function that has multiple optima values, out of which only one is the global optima. Depending on the loss function, it can be very difficult to locate global optima.



Optimizstions

Important Terms

Epoch – The number of times the algorithm runs on the whole training dataset.

Sample – A single row of a dataset or an instance from the dataset that model uses for training.

Batch – It denotes the number of samples to be taken to for updating the model parameters.

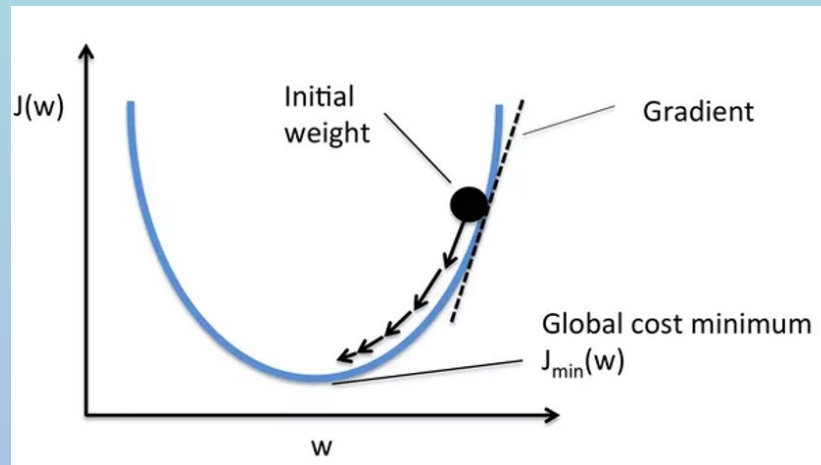
Learning rate – It is a parameter that provides the model a scale of how much model weights should be updated. It determines the step size (the amount by which the weights are updated).

Cost Function/Loss Function – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value.

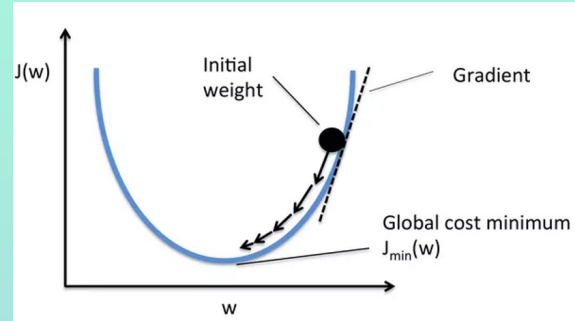
Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.

Optimisations

- Optimisation functions usually calculate the gradient i.e. the partial derivative of loss function with respect to weights, and the weights are modified in the opposite direction of the calculated gradient. This cycle is repeated until we reach the minima of loss function.
- The procedure of repeatedly evaluating the gradient and then performing a parameter update is known as **Gradient Descent**.
- Gradient – direction of increasing error



Optimisations



- Case 1: Positive Gradient
 - Imagine the loss function as a curve. If the gradient at a point is positive, the curve is sloping upward as you increase the weights. Increasing the weights further will increase the loss. To reduce the loss, you need to move in the opposite direction, i.e., decrease the weights.
- Case 2: Negative Gradient
 - If the gradient is negative, the curve is sloping downward as you increase the weights. Decreasing the weights further will increase the loss. To reduce the loss, you need to move in the opposite direction, i.e., increase the weights.

Optimisations

- In most real-life problems, we only have Non-convex functions, but it is hard to find Global minima in these functions. When it comes to using Gradient Descent as an optimisation algorithm for solving non-convex functions, there are two problems associated with that:
- It converges to a minima only slowly
- It gets stuck in the local optima (minima)
- There are different variations of Gradient Descent or Optimisers that can solve the above problems.
- **Popular Optimisers:**
 1. Stochastic Gradient Descent
 2. Stochastic Gradient Descent with Momentum
 3. Mini Batch Gradient Descent
 4. RMSprop
 5. Adam

Optimisations

Stochastic Gradient Descent (SGD)

In SGD a random data points is selected to find the derivative/gradient of the loss function. For example, A dataset consists of 1000 data points, and to calculate the derivative loss function, it will be considering only one data point at a time.

Steps for SGD:

Example:

A data set with n samples, $n=5$;

$D=\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)\}$

1. Randomly pick $((x_1, y_1))$
2. Compute $\widehat{y}_1$ (prediction for x_1)
3. Calculate the loss, L_1 using y_1 and $\widehat{y}_1$
4. Compute the gradient of L_1 with respect to the weights.
5. Update the weights using the gradient and learning rates.
6. Repeat for rest of the data points.
7. Start a new epoch by shuffling the dataset and repeating.

In SGD, convergence to the global minima happens very slowly as it will take a single record in each iteration during forward and backward propagation. As it takes one data point at a time, there will be noise.

Optimisations

Stochastic Gradient Descent (SGD) with momentum

It helps in faster convergence compared to the SGD

Momentum is a combination of how fast something is moving and in which direction is it moving.

When training a model (like using Gradient Descent), weights are adjusted step by step to reduce the loss (error).

Without momentum:

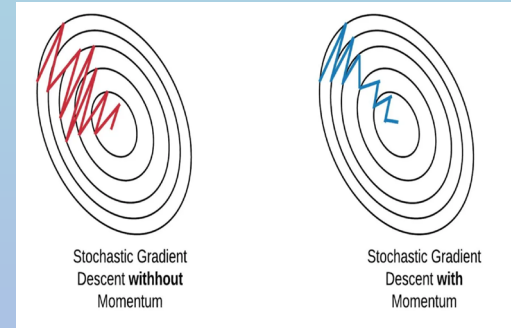
Each step is only based on the current gradient (direction of steepest descent).

This can lead to a lot of back-and-forth zigzagging, especially if the loss function has valleys or noisy gradients.

With momentum:

The optimiser remembers the direction of past steps and combines it with the current gradient.

This makes the updates smoother, reduces zigzagging, and helps the model move faster toward the optimal solution.



Optimisations

Mini Batch Gradient Descent

Instead of taking all the training data (batch gradient) , only a subset of the data is used for calculating the loss function.

Gradient Descent: Uses the entire dataset to calculate the gradient and update the weights in one step. It's slow but gives stable updates.

Mini-Batch Gradient Descent: Divides the dataset into small batches, using each batch to update the weights. It's faster and balances efficiency with accuracy.

Stochastic Gradient Descent (SGD): Uses just one data point at a time to update the weights, making updates very fast but noisier and less stable.

Optimisations

Root Mean Square Propagation (RMSprop)

- It is a gradient based optimisation algorithm.
- It adjust the learning rate to speed up the process.
- It does this by keeping track of the average of the squared gradients for each parameter and uses this to normalize the gradient.
- When the gradient is large, RMSprop reduces the learning rate, preventing the model from making overly large updates (avoiding exploding gradients).
- When the gradient is small, it increases the learning rate to allow for more effective updates (avoiding vanishing gradients).

Exploding gradients: Large gradients lead to unstable updates, causing the model to fail or "explode" in values.

Vanishing gradients: Small gradients lead to tiny updates, making it hard for the model to learn or "vanish" progress.

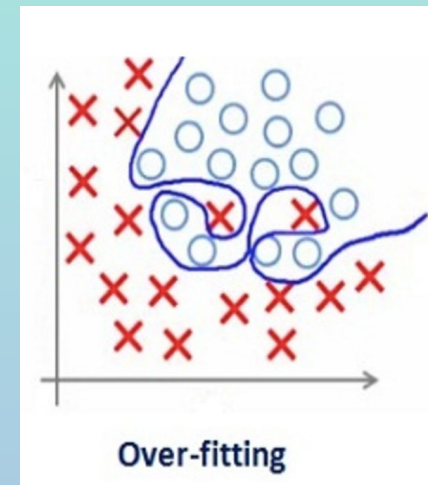
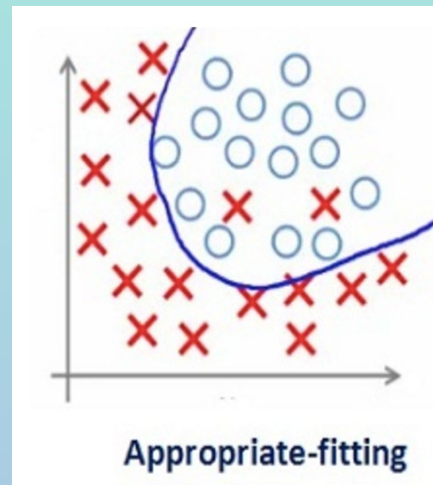
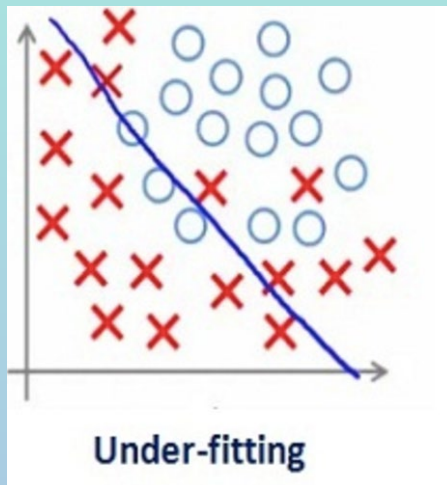
Optimisations

Adaptive Moment Estimation (Adam)

- It combines the advantages of two other algorithms, RMSprop and SDG Momentum, to improve training.
- It keeps track of both the moving average of the gradients (first moment) and the moving average of the squared gradients (second moment).
- This allows Adam to adaptively adjust the learning rate for each parameter, providing more efficient updates.
- The algorithm also includes a bias correction to handle the initial stages of training when the moving averages are biased toward zero.
- As a result, Adam helps the model converge faster and more reliably, making it one of the most popular optimisers for deep learning.

Regularisation Techniques

- Regularisation is a technique used in machine learning and deep learning to prevent overfitting and improve a model's generalization performance. Regularisation add some constraints to the model's complexity during training and ensuring that model to be simpler or more general.



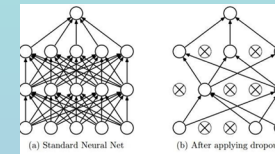
Regularisation Techniques

- L1 Regularisation (Lasso (Least absolute shrinkage and selection operator) regression).
 - It's add the absolute value of the coefficient/weight as a penalty term to the loss function.
 - It helps to penalise large by weights by changing the weights approximately equal to zero, if that feature is not a significant. It enables feature selection and removal.
 - $L_{loss} = L_{original} + \lambda \sum_{i=1}^m |w_i|$ Where λ is the Regularisation parameter, w is the weight and m is number of features.
- L2 Regularisation (Weight Decay)
 - Its add the square of the weights of the loss function as penalty.
 - Shrinks weights towards zero but does not make them exactly zero.
 - Useful when all features are likely relevant, but their impact should be minimised.
 - $L_{loss} = L_{original} + \lambda \sum_{i=1}^m w_i^2$

Regularisation Techniques

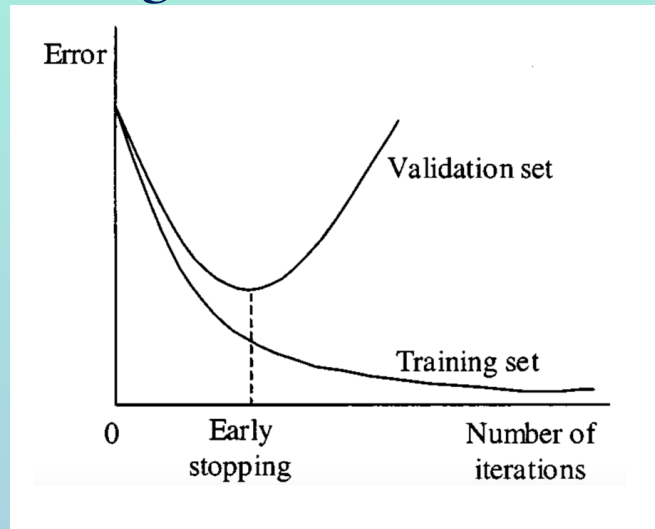
Dropout: A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of the training data. This is handled by using Dropout.

- In dropout, we randomly shut down some fraction of a layer's neurons at each training step by zeroing out the neuron values.
- The fraction of neurons to be zeroed out is known as the dropout rate, say rd .
- The remaining neurons have their values multiplied by $1/(1-rd)$ so that the overall sum of the neuron values remains the same.



Dropout and Early stopping

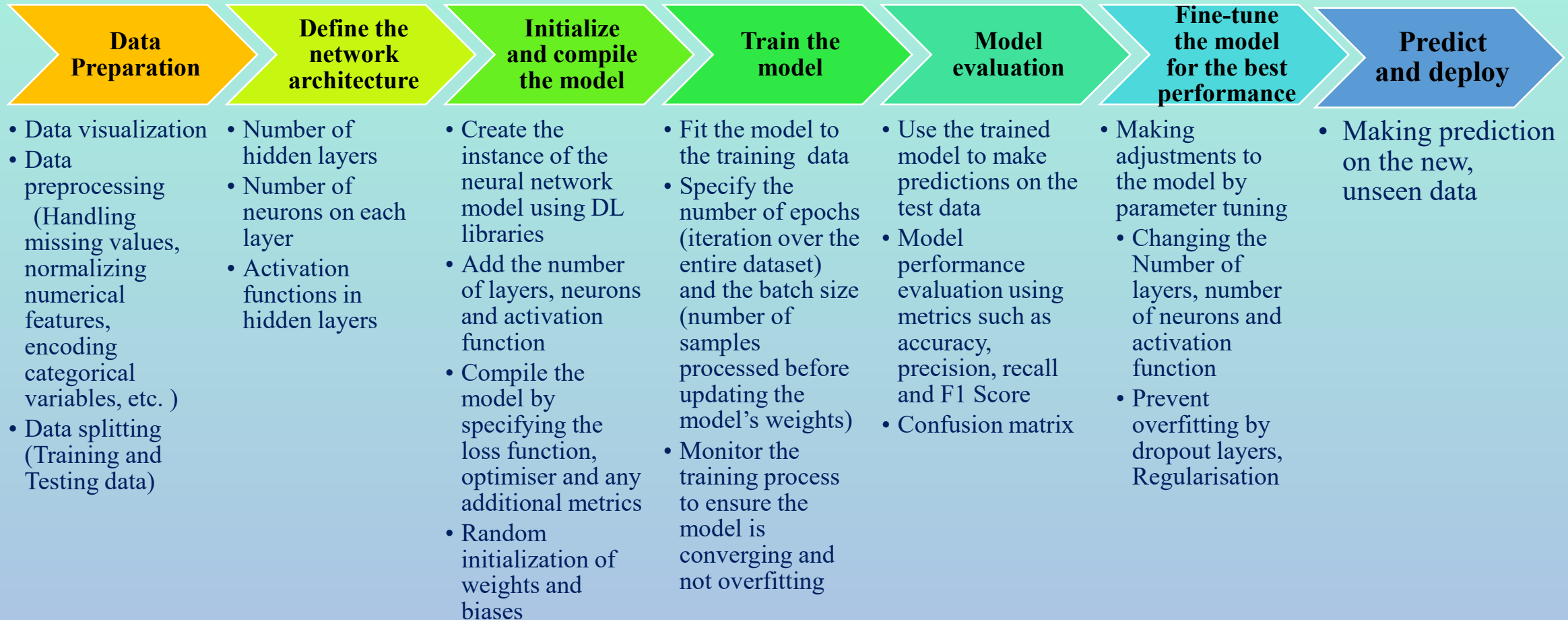
Early Stopping : Early stopping is a technique similar to cross-validation, where a part of training data is kept as the validation data. When the performance of the validation data starts worsening, the model will immediately stop the training.



Loss function Example

- Loss function Examples:
 - Loss function for classification – Binary Cross entropy
 - Loss function for regression – Mean absolute error, Mean Square error, Smooth Mean absolute error.

Building the ANN architecture



Setting up the environment for building an NN in Python

- Why Python
 - Easy-to-Understand Syntax
 - Rich Ecosystem of Libraries and Frameworks (Specifically designed for ML and DL tasks)
 - Numerical computing capabilities
- Install
 - Numpy: Numerical computation
 - Pandas: Data manipulation and analysis
 - Seaborn: data visualization
 - TensorFlow: a Deep learning framework
 - Keras: A high-level neural networks API integrated within TensorFlow
 - scikit-learn: A comprehensive machine learning library that includes various tools for preprocessing data, model selection, and evaluation.

Case-study: Predicting Chances of admission

Context

- The world is fast evolving and constantly seeking the best expertise and experience among people. This inspires people all over the world to excel in their work and pursue higher education degrees that will help them improve their skills and knowledge. As a result, the number of students applying for Master's programs has significantly increased.
- The present admission dataset was developed to forecast admissions to the University of ABC. It was created to assist students in narrowing down their university choices based on their profiles. The expected production offers them a good sense of their chances of acceptance.
- **Objective:** to build a neural network-based classification model to estimate a student's chances of admission to ABC University.

Case-study: Predicting Chances of admission

- Implementation in Python
- Predicting Admission Chance - Jupyter Notebook

Extra material

- [A Neural Network Playground](#)
- [Softmax Activation Function: Everything You Need to Know | Pinecone](#)
- [Regularisation in Machine Learning – GeeksforGeeks](#)
- [\(19\) Understanding Hidden Layers in a Neural Network for Machine Learning Purposes | LinkedIn](#)