AdTrans project, 2023-1-PL01-KA220-HED-000158917

ADTRANS
ADVANCED DATA ANALYSIS FOR TRANSPORTATION AND MOBILITY ACADEMIC TEACHERS

Session 1

Introduction to Parallel Computing & Machine Learning

Paraskevi Fasouli

Co-funded by the European Union

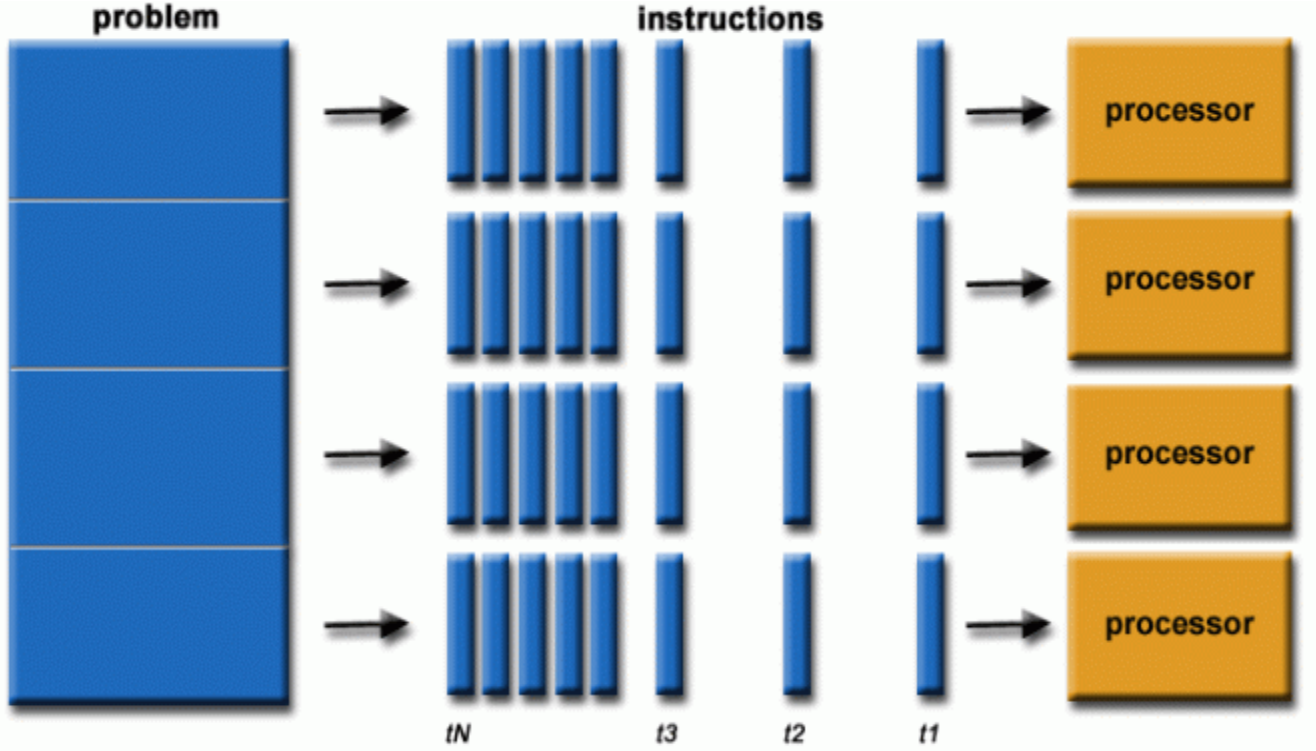
Part 1:

Introduction to Parallel Computing

Overview

- Parallel Computing
- Neurocomputing
- NN Key Concepts
- Historical Timeline
- Advantages of Neurocomputing
- Applications in Robotics

What is Parallel Computing?



Parallel computing is a type of computation in which many calculations or processes are carried out **simultaneously**.

It involves the simultaneous execution of multiple tasks, with the goal of solving a problem more quickly or efficiently than would be possible with a sequential, single-core approach.

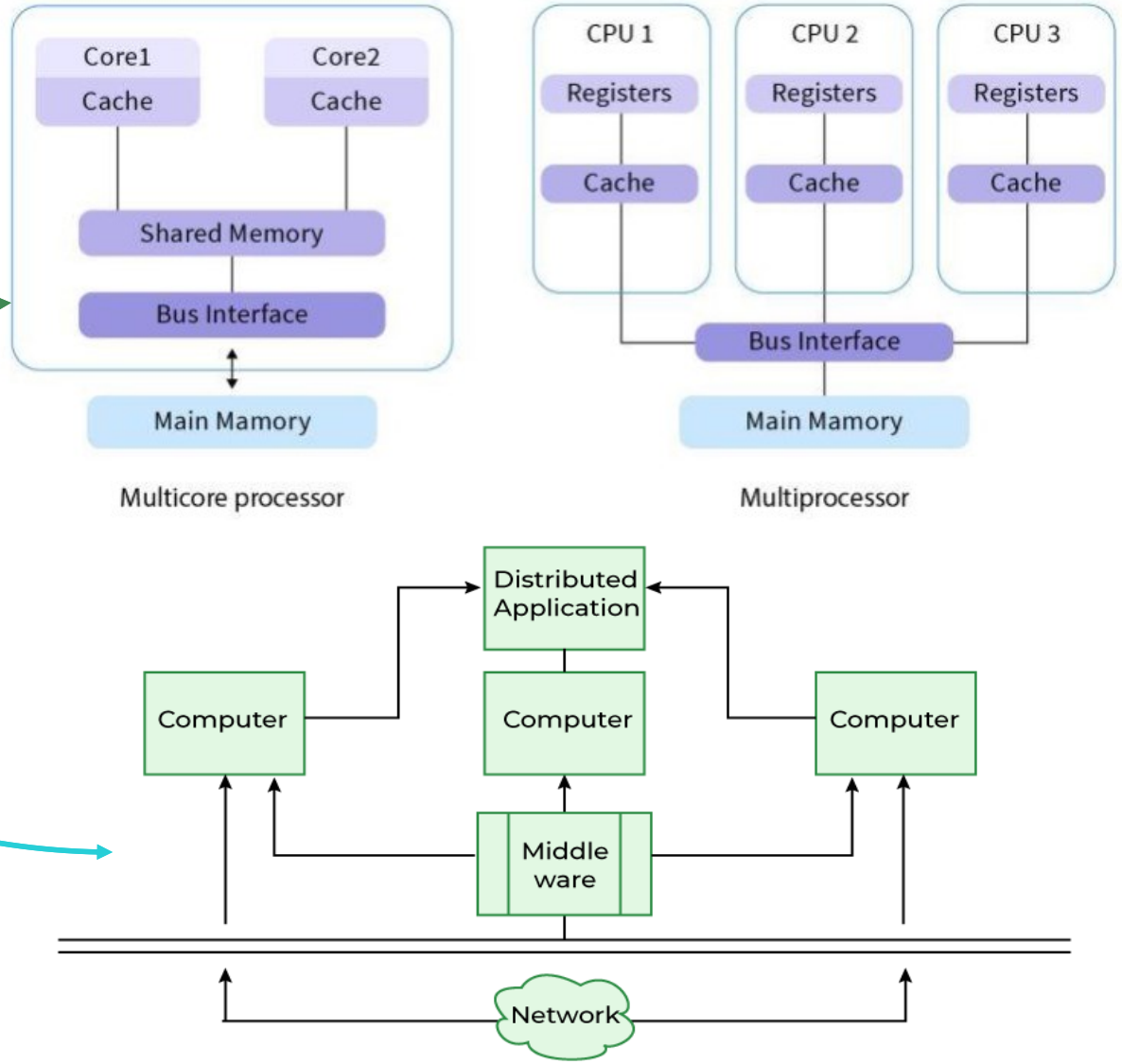
Large problems can often be divided into **smaller ones**, which can then be solved at the same time.

What is Parallel Computing?

It is effective for tasks that can be decomposed into independent subtasks, allowing for concurrent processing and faster problem-solving.

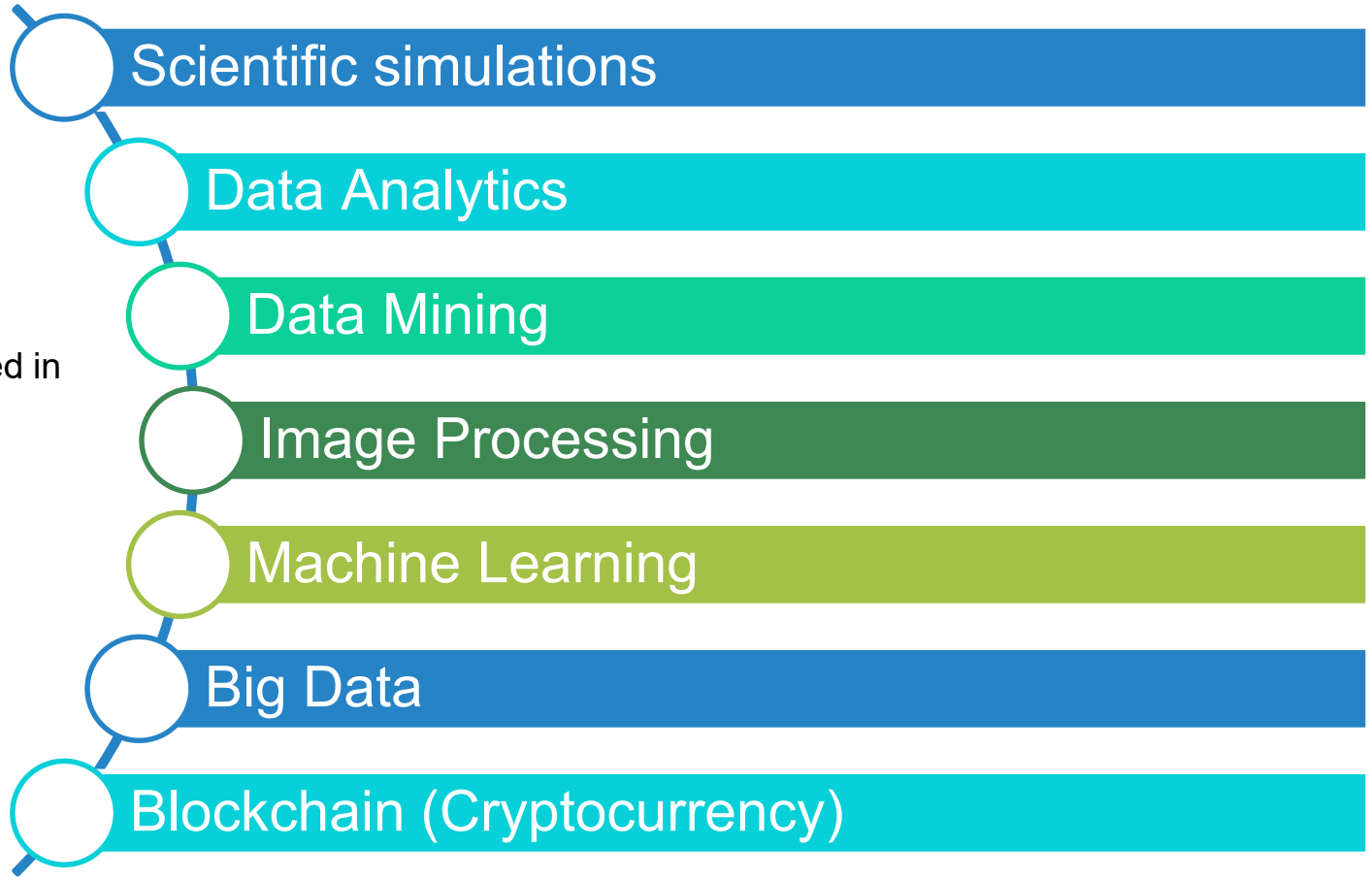
Parallel computing can be implemented on:

- **a single machine with multiple processing cores** or on
- **a distributed network of interconnected computers**



What is Parallel Computing?

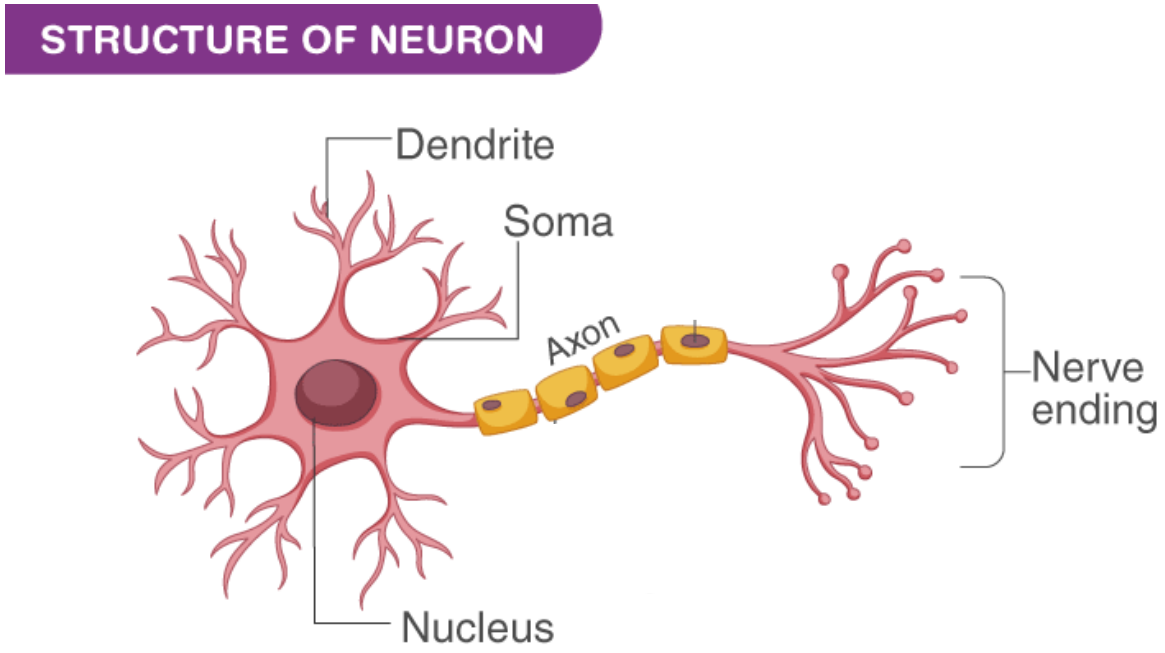
Parallel computing is widely used in various domains:



What is Neurocomputing?

Neurocomputing, also known as neural computing or neural networks, refers to the study and application of algorithms inspired by the structure and function of the human brain.

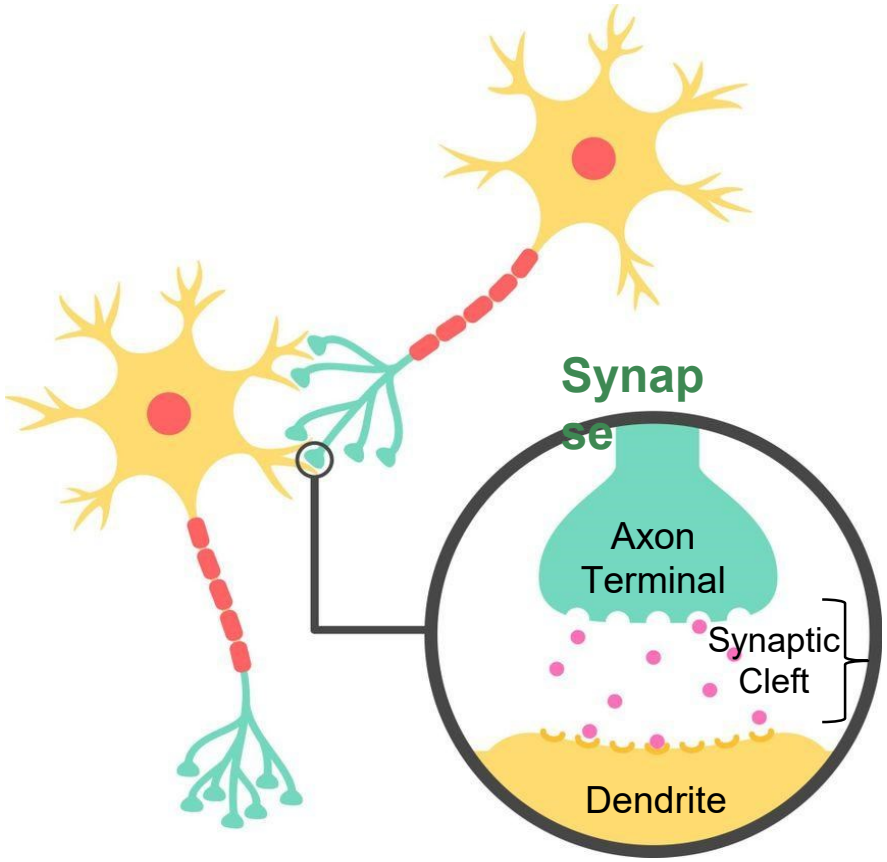
Neural Network of our Brain



Neurons can only be seen using a microscope and can be split into three parts:

- **Soma (cell body)** — this portion of the neuron receives information. It contains the cell's nucleus.
- **Dendrites** — these thin filaments carry information from other neurons to the soma. They are the "input" part of the cell.
- **Axon** — this long projection carries information from the soma and sends it off to other cells. This is the "output" part of the cell. It normally ends with several synapses connecting to the dendrites of other neurons.

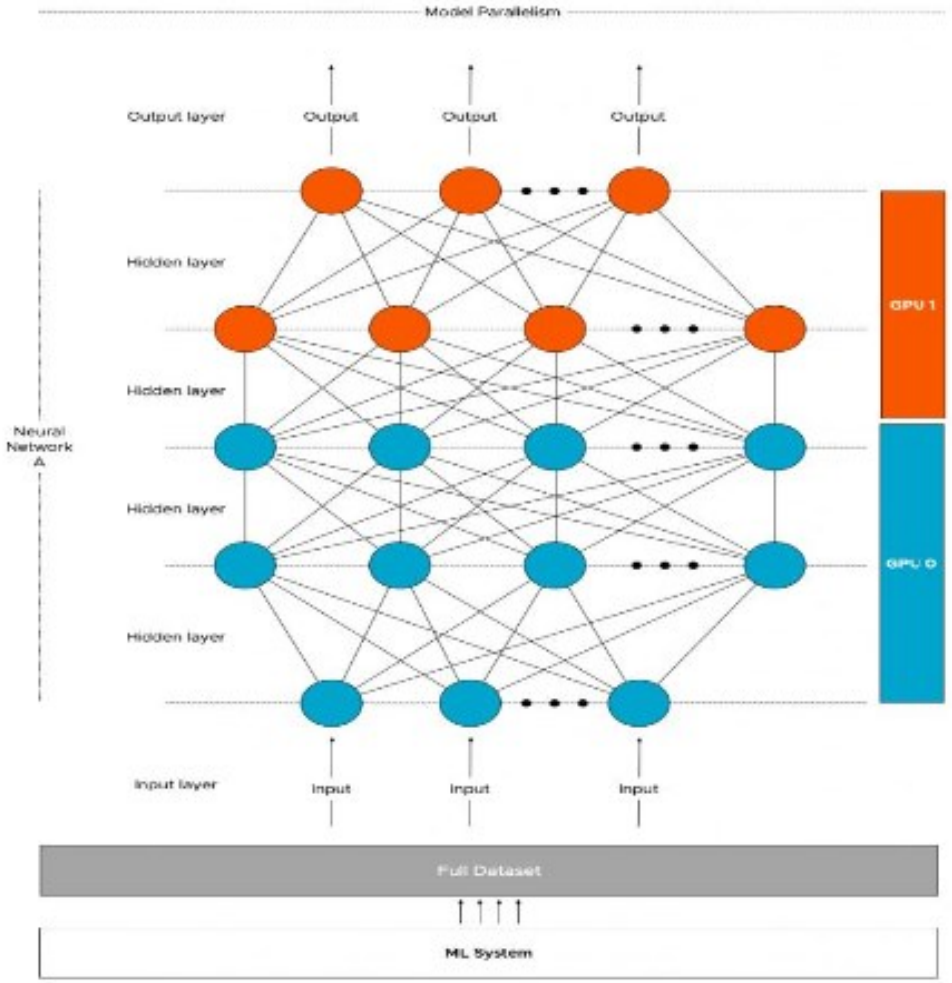
Neural Network of our Brain



Neurons are connected to each other and tissues so that they can communicate messages.
They do not physically touch — there is always a gap between cells called a **synapse**.

Synapses can be **electrical** or **chemical**. The signal that is carried from the first nerve fiber (presynaptic neuron) to the next (postsynaptic neuron) is transmitted by an electrical signal or a chemical one.

Neural Network As Parallel System

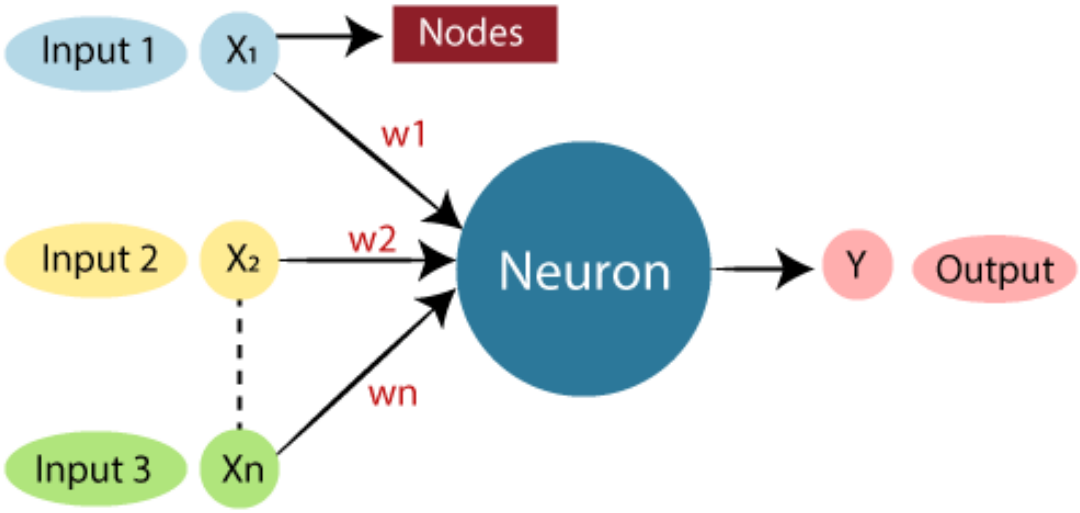


The connection between parallel computing and neurocomputing lies in the application of parallel processing techniques to enhance the training and deployment of neural networks.
Neural networks, especially deep learning models, often involve complex computations that can benefit significantly from parallelism.

Neural Network as Computing System

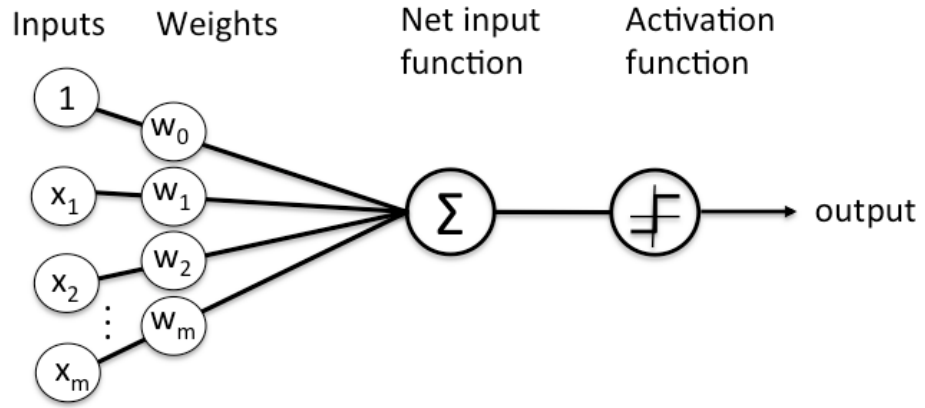
Processing in a neural network is done not by one but by a great number of processors (neurons). They are intricately connected with each other (therefore the name 'network model'), and each of them works by itself, independently of the others.

Each processor can only receive **signals** from other processors, **weight** them (i.e. , assign them different strengths), compute a value called its **activation**
and send an **output** signal, depending on its activation, to one or more other processors.
The storage capacity of a processor includes remembering the weights assigned to the inputs and the previous activation and output values.



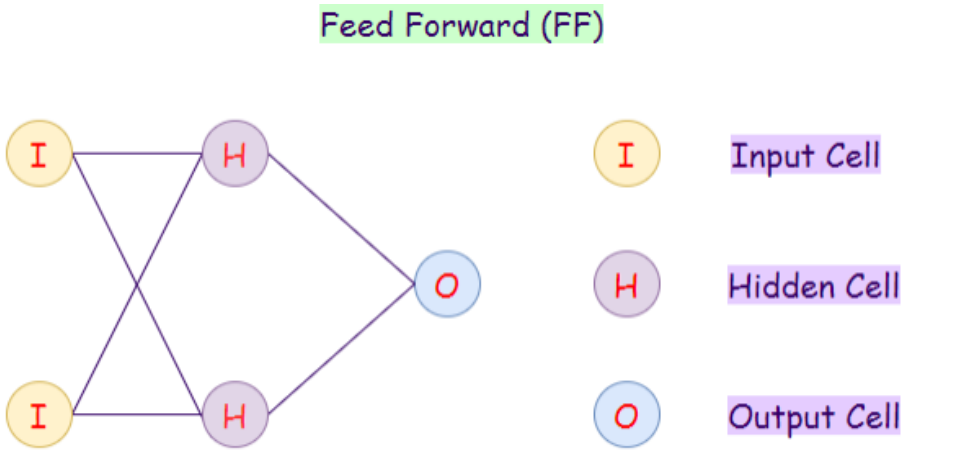
Key Concepts

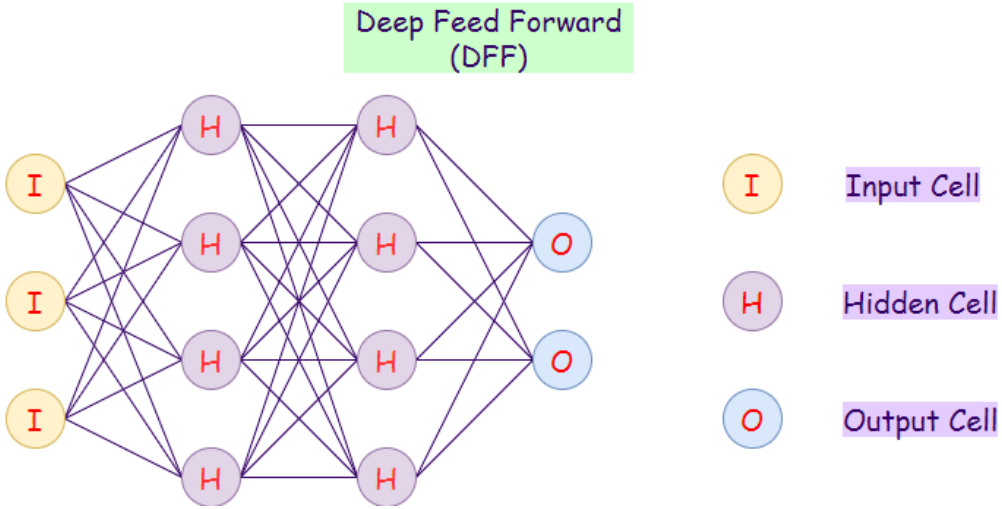
1. **Neurons:**
   - Neurons are fundamental units in neural networks analogous to biological neurons. They receive input signals, process them using weights, and produce an output.
2. **Synapses:**
   - Synapses represent the connections between neurons. Each synapse has a weight that modulates the strength of the connection. Learning in neural networks involves adjusting these weights.
3. **Activation Functions:**
   - Activation functions determine the output of a neuron based on its input. Common functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU), introducing non-linearities critical for learning complex patterns.



Key Concepts

4. **Shallow Feedforward Networks:**
   - In feedforward neural networks, information travels in one direction—from input to output. Each cell processes the input, and the final layer produces the output. (1 hidden layer)
5. **Backpropagation:**
   - Backpropagation is a supervised learning algorithm used to train neural networks. It involves adjusting weights in reverse order from output to input, minimizing the difference between predicted and actual outputs.
6. **Deep Feedforward Networks:**
   - Deep neural networks have many hidden layers. This enables the network to learn hierarchical representations of data, capturing complex features.





Historical Timeline

Milestones

MCCULLOCH & PITTS MODEL

Warren McCulloch and Walter Pitts introduce the first mathematical model of a neural network, laying the theoretical groundwork for later developments.

1943

PERCEPTRON

Frank Rosenblatt develops the perceptron, an early form of a neural network capable of learning and making binary decisions. It becomes a significant milestone in

1957

NEAREST NEIGHBOR

The nearest neighbor algorithm was created, which is the start of basic pattern recognition. The algorithm was used to map routes.

1969

BACKPROPAGATION ALGORITHM

David Rumelhart, Geoffrey Hinton, and Ronald Williams published a paper introducing the backpropagation algorithm, a critical advancement enabling efficient training of multi-layer neural networks.

1986

RENAISSANCE OF NNs

Renewed interest in neural networks emerges, fueled by advancements in computing power, the availability of large datasets, and improved training algorithms. (Reinforcement Learning, SVMs, LSTMs)

1990s

Historical Timeline

Milestones

LENET-5

Yann LeCun introduces LeNet-5, a convolutional neural network (CNN) designed for handwritten digit recognition. It becomes a foundation for later image recognition models.

1998

DEEP BELIEF NETWORKS

Geoffrey Hinton and his team introduce deep belief networks, opening the door to deep learning with unsupervised learning.

2006

IMAGENET COMPETITION

The ImageNet Large Scale Visual Recognition Challenge sees a breakthrough as the deep convolutional neural network (CNN), AlexNet, outperforms traditional computer vision methods, sparking the deep learning revolution.

2012

GATED RECURRENT UNITS

Kyunghyun Cho and his team introduce GRUs, a type of recurrent neural network (RNN), improving the modeling of sequential data.

2013

GENERATIVE ADVERSARIAL NETS

Ian Goodfellow and his collaborators introduce GANs, a novel framework for training generative models, enabling the generation of realistic synthetic data.

2014

Historical Timeline

Milestones

RESIDUAL NETS

Researchers at Microsoft introduced ResNets, a type of deep neural network architecture that significantly eases the training of very deep networks.

2015

ALPHAGO VICTORY

DeepMind's AlphaGo defeats the world champion Go player, 2016 **demonstrating the ability of deep learning to tackle complex strategic games.**

TRANSFORMERS

A team at Google Brain invented the transformer architecture, which allows for faster parallel training of neural networks on sequential data like text.

2017

BERT

BERT, a transformer-based model developed by Google, revolutionizes natural language processing (NLP) by capturing contextual information bidirectionally, enhancing language

2018

GENERATIVE PRETRAINED TRANSFORMER

OpenAI introduces GPT-3, a state-of-the-art autoregressive language model that uses deep learning to produce a variety of computer codes, poetry, and other language tasks almost indistinguishable from those written by humans.

2020

016

Historical Timeline

Milestones

CLIP

OpenAI introduces Contrastive Language-Image Pre-training (CLIP), demonstrating a model capable of understanding and generating text based on images and vice versa, showcasing cross-modal learning.

2021

CHATGPT

ChatGPT, an AI chatbot developed by OpenAI, debuts in November 2022. It is built on top of GPT3.5 model.

2022

STABLE DIFFUSION

Stability AI developed Stable Diffusion, a latent diffusion model, a kind of deep generative artificial neural network. It is primarily used to generate detailed images conditioned on text descriptions .

2023

GEMINI

Google releases Gemini 1.0 Ultra, a family of 3 multimodal large language models of decoder-only transformer architecture.

2024

SORA

OpenAI publicly announces Sora, a text-to-video model for generating videos up to a minute long.

2024

Key Advantages of NNs

Adaptability:

- Neural networks adapt to changes in data by adjusting their internal parameters. This adaptability makes them suitable for tasks where the relationship between inputs and outputs is dynamic.

Parallel Processing:

- Neural networks process information in parallel, mimicking the brain's ability to perform multiple computations simultaneously. This parallelism enhances computational efficiency.

Learning from Data:

- Neural networks learn patterns and relationships from data. Training involves presenting the network with examples, adjusting weights based on errors, and repeating this process iteratively.

Pattern Recognition & Feature Extraction:

- Neural networks excel at recognizing patterns in data, making them valuable for tasks such as image and speech recognition, where complex patterns need to be identified. They can also extract relevant features from raw data, reducing the need for manual feature engineering.

Generalization:

- Neural networks generalize their learning to unseen data. They avoid memorizing specific examples and instead learn underlying patterns, enhancing their ability to make accurate predictions on new inputs.

Bias and Weights:

- Neurons have biases and weights that determine their influence on the output. Bias allows for fine-tuning, while weights adjust the strength of connections between neurons.

Key Advantages of NNs

Integration with Big Data:

- Neural networks can leverage big data for training, enabling them to learn intricate patterns and relationships from vast datasets.

Transfer Learning:

- Neural networks support transfer learning, allowing pre-trained models on one task to be fine-tuned for another task with less available data. This is especially valuable in scenarios where labelled data is scarce.

Prediction Accuracy:

- In many cases, neural networks can achieve high prediction accuracy, outperforming traditional algorithms, especially in complex and high-dimensional data.

Real-world Applications:

- Neurocomputing has found successful applications in areas such as image and speech recognition, autonomous vehicles, medical diagnosis, financial forecasting, and more, showcasing its practical utility.

Real-Time Processing:

- Some neural network architectures, especially those optimized for inference on specialized hardware, can achieve real-time processing, making them suitable for applications with low latency requirements

NN Tasks in Robotics

Object Recognition and Manipulation:

- Neural networks are used to recognize and manipulate objects in a robot's environment. This includes tasks such as identifying objects, determining their position, and planning manipulation strategies.

Sensor Data Processing:

- Neural networks process data from various sensors, such as cameras, lidar, and depth sensors. This allows robots to interpret and make decisions based on sensory information, enhancing their perception capabilities.

Path Planning & Navigation:

- Utilizing neural networks for efficient and adaptive path planning. They can learn and adapt to complex environments, avoiding obstacles and optimizing routes for efficient movement.

Gesture and Speech Recognition:

- Neural networks enable robots to recognize and respond to human gestures and speech. This is valuable in human-robot interaction scenarios, including collaborative tasks and service-oriented applications.

Applications in Robotics

Autonomous Vehicles and Drones:

- In the realm of autonomous vehicles and drones, neurocomputing is used for tasks such as image recognition, obstacle avoidance, and decision-making. Neural networks contribute to making these systems more adaptive and responsive to dynamic environments.

Robotics in Manufacturing:

- Neural networks are employed in robotic systems for tasks like quality control, defect detection, and assembly line optimization. Robots equipped with neurocomputing capabilities can learn from variations in manufacturing processes.

Prosthetics and Exoskeletons:

- Neural networks play a crucial role in the development of advanced prosthetics and exoskeletons. By connecting with the user's nervous system, these devices can respond more intuitively to user movements and intentions.

Humanoid Robotics:

- Humanoid robots benefit from neurocomputing for tasks that require human-like cognitive abilities, such as facial recognition, emotion detection, and natural language processing.

Applications in Robotics

Rehabilitation Robotics:

- Neural networks are used in rehabilitation robots to tailor therapy programs based on the patient's progress and adapt to individual needs, providing personalized rehabilitation exercises.

Swarm Robotics:

- In swarm robotics, neural networks are implemented to coordinate the actions of multiple robots, allowing them to collaborate and achieve collective objectives, such as exploration, search and rescue, or environmental monitoring.

Robotic Learning from Demonstration (LfD):

- LfD involves robots learning new tasks by observing and imitating human demonstrations. Neural networks facilitate the learning process, enabling robots to generalize from observed behaviours.

Cognitive Robotics:

- Cognitive robots leverage neurocomputing to simulate cognitive processes, enabling reasoning, decision-making, and learning from experience. This is particularly useful in creating robots that can adapt to changing environments and unforeseen situations.

Part 2:

Machine Learning & Neural Networks

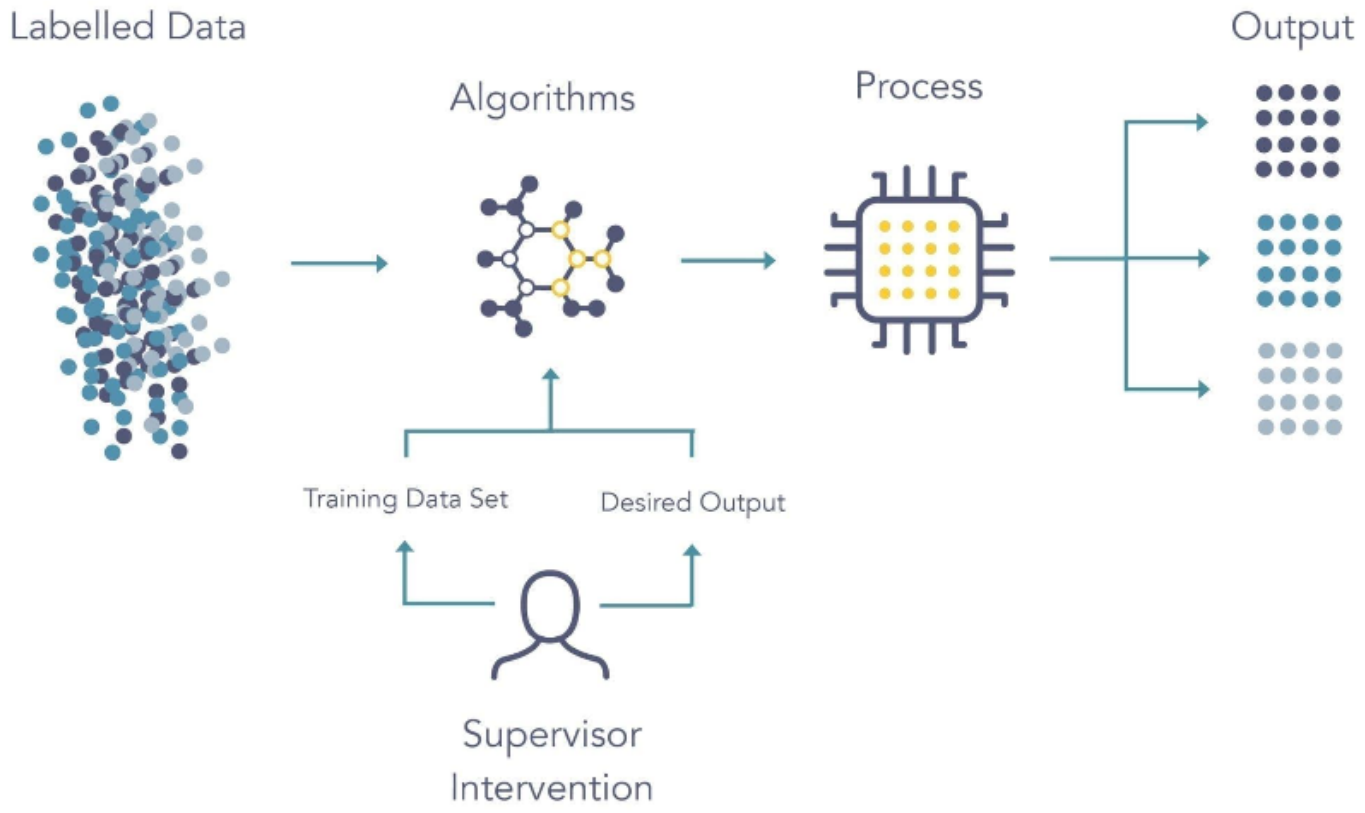
Overview

Machine Learning Definition

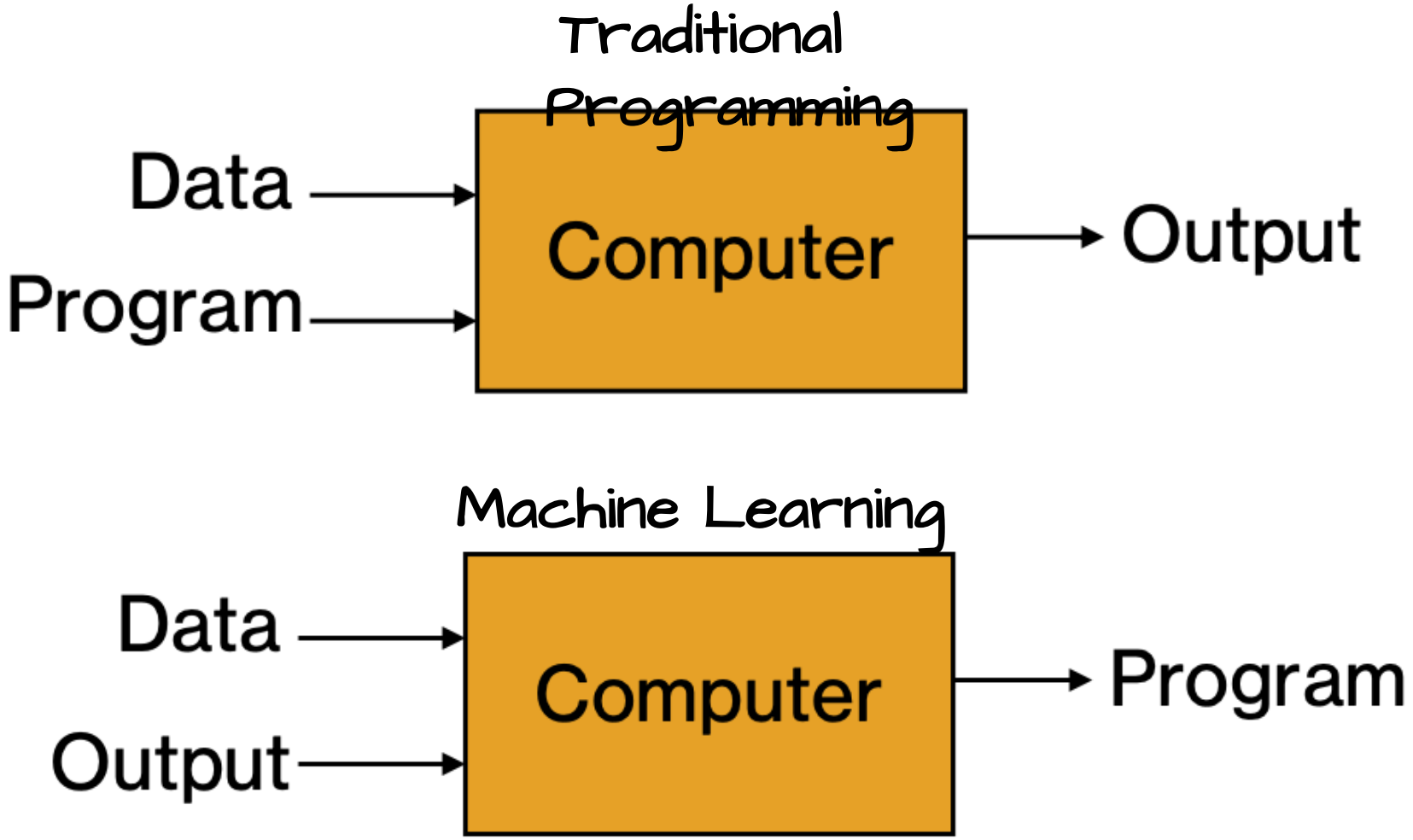
Over/Underfitting Problem

Basic Types of Neural Nets

What is Machine Learning?

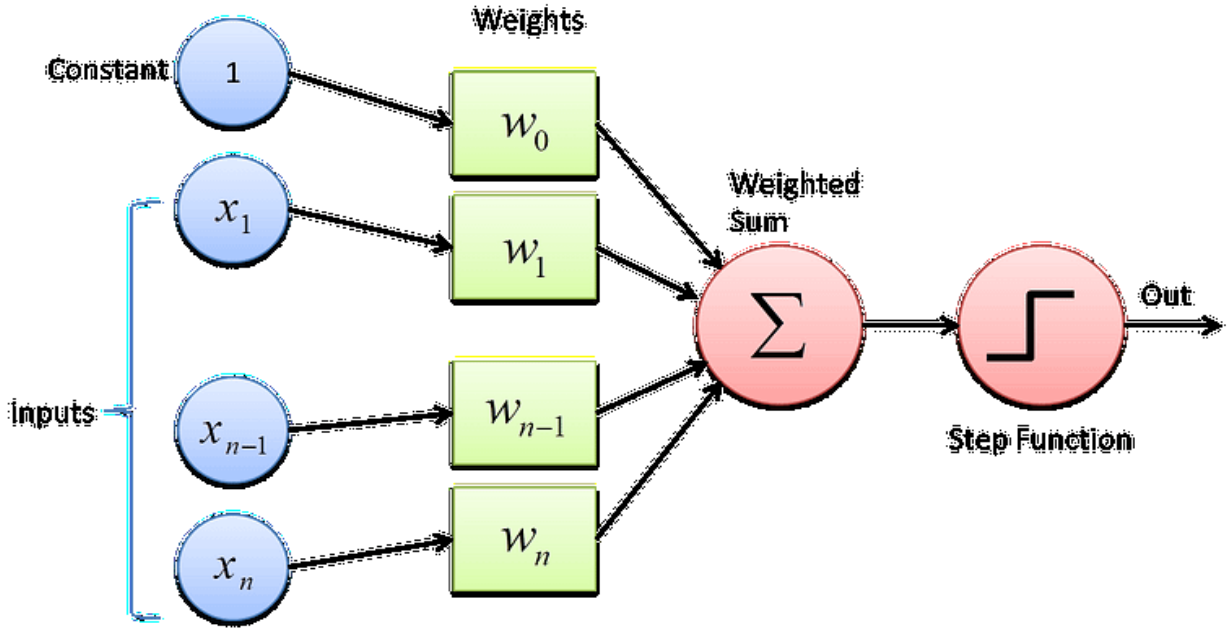


What is Machine Learning?



Perceptron

Perceptron = a single neuron



It receives one or more inputs that are called **nodes**.
The nodes have both a **value** and a **weight**.
Weights show the strength of each node.
A higher value means the input has a **stronger influence** on the output.

The perceptron calculates the **weighted sum** of its inputs.
It multiplies each input by its corresponding weight and sums up the results.

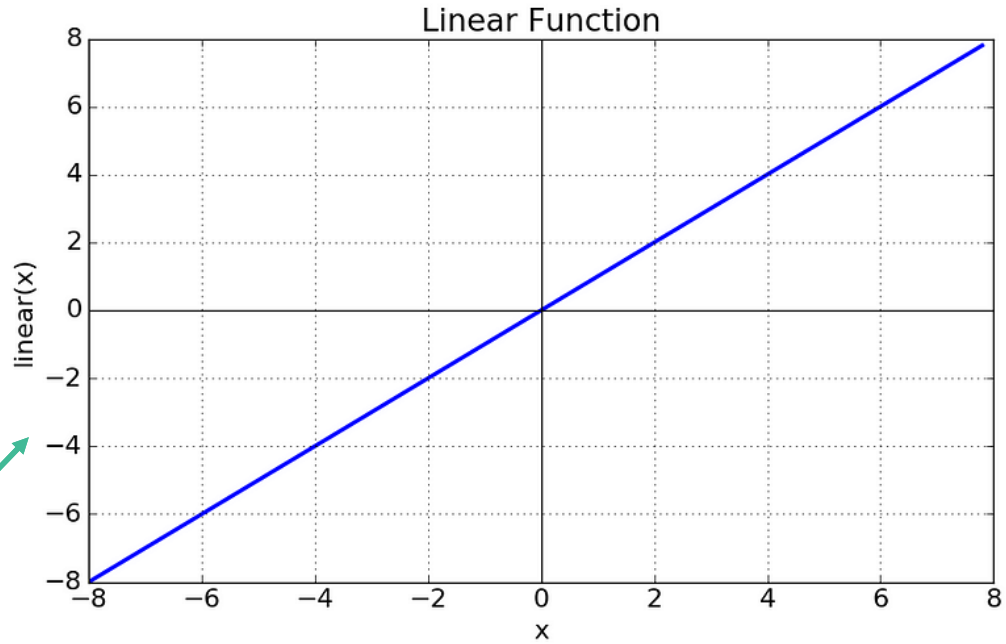
After the summation, the perceptron applies an **activation function** to the sum.
It determines whether the perceptron should fire or not based on the aggregated input, and it can introduce non-linearity into the output.
A Threshold Value typically accompanies the activation function.
The activation function maps the weighted sum into a binary value.
If the result of the activation function exceeds the threshold, the perceptron **fires (outputs 1)**; otherwise, it remains **inactive (outputs 0)**.
The final output represents the perceptron's decision or prediction based on the input and the weights.

Activation Function

It is used to determine the output of a neural network like yes or no.
It maps the resulting values between 0 to 1 or -1 to 1, etc. (depending upon the function).

Activation Functions can be basically divided into 2 types:

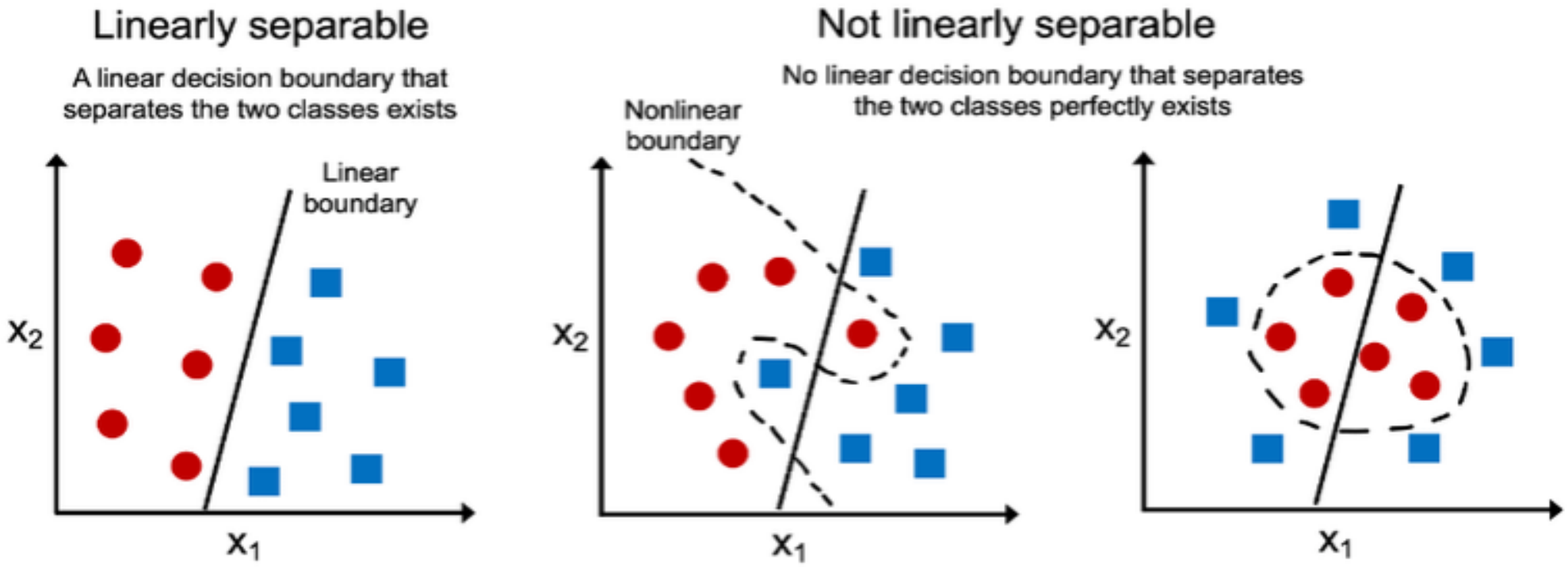
- Linear Activation Function
- Non-linear Activation Functions



Equation : f(x) = x

Range : (-infinity to infinity)

Activation Function



Activation Function

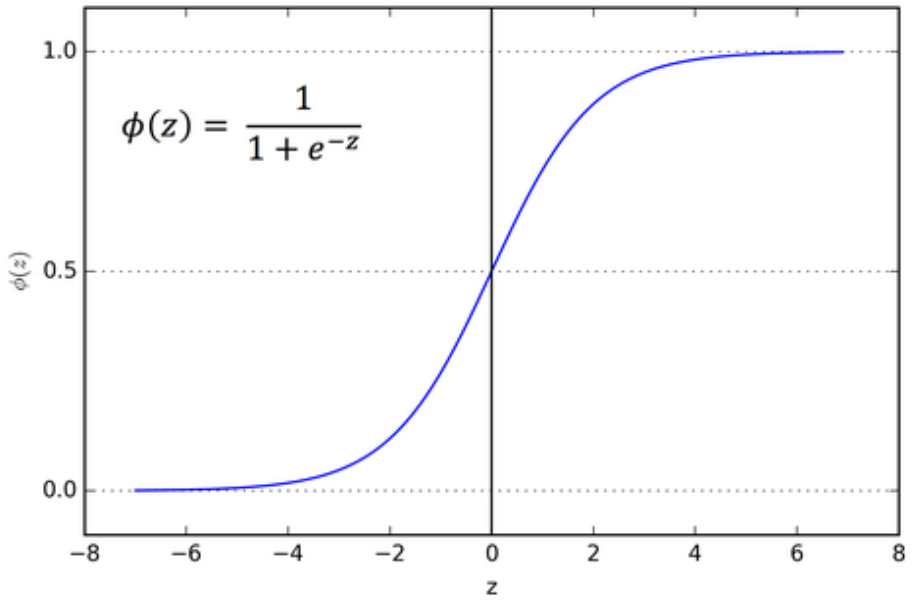
The Nonlinear Activation Functions are the most used activation functions. It makes it easy for the model to generalize or adapt to a variety of data and to differentiate between the output.

The main terminologies needed to understand nonlinear functions are:

- **Derivative or Differential**: Change in y-axis w.r.t. change in x-axis. It is also known as *slope*.
- **Monotonic function**: A function which is either
  - entirely non-increasing [if $a \leq b$, then $f(a) \geq f(b)$] or
  - non-decreasing [if $a \leq b$, then $f(a) \leq f(b)$].

Sigmoid or Logistic Function

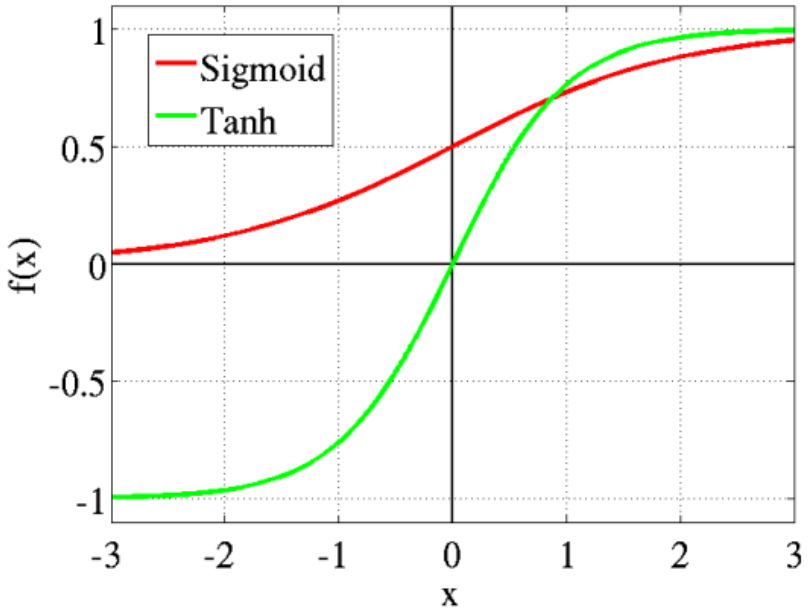
- We mainly use the **sigmoid** function for models where we must predict the **probability** as an output. Since the probability of anything exists only in (0 to 1) sigmoid is the right choice.
- It is **differentiable**. That means we can find the slope of the sigmoid curve at any two points.
- It is **monotonic**, but the function's derivative is not.
- It can cause a neural network to get stuck during training.
- It is a more generalized logistic activation function used for multiclass classification.



Tanh Function

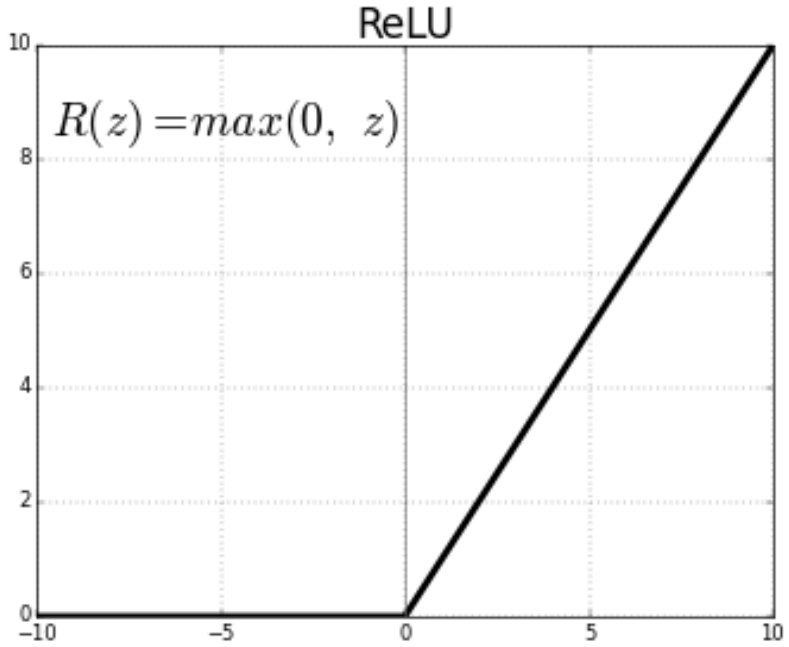
- The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s-shaped).
- The advantage is that the negative inputs will be mapped strongly negative, and the zero inputs will be mapped near zero in the tanh graph.
- The function is **differentiable**.
- It is **monotonic**, while its derivative is not monotonic.
- It is mainly used for binary classification.

Both tanh and logistic sigmoid activation functions are used in feed-forward nets.



Rectified Linear Unit (ReLU) Function

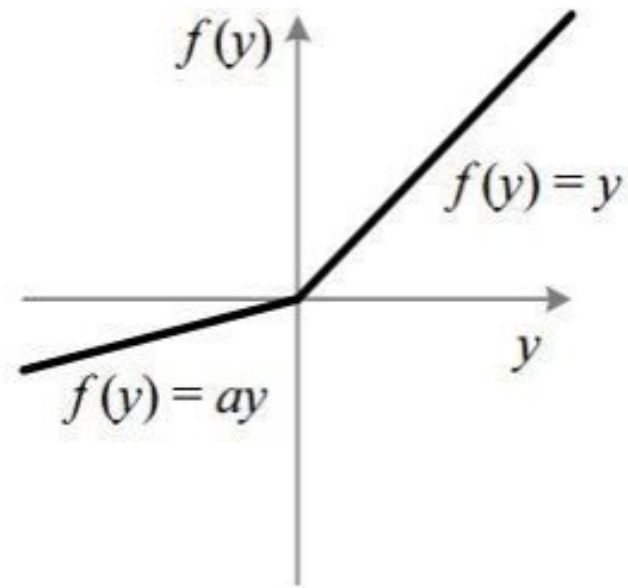
- ReLU is the most used activation function. It is used in almost all convolutional neural networks.
- The ReLU is half rectified (from the bottom). f(z) = 0 when z is less than zero, and f(z) = z when z is above or equal to zero.
- The function is **differentiable**.
- The function and its derivative are both **monotonic**.
- However, the issue is that all the negative values become zero immediately, which decreases the model's ability to fit or train from the data properly. (also known as the dying ReLU problem)



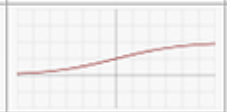
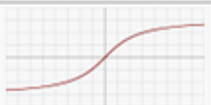
Leaky ReLU

problem.

- The leak helps to increase the range of the ReLU function. Usually, the value of *a* is 0.01 or so.
- When *a* is not 0.01, then it is called Randomized ReLU.
- The range of the Leaky ReLU is (- ∞ to ∞).
- Both Leaky and Randomized ReLU functions are **monotonic** in nature. Also, their derivatives are monotonic in nature.



Activation Function Cheatsheet

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Why we use Derivatives?

When updating the curve, we need to know in which direction and how much to change or update the curve depending upon the slope given by the first order derivative.

That is why we use differentiation in almost every part of Machine Learning and Deep Learning.

The Training

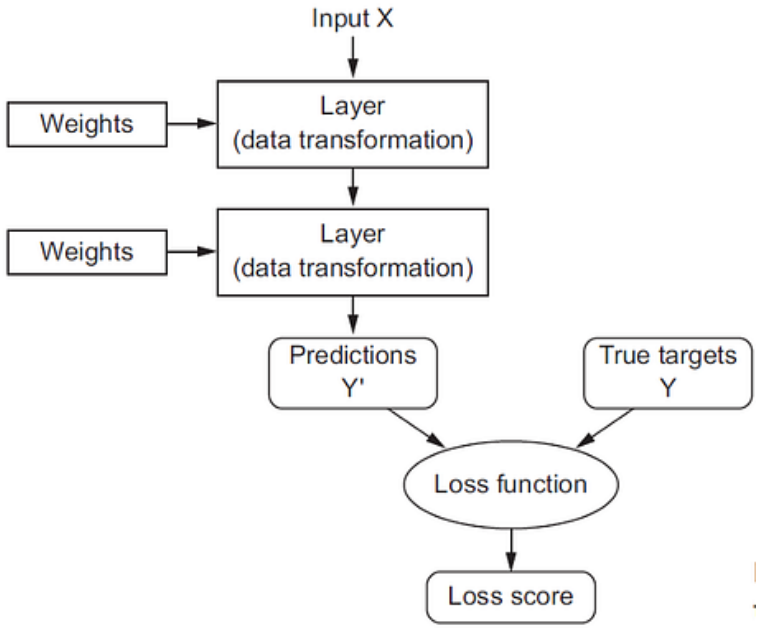
An ML model is Trained by looping over data multiple times.
For each iteration, the weight values are adjusted.
Training is complete when the iterations fail to reduce the Loss.

We need:

- A **Training Function**: that predicts the outcome based on the activation function. This is the Activation function of the **output layer**.
- A **Loss function**: that measures how good the solution is.
- An **Otrimizer**: Every time the prediction is wrong, the neural network should adjust the weights. (backpropagation) After many guesses and adjustments, the weights will be correct.

The Loss Function

- Loss function (cost/error) is a mathematical function that measures the difference between the predicted output of a machine learning model and its actual output.
- The goal of training a machine learning model is to minimize the value of the loss function, which maximizes the accuracy of the model's predictions.
- Different types of loss functions are used for problems, such as mean squared error for regression problems and cross-entropy for classification problems.
- At its core, a loss function is a measure of how good your prediction model is in terms of being able to predict the expected outcome(or value).



Types of Loss Functions

The most commonly used loss functions are:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Log-Likelihood Loss
- Cross Entropy
- Hinge Loss
- Huber Loss

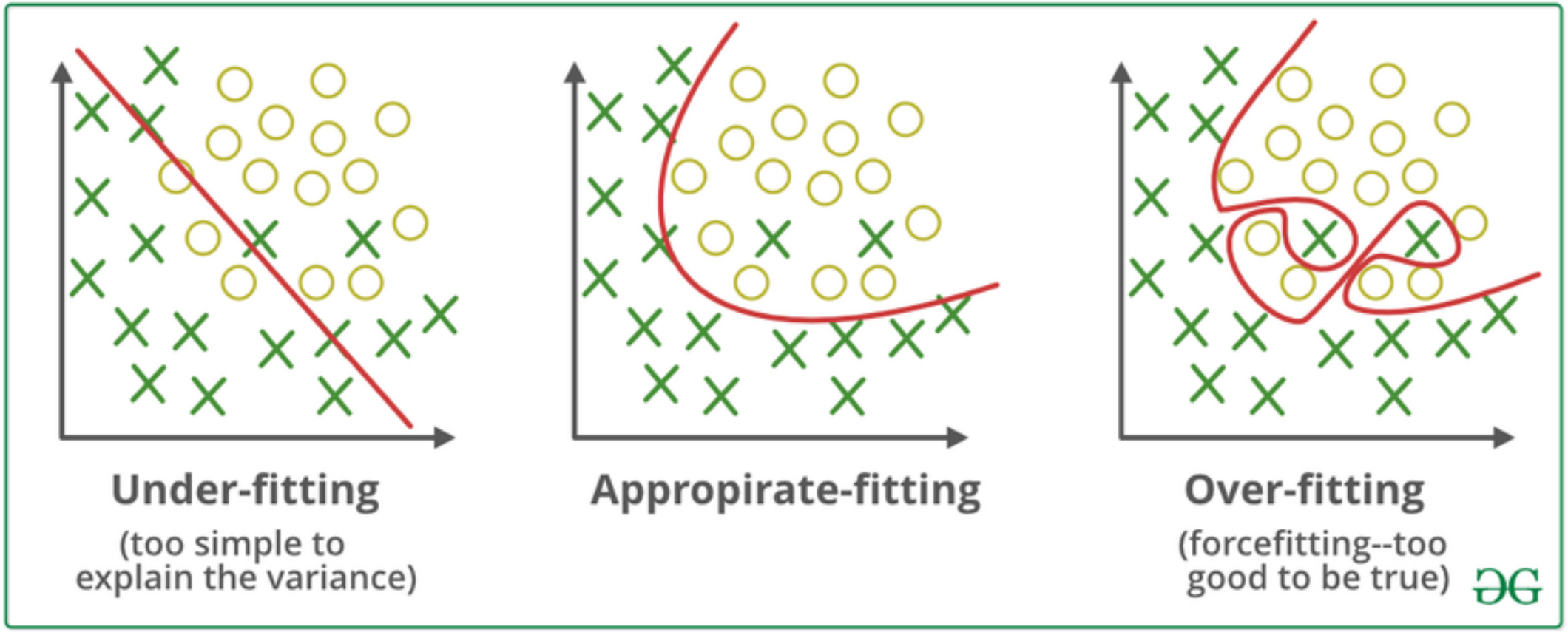
Optimizer

We convert the learning problem into an optimization problem. After we define a loss function, we want to minimize the loss function with an optimization algorithm.

Known algorithms that minimize the loss function are:

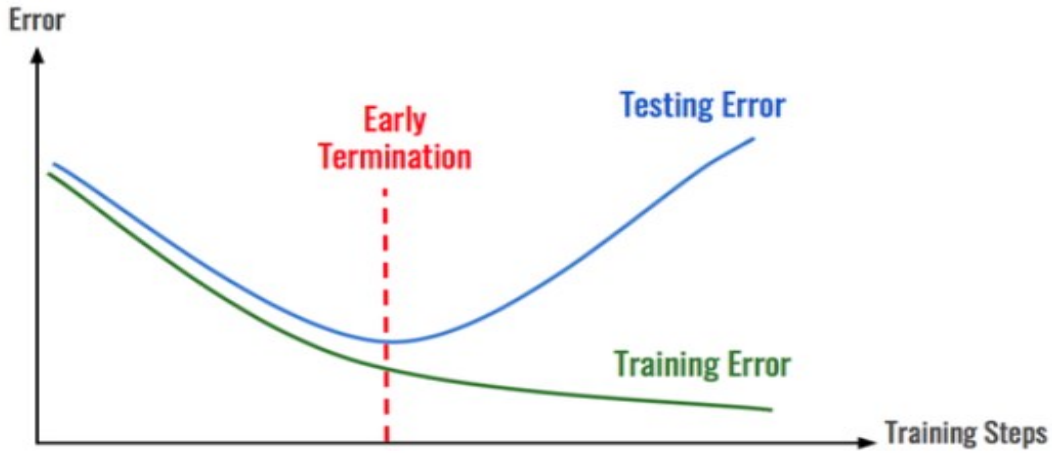
- Stochastic Gradient Descent
- Resillient Backpropagation
- RMSProp
- Adam
- Adagrad

Overfitting & Underfitting



Prevent Overfitting

Standard ways to limit the capacity of a neural net:



✓ Limit the number of hidden units — Dropout

✓ Limit the size of the weights — Regularization

✓ Stop the learning before it has time to overfit — Early stopping

The Regularization

Regularization is about finding the perfect balance between the simple and too complicated, 'augmenting' the input data and forcing models to focus on significant factors for prediction, avoiding memorizing errors (or noise), and ensuring the ability to predict the unseen satisfactorily.

With regularization, we can minimize or shrink the coefficient estimates towards zero to avoid underfitting or overfitting the machine learning model.

Types of Regularization

- L2(Ridge) regularization
- L1(Lasso) regularization
- Dropout regularization

L1 & L2 Regularization

- **L2 regularization** or **ridge regression**, works by adding a penalty or complexity term in the loss function of the standard least squares model, which is proportional to the sum of the squares of weights of each feature.
- **L1 regularization** or **lasso regression**, works by minimizing the weights by including them as a penalty in the loss function. The absolute weights are considered instead of the magnitude of the weights, as in the case of ridge regression. It allows for natural feature selection.

L2 regularization	L1 regularization
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases
Non-sparse outputs	Sparse outputs
No feature selection	Built-in feature selection

L1 & L2 Regularization

L1 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} |w_i|$$

L2 regularization on least squares:

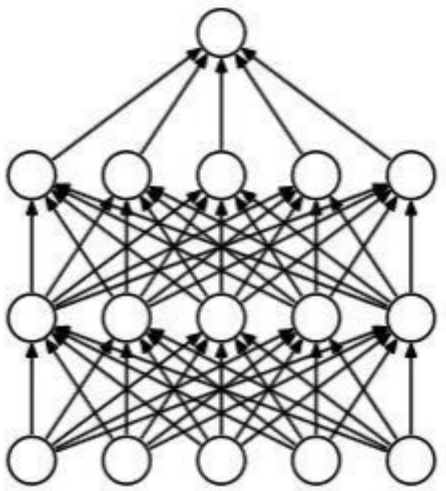
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

Dropout Regularization

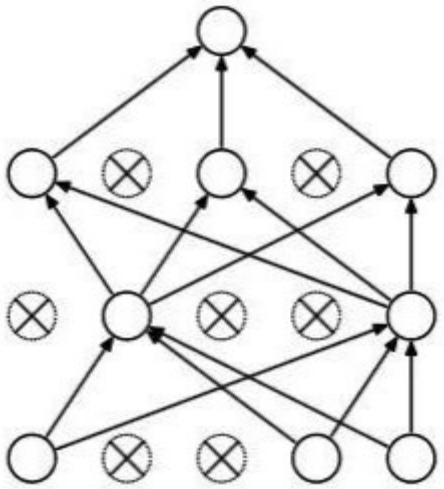
Dropout Regularization works by **randomly dropping out neurons during the training process**. (setting the output of the dropped neurons to zero)

This encourages the other neurons to learn to compensate for the dropped neurons. The model becomes more robust and less prone to overfitting.

The dropout rate should be adjusted based on the complexity of the model and the size of the dataset.
A higher dropout rate can be used for simpler models and smaller datasets, while a lower dropout rate can be used for more complex models and larger datasets.



(a) Standard Neural Net



(b) After applying dropout.

Types of NNs

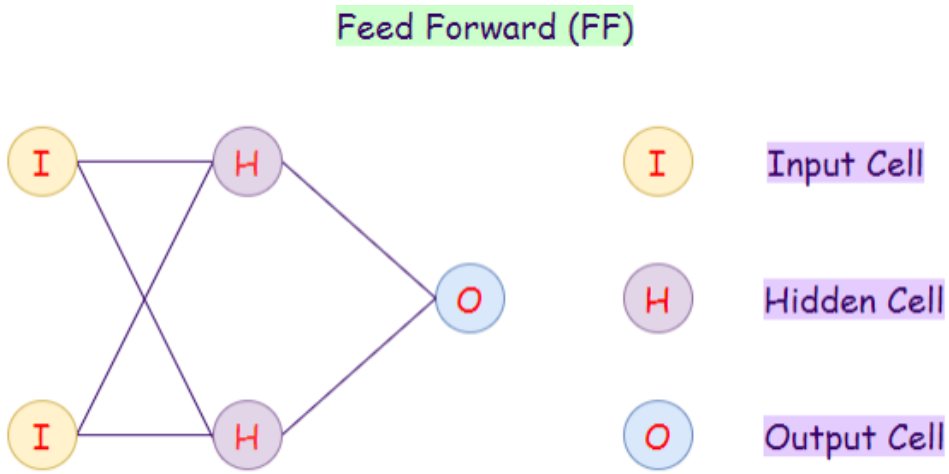
In its simplest form, a neural network is made up of the following:

- An input layer
- A hidden layer
- An output layer

In the Neural Network Model, input data are processed against a hidden layer before producing the final output.

So, Neural Networks are **Multi-Layer Perceptrons**

By combining multiple perceptrons in layers and connecting them in a network structure, these models can learn and represent complex patterns and relationships in data, enabling tasks such as image recognition, natural language processing, and decision-making.



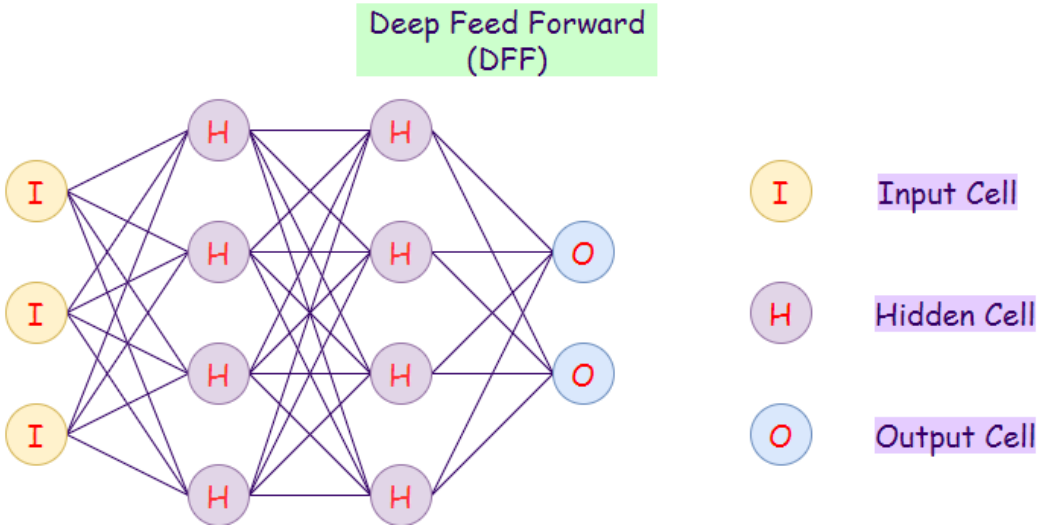
Types of NNs

Deep Neural Networks are made up of several hidden layers of neural networks that perform complex operations on massive amounts of data.
Each successive layer uses the preceding layer as input.
In the Deep Neural Network Model, input data (yellow) are processed against a hidden layer (pink) and modified against more hidden layers to produce the final output (blue).

The First Layer:
Perceptrons make simple decisions based on input. Each single decision is sent to the perceptrons in the next layer.

The Second Layer:
The next perceptrons make decisions by weighing the results from the first layer. This layer makes more complex decisions at a more abstract level than the first layer.

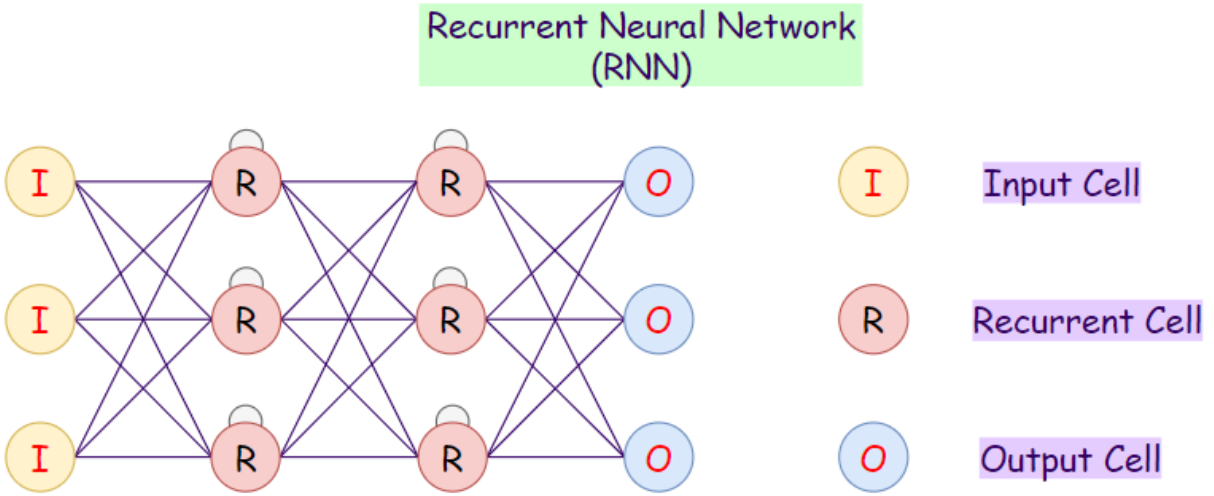
For instance, optical reading uses low layers to identify edges and higher layers to identify letters.



Types Of NNs

In contrast to the unidirectional feedforward neural network, **Recurrent Neural Network (RNN)** is a bi-directional artificial neural network, which allows the output from some nodes to affect subsequent input to the same nodes.

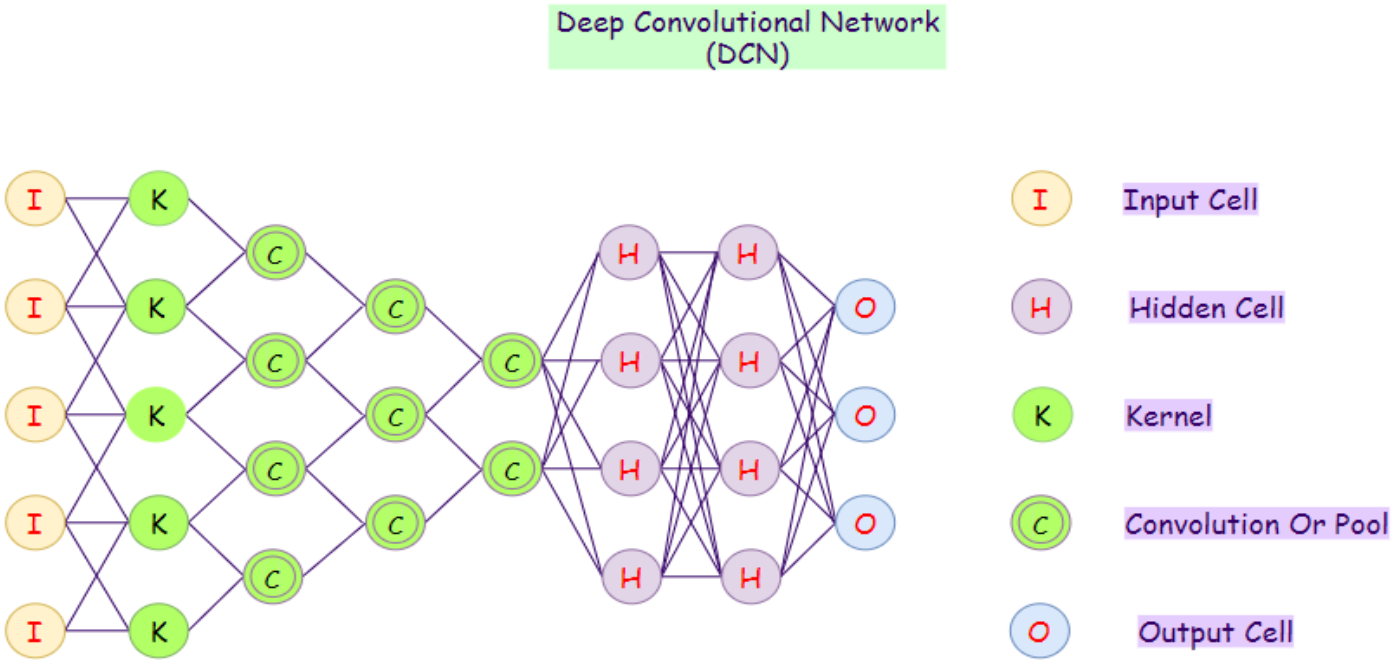
Their ability to use internal state (memory) to process arbitrary sequences of inputs makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

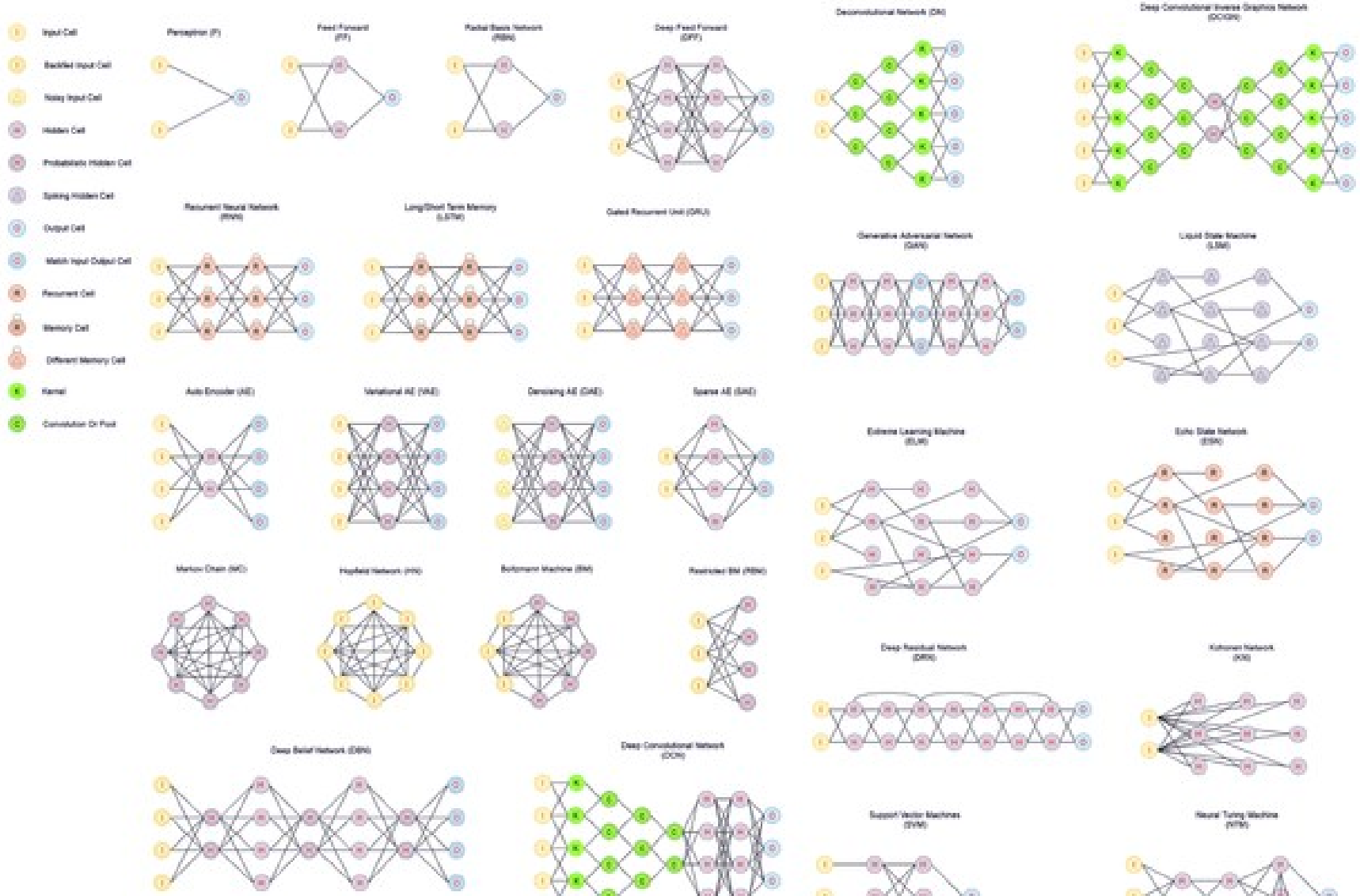


Types Of NNs

A **convolutional neural network (CNN)** is a regularized type of feed-forward neural network that learns feature engineering by itself via filter (or kernel) optimization.

Vanishing gradients and exploding gradients, seen during backpropagation, are prevented by using regularized weights over fewer connections.





Main Types of Neural Networks

End of Session 1