# Deep Learning Course

## Intermediate Level

**Lara WEHBE - August 2024 - TheAIEngineers**

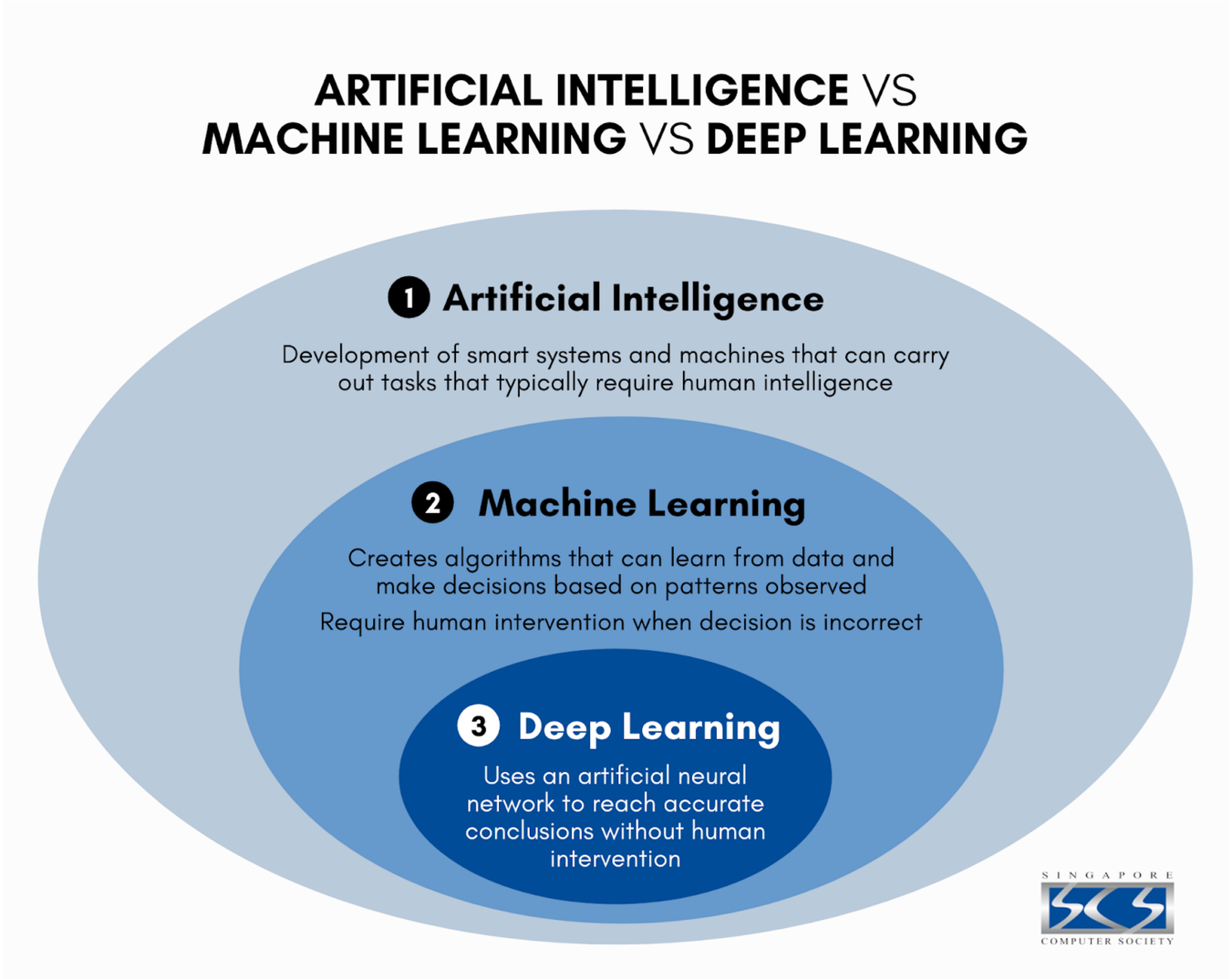
# Outline

- Introduction
- Revolution
- Mathematical Foundations
- Perceptron
- Optimization
- Deep Learning in Practice
- Numerical Example
- Neural Network Architectures

# Introduction

# Deep Learning

## What is Deep Learning?



# Deep Learning

## Difference between Machine Learning and Deep Learning

Machine Learning referred to as “Traditional Learning”

# Deep Learning

## How Deep Learning works?

Neural networks, or artificial neural networks, attempt to mimic the human brain through a combination of data inputs, weights and bias—all acting as silicon neurons. These elements work together to accurately recognize, classify and describe objects within the data.

# Deep Learning

## Main Elements:

The main elements of deep learning are:

1. Layers and Neurons
2. Perceptron
3. Feed Forward (For learning)
4. Back propagation (For calculating errors)
5. Activation Functions

# Deep Learning

## Applications

1. Computer Vision:

**Automotive**: While the age of driverless cars hasn't entirely arrived, the underlying technology has started to make its way into automobiles, improving driver and passenger safety through features such as lane line detection.

- **Healthcare:** Computer vision has been incorporated into radiology technology, enabling doctors to better identify cancerous tumors in healthy anatomy.
- **Marketing:** Social media platforms provide suggestions on who might be in a photograph that has been posted on a profile, making it easier to tag friends in photo albums.
- **Retail:** Visual search has been incorporated into some e-commerce platforms, enabling brands to recommend items that would complement an existing wardrobe.

# Revolution

# The Perceptron (1950s)

## Birth of the Neural Network

How could we create a machine that thinks like a human? Could a machine learn from experience, just as humans do?

Answer:

Birth of Perceptron

# The Perceptron (1950s)

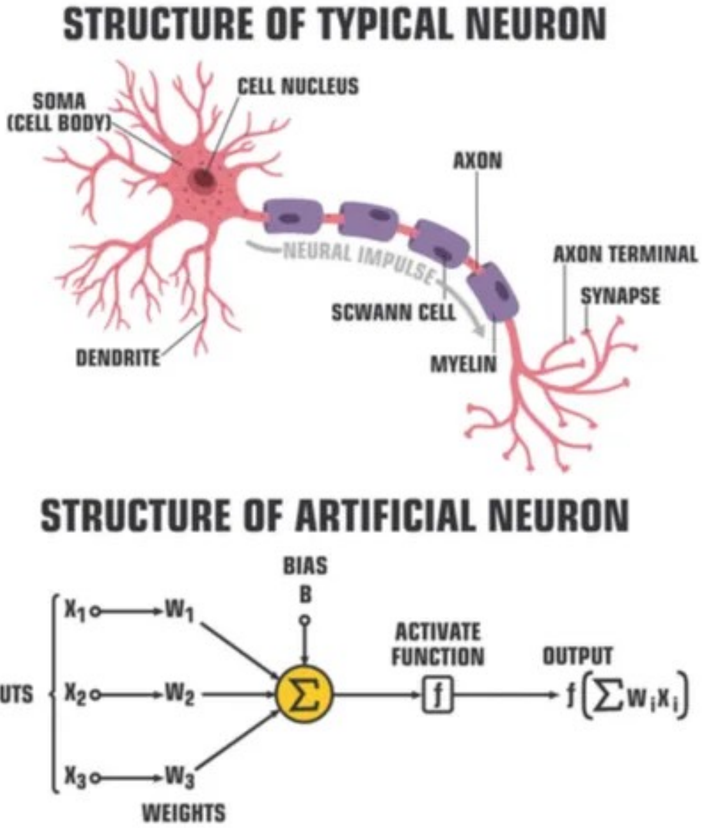
## Frank Rosenblatt: The Visionary

In 1957, Frank Rosenblatt, a psychologist and researcher, embarked on a journey to mimic the human brain’s learning process.

# The Perceptron (1950s)

## Birth of the Neural Network

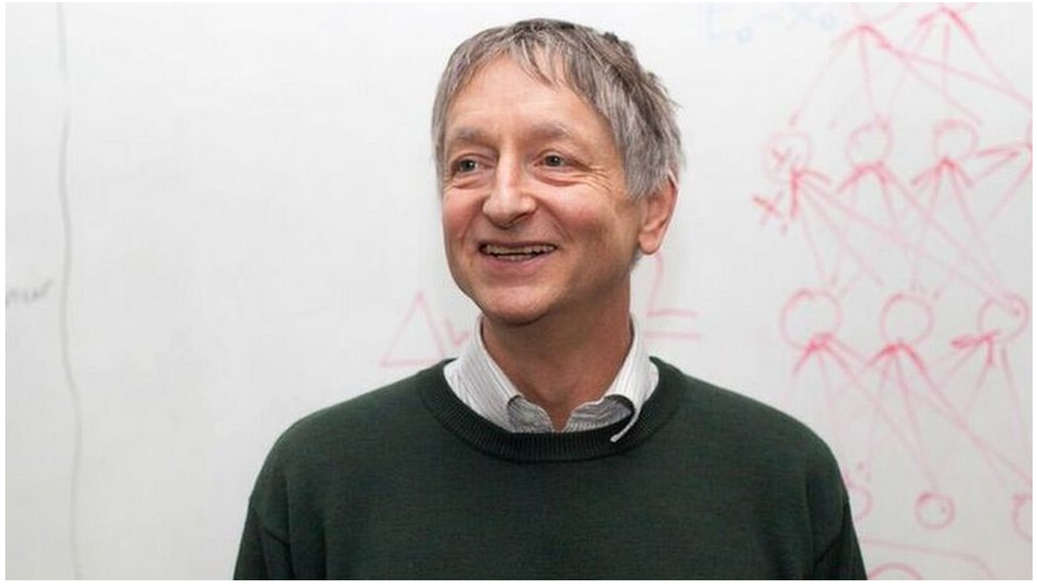
The perceptron is one of the earliest models of a neural network. It is a type of linear classifier that maps an input feature vector to an output decision (e.g., binary classification).



# Backward Propagation (1980s)

## Revolutionizing Training

In his paper entitled “Learning Representations by Back-propagating errors”, alongside with his colleagues, Geoffrey Hinton provided a clear and practical explanation on how to calculate the neural network errors, and update their weights leading to significantly diverged model.



# Backward Propagation (1980s)

## Revolutionizing Training

Letter | Published: 09 October 1986

### Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

*Nature* **323**, 533–536 (1986) | Cite this article



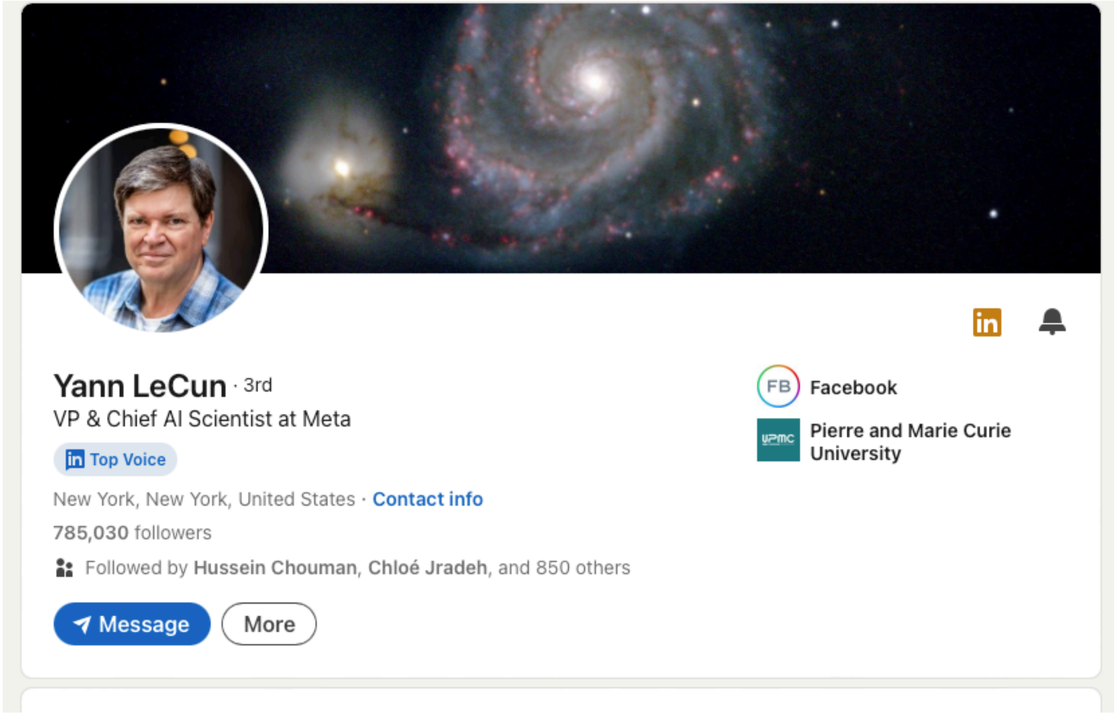
**Abstract**

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

# Convolutional Neural Networks (1990s)

## Transforming Image Recognition

Yan LeCun  is a French-American computer scientist that had significant contribution to the development of CNNs.

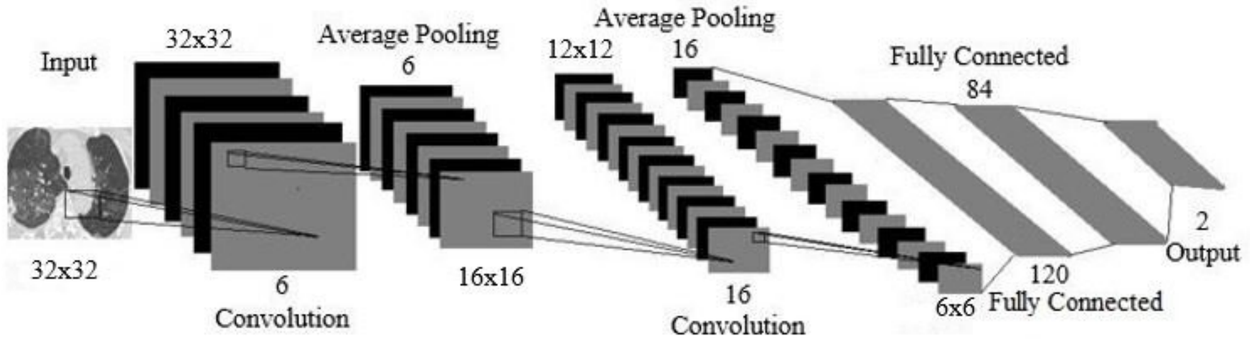


# Convolutional Neural Networks (1990s)

## Transforming Image Recognition

LeCun's work on CNNs culminated in the creation of LeNet-5, one of the earliest CNN architectures, which was designed for handwritten digit recognition.

It became then the foundational model for the field of computer vision



# Deep Learning Breakthroughs (2010s)

## From AlexNet to GPT

Developed by Alex Krizhevsky, **Ilya Sutskever**, and **Geoffrey Hinton**, AlexNet was a convolutional neural network that achieved groundbreaking results in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

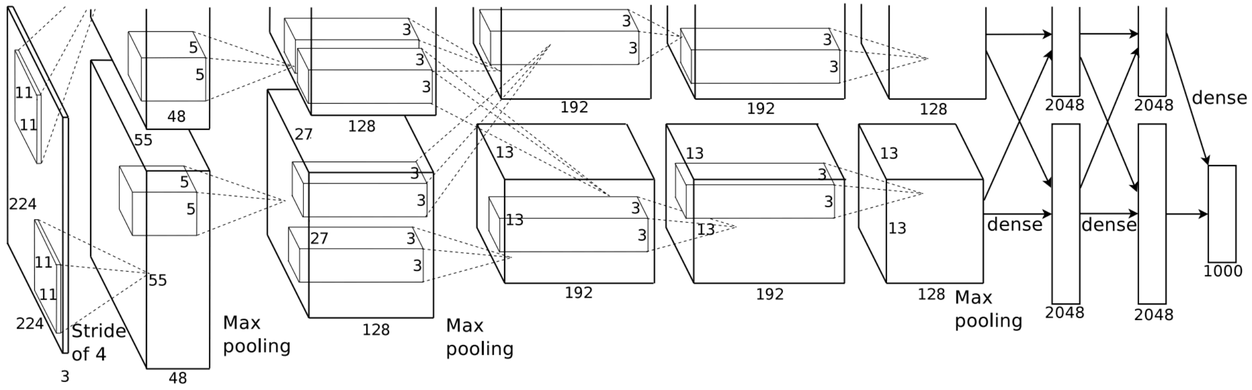



Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

# Deep Learning Breakthroughs (2010s)

## From AlexNet to GPT

AlexNet was an 8-layer network, with 5 convolutional layers followed by 3 fully connected layers, making it deeper than most networks at the time.

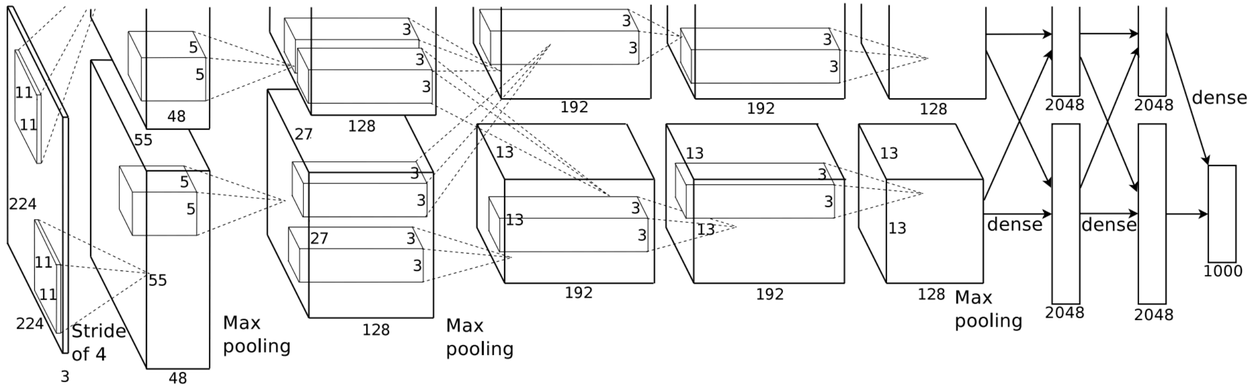


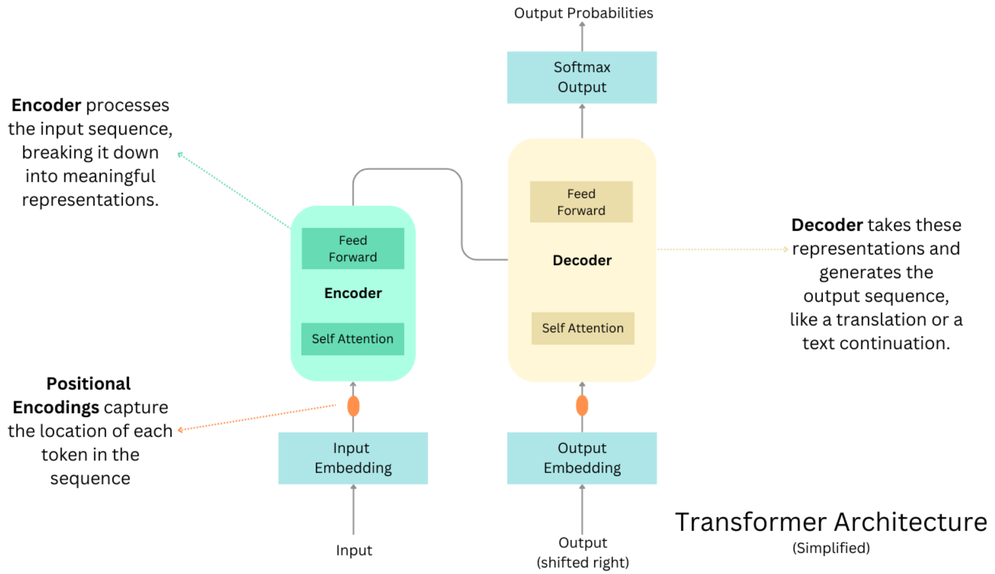

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

# Transformers (2017)

## Attention is all you need!

Transformers Architecture made a plot twist and a clear introduction to the Generative AI World!

Simply put, its ability to remember all the information backward, referred to as “Infinite Window” was the game changer.




Transformer Architecture
(Simplified)

# GPU Acceleration

## Core of training models

GPU acceleration revolutionized deep learning by dramatically reducing training times, enabling the development and scalability of larger, more complex neural networks like AlexNet

GPUs help in handling mathematical calculations needed for deep learning.

# Mathematical Foundations

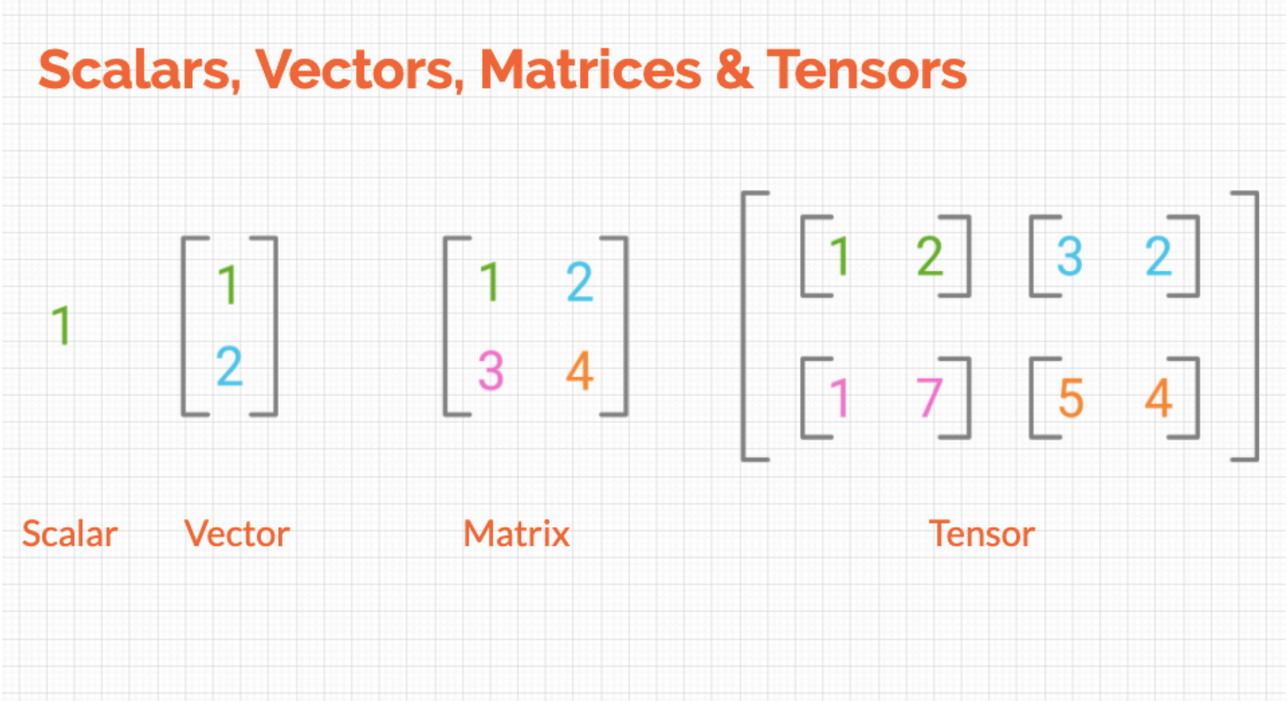
# Linear Algebra Review

## Introduction

In this section, we will dive deeper into the Mathematical fundamentals required to understand what’s going on inside the Deep Neural Network!

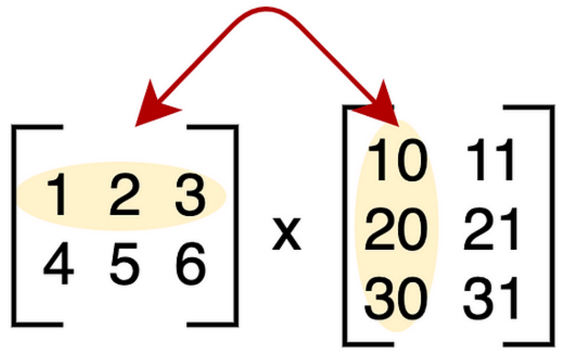
# Linear Algebra Review

## Vectors, Matrices, and Tensors



# Linear Algebra Review

## Matrix Multiplication and Its Role in Neural Networks



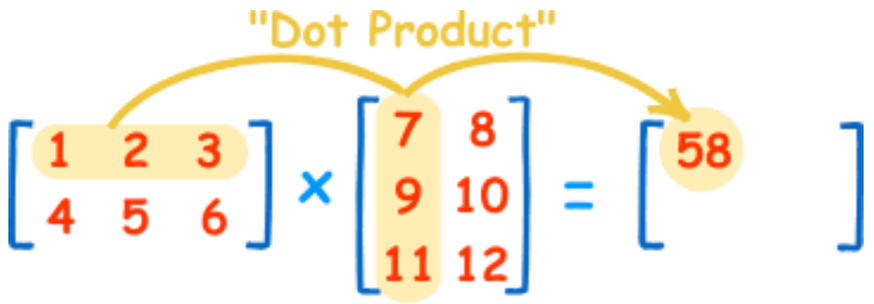
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

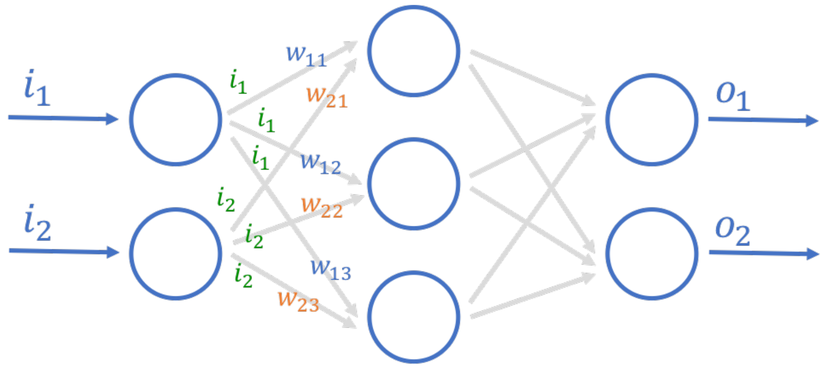
# Linear Algebra Review

## Matrix Multiplication and Its Role in Neural Networks



# Linear Algebra Review

## Matrix Multiplication and Its Role in Neural Networks

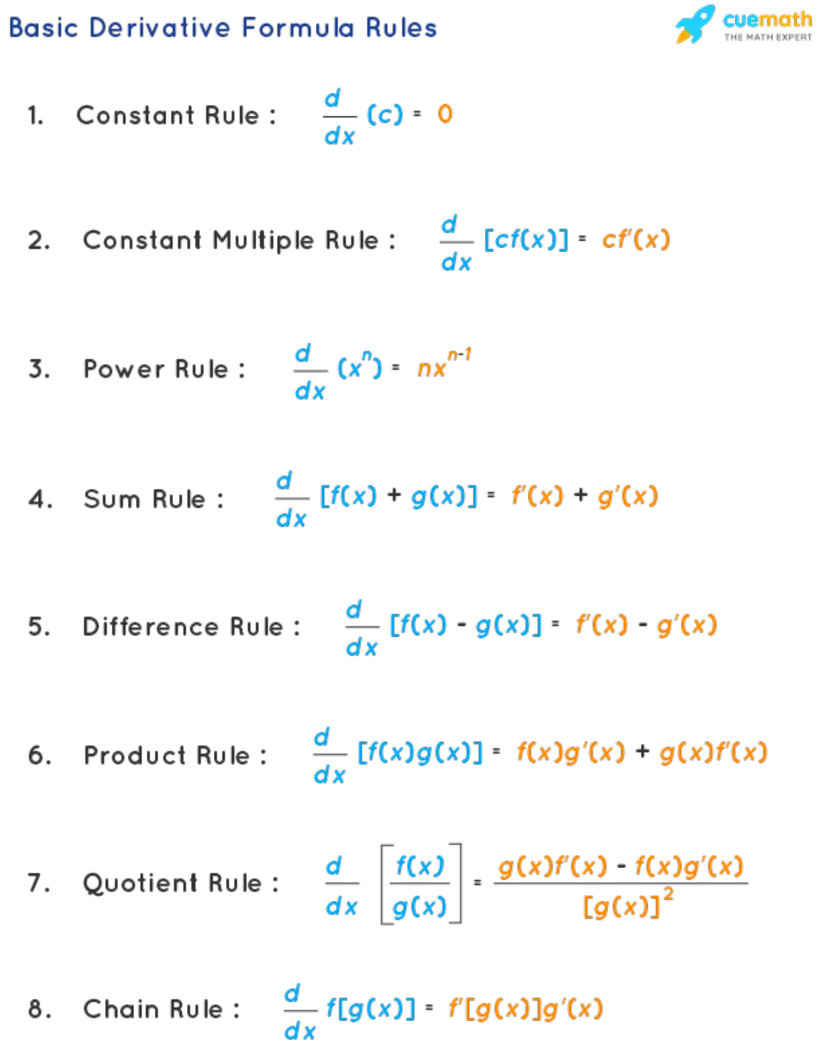



$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

# Calculus in Deep Learning

## Derivatives and Gradients

Differentiation is important in deep learning because it allows us to train neural networks using gradient descent. Gradient descent is an optimization algorithm that finds the minimum of a function by moving in the direction of the negative gradient, which is the direction of steepest descent.



# Perceptron

# Neural Networks Basics

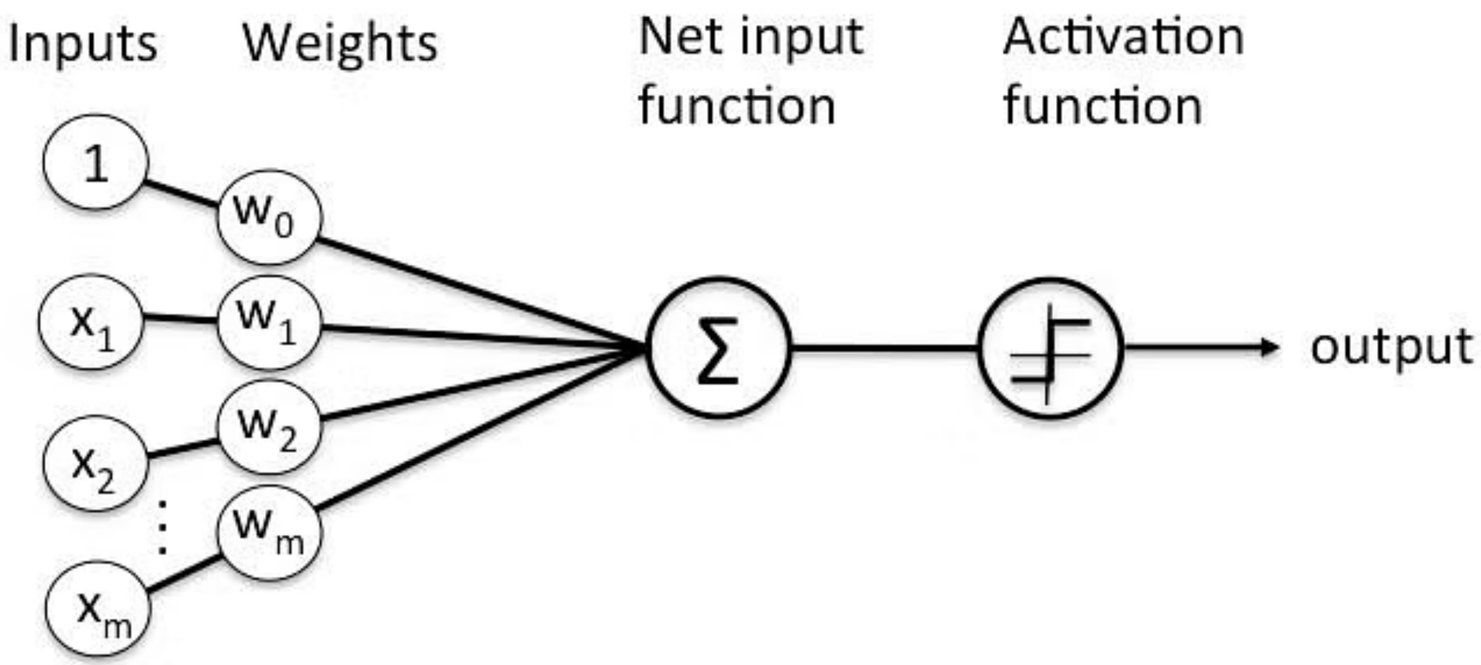
## What is a Neural Network

The Perceptron

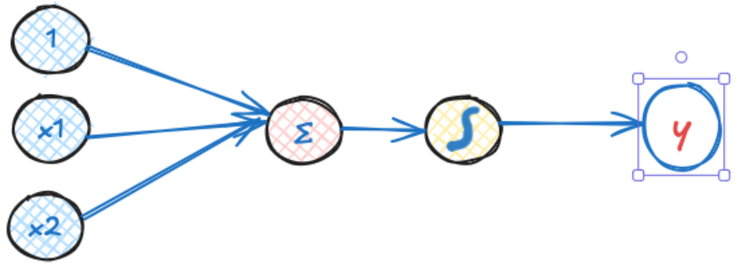
- **Introduction**: The perceptron is the simplest type of artificial neural network, used for binary classification.
- Components of a Perceptron:
  - **Input**: Receives input values (features).
  - **Weights**: Each input has a corresponding weight.
  - **Bias**: An additional parameter that adjusts the output along with the weighted sum of inputs.
  - **Activation Function**: Determines the output (usually a step function in perceptrons).
- **Output**: Produces an output (0 or 1) based on whether the weighted sum of inputs exceeds a certain threshold.
- **Importance**: The perceptron is the building block of more complex neural networks, serving as the foundation for understanding deeper architecture

# Perceptron

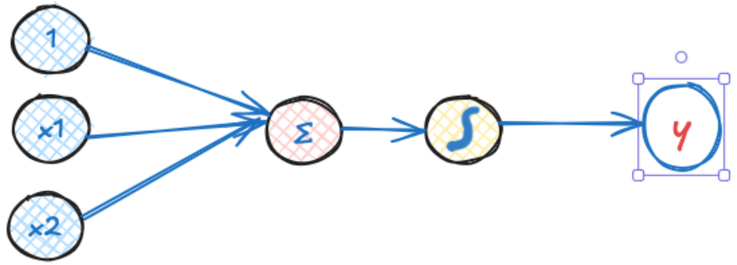
## Introduction



# Perceptron



# Perceptron: Forward Propagation




We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

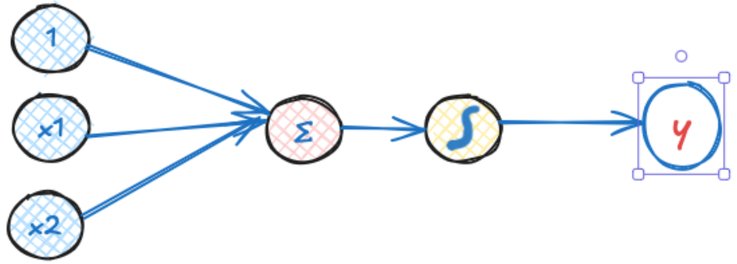
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

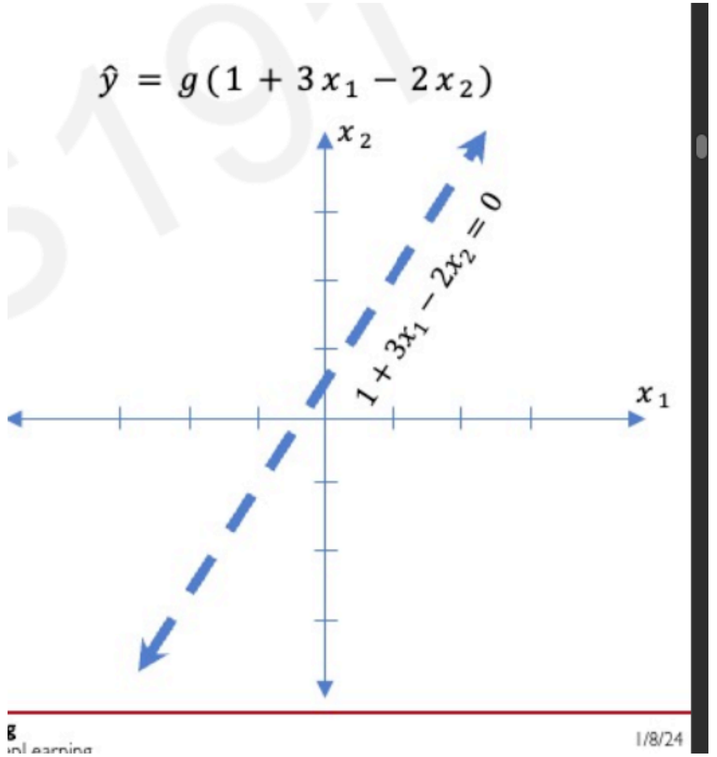
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

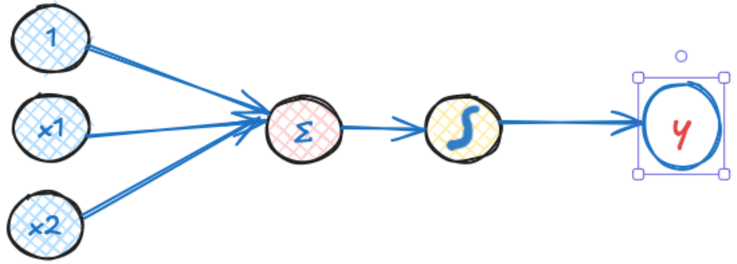
This is just a line in 2D!

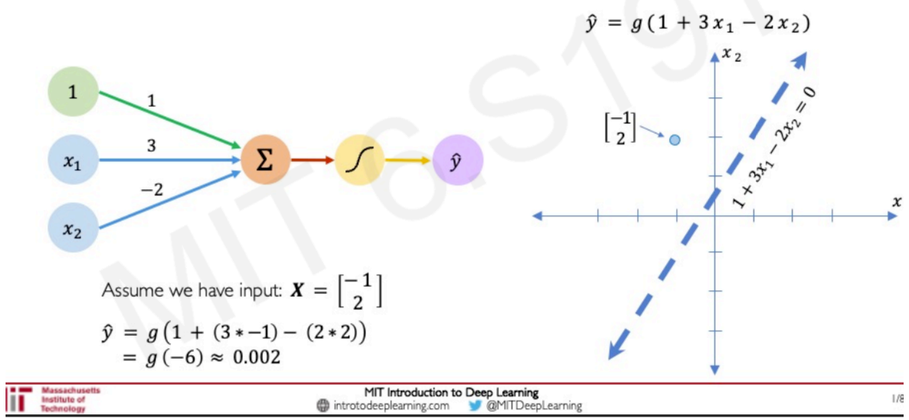
# Perceptron: Forward Propagation



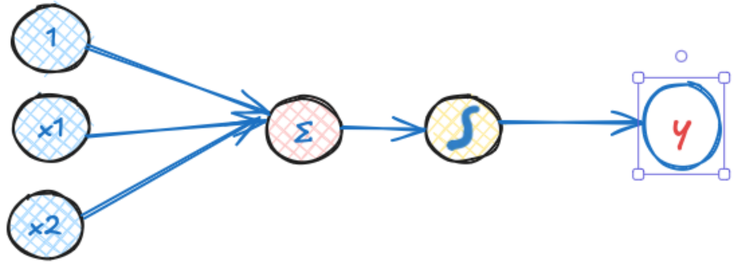



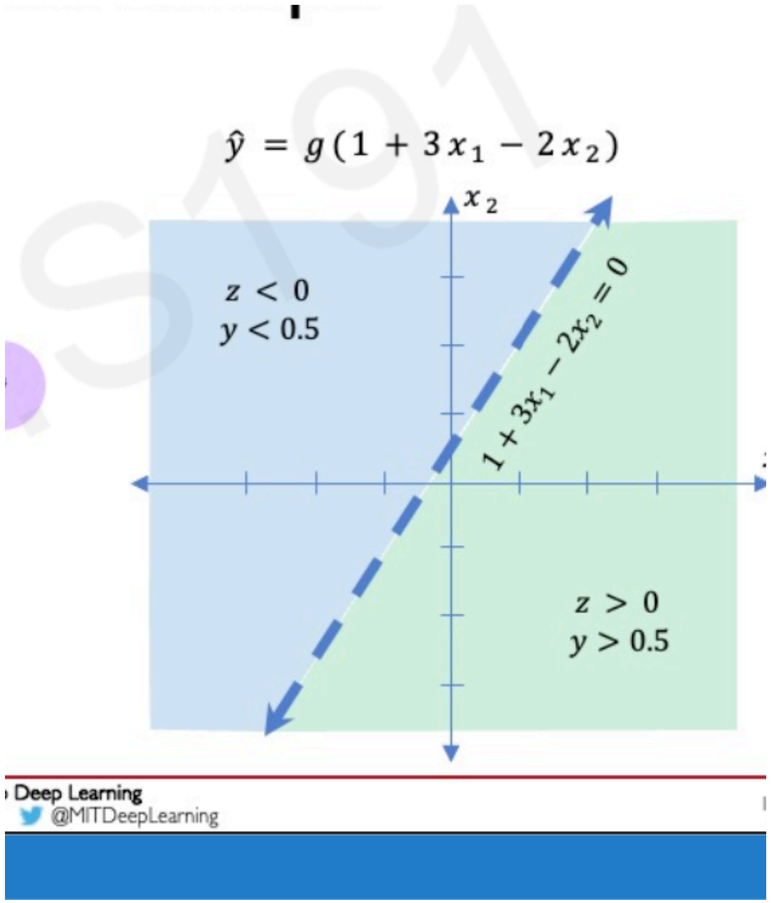
# Perceptron: Forward Propagation



# Perceptron: Forward Propagation



# Neural Networks

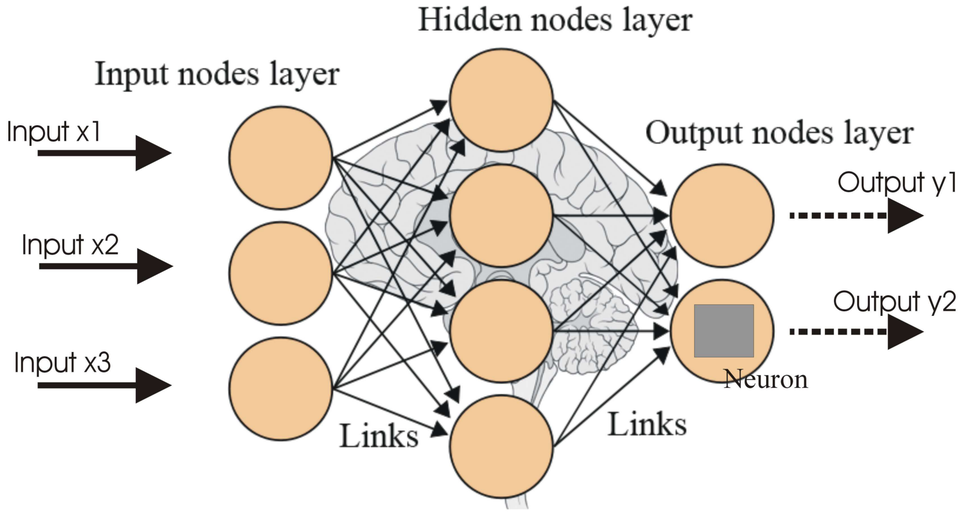
# Geoffrey Hinton: God father of Neural Networks

## What is a Neural Network

A neural network is a computational model inspired by the way biological neural networks in the human brain process information. It consists of interconnected units (neurons) that work together to solve specific problems.

**Basic Structure**: Input layer, hidden layers, and output layer.

**Functionality**: Neural networks learn to perform tasks by considering examples, generally without being programmed with any task-specific rules.



# Neural Networks Basics

## Single Perceptron Limitation

Evolution from Perceptron to Deep Networks

- **Limitations of Perceptron**: **Inability to solve non-linear problems (like XOR).**
- **Introduction of Multi-layer Perceptrons (MLPs)**: Overcoming limitations by adding hidden layers and non-linear activation functions.

# Neural Networks Basics

## Limitation Example

Inputs and Outputs:

- **Input A**: Switch A (can be 0 or 1)
- **Input B**: Switch B (can be 0 or 1)
- **Output**: Light status (should be 1 if A or B is on, but not both; otherwise 0)

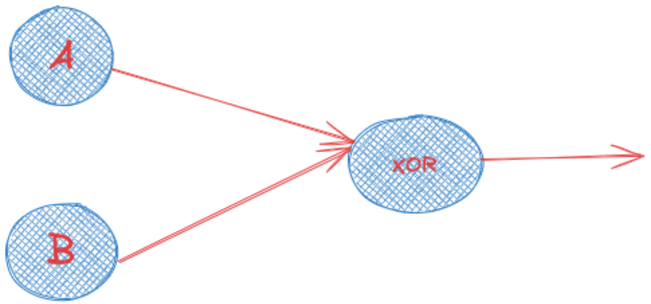
| A | B | A(XOR)B |
|---|---|---|
| **0** | 0 | 0 |
| **0** | 1 | 1 |
| **1** | 0 | 1 |
| **1** | 1 | 0 |

# Neural Networks Basics

## Limitation Example

Inputs and Outputs:

- **Input A**: Switch A (can be 0 or 1)
- **Input B**: Switch B (can be 0 or 1)
- **Output**: Light status (should be 1 if A or B is on, but not both; otherwise 0)

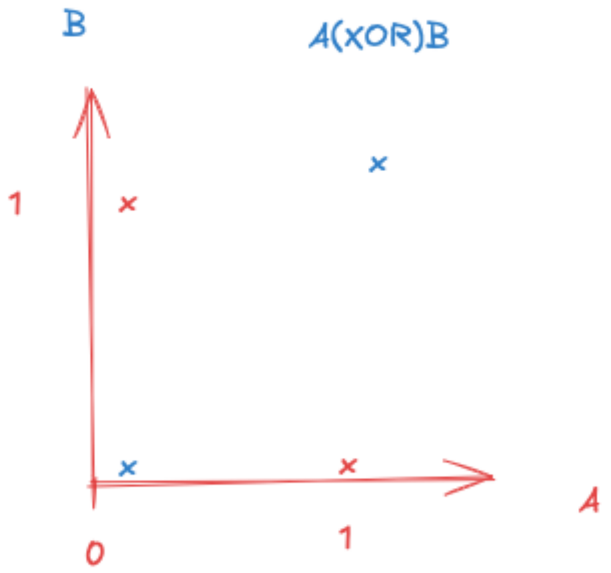


# Neural Networks Basics

## Limitation Example

Inputs and Outputs:

- **Input A**: Switch A (can be 0 or 1)
- **Input B**: Switch B (can be 0 or 1)
- **Output**: Light status (should be 1 if A or B is on, but not both; otherwise 0)



# Activation Functions

## Introduction

**Definition**: An activation function is a mathematical function applied to the output of a neuron in a neural network. It determines whether the neuron should be activated (fired) or not, based on the input signals.

# Activation Functions

## Why do we need activation functions?

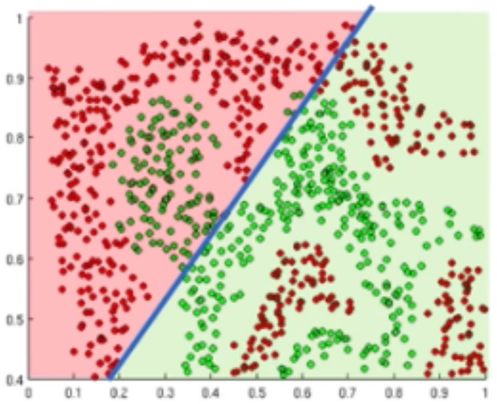
Activation functions are necessary for neural networks because, without them, the output of the model would simply be a linear function of the input. In other words, it wouldn’t be able to handle large volumes of complex data.

# Activation Functions

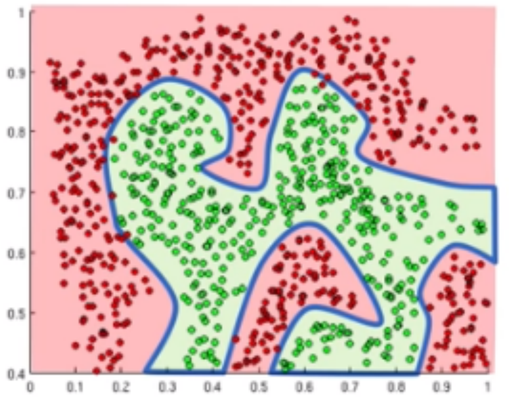
## Importance of Non-linearity

### Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*



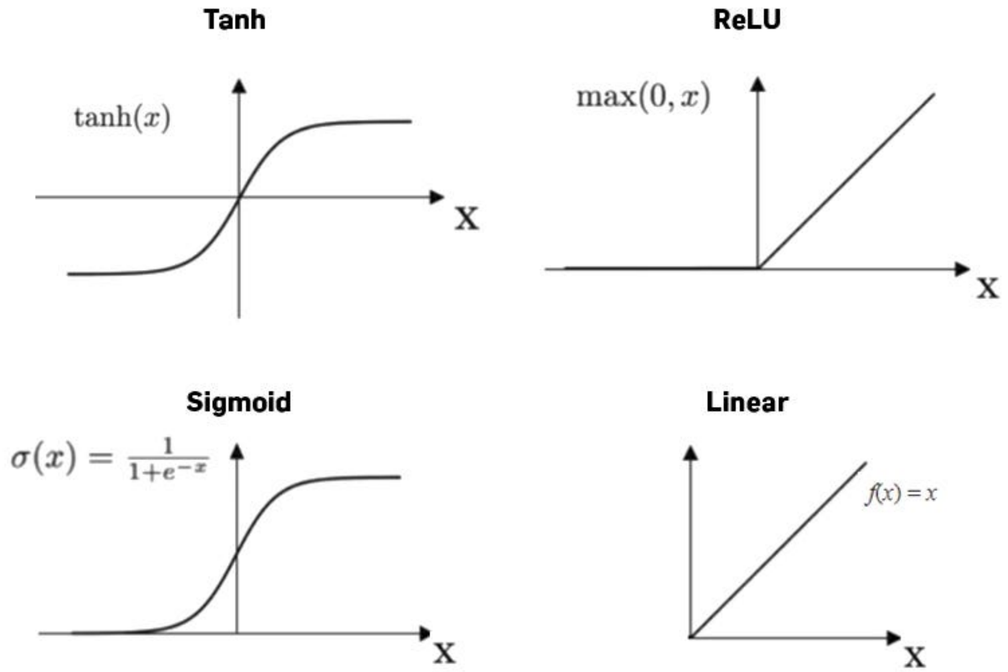

Linear activation functions produce linear decisions no matter the network size




Non-linearities allow us to approximate arbitrarily complex functions

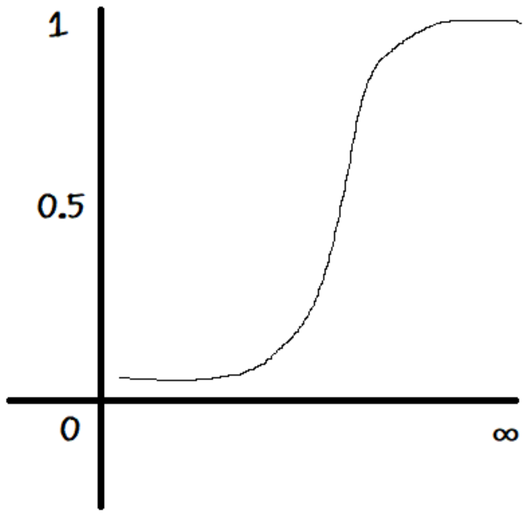
# Activation Functions

## Common Activation Functions

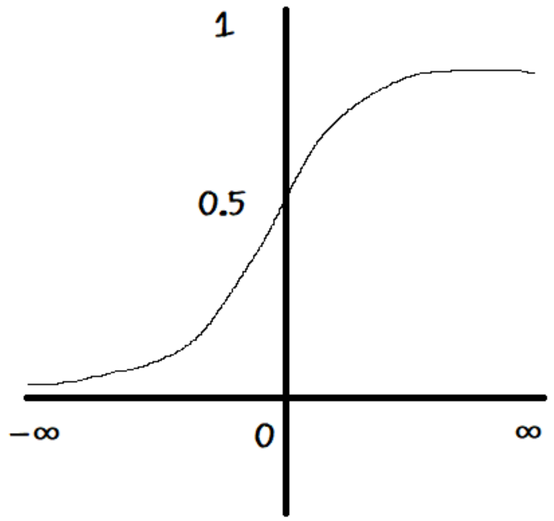


# Activation Functions

## Common Activation Functions




Sigmoid




Softmax

# Activation Functions

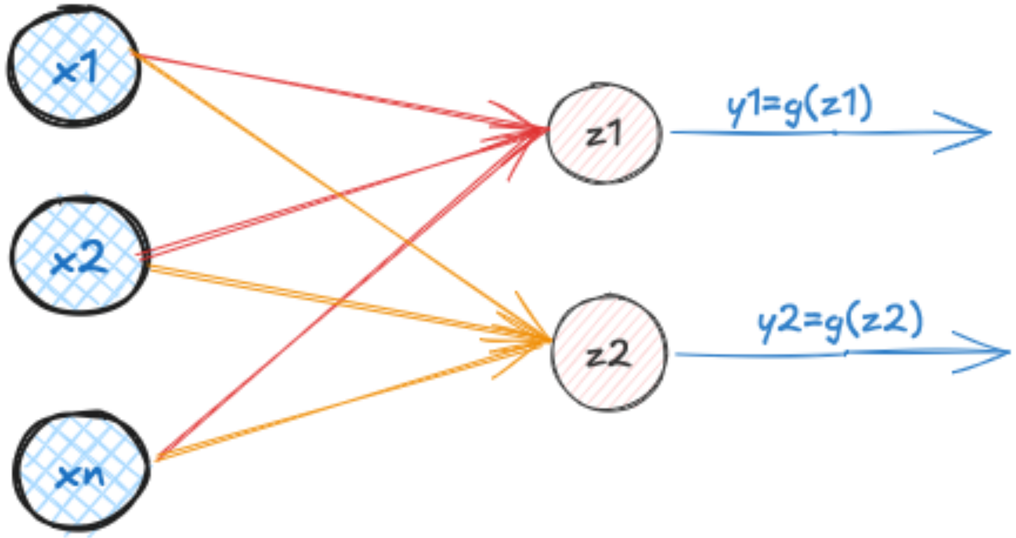
## Choosing the Right Activation Function for Your Model

Choosing the Right Activation Function

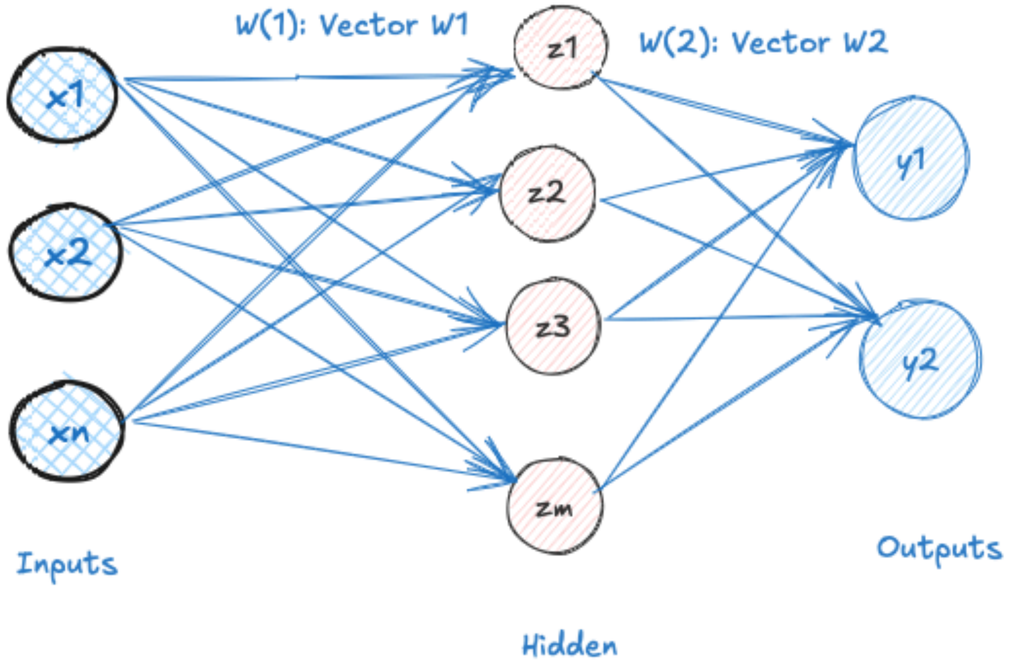
- **Problem-Specific**: The choice of activation function can significantly impact the performance of the neural network. It should be chosen based on the problem at hand and the architecture of the network.

- **Practical Considerations**: Speed of convergence, gradient flow, and computational efficiency.

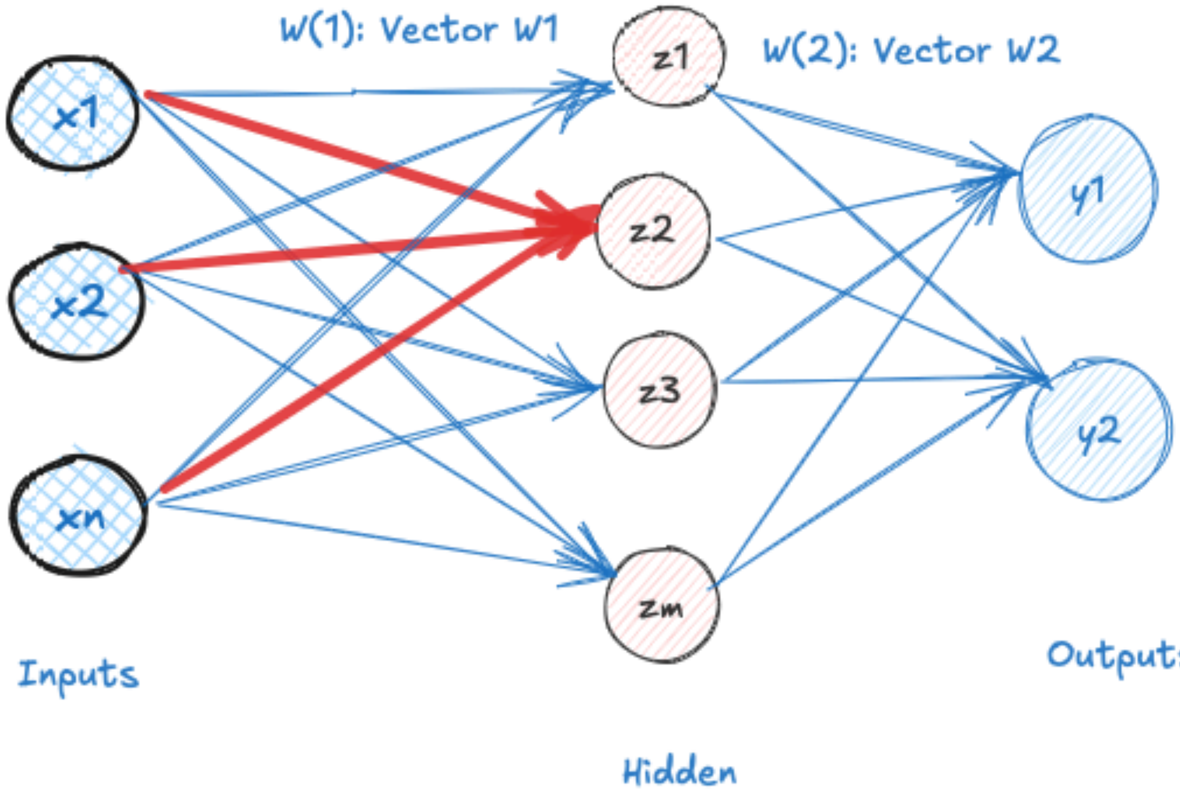
# Multi Output Perceptron



# Single Layer Network



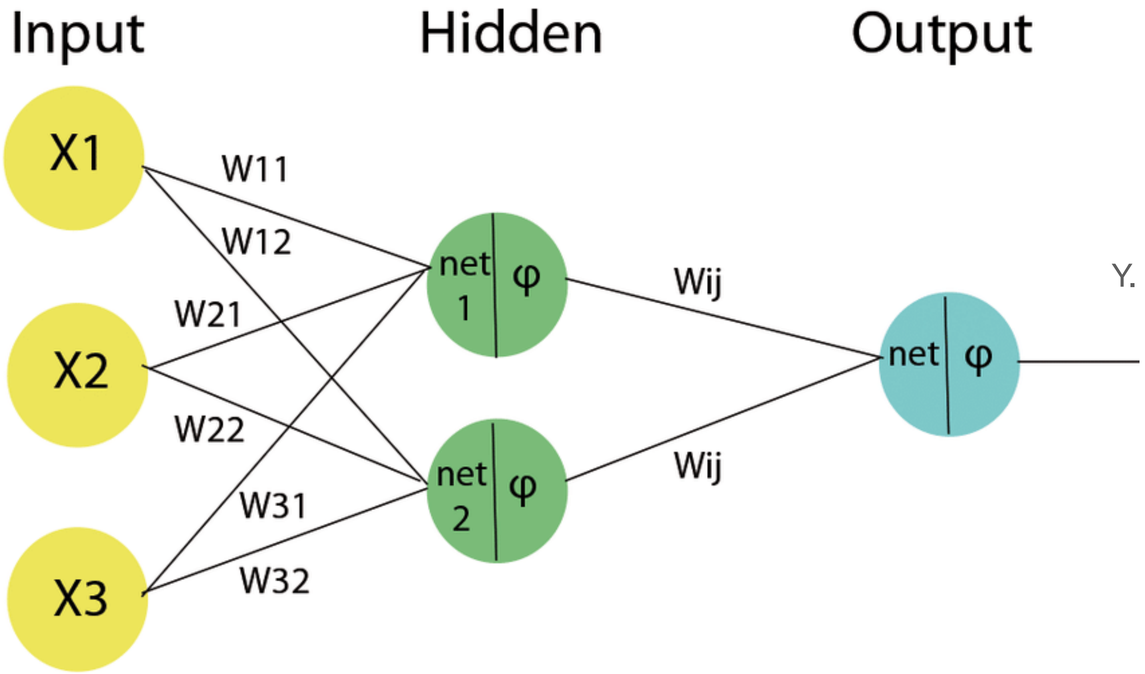
# Single Layer Network




Z2 = x1*w12 + x2*w22 + x3*w32 + bias ( w0)
Y2 = w12*z1 + w22*(x1*w12 + x2*w22 + x3*w32 + bias ( w0)) + ,…

# Multilayer Perceptron

## Deep Neural Networks




Z1 = w11*x1 + w21*x2 + w31*x3 + w0

Y. = w21* (activationfunction(z1)) + w22*activationfunction(z2))

$$\underset{j}{net} = \sum_{i} w_{ij} * X_i \qquad \varphi = \frac{1}{1 + e^{-net}}$$

# Feedforward Process

## How Information Flows through a Neural Networks

The feed-forward **model** is the simplest type of neural network because the input is only processed in one direction. The data always flows in one direction and never backwards, regardless of how many buried nodes it passes through.

# Backward Propagation Process

## Computing Gradients: Back Propagation


D1 = d(j(w)) / d(w1)

D2 = d(j(w)) / d(w2) = dj(w) / d(y ) * dy/d(w2)

W1 = w1 - d1
W2 = w2 -d2

# Backward Propagation Process

## Computing Gradients: Back Propagation

# Optimization

# Optimization Techniques

## Loss function

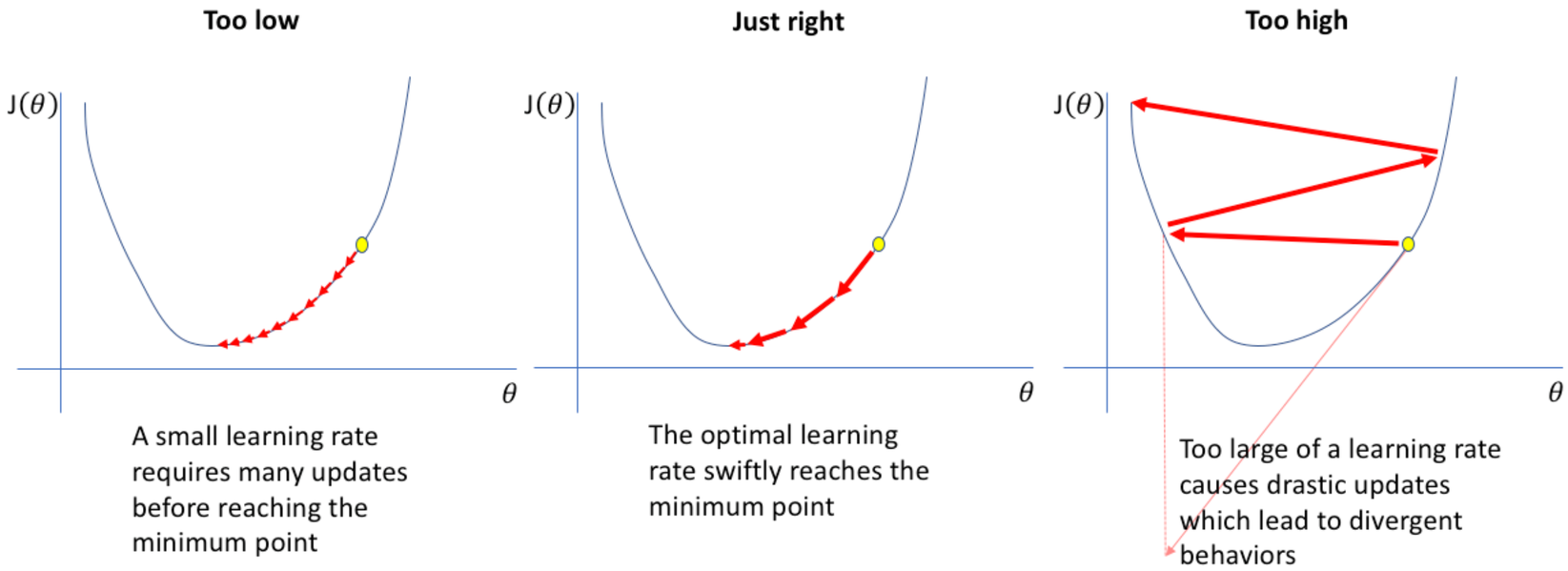
Loss function usually is:

$$W_{new} = W_{old} - \frac{d(J(w))}{d(w)}$$

We will add a new parameter called: ***Learning Rate***

# Optimization Techniques

## Understanding Learning Rate and Its Impact



# Optimization Techniques

## How do we specify the optimal learning rate?

1. Try lots of different learning rates and see what works "just right”
2. Design an adaptive learning rate that “Adapts” to landscape *(We will see this in the hands-on practice)*

# Deep Learning In Practice

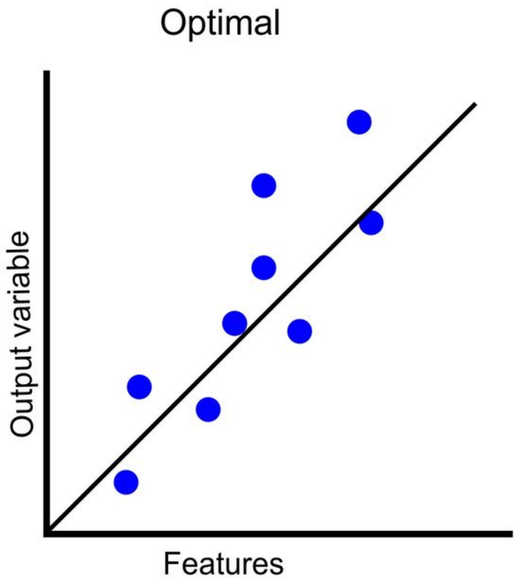
# Overfitting

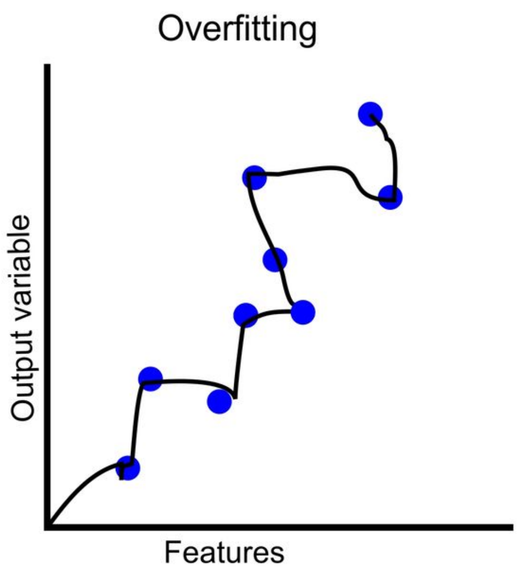
## Define the problem

Overfitting is when the model learns the training dataset, but it’s not able to generalize to the testing dataset.

# Overfitting

## Define the problem



# Overfitting

## Detect Overfitting

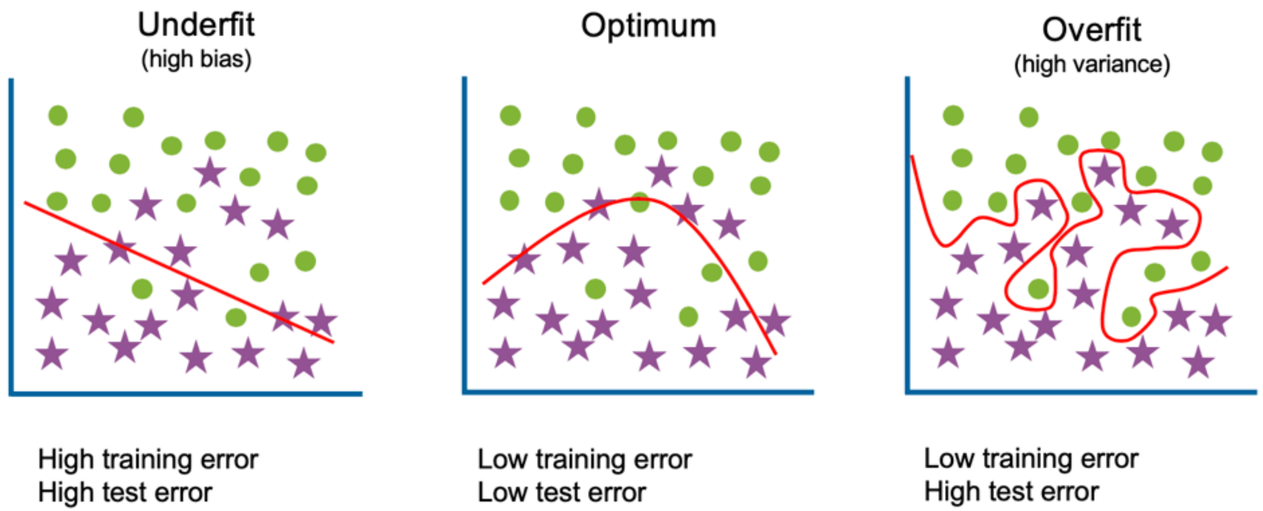
There are several methods to detect overfitting, here are the main ones:

1. Observe the graph of the training metrics

2. If the training loss < testing loss, it means the model was not able to generalize -> Overfitting

# Overfitting

## Detect Overfitting



# Overfitting

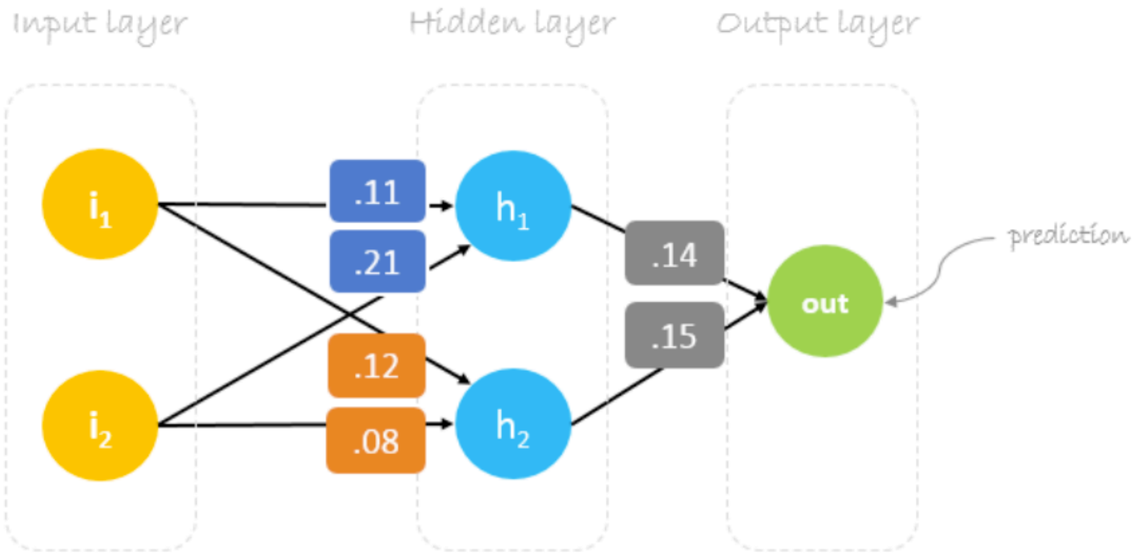
## Avoid Overfitting

1. Early Stopping
2. Train with more data
3. Data Augmentation
4. Regularization

# Numerical Example

# Consider the following Neural Network

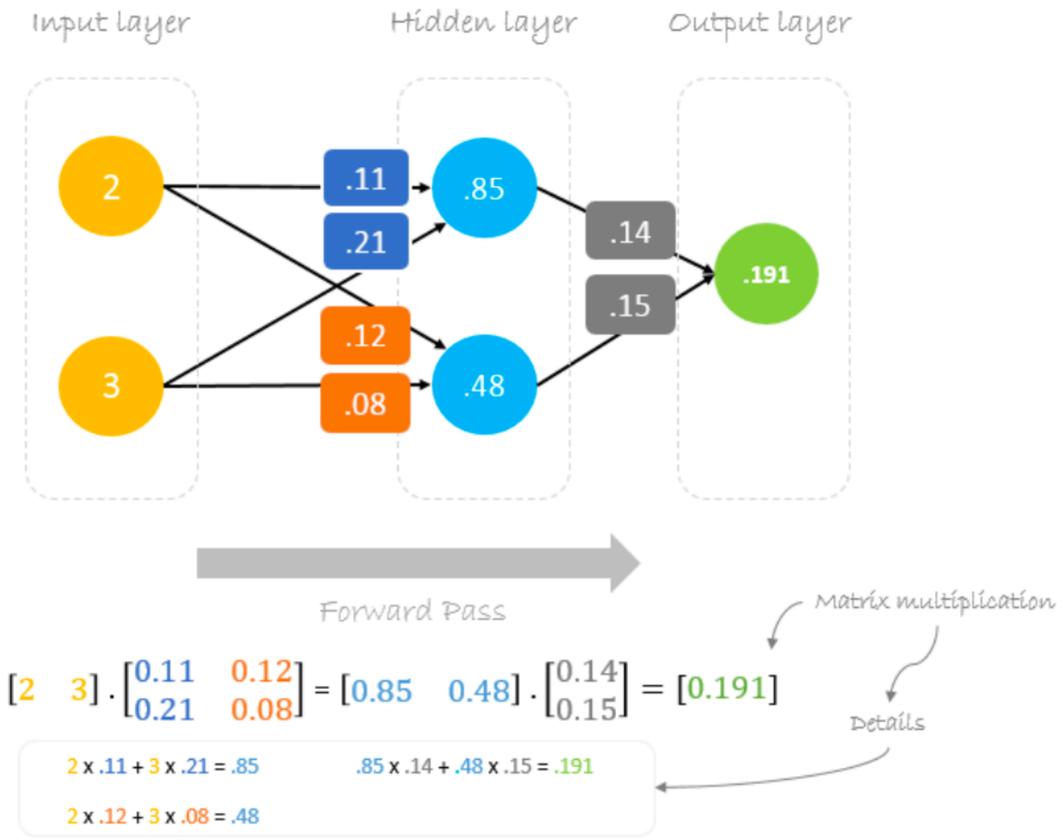
Network with initial weights




One sample from the dataset

# Consider the following Neural Network

- **Step 1: Forward pass**

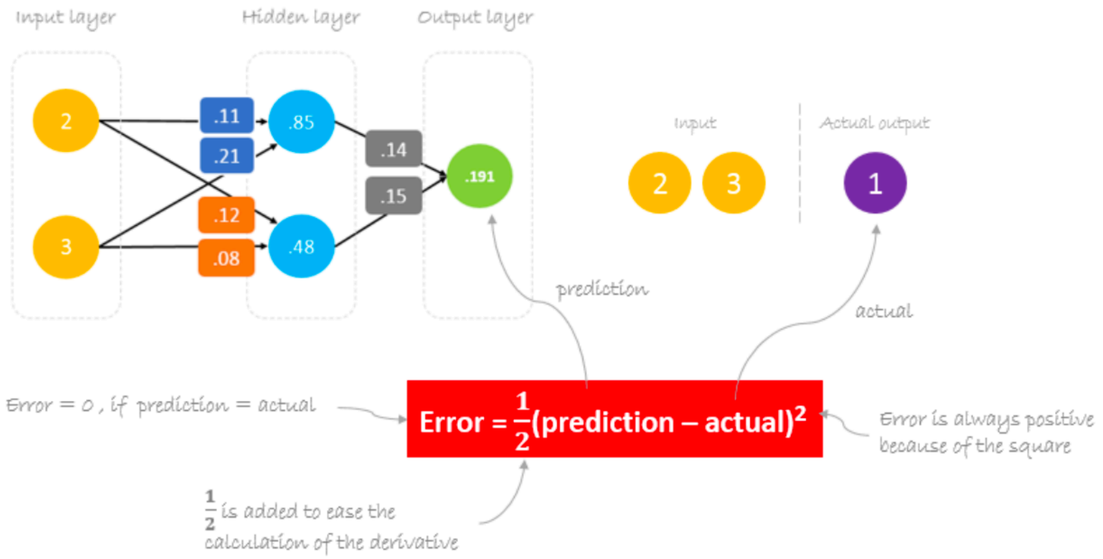



$Z = w0 + \sum wi*xi$

$W*X(T)$

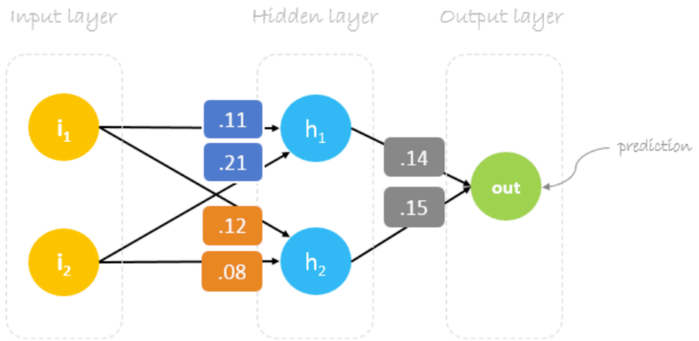
# Consider the following Neural Network

- **Step 2: Calculate error**

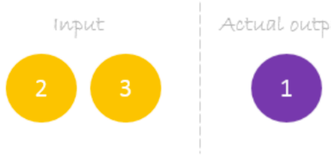


# Consider the following Neural Network

Network with initial weights




One sample from the dataset




- Step 3: Reducing error by updating weights using BP

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

Old weight; Derivative of Error with respect to weight; New weight; Learning rate

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

$$\text{prediction} = \text{out}$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$\text{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

to change ***prediction*** value, we need to change ***weights***

# Consider the following Neural Network

- Step 3: Reducing error by updating weights using BP (Output layer)

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

Error = $\frac{1}{2}$(prediction – actual)$^2$

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial \left(i_1 w_1 + i_2 w_2\right) w_5 + \left(i_1 w_3 + i_2 w_4\right) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$

$h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (\text{predictoin} - \text{actula}) * (h_2)$$

Δ = prediction – actual

delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

$$^*W_6 = W_6 - a\,\Delta h_2 \qquad ^*W_5 = W_5 - a\,\Delta h_1$$

# Consider the following Neural Network

- **Step 3: Reducing error by updating weights using BP (hidden layer)**

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

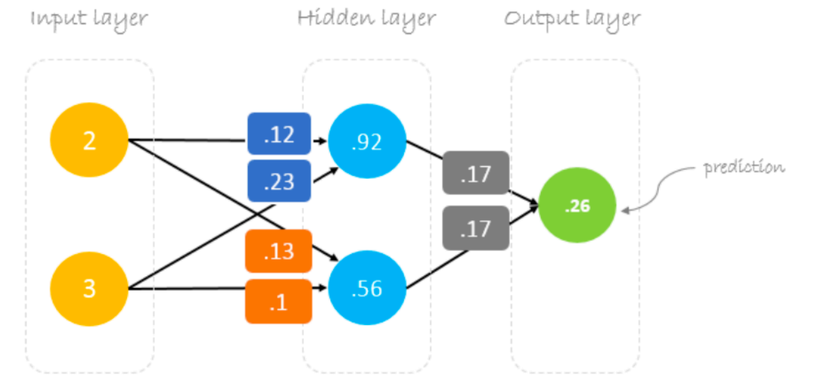
$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix}.[0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

updated weights

$$^{*}w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$
$$^{*}w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$
$$^{*}w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$
$$^{*}w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$
$$^{*}w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$
$$^{*}w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

# Consider the following Neural Network

- Using the updated weights, what happens if we feed the same sample again ?
- Prediction is closer



# Neural Network Architectures

# Neural Network Architectures

## Introduction

Neural network architectures refer to the structure and design of neural networks, including the arrangement and connection of layers, neurons, and the types of operations performed.

# Neural Network Architectures

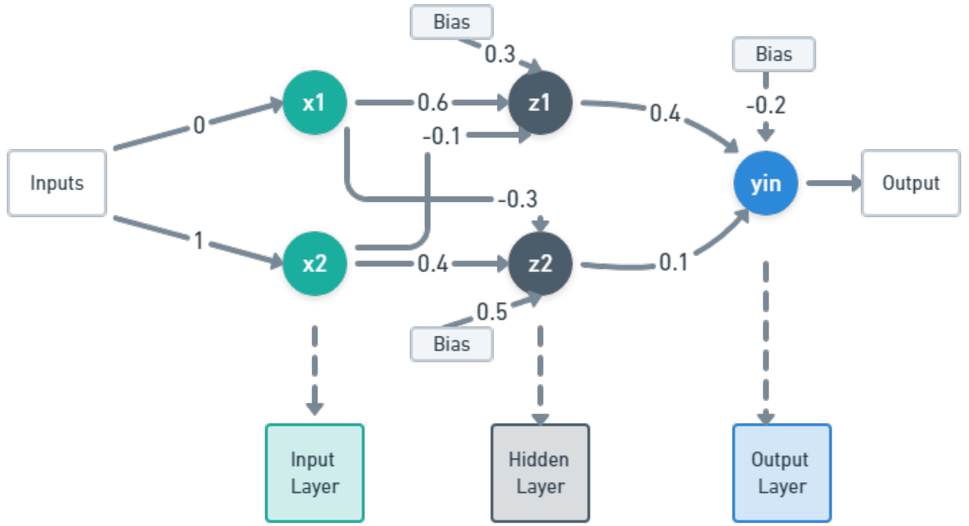
## Introduction

Different architectures are suited to different types of data and tasks, making it crucial to choose the right architecture for the problem at hand.

# Feedforward Neural Networks

## FNN

Neural network architectures refer to the structure and design of neural networks, including the arrangement and connection of layers, neurons, and the types of operations performed.



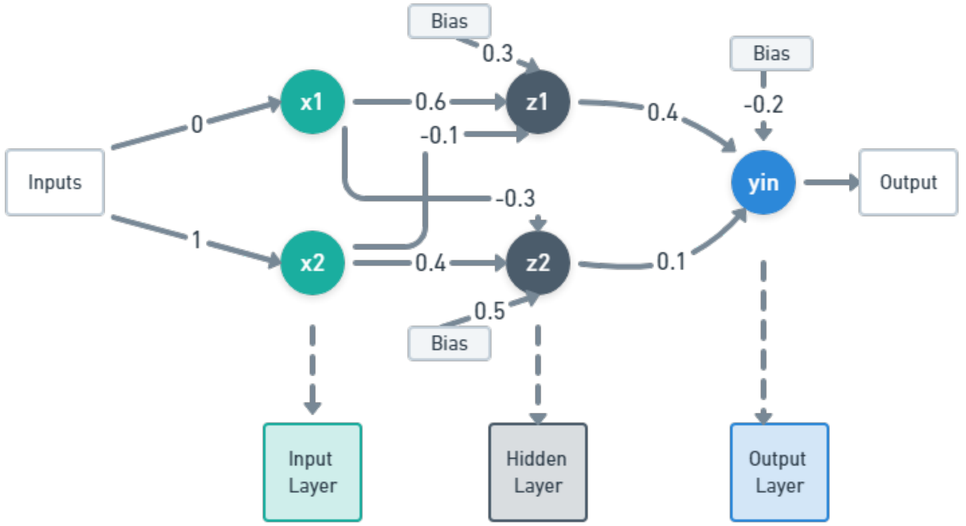
# Feedforward Neural Networks

## FNN

If it’s feedforward, does it do back propagation?

Of course! Feedforward means that the direction of data flow is from input -> hidden -> output, without cycles (As we will see in a bit).

Use cases: Classification and Regression



# Convolutional Neural Network

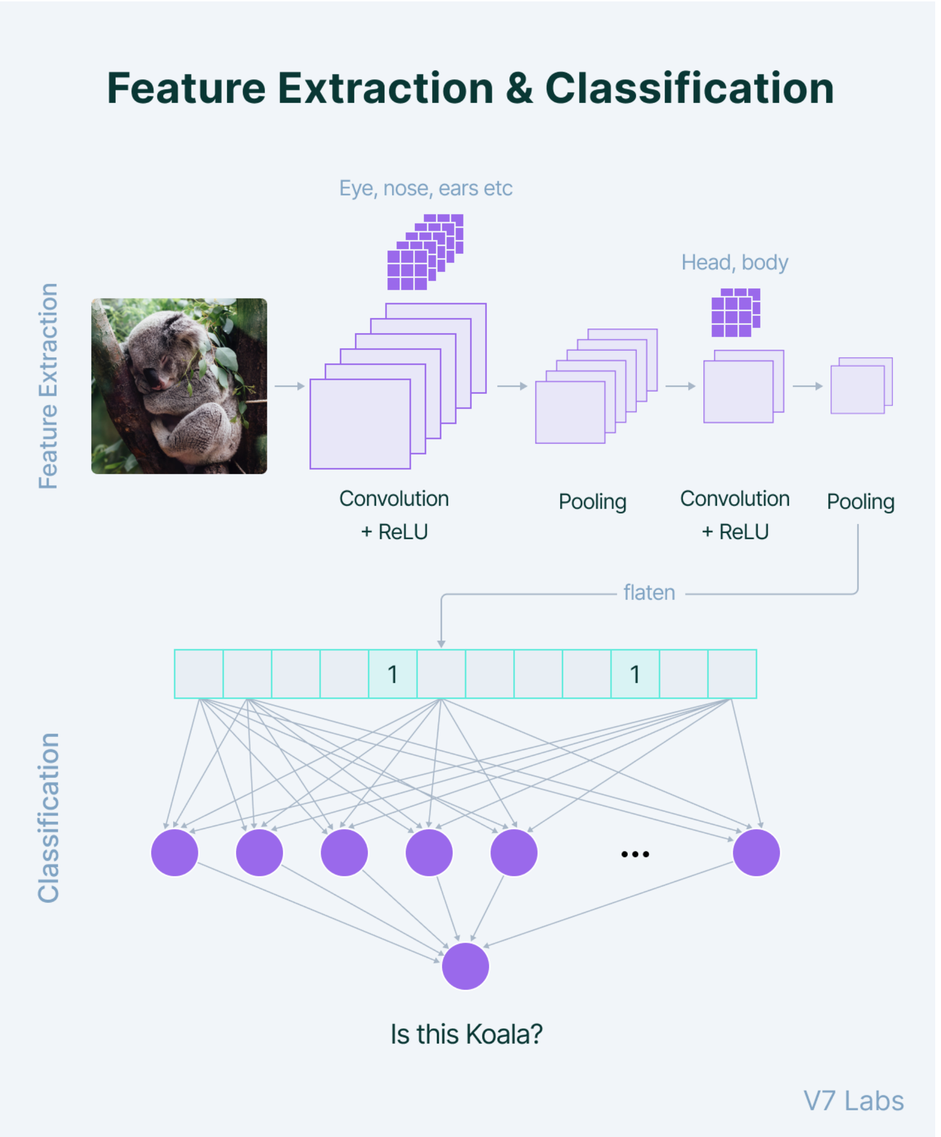
## CNN

Specialized in processing grid-like data, such as images. They use convolutional layers to automatically learn spatial hierarchies of features.

Use case:

Everything related to images

Computer Vision

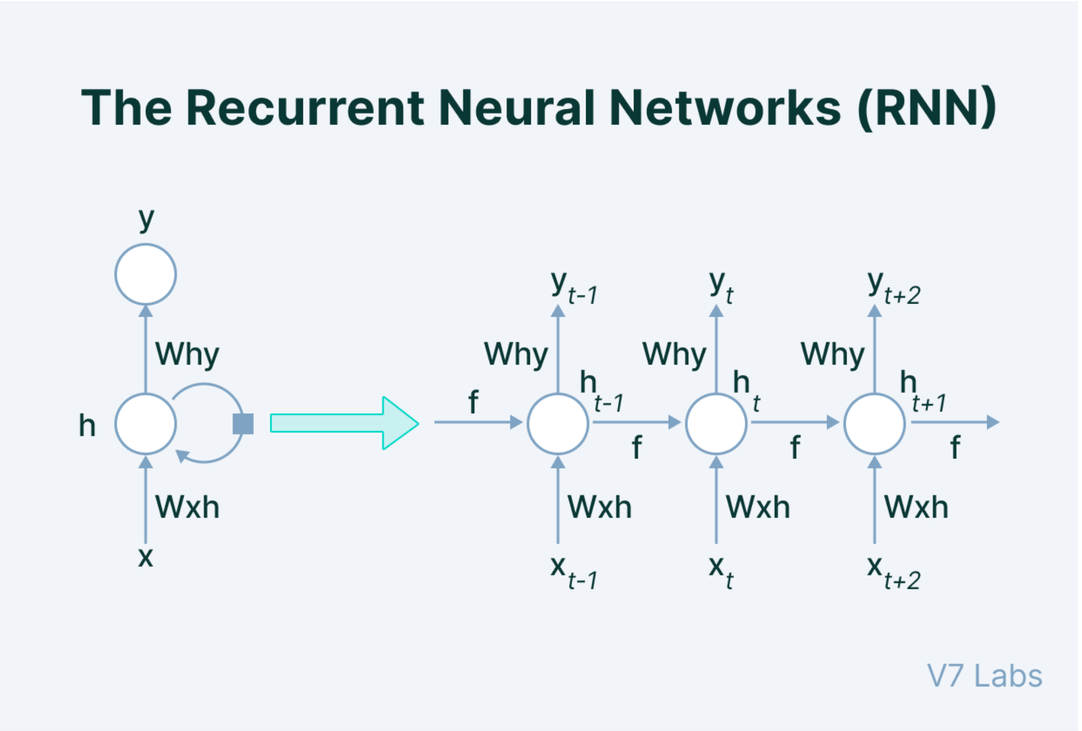


# Recurrent Neural Networks

## RNN

RNNs have the power to remember what it has learned in the past and apply it in future predictions.

The input is in the form of sequential data that is fed into the RNN, which has a hidden internal state that gets updated every time it reads the following sequence of data in the input.



# Recurrent Neural Networks

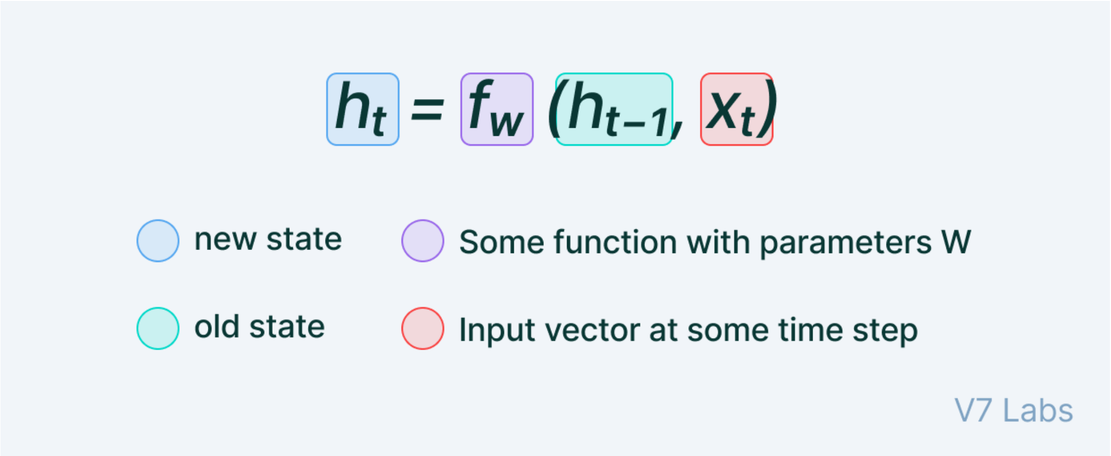
## RNN

Use cases:

1. Natural language processing (NLP)

2. Time series prediction

3. speech recognition.

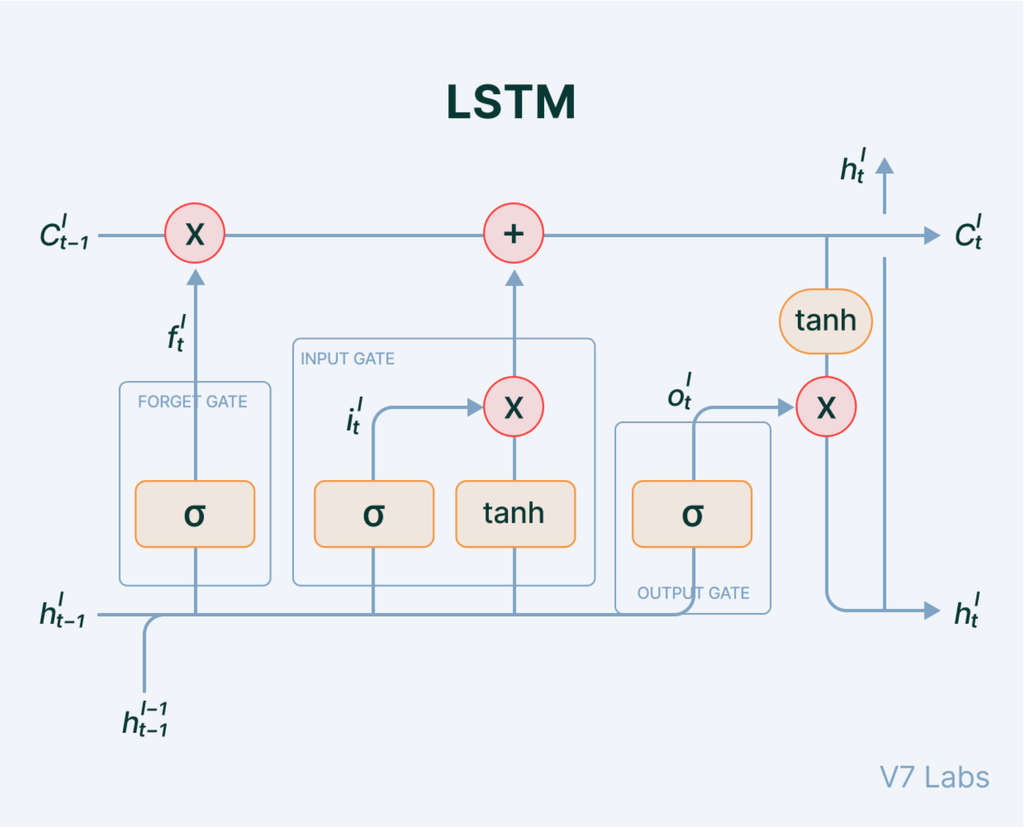


# Long Short Term Memory

## LSTM

In RNN each of our predictions looked only one timestamp back, and it has a very short-term memory. It doesn't use any information from further back.

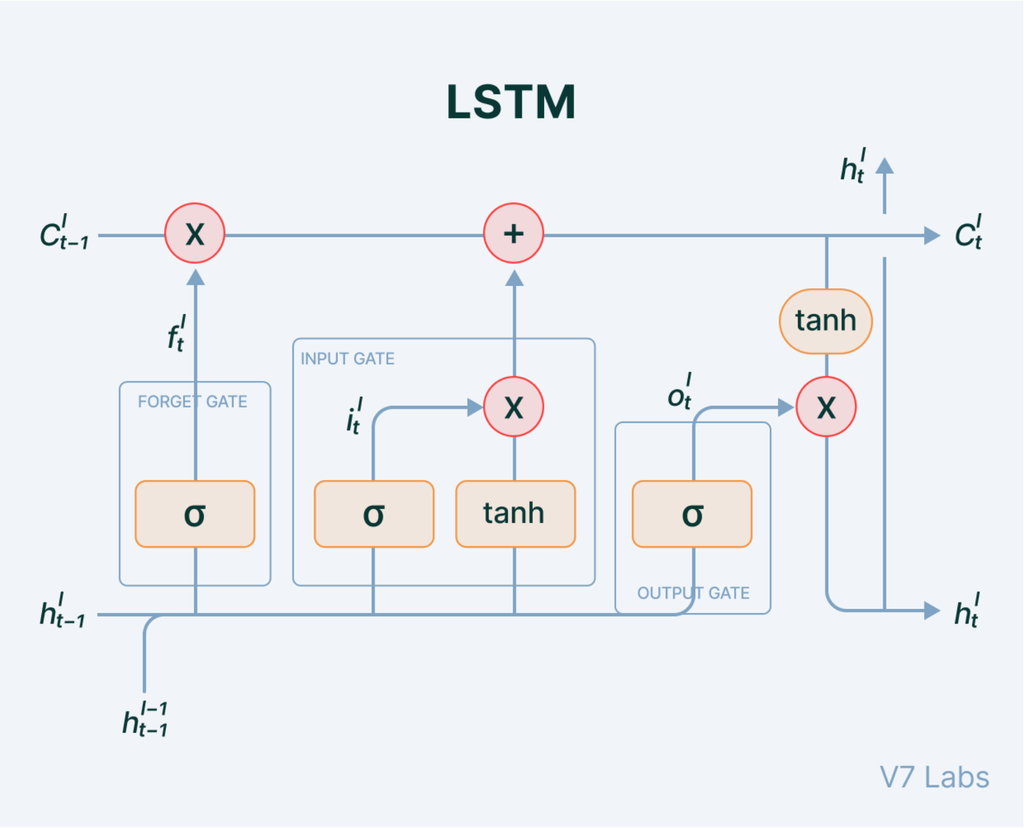
To rectify this, we can take our Recurrent Neural Networks structure and expand it by adding some more pieces to it.



# Long Short Term Memory

## LSTM

A special type of RNN that can learn long-term dependencies by using memory cells that retain information over long periods.



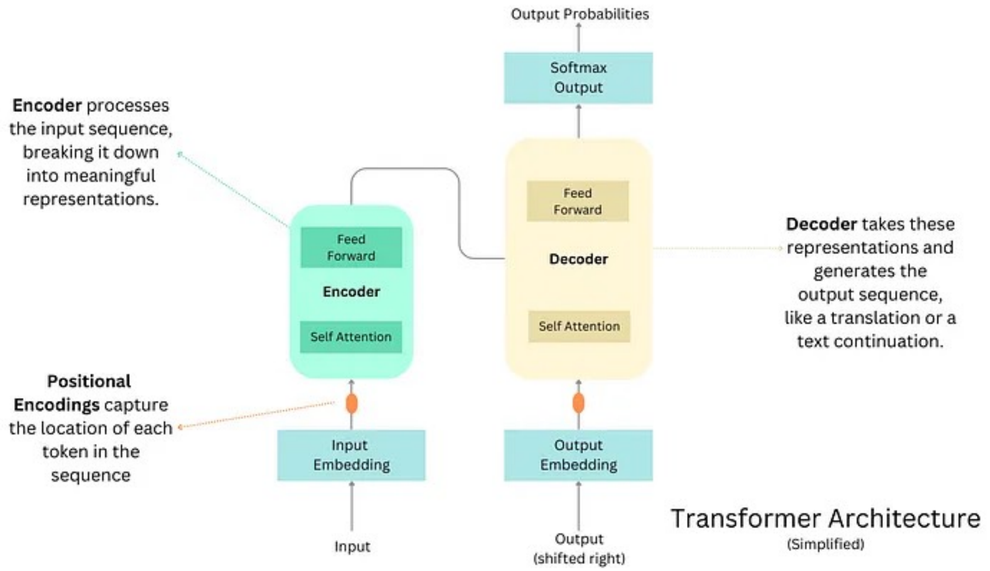

Use Case:

Sequential data tasks where long-term context is important, such as text generation and machine translation.

# Transformers

RNNs are slow and take too much time in training. They are not very good with large sequenced data and lead to vanishing gradients. LSTMs that were introduced to bring memory in the RNN became even slower to train.

For both RNN and LSTM, we need to feed the data sequentially or serially. This does not make use of GPUs.



# Transformers

Transformers are a type of neural network architecture that transforms or changes an input sequence into an output sequence.