# Introduction to Deep learning

Dr. Sagarika B

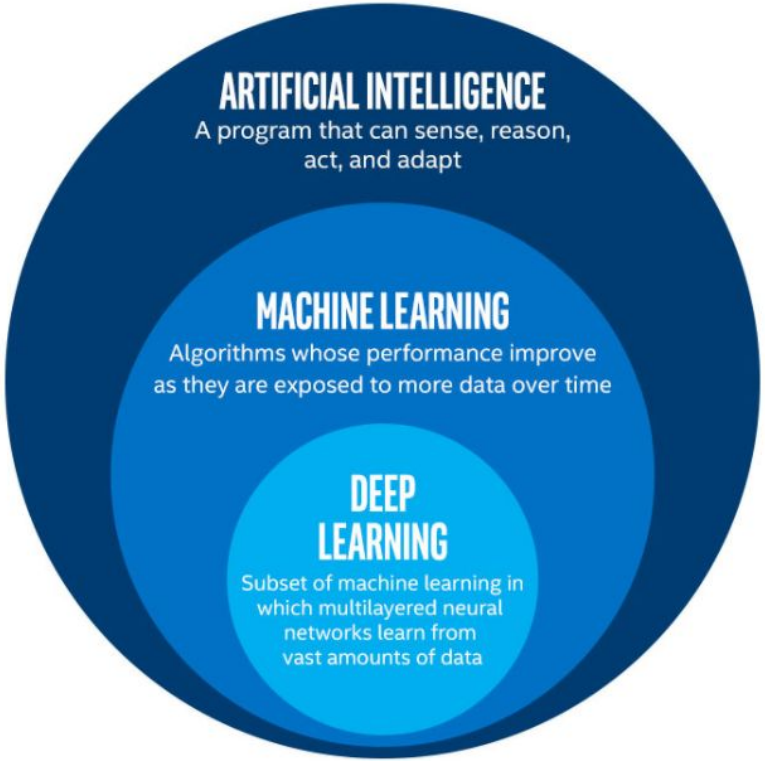
# Session 1

- The What, Why and How of deep learning
- Applications of deep learning
- Introduction to tensorflow and keras
- Basic tensor functions
- Perceptron concept

# What is Deep learning?

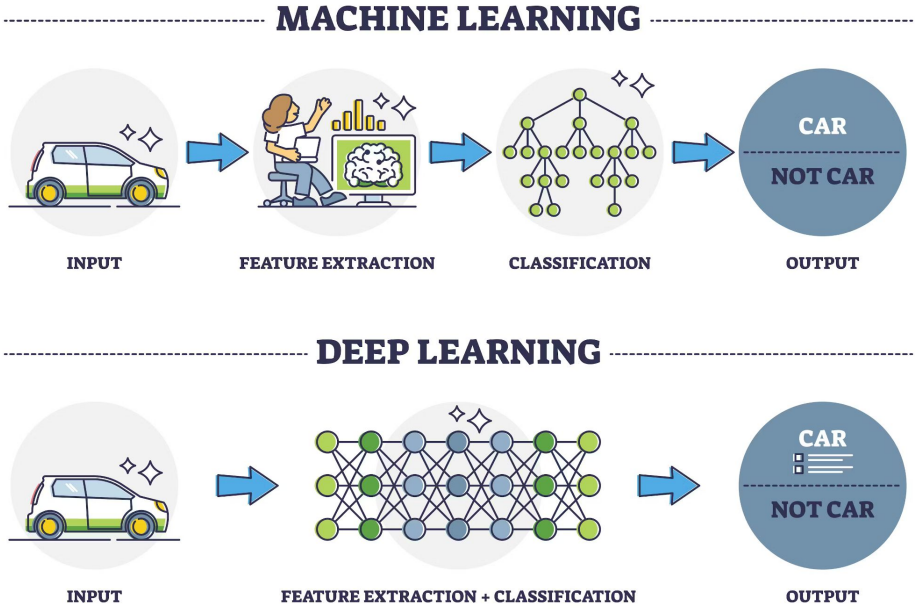
*Definition*: DL is a subfield of AI and ML that is inspired by the structure of human brain.

DL algorithms attempt to draw similar conclusions as humans would by continually analyzing data with given logical structure called neural network.



# How it learns ?

Deep learning algorithms attempt to learn (multiple levels of) representation of data by using a hierarchy of multiple layers.



## Traditional machine learning

**Input** → **Feature extractor** → **Machine learning model** → **Output**

(e.g.,  )

(e.g., color and texture histograms)

(e.g., support vector machines and random forests)

(e.g., presence or not or disease)

## Deep learning

**Input** → **Deep Learning Model** → **Output**

(e.g.,  )

(e.g., convolutional and recurrent neural networks)

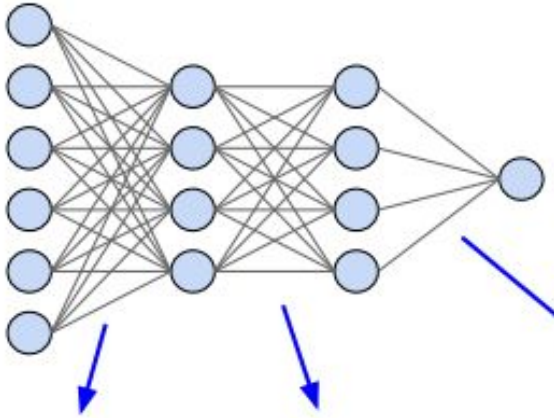
(e.g., presence or not or disease)

# Learning representations from data

- To learn useful representations of the input data at hand, representations that get us closer to the expected output.
  - Input data points
  - Examples of the expected output
  - A way to measure whether the algorithm is doing good job

**Input**

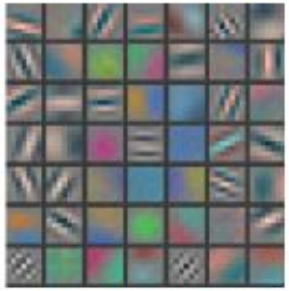
(e.g.,  )



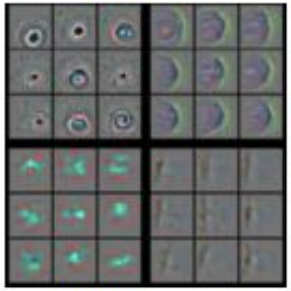
**Output**

(e.g., presence or not or disease)

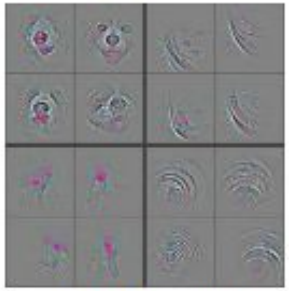
Hierarchical structure of neural networks allows compositional extraction of increasingly complex features



Low-level features



Mid-level features

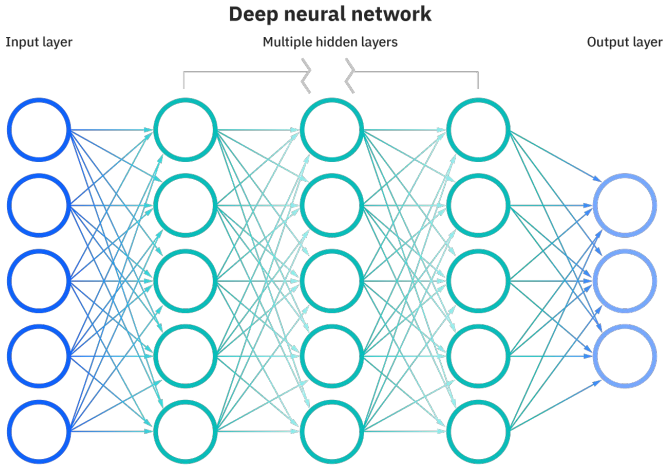


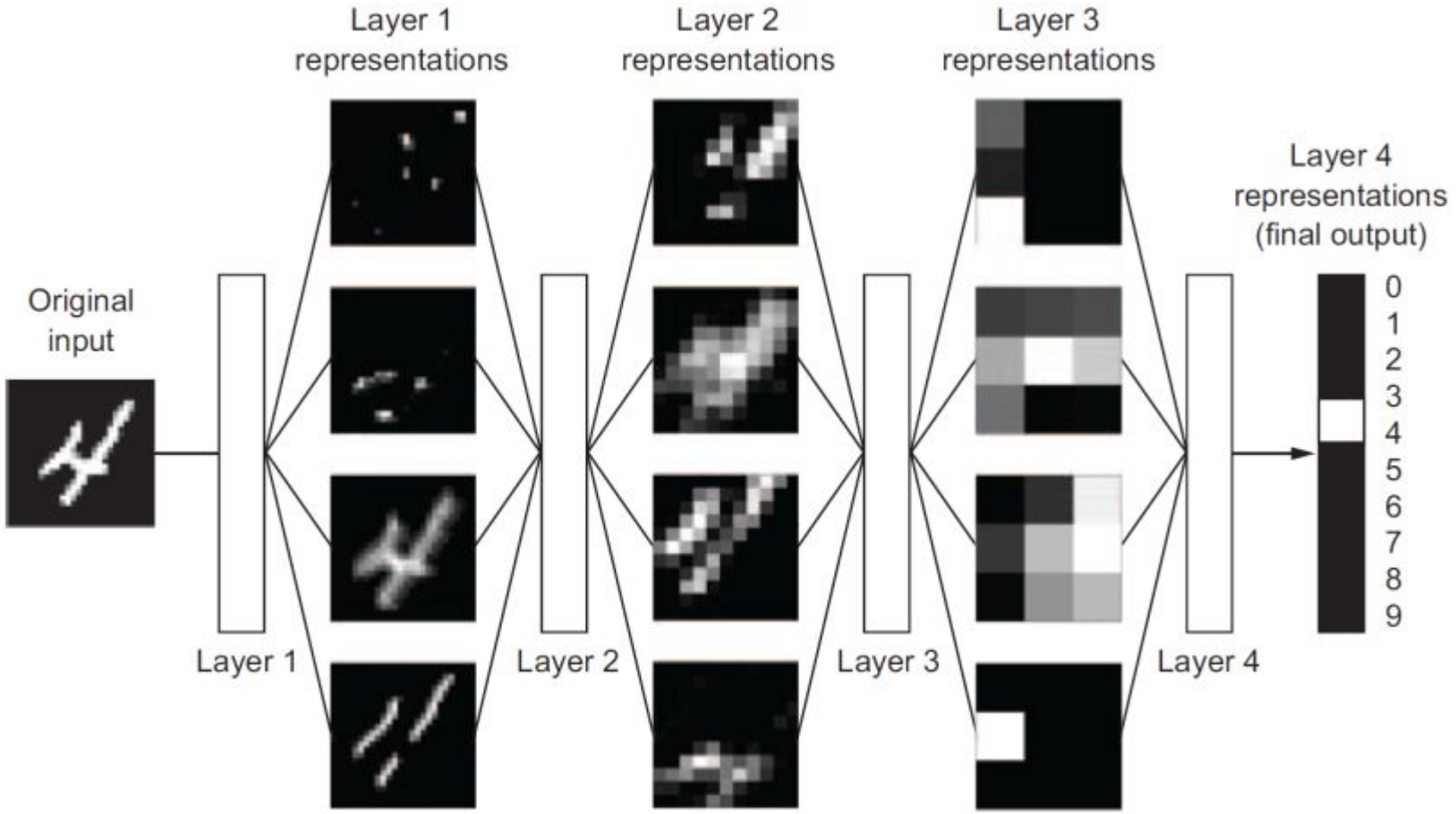
High-level features

Feature visualizations from Zeiler and Fergus 2013

# How “Deep” ?

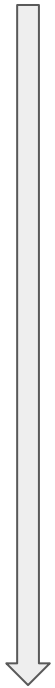
- Learning representations from data that puts an emphasis on incremental learning of successive layers with increasingly meaningful and complex representations
- “Deep” = successive layers of representations. The more the layers, the deeper the architecture.
- These layered representations are learned via models called *neural networks*.






Eg. - Digit classification

# The timeline



1950-1980 : Symbolic AI  (Knowledge base), Rule based systems

1960: Perceptron

1978: Multilayer perceptron

1989: ANN , Yan Lecun

1990-2000: AI winter

2006: Deep belief networks

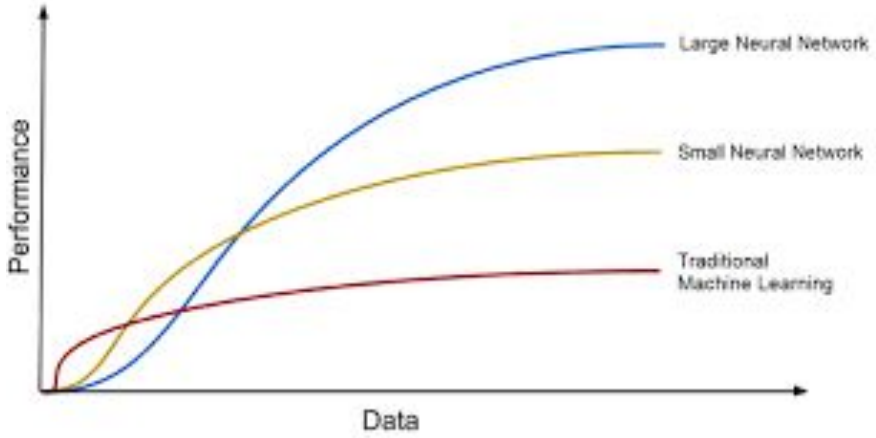
2012: Imagenet computation on GPU

# Why Deep learning now?

- Datasets availability (labeled)
- Hardware
- Frameworks / Libraries

  (Tensorflow, SKlearn, keras, PyTorch etc.)

- Architectures
- Community



# Difference between DL and ML

Data dependency

Hardware dependency

Training time

Feature selection

Interpretability

# Where do we use Deep learning?

- **Google Photos** - Google LLC: Google Photos uses deep learning for various features like image classification, object recognition, and automatic organization of photos based on people, places, and things.
- **Amazon Alexa** - Amazon.com, Inc.: Alexa, Amazon's virtual assistant, employs deep learning for natural language understanding, speech recognition, and personalized responses to user queries.
- **Netflix Recommendations -** Netflix, Inc.: Netflix uses deep learning algorithms to analyze user preferences, viewing history, and interactions to provide personalized movie and TV show recommendations.
- **Tesla Autopilot** - Tesla, Inc.: Tesla's Autopilot feature utilizes deep learning for autonomous driving capabilities, including object detection, lane detection, and decision-making based on sensor data.
- **Facebook News Feed** - Meta Platforms, Inc. (formerly Facebook, Inc.): Facebook employs deep learning algorithms to personalize users' news feeds by analyzing their interests, interactions, and social connections.

- **Apple Siri - Apple Inc**.: Siri, Apple's virtual assistant, utilizes deep learning for natural language understanding, voice recognition, and responding to user commands on Apple devices.
- **IBM Watson** - IBM Corporation: IBM Watson leverages deep learning techniques for various applications, including healthcare diagnostics, natural language processing, and data analysis.
- **Uber Eats Recommendations -** Uber Technologies, Inc.: Uber Eats employs deep learning algorithms to provide personalized food recommendations to users based on their past orders, preferences, and restaurant popularity.
- **Pinterest Visual Search -** Pinterest, Inc.: Pinterest utilizes deep learning for visual search capabilities, allowing users to search for visually similar images by analyzing their visual features.

# What do we need to use a deep learning model?

1. Data
2. Computational Resources
3. Deep Learning Framework
4. Model Architecture
5. Training Process
6. Evaluation Metrics

# TensorFlow

TensorFlow is general purpose Python programming language based open-source end-to end platform Developed by Google Brain Team for creating Machine Learning applications.

Platform for high dimensional computation and implementing complex deep learning models.

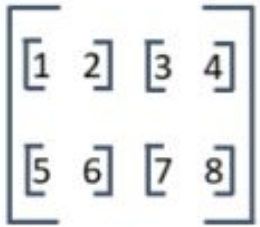
# What is a tensor ?

A tensor is a mathematical object representing a multi-dimensional array of data. In simpler terms, you can think of tensors as containers for data of different dimensions, such as scalars, vectors, and matrices.

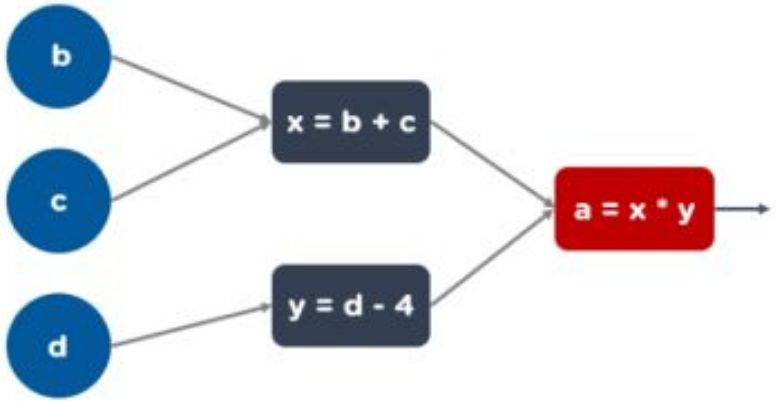
Tensorflow supports GPU computing

TensorFlow

# Tensor + Data Flow Graph

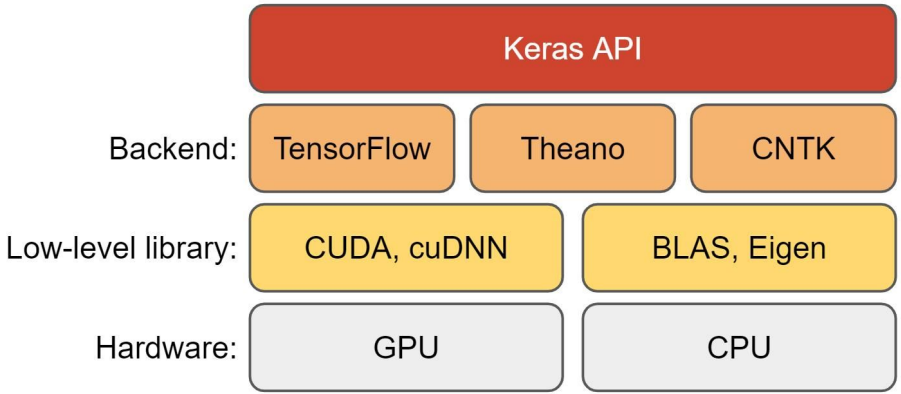



Tensor



# Keras

- Keras is a high-level neural network API
- Written and implemented in Python
- Can run on top of TensorFlow
- It was designed keeping fast experimentation in mind.




- **Supported Platforms**
  - Linux / Ubuntu
  - Windows
  - Mac OS
  - Raspberry Pi

# Google colab :-)

1. Free GPU and TPU Access
2. No Setup Required
3. Collaboration and Sharing
4. Integration with Google Services
5. Pre-installed Libraries and Dependencies
6. Flexible Environment:
7. Resource Scalability

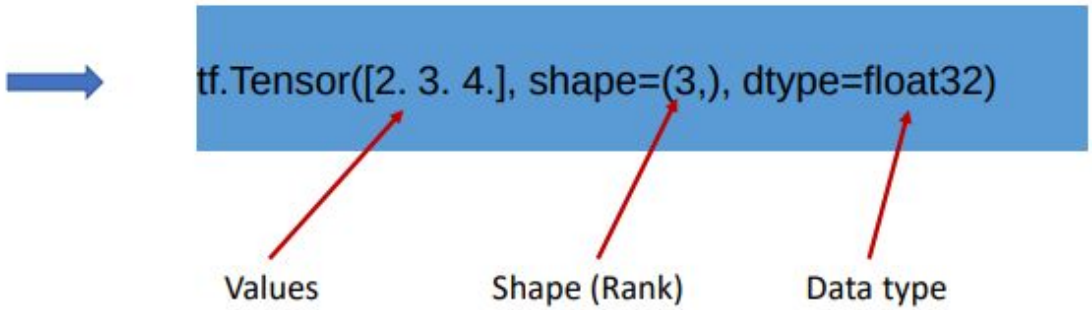
# Data representations for Neural networks

- Scalars (0D tensors)
- Vectors (1D tensors)
- Matrices (2D tensors)
- 3D tensors

# Components of a Tensor?

```
import tensorflow as tf

t = tf.constant( [2.0, 3.0, 4.0] )
print(t)
```



# Basic tensor operations

- Element wise operations
- Broadcasting
- Tensor dot
- Tensor reshaping

https://colab.research.google.com/drive/1018448zu8ZIntwyFthlmFrjPkYVN3SmR#scrollTo=vGvF21oEE1mE

# Simple linear regression (Recap)

- Model $y = \beta_1 x + \beta_0$
- Weights/ parameters $\beta_0, \beta_1$
- Loss function $mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Optimization (GD, Least squares)

$$\beta_{0_{new}} = \beta_{0_{old}} - \eta \frac{\delta loss}{\delta \beta_{0_{old}}}$$

$$\beta_{1_{new}} = \beta_{1_{old}} - \eta \frac{\delta loss}{\delta \beta_{1_{old}}}$$

# Perceptron !!

- 1960, Frank Rosenblatt
- Perceptron is a linear classifier used in supervised learning.
- Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.

# What is Perceptron ?

A perceptron is responsible to find a line of the form: `y= WX + b`

- **X** is dimension size vector which forms the variables (eg. X = [x1, x2]),
- **W** is the coefficient vector corresponding to each X (eg. W = [w1, w2]),
- **b** is a scalar bias value added to the line equation.

- Positive(Blue) data point if `WX + b ≥ 0`
- Negative(Red) data point if `WX + b < 0`

# How does it work?

In the image the Negative point (Red) is in the Positive Region for our given line :

```
(3 * x1) + (4 * x2) — 10 = 0
```

Our objective is to somehow shift the line such that the red point comes below the new updated line, i.e., we need to bring the line closer to the Red point.

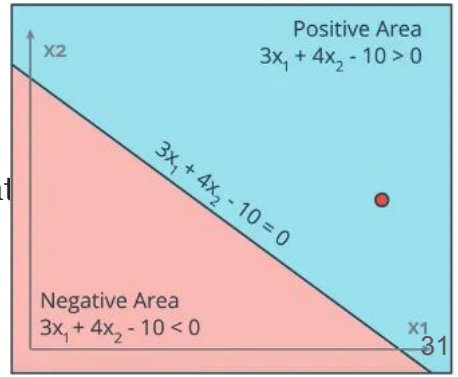
A neat trick for this is to do the following, suppose the red point has the coordinates (4,5):

1. For negative point in Positive region (i.e. WX + b ≥ 0):

- Take the line coefficients and subtract the misclassified point coordinate values from it.
- Also, subtract 1 from the bias value.

2. For positive point in Negative region (i.e. WX + b < 0)

- Take the line coefficients and add the misclassified point coordinate values int
- Also, add 1 to the bias value.



**Algorithm:** Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize **w** randomly;
**while** !*convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$ ;
    **if** $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
    **end**
    **if** $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
    **end**
**end**
//the algorithm converges when all the inputs are classified correctly

***1. Initialization:*** Begin by randomly initializing the weights of the perceptron algorithm. These weights are essentially the parameters that the model learns to adjust during training.

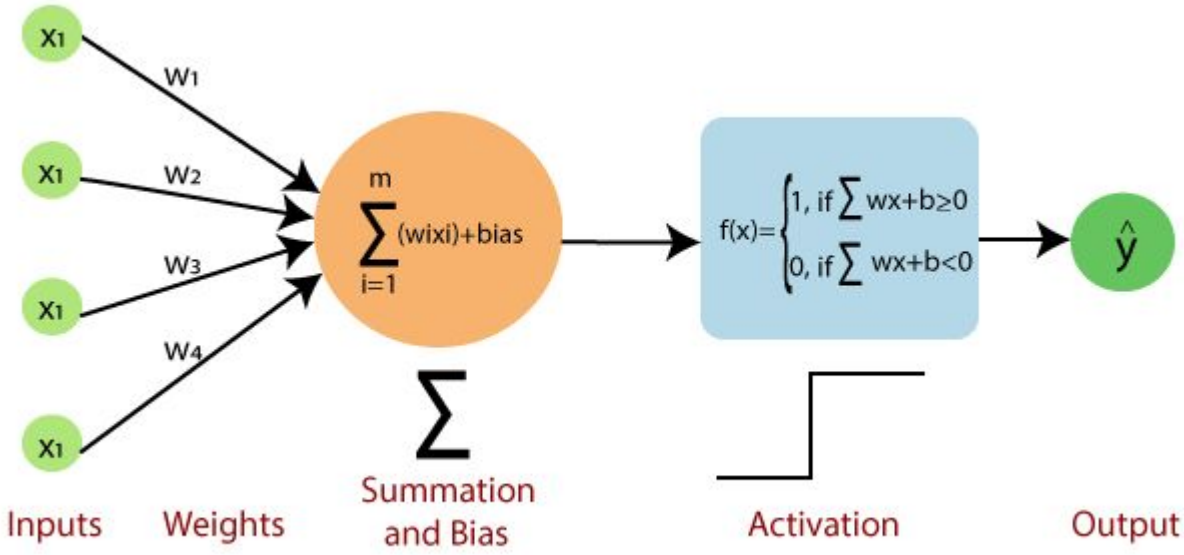
***2. Input Data:*** Feed the input data into the perceptron and compute the output using the current weights.

***3. Error Calculation:*** Compare the predicted output with the actual output. If the prediction is incorrect (i.e., a misclassification occurs), proceed to the next step.

***4. Weight Adjustment:*** Apply the Perceptron Trick to update the weights. This involves adding or subtracting a fraction of the input vector from the current weights, depending on whether the prediction was too high or too low.

***5. Repeat:*** Continue iterating through the dataset, updating the weights after each misclassification, until convergence or a predefined number of iterations.

# Perceptron


x1
x1
x1
x1
w1
w2
w3
w4
m
∑ (wixi)+bias
i=1
f(x)= 1, if ∑ wx+b≥0
0, if ∑ wx+b<0
ŷ
Inputs
Weights
Summation and Bias
Activation
Output

**Why do we need Weights and Bias?**

Weights shows the strength of the particular node.

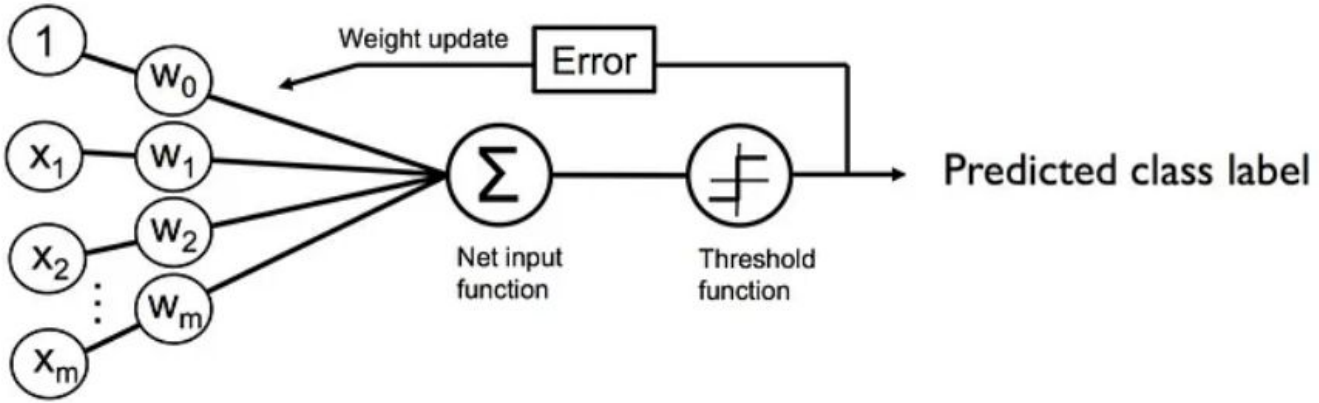
A bias value allows you to shift the activation function curve up or down.

Challenges:

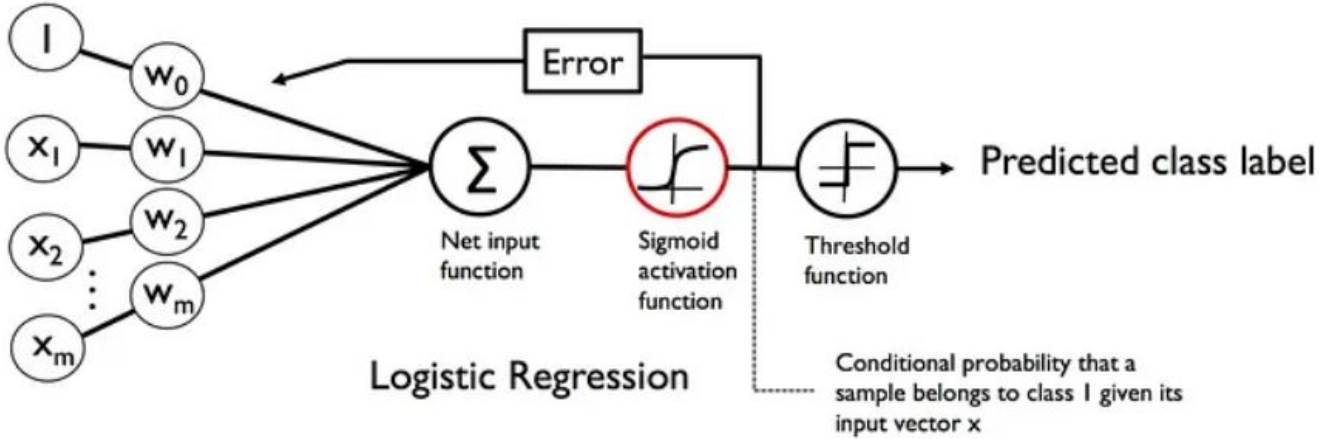
- We cannot quantify the error, How good is your model !!
- Since a step function is not differentiable, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.

Solution:

- Sigmoid activation function !!
- This introduced the probabilistic interpretation of class labels.
- The far the misclassified point from the classification line, the greater the error penalty value.




Perceptron




Logistic Regression

- If we want probability with respect to a particular class !!

Binary cross entropy

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

# Loss functions

Perceptron loss

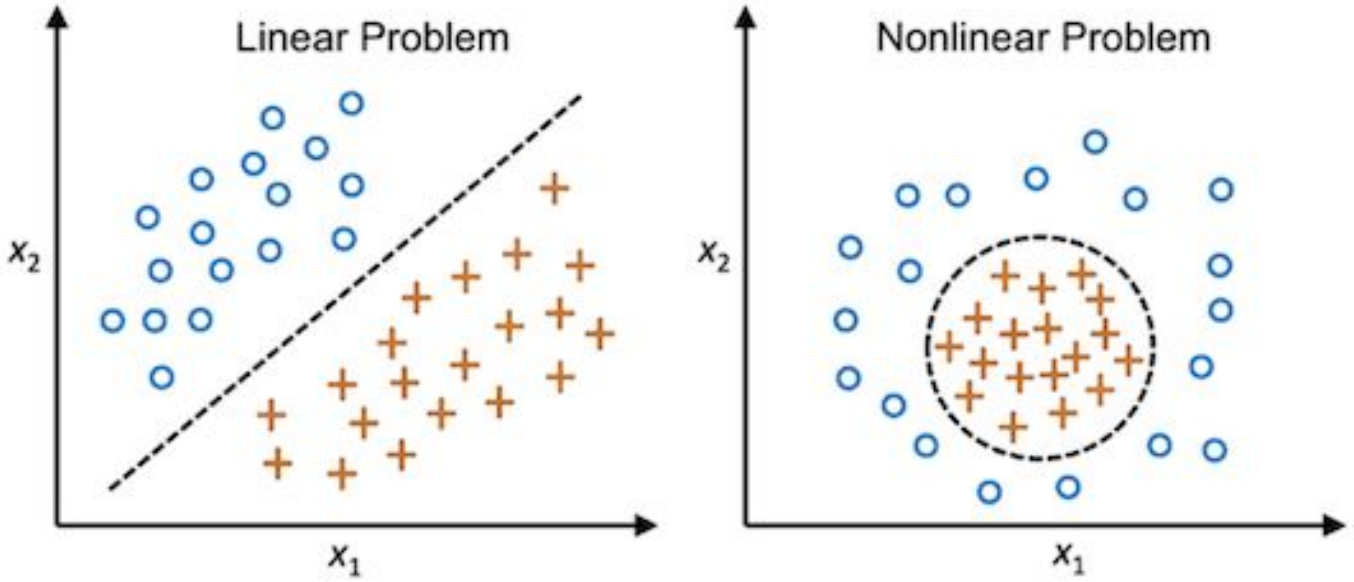
Binary cross entropy

# Where perceptron does not work?

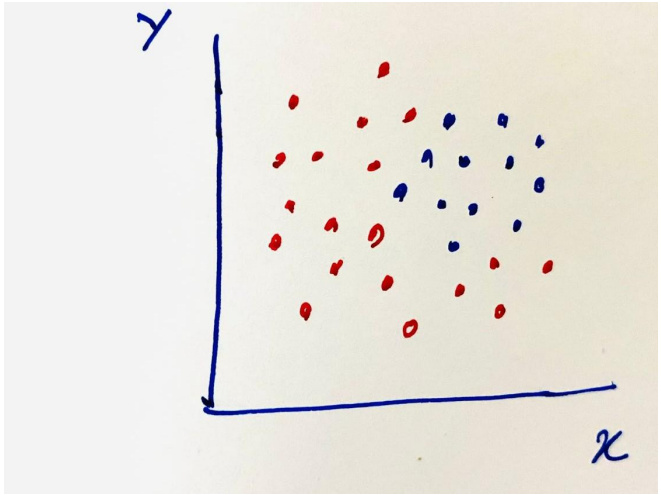
***1. Linear Separability:*** The Perceptron Trick only works when the data is linearly separable. In cases where the decision boundary is nonlinear, the perceptron may fail to converge.

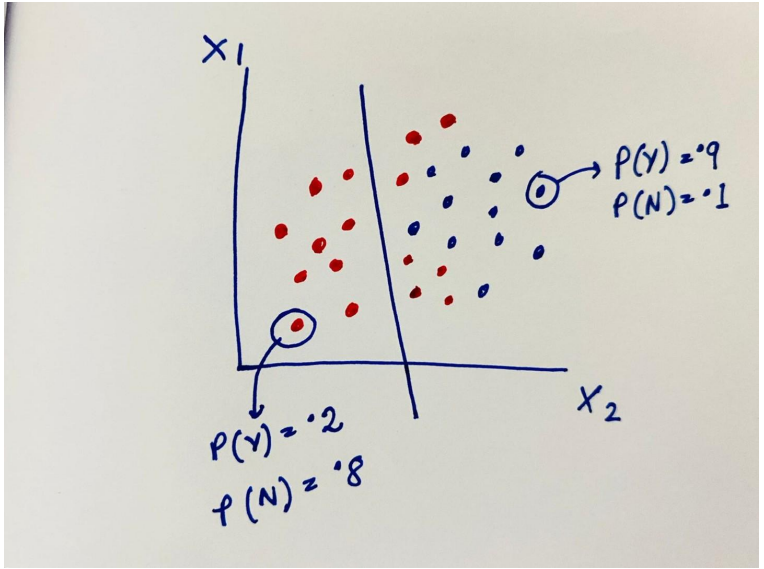
***2. Convergence Guarantee:*** There's no guarantee that the Perceptron Trick will converge if the data is not linearly separable. This can lead to infinite loops or suboptimal solutions.

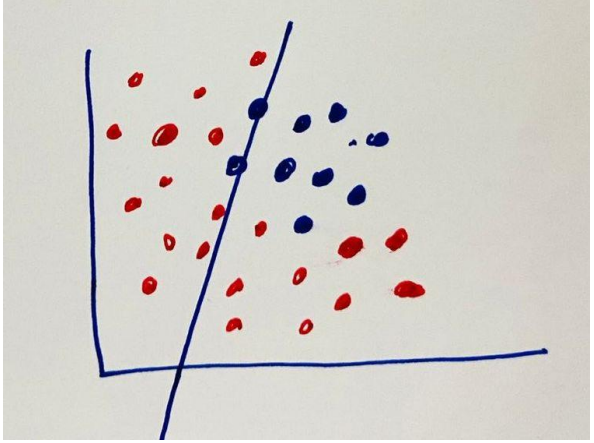
***3. Binary Classification:*** The Perceptron Trick is primarily suited for binary classification tasks and may not generalize well to multiclass problems without modifications.


Linear Problem
$x_2$
$x_1$
Nonlinear Problem
$x_2$
$x_1$

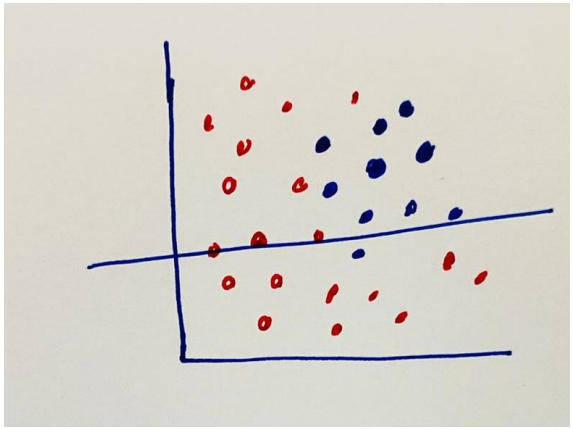
# Perceptron with Sigmoid Activation = Logistic regression



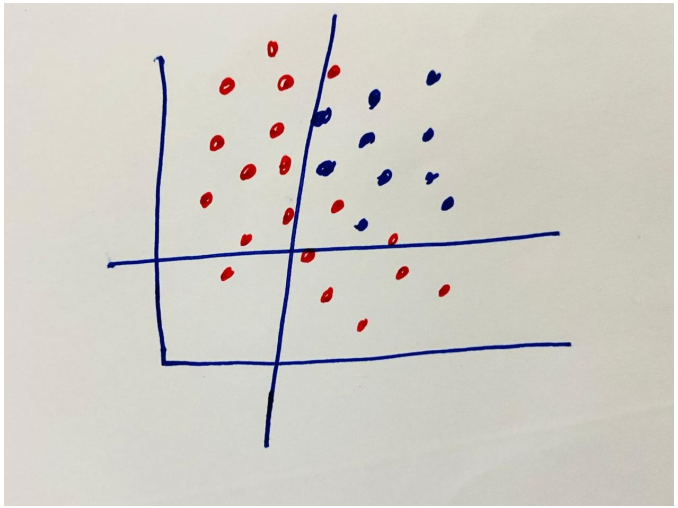


Perceptron 1

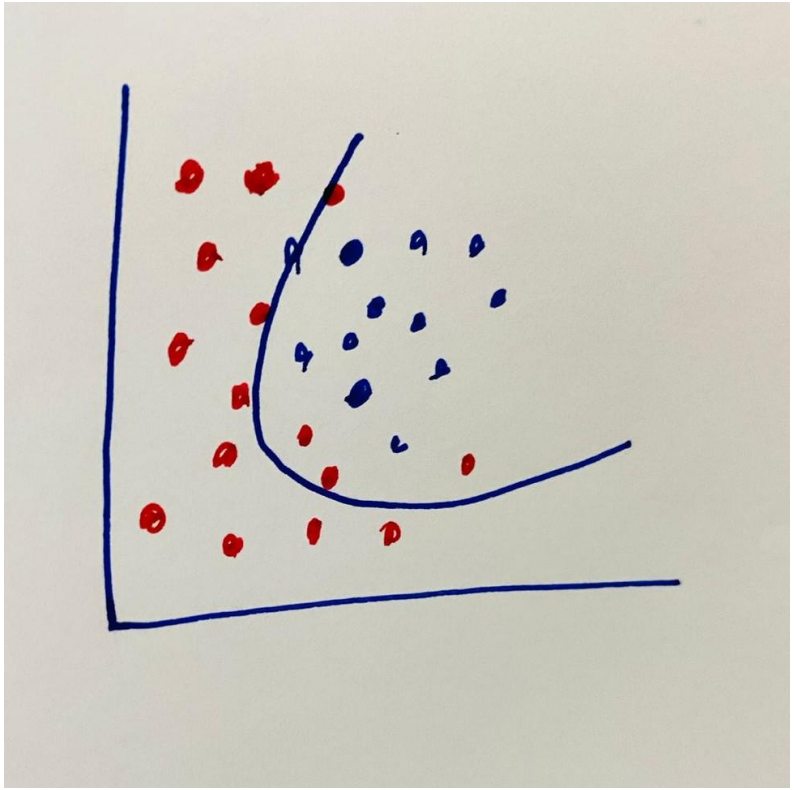


Perceptron 2

Can we use multiple perceptrons to create a non-linear decision boundary?



Ans: YES we can !!

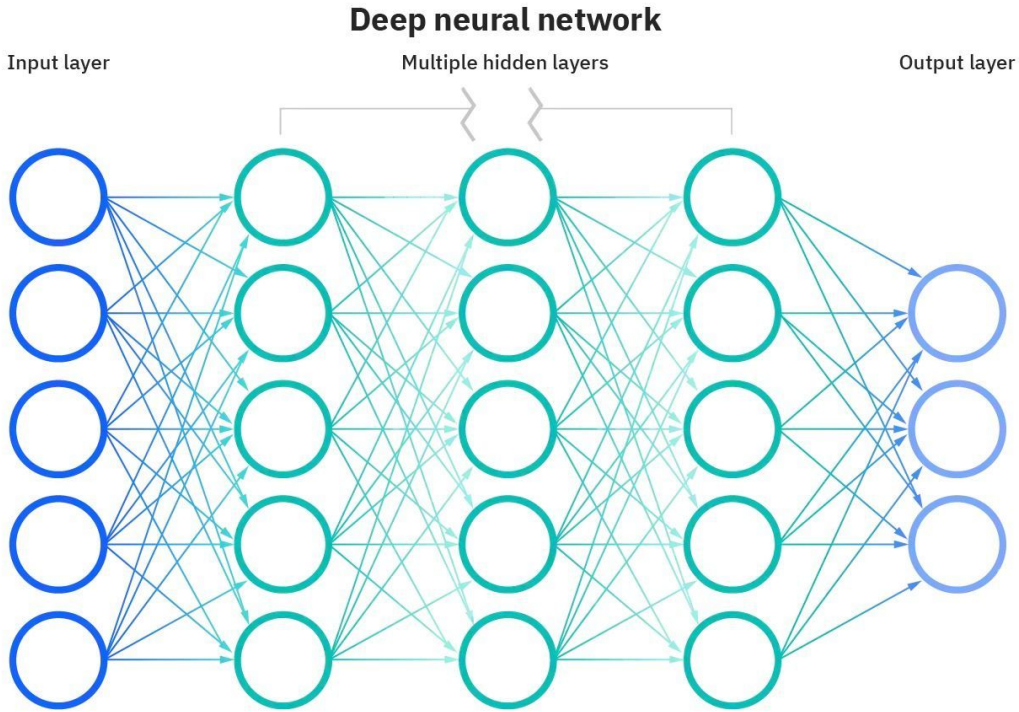


After applying
**Super imposition and smoothening**

**Linear combination** of the perceptron 1 and perceptron 2

# Multilayer Perceptron (MLP)

1. The **Input** layer (interface to accept data)
2. The **Hidden** layer(s) (Actual Neurons)
3. The **Output** layer (Actual Neurons to output the result)



Thank you

Books to refer

- Deep Learning by Grokking
  https://edu.anarcho-copy.org/Algorithm/grokking-deep-learning.pdf

- Deep learning in python
  https://www.manning.com/books/deep-learning-with-python

- Deep learning by Ian Goodfellow et al.
  https://www.deeplearningbook.org/