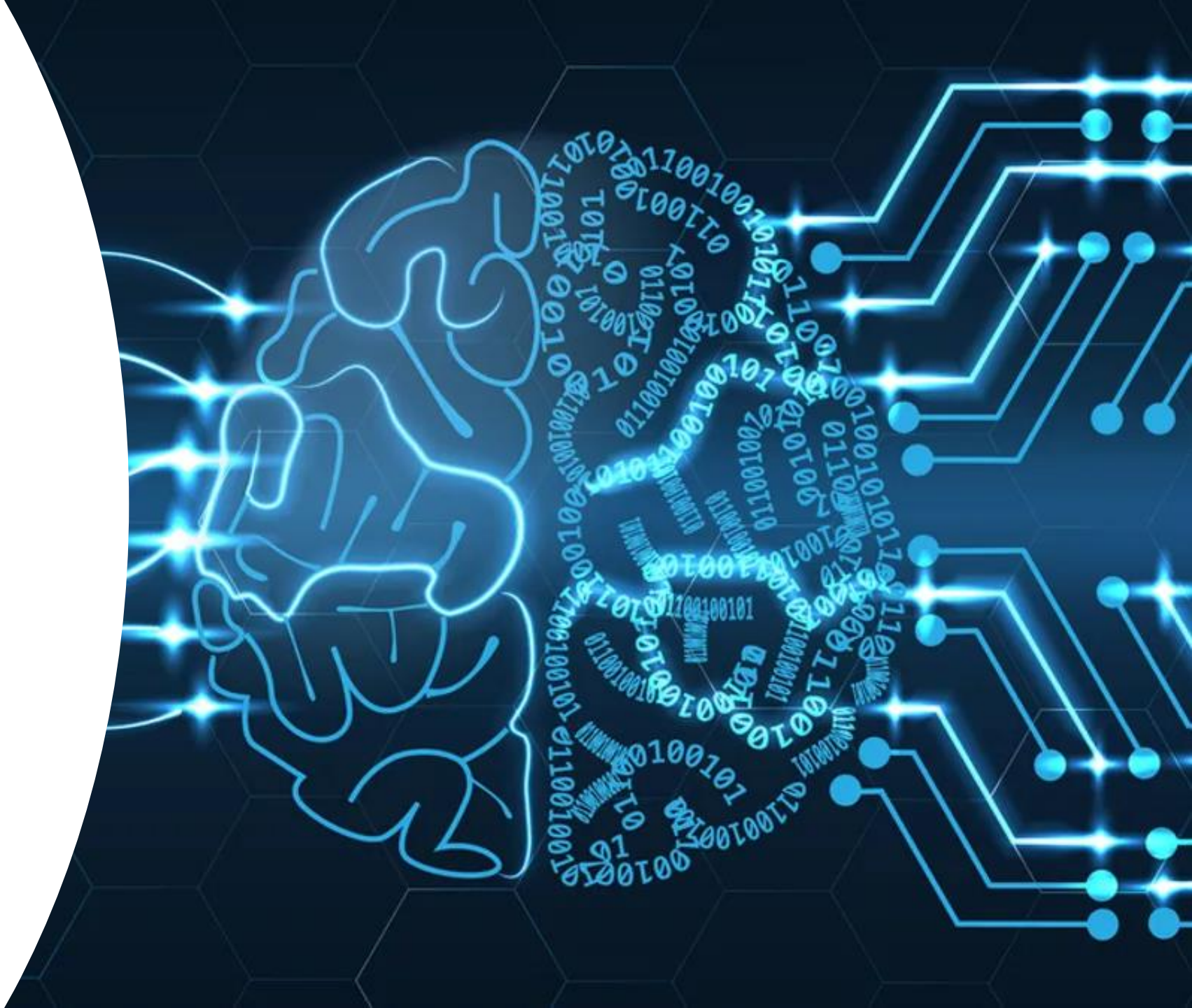
AMITY UNIVERSITY

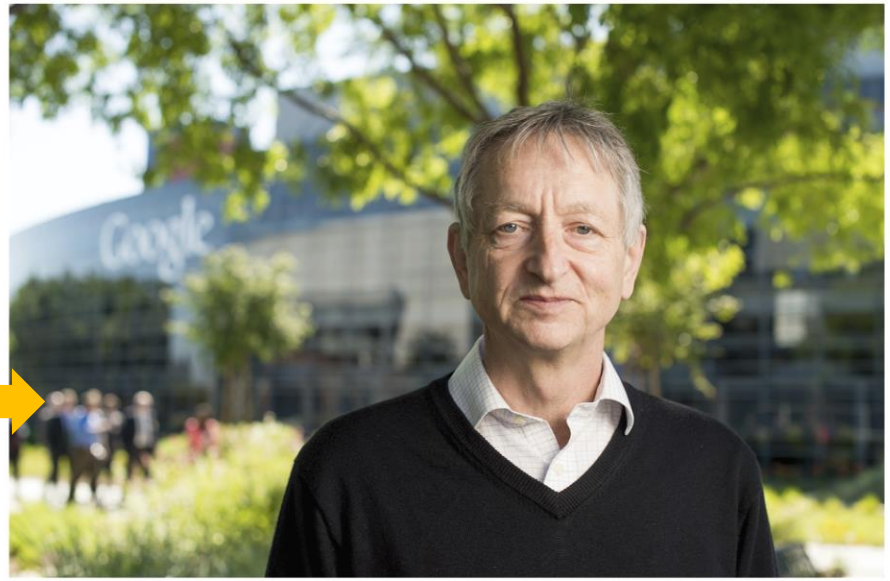
Deep Learning



Father of 'AI'

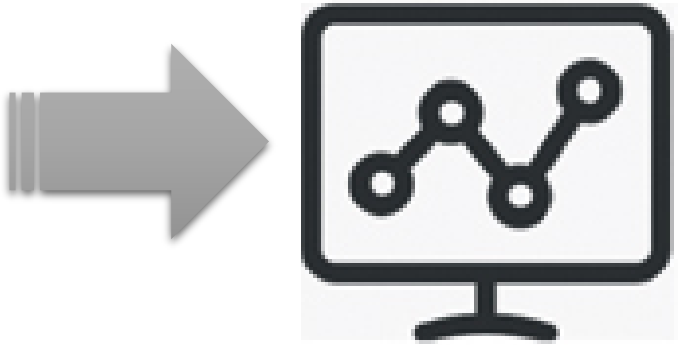
"Once your computer is pretending to be a neural net, you get it to be able to do a particular task by just showing it a whole lot of examples."

By Geoffrey Hinton
(EX. Google (Google Brain)
Prof. at University of Toronto)



Geoffrey Hinton designs machine learning algorithms. His aim is to discover a learning procedure that is efficient at finding complex structure in large, high-dimensional datasets and to show that this is how the brain learns to see.

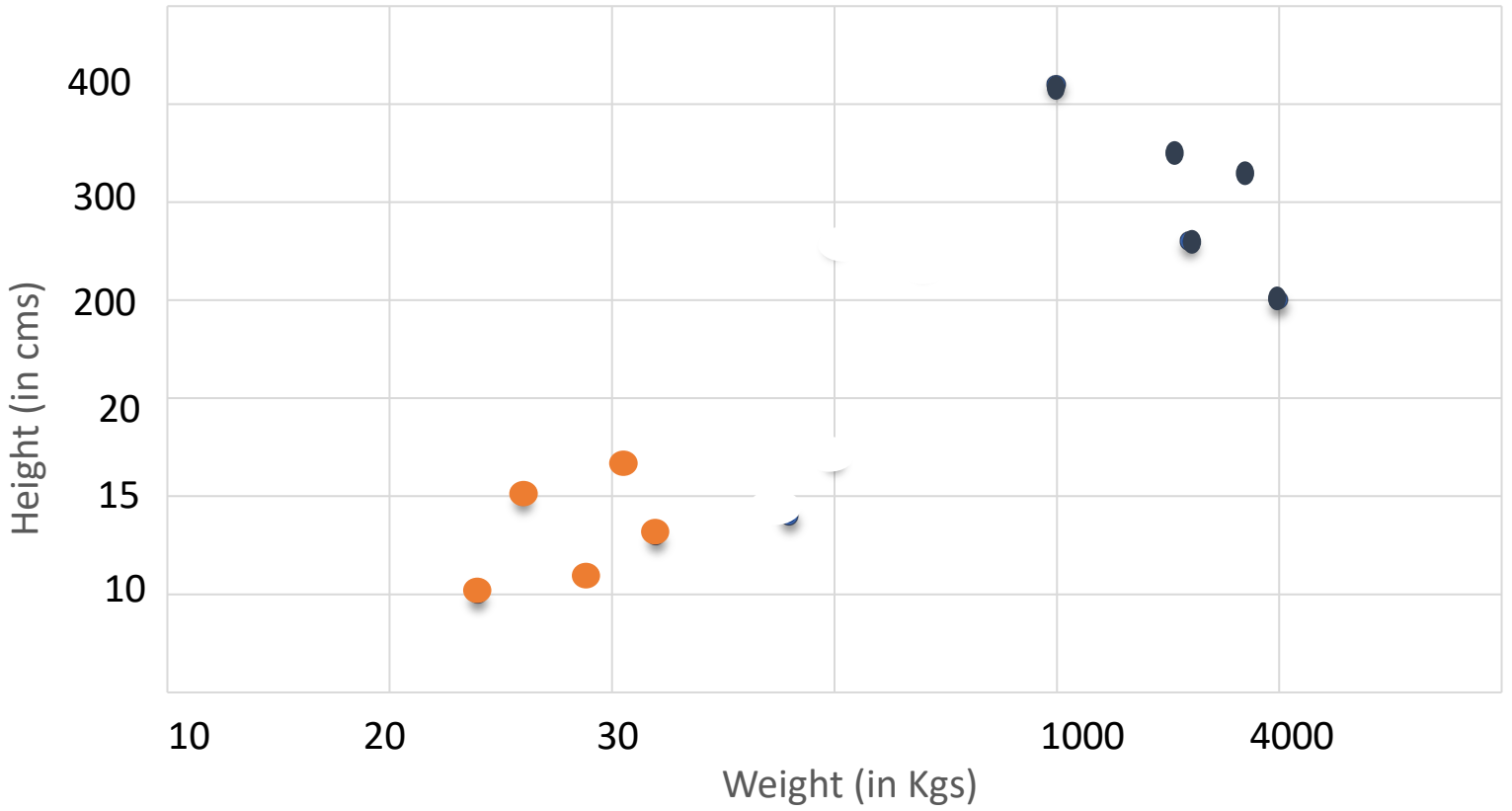
Make a Machine differentiate Elephants from Dogs?



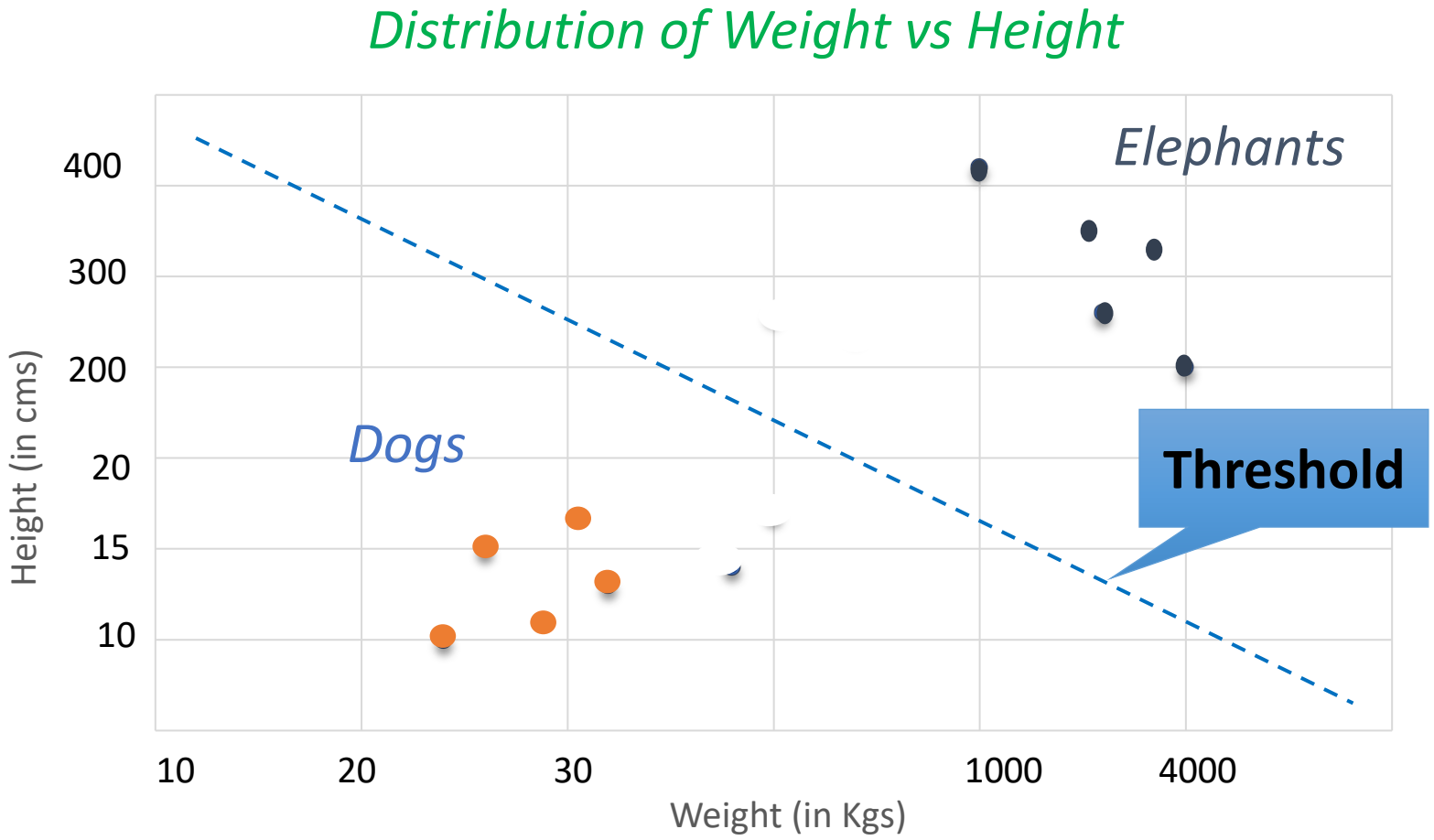
Machine

Elephant OR Dog?

Make a Machine differentiate Elephants from Dogs



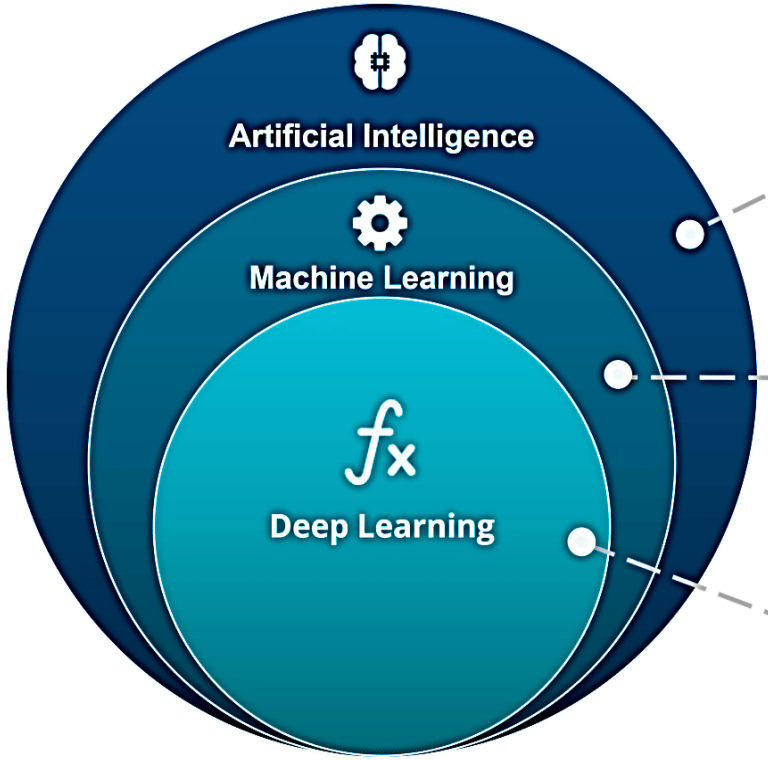
Make a Machine differentiate Elephants from Dogs: Simply Threshold



Differentiate 100 species of Moths with Simple Threshold?

Make Machine Intelligent !

AI vs. ML vs. DL



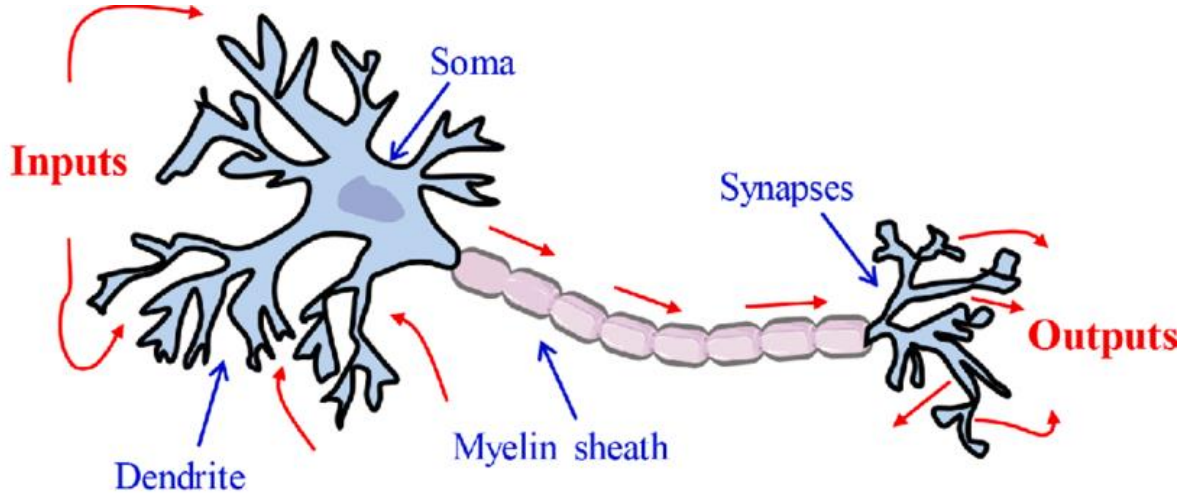
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

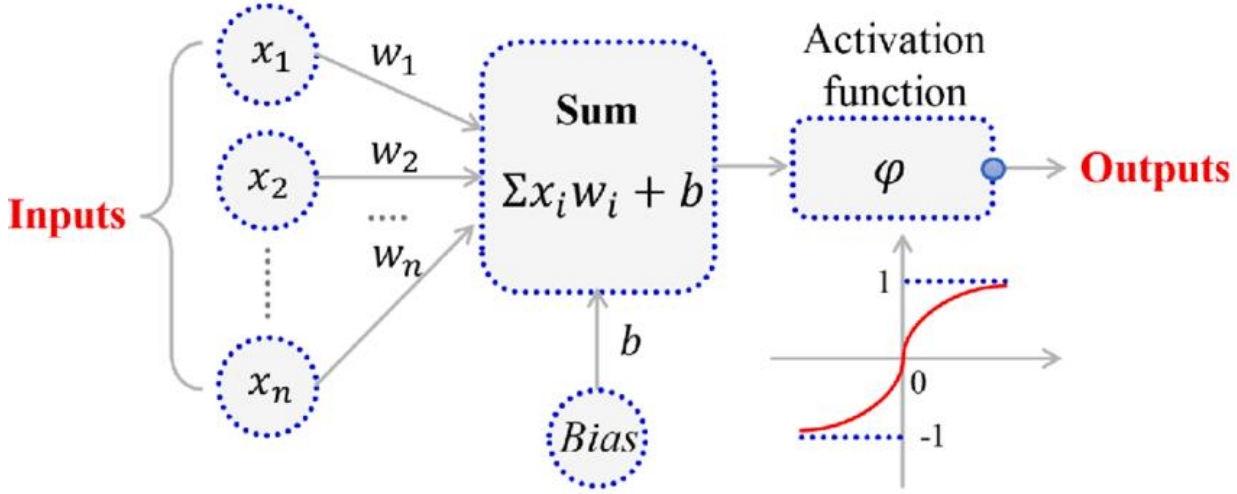
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Timeline

- 1950's
- 1960's
- 1970's
- 1980's
- 1990's
- 2000's
- 2010's

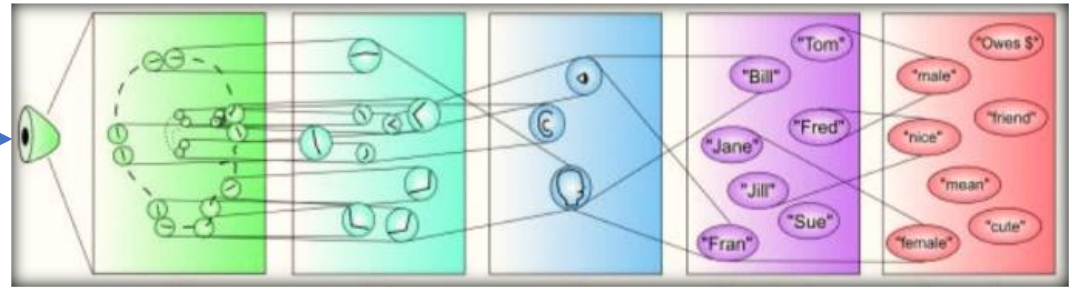


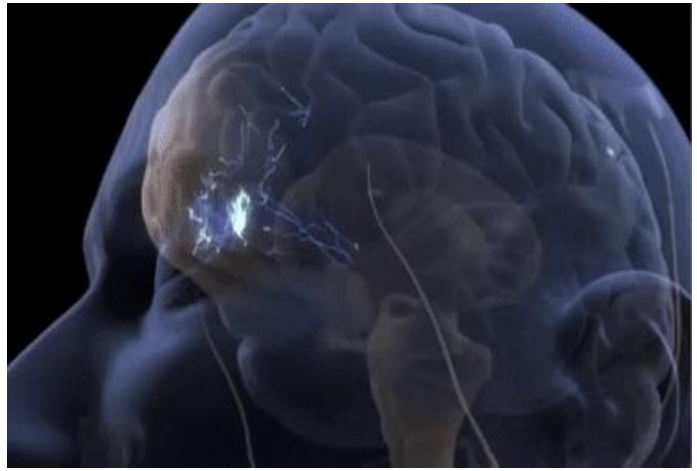
(a) Biological neuron

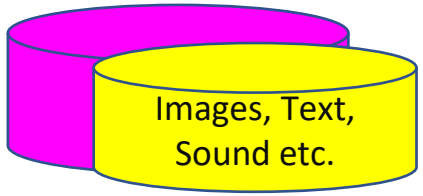


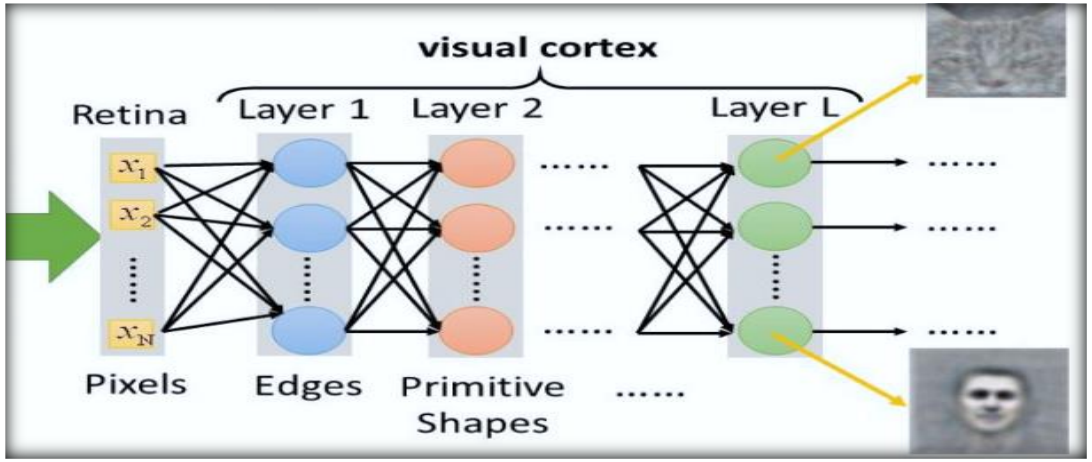
(b) Artificial neuron

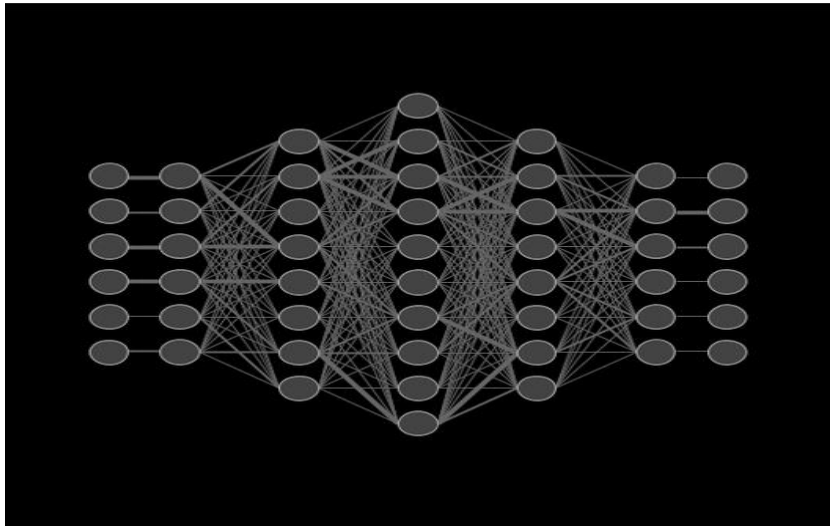
Visual Cortex and Deep Neural Network (DNN)



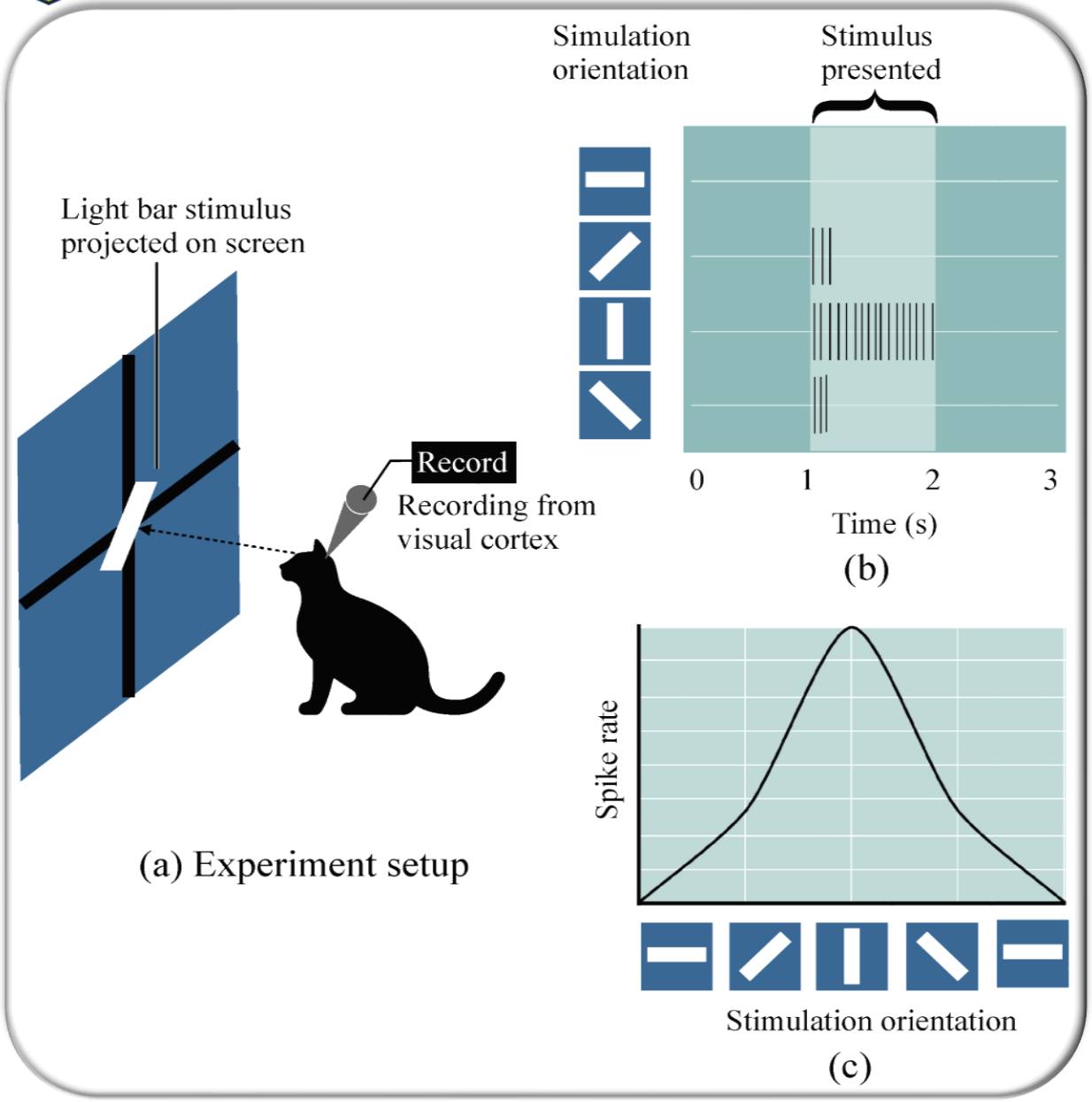


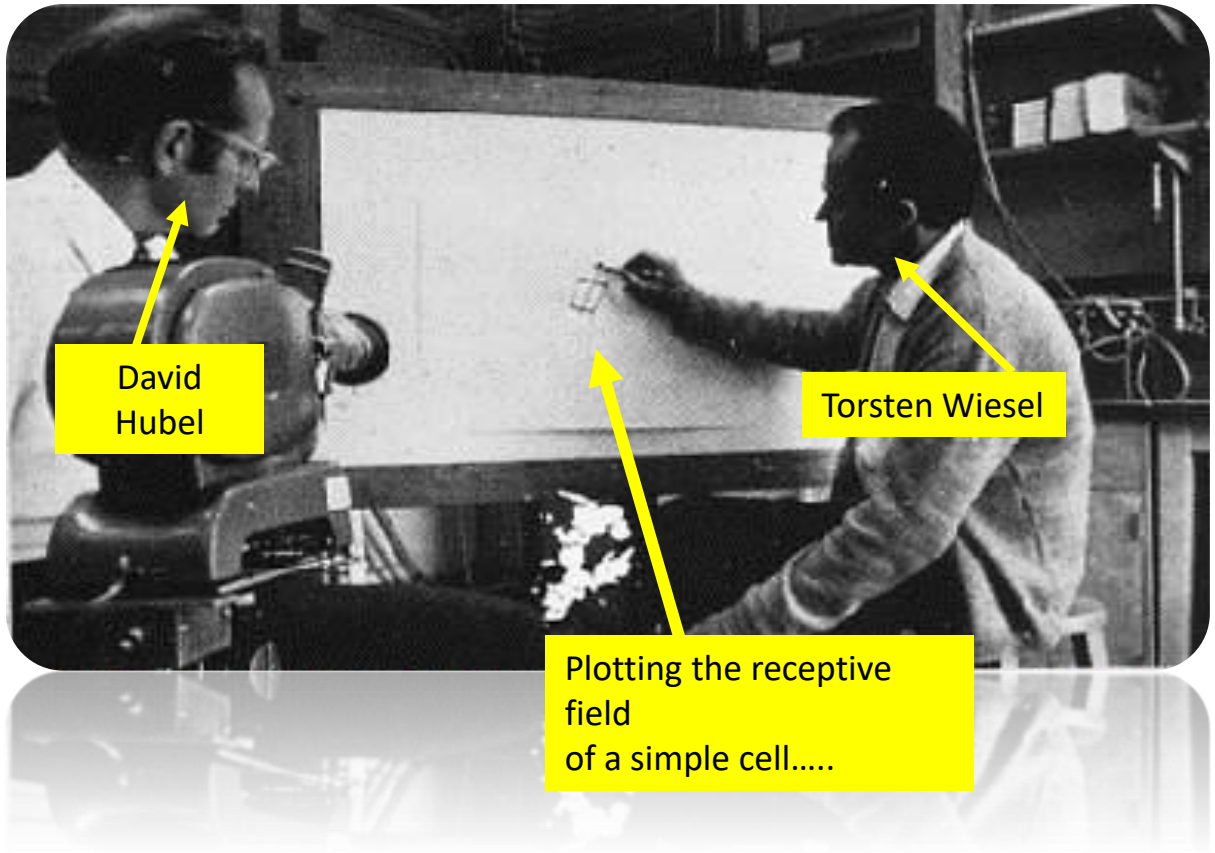




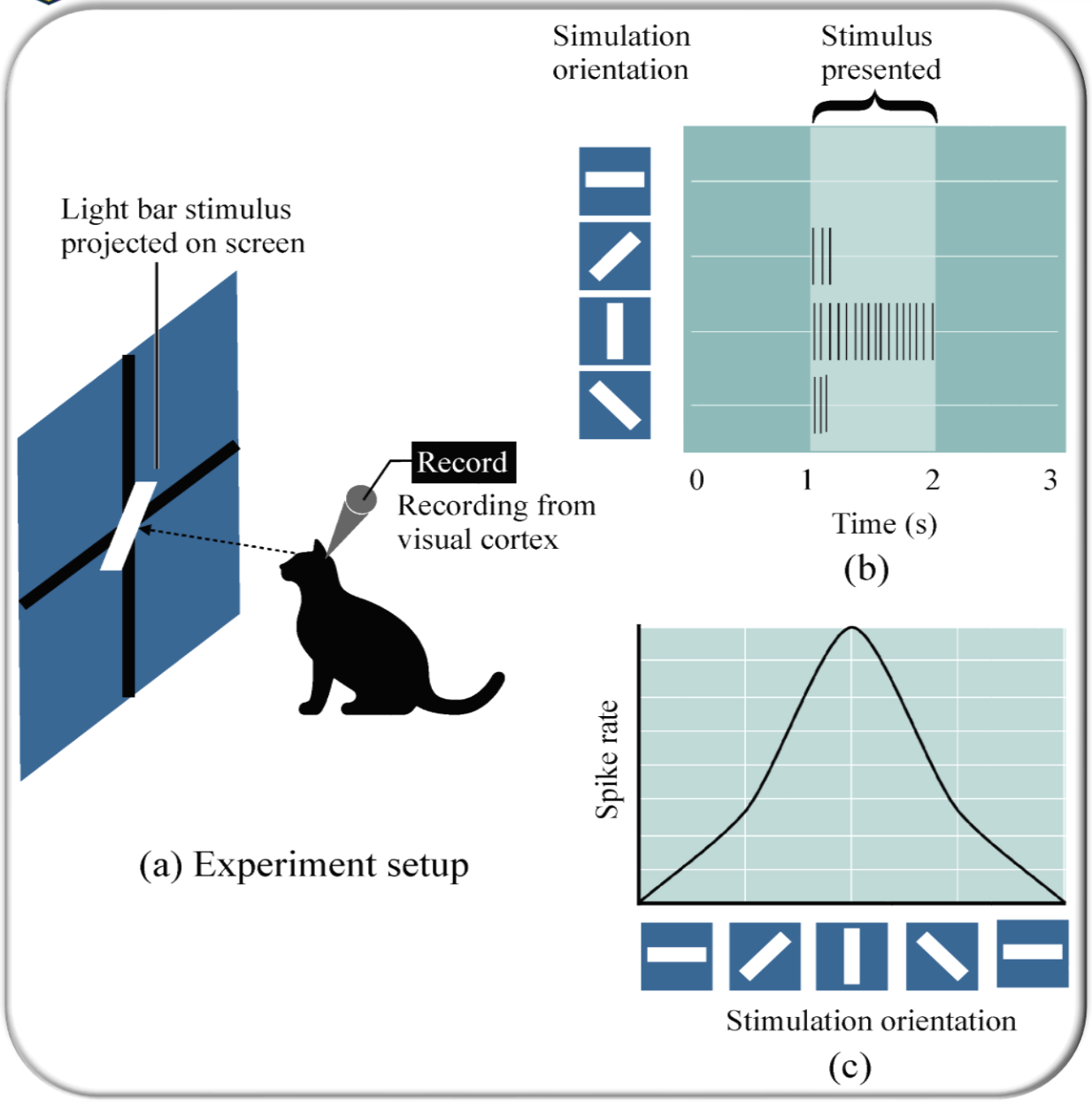


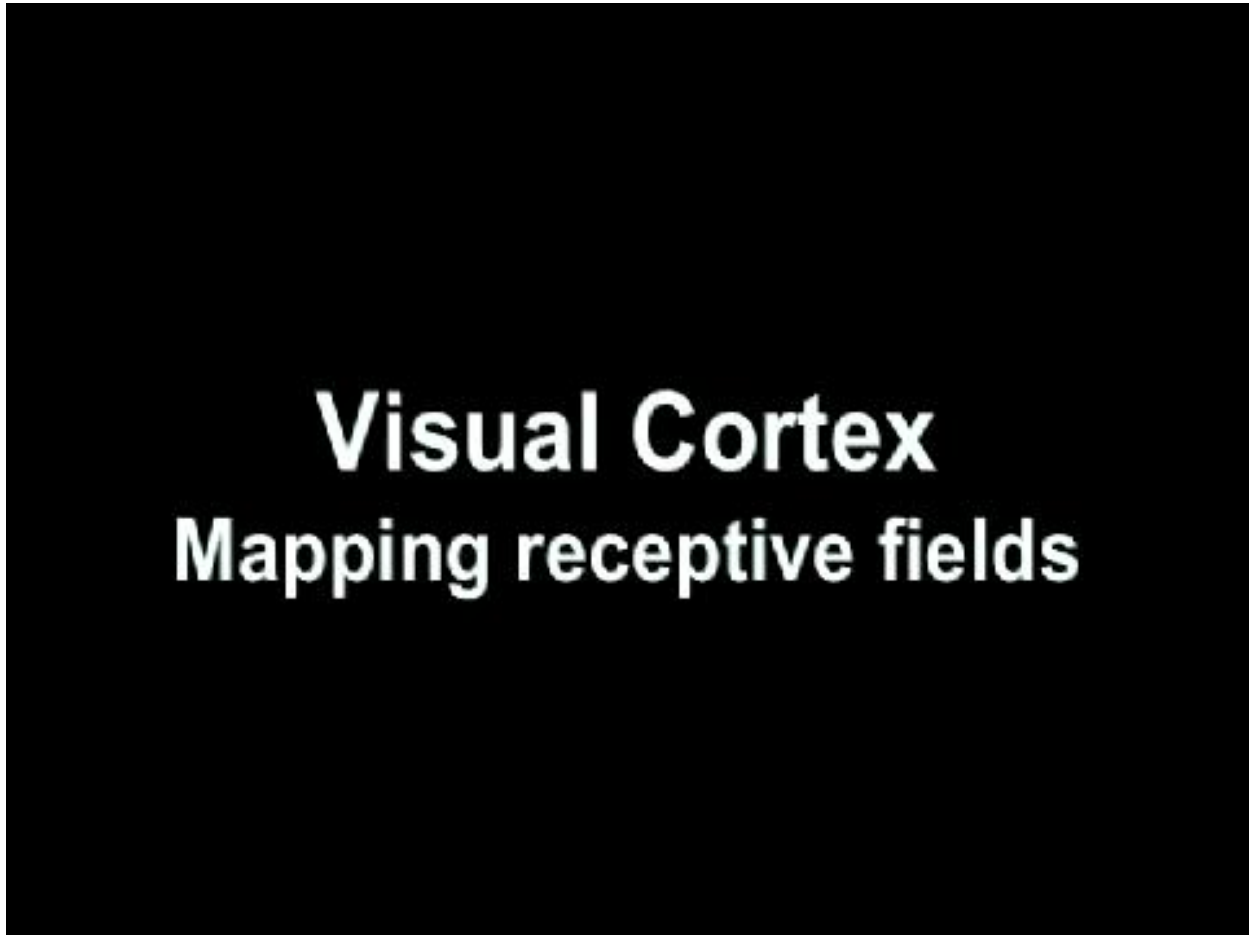
Hubel & Wiesel Experiment





Hubel & Wiesel Experiment





What is Machine Learning?

One of the Earliest definition of Machine Learning:

Arthur Samuel (1959):

"Field of study that gives computers the ability to learn without being explicitly programmed."

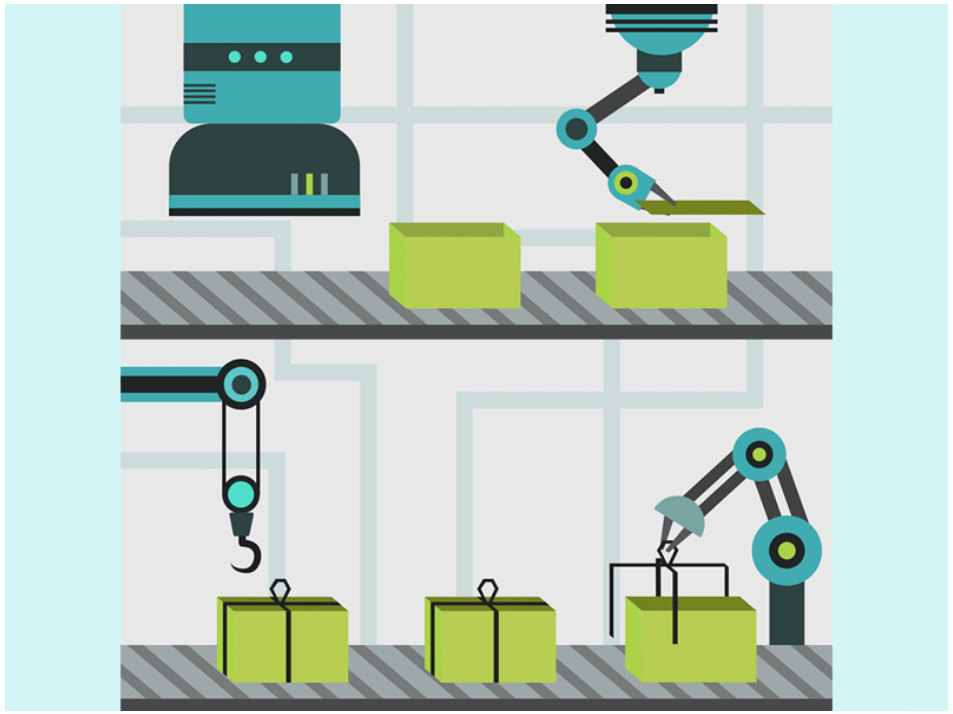
The Samuel Checkers-playing Program was among the world's first successful self-learning programs, and as such a very early demonstration of the fundamental concept of artificial intelligence.

***On February 24, 1956**, Arthur Samuel's Checkers program was developed for play on the IBM 701, In 1962, Self-proclaimed checkers master Robert Nealey played the game on an IBM 7094 computer. The computer won. It is still considered a milestone for artificial intelligence, as an example of the capabilities of an electronic computer.*

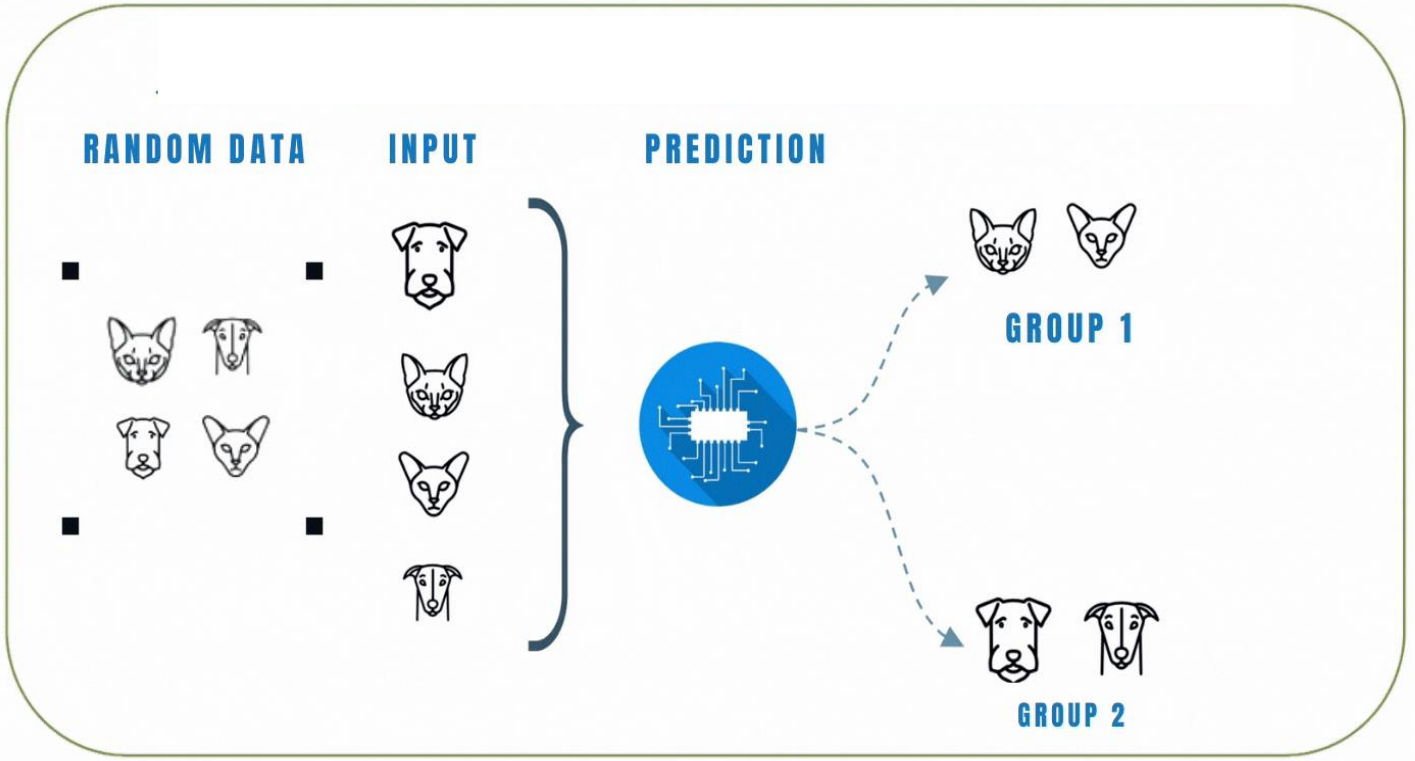
Courtesy: ibm.com

Machine Learning vs Process Automation

Process Automation



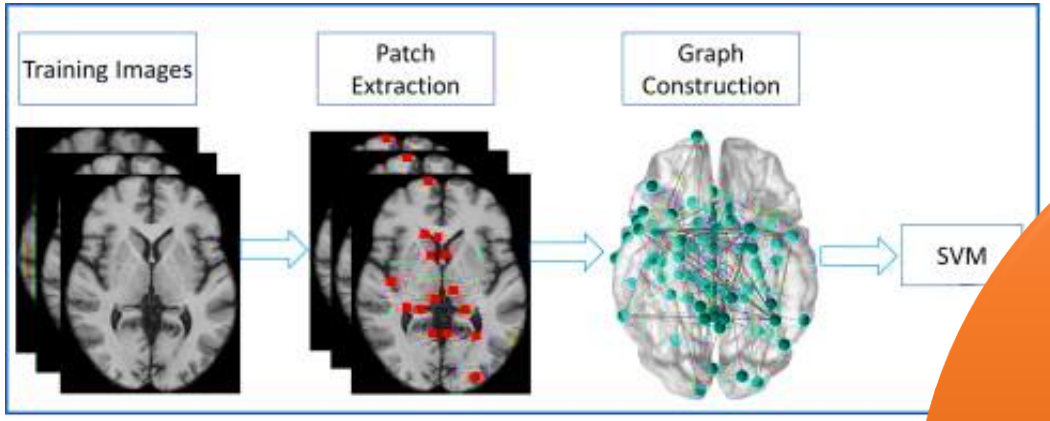
Machine Learning



Follow orders, Pre-programmed Rules to run a process, Monotonous, Repetitive process

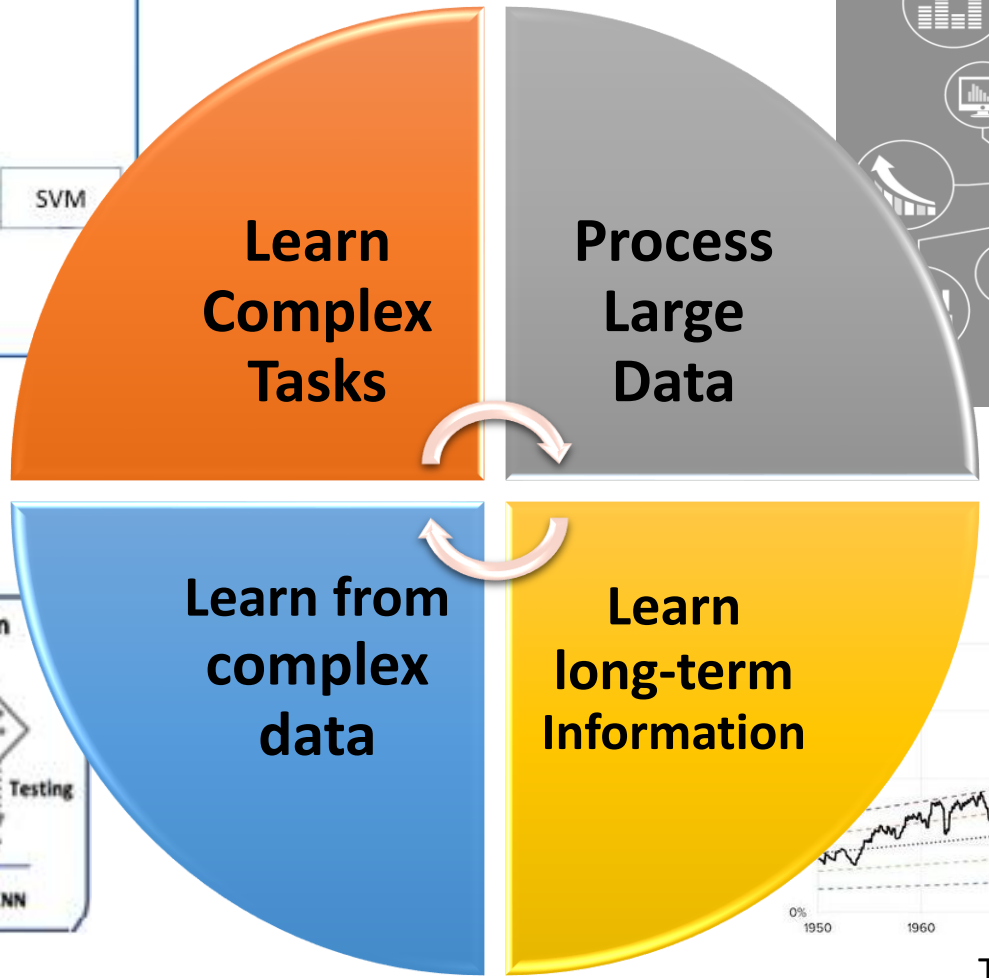
Mimic human-ability to think and do, Machine seeks patterns, adapts with experience

Machine Learning imparts abilities beyond human capability!

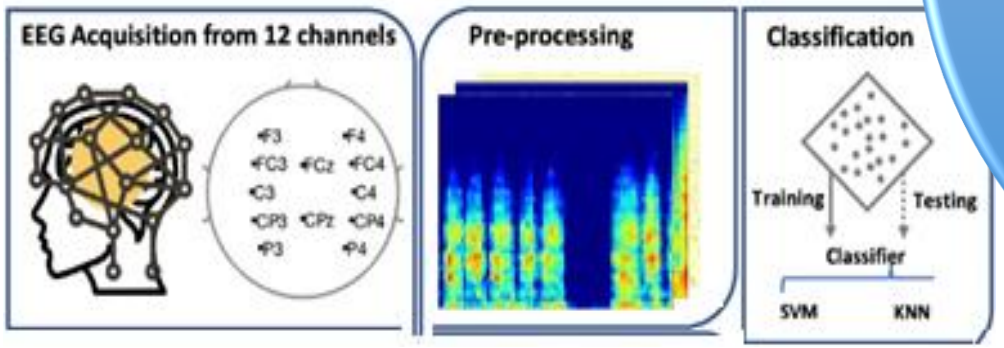


Automatic Disease Detection

https://doi.org/10.1016/j.media.2014.04.006



Process 1000's of Images Simultaneously



Auto-analyse EEG (brain) signals

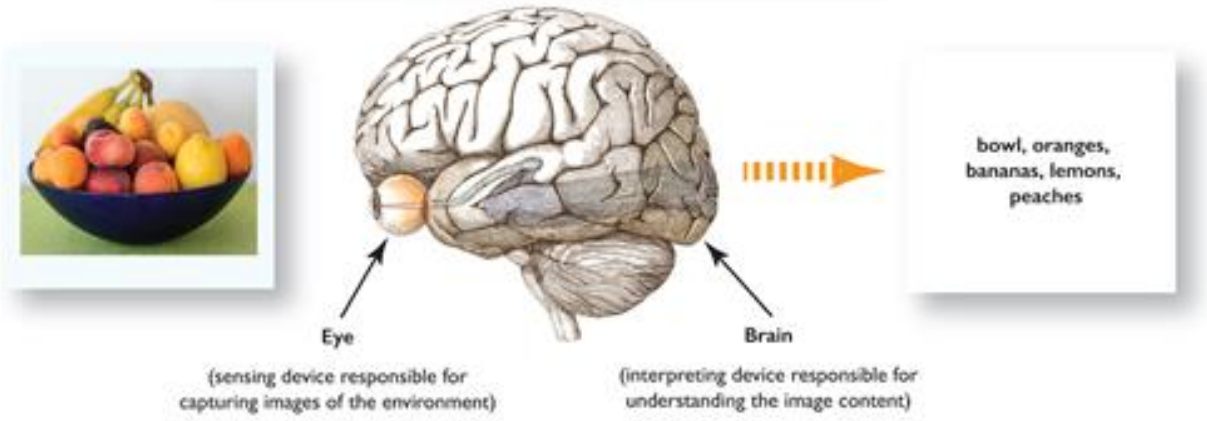
https://doi.org/10.1007/978-3-030-63823-8_6

Trend Analysis

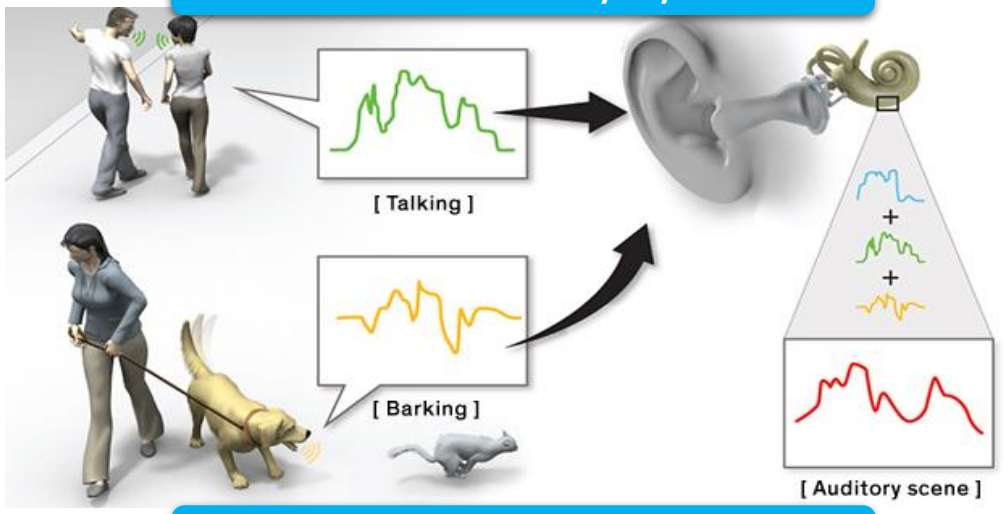
Human Perception vs. Machine Learning

Capture Data -> Process -> Interpret

Human Vision System

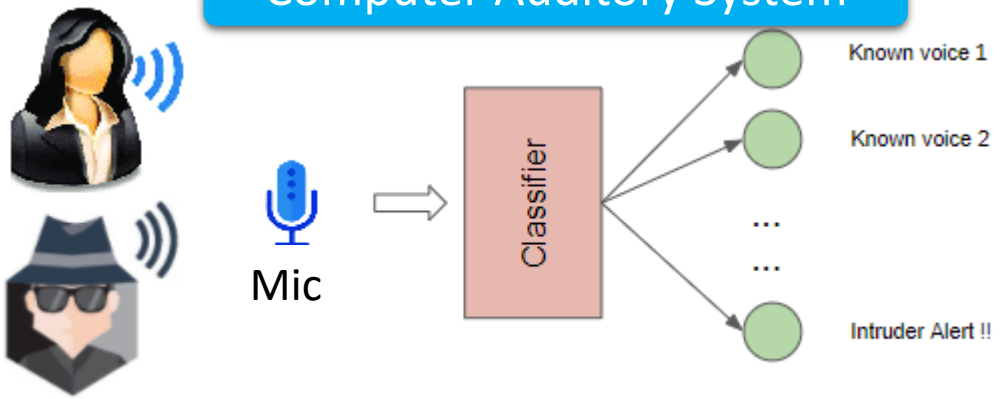


Human Auditory System



Computer Vision System

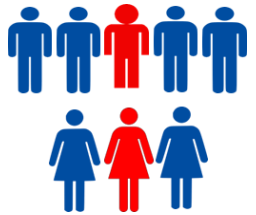
Computer Auditory System



Consider an Example of Computer Vision

Aim: Early Detection of Osteoporosis

- Osteoporosis is a chronic disease characterized by progressive bone loss and changes at microstructural levels resulting in increasing bone weakness and fragility.



Worldwide, 1 in 3 women and 1 in 5 men will experience osteoporotic fractures in their lifetime.

- Current Practice: Conventionally, Dual-Energy X-ray Absorptiometry (DEXA) is used, which is costly and is not available in small towns and villages. DEXA machine which is available only to 0.26/million population in India.

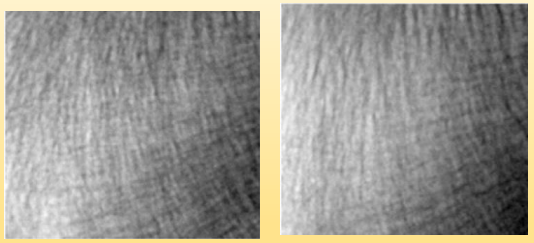
Question: Can we assess bone quality parameters on X-ray radiographs?

Answer: Yes ! Using Computer Vision on X-ray Images

Human Eye: Views Spatial Information in 2D

Computer Vision: Can go Beyond Human Vision!

Similar Visual Information
Computationally Different



Control Subject | Osteoporosis Patient

- Domain Transformations
- Computation Algorithms
- Computer Vision

Optimizations and Parameter Tuning for final Framework

Final Automatic Diagnostic tool based on the most optimized performance.

Classification of the Trabecular Bone Structure of Osteoporotic Patients using Machine Vision

Features Extraction

Table: Features (Principal components) and their corresponding *p-values* (after reduction using PCA)

Features	Healthy subjects (mean ± std. deviation)	Osteoporotic patients (mean ± std. deviation)	p- values
PC 1	(-4.02E-006) ± (6.47E-006)	(4.01E-006) ± (6.85E-006)	1.69E-006
PC 2	(-1.80E-006) ± (4.06E-006)	(1.80E-006) ± (2.83E-006)	2.24E-005
PC 3	(-2.62E-008) ± (1.10E-007)	(2.61E-008) ± (1.96E-007)	0.0013
PC 4	(-1.63E-012) ± (8.32E-011)	(1.63E-012) ± (9.52E-011)	0.8663
PC 5	(-4.79E-012) ± (1.64E-011)	(4.79E-012) ± (2.40E-011)	0.6834
PC 6	(7.84E-014) ± (2.78E-012)	(-7.82E-014) ± (2.63E-012)	0.8436

Table: Linear combinations of the original features that generate the principal components PC1 and PC2

Features		Principal Components (PCs)	
		PC1	PC2
Statistical Features	Mean	-0.5160	-0.4609
	Std. deviation	-0.3014	0.8533
Texture Features	Contrast	0.8017	0.0268
	Correlation	0.0052	-0.1397
	Energy	0.0052	-0.1397
	Homogeneity	0.0052	-0.1397

Classification Methods

Supervised Machine Learning

The discriminatory feature vectors (first and second PCs) were used to separate healthy X-ray images from osteoporotic ones using different classifiers. The four most popular classification algorithms, namely **Support Vector Machine** (SVM), **Naive Bayes classifier** , **Artificial Neural Network** (ANN) and ***k*-Nearest Neighbors** (*k*-NN) classifier, were applied to the first two principal components.

Bone X-ray image database considered in this study

Bone X-ray image	#Samples for training	#Samples for testing	Total samples
Healthy Subjects	40	47	87
Osteoporotic Patients	40	47	87
Total	80	94	174

Performance of the classifiers with feature processing

Classifiers	Sensitivity (%)	Specificity (%)	Accuracy (%)
SVM	100	95.74	97.87
Naive Bayes	97.87	93.61	95.74
k-NN	95.74	97.87	96.80
ANN	97.87	95.74	96.80

Computers in Biology and Medicine 91 (2017) 148–158

Contents lists available at ScienceDirect

Computers in Biology and Medicine

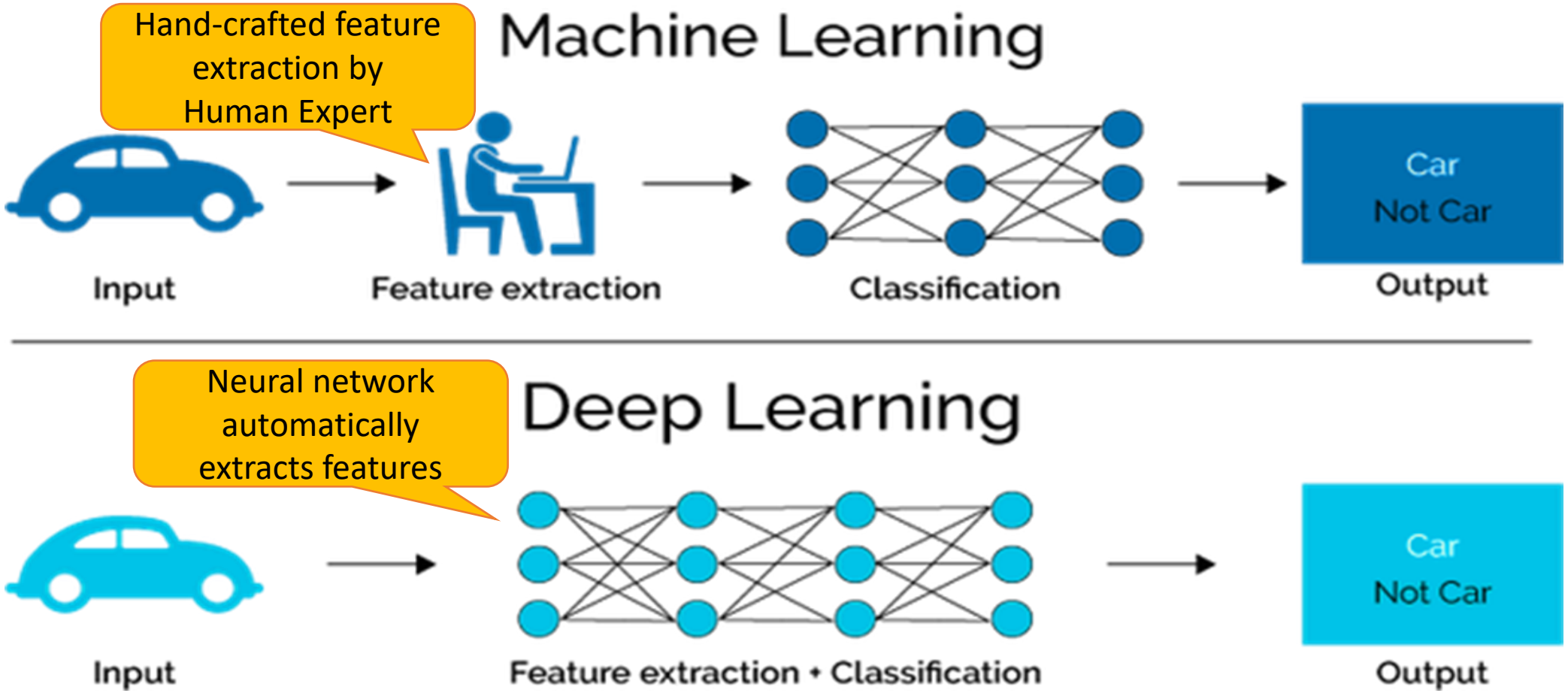
journal homepage: www.elsevier.com/locate/compbiomed

Classification of the trabecular bone structure of osteoporotic patients using machine vision

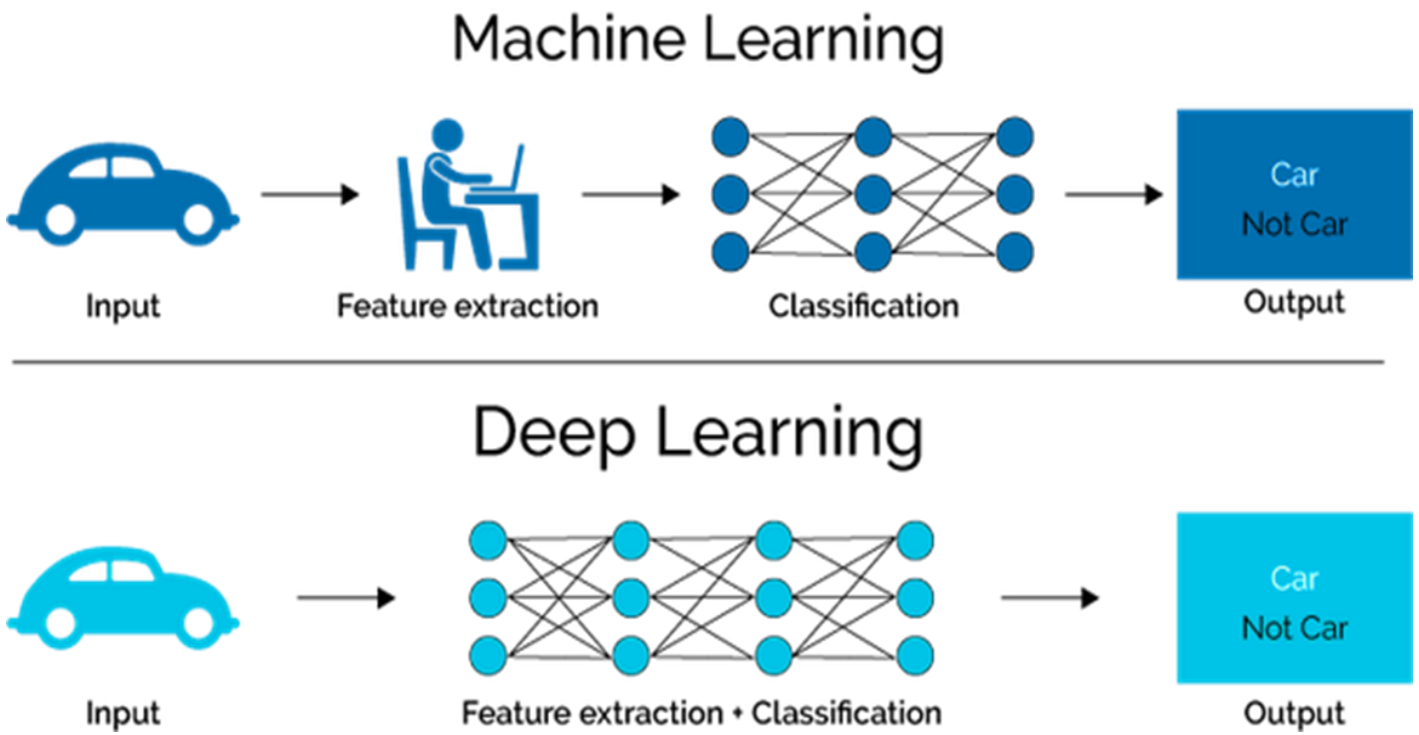
Anushikha Singh [a], Malay Kishore Dutta [a,*], Rachid Jennane [b], Eric Lespessailles [b,c]

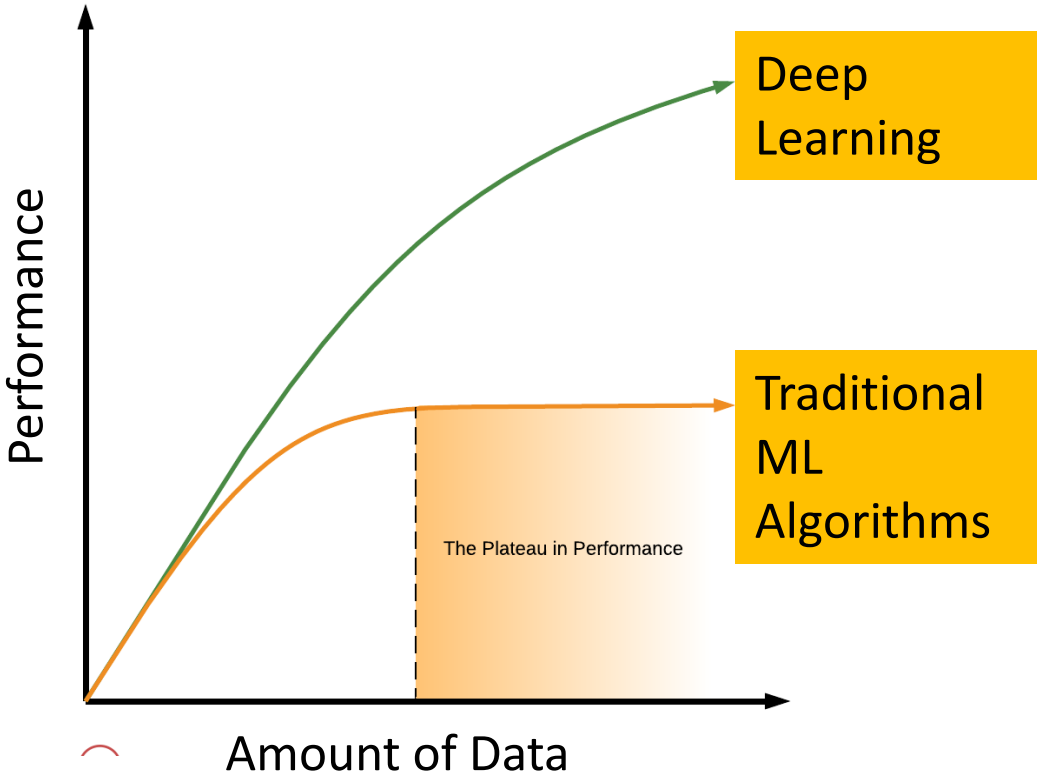
[a] Department of Electronics & Communication Engineering, Amity University, Noida, Uttar Pradesh, India
[b] Univ. Orléans, I3MTO Laboratory, EA 4708, 45067 Orléans, France
[c] Hospital of Orleans, 45067 Orléans, France

ML vs. DL


Hand-crafted feature extraction by Human Expert
Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Neural network automatically extracts features
Deep Learning
Input
Feature extraction • Classification
Car
Not Car
Output

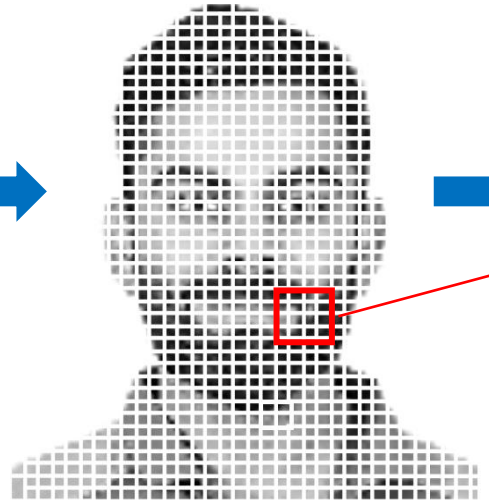
ML vs. DL


Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction • Classification
Car
Not Car
Output


Performance
Amount of Data
Deep Learning
Traditional ML Algorithms
The Plateau in Performance

What do you see?

How computer see the image?



255	200	211	235	1
200	161	217	233	0
218	65	214	237	0
232	29	217	236	1
234	23	216	240	0
102	31	217	234	0

255	200	211	235	1
200	161	217	233	0
218	65	214	237	0
232	29	217	236	0
234	23	216	240	0
102	31	217	234	0

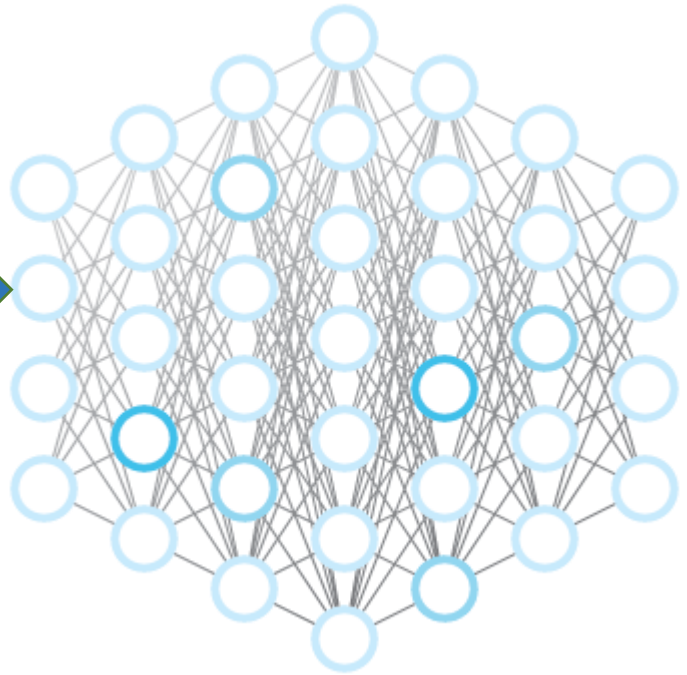
Computer Interpretation

- For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255.
- Similarly for color images, each level is represented by the range of decimal numbers from 0 to 255 (256 levels for each color), equivalent to the range of binary numbers from 00000000 to 11111111, or hexadecimal 00 to FF. The total number of available colors is 256 x 256 x 256, or 16,777,216 possible color.

Input Image

255	65	7	55	6	55	6	5	44	98	134	23	21	142	44	216	216	216
218	65	12	33	255	200	211	235	76	33	66	33	65	33	76	217	217	217
232	45	34	66	200	161	217	233	34	66	44	66	45	66	34	33	33	33
234	76	22	44	218	33	214	237	56	44	66	44	76	44	56	66	66	216
65	34	2	32	232	66	217	236	23	32	32	32	34	32	23	44	44	217
45	56	3	64	234	44	216	240	64	64	64	64	56	64	76	216	216	56
76	23	2	3	65	32	217	234	3	3	3	3	23	3	34	217	217	23
218	45	65	2	45	64	33	33	2	2	2	2	45	2	56	33	33	45
232	78	2	2	76	3	66	66	2	2	65	2	78	2	23	66	66	78
234	22	65	0	34	2	44	44	0	0	45	0	22	0	45	44	44	22

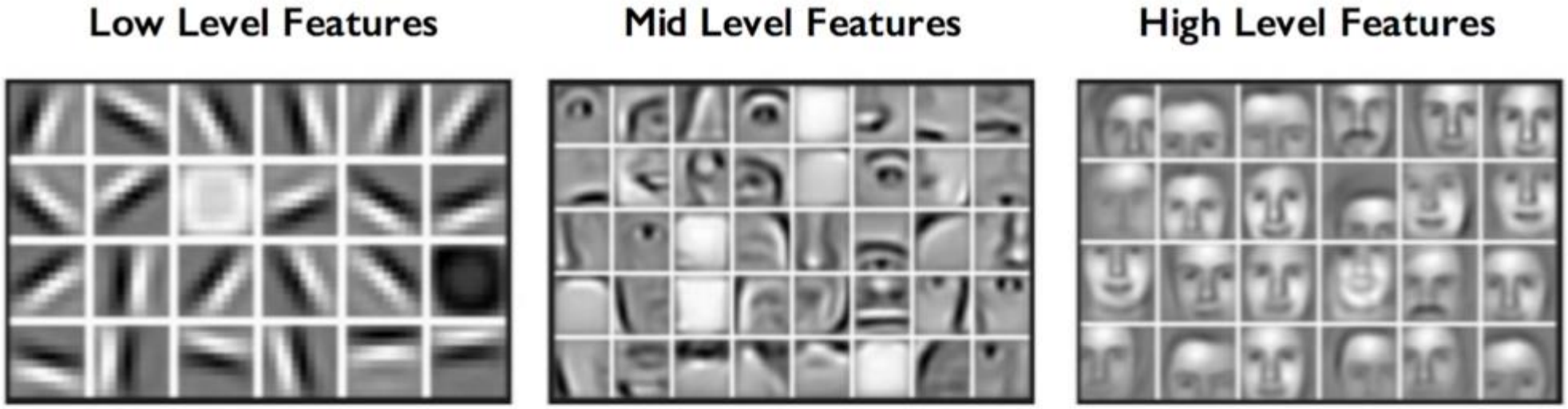
Pixel Representation



Classification Model

Virat Kohli 86%
Rohit Sharma 7%
M.S.Dhoni 5.8%
Sachin Tendulkar 1.2%

Classification



Input---**Shallow Layers**---------**Middle Layers**-------**Deeper Layers** ----> Output

MIT: Alexander Amini, 2018 introtodeeplearning.com

Manual Feature Extraction: Issues

Occlusion

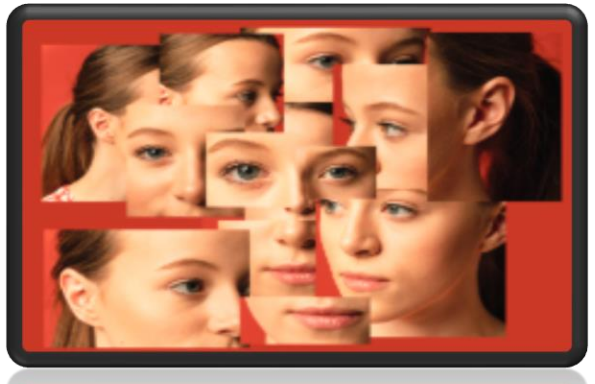
Cluttered Background

Intra-class Diversity

Object Deformation

Scale Variation

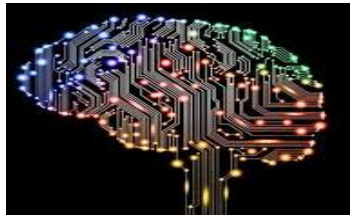
Illumination Variation



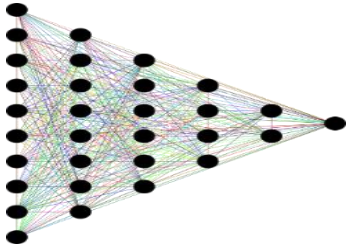
Change in Viewing Angles

Image Deformity

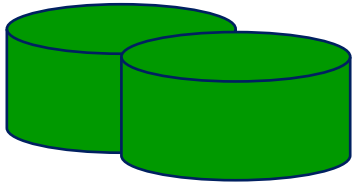
What is Deep Learning?



- Deep learning is a type of machine learning that mimics the neuron of the neural networks present in the human brain.

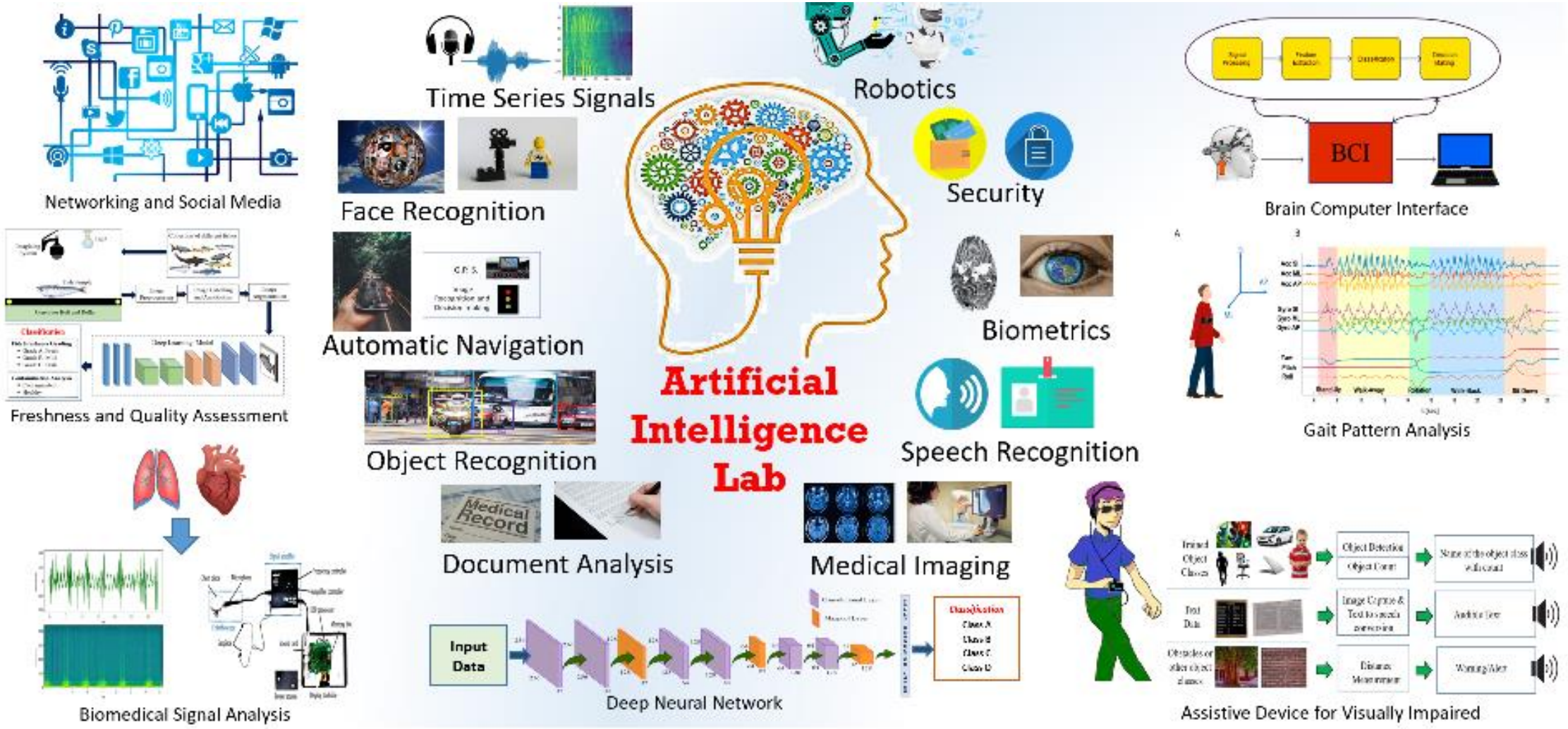


- Deep learning algorithms learn progressively about the input data as it goes through each neural network layer.

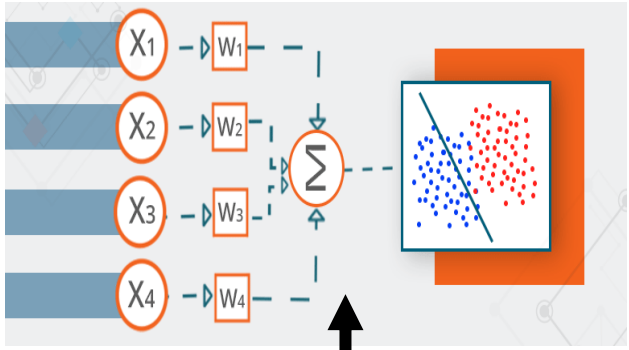


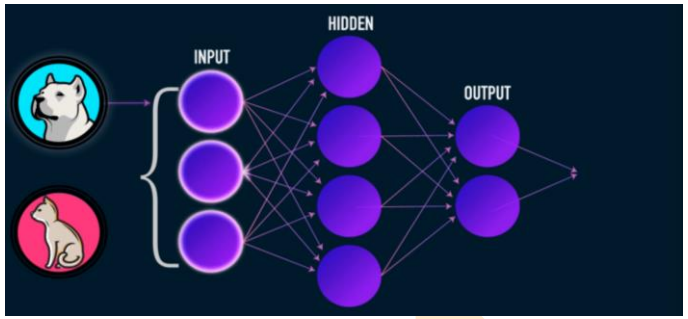
- If the system provided with tons of information, it begins to understand it and respond in useful ways.

Applications of Deep learning



Deep Learning: Historical Overview





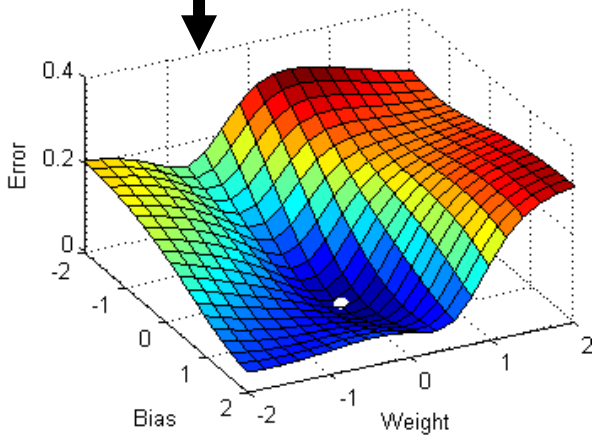
1952
Stochastic Gradient Descent

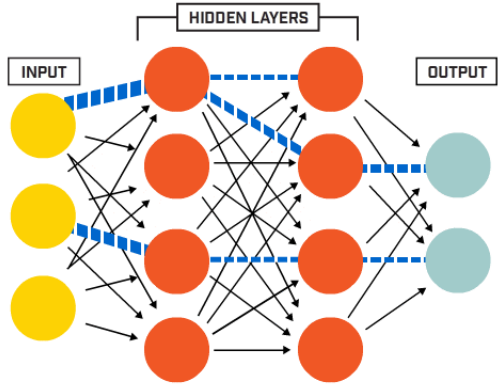
1958
Perceptron: Learning weights

1974
Backpropagation

1986
Backpropagation : Multi layer Perceptron

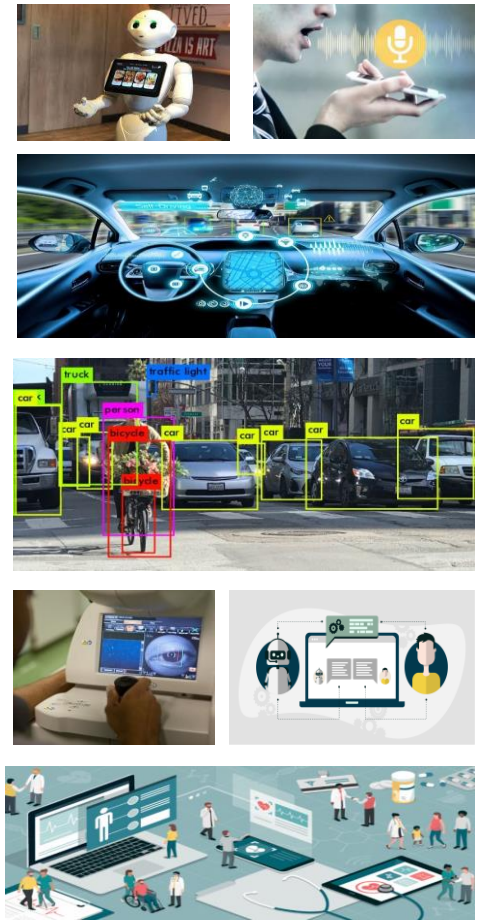
1998
Convolutional Neural Network





Why using Deep Learning Now?

Neural Network date back to 1950s, so why the resurgence?



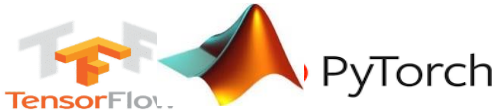
Why using Deep Learning Now?

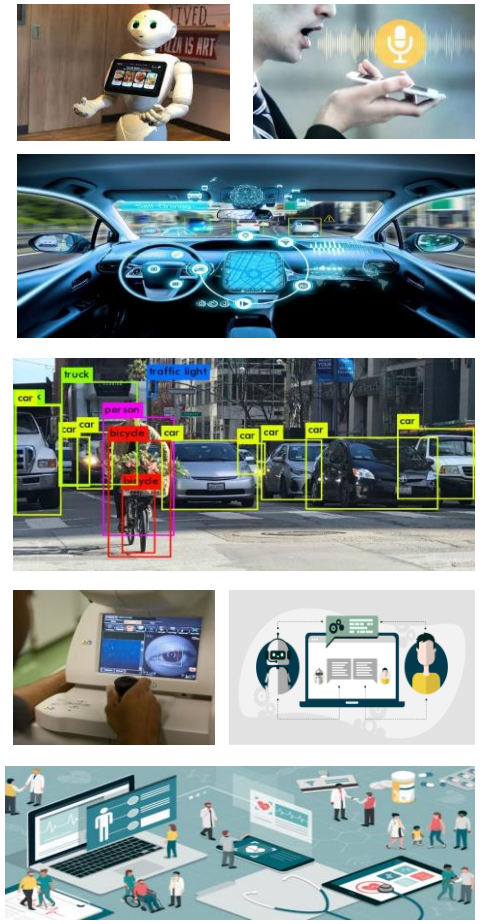
Neural Network date back to 1950s, so why the resurgence?

1. **Big Data**
   - Larger Datasets
   - Easier Collection and storage

2. **Hardware**
   - Graphics Processing Units (GPUs)
   - Massively Parallelizable

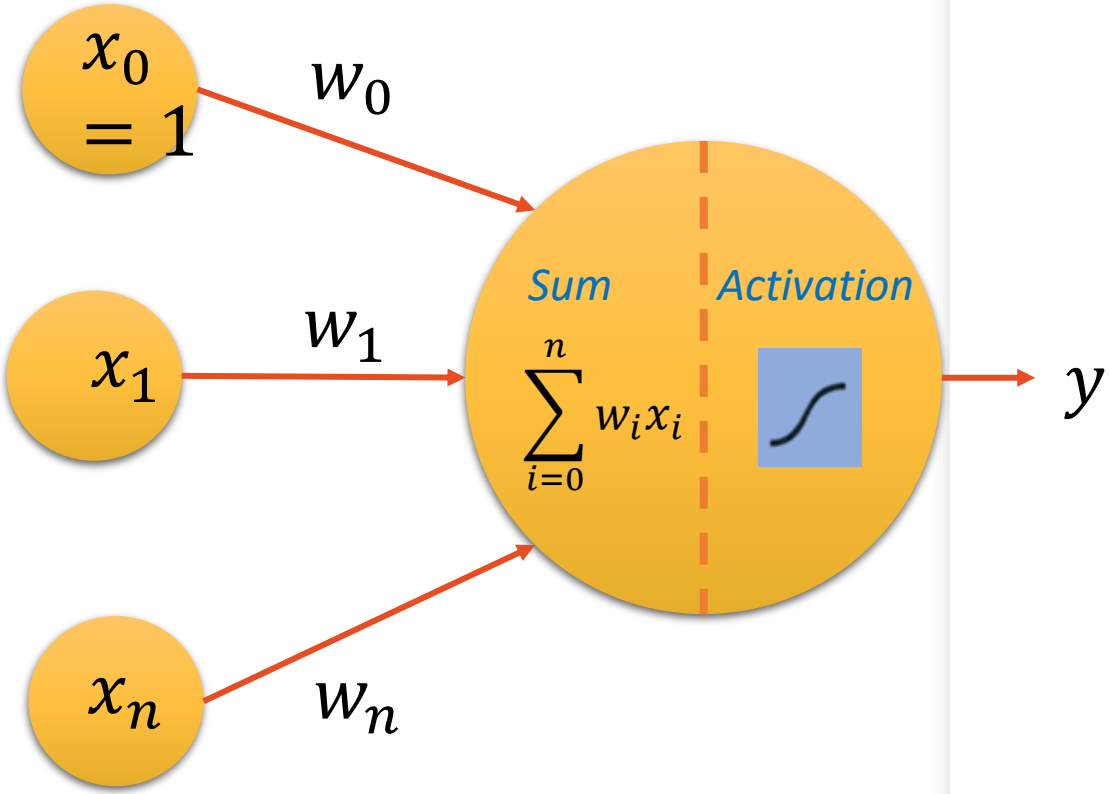
3. **Software**
   - Improved Techniques
   - New Models
   - Toolboxes





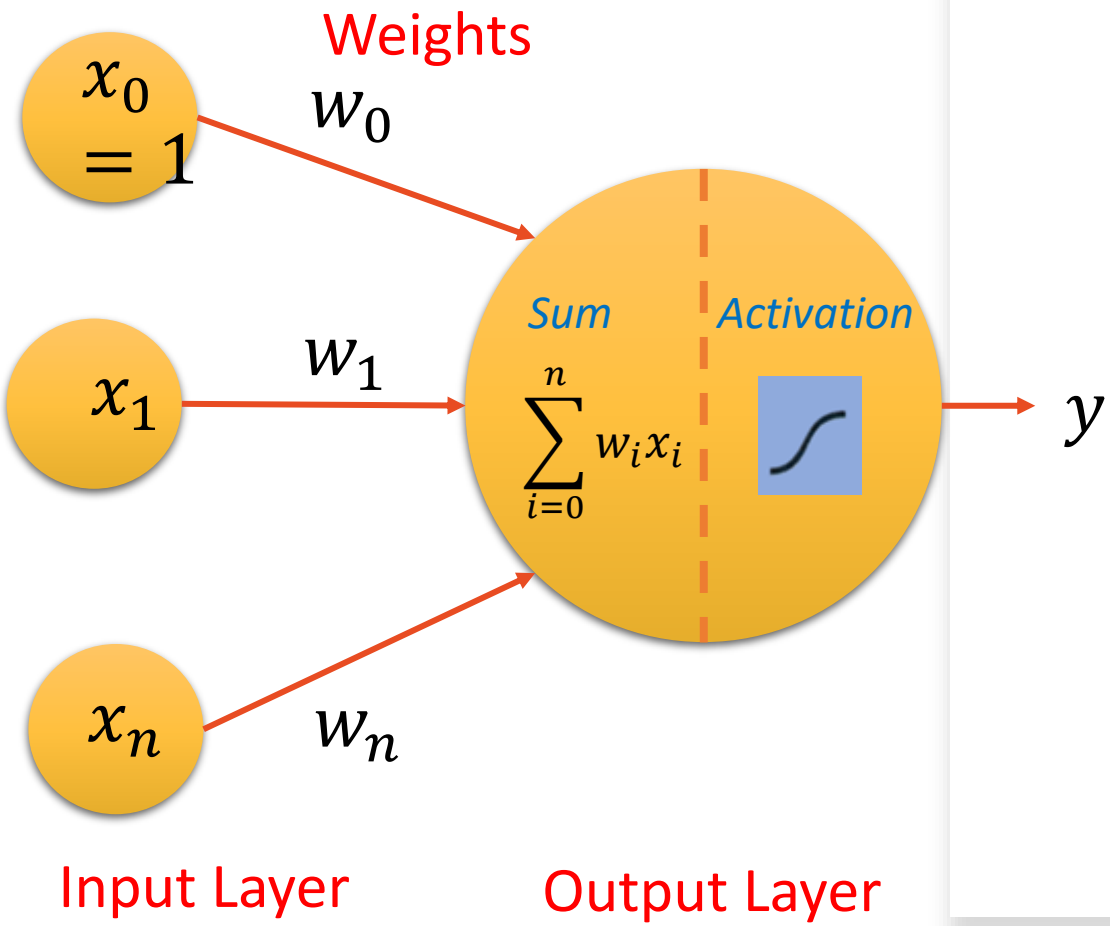
Perceptron

Perceptron



- Perceptron is a *building block* of an Artificial Neural Network
- Neural Network comprising of a *Single Neuron*

Perceptron



- Perceptron is a *building block* of an Artificial Neural Network
- Neural Network comprising of a *Single Neuron*

Input Nodes or Input Layer: Accepts the initial data into the system

Weights: Represents the strength of the connection between units and decide output based on the strength of associated input neuron

Output Nodes or Output Layer: Process inputs and generates output

How Perceptron works?

2 Step Process

Step1: Multiply all input values with corresponding weights & add to determine **weighted sum**. Mathematically, expressed as:

$$\sum_{i=0}^{n} w_i x_i = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

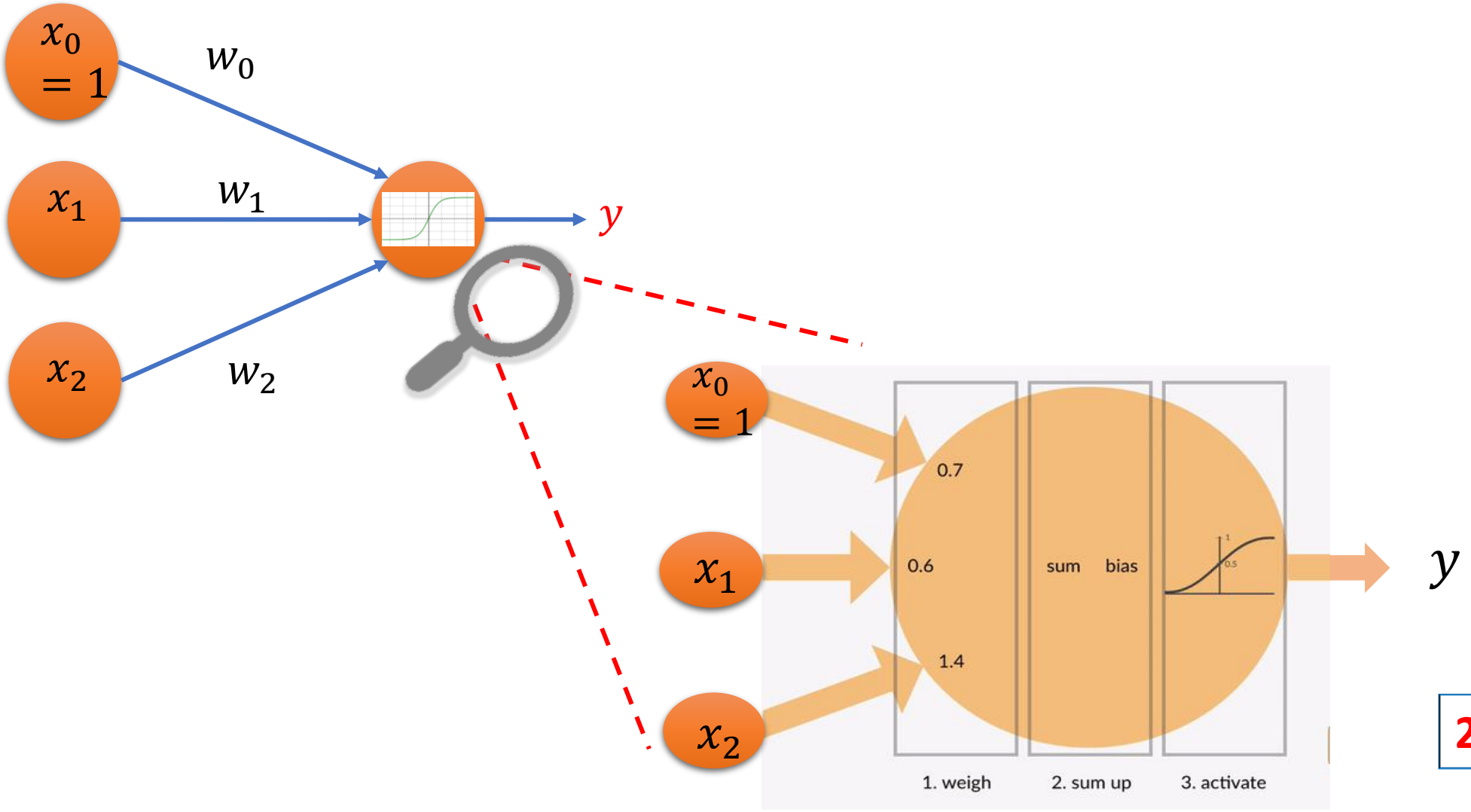
$x_0 = 1$ to introduce a bias term w_0

Step 2: Activation function is applied with weighted sum, which gives us output either in binary form or a continuous value

Activation function decides if the neuron will fire

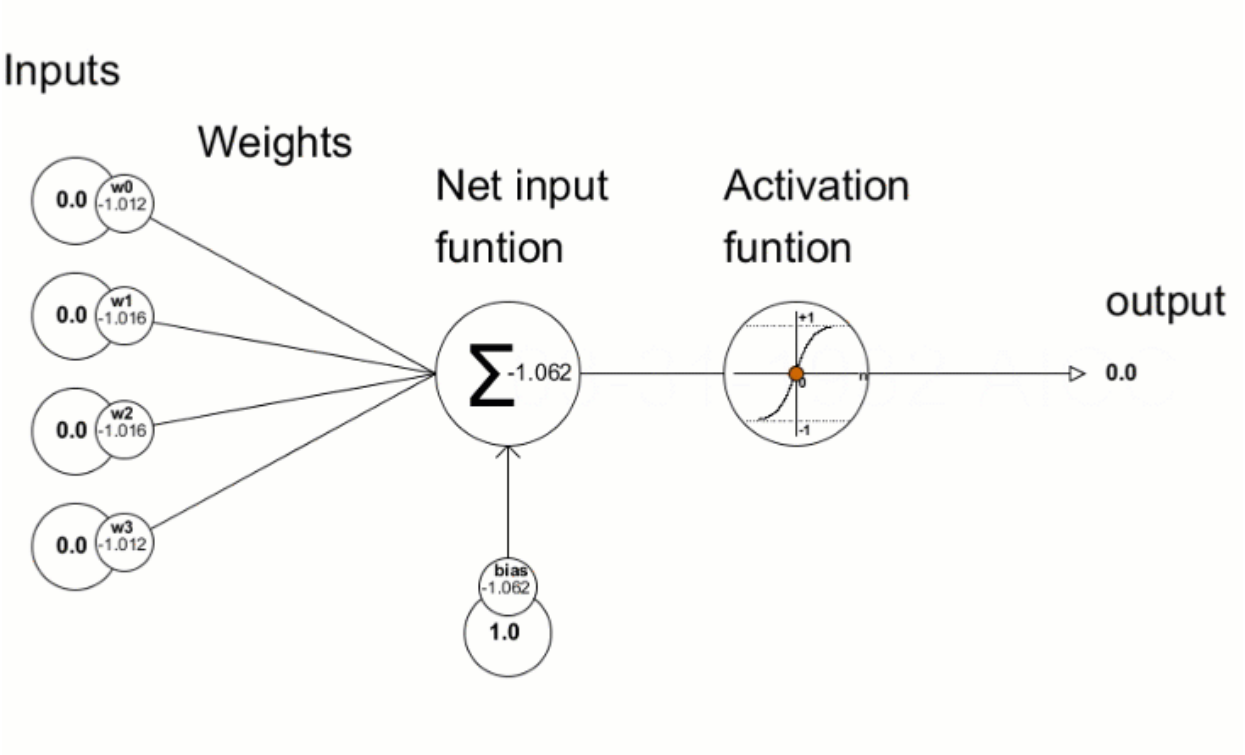
$$y = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

How Perceptron works?



2 Step Process

Activation function

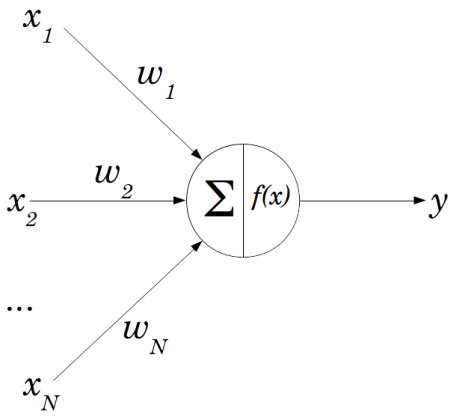


$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

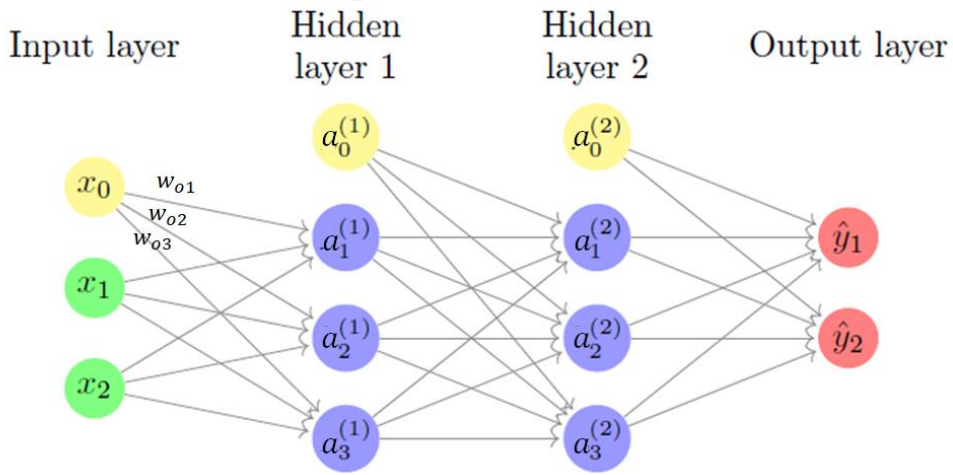
- Decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations

Source: https://medium.com/@MrBam44/activation-functions-in-deep-learning-models-how-to-choose-3ad007eaf998

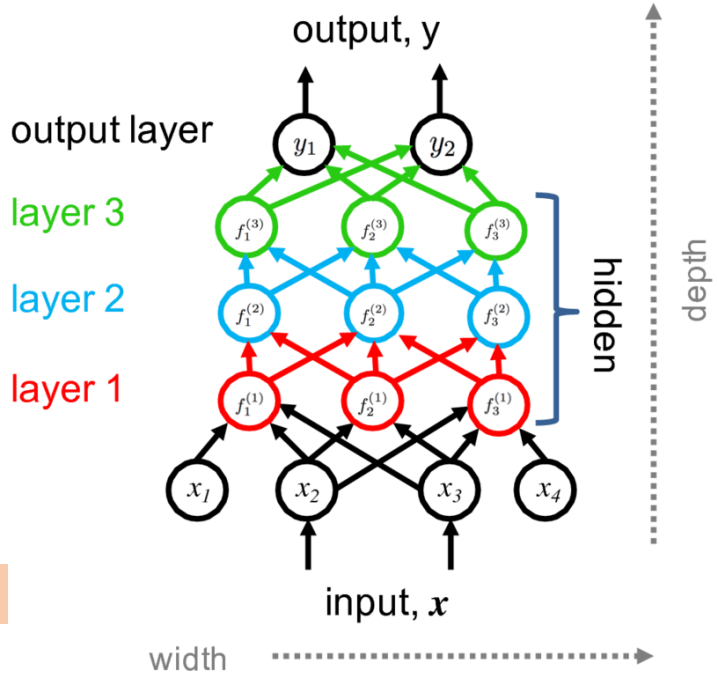
Neural Network



Single Perceptron



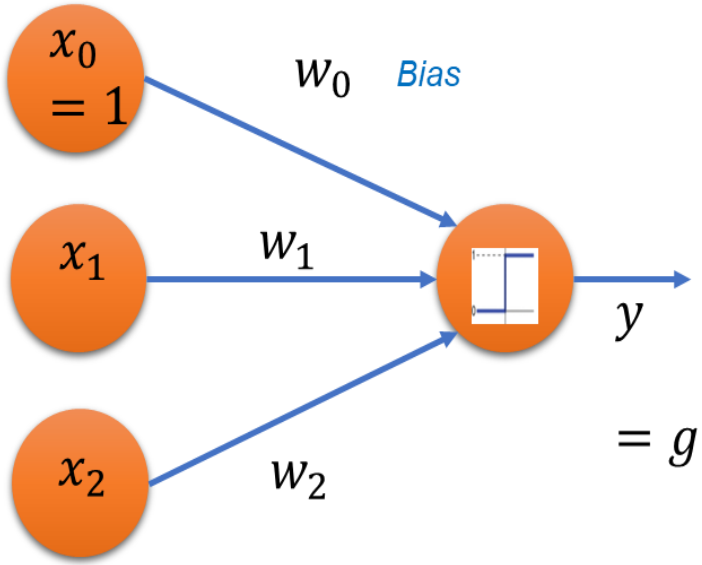
Multi Layer Perceptron (MLP)



- ANN has Nodes or Units connected by directed links
- The computational units are connected to one another through weights

- Weights ≡ strengths of synaptic connections in biological organisms
- Learning occurs by changing the weights connecting the neurons

Single Layer



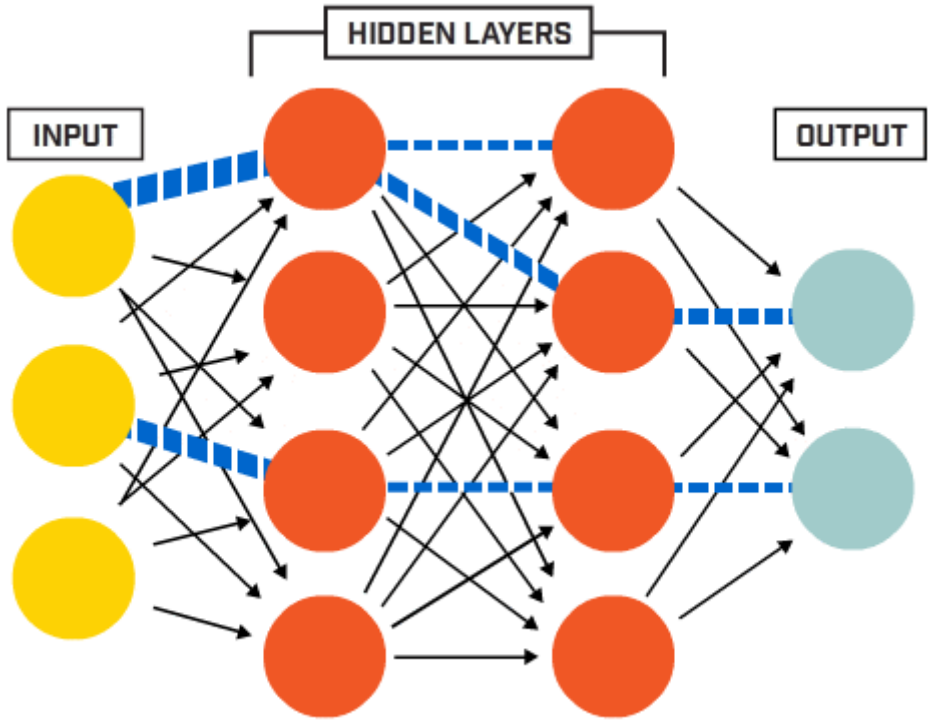
- NN comprising of a Single Neuron
- Single Layer Neural Network
- An artificial neuron using the Heaviside step function as the activation function

Thresholding function

$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

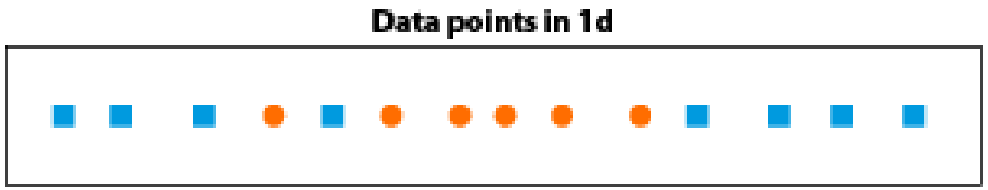
- Single-layer perceptron can learn only linearly separable patterns

Multi Layer Perceptron (MLP)



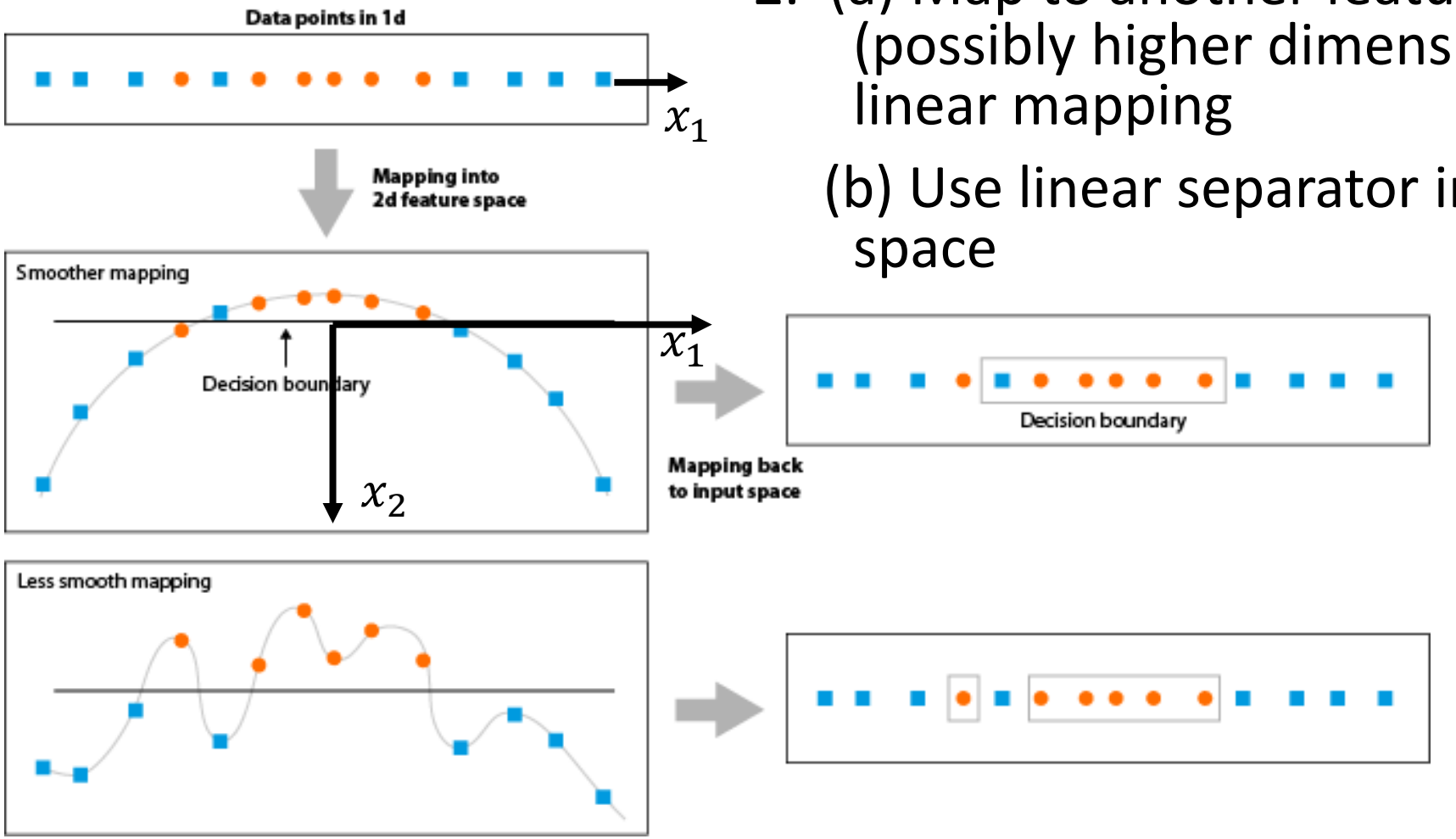
- MLP is a feedforward artificial neural network
- Data flows in forward direction
- MLP must have at least three layers: the input layer, a hidden layer and the output layer
- They are fully connected; each node in one layer connects with a weight to every node in the next layer
- Non linear, activation function can be used such as sigmoid, TanH, ReLU, etc., for deployment.
- Can process linear and non-linear patterns. Can also be implemented with logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Datapoint (1D) Not linearly Separable



1. Non-linear separator

Non-linearly Separable

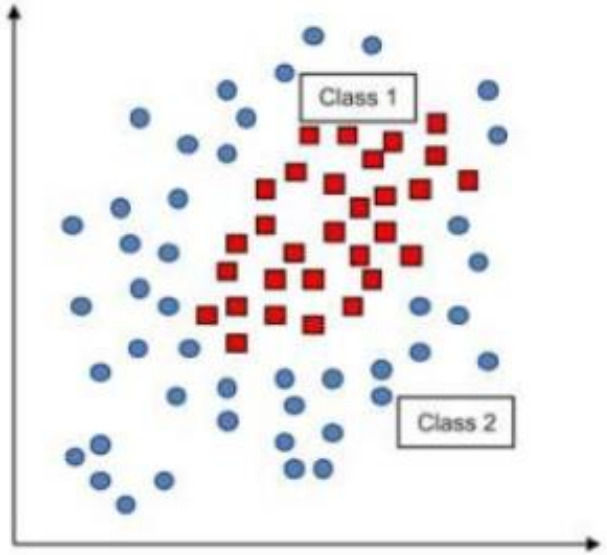


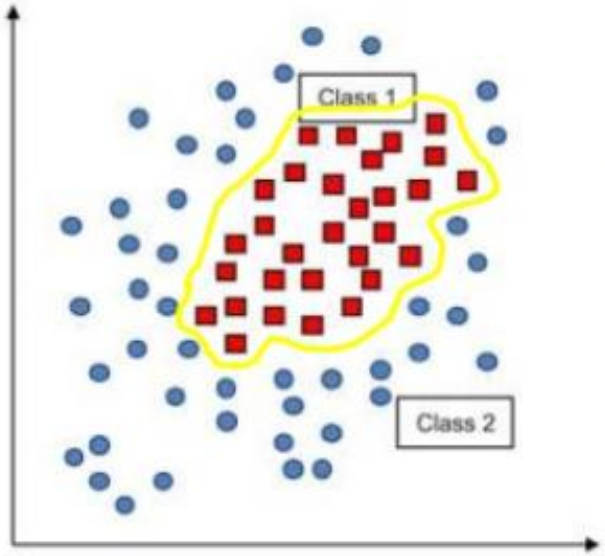
2. (a) Map to another feature space (possibly higher dimension) using non-linear mapping

   (b) Use linear separator in new feature space

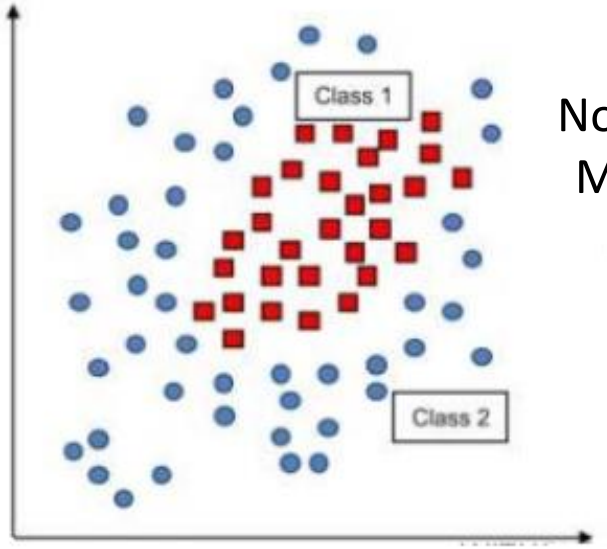
Datapoints in 2D feature space-

Not Linearly Separable

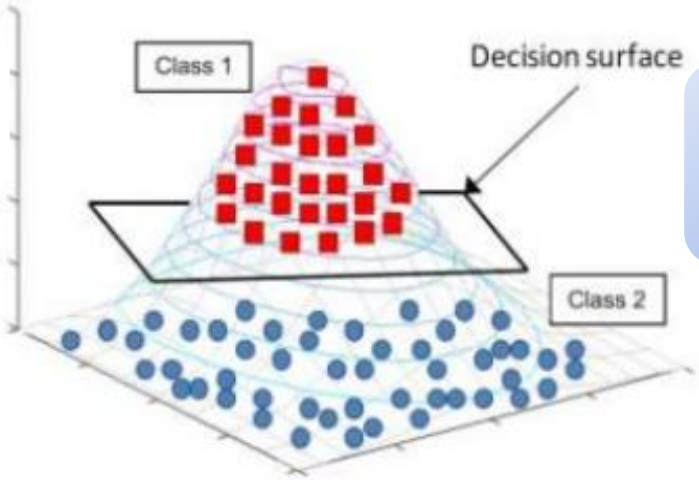




Non-linear Decision Boundary



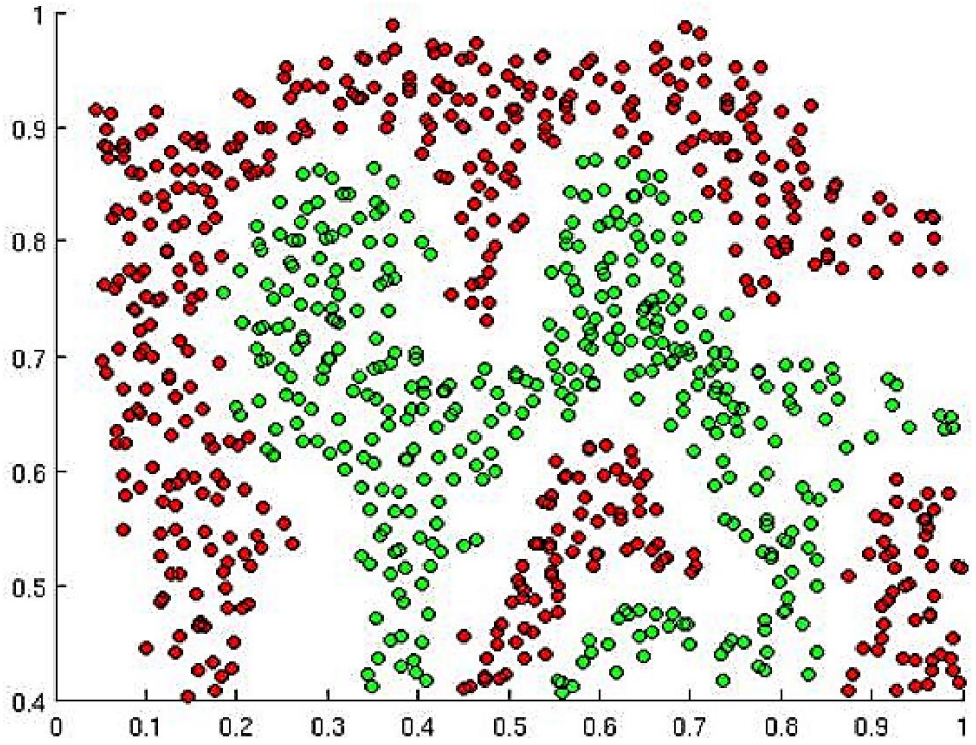
Non-linear Mapping



Linearly Separable

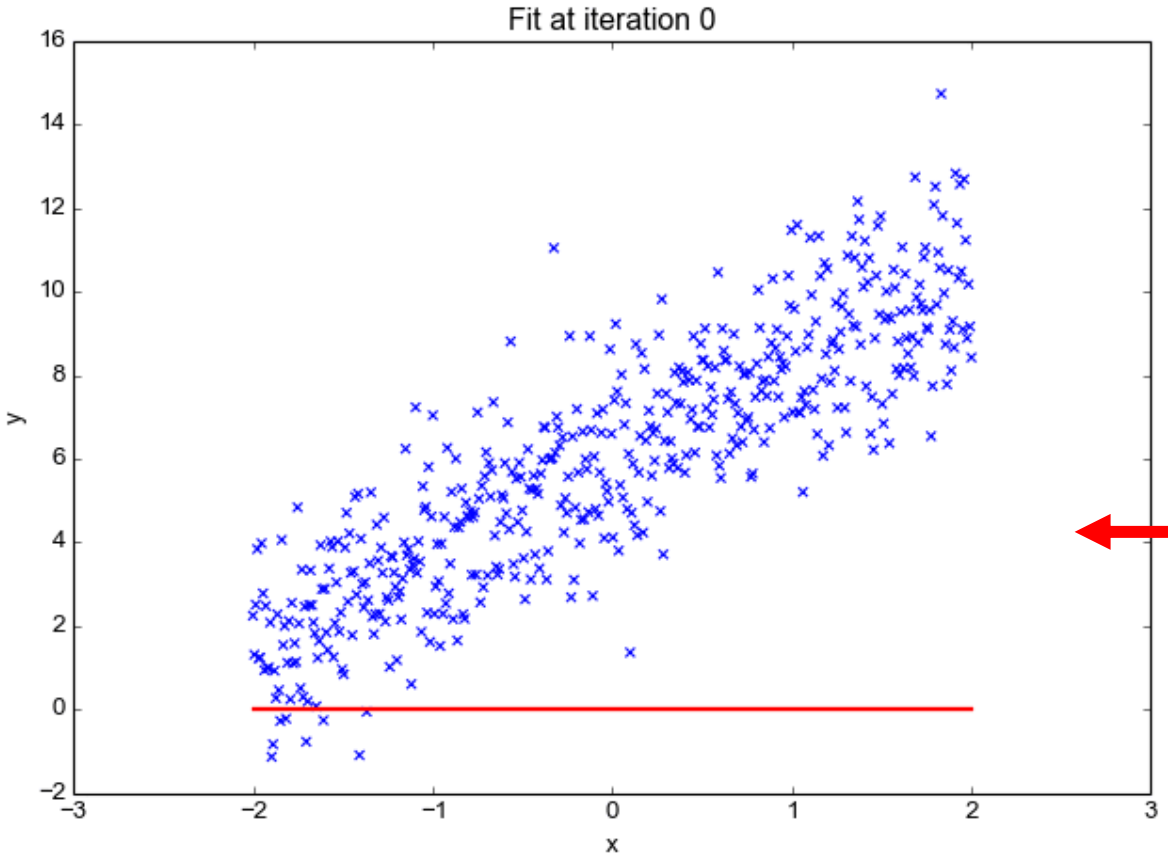
Why Activation Function?

- Application of the activation function tells us that which neurons in each layer will be triggered. Only the neurons with some relevant information are activated in every layer.
- The activation takes place depending on some rule or threshold
- The purpose of the activation function is to **introduce non-linearity** into the network.
- As most of the data in real life is non linear.



Example : **Separating green points from red points in the graph.**

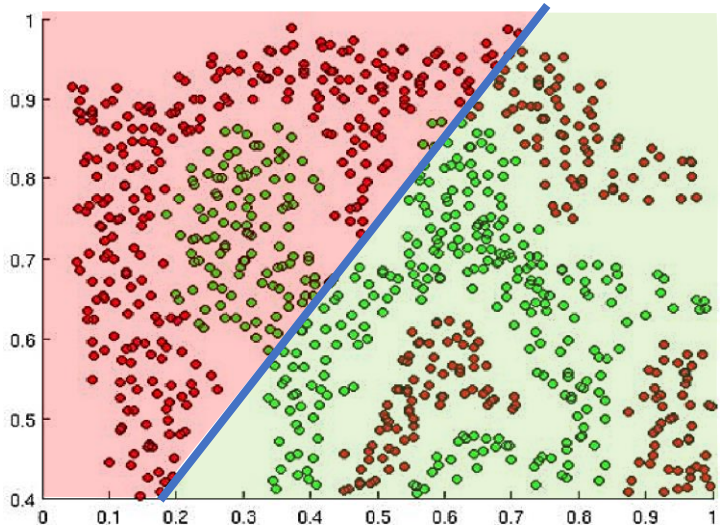
Can we do without an activation function?



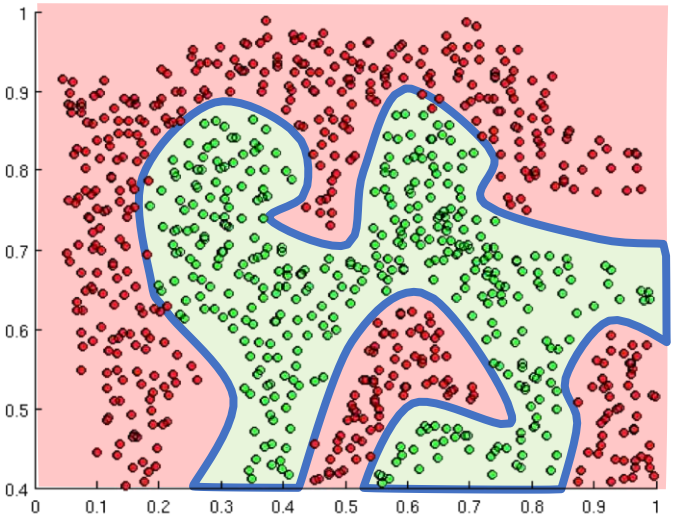
- As it introduces an additional step at each layer during the forward propagation, increases complexity
- In that case, every neuron will only perform a linear transformation on the inputs using the weights and biases that make it simpler and unable to learn the complex patterns from data.
- without an activation function it is just a linear regression model.
- activation function introduces non-linearity in the network.

Importance of Activation Functions

- Separating green points from red points using linear function results into under fitting problem.
- No matter how deep and how large is the network if using linear activation function it just composing lines on top of lines to get another line.
- Whereas using non linear function generates non linear boundaries in the network which is extremely powerful for classification task.



Linear Activation functions produce linear decisions no matter the network size



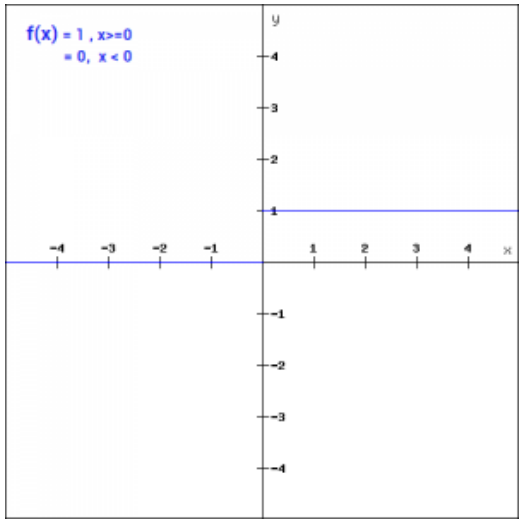
Non-linearities allow us to approximate arbitrarily complex functions

Types of Activation Function

- Binary step
- Linear
- Sigmoid
- Tanh
- ReLu
- Leaky ReLu
- Softmax

Binary Step

f(x) = 1, x>=0
= 0, x<0



Python Code:

```
def
binary_step(x):
    if x<0:
        return 0
    else:
        return 1
binary_step(5),
binary_step(-1)
```

Output:

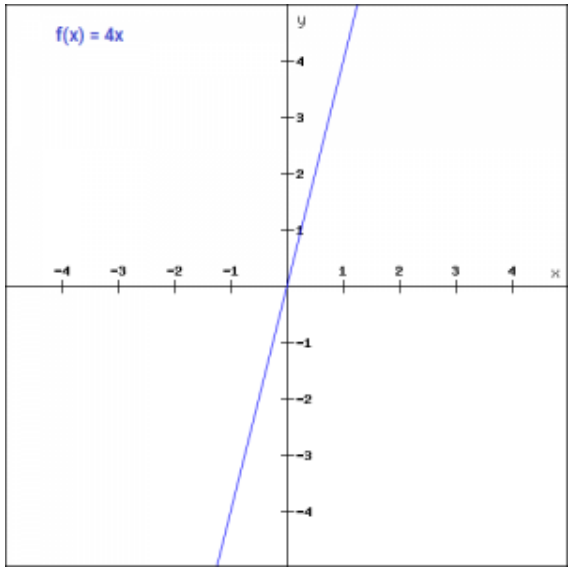
```
(5,0)
```

- Threshold based classifier
- If the input to the activation function is > threshold, then neuron is activated, else deactivated, i.e. its output is not considered for next hidden layer
- Useful for binary class only, not multi class
- The gradient of the step function is '0' which causes a hindrance in the back propagation process.

$$f'(x) = 0, \text{ for all } x$$

Linear Function

f(x)=ax



Python Code:

```
def linear_function(x):
    return 4*x
linear_function(4), linear_function(-2)
```

Output:

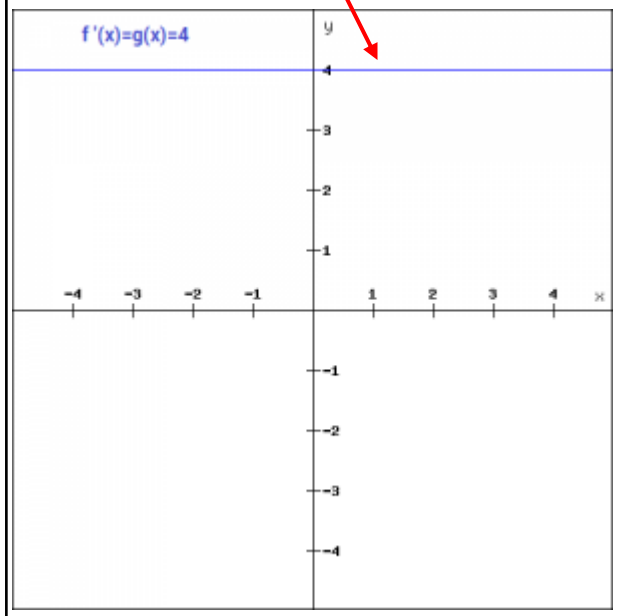
```
(16, -8)
```

- The activation is proportional to the input. The variable ‘a’ in this case can be any constant value
- Differentiate the function with respect to x, the result is the coefficient of x, which is a constant.
- Although the gradient here does not become zero, but it is a constant

$$f'(x) = a$$

Linear Function (Gradient)

f'(x) = a



Python Code:

```
def linear_function(x):
    return 4*x
linear_function(4),
linear_function(-2)
```

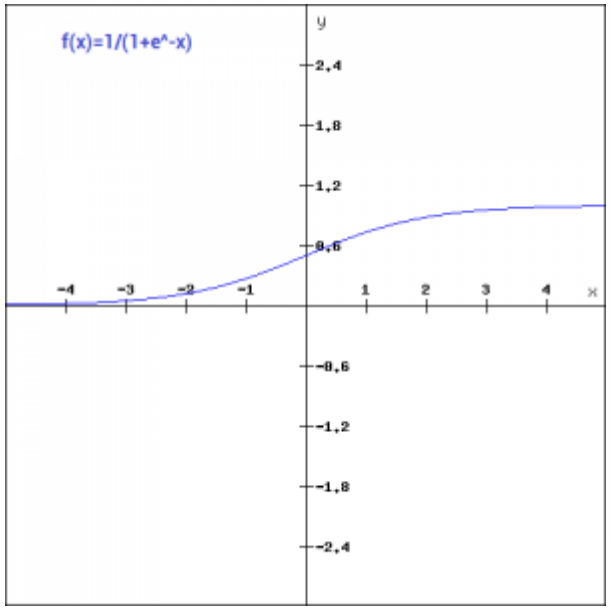
Output:

```
(16, -8)
```

- Implies during the backpropagation process weights & biases get updated with same updating factor.
- Neural network not improve the error as gradient is same for every iteration.
- Not suitable to capture the complex patterns from the data.

Sigmoid

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



Python Code:

```
import numpy as np

def sigmoid_function(x):

    z = (1/(1 + np.exp(-x)))

    return z

sigmoid_function(7),sigmoid_function(-22)
```

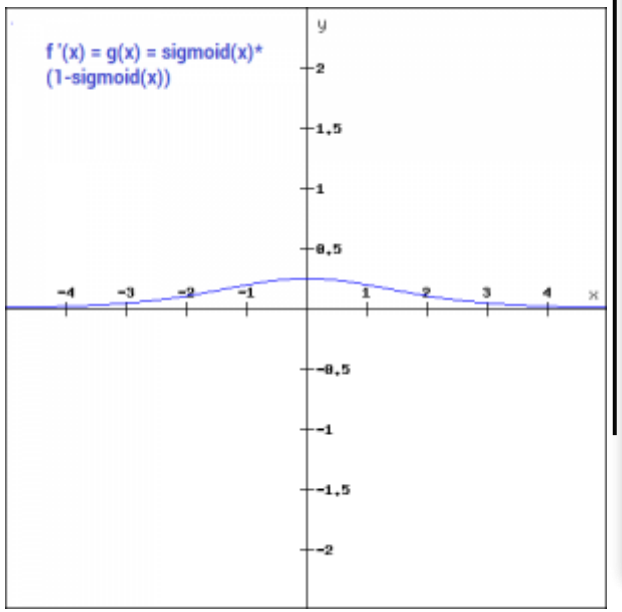
Output:

```
(0.9990889488055994, 2.7894680920908113e-10)
```

- The sigmoid function is not symmetric around 0, thus output of all neurons is of same sign.
  - This can be addressed by scaling the sigmoid function which is exactly what happens in the *tanh function*
- As input values move away from 0, the output value becomes less sensitive. Even a large change in input values results in little to no change in the output value

Sigmoid (gradient)

$F'(x) = 1/(1+e^{-x})$



Python Code:

```
import numpy as np

def sigmoid_function(x):

    z = (1/(1 + np.exp(-x)))

    return z

sigmoid_function(7),sigmoid_function(-22)
```

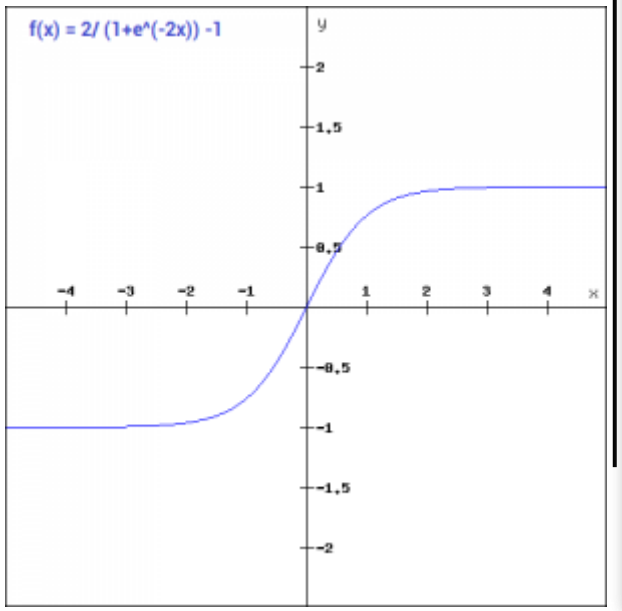
Output:

```
(0.9990889488055994, 2.7894680920908113e-10)
```

- Gradient values are significant for range -3 and 3 but graph is much flatter in other regions. This implies that for values > 3 or < -3, will have very small gradients. As the gradient value approaches zero, the network is not really learning.

Tanh

$\tanh(x) = 2/(1+e^{-2x}) - 1$



Python Code:

```
import numpy as np

def tanh_function(x):

    z = (2/(1 + np.exp(-2*x))) -1

    return z

tanh_function(0.5), tanh_function(-1)
```

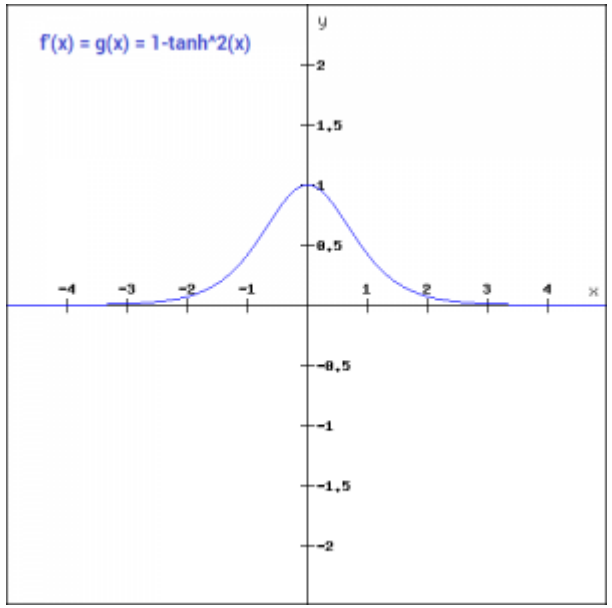
Output:

```
(0.4621171572600098, -0.7615941559557646)
```

- Symmetric around the origin. The range of values in this case is from -1 to 1.
- Thus the inputs to the next layers will not always be of the same sign.

Tanh (gradient)

$\tanh(x) = 2/(1+e^{-2x}) - 1$



Python Code:

```
import numpy as np

def tanh_function(x):

    z = (2/(1 + np.exp(-2*x))) -1

    return z

tanh_function(0.5), tanh_function(-1)
```

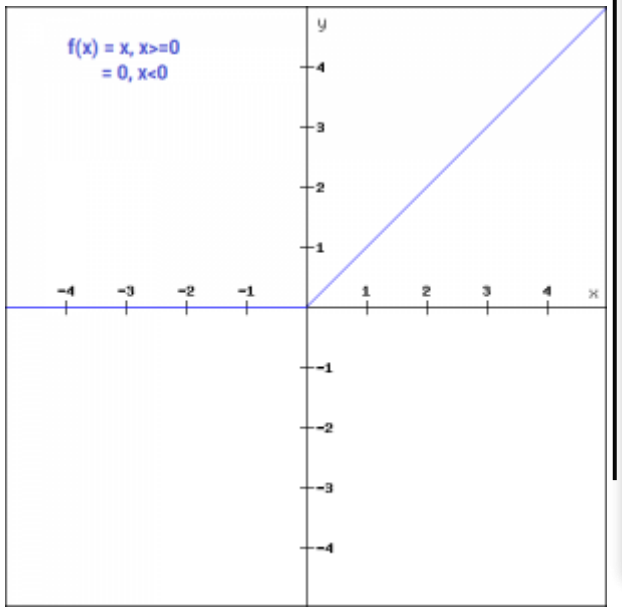
Output:

```
(0.4621171572600098, -0.7615941559557646)
```

- The gradient is steeper as compared to the sigmoid function.
- Usually, tanh is preferred over the sigmoid function since it is zero centered & gradients are not restricted to move in a certain direction.

ReLu (Rectified linear Unit)

$f(x)=max(0,x)$



Python Code:

```
def relu_function(x):
    if x<0:
        return 0
    else:
        return x
relu_function(7),
relu_function(-7)
```

Output:

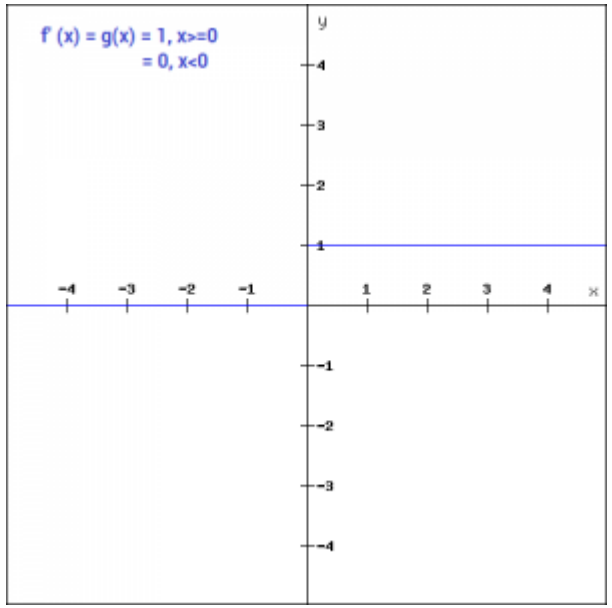
```
(7, 0)
```

- As compared to other activation function it does not activate all the neurons at same time
- The neurons will only be deactivated if the output of linear transformation is < 0.
- Since only a certain no. of neurons are activated, the ReLU is far more computationally efficient compared to sigmoid & tanh function

ReLu (gradient)

f'(x) = 1, x>=0

= 0, x<0



Python Code:

```
def relu_function(x):
    if x<0:
        return 0
    else:
        return x
relu_function(7),
relu_function(-7)
```

Output:

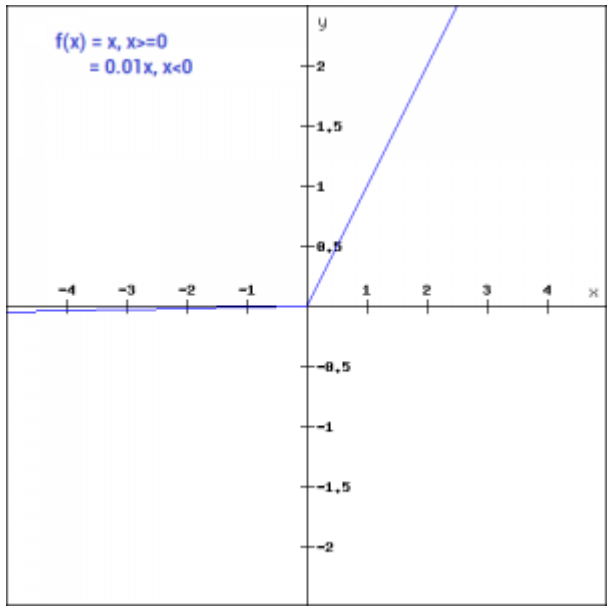
```
(7, 0)
```

- At negative side of graph, the gradient value is 0. Thus during the backpropogation process, the weights and biases for some neurons are not updated.
- This may create dead neurons which never get activated. This is taken care of by the *'Leaky' ReLU function*.

Leaky ReLu

f(x)= 0.01x, x<0

= x, x>=0



Python Code:

```
def
leaky_relu_function(x):
    if x<0:
        return 0.01*x
    else:
        return x
leaky_relu_function(7),
leaky_relu_function(-7)
```

Output:

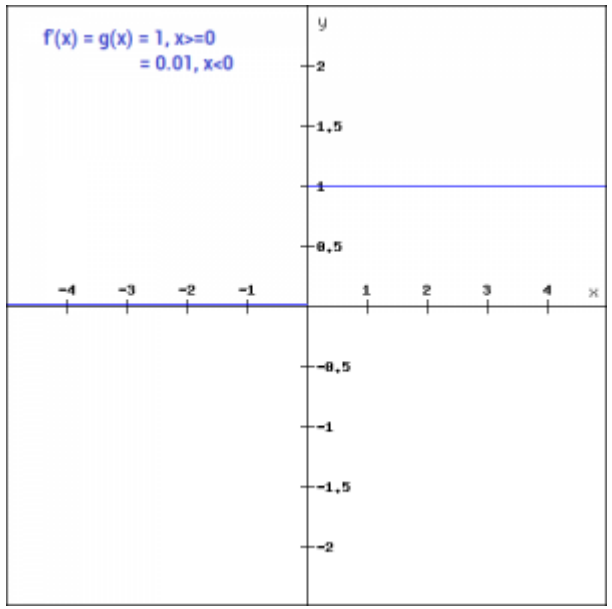
```
(7, -0.07)
```

- The ReLU function, the gradient is 0 for x<0, which would deactivate the neurons in that region
- Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x, we define it as an extremely small linear component of x
- Hence we would no longer encounter dead neurons in that region

Leaky ReLu (gradient)

f'(x) = 1, x>=0

=0.01, x<0



Python Code:

```
def
leaky_relu_function(x):
    if x<0:
        return 0.01*x
    else:
        return x
leaky_relu_function(7),
leaky_relu_function(-7)
```

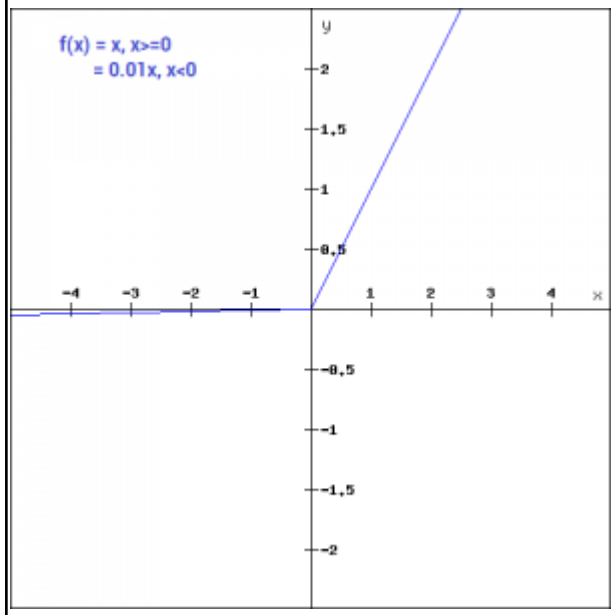
Output:

```
(7, -0.07)
```

- By making this small modification, the gradient of the left side of the graph comes out to be a non-zero value.
- Hence it no longer encounter dead neurons in that region.

Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



Python Code:

```
def softmax_function(x):
    z = np.exp(x)
    z_ = z/z.sum()
    return z_
softmax_function([0.8, 1.2, 3.1])
```

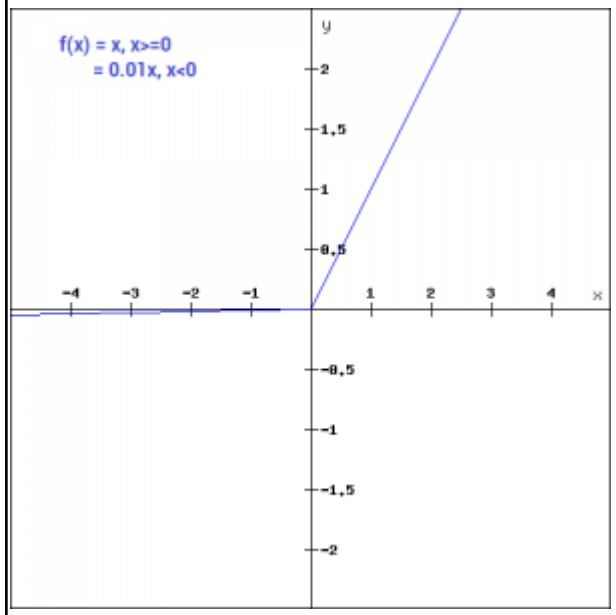
Output:

```
array([0.08021815, 0.11967141, 0.80011044])
```

- Softmax function is a combination of multiple sigmoids. Since, sigmoid returns values between 0 and 1, which can be treated as probabilities of a data point belonging to a particular class..
- The softmax function can be used for multiclass classification problems. This function returns the probability for a datapoint belonging to each individual class

Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



Python Code:

```
def softmax_function(x):
    z = np.exp(x)
    z_ = z/z.sum()
    return z_
softmax_function([0.8, 1.2, 3.1])
```

Output:

```
array([0.08021815, 0.11967141, 0.80011044])
```

- For a multiclass problem, the output layer would have as many neurons as the number of classes in the target. if you have three classes, there would be three neurons in the output layer. Let the output from the neurons as [1.2 , 0.9 , 0.75].
- Applying the softmax function over these values, you will get the following result – [0.42 , 0.31, 0.27].
- These represent the probability for the data point belonging to each class.

Note that the sum of all the values is 1