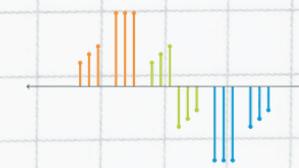


# DSP Lecture 2

## INTRODUCTION TO DEEP LEARNING

Prepared by: Christian Dave T. Navesis

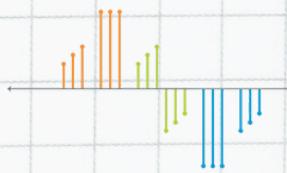


# CONTENTS

---

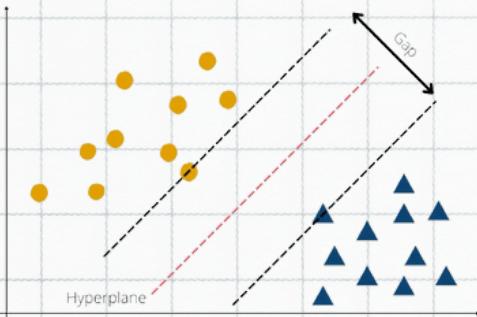
1. ML vs. DL
2. ML\DL Cycle
3. Neural Networks
4. The Perceptron (Artificial Neuron)
5. Weights and Biases
6. Activation Function
7. Loss Functions
8. Gradient Descent and Backpropagation
9. Types of Neural Networks
10. Demo: Let's build a Neural Net!

1. The Convolutional Neural Network (CNN)
2. Kernels and Convolution Operation
3. Pooling
4. Dense Layer
5. Demo: Digits Classification, Cats Vs Dogs



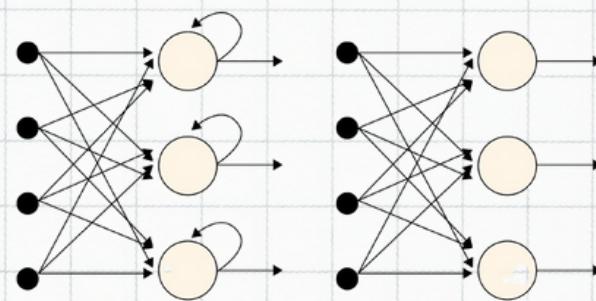
# ML VS. DL

## Machine Learning (Statistical Models)

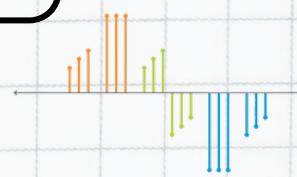


Predetermined Learning.  
Manual Extraction of  
learnable data features.  
Human intervention is  
needed

## Deep Learning (Neural Network Models)

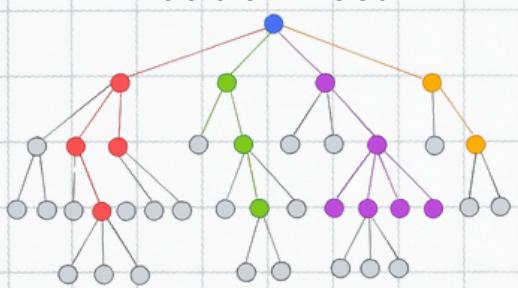


End to end Learning.  
Can learn the learnable  
features from raw data.  
Human intervention is not  
needed



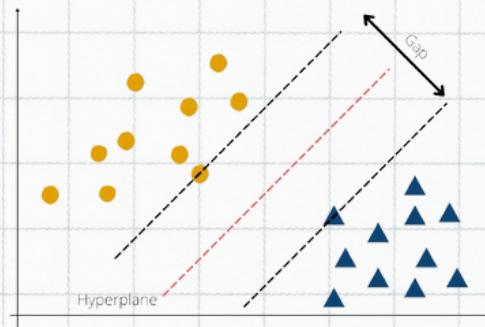
# ML VS. DL

Decision Trees

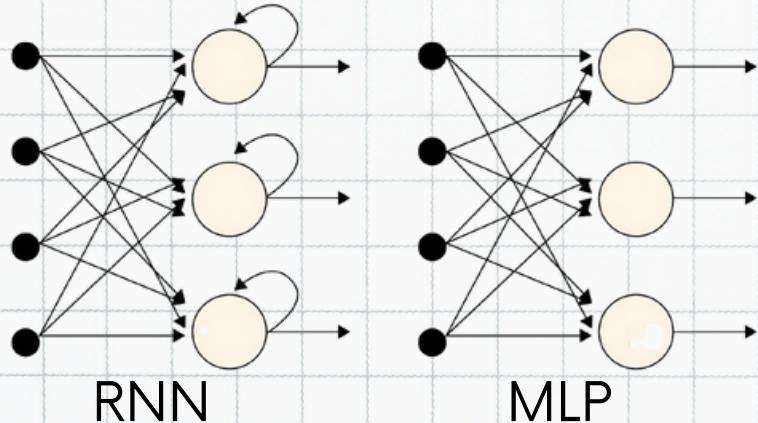


Machine Learning  
(Statistical Models)

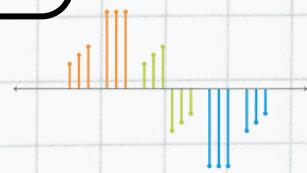
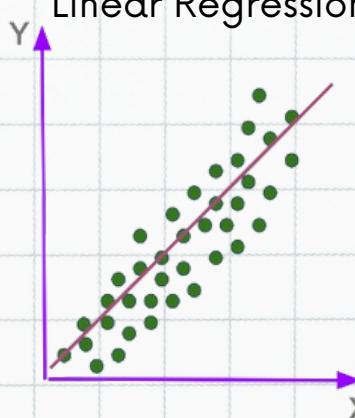
Support Vector Machines



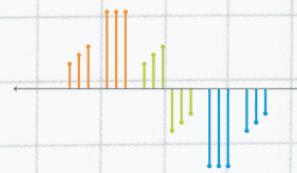
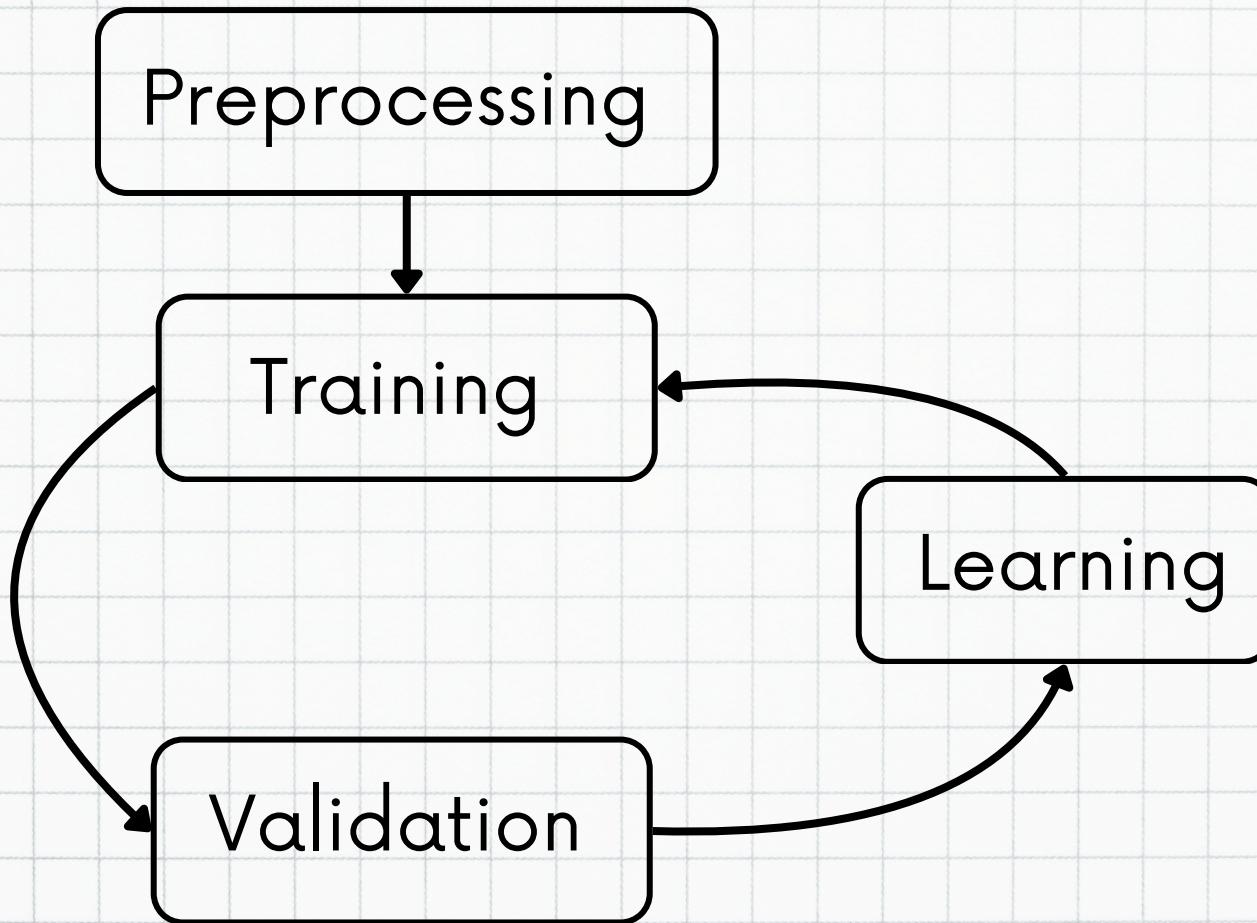
Deep Learning  
(Neural Network Models)



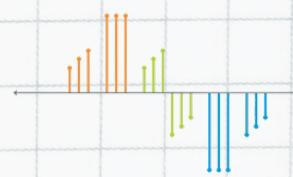
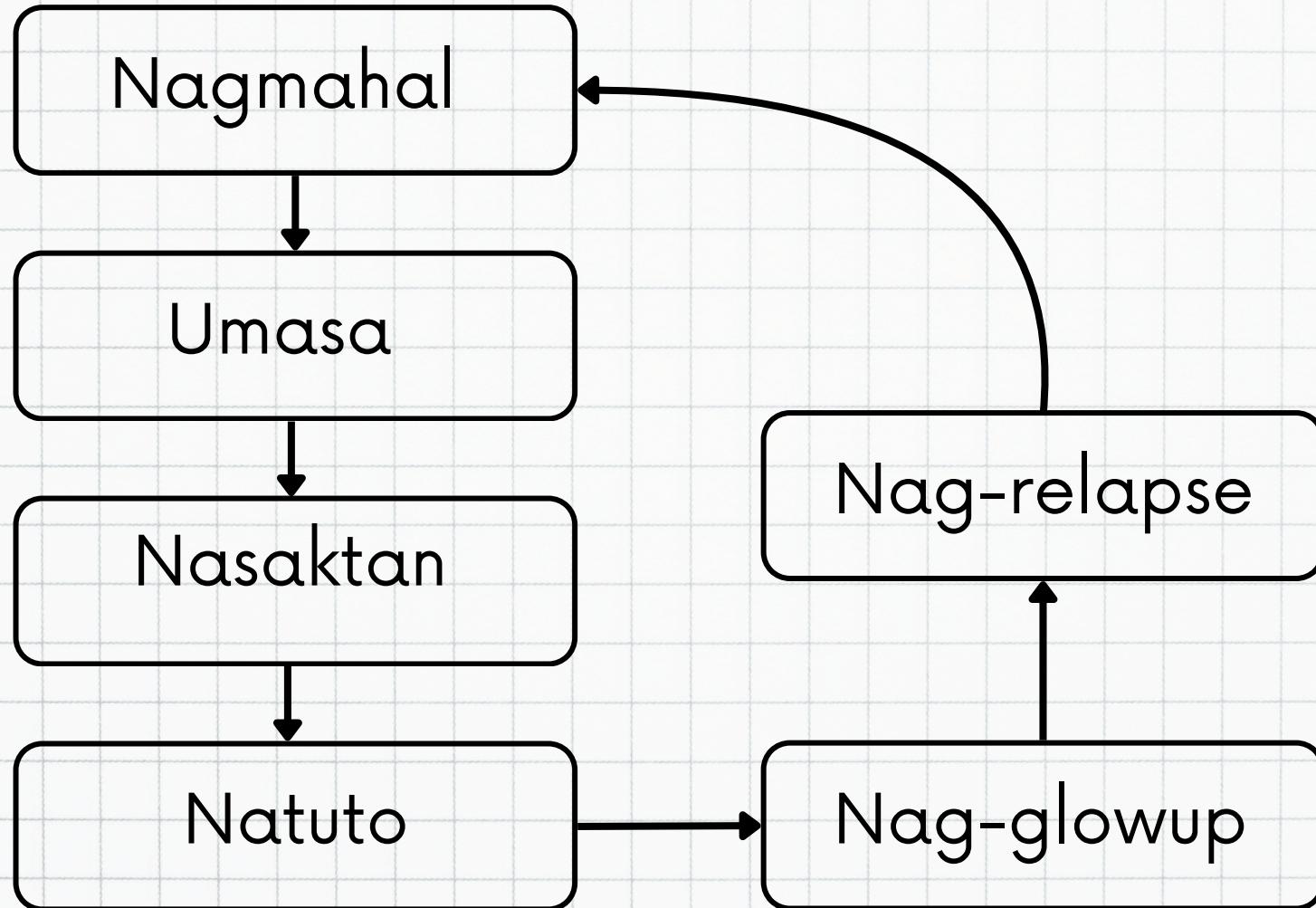
Linear Regression



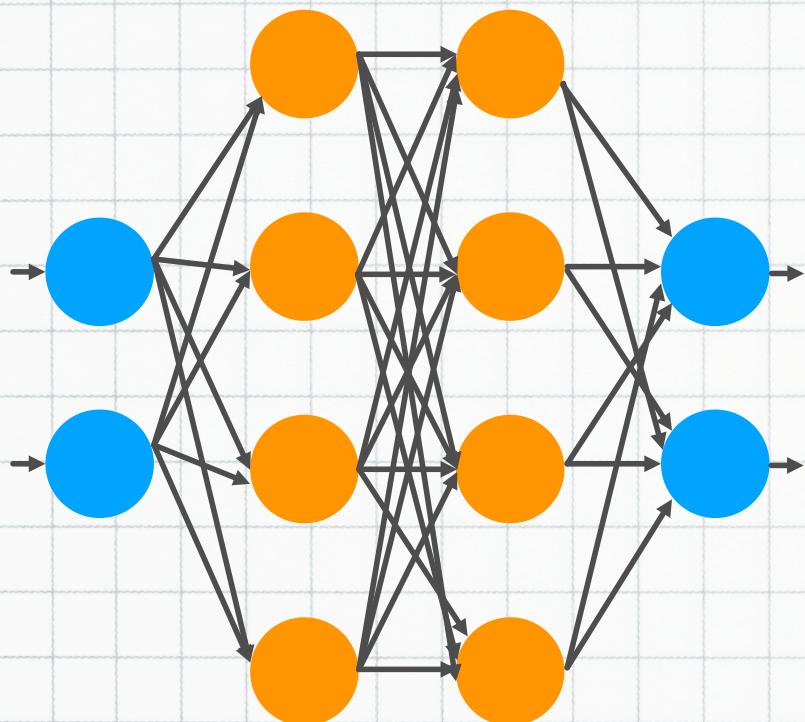
# ML/ DL Cycle



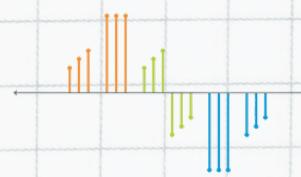
# ML/ DL Cycle



# Neural Networks



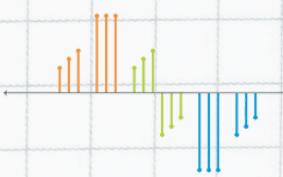
A computational Model that seeks to identify the patterns of a dataset and learn from it. It is inspired by the biological neurons of the brain.



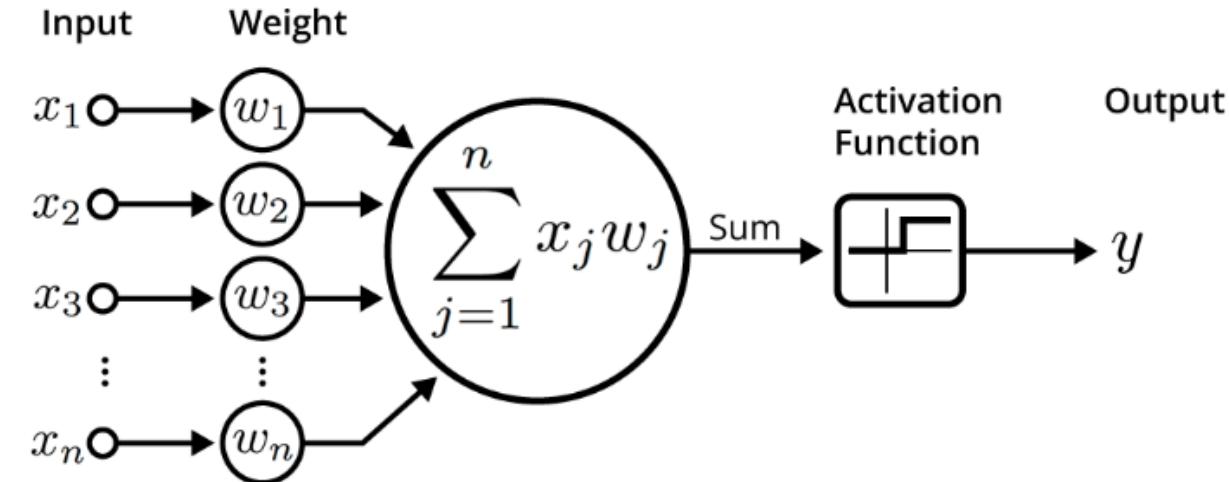
# Neural Networks



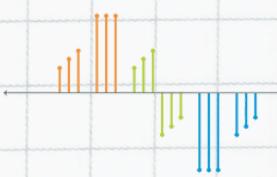
Muffin or  
Chihuahua?



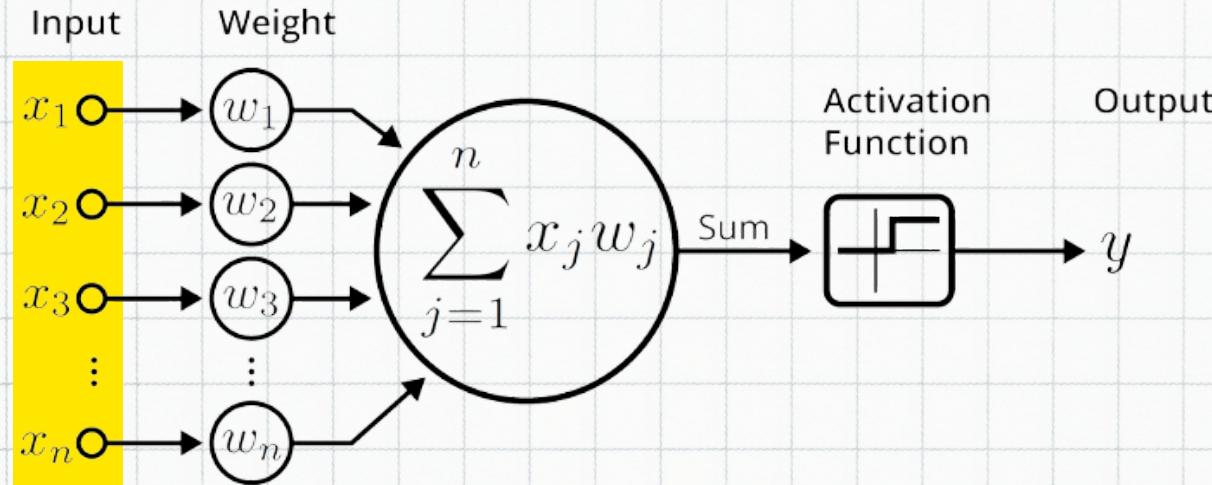
# The Perceptron (Artificial Neuron)



$$y = \sigma \left( \sum_{j=1}^n x_j w_j \right) = \sigma(x_1 w_1 + x_2 w_2 + \dots + x_n w_n)$$



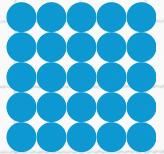
# Input



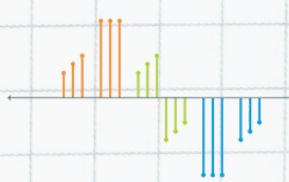
An image, audio signal, or text can be represented as **tensors**.

●  
**scalar**

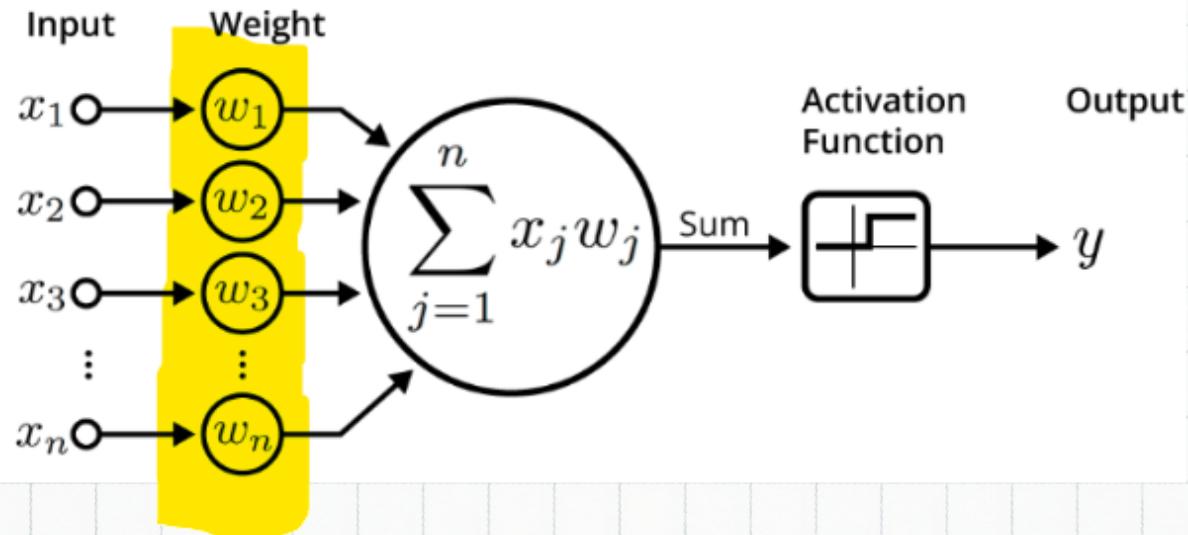
oooo  
**vector**

  
**matrix**

Usually, matrices are flattened to a long vector as inputs to NN.

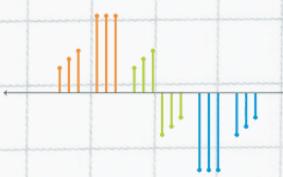


# Weights

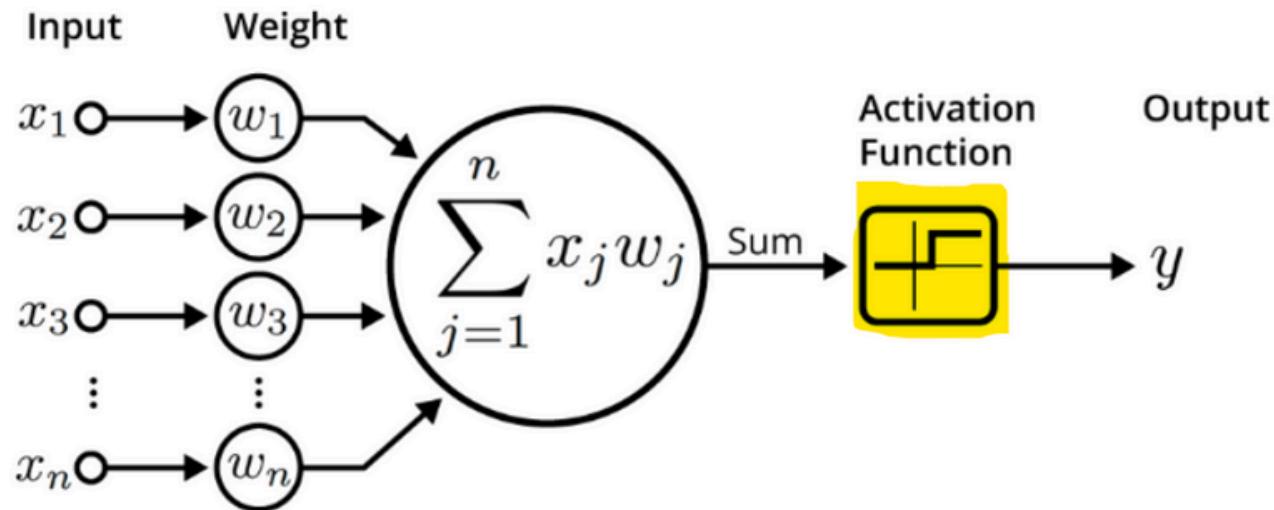


$$y = \sigma \left( \sum_{j=1}^n x_j w_j \right) = \sigma(x_1 w_1 + x_2 w_2 + \dots + x_n w_n)$$

Weights dictates the **influence** of each input to the output of a neuron.

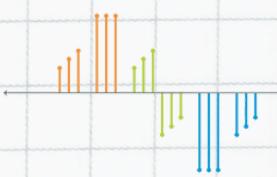


# Activation Function

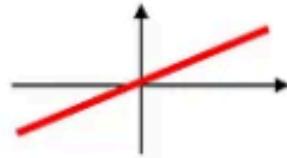
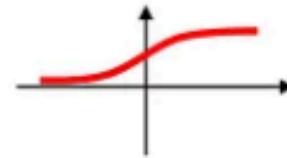
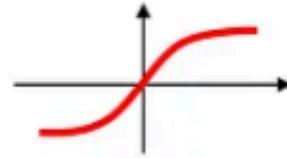
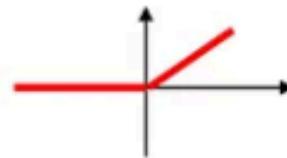


$$y = \sigma \left( \sum_{j=1}^n x_j w_j \right) = \sigma(x_1 w_1 + x_2 w_2 + \dots + x_n w_n)$$

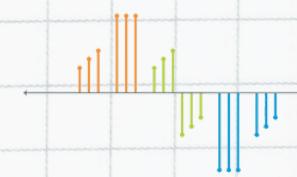
A function that maps the weighted sum of the inputs to an output  $y$ .



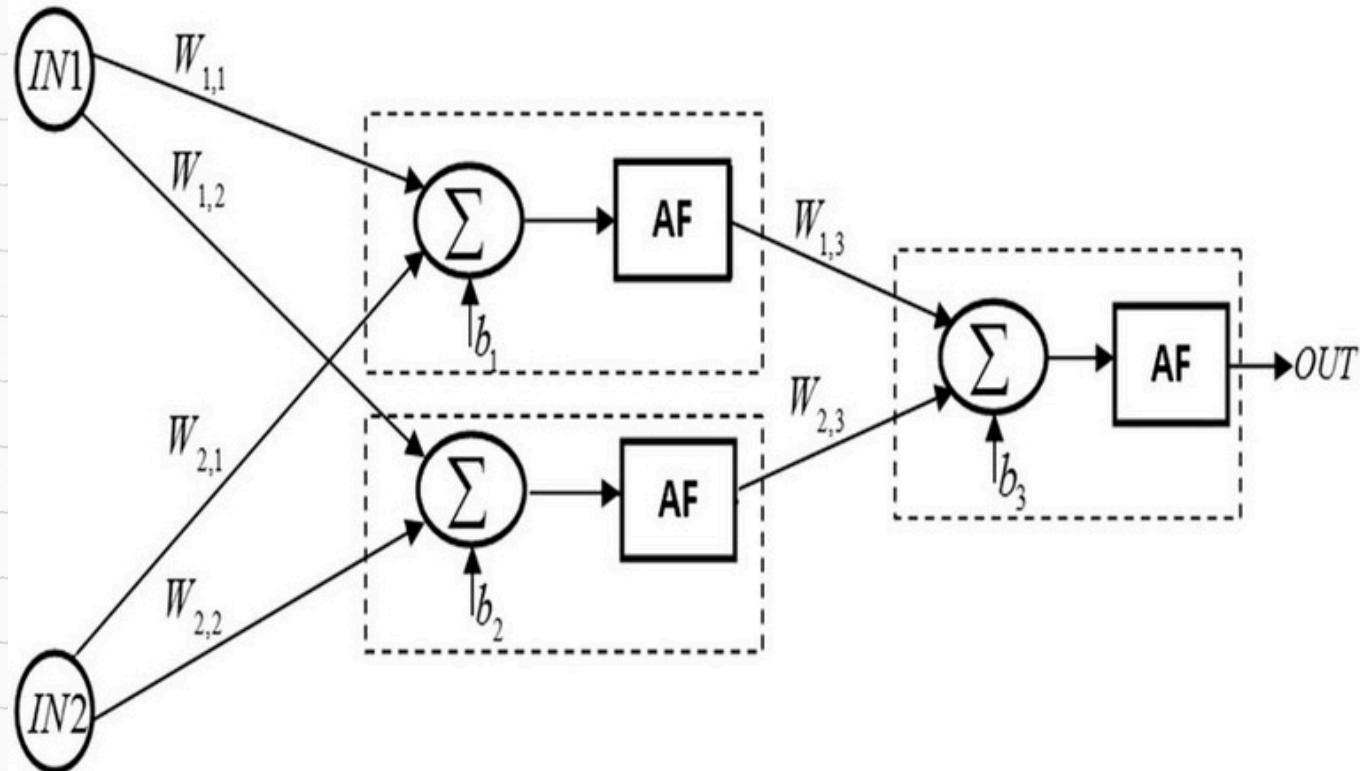
# Activation Function

Activation function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent (Tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	

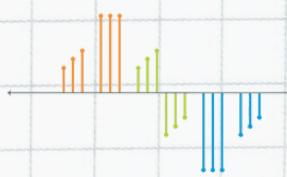
Copyright © Sebastian Raschka 2016  
(<http://sebastianraschka.com>)



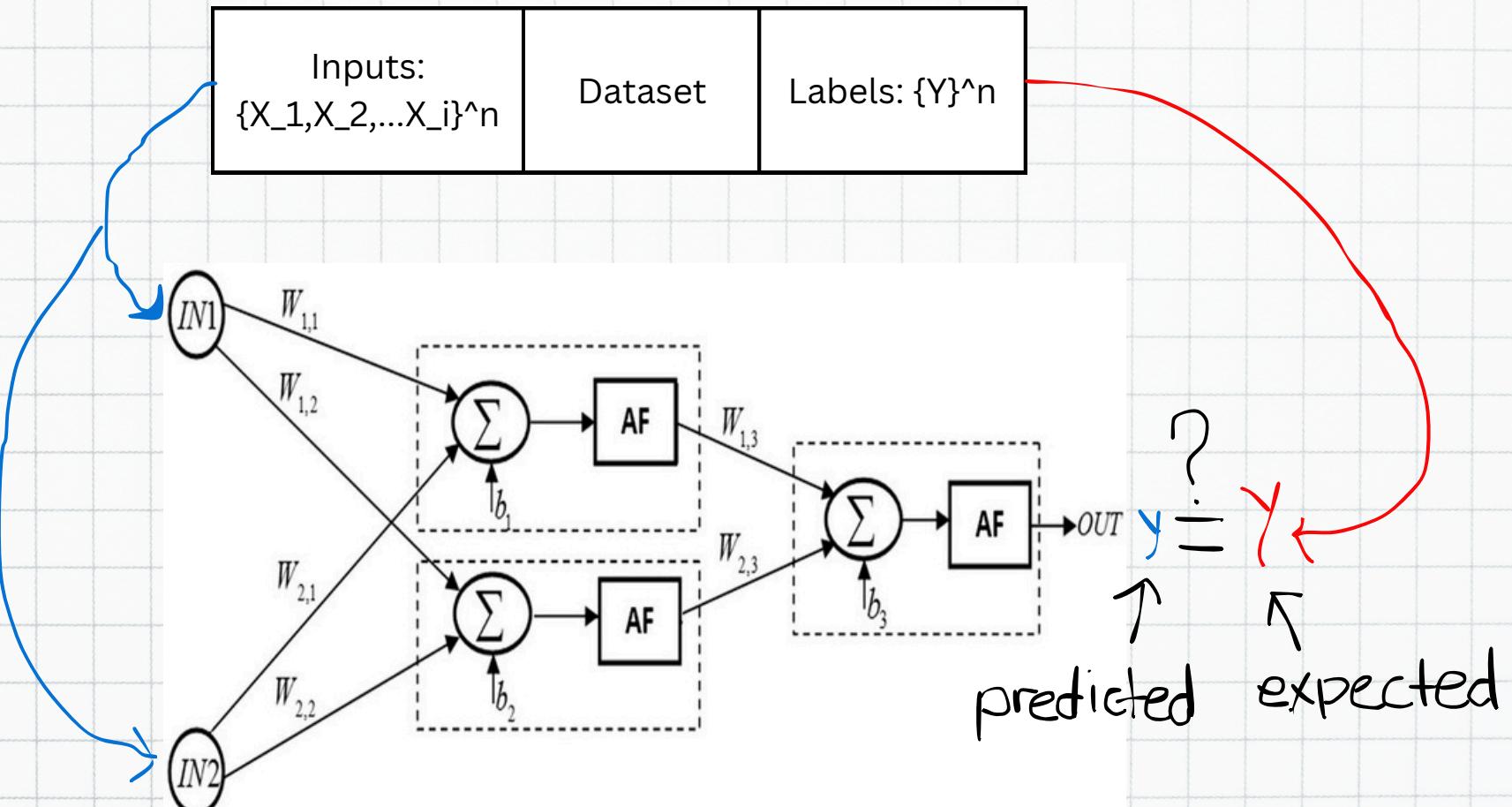
# Multilayered Perceptrons



$$y = \sigma \left( w_{1,3} \sigma(w_{1,1}x_1 + w_{2,1}x_2) + w_{2,3} \sigma(w_{1,2}x_1 + w_{2,2}x_2) \right)$$

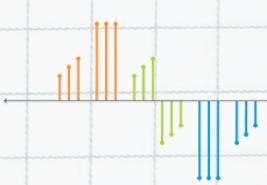


# Multilayered Perceptrons



$$y = \sigma(w_{1,3}\sigma(w_{1,1}x_1 + w_{2,1}x_2) + w_{2,3}\sigma(w_{1,2}x_1 + w_{2,2}x_2))$$

**GOAL:** tweak  $\{w\}$  such that  $y = Y$



# Loss Function

$$LOSS = J(Y, y)$$

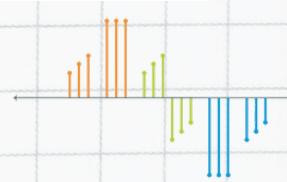
Quantifies how close the prediction  $y$  to the real output  $Y$

$$J_{MSE}(Y, y) = \frac{1}{n} \sum_{i=1}^n (Y_i - y)^2$$

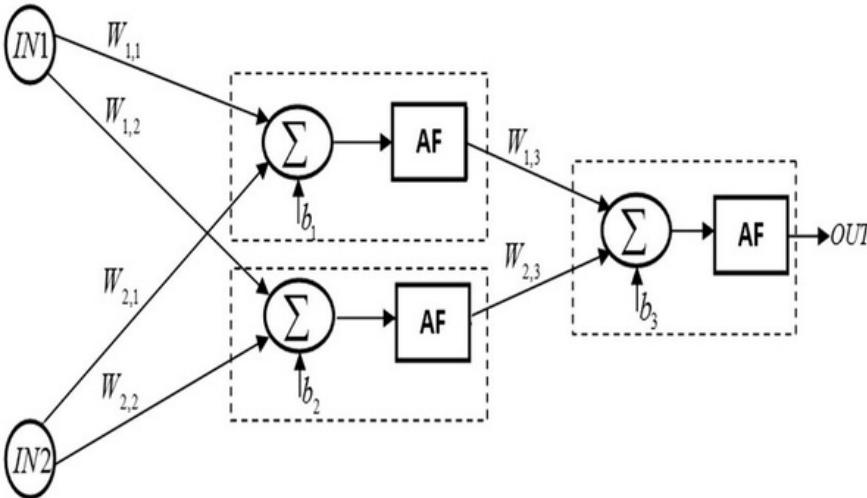
$$J_{MAE}(Y, y) = \frac{1}{n} \sum_{i=1}^n |Y_i - y|$$

$$J_{CE}(Y, y) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(p_{ij})$$

Loss Function	Applicability to Classification	Applicability to Regression
Mean Square Error (MSE) / L2 Loss	✗	✓
Mean Absolute Error (MAE) / L1 Loss	✗	✓
Binary Cross-Entropy Loss / Log Loss	✓	✗
Categorical Cross-Entropy Loss	✓	✗
Hinge Loss	✓	✗
Huber Loss / Smooth Mean Absolute Error	✗	✓
Log Loss	✓	✗

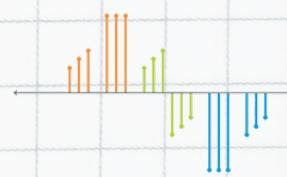


# Multilayered Perceptrons



$$y = \sigma(w_{1,3}\sigma(w_{1,1}x_1 + w_{2,1}x_2) + w_{2,3}\sigma(w_{1,2}x_1 + w_{2,2}x_2))$$

***GOAL:*** tweak  $\{w\}$  to minimize  $J(Y, y)$

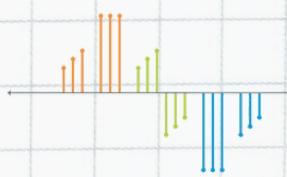


# Gradient Descent

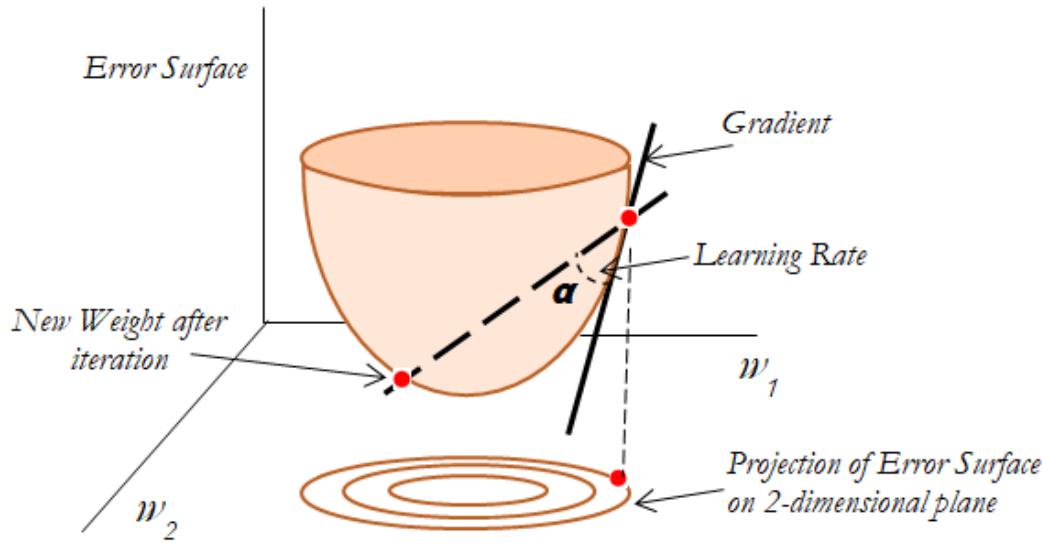
**Gradient Descent** - An algorithm to find the local minimum of our Loss Function?

Why can't we use Math 23 calculus?  
Math 23 Lagrange Multipliers??

- A deep neural network could have thousands to billions of weights. This means we need to find the local minimum of a Billionth dimension function. Now try using Lagrange Multipliers for this.  
Tignan natin kung di ka mabaliw



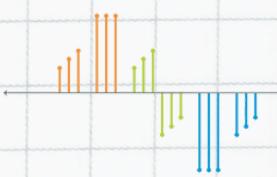
# Gradient Descent



repeat until convergence{

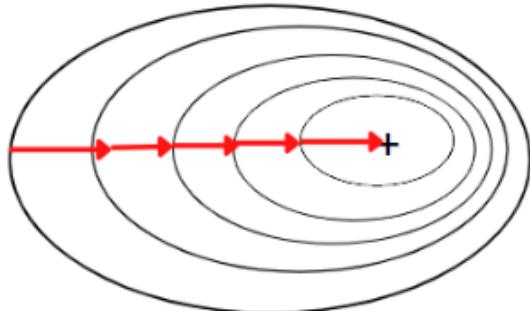
$$w_{i+1} = w_i - \alpha \frac{\partial}{\partial w_i} J(w_i, \{w \setminus w_i\})$$

Learning Rate } update term

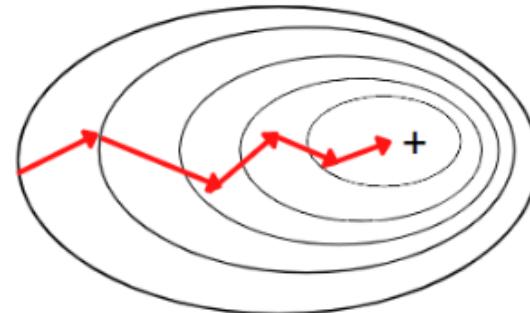


# Types of Gradient Descent

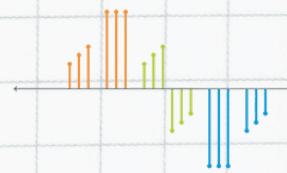
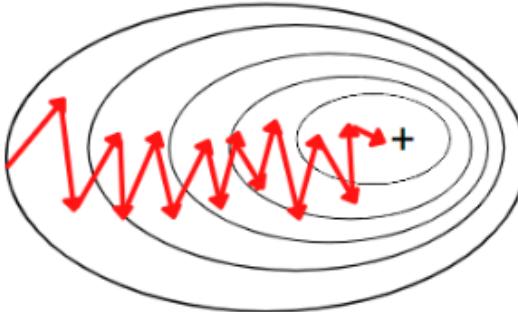
**Batch Gradient Descent**



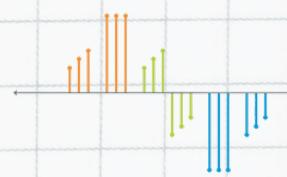
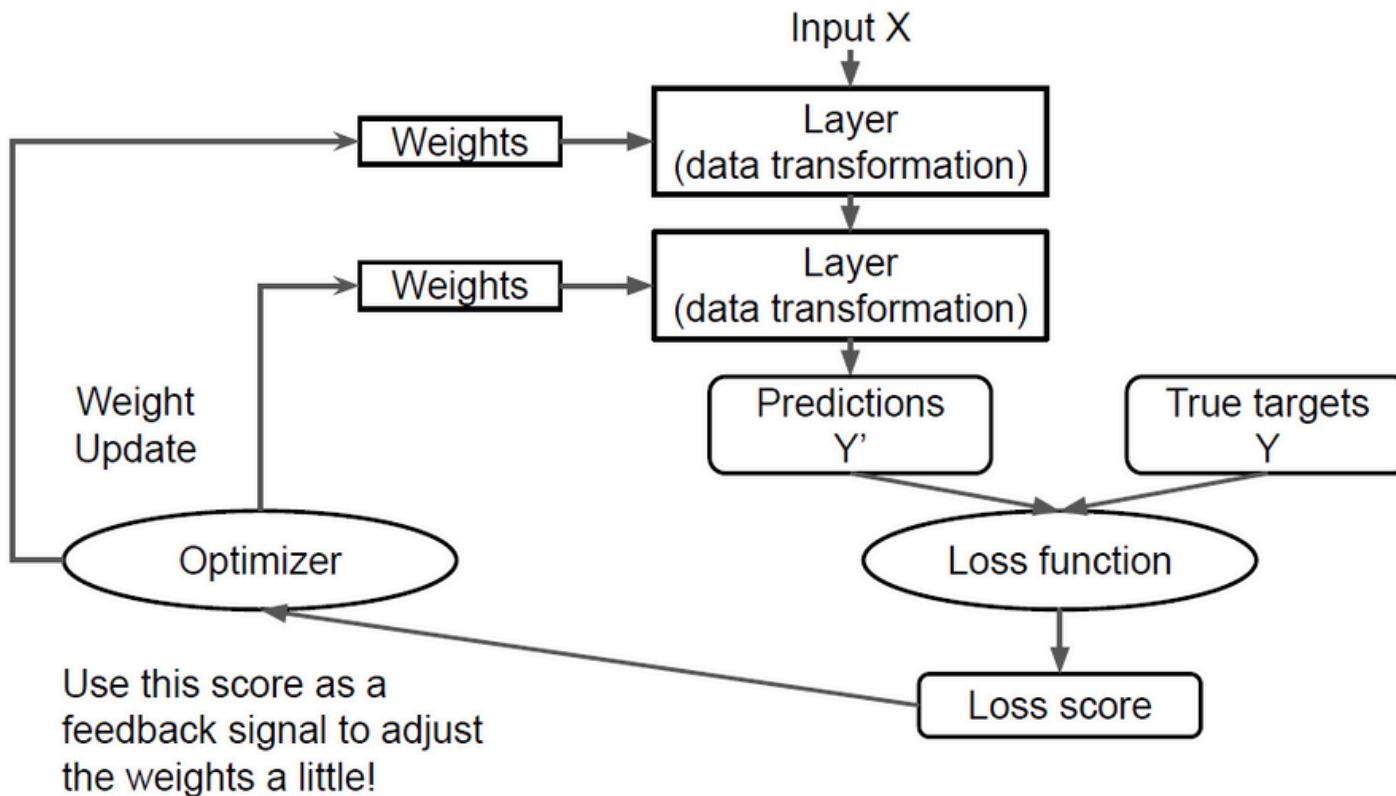
**Mini-Batch Gradient Descent**



**Stochastic Gradient Descent**



# Deep Learning in a nutshell



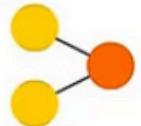
# Types of Neural Networks

*A mostly complete chart of*

## Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

Perceptron (P)



Feed Forward (FF)



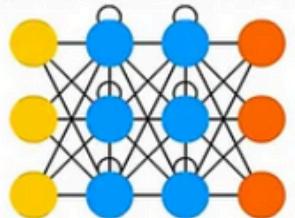
Radial Basis Network (RBF)



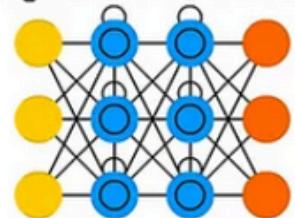
Deep Feed Forward (DFF)



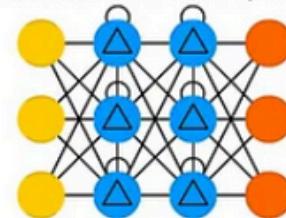
Recurrent Neural Network (RNN)



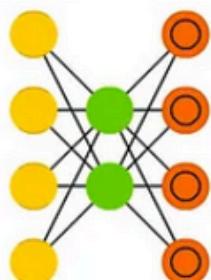
Long / Short Term Memory (LSTM)



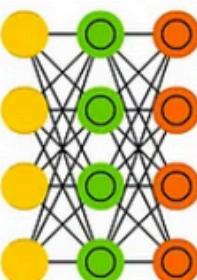
Gated Recurrent Unit (GRU)



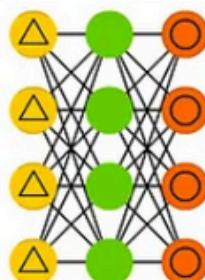
Auto Encoder (AE)



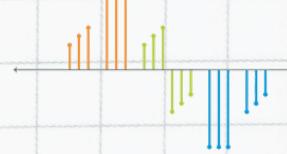
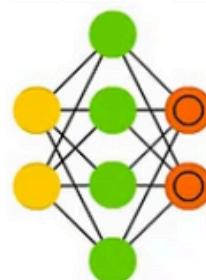
Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



# Types of Neural Networks

Markov Chain (MC)



Hopfield Network (HN)



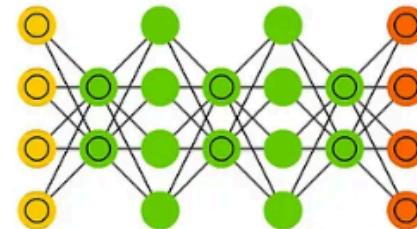
Boltzmann Machine (BM)



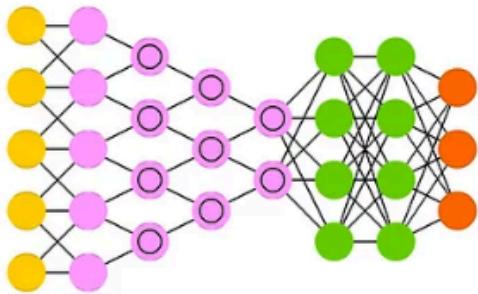
Restricted BM (RBM)



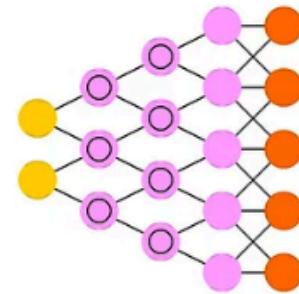
Deep Belief Network (DBN)



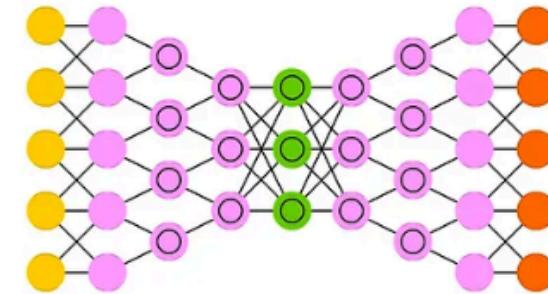
Deep Convolutional Network (DCN)



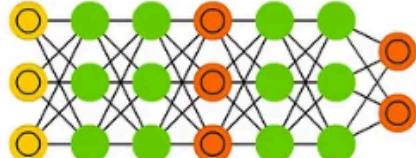
Deconvolutional Network (DN)



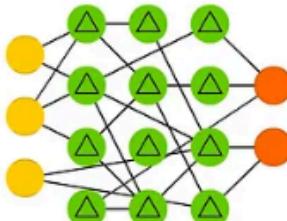
Deep Convolutional Inverse Graphics Network (DCIGN)



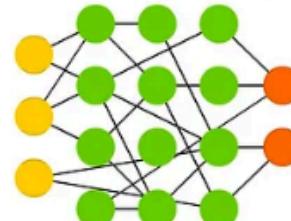
Generative Adversarial Network (GAN)



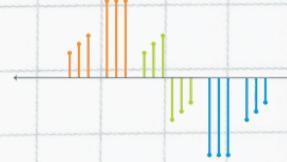
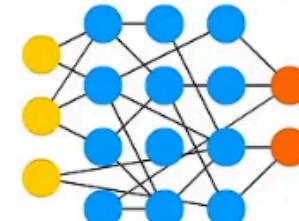
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



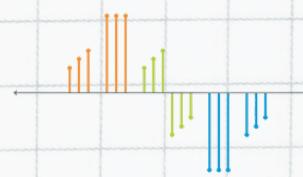
Echo State Network (ESN)



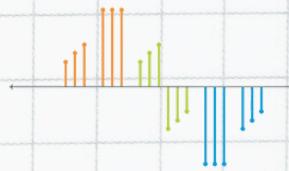
---

# Let's build a Neural Net!

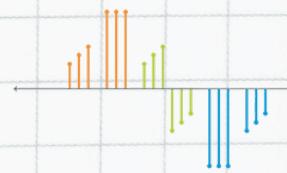
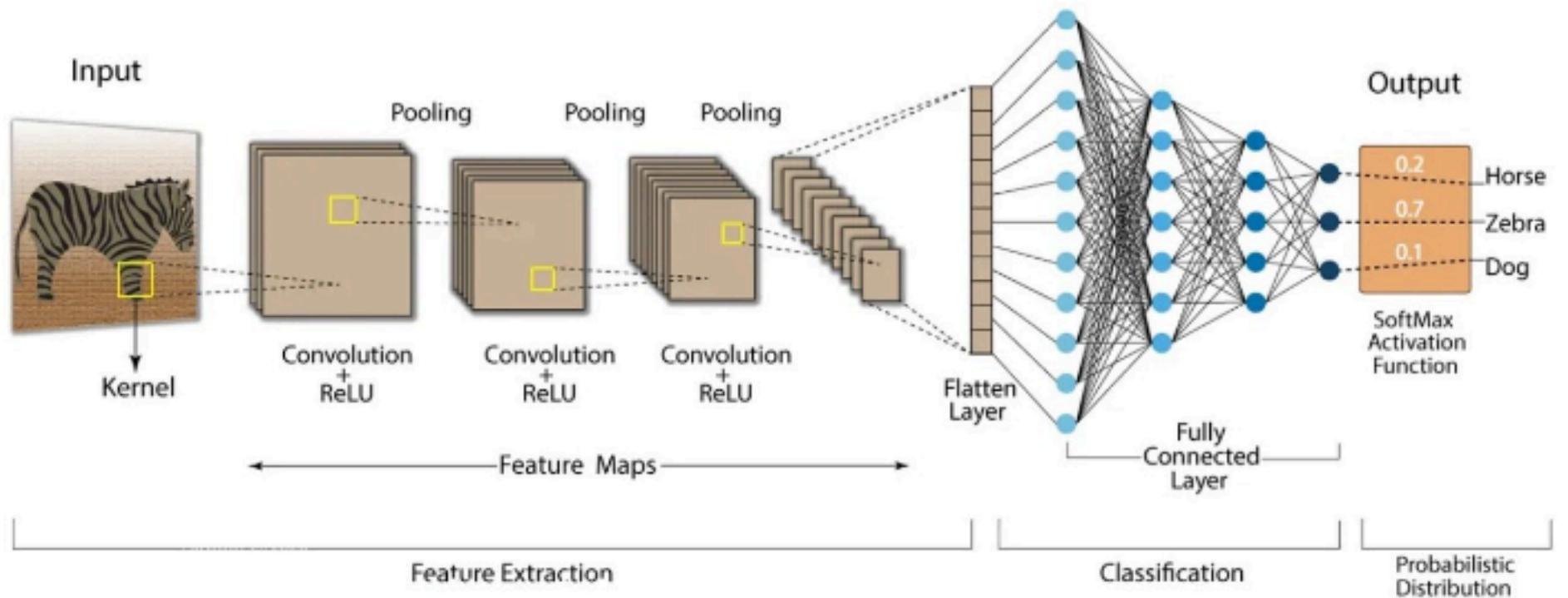
---



**Chill muna 3mins, may part 2 pa ;)**



# Convolutional Neural Network (CNN)



# 2DConvolution

**Input Image:**

10	9	9	4	0
0	6	6	2	2
5	9	8	4	3
7	5	5	4	3
8	10	8	5	0

**Prewitt Kernels**

-1	0	1
-1	0	1
-1	0	1

X-Direction

1	1	1
0	0	0
-1	-1	-1

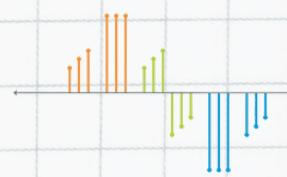
Y-Direction

**X-Direction**

X	X	X	X	X
X	8	-14	-18	X
X	7	-10	-11	X
X	1	-11	-15	X
X	X	X	X	X

**Y-Direction**

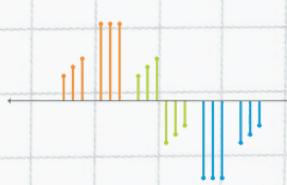
X	X	X	X	X
X	6	1	-2	X
X	-5	0	-2	X
X	-4	-2	2	X
X	X	X	X	X



# 2DConvolution

<table border="1"><tr><td>8</td><td>-14</td><td>-18</td></tr><tr><td>7</td><td>-10</td><td>-11</td></tr><tr><td>1</td><td>-11</td><td>-15</td></tr></table>	8	-14	-18	7	-10	-11	1	-11	-15	X-Direction	<b>Magnitude</b>									
8	-14	-18																		
7	-10	-11																		
1	-11	-15																		
<table border="1"><tr><td>6</td><td>1</td><td>-2</td></tr><tr><td>-5</td><td>0</td><td>-2</td></tr><tr><td>-4</td><td>-2</td><td>2</td></tr></table>	6	1	-2	-5	0	-2	-4	-2	2	Y-Direction	<table border="1"><tr><td>10</td><td>14</td><td>18</td></tr><tr><td>9</td><td>10</td><td>11</td></tr><tr><td>4</td><td>11</td><td>15</td></tr></table>	10	14	18	9	10	11	4	11	15
6	1	-2																		
-5	0	-2																		
-4	-2	2																		
10	14	18																		
9	10	11																		
4	11	15																		

<b>Magnitude</b>	<b>Edge</b>																		
<table border="1"><tr><td>10</td><td>14</td><td>18</td></tr><tr><td>9</td><td>10</td><td>11</td></tr><tr><td>4</td><td>11</td><td>15</td></tr></table>	10	14	18	9	10	11	4	11	15	<table border="1"><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	0	1	1	0	0	0	0	0	1
10	14	18																	
9	10	11																	
4	11	15																	
0	1	1																	
0	0	0																	
0	0	1																	
<b>Threshold = 11</b>																			
<b>Threshold = <math>(10 + 14 + 18 + 9 + 10 + 11 + 4 + 11 + 15) / 9 = 11</math></b>																			



# Max/Avg. Pooling

Max Pooling

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

$2 \times 2$   
pool size

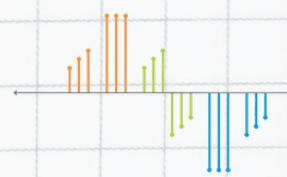
100	184
12	45

Average Pooling

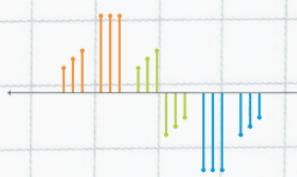
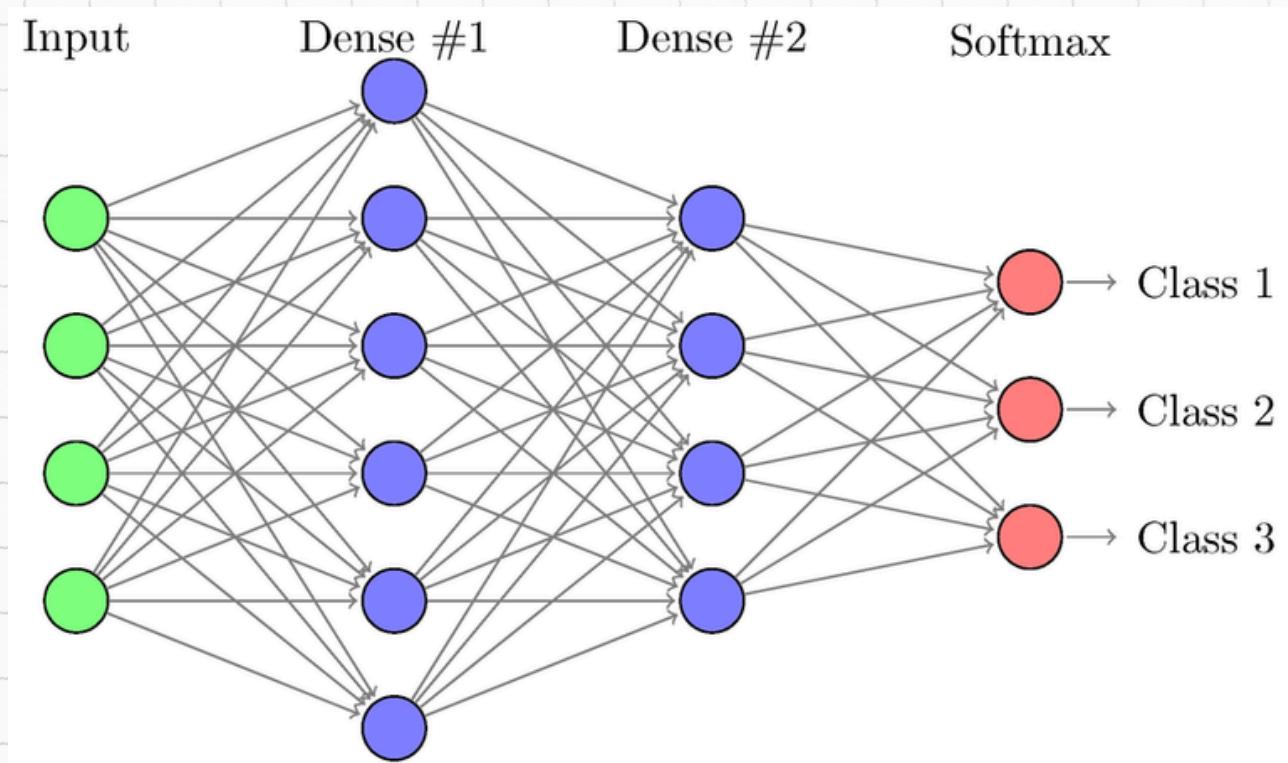
31	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

$2 \times 2$   
pool size

36	80
12	15



# Dense Layer



---

# **Let's build our first CNN !**

---

