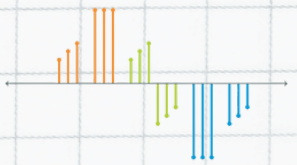
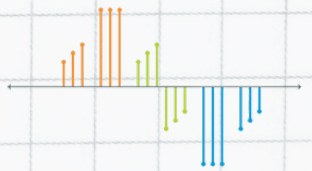
DSP Lecture 2

INTRODUCTION TO DEEP LEARNING

Prepared by: Christian Dave T. Navesis

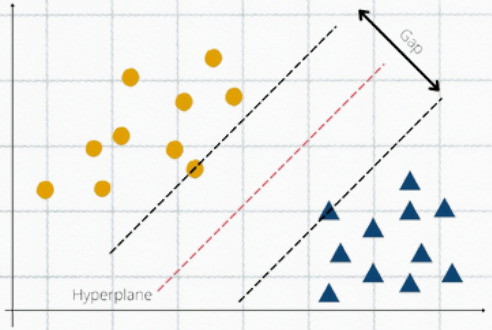


CONTENTS



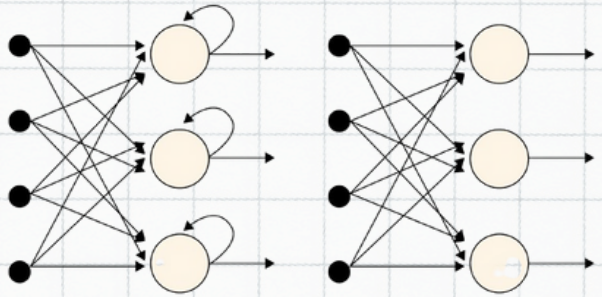
ML VS. DL

Machine Learning (Statistical Models)



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

Deep Learning (Neural Network Models)



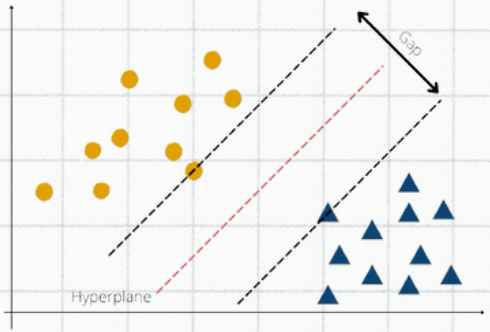
End to end Learning. Can learn the learnable features from raw data. Human intervention is not needed

ML VS. DL

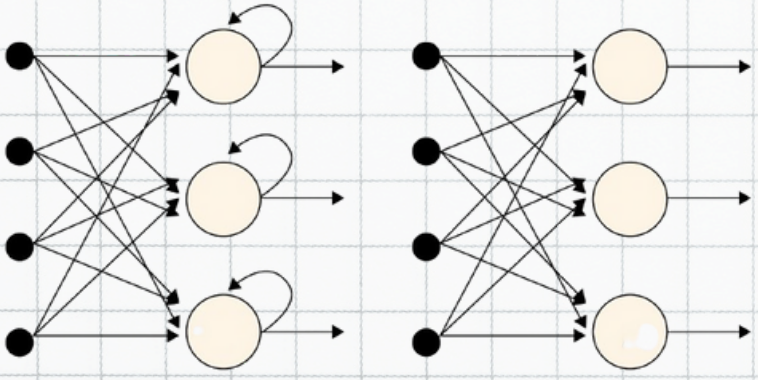
Machine Learning (Statistical Models)

Decision Trees

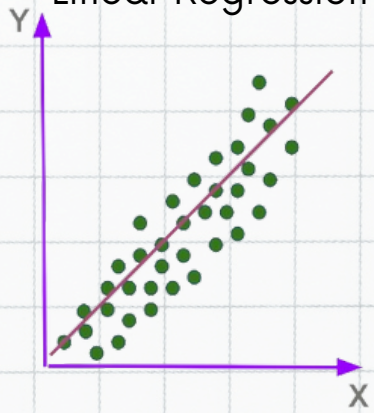
Support Vector Machines



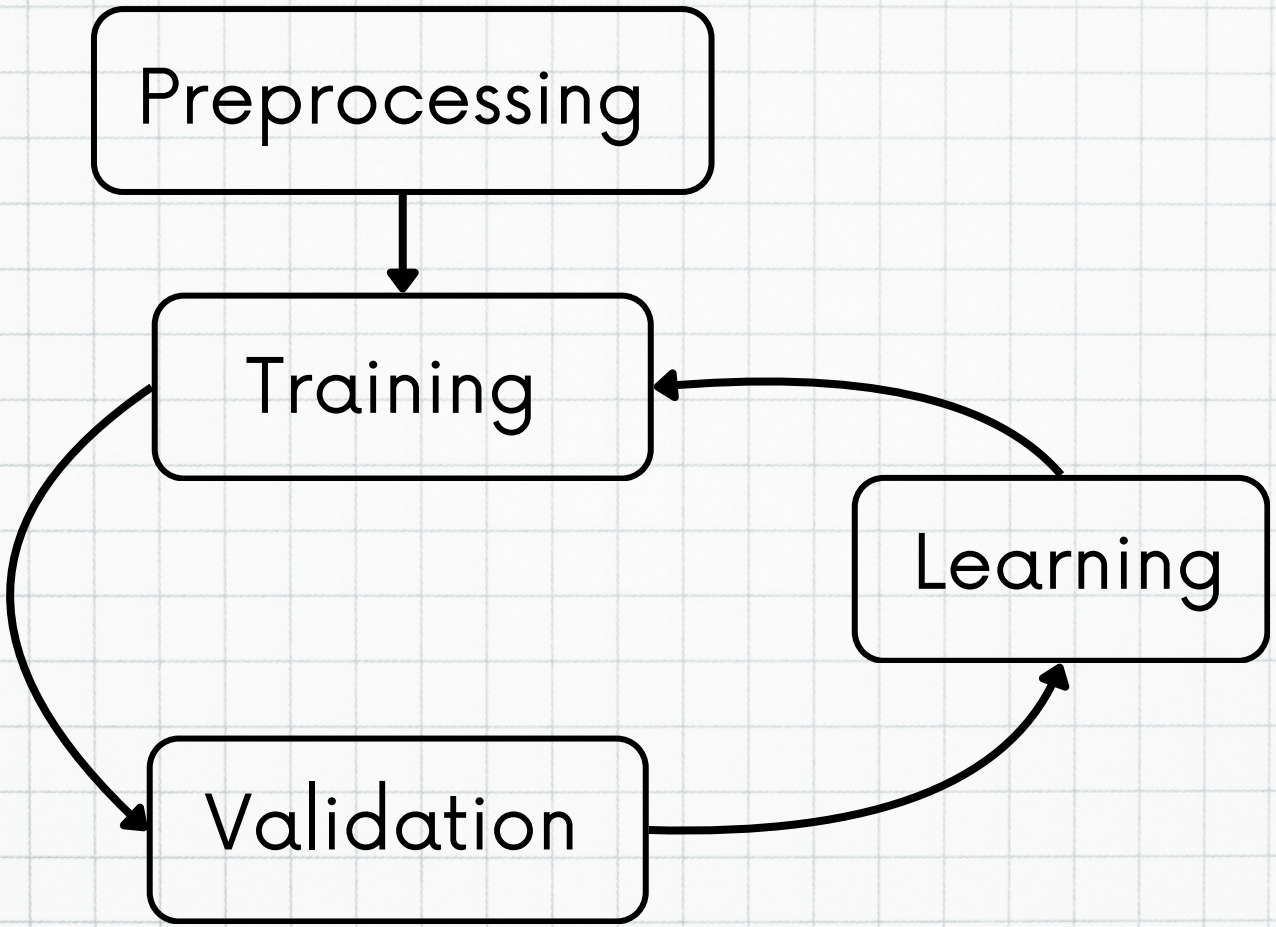
Deep Learning (Neural Network Models)



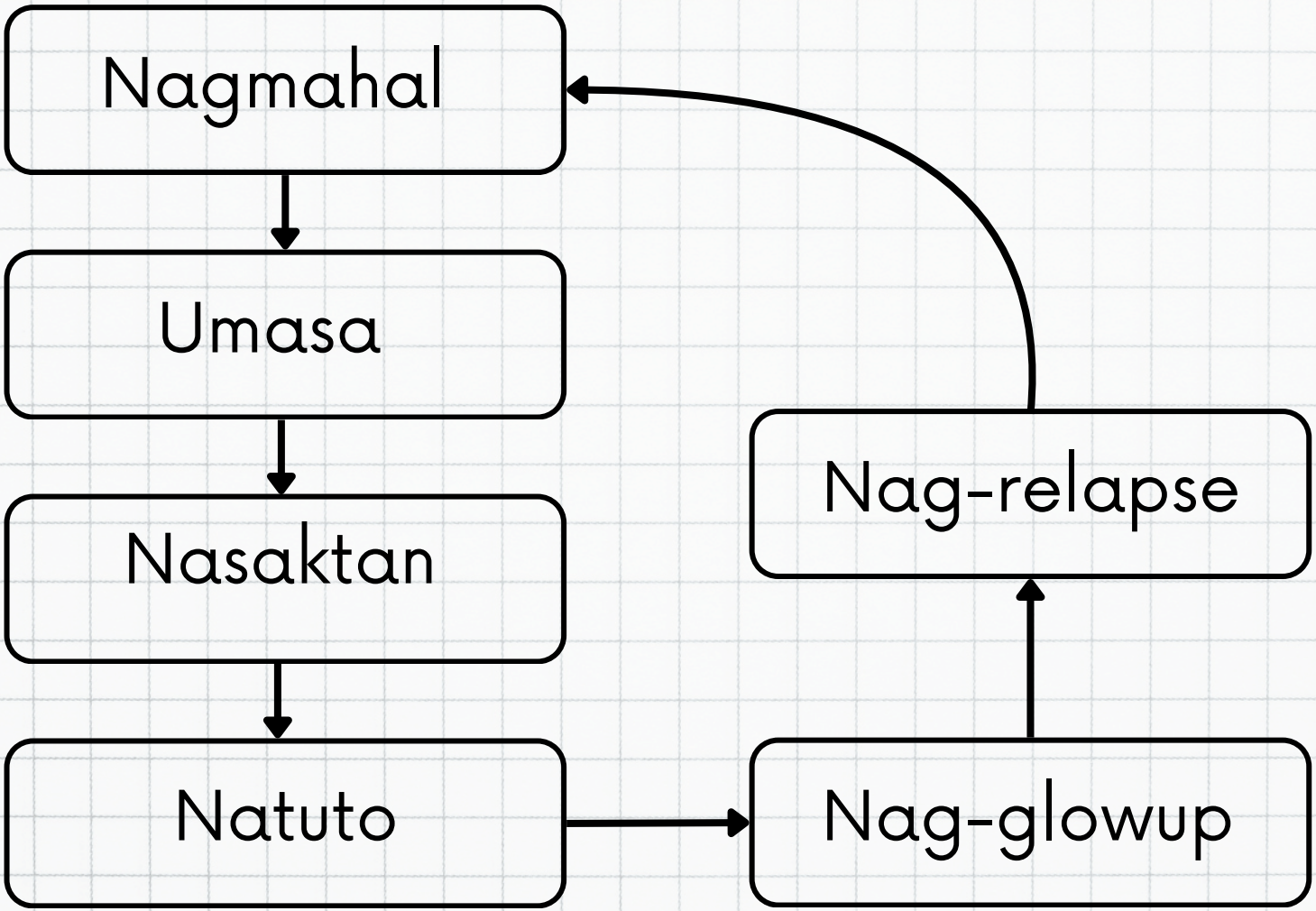
Linear Regression



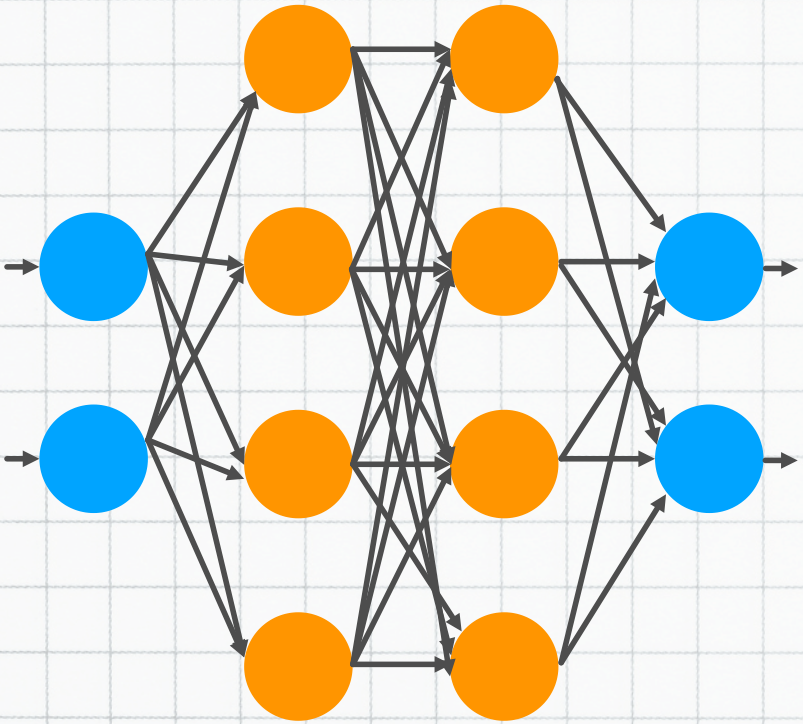
ML/ DL Cycle



ML/ DL Cycle



Neural Networks

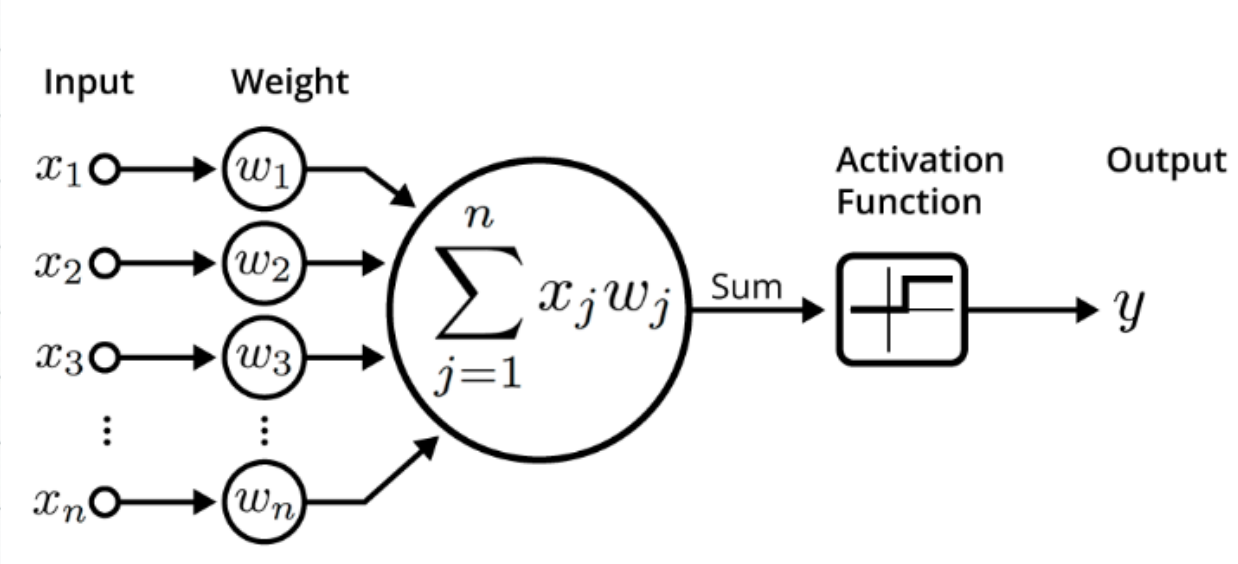


A computational Model that seeks to identify the patterns of a dataset and learn from it. It is inspired by the biological neurons of the brain.

Neural Networks

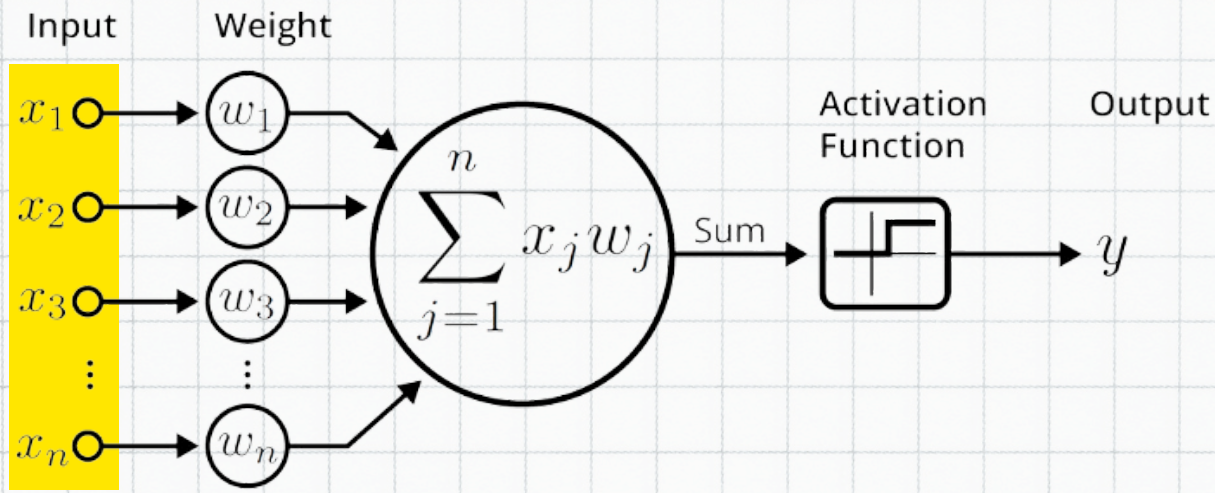
Muffin or Chihuahua?

The Perceptron (Artificial Neuron)



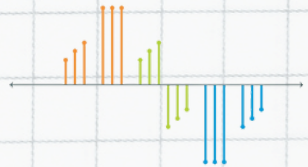
$$y = \sigma\left(\sum_{j=1}^{n} x_j w_j\right) = \sigma(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n)$$

Input

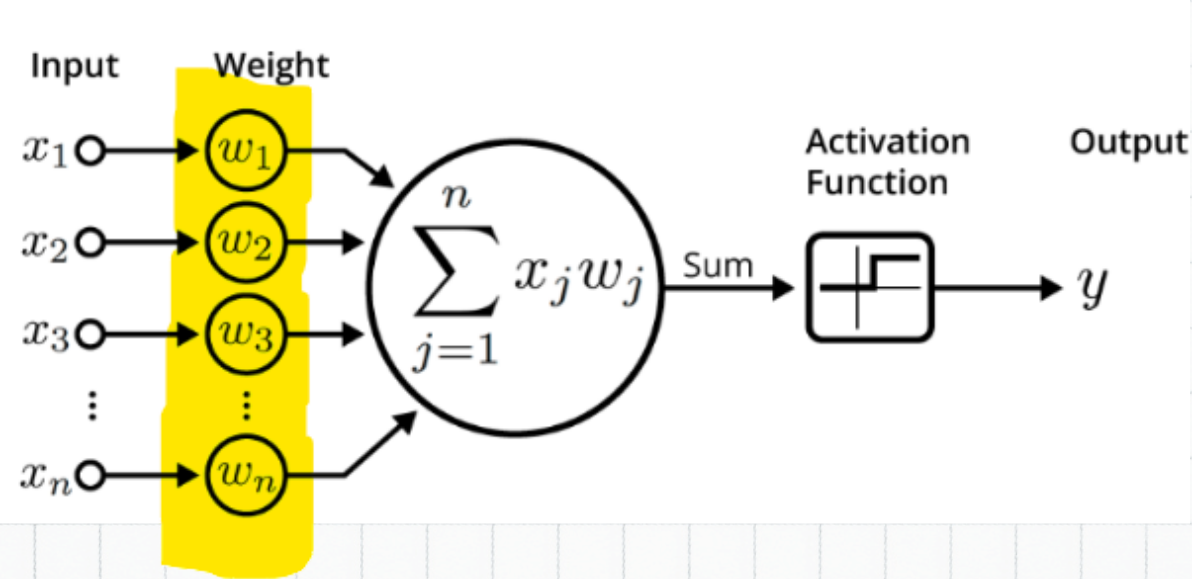


An image, audio signal, or text can be represented as **tensors.**

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



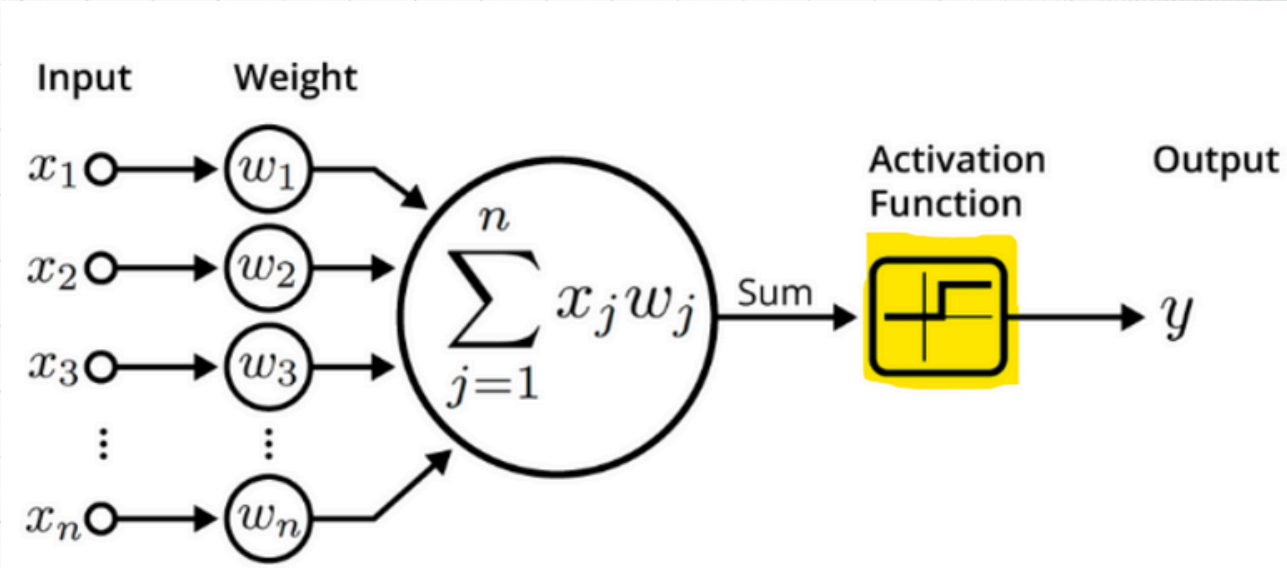
Weights



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

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

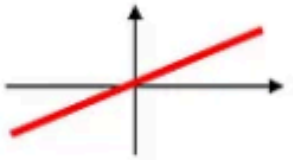
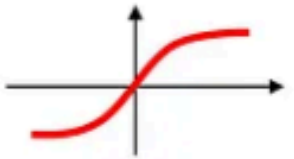
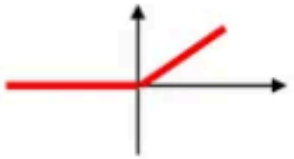
Activation Function

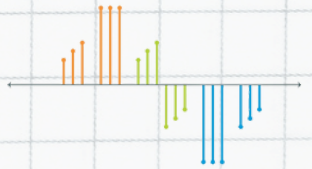


$$y = \sigma\left(\sum_{j=1}^{n} x_j w_j\right) = \sigma(x_1 w_1 + x_2 w_2 + \cdots + x_n w\_n)$$

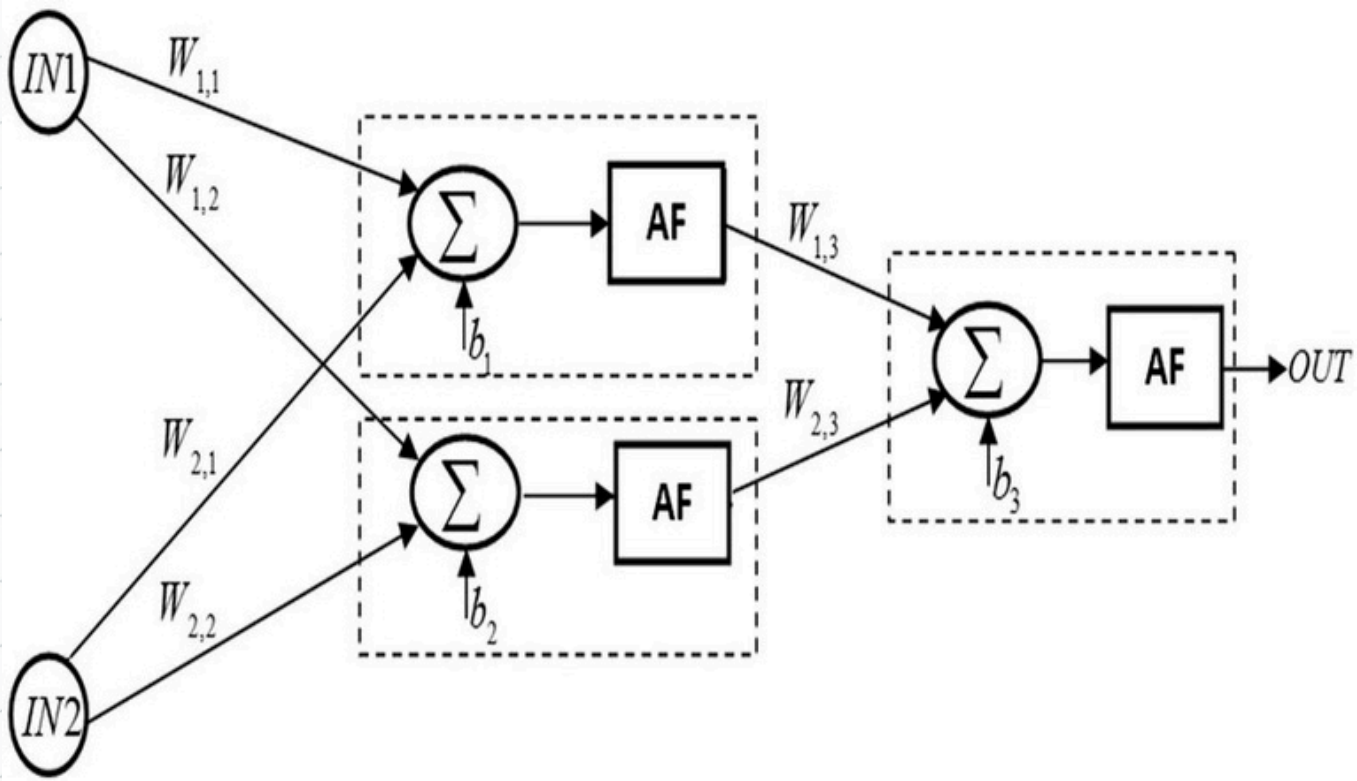
A function that maps the weighted sum of the inputs to an output y.

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	



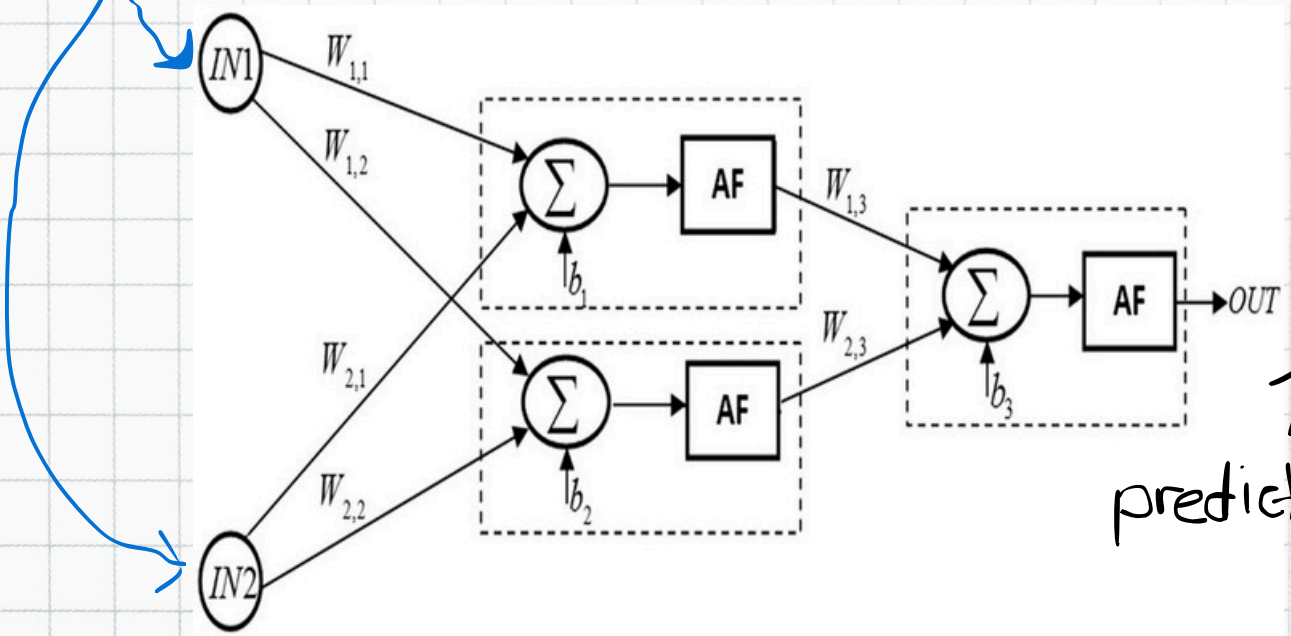
Multilayered Perceptrons



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

Multilayered Perceptrons

Inputs: {X_1,X_2,...X_i}^n	Dataset	Labels: {Y}^n



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

$y \stackrel{?}{=} Y$

predicted expected

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

Loss Function

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

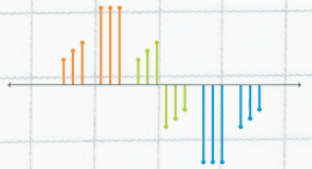
Quantifies how close the prediction y to the real output Y

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

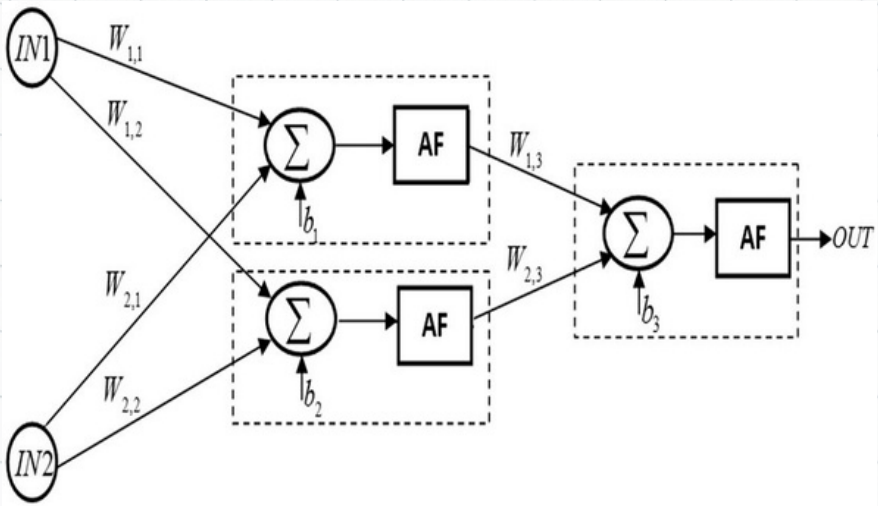
$$J_{MAE}(Y, y) = \frac{1}{n}\sum_{i=1}^{n}|Y_i - y|$$

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

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



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

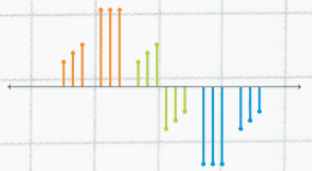
$$\textbf{GOAL: } tweak\ \{w\}\ to\ \textbf{minimize}\ J(Y, y)$$

Gradient Descent

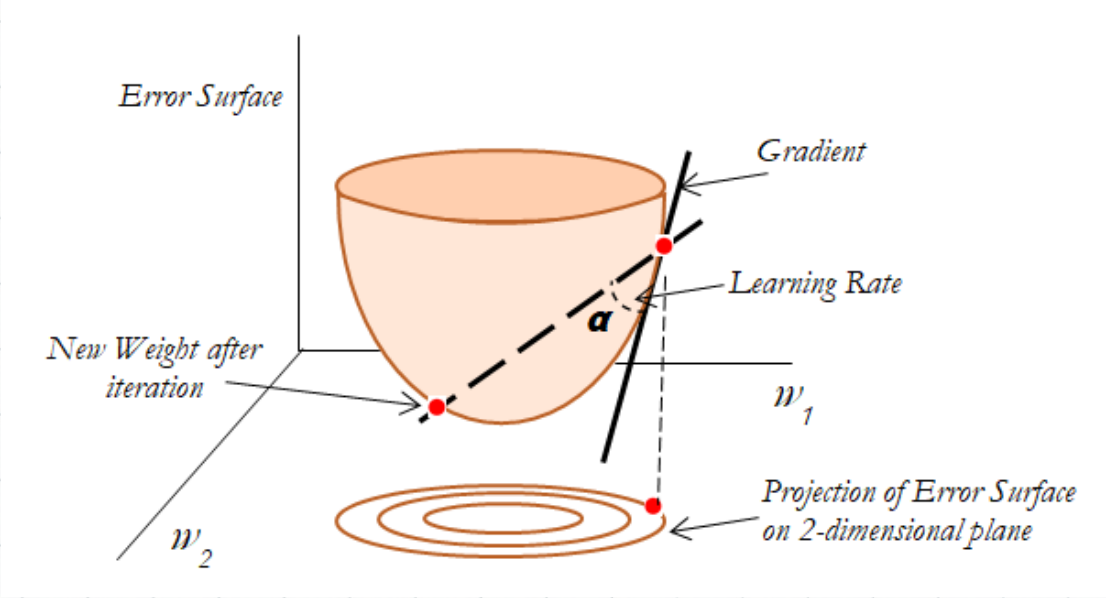
Gradient Descent - An algorithm to find the local minimum of our Loss Function?

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{

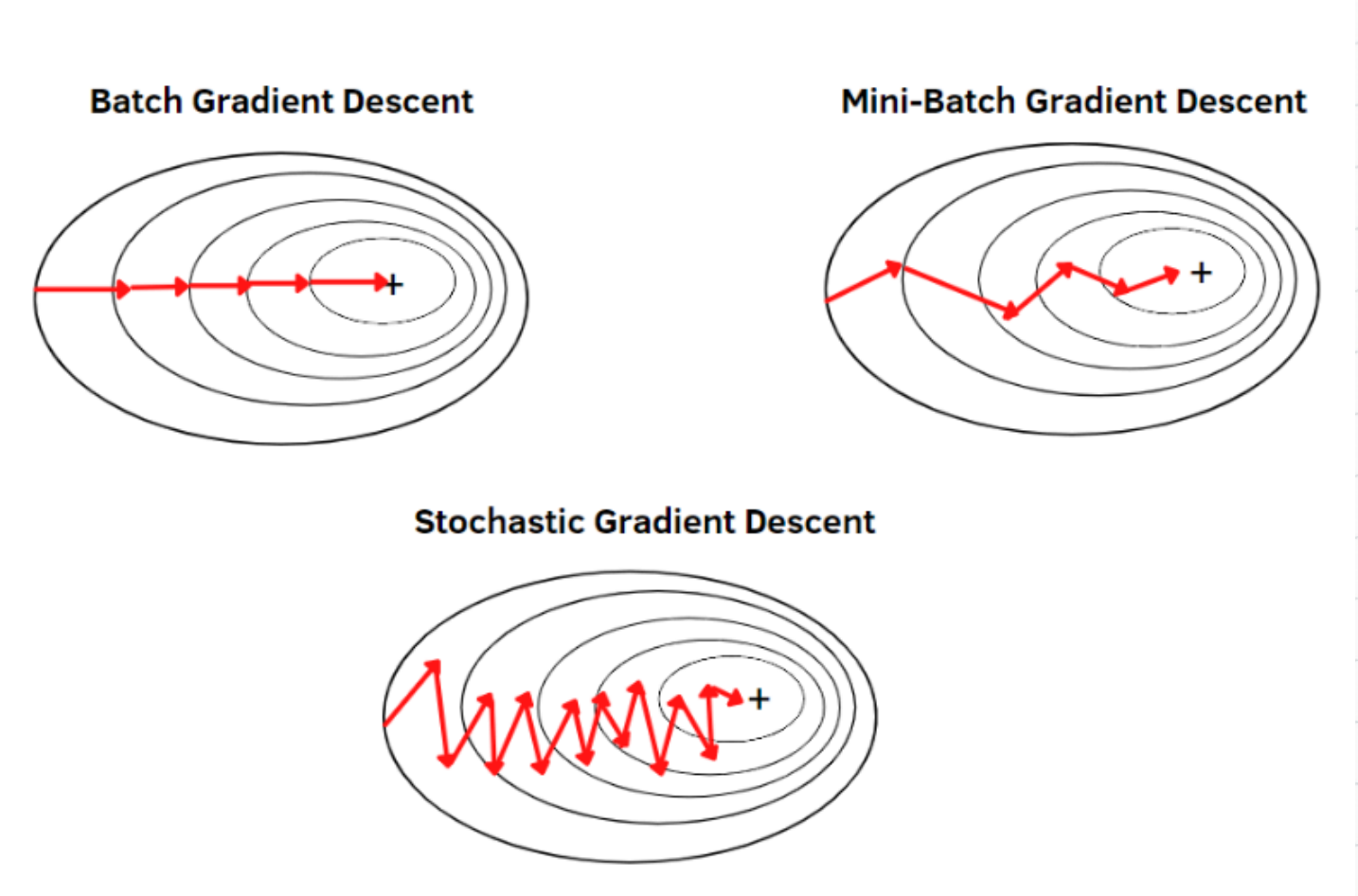
$$w_{i+1} = w_i - \alpha \frac{\partial}{\partial w_i} J(w_i, \{w \backslash w_i\})$$

}

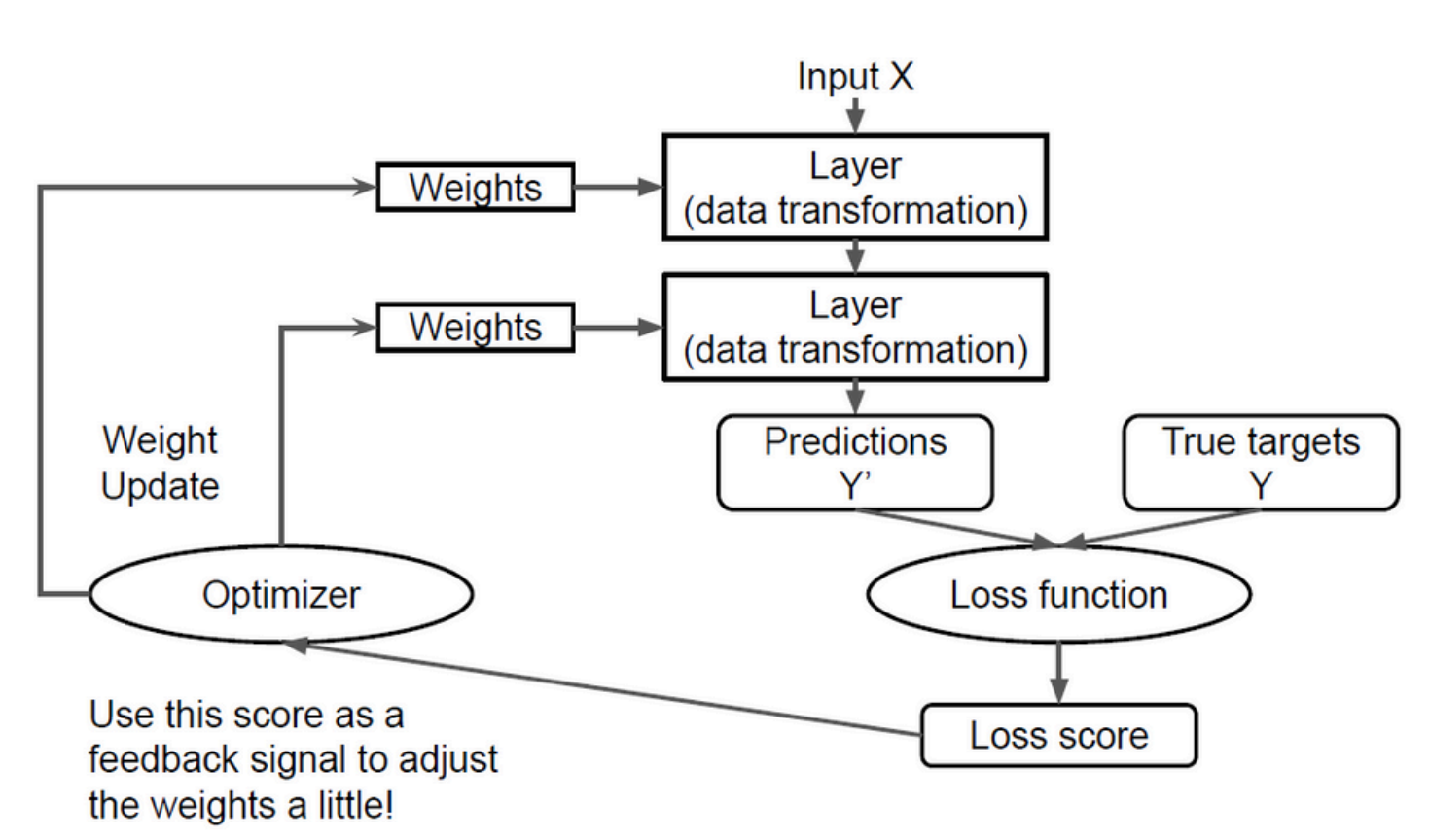
Learning Rate

update term

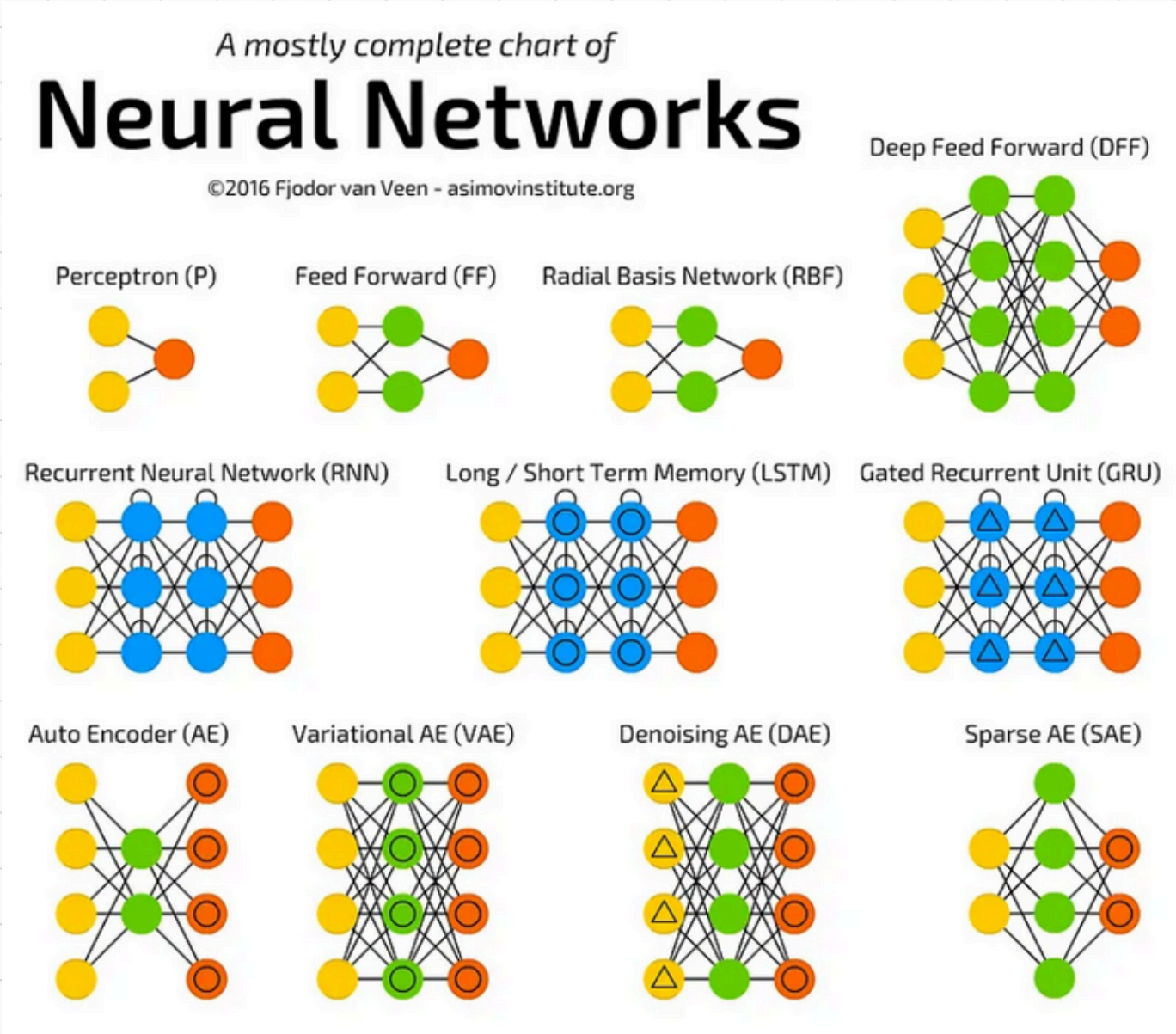
Types of Gradient Descent



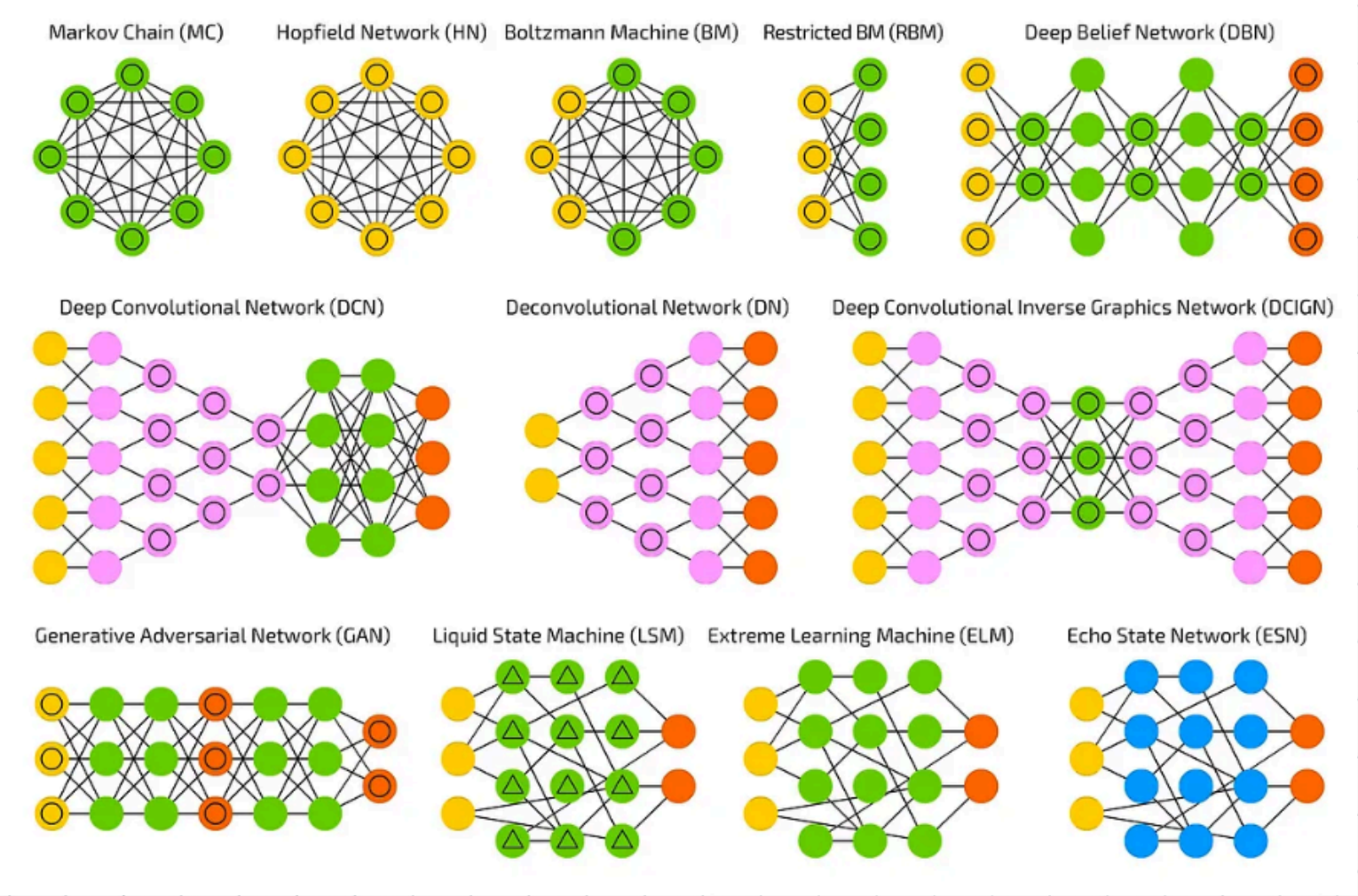
Deep Learning in a nutshell



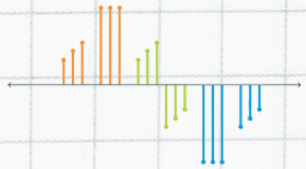
Types of Neural Networks



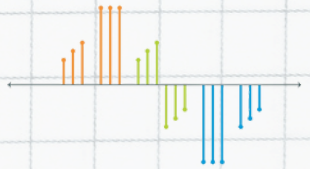
Types of Neural Networks



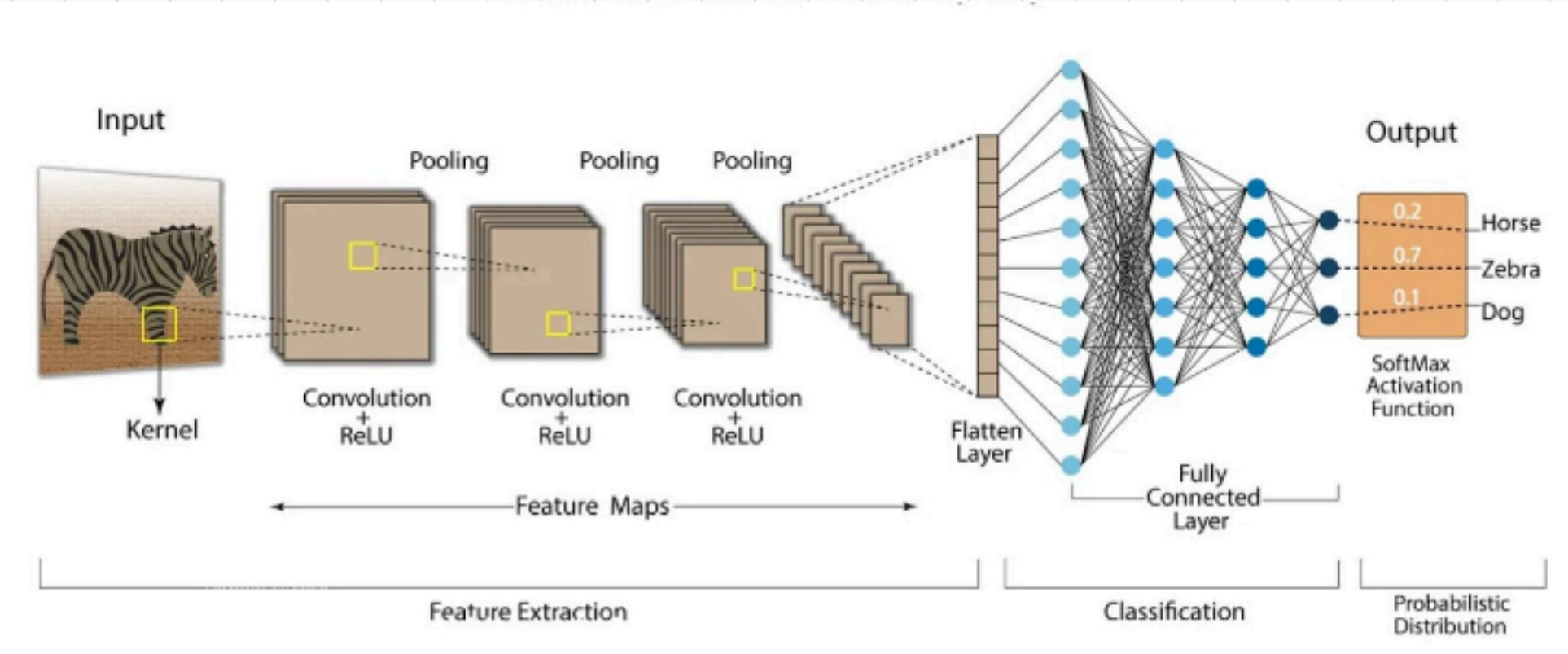
Let's build a Neural Net!



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

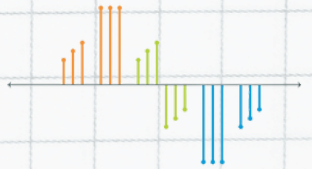
Y-Direction

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

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

X-Direction

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

Y-Direction

Magnitude

10	14	18
9	10	11
4	11	15

Magnitude

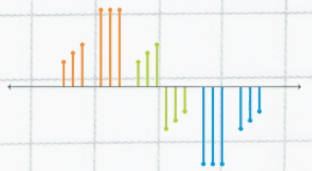
10	14	18
9	10	11
4	11	15

Threshold = 11

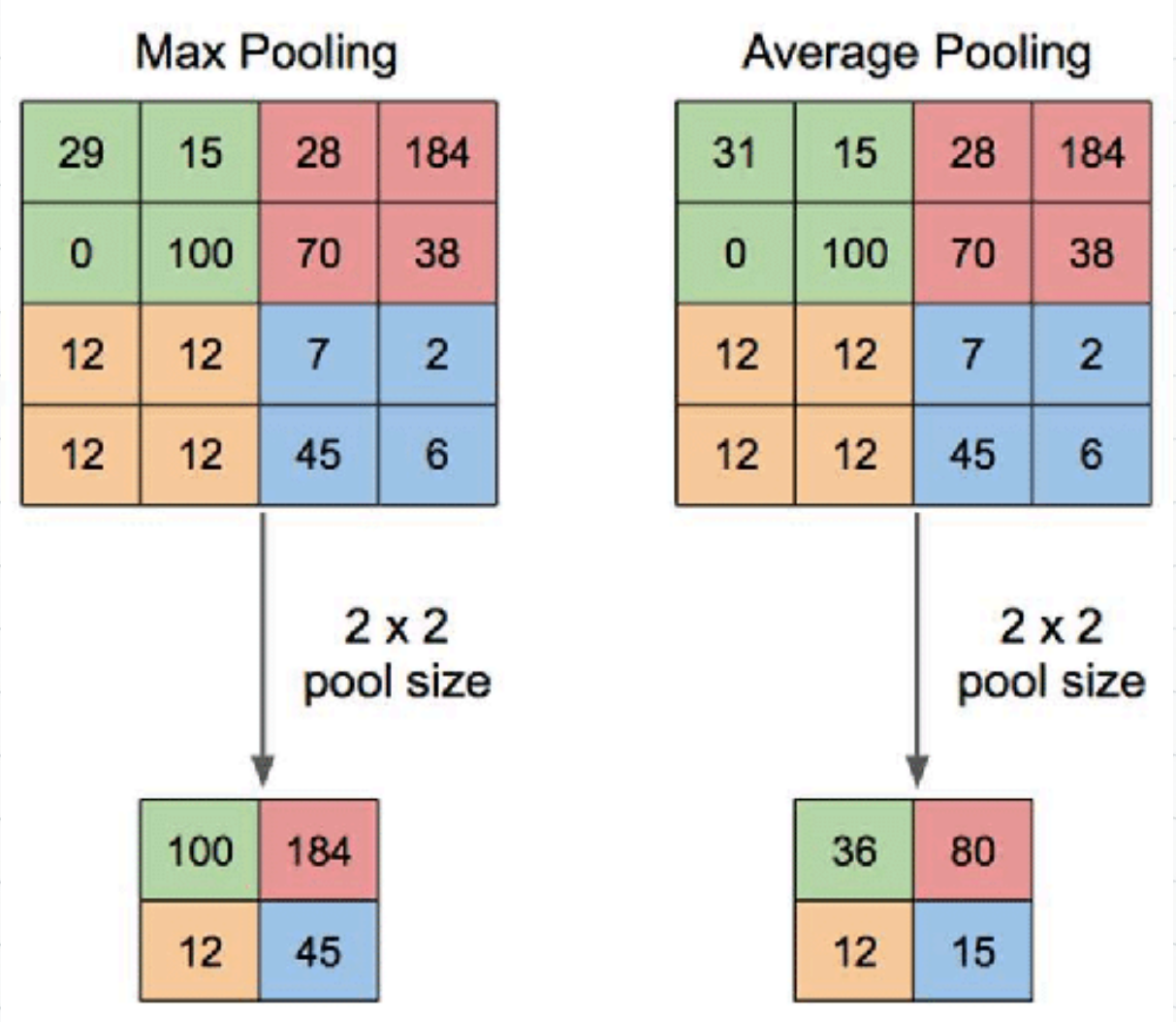
Edge

0	1	1
0	0	0
0	0	1

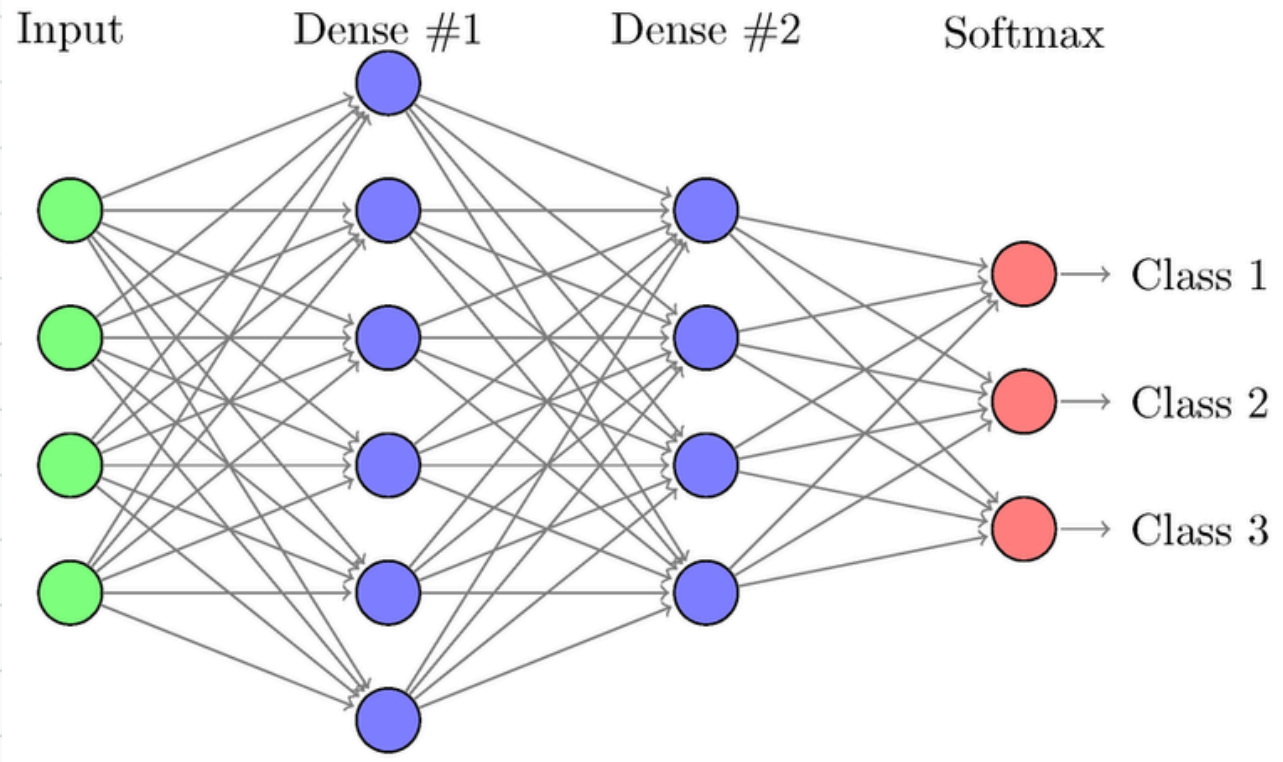
Threshold = (10 + 14 + 18 + 9 + 10 + 11 + 4 + 11 + 15) / 9 = 11



Max/Avg. Pooling



Dense Layer



Let's build our first CNN !