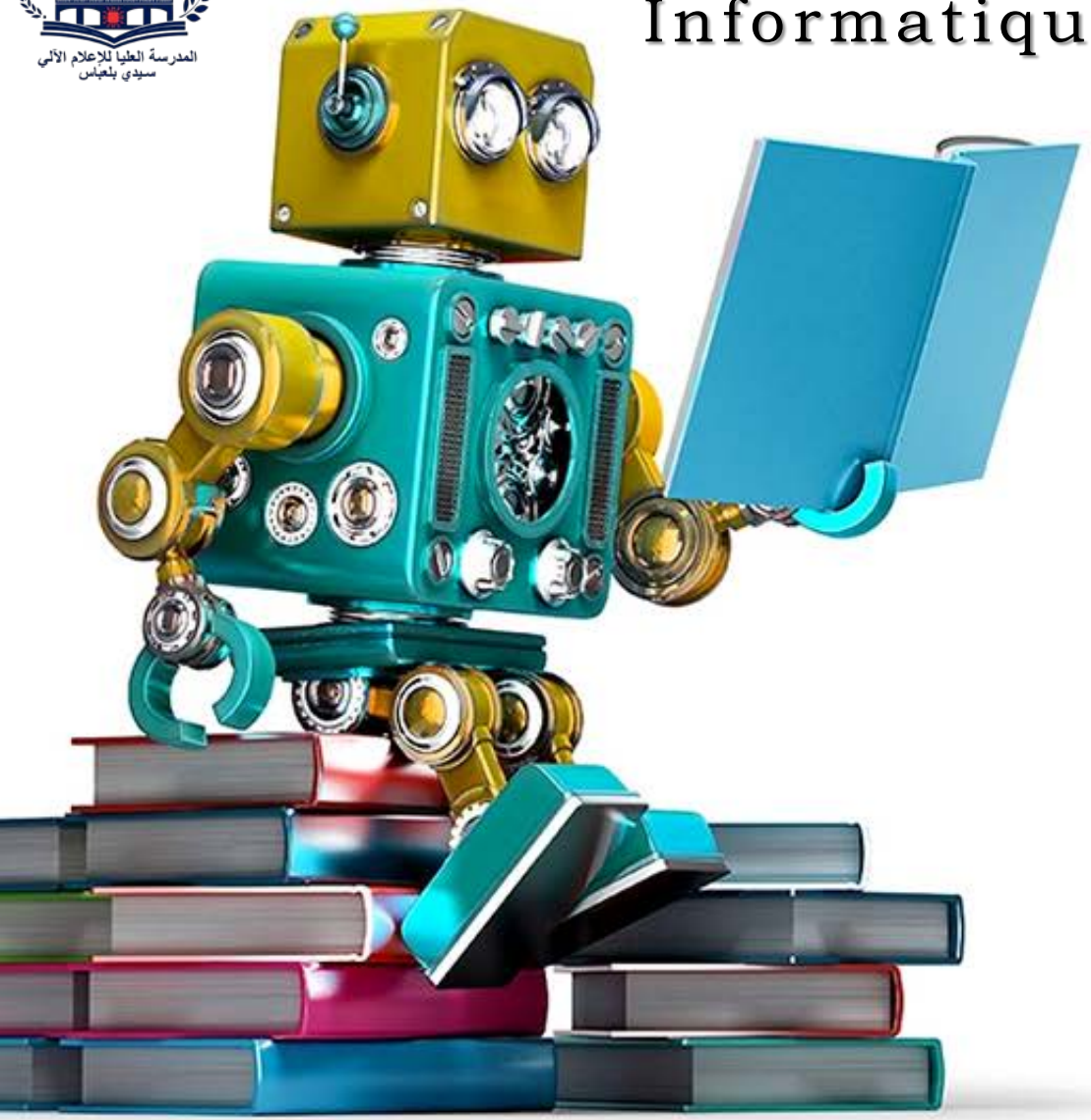
ESI-SBA - École Supérieure en Informatique 08-MAI-1945



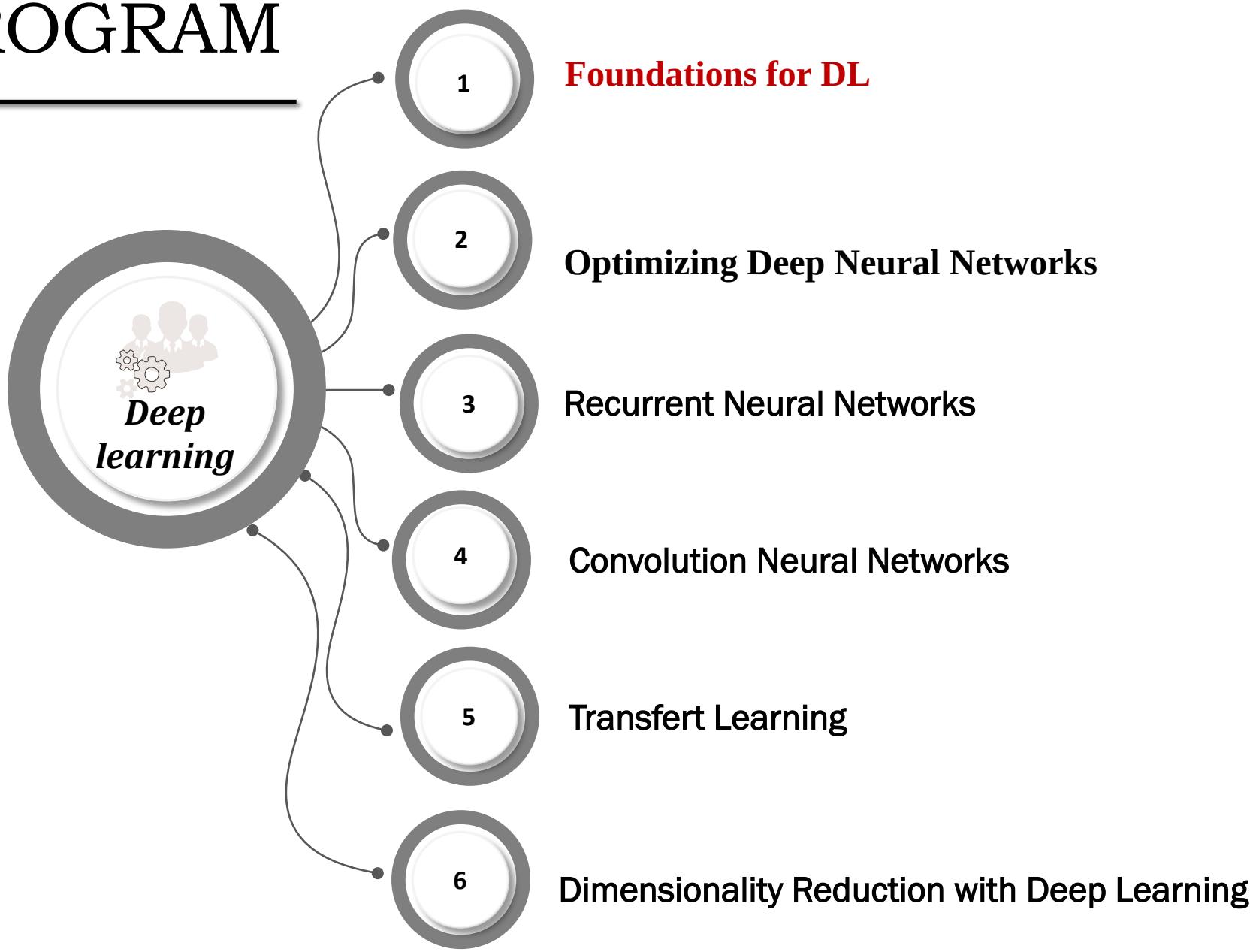
DEEP LEARNING

DR N. DIF

"Just as electricity transformed almost everything 100 years ago, today I actually have a hard time thinking of an industry that I don't think AI (Artificial Intelligence) will transform in the next several years."

Andrew Ng

PROGRAM



Foundations for DL

Optimizing Deep Neural Networks

Recurrent Neural Networks

Convolution Neural Networks

Transfert Learning

Dimensionality Reduction with Deep Learning

REFERENCES

Aggarwal, C. C. (2018). Neural networks and deep learning. *Springer*, *10*(978), 3.

Chollet, F. (2018). *Deep learning mit python und keras: das praxis-handbuch vom entwickler der keras-bibliothek*. MITP-Verlags GmbH & Co. KG.

Book, Deep Learning. "Ian Goodfellow, Yoshua Bengio, Aaron Courville." (2017).

Pattanayak, S., Pattanayak, & John, S. (2017). Pro deep learning with tensorflow (pp. 153-278). New York, NY, USA:: Apress.

Machine Learning Mastery: https://machinelearningmastery.com/

DeepLearningAI : Tensorflow Developer Specialization.

DeepLearningAI : Deep Learning Specialization.

EVALUATION

TOTAL = (EXAM 1 + EXAM 2 + CC)/3.

CC = Assiduity + Preparation + Projet.

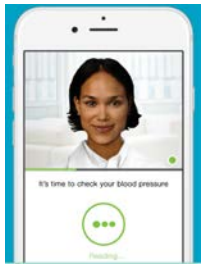
1. Introduction to Deep Learning

In the past few years, artificial intelligence (AI) has been a subject of many applications.

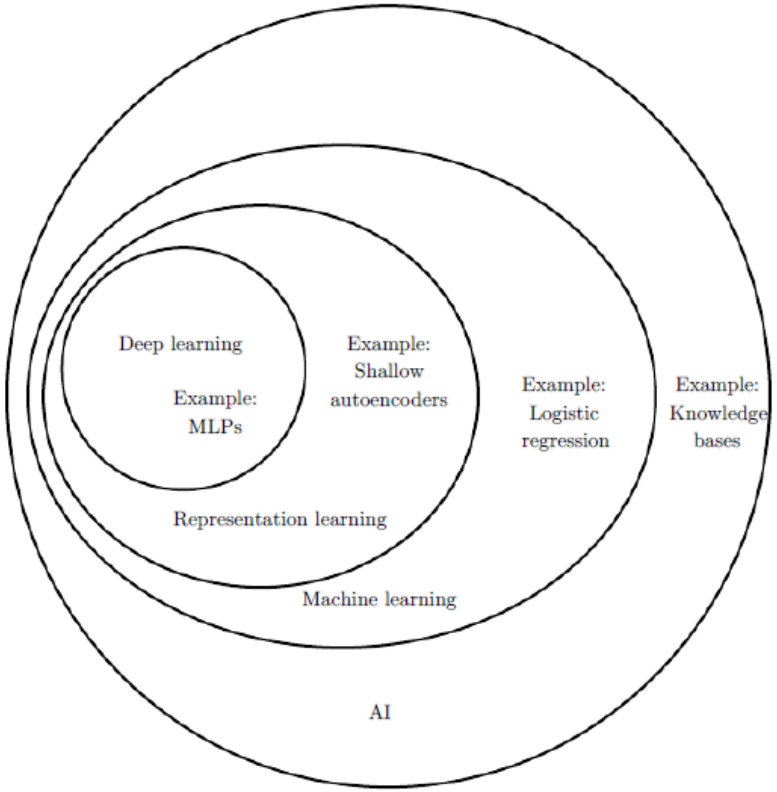
Machine learning, deep learning, and AI come up in countless articles, often outside of technology-minded publications.

Intelligent chatbots (ChatGPT)

Virtual assistants



Self-driving cars



1. Introduction to Deep Learning

ChatGPT: Optimizing Language Models for Dialogue

ChatGPT is a model that was trained to interact in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, and reject inappropriate requests.

The technology itself **isn't groundbreaking**. GPT stands for "generative pre-trained transformer," which is an autoregressive language model that uses deep learning to produce human-like speech

https://www.youtube.com/watch?v=aCnPjtOtDlg

1. Introduction to Deep Learning

Deep Fake

A video of a person in which their face or body has been digitally altered so that they appear to be someone else, typically used maliciously or to spread false information.

REFACE

Reface's deepfake effects are powered by a class of deep learning models known as GANs . it generates a new animated face using the twin inputs (the selfie and the target video or image), rather than trying to mask one on top of the other

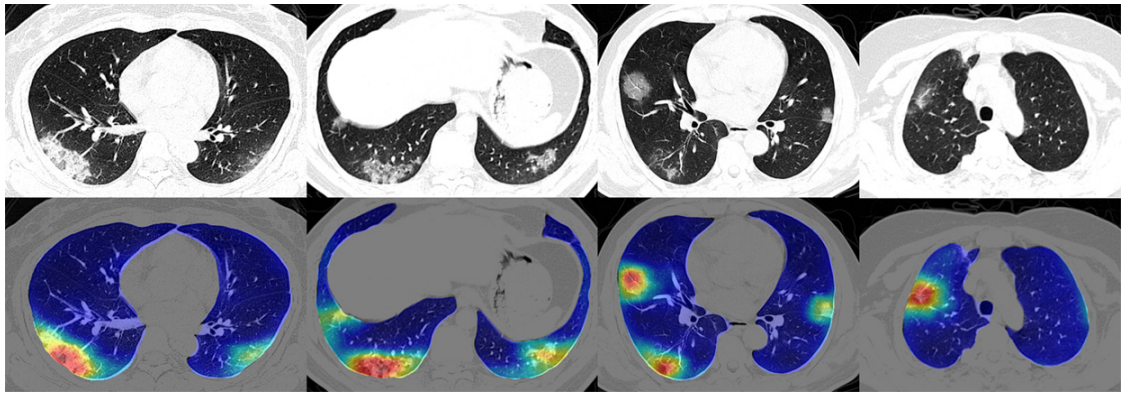
1. Introduction to Deep Learning

Google Dream: create synthetic images from real ones (GAN).

1. Introduction to Deep Learning

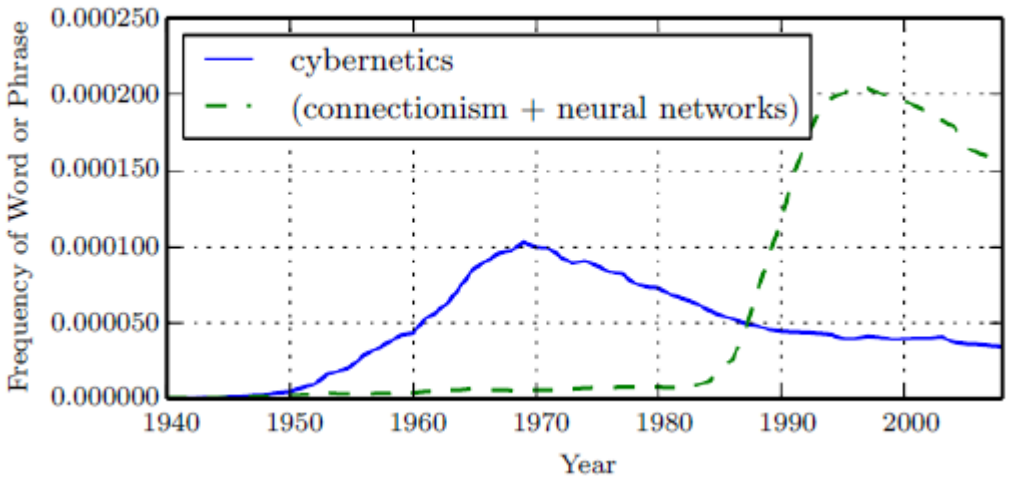
Masque detection with YOLO

Gradient weighted Class Activation Mapping (**Grad-CAM**) to visualize activated features during covid-19 detection from CT-scans

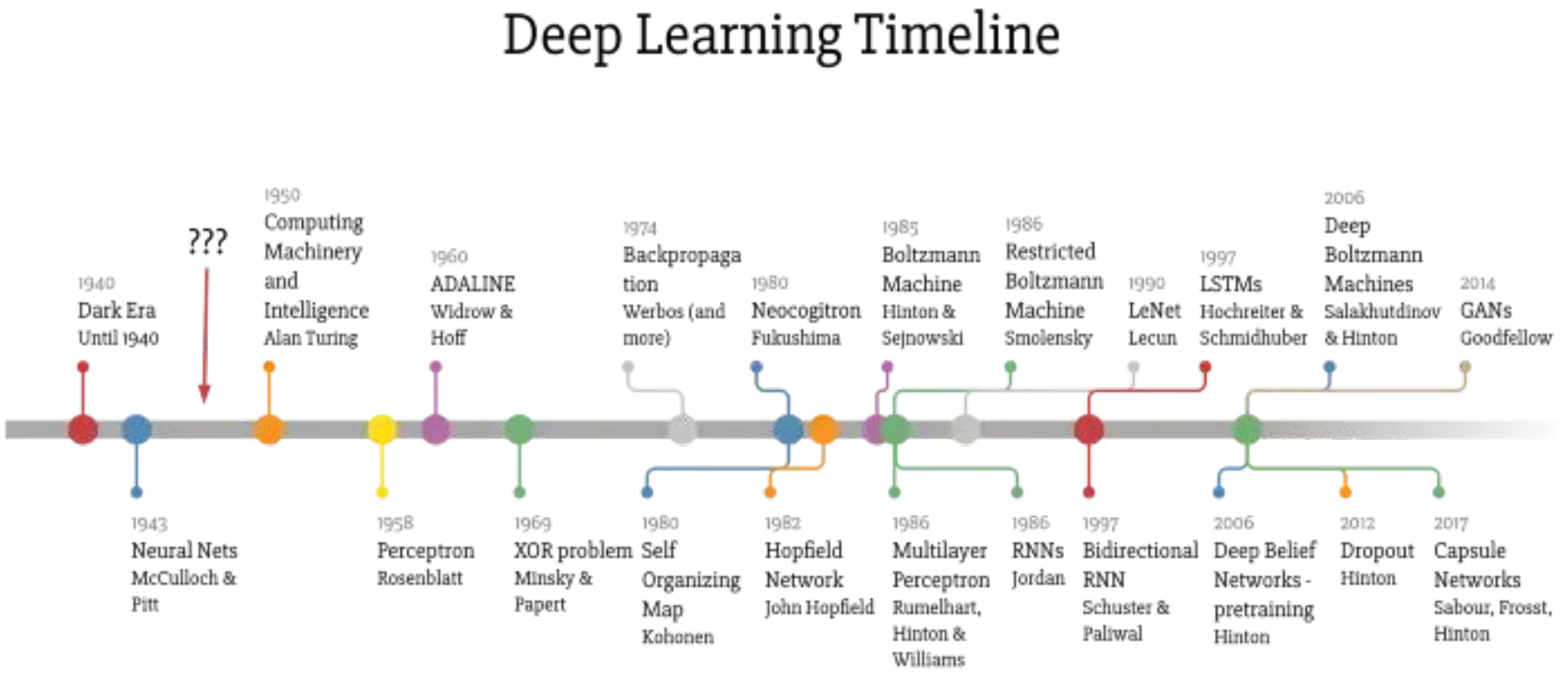


1. Introduction to Deep Learning

- Deep learning only appears to be new. In fact, it dates back to the 1940s, because it was relatively unpopular for several years preceding it current popularity, and because it has gone though many different names and recently become called “deep learning”.
- There has been three waves of development of deep learning : 1940-1960 (cybernetics), 1980s- 1990s (connectionism), and deep learning beginning from 2006.



1. Introduction to Deep Learning



Source : https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a

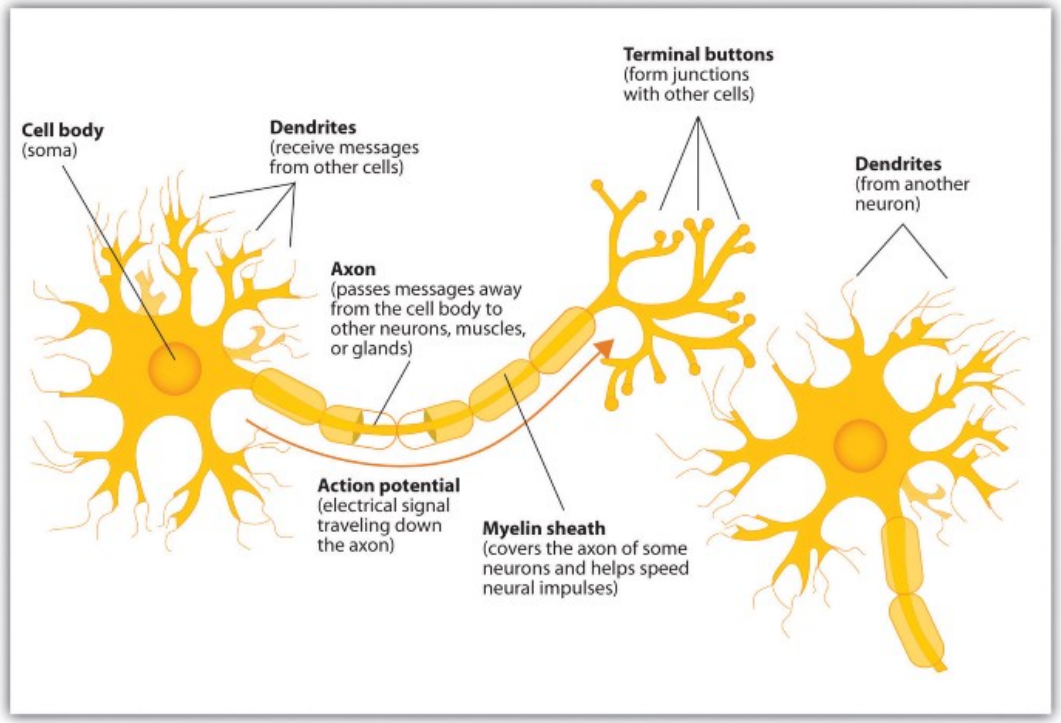
2. Perceptron and Supervised Learning with Neural Networks

The Story Started From 1943.

1.1. McCulloch-Pitts Neuron (MCP)

The first step towards the *perceptron* we use today was taken in **1943 by McCulloch and Pitts [1]**, by mimicking the functionality of a biological neuron. They proposed the first computational model of a neuron.

Basically, a neuron takes an input signal (dendrite), processes it (soma), passes the output through a cable like structure to other connected neurons (axon to synapse to other neuron's dendrite)—takes an input, processes it, throws out an output.



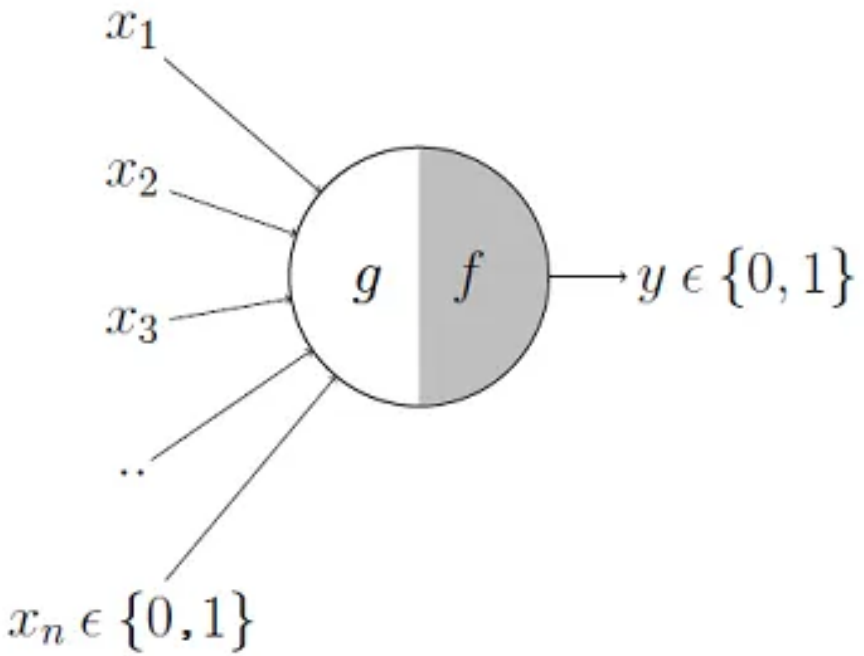
2. Perceptron and Supervised Learning with Neural Networks

1.1. McCulloch-Pitts Neuron (MCP)

- g takes an input (dendrite), performs an aggregation. It Aggregates the weighted inputs into single numeric value, and based on the aggregated value the second part, f makes a decision.
- θ presents a thresholding parameter, if the output is greater than the threshold than the neuron will fire.
- w_i are weights that could be set by human operator.

$$g(x_1, x_2, \ldots, x_n) = \mathrm{g(x)} = \sum_{i=1}^{n} w_i x_i$$

$$y = f\big(g(x)\big) = \begin{cases} 1 \; if \; g(x) \geq \theta \;\; \boldsymbol{active\; neurone} \\ 0 \; if \; g(x) < \theta \;\; \boldsymbol{silent\; neurone} \end{cases}$$



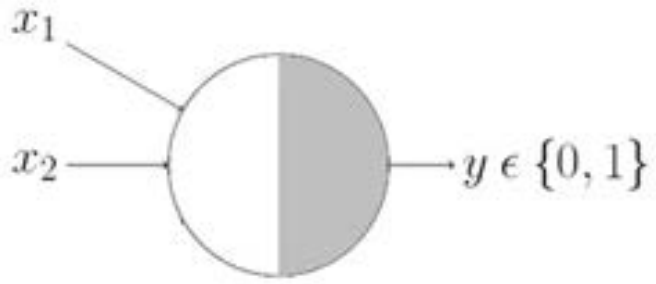
2. Perceptron and Supervised Learning with Neural Networks

1.1. McCulloch-Pitts Neuron (MCP)

We consider that $w_i = 1$.

AND Function

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



An AND function neuron would only fire when ALL the inputs are ON i.e., $\boldsymbol{g(x)} \geq 2$ here.

OR Function

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

An OR function neuron would fire if ANY of the inputs is ON i.e., $\boldsymbol{g(x)} \geq 1$ here.

2. Perceptron and Supervised Learning with Neural Networks

Sorry, I Cant Solve Non Linear Problems.

1.1. McCulloch-Pitts Neuron (MCP)

AND and OR are linear functions. What about **XOR** witch is a non linear function ?

XOR Function

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Minsky and Papert wrote a book called Perceptrons in 1969. Although the book said many other things, the only thing most people remembered about the book was that: “A linear classifier cannot learn an XOR function.” • Because of that statement, most people gave up working on neural networks from about 1969 to about 2006.

No way to fix a threshold !

2. Perceptron and Supervised Learning with Neural Networks

1.1. McCulloch-Pitts Neuron (MCP) : limitations

Limitations

- MCP neuron can not treat non-boolean inputs.
- The threshold is manually changed.
- Weights are equal and set manually.

2. Perceptron and Supervised Learning with Neural Networks

2.1. Perceptron

Overcoming the limitations of the M-P neuron**, Frank Rosenblatt [2],** an American psychologist, proposed the classical perception model, in 1958. It is more generalized computational model than the McCulloch-Pitts neuron where weights and thresholds can be learnt over time.

The learning process was inspired by the Hebbian theory of synaptic plasticity (i.e. the adaptation of brain neurons during the learning process).

Rosenblatt's perceptron can handle only classification tasks for linearly separable classes (The XOR problem is always present).

2. Perceptron and Supervised Learning with Neural Networks

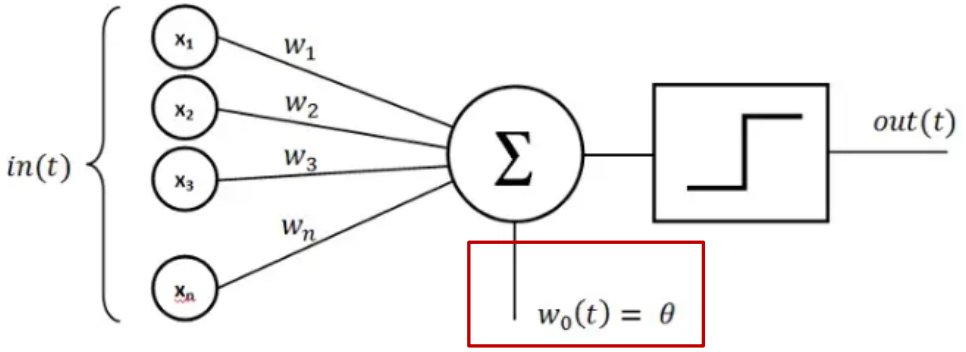
2.1. Perceptron

The perceptron has some major differences, compared to MCP :

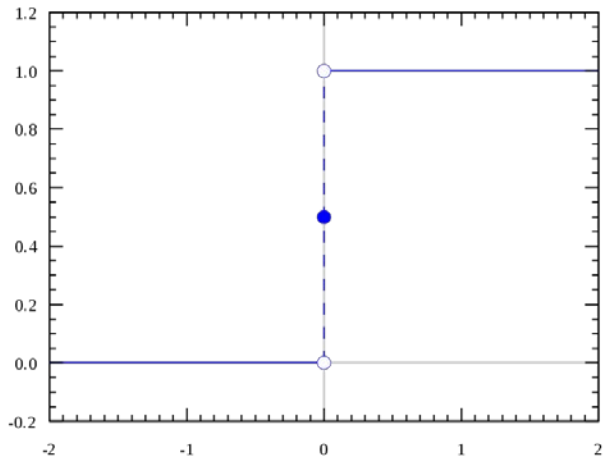
1. The neuron takes an extra constant input associated with a synaptic weight b (bias). It is the negative of the activation threshold.
2. The synaptic weights w_k are not restricted to unity, thus, it allows weighting.

$$g(W^T X + b) = \begin{cases} 1 \; if \; W^T X + b \geq 0 \\ 0 \; otherwise \end{cases}$$

g corresponds to the **activation function** : **Heaviside**



Artificial neuron used by the perceptron. From Wikipedia.



Heaviside function
Source : Wikipedia

2. Perceptron and Supervised Learning with Neural Networks

2.1. Perceptron

Given a set of M examples (x_m, y_m), how can the perceptron learn the correct synaptic weights w and bias b to correctly separate the two classes.

Weights vector : W = $[w_1, w_2, \ldots, w_n]$,
n is the number of features.
m : number of samples.
r : learning rate $\in [0,1]$

Perceptron Learning Algorithm

Input: Training examples $\{x_i, y_i\}_{i=1}^{m}$
Initialize W and b randomly
For i=1 to L do
For j=1 to m do

$$error = y_j - g\left(W^T x_j + b\right)$$
$$W = W + r \times error \times x_j$$
$$b = b + r \times error$$

Output : Set of weights w and bias b for the perceptron

2. Perceptron and Supervised Learning with Neural Networks

2.1. Perceptron : example

Consider the following training set:

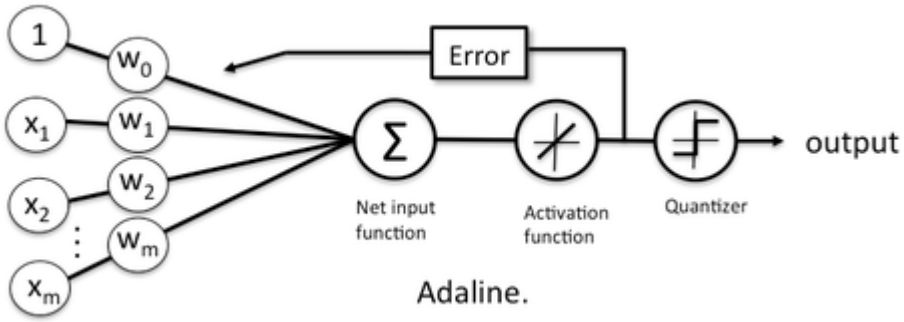
x_i	y_i
$[3, 2, 1]$	0
$[1, 1, 1]$	1
$[1, 2, 3]$	1

1- Manually simulate the perceptron algorithm on this data in one iteration in the order from top to bottom. By initializing the learning rate r to 0.1, the weights vector W = [0,0,0], and the bias b to 0.5.

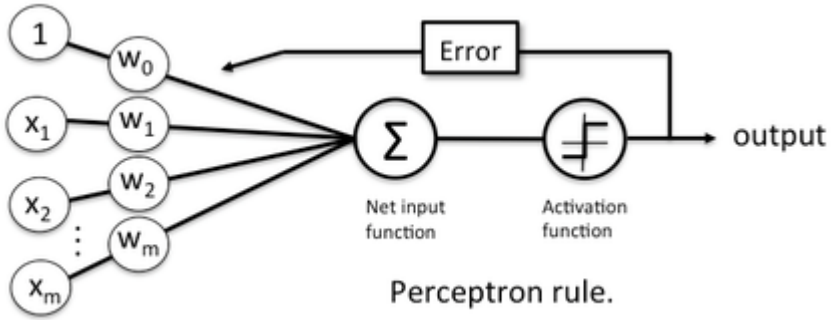
2. Perceptron and Supervised Learning with Neural Networks

2.2. Adaline (Adaptive Linear Element)

- It was developed by Bernard Widrow and Marcian Hoff [4].
- The perceptron uses the Heaviside activation function, whereas, the Adaline uses a linear activation function.
- Adaline uses continuous predicted values (from the net input) to learn the model coefficients, which is more “powerful” since it tells us by “how much” we were right or wrong.
- SGD cant be used in the perceptron, the derivative of **the error function is always 0**.
- This weight update in Adaline is basically just taking the “opposite step” in direction of the sum-of-squared error cost gradient (SGD).



Linear activation function F(x)=x

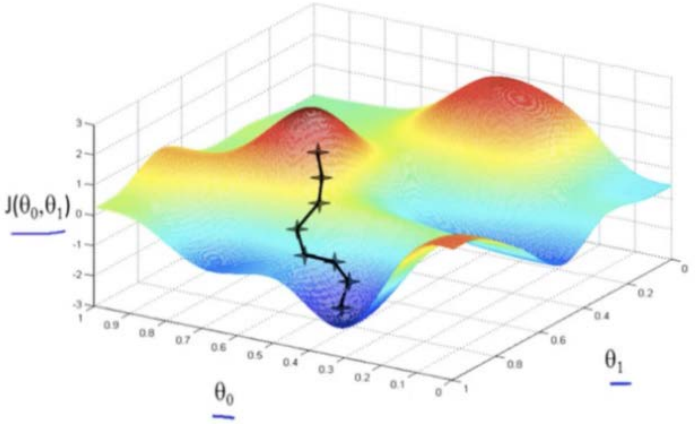


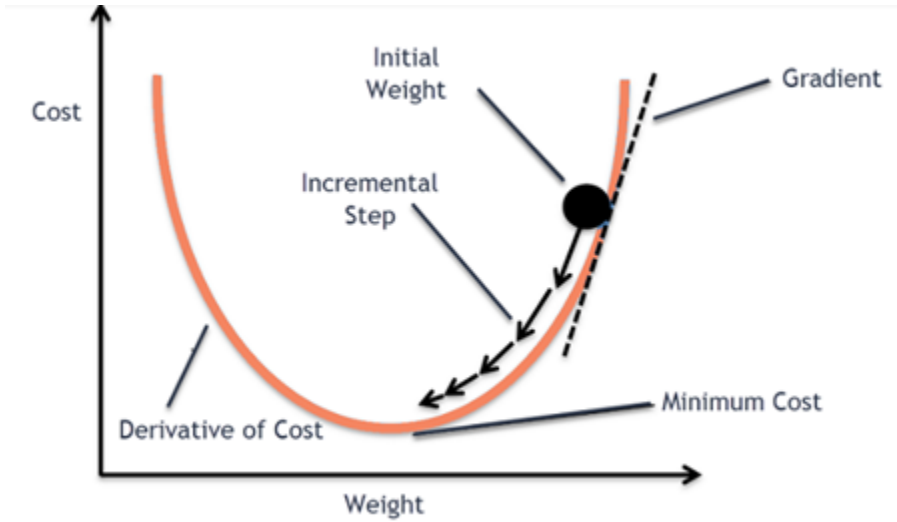
Heaviside activation function

REMINDER

Gradient descent is an iterative stochastic optimization algorithm that helps to find the minimum of a function.

What is the inspiration of gradient descent? Why using partial derivative? What is a local and a global minimum?





MSE cost function

2. Perceptron and Supervised Learning with Neural Networks

2.2. Adaline (Adaptive Linear Element)

ADALINE Algorithm

Input: Training examples $\{x_i, y_i\}_{i=1}^{m}$
Initialize W and b randomly
For l=1 to L do
For i=1 to m do

$$W_j^{(t)} = W_j^{(t-1)} + \alpha * (y^{(i)} - F(x^{(i)}))X_j^{(i)}$$

$$b = b + \alpha * (y^{(i)} - F(x^{(i)}))$$

Output : Set of weights w and bias b for the perceptron

This weight update in Adaline is basically just taking the "opposite step" in direction of the sum-of-squared error cost gradient

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}[y^{(i)} - F(x^{(i)})]^2$$

J *is the cost function and* n *and* $\frac{\partial}{\partial w}\mathrm{J}(w)$ is the partial dever*ative on* w_j, α is the learning rate.

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}\mathrm{J}(w), j\epsilon\ (1,2,\dots,n)$$

$$w_j = w_j - \alpha(y^{(i)} - F(x^{(i)}))(-X_j^{(i)})$$

$$b = b - \alpha * (y^{(i)} - F(x^{(i)}))$$

For one sample

$$w_j = w_j - \alpha * (y^{(i)} - F(x^{(i)}))(-X_j^{(i)})$$

$$b = b - \alpha * (y^{(i)} - F(x^{(i)}))(-1)$$

2. Perceptron and Supervised Learning with Neural Networks

Minsky and Papert critic

- Minsky and Papert [3] proved that a single-layered perceptron is unable to classify non-linear patterns. its classificatory capacities are limited to patterns that are linearly separable (XOR function).

- The classical perceptron had only one layer of neurons. A single-layered neural net cannot compute much, and more layers are needed to perform more complex operations.

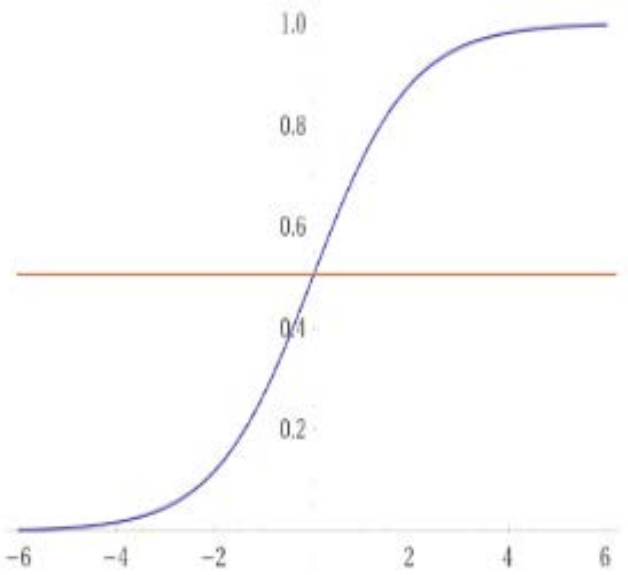
- Minsky and Papert's book was often *interpreted* as *implying* that, even with multi-layered perceptrons, the problem of linear separability could not be solved.

- Many connectionist AI researchers discussed the learning difficulties of multi-layer perceptrons.

REMINDER

Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

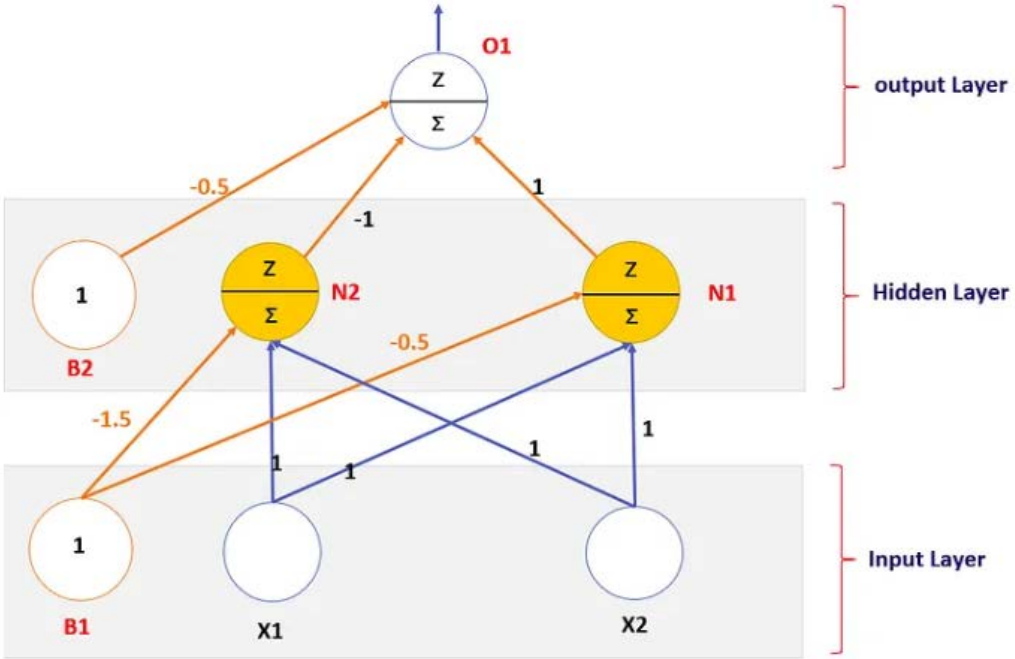


2. Perceptron and Supervised Learning with Neural Networks

2.3. Multi Layer Perceptron

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

h1 node : (x2 **and not** x1) .
h2 node : (x1 **and not** x2).
y node : (h1 **or** h2).
Activation function : Heaviside

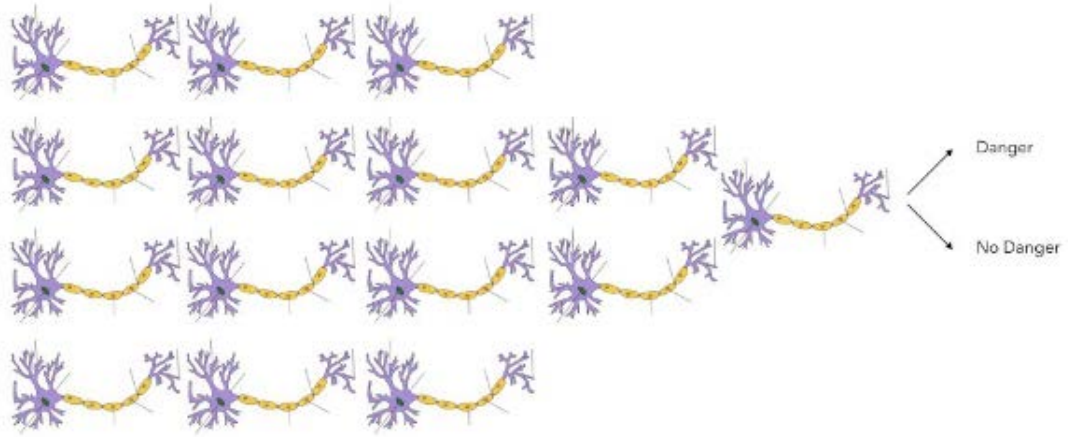


https://medium.com/analytics-vidhya/xor-gate-with-multilayer-perceptron-66e78671acd4

2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : inspiration

- Neurons in the brain are strongly connected. About 100 billion neurons form a complex and interconnected network in our brain, allowing us to generate complex thought patterns and actions. Neurons come in all sizes and shapes, but they mostly have long protrusions that connect to neighboring cells through specialized information-transmission structures called synapses.
- The architecture of Multi-Layer Perceptron's are inspired from the complex and interconnected network in our brain. It is hard to presents all interconnexion in the brain due the difficulty of their comprehension and the limitations of the available systems.



Source https://maelfabien.github.io/deeplearning/Perceptron/#why-neurons

2. Multi-layer Perceptron (MLP) and Deep Neural Network

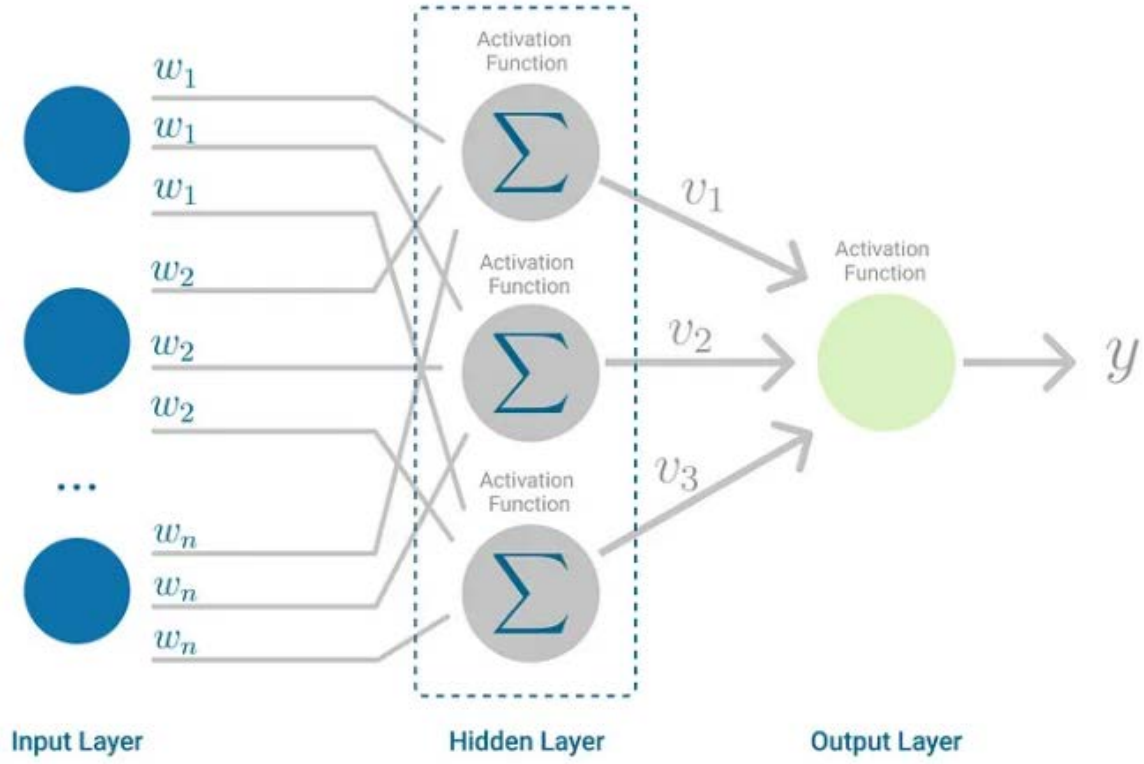
2.1. Multi-layer Perceptron (MLP)

- An MLP is characterized by one input layers, hidden layers, and one output layer.

- **The input layer** : contains n neurons (n is the number of features). Its helps to receive data.

- **The hidden layers** : Layers of nodes between the input and output layers. There may be one or more of these layers.

- **Output Layer**: A layer of nodes that produce the output variables. Generally, used in supervised problems. The number of neurons presents the number of classes.

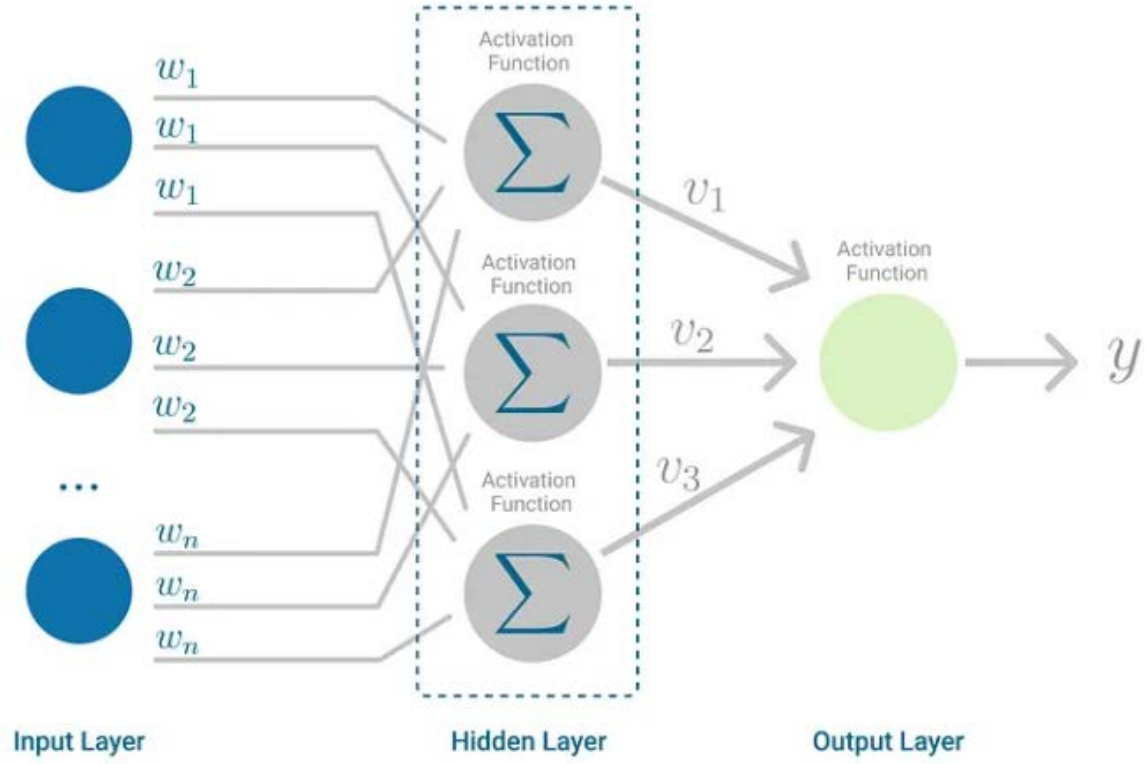


2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP)

To to describe the shape and capability of a neural network :

Size: The number of nodes in the model.
Width: The number of nodes in a specific layer.
Depth: The number of layers in a neural network.
Architecture: The specific arrangement of the layers and nodes in the network.

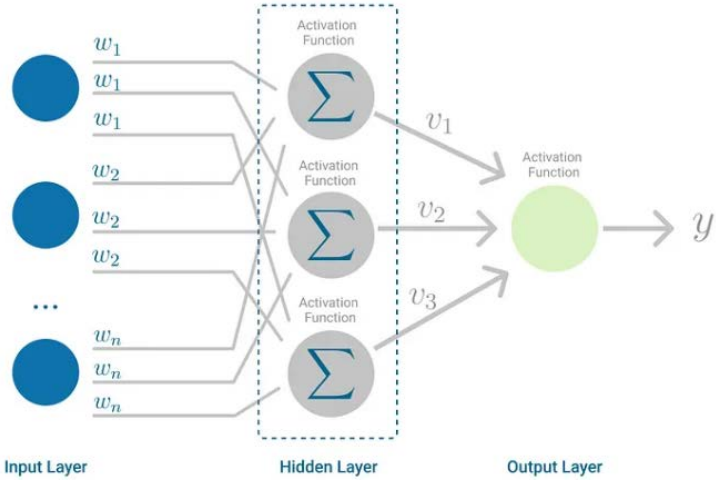


2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP)

How Many Layers and Nodes to Use?

- **Experimentation.**
- **Intuition :** an intuition that a deep network is required to address a specific predictive modeling problem. This intuition can come from experience with the domain, experience with modeling problems with neural networks, or some mixture of the two.
- **Go For Depth :**deep networks helps to solve complex problems (pay attention to overfitting).
- **Borrow Ideas :** the network layers and number of nodes used on related problems is a good starting point for testing ideas.
- **Search** : use strategies such as : Random, Heuristic.



Source : https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

2. Multi-layer Perceptron (MLP) and Deep Neural Network

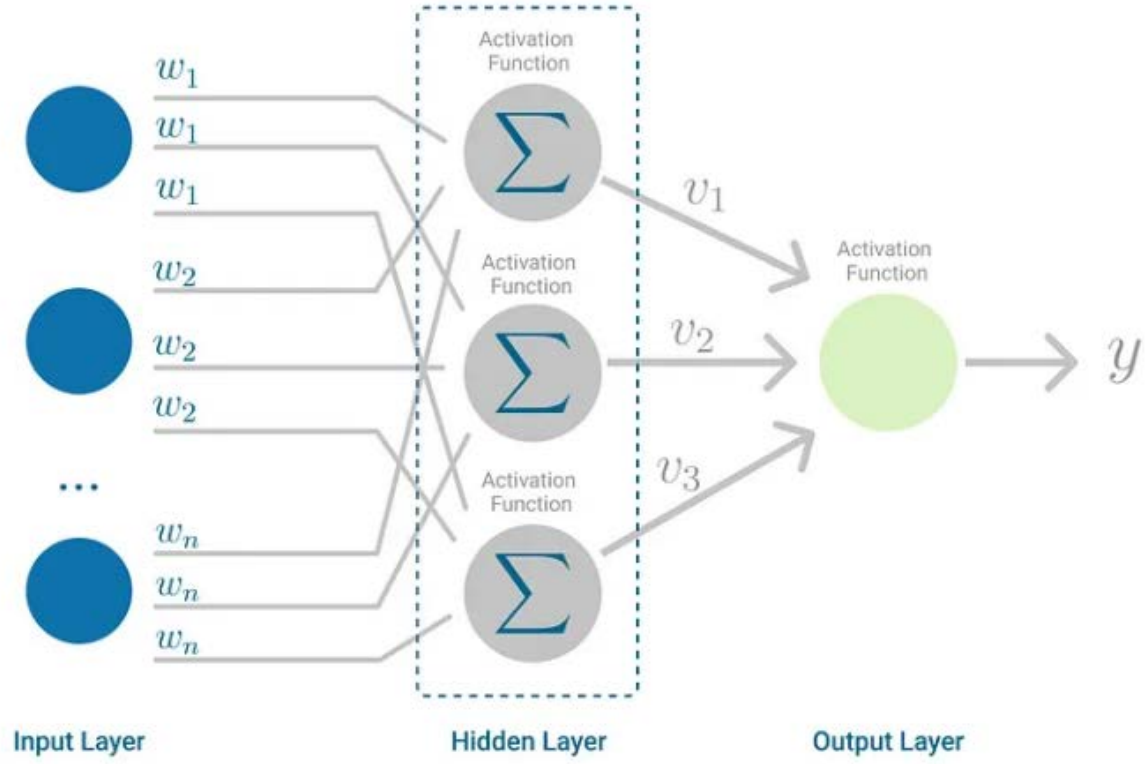
2.1. Multi-layer Perceptron (MLP) : forward propagation

The input data is fed in the forward direction through the network. Each hidden layer accepts the input data, processes it by the activation function and passes to the successive layer.

At each neuron in a hidden or output layer, the processing happens in two steps:

1. **Pre-activation :** a weighted sum of inputs .

2. **Activation :** the calculated weighted sum of inputs is passed to the activation function. An activation function is a mathematical function which adds non-linearity to the network.



2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : forward propagation

One hot encoding of Y

$$yv = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

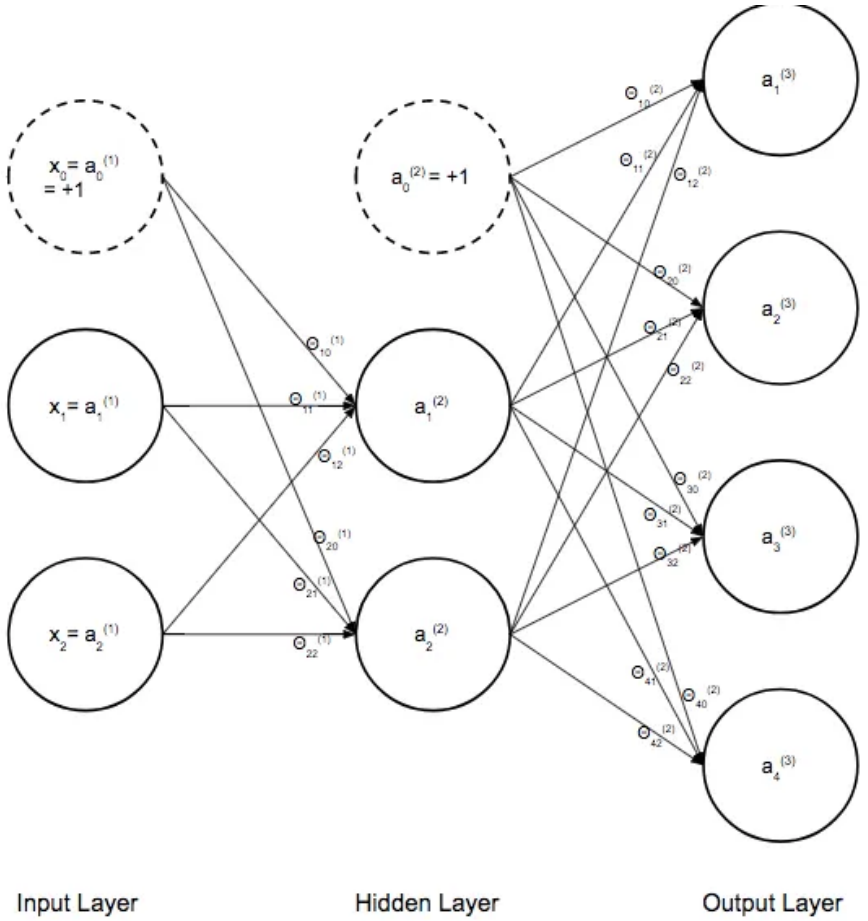
Bias x_0 | Feature x_1 | Feature x_2

$$X = a^{(1)} = \begin{bmatrix} 1.0000 & 0.5403 & -0.4161 \\ 1.0000 & -0.9900 & -0.6536 \\ 1.0000 & 0.2837 & 0.9602 \end{bmatrix}$$

← Example 1 input data, $x^{(1)}$

← Example 2 input data, $x^{(2)}$

← Example 3 input data, $x^{(3)}$



Source : https://towardsdatascience.com/under-the-hood-of-neural-network-forward-propagation-the-dreaded-matrix-multiplication-a5360b33426

2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : forward propagation

Theta1 = $\begin{bmatrix} 0.1 \; \Theta_{10}^{(1)} & 0.3 \; \Theta_{11}^{(1)} & 0.5 \; \Theta_{12}^{(1)} \\ 0.2 \; \Theta_{20}^{(1)} & 0.4 \; \Theta_{21}^{(1)} & 0.6 \; \Theta_{22}^{(1)} \end{bmatrix}$

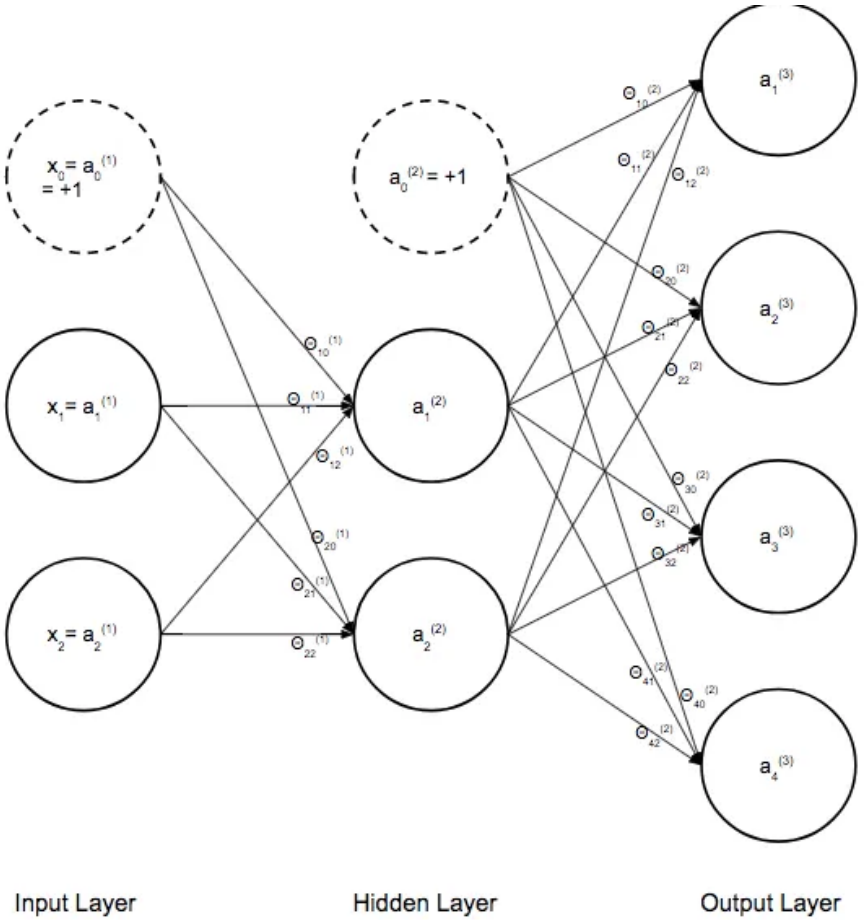
← Weights Making up 1st Activation Unit of 2nd Layer

← Weights Making up 2nd Activation Unit of 2nd Layer

$$a^{(2)} = Sigmoid\ (Theta1 \times (a^{(1)})^T)$$

Transpose $a^{(1)}$

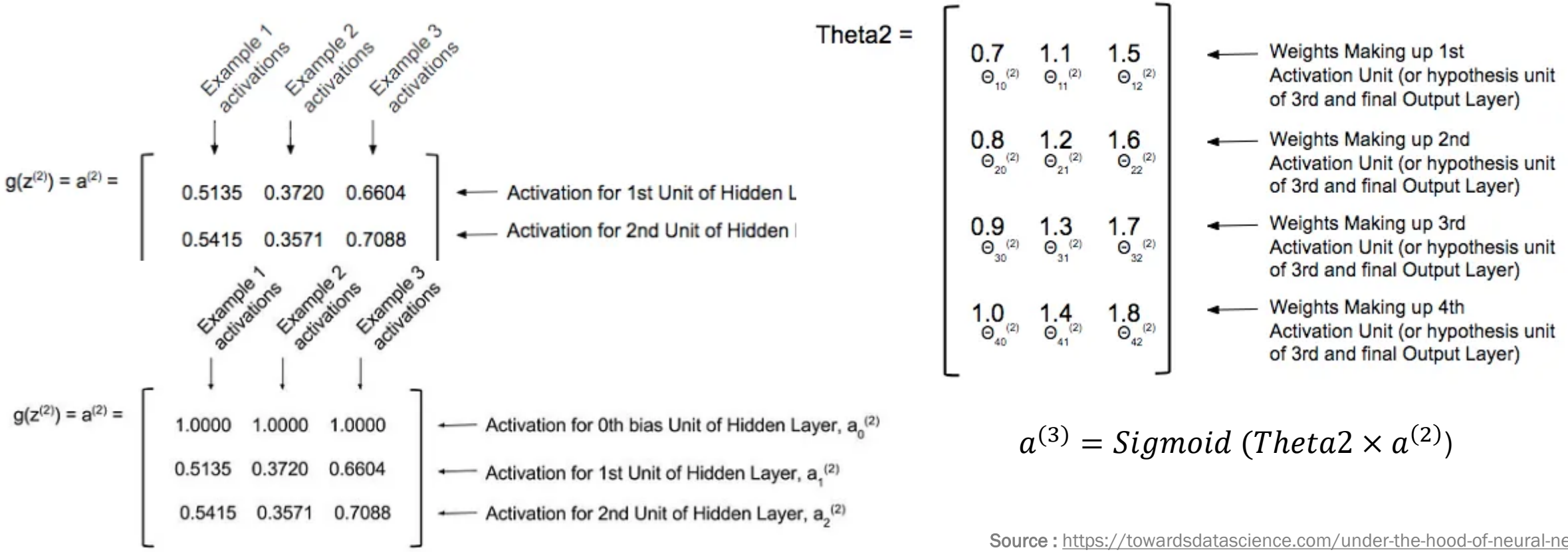
Compute $a^{(2)}$ and define the values of biases



Source : https://towardsdatascience.com/under-the-hood-of-neural-network-forward-propagation-the-dreaded-matrix-multiplication-a5360b33426

2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : forward propagation

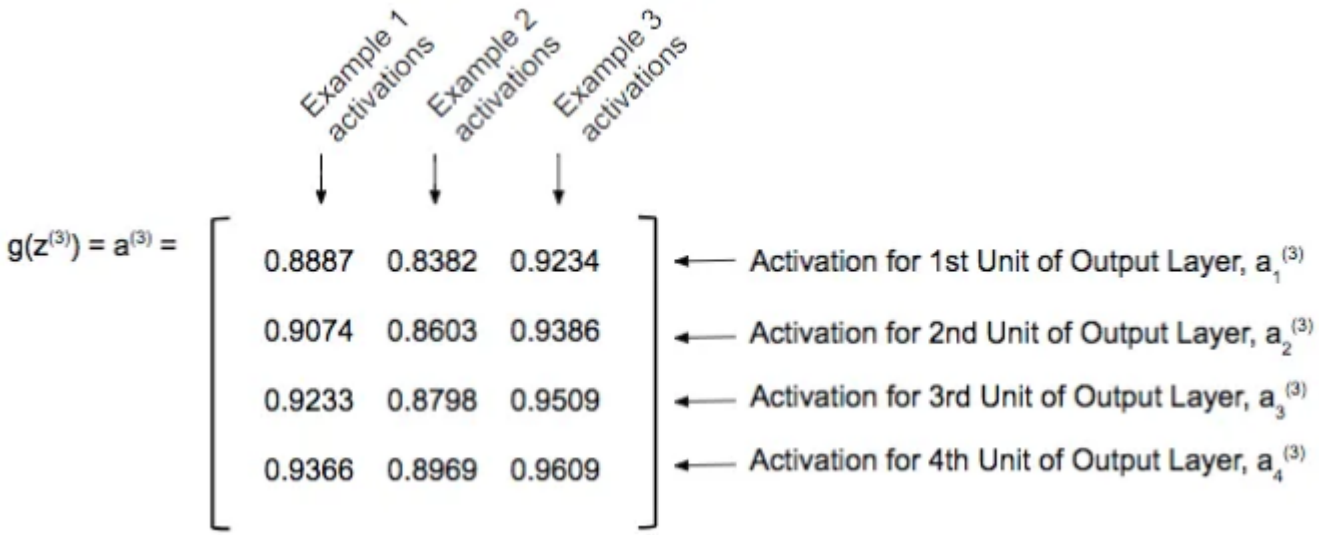


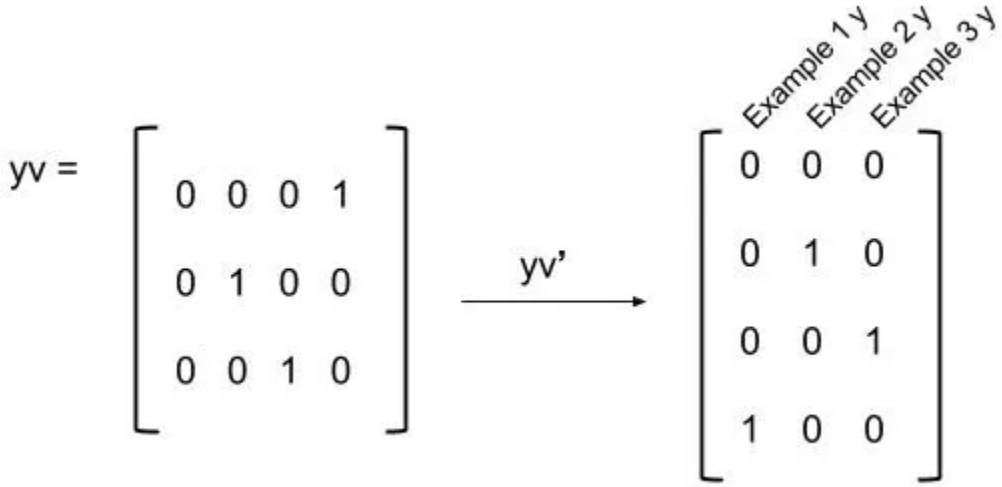
$$a^{(3)} = Sigmoid\ (Theta2 \times a^{(2)})$$

Source : https://towardsdatascience.com/under-the-hood-of-neural-network-forward-propagation-the-dreaded-matrix-multiplication-a5360b33426

2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : forward propagation





Cost Function : cross entropy loss

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(h_\theta\left(x^{(i)}\right)_k\right)\right]$$

Source : https://towardsdatascience.com/under-the-hood-of-neural-network-forward-propagation-the-dreaded-matrix-multiplication-a5360b33426

REMINDER

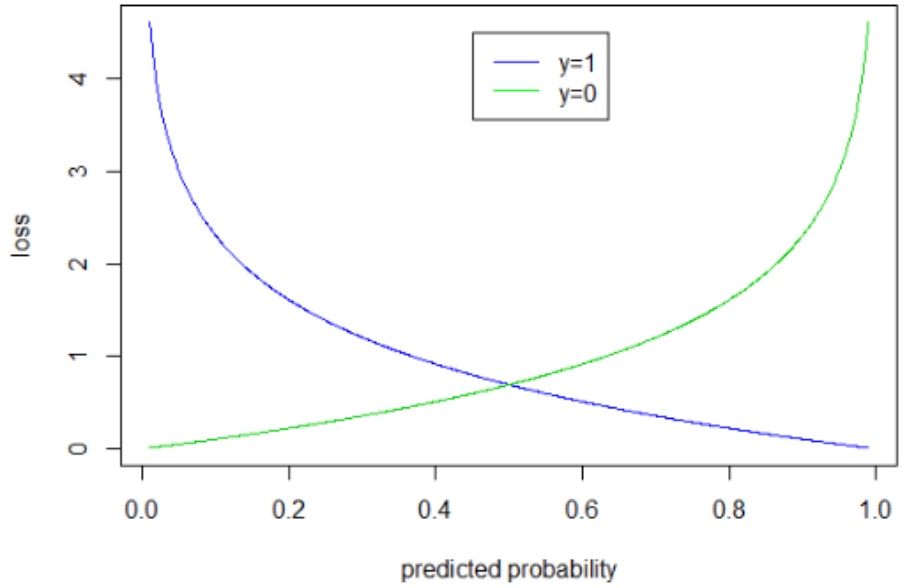
Cross entropy loss

Multi-classification problems

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(h_\theta\left(x^{(i)}\right)_k\right)\right]$$

Binary classification problems

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - \left(1-y^{(i)}\right)\log(1-h_\theta\left(x^{(i)}\right))\right]$$

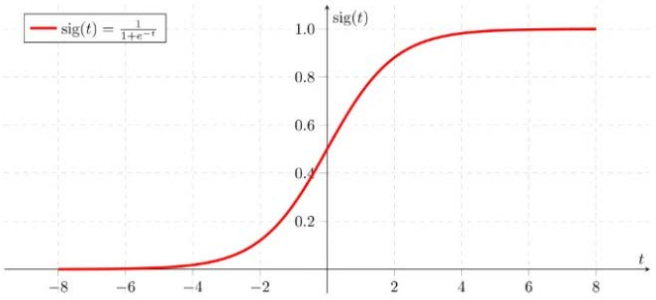


2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : activation function

Why using activation functions ? : ***Non-linear activation functions: Without an activation function, a Neural Network is just a linear regression model***. The activation function transforms the input in a non-linear way, allowing it to learn and as well as accomplish more complex tasks

Sigmoid : Used in hidden and last layer for binary classification problems. It is computationally expensive, causes vanishing gradient problems

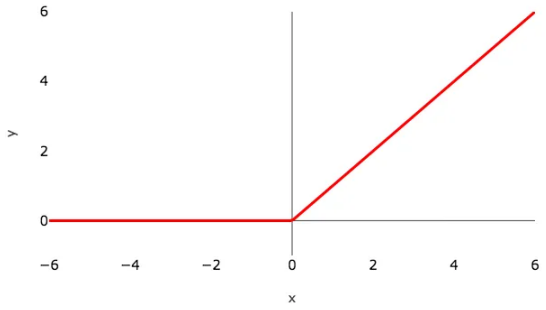


Softmax: The softmax is a more generalised form of the sigmoid. It is used in **multi-class classification problems**. it is used in the final layer in classification models

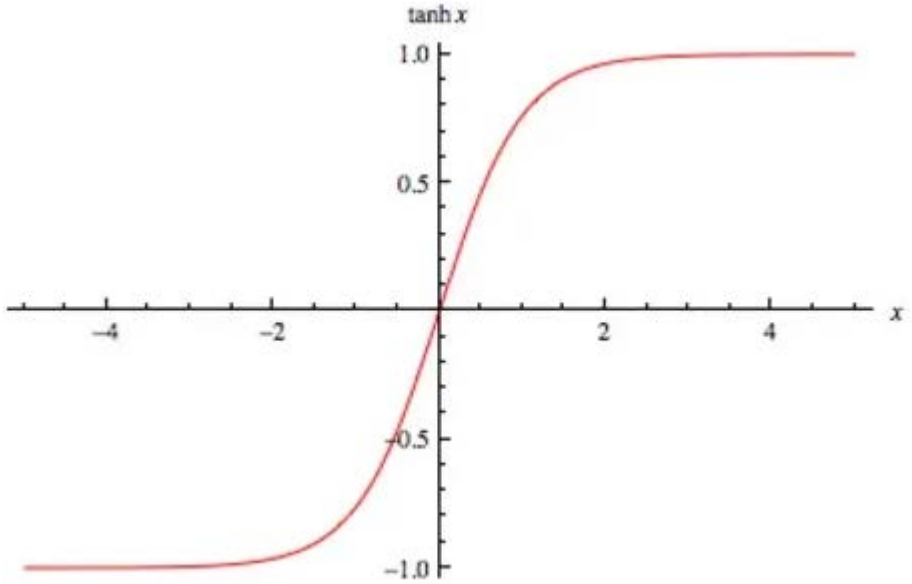
2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : activation function

ReLU (Rectified Linear Unit) is defined as **f(x) = max(0,x).** widely used activation function, especially with Convolutional Neural networks. It is easy to compute and does not saturate and does not cause the Vanishing Gradient Problem. It suffers from **"dying ReLU"** problem. Since the output is zero for all negative inputs. It causes some nodes to completely die and not learn anything



Tanh: The tanh is defined as:

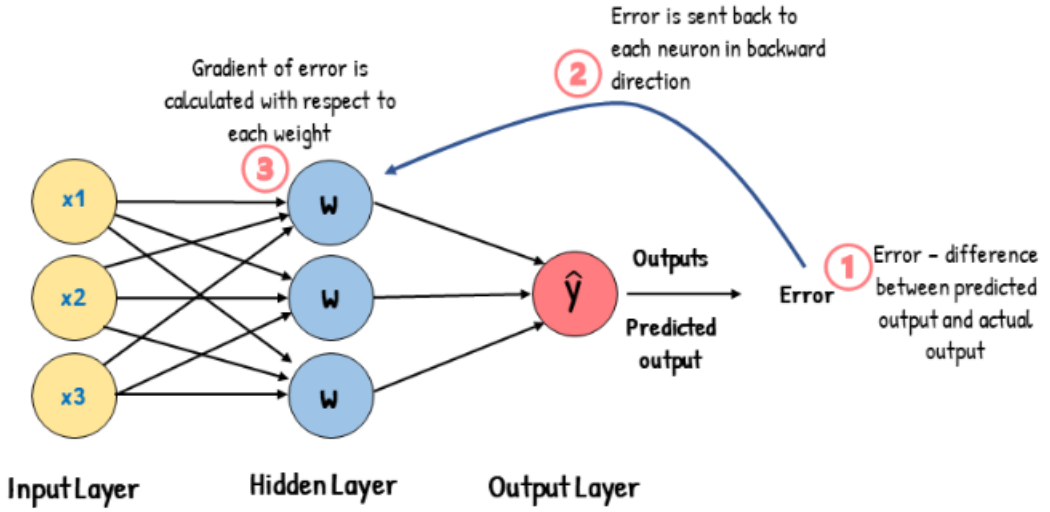


2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : backpropagation gradient descent

How to update weights in an MLP ? → Based on backpropagation

- The error is propagated backward during backpropagation from the output to the input layer. This error is then used to calculate the gradient of the cost function with respect to each weight.
- Essentially, backpropagation aims to calculate the negative gradient of the cost function.
- Back-propagation is the process of calculating the derivatives and gradient descent is the process of descending through the gradient



Source : https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/

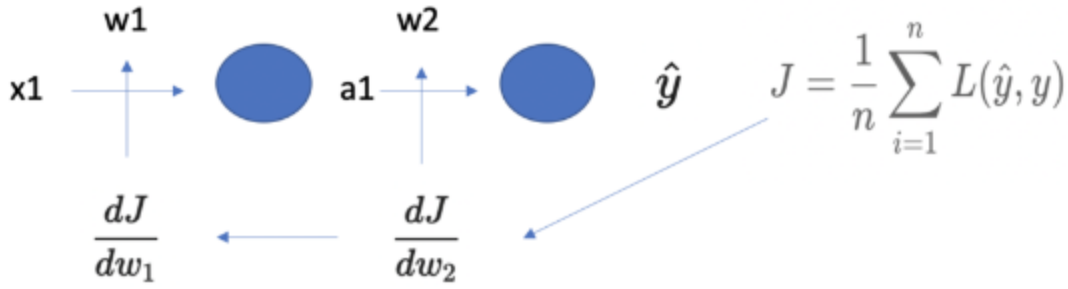
2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : backpropagation gradient descent

How the change in weights affects the error expressed by the cost J?

Find the set of weights that minimizes the cost and thus the error

- Two layers
- Each layer has one neuron.
- We will ignore the bias b .
  - We need to calculate the derivatives of the cost J with respect to w2 and w1 and update the respective weights.
- n is the number of samples.



$$z_1 = w_1\, x_1$$
$$a_1 = \sigma(z_1)$$
$$z_2 = w_2\, a_1$$
$$\hat{y} = \sigma(z_2) = \sigma(w_2\sigma(z_1))$$
$$J = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}, y)$$

w2 and w1 are not directly related to the cost

2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : backpropagation gradient descent

The backward differentiation through several intermediate functions is done using the chain rule.

Calculating the Adjustment to W2

J is related to w2 through the following equations in the forward pass

$$z_2 = w_2 a_1 \quad ; \quad \hat{y} = \sigma(z_2) \quad ; \quad J = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}, y)$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial w_2} \quad \textbf{(1)}$$

$$J = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}, y) = \frac{1}{n}\sum_{i=n}[-y\log(\hat{y}) - (1-y)\log(1-\hat{y})]$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left(\frac{1}{n}\sum_{i=n}[-y\log(\hat{y}) - (1-y)\log(1-\hat{y})]\right)$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \quad \textbf{(2)}$$

$$\frac{\partial \hat{y}}{\partial z_2} = \frac{\partial \hat{y}}{\partial z_2} = \frac{\partial}{\partial z_2}\sigma(z_2) = \frac{\partial}{\partial z_2}\left(\frac{1}{1+e^{-z_2}}\right)$$

$$\frac{\partial \hat{y}}{\partial z_2} = \frac{1}{1+e^{-z_2}}\left(1 - \frac{1}{1+e^{-z_2}}\right) \quad \textbf{(3)}$$

$$\frac{\partial z_2}{\partial w_2} = \frac{\partial}{\partial w_2}(w_2 a_1) = a_1 \quad \textbf{(4)}$$

By replacing (2), (3), and (4) in (1), we obtain :

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial w_2} = \left(\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right)\left(\frac{1}{1+e^{-z_2}}\left(1 - \frac{1}{1+e^{-z_2}}\right)\right)a_1$$

$$\frac{\partial J}{\partial w_2} = \left(\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right)\left(\frac{1}{1+e^{-w_2 a_1}}\left(1 - \frac{1}{1+e^{-w_2 a_1}}\right)\right)a_1$$

Source https://programmathically.com/understanding-backpropagation-with-gradient-descent/

2. Multi-layer Perceptron (MLP) and Deep Neural Network

2.1. Multi-layer Perceptron (MLP) : backpropagation gradient descent

Calculating the Adjustment to W1

J is related to w1 through the following equations in the forward pass, but we have to go back even further.

$$z_1 = w_1 x_1 \quad ; a_1 = \sigma(z_1) \quad ; z_2 = w_2 a_1 \; ; \hat{y} = \sigma(z_2) \quad ; \quad J = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}, y)$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} \quad \textbf{(1)}$$

Compute $\frac{\partial J}{\partial w_1}$

Weights update

$$w_2^{(t+1)} = w_2^{(t)} - \alpha \frac{\partial J}{\partial w_2}$$

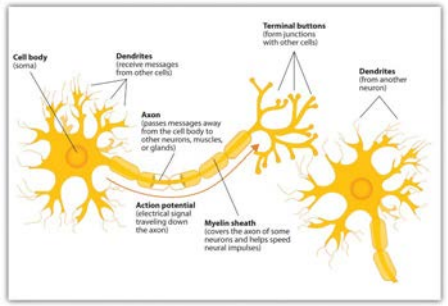
$$w_1^{(t+1)} = w_1^{(t)} - \alpha \frac{\partial J}{\partial w_1}$$

α is the learning rate.

Source https://programmathically.com/understanding-backpropagation-with-gradient-descent/

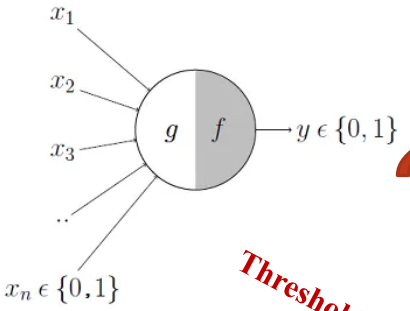
Conclusion

I Learned Multiple Concepts



How a biological neuron works ?

1



Heaviside

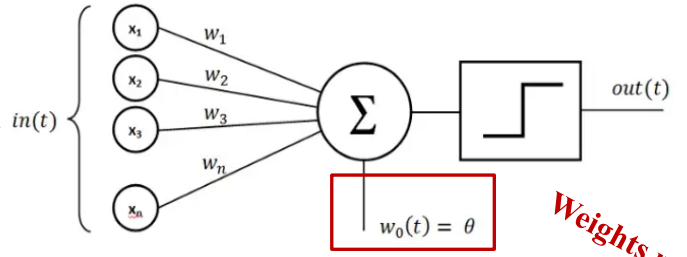
Threshold

McCulloch-Pitts Neuron (MCP)

Linear Problems

No Training Process !

2



Artificial neuron used by the perceptron. From Wikipedia.

Weights update

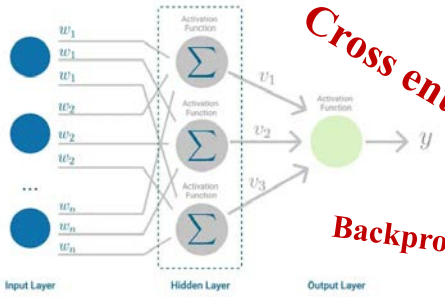
Bias

Perceptron

Training process !

Learning rate

activation functions



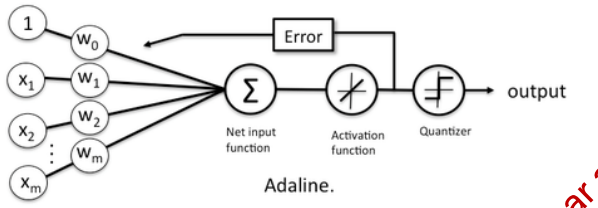
Cross entropy loss

Forward Propagation

Backpropagation

4

3



Sum-of-squared error cost

SGD

ADALINE

Linear activation function

Multi-layer Perceptron (MLP)

REFERENCES

[1] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*, 115-133.
[2] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.
[3] Marvin, M., & Seymour, A. P. (1969). Perceptrons. *Cambridge, MA: MIT Press*, *6*, 318-362.
[4] Widrow, B., & Hoff, M. E. (1960). *Adaptive switching circuits*. Stanford Univ Ca Stanford Electronics Labs.

SOURCES

https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a

https://maelfabien.github.io/deeplearning/Perceptron/#the-classic-model

https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html