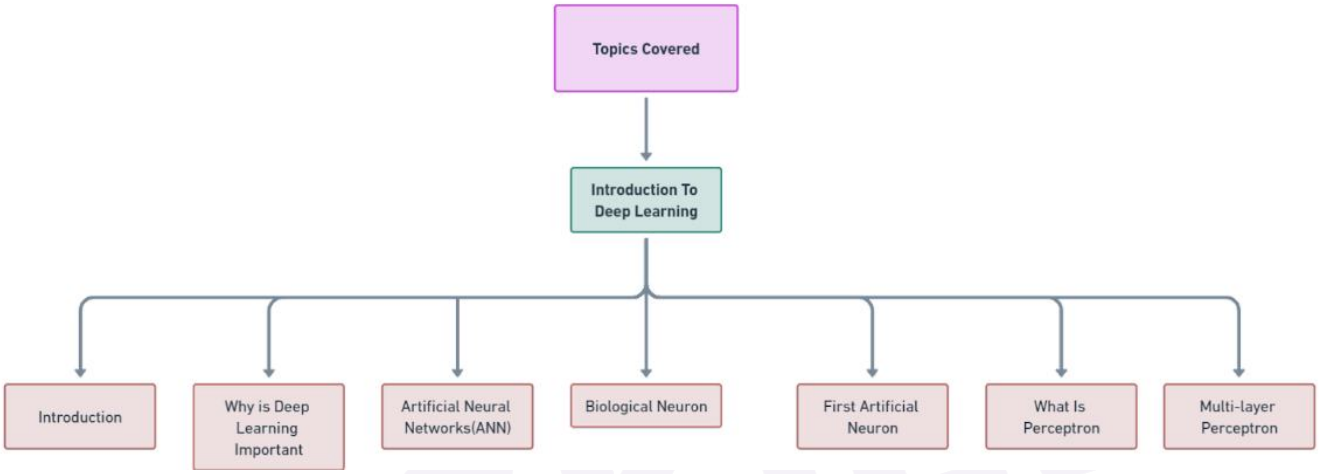
PW SKILLS

Lesson Plan

Introduction to Deep Learning

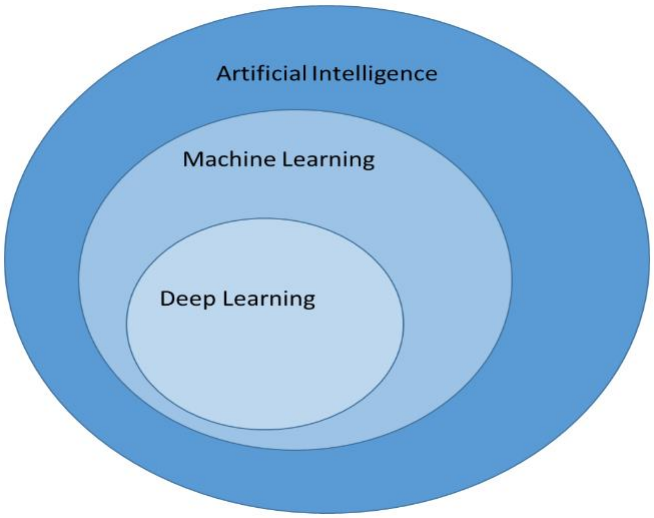
Topics Covered

- Introduction
- Why is Deep Learning Important
- Artificial Neural Networks(ANN)
- Biological Neuron
- First Artificial Neuron
- What Is Perceptron
- Multi-layer Perceptron



Introduction

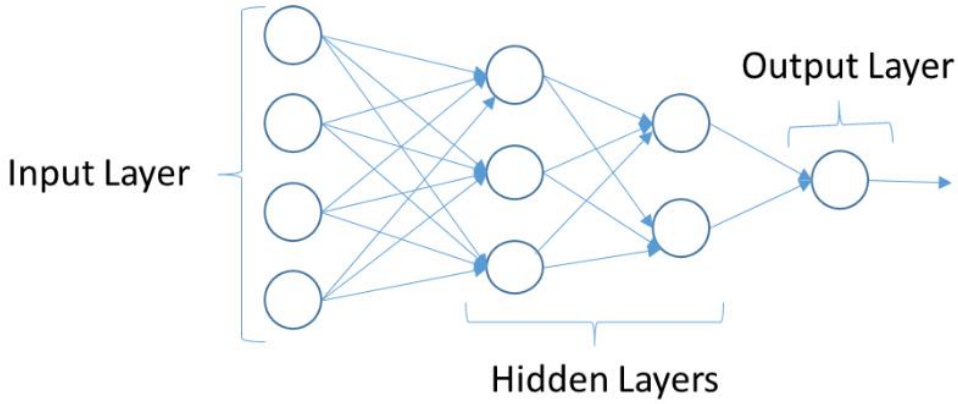
To understand what deep learning is, we first need to understand the relationship deep learning has with machine learning, neural networks, and artificial intelligence. The best way to think of this relationship is to visualize them as concentric circles.



- At the outermost ring, you have artificial intelligence (using computers to reason). One layer inside of that is machine learning with artificial neural networks and deep learning at the center.

- Broadly speaking, deep learning is a more approachable name for an artificial neural network. The “deep” in deep learning refers to the depth of the network. An artificial neural network can be very shallow.

- Neural networks are inspired by the structure of the cerebral cortex. At the basic level is the perceptron, the mathematical representation of a biological neuron. Like in the cerebral cortex, there can be several layers of interconnected perceptrons.

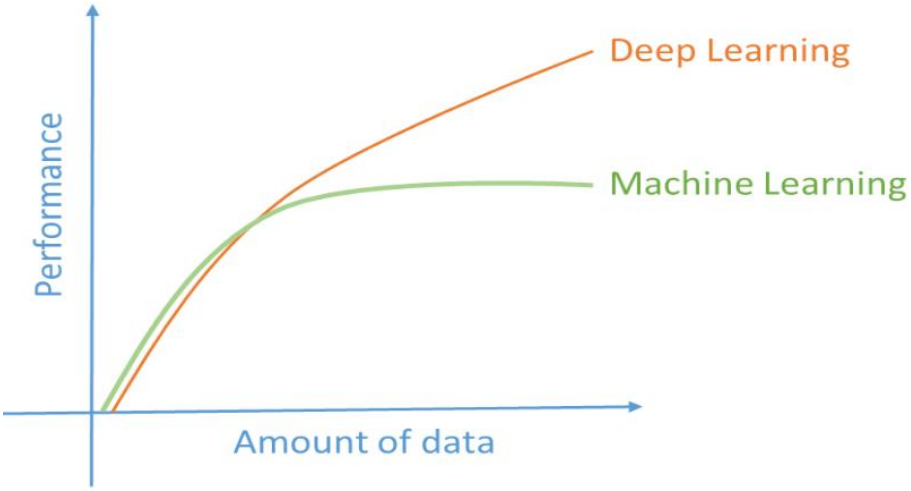


- The first layer is the input layer. Each node in this layer takes an input, and then passes its output as the input to each node in the next layer. There are generally no connections between nodes in the same layer and the last layer produces the outputs.

- We call the middle part the hidden layer. These neurons have no connection to the outside (e.g. input or output) and are only activated by nodes in the previous layer.

- Machine learning is considered a branch or approach of Artificial intelligence, whereas deep learning is a specialized type of machine learning.

Why is Deep Learning Important



- Computers have long had techniques for recognizing features inside of images. The results weren't always great. Computer vision has been a main beneficiary of deep learning. Computer vision using deep learning now rivals humans on many image recognition tasks.

- Facebook has had great success with identifying faces in photographs by using deep learning. It's not just a marginal improvement, but a game changer: "Asked whether two unfamiliar photos of faces show the same person, a human being will get it right 97.53 percent of the time. New software developed by researchers at Facebook can score 97.25 percent on the same challenge, regardless of variations in lighting or whether the person in the picture is directly facing the camera."

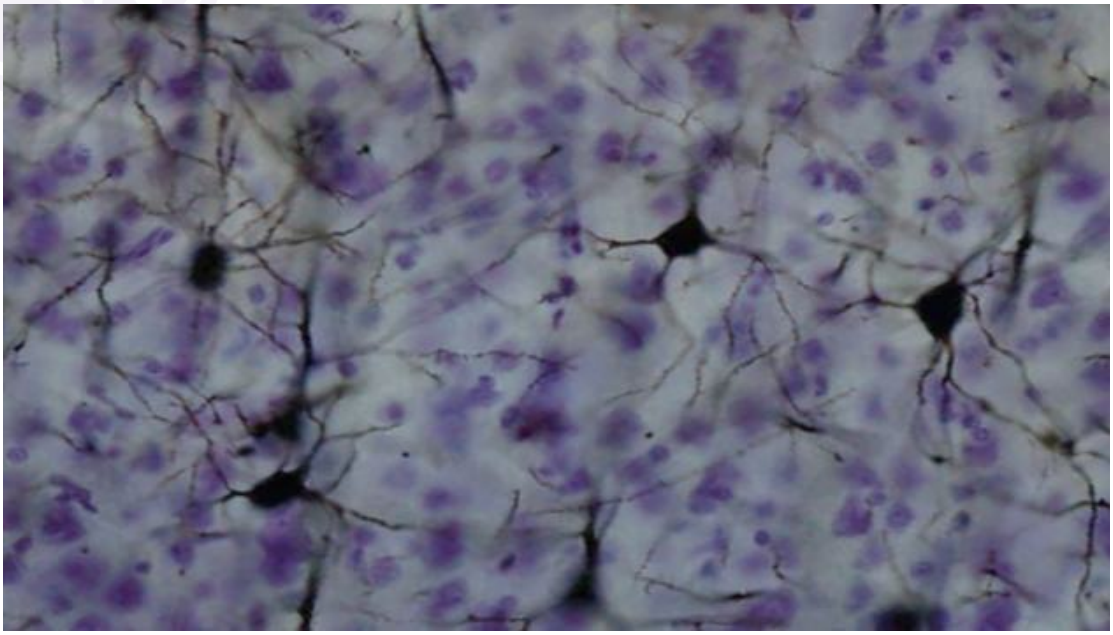
- Speech recognition is another area that's felt deep learning's impact. Spoken languages are so vast and ambiguous. Baidu – one of the leading search engines of China – has developed a voice recognition system that is faster and more accurate than humans at producing text on a mobile phone. In both English and Mandarin.

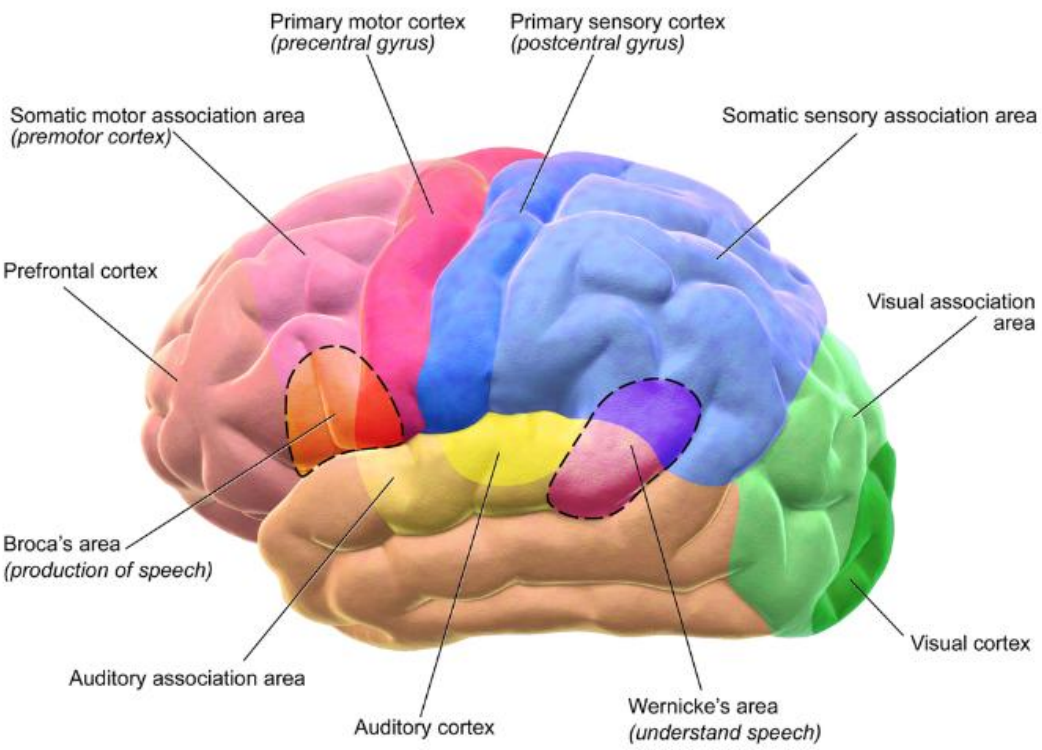
- What is particularly fascinating, is that generalizing the two languages didn't require much additional design effort: "Historically, people viewed Chinese and English as two vastly different languages, and so there was a need to design very different features," Andrew Ng says, chief scientist at Baidu. "The learning algorithms are now so general that you can just learn."

- Google is now using deep learning to manage the energy at the company's data centers. They've cut their energy needs for cooling by 40%. That translates to about a 15% improvement in power usage efficiency for the company and hundreds of millions of dollars in savings.

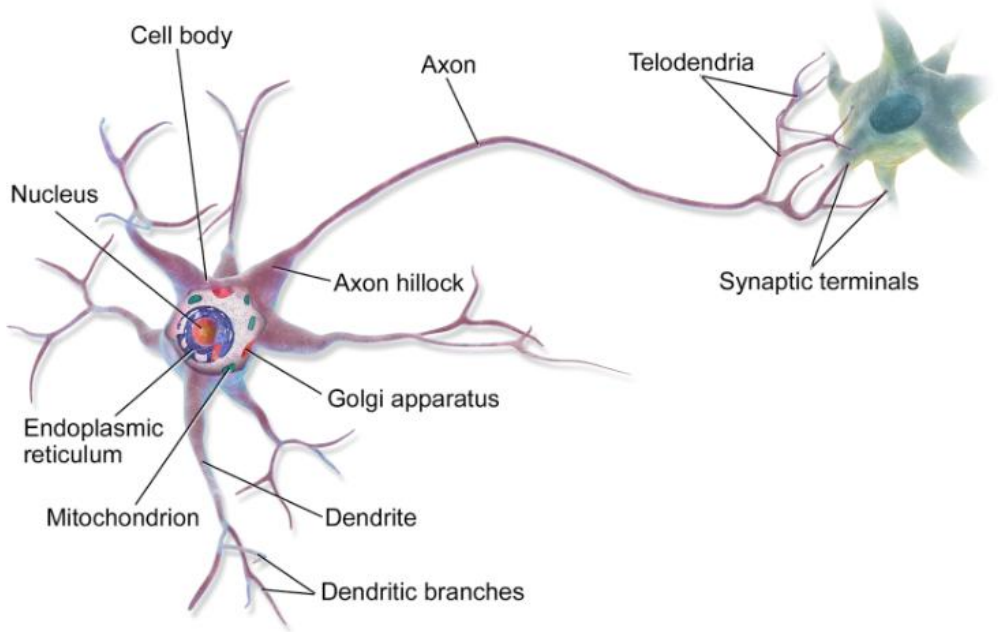
Artificial Neural Networks (ANN)

- ANNs are inspired by biological neurons found in the cerebral cortex of our brain.
- The cerebral cortex (plural cortices), also known as the cerebral mantle, is the outer layer of neural tissue of the cerebrum of the brain in humans and other mammals. - Wikipedia





- ANNs are the core of deep learning. Hence one of the most important topics to understand.
- ANNs are versatile, scalable and powerful. Thus it can tackle highly complex ML tasks like classifying images, identifying objects, speech recognition etc.



- Biological Neurons produce short electrical impulses known as action potentials which travel through axons to the synapses which release chemical signals i.e neurotransmitters.

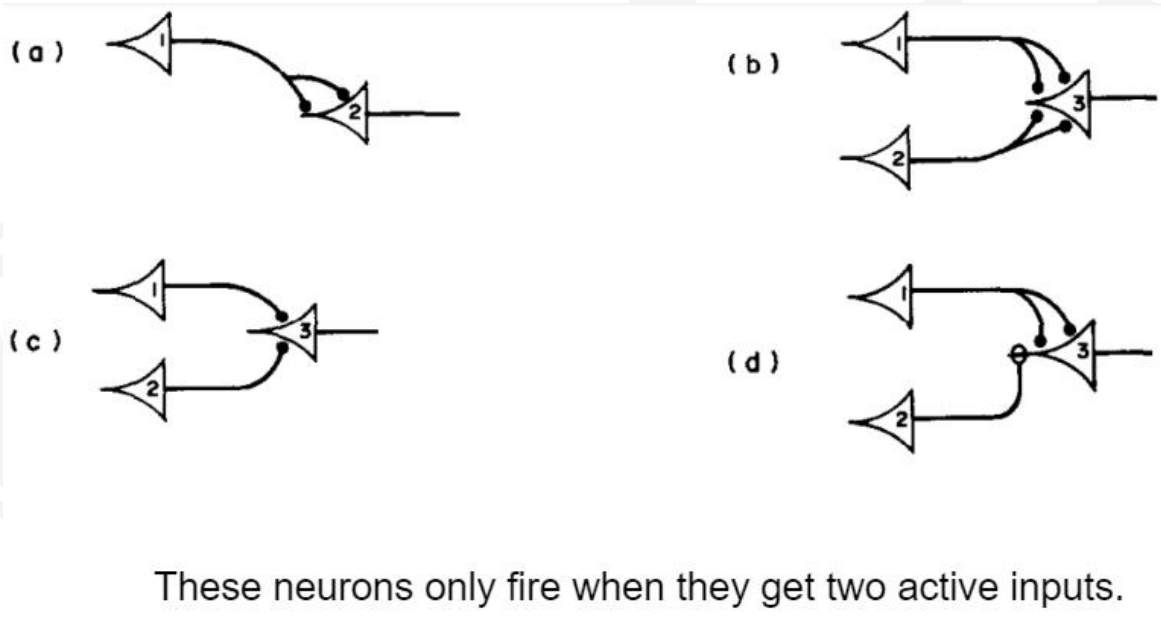
- When a connected neuron receives a sufficient amount of these neurotransmitters within a few milliseconds, it fires (or does not fire, think of a NOT gate here) its own action potential or electrical impulse.

- These simple units form a strong network known as the Biological Neural Network (BNN) to perform very complex computation tasks.

First Artificial Neuron

- It was in year 1943, Artificial neuron was introduced by
  - Neurophysiologist Warren McCulloch and
  - Mathematician Walter Pitts
- They have shown that these simple neurons can perform small logical operations like OR, NOT, AND gate etc.

- The following figure represents these ANs which can perform (a) Buffer, (b) OR, (c) AND and (d) A-B operation



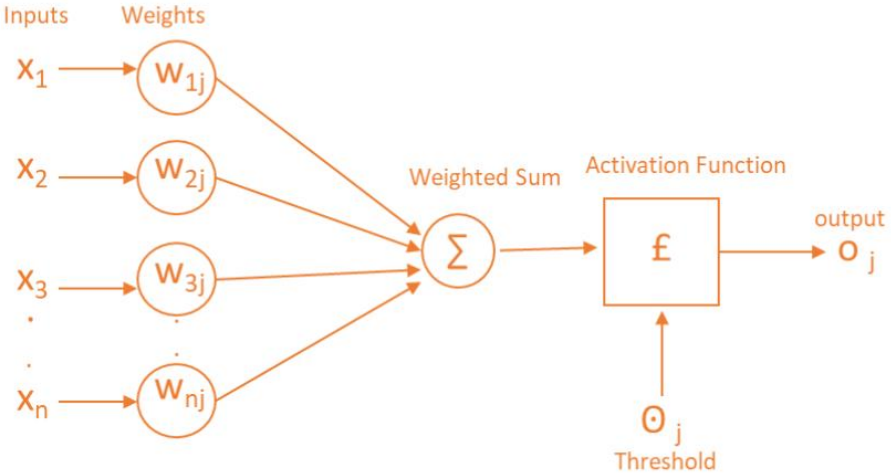
These neurons only fire when they get two active inputs.

- **The Perceptron**
  - It's the simplest ANN architecture. It was invented by Frank Rosenblatt in 1957 and published as Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408. doi:10.1037/h0042519 .

  - It has a different architecture than the first neuron that we have seen above. It is known as threshold logic unit(TLU) or linear threshold unit (LTU).

  - Here inputs are not just binary.

  - Let's see the architecture shown below -



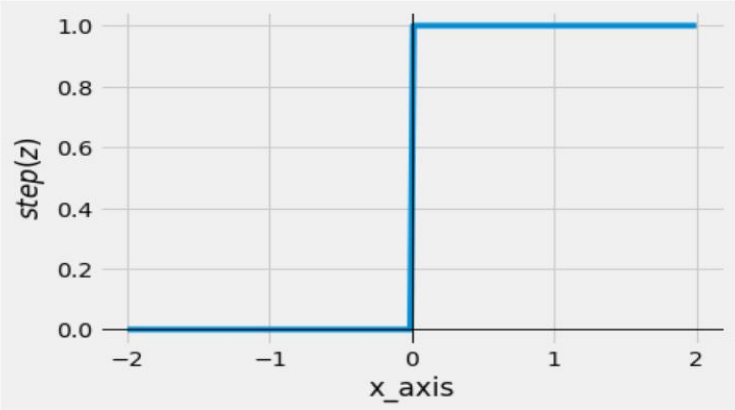
Common activation functions used for Perceptrons are (with threshold at 0)

$$step(z) \text{ or } heaviside(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")

x_axis = np.linspace(-2,2,200)
step = np.where(x_axis < 0, 0, 1)

plt.plot(x_axis, step)
plt.xlabel("x_axis")
plt.ylabel(r"$step(z)$")
plt.axhline(0, color='k', lw=1);
plt.axvline(0, color='k', lw=1);
```

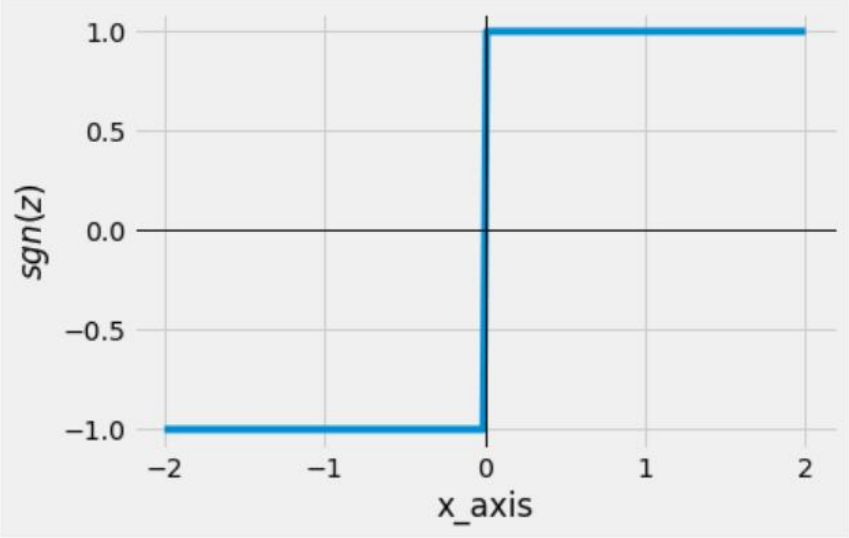


$$sgn(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$$

```
def sgn(x):
    if x < 0:
        return -1
    elif x > 0:
        return 1
    return 0

sgn = np.array(list(map(sgn, x_axis)))

plt.plot(x_axis, sgn)
plt.xlabel("x_axis")
plt.ylabel(r"$sgn(z)$")
plt.axhline(0, colour='k', lw=1);
plt.axvline(0, colour='k', lw=1);
```



The activation function is analogous to firing of a neurotransmitter at a specific threshold.

Perceptron learning Rule:

$$w_{i,j} \leftarrow w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

Where,

Wi,j: connection weight between ith input and jth output neuron

xi: ith value.

y^: output jth output neuron.

yj: target output of jth output neuron

n: Learning rate
It can also be written as for jth element of w vector

$$w_j = w_j + \triangle w_j$$
$$where,\ \triangle w_j = \eta(y^{(i)} - \hat{y}_j{}^{(i)})x_j^{(i)}$$

- Single TLUs are simple linear binary classifiers hence not suitable for non-linear operation.
- Rosenblatt proved that if the data is linearly separable then only this algorithm will converge which is known as the Perceptron learning theorem.
- Rosenblatt proved that if the data is linearly separable then only this algorithm will converge which is known as the Perceptron learning theorem.

The perceptron tries to find a separating hyperplane for the two response classes. Namely, a set of weights that specifies

$X_1W^T = 0$ and $X_2W^T = 0$

Hence,

$X_1W^T = X_2W^T$

$(X_1 - X_2)W^T = 0$

This means that either the norms of 1- 2 or are zero, or the cosine of the angle between them is equal to zero, due to the identity.

ab=|| |||| ||cos
Since there is no reason for the norms to be zero in general, we need the two vectors to be at right angles to one another. So, we need a weight vector that is perpendicular to the decision boundary.

$$ab=\|a\|\|b\|\cos\theta$$

Since there is no reason for the norms to be zero in general, we need the two vectors to be at right angles to one another. So, we need a weight vector that is perpendicular to the decision boundary.

Derivation :

- Let's assume that you are doing a binary classification with classes +1 and -1
- Let there be decision function ( ).
- which takes a linear combination of certain inputs " ".
- corresponding weights " " and net input = 1 1+ 2 2+...+ .

So in vector form, we have

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

So,

Z = WT X
Now if for a sample x

$$\phi(z) = \begin{cases} +1 & \text{if } z \geq \theta \\ -1 & \text{if } z < \theta \end{cases}$$

Let's simplify the above equation.

$$\phi(z) = \begin{cases} +1 & \text{if } z - \theta \geq 0 \\ -1 & \text{if } z - \theta < 0 \end{cases}$$

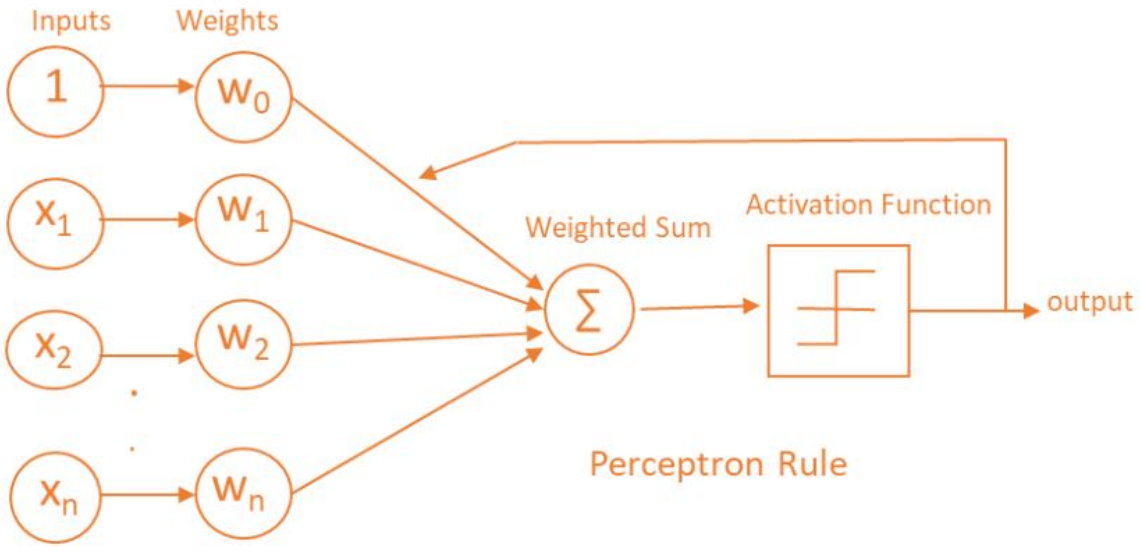
Suppose w0 = - and x0 = 1 then,

$$z' = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

and

$$\phi(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

Here, w0x0 is usually known as a bias unit.



Multi-layer Perceptron

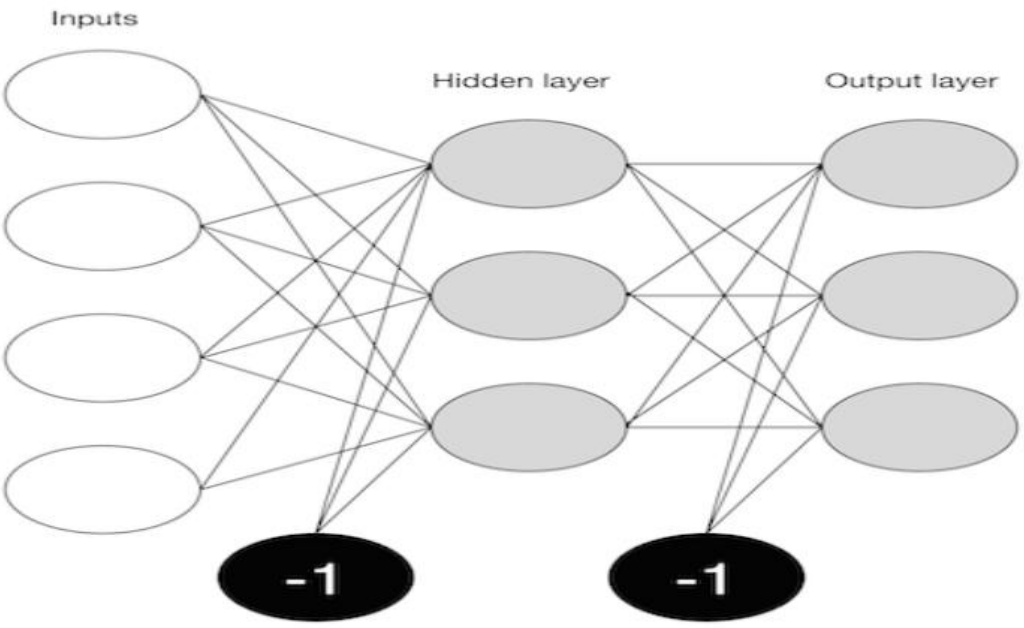
- The solution to fitting more complex (i.e. non-linear) models with neural networks is to use a more complex network that consists of more than just a single perceptron. The take-home message from the perceptron is that all of the learning happens by adapting the synapse weights until the prediction is satisfactory. Hence, a reasonable guess at how to make a perceptron more complex is to simply add more weights.

There are two ways to add complexity.

- Add backward connections, so that output neurons feedback to input nodes, resulting in a recurrent network.

- Add neurons between the input nodes and the outputs, creating an additional ("hidden") layer to the network, resulting in a multi-layer perceptron.

The latter approach is more common in applications of neural networks.



How to train a multilayer network is not intuitive. Propagating the inputs forward over two layers is straightforward since the outputs from the hidden layer can be used as inputs for the output layer. However, the process for updating the weights based on the prediction error is less clear, since it is difficult to know whether to change the weights on the input layer or on the hidden layer in order to improve the prediction.

Updating a multi-layer perceptron (MLP) is a matter of:

- moving forward through the network, calculating outputs given inputs and current weight estimates.
- moving backward updating weights according to the resulting error from forward propagation.

In this sense, it is similar to a single-layer perceptron, except it has to be done twice, once for each layer.