NPTEL | DEEP LEARNING | CS1075

Reticular theory is an obsolete scientific theory in neurobiology that stated that everything in the nervous system, such as brain, is a single continuous network. The concept was postulated by a German anatomist Joseph von Gerlach in 1871, and was most popularised by the Nobel laureate Italian physician Camillo Golgi.

Reticular theory – single cell; Joseph von Gerlach, 1871
Camillo Golgi *– proponent of Reticular theory (single continuous network); Staining technique*
Santiago Cajal *– not a single one; collection of neurons – neuron doctrine!*
Heinrich Gottfried *– coined neuron and chromosome; consolidated into Neuron Doctrine!*
1906 Nobel in Medicine *– Camillo Golgi, Santiago Ramón y Cajal*
Electron microscope *– gap between neurons; 1950's, debate settled!*

McCulloch-Pitts Neuron *– 1943 + AI*
AI – *1956, coined the word AI*
Perceptron *–*
Deep Learning – large number of neurons, inter-connected to perform certain activities
Limitations *– it's possible that a perceptron cannot make simple decisions; not able to do basic XOR.*
Types of AI *– Connectionist AI and Symbolic AI; we'll focus on Connectionism*
"A multi-layered n/w of neurons can do – whatever cited as limitation {a single neuron cannot do simple tasks!}"
Back Propagation
Gradient descent
Universal Approximation theorem

McCulloch Pitts neuron [MP neuron]
Threshold –theta
Exhibitors and inhibitors
Classifying 0 or 1 | linear decision boundary | linearly separable function
Multi-dimensional planes

Bias (w0 = -theta)
Sigma i=0 to n, w0, w1, w2,.... wn

Perceptron
Give data
M movies and labels (class) – liked or not/ binary decision
Data+labels
Perceptron is supposed to Adjust the weights – in such a way that we should be able to separate
Perfect match
Feed movies – return same label

PLA – Perceptron Learning Algorithm
P -> inputs with label 1
N -> inputs with label 0
Assign weights randomly!
As we don't know the weights as of now....

Convergence – when there're no more errors...

W = [w0, w1, w2,... wn] / n+1 dimension
X is n-dimension
X = [1,x1,x2,...xn]

While convergence != '' do

Pick random x element of P Union N

If X element of P AND If summation ($w^T X$) > 0 THEN

W = W + X

If X element of N AND If summation ($w^T X$) <= 0

W = W - X

Dot Product:

*w.x = $w^T X$ = SUM(Wi * Xi) ; i= 0 to n*

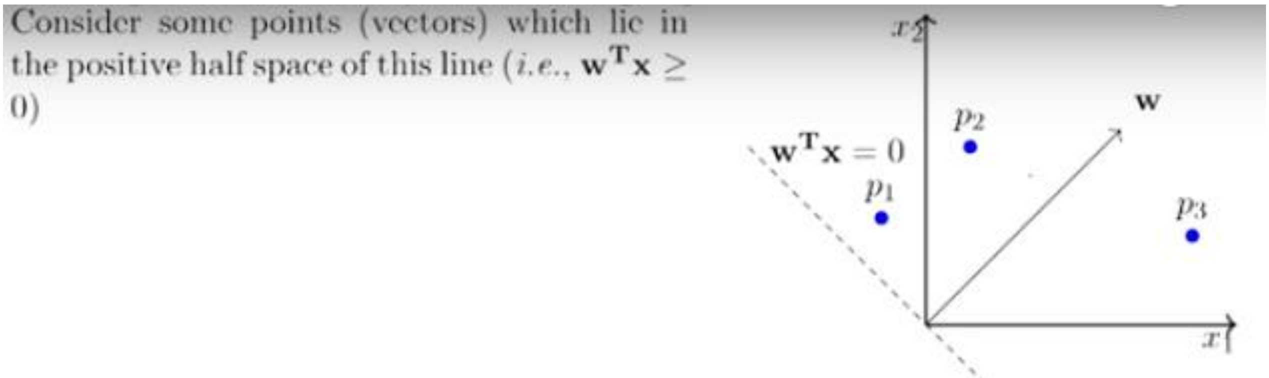
Perceptron rule:

y = 1 if $w^T X$ >= 0
y = 0 if $w^T X$ < 0

$w^T X$ => divides the input space into two...

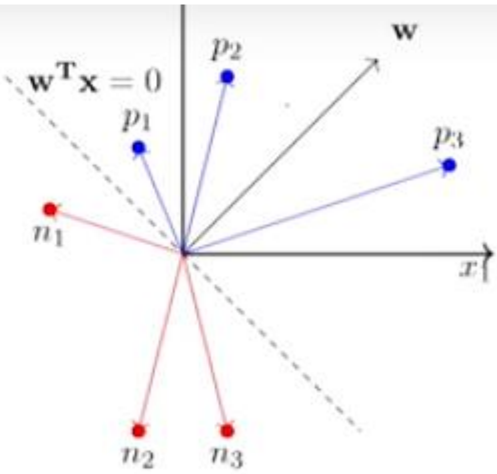
Need the line $w^T X = 0$; perpendicular

Every point (x) on the line satisfies the equation $w^T X = 0$
Angle alpha = 90 / Cos 90 = 0*



0)

- What will be the angle between any such vector and $\mathbf{w}$? Obviously, less than 90°
- What about points (vectors) which lie in the negative half space of this line (*i.e.*, $\mathbf{w^T x} < 0$)
- What will be the angle between any such vector and $\mathbf{w}$? Obviously, greater than 90°



- Of course, this also follows from the formula ($cos\alpha = \frac{w^T x}{||w|| ||x||}$)

- For $\mathbf{x} \in P$ if $\mathbf{w.x} < 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is greater than 90° (but we want α to be less than 90°)
- What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T \mathbf{x} \\ &\propto (\mathbf{w} + \mathbf{x})^T \mathbf{x} \\ &\propto \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} \\ &\propto cos\alpha + \mathbf{x}^T \mathbf{x} \\ cos(\alpha_{new}) &> cos\alpha \end{aligned}$$

- For $\mathbf{x} \in N$ if $\mathbf{w.x} \geq 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is less than 90° (but we want α to be greater than 90°)
- What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T \mathbf{x} \\ &\propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \\ &\propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \\ &\propto cos\alpha - \mathbf{x}^T \mathbf{x} \end{aligned}$$

$$cos(\alpha_{new}) < cos\alpha$$

- Thus α_{new} will be greater than α and this is exactly what we want

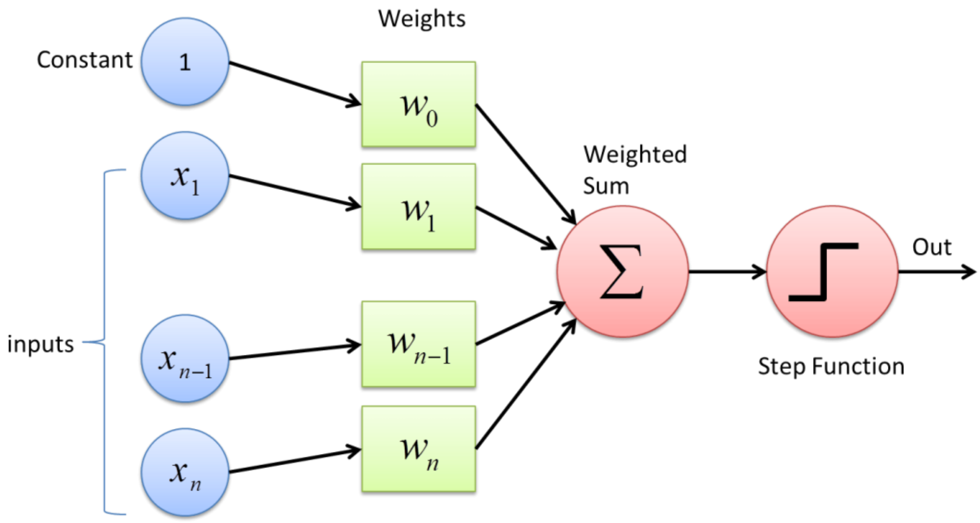
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} x_i \geq \theta \\
&= 0 \quad if \sum_{i=0}^{n} x_i < \theta
\end{aligned}
$$

Perceptron

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq \theta \\
&= 0 \quad if \sum_{i=0}^{n} w_i * x_i < \theta
\end{aligned}
$$

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side
- In other words, a single perceptron can only be used to implement linearly separable functions
- Then what is the difference? The weights (including threshold) can be learned and the inputs can be real valued



c. ***Apply*** that weighted sum to the correct ***Activation Function.***

For Example : Unit Step Activation Function.

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$

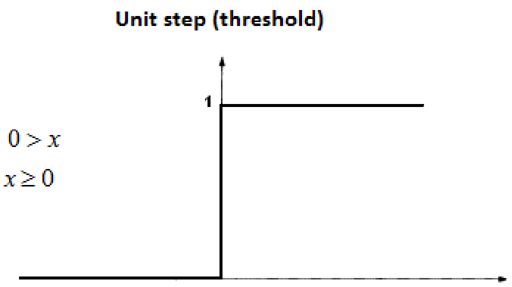


Fig: Unit Step Activation Function

Why do we need Activation Function?

> *In short, the activation functions are used to map the input between the required values like (0, 1) or (-1, 1).*

a. All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k***.

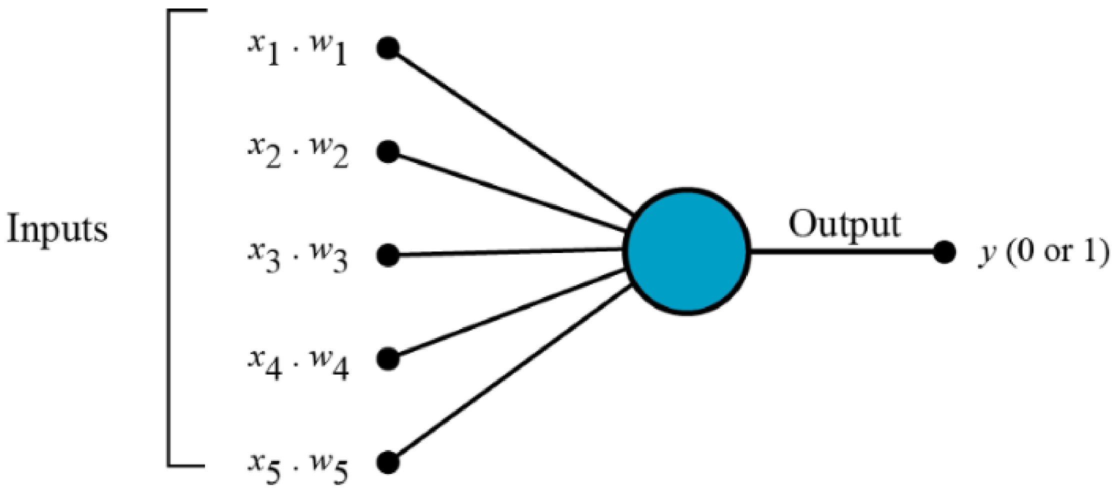


Fig: Multiplying inputs with weights for 5 inputs

b. ***Add*** all the multiplied values and call them ***Weighted Sum.***

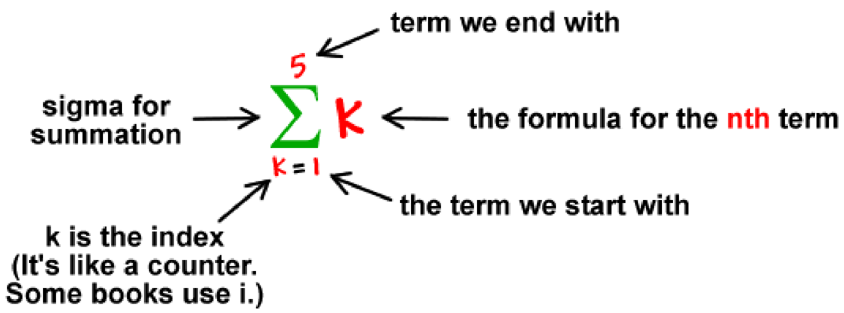
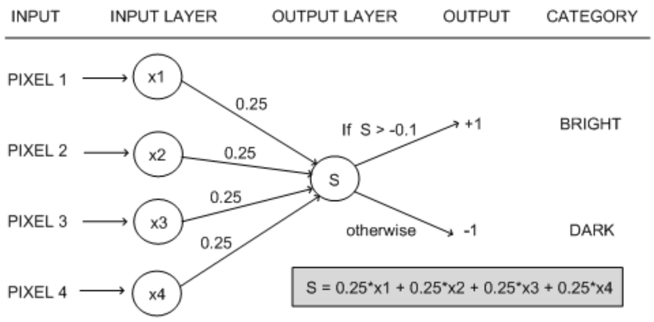


Fig: Adding with Summation

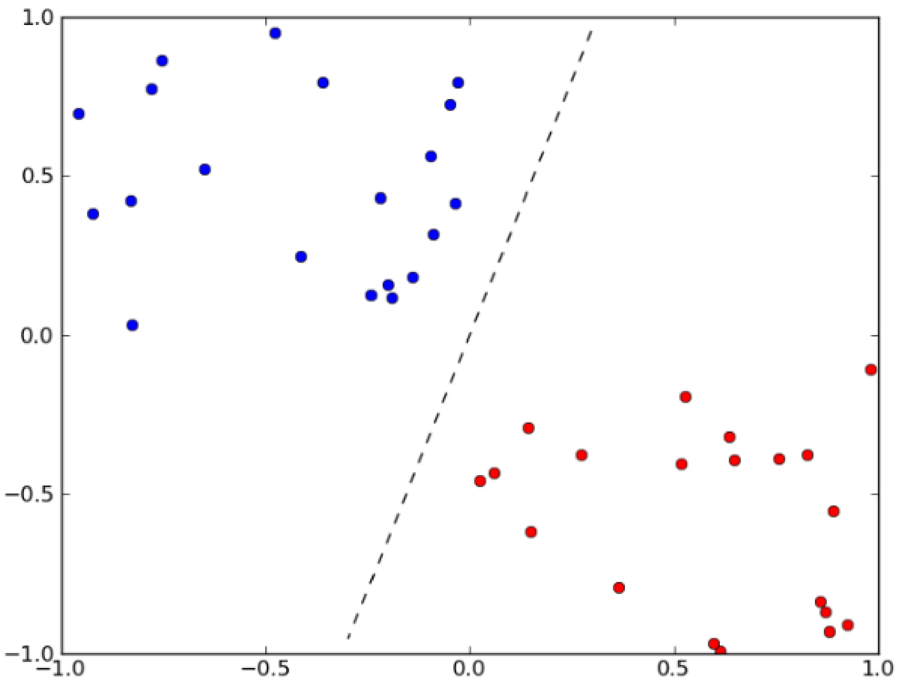
Why do we need Weights and Bias?

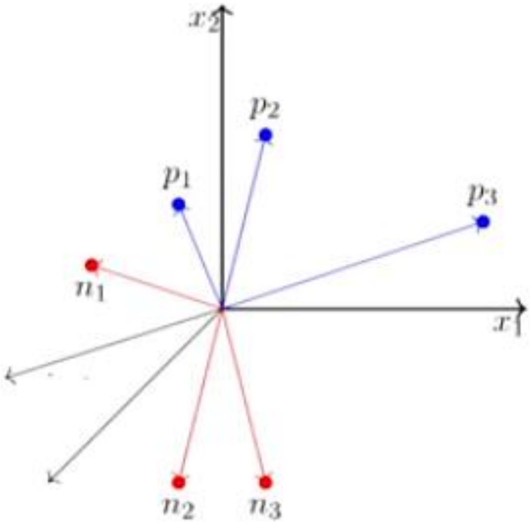
> *Weights shows the strength of the particular node.*
>
> *A bias value allows you to shift the activation function curve up or down.*



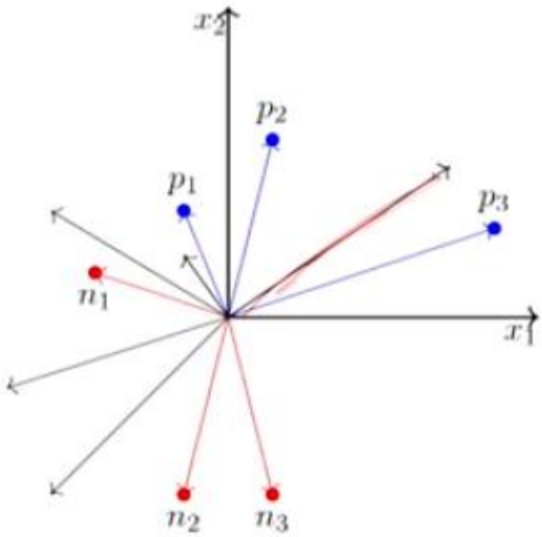
Where we use Perceptron?

> *Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.*





- We initialized $\mathbf{w}$ to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $> 90°$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $< 90°$) for all the negative points (the situation is exactly oppsite of what we actually want it to be)
- We now run the algorithm by randomly going over the points
- Randomly pick a point (say, p_2), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x}$ $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)



- We initialized $\mathbf{w}$ to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle $> 90°$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $< 90°$) for all the negative points (the situation is exactly oppsite of what we actually want it to be)
- We now run the algorithm by randomly going over the points
- Randomly pick a point (say, n_1), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

Theorem

Definition: Two sets P and N of points in an n-dimensional space are called absolutely linearly separable if $n+1$ real numbers $w_0, w_1, ..., w_n$ exist such that every point $(x_1, x_2, ..., x_n) \in P$ satisfies $\sum_{i=1}^{n} w_i * x_i > w_0$ and every point $(x_1, x_2, ..., x_n) \in N$ satisfies $\sum_{i=1}^{n} w_i * x_i < w_0$.

W2

function which
is linearly separable
OR

0 0 0
0 1 1
1 0 1
1 1 1

is not linearly separable
XOR

0 0 0
0 1 1
1 0 1
1 1 0

THERE DOESN'T EXIST A LINE
WHICH SEPARATES X-Y

MOST REAL WORLD DATA IS NOT LINEARLY SEPARABLE

> *n inputs*
> *2^2n functions*
> *- how many are linearly separable?*
>
> *2 inputs*
> *16 functions*
> *2 are not linearly separable*
> *XOR and !XOR*

HOW any BOOLEAN function CAN be REPRESENTED using A perceptron
- model/ input/ truth table/ output/

Boolean function!

REPRESENTED – feeding any value of x,y

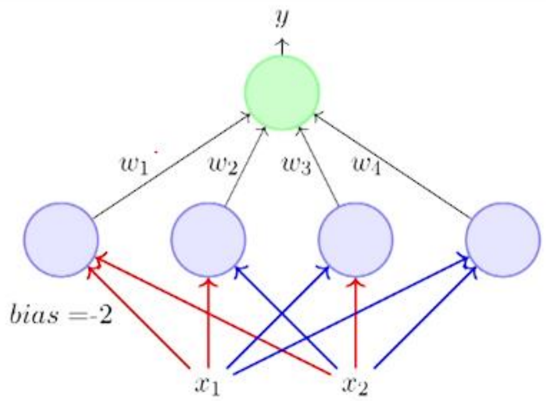
- *Network will give the same output as that of the truth table*
-

0 or -1

Linearly separable

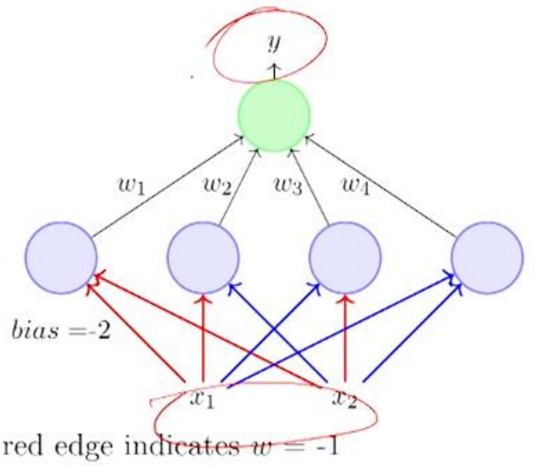
Layers in a Perceptron

Weights



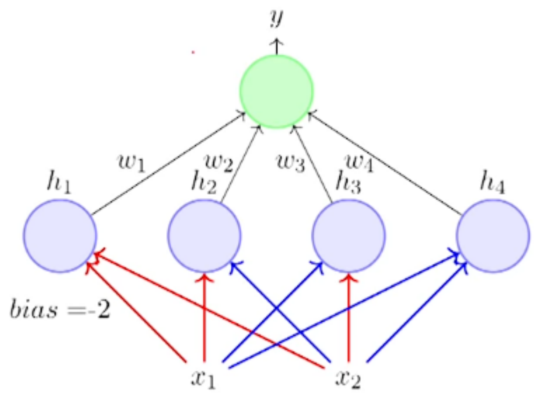
- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)
- The output of this perceptron (y) is the output of this network

- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4



red edge indicates w = -1
blue edge indicates $w = +1$

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)
- The output of this perceptron (y) is the output of this network



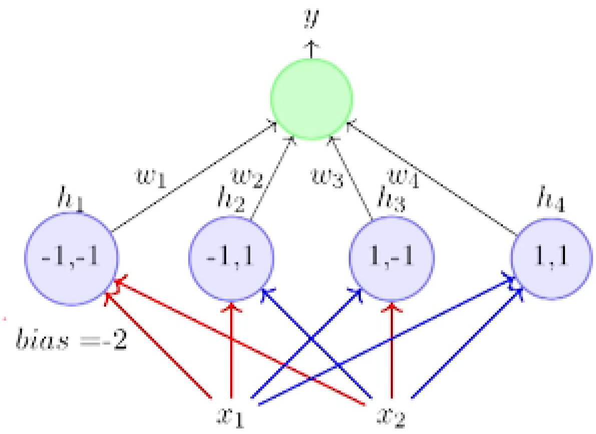
Terminology:

- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on

- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- Let us see why this network works by taking an example of the XOR function

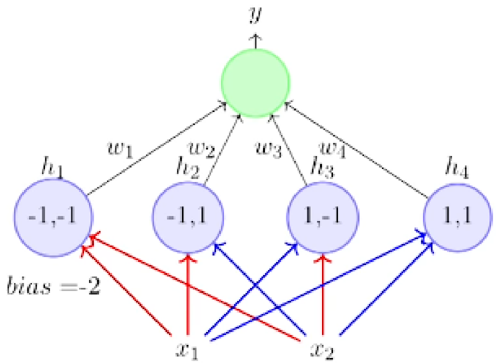
implement **any** boolean function (linearly separable or not) !



red edge indicates w = -1

blue edge indicates w = +1

- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- the fourth perceptron fires for {1,1}



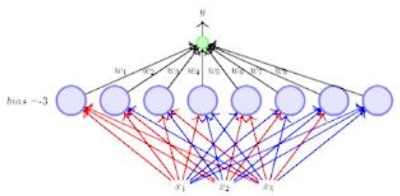
red edge indicates w = -1

blue edge indicates w = +1

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

- This results in the following four conditions to implement XOR: $w_1 < w_0, w_2 \geq w_0, w_3 \geq w_0, w_4 < w_0$



$$
\begin{array}{c}
p_1 \\ p_2 \\ \vdots \\ n_1 \\ n_2 \\ \vdots
\end{array}
\begin{bmatrix}
x_{11} & x_{12} & \dots & x_{1n} & y_1 = 1 \\
x_{21} & x_{22} & \dots & x_{2n} & y_2 = 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_{k1} & x_{k2} & \dots & x_{kn} & y_i = 0 \\
x_{j1} & x_{j2} & \dots & x_{jn} & y_j = 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
$$

- We are given this data about our past movie experience
- For each movie, we are given the values of the various factors ($x_1, x_2, \dots, x_n$) that we base our decision on and we are also also given the value of y (like/dislike)
- p_i's are the points for which the output was 1 and n_i's are the points for which it was 0
- The data may or may not be linearly separable
- But the proof that we just saw tells us that it is possible to have a network of perceptrons and learn the weights in this network such that for any given p_i or n_j the output of the network will be the same as y_i or y_j (i.e., we can separate the positive and the negative points)

Theorem

Any boolean function of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons and one output layer containing 1 perceptron

Proof (informal:) We just saw how to construct such a network

Note: A network of $2^n + 1$ perceptrons is not necessary but sufficient. For example, we already saw how to represent AND function with just 1 perceptron

Catch: As n increases the number of perceptrons in the hidden layers obviously increases exponentially

TAKEAWAY:

PERCEPTRON CAN IMPLEMENT ANY BOOLEAN FUNCTION, WHETHER LINEARLY SEPARABLE OR NOT

SIGMOID FUNCTION

OIL wells

Function which takes x (vector with n variables) and gives y

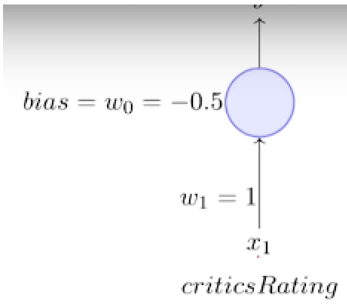
- *Whether to drill or not*
-

The story ahead ...

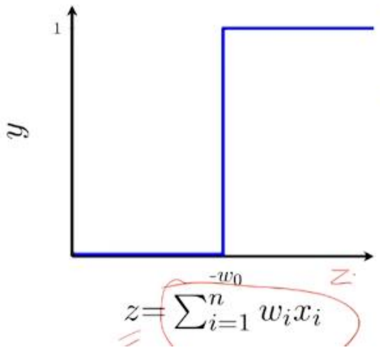
- Enough about boolean functions!
- What about arbitrary functions of the form $y = f(x)$ where $x \in \mathbb{R}^n$ (instead of $\{0, 1\}^n$) and $y \in \mathbb{R}$ (instead of $\{0, 1\}$) ?

- *From arbitrary neurons to sigmoid neurons*
  - *Real functions*
  - *Perceptron will fire when the weighted sum is greater than the threshold*

- *Networks which could represent Boolean functions*
  - *1 hidden layer minimum*
  - *Hidden layer grows exponentially!*
  - *Harsh logic used by perceptron*
  - *0.49 and 0.51 lies pretty close but treated differently by perceptron functions*
  -

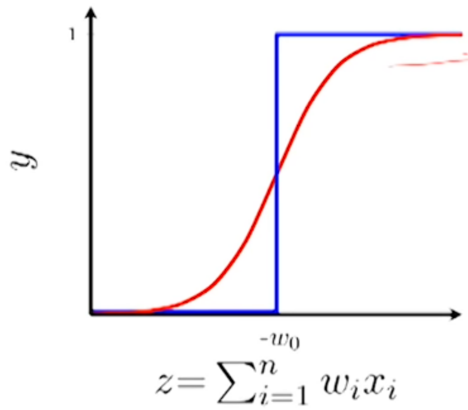


- The thresholding logic used by a perceptron is very harsh !
- For example, let us return to our problem of deciding whether we will like or dislike a movie
- Consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$?



- This behavior is not a characteristic of the specific problem we chose or the specific weight and threshold that we chose
- It is a characteristic of the perceptron function itself which behaves like a step function

Sigmoid Neuron



- Introducing sigmoid neurons where the output function is much smoother than the step function
- Here is one form of the sigmoid function called the logistic function

$$y = \frac{1}{1 + e^{-\underbrace{(w_0 + \sum_{i=1}^{n} w_i x_i)}}}$$

$$\omega^T x = \sum_{i=0}^{n} \omega_i x_i$$

$$\frac{1}{1 + e^{-\omega^T x}}$$

$$\omega^T x \to \infty$$

$i=0$

$$\frac{1}{1 + e^{-\omega^T x}}$$

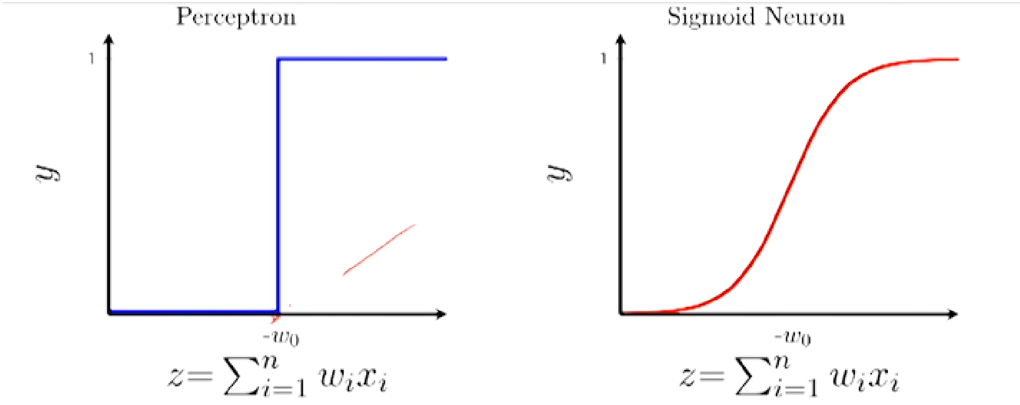
$$\omega^T x = 0$$

$$\omega^T x \to \infty$$

$$\omega^T x \to -\infty$$

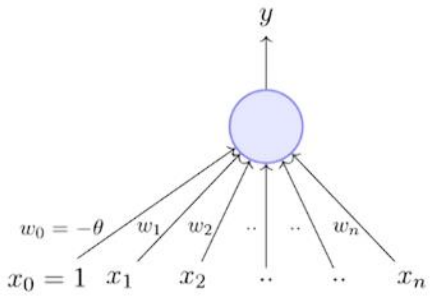
Sigmoid function range lies between 0 and 1
Same goes with probability

Not harsh anymore!

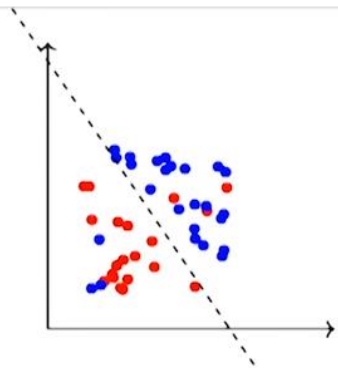


Not smooth, not continuous (at $w0$), **not differentiable**

Sigmoid (logistic) Neuron



Weights



- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?
- What would happen if we use a perceptron model to classify this data ?
- We would probably end up with a line like this ...
- This line doesn't seem to be too bad

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between $\mathbf{x}$ and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w^T x})}}$$

$$or \quad \hat{y} = \mathbf{w^T x}$$

$$or \quad \hat{y} = \mathbf{x^T W x}$$

$1\times n$ $\quad n\times n$ $\quad n\times 1.$

  or just about any functionx

- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data

Learning algorithm – algorithm for learning the parameters (w) of the model!

- *Gradient descent*
- *Perceptron learning algorithm*

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between $\mathbf{x}$ and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w^T x})}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:** One possibility is

$$\mathscr{L}(\mathbf{w}) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

The learning algorithm should aim to find a w whicl function (squared error between y and $\hat{y}$)