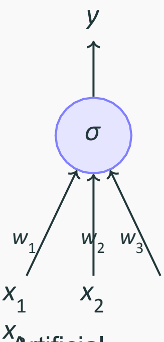
CS7015 (Deep Learning) : Lecture 2

McCulloch Pitts Neuron, Thresholding Logic, Perceptrons, Perceptron Learning Algorithm and Convergence, Multilayer Perceptrons (MLPs), Representation Power of MLPs

Mitesh M. Khapra

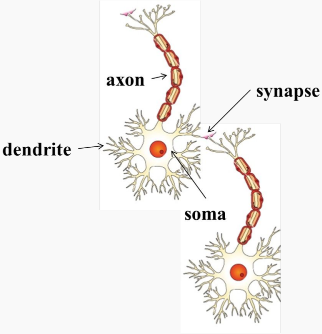
Department of Computer Science and Engineering Indian Institute of Technology Madras

Module 2.1: Biological Neurons



Artificial Neuron

- The most fundamental unit of a deep neural network is called an *artificial neuron*
- Why is it called a neuron ? Where does the inspiration come from ?
- The inspiration comes from biology (more specifically, from the *brain*)
- *biological neurons* = *neural cells* = *neural processing units*
- We will first see what a biological neuron looks like ...

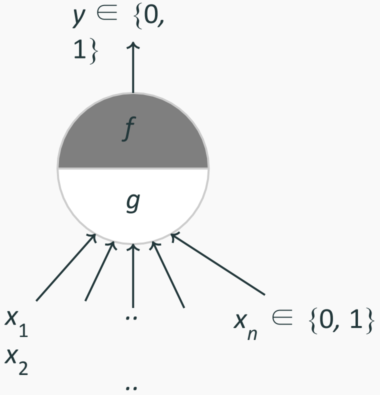


Biological
Neurons*

- **dendrite:** receives signals from other neurons
- **synapse:** point of connection to other neurons
- **soma:** processes the information
- **axon:** transmits the output of this neuron

*Image adapted from https://cdn.vectorstock.com/i/composite/12,25/neuron-cell-vector-81225.jpg

Module 2.2: McCulloch Pitts Neuron



- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- g aggregates the inputs and the function f takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- $y = 0$ if any x_i is inhibitory, else

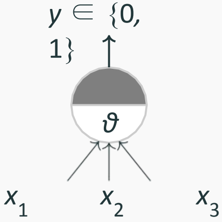
$$g(x_1, x_2, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad \text{if} \quad g(\mathbf{x}) \geq \vartheta$$

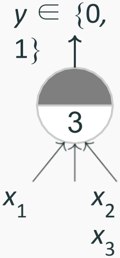
$$= 0 \quad \text{if} \quad g(\mathbf{x}) < \vartheta$$

- ϑ is called the thresholding parameter

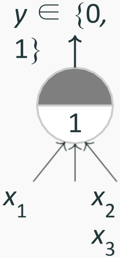
Let us implement some boolean functions using this McCulloch Pitts (MP) neuron ...



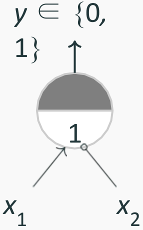
A McCulloch Pitts unit



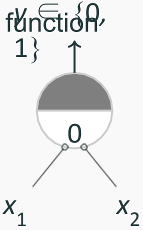
AND function



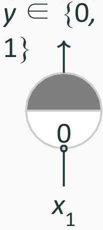
OR function



x_1 AND !x_2*



NOR function

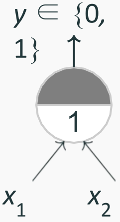


NOT function

*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0

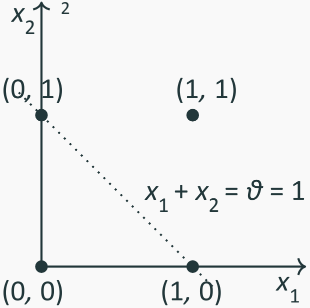
- Can any boolean function be represented using a McCulloch Pitts unit ?
- Before answering this question let us first see the geometric interpretation of a MP unit

  ...

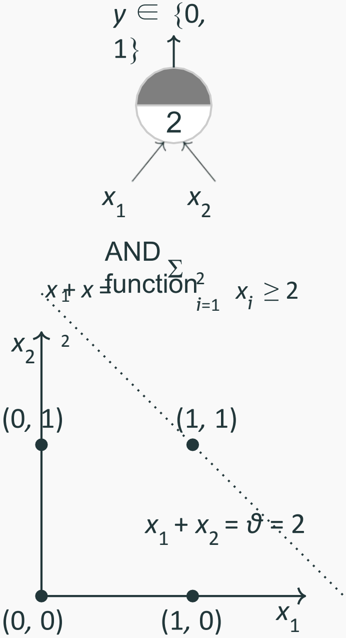


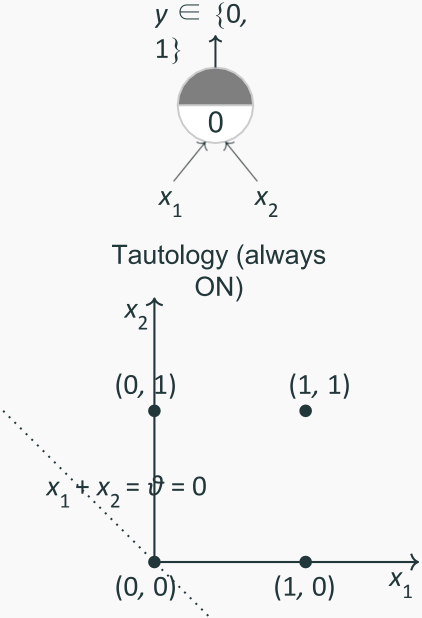
OR function

$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$

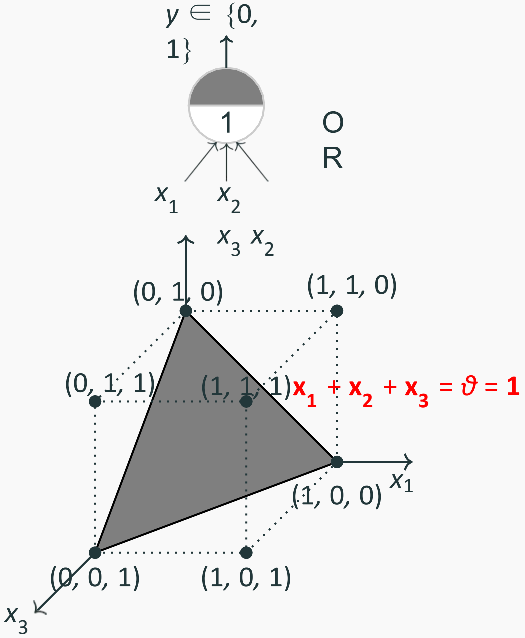


- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves
- Points lying on or above the line $\sum_{i=1}^{n} x_i - \vartheta = 0$ and points lying below this line
- In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} x_i < \vartheta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^{n} x_i \geq \vartheta$) of this line
- Let us convince ourselves about this with a few more examples (if it is not already clear from the math)


$y \in \{0, 1\}$
2
x_1
x_2
AND function
$\sum_{i=1}^{2} x_i \geq 2$
(0, 1)
(1, 1)
$x_1 + x_2 = \vartheta = 2$
(0, 0)
(1, 0)


$y \in \{0, 1\}$
0
x_1
x_2
Tautology (always ON)
(0, 1)
(1, 1)
$x_1 + x_2 = \vartheta = 0$
(0, 0)
(1, 0)

- What if we have more than 2 inputs?
- Well, instead of a line we will have a plane
- For the OR function, we want a plane such that the point (0,0,0) lies on one side and the remaining 7 points lie on the other side of the plane



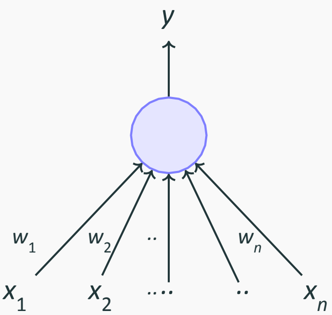
The story so far ...

- A single McCulloch Pitts Neuron can be used to represent boolean functions which are linearly separable
- Linear separability (for boolean functions) : There exists a line (plane) such that all in- puts which produce a 1 lie on one side of the line (plane) and all inputs which produce a 0 lie on other side of the line (plane)

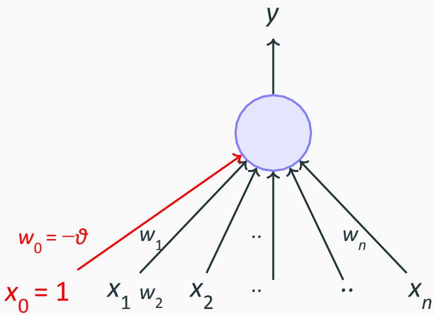
Module 2.3: Perceptron

The story ahead ...

- What about non-boolean (say, real) inputs ?
- Do we always need to hand code the threshold ?
- Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?
- What about functions which are not linearly separable ?



- Frank Rosenblatt, an American psychologist, pro- posed the **classical perceptron** model (1958)
- A more general computational model than McCulloch–Pitts neurons
- **Main differences:** Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values
- Refined and carefully analyzed by Minsky and Pa- pert (1969) - their model is referred to as the **per- ceptron** model here



A more accepted convention,

$$y = 1 \quad if \quad \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \quad \sum_{i=0}^{n} w_i * x_i < 0$$

where, $x_0 = 1$ *and* $w_0 = -\vartheta$

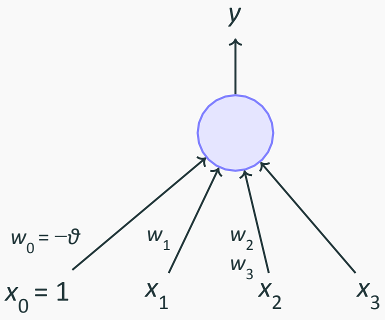
Rewriting the above,

$$y = 1 \quad if \quad \sum_{i=1}^{n} w_i * x_i \geq \vartheta$$
$$= 0 \quad if \quad \sum_{i=1}^{n} w_i * x_i < \vartheta$$

$$y = 1 \quad if \quad \sum_{i=1}^{n} w_i * x_i - \vartheta \geq 0$$
$$= 0 \quad if \quad \sum_{i=1}^{n} w_i * x_i - \vartheta < 0$$

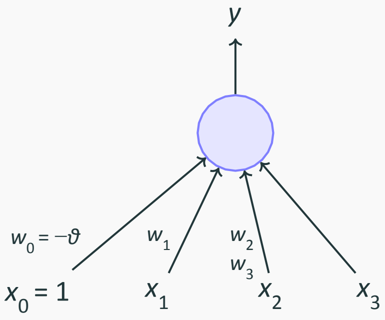
We will now try to answer the following questions:

- Why are we trying to implement boolean functions?
- Why do we need weights ?
- Why is $w_0 = -\vartheta$ called the bias ?



x_1 = *isActorDamon* x_2 = *isGenreThriller* x_3 = *isDirectorNolan*

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for sim- plicity)
- Based on our past viewing experience (**data**), we may give a high weight to *isDirectorNolan* as compared to the other inputs
- Specifically, even if the actor is not *Matt Damon* and the genre is not *thriller* we would still want to cross the threshold ϑ by assigning a high weight to *isDirectorNolan*



x_1 = *isActorDamon* x_2 = *isGenreThriller* x_3 = *isDirectorNolan*

- w_0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director [$\vartheta = 0$]
- On the other hand, a selective viewer may only watch thrillers starring Matt Damon and directed by Nolan [$\vartheta = 3$]
- The weights ($w_1, w_2, \ldots, w_n$) and the bias (w_0) will depend on the data (viewer history in this case)

What kind of functions can be implemented using the perceptron? Any difference from McCulloch Pitts neurons?

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} x_i < 0$$

Perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

- 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 (includ- ing threshold) can be learned and the inputs can be real valued
- We will first revisit some boolean functions and then see the perceptron learning algorithm (for learning weights)

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

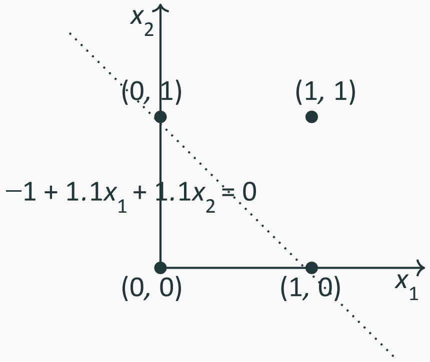
$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0$$

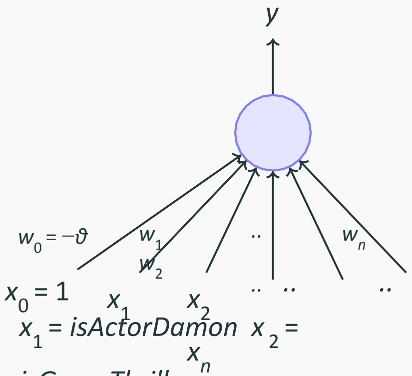
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0$$

- One possible solution to this set of inequalities is $w_0 = -1$, $w_1 = 1.1$, $w_2 = 1.1$ (and various other solutions are possible)



- Note that we can come up with a similar set of inequalities and find the value of ϑ for a McCul- loch Pitts neuron also (Try it!)

Module 2.5: Perceptron Learning Algorithm



$x_1 = isActorDamon$ $x_2 = isGenreThriller$ $x_3 = isDirectorNolan$

$x_4 = imdbRating\ (scaled\ to\ 0\ to\ 1)$

... ...

$x_n = criticsRating\ (scaled\ to\ 0\ to\ 1)$

- Let us reconsider our problem of deciding whether to watch a movie or not
- Suppose we are given a list of m movies and a la- bel (class) associated with each movie indicating whether the user liked this movie or not : binary decision
- Further, suppose we represent each movie with n features (some boolean, some real valued)
- We will assume that the data is linearly separable and we want a perceptron to learn how to make this decision
- In other words, we want the perceptron to find the equation of this separating plane (or find the val- ues of $w_0, w_1, w_2, .., w_m$)

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$

$N \leftarrow inputs \quad with \quad label \quad 0;$

Initialize **w** randomly;

while $!convergence$ **do**

  Pick random $\mathbf{x} \in P \cup N$;

  if $\mathbf{x} \in P$ *and* $\sum_{i=0}^{n} w_i * x_i < 0$ **then**

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

  if $\mathbf{x} \in N$ *and* $\sum_{i=0}^{n} w_i * x_i \geq 0$ **then**

    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

  end

end

//the algorithm converges when all the inputs are classified correctly

- Why would this work ?

- Consider two vectors **w** and **x**

$$\mathbf{w} = [w_0, w_1, w_2, \ldots, w_n]$$

$$\mathbf{x} = [1, x_1, x_2, \ldots, x_n]$$

$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

- We can thus rewrite the perceptron rule as

$$y = 1 \quad if \quad \mathbf{w}^T\mathbf{x} \geq 0$$

$$= 0 \quad if \quad \mathbf{w}^T\mathbf{x} < 0$$

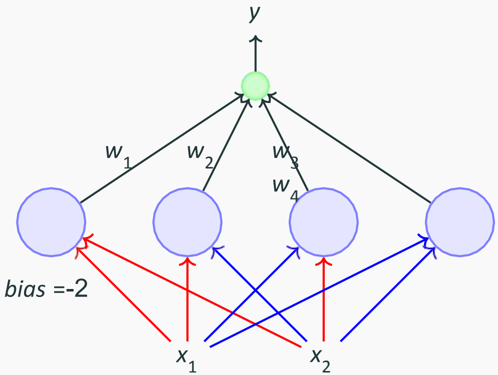
- We are interested in finding the line $\mathbf{w}^T\mathbf{x} = 0$ which divides the input space into two halves
- Every point ($\mathbf{x}$) on this line satisfies the equation $\mathbf{w}^T\mathbf{x} = 0$
- What can you tell about the angle (α) between **w** and any point ($\mathbf{x}$) which lies on this line ?
- The angle is 90° ($\because \cos\alpha = \frac{w^T x}{||w||||x||} = 0$)
- Since the vector **w** is perpendicular to every point on the line it is actually per- pendicular to the line itself

- How many boolean functions can you design from 2 inputs ?
- Let us begin with some easy ones which you already

x_1	x_2	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

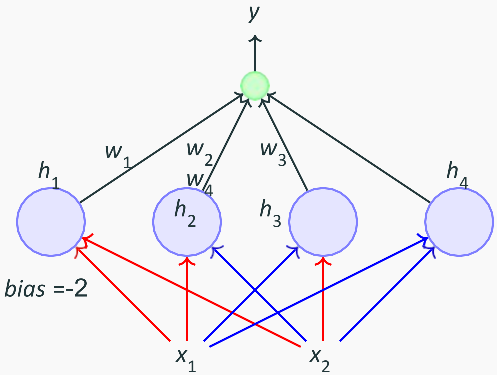
- Of these, how many are linearly separable ? (turns out all except XOR and !XOR - feel free to verify)
- In general, how many boolean functions can you have for n inputs ? 2^{2^n}
- How many of these 2^{2^n} functions are not linearly separable ? For the time being, it suffices to know that at least some of these may not be linearly inseparable (I encourage you to figure out the exact answer :-))

Module 2.8: Representation Power of a Network of Perceptrons



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

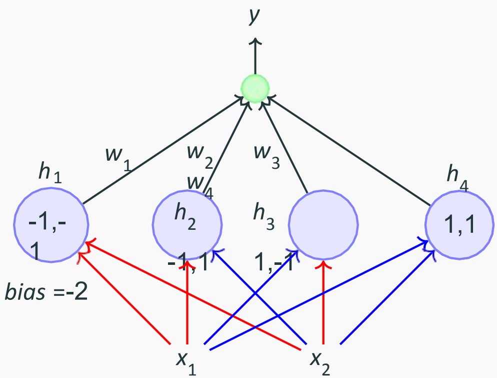
- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each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



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

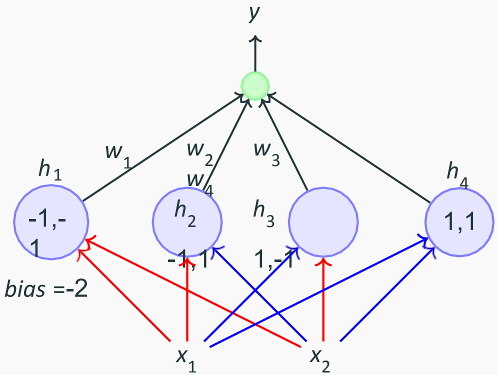
Terminology:

- 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 hid- den layer are denoted by h_1, h_2, h_3, h_4
- The red and blue edges are called layer 1 weights
- w_1, w_2, w_3, w_4 are called layer 2 weights



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

- We claim that this network can be used to im- plement **any** boolean function (linearly separ- able or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean func- tion can be represented by this network
- Astonishing claim! Well, not really, if you un- derstand 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

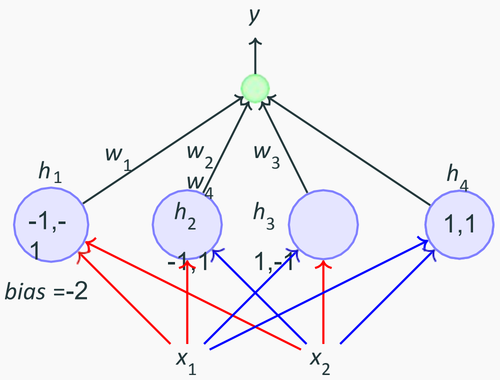


red edge indicates w = -1 blue edge indicates w = +1

- 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: $w_1 < w_0$, $w_2 \geq w_0$, $w_3 \geq w_0$, $w_4 < w_0$
- Unlike before, there are no contradictions now and the system of inequalities can be satisfied
- Essentially each w_i is now responsible for one of the 4 possible inputs and can be adjusted

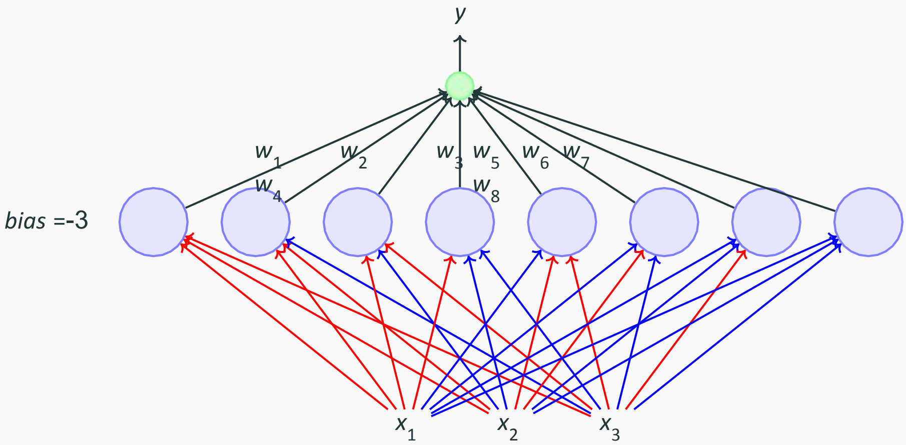


red edge indicates w = -1 blue edge indicates w = +1

- It should be clear that the same network can be used to represent the remaining 15 boolean functions also
- Each boolean function will result in a different set of non-contradicting inequalities which can be satisfied by appropriately setting w_1, w_2, w_3, w_4
- Try it!

- What if we have more than 3 inputs ?

- Again each of the 8 perceptorns will fire only for one of the 8 inputs
- Each of the 8 weights in the second layer is responsible for one of the 8 inputs and can be adjusted to produce the desired output for that input



- What if we have n inputs ?

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

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

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

- Again, why do we care about boolean functions ?
- How does this help us with our original problem: which was to predict whether we like a movie or not?

The story so far ...

- Networks of the form that we just saw (containing, an input, output and one or more hidden layers) are called Multilayer Perceptrons (MLP, in short)
- More appropriate terminology would be"Multilayered Network of Perceptrons" but MLP is the more commonly used name
- The theorem that we just saw gives us the representation power of a MLP with a single hidden layer
- Specifically, it tells us that a MLP with a single hidden layer can represent **any** boolean function