Deep Learning
Neuron to Perceptron

Puneet Kumar Jain

CSE Department
National Institute of Technology Rourkela

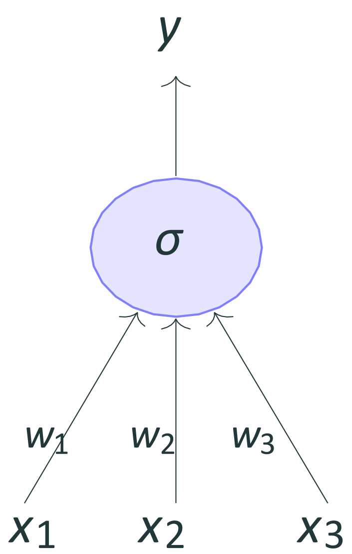
References:

The Slides are prepared from the following major source:

- **"CS7105-Deep Learning" by Mitesh M. Khapra, IIT Madras. http://www.cse.iitm.ac.in/~miteshk/CS7015_2018.html**

- **Biological Neurons**

Neuron



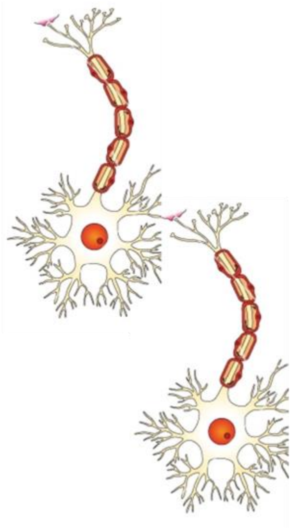
Artificial Neuron

The most fundamental unit of a deep neural network is called an *artificial neuron*

The inspiration comes from biology (more specifically, from the *brain*)

biological neurons = neural cells = neural processing units

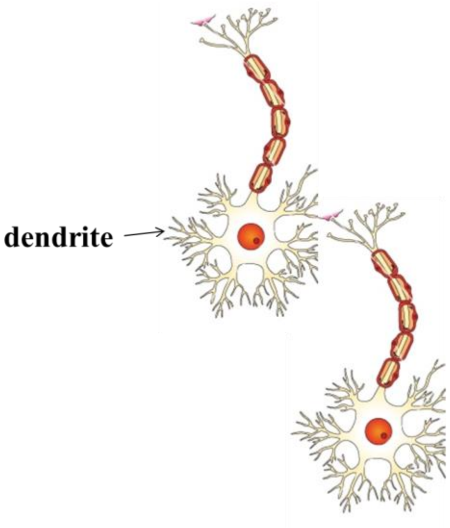
Biological neuron



Biological Neurons*

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

Biological neuron

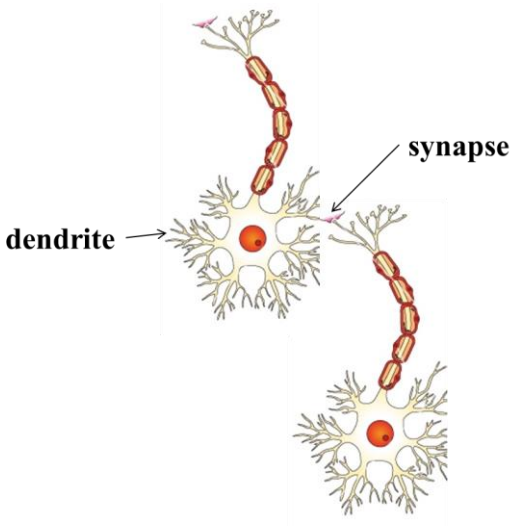


Biological Neurons*

dendrite: receives signals from other neurons

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

Biological neuron



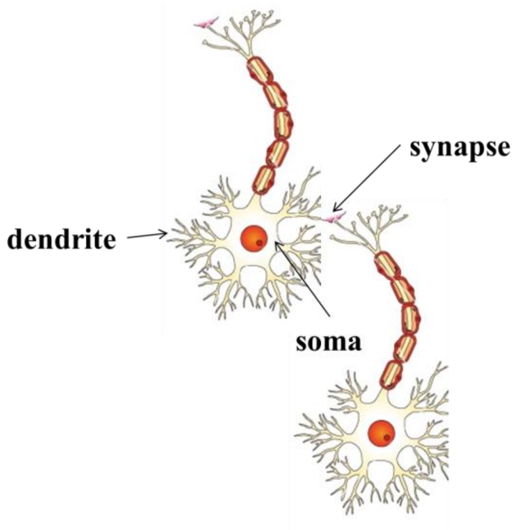
Biological Neurons*

dendrite: receives signals from other neurons

synapse: point of connection to other neurons

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

Biological neuron



Biological Neurons*

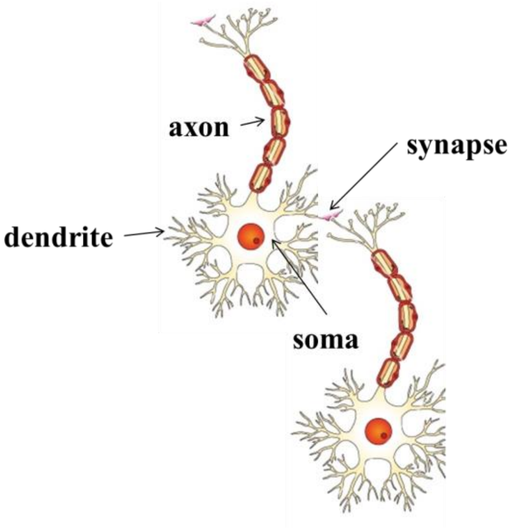
dendrite: receives signals from other neurons

synapse: point of connection to other neurons

soma: processes the information

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

Biological neuron



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

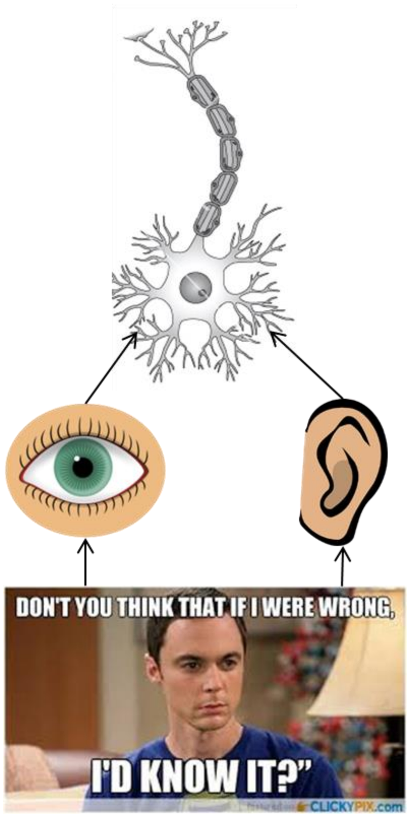
*Image adapted from https://cdn.vectorstock.com/i/composite/12,25/neuron-cell-vector-81225.jpg

Biological neuron

Let us see a very cartoonish illustration of how a neuron works

Our sense organs interact with the out- side world

Biological neuron

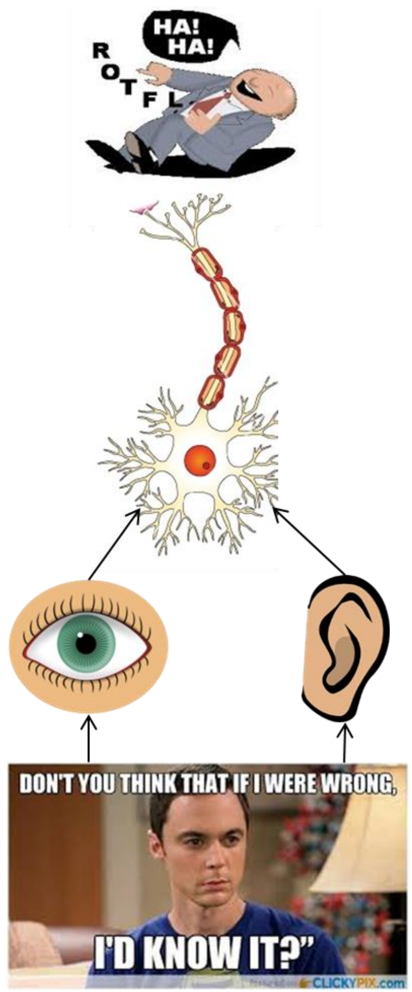


Let us see a very cartoonish illustration of how a neuron works

Our sense organs interact with the outside world

They relay information to the neurons

Biological neuron



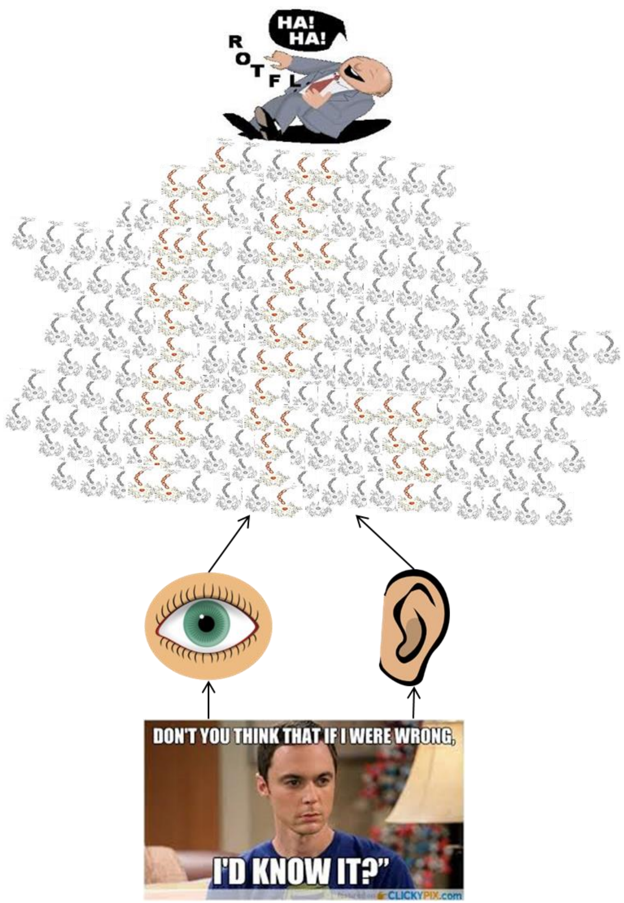
Let us see a very cartoonish illustration of how a neuron works

Our sense organs interact with the outside world

They relay information to the neurons

The neurons (may) get activated and produces a response (laughter in this case)

Biological neuron



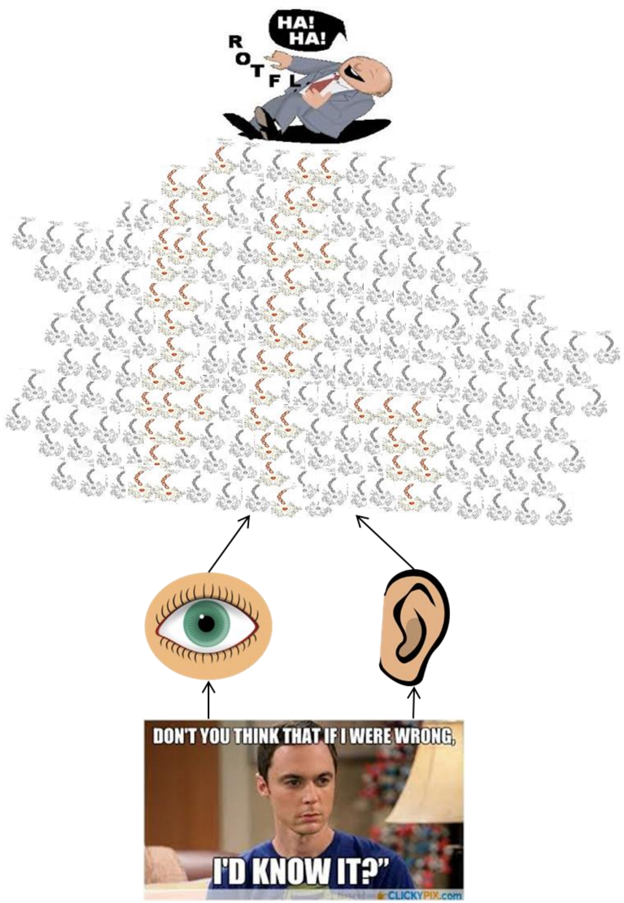
Of course, in reality, there is a massively parallel interconnected network of neurons

The sense organs relay information to the lowest layer of neurons

Some of these neurons may fire (in red) in response to this information and in turn relay information to other neurons they are connected to

These neurons may also fire (again, in red) and the process continues, eventually resulting in a response (laughter in this case)

Biological neuron



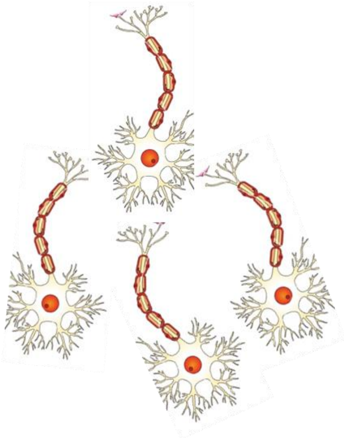
An average human brain has around 10^{11} (100 billion) neurons!

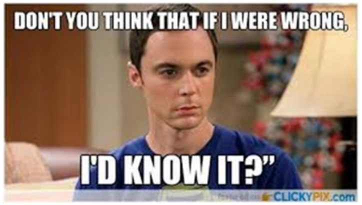
This massively parallel network also ensures that there is division of work

Biological neuron

This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus



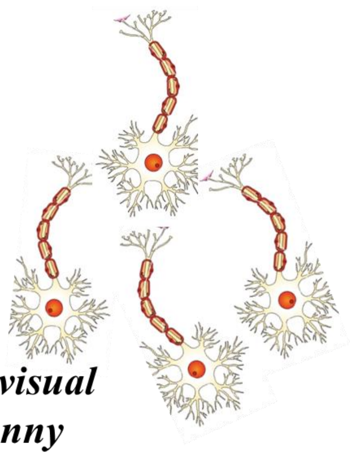


A simplified illustration

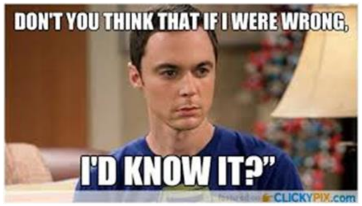
Biological neuron

This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus



fires if visual is funny

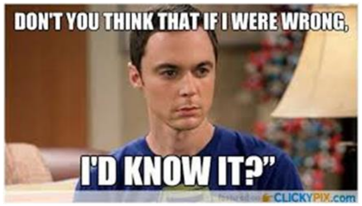


A simplified illustration

Biological neuron

This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus

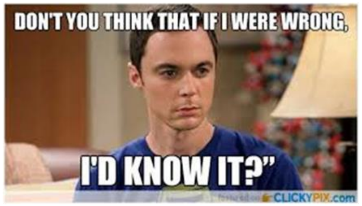


A simplified illustration

Biological neuron

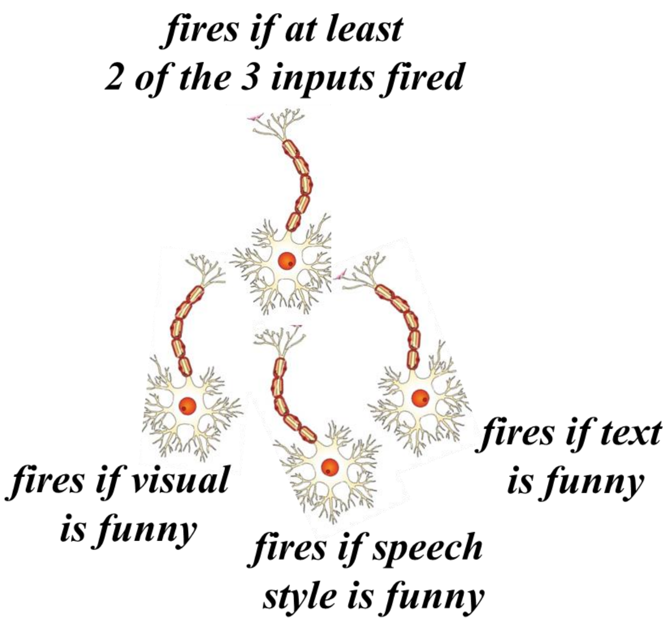
This massively parallel network also ensures that there is division of work

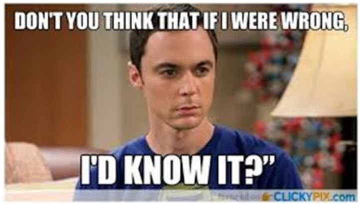
Each neuron may perform a certain role or respond to a certain stimulus



A simplified illustration

Biological neuron



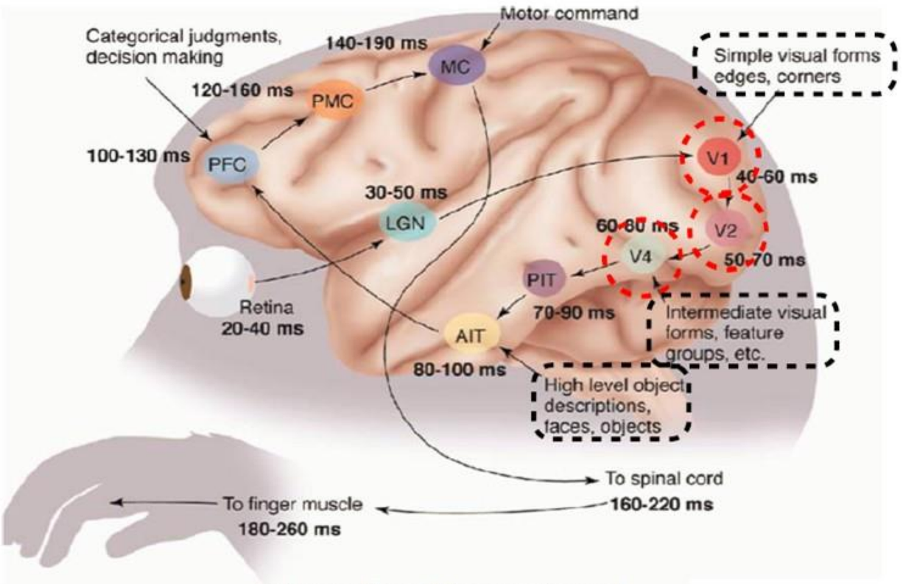


A simplified illustration

This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus

Biological neuron (hierarchy)



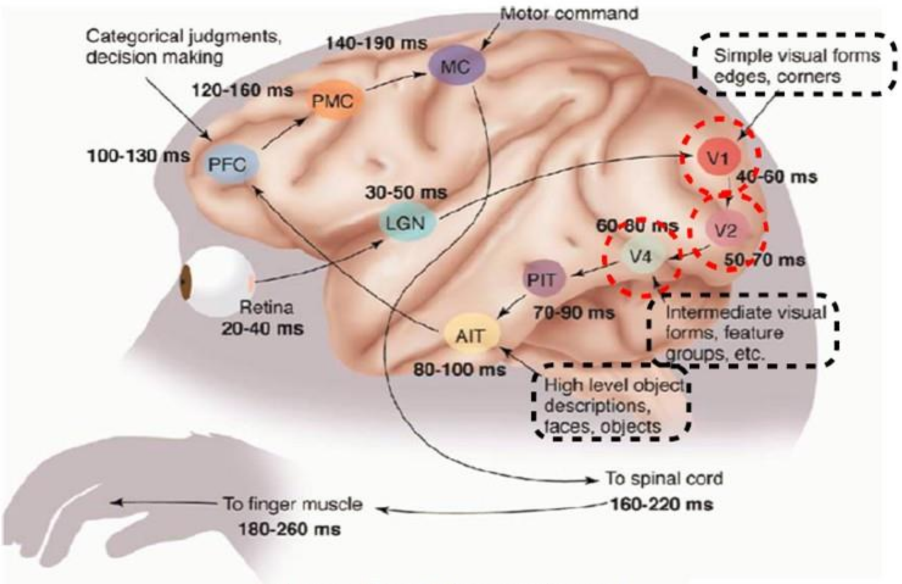
[picture from Simon Thorpe]

Layer 1: detect edges & corners

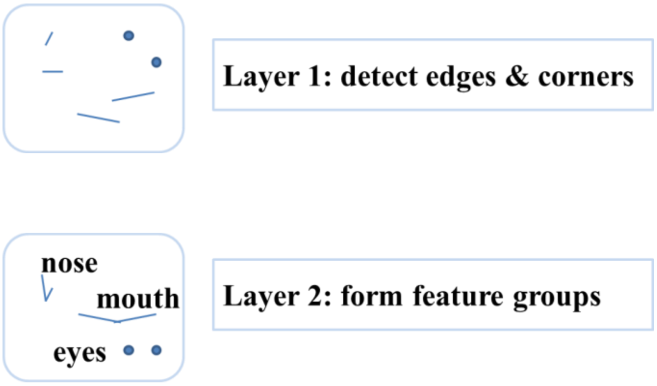
Sample illustration of hierarchical processing*

*Idea borrowed from Hugo Larochelle's lecture slides

Biological neuron (hierarchy)



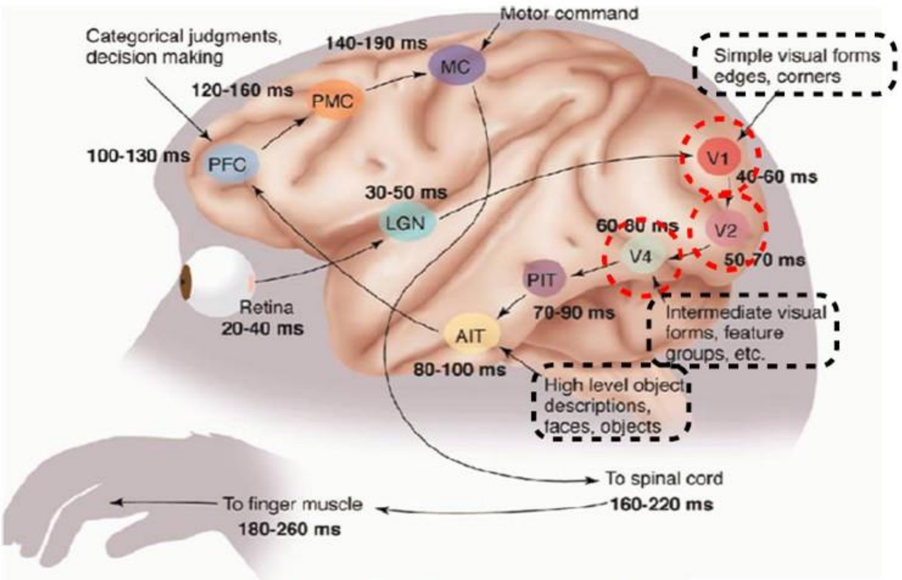
[picture from Simon Thorpe]



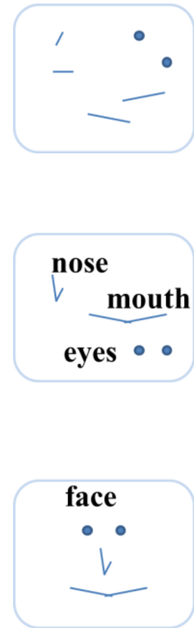
Sample illustration of hierarchical processing*

*Idea borrowed from Hugo Larochelle's lecture slides

Biological neuron (hierarchy)



[picture from Simon Thorpe]



Layer 1: detect edges & corners

Layer 2: form feature groups

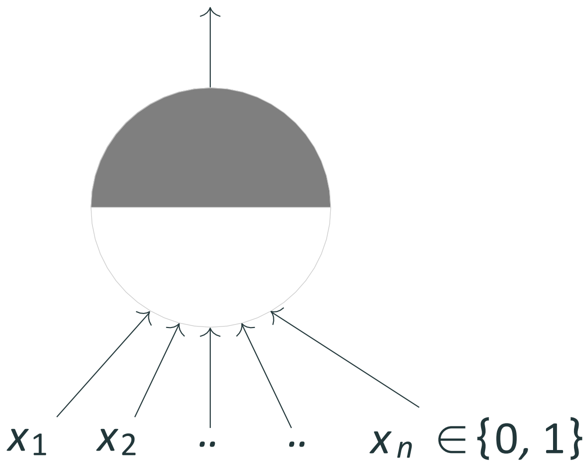
Layer 3: detect high level objects, faces, etc.

Sample illustration of hierarchical processing*

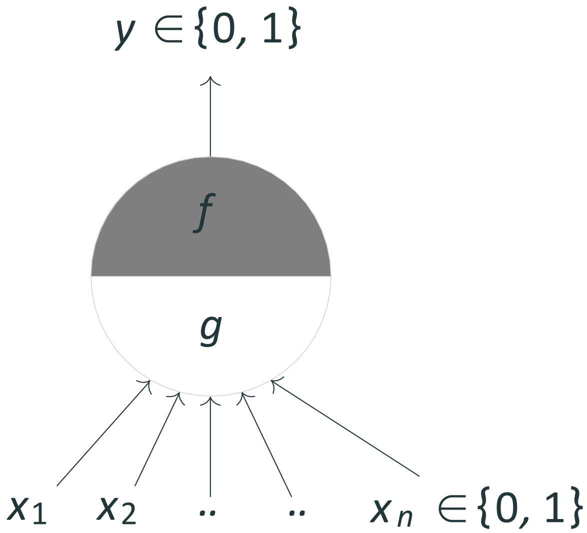
---

*Idea borrowed from Hugo Larochelle's lecture slides

- **McCulloch Pitts Neuron**

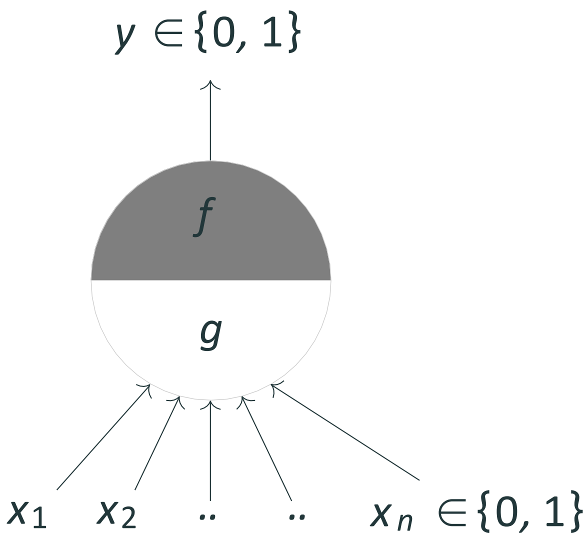


McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)



McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)

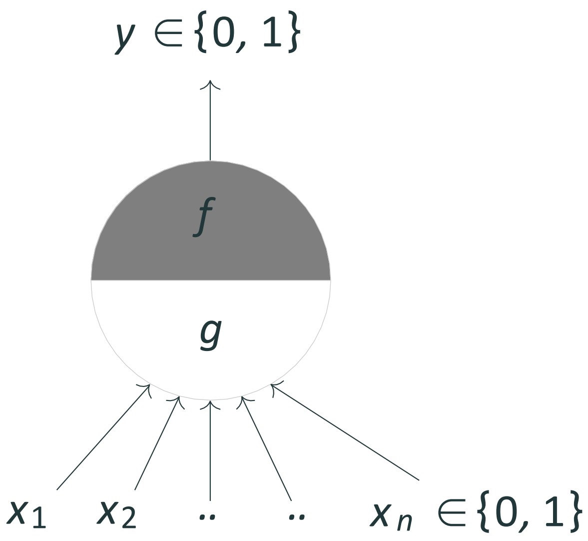
g aggregates the inputs and the function f takes a decision based on this aggregation



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

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



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

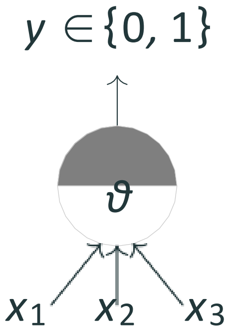
$$g(x_1, x_2, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \vartheta \\ &= 0 \quad if \quad g(\mathbf{x}) < \vartheta \end{aligned}$$

ϑ is called the thresholding parameter

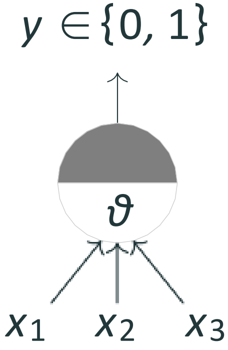
Let us implement some

boolean functions using MP neuron ...

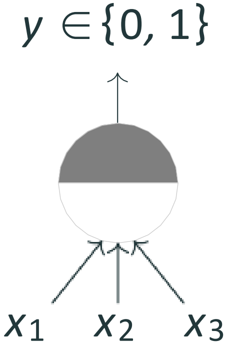


A McCulloch Pitts unit

boolean functions using MP neuron …

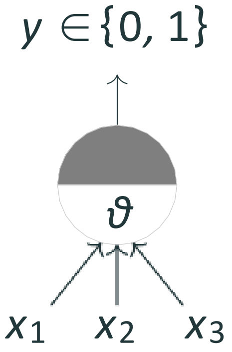


A McCulloch Pitts unit

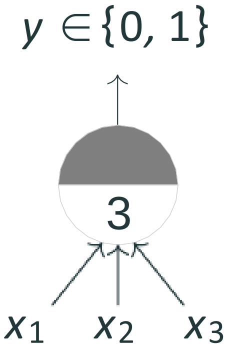


AND function

boolean functions using MP neuron ...

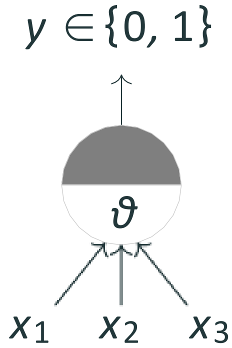


A McCulloch Pitts unit

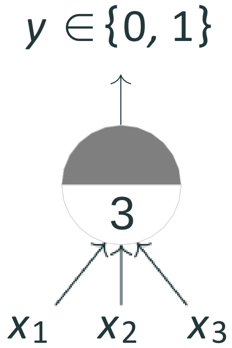


AND function

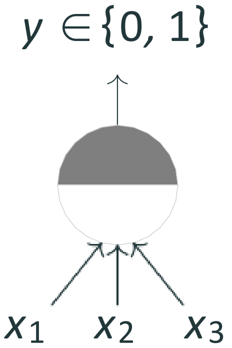
boolean functions using MP neuron ...



A McCulloch Pitts unit

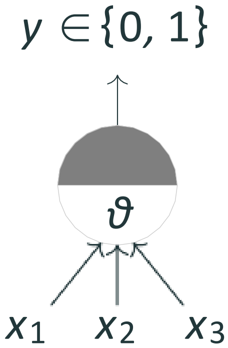


AND function

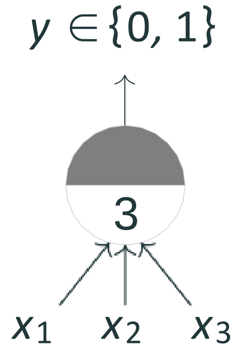


OR function

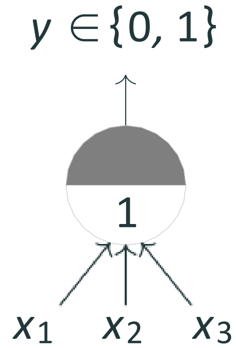
boolean functions using MP neuron ...



A McCulloch Pitts unit

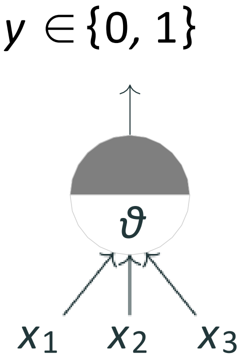


AND function

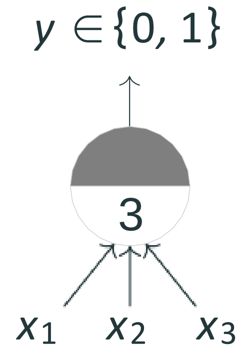


OR function

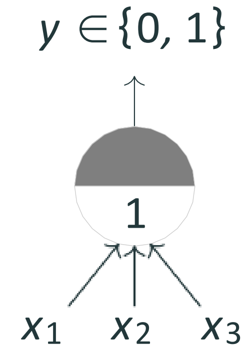
boolean functions using MP neuron ...



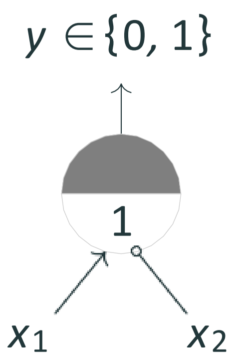
A McCulloch Pitts unit



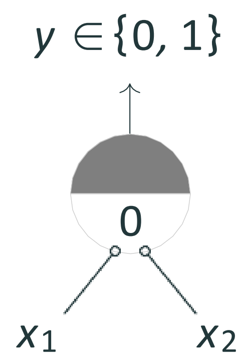
AND function



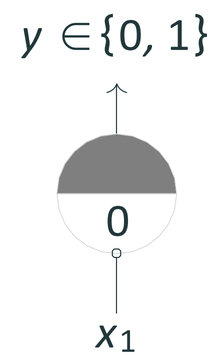
OR function



x_1 AND $!x_2$*



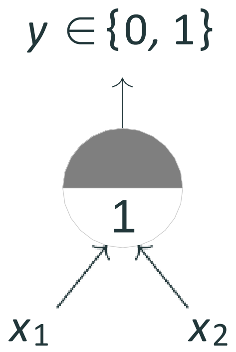
NOR function



NOT function

---

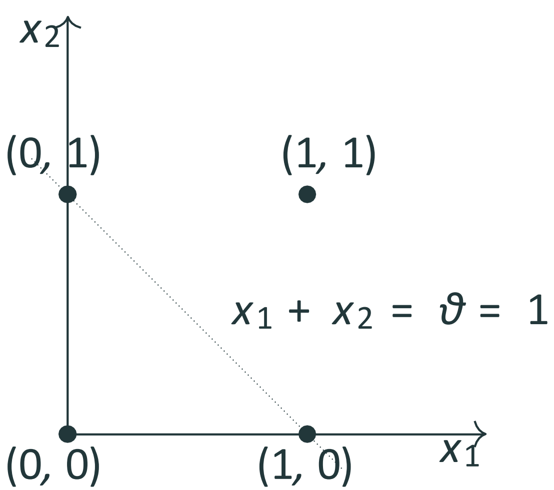
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0



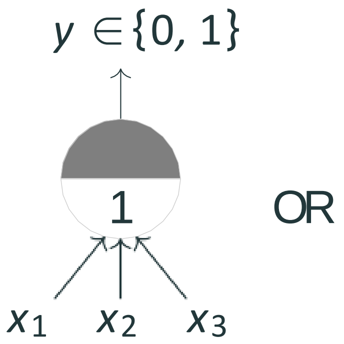
OR function

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

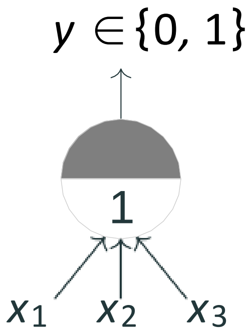
A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves



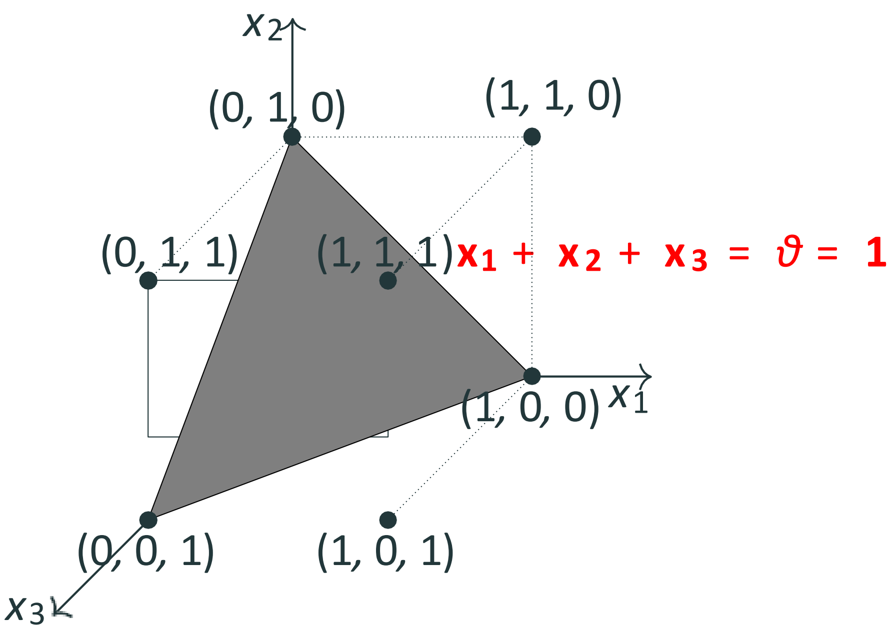
Points lying on or above the line $\sum_{i=1}^{n} x_i - \vartheta = 0$ and points lying below this line



What if we have more than 2 inputs?



OR



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

The story ahead …

What about non-boolean (say, real) inputs ?

The story ahead ...

What about non-boolean (say, real) inputs ?

Do we always need to hand code the threshold ?

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 ?

The story ahead ...

What about non-boolean (say, real) inputs ?

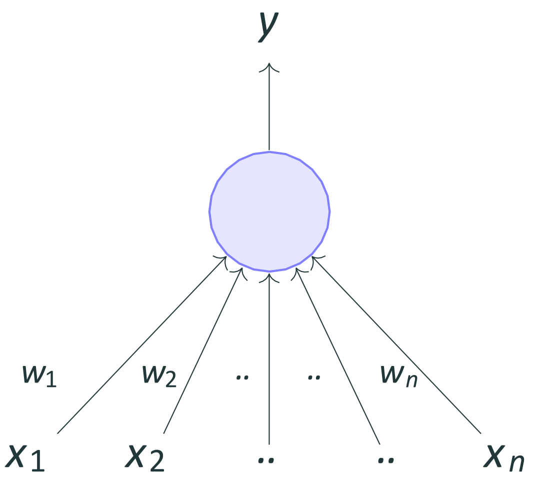
Do we always need to hand code the threshold ?

Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?

What about functions which are not linearly separable ?

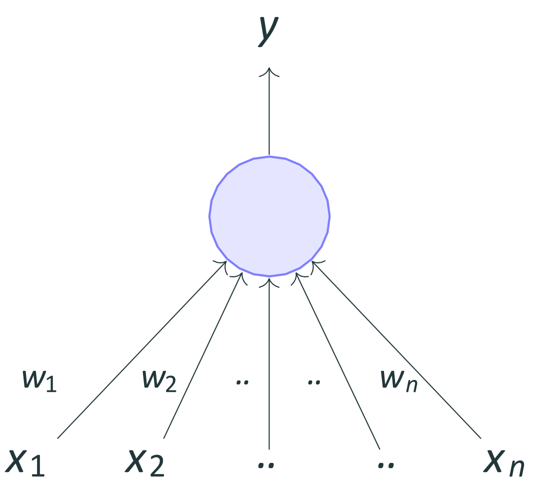
- **Perceptron**

Perceptron



Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)

Perceptron



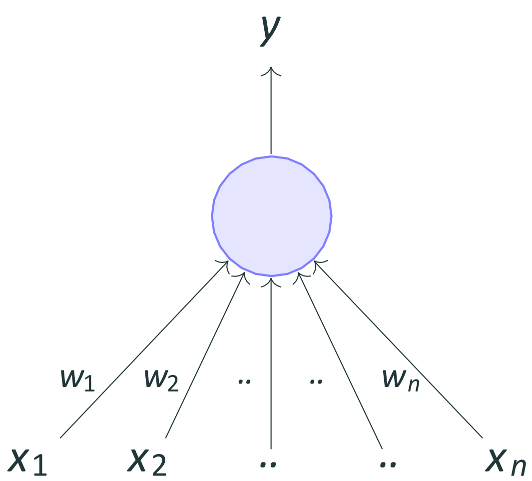
Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)

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 Papert (1969) - their model is referred to as the **perceptron** model here

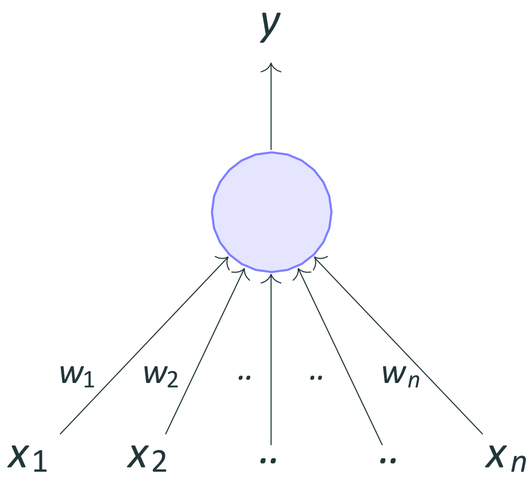
Perceptron



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

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

Perceptron

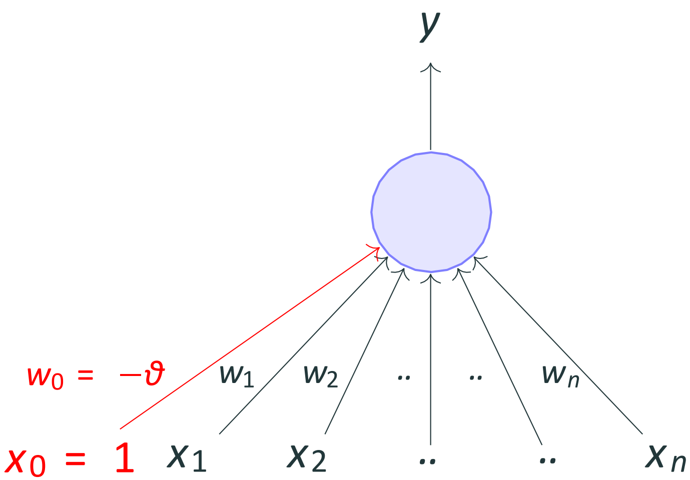


$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \vartheta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \vartheta$$

Rewriting the above,

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

Perceptron



A more accepted convention,

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

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

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

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

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

Rewriting the above,

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

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

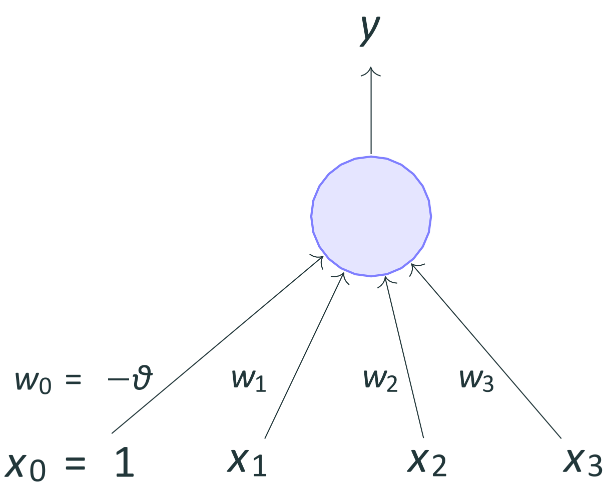
Answer the following questions

Why are we trying to implement boolean functions?

Why do we need weights ?

Why is $w_0 = -\vartheta$ called the bias ?

Why do we need weights ?



$x_1 = isActorDamon$

$x_2 = isGenreThriller$

$x_3 = isDirectorNolan$

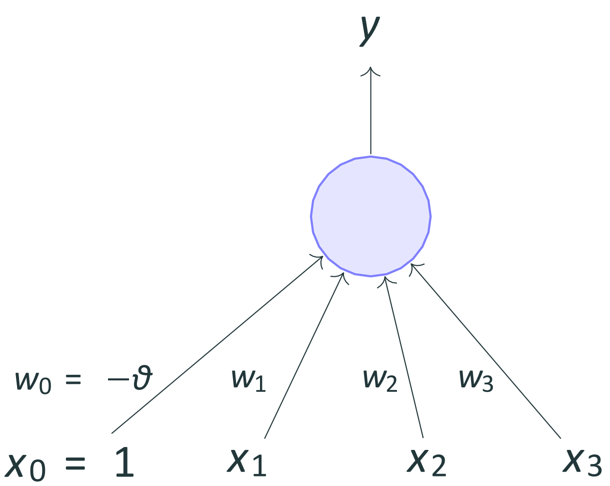
Consider the task of predicting whether we would like a movie or not

Suppose, we base our decision on 3 inputs (binary, for simplicity)

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*

Why is $w_0 = -\theta$ called the bias ?



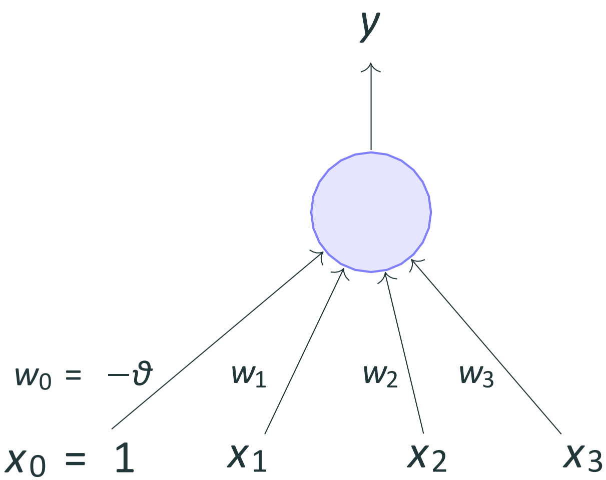
$x_1 = isActorDamon$

$x_2 = isGenreThriller$

$x_3 = isDirectorNolan$

w_0 is called the bias as it represents the prior (prejudice)

Why is $w_0 = -\theta$ called the bias ?



$x_1 = isActorDamon$

$x_2 = isGenreThriller$

$x_3 = isDirectorNolan$

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 ($w_1, w_2, ..., w_n$) and the bias (w_0) will depend on the data (viewer history in this case)

Perceptron vs McCulloch Pitts Neuron

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

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

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

A perceptron separates the input space into two halves

In other words, a single perceptron can only be used to implement linearly separable functions

Then what is the difference?

Perceptron

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

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

Perceptron vs McCulloch Pitts Neuron

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

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

A perceptron separates the input space into two halves

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

perceptron learning algorithm (for learning weights)

Learning the 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	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

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

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

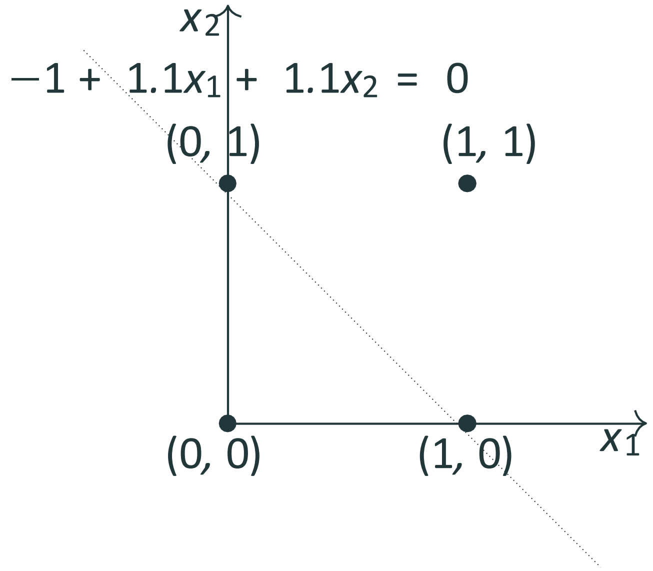
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0$$

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

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)

Errors and Error Surfaces

Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2

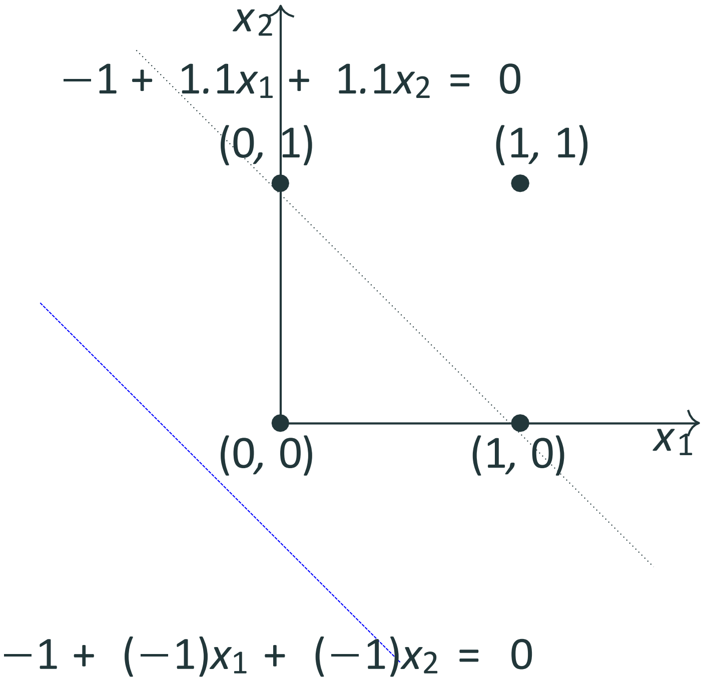


Errors and Error Surfaces

Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2

Say, $w_1 = -1, w_2 = -1$

What is wrong with this line? We make an error on 1 out of the 4 inputs



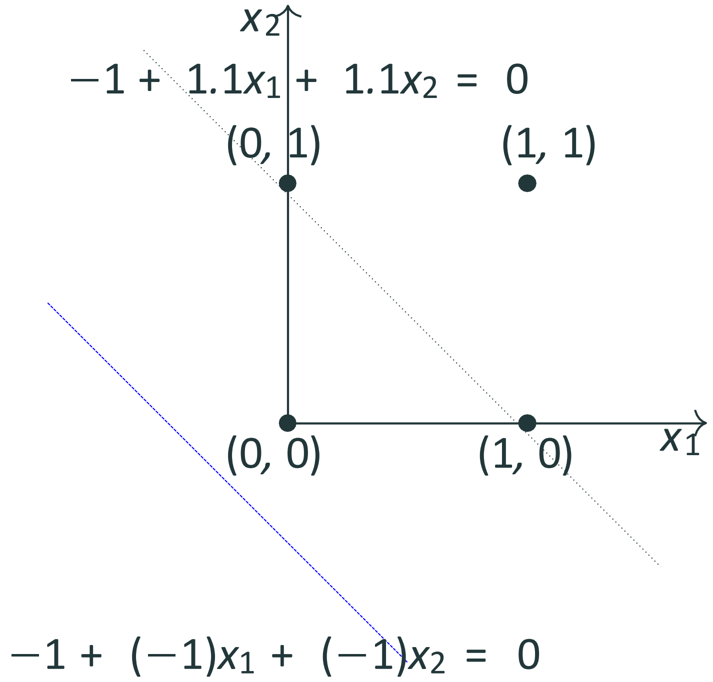
Errors and Error Surfaces

Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2

Say, $w_1 = -1, w_2 = -1$

What is wrong with this line? We make an error on 1 out of the 4 inputs

Lets try some more values of w_1, w_2 and note how many errors we make



Errors and Error Surfaces

Let us fix the threshold ($-w_0 = 1$) and try different values of w_1, w_2

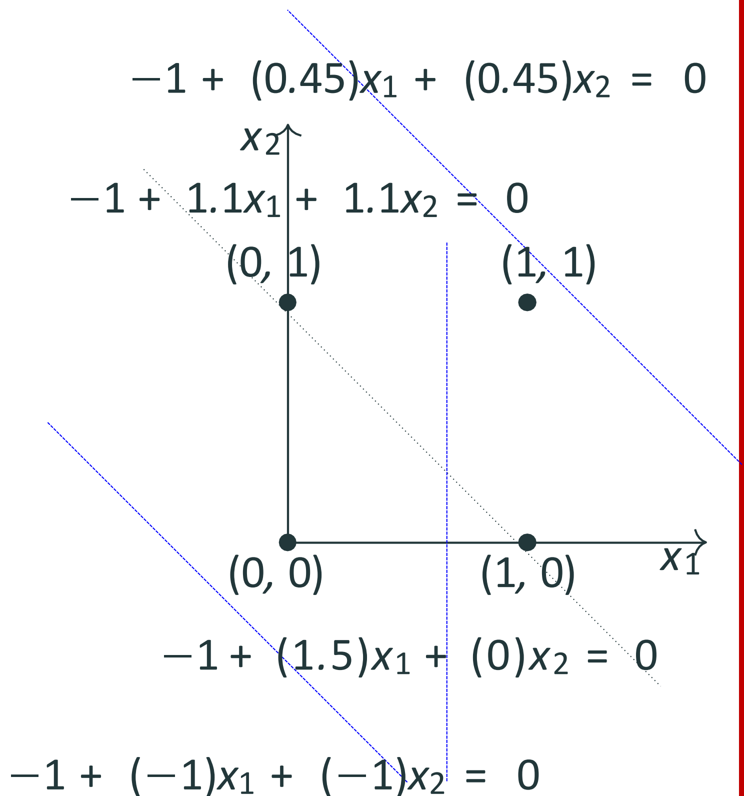
Say, $w_1 = -1, w_2 = -1$

What is wrong with this line? We make an error on 1 out of the 4 inputs

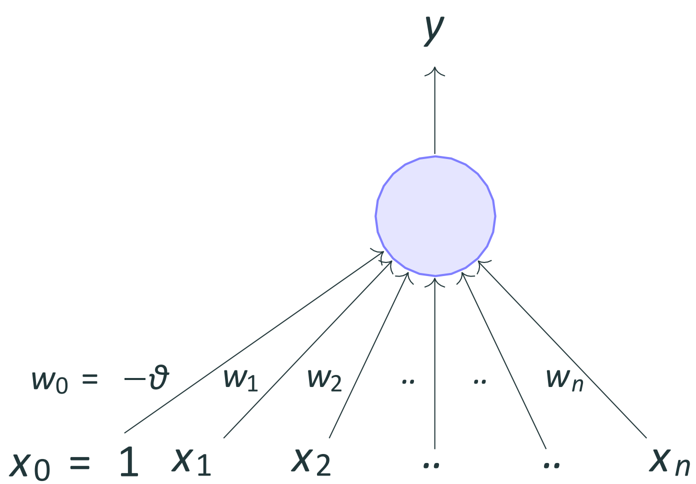
Lets try some more values of w_1, w_2 and note how many errors we make

w_1	w_2	errors
-1	-1	3
1.5	0	1
0.45	0.45	3

We are interested in those values of w_0, w_1, w_2 which result in 0 error



Learning weights: movie review



$x_1 = isActorDamon$

$x_2 = isGenreThriller$

$x_3 = isDirectorNolan$

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

... ...

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

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 label (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 values of $w_0, w_1, w_2, .., w_m$)

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

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

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

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

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**

end

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

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

end

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

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

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

  end

end

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

---

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

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

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

  end

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

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

  end

end

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

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

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

  end

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

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

  end

end

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

Why would this work ?

To understand why this works we will have to get into a bit of Linear Algebra and a bit of geometry...

Consider two vectors $\mathbf{w}$ and $\mathbf{x}$

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

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

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

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

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

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

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

We can thus rewrite the perceptron rule as

$$\begin{aligned} y &= 1 \quad if \quad \mathbf{w}^\mathbf{T} \mathbf{x} \geq 0 \\ &= 0 \quad if \quad \mathbf{w}^\mathbf{T} \mathbf{x} < 0 \end{aligned}$$

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

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

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

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

We can thus rewrite the perceptron rule as

$$\begin{aligned} y &= 1 \quad if \quad \mathbf{w}^{\mathbf{T}} \mathbf{x} \geq 0 \\ &= 0 \quad if \quad \mathbf{w}^{\mathbf{T}} \mathbf{x} < 0 \end{aligned}$$

We are interested in finding the line $\mathbf{w}^{\mathbf{T}} \mathbf{x} = 0$ which divides the input space into two halves

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

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

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

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

We can thus rewrite the perceptron rule as

$$\begin{aligned} y &= 1 \quad if \quad \mathbf{w}^{\mathbf{T}} \mathbf{x} \geq 0 \\ &= 0 \quad if \quad \mathbf{w}^{\mathbf{T}} \mathbf{x} < 0 \end{aligned}$$

We are interested in finding the line $\mathbf{w}^{\mathbf{T}} \mathbf{x} = 0$ which divides the input space into two halves

Every point ($\mathbf{x}$) on this line satisfies the equation $\mathbf{w}^{\mathbf{T}} \mathbf{x} = 0$

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

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

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

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

We can thus rewrite the perceptron rule as

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

We are interested in finding the line $\mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$ which divides the input space into two halves

Every point (**x**) on this line satisfies the equation $\mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$

What can you tell about the angle (α) between **w** and any point (**x**) which lies on this line ?

Consider two vectors $\mathbf{w}$ and $\mathbf{x}$

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

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

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

We can thus rewrite the perceptron rule as

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

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

We are interested in finding the line $\mathbf{w}^{\mathbf{T}} \mathbf{x} = 0$ which divides the input space into two halves

Every point ($\mathbf{x}$) on this line satisfies the equation $\mathbf{w}^{\mathbf{T}} \mathbf{x} = 0$

What can you tell about the angle (α) between $\mathbf{w}$ and any point ($\mathbf{x}$) which lies on this line ?

The angle is 90° ($\because cos\alpha = \frac{w^T x}{||w||||x||} = 0$)

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

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

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

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

We can thus rewrite the perceptron rule as

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

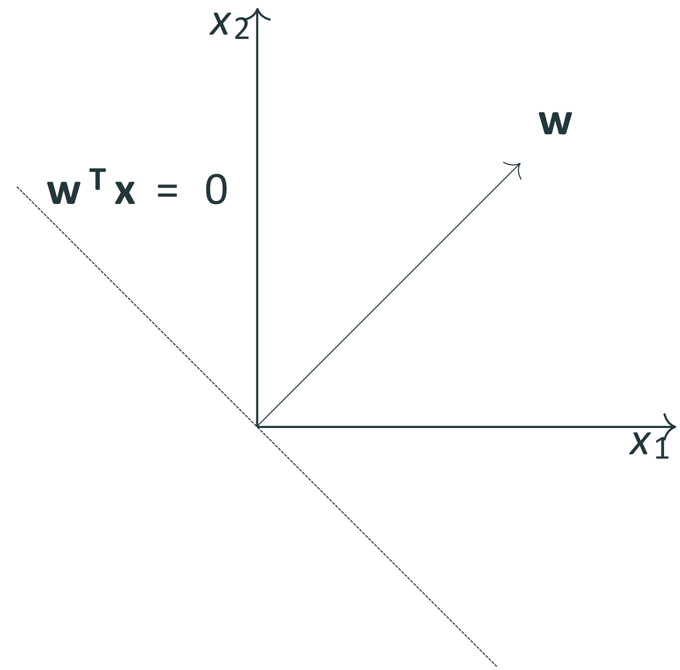
We are interested in finding the line $\mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$ which divides the input space into two halves

Every point ($\mathbf{x}$) on this line satisfies the equation $\mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$

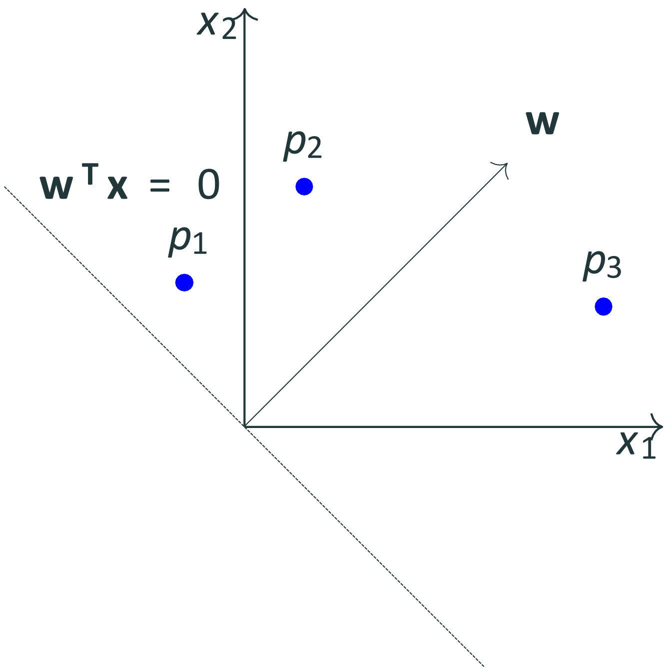
What can you tell about the angle (α) between **w** and any point (**x**) which lies on this line ?

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 perpendicular to the line itself

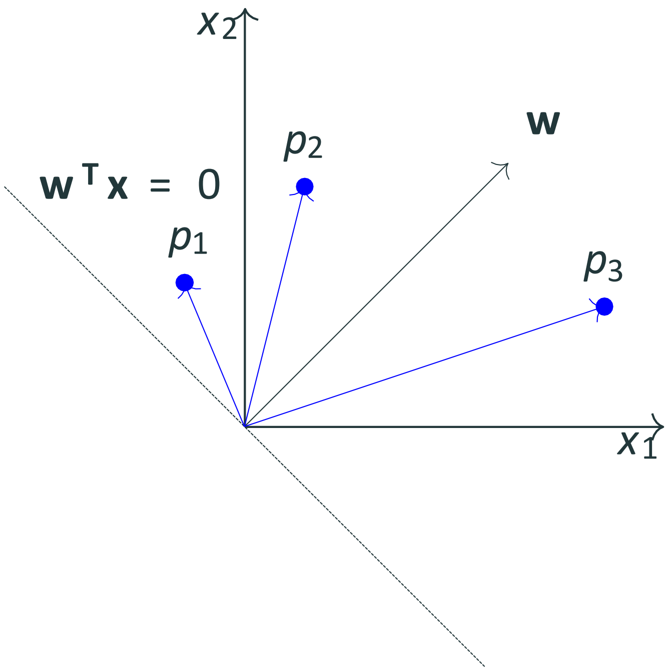

x_2
$\mathbf{w}$
$\mathbf{w}^\mathsf{T}\mathbf{x} = 0$
x_1

Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^\mathsf{T}\mathbf{x} \geq 0$)



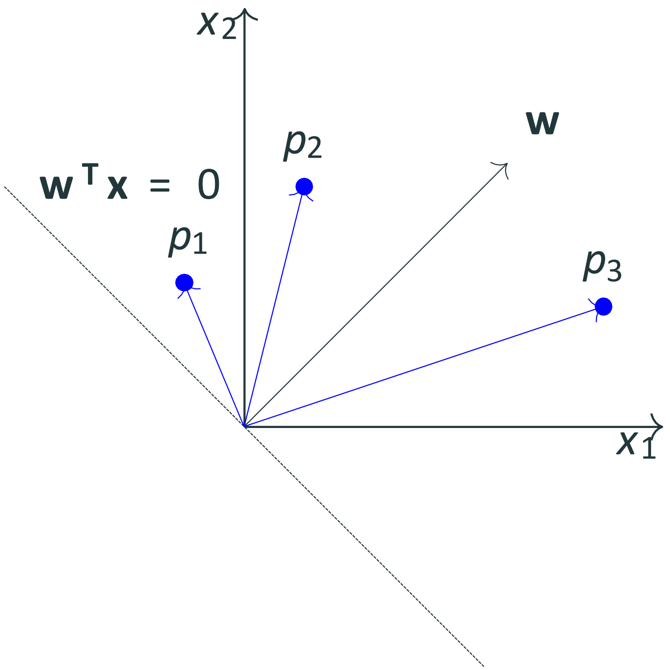
Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^{\mathsf{T}}\mathbf{x} \geq 0$)

What will be the angle between any such vector and $\mathbf{w}$?



Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^T\mathbf{x} \geq 0$)

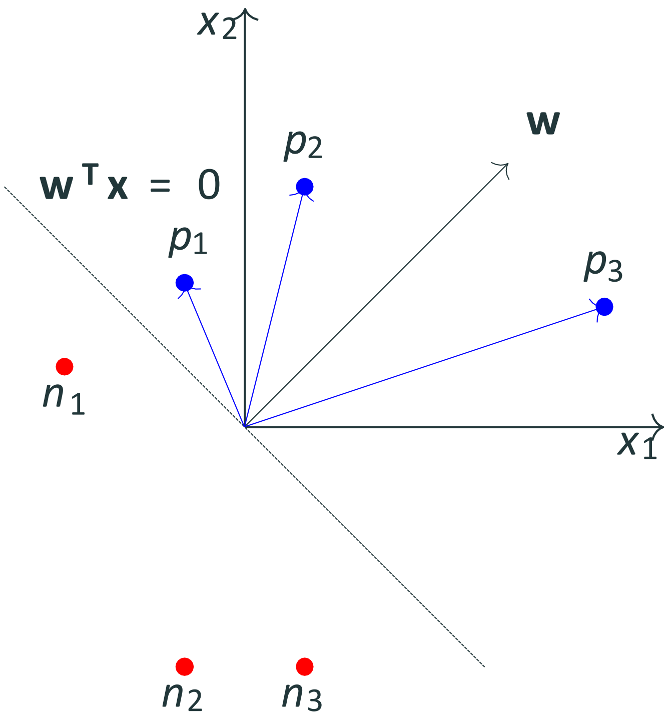
What will be the angle between any such vector and $\mathbf{w}$? Obviously, less than 90°



Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^T\mathbf{x} \geq 0$)

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.*, $\mathbf{w}^T\mathbf{x} < 0$)

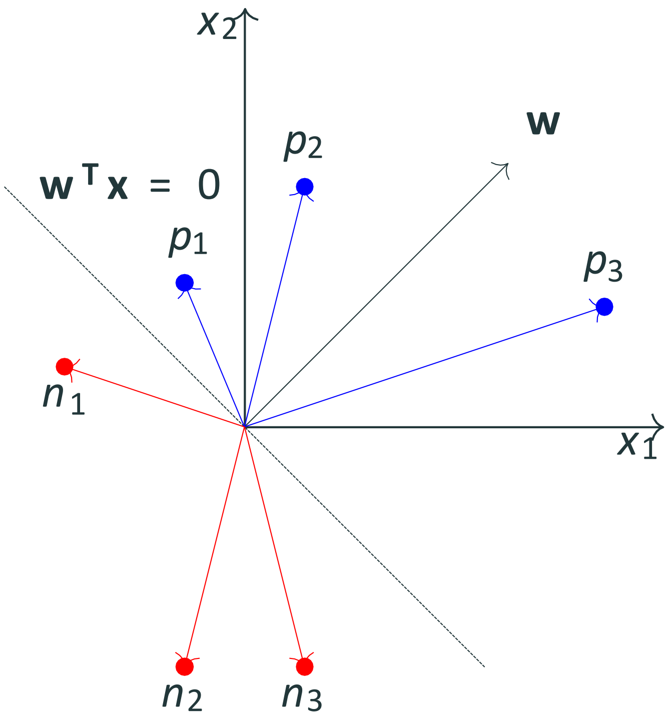


Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^T\mathbf{x} \geq 0$)

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.*, $\mathbf{w}^T\mathbf{x} < 0$)

What will be the angle between any such vector and $\mathbf{w}$?

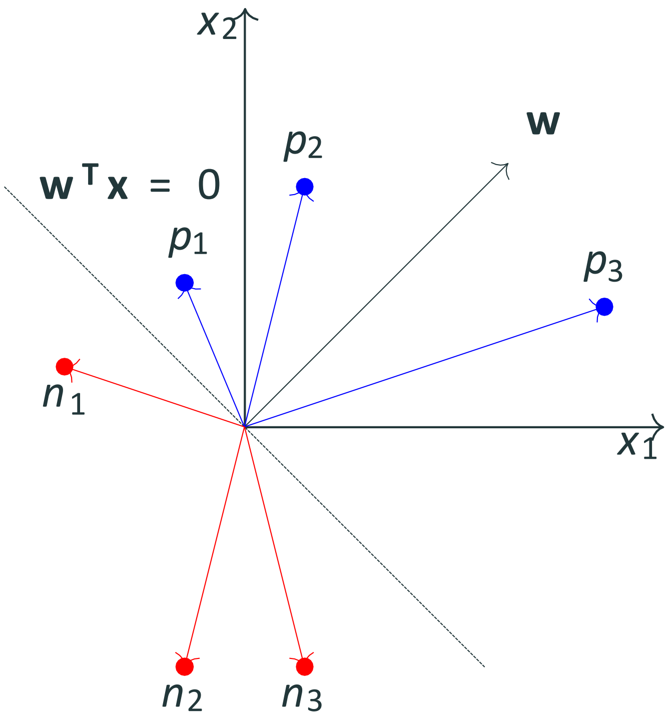


Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^T\mathbf{x} \geq 0$)

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.*, $\mathbf{w}^T\mathbf{x} < 0$)

What will be the angle between any such vector and $\mathbf{w}$? Obviously, greater than 90°



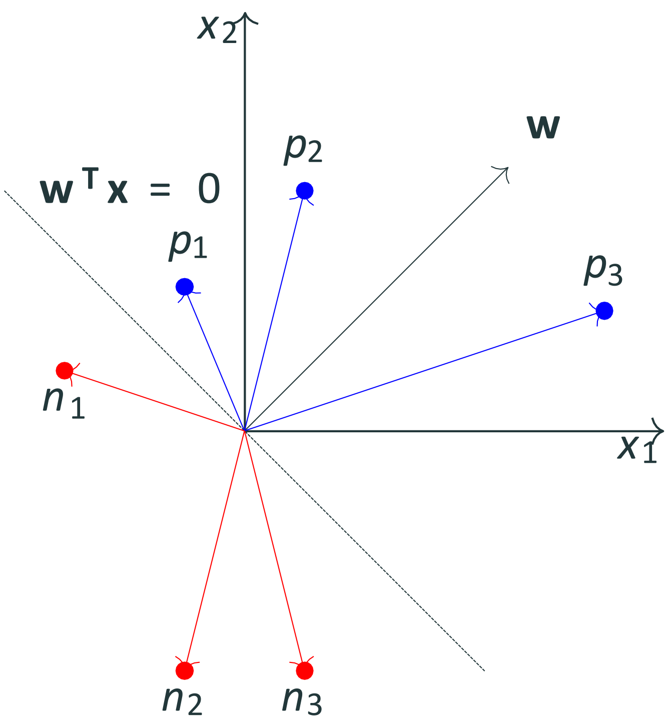
Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^{\mathsf{T}}\mathbf{x} \geq 0$)

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.*, $\mathbf{w}^{\mathsf{T}}\mathbf{x} < 0$)

What will be the angle between any such vector and $\mathbf{w}$? Obviously, greater than 90°

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



Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w}^{\mathsf{T}}\mathbf{x} \geq 0$)

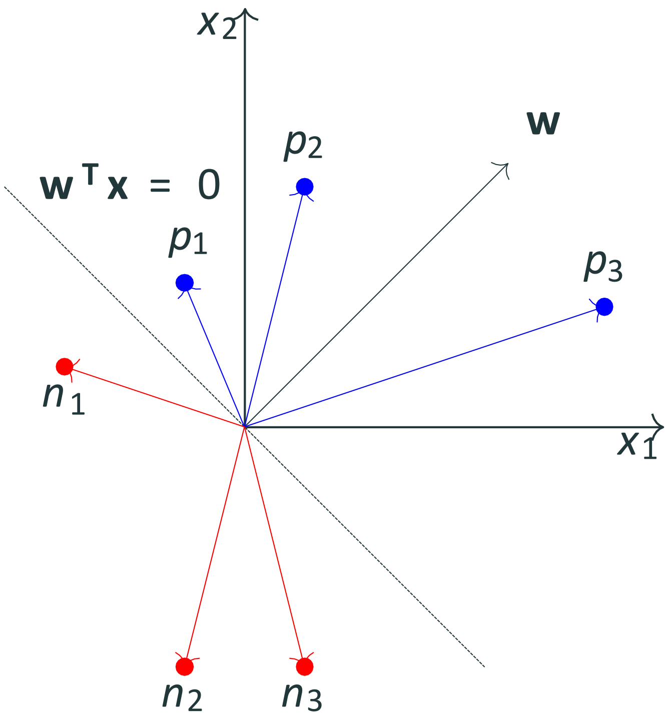
What will be the angle between any such vector and $\mathbf{w}$? Obviously, less than 90°

What about points (vectors) which lie in the negative half space of this line (*i.e.*, $\mathbf{w}^{\mathsf{T}}\mathbf{x} < 0$)

What will be the angle between any such vector and $\mathbf{w}$? Obviously, greater than 90°

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

Keeping this picture in mind let us revisit the algorithm



Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

end

if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

end

end

//the algorithm converges when all the inputs

are classified correctly

$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{}$

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{}$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is greater than 90°

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while !$convergence$ **do**

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

    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

    end

    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

    end

end

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

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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°)

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 $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$$cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$$cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

$$cos(\alpha_{new}) \propto \mathbf{w_{new}}^T \mathbf{x}$$

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P$ and $\mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N$ and $\mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

---

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

$$cos(\alpha_{new}) \propto \mathbf{w_{new}}^T\mathbf{x}$$

$$\propto (\mathbf{w} + \mathbf{x})^T\mathbf{x}$$

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P$ and $\mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N$ and $\mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

//the algorithm converges when all the inputs

are classified correctly

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

$$cos(\alpha_{new}) \propto \mathbf{w_{new}}^T\mathbf{x}$$
$$\propto (\mathbf{w} + \mathbf{x})^T\mathbf{x}$$
$$\propto \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{x}$$

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$$cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

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

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$$cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

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

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

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$$cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{}$$

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ 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 $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

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

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

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

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

For $\mathbf{x} \in N$ if $\mathbf{w}.\mathbf{x} \geq 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is less than 90°

$$cos\alpha = \frac{\mathbf{w}^T \mathbf{x}}{}$$

Algorithm: Perceptron Learning Algorithm

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

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

Initialize $\mathbf{w}$ randomly;

while $!convergence$ **do**

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

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

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

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

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

  end

end

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

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{}$$

For $\mathbf{x} \in N$ if $\mathbf{w}.\mathbf{x} \geq 0$ 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°)

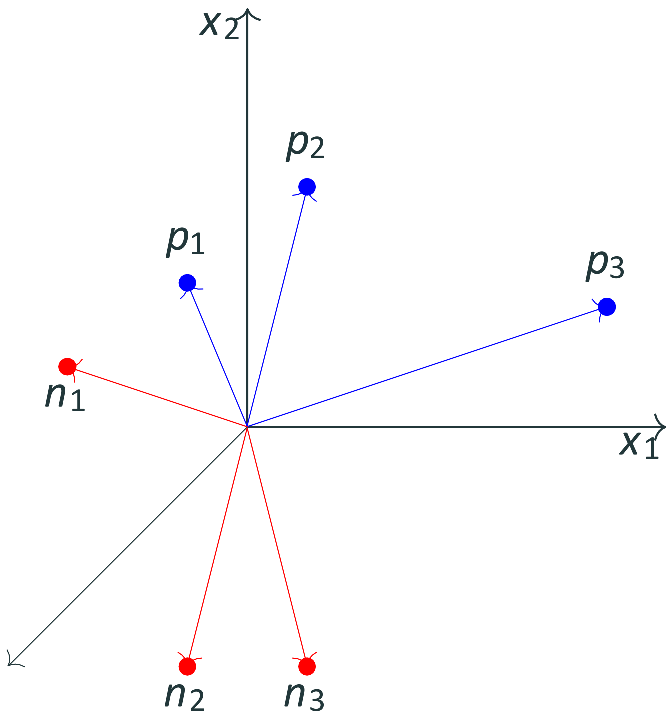
What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

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

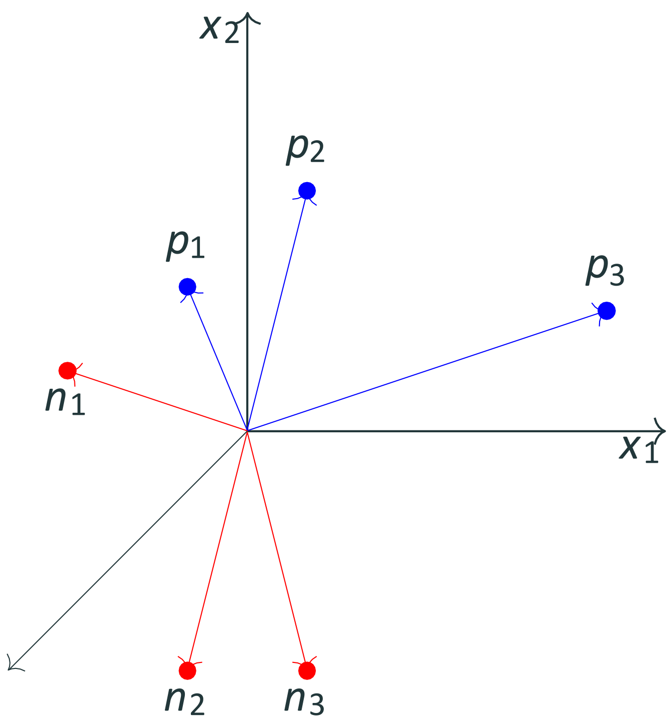
$$cos(\alpha_{new}) < cos\alpha$$

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

We will now see this algorithm in action for a toy dataset

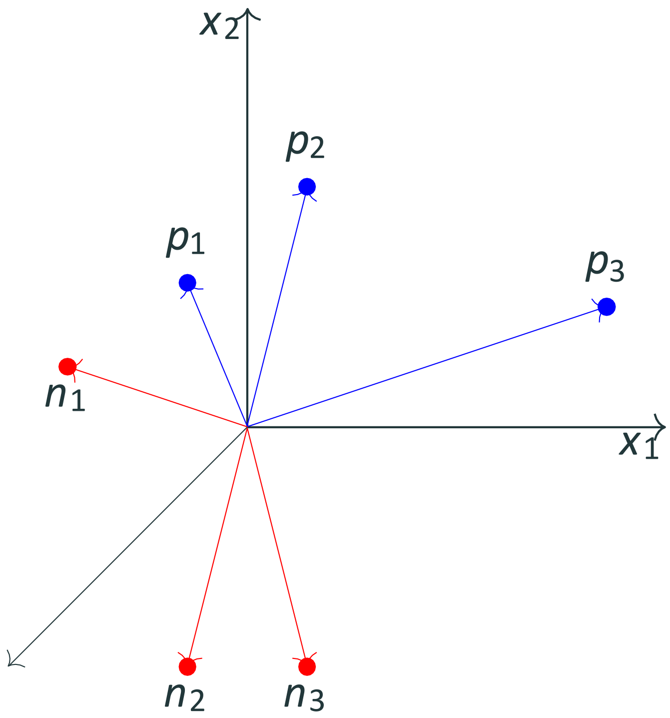


We initialized **w** to a random value



We initialized $\mathbf{w}$ to a random value

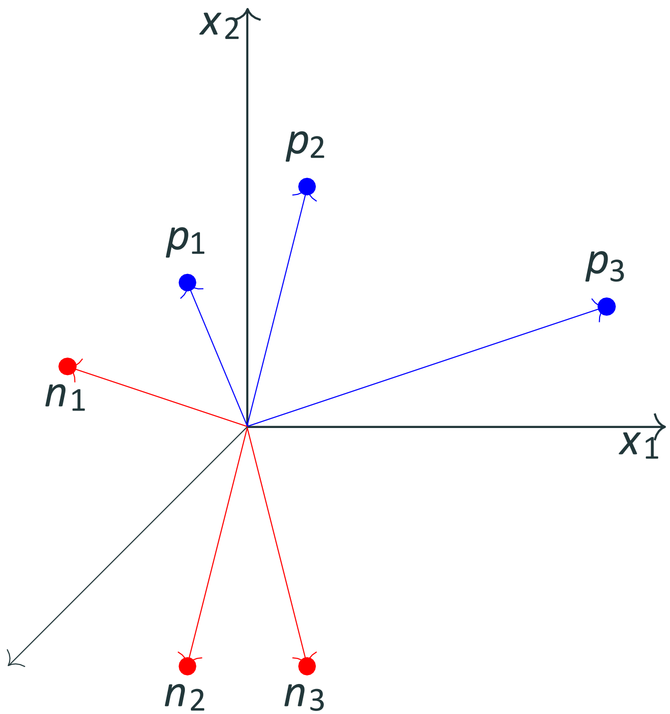
We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\because$ angle > 90°) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle < 90°) for all the negative points (the situation is exactly oppsite of what we actually want it to be)



We 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

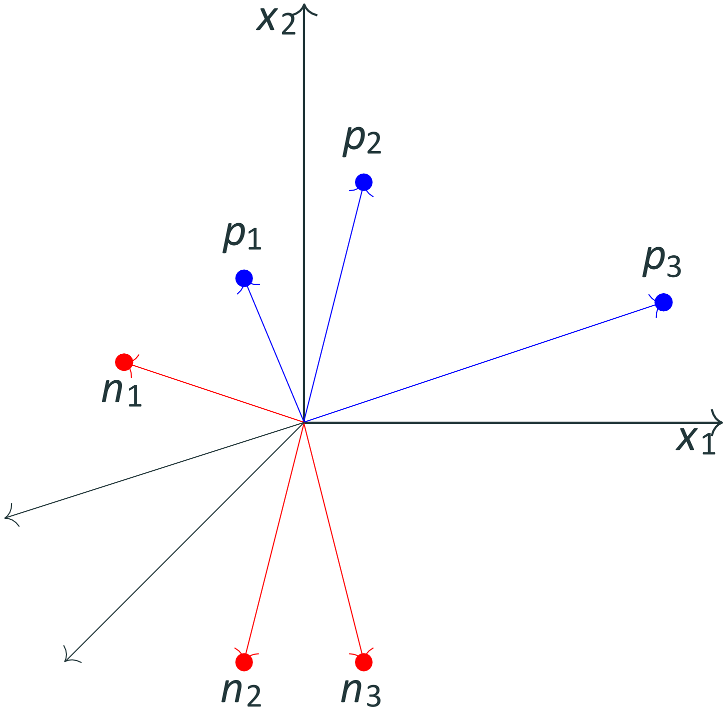


We initialized $\mathbf{w}$ to a random value

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_1), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

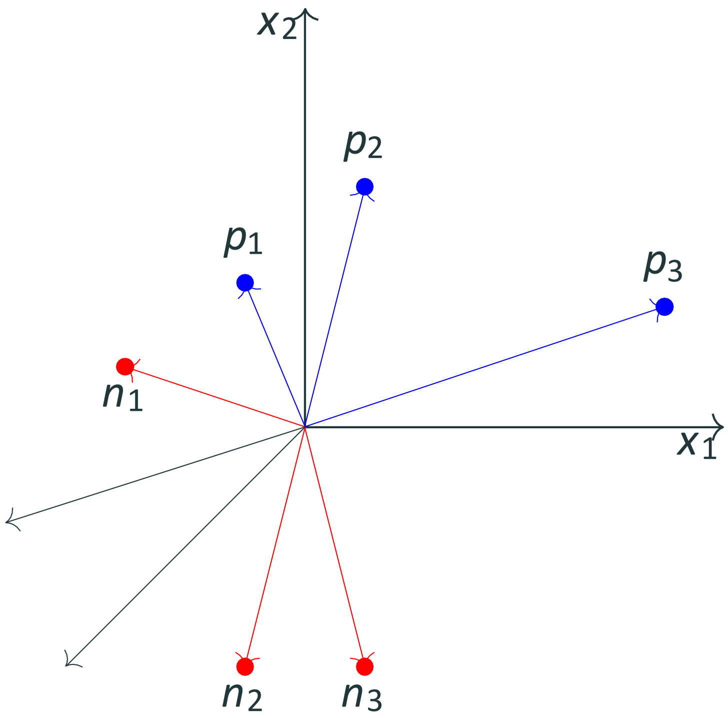


We initialized $\mathbf{w}$ to a random value

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, p_1), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

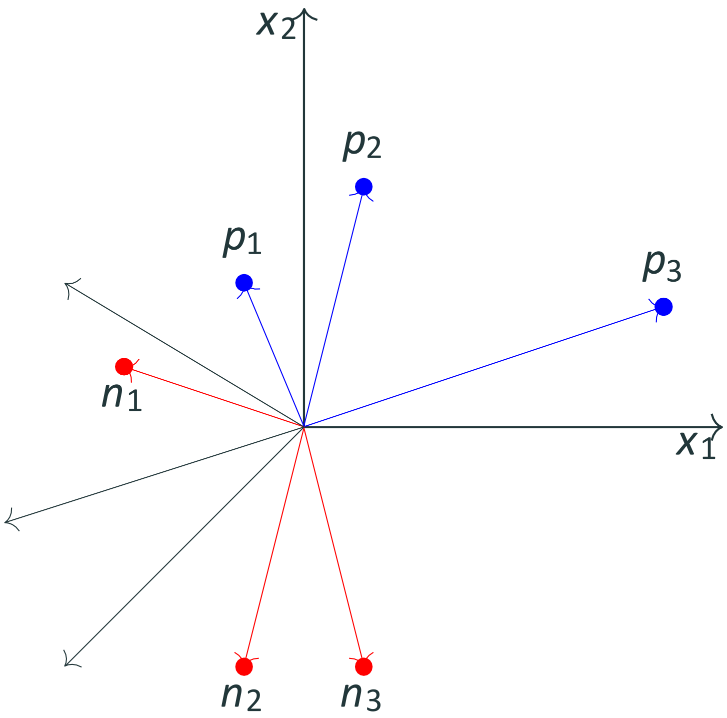


We initialized $\mathbf{w}$ to a random value

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, p_2), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

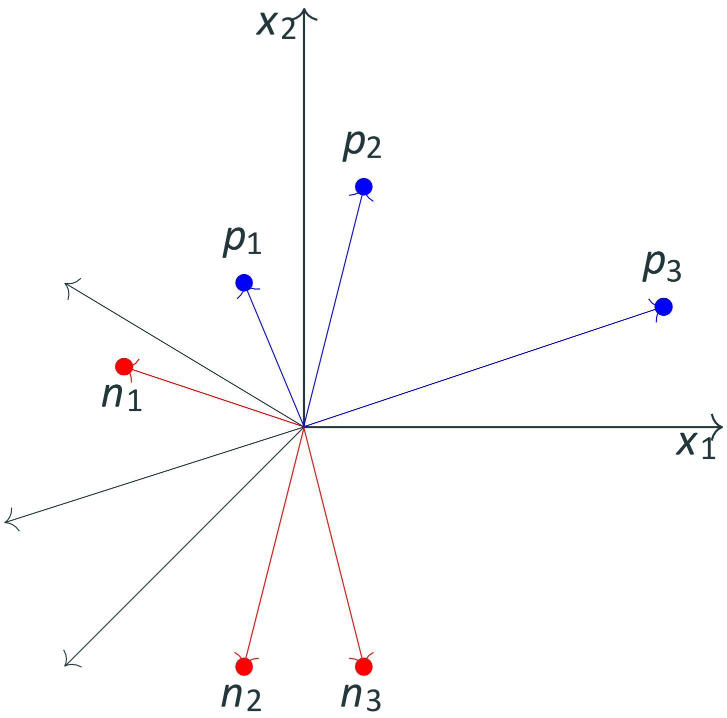


We initialized $\mathbf{w}$ to a random value

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, p_2), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

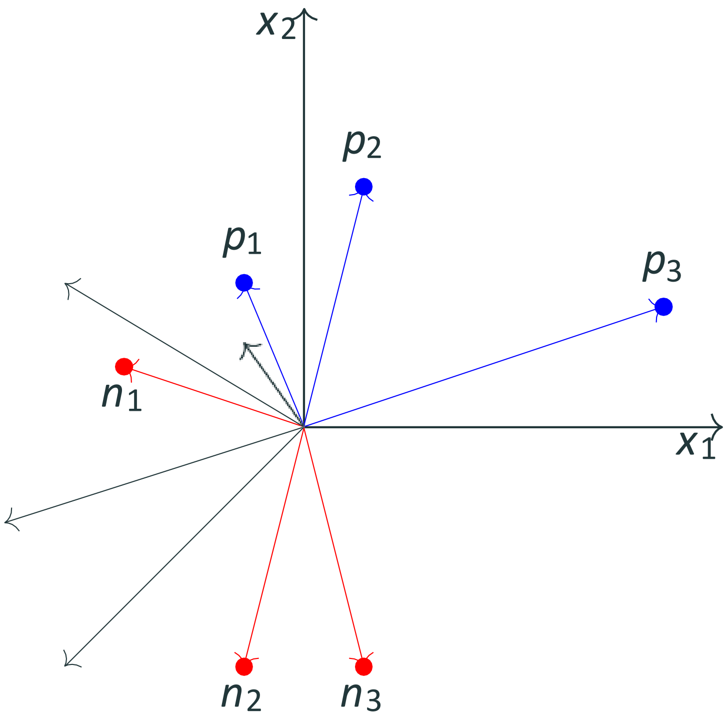


We initialized $\mathbf{w}$ to a random value

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), apply correction $\mathbf{w} = \mathbf{w} - \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

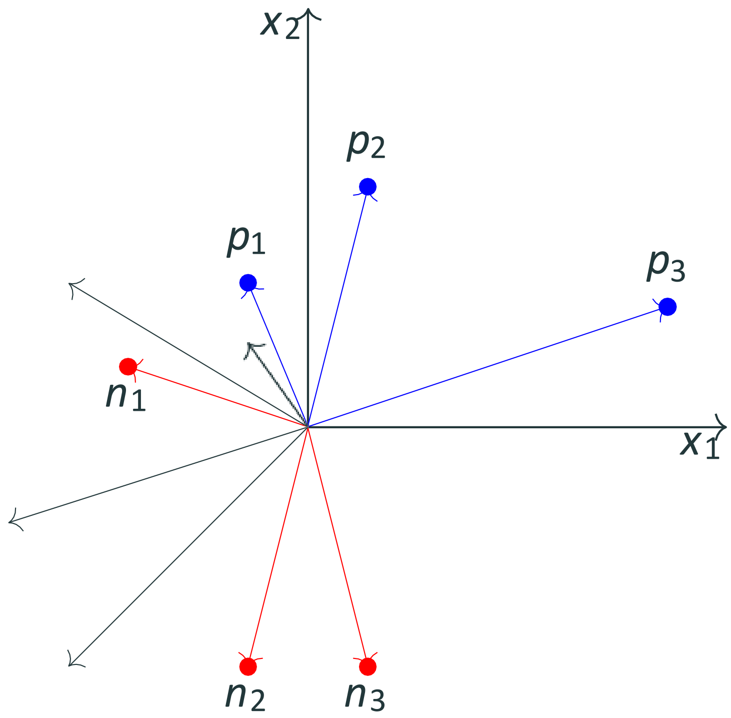


We initialized $\mathbf{w}$ to a random value

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), apply correction $\mathbf{w} = \mathbf{w} - \mathbf{x}$ $\because \mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

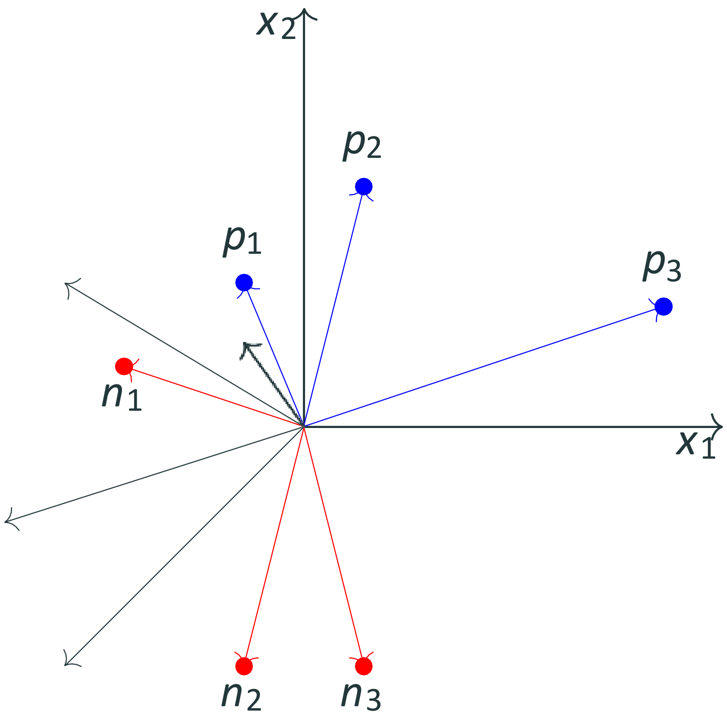


We initialized $\mathbf{w}$ to a random value

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_3), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

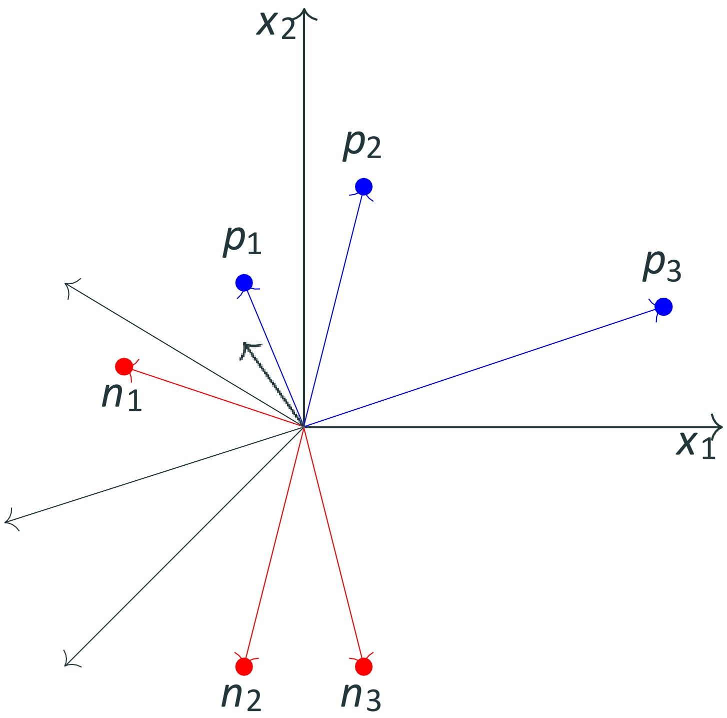


We initialized $\mathbf{w}$ to a random value

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_3), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

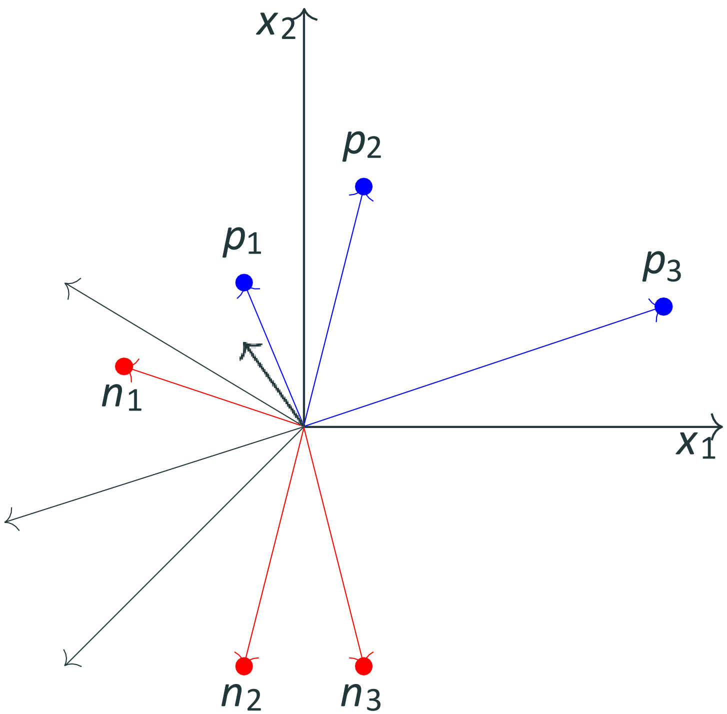


We initialized $\mathbf{w}$ to a random value

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_2), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

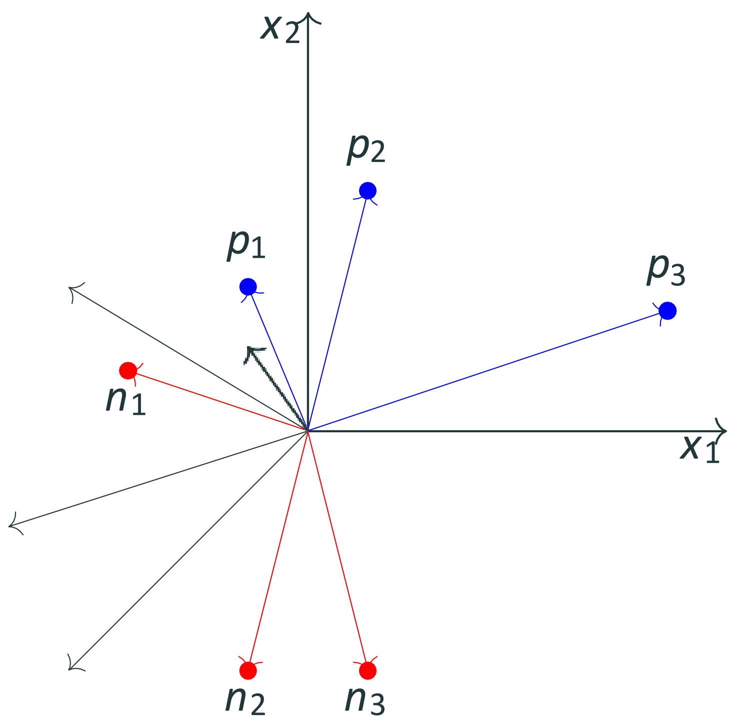


We initialized $\mathbf{w}$ to a random value

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_2), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

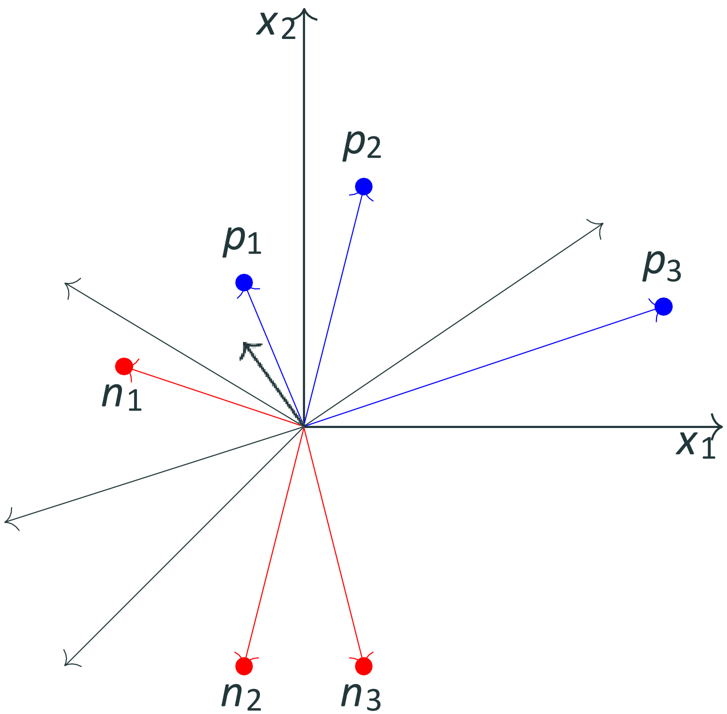


We initialized $\mathbf{w}$ to a random value

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, p_3), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

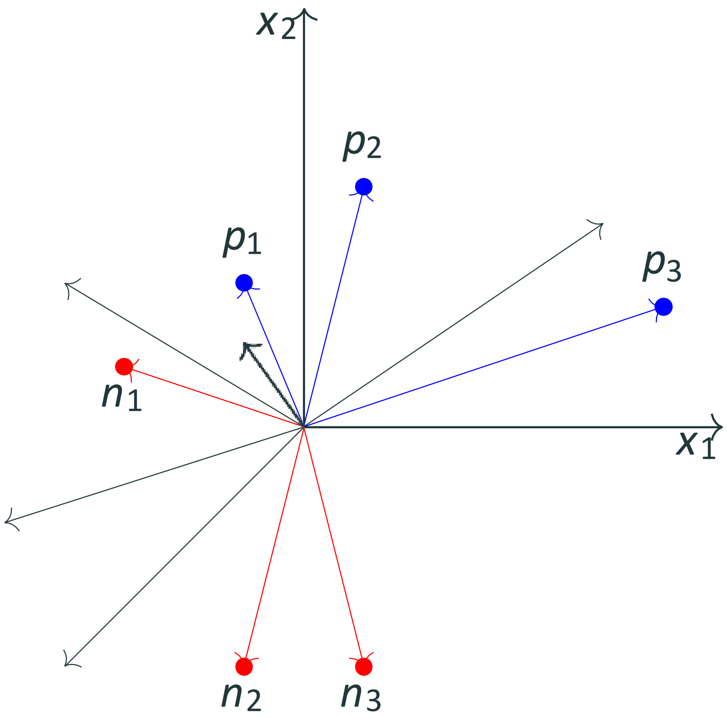


We initialized $\mathbf{w}$ to a random value

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, p_3), apply correction $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

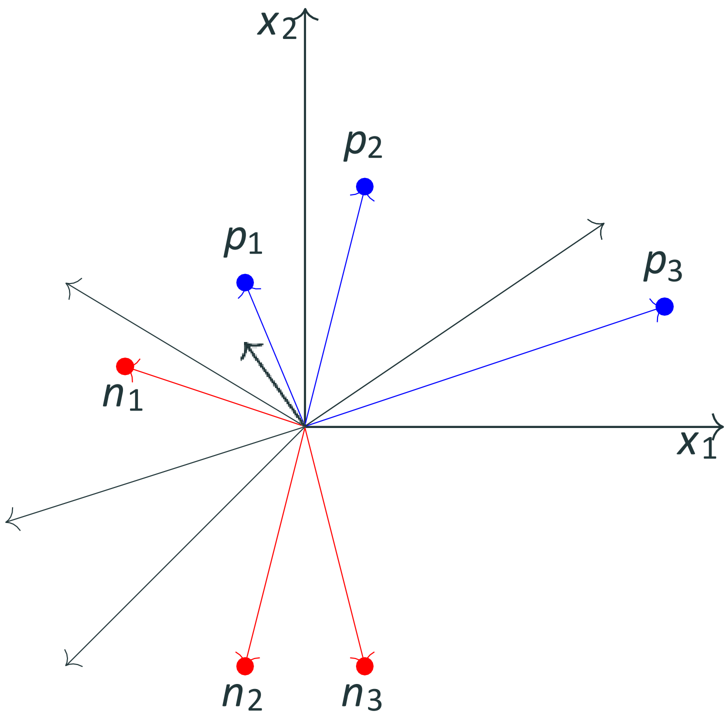


We initialized $\mathbf{w}$ to a random value

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, p_1), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

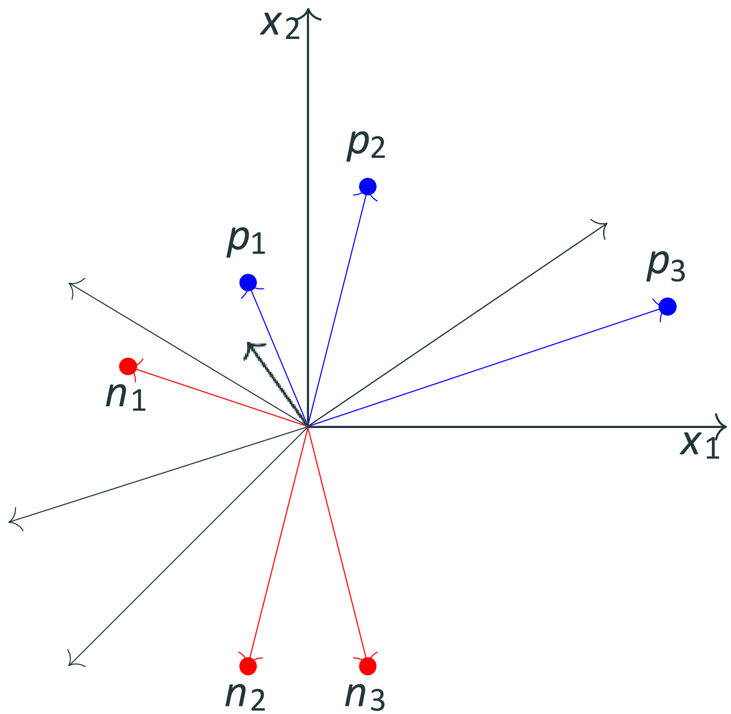


We initialized $\mathbf{w}$ to a random value

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, p_1), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

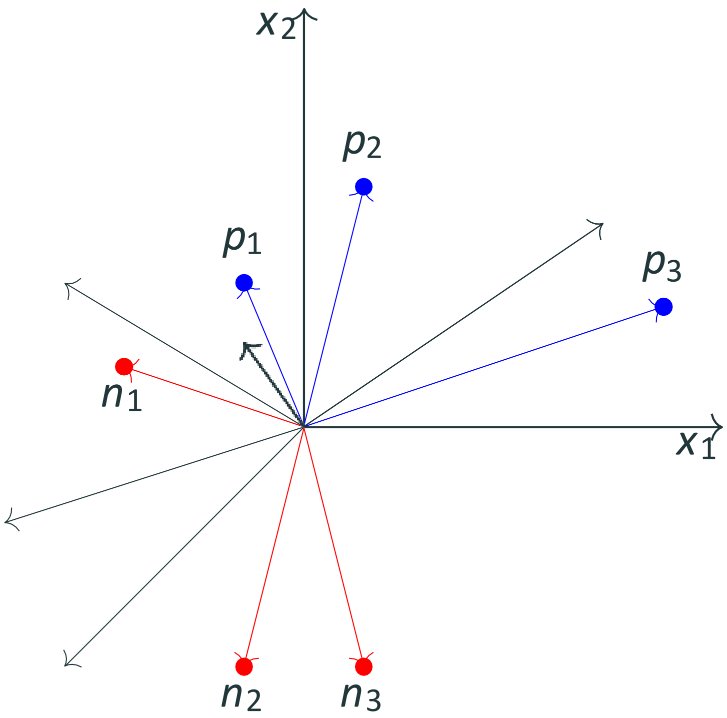


We initialized $\mathbf{w}$ to a random value

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, p_2), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

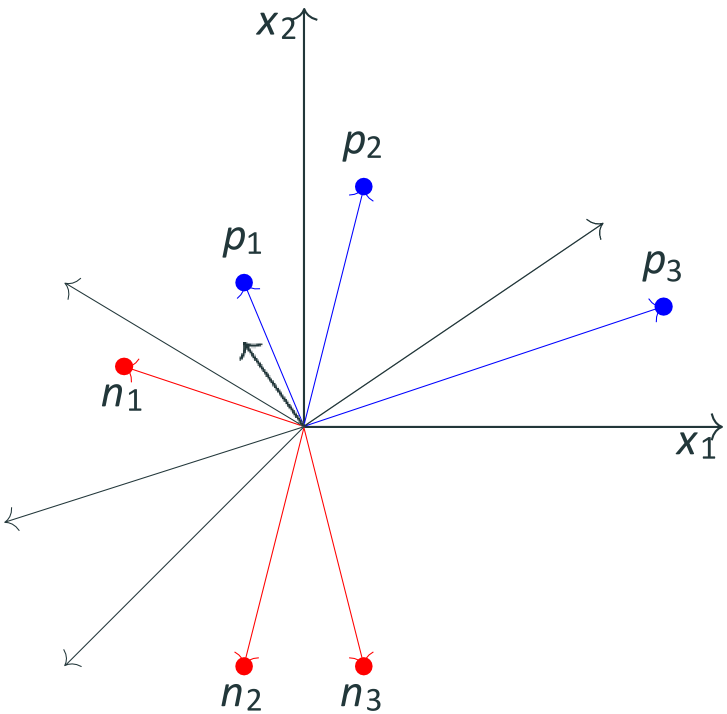


We initialized $\mathbf{w}$ to a random value

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, p_2), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (you can check the angle visually)

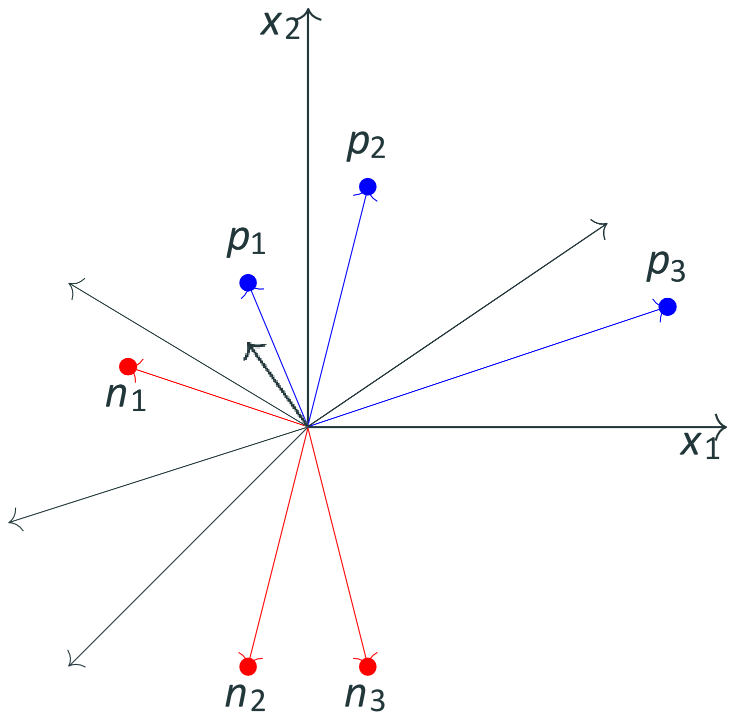


We initialized $\mathbf{w}$ to a random value

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)

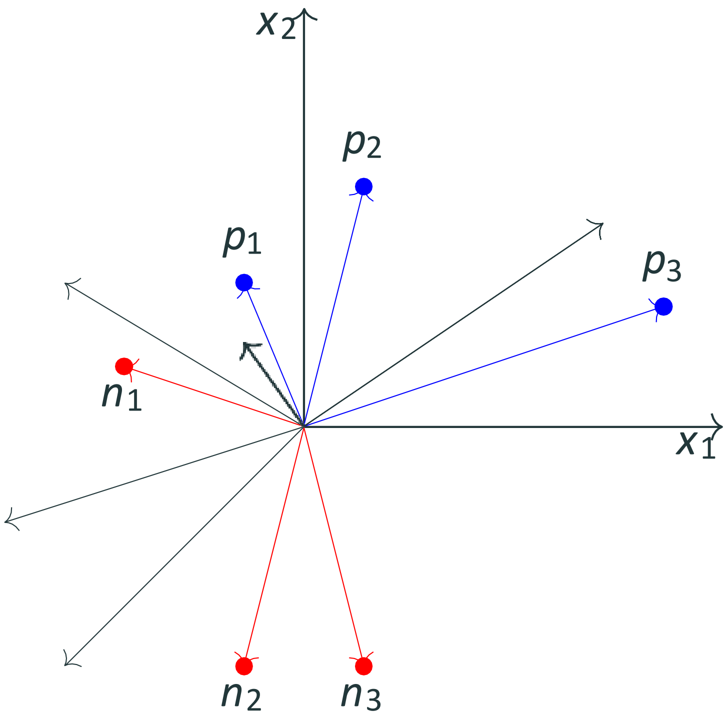


We initialized $\mathbf{w}$ to a random value

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)

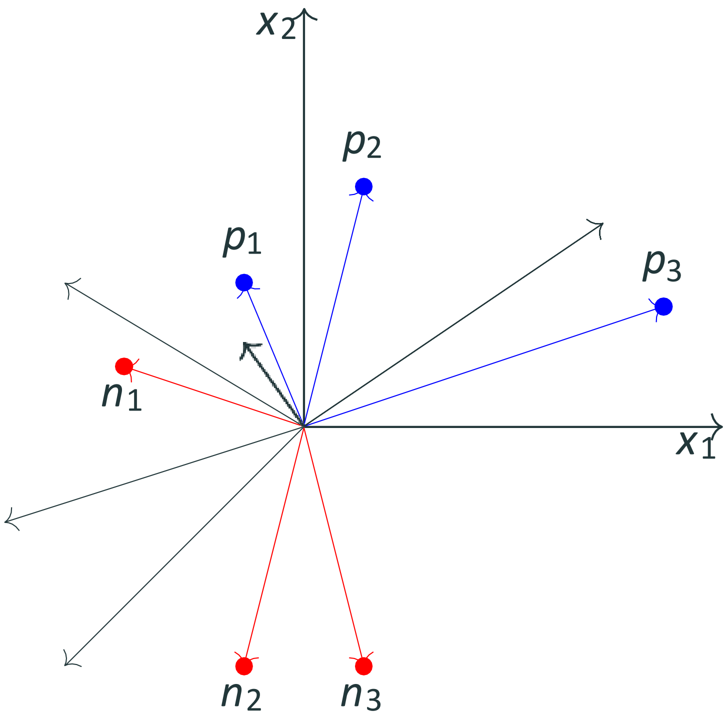


We initialized $\mathbf{w}$ to a random value

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_3), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

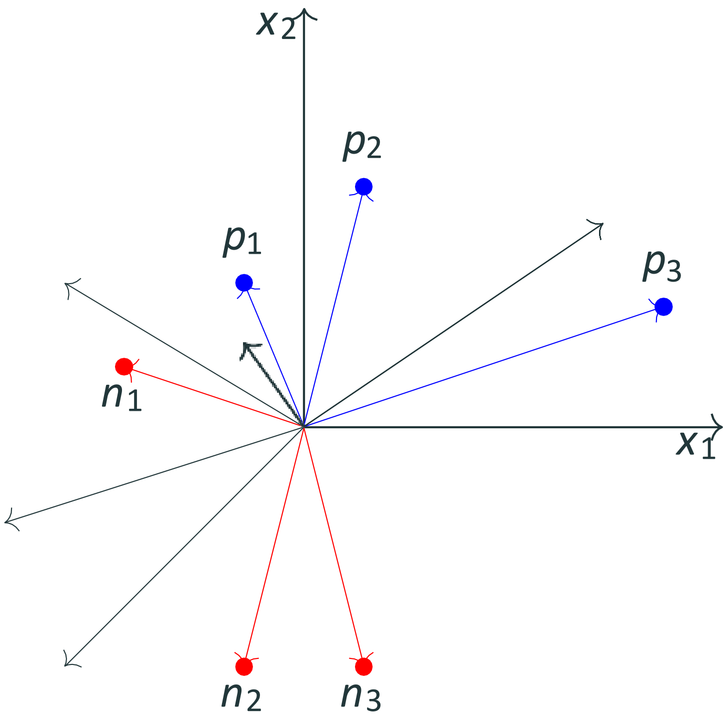


We initialized $\mathbf{w}$ to a random value

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_3), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

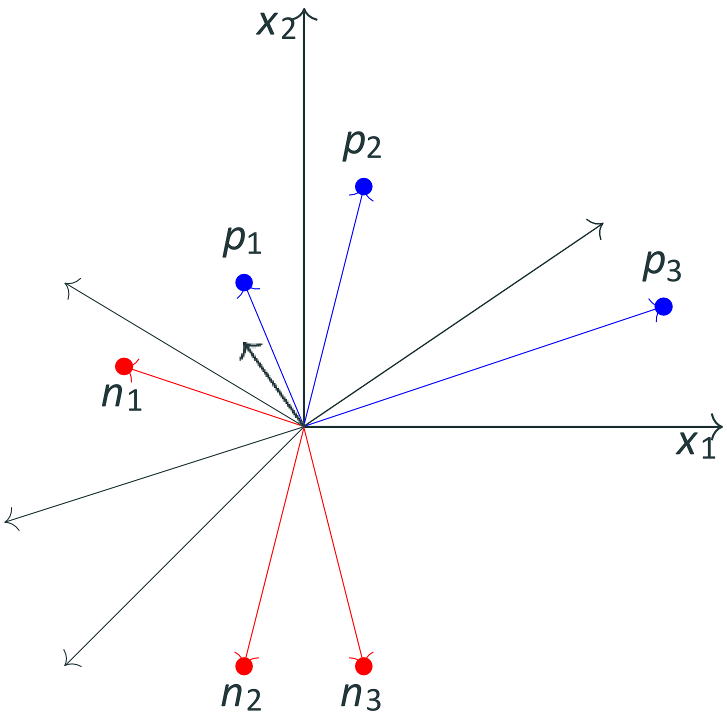


We initialized $\mathbf{w}$ to a random value

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_2), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

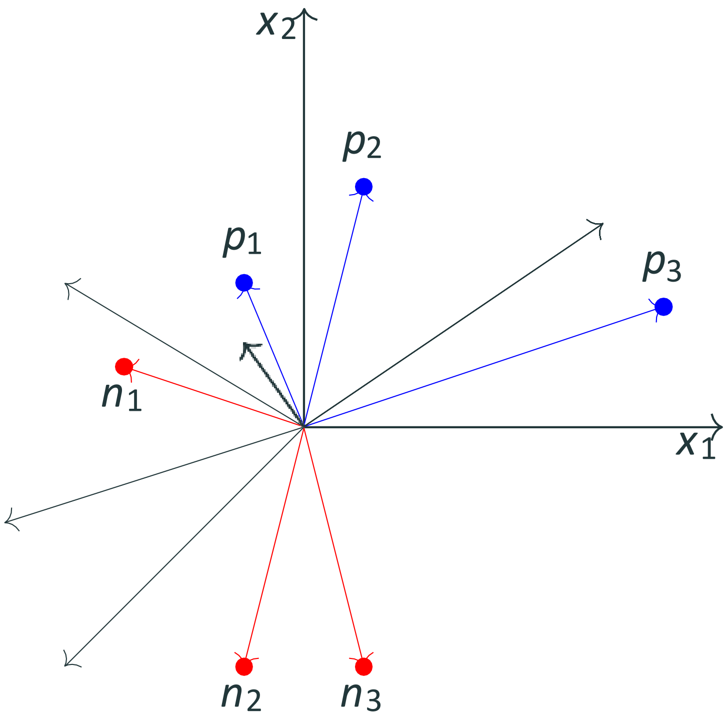


We initialized $\mathbf{w}$ to a random value

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_2), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} < \mathbf{0}$ (you can check the angle visually)

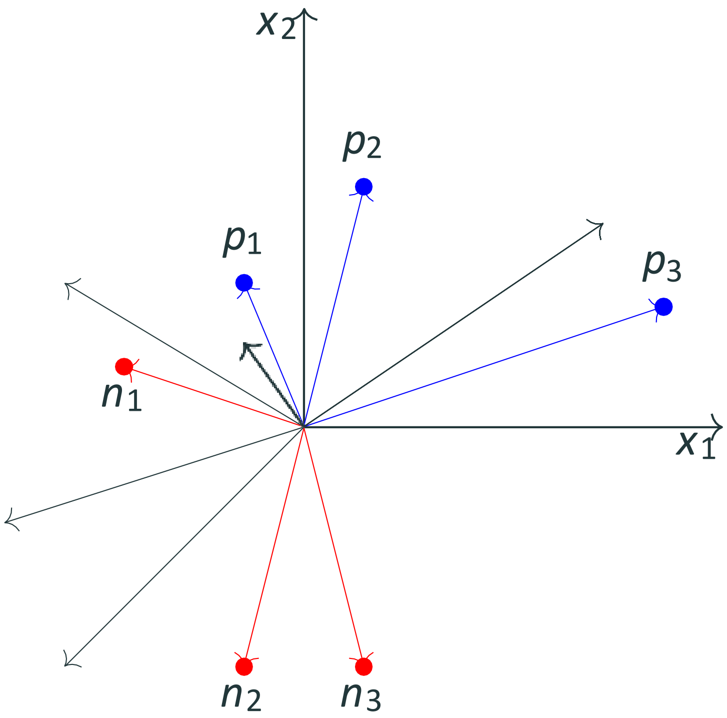


We initialized $\mathbf{w}$ to a random value

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, p_3), no correction needed $\because$ $\mathbf{w} \cdot \mathbf{x} \geq \mathbf{0}$ (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

The algorithm has converged

Coming back to our questions …

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 ?

Coming back to our questions ...

What about non-boolean (say, real) inputs? Real valued inputs are allowed in perceptron

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 ?

Coming back to our questions . . .

What about non-boolean (say, real) inputs? Real valued inputs are allowed in perceptron

Do we always need to hand code the threshold? No, we can learn 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 ?

Coming back to our questions ...

What about non-boolean (say, real) inputs? Real valued inputs are allowed in per- ceptron

Do we always need to hand code the threshold? No, we can learn the threshold

Are all inputs equal? What if we want to assign more weight (importance) to some inputs? Aperceptron allows weights to be assigned to inputs

What about functions which are not linearly separable ?

Coming back to our questions . . .

What about non-boolean (say, real) inputs? Real valued inputs are allowed in perceptron

Do we always need to hand code the threshold? No, we can learn the threshold

Are all inputs equal? What if we want to assign more weight (importance) to some inputs? A perceptron allows weights to be assigned to inputs

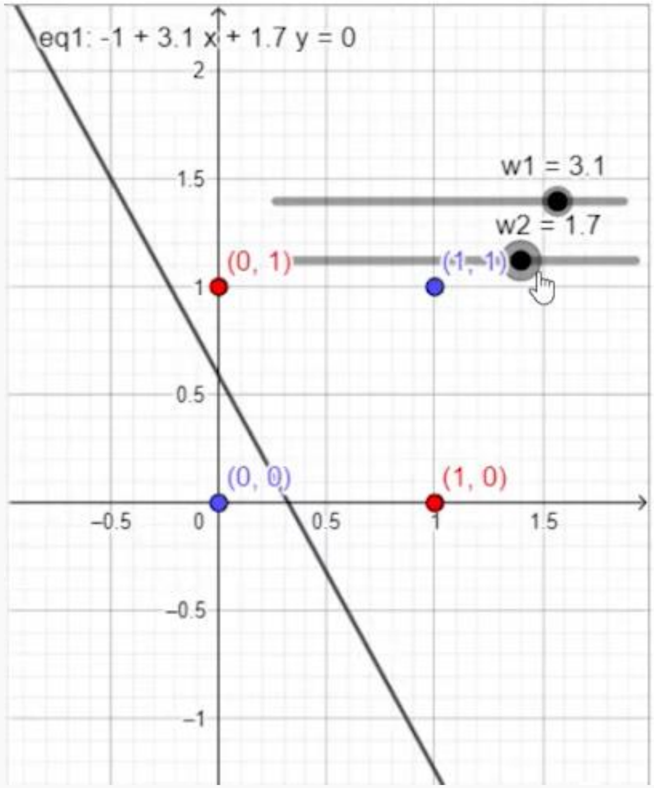
What about functions which are not linearly separable ? Not possible with a single perceptron but we will see how to handle this ..

- **Non-Linearly Separable Boolean Functions**

So what do we do about functions which are not linearly separable ?

Let us see one such simple boolean function first ?

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



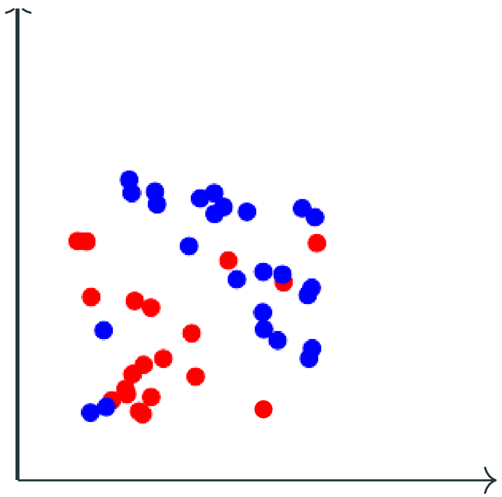
$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

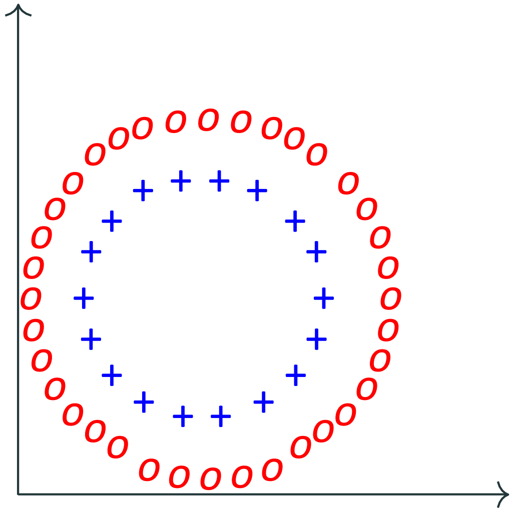
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$$

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

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

- The fourth condition contradicts conditions 2 and 3
- Hence we cannot have a solution to this set of in-equalities





Most real world data is not linearly separable and will always contain some outliers

In fact, sometimes there may not be any outliers but still the data may not be linearly separable

While a single perceptron cannot deal with such data, we will show that a network of perceptrons can indeed deal with such data

Before seeing how a network of perceptrons can deal with linearly inseparable data, we will discuss boolean functions in some more detail ...

How many boolean functions can you design from 2 inputs ?

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

x_1	x_2	f_1
0	0	0
0	1	0
1	0	0
1	1	0

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

x_1	x_2	f_1	f_{16}
0	0	0	1
0	1	0	1
1	0	0	1
1	1	0	1

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

x_1	x_2	f_1	f_2	f_{16}
0	0	0	0	1
0	1	0	0	1
1	0	0	0	1
1	1	0	1	1

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

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

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

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 ?

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

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)

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

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)

In general, how many boolean functions can you have for n inputs ? 2^{2^n}

How many boolean functions can you design from 2 inputs ?

Let us begin with some easy ones which you already know ..

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)

In general, how many boolean functions can you have for n inputs ? 2^{2^n}

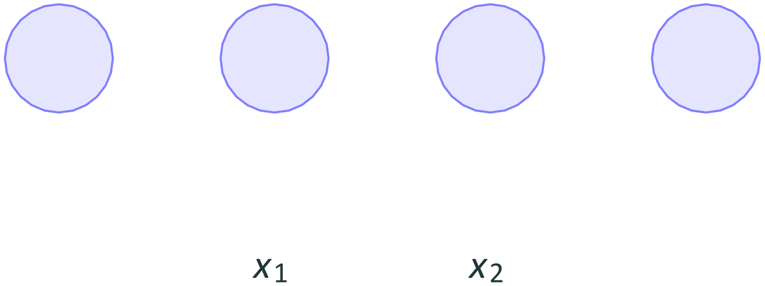
How many of these 2^{2^n} functions are not linearly separable ?

For the time being, it suffices to know that at least some of these may not be linearly inseparable

Module 2.8: Representation Power of a Network of Perceptrons

For this discussion, we will assume True = +1 and False = -1

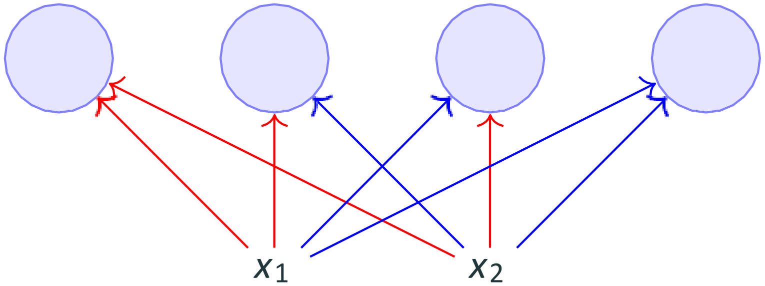
We consider 2 inputs and 4 perceptrons



For this discussion, we will assume True = +1 and False = -1

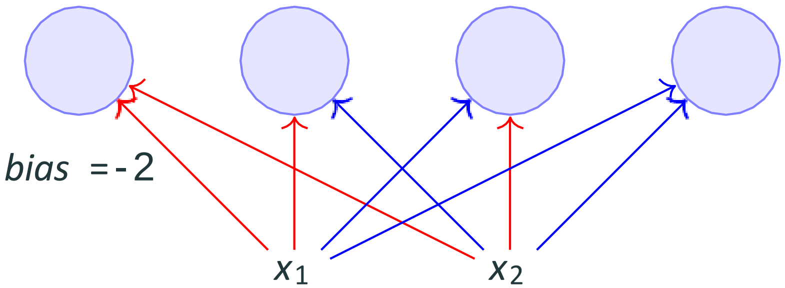
We consider 2 inputs and 4 perceptrons

Each input is connected to all the 4 perceptrons with specific weights



red edge indicates w = -1

blue edge indicates w = +1



red edge indicates $w = -1$

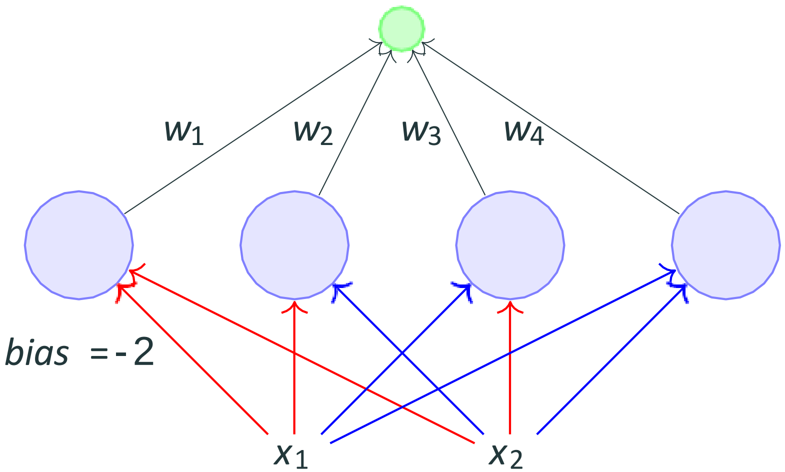
blue edge indicates $w = +1$

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)



red edge indicates $w = -1$

blue edge indicates $w = +1$

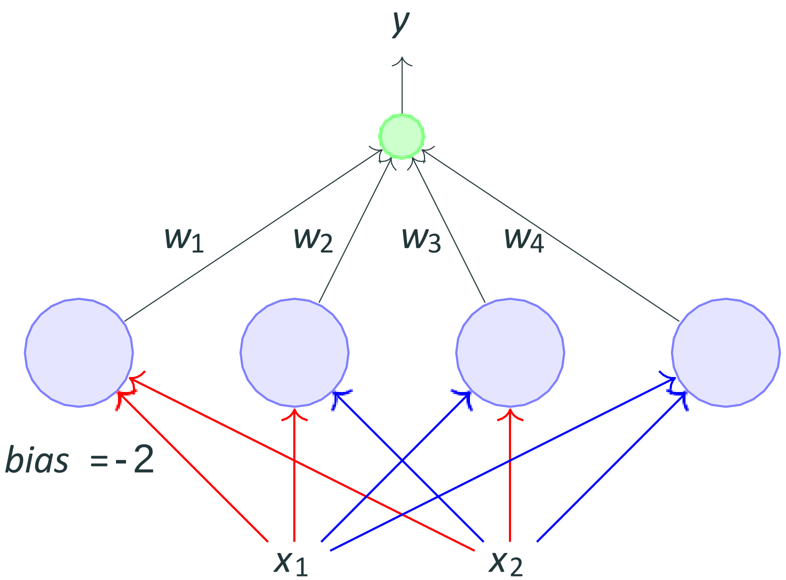
For this discussion, we will assume True = +1 and False = -1

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)



red edge indicates w = -1

blue edge indicates w = +1

For this discussion, we will assume True = +1 and False = -1

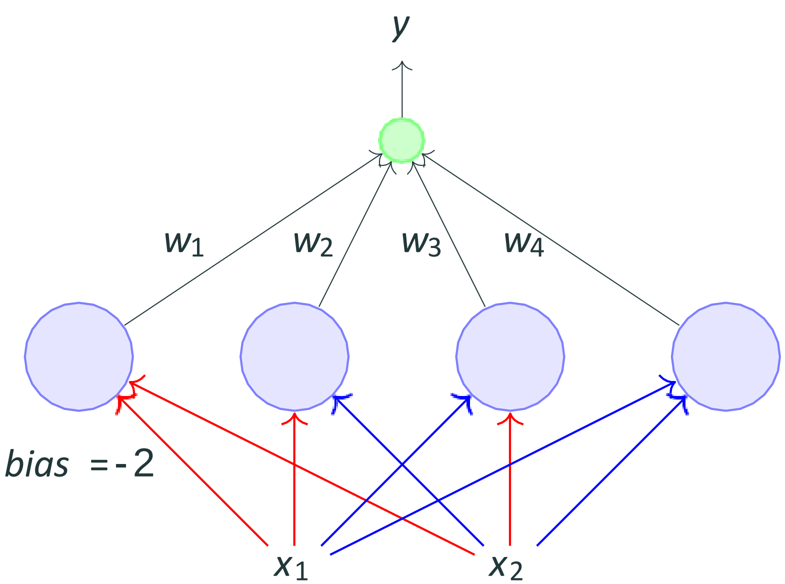
We consider 2 inputs and 4 perceptrons

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 $\geq$ 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 $w = +1$

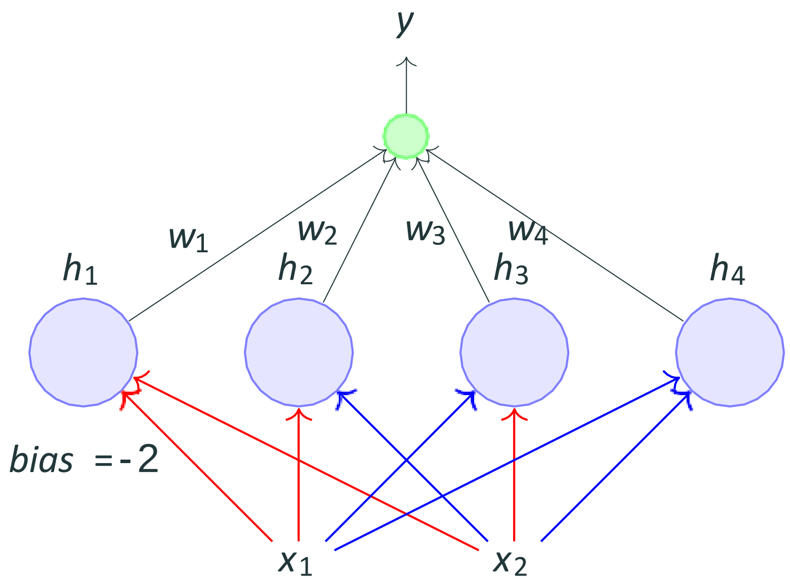
Terminology:

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**



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**

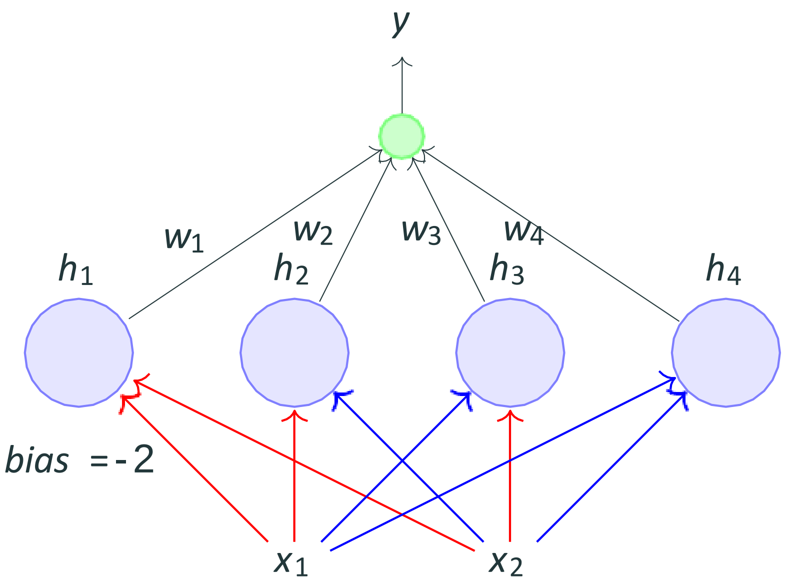
The final layer containing one output neuron is called the **output layer**

The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4

The red and blue edges are called layer 1 weights

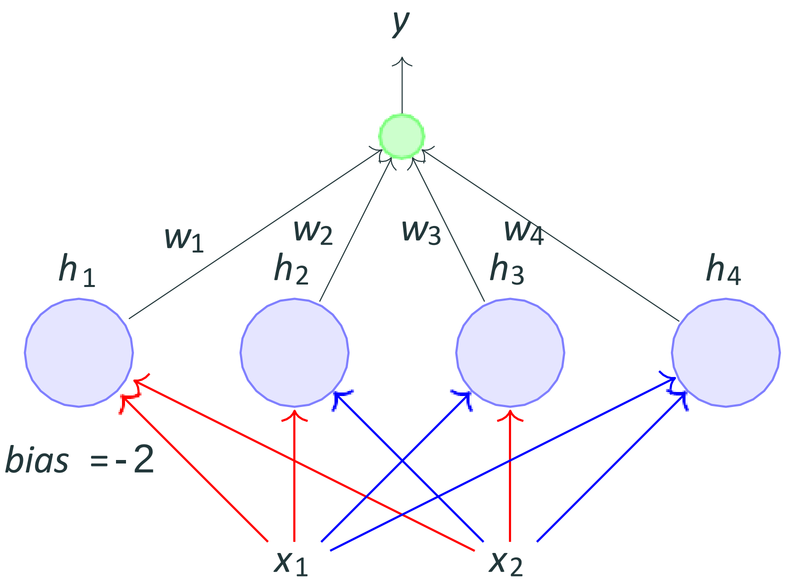
w_1, w_2, w_3, w_4 are called layer 2 weights

We claim that this network can be used to implement **any** boolean function (linearly separable or not) !



red edge indicates $w = -1$

blue edge indicates $w = +1$

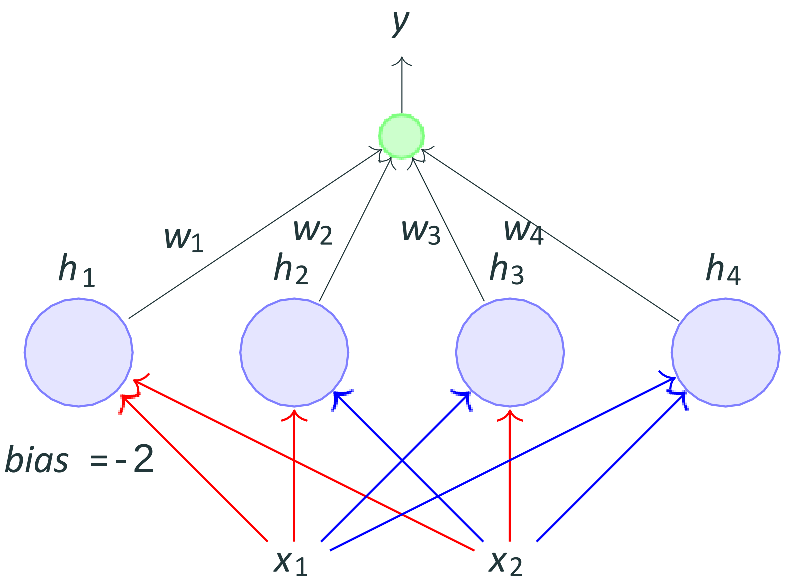


red edge indicates w = -1

blue edge indicates w = +1

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



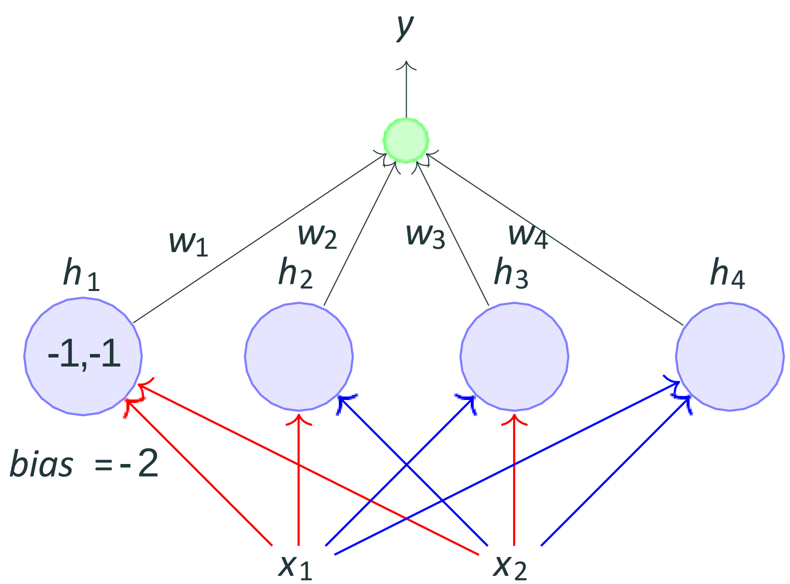
red edge indicates w = -1

blue edge indicates w = +1

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



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

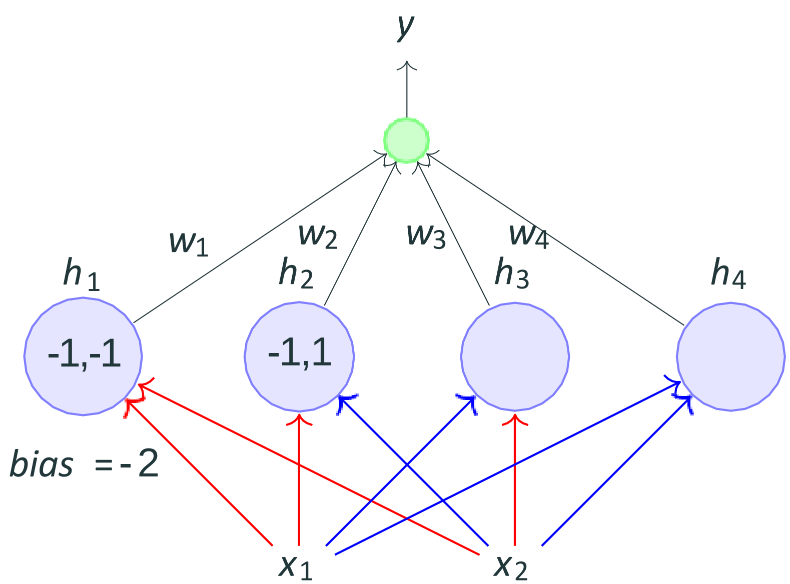
We claim that this network can be used to implement any boolean function (linearly separable or not) !

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)

the first perceptron fires for {-1,-1}



red edge indicates w = -1

blue edge indicates w = +1

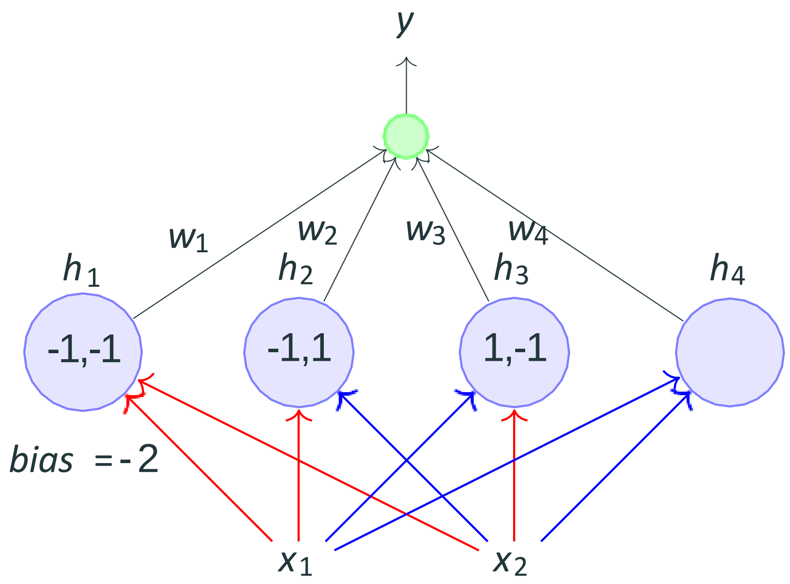
We claim that this network can be used to implement any boolean function (linearly separable or not) !

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)

the second perceptron fires for {-1,1}



red edge indicates w = -1

blue edge indicates w = +1

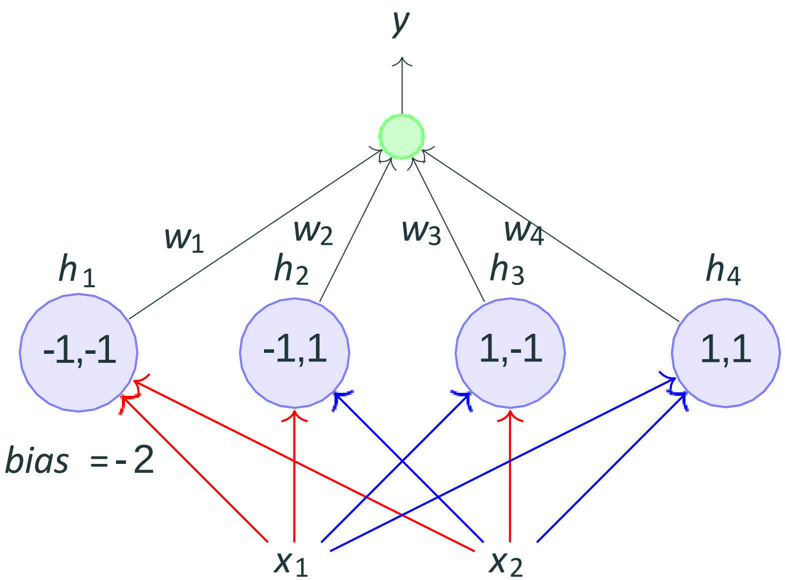
We claim that this network can be used to implement any boolean function (linearly separable or not) !

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)

the third perceptron fires for {1,-1}



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

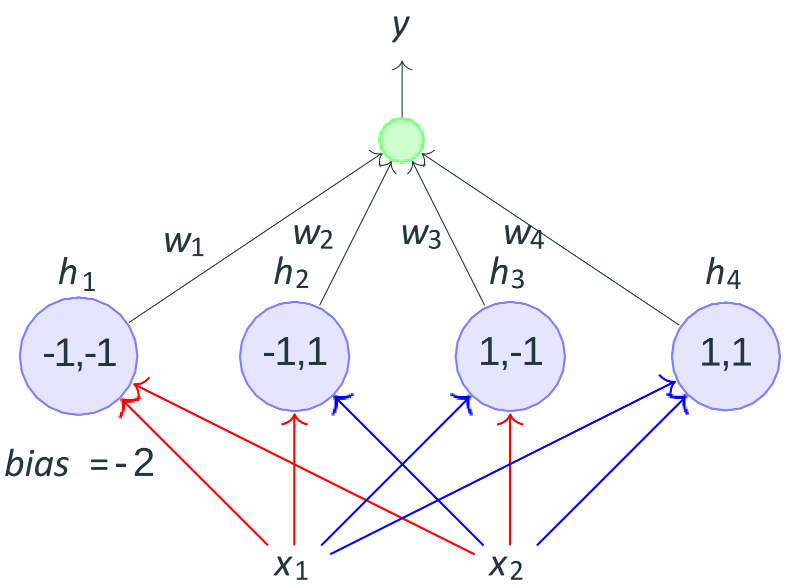
We claim that this network can be used to implement any boolean function (linearly separable or not) !

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)

the fourth perceptron fires for {1,1}



red edge indicates w = -1

blue edge indicates w = +1

We claim that this network can be used to implement any boolean function (linearly separable or not) !

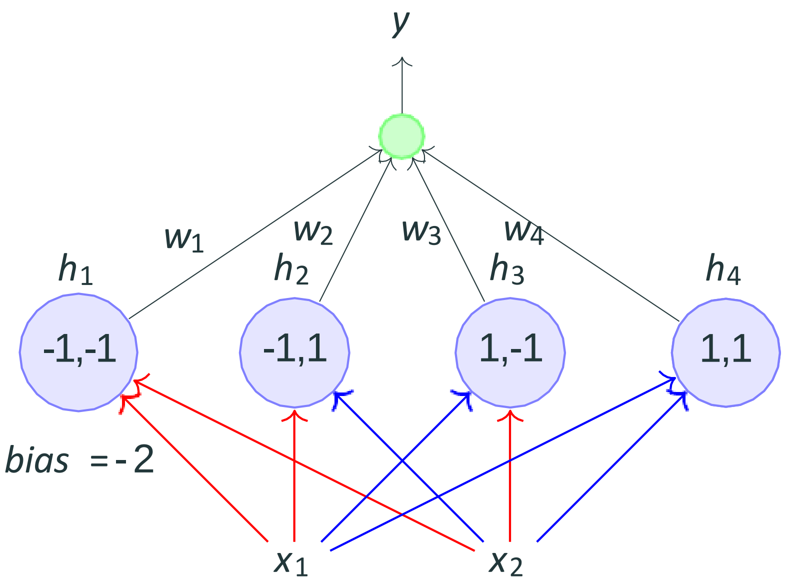
In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network

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)

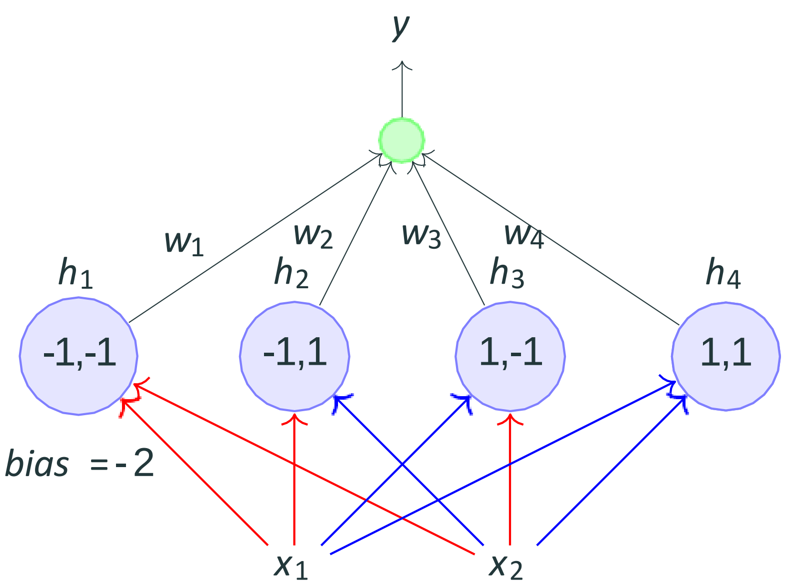
Let us see why this network works by taking an example of the XOR function

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



red edge indicates w = -1

blue edge indicates w = +1



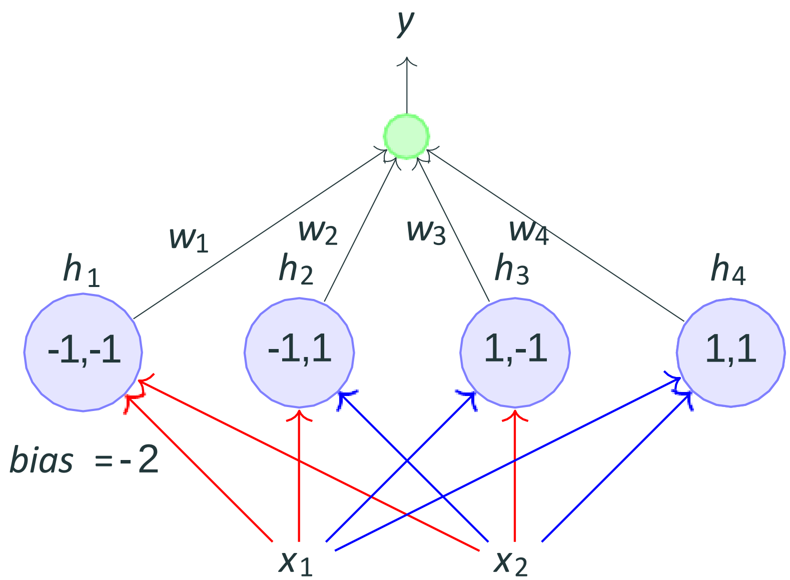
red edge indicates w = -1

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

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



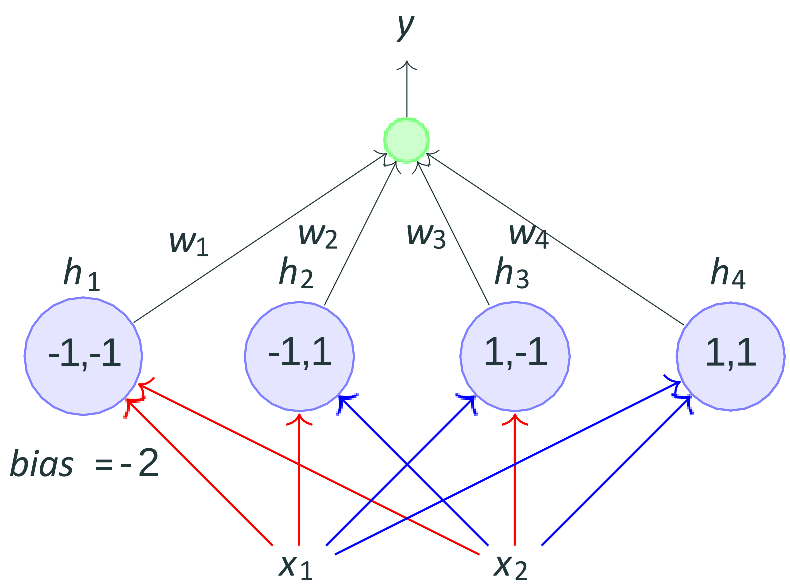
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2

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 $\sum_{i=1}^{4} w_i h_i \geq w_0$)

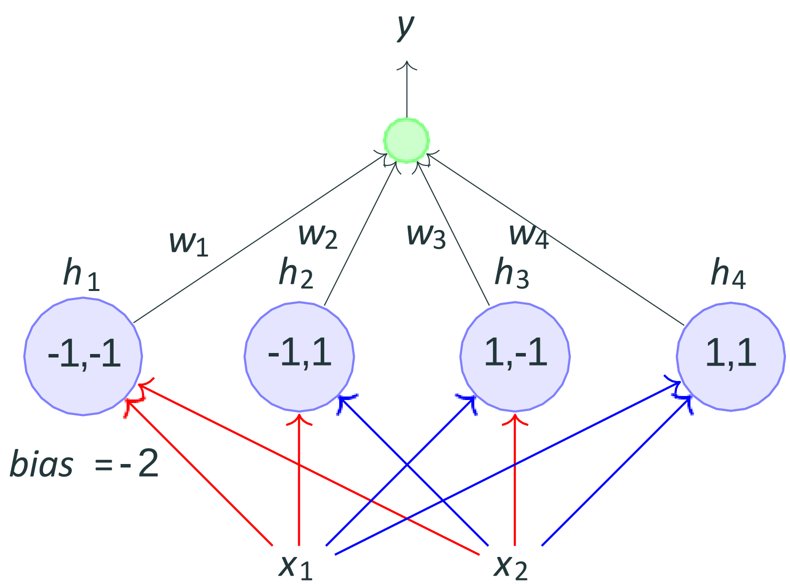
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3



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 $\sum_{i=1}^{4} w_i h_i \geq w_0$)

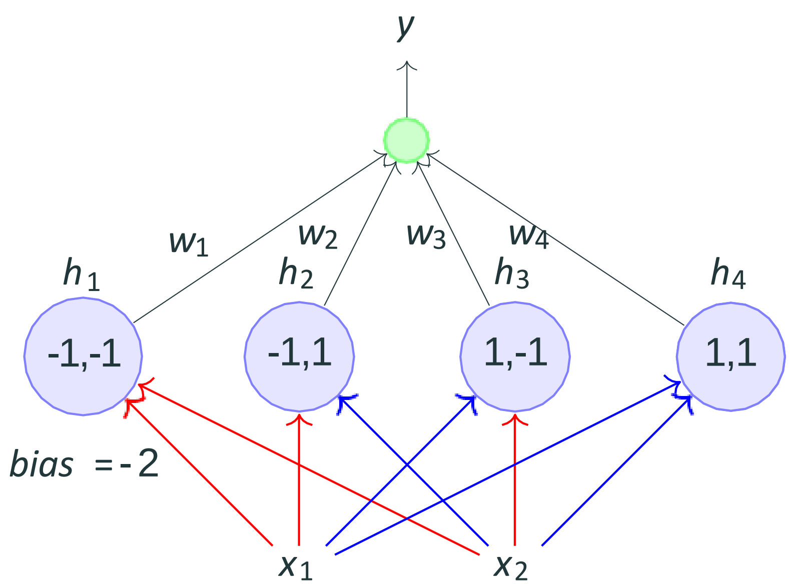


x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

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 $\sum_{i=1}^{4} w_i h_i \geq w_0$)



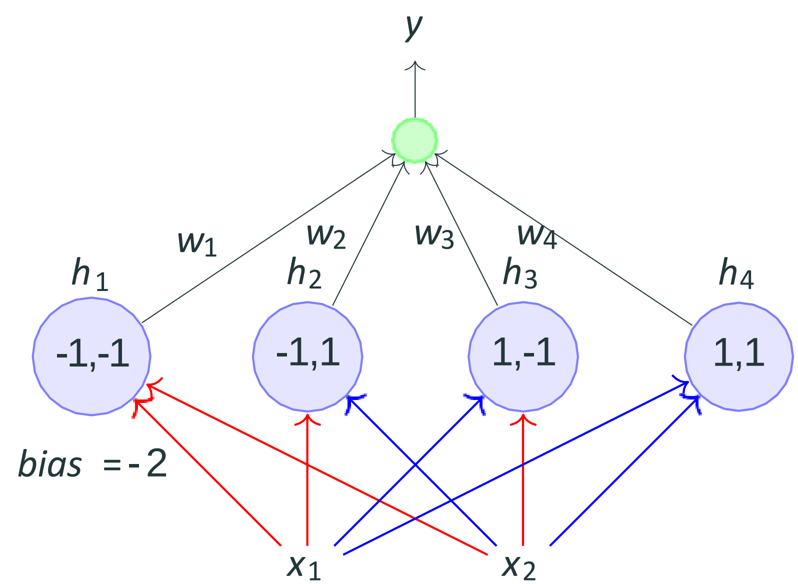
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

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

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 $\sum_{i=1}^{4} w_i h_i \geq w_0$)



red edge indicates w = -1

blue edge indicates w = +1

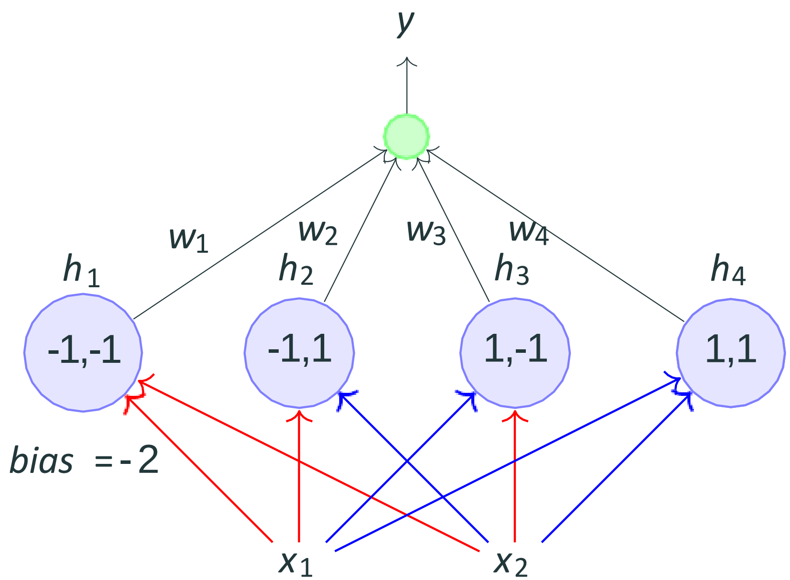
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

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

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



red edge indicates w = -1

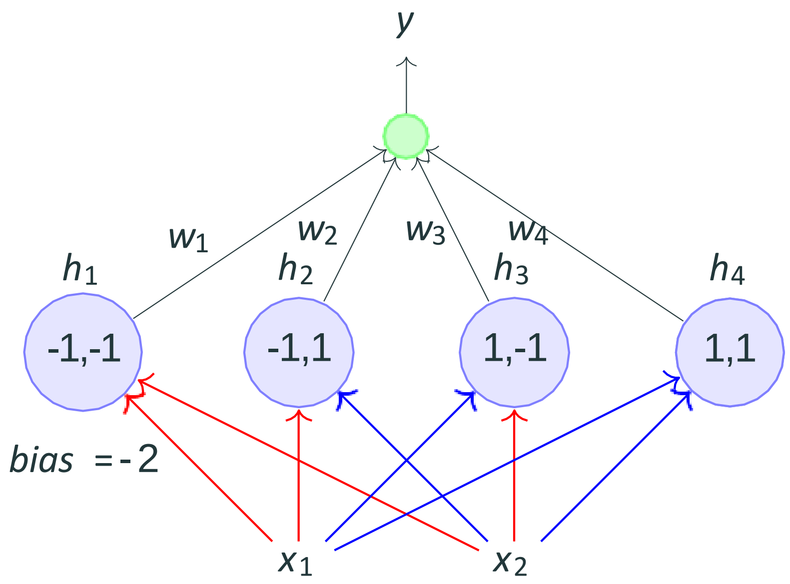
blue edge indicates w = +1

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

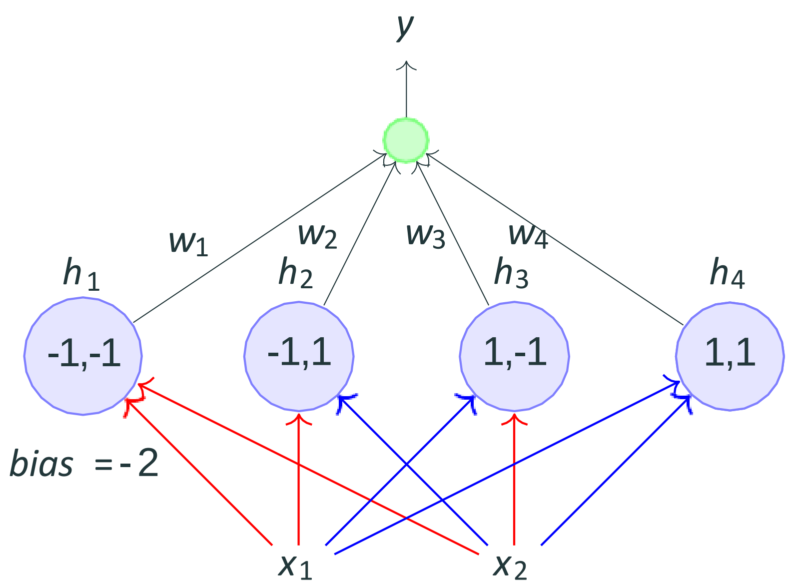
Essentially each w_i is now responsible for one of the 4 possible inputs and can be adjusted to get the desired output for that input

It should be clear that the same network can be used to represent the remaining 15 boolean functions also



red edge indicates w = -1

blue edge indicates w = +1



red edge indicates w = -1

blue edge indicates w = +1

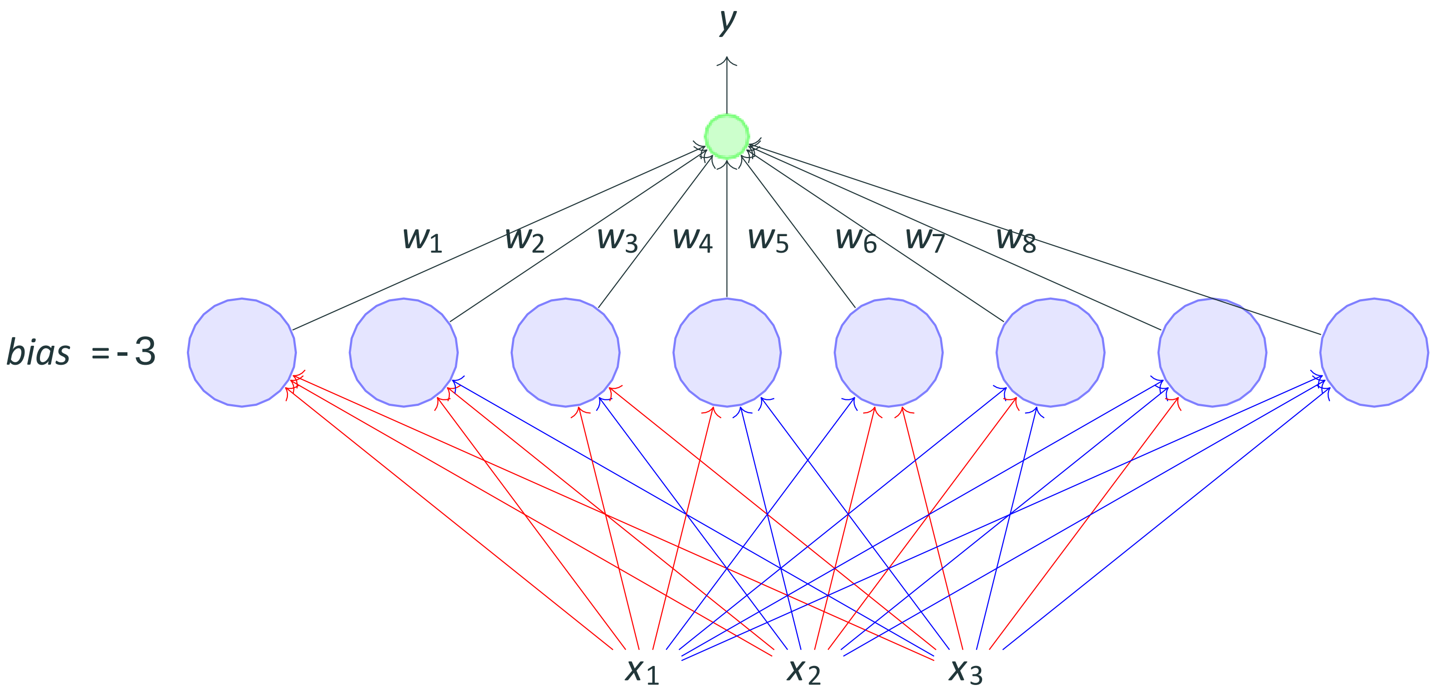
It should be clear that the same network can be used to represent the remaining 15 boolean functions also

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

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

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

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