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

![Diagram of an artificial neuron](image)

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?

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*).
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.
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

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

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

- **dendrite**: receives signals from other neurons
- **synapse**: point of connection to other neurons
- **soma**: processes the information
- **axon**: transmits the output of this neuron
• **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 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
- **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*
Let us see a very cartoonish illustration of how a neuron works.
• Let us see a very cartoonish illustration of how a neuron works

• Our sense organs interact with the outside world
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.
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).
- Of course, in reality, it is not just a single neuron which does all this.
• Of course, in reality, it is not just a single neuron which does all this
• There is a massively parallel interconnected network of neurons
• Of course, in reality, it is not just a single neuron which does all this
• There is a massively parallel interconnected network of neurons
• The sense organs relay information to the lowest layer of neurons

"DON'T YOU THINK THAT IF I WERE WRONG, I'D KNOW IT?"
• Of course, in reality, it is not just a single neuron which does all this
• 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

An average human brain has around $10^{11}$ (100 billion) neurons!
• Of course, in reality, it is not just a single neuron which does all this
• 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

DON'T YOU THINK THAT IF I WERE WRONG, I'D KNOW IT?"
• Of course, in reality, it is not just a single neuron which does all this
• 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)
Of course, in reality, it is not just a single neuron which does all this.

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).

An average human brain has around $10^{11}$ (100 billion) neurons!
This massively parallel network also ensures that there is division of work.
• 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
• 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
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.
• 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
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.
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.
The neurons in the brain are arranged in a hierarchy.

We illustrate this with the help of visual cortex (part of the brain) which deals with processing visual information. Starting from the retina, the information is relayed to several layers (follow the arrows). We observe that the layers $V_1, V_2$ to $A_1$ form a hierarchy (from identifying simple visual forms to high level objects).
• The neurons in the brain are arranged in a hierarchy
• We illustrate this with the help of visual cortex (part of the brain) which deals with processing visual information
• The neurons in the brain are arranged in a hierarchy
• We illustrate this with the help of visual cortex (part of the brain) which deals with processing visual information
• Starting from the retina, the information is relayed to several layers (follow the arrows)
• The neurons in the brain are arranged in a hierarchy
• We illustrate this with the help of visual cortex (part of the brain) which deals with processing visual information
• Starting from the retina, the information is relayed to several layers (follow the arrows)
• We observe that the layers \( V_1, V_2 \) to \( AIT \) form a hierarchy (from identifying simple visual forms to high level objects)
Sample illustration of hierarchical processing*

*Idea borrowed from Hugo Larochelle’s lecture slides
Sample illustration of hierarchical processing*

*Idea borrowed from Hugo Larochelle’s lecture slides
Sample illustration of hierarchical processing*

*Idea borrowed from Hugo Larochelle’s lecture slides
Disclaimer

- I understand very little about how the brain works!
- What you saw so far is an overly simplified explanation of how the brain works!
- But this explanation suffices for the purpose of this course!
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 $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i y = f(g(x)) = 1$ if $g(x) \geq \theta = 0$ if $g(x) < \theta$.

$\theta$ is called the thresholding parameter.

This is called Thresholding Logic.
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)

\[
g \left( x_1, x_2, \ldots, x_n \right) = g \left( x \right) = \sum_{i=1}^{n} x_i y = f \left( g \left( x \right) \right) = \begin{cases} 1 & \text{if } g \left( x \right) \geq \theta \\ 0 & \text{if } g \left( x \right) < \theta \end{cases}
\]

\( \theta \) is called the thresholding parameter.

This is called Thresholding Logic.
• McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)

\[
g(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i
\]

\[
y = f(g(x)) = \begin{cases} 
1 & \text{if } g(x) \geq \theta \\
0 & \text{if } g(x) < \theta 
\end{cases}
\]

\[
\theta \text{ is called the thresholding parameter}
\]

• This is called Thresholding Logic
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 $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$

$y = f(g(x)) = 1$ if $g(x) \geq \theta$

$y = 0$ if $g(x) < \theta$

$\theta$ is called the thresholding parameter.

This is called Thresholding Logic.
McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)

\begin{align*}
    g(x_1, x_2, \ldots, x_n) &= g(x) = \sum_{i=1}^{n} x_i y = f(g(x)) = 1 \quad \text{if } g(x) \geq \theta \\
    &= 0 \quad \text{if } g(x) < \theta
\end{align*}
McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)

- $g$ aggregates the inputs

- $y = 0$ if any $x_i$ is inhibitory, else $y = f(g(x)) = 1$ if $g(x) \geq \theta = 0$ if $g(x) < \theta$

- $\theta$ is called the thresholding parameter

- This is called Thresholding Logic
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

$y = 0$ if any $x_i$ is inhibitory, else

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

$y = f(g(x))$

$y = 1$ if $g(x) \geq \theta$

$y = 0$ if $g(x) < \theta$
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.

\[ y \in \{0, 1\} \]

\[ x_1 \ x_2 \ \ldots \ \ldots \ x_n \in \{0, 1\} \]
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.

\[
y \in \{0, 1\} = g(x_1, x_2, \ldots, x_n) \leq \theta
\]

\[
y = f(g(x_1, x_2, \ldots, x_n)) = \begin{cases} 
1 & \text{if } g(x_1, x_2, \ldots, x_n) \geq \theta \\
0 & \text{if } g(x_1, x_2, \ldots, x_n) < \theta \end{cases}
\]
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...
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

$$g(x_1, x_2, \ldots, x_n) = g(x) = \sum_{i=1}^{n} x_i$$
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
- The inputs can be excitatory or inhibitory
- $y = 0$ if any $x_i$ is inhibitory, else

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

$$y = f(g(x)) = 1 \quad if \quad g(x) \geq \theta$$
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

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

$$y = f(g(x)) = 1 \text{ if } g(x) \geq \theta$$

$$= 0 \text{ if } g(x) < \theta$$
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

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

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

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

- $\theta$ is called the thresholding parameter.
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

$$y = f(g(x)) = \begin{cases} 1 & \text{if } g(x) \geq \theta \\ 0 & \text{if } g(x) < \theta \end{cases}$$

- $\theta$ is called the thresholding parameter.
- This is called Thresholding Logic.
Let us implement some boolean functions using this McCulloch Pitts (MP) neuron ...
$y \in \{0, 1\}$

A McCulloch Pitts unit
\[ y \in \{0, 1\} \]

A McCulloch Pitts unit

AND function
\[ y \in \{0, 1\} \]

A McCulloch Pitts unit

\[ y \in \{0, 1\} \]

AND function
\( y \in \{0, 1\} \)

A McCulloch Pitts unit

\( y \in \{0, 1\} \)

AND function

\( y \in \{0, 1\} \)

OR function
\[ y \in \{0, 1\} \]

A McCulloch Pitts unit

\[ y \in \{0, 1\} \]

AND function

\[ y \in \{0, 1\} \]

OR function
\( y \in \{0, 1\} \)

A McCulloch Pitts unit

\[ y \in \{0, 1\} \]

AND function

\[ y \in \{0, 1\} \]

OR function

\[ x_1 \text{ AND } !x_2 \]

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

$y \in \{0, 1\}$

$\theta$

$x_1 \quad x_2 \quad x_3$

AND function

$y \in \{0, 1\}$

$3$

$x_1 \quad x_2 \quad x_3$

OR function

$y \in \{0, 1\}$

$1$

$x_1 \quad x_2 \quad x_3$

$x_1 \ AND \ !x_2^*$

*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0
$y \in \{0, 1\}$

A McCulloch Pitts unit

$y \in \{0, 1\}$

AND function

$y \in \{0, 1\}$

OR function

$x_1 \text{ AND } !x_2^*$

NOR function

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

$y \in \{0, 1\}$

$\theta$

$x_1, x_2, x_3$

AND function

$y \in \{0, 1\}$

$3$

$x_1, x_2, x_3$

OR function

$y \in \{0, 1\}$

$1$

$x_1, x_2, x_3$

$y \in \{0, 1\}$

$0$

$x_1, x_2$

NOR function

$x_1 \text{ AND } !x_2^*$

*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0
$y \in \{0, 1\}$

A McCulloch Pitts unit

$y \in \{0, 1\}$

AND function

$y \in \{0, 1\}$

OR function

$x_1 \text{ AND } \neg x_2^*$

$y \in \{0, 1\}$

NOR function

$y \in \{0, 1\}$

NOT function

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

\[ y \in \{0, 1\} \]

\[ x_1 \quad x_2 \quad x_3 \]

\[ \theta \]

**AND function**

\[ y \in \{0, 1\} \]

\[ x_1 \quad x_2 \quad x_3 \]

\[ 3 \]

\[ x_1 \quad x_2 \quad x_3 \]

\[ 1 \]

\[ 1 \text{ AND } !x_2^* \]

**NOR function**

\[ y \in \{0, 1\} \]

\[ x_1 \quad x_2 \]

\[ 0 \]

**OR function**

\[ y \in \{0, 1\} \]

\[ x_1 \quad x_2 \]

\[ 0 \]

**NOT function**

\[ y \in \{0, 1\} \]

\[ x_1 \]

---

*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?
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...
\[ y \in \{0, 1\} \]

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 \( \sum_{i=1}^{2} x_i - \theta = 0 \) and points lying below this line. In other words, all inputs which produce an output 0 will be on one side \( \sum_{i=1}^{2} x_i < \theta \) of the line and all inputs which produce an output 1 will lie on the other side \( \sum_{i=1}^{2} x_i \geq \theta \) of this line. Let us convince ourselves about this with a few more examples (if it is not already clear from the math).
$y \in \{0, 1\}$

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 $\sum_{i=1}^{n} x_i - \theta = 0$ and points lying below this line.

In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} x_i < \theta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^{n} x_i \geq \theta$) of this line.

Let us convince ourselves about this with a few more examples (if it is not already clear from the math). 

$\theta = 1$ 

Point $(0, 0)$ is on the line $x_1 + x_2 = \theta = 1$.
\[ y \in \{0, 1\} \]

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

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

The OR function

\[ x_1 + x_2 = \theta = 1 \]

- Points lying on or above the line \( \sum_{i=1}^{2} x_i - \theta = 0 \) and points lying below this line
- In other words, all inputs which produce an output 0 will be on one side (\( \sum_{i=1}^{2} x_i < \theta \)) of the line and all inputs which produce an output 1 will lie on the other side (\( \sum_{i=1}^{2} x_i \geq \theta \)) of this line

Let us convince ourselves about this with a few more examples (if it is not already clear from the math)
A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves.

Points lying on or above the line \( \sum_{i=1}^{n} x_i - \theta = 0 \) and points lying below this line.

In other words, all inputs which produce an output 0 will be on one side (\( \sum_{i=1}^{n} x_i < \theta \)) of the line and all inputs which produce an output 1 will lie on the other side (\( \sum_{i=1}^{n} x_i \geq \theta \)) of this line.

Let us convince ourselves about this with a few more examples (if it is not already clear from the math).
A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves.

Points lying on or above the line \( \sum_{i=1}^{n} x_i - \theta = 0 \) and points lying below this line.

In other words, all inputs which produce an output 0 will be on one side \( (\sum_{i=1}^{n} x_i < \theta) \) of the line and all inputs which produce an output 1 will lie on the other side \( (\sum_{i=1}^{n} x_i \geq \theta) \) of this line.
A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves.

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

In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} x_i < \theta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^{n} x_i \geq \theta$) of this line.

Let us convince ourselves about this with a few more examples (if it is not already clear from the math).
\[ y \in \{0, 1\} \]

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

**AND function**
$y \in \{0, 1\}$

**AND function**

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

The diagram illustrates the logic of an AND function with inputs $x_1$ and $x_2$, and output $y$. Points $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$ are marked to show the possible input combinations and their corresponding outputs.
$y \in \{0, 1\}$

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

AND function

\[
x_1 + x_2 = \theta = 2
\]
\( y \in \{0, 1\} \)

AND function
\[
x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2
\]

\((0, 1)\) \quad \((1, 1)\)

Tautology (always ON)
\[ y \in \{0, 1\} \]

**AND function**

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

**Tautology (always ON)**

\[ x_1 + x_2 = \theta = 2 \]
\[ y \in \{0, 1\} \]

**AND function**

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

\[ x_1 + x_2 = \theta = 2 \]

\[ (0, 0), (1, 0), (0, 1), (1, 1) \]

**Tautology (always ON)**

\[ y \in \{0, 1\} \]

\[ x_1 \]

\[ (0, 0), (1, 0), (0, 1), (1, 1) \]
\( y \in \{0, 1\} \)

**AND function**
\[
x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2
\]

\( x_1 + x_2 = \theta = 2 \)

**Tautology (always ON)**
\[
x_1 + x_2 = \theta = 0
\]
What if we have more than 2 inputs?

What if we have more than 2 inputs?

What if we have more than 2 inputs?
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.
What if we have more than 2 inputs?  
Well, instead of a line we will have a plane
\( y \in \{0, 1\} \)

- 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.
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.

\[
\begin{align*}
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 &= \theta = 1
\end{align*}
\]
The story so far ...

- A single McCulloch Pitts Neuron can be used to represent boolean functions which are linearly separable
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 inputs 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?
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?
• Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)
• Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)
• Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)

• A more general computational model than McCulloch–Pitts neurons
• Frank Rosenblatt, an American psychologist, proposed 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
Frank Rosenblatt, an American psychologist, proposed 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
Frank Rosenblatt, an American psychologist, proposed 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 Papert (1969) - their model is referred to as the **perceptron** model here
A more accepted convention, $y = 1$ if $n \sum_{i=0}^{n} w_i x_i \geq 0$.

Rewriting the above, $y = 1$ if $n \sum_{i=1}^{n} w_i x_i - \theta \geq 0$. 

$w_1, w_2, \ldots, w_n$ are the weights, and $x_1, x_2, \ldots, x_n$ are the inputs.
A more accepted convention, $y = 1$ if $n \sum_{i=0}^{n} w_i \cdot x_i \geq 0$.

Rewriting the above, $y = 1$ if $\sum_{i=1}^{n} w_i \cdot x_i \geq \theta$.
A more accepted convention, $y = 1$ if $\sum_{i=0}^{n} w_i x_i \geq 0$

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

where, $x_0 = 1$ and $w_0 = -\theta$

Rewriting the above,

$y = 1$ if $\sum_{i=1}^{n} w_i x_i \geq \theta$

$= 0$ if $\sum_{i=1}^{n} w_i x_i < \theta$
y = 1 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i \geq \theta
\]
\[= 0 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i < \theta
\]

Rewriting the above,
A more accepted convention,

\[ y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i \geq \theta \]

\[ = 0 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i < \theta \]

Rewriting the above,

\[ y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \]
A more accepted convention, $y = 1$ if $\sum_{i=1}^{n} w_i x_i \geq 0$

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

Rewriting the above,

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

$= 0$ if $\sum_{i=1}^{n} w_i x_i - \theta < 0$
A more accepted convention,

\[ y = 1 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i \geq \theta \]
\[ = 0 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i < \theta \]

Rewriting the above,

\[ y = 1 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \]
\[ = 0 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i - \theta < 0 \]
A more accepted convention,

\[ y = 1 \text{ if } \sum_{i=0}^{n} w_i \cdot x_i \geq 0 \]

\[ = 0 \text{ if } \sum_{i=0}^{n} w_i \cdot x_i < 0 \]

Rewriting the above,

\[ y = 1 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i \geq \theta \]

\[ = 0 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i < \theta \]
A more accepted convention,

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i \cdot x_i \geq 0 \]

\[ y = 0 \quad \text{if} \quad \sum_{i=0}^{n} w_i \cdot x_i < 0 \]

Rewriting the above,

\[ y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \]

\[ y = 0 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i - \theta < 0 \]

\[ \text{where, } \quad x_0 = 1 \quad \text{and} \quad w_0 = -\theta \]
A more accepted convention,

\[
y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i \cdot x_i \geq 0
\]

\[
y = 0 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i < \theta
\]

Rewriting the above,

\[
y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0
\]

\[
y = 0 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i - \theta < 0
\]

where, \( x_0 = 1 \) and \( w_0 = -\theta \)
A more accepted convention,

\[
y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i \cdot x_i \geq 0
\]

\[
y = 0 \quad \text{if} \quad \sum_{i=0}^{n} w_i \cdot x_i < 0
\]

where, \( x_0 = 1 \) and \( w_0 = -\theta \)

Rewriting the above,

\[
y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i \geq \theta
\]

\[
y = 0 \quad \text{if} \quad \sum_{i=1}^{n} w_i \cdot x_i < \theta
\]
We will now try to answer the following questions:

- Why are we trying to implement boolean functions?
- Why do we need weights?
- Why is $w_0 = -\theta$ called the bias?
Consider the task of predicting whether we would like a movie or not

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

\[ y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \]

\[ w_0 = -\theta \]

\[ x_0 = 1 \]

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 \( \text{isDirectorNolan} \) 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 \( \theta \) by assigning a high weight to \( \text{isDirectorNolan} \).
Consider the task of predicting whether we would like a movie or not.

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

\[
\begin{align*}
    w_0 &= -\theta \\
    x_0 &= 1 \\
    x_1 \\
    x_2 \\
    x_3 \\
\end{align*}
\]
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 \textit{isDirectorNolan} as compared to the other inputs

\begin{align*}
x_0 &= 1 \\
x_1 &= isActorDamon \\
x_2 &= isGenreThriller \\
x_3 &= isDirectorNolan
\end{align*}
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 \textit{isDirectorNolan} as compared to the other inputs.

Specifically, even if the actor is not \textit{Matt Damon} and the genre is not \textit{thriller} we would still want to cross the threshold $\theta$ by assigning a high weight to \textit{isDirectorNolan}.

\[
\begin{align*}
\text{\textit{x}_0} &= 1 \\
\text{\textit{x}_1} &= \text{isActorDamon} \\
\text{\textit{x}_2} &= \text{isGenreThriller} \\
\text{\textit{x}_3} &= \text{isDirectorNolan}
\end{align*}
\]
• $w_0$ is called the bias as it represents the prior (prejudice).

\[ w_0 = -\theta \]
\[ x_0 = 1 \]
\[ \begin{align*}
    x_1 &= \text{isActorDamon} \\
    x_2 &= \text{isGenreThriller} \\
    x_3 &= \text{isDirectorNolan}
\end{align*} \]
• \( w_0 \) 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 \([\theta = 0]\)

\[
\begin{align*}
\begin{align*}
x_0 &= 1, \\
x_1 &= \text{isActorDamon}, \\
x_2 &= \text{isGenreThriller}, \\
x_3 &= \text{isDirectorNolan}
\end{align*}
\]
• $w_0$ 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 [$\theta = 0$]
• On the other hand, a selective viewer may only watch thrillers starring Matt Damon and directed by Nolan [$\theta = 3$]

$x_0 = 1$
$x_1 = \text{isActorDamon}$
$x_2 = \text{isGenreThriller}$
$x_3 = \text{isDirectorNolan}$
$w_0$ 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 [$\theta = 0$].

On the other hand, a selective viewer may only watch thrillers starring Matt Damon and directed by Nolan [$\theta = 3$].

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

$x_1 = \text{isActorDamon}$
$x_2 = \text{isGenreThriller}$
$x_3 = \text{isDirectorNolan}$
What kind of functions can be implemented using the perceptron? Any difference from McCulloch Pitts neurons?
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} x_i \geq 0 \]

\[ = 0 \quad \text{if} \quad \sum_{i=0}^{n} x_i < 0 \]

Perceptron

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i \times x_i \geq 0 \]

\[ = 0 \quad \text{if} \quad \sum_{i=0}^{n} w_i \times x_i < 0 \]
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

\[ y = 1 \text{ if } \sum_{i=0}^{n} x_i \geq 0 \]
\[ = 0 \text{ if } \sum_{i=0}^{n} x_i < 0 \]

Perceptron

\[ y = 1 \text{ if } \sum_{i=0}^{n} w_i \cdot x_i \geq 0 \]
\[ = 0 \text{ if } \sum_{i=0}^{n} w_i \cdot x_i < 0 \]

- From the equations it should be clear that even a perceptron separates the input space into two halves.
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} x_i \geq 0 \]
\[ = 0 \quad \text{if} \quad \sum_{i=0}^{n} x_i < 0 \]

Perceptron

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i \times x_i \geq 0 \]
\[ = 0 \quad \text{if} \quad \sum_{i=0}^{n} w_i \times x_i < 0 \]

- 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
- The weights (including 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)
McCulloch Pitts Neuron (assuming no inhibitory inputs)

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

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

Perceptron

\[ y = 1 \quad if \quad \sum_{i=0}^{n} w_i \times x_i \geq 0 \]

\[ = 0 \quad if \quad \sum_{i=0}^{n} w_i \times x_i < 0 \]

- 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
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} x_i \geq 0 \]

\[ = 0 \quad \text{if} \quad \sum_{i=0}^{n} x_i < 0 \]

Perceptron

\[ y = 1 \quad \text{if} \quad \sum_{i=0}^{n} w_i \ast x_i \geq 0 \]

\[ = 0 \quad \text{if} \quad \sum_{i=0}^{n} w_i \ast x_i < 0 \]

- 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 (including 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)
McCulloch Pitts Neuron 
(assuming no inhibitory inputs)

\[
y = 1 \iff \sum_{i=0}^{n} x_i \geq 0
\]

\[
y = 0 \iff \sum_{i=0}^{n} x_i < 0
\]

Perceptron

\[
y = 1 \iff \sum_{i=0}^{n} w_i * x_i \geq 0
\]

\[
y = 0 \iff \sum_{i=0}^{n} w_i * x_i < 0
\]

- 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 (including threshold) can be learned and the inputs can be real valued
McCulloch Pitts Neuron
(assuming no inhibitory inputs)

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

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

Perceptron

\[ y = 1 \quad if \quad \sum_{i=0}^{n} w_i \ast x_i \geq 0 \]

\[ = 0 \quad if \quad \sum_{i=0}^{n} w_i \ast x_i < 0 \]

- 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 (including 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)
\[
\begin{align*}
&w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \\
&\Rightarrow w_0 < 0 \\
&w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \\
&\Rightarrow w_2 \geq -w_0 \\
&w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \\
&\Rightarrow w_1 \geq -w_0 \\
&w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \\
&\Rightarrow w_1 + w_2 \geq -w_0
\end{align*}
\]

One possible solution to this set of inequalities is

\[w_0 = -1, w_1 = 1, w_2 = 1\]

(and various other solutions are possible)

\[
\begin{array}{c|c|c|c|c|c}
\hline
x_1 & x_2 & OR \\
\hline
0 & 0 & \\
\hline
\end{array}
\]

Note that we can come up with a similar set of inequalities and find the value of \(\theta\) for a McCulloch Pitts neuron also (Try it!)
\[
\begin{array}{ccc}
  x_1 & x_2 & \text{OR} \\
  0 & 0 & 0 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$w_0 + \sum_{i=1}^{2} w_i x_i$
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### OR

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ w_0 + \sum_{i=1}^{2} w_i x_i < 0 \]
\[
\begin{array}{ccc}
\hline
x_1 & x_2 & \text{OR} \\
\hline
0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
\hline
\end{array}
\]

One possible solution to this set of inequalities is 
\[w_0 = -1, w_1 = 1, w_2 = 1\] (and various other solutions are possible).

Note that we can come up with a similar set of inequalities and find the value of \(\theta\) for a McCulloch Pitts neuron also (Try it!).
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

One possible solution to this set of inequalities is $w_0 = -1$, $w_1 = 1$, $w_2 = 1$ (and various other solutions are possible).
\[
\begin{array}{ccc}
  x_1 & x_2 & \text{OR} \\
  0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
  1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
  0 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
\[
\begin{array}{ccc}
 x_1 & x_2 & \text{OR} \\
0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
0 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
1 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
\end{array}
\]

\[w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0\]
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0 \]
\[ w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0 \]
### OR

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$w_0 + \sum_{i=1}^{2} w_i x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i &lt; 0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$</td>
</tr>
</tbody>
</table>

\begin{align*}
  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 \\
\end{align*}

\[ w_0 + w_1 \cdot 0 + w_2 \cdot 0 + 1 \leq 0 \implies w_0 \leq 0 \]
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i &lt; 0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$</td>
</tr>
</tbody>
</table>

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

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

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

$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0$
\[
\begin{array}{ccc}
 x_1 & x_2 & \text{OR} \\
 0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
 1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
 0 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
 1 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
\end{array}
\]

\[w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0\]
\[w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0\]
\[w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0\]
\[w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0\]

- 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)
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0
\]
\[
w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0
\]
\[
w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0
\]
\[
w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0
\]

- 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)
\[
\begin{array}{ccc}
x_1 & x_2 & \text{OR} \\
0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
0 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
1 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
\end{array}
\]

\[
w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0 \\
w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0 \\
\]

- 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)
\[
\begin{array}{c|c|c}
\text{x}_1 & \text{x}_2 & \text{OR} \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{align*}
\text{w}_0 + \sum_{i=1}^{2} \text{w}_i \times i &< 0 \\
\text{w}_0 + \sum_{i=1}^{2} \text{w}_i \times i &\geq 0 \\
\text{w}_0 + \sum_{i=1}^{2} \text{w}_i \times i &\geq 0
\end{align*}
\]

\[
\begin{align*}
\text{w}_0 + \text{w}_1 \cdot 0 + \text{w}_2 \cdot 0 &< 0 \implies \text{w}_0 < 0 \\
\text{w}_0 + \text{w}_1 \cdot 0 + \text{w}_2 \cdot 1 &\geq 0 \implies \text{w}_2 \geq -\text{w}_0 \\
\text{w}_0 + \text{w}_1 \cdot 1 + \text{w}_2 \cdot 0 &\geq 0 \implies \text{w}_1 \geq -\text{w}_0 \\
\text{w}_0 + \text{w}_1 \cdot 1 + \text{w}_2 \cdot 1 &\geq 0 \implies \text{w}_1 + \text{w}_2 \geq -\text{w}_0
\end{align*}
\]

- 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)

\[ -1 + 1.1x_1 + 1.1x_2 = 0 \]

- Note that we can come up with a similar set of inequalities and find the value of \( \theta \) for a McCulloch Pitts neuron also
\[ w_0 + \sum_{i=1}^{2} w_i x_i < 0 \]
\[ w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \]
\[ w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \]

\[ w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0 \]
\[ w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0 \]
\[ w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0 \]
\[ w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0 \]

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 \( \theta \) for a McCulloch Pitts neuron also (Try it!)
Module 2.4: Errors and Error Surfaces
Let us fix the threshold \((-w_0 = 1\) and try different values of \(w_1, w_2\)
Let us fix the threshold \((-w_0 = 1)\) and try different values of \(w_1, w_2\).

Say, \(w_1 = -1, w_2 = -1\)

\[-1 + 1.1x_1 + 1.1x_2 = 0\]

\[-1 + (-1)x_1 + (-1)x_2 = 0\]
• 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?
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

\[-1 + \frac{1}{1} x_1 + \frac{1}{1} x_2 = 0\]

\((0, 1)\) \hspace{1cm} \((1, 1)\)

\[-1 + (-1)x_1 + (-1)x_2 = 0\]

\((0, 0)\) \hspace{1cm} \((1, 0)\)
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

Let us try some more values of $w_1, w_2$ and note how many errors we make

\[ -1 + 1.1x_1 + 1.1x_2 = 0 \]

\[ -1 + (-1)x_1 + (-1)x_2 = 0 \]
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

<table>
<thead>
<tr>
<th>(w_1)</th>
<th>(w_2)</th>
<th>errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

\[-1 + 1.1x_1 + 1.1x_2 = 0\]
\[-1 + (-1)x_1 + (-1)x_2 = 0\]
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

Let us try some more values of \(w_1, w_2\) and note how many errors we make

<table>
<thead>
<tr>
<th>(w_1)</th>
<th>(w_2)</th>
<th>errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
-1 + (1.5)x_1 + (0)x_2 = 0 \\
-1 + (-1)x_1 + (-1)x_2 = 0
\]
• 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

• Let us plot the error surface corresponding to different values of \(w_0, w_1, w_2\)

\[
\begin{array}{ccc}
  w_1 & w_2 & \text{errors} \\
  -1 & -1 & 3 \\
  1.5 & 0 & 1 \\
  0.45 & 0.45 & 3 \\
\end{array}
\]
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

Let's try some more values of \(w_1, w_2\) and note how many errors we make

<table>
<thead>
<tr>
<th>(w_1)</th>
<th>(w_2)</th>
<th>errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.45</td>
<td>0.45</td>
<td>3</td>
</tr>
</tbody>
</table>

We are interested in those values of \(w_0, w_1, w_2\) which result in 0 error
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.

Let's try some more values of \(w_1, w_2\) and note how many errors we make:

\[
\begin{array}{ccc}
  w_1 & w_2 & \text{errors} \\
  -1 & -1 & 3 \\
  1.5 & 0 & 1 \\
  0.45 & 0.45 & 3 \\
\end{array}
\]

We are interested in those values of \(w_0, w_1, w_2\) which result in 0 error.

Let us plot the error surface corresponding to different values of \(w_0, w_1, w_2\).
For ease of analysis, we will keep $w_0$ fixed (-1) and plot the error for different values of $w_1, w_2$. 

For a given $w_0, w_1, w_2$ we will compute 

$$-w_0 + w_1 \times x_1 + w_2 \times x_2$$

for all combinations of $(x_1, x_2)$ and note down how many errors we make.

For the OR function, an error occurs if $(x_1, x_2) = (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 \geq 0$ or if $(x_1, x_2) \neq (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 < 0$.

We are interested in finding an algorithm which finds the values of $w_1, w_2$ which minimize this error.
• For ease of analysis, we will keep $w_0$ fixed (-1) and plot the error for different values of $w_1, w_2$

• For a given $w_0, w_1, w_2$ we will compute $-w_0 + w_1 * x_1 + w_2 * x_2$ for all combinations of $(x_1, x_2)$ and note down how many errors we make
For ease of analysis, we will keep $w_0$ fixed (-1) and plot the error for different values of $w_1, w_2$

For a given $w_0, w_1, w_2$ we will compute $-w_0 + w_1 \times x_1 + w_2 \times x_2$ for all combinations of $(x_1, x_2)$ and note down how many errors we make

For the OR function, an error occurs if $(x_1, x_2) = (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 \geq 0$ or if $(x_1, x_2) \neq (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 < 0$
For ease of analysis, we will keep $w_0$ fixed (-1) and plot the error for different values of $w_1, w_2$

For a given $w_0, w_1, w_2$ we will compute $-w_0 + w_1 \times x_1 + w_2 \times x_2$ for all combinations of $(x_1, x_2)$ and note down how many errors we make.

For the OR function, an error occurs if $(x_1, x_2) = (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 \geq 0$ or if $(x_1, x_2) \neq (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 < 0$
For ease of analysis, we will keep $w_0$ fixed (-1) and plot the error for different values of $w_1, w_2$

For a given $w_0, w_1, w_2$ we will compute $-w_0 + w_1 \times x_1 + w_2 \times x_2$ for all combinations of $(x_1, x_2)$ and note down how many errors we make

For the OR function, an error occurs if $(x_1, x_2) = (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 \geq 0$ or if $(x_1, x_2) \neq (0, 0)$ but $-w_0 + w_1 \times x_1 + w_2 \times x_2 < 0$

We are interested in finding an algorithm which finds the values of $w_1, w_2$ which minimize this error
Module 2.5: Perceptron Learning Algorithm
We will now see a more principled approach for learning these weights and threshold but before that let us answer this question...
We will now see a more principled approach for learning these weights and threshold but before that let us answer this question...

Apart from implementing boolean functions (which does not look very interesting) what can a perceptron be used for?
We will now see a more principled approach for learning these weights and threshold but before that let us answer this question...

Apart from implementing boolean functions (which does not look very interesting) what can a perceptron be used for?

Our interest lies in the use of perceptron as a binary classifier. Let us see what this means...
Let us reconsider our problem of deciding whether to watch a movie or not.
• 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
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).
• 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)

\[
x_1 = \text{isActorDamon}
\]
\[
x_2 = \text{isGenreThriller}
\]
\[
x_3 = \text{isDirectorNolan}
\]
\[
x_4 = \text{imdbRating(scaled to 0 to 1)}
\]
\[
\ldots \quad \ldots
\]
\[
x_n = \text{criticsRating(scaled to 0 to 1)}
\]
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.

$x_0 = 1$
$x_1 = \text{isActorDamon}$
$x_2 = \text{isGenreThriller}$
$x_3 = \text{isDirectorNolan}$
$x_4 = \text{imdbRating}(\text{scaled to 0 to 1})$

... ... 

$x_n = \text{criticsRating}(\text{scaled to 0 to 1})$
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 $w_0, w_1, w_2, \ldots, w_m$).

\[ x_1 = isActorDamon \]
\[ x_2 = isGenreThriller \]
\[ x_3 = isDirectorNolan \]
\[ x_4 = imdbRating(scaled \text{ to } 0 \text{ to } 1) \]

... ...

\[ x_n = criticsRating(scaled \text{ to } 0 \text{ to } 1) \]
Algorithm: Perceptron Learning Algorithm

P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
  Pick random \( x \in P \cup N \);
  if \( x \in P \) and \( \sum_{i} w_i \cdot x_i < 0 \) then
    \( w = w + x \);
  end
  if \( x \in N \) and \( \sum_{i} w_i \cdot x_i \geq 0 \) then
    \( w = w - x \);
  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...
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
**Algorithm: Perceptron Learning Algorithm**

\[ P \leftarrow \text{inputs with label 1}; \]

\[ N \leftarrow \text{inputs with label 0}; \]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]

Initialize \( \mathbf{w} \) randomly;

// 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...
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( w \) randomly;

**while** !convergence **do**

end

• 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...
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]

Initialize \( \mathbf{w} \) randomly;

**while** !\textit{convergence} **do**

\[ \text{Pick random } x \in P \cup N; \]
\[ \text{if } x \in P \text{ and } \sum \mathbf{w}_i x_i < 0 \quad \text{then} \]
\[ \mathbf{w} = \mathbf{w} + x; \]
\[ \text{end} \]
\[ \text{if } x \in N \text{ and } \sum \mathbf{w}_i x_i \geq 0 \quad \text{then} \]
\[ \mathbf{w} = \mathbf{w} - x; \]
\[ \text{end} \]

**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...
Algorithm: Perceptron Learning Algorithm

\( P \leftarrow \text{inputs with label } 1 \);  
\( N \leftarrow \text{inputs with label } 0 \);  
Initialize \( \mathbf{w} \) randomly;  
\textbf{while } !\text{convergence} \textbf{ do}  
\hspace{1em} \text{Pick random } \mathbf{x} \in P \cup N ;  
\textbf{end}  
//the algorithm converges when all the inputs  
\hspace{1em} \text{are classified correctly}
Algorithm: Perceptron Learning Algorithm

\( P \leftarrow \text{inputs with label } 1; \)

\( N \leftarrow \text{inputs with label } 0; \)

Initialize \( \mathbf{w} \) randomly;

\[
\text{while } \neg \text{convergence do}
\]

\[
\quad \text{Pick random } \mathbf{x} \in P \cup N ;
\]

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

\[
\qquad \text{end}
\]

\[
\text{end}
\]

//the algorithm converges when all the inputs
are classified correctly
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( w \) randomly;

\textbf{while} !convergence \textbf{do}

\hspace{1em} \text{Pick random } x \in P \cup N ;

\hspace{1em} \text{if } x \in P \text{ and } \sum_{i=0}^{n} w_i \times x_i < 0 \text{ then}

\hspace{1.5em} w = w + x ;

\hspace{1em} \text{end}

\textbf{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...
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]
Initialize \( \mathbf{w} \) randomly;

\textbf{while} !convergence \textbf{do}

\hspace{1em} \text{Pick random} \ x \in P \cup N ;

\hspace{1em} \text{if} \ x \in P \ \text{and} \ \sum_{i=0}^{n} w_i \cdot x_i < 0 \ \text{then}

\hspace{2em} \mathbf{w} = \mathbf{w} + \mathbf{x} ;

\hspace{1em} \text{end}

\hspace{1em} \text{if} \ x \in N \ \text{and} \ \sum_{i=0}^{n} w_i \cdot x_i \geq 0 \ \text{then}

\hspace{2em} \text{end}

\textbf{end}

//the algorithm converges when all the inputs are classified correctly
Algorithm: Perceptron Learning Algorithm

\( P \leftarrow \text{inputs with label 1}; \)
\( N \leftarrow \text{inputs with label 0}; \)
Initialize \( \mathbf{w} \) randomly;

\[ \text{while } \neg \text{convergence do} \]
\[ \quad \text{Pick random } \mathbf{x} \in P \cup N; \]
\[ \quad \text{if } \mathbf{x} \in P \text{ and } \sum_{i=0}^{n} w_i \cdot x_i < 0 \text{ then} \]
\[ \quad \quad \mathbf{w} = \mathbf{w} + \mathbf{x}; \]
\[ \quad \text{end} \]
\[ \quad \text{if } \mathbf{x} \in N \text{ and } \sum_{i=0}^{n} w_i \cdot x_i \geq 0 \text{ then} \]
\[ \quad \quad \mathbf{w} = \mathbf{w} - \mathbf{x}; \]
\[ \quad \text{end} \]
\[ \text{end} \]

//the algorithm converges when all the inputs are classified correctly
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]

Initialize \( w \) randomly;

\[ \text{while } \neg \text{convergence do} \]
\[ \quad \text{Pick random } x \in P \cup N; \]
\[ \quad \text{if } x \in P \text{ and } \sum_{i=0}^{n} w_i * x_i < 0 \text{ then} \]
\[ \quad \quad w = w + x; \]
\[ \quad \text{end} \]
\[ \quad \text{if } x \in N \text{ and } \sum_{i=0}^{n} w_i * x_i \geq 0 \text{ then} \]
\[ \quad \quad w = w - x; \]
\[ \quad \text{end} \]
\[ \text{end} \]

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

- Why would this work?
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( \mathbf{w} \) randomly;

\textbf{while} !convergence \textbf{do}

\hspace{1em} \text{Pick random } \mathbf{x} \in P \cup N ;

\hspace{2em} \text{if } \mathbf{x} \in P \text{ and } \sum_{i=0}^{n} w_i \cdot x_i < 0 \text{ then}

\hspace{3em} \mathbf{w} = \mathbf{w} + \mathbf{x} ;

\hspace{2em} \text{end}

\hspace{2em} \text{if } \mathbf{x} \in N \text{ and } \sum_{i=0}^{n} w_i \cdot x_i \geq 0 \text{ then}

\hspace{3em} \mathbf{w} = \mathbf{w} - \mathbf{x} ;

\hspace{2em} \text{end}

\textbf{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 $w$ and $x$. We can thus rewrite the perceptron rule as

$$w^T x = 0$$

which divides the input space into two halves. Every point $(x)$ on this line satisfies the equation $w^T x = 0$.

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

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

Since the vector $w$ is perpendicular to every point on the line, it is actually perpendicular to the line itself.
Consider two vectors $\mathbf{w}$ and $\mathbf{x}$

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

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

$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x} = \sum_{i=0}^{n} w_i x_i$
Consider two vectors $\mathbf{w}$ and $\mathbf{x}$

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

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

\[
\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x} = \sum_{i=0}^{n} w_i \ast x_i
\]

We can thus rewrite the perceptron rule as

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 $(\alpha)$ between $\mathbf{w}$ and any point $(\mathbf{x})$ which lies on this line?

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

Since the vector $\mathbf{w}$ is perpendicular to every point on the line it is actually perpendicular to the line itself.
Consider two vectors $\mathbf{w}$ and $\mathbf{x}$

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

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

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

We can thus rewrite the perceptron rule as

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 ($\alpha$) between $\mathbf{w}$ and any point $(\mathbf{x})$ which lies on this line?

The angle is $90^\circ$ ($\because \cos \alpha = \frac{\mathbf{w} \cdot \mathbf{x}}{|| \mathbf{w} || \times || \mathbf{x} ||} = 0$)

Since the vector $\mathbf{w}$ is perpendicular to every point on the line it is actually perpendicular to the line itself.

35
Consider two vectors \( \mathbf{w} \) and \( \mathbf{x} \)

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

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

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

We can thus rewrite the perceptron rule as

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

\[
y = 0 \quad if \quad \mathbf{w}^T \mathbf{x} < 0
\]
Consider two vectors $w$ and $x$

$$w = [w_0, w_1, w_2, \ldots, w_n]$$
$$x = [1, x_1, x_2, \ldots, x_n]$$

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

We can thus rewrite the perceptron rule as

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

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

\[
\mathbf{w} = [w_0, w_1, w_2, \ldots, w_n]
\]
\[
\mathbf{x} = [1, x_1, x_2, \ldots, x_n]
\]

\[
\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x} = \sum_{i=0}^{n} w_i \times x_i
\]

We can thus rewrite the perceptron rule as

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

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 \( \alpha \) between \( \mathbf{w} \) and any point \( \mathbf{x} \) which lies on this line?

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

Since the vector \( \mathbf{w} \) is perpendicular to every point on the line it is actually perpendicular to the line itself.
Consider two vectors \( \mathbf{w} \) and \( \mathbf{x} \)

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

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

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

We can thus rewrite the perceptron rule as

\[
y = 1 \quad \text{if} \quad \mathbf{w}^T \mathbf{x} \geq 0
\]

\[
y = 0 \quad \text{if} \quad \mathbf{w}^T \mathbf{x} < 0
\]

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 \( \alpha \) between \( \mathbf{w} \) and any point \( \mathbf{x} \) which lies on this line?
Consider two vectors \( \mathbf{w} \) and \( \mathbf{x} \)

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

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

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

We can thus rewrite the perceptron rule as

\[
y = 1 \quad \text{if} \quad \mathbf{w}^T \mathbf{x} \geq 0
\]

\[
y = 0 \quad \text{if} \quad \mathbf{w}^T \mathbf{x} < 0
\]

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 (\( \alpha \)) between \( \mathbf{w} \) and any point (\( \mathbf{x} \)) which lies on this line?

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

Since the vector \( \mathbf{w} \) is perpendicular to every point on the line it is actually perpendicular to the line itself.
Consider two vectors \( \mathbf{w} \) and \( \mathbf{x} \)

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

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

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

We can thus rewrite the perceptron rule as

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

\[
y = 0 \quad if \quad \mathbf{w}^T \mathbf{x} < 0
\]

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 \( \alpha \) between \( \mathbf{w} \) and any point \( \mathbf{x} \) which lies on this line?

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

Since the vector \( \mathbf{w} \) is perpendicular to every point on the line it is actually perpendicular to the line itself.
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 $\mathbf{w}$?

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 $\mathbf{w}$?

Of course, this also follows from the formula

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

Keeping this picture in mind let us revisit the algorithm.
Consider some points (vectors) which lie in the positive half space of this line (i.e., $w^T x \geq 0$)

What will be the angle between any such vector and $w$?

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

What will be the angle between any such vector and $w$?

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

\[ w^T x = 0 \]
• Consider some points (vectors) which lie in the positive half space of this line (i.e., $w^T x \geq 0$)

• What will be the angle between any such vector and $w$?

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

• Keeping this picture in mind let us revisit the algorithm.
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 $90^\circ$

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 $\mathbf{w}$?

Of course, this also follows from the formula

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

Keeping this picture in mind let us revisit the algorithm
Consider some points (vectors) which lie in the positive half space of this line (i.e., $w^T x \geq 0$).

What will be the angle between any such vector and $w$? Obviously, less than 90°.

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

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°

Of course, this also follows from the formula

\[
\cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||}
\]
Consider some points (vectors) which lie in the positive half space of this line (i.e., $w^T x \geq 0$).

What will be the angle between any such vector and $w$? Obviously, less than 90°.

What about points (vectors) which lie in the negative half space of this line (i.e., $w^T x < 0$)?

What will be the angle between any such vector and $w$? 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

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( \mathbf{w} \) randomly;

\[ \text{while } \neg \text{convergence do} \]
  \[ \text{Pick random } \mathbf{x} \in P \cup N; \]
  \[ \quad \text{if } \mathbf{x} \in P \quad \text{and} \quad \mathbf{w} \cdot \mathbf{x} < 0 \text{ then} \]
  \[ \quad \quad \mathbf{w} = \mathbf{w} + \mathbf{x}; \]
  \[ \quad \text{end} \]
  \[ \quad \text{if } \mathbf{x} \in N \quad \text{and} \quad \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ then} \]
  \[ \quad \quad \mathbf{w} = \mathbf{w} - \mathbf{x}; \]
  \[ \quad \text{end} \]
\[ \text{end} \]

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||} \]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( w \) randomly;

\[ \text{while } \neg \text{convergence do} \]
| \[ \text{Pick random } x \in P \cup N; \] |
| \[ \text{if } x \in P \text{ and } w \cdot x < 0 \text{ then} \] |
| \[ \quad \quad w = w + x; \] |
| \[ \text{end} \] |
| \[ \text{if } x \in N \text{ and } w \cdot x \geq 0 \text{ then} \] |
| \[ \quad \quad w = w - x; \] |
| \[ \text{end} \] |

\[ \text{end} \]

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

\[ \cos \alpha = \frac{w^T x}{||w|| \cdot ||x||} \]

For \( x \in P \) if \( w \cdot x < 0 \) then it means that the angle (\( \alpha \)) between this \( x \) and the current \( w \) is greater than 90°
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]
Initialize \( w \) randomly;

\[ \text{while } \neg \text{convergence do} \]
\[ \quad \text{Pick random } x \in P \cup N; \]
\[ \quad \text{if } x \in P \text{ and } w \cdot x < 0 \text{ then} \]
\[ \quad \quad w = w + x; \]
\[ \quad \text{end} \]
\[ \quad \text{if } x \in N \text{ and } w \cdot x \geq 0 \text{ then} \]
\[ \quad \quad w = w - x; \]
\[ \quad \text{end} \]
\[ \text{end} \]

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

\[ \cos \alpha = \frac{w^T x}{||w|| ||x||} \]
Algorithm: Perceptron Learning Algorithm

$P \leftarrow \text{inputs with label 1}$;
$N \leftarrow \text{inputs with label 0}$;
Initialize $w$ randomly;

while !convergence do

Pick random $x \in P \cup N$;
if $x \in P$ and $w \cdot x < 0$ then
    $w = w + x$;
end
if $x \in N$ and $w \cdot x \geq 0$ then
    $w = w - x$;
end
end

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

For $x \in P$ if $w \cdot x < 0$ then it means that the angle ($\alpha$) between this $x$ and the current $w$ is greater than 90° (but we want $\alpha$ to be less than 90°)

What happens to the new angle ($\alpha_{\text{new}}$) when $w_{\text{new}} = w + x$

$$
\cos \alpha = \frac{w^T x}{||w|| ||x||}
$$
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( \mathbf{w} \) randomly;

\textbf{while} \! \text{convergence} \textbf{do}

\hspace{1em} \text{Pick random } \mathbf{x} \in P \cup N ;

\hspace{2em} \text{if } \mathbf{x} \in P \text{ and } \mathbf{w} \cdot \mathbf{x} < 0 \text{ then}

\hspace{3em} \mathbf{w} = \mathbf{w} + \mathbf{x} ;

\hspace{2em} \text{end}

\hspace{2em} \text{if } \mathbf{x} \in N \text{ and } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ then}

\hspace{3em} \mathbf{w} = \mathbf{w} - \mathbf{x} ;

\hspace{2em} \text{end}

\textbf{end}

\begin{align*}
\text{\texttt{//the algorithm converges when all the inputs} } & \\
\text{\texttt{are classified correctly}} & \\
\end{align*}

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||} \]

\begin{itemize}
\item For \( \mathbf{x} \in P \) if \( \mathbf{w} \cdot \mathbf{x} < 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is greater than 90° (but we want \( \alpha \) to be less than 90°)
\item What happens to the new angle (\( \alpha_{\text{new}} \)) when \( \mathbf{w}_{\text{new}} = \mathbf{w} + \mathbf{x} \)
\begin{align*}
\cos(\alpha_{\text{new}}) & \propto \mathbf{w}_{\text{new}}^T \mathbf{x} \\
& \propto (\mathbf{w} + \mathbf{x})^T \mathbf{x} \\
& \propto \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} \\
& > \cos \alpha
\end{align*}
\item Thus \( \alpha_{\text{new}} \) will be less than \( \alpha \) and this is exactly what we want.
\end{itemize}
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]

Initialize \( \mathbf{w} \) randomly;

\[ \text{while !convergence do} \]
\[ \text{Pick random } \mathbf{x} \in P \cup N; \]
\[ \text{if } \mathbf{x} \in P \text{ and } \mathbf{w} \cdot \mathbf{x} < 0 \text{ then} \]
\[ \mathbf{w} = \mathbf{w} + \mathbf{x}; \]
\[ \text{end} \]
\[ \text{if } \mathbf{x} \in N \text{ and } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ then} \]
\[ \mathbf{w} = \mathbf{w} - \mathbf{x}; \]
\[ \text{end} \]
\[ \text{end} \]

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||} \]

- For \( \mathbf{x} \in P \) if \( \mathbf{w} \cdot \mathbf{x} < 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is greater than 90° (but we want \( \alpha \) to be less than 90°)

- What happens to the new angle (\( \alpha_{new} \)) 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

\( P \leftarrow \text{inputs with label 1}; \)
\( N \leftarrow \text{inputs with label 0}; \)
Initialize \( w \) randomly;

while !convergence do

Pick random \( x \in P \cup N \);
if \( x \in P \) and \( w \cdot x < 0 \) then
    \( w = w + x; \)
end
if \( x \in N \) and \( w \cdot x \geq 0 \) then
    \( w = w - x; \)
end
end

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

For \( x \in P \) if \( w \cdot x < 0 \) then it means that the angle (\( \alpha \)) between this \( x \) and the current \( w \) is greater than 90° (but we want \( \alpha \) to be less than 90°)

What happens to the new angle (\( \alpha_{new} \)) when \( w_{new} = w + x \)

\[
\cos(\alpha_{new}) \propto w_{new}^T x \\
\propto (w + x)^T x \\
\propto w^T x + x^T x
\]

\[
\cos \alpha = \frac{w^T x}{||w|| ||x||}
\]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( w \) randomly;

\[ \text{while } \neg \text{convergence do} \]
\[ \quad \text{Pick random } x \in P \cup N; \]
\[ \quad \text{if } x \in P \text{ and } w \cdot x < 0 \text{ then} \]
\[ \quad \quad w = w + x; \]
\[ \quad \text{end} \]
\[ \quad \text{if } x \in N \text{ and } w \cdot x \geq 0 \text{ then} \]
\[ \quad \quad w = w - x; \]
\[ \quad \text{end} \]
\[ \text{end} \]

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

\[ \cos \alpha = \frac{w^T x}{\|w\| \|x\|} \]

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

\[ \cos(\alpha_{\text{new}}) \propto w_{\text{new}}^T x \]
\[ \propto (w + x)^T x \]
\[ \propto w^T x + x^T x \]
\[ \propto \cos \alpha + x^T x \]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]
Initialize \( w \) randomly;

\textbf{while} !convergence \textbf{do}

\hspace{1em} Pick random \( x \in P \cup N \);

\hspace{2em} if \( x \in P \) and \( w \cdot x < 0 \) then

\hspace{3em} \( w = w + x \);  

\hspace{2em} end

\hspace{2em} if \( x \in N \) and \( w \cdot x \geq 0 \) then

\hspace{3em} \( w = w - x \);  

\hspace{2em} end

\textbf{end}

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

\[ \cos \alpha = \frac{w^T x}{||w|| \cdot ||x||} \]

\[ \text{For } x \in P \text{ if } w \cdot x < 0 \text{ then it means that the angle (\( \alpha \)) between this } x \text{ and the current } w \text{ is greater than } 90^\circ \text{ (but we want } \alpha \text{ to be less than } 90^\circ) \]

\[ \text{What happens to the new angle (} \alpha_{\text{new}} \text{) when } w_{\text{new}} = w + x \]

\[ \cos(\alpha_{\text{new}}) \propto w_{\text{new}}^T x \]

\[ \propto (w + x)^T x \]

\[ \propto w^T x + x^T x \]

\[ \propto \cos \alpha + x^T x \]

\[ \cos(\alpha_{\text{new}}) > \cos \alpha \]
**Algorithm:** Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( \mathbf{w} \) randomly;

**while** !convergence **do**

\[ \text{Pick random } \mathbf{x} \in P \cup N; \]
\[ \text{if } \mathbf{x} \in P \text{ and } \mathbf{w} \cdot \mathbf{x} < 0 \text{ then} \]
\[ \quad \mathbf{w} = \mathbf{w} + \mathbf{x}; \]
\[ \text{end} \]
\[ \text{if } \mathbf{x} \in N \text{ and } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ then} \]
\[ \quad \mathbf{w} = \mathbf{w} - \mathbf{x}; \]
\[ \text{end} \]

**end**

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

\[
cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| \cdot ||\mathbf{x}||} \]

- For \( \mathbf{x} \in P \) if \( \mathbf{w} \cdot \mathbf{x} < 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is greater than 90° (but we want \( \alpha \) to be less than 90°)

- What happens to the new angle (\( \alpha_{\text{new}} \)) when \( \mathbf{w}_{\text{new}} = \mathbf{w} + \mathbf{x} \)

\[
cos(\alpha_{\text{new}}) \propto \mathbf{w}_{\text{new}}^T \mathbf{x} \\
\propto (\mathbf{w} + \mathbf{x})^T \mathbf{x} \\
\propto \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} \\
\propto \cos \alpha + \mathbf{x}^T \mathbf{x} \]

\[
cos(\alpha_{\text{new}}) > \cos \alpha \]

Thus \( \alpha_{\text{new}} \) will be less than \( \alpha \) and this is exactly what we want
**Algorithm**: Perceptron Learning Algorithm

\( P \leftarrow \text{inputs with label 1;} \)

\( N \leftarrow \text{inputs with label 0;} \)

Initialize \( \mathbf{w} \) randomly;

**while** !convergence **do**

- Pick random \( \mathbf{x} \in P \cup N \);
  - if \( \mathbf{x} \in P \) and \( \mathbf{w} \cdot \mathbf{x} < 0 \) then
    - \( \mathbf{w} = \mathbf{w} + \mathbf{x} \);
  - end
  - if \( \mathbf{x} \in N \) and \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) then
    - \( \mathbf{w} = \mathbf{w} - \mathbf{x} \);
  - end

**end**

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

\[
\cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||}
\]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( w \) randomly;

while !convergence do

Pick random \( x \in P \cup N \);

if \( x \in P \) and \( w \cdot x < 0 \) then

\[ w = w + x; \]

end

if \( x \in N \) and \( w \cdot x \geq 0 \) then

\[ w = w - x; \]

end

end

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

\[
\cos \alpha = \frac{w^T x}{||w|| ||x||}
\]
**Algorithm: Perceptron Learning Algorithm**

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]
Initialize \( \mathbf{w} \) randomly;

while !convergence do

| Pick random \( \mathbf{x} \in P \cup N \); |
| if \( \mathbf{x} \in P \) and \( \mathbf{w} \cdot \mathbf{x} < 0 \) then |
| \( \mathbf{w} = \mathbf{w} + \mathbf{x} \); |
| end |
| if \( \mathbf{x} \in N \) and \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) then |
| \( \mathbf{w} = \mathbf{w} - \mathbf{x} \); |
| end |

end

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| \cdot ||\mathbf{x}||} \]

For \( \mathbf{x} \in N \) if \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is less than 90° (but we want \( \alpha \) to be greater than 90°)
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( \mathbf{w} \) randomly;

while \( \neg \text{convergence} \) do

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

if \( \mathbf{x} \in P \text{ and } \mathbf{w} \cdot \mathbf{x} < 0 \) then

\[ \mathbf{w} = \mathbf{w} + \mathbf{x}; \]

end

if \( \mathbf{x} \in N \text{ and } \mathbf{w} \cdot \mathbf{x} \geq 0 \) then

\[ \mathbf{w} = \mathbf{w} - \mathbf{x}; \]

end

end

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| \cdot ||\mathbf{x}||}; \]

For \( \mathbf{x} \in N \) if \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is less than 90° (but we want \( \alpha \) to be greater than 90°)

What happens to the new angle (\( \alpha_{\text{new}} \)) when \( \mathbf{w}_{\text{new}} = \mathbf{w} - \mathbf{x} \)
Algorithm: Perceptron Learning Algorithm

$P \leftarrow \text{inputs with label 1}$;
$N \leftarrow \text{inputs with label 0}$;
Initialize $w$ randomly;

while !convergence do

    Pick random $x \in P \cup N$;
    if $x \in P$ and $w \cdot x < 0$ then
        $w = w + x$;
    end
    if $x \in N$ and $w \cdot x \geq 0$ then
        $w = w - x$;
    end

end

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

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

For $x \in N$ if $w \cdot x \geq 0$ then it means that the angle ($\alpha$) between this $x$ and the current $w$ is less than $90^\circ$ (but we want $\alpha$ to be greater than $90^\circ$)

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

$cos(\alpha_{\text{new}}) \propto w_{\text{new}}^T x$
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( \mathbf{w} \) randomly;

\textbf{while} \! \text{convergence} \textbf{do}

\hspace{1em} \text{Pick random } \mathbf{x} \in P \cup N ;

\hspace{2em} \text{if } \mathbf{x} \in P \text{ and } \mathbf{w} \cdot \mathbf{x} < 0 \text{ then}

\hspace{3em} \mathbf{w} = \mathbf{w} + \mathbf{x} ;

\hspace{2em} \text{end}

\hspace{2em} \text{if } \mathbf{x} \in N \text{ and } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ then}

\hspace{3em} \mathbf{w} = \mathbf{w} - \mathbf{x} ;

\hspace{2em} \text{end}

\textbf{end}

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| \cdot ||\mathbf{x}||} \]

- For \( \mathbf{x} \in N \) if \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is less than 90° (but we want \( \alpha \) to be greater than 90°)

- What happens to the new angle (\( \alpha_{\text{new}} \)) when \( \mathbf{w}_{\text{new}} = \mathbf{w} - \mathbf{x} \)

\[ \cos(\alpha_{\text{new}}) \propto \mathbf{w}_{\text{new}}^T \mathbf{x} \]
\[ \propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( \mathbf{w} \) randomly;

while !convergence do
  Pick random \( x \in P \cup N \);
  if \( x \in P \) and \( \mathbf{w} \cdot x < 0 \) then
    \( \mathbf{w} = \mathbf{w} + x; \)
  end
  if \( x \in N \) and \( \mathbf{w} \cdot x \geq 0 \) then
    \( \mathbf{w} = \mathbf{w} - x; \)
  end
end

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

For \( x \in N \) if \( \mathbf{w} \cdot x \geq 0 \) then it means that the angle (\( \alpha \)) between this \( x \) and the current \( \mathbf{w} \) is less than 90° (but we want \( \alpha \) to be greater than 90°)

What happens to the new angle (\( \alpha_{\text{new}} \)) when \( \mathbf{w}_{\text{new}} = \mathbf{w} - x \)

\[
\cos(\alpha_{\text{new}}) \propto \mathbf{w}_{\text{new}}^T x \\
\propto (\mathbf{w} - x)^T x \\
\propto \mathbf{w}^T x - x^T x
\]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]

Initialize \( \mathbf{w} \) randomly;

\begin{algorithmic}
\While{!convergence}
\State Pick random \( \mathbf{x} \in P \cup N \);
\If{\( \mathbf{x} \in P \) and \( \mathbf{w} \cdot \mathbf{x} < 0 \)}
\State \( \mathbf{w} = \mathbf{w} + \mathbf{x} \);
\EndIf
\If{\( \mathbf{x} \in N \) and \( \mathbf{w} \cdot \mathbf{x} \geq 0 \)}
\State \( \mathbf{w} = \mathbf{w} - \mathbf{x} \);
\EndIf
\EndWhile
\Endalgorithmic

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||} \]

- For \( \mathbf{x} \in N \) if \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) then it means that the angle (\( \alpha \)) between this \( \mathbf{x} \) and the current \( \mathbf{w} \) is less than 90° (but we want \( \alpha \) to be greater than 90°)

- What happens to the new angle (\( \alpha_{\text{new}} \)) when \( \mathbf{w}_{\text{new}} = \mathbf{w} - \mathbf{x} \)

\[ \cos(\alpha_{\text{new}}) \propto \mathbf{w}_{\text{new}}^T \mathbf{x} \]
\[ \propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \]
\[ \propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \]
\[ \propto \cos \alpha - \mathbf{x}^T \mathbf{x} \]
**Algorithm: Perceptron Learning Algorithm**

$P \leftarrow \text{inputs with label } 1$;

$N \leftarrow \text{inputs with label } 0$;

Initialize $\mathbf{w}$ randomly;

**while** !convergence **do**

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

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

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

**end**

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

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

**end**

**end**

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

\[ \cos \alpha = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}|| ||\mathbf{x}||} \]

- For $\mathbf{x} \in N$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ then it means that the angle ($\alpha$) between this $\mathbf{x}$ and the current $\mathbf{w}$ is less than 90° (but we want $\alpha$ to be greater than 90°)

- What happens to the new angle ($\alpha_{\text{new}}$) when $\mathbf{w}_{\text{new}} = \mathbf{w} - \mathbf{x}$

\[ \cos(\alpha_{\text{new}}) \propto \mathbf{w}_{\text{new}}^T \mathbf{x} \]
\[ \propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \]
\[ \propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \]
\[ \propto \cos \alpha - \mathbf{x}^T \mathbf{x} \]

\[ \cos(\alpha_{\text{new}}) < \cos \alpha \]
Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label } 1; \]
\[ N \leftarrow \text{inputs with label } 0; \]
Initialize \( w \) randomly;

\[ \textbf{while} \ \neg \text{convergence} \ \textbf{do} \]
\[ \quad \text{Pick random } x \in P \cup N; \]
\[ \quad \text{if } x \in P \text{ and } w \cdot x < 0 \text{ then} \]
\[ \quad \quad w = w + x; \]
\[ \quad \text{end} \]
\[ \quad \text{if } x \in N \text{ and } w \cdot x \geq 0 \text{ then} \]
\[ \quad \quad w = w - x; \]
\[ \quad \text{end} \]
\[ \textbf{end} \]

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

\[
\cos \alpha = \frac{w^T x}{\|w\| \|x\|}
\]

For \( x \in N \) if \( w \cdot x \geq 0 \) then it means that the angle \((\alpha)\) between this \( x \)
and the current \( w \) is less than 90° (but we want \( \alpha \) to be greater than
90°)

What happens to the new angle \((\alpha_{\text{new}})\) when \( w_{\text{new}} = w - x \)

\[
\cos(\alpha_{\text{new}}) \propto w_{\text{new}}^T x \\
\propto (w - x)^T x \\
\propto w^T x - x^T x \\
\propto \cos \alpha - x^T x \\
\]

\[
\cos(\alpha_{\text{new}}) < \cos \alpha
\]

Thus \( \alpha_{\text{new}} \) will be greater than \( \alpha \) and this is exactly what we want
We will now see this algorithm in action for a toy dataset
- We initialized $\mathbf{w}$ to a random value.

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

- We now run the algorithm by randomly going over the points.

- The algorithm has converged.
- We initialized $\mathbf{w}$ to a random value
- We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\therefore$ angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\therefore$ angle $< 90^\circ$) for all the negative points (the situation is exactly opposite of what we actually want it to be)
• We initialized \( w \) to a random value
• We observe that currently, \( w \cdot x < 0 \) (\( \because \) angle > 90°) for all the positive points and \( w \cdot x \geq 0 \) (\( \because \) angle < 90°) for all the negative points (the situation is exactly opposite 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 > $90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle < $90^\circ$) for all the negative points (the situation is exactly opposite 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 $\mathbf{w} = \mathbf{w} + \mathbf{x} \because \mathbf{w} \cdot \mathbf{x} < 0$ (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 opposite 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 \( \mathbf{w} = \mathbf{w} + \mathbf{x} \) \( \therefore \) \( \mathbf{w} \cdot \mathbf{x} < 0 \) (you can check the angle visually).
• We initialized $\mathbf{w}$ to a random value

• We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0 (\because \text{angle } > 90^\circ)$ for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0 (\because \text{angle } < 90^\circ)$ for all the negative points (the situation is exactly opposite 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 $\mathbf{w} = \mathbf{w} + \mathbf{x} \therefore \mathbf{w} \cdot \mathbf{x} < 0$ (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 opposite 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 \( \mathbf{w} = \mathbf{w} + \mathbf{x} \) \( \because \) \( \mathbf{w} \cdot \mathbf{x} < 0 \) (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 opposite 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 \( \mathbf{w} = \mathbf{w} - \mathbf{x} \) : \( \because \mathbf{w} \cdot \mathbf{x} \geq 0 \) (you can check the angle visually).
• We initialized \( \mathbf{w} \) to a random value
• We observe that currently, \( \mathbf{w} \cdot \mathbf{x} < 0 (\because \text{angle} > 90^\circ) \) for all the positive points and \( \mathbf{w} \cdot \mathbf{x} \geq 0 (\because \text{angle} < 90^\circ) \) for all the negative points (the situation is exactly opposite 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 \( \mathbf{w} = \mathbf{w} - \mathbf{x} \) \( \because \mathbf{w} \cdot \mathbf{x} \geq 0 \) (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 $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $< 90^\circ$) for all the negative points (the situation is exactly opposite 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 $\because \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)
• We initialized $\mathbf{w}$ to a random value

• We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0 (\because \text{angle } > 90^\circ)$ for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0 (\because \text{angle } < 90^\circ)$ for all the negative points (the situation is exactly opposite 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 $\because \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)
• We initialized \( w \) to a random value

• We observe that currently, \( w \cdot x < 0 \) (\( \because \) angle > 90°) for all the positive points and \( w \cdot x \geq 0 \) (\( \because \) angle < 90°) for all the negative points (the situation is exactly opposite 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 \( \because \) \( w \cdot x < 0 \) (you can check the angle visually)
- We initialized \( w \) to a random value
- We observe that currently, \( w \cdot x < 0 \) (\( \because \) angle > 90°) for all the positive points and \( w \cdot x \geq 0 \) (\( \because \) angle < 90°) for all the negative points (the situation is exactly opposite 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 \( \therefore w \cdot x < 0 \) (you can check the angle visually)
- We initialized $w$ to a random value
- We observe that currently, $w \cdot x < 0 \ (\because \text{angle} > 90^\circ)$ for all the positive points and $w \cdot x \geq 0 \ (\because \text{angle} < 90^\circ)$ for all the negative points (the situation is exactly opposite 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 $w = w + x \because w \cdot x < 0$ (you can check the angle visually)
• We initialized $w$ to a random value
• We observe that currently, $w \cdot x < 0 \ (\because \ \text{angle} > 90^\circ)$ for all the positive points and $w \cdot x \geq 0 \ (\because \ \text{angle} < 90^\circ)$ for all the negative points (the situation is exactly opposite 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 $w = w + x \ \therefore \ w \cdot x < 0$ (you can check the angle visually)
• We initialized $\mathbf{w}$ to a random value
• We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0 \ (\because \text{angle} > 90^\circ)$ for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0 \ (\because \text{angle} < 90^\circ)$ for all the negative points (the situation is exactly opposite 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 $\because \mathbf{w} \cdot \mathbf{x} \geq 0$ (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 opposite 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 \( \because \) \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) (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 $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\(\because\) angle $< 90^\circ$) for all the negative points (the situation is exactly opposite 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 $\therefore \mathbf{w} \cdot \mathbf{x} \geq 0$ (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 $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite 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 $\because \mathbf{w} \cdot \mathbf{x} \geq 0$ (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 opposite 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 \( \because \) \( \mathbf{w} \cdot \mathbf{x} < 0 \) (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 $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ (\because angle $< 90^\circ$) for all the negative points (the situation is exactly opposite 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 $\therefore \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually).
• We initialized \( \mathbf{w} \) to a random value
• We observe that currently, \( \mathbf{w} \cdot \mathbf{x} < 0 \) (\( \therefore \) angle > 90°) for all the positive points and \( \mathbf{w} \cdot \mathbf{x} \geq 0 \) (\( \therefore \) angle < 90°) for all the negative points (the situation is exactly opposite 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 \( \therefore \) \( \mathbf{w} \cdot \mathbf{x} < 0 \) (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 opposite 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 \( \because \) \( \mathbf{w} \cdot \mathbf{x} < 0 \) (you can check the angle visually)
We initialized $\mathbf{w}$ to a random value

We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0$ ($\therefore$ angle $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\therefore$ angle $< 90^\circ$) for all the negative points (the situation is exactly opposite 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 $\therefore \mathbf{w} \cdot \mathbf{x} < 0$ (you can check the angle visually)
• We initialized \( w \) to a random value

• We observe that currently, \( w \cdot x < 0 \) (\( \therefore \) angle > 90°) for all the positive points and \( w \cdot x \geq 0 \) (\( \therefore \) angle < 90°) for all the negative points (the situation is exactly opposite 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 \( \therefore w \cdot x < 0 \) (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 $> 90^\circ$) for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0$ ($\because$ angle $< 90^\circ$) for all the negative points (the situation is exactly opposite 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 $\because \mathbf{w} \cdot \mathbf{x} \geq 0$ (you can check the angle visually)
• We initialized $\mathbf{w}$ to a random value
• We observe that currently, $\mathbf{w} \cdot \mathbf{x} < 0 \ (\because \ \text{angle} > 90^\circ)$ for all the positive points and $\mathbf{w} \cdot \mathbf{x} \geq 0 \ (\because \ \text{angle} < 90^\circ)$ for all the negative points (the situation is exactly opposite of what we actually want it to be)
• We now run the algorithm by randomly going over the points
• The algorithm has converged
Module 2.6: Proof of Convergence
Now that we have some faith and intuition about why the algorithm works, we will see a more formal proof of convergence...
**Theorem**

**Definition:** Two sets $P$ and $N$ of points in an $n$-dimensional space are called absolutely linearly separable if

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

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

Proposition: If the sets $P$ and $N$ are finite and linearly separable, the perceptron learning algorithm updates the weight vector $w_t$ a finite number of times. In other words: if the vectors in $P$ and $N$ are tested cyclically one after the other, a weight vector $w_t$ is found after a finite number of steps $t$ which can separate the two sets.

Proof: On the next slide
Theorem

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

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

**Proposition:**
**Theorem**

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

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

Proposition: If the sets $P$ and $N$ are finite and linearly separable, the perceptron learning algorithm updates the weight vector $\mathbf{w}_t$ a finite number of times.
**Theorem**

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

**Proposition:** If the sets $P$ and $N$ are finite and linearly separable, the perceptron learning algorithm updates the weight vector $w_t$ a finite number of times. In other words: if the vectors in $P$ and $N$ are tested cyclically one after the other,
**Theorem**

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

**Proposition:** If the sets $P$ and $N$ are finite and linearly separable, the perceptron learning algorithm updates the weight vector $w_t$ a finite number of times. In other words: if the vectors in $P$ and $N$ are tested cyclically one after the other, a weight vector $w_t$ is found after a finite number of steps $t$ which can separate the two sets.
Definition: Two sets $P$ and $N$ of points in an $n$-dimensional space are called absolutely linearly separable if $n + 1$ real numbers $w_0, w_1, ..., w_n$ exist such that every point $(x_1, x_2, ..., x_n) \in P$ satisfies $\sum_{i=1}^{n} w_i \ast x_i > w_0$ and every point $(x_1, x_2, ..., x_n) \in N$ satisfies $\sum_{i=1}^{n} w_i \ast x_i < w_0$.

Proposition: If the sets $P$ and $N$ are finite and linearly separable, the perceptron learning algorithm updates the weight vector $w_t$ a finite number of times. In other words: if the vectors in $P$ and $N$ are tested cyclically one after the other, a weight vector $w_t$ is found after a finite number of steps $t$ which can separate the two sets.

Proof: On the next slide
Setup:

- If $x \in N$ then $-x \in P$ (\(\therefore\))

\[
w^T x < 0 \implies w^T (-x) \geq 0
\]
Setup:

- If \( x \in N \) then \(-x \in P \): \( w^T x < 0 \implies w^T (-x) \geq 0 \)

- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)
Setup:

- If \( x \in N \) then \(-x \in P \) (\( w^T x < 0 \implies w^T (-x) \geq 0 \))
- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)

Algorithm: Perceptron Learning Algorithm

\[
P \leftarrow \text{inputs with label } 1;
\]

\[
P' \leftarrow \text{inputs with label } 0;
\]

\[
P' \leftarrow \text{inputs with label } 0;
\]
Setup:

- If $x \in N$ then $-x \in P$ ($\therefore w^T x < 0 \implies w^T (-x) \geq 0$)
- We can thus consider a single set $P' = P \cup N^-$ and for every element $p \in P'$ ensure that $w^T p \geq 0$

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1;
$N \leftarrow$ inputs with label 0;
Setup:
- If $x \in N$ then $-x \in P$ ($\because w^T x < 0 \implies w^T (-x) \geq 0$)
- We can thus consider a single set $P' = P \cup N^-$ and for every element $p \in P'$ ensure that $w^T p \geq 0$

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1;
$N \leftarrow$ inputs with label 0;
$N^-$ contains negations of all points in $N$;

Initialize $w$ randomly;
while !convergence do
  Pick random $p \in P'$;
  $p \leftarrow \frac{p}{||p||}$ (so now, $||p|| = 1$);
  if $w \cdot p < 0$ then
    $w = w + p$;
  end
end

//the algorithm converges when all the inputs are classified correctly
//notice that we do not need the other if condition because by construction we want all points in $P'$ to lie in the positive half space $w \cdot p \geq 0$
Setup:

- If $x \in N$ then $-x \in P$ ($\therefore w^T x < 0 \implies w^T (-x) \geq 0$)

- We can thus consider a single set $P' = P \cup N^-$ and for every element $p \in P'$ ensure that $w^T p \geq 0$

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1;
$N \leftarrow$ inputs with label 0;
$N^-$ contains negations of all points in $N$;
$P' \leftarrow P \cup N^-$;
Setup:

- If \( x \in N \) then \(-x \in P \) (\( \because w^T x < 0 \implies w^T(-x) \geq 0 \))
- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)

Algorithm: Perceptron Learning Algorithm

- \( P \leftarrow \) inputs with label 1;
- \( N \leftarrow \) inputs with label 0;
- \( N^- \) contains negations of all points in \( N \);
- \( P' \leftarrow P \cup N^- \);
- Initialize \( w \) randomly;
Setup:

- If \( x \in N \) then \(-x \in P \) (\( \because w^T x < 0 \implies w^T (-x) \geq 0 \))
- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)

Algorithm: Perceptron Learning Algorithm

\[
P \leftarrow \text{inputs with label 1};
N \leftarrow \text{inputs with label 0};
N^- \text{ contains negations of all points in } N;
P' \leftarrow P \cup N^-;
\]

Initialize \( w \) randomly;

\[\text{while } \neg \text{convergence do}\]

\[\text{end}\]

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

- If \( x \in N \) then \(-x \in P \) (\( \because w^T x < 0 \implies w^T(-x) \geq 0 \))
- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)

Algorithm: Perceptron Learning Algorithm

\( P \leftarrow \text{inputs with label 1}; \)
\( N \leftarrow \text{inputs with label 0}; \)
\( N^- \) contains negations of all points in \( N; \)
\( P' \leftarrow P \cup N^-; \)
Initialize \( w \) randomly;
while !convergence do
  Pick random \( p \in P' \);
end

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

- If $x \in N$ then $-x \in P$ ($\because w^T x < 0 \implies w^T (-x) \geq 0$)
- We can thus consider a single set $P' = P \cup N^-$ and for every element $p \in P'$ ensure that $w^T p \geq 0$

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ inputs with label 1;
$N \leftarrow$ inputs with label 0;
$N^-$ contains negations of all points in $N$;
$P' \leftarrow P \cup N^-$;
Initialize $w$ randomly;

while !convergence do
    Pick random $p \in P'$;
    if $w.p < 0$ then
        $w = w + p$;
    end
end

//the algorithm converges when all the inputs are classified correctly
Setup:
- If \( x \in N \) then \(-x \in P \) (\( w^T x < 0 \implies w^T (-x) \geq 0 \))
- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)

Algorithm: Perceptron Learning Algorithm

\[
P \leftarrow \text{inputs with label } 1; \\
N \leftarrow \text{inputs with label } 0; \\
N^- \text{ contains negations of all points in } N; \\
P' \leftarrow P \cup N^-;
\]

Initialize \( w \) randomly;

\[
\text{while } !\text{convergence do} \\
\quad \text{Pick random } p \in P' \\
\quad \text{if } w \cdot p < 0 \text{ then} \\
\quad \quad w = w + p ;
\]

end

//the algorithm converges when all the inputs are classified correctly
//notice that we do not need the other if condition because by construction we want all points in \( P' \) to lie...
Setup:

- If \( x \in N \) then \(-x \in P \) (\( \because \) \( w^T x < 0 \implies w^T (-x) \geq 0 \))

- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)

- Further we will normalize all the \( p \)'s so that \( ||p|| = 1 \) (notice that this does not affect the solution: \( \because \) \( if \quad w^T \frac{p}{||p||} \geq 0 \) then \( w^T p \geq 0 \))

Algorithm: Perceptron Learning Algorithm

\[ P \leftarrow \text{inputs with label 1}; \]
\[ N \leftarrow \text{inputs with label 0}; \]
\[ N^- \text{contains negations of all points in } N; \]
\[ P' \leftarrow P \cup N^-; \]

Initialize \( w \) randomly;

\[ \text{while } !\text{convergence} \quad \text{do} \]

\[ \quad \text{Pick random } p \in P'; \]

\[ \quad \text{if } w. p < 0 \text{ then} \]

\[ \quad \quad w = w + p; \]

\[ \text{end} \]

\[ \text{//the algorithm converges when all the inputs are classified correctly} \]

\[ \text{//notice that we do not need the other if condition because by construction we want all points in } P' \text{ to lie} \]
Setup:

- If $x \in N$ then $-x \in P$ (∵ $w^T x < 0 \implies w^T (-x) \geq 0$)

- We can thus consider a single set $P' = P \cup N^-$ and for every element $p \in P'$ ensure that $w^T p \geq 0$

- Further we will normalize all the $p$’s so that $||p|| = 1$ (notice that this does not affect the solution ∴ if $w^T p \geq 0$ then $w^T p \geq 0$)

**Algorithm: Perceptron Learning Algorithm**

$P \leftarrow$ inputs with label 1;

$N \leftarrow$ inputs with label 0;

$N^-$ contains negations of all points in $N$;

$P' \leftarrow P \cup N^-$;

Initialize $w$ randomly;

while !convergence do

- Pick random $p \in P'$;
- $p \leftarrow \frac{p}{||p||}$ (so now,$||p|| = 1$);
- if $w \cdot p < 0$ then
  - $w = w + p$;

end

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

// notice that we do not need the other if condition because by construction we want all points in $P'$ to lie
Setup:
- If \( x \in N \) then \(-x \in P \) (\( \because \) \( w^T x < 0 \) \( \implies \) \( w^T (-x) \geq 0 \))
- We can thus consider a single set \( P' = P \cup N^- \) and for every element \( p \in P' \) ensure that \( w^T p \geq 0 \)
- Further we will normalize all the \( p \)'s so that \( ||p|| = 1 \) (notice that this does not affect the solution \( \because \) if \( w^T p \frac{p}{||p||} \geq 0 \) then \( w^T p \geq 0 \))
- Let \( w^* \) be the normalized solution vector (we know one exists as the data is linearly separable)

<table>
<thead>
<tr>
<th>Algorithm: Perceptron Learning Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P \leftarrow \text{inputs with label 1;} )</td>
</tr>
<tr>
<td>( N \leftarrow \text{inputs with label 0;} )</td>
</tr>
<tr>
<td>( N^- ) contains negations of all points in ( N );</td>
</tr>
<tr>
<td>( P' \leftarrow P \cup N^- ; )</td>
</tr>
<tr>
<td>Initialize ( w ) randomly;</td>
</tr>
<tr>
<td>\underline{while} !convergence \underline{do}</td>
</tr>
<tr>
<td>\quad Pick random ( p \in P' );</td>
</tr>
<tr>
<td>\quad ( p \leftarrow \frac{p}{</td>
</tr>
<tr>
<td>\quad if ( w.p &lt; 0 ) then</td>
</tr>
<tr>
<td>\qquad ( w = w + p );</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>//the algorithm converges when all the inputs are classified correctly</td>
</tr>
</tbody>
</table>
| //notice that we do not need the other \textbf{if} condition because by construction we want all points in \( P' \) to lie
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$.

We make a correction $w_{t+1} = w_t + p_i$.

Let $\beta$ be the angle between $w^*$ and $w_{t+1}$.

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{|w_{t+1}|}.
\]

Numerator $= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$.

\[
\geq w^* \cdot w_t + \delta \quad (\delta = \min \{w^* \cdot p_i \mid \forall i\}).
\]

\[
\geq w^* \cdot (w_t - 1 + p_j) + \delta \geq w^* \cdot (w_0 + 2\delta).
\]

(By induction)
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step
- So at time-step $t$ we would have made only $k (\leq t)$ corrections
- Every time we make a correction a quantity $\delta$ gets added to the numerator
- So by time-step $t$, a quantity $k \delta$ gets added to the numerator

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

$$
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
$$
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
\]

\[
\text{Numerator} = w^* \cdot w_{t+1}
\]
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

$$
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
$$

Numerator $= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

$$\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}$$

Numerator

$$w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$$

$$= w^* \cdot w_t + w^* \cdot p_i$$
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
\]

Numerator $= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$

$= w^* \cdot w_t + w^* \cdot p_i$

$\geq w^* \cdot w_t + \delta \quad (\delta = \min \{w^* \cdot p_i | \forall i\}$
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
\]

Numerator

\[
= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)
\]

\[
= w^* \cdot w_t + w^* \cdot p_i
\]

\[
\geq w^* \cdot w_t + \delta \quad (\delta = \min \{w^* \cdot p_i | \forall i\})
\]

\[
\geq w^* \cdot (w_{t-1} + p_j) + \delta
\]
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$
  \[
  \cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
  \]
  \[
  \text{Numerator} = w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)
  = w^* \cdot w_t + w^* \cdot p_i
  \geq w^* \cdot w_t + \delta \quad (\delta = \min\{w^* \cdot p_i | \forall i\})
  \geq w^* \cdot (w_{t-1} + p_j) + \delta
  \geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta
  \]
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

$$
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
$$

\[
\text{Numerator} = w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i) = w^* \cdot w_t + w^* \cdot p_i \\
\geq w^* \cdot w_t + \delta \quad (\delta = \min\{w^* \cdot p_i | \forall i\}) \\
\geq w^* \cdot (w_{t-1} + p_j) + \delta \\
\geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta \\
\geq w^* \cdot w_{t-1} + 2\delta
\]
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
\]

Numerator = $w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$

\[
= w^* \cdot w_t + w^* \cdot p_i
\]

\[
\geq w^* \cdot w_t + \delta \quad (\delta = \min \{w^* \cdot p_i | \forall i\})
\]

\[
\geq w^* \cdot (w_{t-1} + p_j) + \delta
\]

\[
\geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta
\]

\[
\geq w^* \cdot w_{t-1} + 2\delta
\]
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$
  \[
  \cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
  \]
- Numerator $= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$
  \[
  = w^* \cdot w_t + w^* \cdot p_i \geq w^* \cdot w_t + \delta \quad (\delta = \min\{w^* \cdot p_i | \forall i\})
  \]
  \[
  \geq w^* \cdot (w_{t-1} + p_j) + \delta \\
  \geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta \geq w^* \cdot w_{t-1} + 2\delta
  \]
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step
- So at time-step $t$ we would have made only $k \leq t$ corrections

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}
\]

$Numerator = w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$

\[
= w^* \cdot w_t + w^* \cdot p_i
\geq w^* \cdot w_t + \delta \quad (\delta = \min \{w^* \cdot p_i \mid \forall i\})
\geq w^* \cdot (w_{t-1} + p_j) + \delta
\geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta
\geq w^* \cdot w_{t-1} + 2\delta
\]
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step
- So at time-step $t$ we would have made only $k$ ($\leq t$) corrections
- Every time we make a correction a quantity $\delta$ gets added to the numerator

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$
  \[
  \cos \beta = \frac{w^* \cdot w_{t+1}}{|w_{t+1}|}
  \]  
  Numerator $= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$
  \[
  = w^* \cdot w_t + w^* \cdot p_i
  \]  
  \[
  \geq w^* \cdot w_t + \delta \quad (\delta = \min \{ w^* \cdot p_i | \forall i \})
  \]  
  \[
  \geq w^* \cdot (w_{t-1} + p_j) + \delta
  \]  
  \[
  \geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta
  \]  
  \[
  \geq w^* \cdot w_{t-1} + 2\delta
  \]
Observations:

- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step
- So at time-step $t$ we would have made only $k$ ($\leq t$) corrections
- Every time we make a correction a quantity $\delta$ gets added to the numerator
- So by time-step $t$, a quantity $k\delta$ gets added to the numerator

Proof:

- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$

$$\cos\beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||}$$

Numerator $= w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)$

$$= w^* \cdot w_t + w^* \cdot p_i$$

$$\geq w^* \cdot w_t + \delta \quad (\delta = \min\{w^* \cdot p_i | \forall i\})$$

$$\geq w^* \cdot (w_{t-1} + p_j) + \delta$$

$$\geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta$$

$$\geq w^* \cdot w_{t-1} + 2\delta$$
Observations:
- $w^*$ is some optimal solution which exists but we don’t know what it is
- We do not make a correction at every time-step
- We make a correction only if $w^T \cdot p_i \leq 0$ at that time step
- So at time-step $t$ we would have made only $k$ ($\leq t$) corrections
- Every time we make a correction a quantity $\delta$ gets added to the numerator
- So by time-step $t$, a quantity $k\delta$ gets added to the numerator

Proof:
- Now suppose at time step $t$ we inspected the point $p_i$ and found that $w^T \cdot p_i \leq 0$
- We make a correction $w_{t+1} = w_t + p_i$
- Let $\beta$ be the angle between $w^*$ and $w_{t+1}$
  \[
  \cos \beta = \frac{w^* \cdot w_{t+1}}{|w_{t+1}|}
  \]
  \[
  \text{Numerator} = w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i)
  \]
  \[
  = w^* \cdot w_t + w^* \cdot p_i
  \]
  \[
  \geq w^* \cdot w_t + \delta \quad (\delta = \min\{w^* \cdot p_i | \forall i\})
  \]
  \[
  \geq w^* \cdot (w_{t-1} + p_j) + \delta
  \]
  \[
  \geq w^* \cdot w_{t-1} + w^* \cdot p_j + \delta
  \]
  \[
  \geq w^* \cdot w_{t-1} + 2\delta
  \]
  \[
  \geq w^* \cdot w_0 + (k)\delta \quad \text{(By induction)}
  \]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)
Proof (continued:)

So far we have,  \( w^T \cdot p_i \leq 0 \)  (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k\delta \quad \text{(proved by induction)}
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k\delta \quad \text{(proved by induction)}
\]

\[
\text{Denominator}^2 = ||w_{t+1}||^2
\]
Proof (continued:)

So far we have, $w^T \cdot p_i \leq 0$  (and hence we made the correction)

$$
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
$$

Numerator $\geq w^* \cdot w_0 + k\delta$  \text{(proved by induction)}

Denominator $^2 = ||w_{t+1}||^2$

$= \left( w_t + p_i \right) \cdot \left( w_t + p_i \right)$
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|} \quad \text{(by definition)}
\]

Numerator \( \geq w^* \cdot w_0 + k\delta \) (proved by induction)

Denominator \(^2 = \|w_{t+1}\|^2 \)

\[
= (w_t + p_i) \cdot (w_t + p_i)
\]

\[
= \|w_t\|^2 + 2w_t \cdot p_i + \|p_i\|^2
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

Numerator \( \geq w^* \cdot w_0 + k\delta \) \quad \text{(proved by induction)}

Denominator\(^2 = ||w_{t+1}||^2 \]
\[
= (w_t + p_i) \cdot (w_t + p_i)
\]
\[
= ||w_t||^2 + 2w_t \cdot p_i + ||p_i||^2
\]
\[
\leq ||w_t||^2 + ||p_i||^2 \quad (\because w_t \cdot p_i \leq 0)
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

**Numerator** \( \geq w^* \cdot w_0 + k\delta \quad \text{(proved by induction)} \)

**Denominator** \( ^2 = ||w_{t+1}||^2 \)

\[
= (w_t + p_i) \cdot (w_t + p_i)
\]

\[
= ||w_t||^2 + 2w_t \cdot p_i + ||p_i||^2 \]

\[
\leq ||w_t||^2 + ||p_i||^2 \quad (\because w_t \cdot p_i \leq 0)
\]

\[
\leq ||w_t||^2 + 1 \quad (\because ||p_i||^2 = 1)
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

**Numerator** \( \geq w^* \cdot w_0 + k\delta \) (proved by induction)

**Denominator**\(^2 = ||w_{t+1}||^2 \)

\[
= (w_t + p_i) \cdot (w_t + p_i)
\]

\[
= ||w_t||^2 + 2w_t \cdot p_i + ||p_i||^2
\]

\[
\leq ||w_t||^2 + ||p_i||^2 \quad (\because w_t \cdot p_i \leq 0)
\]

\[
\leq ||w_t||^2 + 1 \quad (\because ||p_i||^2 = 1)
\]

\[
\leq (||w_{t-1}||^2 + 1) + 1
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k\delta \quad \text{(proved by induction)}
\]

\[
\text{Denominator}^2 = ||w_{t+1}||^2
\]

\[
= (w_t + p_i) \cdot (w_t + p_i)
\]

\[
= ||w_t||^2 + 2w_t \cdot p_i + ||p_i||^2
\]

\[
\leq ||w_t||^2 + ||p_i||^2 \quad (\therefore w_t \cdot p_i \leq 0)
\]

\[
\leq ||w_t||^2 + 1 \quad (\therefore ||p_i||^2 = 1)
\]

\[
\leq (||w_{t-1}||^2 + 1) + 1
\]

\[
\leq ||w_{t-1}||^2 + 2
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k \delta \quad \text{(proved by induction)}
\]

\[
\text{Denominator}^2 = ||w_{t+1}||^2
\]

\[
= (w_t + p_i) \cdot (w_t + p_i)
\]

\[
= ||w_t||^2 + 2w_t \cdot p_i + ||p_i||^2
\]

\[
\leq ||w_t||^2 + ||p_i||^2 \quad (\because w_t \cdot p_i \leq 0)
\]

\[
\leq ||w_t||^2 + 1 \quad (\because ||p_i||^2 = 1)
\]

\[
\leq (||w_{t-1}||^2 + 1) + 1
\]

\[
\leq ||w_{t-1}||^2 + 2
\]

\[
\leq ||w_0||^2 + (k) \quad \text{(By same observation that we made about} \, \delta)\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k\delta \quad \text{(proved by induction)}
\]

\[
\text{Denominator}^2 \leq ||w_0||^2 + k \quad \text{(By same observation that we made about } \delta)\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

Numerator \( \geq w^* \cdot w_0 + k\delta \) (proved by induction)

Denominator \( \leq ||w_0||^2 + k \) (By same observation that we made about \( \delta \))

\[
\cos \beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{||w_0||^2 + k}}
\]
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) \hspace{1em} (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \hspace{1em} \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k\delta \hspace{1em} \text{(proved by induction)}
\]

\[
\text{Denominator}^2 \leq ||w_0||^2 + k \hspace{1em} \text{(By same observation that we made about } \delta \text{)}
\]

\[
\cos \beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{||w_0||^2 + k}}
\]

- \( \cos \beta \) thus grows proportional to \( \sqrt{k} \)
Proof (continued:)

So far we have, $w^T \cdot p_i \leq 0$ (and hence we made the correction)

$$
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
$$

Numerator $\geq w^* \cdot w_0 + k\delta$ (proved by induction)

Denominator $^2 \leq ||w_0||^2 + k$ (By same observation that we made about $\delta$)

$$
\cos \beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{||w_0||^2 + k}}
$$

- $\cos \beta$ thus grows proportional to $\sqrt{k}$
- As $k$ (number of corrections) increases $\cos \beta$ can become arbitrarily large
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

Numerator \( \geq w^* \cdot w_0 + k\delta \) (proved by induction)

Denominator \( \leq ||w_0||^2 + k \) (By same observation that we made about \( \delta \))

\[
\cos \beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{||w_0||^2 + k}}
\]

- \( \cos \beta \) thus grows proportional to \( \sqrt{k} \)
- As \( k \) (number of corrections) increases \( \cos \beta \) can become arbitrarily large
- But since \( \cos \beta \leq 1 \), \( k \) must be bounded by a maximum number
Proof (continued:)

So far we have, \( w^T \cdot p_i \leq 0 \) (and hence we made the correction)

\[
\cos \beta = \frac{w^* \cdot w_{t+1}}{||w_{t+1}||} \quad \text{(by definition)}
\]

\[
\text{Numerator} \geq w^* \cdot w_0 + k\delta \quad \text{(proved by induction)}
\]

\[
\text{Denominator}^2 \leq ||w_0||^2 + k \quad \text{(By same observation that we made about } \delta)\]

\[
\cos \beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{||w_0||^2 + k}}
\]

\begin{itemize}
  \item \( \cos \beta \) thus grows proportional to \( \sqrt{k} \)
  \item As \( k \) (number of corrections) increases \( \cos \beta \) can become arbitrarily large
  \item But since \( \cos \beta \leq 1 \), \( k \) must be bounded by a maximum number
  \item Thus, there can only be a finite number of corrections \( (k) \) to \( w \) and the algorithm will converge!
\end{itemize}
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 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?
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 ..
Module 2.7: Linearly Separable Boolean Functions
So what do we do about functions which are not linearly separable?
• So what do we do about functions which are not linearly separable?
• Let us see one such simple boolean function first?
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The fourth condition contradicts conditions 2 and 3. Hence we cannot have a solution to this set of inequalities.
\[
\begin{array}{c|cc|c}
 x_1 & x_2 & \text{XOR} \\
\hline
0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
0 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
1 & 1 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
\end{array}
\]

\[
w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0
\]
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$
The fourth condition contradicts conditions 2 and 3. Hence we cannot have a solution to this set of inequalities.

And indeed you can see that it is impossible to draw a line which separates the red points from the blue points.

### Table: XOR

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$1$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$0$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>XOR</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

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

$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \implies w_1 + w_2 < -w_0$
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

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

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

- The fourth condition contradicts conditions 2 and 3
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

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

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

\[
\begin{align*}
w_0 + w_1 \cdot 0 + w_2 \cdot 0 &< 0 \implies w_0 < 0 \\
w_0 + w_1 \cdot 0 + w_2 \cdot 1 &\geq 0 \implies w_2 \geq -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 0 &\geq 0 \implies w_1 \geq -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 1 &< 0 \implies w_1 + w_2 < -w_0
\end{align*}
\]

- The fourth condition contradicts conditions 2 and 3
- Hence we cannot have a solution to this set of inequalities
\[
\begin{array}{c|cc|c}
 x_1 & x_2 & \text{XOR} \\
\hline
0 & 0 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
1 & 0 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
0 & 1 & 1 & w_0 + \sum_{i=1}^{2} w_i x_i \geq 0 \\
1 & 1 & 0 & w_0 + \sum_{i=1}^{2} w_i x_i < 0 \\
\end{array}
\]

\[w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0\]
\[w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0\]
\[w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0\]
\[w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \implies w_1 + w_2 < -w_0\]

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

And indeed you can see that it is impossible to draw a line which separates the red points from the blue points
Most real world data is not linearly separable and will always contain some outliers.
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.
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.

We need computational units (models) which can deal with such data.
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.

We need computational units (models) which can deal with such data.

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?

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
<th>f6</th>
<th>f7</th>
<th>f8</th>
<th>f9</th>
<th>f10</th>
<th>f11</th>
<th>f12</th>
<th>f13</th>
<th>f14</th>
<th>f15</th>
<th>f16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

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

Of these, how many are linearly separable? (turns out all except XOR and NOT-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 :-))
- How many boolean functions can you design from 2 inputs?
- Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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?

$2^2 = 4$

How many of these $2^2$ 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 :-) )
- How many boolean functions can you design from 2 inputs?
- Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
• How many boolean functions can you design from 2 inputs?
• Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_8$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• How many boolean functions can you design from 2 inputs?
• Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_8$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How many boolean functions can you design from 2 inputs?
Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_8$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- How many boolean functions can you design from 2 inputs?
- Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_8$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Of these, how many are linearly separable?

In general, how many boolean functions can you have for $n$ inputs?

$2^2n$

How many of these 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 :-))

54
• How many boolean functions can you design from 2 inputs?
• Let us begin with some easy ones which you already know ..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_8$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Of these, how many are linearly separable?

In general, how many boolean functions can you have for $n$ inputs?

$2^n$

How many of these $2^n$ 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 :-) )
- How many boolean functions can you design from 2 inputs?
- Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
- How many boolean functions can you design from 2 inputs?
- Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
• How many boolean functions can you design from 2 inputs?
• Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• How many boolean functions can you design from 2 inputs?
• Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
How many boolean functions can you design from 2 inputs?
Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How many boolean functions can you design from 2 inputs?

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Of these, how many are linearly separable?

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

|   |   | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12 | f13 | f14 | f15 | f16 |
|---|---|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|----|
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1   | 1   | 1   | 1   | 1   | 1   |
| 0 | 1 | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 0  | 0   | 0   | 1   | 1   | 1   | 1   |
| 1 | 0 | 0  | 0  | 1  | 1  | 0  | 0  | 1  | 1  | 0  | 1   | 0   | 0   | 1   | 1   | 1   |
| 1 | 1 | 0  | 1  | 0  | 1  | 0  | 1  | 0  | 1  | 0  | 1   | 0   | 1   | 0   | 1   | 0   |

- 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..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Of these, how many are linearly separable? (turns out all except XOR and !XOR - feel free to verify)
• How many boolean functions can you design from 2 inputs?
• Let us begin with some easy ones which you already know..

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

• 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?
How many boolean functions can you design from 2 inputs?

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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 boolean functions can you design from 2 inputs?

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
<th>$f_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>f₁</th>
<th>f₂</th>
<th>f₃</th>
<th>f₄</th>
<th>f₅</th>
<th>f₆</th>
<th>f₇</th>
<th>f₈</th>
<th>f₉</th>
<th>f₁₀</th>
<th>f₁₁</th>
<th>f₁₂</th>
<th>f₁₃</th>
<th>f₁₄</th>
<th>f₁₅</th>
<th>f₁₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- 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
We will now see how to implement any boolean function using a network of 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

• 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
• For this discussion, we will assume True = +1 and False = -1
• We consider 2 inputs and 4 perceptrons

\[ y = w_1 x_1 + w_2 x_2 + w_3 x_1 + w_4 x_2 + w_0 \]

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 red edge indicates \( w = -1 \)

The blue edge indicates \( w = +1 \)

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
• 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

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 $w = +1$
• 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 $w = +1$
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 $w = +1$
• 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 $w = +1$
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 $w = +1$
• 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 \(\geq 2\))

\[
\begin{align*}
\text{bias} &= -2 \\
x_1 &\quad \text{red edge indicates } w = -1 \\
x_2 &\quad \text{blue edge indicates } w = +1
\end{align*}
\]
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)
• 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)

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).

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

$\text{bias} = -2$

Red edge indicates $w = -1$

Blue edge indicates $w = +1$
Terminology:
- This network contains 3 layers

**Red edge indicates** $w = -1$
**Blue edge indicates** $w = +1$

bias $=-2$
**Terminology:**

- This network contains 3 layers
- The layer containing the inputs \((x_1, x_2)\) is called the **input layer**

![Diagram of a neural network with two input nodes, four hidden nodes, and one output node.](image)

- The red and blue edges are called layer 1 **weights**
- \(w_1, w_2, w_3, w_4\) are called layer 2 **weights**

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

- **bias** = -2

- 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**

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**

\[ y = w_1x_1 + w_2x_2 + bias \]

- 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 \(h_1, h_2, h_3, h_4\)

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

Bias: \(\text{bias} = -2\)

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

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)!

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.

- 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 $w = +1$
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!

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.

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)
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.

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).

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

red edge indicates $w = -1$
blue edge indicates $w = +1$
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 $\{1,1\}$
• 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 \( \sum_{i=1}^{4} w_i h_i \geq w_0 \))

This results in the following four conditions to implement XOR:

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

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.

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

\[
\begin{array}{cccccccc}
x_1 & x_2 & XOR & h_1 & h_2 & h_3 & h_4 & \sum_{i=1}^{4} w_i h_i \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>$h_4$</th>
<th>$\sum_{i=1}^{4} w_i h_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$w_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$w_2$</td>
</tr>
</tbody>
</table>

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

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.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>$h_4$</th>
<th>$\sum_{i=1}^{4} w_i h_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$w_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$w_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$w_3$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>$h_4$</th>
<th>$\sum_{i=1}^{4} w_i h_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$w_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$w_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$w_3$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$w_4$</td>
</tr>
</tbody>
</table>

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

- 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.

This results in the following four conditions to implement XOR: $w_1 < w_0$, $w_2 \geq w_0$, $w_3 \geq w_0$, $w_4 < w_0$.
Let $w_0$ be the bias output of the neuron (i.e., it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>$h_4$</th>
<th>$\sum_{i=1}^{4} w_i h_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$w_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$w_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$w_3$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$w_4$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>XOR</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>$h_4$</th>
<th>$\sum_{i=1}^{4} w_i h_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$w_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$w_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$w_3$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$w_4$</td>
</tr>
</tbody>
</table>

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 $w = +1$

bias = -2
- 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 $w_1, w_2, w_3, w_4$

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!

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

Catch:
As $n$ increases the number of perceptrons in the hidden layers obviously increases exponentially.
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?
• 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?
• 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? Let us see!
• We are given this data about our past movie experience

\[
\begin{pmatrix}
\begin{array}{cccc}
  x_{11} & x_{12} & \ldots & x_{1n} \\
  x_{21} & x_{22} & \ldots & x_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{k1} & x_{k2} & \ldots & x_{kn} \\
\end{array}
\end{pmatrix}
\begin{pmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_i \\
\end{pmatrix}
\begin{pmatrix}
  p_1 \\
  p_2 \\
  \vdots \\
  n_1 \\
\end{pmatrix}
\begin{pmatrix}
  1 \\
  1 \\
  \vdots \\
  0 \\
\end{pmatrix}
\]
We are given this data about our past movie experience

For each movie, we are given the values of the various factors \((x_1, x_2, \ldots, x_n)\) that we base our decision on and we are also also given the value of \(y\) (like/dislike)

\[
\begin{bmatrix}
p_1 & x_{11} & x_{12} & \ldots & x_{1n} & y_1 = 1 \\
p_2 & x_{21} & x_{22} & \ldots & x_{2n} & y_2 = 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
n_1 & x_{k1} & x_{k2} & \ldots & x_{kn} & y_i = 0 \\
n_2 & x_{j1} & x_{j2} & \ldots & x_{jn} & y_j = 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots
\end{bmatrix}
\]
We are given this data about our past movie experience

For each movie, we are given the values of the various factors \((x_1, x_2, \ldots, x_n)\) that we base our decision on and we are also also given the value of \(y\) (like/dislike)

\[
\begin{align*}
 p_1 & \quad \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} & y_1 = 1 \
 p_2 & \quad \begin{bmatrix} x_{21} & x_{22} & \cdots & x_{2n} & y_2 = 1 \\
 \vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots \\
 n_1 & \quad \begin{bmatrix} x_{k1} & x_{k2} & \cdots & x_{kn} & y_i = 0 \\
 n_2 & \quad \begin{bmatrix} x_{j1} & x_{j2} & \cdots & x_{jn} & y_j = 0 \\
 \vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots 
\end{align*}
\]

\(p_i\)’s are the points for which the output was 1 and \(n_i\)’s are the points for which it was 0
We are given this data about our past movie experience

For each movie, we are given the values of the various factors \((x_1, x_2, \ldots, x_n)\) that we base our decision on and we are also given the value of \(y\) (like/dislike)

- \(p_i\)'s are the points for which the output was 1 and \(n_i\)'s are the points for which it was 0
- The data may or may not be linearly separable
We are given this data about our past movie experience:

For each movie, we are given the values of the various factors \((x_1, x_2, \ldots, x_n)\) that we base our decision on and we are also given the value of \(y\) (like/dislike).

\(p_i\)'s are the points for which the output was 1 and \(n_i\)'s are the points for which it was 0.

The data may or may not be linearly separable.

The proof that we just saw tells us that it is possible to have a network of perceptrons and learn the weights in this network such that for any given \(p_i\) or \(n_j\) the output of the network will be the same as \(y_i\) or \(y_j\) (i.e., we can separate the positive and the negative points).
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)
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 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.
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