EE5904/ME5404: Neural Networks
Lecture 02

Xingyu Liu
xyl@nus.edu.sg

Assistant Professor
Department of Electrical & Computer Engineering
National University of Singapore

Assignment 1 is out!

Due 23:59 (SGT), Sunday, 15 February 2026.

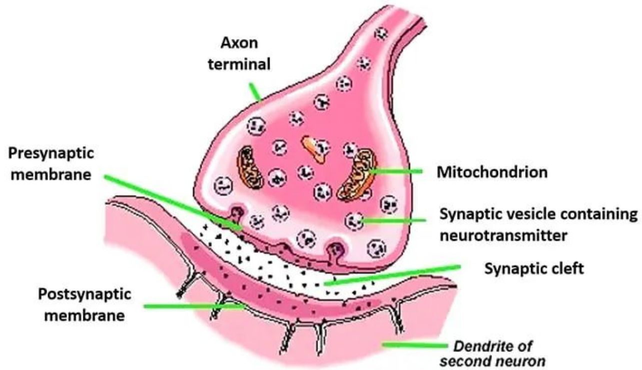
Submission instructions

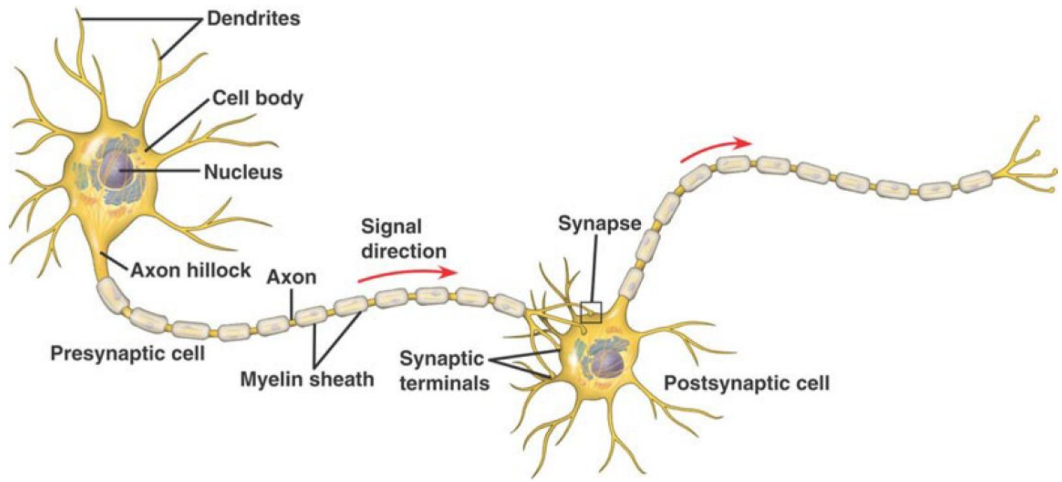
- Submit the assignment via Canvas.
- Any Python code should be included as an attachment.

Handwritten submissions are encouraged!

- If all questions (except the Python code) are handwritten, you will receive a **10% bonus** on the assignment score.
- For handwritten work, **take clear photos of the pages** and upload them to Canvas.
- Ensure that your handwriting and photos are **clear and legible**. Illegible submissions may lose marks.

Recap of last lecture





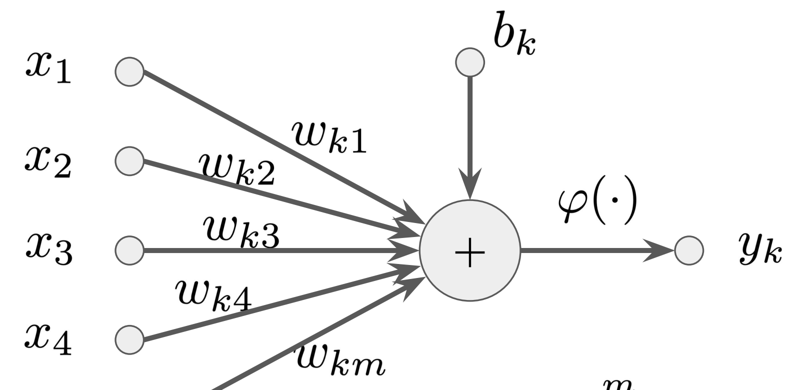
Biological neurons:

1. Receives information in the form of electrical pulses from many other neurons.
2. Does a complex dynamic **sum of these inputs** by receiving / sending all the charged ions from / to all synapses.
3. Sends out information in the form of a stream of electrical impulses along its axon and to many other neurons.
4. The connections (synapses) are crucial for excitation or inhibition of the cells.
5. Learning is done by adjusting the synapses.

Recap of last lecture

Artificial neurons:

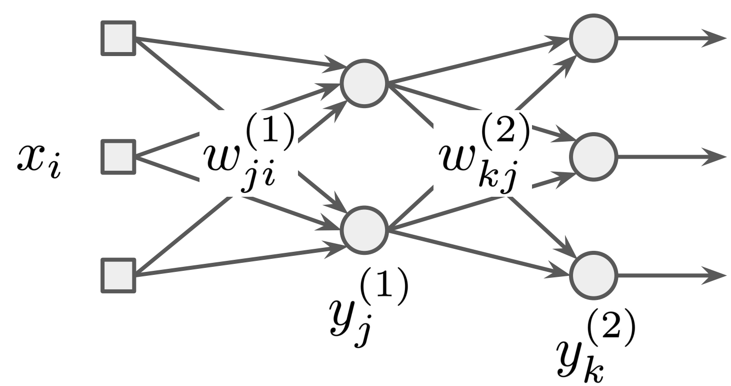
1. A set of **synapses** with weights;
2. An **adder** for weighted summation of the input signals;
3. An **activation function** for modulating the neuron output.



$$y_k = \varphi(-b_k + \sum_{i=1}^{m} w_{ki}x_i)$$

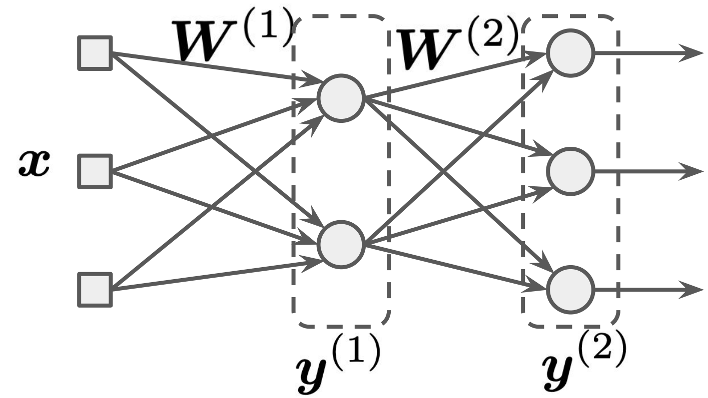
Artificial neural networks:

1. Networks consisting of a large number of interconnected neurons.
2. Network architecture defines the number of neurons and how they are connected.



Recap of last lecture

Layered feedforward Networks:
- Neurons partitioned into layers;
- No connections from layer j to layer i if $j > i$;
- Directed graph, no cyclic connections



Vectorized computation

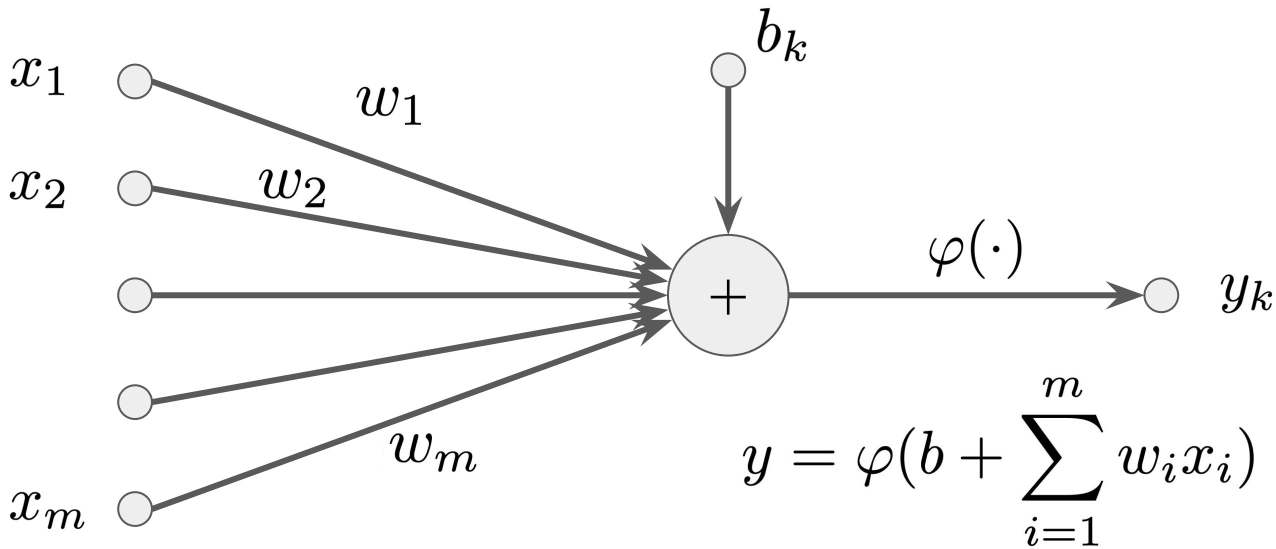
$$\boldsymbol{x} = [x_1, x_2, \ldots, x_i, \ldots]^T$$
$$\boldsymbol{y}^{(1)} = [y_1^{(1)}, y_2^{(1)}, \ldots, y_j^{(1)}, \ldots]^T$$
$$\boldsymbol{y}^{(2)} = [y_1^{(2)}, y_2^{(2)}, \ldots, y_k^{(2)}, \ldots]^T$$

$$\boldsymbol{W}^{(1)} = [w_{ji}^{(1)}]$$
$$\boldsymbol{W}^{(2)} = [w_{kj}^{(2)}]$$

$$\boldsymbol{y}^{(1)} = \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x}) \qquad \Rightarrow \qquad \boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\varphi(\boldsymbol{W}^{(1)}\boldsymbol{x})$$
$$\boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{y}^{(1)}$$

Perceptron

Perceptron: definition



$$y = \varphi(b + \sum_{i=1}^{m} w_i x_i)$$

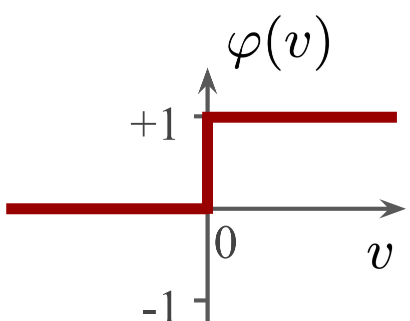
A single artificial neuron with threshold activation function

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

$$\boldsymbol{x} = [1, x_1, x_2, \ldots, x_m]^T$$

$$\boldsymbol{w} = [b, w_1, w_2, \ldots, w_m]^T$$

$$\boldsymbol{y} = \varphi(\boldsymbol{w}^T \boldsymbol{x})$$



Learning

Definition of learning
Learning is a process of adjusting the parameters of a computation model to optimize an objective under data drawn from the environment.

Learning

Definition of learning
Learning is a process of adjusting the parameters of a computation model to optimize an objective under data drawn from the environment.

Suppose $y = f_\theta(x), (x, y) \sim \mathcal{D}$

Learning

Definition of learning
Learning is a process of adjusting the parameters of a computation model to optimize an objective under data drawn from the environment.

Suppose $y = f_\theta(x), (x, y) \sim \mathcal{D}$

Learning: update θ to find $\theta^* = \arg\min_\theta \; \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(f_\theta(x), y)\right]$

- $f_\theta(x)$: computational model that produces outputs from inputs
- θ : parameters of the computation model
- $\mathcal{D}$: environment data distribution / dataset
- $\mathcal{L}$: objective function

Learning

Definition of learning

Learning is a process of adjusting the parameters of a computation model to optimize an objective under data drawn from the environment.

Suppose $y = f_\theta(x), (x, y) \sim \mathcal{D}$

Learning: update θ to find $\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(f_\theta(x), y)\right]$

- $f_\theta(x)$: computational model that produces outputs from inputs
- θ : parameters of the computation model
- $\mathcal{D}$: environment data distribution / dataset
- $\mathcal{L}$: objective function

Perceptron and neural networks are just instantiation of the computation models $y = f_\theta(x)$

Learning

Supervised Learning:

- **Data** $(x, y) \sim \mathcal{D}$: both data from environment and target label
- **Objective** $\mathcal{L}(f_\theta(x), y)$: loss function to quantify prediction error
  - e.g. L2 loss: $||f_\theta(x) - y||_2^2$ for regression problems

Learning

Supervised Learning:
- **Data** $(x, y) \sim \mathcal{D}$: both data from environment and target label
- **Objective** $\mathcal{L}(f_\theta(x), y)$: loss function to quantify prediction error
  - e.g. L2 loss: $||f_\theta(x) - y||_2^2$ for regression problems

Unsupervised Learning:
- **Data** $x \sim \mathcal{D}$: data from environment only
- **Objective** $\mathcal{L}(f_\theta(x))$: probabilistic objective function for capturing internal structure of the data
  - e.g. log-likelihood $\log p_\theta(x)$ generative modeling problems

Learning

Supervised Learning:

- **Data** $(x, y) \sim \mathcal{D}$: both data from environment and target label
- **Objective** $\mathcal{L}(f_\theta(x), y)$: loss function to quantify prediction error
  - e.g. L2 loss: $||f_\theta(x) - y||_2^2$ for regression problems

Unsupervised Learning:

- **Data** $x \sim \mathcal{D}$: data from environment only
- **Objective** $\mathcal{L}(f_\theta(x))$: probabilistic objective function for capturing internal structure of the data
  - e.g. log-likelihood $\log p_\theta(x)$ generative modeling problems

Reinforcement Learning:

- **Data** $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$: state, action and reward trajectory generated from interaction with environment using actions a_t by follows policy π_θ: $a_t \sim \pi_\theta(\cdot \mid s_t)$
- **Objective** $R(\tau) = \sum_t \gamma^t r_t$: accumulated discounted reward

Two phases of learning

1. Training / Learning

Optimize θ using data and objective function to find the optimal

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(f_\theta(x), y)\right]$$

Two phases of learning

1. Training / Learning

Optimize θ using data and objective function to find the optimal

$$\theta^* = \arg\min_{\theta} \ \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(f_\theta(x), y)\right]$$

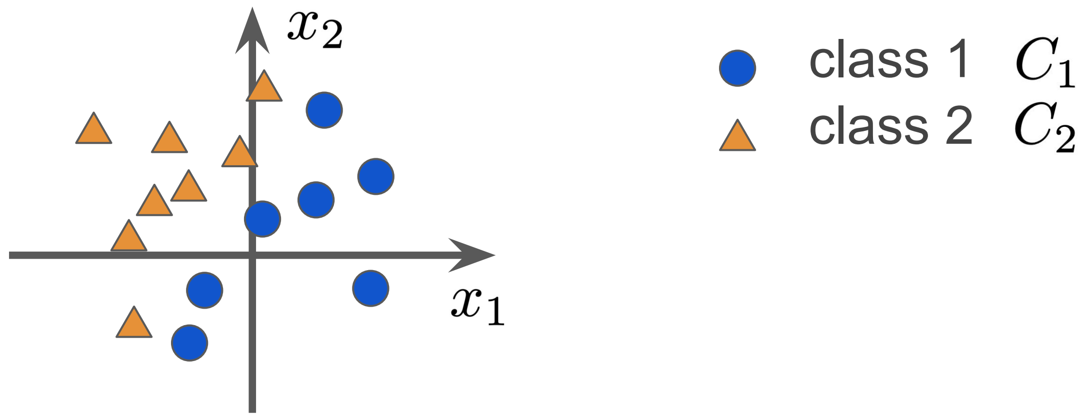
2. Testing / Evaluation

Using the trained θ^* to predict the output of **<u>new / unseen</u>** data

$$y_{\text{test}} = f_{\theta^*}(x_{\text{test}})$$

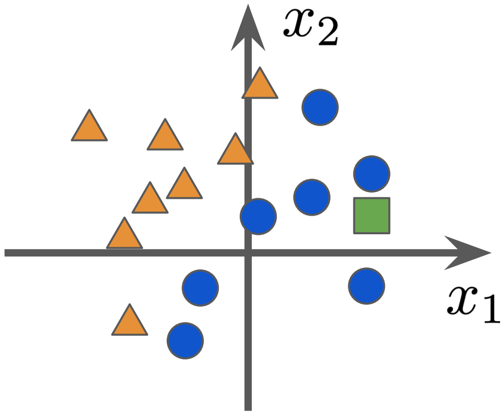
<u>The test data cannot be touched during training!</u>

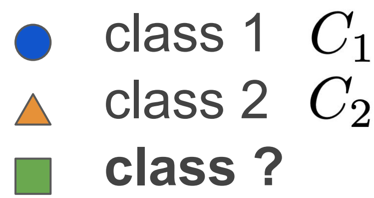
Perceptron: application to classification problem



Problem formulation: categorize the (continuous) input data into **discrete** categories / classes.

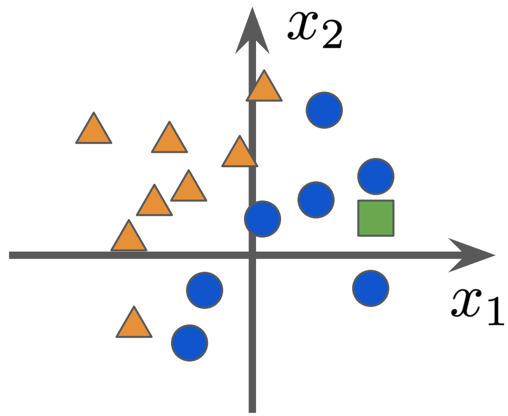
Perceptron: application to classification problem





Problem formulation: categorize the (continuous) input data $\boldsymbol{x}$ into **discrete** categories / classes.

Perceptron: application to classification problem

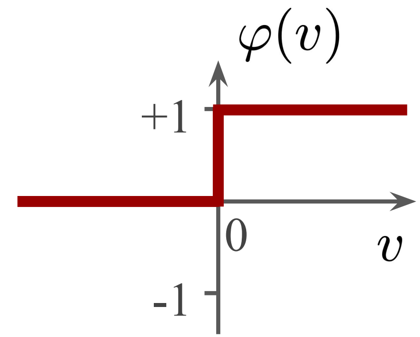


class 1 C_1
class 2 C_2
class ?

Problem formulation: categorize the (continuous) input data $\boldsymbol{x}$ into **discrete** categories / classes.

$$\boldsymbol{y} = \varphi(\boldsymbol{w}^T \boldsymbol{x} + b)$$

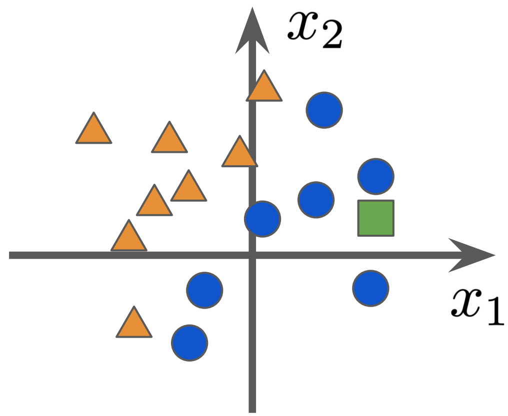
$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$



$\boldsymbol{y} = 0 \Rightarrow$ class 1

$\boldsymbol{y} = 1 \Rightarrow$ class 2

Perceptron: application to classification problem

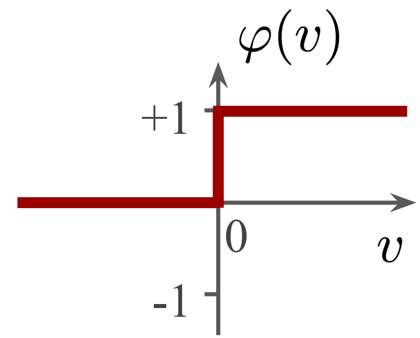


- class 1 C_1
- class 2 C_2
- **class ?**

Problem formulation: categorize the (continuous) input data $\boldsymbol{x}$ into **discrete** categories / classes.

$$\boldsymbol{y} = \varphi(\boldsymbol{w}^T \boldsymbol{x} + b)$$

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$



$\boldsymbol{y} = 0 \Rightarrow$ class 1

$\boldsymbol{y} = 1 \Rightarrow$ class 2

Does it make sense to use perceptron in this way to solve classification?

Perceptron: application to classification problem

class 1: $\boldsymbol{y} = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{x} + b < 0$

class 2: $\boldsymbol{y} = 1 \Rightarrow \boldsymbol{w}^T \boldsymbol{x} + b > 0$

decision boundary: $\boldsymbol{w}^T \boldsymbol{x} + b = 0$

Perceptron: application to classification problem

class 1: $\boldsymbol{y} = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{x} + b < 0$

class 2: $\boldsymbol{y} = 1 \Rightarrow \boldsymbol{w}^T \boldsymbol{x} + b > 0$

decision boundary: $\boldsymbol{w}^T \boldsymbol{x} + b = 0$

What is the geometric shape of the decision boundary?

Perceptron: application to classification problem

class 1: $\boldsymbol{y} = 0 \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b < 0$

class 2: $\boldsymbol{y} = 1 \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b > 0$

decision boundary: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$

<u>What is the geometric shape of the decision boundary?</u>

$\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$

If $m = 1$ $\quad w_1 x_1 + b = 0$ ⇨ A point on a line

Perceptron: application to classification problem

class 1: $\boldsymbol{y} = 0 \Rightarrow \boldsymbol{w}^T \boldsymbol{x} + b < 0$

class 2: $\boldsymbol{y} = 1 \Rightarrow \boldsymbol{w}^T \boldsymbol{x} + b > 0$

decision boundary: $\boldsymbol{w}^T \boldsymbol{x} + b = 0$

<u>What is the geometric shape of the decision boundary?</u>

$\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$

If $m = 1$ $\quad w_1 x_1 + b = 0$ ⇨ A point on a line

If $m = 2$ $\quad w_1 x_1 + w_2 x_2 + b = 0$ ⇨ A line in a 2D plane

Perceptron: application to classification problem

class 1: $\boldsymbol{y} = 0 \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b < 0$

class 2: $\boldsymbol{y} = 1 \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b > 0$

decision boundary: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$

<u>What is the geometric shape of the decision boundary?</u>

$\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$

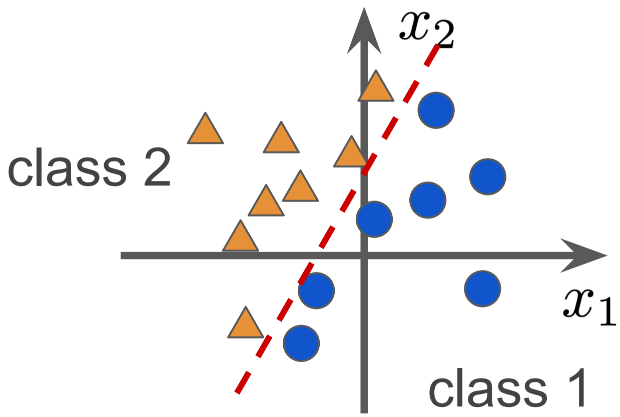
If $m = 1$ $\quad w_1x_1 + b = 0$ ⇨ A point on a line

If $m = 2$ $\quad w_1x_1 + w_2x_2 + b = 0$ ⇨ A line in a 2D plane

If $m = 3$ $\quad w_1x_1 + w_2x_2 + w_3x_3 + b = 0$ ⇨ A plane in 3D space

For a general m, it is a **hyperplane** in the m-dimensional space; Classes 1 and 2 corresponds to the two sub-spaces separated by the hyperplane

Perceptron: application to classification problem

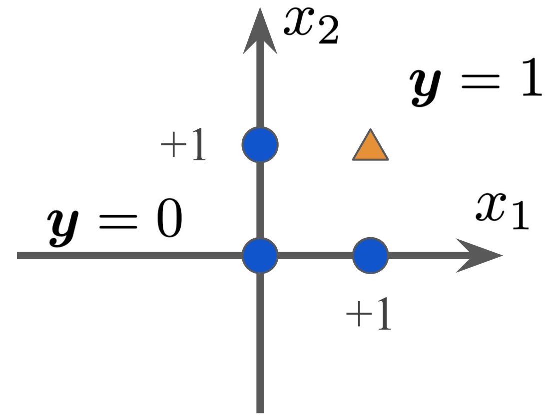


How to properly set the weights and biases that can produce desired decision boundary?

This is also the “**ONLY**” question for machine learning in general!

Example 1: AND gate

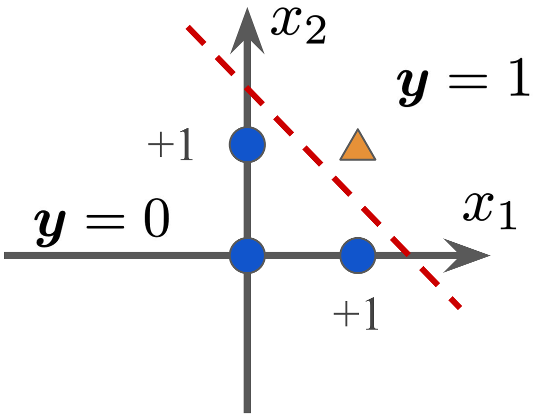
x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1



Let's first try to set the weights and bias of a desired decision boundary!

Example 1: AND gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1

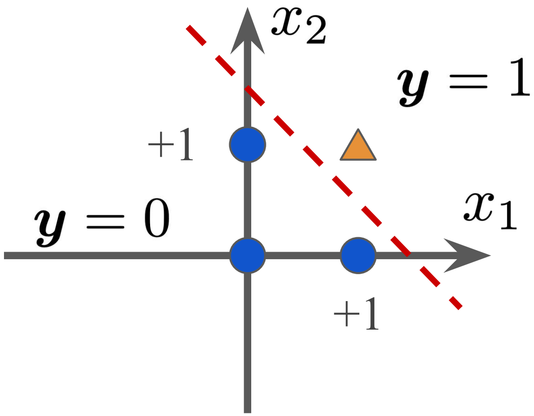


Let's first try to set the weights and bias of a desired decision boundary!

A line that go through (0.5, 1) and (1, 0.5) should work!

Example 1: AND gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1



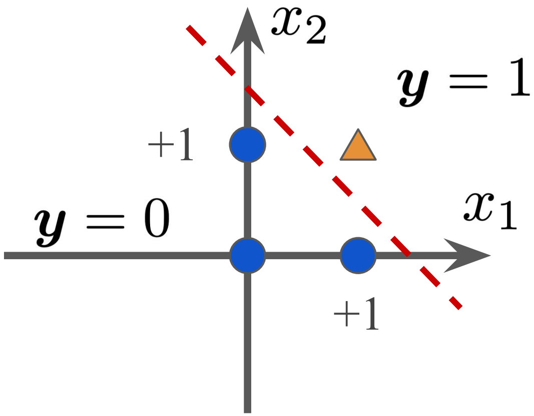
Let's first try to set the weights and bias of a desired decision boundary!

A line that go through (0.5, 1) and (1, 0.5) should work!

Decision boundary: $x_1 + x_2 - 1.5 = 0$

Example 1: AND gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1



Let's first try to set the weights and bias of a desired decision boundary!

A line that go through (0.5, 1) and (1, 0.5) should work!

Decision boundary: $x_1 + x_2 - 1.5 = 0$

<u>What is the corresponding perceptron?</u>

Perceptron from decision boundary

If the decision boundary hyperplane is provided, what is the corresponding perceptron?

1. Put the equation for the hyper-plane in the following form:

$$b + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = 0$$

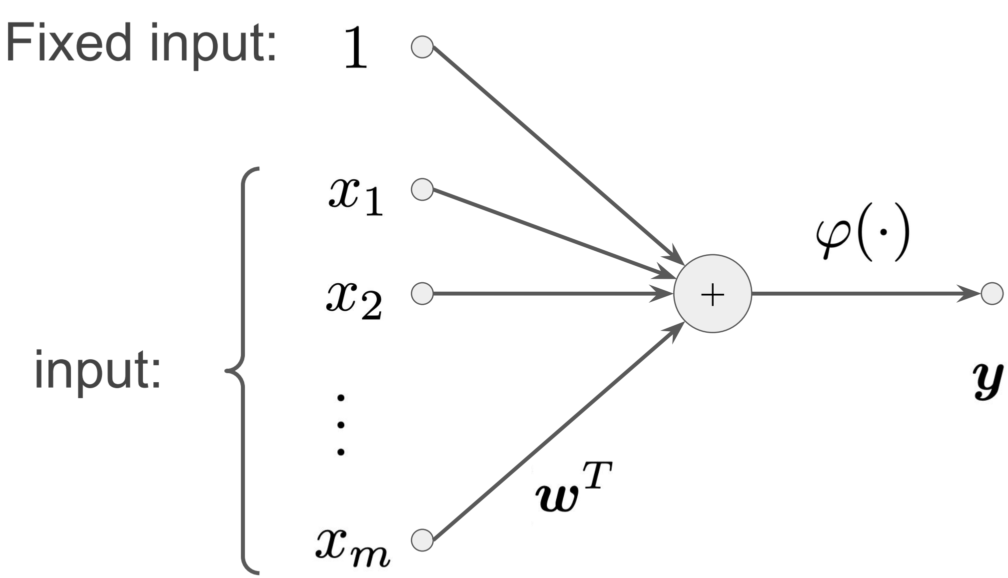
Perceptron from decision boundary

If the decision boundary hyperplane is provided, what is the corresponding perceptron?

1. Put the equation for the hyper-plane in the following form:

$$b + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = 0$$

2. The corresponding perceptron is:



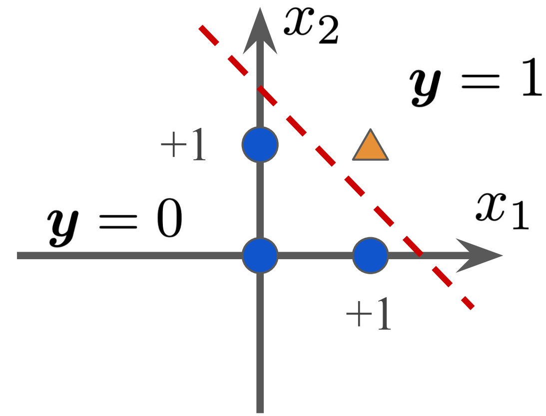
$$\boldsymbol{x} = [1, x_1, x_2, \ldots, x_m]^T$$

$$\boldsymbol{w} = [b, w_1, w_2, \ldots, w_m]^T$$

$$\boldsymbol{y} = \varphi(\boldsymbol{w}^T \boldsymbol{x})$$

Example 1: AND gate

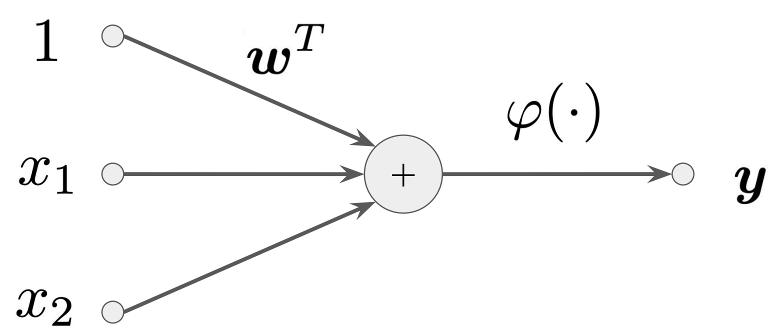
x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1



Let's first try to set the weights and bias of a desired decision boundary!

A line that go through (0.5, 1) and (1, 0.5) should work!

Decision boundary: $x_1 + x_2 - 1.5 = 0$

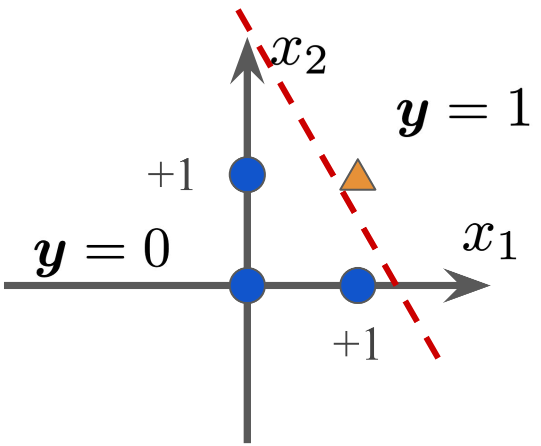


$$\boldsymbol{w} = [-1.5, 1, 1]^T$$

(Recall $\boldsymbol{w} = [b, w_1, w_2, \ldots, w_m]^T$)

Example 1: AND gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1

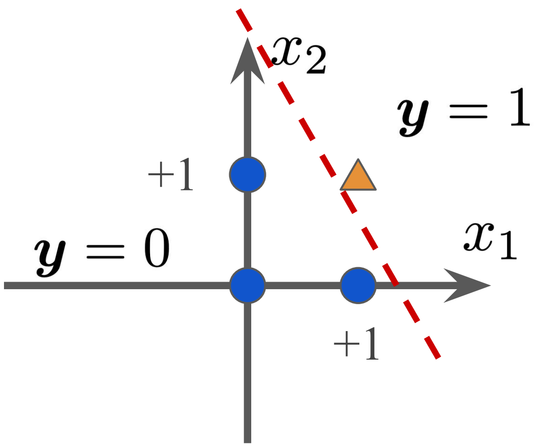


The viable decision boundary may not be unique!

For example, a line that go through (0.75, 1) and (1, 0.5) also works!

Example 1: AND gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	0	0	1

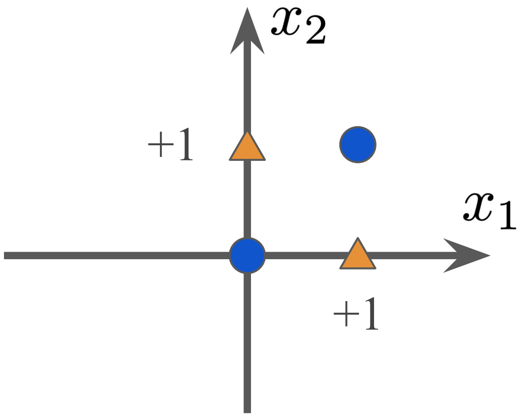


The viable decision boundary may not be unique!
For example, a line that go through (0.75, 1) and (1, 0.5) also works!

Decision boundary would be $2x_1 + x_2 - 2.5 = 0$

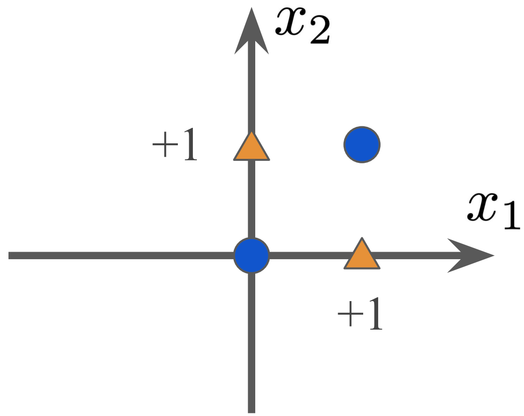
Example 2: XOR gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	1	1	0



Example 2: XOR gate

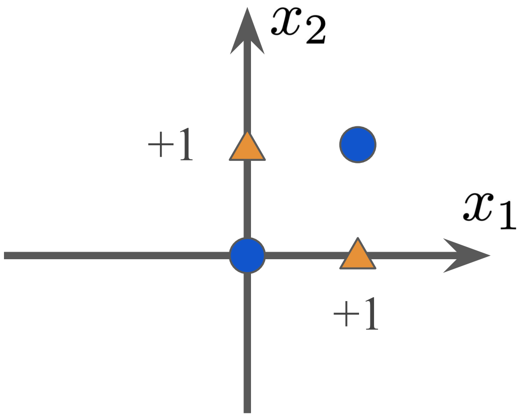
x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	1	1	0



Can the four data points of the XOR logic function be separated by one hyperplane decision boundary?

Example 2: XOR gate

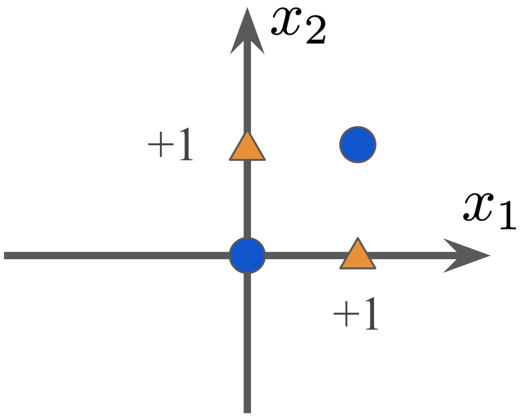
x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	1	1	0



Can the four data points of the XOR logic function be separated by one hyperplane decision boundary?
No.

Example 2: XOR gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	1	1	0



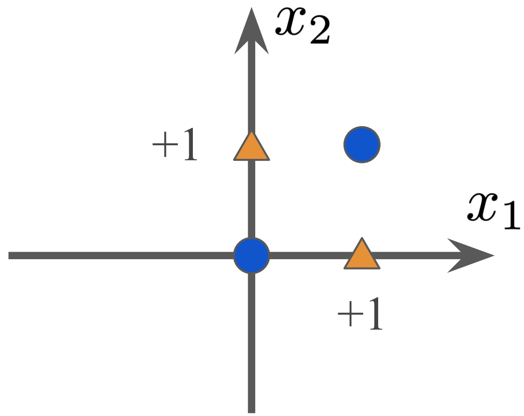
<u>Can the four data points of the XOR logic function be separated by one hyperplane decision boundary?</u>

No.

<u>Can a single perceptron produce more than one hyperplane decision boundary?</u>

Example 2: XOR gate

x_1	0	0	1	1
x_2	0	1	0	1
$\boldsymbol{y}$	0	1	1	0



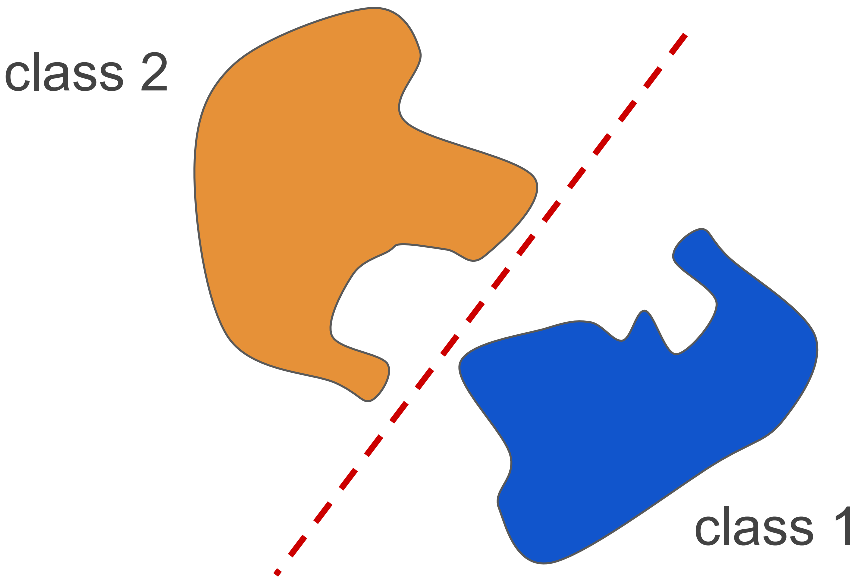
Can the four data points of the XOR logic function be separated by one hyperplane decision boundary?
No.

Can a single perceptron produce more than one hyperplane decision boundary?
No.

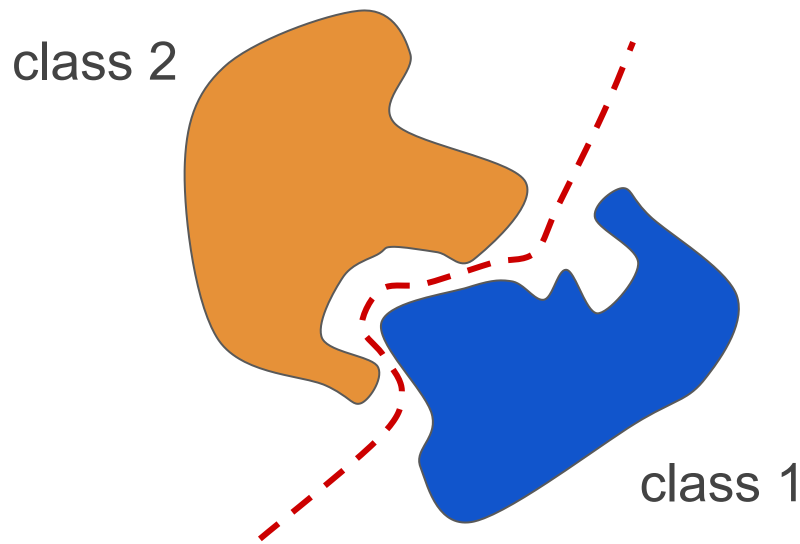
Linearly separable

Definition
Data samples of two classes can be separated by one hyperplane.

Two classes are linearly separable if and only if there exists a weight $\boldsymbol{w}$ with which the perceptron can correctly perform classification.



Linearly separable.



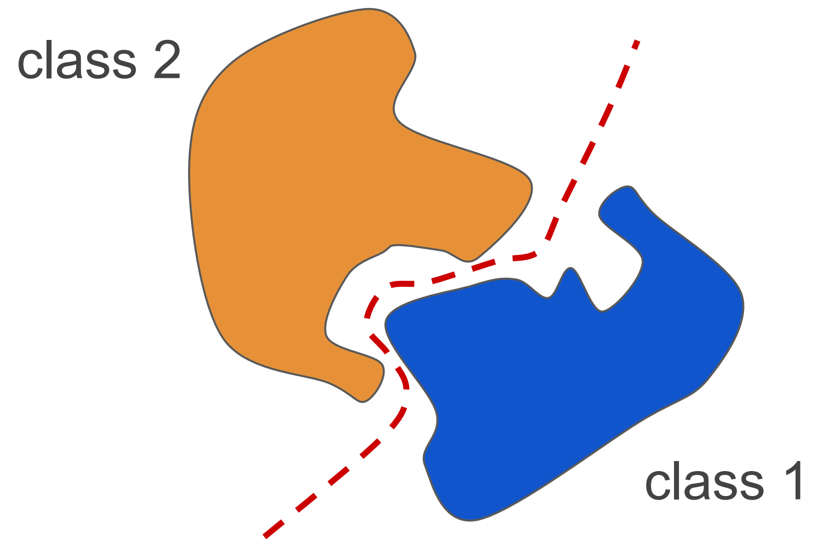
NOT Linearly separable.

Linearly separable

Definition
Data samples of two classes can be separated by one hyperplane.

Two classes are linearly separable if and only if there exists a weight $\boldsymbol{w}$ with which the perceptron can correctly perform classification.

If the samples are NOT linearly separable, there cannot be any simple perceptron that achieves the classification task.



NOT Linearly separable.

Perceptron from decision boundary

Geometric visualization of decision boundaries are possible for 2-dimensional or 3-dimensional data

Perceptron from decision boundary

Geometric visualization of decision boundaries are possible for 2-dimensional or 3-dimensional data

How about 4-dimensional data? Not so straightforward.

Perceptron from decision boundary

Geometric visualization of decision boundaries are possible for 2-dimensional or 3-dimensional data

How about 4-dimensional data? Not so straightforward.

In real-world applications, the dimension of the input pattern is very high, e.g. the data dimension of a 100×100 resolution image is 10000.
How about 10000-dimensional data?

Perceptron from decision boundary

Geometric visualization of decision boundaries are possible for 2-dimensional or 3-dimensional data

How about 4-dimensional data? Not so straightforward.

In real-world applications, the dimension of the input pattern is very high, e.g. the data dimension of a 100×100 resolution image is 10000.
How about 10000-dimensional data?

In 1958, Rosenblatt demonstrated that Perceptron can learn to do the job correctly!

Perceptron was the first computer that could learn new skills simply by trial and error.

Perceptron learning

Now, suppose that the input variables of the perceptron are from **TWO** linearly separable classes C_1 and C_2.

So there must exist a $\boldsymbol{w}_0$ such that

$$\boldsymbol{w}_0^T \boldsymbol{x} < 0, \forall \boldsymbol{x} \in C_1$$
$$\boldsymbol{w}_0^T \boldsymbol{x} > 0, \forall \boldsymbol{x} \in C_2$$

Perceptron learning

Now, suppose that the input variables of the perceptron are from **TWO** linearly separable classes C_1 and C_2.

So there must exist a $\boldsymbol{w}_0$ such that

$$\boldsymbol{w}_0^T \boldsymbol{x} < 0, \forall \boldsymbol{x} \in C_1$$
$$\boldsymbol{w}_0^T \boldsymbol{x} > 0, \forall \boldsymbol{x} \in C_2$$

In other words, we know there is a solution. But where is it?

Perceptron learning

Now, suppose that the input variables of the perceptron are from **TWO** linearly separable classes C_1 and C_2.

So there must exist a $\boldsymbol{w}_0$ such that

$$\boldsymbol{w}_0^T \boldsymbol{x} < 0, \forall \boldsymbol{x} \in C_1$$
$$\boldsymbol{w}_0^T \boldsymbol{x} > 0, \forall \boldsymbol{x} \in C_2$$

In other words, we know there is a solution. But where is it?

If can be found via **trial and error**!

Perceptron learning

Feed a pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Suppose $\boldsymbol{w}^T\boldsymbol{x} < 0 \Rightarrow y = 0$

Perceptron learning

Feed a pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Suppose $\boldsymbol{w}^T\boldsymbol{x} < 0 \Rightarrow y = 0$

If the correct label / desired output is also $y^* = 0$

Perceptron learning

Feed a pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Suppose $\boldsymbol{w}^T\boldsymbol{x} < 0 \Rightarrow y = 0$

If the correct label / desired output is also $y^* = 0$
There is nothing needs to be done!

Perceptron learning

Feed a pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Suppose $\boldsymbol{w}^T\boldsymbol{x} < 0 \Rightarrow y = 0$

If the correct label / desired output is also $y^* = 0$
There is nothing needs to be done!

If the correct label / desired output is $y^* = 1$
<u>We should update the perceptron weights to fix the mistake!</u>

Perceptron learning

Feed a pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Suppose $\boldsymbol{w}^T\boldsymbol{x} < 0 \Rightarrow y = 0$

If the correct label / desired output is also $y^* = 0$
There is nothing needs to be done!

If the correct label / desired output is $y^* = 1$
<u>We should update the perceptron weights to fix the mistake!</u>

$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$$

How to choose $\Delta\boldsymbol{w}$?

Perceptron learning

Feed a pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Suppose $\boldsymbol{w}^T\boldsymbol{x} < 0 \Rightarrow y = 0$

If the correct label / desired output is also $y^* = 0$
There is nothing needs to be done!

If the correct label / desired output is $y^* = 1$
We should update the perceptron weights to fix the mistake!

$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$$

How to choose $\Delta\boldsymbol{w}$?
We choose $\Delta\boldsymbol{w}$ such that $\boldsymbol{w}^T\boldsymbol{x}$ can eventually become positive!

Perceptron learning

Let’s check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

<u>What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?</u>

$\Delta \boldsymbol{w}^T \boldsymbol{x}$

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

$$\Delta \boldsymbol{w}^T \boldsymbol{x}$$

If $\boldsymbol{w}^T \boldsymbol{x} < 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become positive, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

$$\Delta \boldsymbol{w}^T \boldsymbol{x}$$

If $\boldsymbol{w}^T \boldsymbol{x} < 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become positive, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?

Positive.

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

<u>What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?</u>

$\Delta \boldsymbol{w}^T \boldsymbol{x}$

If $\boldsymbol{w}^T \boldsymbol{x} < 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become positive, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?
Positive.

<u>What is the simplest way to make $\Delta \boldsymbol{w}^T \boldsymbol{x}$ positive, for a given $\boldsymbol{x}$?</u>

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

$\Delta \boldsymbol{w}^T \boldsymbol{x}$

If $\boldsymbol{w}^T \boldsymbol{x} < 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become positive, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?
Positive.

What is the simplest way to make $\Delta \boldsymbol{w}^T \boldsymbol{x}$ positive, for a given $\boldsymbol{x}$?
Just set $\Delta \boldsymbol{w} = \boldsymbol{x}$!

To avoid too big jump of the weights, let's use a small step size $\Delta \boldsymbol{w} = \eta \boldsymbol{x}$ where η is the **learning rate**.

$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \eta \boldsymbol{x}$$

Perceptron learning

Feed another pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Let's consider another case: $\boldsymbol{w}^T\boldsymbol{x} > 0 \Rightarrow y = 1$

Perceptron learning

Feed another pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Let's consider another case: $\boldsymbol{w}^T\boldsymbol{x} > 0 \Rightarrow y = 1$

If the correct label / desired output is also $y^* = 1$

Perceptron learning

Feed another pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Let's consider another case: $\boldsymbol{w}^T\boldsymbol{x} > 0 \Rightarrow y = 1$

If the correct label / desired output is also $y^* = 1$
There is nothing needs to be done!

Perceptron learning

Feed another pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Let's consider another case: $\boldsymbol{w}^T\boldsymbol{x} > 0 \Rightarrow y = 1$

If the correct label / desired output is also $y^* = 1$
There is nothing needs to be done!

If the correct label / desired output is $y^* = 0$
We should update the perceptron weights to fix the mistake!

Perceptron learning

Feed another pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Let's consider another case: $\boldsymbol{w}^T\boldsymbol{x} > 0 \Rightarrow y = 1$

If the correct label / desired output is also $y^* = 1$
There is nothing needs to be done!

If the correct label / desired output is $y^* = 0$
We should update the perceptron weights to fix the mistake!

$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$

How to choose $\Delta\boldsymbol{w}$?

Perceptron learning

Feed another pattern $\boldsymbol{x}$ to the perceptron with weight vector $\boldsymbol{w}$, it will produce a binary output y

Let's consider another case: $\boldsymbol{w}^T\boldsymbol{x} > 0 \Rightarrow y = 1$

If the correct label / desired output is also $y^* = 1$
There is nothing needs to be done!

If the correct label / desired output is $y^* = 0$
We should update the perceptron weights to fix the mistake!

$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$$

How to choose $\Delta\boldsymbol{w}$?
We choose $\Delta\boldsymbol{w}$ such that $\boldsymbol{w}^T\boldsymbol{x}$ can eventually become negative!

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

<u>What is the change of value of $\boldsymbol{w}^T\boldsymbol{x}$?</u>

$\Delta \boldsymbol{w}^T\boldsymbol{x}$

If $\boldsymbol{w}^T\boldsymbol{x} > 0$, but we want $\boldsymbol{w}^T\boldsymbol{x}$ to eventually become negative, should $\Delta \boldsymbol{w}^T\boldsymbol{x}$ be positive or negative?

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

$\Delta \boldsymbol{w}^T \boldsymbol{x}$

If $\boldsymbol{w}^T \boldsymbol{x} > 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become negative, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?

Negative.

Perceptron learning

Let's check the effect of applying $\boldsymbol{\Delta w}$:

What is the change of value of $\boldsymbol{w}^T\boldsymbol{x}$?

$\boldsymbol{\Delta w}^T\boldsymbol{x}$

If $\boldsymbol{w}^T\boldsymbol{x} > 0$, but we want $\boldsymbol{w}^T\boldsymbol{x}$ to eventually become negative, should $\boldsymbol{\Delta w}^T\boldsymbol{x}$ be positive or negative?
Negative.

What is the simplest way to make $\boldsymbol{\Delta w}^T\boldsymbol{x}$ negative, for a given $\boldsymbol{x}$?

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

$\Delta \boldsymbol{w}^T \boldsymbol{x}$

If $\boldsymbol{w}^T \boldsymbol{x} > 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become negative, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?
Negative.

What is the simplest way to make $\Delta \boldsymbol{w}^T \boldsymbol{x}$ negative, for a given $\boldsymbol{x}$?
Just set $\Delta \boldsymbol{w} = -\boldsymbol{x}$!

Perceptron learning

Let's check the effect of applying $\Delta \boldsymbol{w}$:

What is the change of value of $\boldsymbol{w}^T \boldsymbol{x}$?

$$\Delta \boldsymbol{w}^T \boldsymbol{x}$$

If $\boldsymbol{w}^T \boldsymbol{x} > 0$, but we want $\boldsymbol{w}^T \boldsymbol{x}$ to eventually become negative, should $\Delta \boldsymbol{w}^T \boldsymbol{x}$ be positive or negative?
Negative.

What is the simplest way to make $\Delta \boldsymbol{w}^T \boldsymbol{x}$ negative, for a given $\boldsymbol{x}$?
Just set $\Delta \boldsymbol{w} = -\boldsymbol{x}$!

To avoid too big jump of the weights, let's use a small step size $\Delta \boldsymbol{w} = -\eta \boldsymbol{x}$ where η is the **learning rate**.

$$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \eta \boldsymbol{x}$$

Perceptron learning

Put them together:

- If the true label is $y^* = 1$, and the perceptron makes a mistake to predict $y = 0$, its synaptic weights are adjusted by
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \eta \boldsymbol{x}$$
- If the true label is $y^* = 0$, and the perceptron makes a mistake to predict $y = 1$, its synaptic weights are adjusted by
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \eta \boldsymbol{x}$$

Perceptron learning

Put them together:

- If the true label is $y^* = 1$, and the perceptron makes a mistake to predict $y = 0$, its synaptic weights are adjusted by
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \eta \boldsymbol{x}$$
- If the true label is $y^* = 0$, and the perceptron makes a mistake to predict $y = 1$, its synaptic weights are adjusted by
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \eta \boldsymbol{x}$$

Let's unify these two case! The core is to introduce prediction error
$$e = y^* - y$$
that shows in which direction should the prediction change to be correct

Perceptron learning

Put them together:

- If the true label is $y^* = 1$, and the perceptron makes a mistake to predict $y = 0$, its synaptic weights are adjusted by
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \eta \boldsymbol{x}$$
- If the true label is $y^* = 0$, and the perceptron makes a mistake to predict $y = 1$, its synaptic weights are adjusted by
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \eta \boldsymbol{x}$$

Let's unify these two case! The core is to introduce prediction error
$$e = y^* - y$$
that shows in which direction should the prediction change to be correct

Then
$$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \eta e \boldsymbol{x}$$

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
draw the next misclassified sample $(\boldsymbol{x}^{(n)}, y^{*(n)})$
compute prediction and prediction error:

$$y^{(n)} \leftarrow \varphi({\boldsymbol{w}^{(n)}}^T \boldsymbol{x}^{(n)})$$

$$e^{(n)} \leftarrow y^{*(n)} - y^{(n)}$$

update the weight vector

$$\boldsymbol{w}^{(n+1)} \leftarrow \boldsymbol{w}^{(n)} + \eta \cdot e^{(n)} \boldsymbol{x}^{(n)}$$

increment n
end

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
    draw the next misclassified sample $(\boldsymbol{x}^{(n)}, y^{*(n)})$

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
draw the next misclassified sample $(\boldsymbol{x}^{(n)}, {y^*}^{(n)})$
compute prediction and prediction error:

$$y^{(n)} \leftarrow \varphi({\boldsymbol{w}^{(n)}}^T \boldsymbol{x}^{(n)})$$

$$e^{(n)} \leftarrow {y^*}^{(n)} - y^{(n)}$$

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
draw the next misclassified sample $(\boldsymbol{x}^{(n)}, y^{*(n)})$
compute prediction and prediction error:

$$y^{(n)} \leftarrow \varphi({\boldsymbol{w}^{(n)}}^T \boldsymbol{x}^{(n)})$$

$$e^{(n)} \leftarrow y^{*(n)} - y^{(n)}$$

update the weight vector

$$\boldsymbol{w}^{(n+1)} \leftarrow \boldsymbol{w}^{(n)} + \eta \cdot e^{(n)} \boldsymbol{x}^{(n)}$$

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
    draw the next misclassified sample $(\boldsymbol{x}^{(n)}, {y^*}^{(n)})$
    compute prediction and prediction error:

$$y^{(n)} \leftarrow \varphi({\boldsymbol{w}^{(n)}}^T \boldsymbol{x}^{(n)})$$

$$e^{(n)} \leftarrow {y^*}^{(n)} - y^{(n)}$$

    update the weight vector

$$\boldsymbol{w}^{(n+1)} \leftarrow \boldsymbol{w}^{(n)} + \eta \cdot e^{(n)} \boldsymbol{x}^{(n)}$$

    increment n
end

Perceptron learning: algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
draw the next misclassified sample $(\boldsymbol{x}^{(n)}, y^{*(n)})$
compute prediction and prediction error:

$$y^{(n)} \leftarrow \varphi({\boldsymbol{w}^{(n)}}^T \boldsymbol{x}^{(n)})$$

$$e^{(n)} \leftarrow y^{*(n)} - y^{(n)}$$

update the weight vector

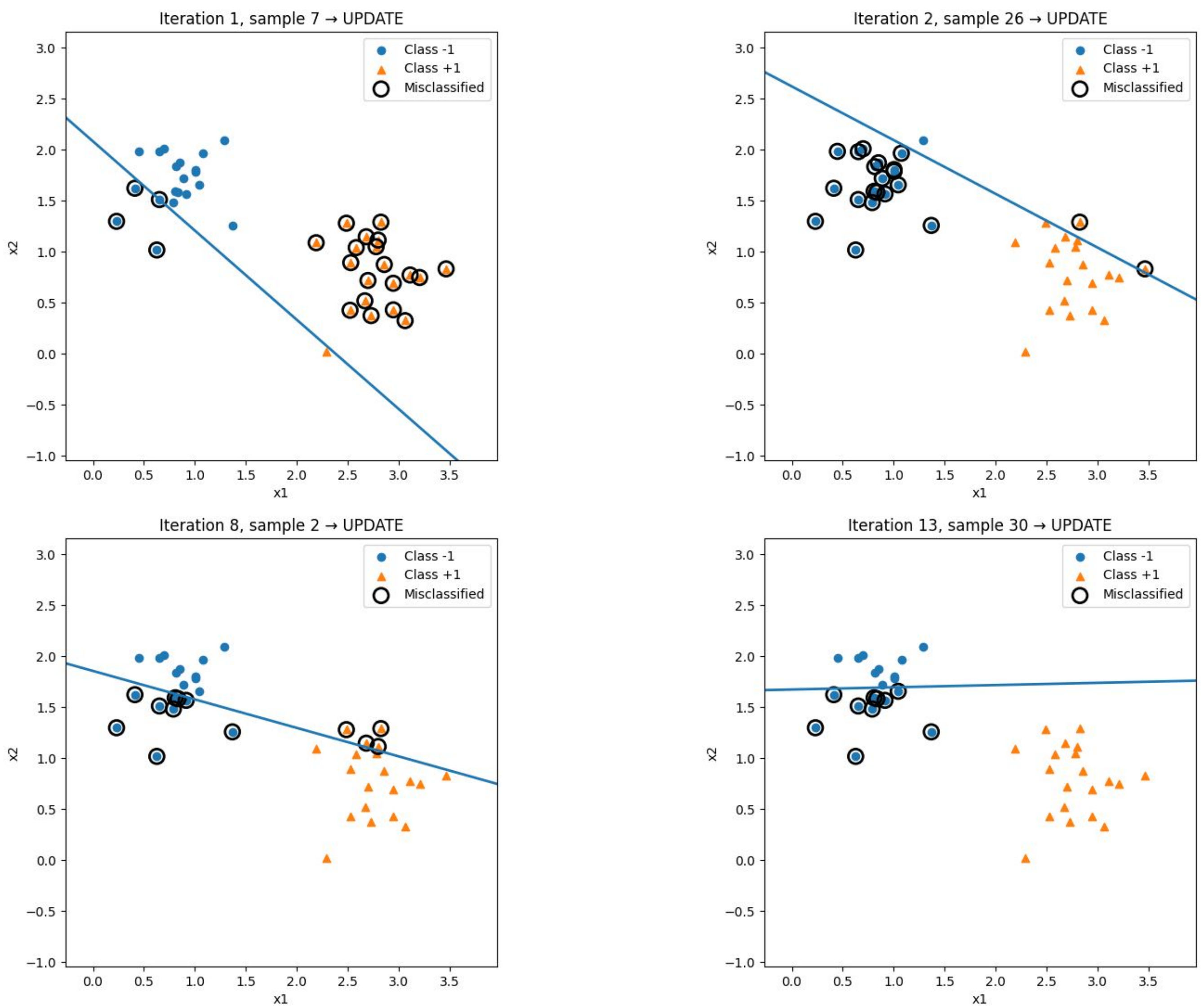
$$\boldsymbol{w}^{(n+1)} \leftarrow \boldsymbol{w}^{(n)} + \eta \cdot e^{(n)} \boldsymbol{x}^{(n)}$$

increment n
end

Imagine that you were a student back in 1958, would you be able to discover this simple algorithm as Rosenblatt did?

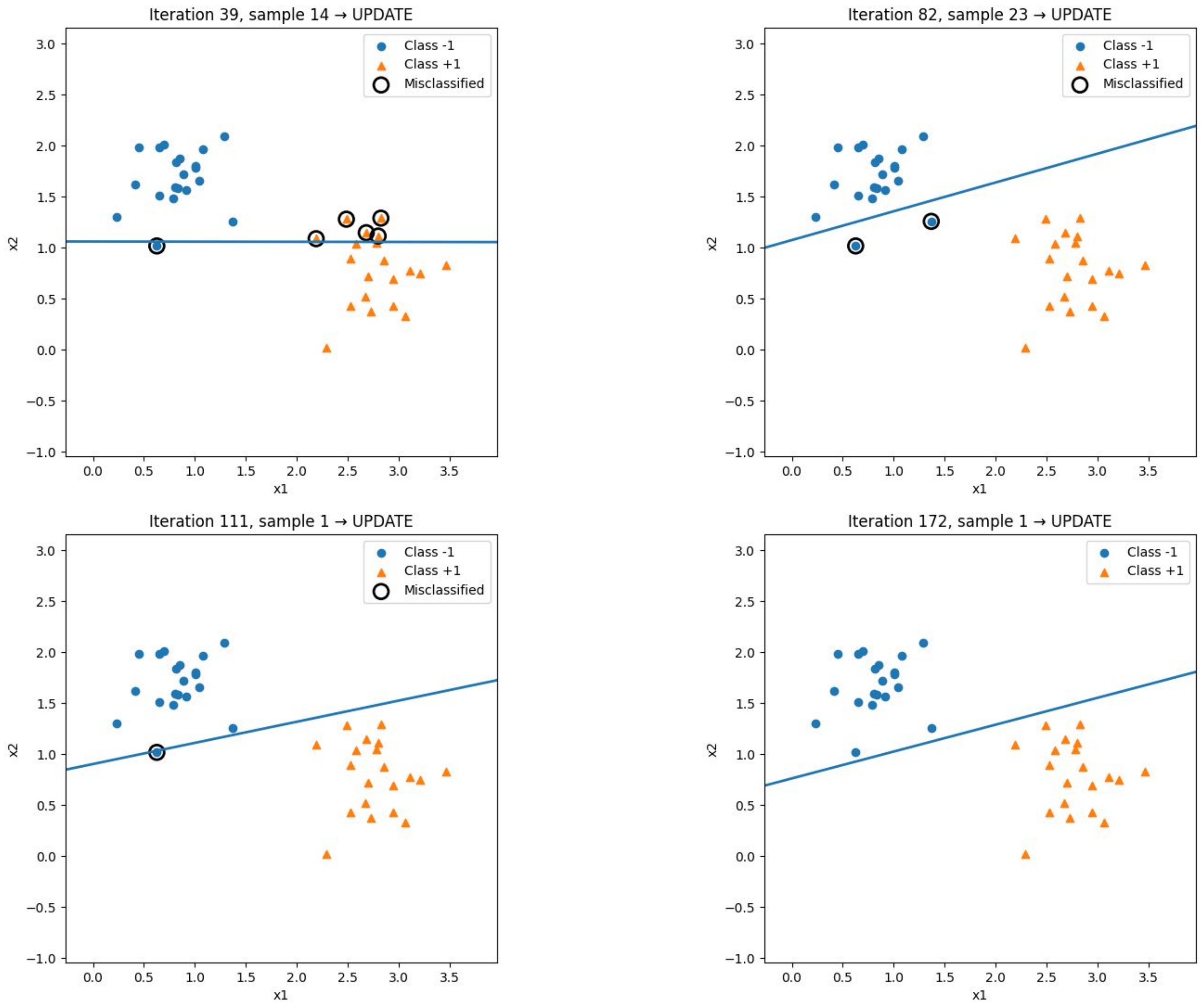
Perceptron learning algorithm: an example

There are two classes of patterns surrounding (0.75, 1.75) and (2.75, 0.75) as shown in the following figures



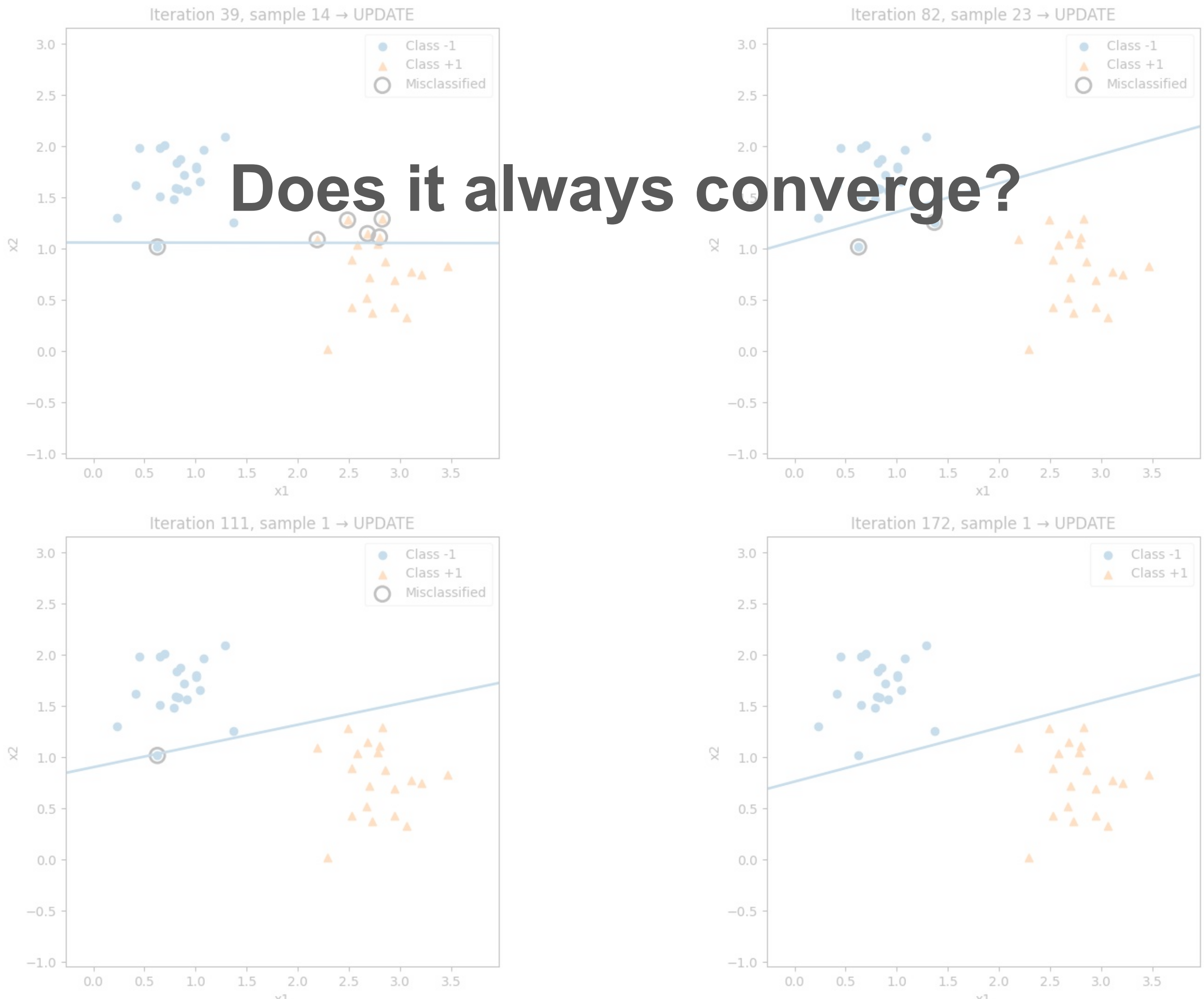
Perceptron learning algorithm: an example

There are two classes of patterns surrounding (0.75, 1.75) and (2.75, 0.75) as shown in the following figures



Perceptron learning algorithm: an example

There are two classes of patterns surrounding (0.75, 1.75) and (2.75, 0.75) as shown in the following figures



Does it always converge?

A short break

We will be back in 5 mins

Perceptron Convergence Theorem (Rosenblatt, 1962)

Theorem

If two classes C_1 and C_2 are linearly separable, then after a **finite number** of perceptron learning steps, the perceptron will **correctly classifies all data points** in the training set.

Perceptron Convergence Theorem (Rosenblatt, 1962)

Theorem
If two classes C_1 and C_2 are linearly separable, then after a **finite number** of perceptron learning steps, the perceptron will **correctly classifies all data points** in the training set.

How to prove it?

Perceptron Convergence Theorem (Rosenblatt, 1962)

Theorem
If two classes C_1 and C_2 are linearly separable, then after a **finite number** of perceptron learning steps, the perceptron will **correctly classifies all data points** in the training set.

How to prove it?

It may be hard to prove that after a finite number of steps, the perceptron classification results will be all correct.

Perceptron Convergence Theorem (Rosenblatt, 1962)

Theorem
If two classes C_1 and C_2 are linearly separable, then after a **finite number** of perceptron learning steps, the perceptron will **correctly classifies all data points** in the training set.

How to prove it?

It may be hard to prove that after a finite number of steps, the perceptron classification results will be all correct.

But, if the weights stop changing after a finite number of steps, does it imply that the perceptron can classify all the data points correctly?

Perceptron Convergence Theorem (Rosenblatt, 1962)

Theorem
If two classes C_1 and C_2 are linearly separable, then after a **finite number** of perceptron learning steps, the perceptron will **correctly classifies all data points** in the training set.

How to prove it?

It may be hard to prove that after a finite number of steps, the perceptron classification results will be all correct.

But, if the weights stop changing after a finite number of steps, does it imply that the perceptron can classify all the data points correctly?
Yes**.**

Remember the perceptron weights are updated if and only if there is still misclassification!

Perceptron Convergence Theorem (Rosenblatt, 1962)

We only prove the case where $\boldsymbol{w}^{(1)} = [0, \ldots, 0]^T, \eta = 1$

Perceptron Convergence Theorem (Rosenblatt, 1962)

We only prove the case where $\boldsymbol{w}^{(1)} = [0, \ldots, 0]^T, \eta = 1$

Proof. Since $\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)}$ and $\boldsymbol{w}^{(1)} = \mathbf{0}$

We have
$$\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + e^{(1)}\boldsymbol{x}^{(1)} = e^{(1)}\boldsymbol{x}^{(1)}$$
$$\boldsymbol{w}^{(3)} = \boldsymbol{w}^{(2)} + e^{(2)}\boldsymbol{x}^{(2)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

We only prove the case where $\boldsymbol{w}^{(1)} = [0, \ldots, 0]^T, \eta = 1$

Proof. Since $\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)}$ and $\boldsymbol{w}^{(1)} = \mathbf{0}$

We have
$$\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + e^{(1)}\boldsymbol{x}^{(1)} = e^{(1)}\boldsymbol{x}^{(1)}$$
$$\boldsymbol{w}^{(3)} = \boldsymbol{w}^{(2)} + e^{(2)}\boldsymbol{x}^{(2)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)}$$

So

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)} + \ldots + e^{(n)}\boldsymbol{x}^{(n)}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

We only prove the case where $\boldsymbol{w}^{(1)} = [0, \ldots, 0]^T, \eta = 1$

Proof. Since $\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)}$ and $\boldsymbol{w}^{(1)} = \boldsymbol{0}$

We have
$$\begin{aligned}\boldsymbol{w}^{(2)} &= \boldsymbol{w}^{(1)} + e^{(1)}\boldsymbol{x}^{(1)} = e^{(1)}\boldsymbol{x}^{(1)} \\ \boldsymbol{w}^{(3)} &= \boldsymbol{w}^{(2)} + e^{(2)}\boldsymbol{x}^{(2)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)}\end{aligned}$$

So

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)} + \ldots + e^{(n)}\boldsymbol{x}^{(n)}$$

Let's first find the lower bound of $||\boldsymbol{w}^{(n)}||$

Perceptron Convergence Theorem (Rosenblatt, 1962)

We only prove the case where $\boldsymbol{w}^{(1)} = [0, \ldots, 0]^T, \eta = 1$

Proof. Since $\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)} \boldsymbol{x}^{(n)}$ and $\boldsymbol{w}^{(1)} = \mathbf{0}$

We have
$$\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + e^{(1)}\boldsymbol{x}^{(1)} = e^{(1)}\boldsymbol{x}^{(1)}$$
$$\boldsymbol{w}^{(3)} = \boldsymbol{w}^{(2)} + e^{(2)}\boldsymbol{x}^{(2)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)}$$

So

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)} = e^{(1)}\boldsymbol{x}^{(1)} + e^{(2)}\boldsymbol{x}^{(2)} + \ldots + e^{(n)}\boldsymbol{x}^{(n)}$$

Let's first find the lower bound of $||\boldsymbol{w}^{(n)}||$

Since two classes C_1 and C_2 are linearly separable, there must exist a perceptron with weight $\boldsymbol{w}_0$ that can correctly classify all data points.

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Note the algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified. So all the terms in the above summation are misclassified terms.

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Note the algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified. So all the terms in the above summation are misclassified terms.

If $\boldsymbol{w}_0^T \boldsymbol{x}^{(i)} > 0 \Rightarrow {y^*}^{(i)} = 1$ (the definition of $\boldsymbol{w}_0$)

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Note the algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified. So all the terms in the above summation are misclassified terms.

If $\boldsymbol{w}_0^T \boldsymbol{x}^{(i)} > 0 \Rightarrow y^{*(i)} = 1$ (the definition of $\boldsymbol{w}_0$)

$\Rightarrow y^{(i)} = 0$ (misclassified!)

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Note the algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified. So all the terms in the above summation are misclassified terms.

If $\boldsymbol{w}_0^T \boldsymbol{x}^{(i)} > 0 \Rightarrow y^{*(i)} = 1$ (the definition of $\boldsymbol{w}_0$)

$\Rightarrow y^{(i)} = 0$ (misclassified!)

$\Rightarrow e^{(i)} = y^{*(i)} - y^{(i)} = 1$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Note the algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified. So all the terms in the above summation are misclassified terms.

If $\boldsymbol{w}_0^T \boldsymbol{x}^{(i)} > 0 \Rightarrow y^{*(i)} = 1$ (the definition of $\boldsymbol{w}_0$)

$\Rightarrow y^{(i)} = 0$ (misclassified!)

$\Rightarrow e^{(i)} = y^{*(i)} - y^{(i)} = 1$

$\Rightarrow e^{(i)} \boldsymbol{w}_0^T \boldsymbol{x}^{(i)} = |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}| > 0$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Multiply both sides with $\boldsymbol{w}_0^T$:

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

Note the algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified. So all the terms in the above summation are misclassified terms.

If $\boldsymbol{w}_0^T \boldsymbol{x}^{(i)} > 0 \Rightarrow y^{*(i)} = 1$ (the definition of $\boldsymbol{w}_0$)

$\Rightarrow y^{(i)} = 0$ (misclassified!)

$\Rightarrow e^{(i)} = y^{*(i)} - y^{(i)} = 1$

$\Rightarrow e^{(i)} \boldsymbol{w}_0^T \boldsymbol{x}^{(i)} = |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}| > 0$

If $\boldsymbol{w}_0^T \boldsymbol{x}^{(i)} < 0 \Rightarrow y^{*(i)} = 0$ (the definition of $\boldsymbol{w}_0$)

$\Rightarrow y^{(i)} = 1$ (misclassified!)

$\Rightarrow e^{(i)} = y^{*(i)} - y^{(i)} = -1$

$\Rightarrow e^{(i)} \boldsymbol{w}_0^T \boldsymbol{x}^{(i)} = |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}| > 0$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

Then $$\begin{aligned}\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} &= e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)} \\ &= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|\end{aligned}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

Then $$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

$$= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$$

What is the lower bound of the absolute values of all $|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$:

$$\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

Then $\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$

$$= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$$

What is the lower bound of the absolute values of all $|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$:

$$\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$$

Is it possible that $\alpha = 0$?

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

Then $$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

$$= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$$

What is the lower bound of the absolute values of all $|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$:

$$\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$$

<u>Is it possible that $\alpha = 0$?</u>

No, we assume that no samples lie on the boundary, weight $\boldsymbol{w}_0$ that can linearly separate and correctly classify all data points.

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

Then $$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$

$$= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$$

What is the lower bound of the absolute values of all $|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$:

$$\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$$

Is it possible that $\alpha = 0$?

No, we assume that no samples lie on the boundary, weight $\boldsymbol{w}_0$ that can linearly separate and correctly classify all data points.

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} \geq n\alpha > 0$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Then $$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$$
$$= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$$

What is the lower bound of the absolute values of all $|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$:

$$\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$$

Is it possible that $\alpha = 0$?
No, we assume that no samples lie on the boundary, weight $\boldsymbol{w}_0$ that can linearly separate and correctly classify all data points.

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} \geq n\alpha > 0$$

Cauchy-Schwarz Inequality:

$$||\boldsymbol{w}_0|| \cdot ||\boldsymbol{w}^{(n+1)}|| \geq |\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)}| \geq n\alpha$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Then $\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} = e^{(1)} \boldsymbol{w}_0^T \boldsymbol{x}^{(1)} + e^{(2)} \boldsymbol{w}_0^T \boldsymbol{x}^{(2)} + \ldots + e^{(n)} \boldsymbol{w}_0^T \boldsymbol{x}^{(n)}$

$$= \sum_{i=1}^{n} |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$$

What is the lower bound of the absolute values of all $|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|$:

$$\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$$

Is it possible that $\alpha = 0$?
No, we assume that no samples lie on the boundary, weight $\boldsymbol{w}_0$ that can linearly separate and correctly classify all data points.

$$\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)} \geq n\alpha > 0$$

Cauchy-Schwarz Inequality:

$$||\boldsymbol{w}_0|| \cdot ||\boldsymbol{w}^{(n+1)}|| \geq |\boldsymbol{w}_0^T \boldsymbol{w}^{(n+1)}| \geq n\alpha$$

$$||\boldsymbol{w}^{(n+1)}|| \geq \frac{n\alpha}{||\boldsymbol{w}_0||}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

$$\begin{aligned}\|\boldsymbol{w}^{(n+1)}\|^2 &= \boldsymbol{w}^{(n+1)^T}\boldsymbol{w}^{(n+1)} = (\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})^T(\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)}) \\ &= \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2\end{aligned}$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \boldsymbol{w}^{(n+1)^T}\boldsymbol{w}^{(n+1)} = (\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})^T(\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})$$
$$= \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2$$

Again, algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified.
Suppose $e^{(i)}$ is non-zero:

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \boldsymbol{w}^{(n+1)^T}\boldsymbol{w}^{(n+1)} = (\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})^T(\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})$$

$$= \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2$$

Again, algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified.
Suppose $e^{(i)}$ is non-zero:

If $\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} \geq 0 \Rightarrow y^{(n)} = 1$ (current prediction)

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \boldsymbol{w}^{(n+1)^T}\boldsymbol{w}^{(n+1)} = (\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})^T(\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})$$
$$= \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2$$

Again, algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified.
Suppose $e^{(i)}$ is non-zero:

If $\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} \geq 0 \Rightarrow y^{(n)} = 1$ (current prediction)

$\Rightarrow y^{*(n)} = 0$ (misclassified!)

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \boldsymbol{w}^{(n+1)^T}\boldsymbol{w}^{(n+1)} = (\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})^T(\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})$$

$$= \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2$$

Again, algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified.
Suppose $e^{(i)}$ is non-zero:

If $\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} \geq 0 \Rightarrow y^{(n)} = 1$ (current prediction)

$\Rightarrow y^{*(n)} = 0$ (misclassified!)

$\Rightarrow e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} \leq 0$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).
Let's then find the upper bound of $||\boldsymbol{w}^{(n)}||$

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \boldsymbol{w}^{(n+1)^T}\boldsymbol{w}^{(n+1)} = (\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})^T(\boldsymbol{w}^{(n)} + e^{(n)}\boldsymbol{x}^{(n)})$$
$$= \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2$$

Again, algorithm iterates only when $\boldsymbol{x}^{(i)}$ is misclassified.
Suppose $e^{(i)}$ is non-zero:

If $\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} \geq 0 \Rightarrow y^{(n)} = 1$ (current prediction)
$\Rightarrow y^{*(n)} = 0$ (misclassified!)
$\Rightarrow e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} \leq 0$

If $\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} < 0 \Rightarrow y^{(n)} = 0$ (current prediction)
$\Rightarrow y^{*(n)} = 1$ (misclassified!)
$\Rightarrow e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} < 0$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2 \leq \|\boldsymbol{w}^{(n)}\|^2 + \|\boldsymbol{x}^{(n)}\|^2$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2 \leq \|\boldsymbol{w}^{(n)}\|^2 + \|\boldsymbol{x}^{(n)}\|^2$$

Therefore:

$$\|\boldsymbol{w}^{(2)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 \leq \|\boldsymbol{x}^{(1)}\|^2$$

$$\|\boldsymbol{w}^{(3)}\|^2 - \|\boldsymbol{w}^{(2)}\|^2 \leq \|\boldsymbol{x}^{(2)}\|^2$$

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(n)}\|^2 \leq \|\boldsymbol{x}^{(n)}\|^2$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2 \leq \|\boldsymbol{w}^{(n)}\|^2 + \|\boldsymbol{x}^{(n)}\|^2$$

Therefore:

$$\|\boldsymbol{w}^{(2)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 \leq \|\boldsymbol{x}^{(1)}\|^2$$

$$\|\boldsymbol{w}^{(3)}\|^2 - \|\boldsymbol{w}^{(2)}\|^2 \leq \|\boldsymbol{x}^{(2)}\|^2$$

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(n)}\|^2 \leq \|\boldsymbol{x}^{(n)}\|^2$$

Summing up all the inequalities:

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 = \|\boldsymbol{w}^{(n+1)}\|^2 \leq \sum_{i=1}^{n} \|\boldsymbol{x}^{(i)}\|^2$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2 \leq \|\boldsymbol{w}^{(n)}\|^2 + \|\boldsymbol{x}^{(n)}\|^2$$

Therefore:

$$\|\boldsymbol{w}^{(2)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 \leq \|\boldsymbol{x}^{(1)}\|^2$$

$$\|\boldsymbol{w}^{(3)}\|^2 - \|\boldsymbol{w}^{(2)}\|^2 \leq \|\boldsymbol{x}^{(2)}\|^2$$

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(n)}\|^2 \leq \|\boldsymbol{x}^{(n)}\|^2$$

Summing up all the inequalities:

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 = \|\boldsymbol{w}^{(n+1)}\|^2 \leq \sum_{i=1}^{n} \|\boldsymbol{x}^{(i)}\|^2$$

Let $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

$$\|\boldsymbol{w}^{(n+1)}\|^2 = \|\boldsymbol{w}^{(n)}\|^2 + 2e^{(n)}\boldsymbol{w}^{(n)T}\boldsymbol{x}^{(n)} + \|\boldsymbol{x}^{(n)}\|^2 \leq \|\boldsymbol{w}^{(n)}\|^2 + \|\boldsymbol{x}^{(n)}\|^2$$

Therefore:

$$\|\boldsymbol{w}^{(2)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 \leq \|\boldsymbol{x}^{(1)}\|^2$$

$$\|\boldsymbol{w}^{(3)}\|^2 - \|\boldsymbol{w}^{(2)}\|^2 \leq \|\boldsymbol{x}^{(2)}\|^2$$

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(n)}\|^2 \leq \|\boldsymbol{x}^{(n)}\|^2$$

Summing up all the inequalities:

$$\|\boldsymbol{w}^{(n+1)}\|^2 - \|\boldsymbol{w}^{(1)}\|^2 = \|\boldsymbol{w}^{(n+1)}\|^2 \leq \sum_{i=1}^{n} \|\boldsymbol{x}^{(i)}\|^2$$

Let $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$

Then $\|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

On the one hand $$||\boldsymbol{w}^{(n+1)}|| \geq \frac{n\alpha}{||\boldsymbol{w}_0||}$$

On the other hand $$\|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

On the one hand $$||\boldsymbol{w}^{(n+1)}|| \geq \frac{n\alpha}{||\boldsymbol{w}_0||}$$

On the other hand $$\|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Put these two inequalities together: $$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

On the one hand $$||\boldsymbol{w}^{(n+1)}|| \geq \frac{n\alpha}{||\boldsymbol{w}_0||}$$

On the other hand $$\|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Put these two inequalities together: $$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

If the algorithm never stops and n continue to increase (to infinity), can the inequality always hold?

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

On the one hand $$||\boldsymbol{w}^{(n+1)}|| \geq \frac{n\alpha}{||\boldsymbol{w}_0||}$$

On the other hand $$\|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Put these two inequalities together: $$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

If the algorithm never stops and n continue to increase (to infinity), can the inequality always hold?

No.

Perceptron Convergence Theorem (Rosenblatt, 1962)

Proof (continued).

On the one hand $$||\boldsymbol{w}^{(n+1)}|| \geq \frac{n\alpha}{||\boldsymbol{w}_0||}$$

On the other hand $$\|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Put these two inequalities together: $$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

<u>If the algorithm never stops and n continue to increase (to infinity), can the inequality always hold?</u>

No.

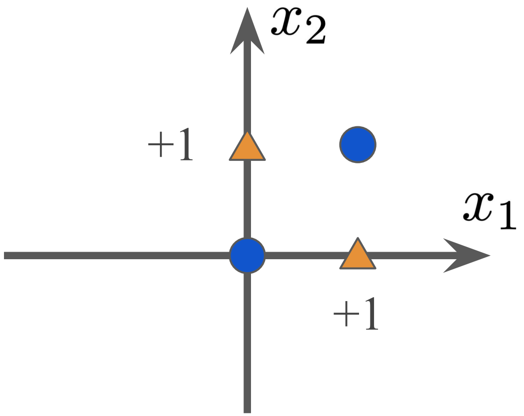
So the algorithm will stop in a finite number of step!

After a finite number of steps, perceptron weights will stop changing, which implies that the perceptron will classify all the data points correctly.

$\square$

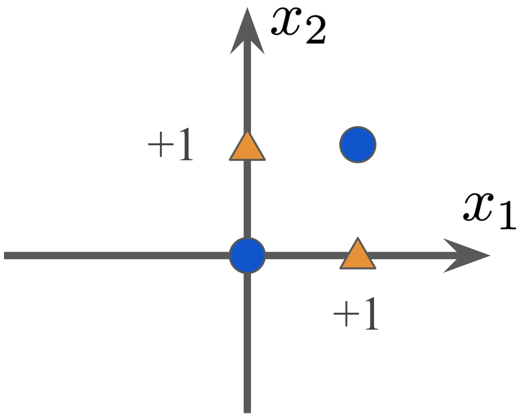
Perceptron learning

What would happen if the patterns are not linearly separable?



Perceptron learning

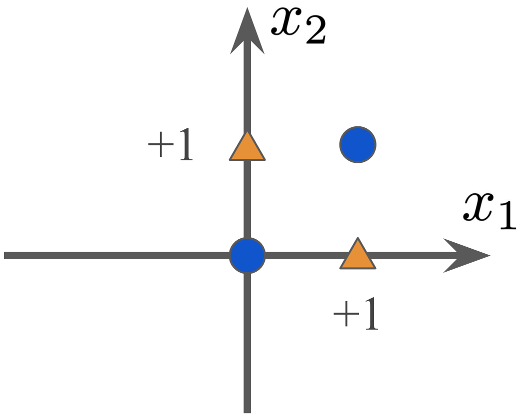
What would happen if the patterns are not linearly separable?



<u>Will the algorithm stop to update the weights of the perceptron?</u>

Perceptron learning

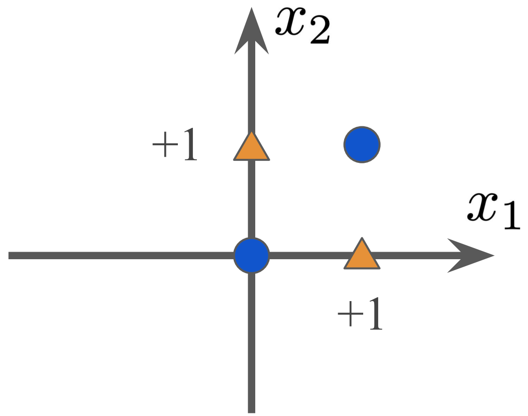
What would happen if the patterns are not linearly separable?



<u>Will the algorithm stop to update the weights of the perceptron?</u>
No. There will always be at least one misclassified data point.

Perceptron learning

What would happen if the patterns are not linearly separable?

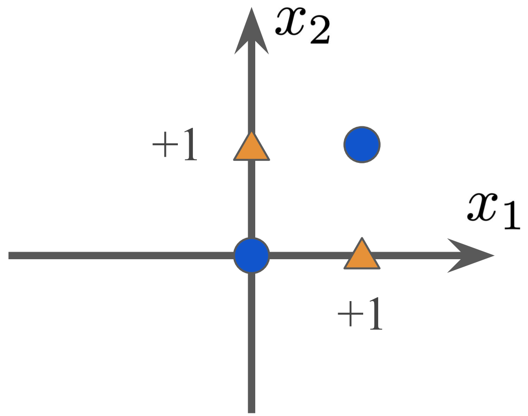


Will the algorithm stop to update the weights of the perceptron?
No. There will always be at least one misclassified data point.

In real-world problems, the dimension of the data is usually very high, how to determine whether the problem is linearly separable?

Perceptron learning

What would happen if the patterns are not linearly separable?

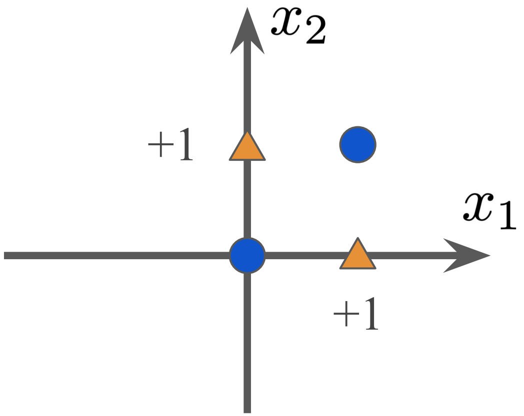


<u>Will the algorithm stop to update the weights of the perceptron?</u>
No. There will always be at least one misclassified data point.

<u>In real-world problems, the dimension of the data is usually very high, how to determine whether the problem is linearly separable?</u>
Just run the perceptron learning algorithm!
If it converges, then it must be linearly separable.

Perceptron learning

What would happen if the patterns are not linearly separable?



<u>**Will the algorithm stop to update the weights of the perceptron?**</u>
No. There will always be at least one misclassified data point.

<u>**In real-world problems, the dimension of the data is usually very high, how to determine whether the problem is linearly separable?**</u>
Just run the perceptron learning algorithm!
If it converges, then it must be linearly separable.

What if the algorithm does not stop after A LOT OF updates?
Can we safely say the problem is not linearly separable?

Perceptron learning

Recall definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$ $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$

If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Perceptron learning

Recall definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\} \quad \beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$
If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

This means $n \leq \dfrac{\|\boldsymbol{w}_0\|^2\beta}{\alpha^2} = \dfrac{\|\boldsymbol{w}_0\|^2 \cdot \max_i \|\boldsymbol{x}^{(i)}\|^2}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|^2}$

$$= \left(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}| / \|\boldsymbol{w}_0\|}\right)^2$$

Perceptron learning

Recall definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$ $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$
If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

This means $n \leq \frac{\|\boldsymbol{w}_0\|^2\beta}{\alpha^2} = \frac{\|\boldsymbol{w}_0\|^2 \cdot \max_i \|\boldsymbol{x}^{(i)}\|^2}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|^2}$

$$= \Big(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|/\|\boldsymbol{w}_0\|}\Big)^2 = \Big(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\Big)^2$$

What is $\frac{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \min_i \frac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$?

Perceptron learning

Recall definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$ $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$
If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

This means $n \leq \frac{\|\boldsymbol{w}_0\|^2\beta}{\alpha^2} = \frac{\|\boldsymbol{w}_0\|^2 \cdot \max_i \|\boldsymbol{x}^{(i)}\|^2}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|^2}$

$$= \Big(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|/\|\boldsymbol{w}_0\|}\Big)^2 = \Big(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\Big)^2$$

What is $\frac{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \min_i \frac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$?

Recall 2D case:
The perpendicular distance from point (x_0, y_0) *to line* $ax + by = 0$ *is*

$$d = \frac{|ax_0 + by_0|}{\sqrt{a^2 + b^2}}$$

Perceptron learning

$\frac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$ is the perpendicular distance from $\boldsymbol{x}^{(i)}$ to the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

Perceptron learning

$\dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$ is the perpendicular distance from $\boldsymbol{x}^{(i)}$ to the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

$\min\limits_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$ is the minimum distance from all data samples to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

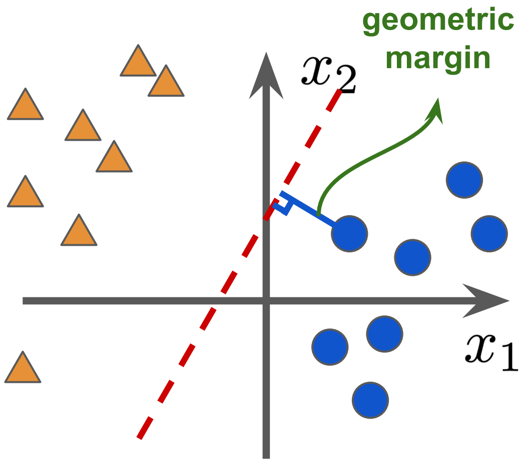
Perceptron learning

$\dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$ is the perpendicular distance from $\boldsymbol{x}^{(i)}$ to the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

$\min\limits_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$ is the minimum distance from all data samples to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

In other words, γ is the geometric margin of the dataset with respect to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

(Data samples are not only correctly separated, but with a margin of at least γ to the decision boundary hyperplane)

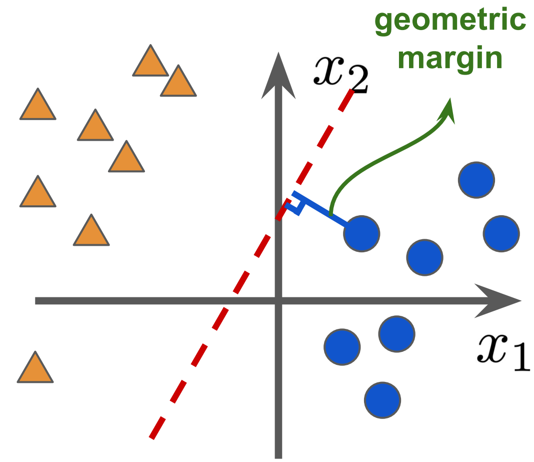


Perceptron learning

$\dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$ is the perpendicular distance from $\boldsymbol{x}^{(i)}$ to the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

$\min\limits_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$ is the minimum distance from all data samples to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

In other words, γ is the geometric margin of the dataset with respect to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$
(Data samples are not only correctly separated, but with a margin of at least γ to the decision boundary hyperplane)



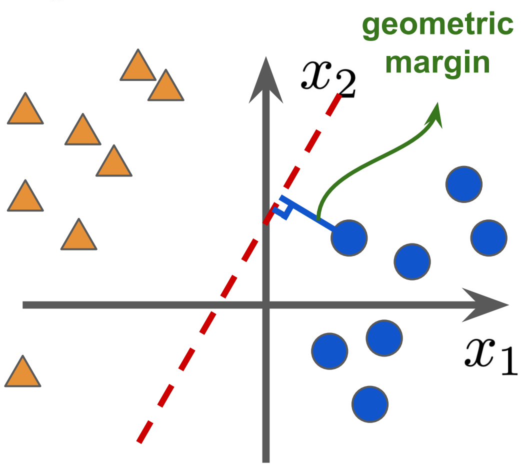
We then have $$n \leq \left(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2 \Rightarrow \gamma \leq \frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\sqrt{n}}$$

Perceptron learning

$\dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$ is the perpendicular distance from $\boldsymbol{x}^{(i)}$ to the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

$\min\limits_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$ is the minimum distance from all data samples to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

In other words, γ is the geometric margin of the dataset with respect to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$
(Data samples are not only correctly separated, but with a margin of at least γ to the decision boundary hyperplane)



We then have $\quad n \leq \left(\dfrac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2 \Rightarrow \gamma \leq \dfrac{\max_i \|\boldsymbol{x}^{(i)}\|}{\sqrt{n}}$

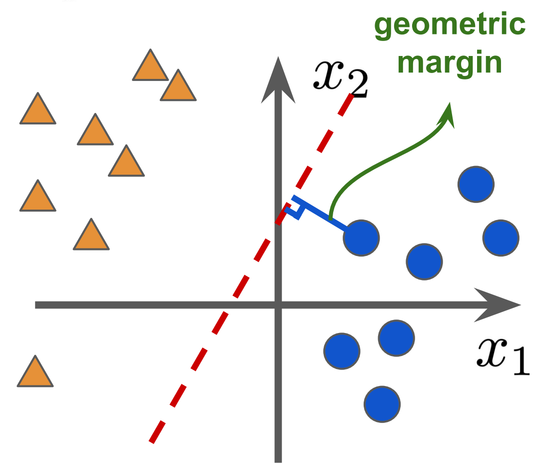
Do these inequalities hold for all possible geometric margins on that dataset, including the ***largest achievable margin****?*

Perceptron learning

$\dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$ is the perpendicular distance from $\boldsymbol{x}^{(i)}$ to the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

$\min_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$ is the minimum distance from all data samples to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$

In other words, γ is the geometric margin of the dataset with respect to the separating hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$
(Data samples are not only correctly separated, but with a margin of at least γ to the decision boundary hyperplane)



We then have $\quad n \leq \left(\dfrac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2 \Rightarrow \gamma \leq \dfrac{\max_i \|\boldsymbol{x}^{(i)}\|}{\sqrt{n}}$

*Do these inequalities hold for all possible geometric margins on that dataset, including the **largest achievable margin**?*
Yes.

Perceptron learning

Summary

If the data samples are linearly separable by margin of at least γ, the number of updates that perceptron learning algorithm has is at most

$$\left(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2$$

Perceptron learning

Summary

If the data samples are linearly separable by margin of at least γ, the number of updates that perceptron learning algorithm has is at most

$$\left(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2$$

In other words, if the perceptron learning algorithm does not end after n updates, it means the data samples are either 1) not linearly separable, or 2) they are linearly separable but the separation margin is less than

$$\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\sqrt{n}}$$

Perceptron learning

How about the choice of the initial weights $\boldsymbol{w}^{(1)}$? Will the perceptron converge if the initial weights are chosen randomly?

Perceptron learning

How about the choice of the initial weights $\boldsymbol{w}^{(1)}$? Will the perceptron converge if the initial weights are chosen randomly?

Yes. (think about why)

Perceptron learning

<u>How about the choice of the initial weights $\boldsymbol{w}^{(1)}$? Will the perceptron converge if the initial weights are chosen randomly?</u>
Yes. (think about why)

<u>How about the choice of the learning rate η? Will the perceptron converge with a different learning rate?</u>

Perceptron learning

How about the choice of the initial weights $\boldsymbol{w}^{(1)}$? Will the perceptron converge if the initial weights are chosen randomly?
Yes. (think about why)

How about the choice of the learning rate η? Will the perceptron converge with a different learning rate?
Yes. (think about why)

Perceptron learning

How about the choice of the initial weights $\boldsymbol{w}^{(1)}$? Will the perceptron converge if the initial weights are chosen randomly?
Yes. (think about why)

How about the choice of the learning rate η? Will the perceptron converge with a different learning rate?
Yes. (think about why)

Although η can be chosen to be any positive value, choosing it properly will decide how fast the learning algorithm converge.

Perceptron learning

How about the choice of the initial weights $\boldsymbol{w}^{(1)}$? Will the perceptron converge if the initial weights are chosen randomly?
Yes. (think about why)

How about the choice of the learning rate η? Will the perceptron converge with a different learning rate?
Yes. (think about why)

Although η can be chosen to be any positive value, choosing it properly will decide how fast the learning algorithm converge.

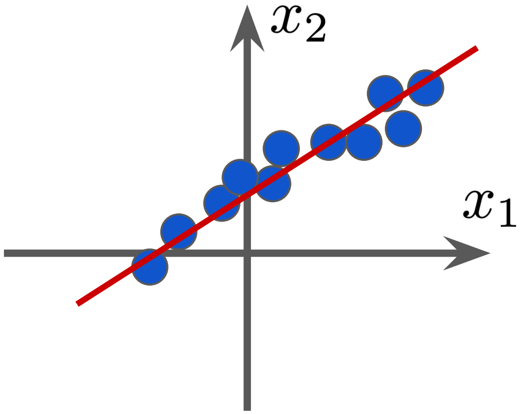
Too small: learning is slow for all data points.
Too large: learning is fast for the current data point, but will spoil the learning that has taken place earlier with respect to other data points.

So an intermediate value is the best! The choice is problem-dependent.

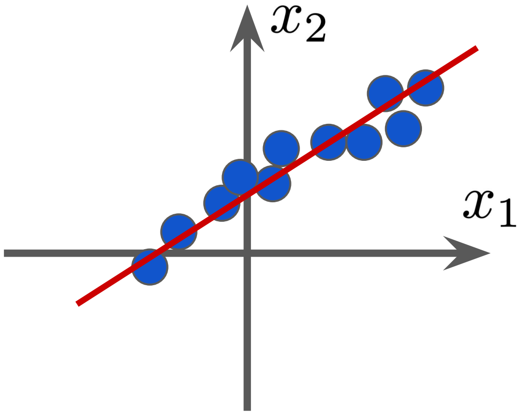
Perceptron for regression problems

Classification problems:
Predict discrete categories

Regression problems:
Predict continuous quantities



Perceptron for regression problems



Classification problems:
Predict discrete categories

Regression problems:
Predict continuous quantities

Applying perceptrons (**w/o activation function**) to regression problems:

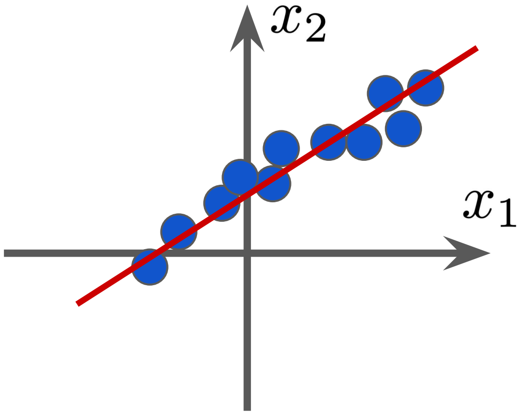
$$y = \boldsymbol{w}^T \boldsymbol{x}$$

Total prediction error is:

$$E(\boldsymbol{w}) = \sum_{i=1}^{n} \left[{y^*}^{(i)} - y^{(i)} \right]^2 = \sum_{i=1}^{n} \left[{y^*}^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)} \right]^2 = \sum_{i=1}^{n} {e^{(i)}}^2 = \boldsymbol{e}^T \boldsymbol{e}$$

where $\boldsymbol{e} = \boldsymbol{y}^* - X\boldsymbol{w}$ is the error signal, and $X = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ \vdots \\ {\boldsymbol{x}^{(n)}}^T \end{bmatrix}, \boldsymbol{y}^* = \begin{bmatrix} y_1^* \\ \vdots \\ y_n^* \end{bmatrix}$

Perceptron for regression problems



Classification problems:
Predict discrete categories

Regression problems:
Predict continuous quantities

Applying perceptrons (**<u>w/o activation function</u>**) to regression problems:

$$y = \boldsymbol{w}^T \boldsymbol{x}$$

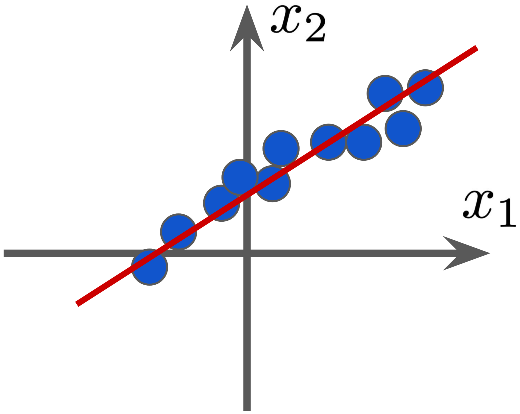
Total prediction error is:

$$E(\boldsymbol{w}) = \sum_{i=1}^{n} \left[y^{*(i)} - y^{(i)} \right]^2 = \sum_{i=1}^{n} \left[y^{*(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)} \right]^2 = \sum_{i=1}^{n} e^{(i)^2} = \boldsymbol{e}^T \boldsymbol{e}$$

where $\boldsymbol{e} = \boldsymbol{y}^* - X\boldsymbol{w}$ is the error signal, and $X = \begin{bmatrix} \boldsymbol{x}^{(1)^T} \\ \vdots \\ \boldsymbol{x}^{(n)^T} \end{bmatrix}, \boldsymbol{y}^* = \begin{bmatrix} y_1^* \\ \vdots \\ y_n^* \end{bmatrix}$

The goal is to find $\arg\min_{\boldsymbol{w}} E(\boldsymbol{w})$

Perceptron for regression problems



Classification problems:
Predict discrete categories

Regression problems:
Predict continuous quantities

Applying perceptrons (**w/o activation function**) to regression problems:

$$y = \boldsymbol{w}^T \boldsymbol{x}$$

Total prediction error is:

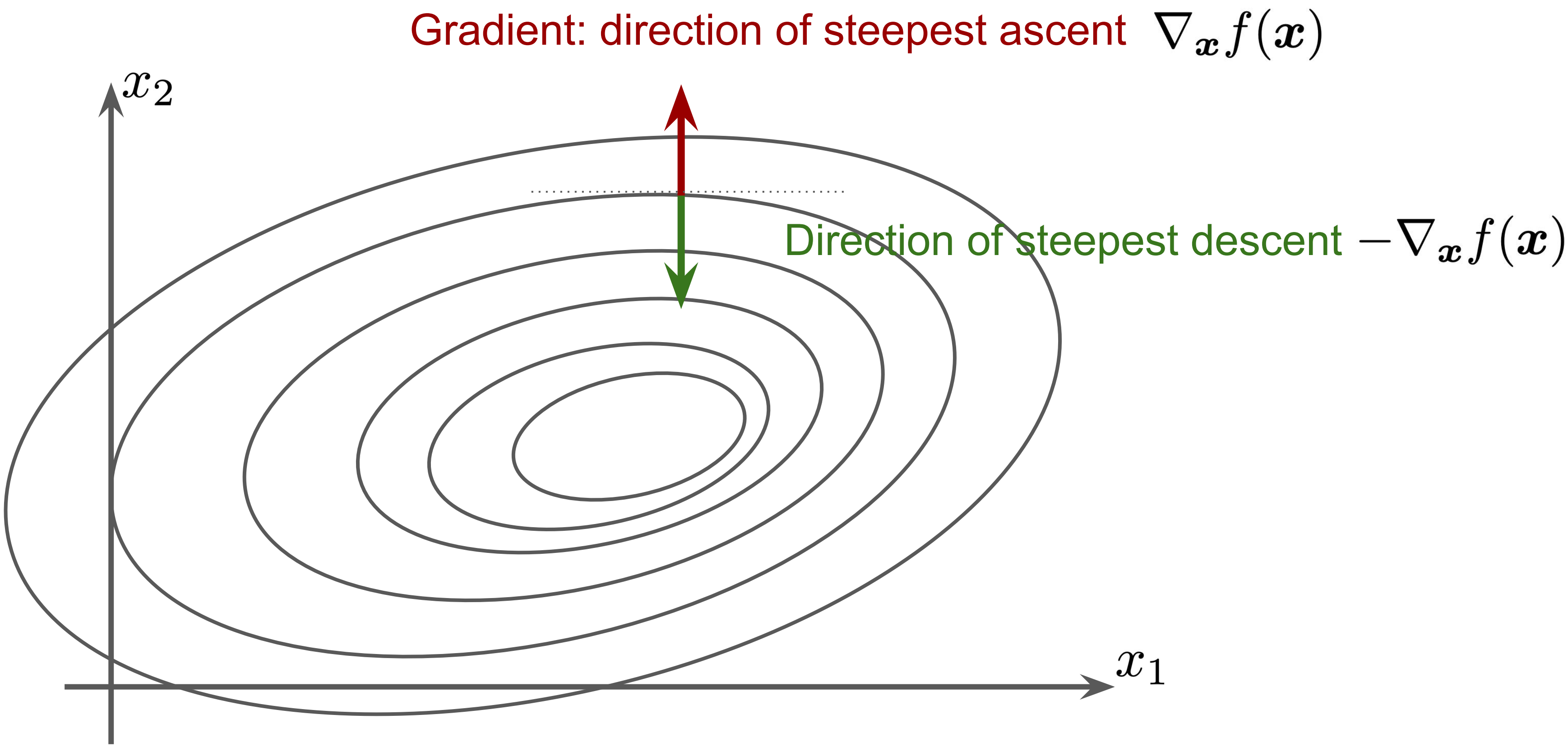
$$E(\boldsymbol{w}) = \sum_{i=1}^{n} \left[y^{*(i)} - y^{(i)} \right]^2 = \sum_{i=1}^{n} \left[y^{*(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)} \right]^2 = \sum_{i=1}^{n} e^{(i)^2} = \boldsymbol{e}^T \boldsymbol{e}$$

where $\boldsymbol{e} = \boldsymbol{y}^* - X\boldsymbol{w}$ is the error signal, and $X = \begin{bmatrix} \boldsymbol{x}^{(1)^T} \\ \vdots \\ \boldsymbol{x}^{(n)^T} \end{bmatrix}, \boldsymbol{y}^* = \begin{bmatrix} y_1^* \\ \vdots \\ y_n^* \end{bmatrix}$

The goal is to find $\arg\min_{\boldsymbol{w}} E(\boldsymbol{w})$

When is $E(\boldsymbol{w})$ minimized?

Derivatives: two-dimensional example



1. When is $f(\boldsymbol{x})$ minimized? When its derivatives is zero!
2. Optimization can be done by moving $\boldsymbol{x}$ in the opposite direction of gradient!

Some basics of matrix calculus

Given a vector-valued function $F : \mathbb{R}^m \to \mathbb{R}^n, F(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \dots, \\ f_n(\boldsymbol{x}) \end{bmatrix}$

Its derivative w.r.t. $\boldsymbol{x}$ is defined as its Jacobian:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} \nabla_{\boldsymbol{x}} f_1(\boldsymbol{x}) \\ \nabla_{\boldsymbol{x}} f_2(\boldsymbol{x}) \\ \dots, \\ \nabla_{\boldsymbol{x}} f_n(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

Some basics of matrix calculus

Given a vector-valued function $F : \mathbb{R}^m \rightarrow \mathbb{R}^n, F(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \dots, \\ f_n(\boldsymbol{x}) \end{bmatrix}$

Its derivative w.r.t. $\boldsymbol{x}$ is defined as its Jacobian:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} \nabla_{\boldsymbol{x}} f_1(\boldsymbol{x}) \\ \nabla_{\boldsymbol{x}} f_2(\boldsymbol{x}) \\ \dots, \\ \nabla_{\boldsymbol{x}} f_n(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

Special case: if $F(\boldsymbol{x})$ is a scalar-valued function, $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ becomes a row vector!

Example 1:

$$\nabla_{\boldsymbol{x}} \boldsymbol{x} = \begin{bmatrix} \frac{\partial x_1}{\partial x_1} & \cdots & \frac{\partial x_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_m}{\partial x_1} & \cdots & \frac{\partial x_m}{\partial x_m} \end{bmatrix} = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} = I_m$$

Some basics of matrix calculus

Product rule:

$$\nabla_{\boldsymbol{x}}(A(\boldsymbol{x})^T \cdot B(\boldsymbol{x})) = (\nabla_{\boldsymbol{x}} A(\boldsymbol{x}))^T \cdot B(\boldsymbol{x}) + (\nabla_{\boldsymbol{x}} B(\boldsymbol{x}))^T \cdot A(\boldsymbol{x})$$

Some basics of matrix calculus

Product rule:

$$\nabla_{\boldsymbol{x}}(A(\boldsymbol{x})^T \cdot B(\boldsymbol{x})) = (\nabla_{\boldsymbol{x}} A(\boldsymbol{x}))^T \cdot B(\boldsymbol{x}) + (\nabla_{\boldsymbol{x}} B(\boldsymbol{x}))^T \cdot A(\boldsymbol{x})$$

Example 2:

$$\nabla_{\boldsymbol{x}}(\boldsymbol{x}^T \boldsymbol{x}) = (\nabla_{\boldsymbol{x}} \boldsymbol{x})^T \cdot \boldsymbol{x} + (\nabla_{\boldsymbol{x}} \boldsymbol{x})^T \cdot \boldsymbol{x} = I^T \boldsymbol{x} + I^T \boldsymbol{x} = 2\boldsymbol{x}$$

Some basics of matrix calculus

Product rule:

$$\nabla_{\boldsymbol{x}}(A(\boldsymbol{x})^T \cdot B(\boldsymbol{x})) = (\nabla_{\boldsymbol{x}} A(\boldsymbol{x}))^T \cdot B(\boldsymbol{x}) + (\nabla_{\boldsymbol{x}} B(\boldsymbol{x}))^T \cdot A(\boldsymbol{x})$$

Example 2:

$$\nabla_{\boldsymbol{x}}(\boldsymbol{x}^T \boldsymbol{x}) = (\nabla_{\boldsymbol{x}} \boldsymbol{x})^T \cdot \boldsymbol{x} + (\nabla_{\boldsymbol{x}} \boldsymbol{x})^T \cdot \boldsymbol{x} = I^T \boldsymbol{x} + I^T \boldsymbol{x} = 2\boldsymbol{x}$$

Chain rule:

$$\nabla_{\boldsymbol{x}} f(g(\boldsymbol{x})) = (\nabla_{\boldsymbol{x}} g(\boldsymbol{x}))^T \nabla_g f(g)$$

Standard Linear Least Squares

Goal: minimize $E(\boldsymbol{w}) = \boldsymbol{e}^T \boldsymbol{e}$

Standard Linear Least Squares

Goal: minimize $E(\boldsymbol{w}) = \boldsymbol{e}^T \boldsymbol{e}$

When $E(\boldsymbol{w})$ is minimized:

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \boldsymbol{e} \cdot \nabla_{\boldsymbol{e}} E \qquad \text{(chain rule)}$$

Standard Linear Least Squares

Goal: minimize $E(\boldsymbol{w}) = \boldsymbol{e}^T\boldsymbol{e}$

When $E(\boldsymbol{w})$ is minimized:

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \boldsymbol{e} \cdot \nabla_{\boldsymbol{e}} E \qquad \text{(chain rule)}$$

$$= \nabla_{\boldsymbol{w}} (\boldsymbol{y}^* - X\boldsymbol{w}) \cdot \nabla_{\boldsymbol{e}} \boldsymbol{e}^T \boldsymbol{e}$$

Standard Linear Least Squares

Goal: minimize $E(\boldsymbol{w}) = \boldsymbol{e}^T \boldsymbol{e}$

When $E(\boldsymbol{w})$ is minimized:

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \boldsymbol{e} \cdot \nabla_{\boldsymbol{e}} E \qquad \text{(chain rule)}$$

$$= \nabla_{\boldsymbol{w}} (\boldsymbol{y}^* - X\boldsymbol{w}) \cdot \nabla_{\boldsymbol{e}} \boldsymbol{e}^T \boldsymbol{e}$$

$$= -2X^T \boldsymbol{e} = 0 \qquad \text{(product rule)}$$

Standard Linear Least Squares

Goal: minimize $E(\boldsymbol{w}) = \boldsymbol{e}^T\boldsymbol{e}$

When $E(\boldsymbol{w})$ is minimized:

$$\begin{aligned}\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) &= \nabla_{\boldsymbol{w}}\boldsymbol{e} \cdot \nabla_{\boldsymbol{e}} E && \text{(chain rule)}\\ &= \nabla_{\boldsymbol{w}}(\boldsymbol{y}^* - X\boldsymbol{w}) \cdot \nabla_{\boldsymbol{e}}\boldsymbol{e}^T\boldsymbol{e} \\ &= -2X^T\boldsymbol{e} = 0 && \text{(product rule)}\end{aligned}$$

By definition of $\boldsymbol{e}$:

$$X^T\boldsymbol{e} = X^T(\boldsymbol{y}^* - X\boldsymbol{w}) = 0$$

Standard Linear Least Squares

Goal: minimize $E(\boldsymbol{w}) = \boldsymbol{e}^T\boldsymbol{e}$

When $E(\boldsymbol{w})$ is minimized:

$$\begin{aligned}\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) &= \nabla_{\boldsymbol{w}}\boldsymbol{e} \cdot \nabla_{\boldsymbol{e}} E && \text{(chain rule)}\\ &= \nabla_{\boldsymbol{w}}(\boldsymbol{y}^* - X\boldsymbol{w}) \cdot \nabla_{\boldsymbol{e}}\boldsymbol{e}^T\boldsymbol{e} \\ &= -2X^T\boldsymbol{e} = 0 && \text{(product rule)}\end{aligned}$$

By definition of $\boldsymbol{e}$:

$$X^T\boldsymbol{e} = X^T(\boldsymbol{y}^* - X\boldsymbol{w}) = 0$$

If X^TX is an invertible matrix (which is very likely):

$$\boldsymbol{w} = (X^TX)^{-1}X^T\boldsymbol{y}^*$$

Directly solving $\boldsymbol{w}$ using this equation: **Linear Least Squares** method

Standard Linear Least Squares

What is the dimension of $X = \begin{bmatrix} \boldsymbol{x}^{(1)^T} \\ \vdots \\ \boldsymbol{x}^{(n)^T} \end{bmatrix}$ **?**

Standard Linear Least Squares

What is the dimension of $X = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ \vdots \\ {\boldsymbol{x}^{(n)}}^T \end{bmatrix}$ **?**

$n \times m$!

Standard Linear Least Squares

What is the dimension of $X = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ \vdots \\ {\boldsymbol{x}^{(n)}}^T \end{bmatrix}$ **?**

$n \times m$!

If we load all data to construct matrix X, we may run out of memory when the number of data samples is large!

Standard Linear Least Squares

What is the dimension of $X = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ \vdots \\ {\boldsymbol{x}^{(n)}}^T \end{bmatrix}$ **?**

$n \times m$!

If we load all data to construct matrix X , we may run out of memory when the number of data samples is large!

<u>Can we use on one / a small number of data samples every time?</u>

Standard Linear Least Squares

What is the dimension of $X = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ \vdots \\ {\boldsymbol{x}^{(n)}}^T \end{bmatrix}$ **?**

$n \times m$!

If we load all data to construct matrix X, we may run out of memory when the number of data samples is large!

<u>Can we use on one / a small number of data samples every time?</u>

Yes! The core idea is to use the gradient of the error signal of each data point w.r.t. weights to update the weights

$$E(\boldsymbol{w}) = \frac{1}{2}{e^{(i)}}^2 \qquad \nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = -e^{(i)} \boldsymbol{x}^{(i)}$$

instead of computing the gradient of the total error w.r.t. the weights

Least-Mean-Squares algorithm (1960)

Proposed by Widrow and Hoff at Stanford University in 1960

Least-Mean-Squares algorithm (1960)

Proposed by Widrow and Hoff at Stanford University in 1960

Notation: use (n) to denote variables in n-th iteration

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
for $i = 1, 2, \ldots, n$ **do**

$$e^{(i)} \leftarrow y^{(i)} - {\boldsymbol{w}^{(i)}}^T \boldsymbol{x}^{(i)}$$

$$w^{(i+1)} \leftarrow w^{(i)} + \eta e^{(i)} \boldsymbol{x}^{(i)}$$

end

Least-Mean-Squares algorithm (1960)

Proposed by Widrow and Hoff at Stanford University in 1960

Notation: use (n) to denote variables in n-th iteration

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
for $i = 1, 2, \ldots, n$ **do**

$$e^{(i)} \leftarrow y^{(i)} - {\boldsymbol{w}^{(i)}}^T \boldsymbol{x}^{(i)}$$

$$w^{(i+1)} \leftarrow w^{(i)} + \eta e^{(i)} \boldsymbol{x}^{(i)}$$

end

This is the same algorithm as the Perceptron Learning algorithm!

Least-Mean-Squares algorithm (1960)

Proposed by Widrow and Hoff at Stanford University in 1960

Notation: use (n) to denote variables in n-th iteration

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
for $i = 1, 2, \ldots, n$ **do**

$$e^{(i)} \leftarrow y^{(i)} - \boldsymbol{w}^{(i)^T} \boldsymbol{x}^{(i)}$$

$$w^{(i+1)} \leftarrow w^{(i)} + \eta e^{(i)} \boldsymbol{x}^{(i)}$$

end

This is the same algorithm as the Perceptron Learning algorithm!

Why aren't we able to derive the Perceptron Learning algorithm from the simple idea of gradient descent?
We cannot do it because the threshold activation function $\varphi(\cdot)$ does not have a non-zero gradient!

Why are we still studying the perceptron in neural network era?

Why are we still studying the perceptron in neural network era?

- It is still being actively used

Why are we still studying the perceptron in neural network era?

- It is still being actively used
  - For example, high-frequency trading
    - Explainable
    - Efficient

Why are we still studying the perceptron in neural network era?

- It is still being actively used
  - For example, high-frequency trading
    - Explainable
    - Efficient
- Reflecting on the past can shed light on the future.

Why are we still studying the perceptron in neural network era?

- It is still being actively used
  - For example, high-frequency trading
    - Explainable
    - Efficient
- Reflecting on the past can shed light on the future.
  - Imagine that you were a student back in 1958 or 1960, would you be able to discover the same algorithms as Rosenblatt, Widrow and Hoff did?

Why are we still studying the perceptron in neural network era?

- It is still being actively used
  - For example, high-frequency trading
    - Explainable
    - Efficient
- Reflecting on the past can shed light on the future.
  - Imagine that you were a student back in 1958 or 1960, would you be able to discover the same algorithms as Rosenblatt, Widrow and Hoff did?
    - If you were able to, what would motivate you to come up with this idea?
    - If you weren’t able to, what would prevent you from thinking in that direction?

Thanks