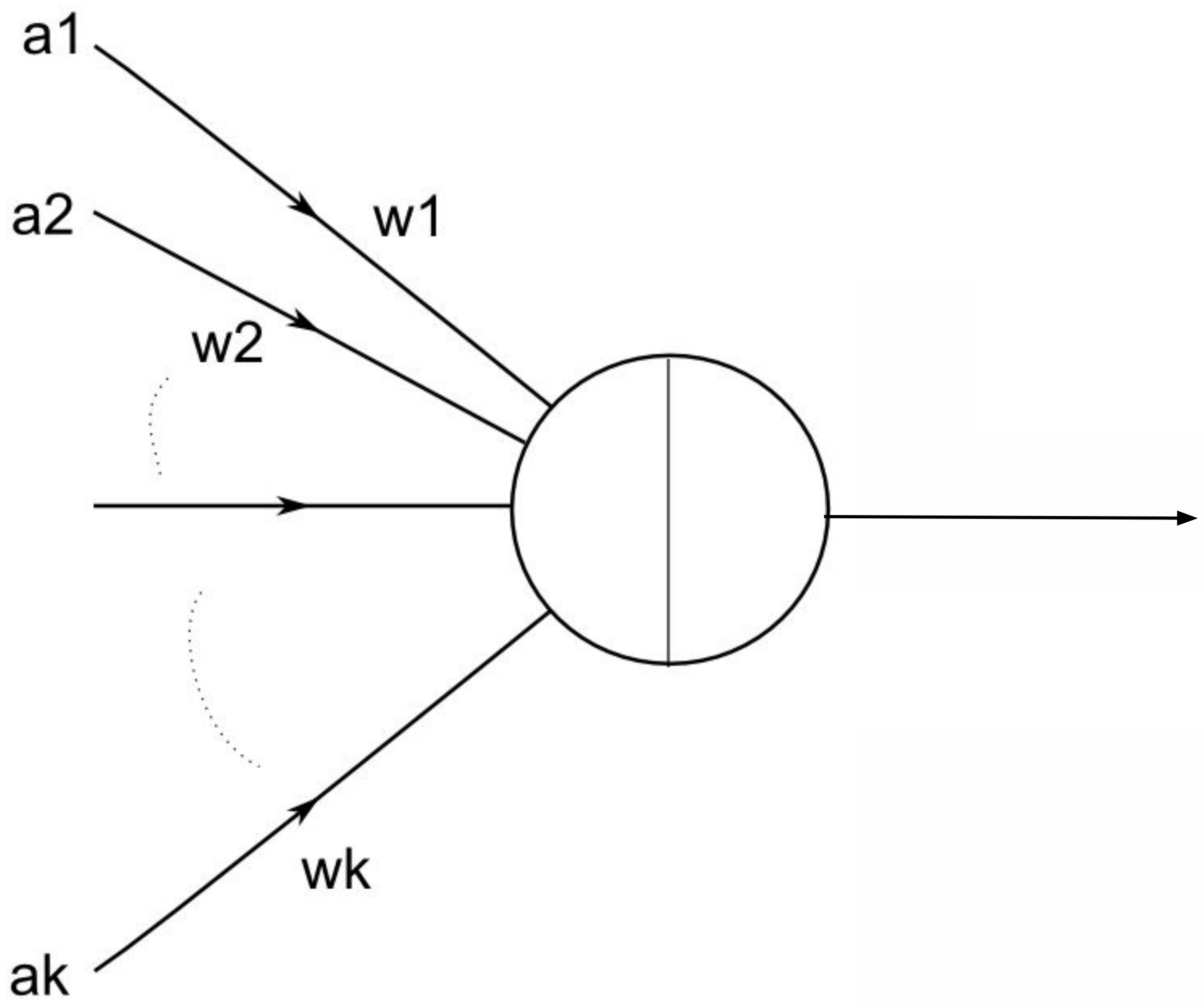
An Introduction to Neural Networks

MUSTAFA HAJIJ

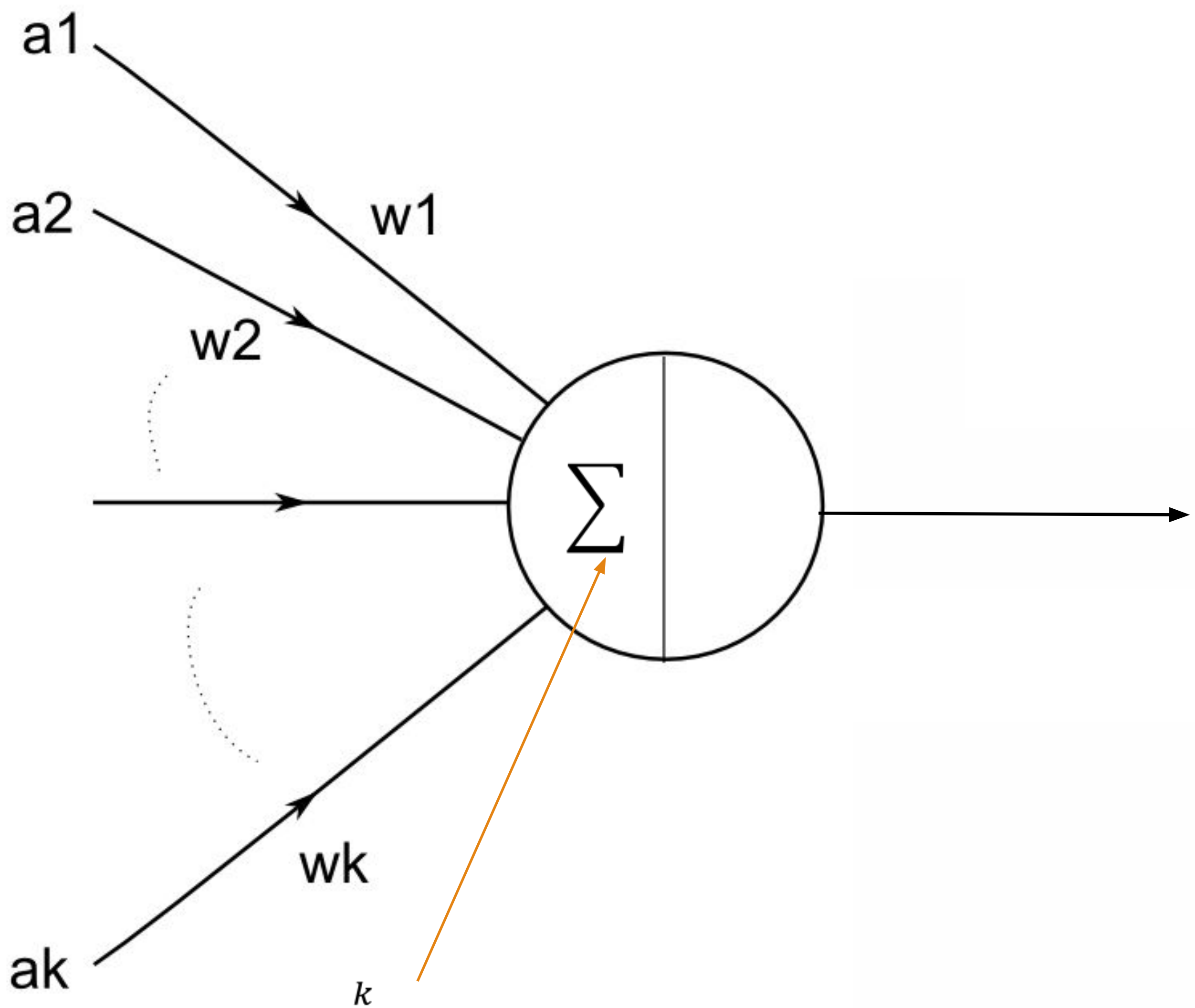
Objectives:

- Definition of a perceptron : the simplest neural network ever.
- How perceptron can be used for binary-labeled data classification
- Training of a perceptron
- Optimization and the general gradient descent algorithm
- General neural network
- Feedforward of a neural network
- An introduction to training neural networks: the backprop algorithm

Perceptron



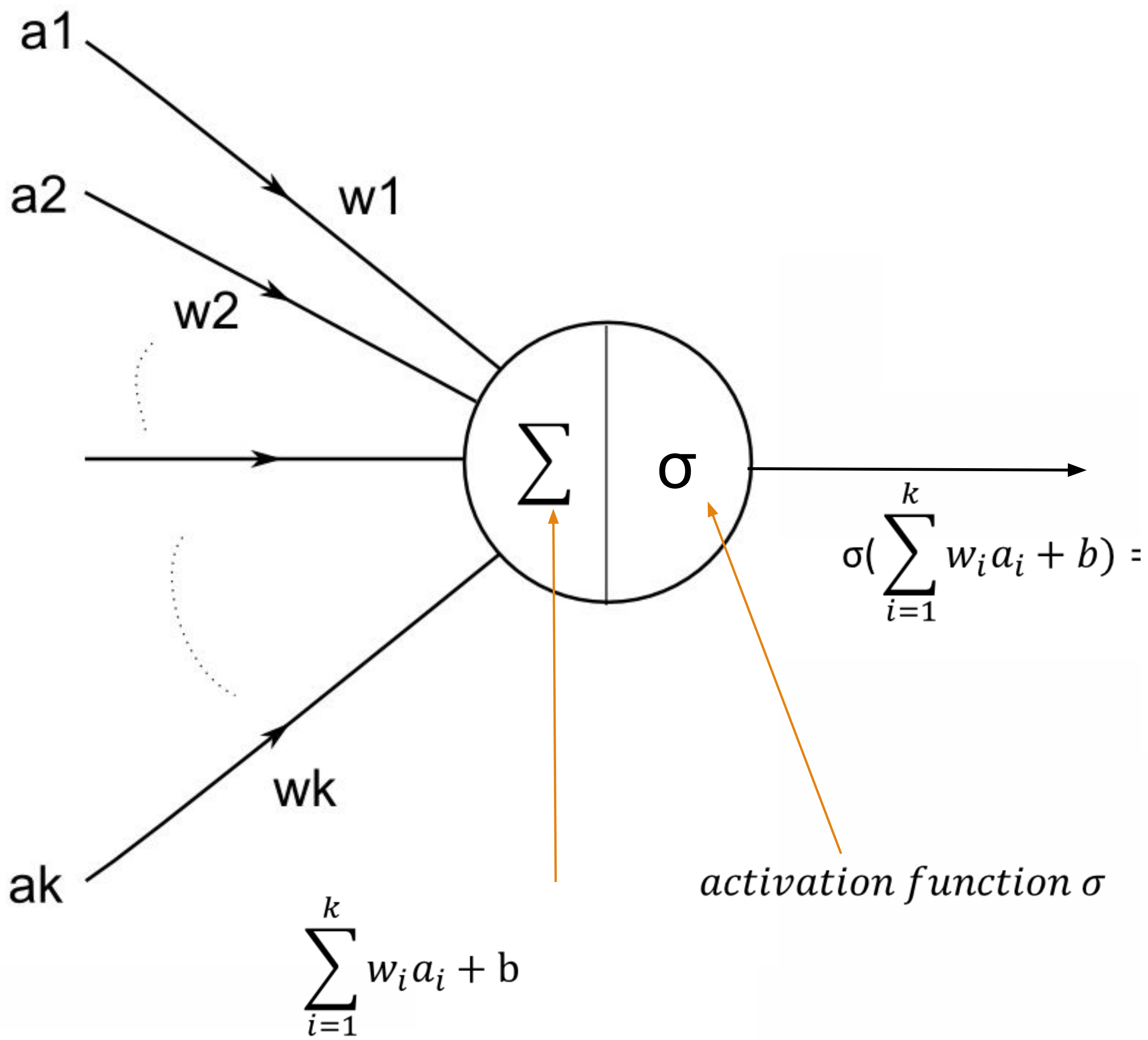
Perceptron



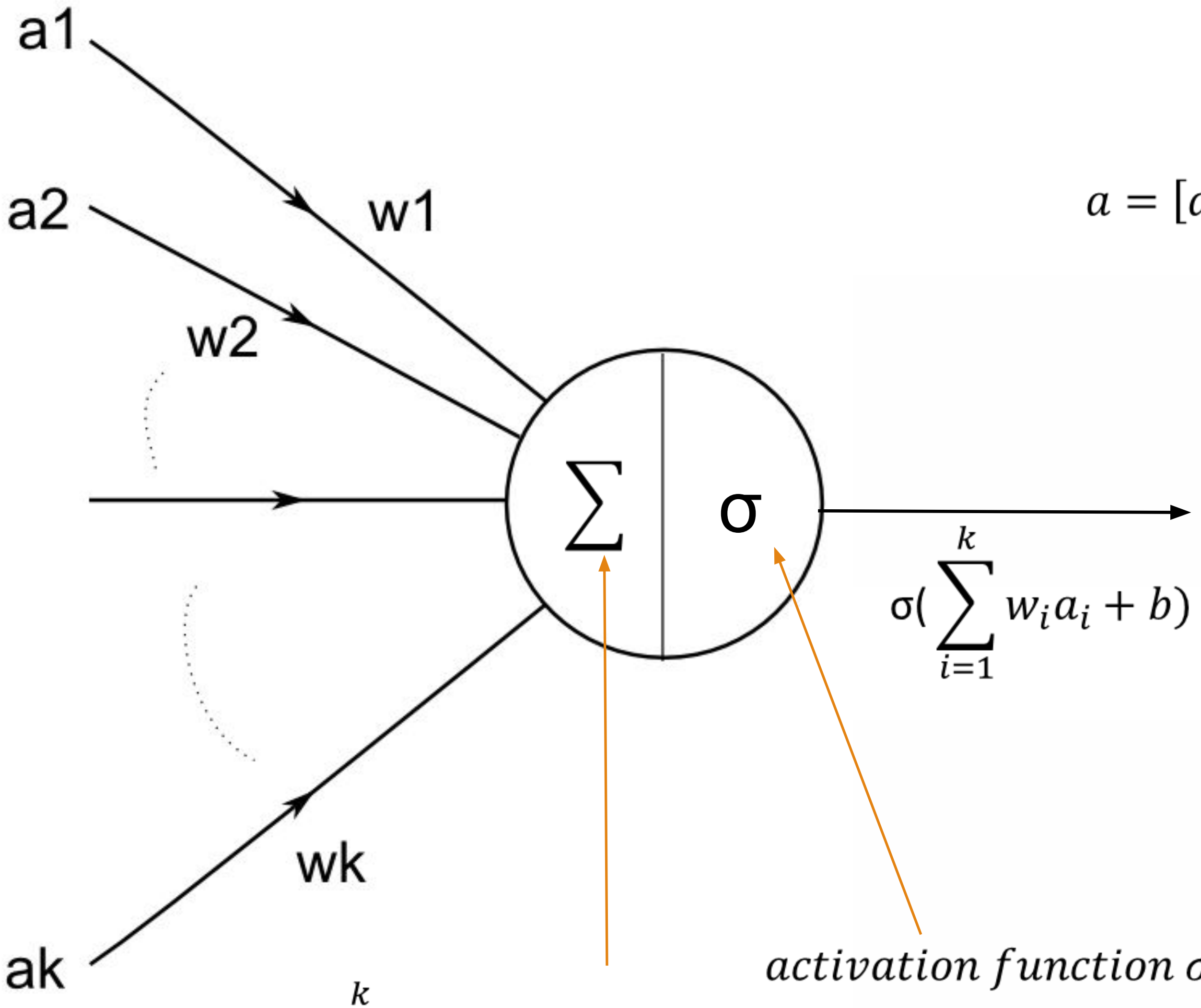
$$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$$

b is called a bias term.

Perceptron


a1
a2
w1
w2
wk
ak
$\sum$
σ
$\sigma(\sum_{i=1}^{k} w_i a_i + b)$ =
$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$
$activation\ function\ \sigma$

Perceptron



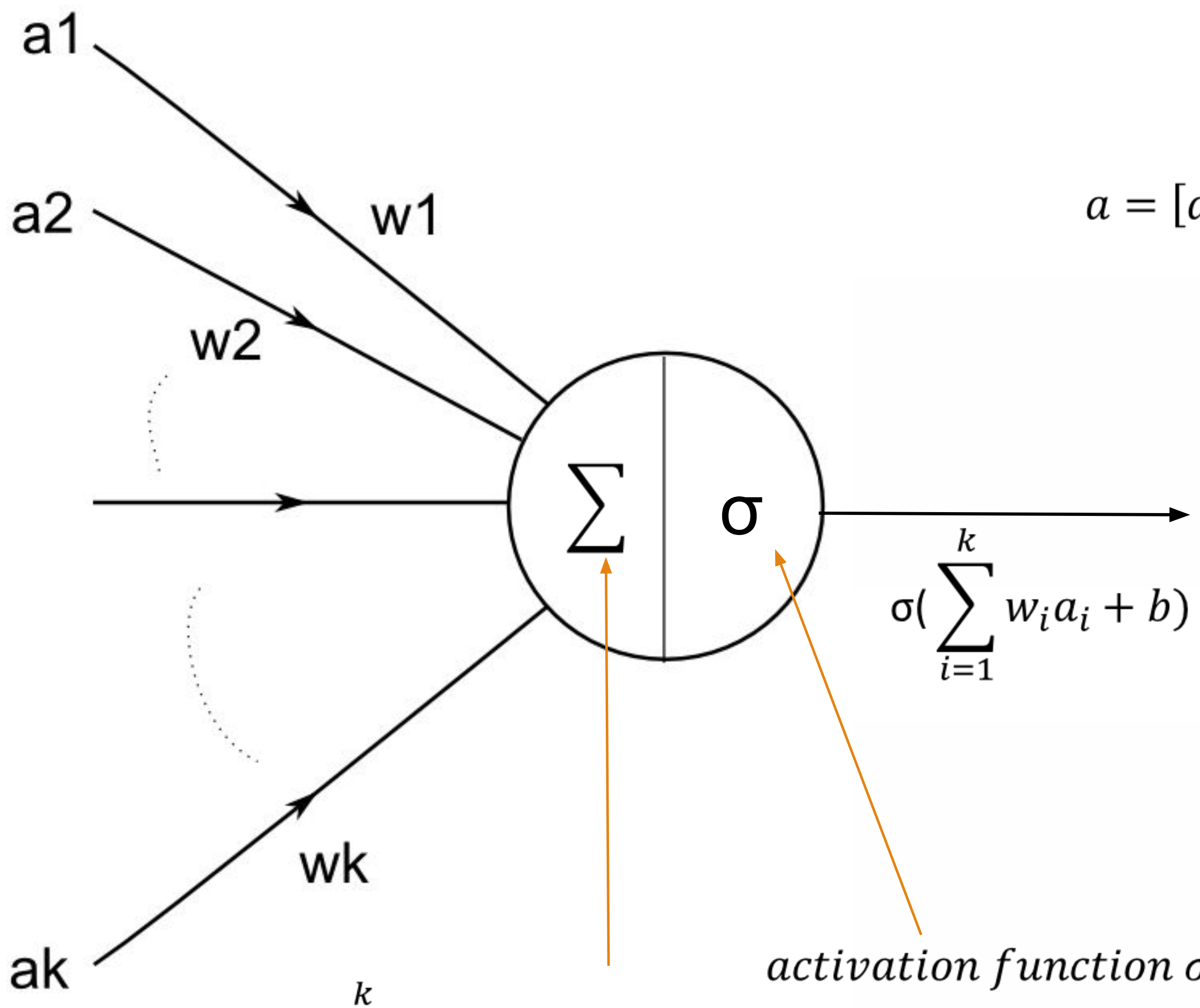
$$a = [a_1, \dots, a_k]^T \qquad w^T = [w_1, \dots, w_k]$$

$$\sigma\left(\sum_{i=1}^{k} w_i a_i + b\right) = \sigma(w^T a + b)$$

$$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$$

activation function σ

Perceptron



$a = [a_1, \ldots, a_k]^T \quad w^T = [w_1, \ldots, w_k]$

$\sigma(\sum_{i=1}^{k} w_i a_i + b) = \sigma(w^T a + b)$

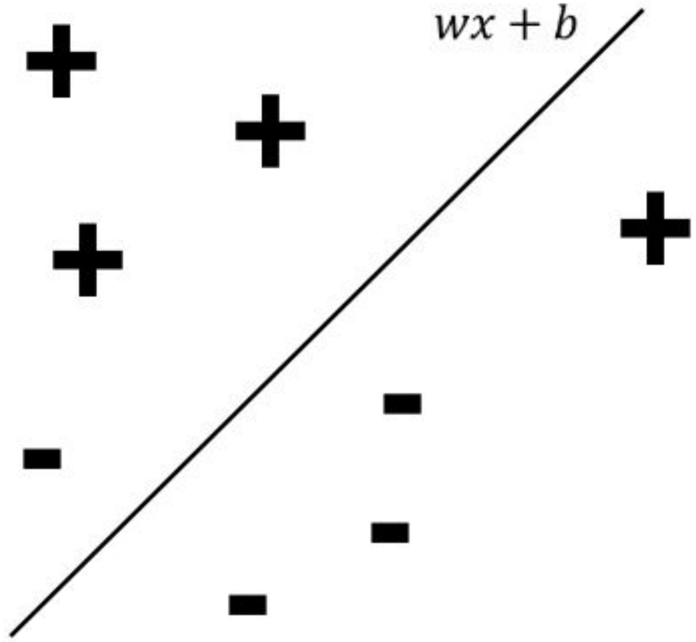
$activation\ function\ \sigma$

$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$

We usually think about **w** as the parameters, **a** as the input data and the entire **perceptron** as the model

Training Perceptron

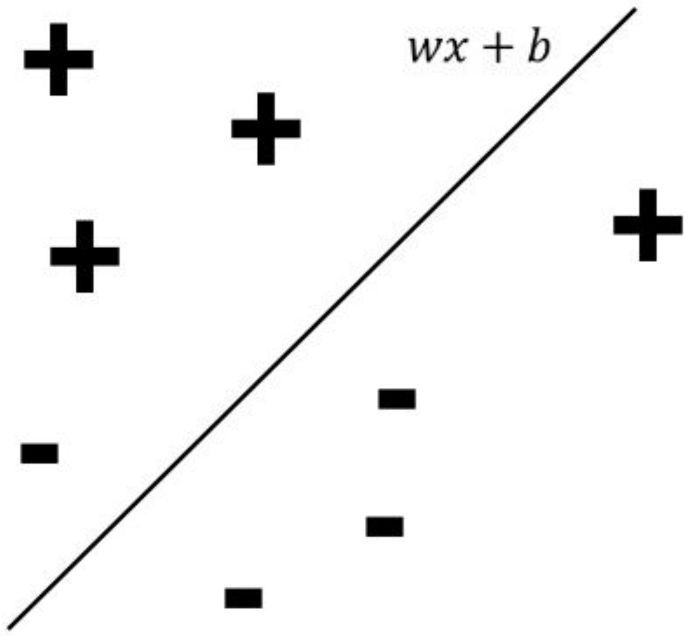
Given a collection of points $(x_1, y_1), \ldots., (x_n, y_n)$ where
x_i is a points in R^d and y_i is a label that takes values in $\{-1, +1\}$



Training Perceptron

Given a collection of points $(x_1, y_1), \ldots., (x_n, y_n)$ where
x_i is a points in R^d and y_i is a label that takes values in $\{-1, +1\}$

We want to choose $w = [w_1, \ldots w_d]$ and b such that the hyperplane determined by the $wx + b = 0$ separates the points x_i according to their labels. In other words, we want to choose the plane $wx + b = 0$ so that all points with positive sign on one side and all points with negative sign on the other side.

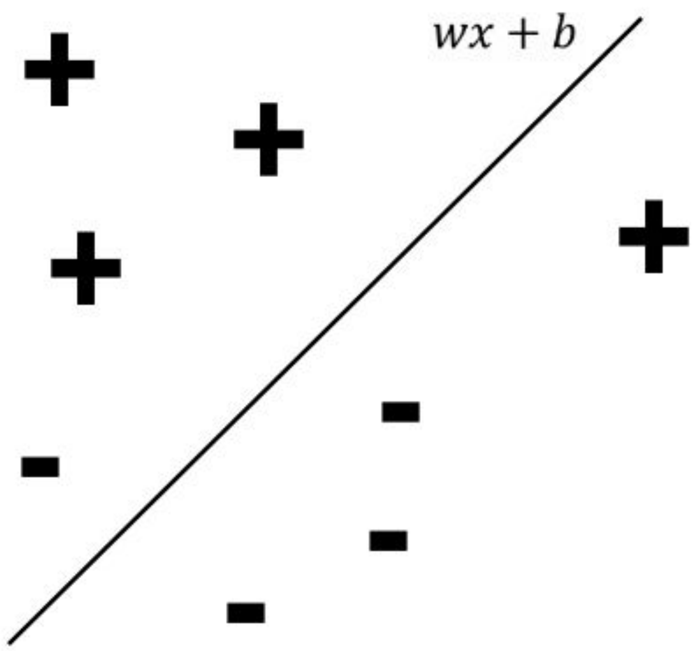


Training Perceptron

Given a collection of points $(x_1, y_1), \ldots., (x_n, y_n)$ where
x_i is a points in R^d and y_i is a label that takes values in $\{-1, +1\}$

We want to choose $w = [w_1, \ldots w_d]$ and b such that the hyperplane determined by the $wx + b = 0$ separates the points x_i according to their labels. In other words, we want to choose the plane $wx + b = 0$ so that all points with positive sign on one side and all points with negative sign on the other side.

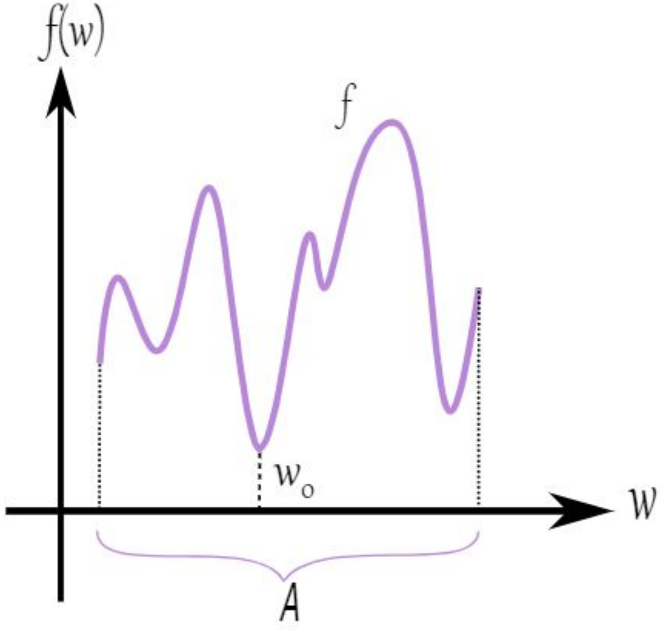
As usual we have to define a cost function and a notion of error.



Lets recall what that means in a bit more details.

Optimization

For continuous optimization problem the problem can be generally stated as follows. For a given set A in the Euclidean space $\mathbf{R}^n$ we are giving a function $f : A \longrightarrow \mathbf{R}$, usually called the *cost or the loss function*, and the goal is to find the point $w_0 \in A$ such that $f(w_0) \leq f(w)$ for every point $w \in A$.



Optimization

Question : why this problem is hard?

Question : why this problem is important?

Question : Is it always possible to find a solution for an optimization problem?

Optimization : practical examples

Portfolio optimization is an optimization problem in finance where the objective is to find the best allocation of investments among different assets. The goal is to maximize expected return while minimizing risk.

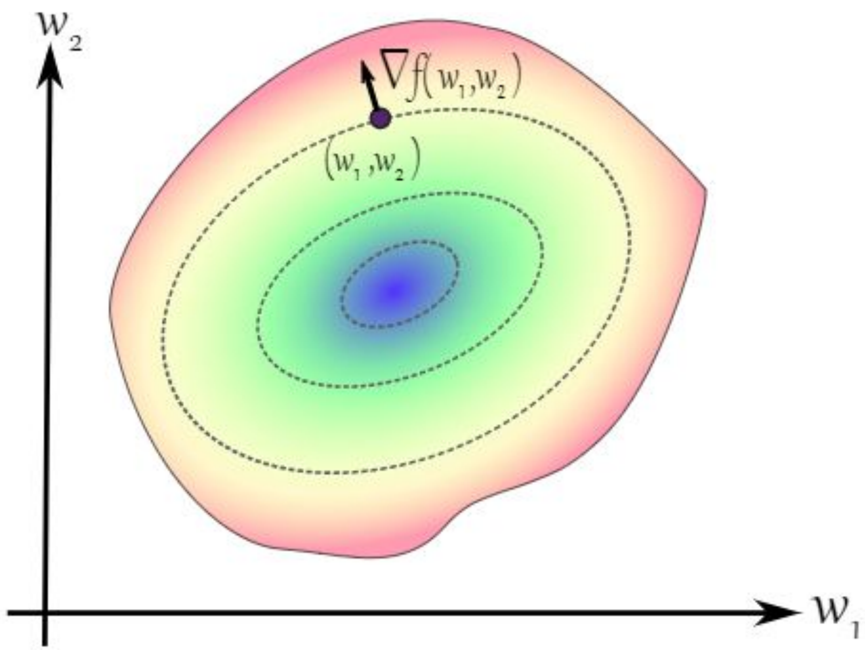
By determining the weights of each asset in the portfolio, the optimization process aims to strike a balance between maximizing returns and managing risk.

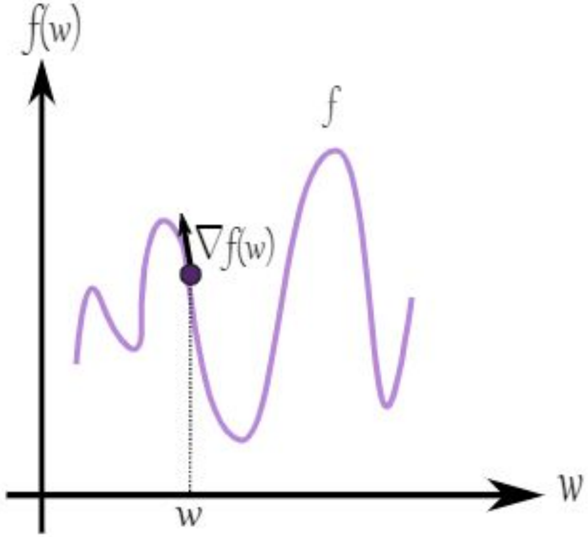
Optimization : practical examples

The calibration problem is an optimization problem because it involves finding the best parameter values that minimize the discrepancy between a model's predictions and observed data.

The objective is to optimize the model's parameters to align its output with the desired targets, typically by minimizing an objective function.

Differentiable functions





When the function f is differentiable then we can compute the gradient. This can be used for a useful algorithm (the gradient descent) that allows us finding a local min of a diff function

General Gradient Descent Algorithm

Gradient descent is a powerful optimization algorithm that allows models to learn from data by iteratively adjusting their parameters to minimize the loss. It is widely used in training machine learning models and forms the basis for many advanced optimization techniques used in deep learning.

General Gradient Descent Algorithm

Suppose that we are given a differentiable function $f(w_1, \dots, w_d)$

General Gradient Descent Algorithm

Suppose that we are given a differentiable function $f(w_1, \dots, w_d)$

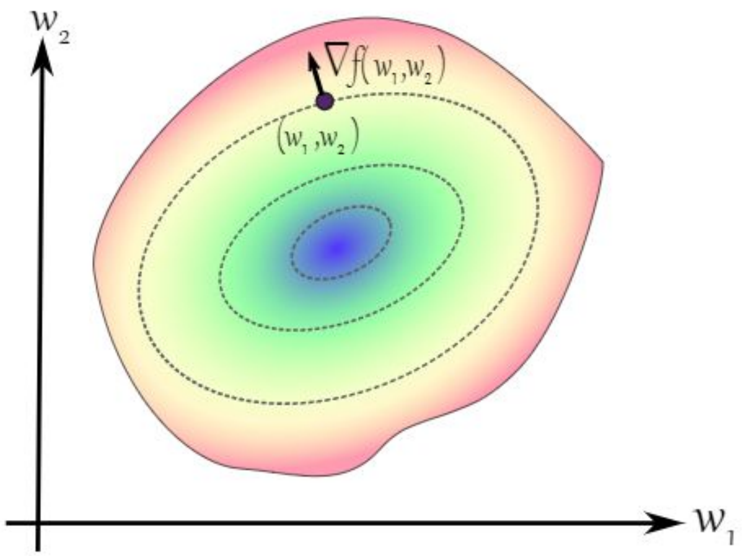
Want to find $w_1, \dots, w_d$ such that $f(w_1, \dots, w_d)$ is minimal

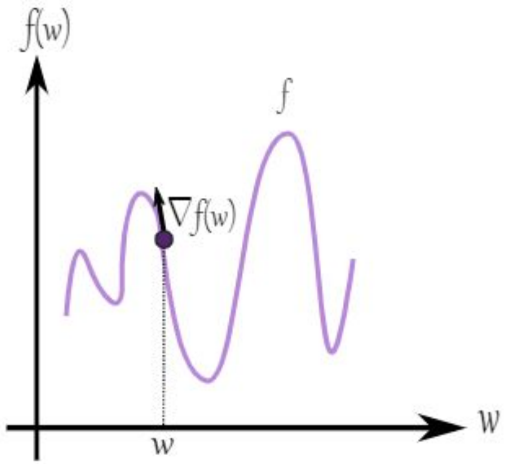
maxpixel.freegreatpicture.com
Walking Man Free Stock Photo - Public Domain Pictures

General Gradient Descent Algorithm

Suppose that we are given a differentiable function $f(w_1, \ldots, w_d)$

Want to find $w_1, \ldots, w_d$ such that $f(w_1, \ldots, w_d)$ is minimal. Gradient descent gives a way to find a local minimum for f





General Gradient Descent Algorithm

Suppose that we are given a differentiable function $f(w_1, \dots, w_d)$

Want to find $w_1, \dots, w_d$ such that $f(w_1, \dots, w_d)$ is minimal. Gradient descent gives a way to find a local minimum for f

Outline :

(1) Initiate $w_1, \dots, w_d$ randomly
(2) keep changing $w_1, \dots, w_d$ until hopefully $f(w_1, \dots, w_d)$ is minimal

General Gradient Descent Algorithm

Suppose that we are given a differentiable function $f(w_1, \dots, w_d)$

Want to find $w_1, \dots, w_d$ such that $f(w_1, \dots, w_d)$ is minimal. Gradient descent gives a way to find a local minimum for f

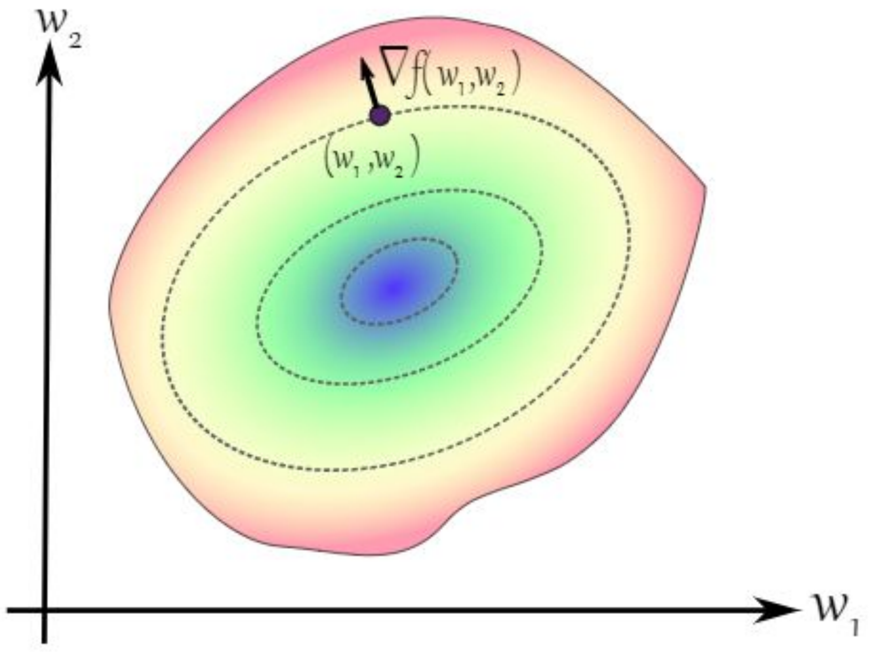
Outline :

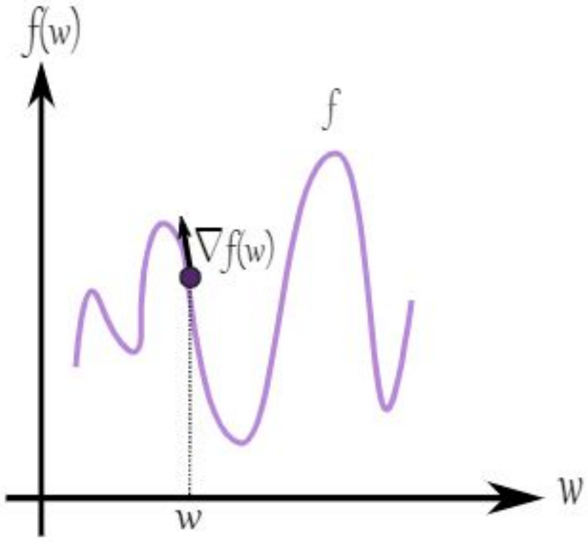
(1) Initiate $w_1, \dots, w_d$ randomly
(2) keep changing $w_1, \dots, w_d$ until hopefully $f(w_1, \dots, w_d)$ is minimal

But how exactly do we change $w_1, \dots, w_d$?

General Gradient Descent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.





General Gradient Descent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

(1) Initiate $w_1, \dots, w_d$ randomly

General Gradient Descent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

(1) Initiate $w_1, \dots, w_d$ randomly
(2) Repeat until convergence :
    (1) For every i in range(1,d):
        $(1) w_i \coloneqq w_i - q \frac{\partial f}{\partial w_i}$ (here we do simultaneous update for the parameters w_i)

General Gradient Descent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

(1) Initiate $w_1, \dots, w_d$ randomly
(2) Repeat until convergence :
    (1) For every i in range(1,d):
        $(1) w_i := w_i - q \frac{\partial f}{\partial w_i}$ (here we do simultaneous update for the parameters w_i)

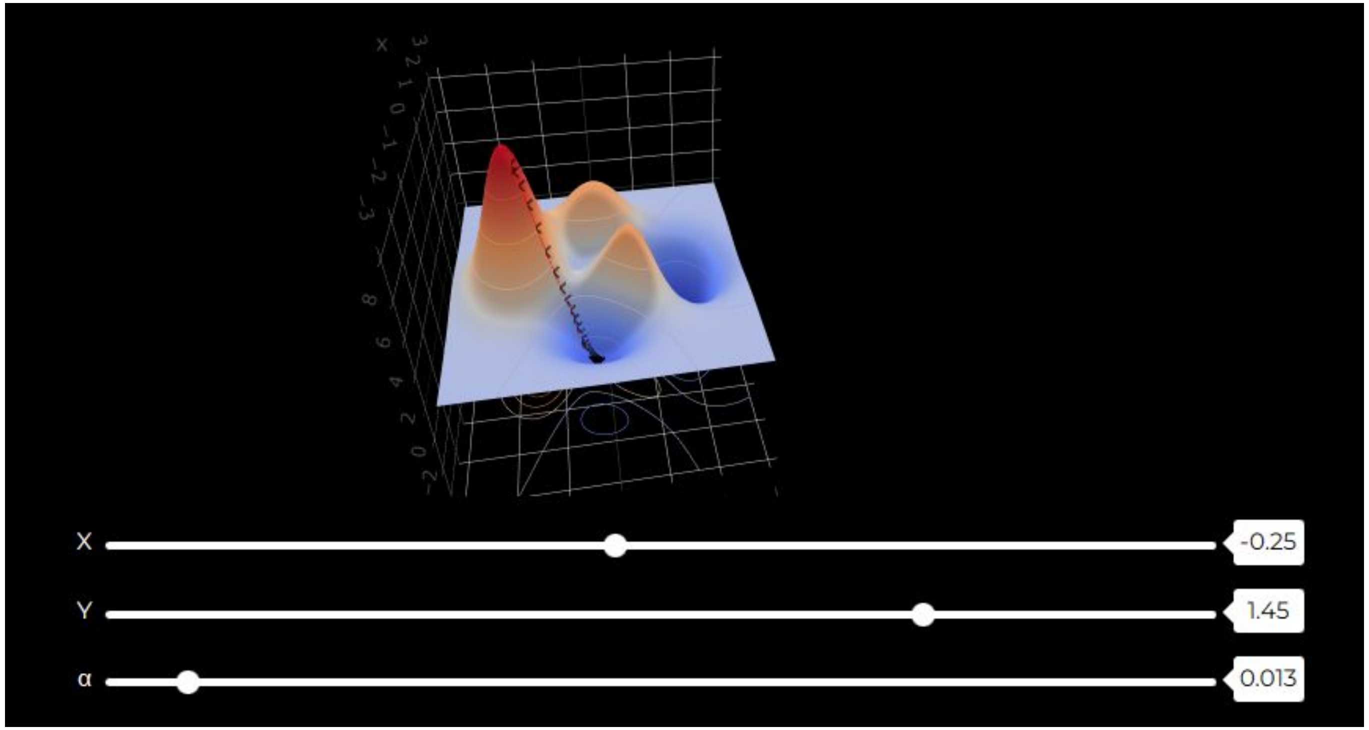
Gradient decent asserts that the values of the function f when we update as described above are non-increasing :

$$f(old\, w_i) \geq f(new\, w_i)$$

General Gradient Descent Algorithm

Lets explore this algorithm interactively

[Interactive Gradient Descent Demo · Sasha Kuznetsov's Blog (skz.dev)](https://skz.dev)



Training Perceptron

Now back to training a perceptron. We need some facts.

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Hence

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Hence

$w^T (x_1 - x_2)$=0

Hence the vector w^T is orthogonal to $(x_1 - x_2)$

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Hence

$w^T(x_1 - x_2)$=0

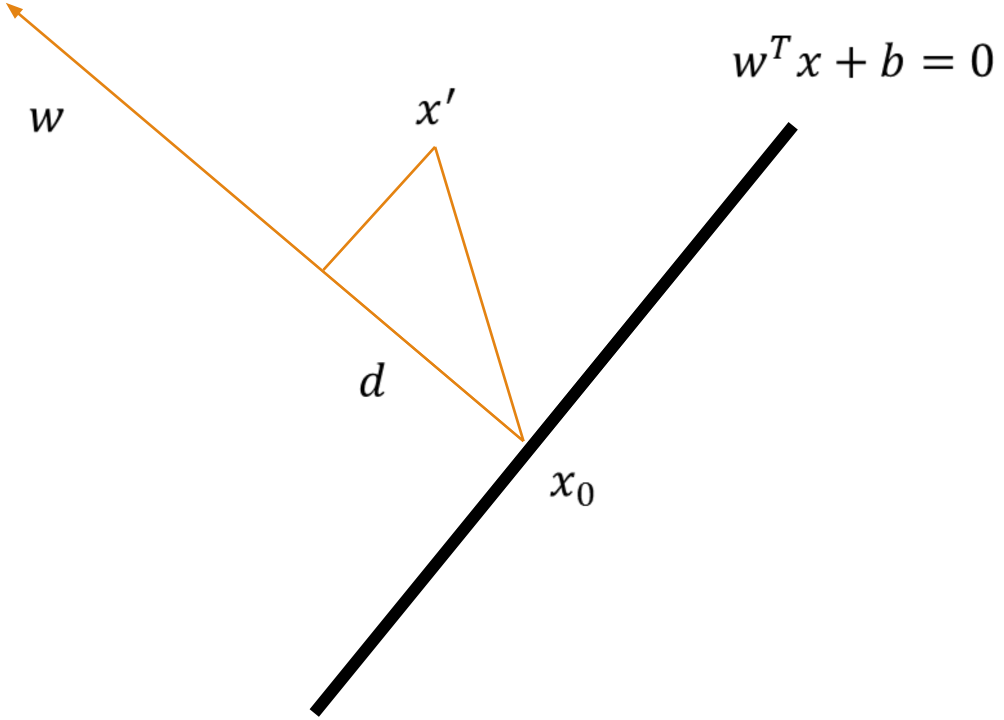
Hence the vector w^T is orthogonal to $(x_1 - x_2)$

Moreover, for any x_0 on the plane $w^T x + b = 0$ we have

$$b = -w^T x_0$$

Training Perceptron

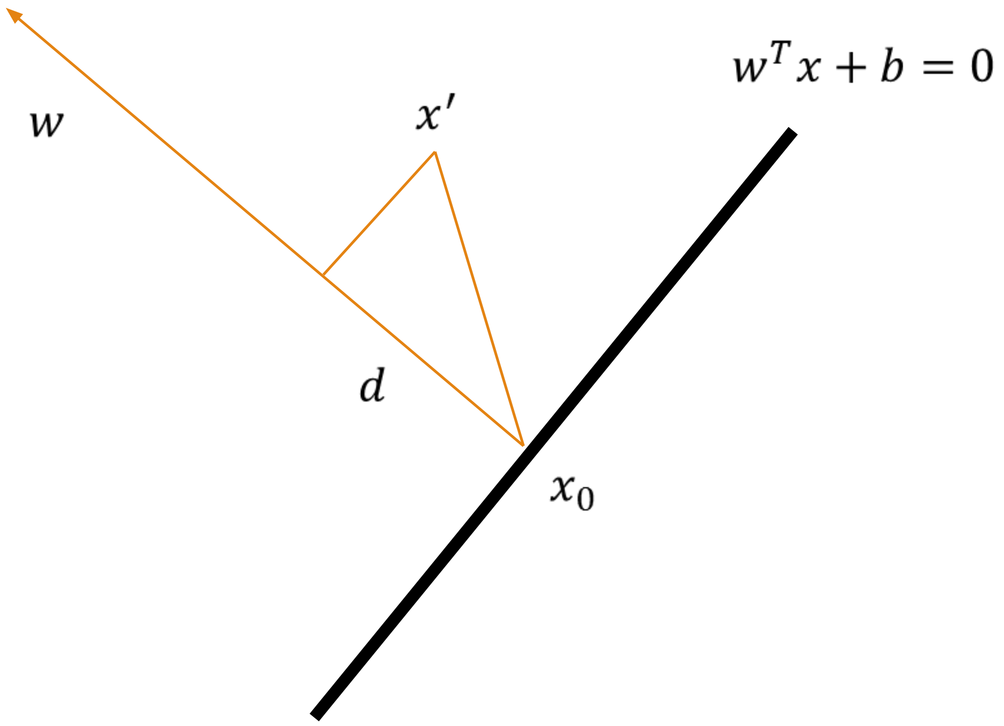
$$d = w^T(x' - x_0) = w^T x' - w^T x_0 = w^T x' + b$$



e

Training Perceptron

$$d = w^T(x' - x_0) = w^T x' - w^T \mathrm{x}_0 = w^T x' + b$$



So if we have a point and we want to see where it is located on with respect to the plan, then all we have to do is to plug it in the equation of the plane.

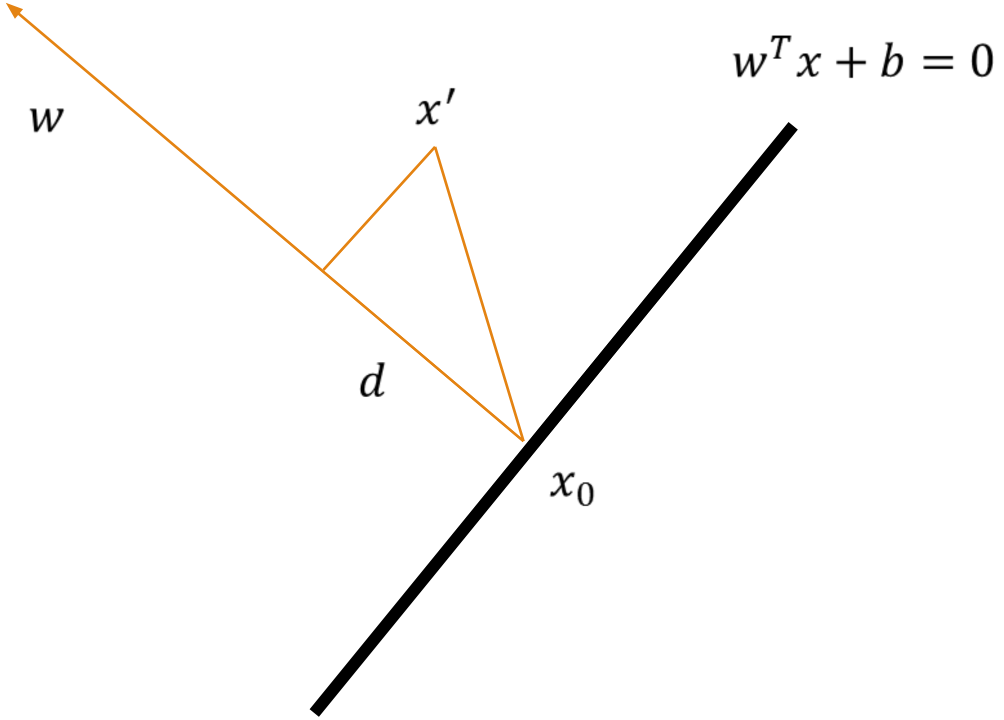
Training Perceptron

Write

$$d_i = y_i(w^T x_i + b)$$

Where (x_i, y_i) is a training example

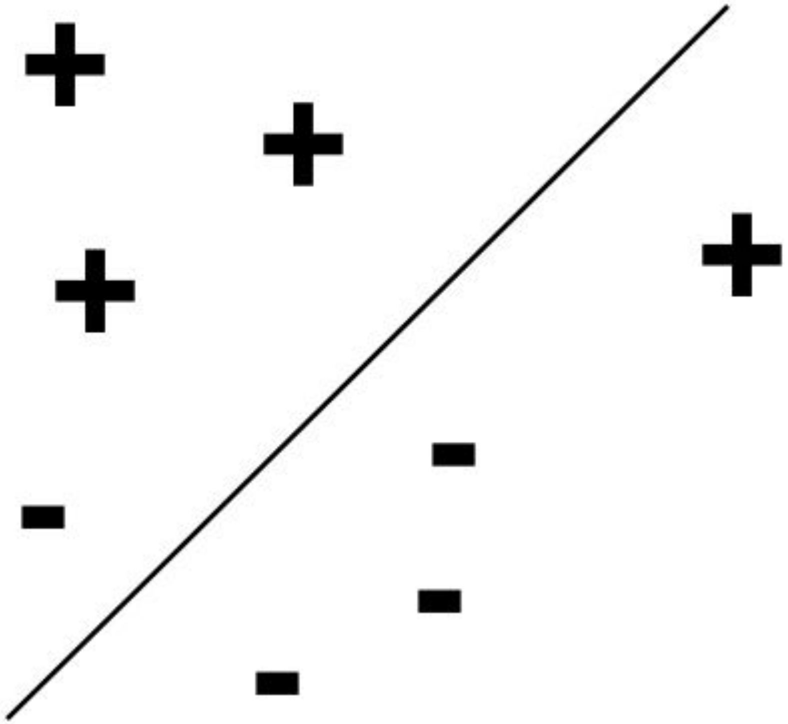
Note that $d_i \geq 0$



Training Perceptron

Define

$$error(w, b) := -\sum_M y_i(w^T x_i + b_0\ )$$

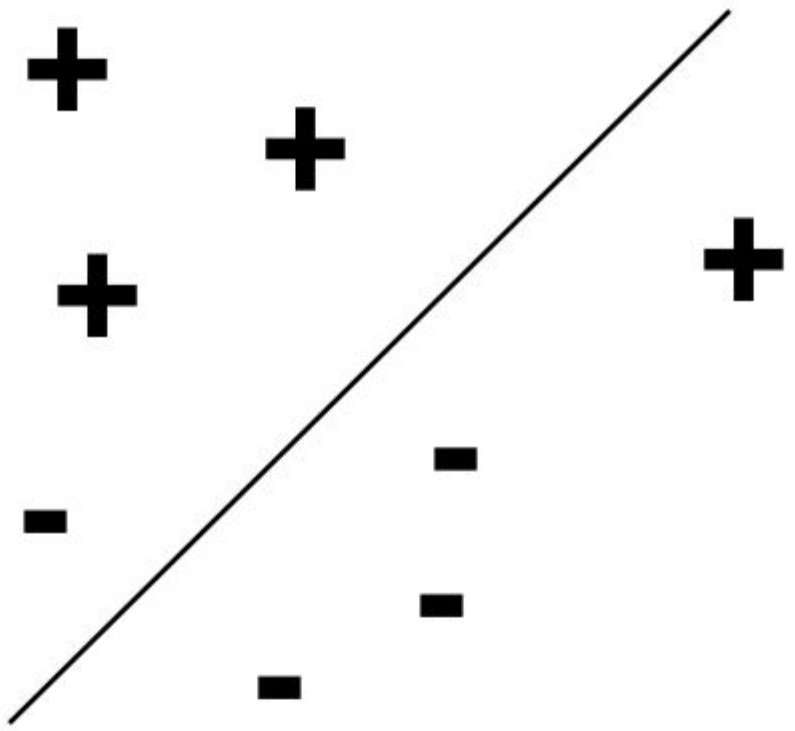


Training Perceptron

Define

$$error(w, b) \coloneqq -\sum_M y_i (w^T x_i + b_0\ )$$

Where M is the set of misclassified points

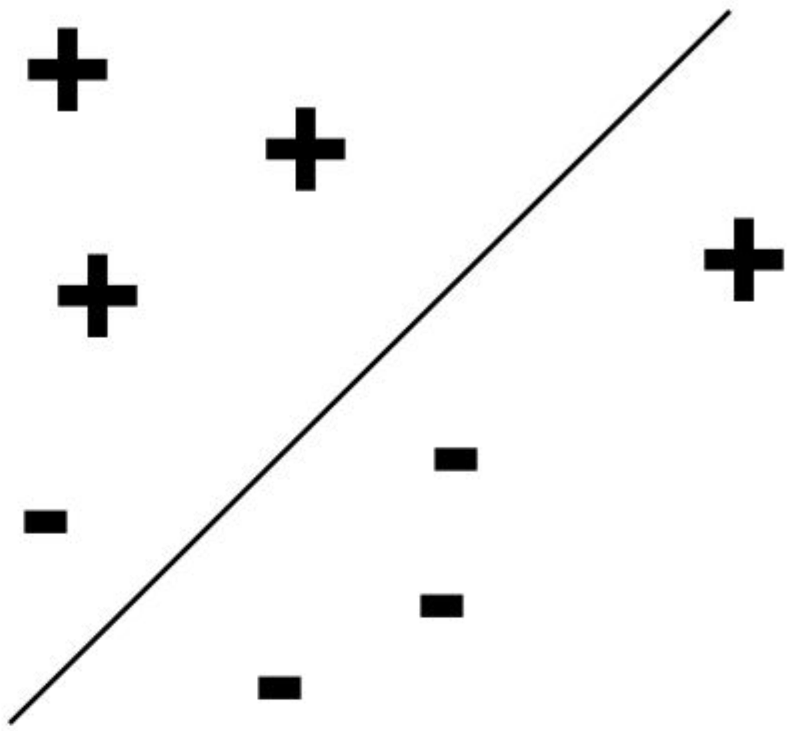


Training Perceptron

Define

$$error(w,b) := -\sum_M y_i(w^T x_i + b_0\ )$$

Where M is the set of misclassified points



We want to apply gradient decent on the function $error(w,b)$

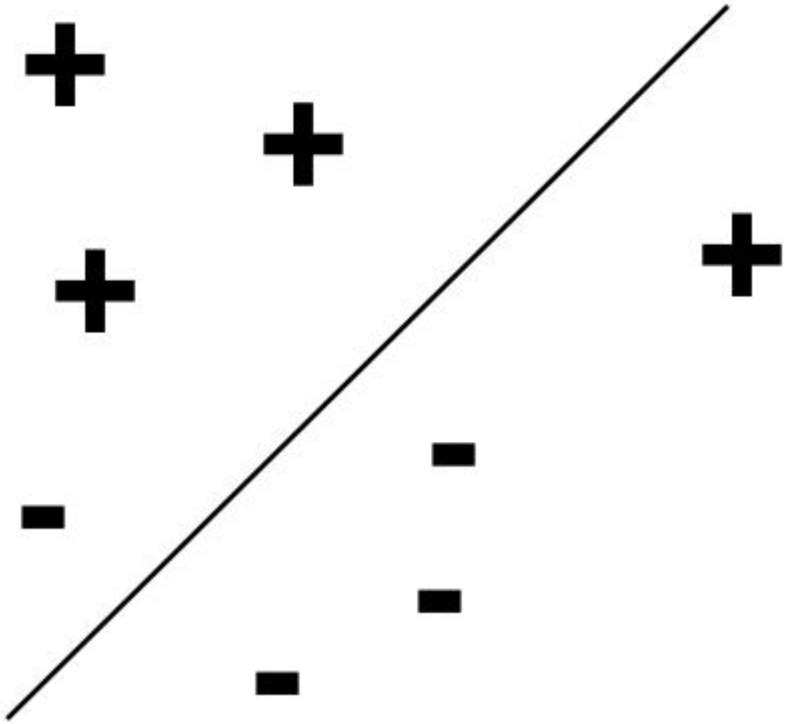
$$\frac{\partial\ error(w,b)}{\partial\ w} = \sum_M y_i x_i$$

$$\frac{\partial\ error(w,b)}{\partial\ b} = \sum_M y_i$$

Training Perceptron

To train a perceptron

(1) Assign the weights w randomly

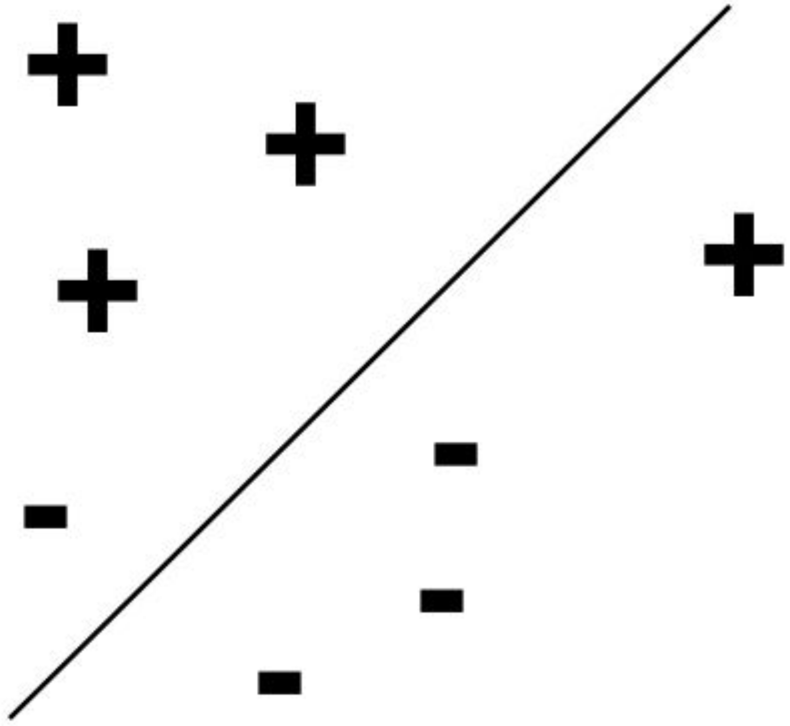


$$error(w, b) := -\sum_M y_i(w^T x_i + b_0)$$

Training Perceptron

To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence



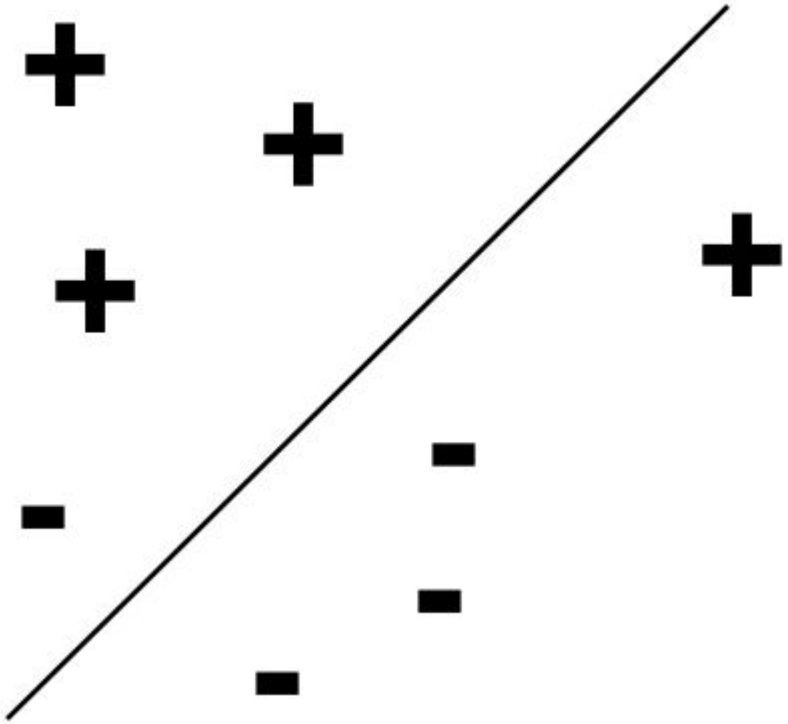
$$error(w, b) := -\sum_{M} y_i (w^T x_i + b_0)$$

Training Perceptron

To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence

$$w_{new} := w_{old} - q\frac{\partial\, error(w,b)}{\partial\, w}$$
$$b_{new} := b_{old} - q\frac{\partial\, error(w,b)}{\partial\, b}$$



$$error(w,b) := -\sum_M y_i(w^T x_i + b_0)$$

Training Perceptron

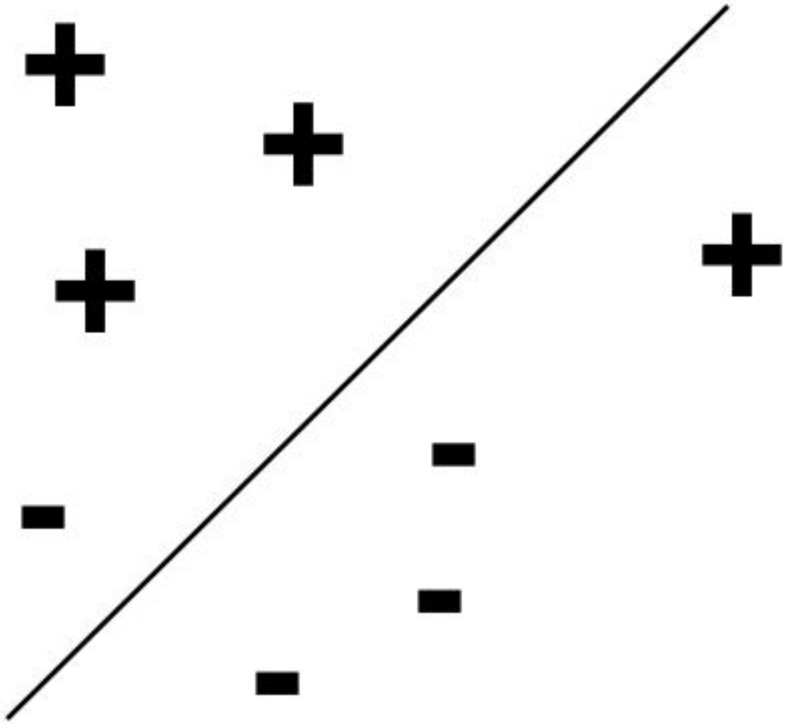
To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence

$$w_{new} := w_{old} - q \frac{\partial \, error(w,b)}{\partial \, w}$$

$$b_{new} := b_{old} - q \frac{\partial \, error(w,b)}{\partial \, b}$$

if the examples are linearly separable then the above model classifies the points



$$error(w,b) := -\sum_M y_i (w^T x_i + b_0)$$

Training Perceptron

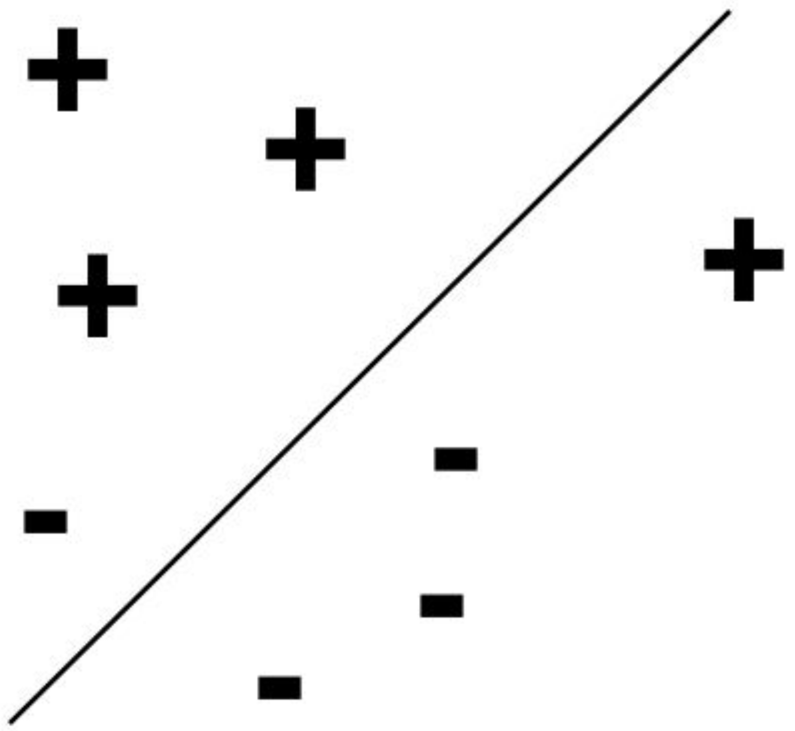
To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence

$$w_{new} := w_{old} - q y_i x_i$$

$$b_{new} := b_{old} - q x_i$$

Stochastic gradient decent



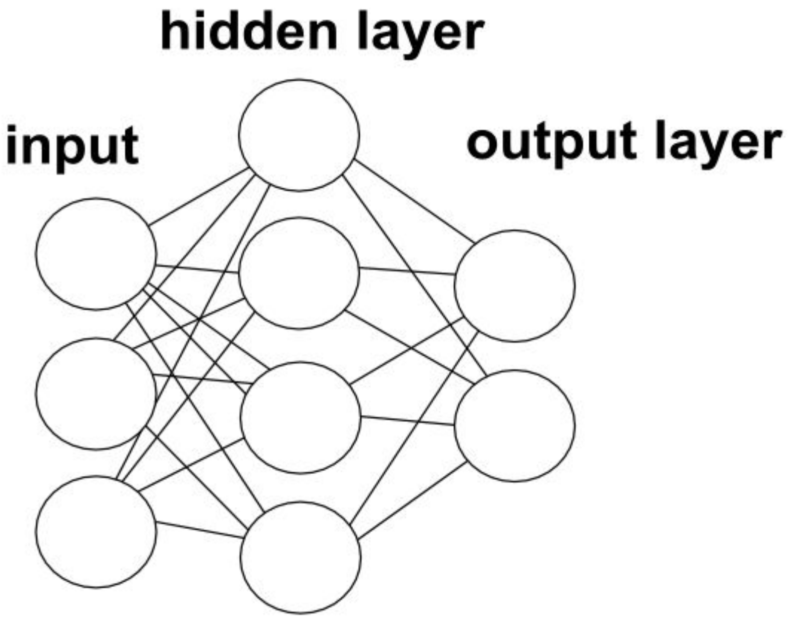
$$error(w, b) := -\sum_M y_i (w^T x_i + b_0)$$

if the examples are linearly separable then the above model classifies the points

Neural Network

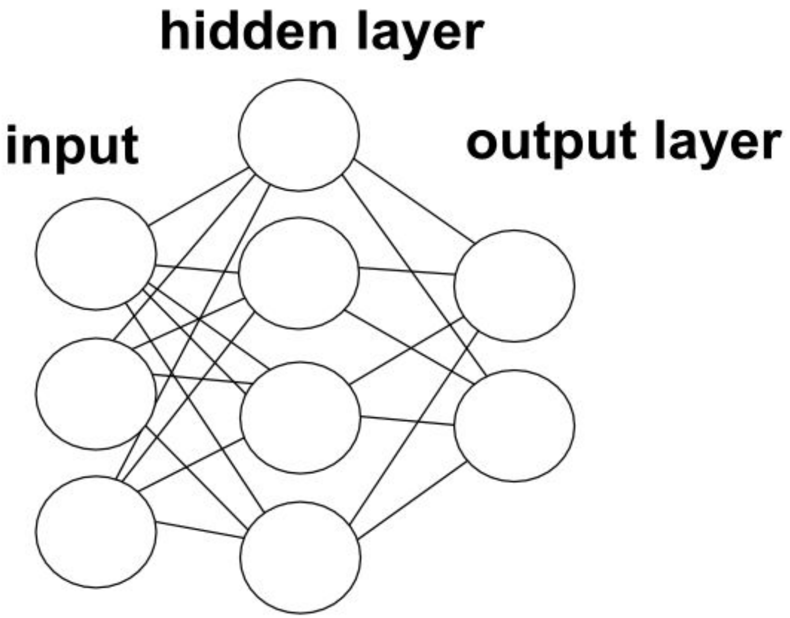
Clearly there are some data that cannot be classified using a single perceptron. Perceptron is the building block of a neural network.

The idea of neural network is to stack together multiple layers of perceptrons in order to be able to learn more complicated functions



Neural Network

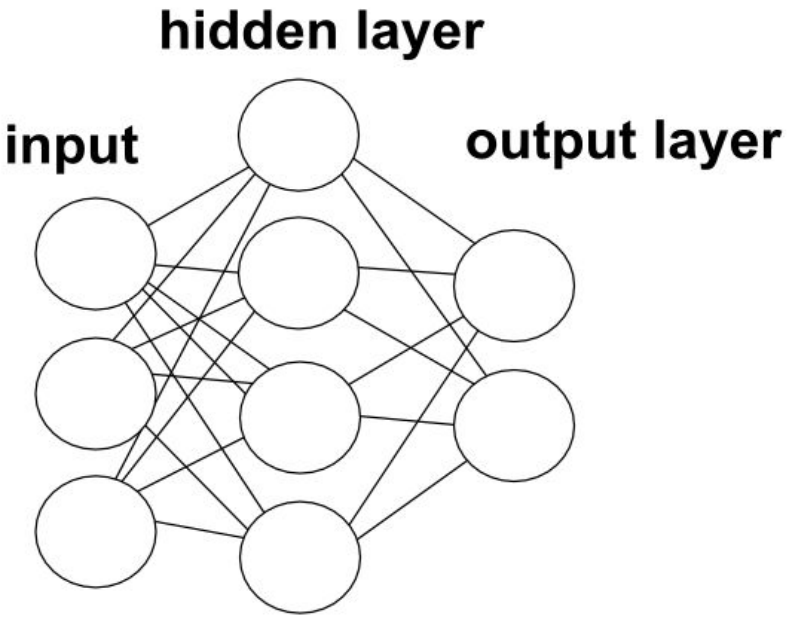
Mathematically, a neural network is a function f that takes x as input and produces an output y=f(x)



Neural Network

Mathematically, a neural network is a function f that takes x as input and produces an output y=f(x)

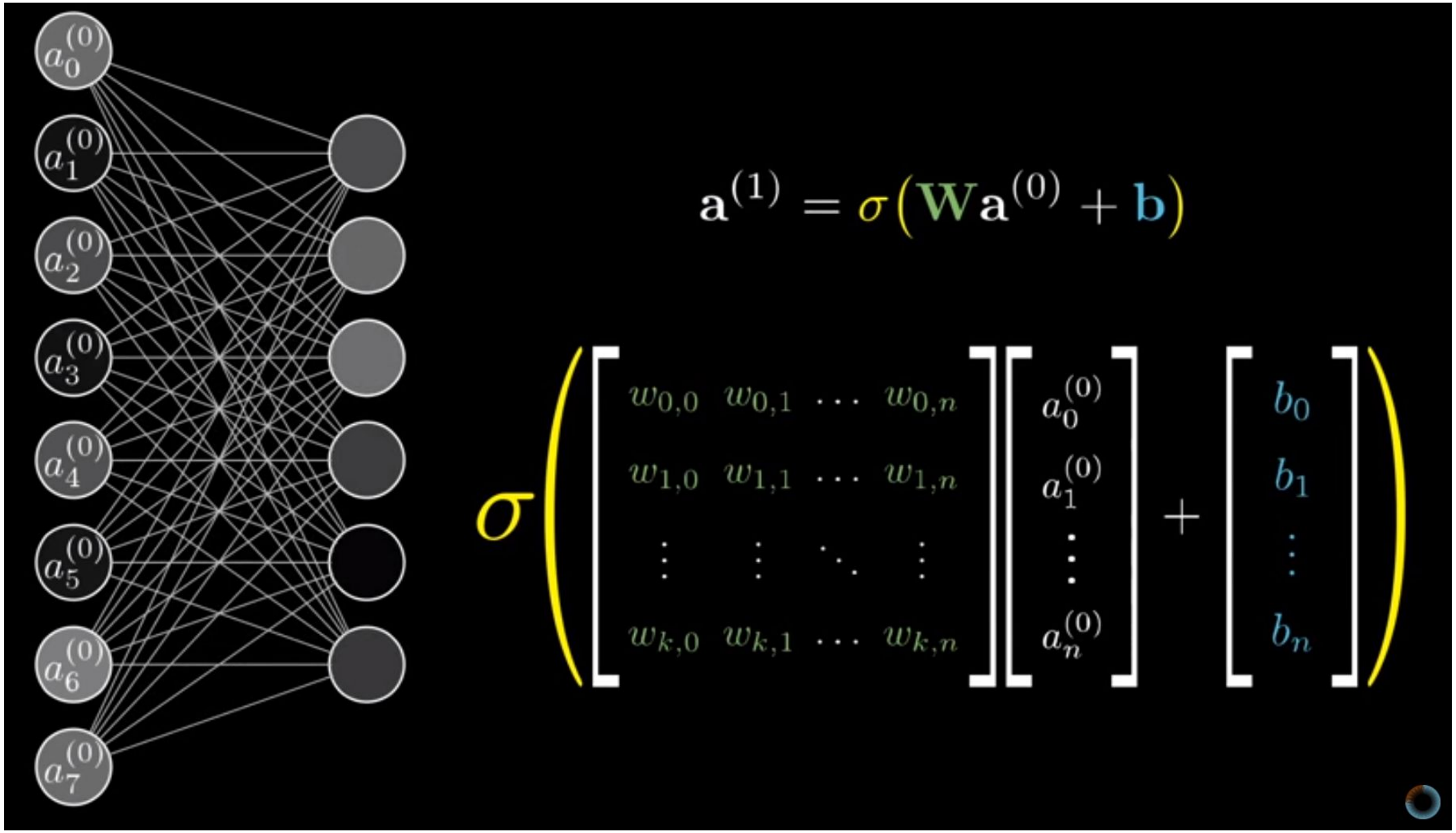
The training of a neural network means to tune the weights in all layers so that the output of the function f matches the label of x. The process of updating the weights for a feedforward neural network is called *backpropagation*.



Neural Network

How exactly do we compute the output of a given neural network ?

Watch this



Feedforward Neural Network

How do we compute a feedforward neural network on an input x ?

Feedforward Neural Network

Start with an input $x = a^{(0)}$. In the picture, this is represented by the first layer of nodes. We will call this layer 0.

$$x = a^{(0)}$$

Feedforward Neural Network

We apply the weight $W^{(1)}$ coming from the edges between layer 0 and layer 1 and add the biases and then apply the Activation function on the resulting vector coordinate-wise.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})}$$

$W^{(1)}$: Edges between
layer 0 and layer 1
$a^{(0)}$: input
$b^{(1)}$: biases applied to layer 1
σ : activation function

Feedforward Neural Network

We will call the output of this computation $a^{(1)}$. This is now represented by the nodes in layer 1.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}}$$

$W^{(1)}$: Edges between layer 0 and layer 1
$a^{(0)}$: input
$b^{(1)}$: biases applied to layer 1
σ : activation function

Feedforward Neural Network

Repeat.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}} \boxed{\sigma(W^{(2)}a^{(1)}+b^{(2)})} \xrightarrow{a^{(2)}}$$

$W^{(2)}$: Edges between layer 1 and layer 2
$a^{(1)}$: input from layer 1
$b^{(2)}$: biases applied to layer 2
σ : activation function

Feedforward Neural Network

Until you finish the neural network and get the final output.

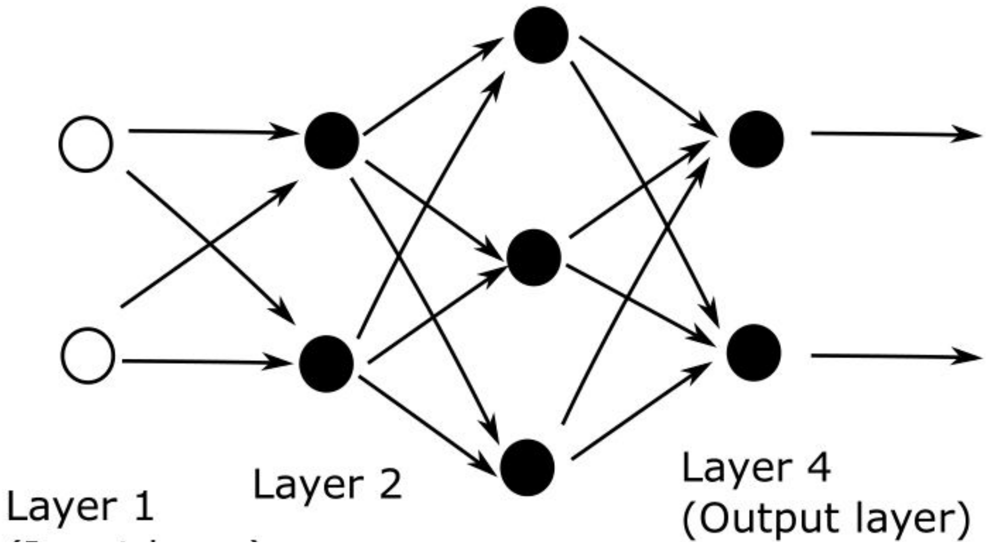
$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}} \boxed{\sigma(W^{(2)}a^{(1)}+b^{(2)})} \xrightarrow{a^{(2)}} \boxed{\sigma(W^{(3)}a^{(2)}+b^{(3)})} \dashrightarrow$$

$$\dashrightarrow \boxed{\sigma(W^{(n)}a^{(n-1)}+b^{(n)})} \longrightarrow a^{(n)}=y$$

Example

We will use an example from this paper.

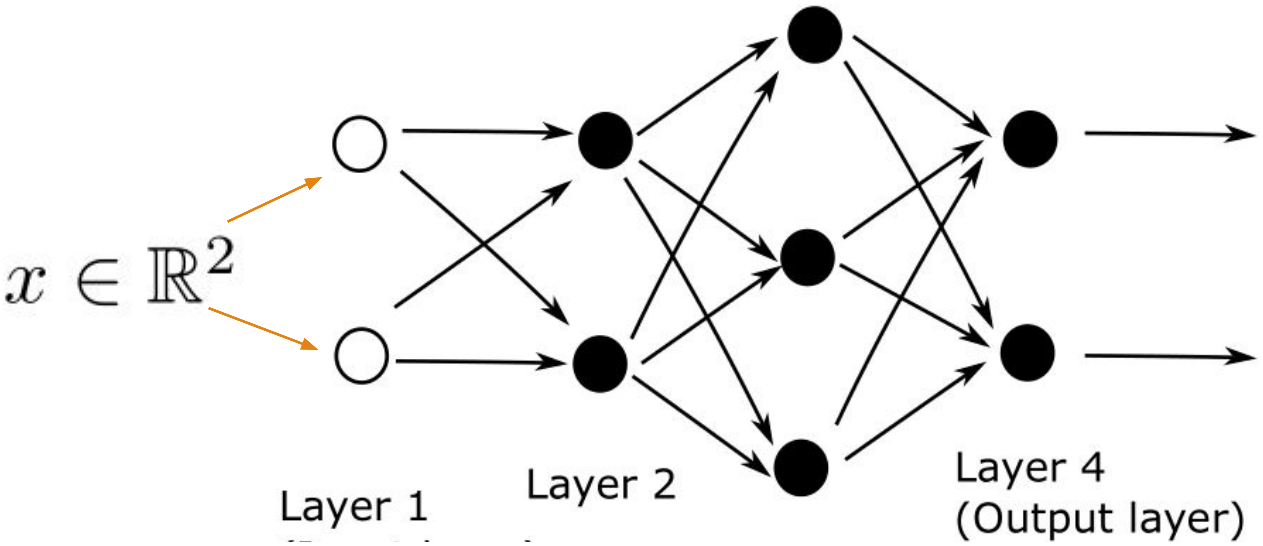
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

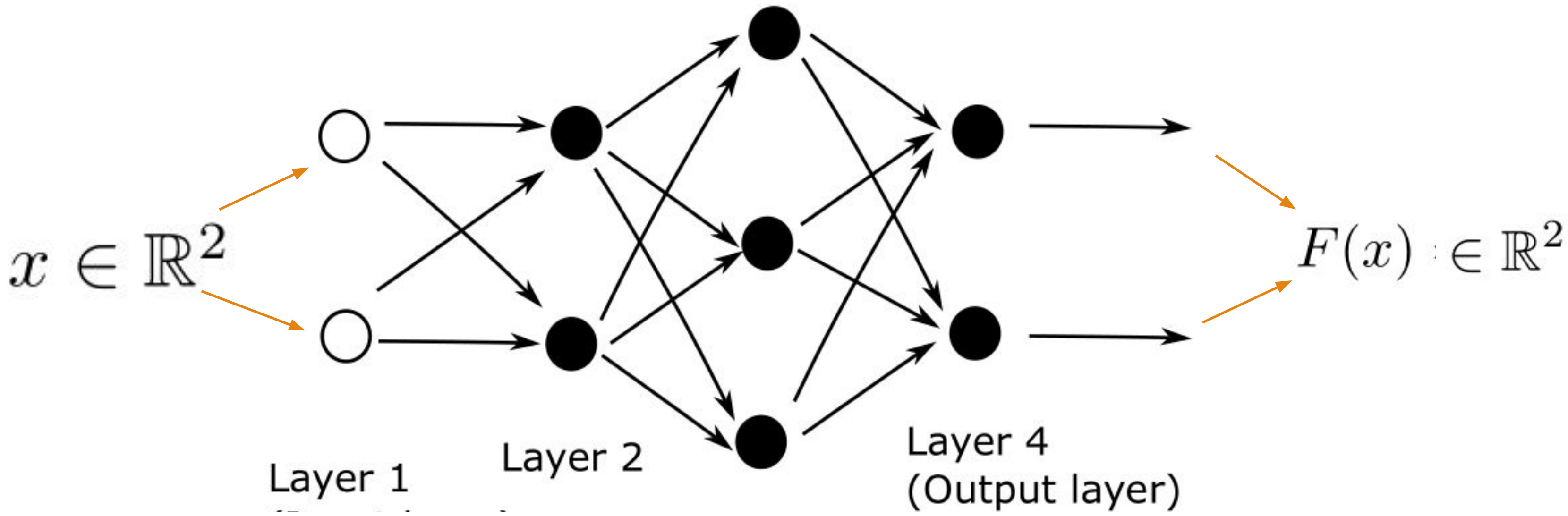
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

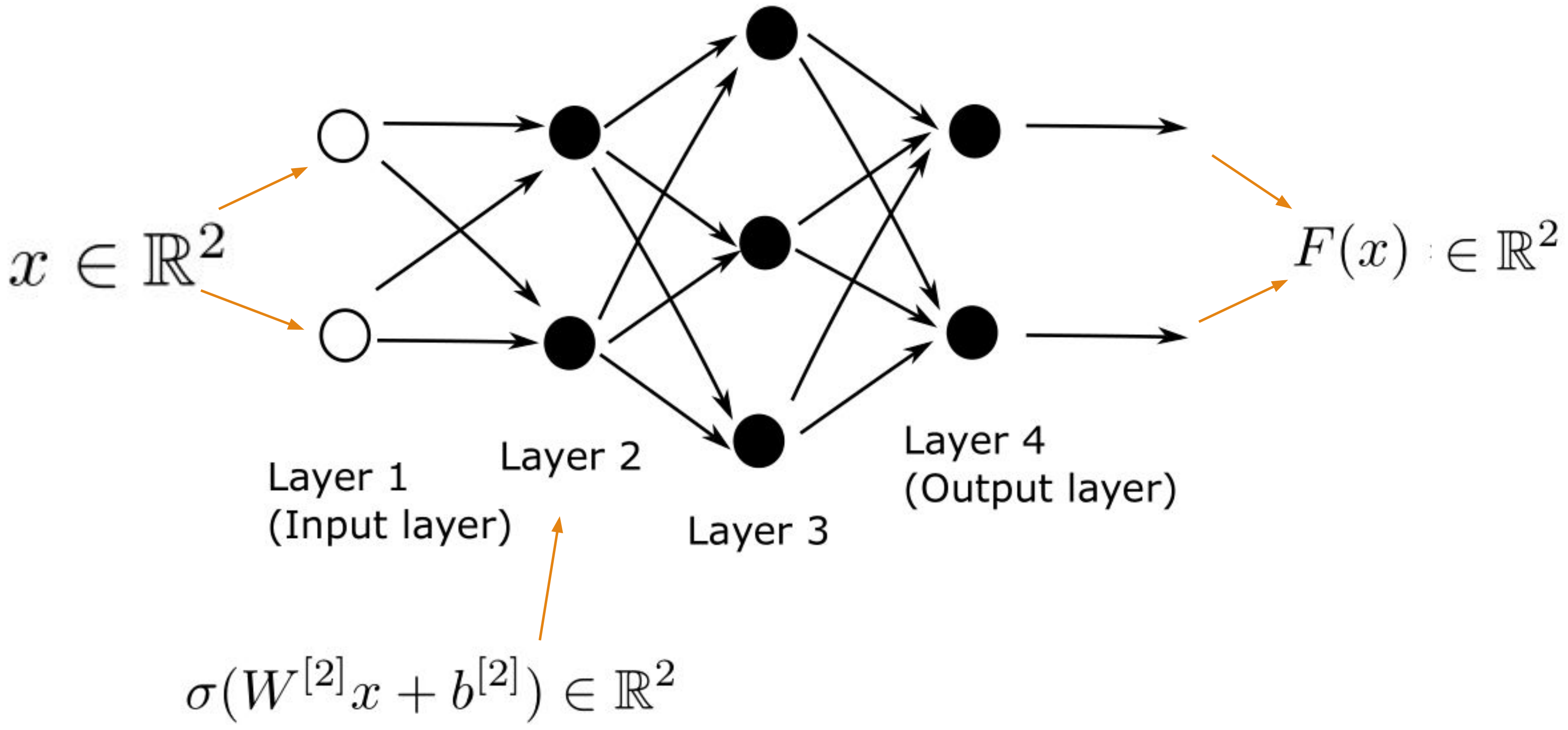
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

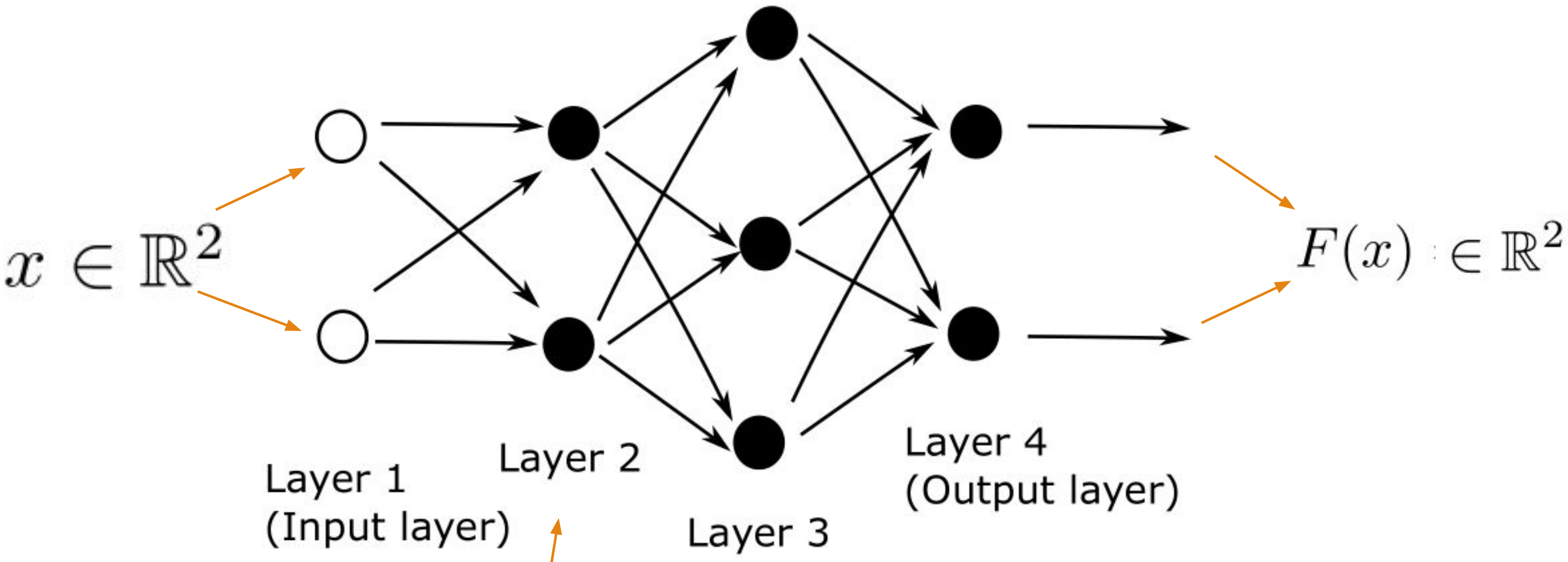
(note that the convention of the index is a little different here)



Example

We will use an example from this paper.

(note that the convention of the index is a little different here)



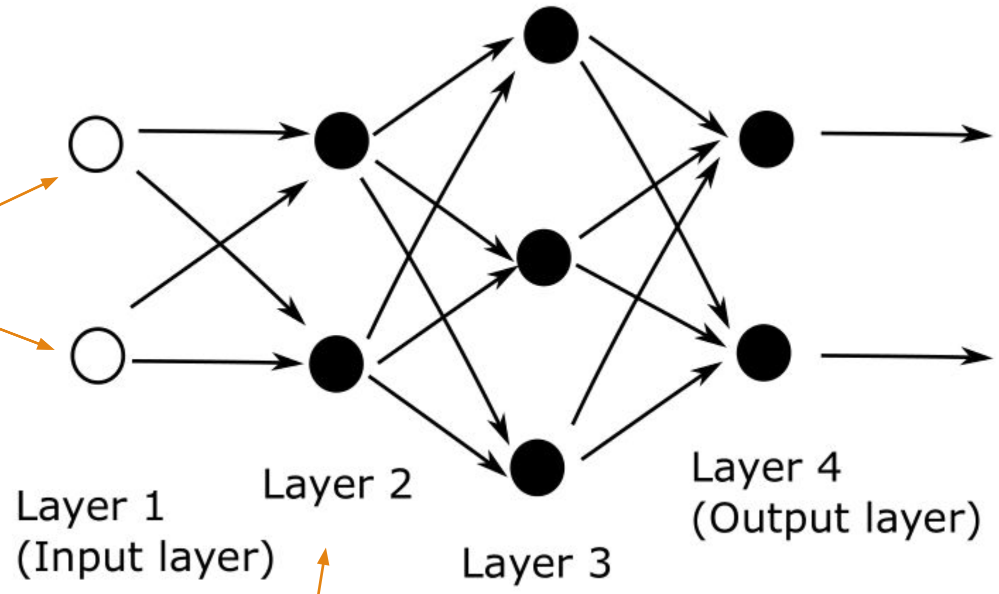
$$\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2$$

$$\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) \in \mathbb{R}^3$$

Example

We will use an example from <u>this</u> paper.

(note that the convention of the index is a little different here)



$x \in \mathbb{R}^2$

$F(x) \in \mathbb{R}^2$

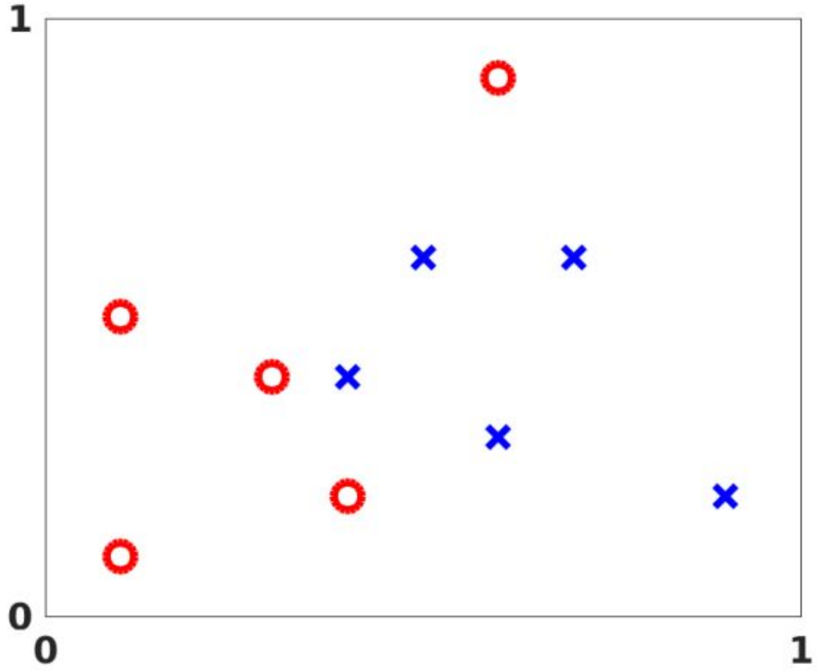
$\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2$

$\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) \in \mathbb{R}^3$

Final function representing the neural network

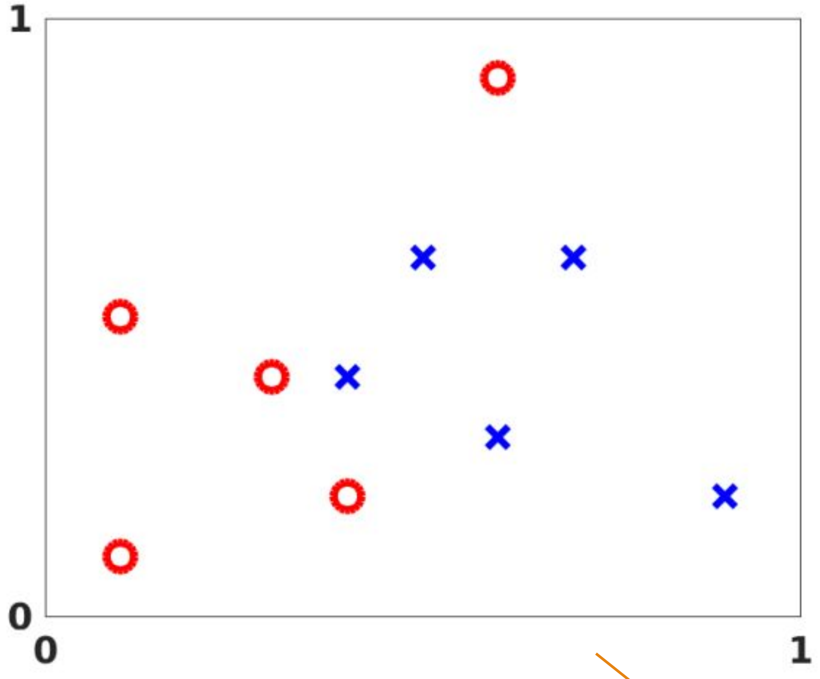
$$F(x) = \sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) + b^{[4]}\right) \in \mathbb{R}^2.$$

Example



Input : labeled data X

Example

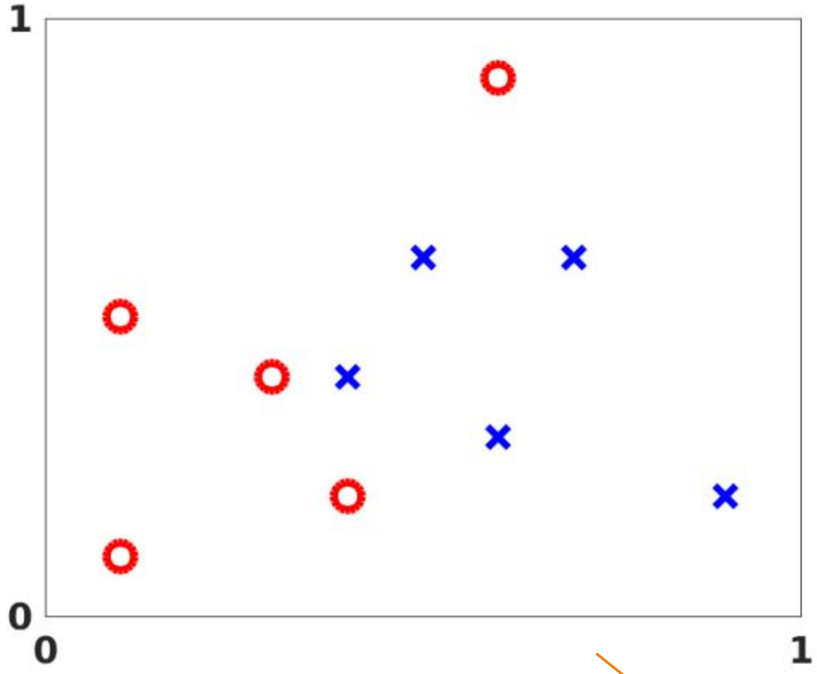


Input : labeled data X

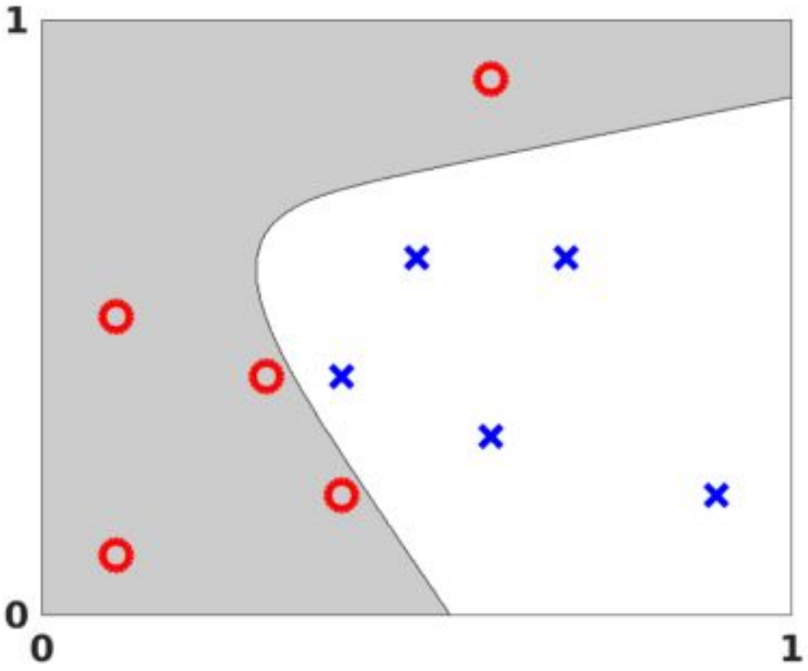
$$\mathrm{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \tfrac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2.$$

the difference between the output given by the network and the actual label

Example



Input : labeled data X



Minimize the cost function

$$\mathrm{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \frac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2.$$

the difference between the output given by the network and the actual label

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron (in the context of binary classification) is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify arbitrary binary-labeled data.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron (in the context of binary classification) is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify arbitrary binary-labeled data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron (in the context of binary classification) is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify arbitrary binary-labeled data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron (in the context of binary classification) is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify arbitrary binary-labeled data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

How do we adjust the weights ? As before, we define a notion of cost function (which will be a function with respect to all weights in the neural network) and then we try to minimize that function using the gradient descent algorithm

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron (in the context of binary classification) is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify arbitrary binary-labeled data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

How do we adjust the weights ? As before, we define a notion of cost function (which will be a function with respect to all weights in the neural network) and then we try to minimize that function using the gradient descent algorithm

Watch this video

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron (in the context of binary classification) is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify arbitrary binary-labeled data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

How do we adjust the weights ? As before, we define a notion of cost function (which will be a function with respect to all weights in the neural network) and then we try to minimize that function using the gradient descent algorithm

Watch this video

The process of updating the weights for a feedforward neural network is called *backpropagation*.