Basics of deep learning part 1 and 2

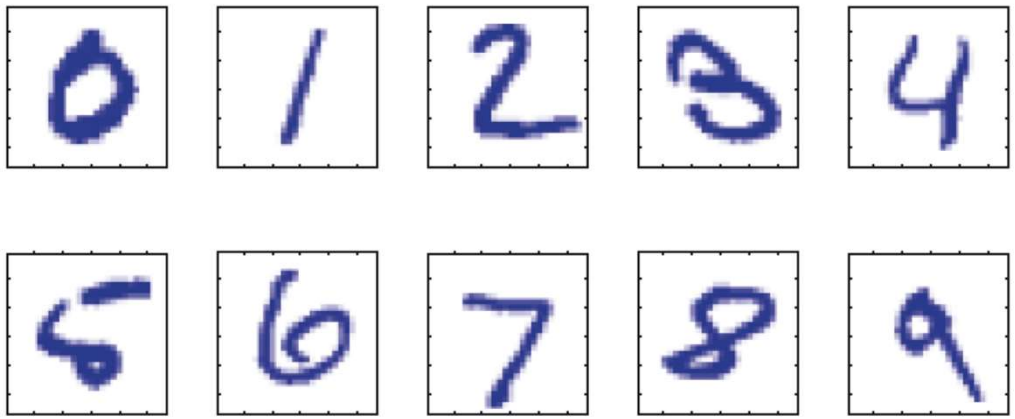
By

Pierre-Marc Jodoin

UNIVERSITÉ DE SHERBROOKE

What is machine learning?

Question : how can one recognize hand written digits?

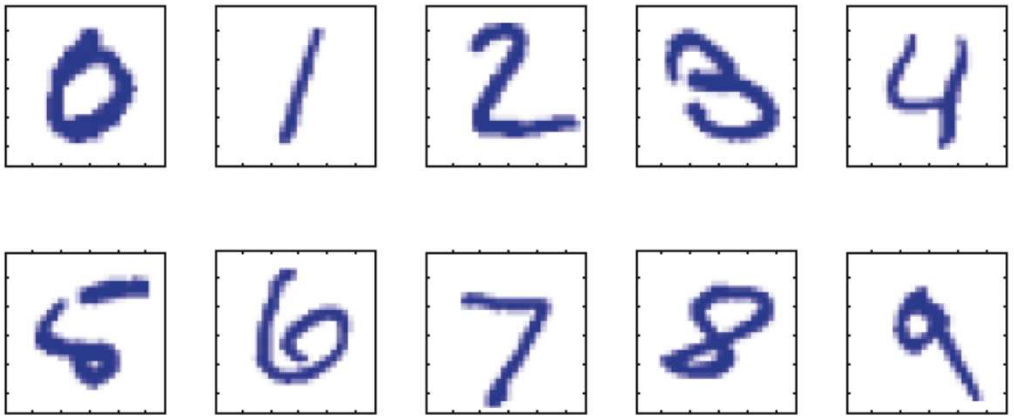


Answer : Design your own rules?

- A series of aligned pixels => ‘1’
- A circle of pixels => ‘0’
- Etc.

(Hugo Larochelle)

Question : how can one recognize hand written digits?

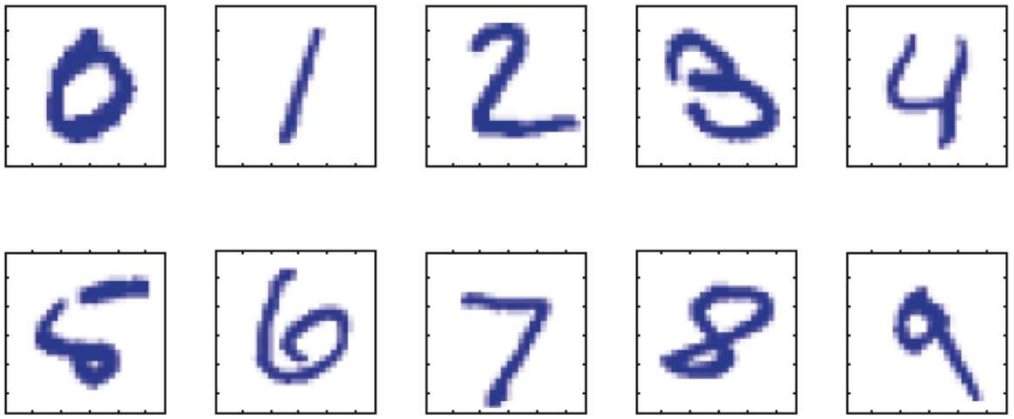


Answer : ~~Design your own rules?~~ **Wrong**

- Bad generalization

(Hugo Larochelle)

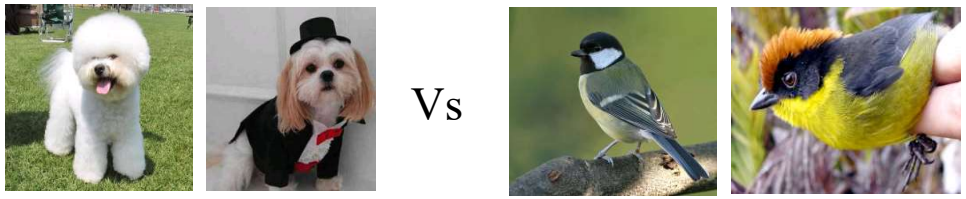
Question : how can one recognize hand written digits?



Answer : ~~Design your own rules?~~ **Wrong**

- Bad generalization

- Often difficult



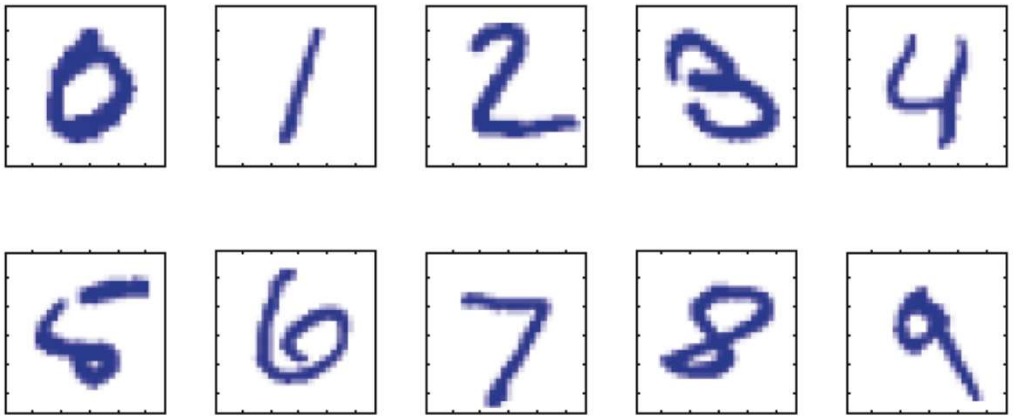
Vs

Dogs

Birds

(Hugo Larochelle)

Question : how can one recognize hand written digits?

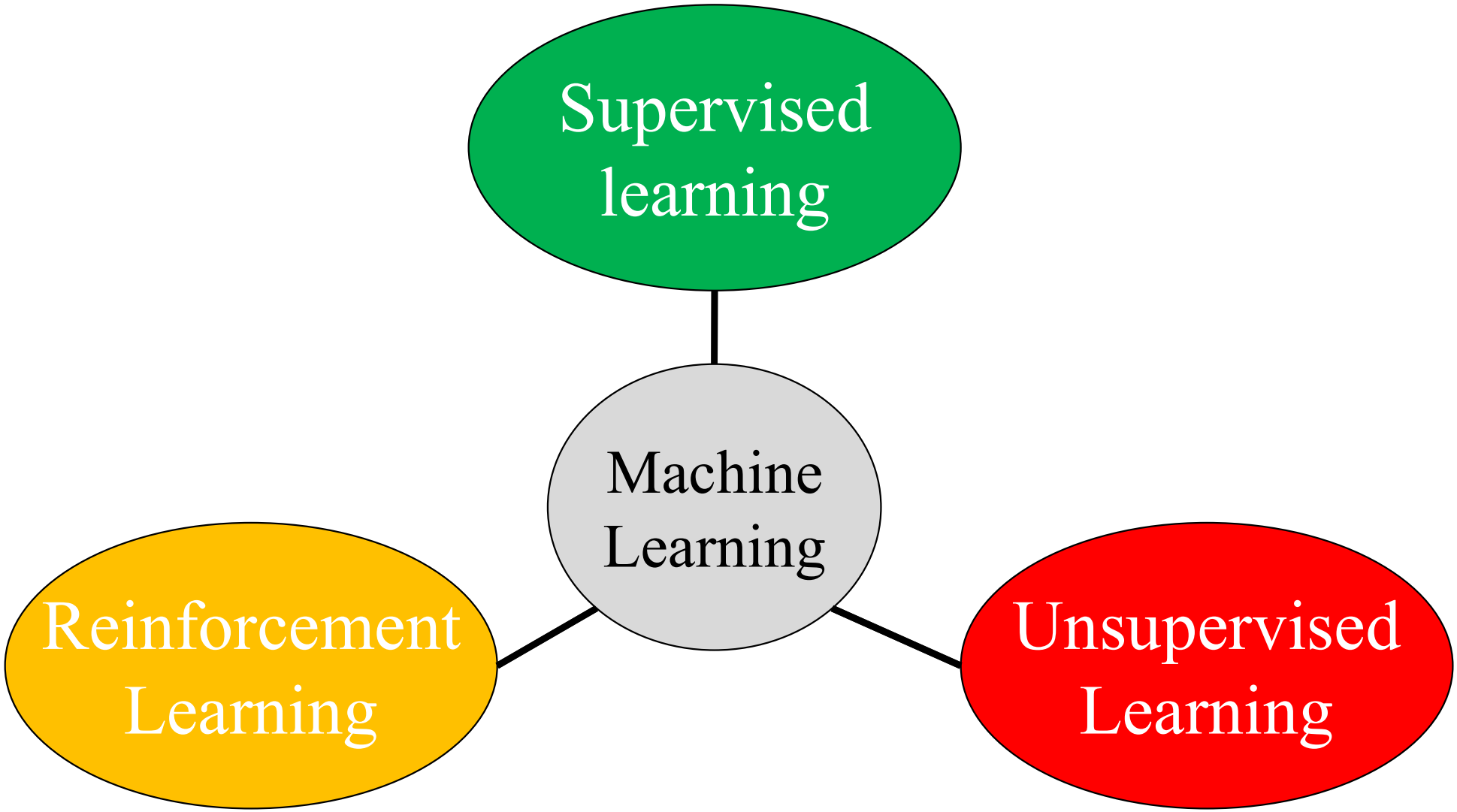


Answer : Let the computer « **learn** » the rules

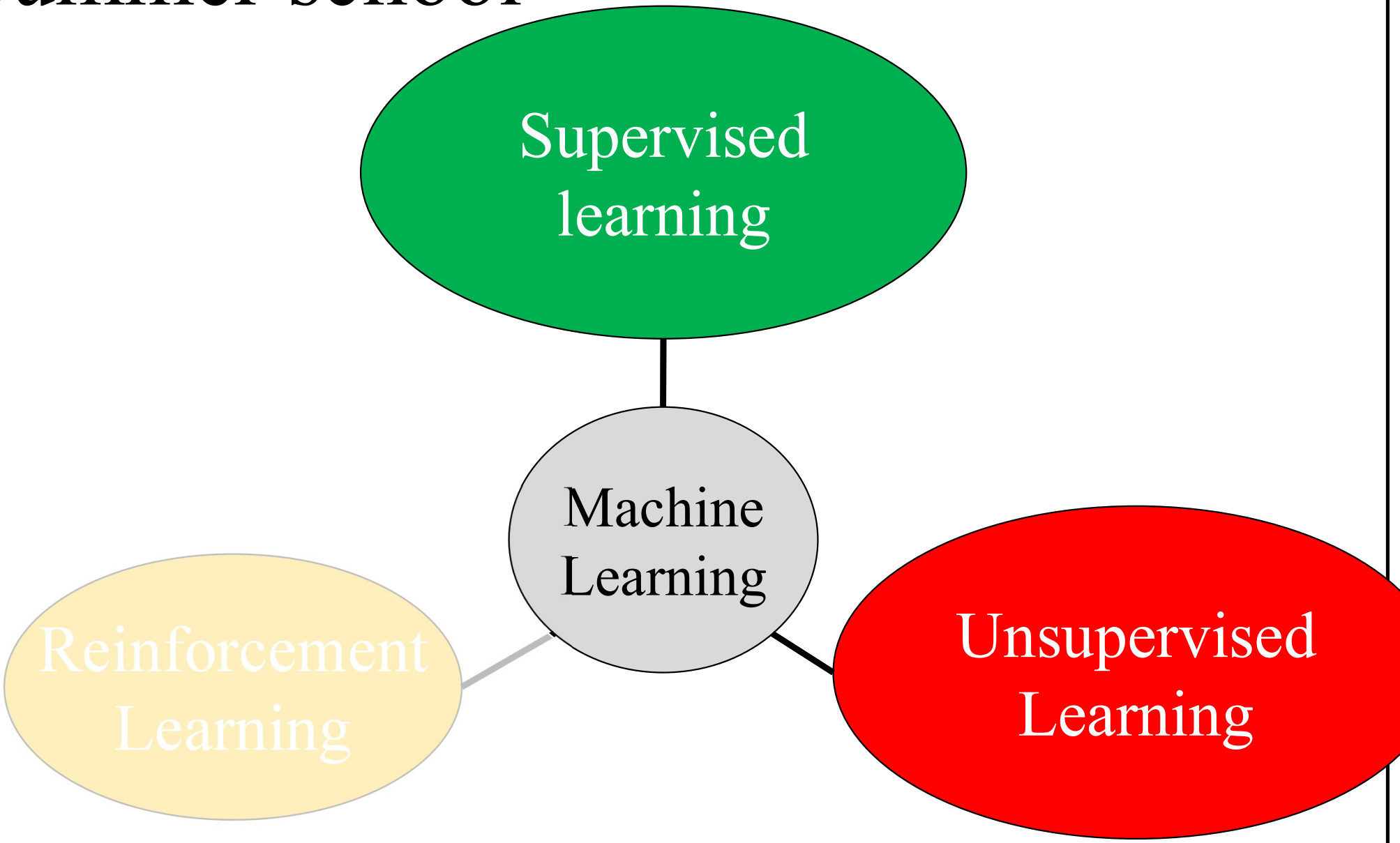
- *Main goal of machine learning*

(Hugo Larochelle)

Three large families



Summer school



Supervised learning

Provide the algorithm with **annotated training data**

‘0’ ‘0’ ‘0’ ‘1’ ‘0’ ‘1’ ‘1’

…and the algorithm returns a function capable of **generalizing** on new data

? ? ? ? ? ? ?

Supervised learning

Provide the algorithm with **annotated training data**

‘0’ ‘0’ ‘0’ ‘1’ ‘0’ ‘1’ ‘1’

The **training dataset**

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$$

where $\vec{x}_i \in \Re^d$ is an **input** and t_i is a **target**

Goal of a supervised machine learning method

From a **training dataset:** $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$

the goal **is to learn** a function that may predict t_i given $\vec{x}_i$

$$y_W(\vec{x}_i) \rightarrow t_i$$

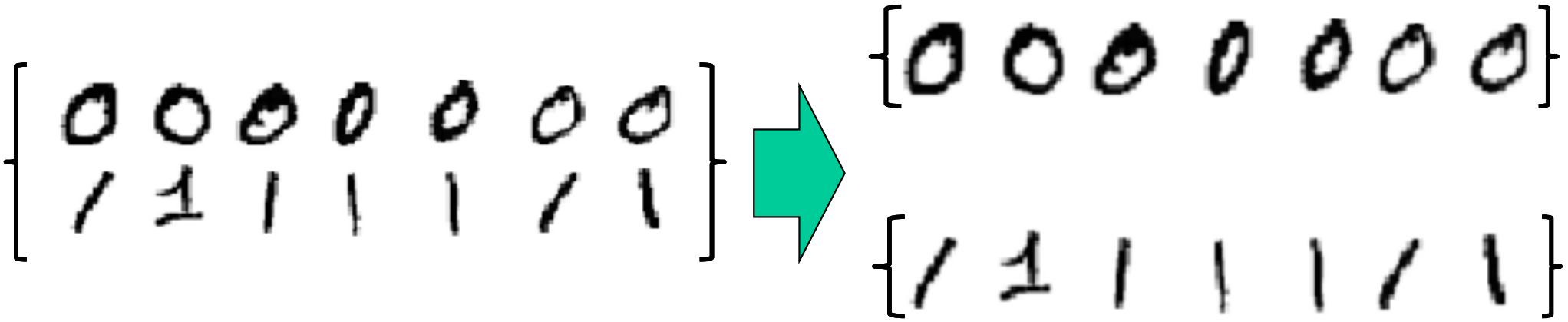
where W are the **parameters** of the model.

Unsupervised learning

When no target is explicitly provided

- E.g. data *clustering*

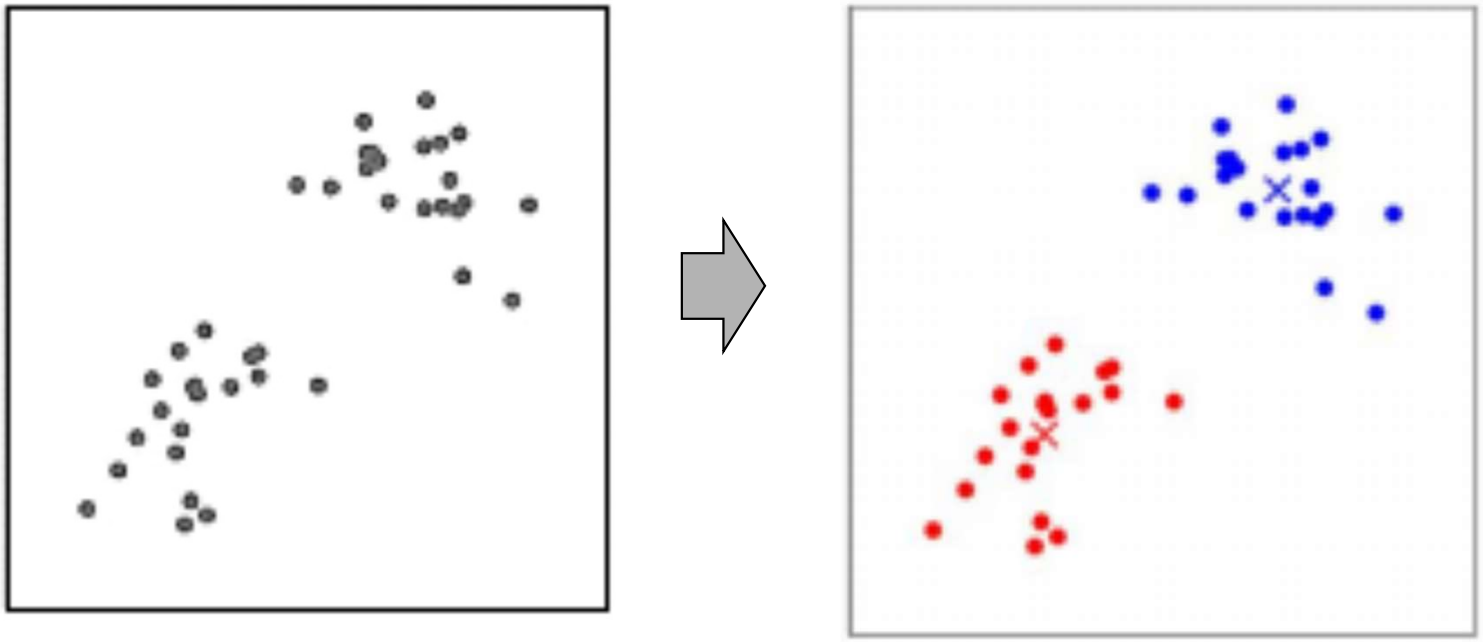
D



Unsupervised learning

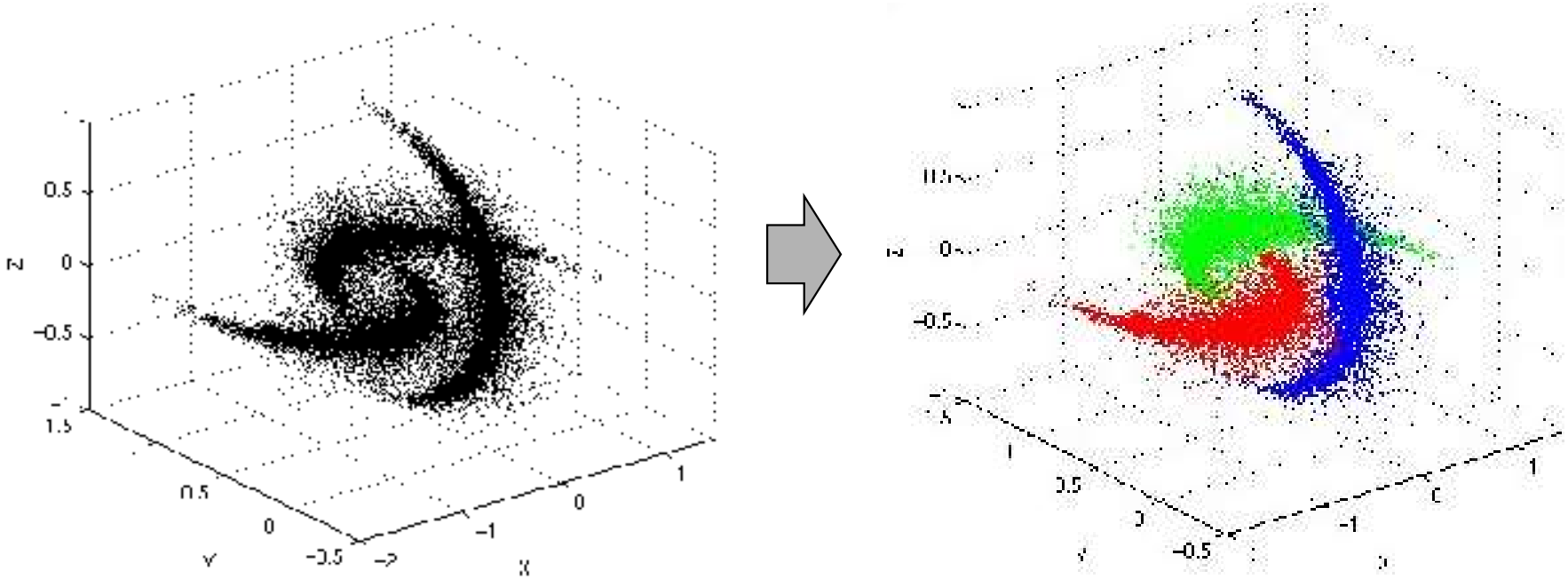
When no target is explicitly provided

- E.g. data *clustering*



Unsupervised learning

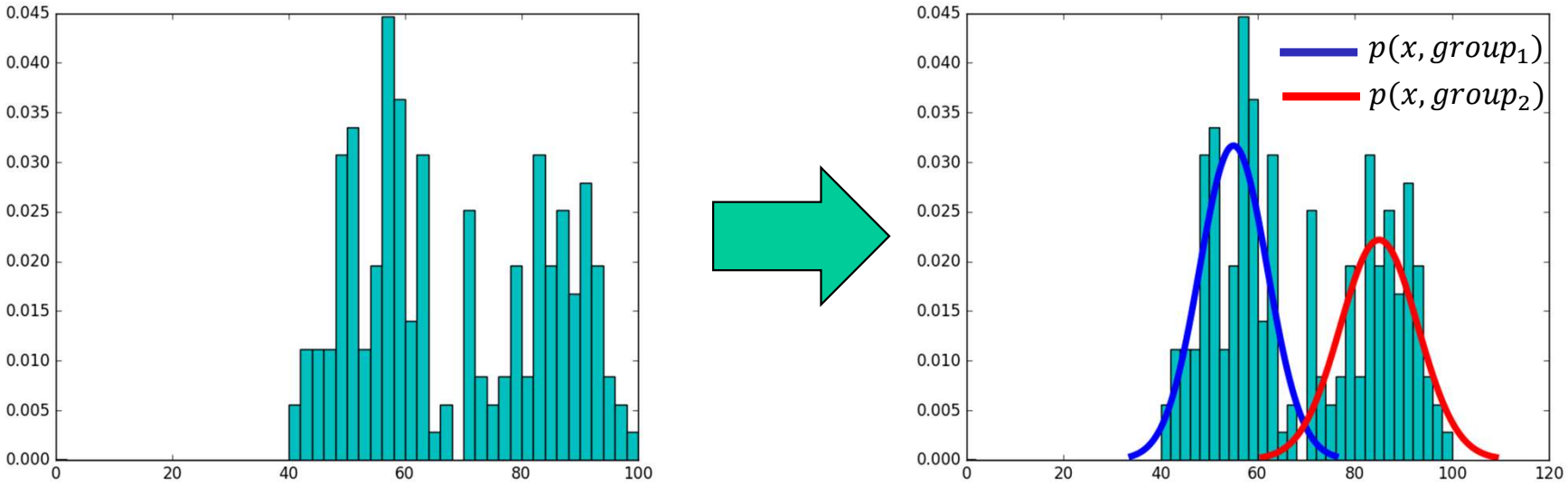
No limit to dimensionality. Could be 3D, 4D,…100kD



Unsupervised learning

Probability density function estimation

Example : find two groups of patients following a memory test



Supervised vs non-supervised

Supervised learning : there is a target

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$$

Unsupervised learning : unknown target

$$D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$$

Supervised vs non-supervised

Supervised learning : there is a tar

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2)$$

Logistic regression
Perceptron
Multilayer perceptron
Convolutional neural networks
Semi-supervised learning
Graph Neural Nets
Transformers
Etc.

Unsupervised learning : unknown target

$$D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$$

Supervised vs non-supervised

Supervised learning : there is a target

$$D = \left\{ \left(\vec{x}_1, t_1 \right), \left(\vec{x}_2, t_2 \right), \ldots, \left(\vec{x}_N, t_N \right) \right\}$$

Unsupervised learning : unknown

$$D = \left\{ \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N \right\}$$

Autoencoders
Variational autoencoders
GANs
Diffusion networks

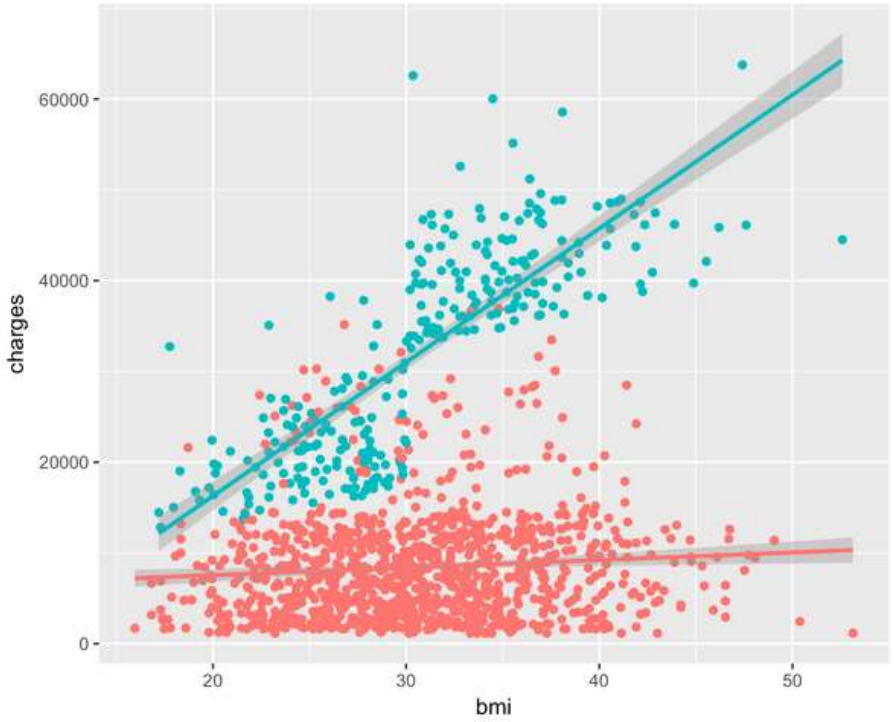
Supervised learning

Classification vs regression

Supervised learning
(linear models)

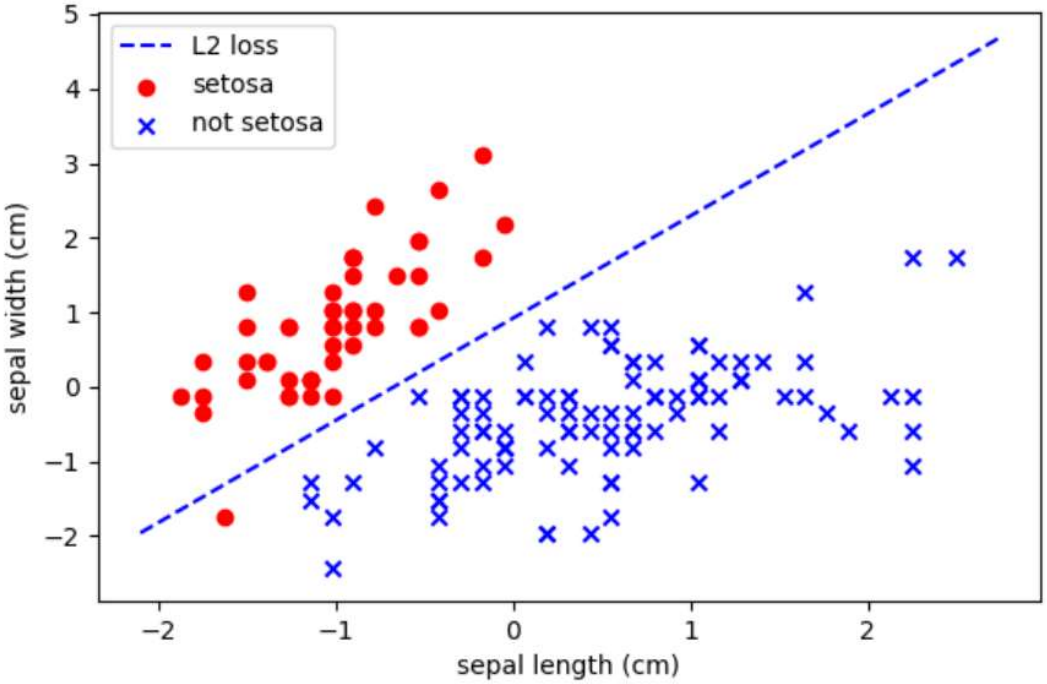
Two main applications

Regression



https://rpubs.com/koki25ando/medicalcost

Classification (2 classes)

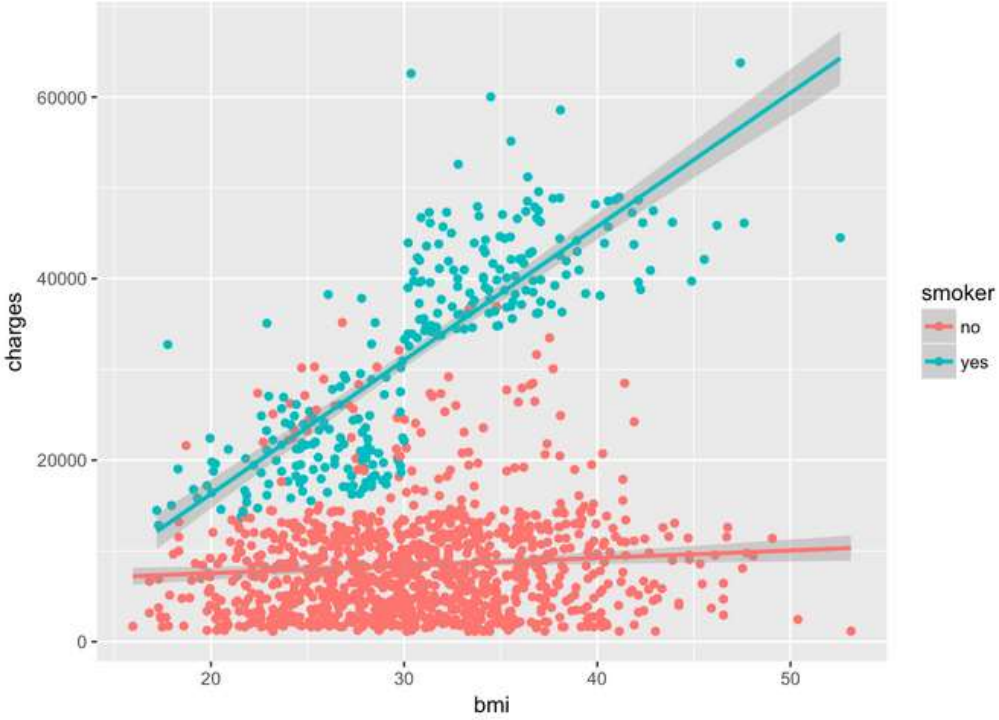


https://winder.ai/403-linear-classification/

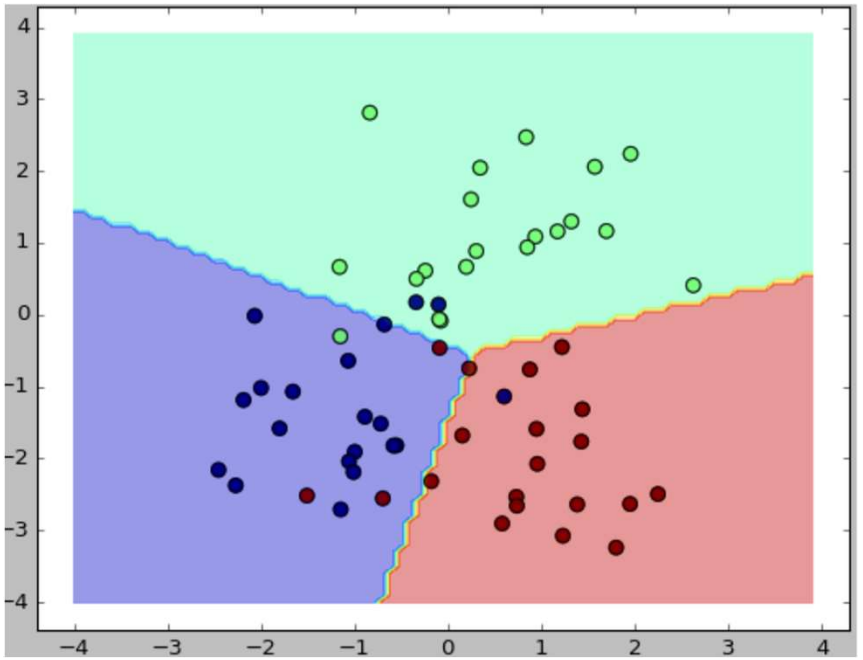
Supervised learning
(linear models)

Two main applications

Regression



Classification (3 classes)

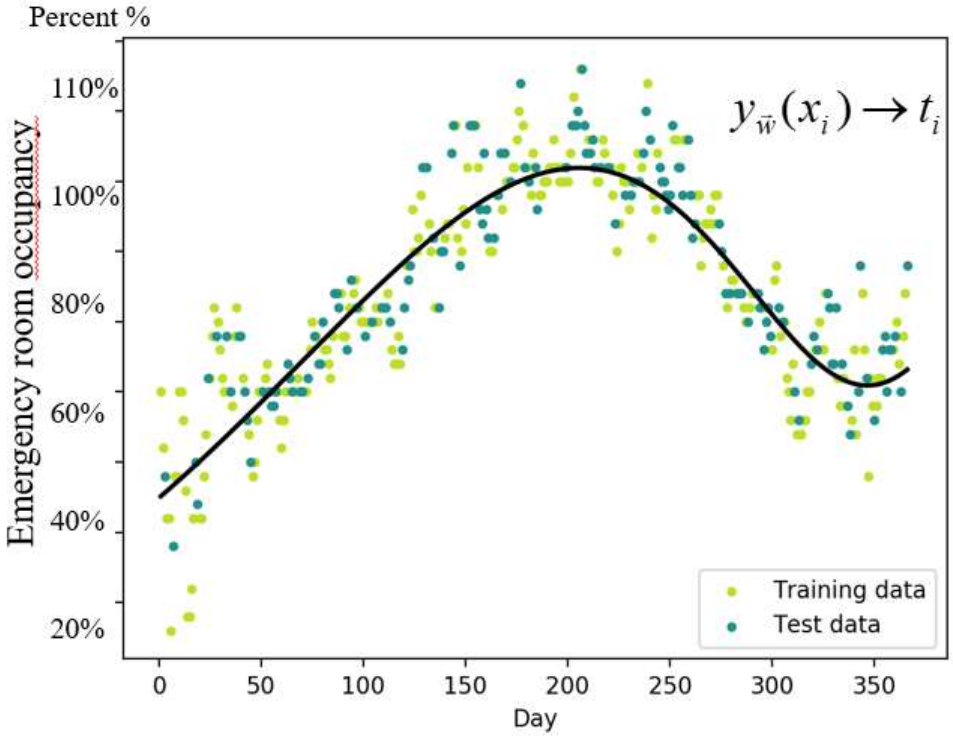


https://rpubs.com/koki25ando/medicalcost

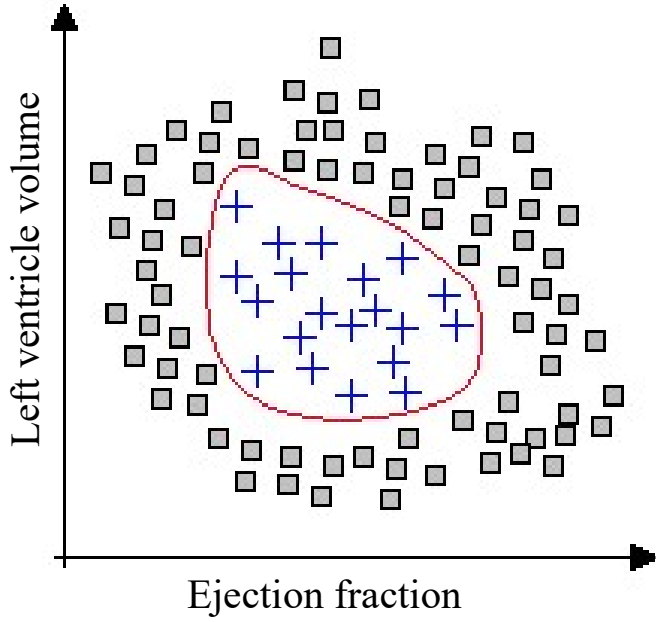
Supervised learning
(non-linear models)

Two main applications

Regression

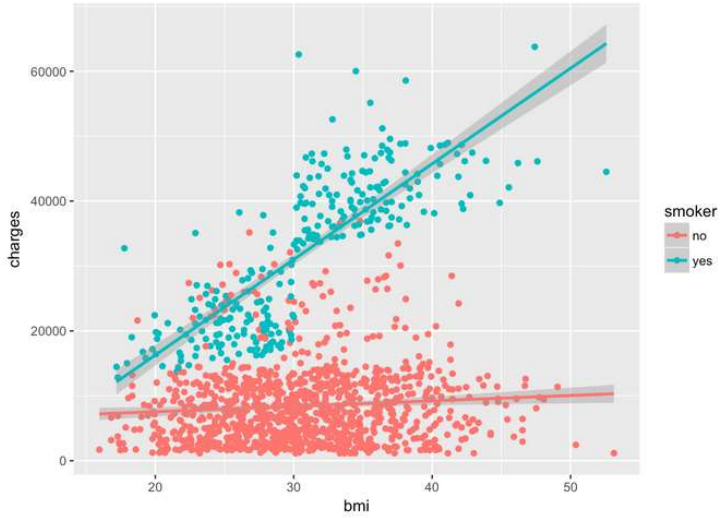


Classification (2 classes)

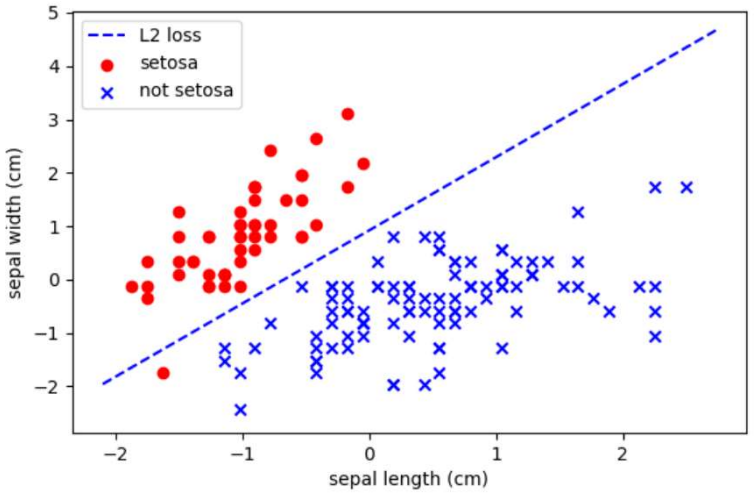


http://www.statistics4u.com/fundstat_eng/cc_classif_calib.html

Let's start with **linear models**



https://rpubs.com/koki25ando/medicalcost



https://winder.ai/403-linear-classification/

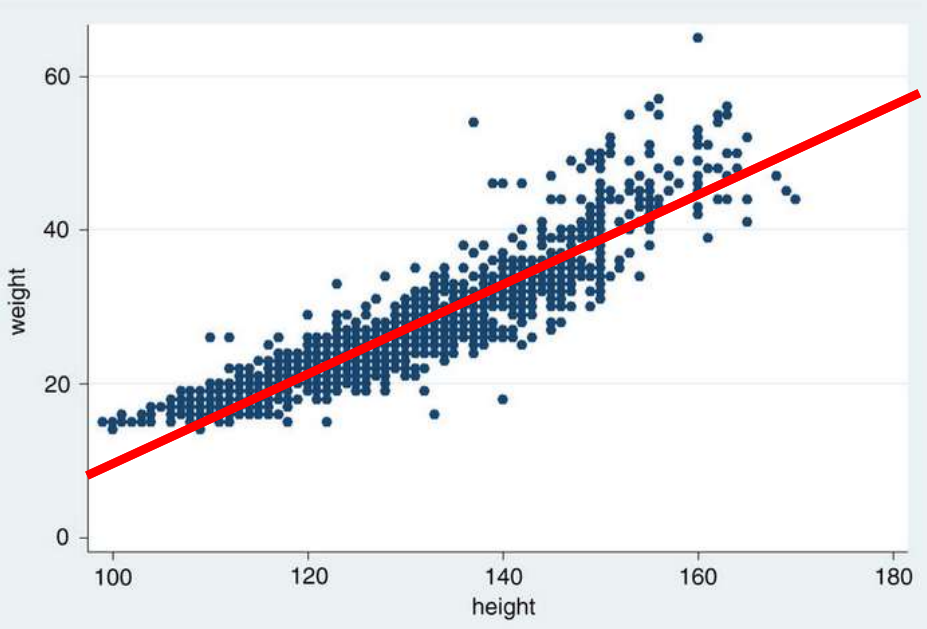
Deep neural nets … **linear models**

Vs
?

Linear models are to deep neural nets what transistors are to modern processors

Linear models are still relevant

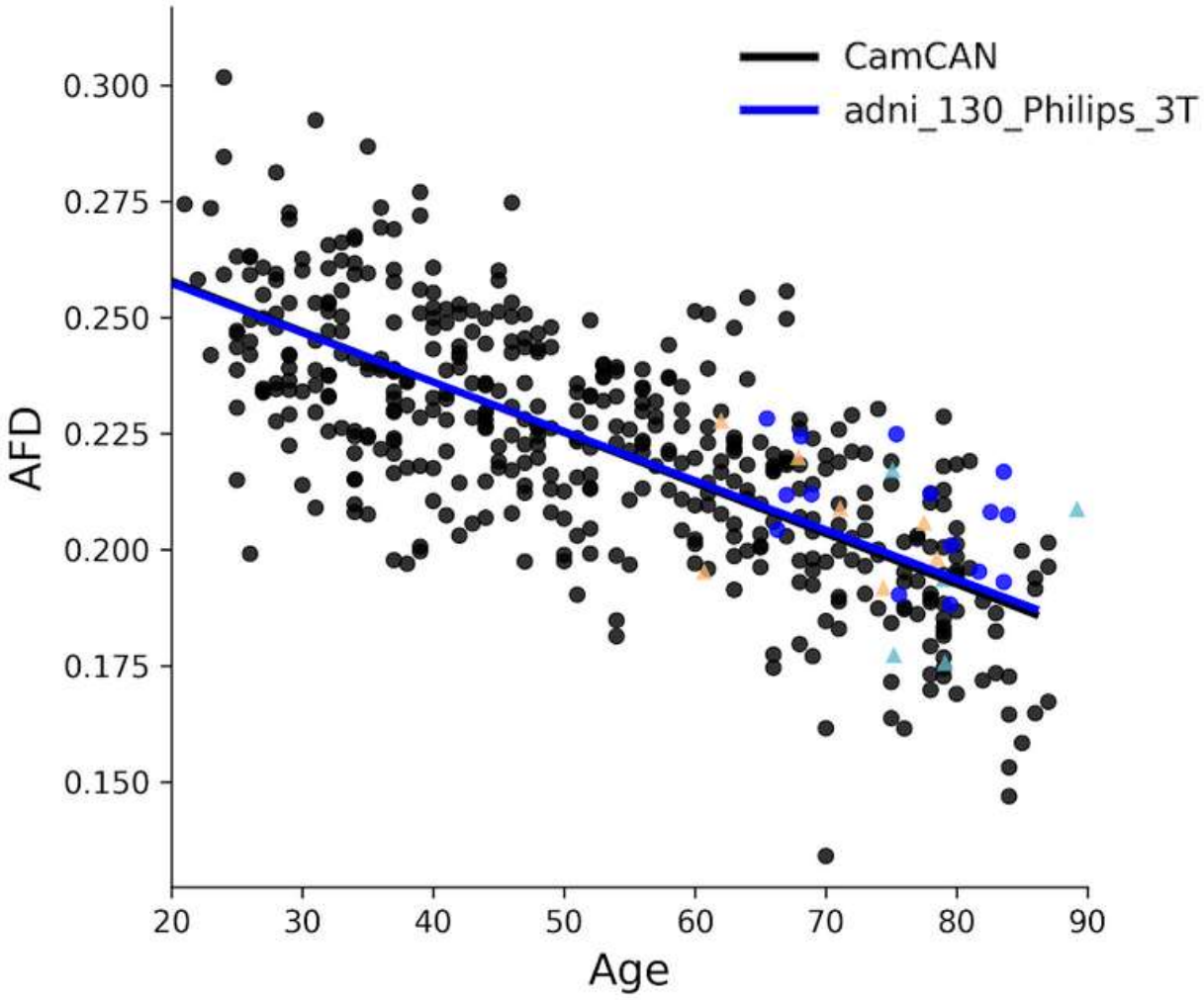
1,694 children surveyed in Tanzania.

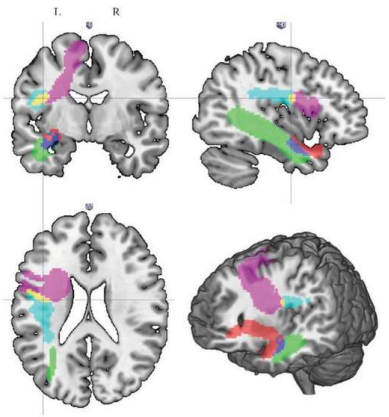


Nordin P, Poggensee G, Mtweve S, Krantz I**. From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

Linear models are still relevant

Apparent Fiber Density
in the white matter



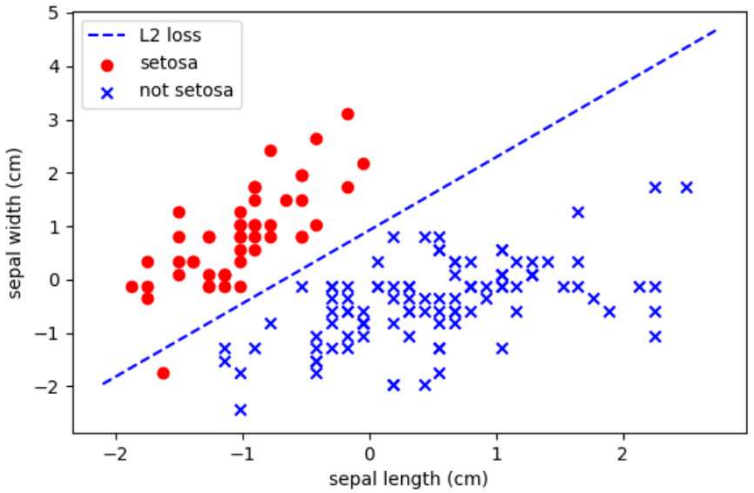


https://commons.wikimedia.org/

Linear models

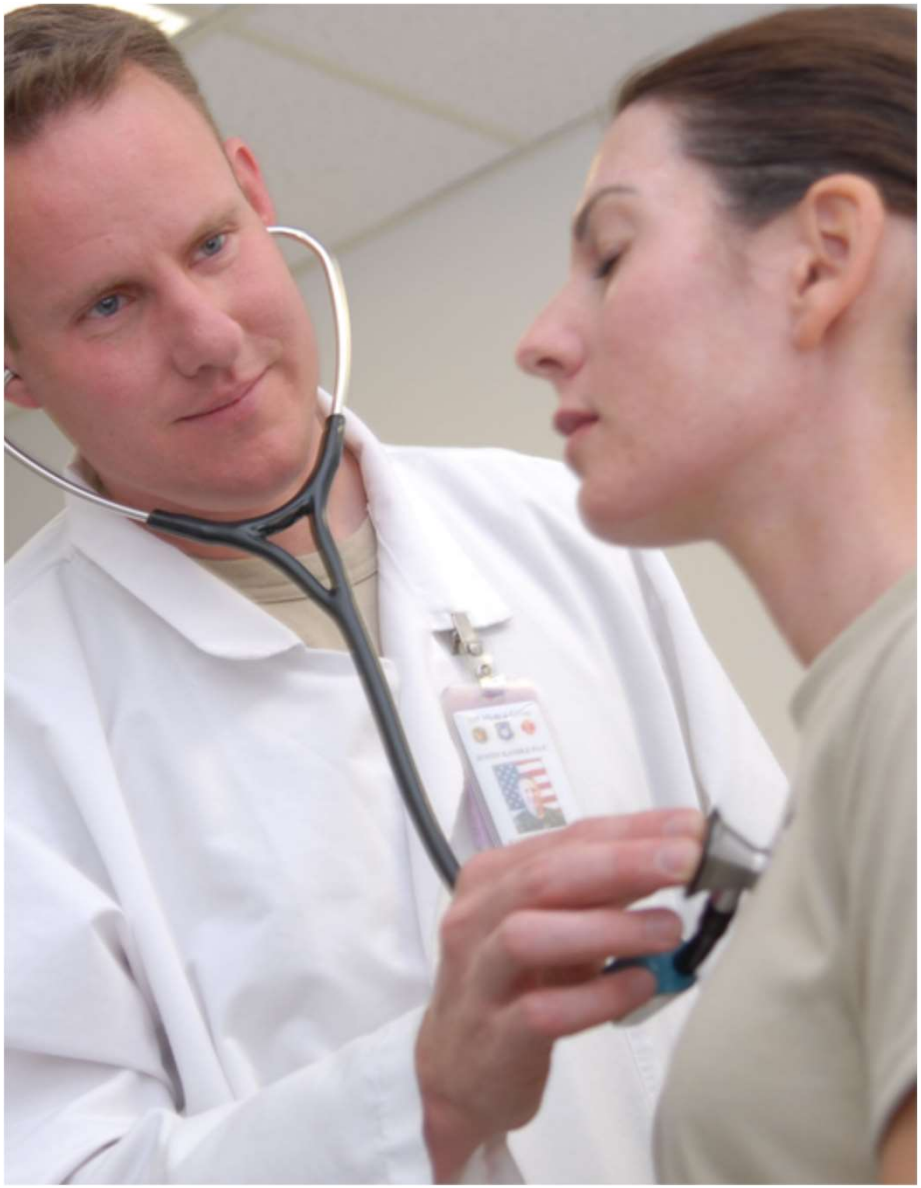
https://rpubs.com/koki25ando/medicalcost

Classification



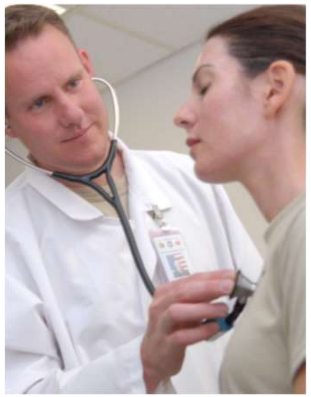
https://winder.ai/403-linear-classification/

Simple example of binary classification



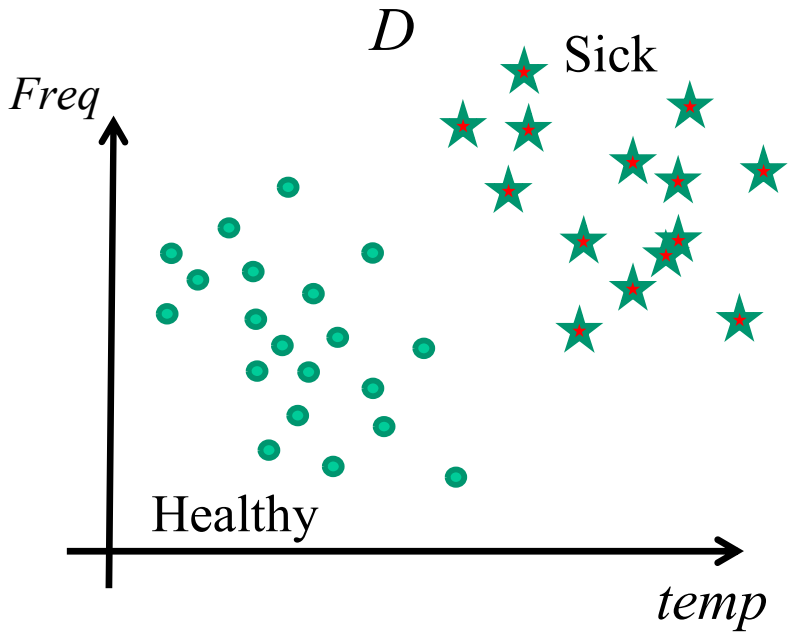
From Wikimedia Commons
the free media repository

Simple example of binary classification



D

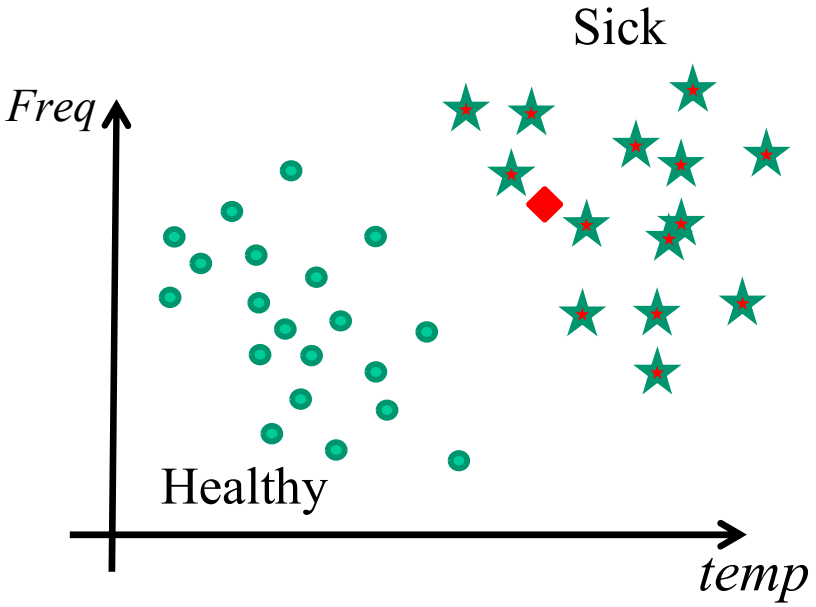
	(temp, freq)	Diagnostic
Patient 1	(37.5, 72)	hearthy
Patient 2	(39.1, 103)	sick
Patient 3	(38.3, 100)	sick
	(…)	…
Patient N	(36.7, 88)	hearthy
	$\vec{x}$	t



Simple example of binary classification

A new patient shows up at the hospital
How can we predict its state?

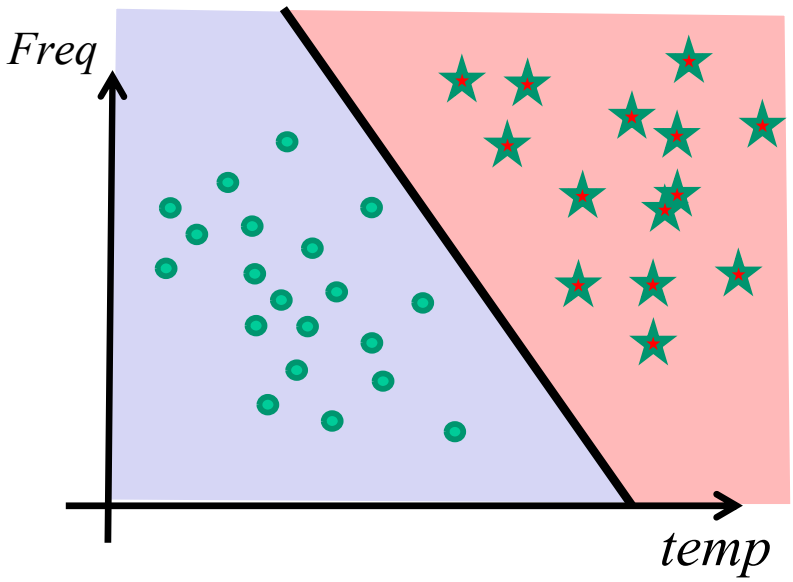
From Wikimedia Commons
the free media repository



Solution

From Wikimedia Commons
the free media repository

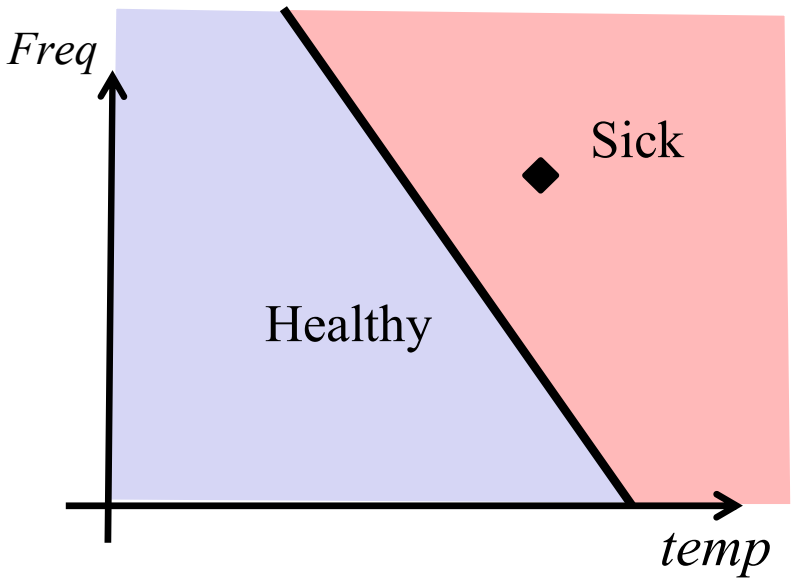
Divide the feature space in two regions : healthy and sick



Solution

From Wikimedia Commons
the free media repository

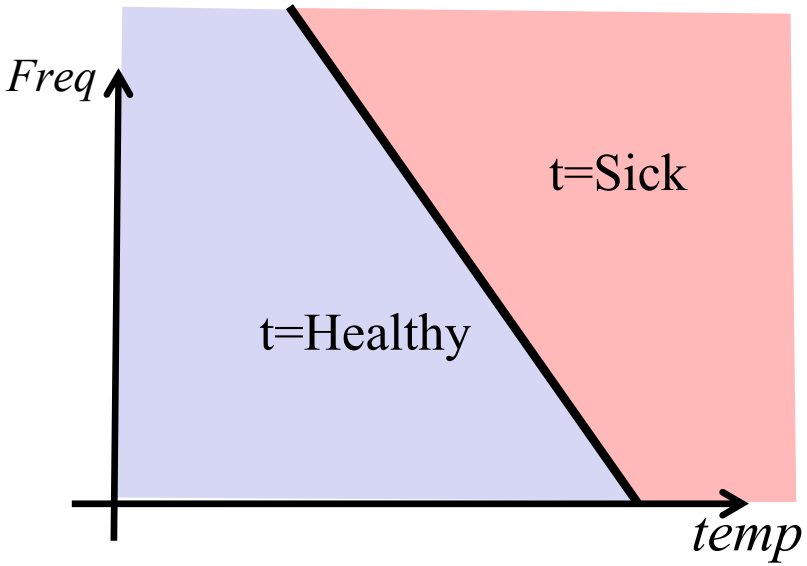
Divide the feature space in two regions : healthy and sick



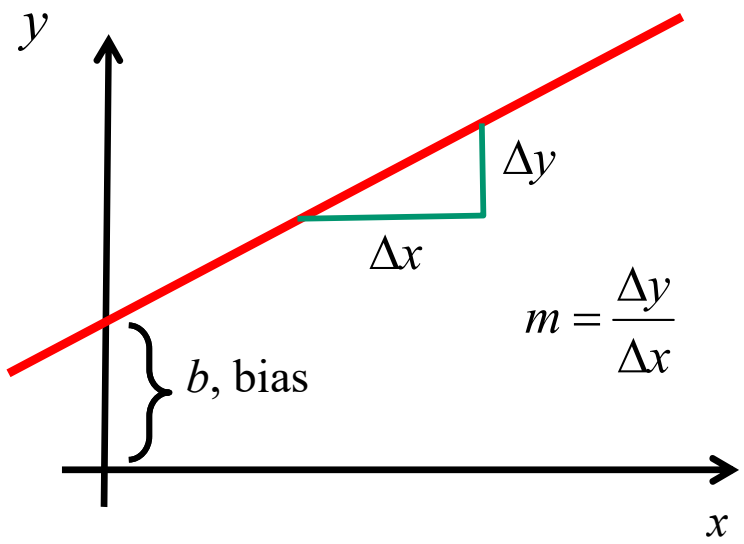
More formally

From Wikimedia Commons
the free media repository

$$y_W(\vec{x}) = \begin{cases} Healthy \text{ if } \vec{x} \text{ is in the blue region} \\ Sick \quad \text{otherwise} \end{cases}$$



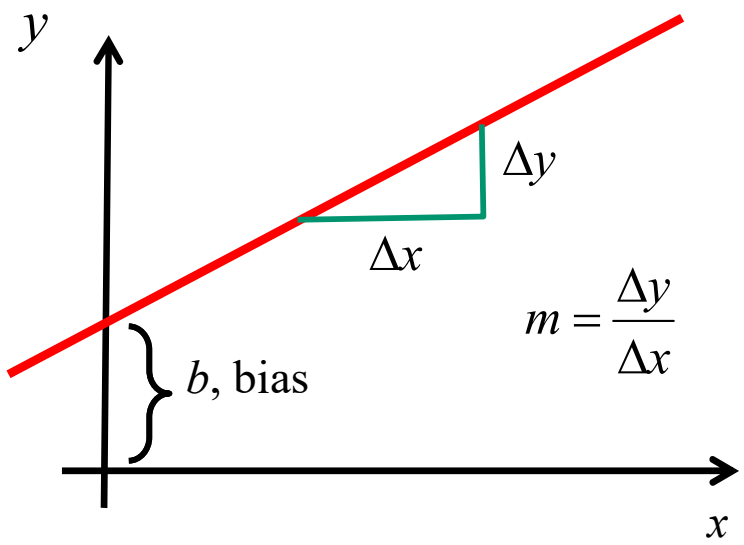
Definition … a line!



$$y = mx + b$$

slope bias

Definition … a line!



$$y = mx + b$$

$$y = \frac{\Delta y}{\Delta x} x + b$$

$$y\Delta x = \Delta y x + b\Delta x$$

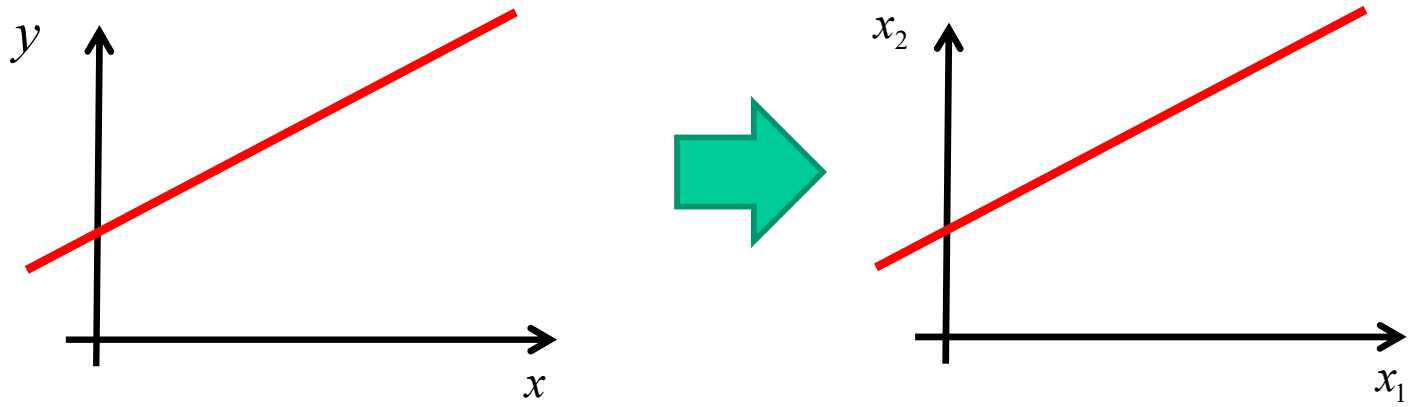
$$0 = \Delta y x - \Delta x y + b\Delta x$$

Rename variables

$$0 = \underbrace{\Delta y x}_{w_1} - \underbrace{\Delta x y}_{w_2} + \underbrace{b \Delta x}_{w_0}$$

Rename variables

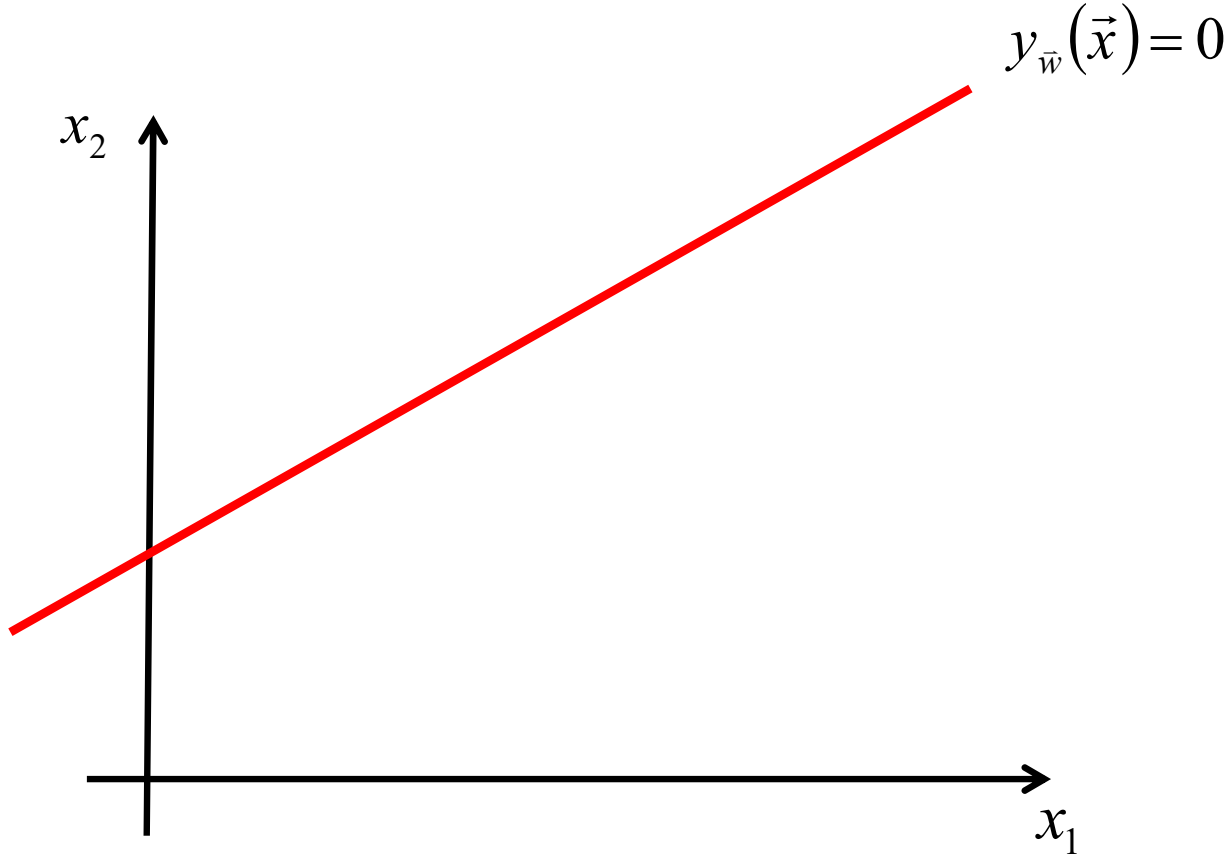
$$0 = w_1 x + w_2 y + w_0$$



$$0 = w_1 x_1 + w_2 x_2 + w_0$$

Implicit function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$



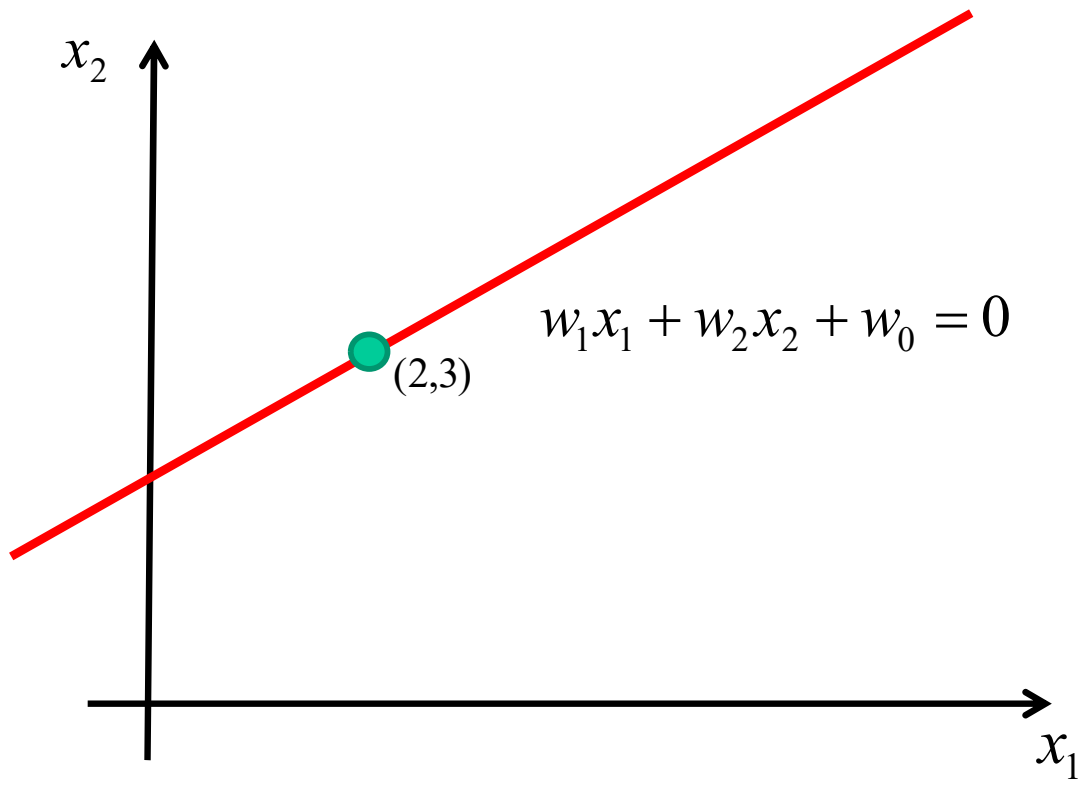
Implicit function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

$w_1 = 1.0$

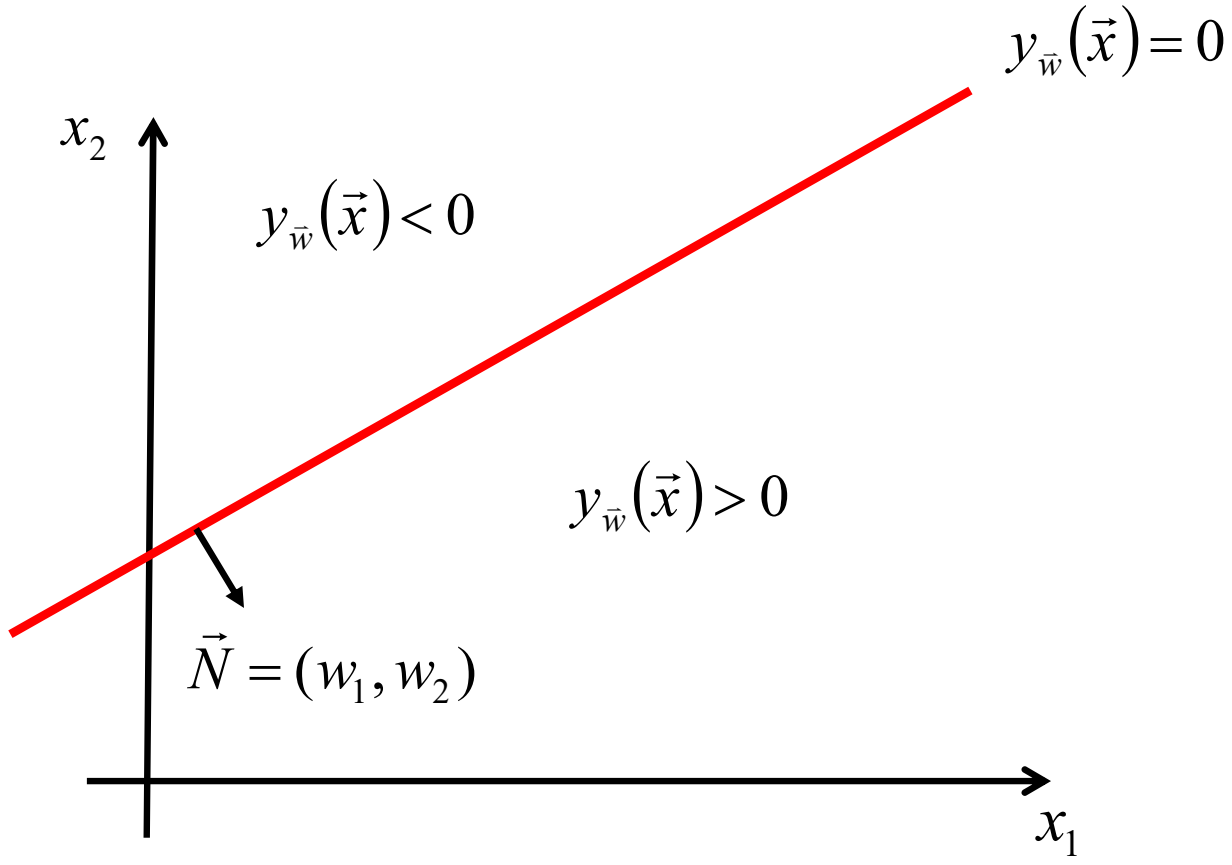
$w_2 = -2.0$

$w_0 = 4.0$



Classification function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$



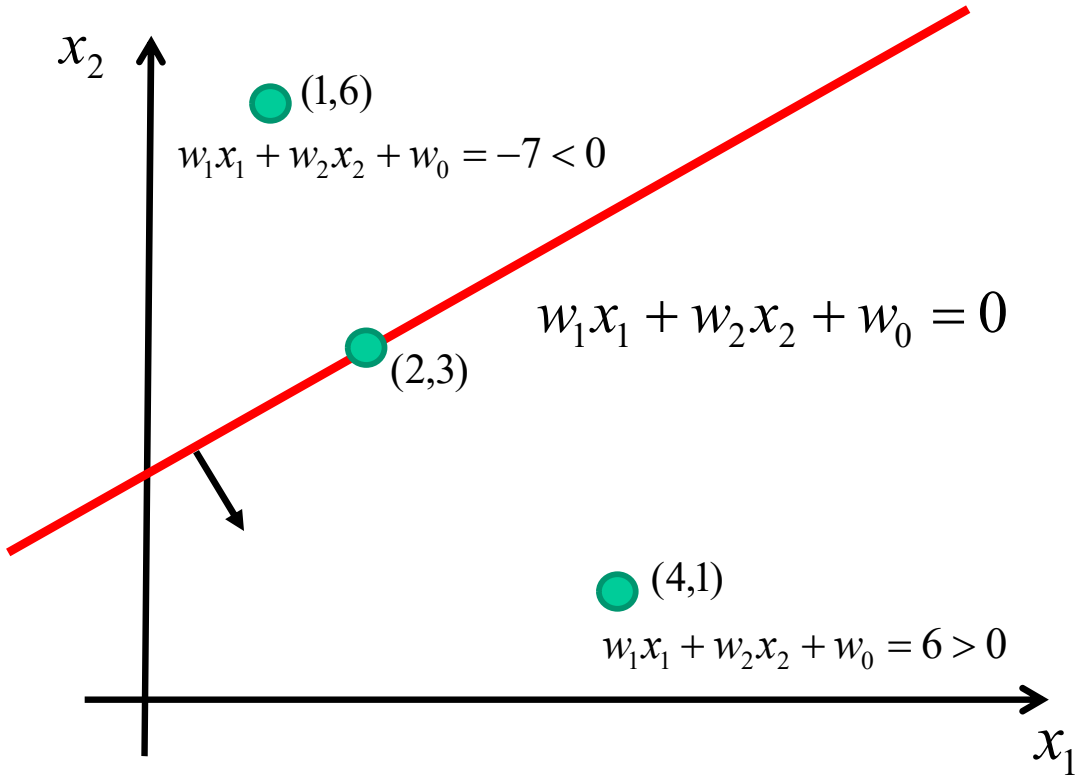
Classification function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

$w_1 = 1.0$

$w_2 = -2.0$

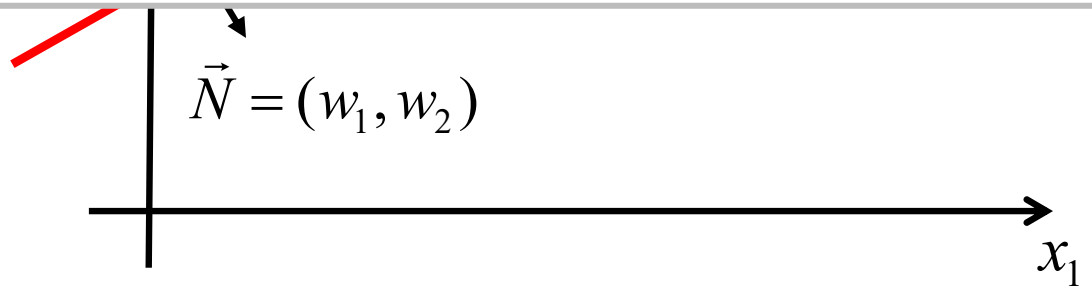
$w_0 = 4.0$



Implicit function

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$
$$= \underbrace{(w_0, w_1, w_2)}_{\vec{w}} \cdot \underbrace{(1, x_1, x_2)}_{\vec{x}}$$

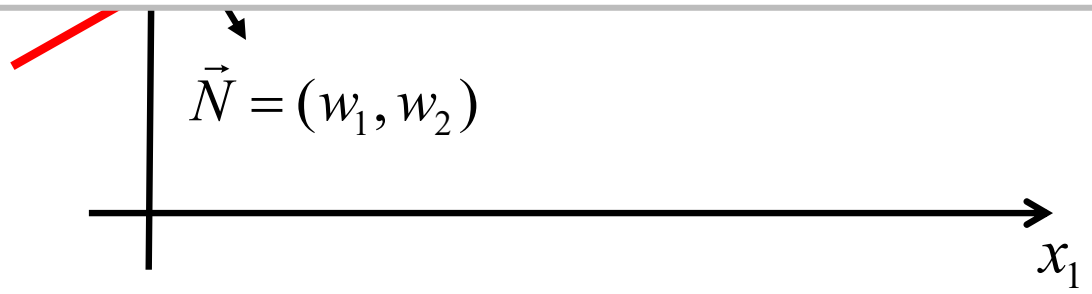
DOT product



Implicit function

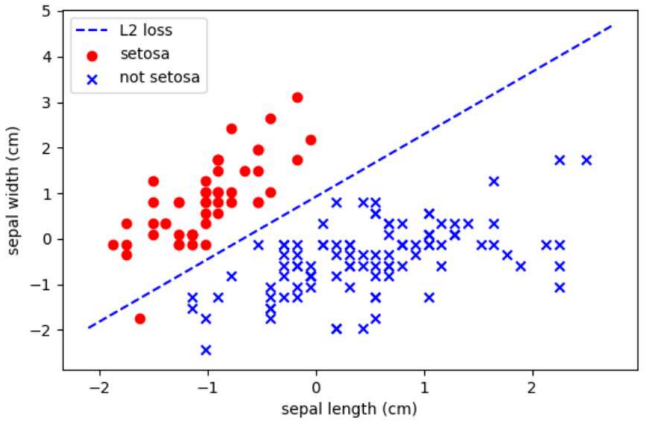
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$
$$= (w_0, w_1, w_2) \cdot (1, x_1, x_2)$$
$$= \vec{w}^T \vec{x}$$

DOT product

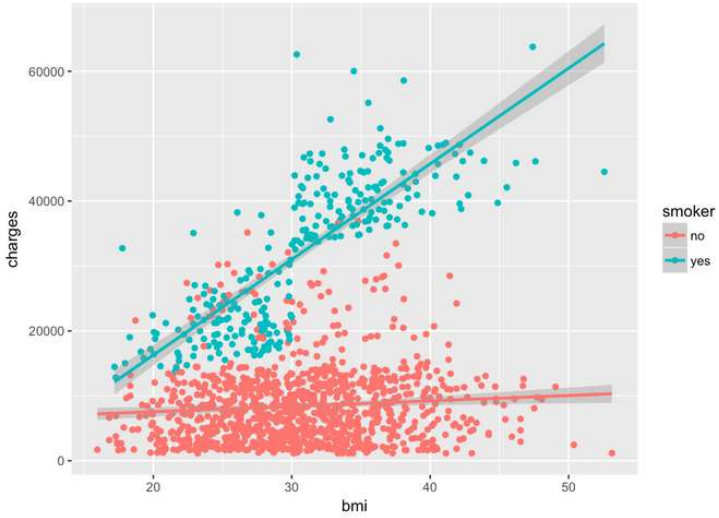


Linear classifier = dot product

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$



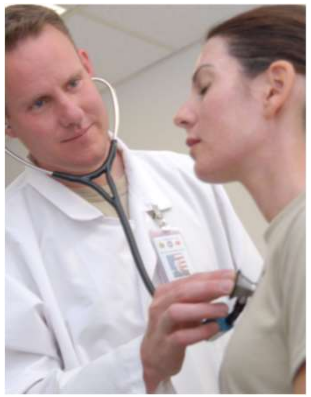
Linear models



https://rpubs.com/koki25ando/medicalcost

https://winder.ai/403-linear-classification/

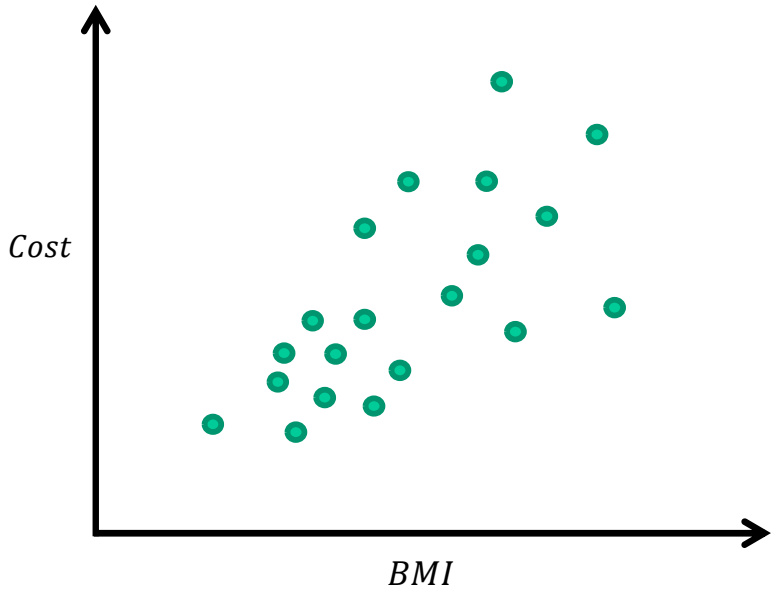
Simple example of linear regression



$D_{smokers}$

	(Body mass index)	Medical cost $
Patient 1	23	12,500
Patient 2	44	34,000
Patient 3	27	21,000
	(…)	…
Patient N	31	42,000
	x	t

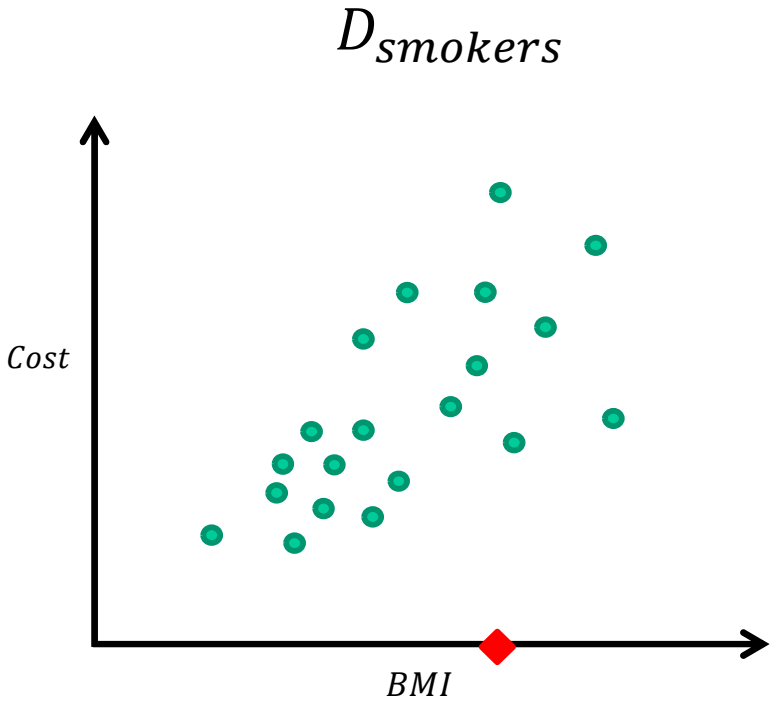
$D_{smokers}$



Simple example of binary classification

A new patient shows up at the hospital
How can we predict his/her medical cost?

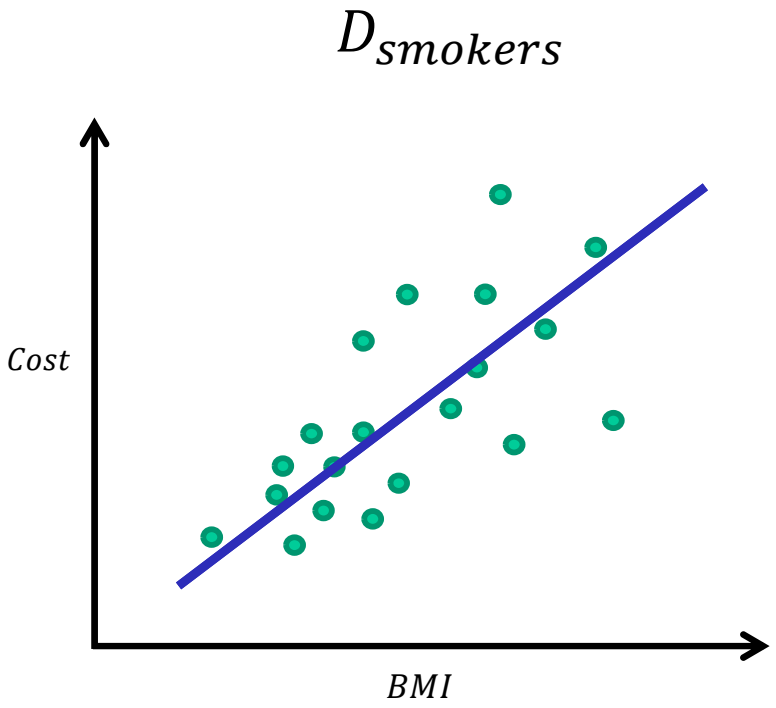
chatgpt.com



Solution

From Wikimedia Commons
the free media repository

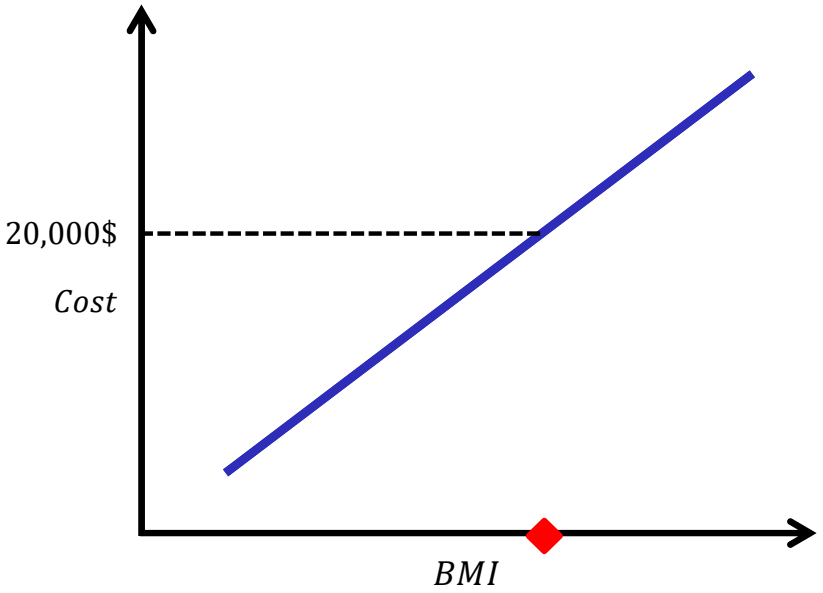
Fit a line onto the data



Solution

From Wikimedia Commons
the free media repository

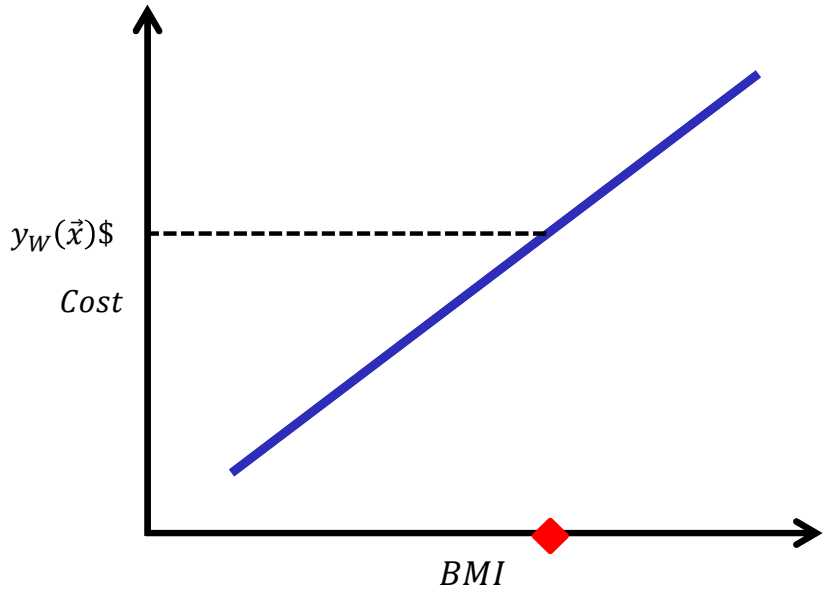
No need to keep the data once you have your regressor



More formally

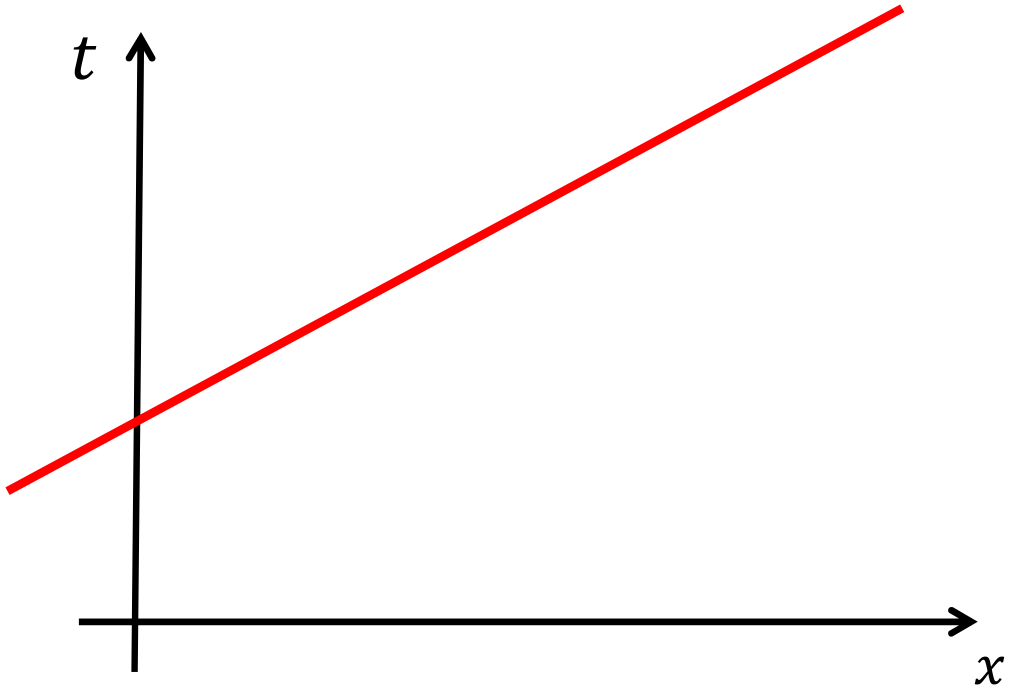
From Wikimedia Commons
the free media repository

$$y_W(\vec{x}) = cost$$



Rename, t for target

$$t = w_1 x + w_0$$



In our previous exemple

$t \rightarrow medical\ cost$
$x \rightarrow BMI$

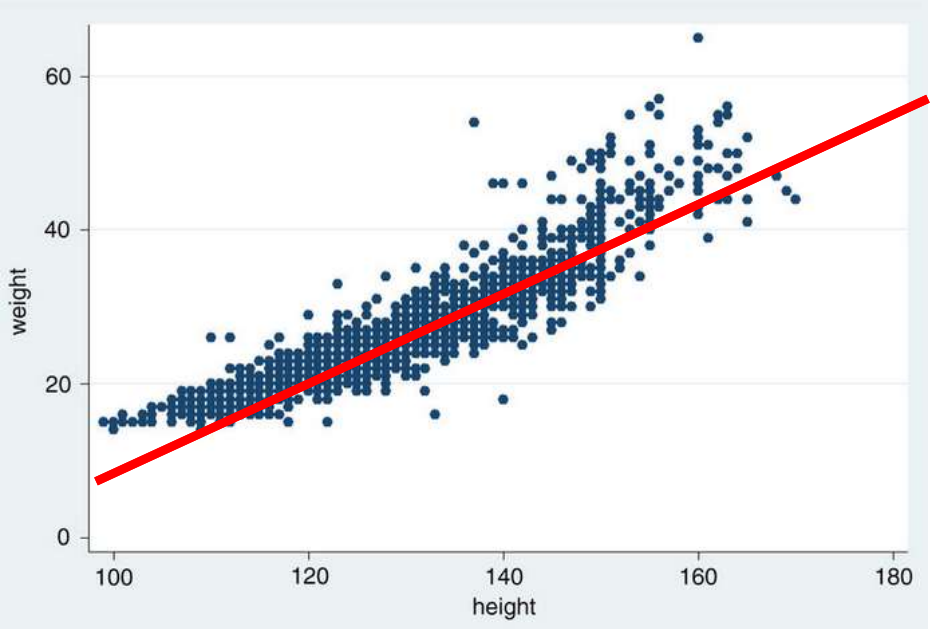
cost$= w_1 BMI + w_0$

Another example

Children

Weight Height

1,694 children surveyed in Tanzania.



$$y_{\vec{w}}(x) = w_1 x + w_0$$

$$y_{\vec{w}}(x_i) = t_i \quad \forall i$$

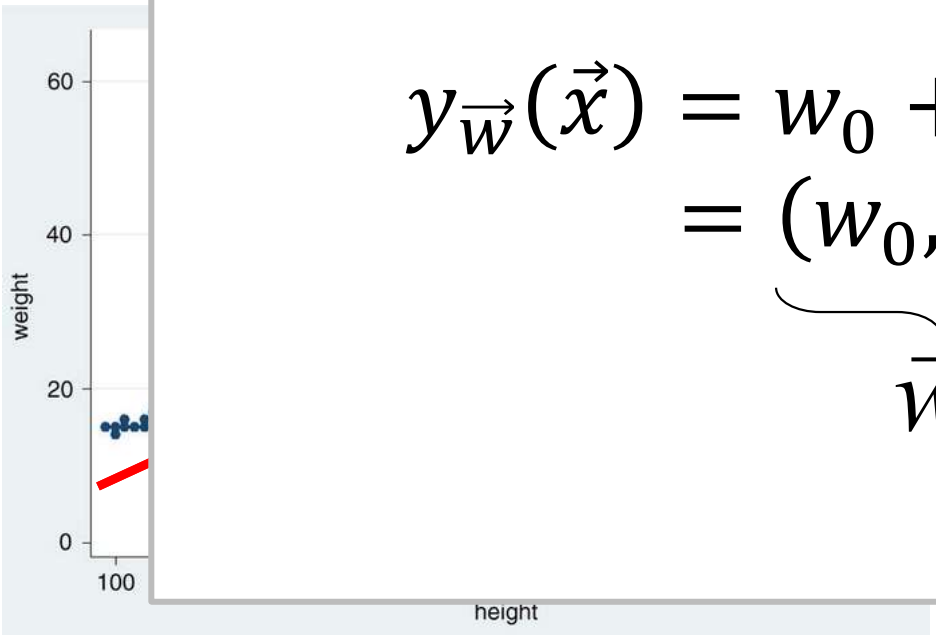
Nordin P, Poggensee G, Mtweve S, Krantz I**. From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

Another example

Children

Weight Height

1,6



$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$$
$$= \underbrace{(w_0, w_1)}_{\vec{w}} \cdot \underbrace{(1, x)}_{\vec{x}}$$

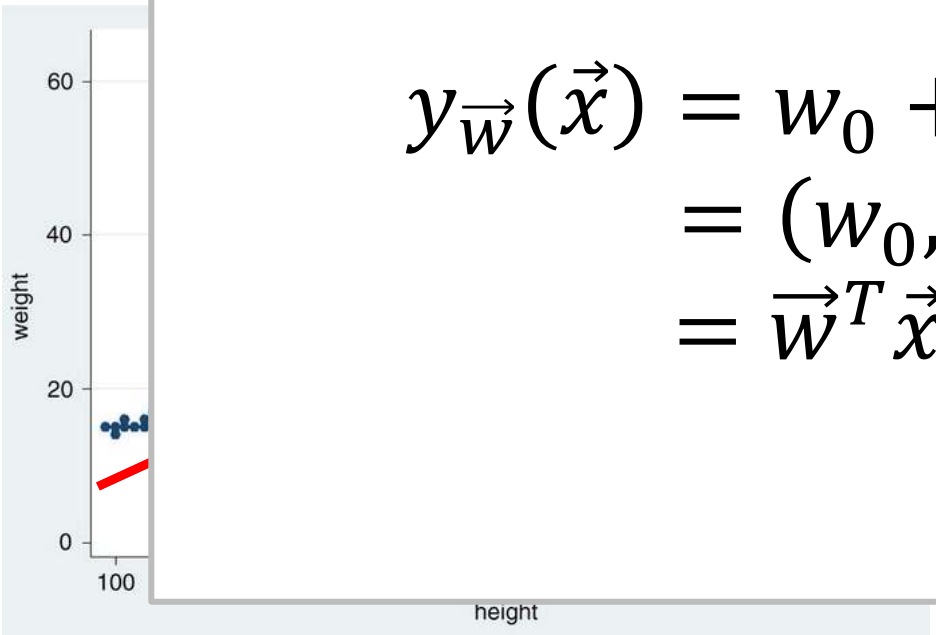
DOT product

Nordin P, Poggensee G, Mtweve S, Krantz I**. From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

Another example

Children
Weight Height

1,6



$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$$
$$= (w_0, w_1) \cdot (1, x)$$
$$= \vec{w}^T \vec{x}$$

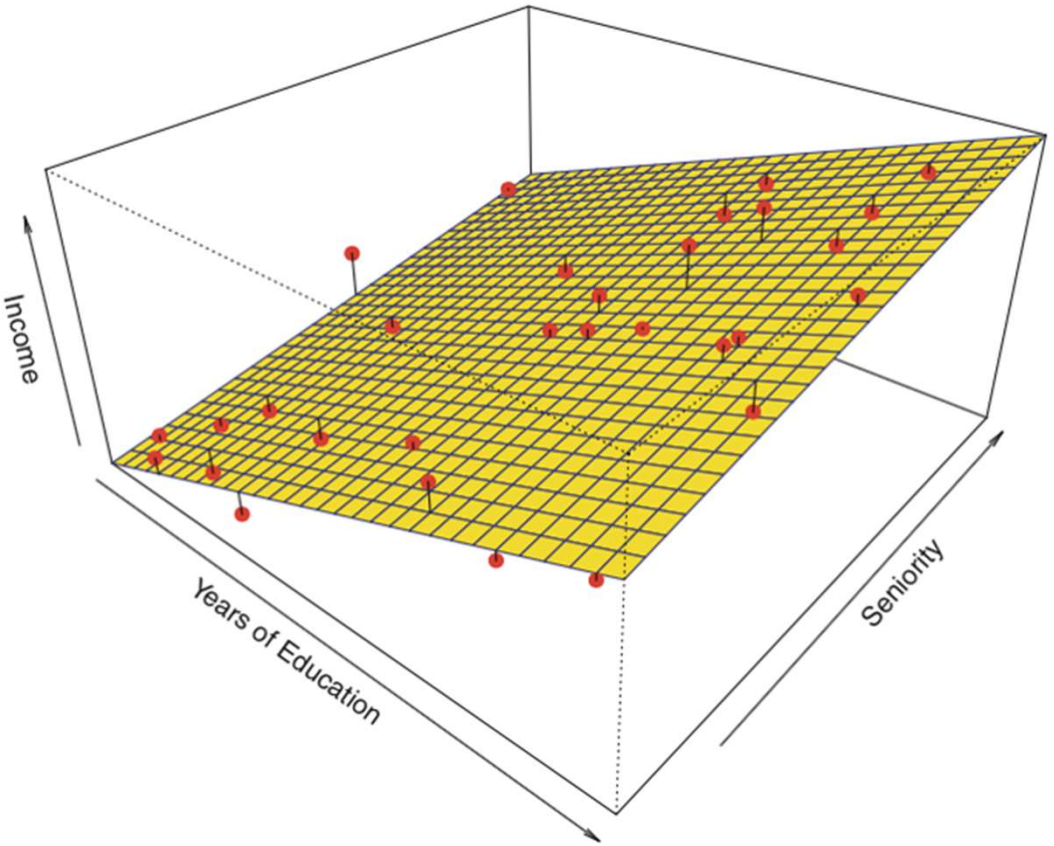
DOT product

Nordin P, Poggensee G, Mtweve S, Krantz I**. From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

Works with more than 1 variable

Example with 2 variables

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$



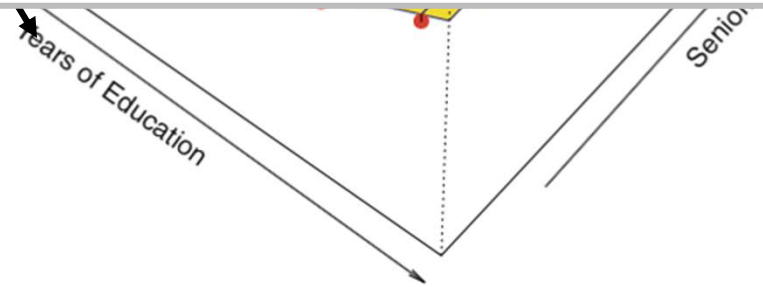
Credit : sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html

Works with more than 1 variable

Example with 2 variables

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$
$$= \underbrace{(w_0, w_1, w_2)}_{\vec{w}} \cdot \underbrace{(1, x_1, x_2)}_{\vec{x}}$$

DOT product



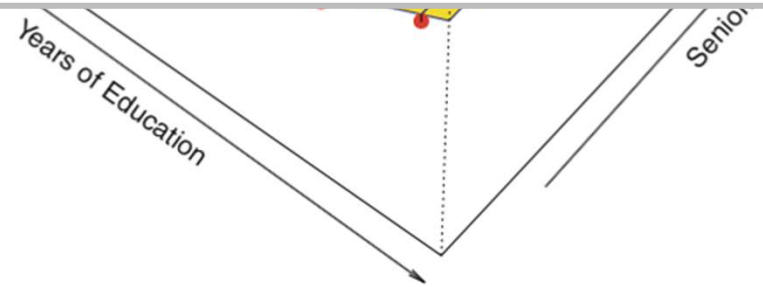
Credit : sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html

Works with more than 1 variable

Example with 2 variables

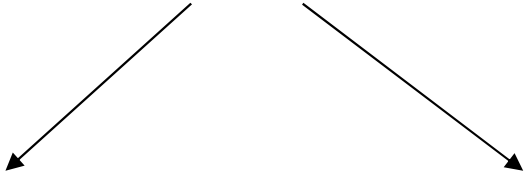
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$
$$= (w_0, w_1, w_2) \cdot (1, x_1, x_2)$$
$$= \vec{w}^T \vec{x}$$

DOT product



Credit : sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html

Linear models



Classification

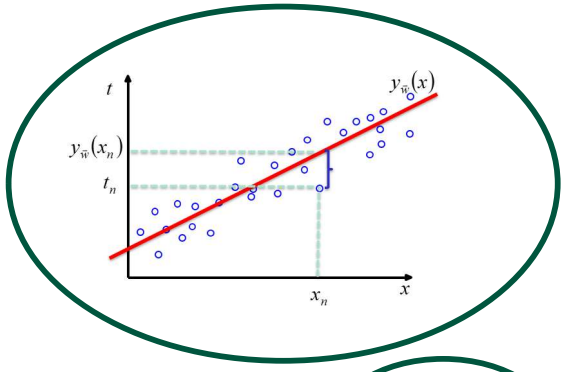
$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

Dot product

Regression

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x} = t$$

Dot product



Linear models

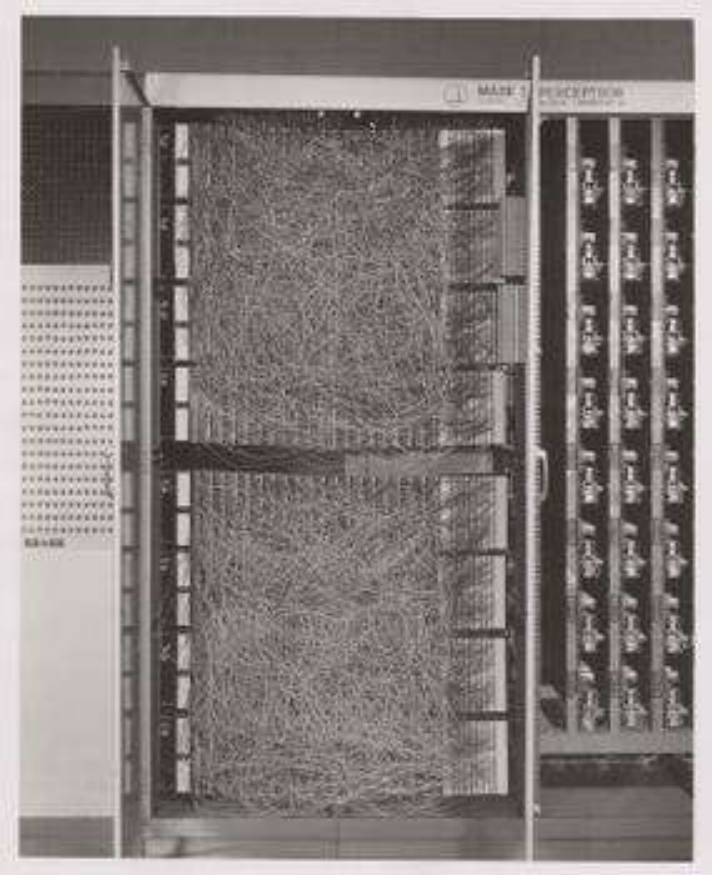
vs

deep learning

At the heart of deep neural nets are… **linear models**

Perceptron
Logistic regression
Multi-layer perceptron

Perceptron



Rosenblatt, Frank (1958), **The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain**, Psychological Review, v65, No. 6, pp. 386–408

Linear models

(neural networks)

Classification

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

Dot product

Regression

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x} = t$$

Dot product

Linear models

(neural networks)

Classification

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

Dot product

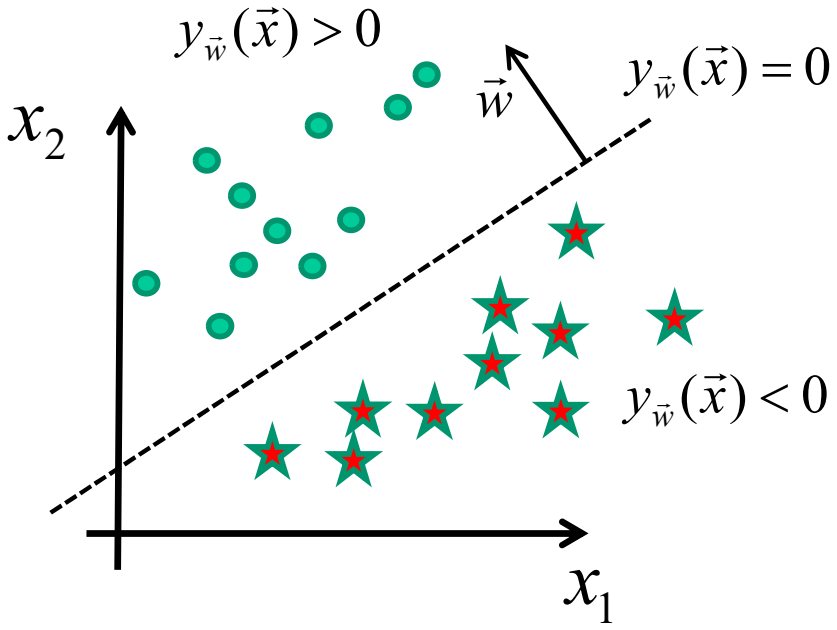
Regression

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = t$$

Dot product

Linear classifier

(2D and 2 classes)



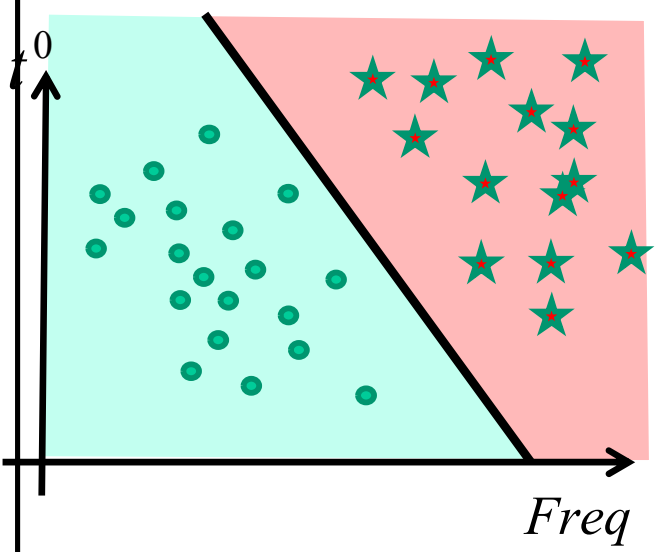
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_1$$

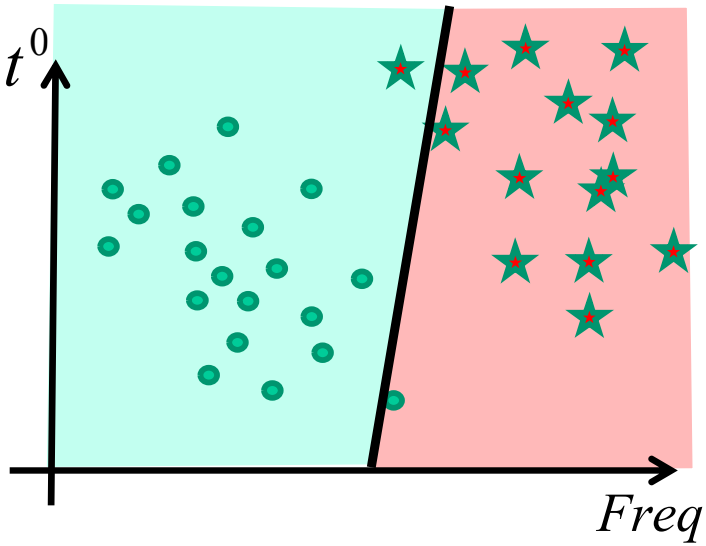
$$= \vec{w}^{\mathrm{T}} \vec{x}$$

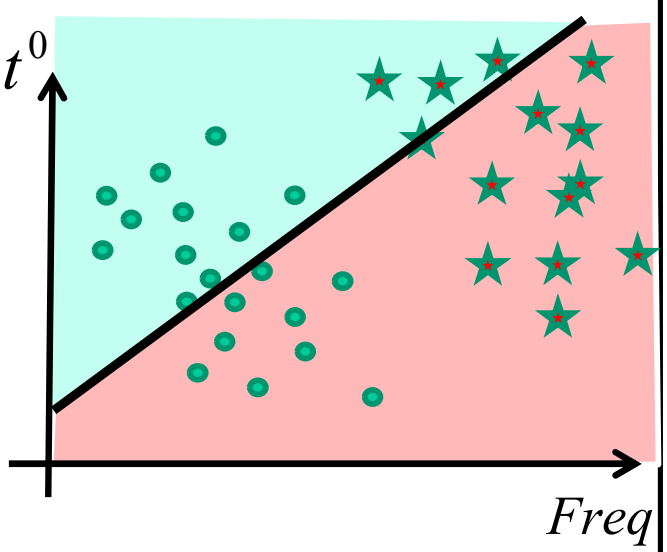
So far…

1. Training dataset: D
2. Classification function (a line in 2D) : $y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$

How good is a classifier?





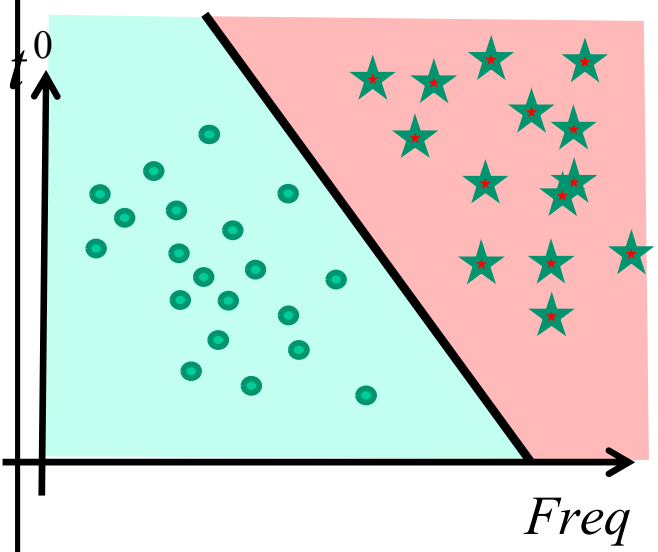


Good!

Medium

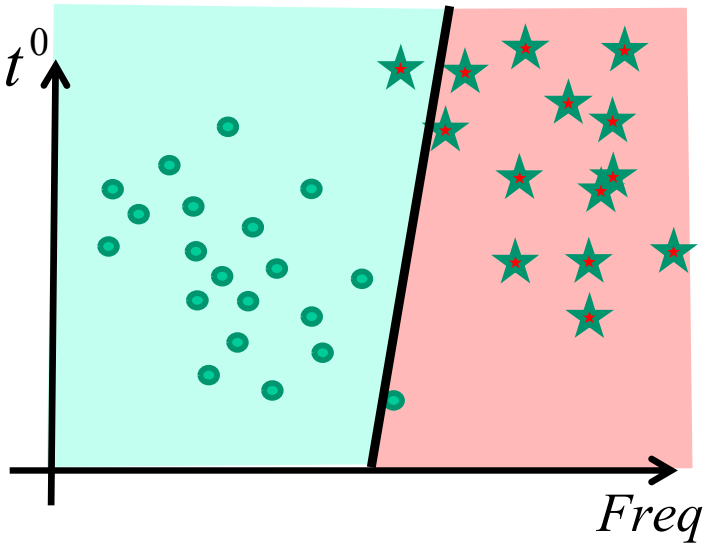
BAD!

Loss function



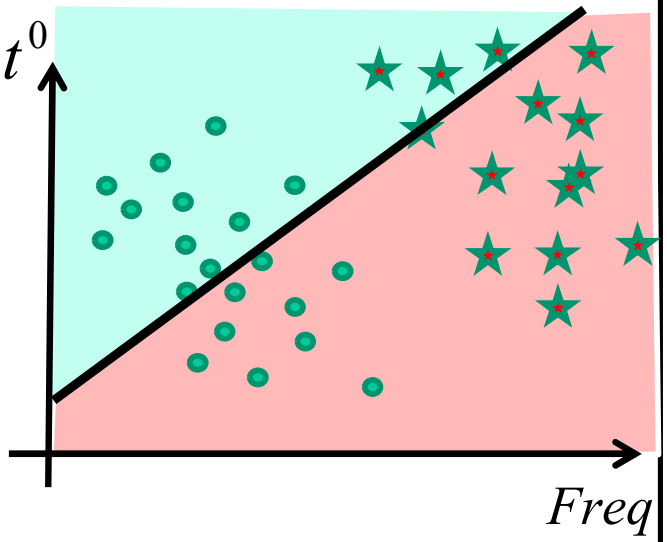
$L(y_{\vec{w}}(\vec{x}), D) \approx 0$

Good!



$L(y_{\vec{w}}(\vec{x}), D) > 0$

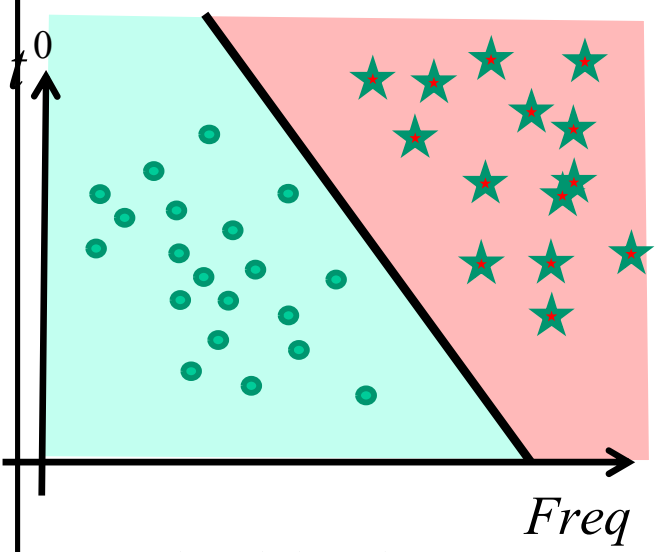
Medium



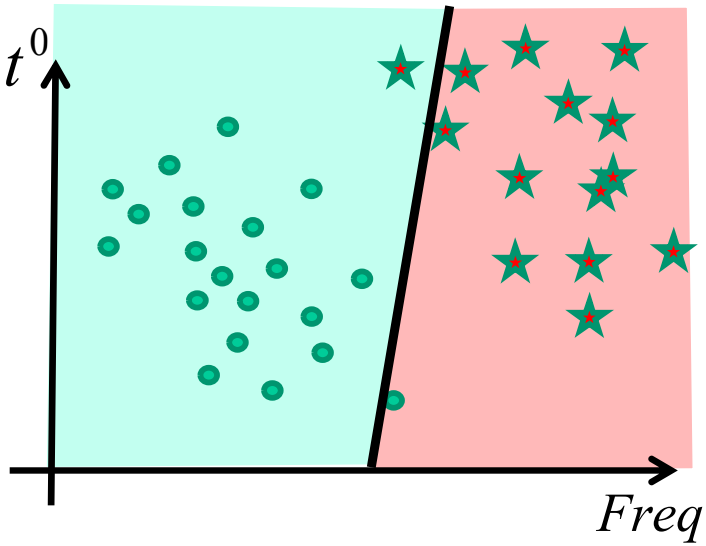
$L(y_{\vec{w}}(\vec{x}), D) >> 0$

BAD!

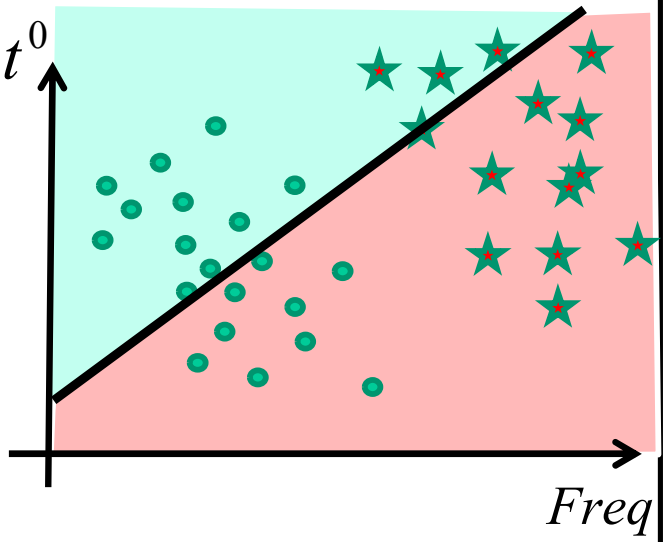
Training



$L(y_{\vec{w}}(\vec{x}), D) \approx 0$



$L(y_{\vec{w}}(\vec{x}), D) > 0$



$L(y_{\vec{w}}(\vec{x}), D) >> 0$

$$\vec{w} = argmin_{\vec{w}} L(y_{\vec{w}}(\vec{x}), D)$$

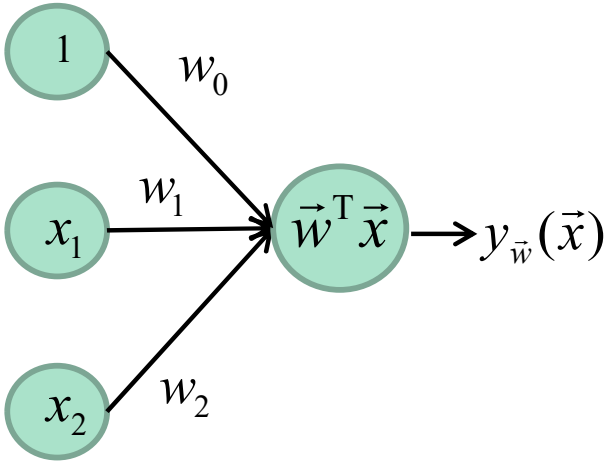
So far…

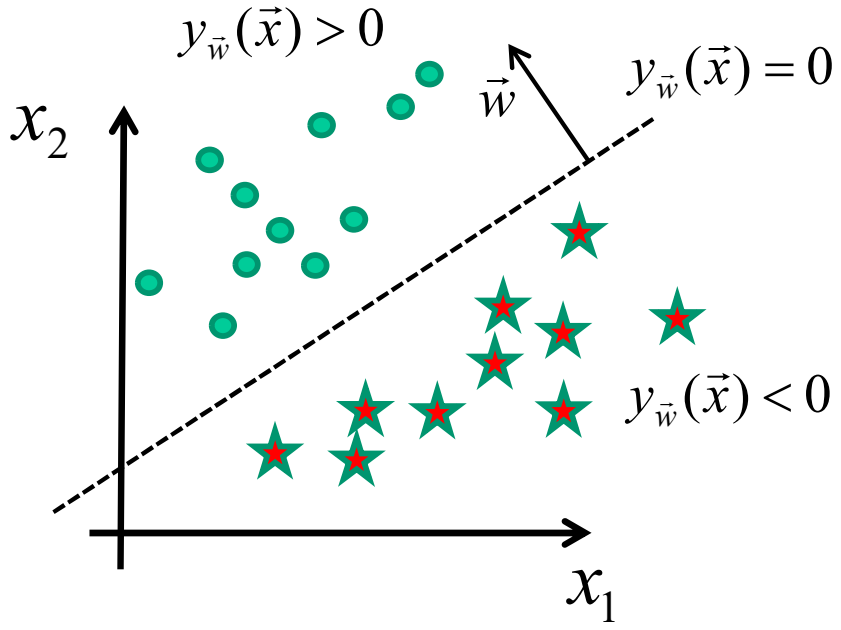
1. Training dataset: D
2. Classification function (a line in 2D) : $y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D)$

4. **Training** : find (w_0, w_1, w_2) that minimize $L(y_{\vec{w}}(\vec{x}), D)$

Perceptron

(2D and 2 classes)



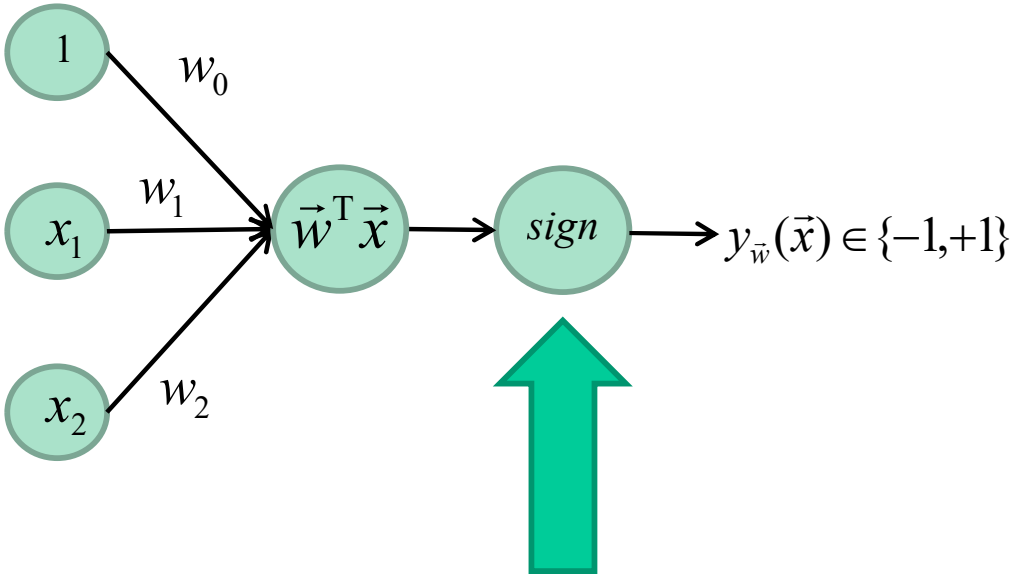


$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_1$$

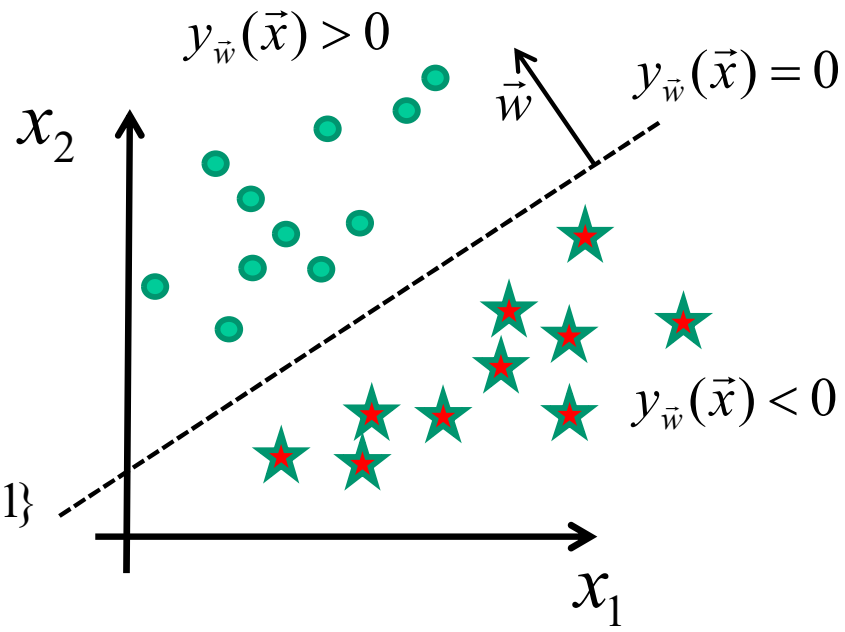
$$= \vec{w}^{\mathrm{T}}\vec{x}$$

Perceptron

(2D and 2 classes)



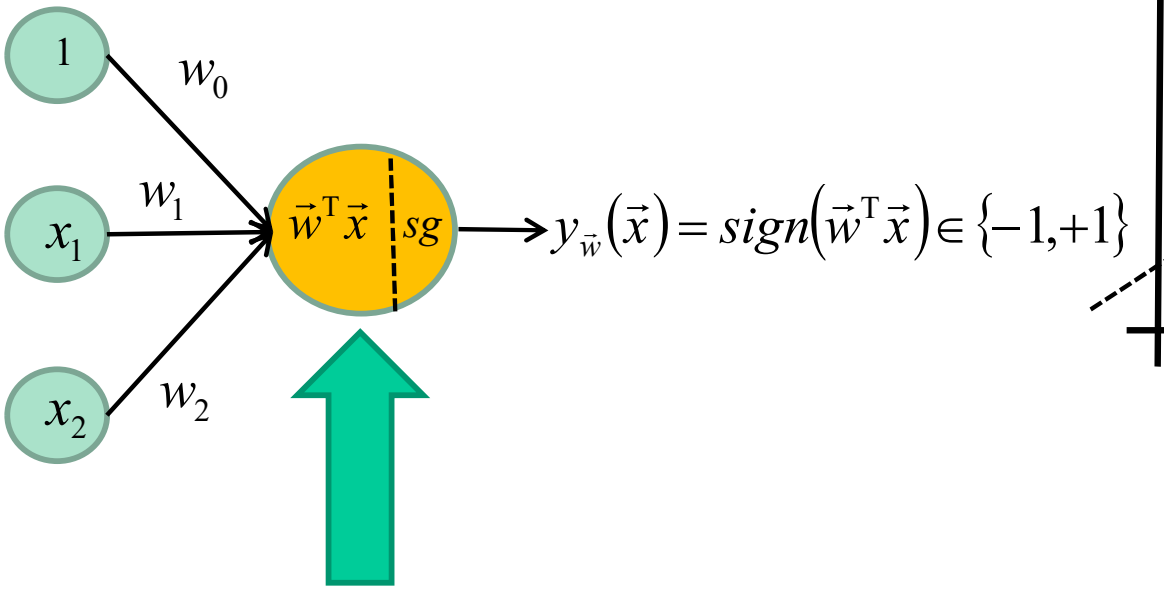
Activation function



$$y_{\vec{w}}(\vec{x}) = sign(\vec{w}^T\vec{x})$$

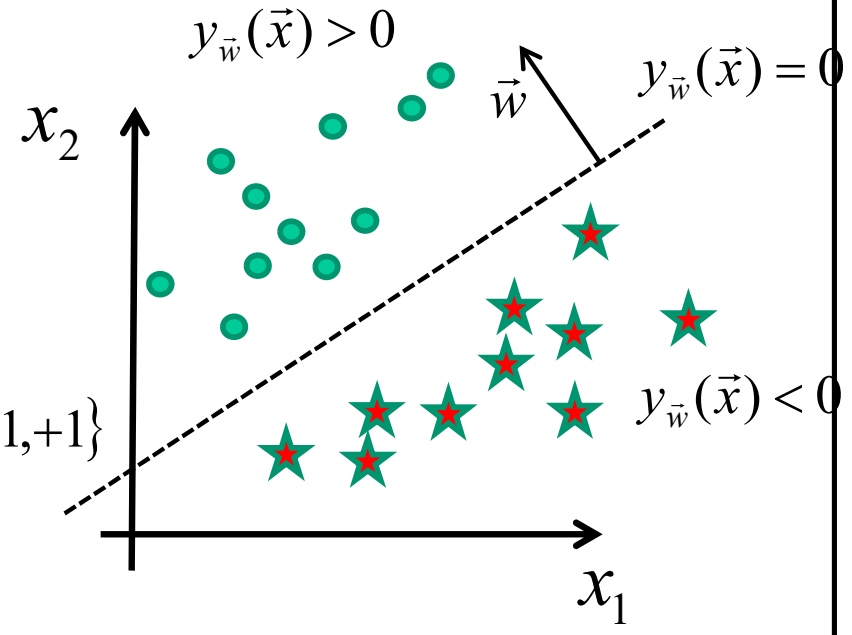
Perceptron

(2D and 2 classes)



Neuron

Dot product + activation function

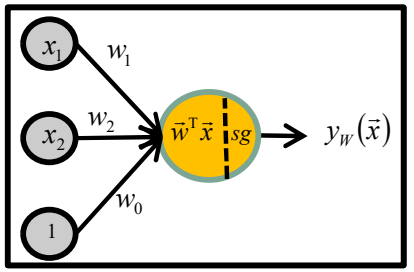


So far…

1. Training dataset: D
2. Classification function (a line in 2D) : $y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D)$

So far…

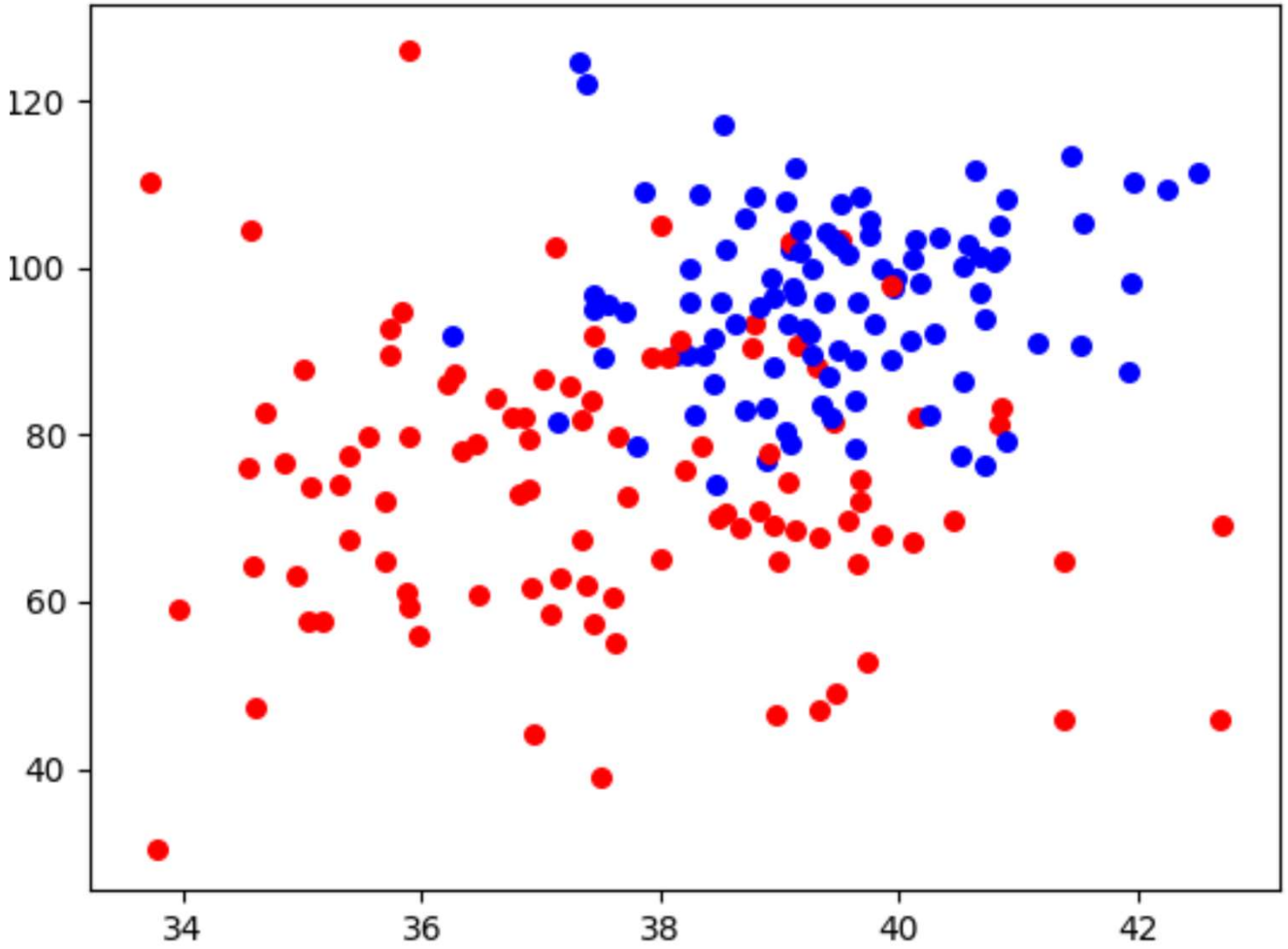
1. Training dataset: D
2. Classification function (a line in 2D) :
3. Loss function: $L\left(y_{\vec{w}}(\vec{x}), D\right)$



4. Training : find (w_0, w_1, w_2) that minimize $L\left(y_{\vec{w}}(\vec{x}), D\right)$

Linear classifiers have **limits**

Non-linearly separable training data



Linear classifier = **large error rate**

Non-linearly separable training data

Three classical solutions

1. Acquire more observations
2. Use a non-linear classifier
3. Transform the data

Non-linearly separable training data

Three classical solutions

1. **More observations**
2. Use a non-linear classifier
3. Transform the data

Acquire more data

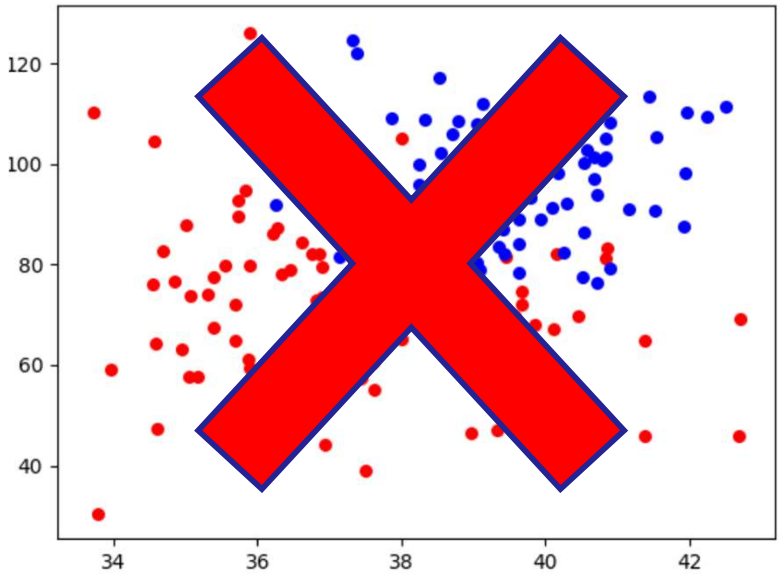
D

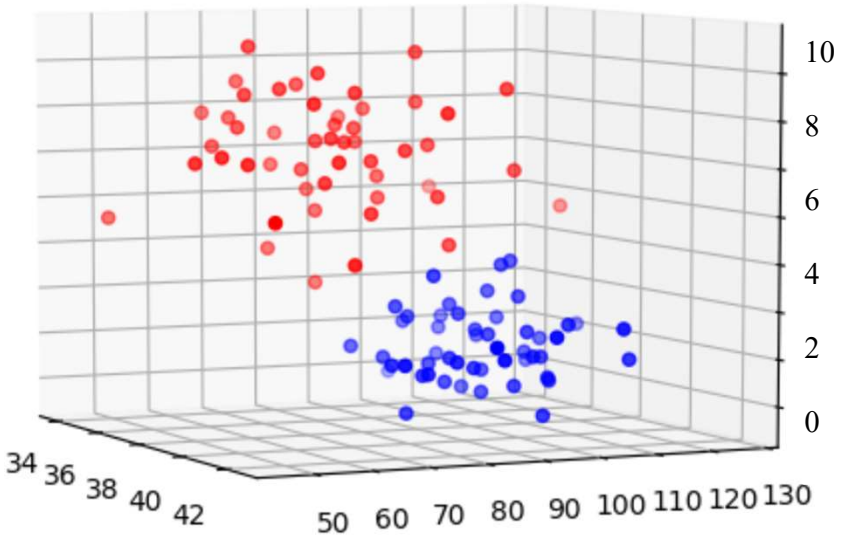
	(temp, freq)	diagnostic
Patient 1	(37.5, 72)	healthy
Patient 2	(39.1, 103)	sick
Patient 3	(38.3, 100)	sick
	(…)	…
Patient N	(36.7, 88)	healthy
	$\vec{x}$	t

D

	(temp, freq, headache)	Diagnostic
Patient 1	(37.5, 72, 2)	healthy
Patient 2	(39.1, 103, 8)	sick
Patient 3	(38.3, 100, 6)	sick
	(…)	…
Patient N	(36.7, 88, 0)	healthy
	$\vec{x}$	t

Non-linearly separable training data





$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

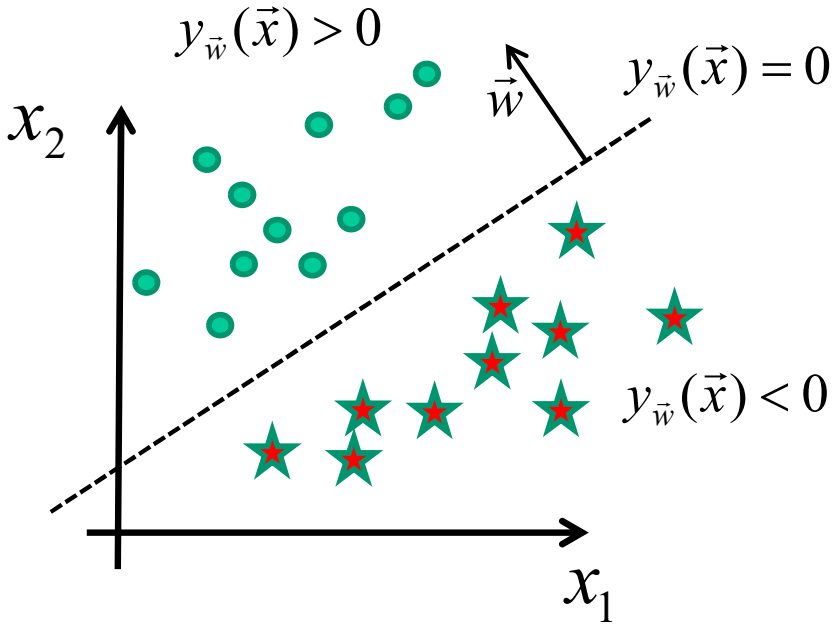
(line)

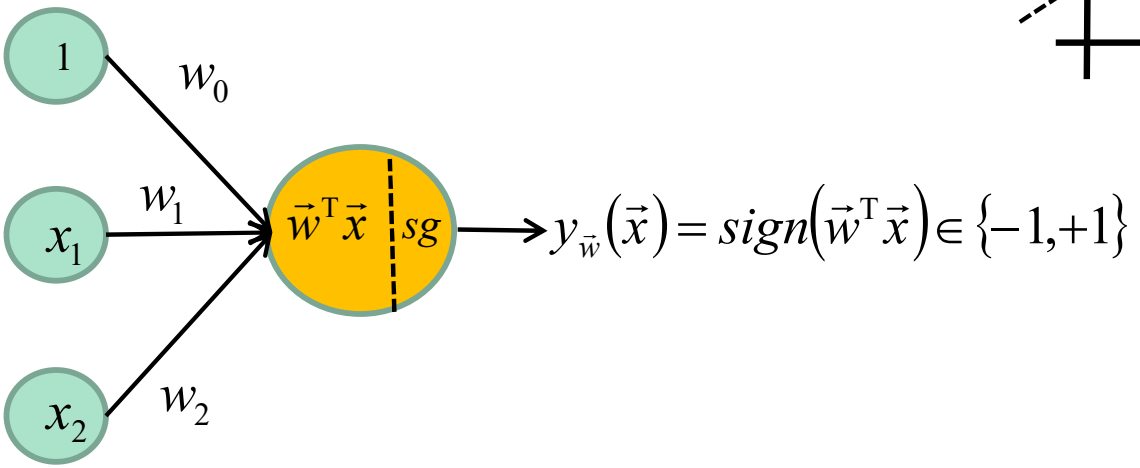
$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0$$

(plane)

Perceptron

(2D and 2 classes)



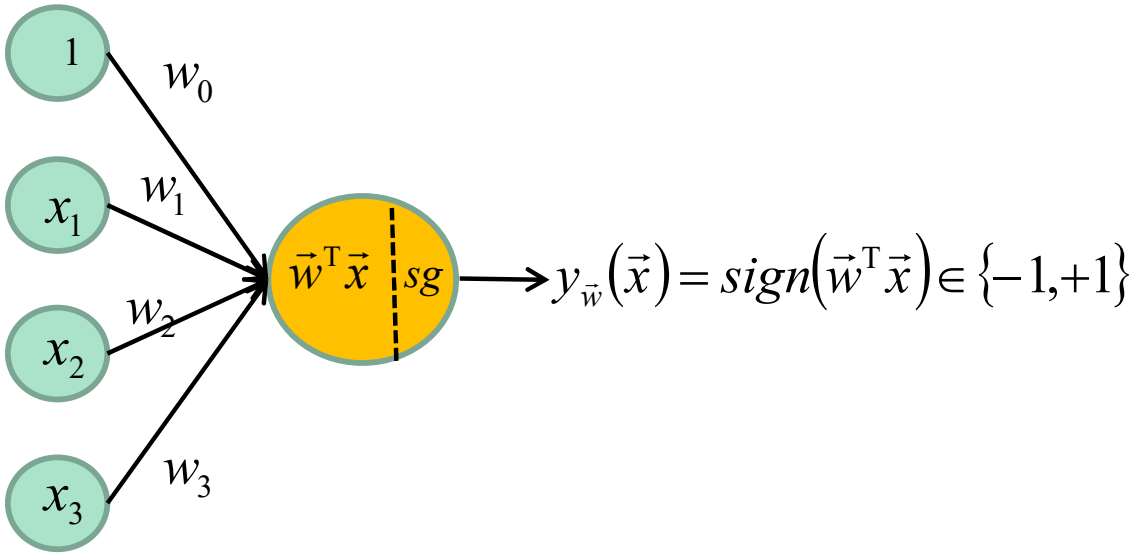


$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

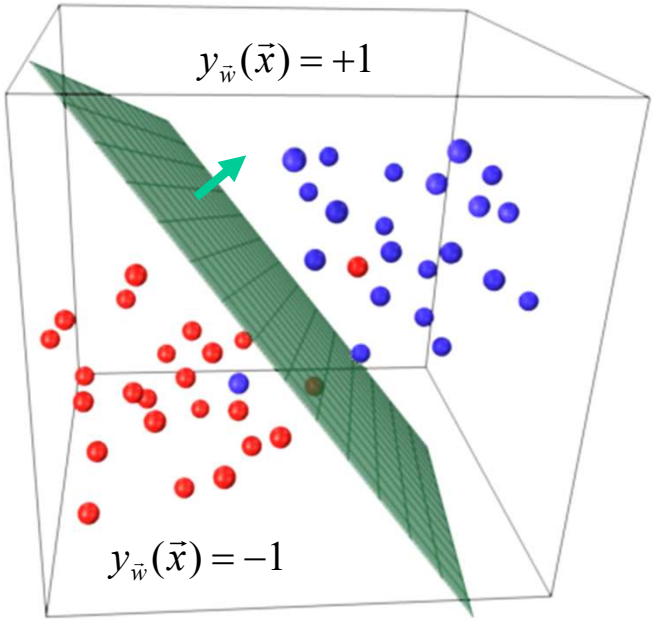
(line)

Perceptron

(3D and 2 classes)



Example 3D

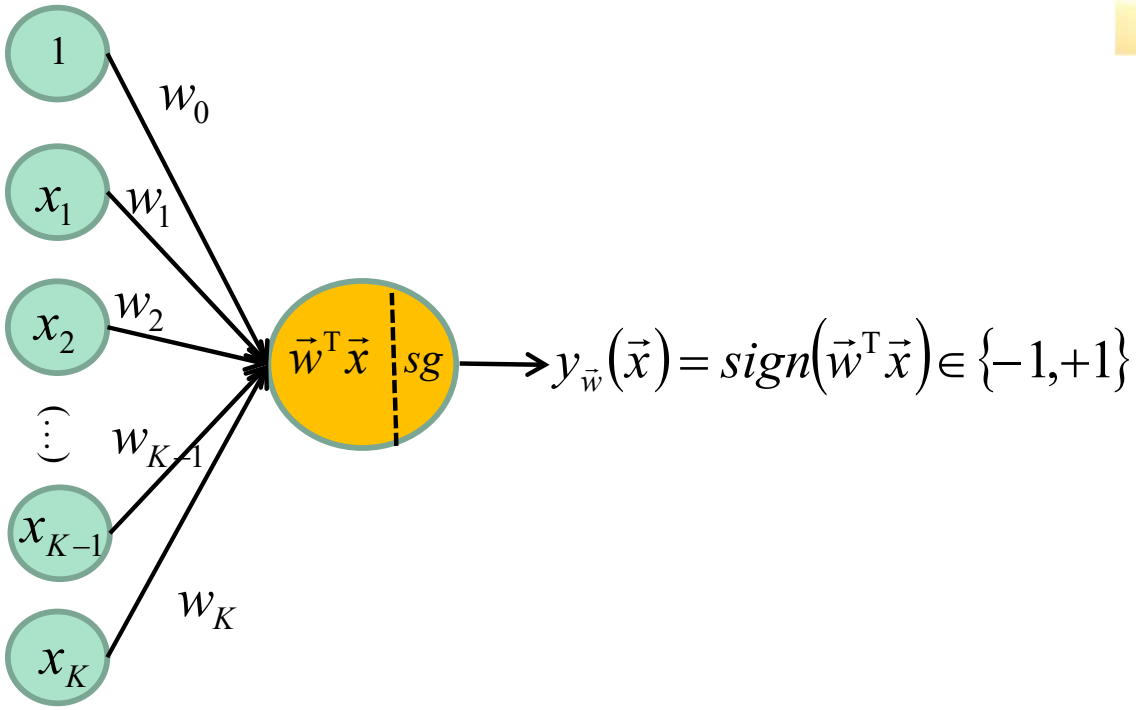


$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0$$

(plane)

Perceptron

(K-D and 2 classes)

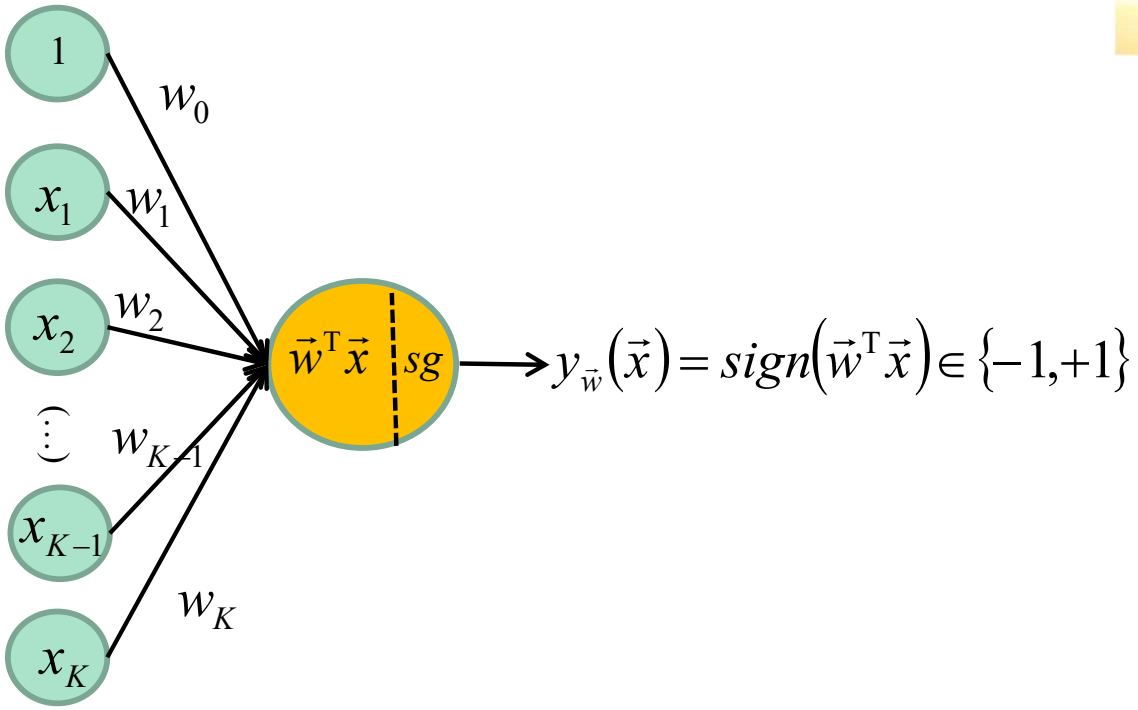


$$y_{\vec{w}}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_Kx_K + w_0$$

(hyperplane)

Perceptron

(K-D and 2 classes)



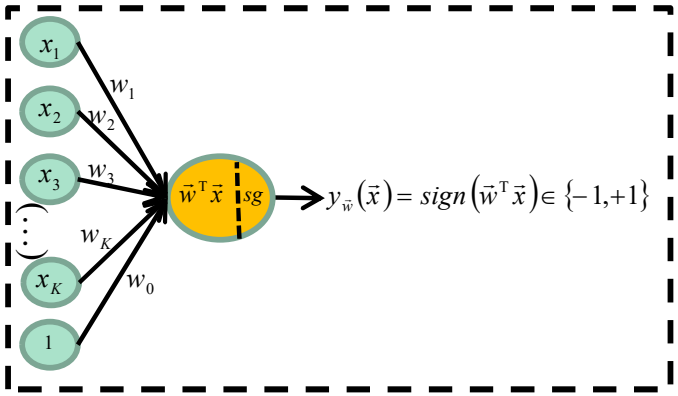
Dot product!!

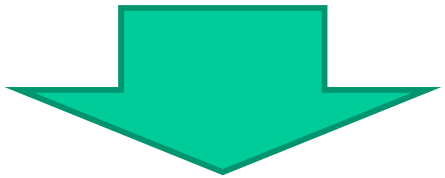
$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x}$$

(hyperplane)

So far…

1. Training dataset: D
2. Linear classification function:
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D)$

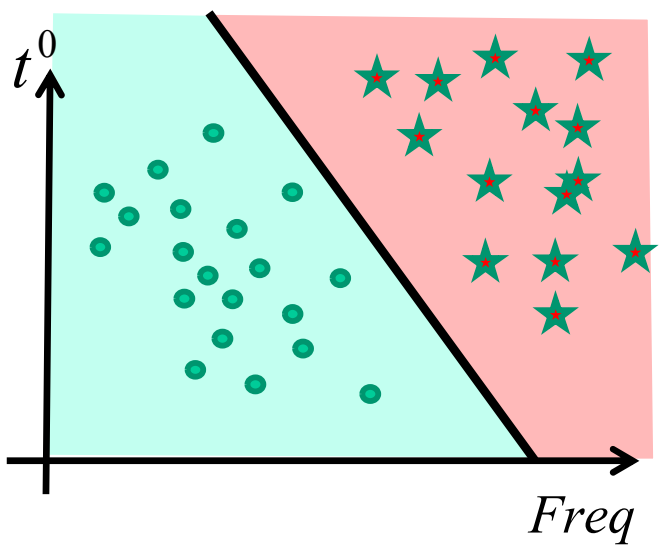




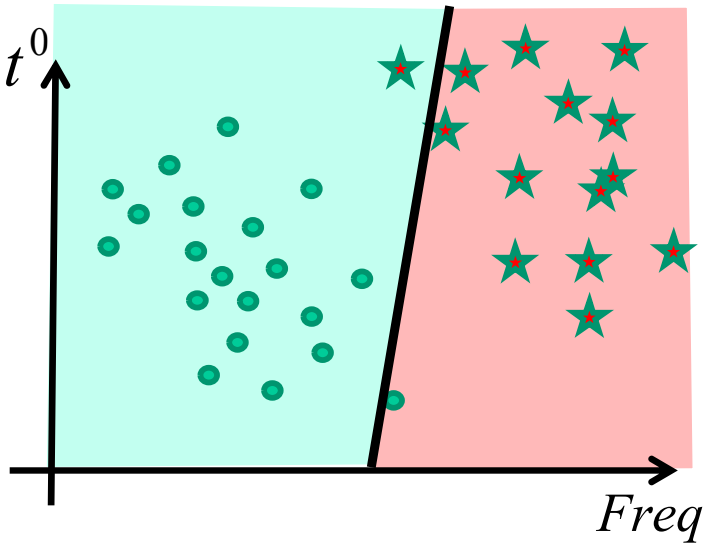
4. Training : find $\vec{w}$ that minimizes $L(y_{\vec{w}}(\vec{x}), D)$

How do we train a Perceptron?

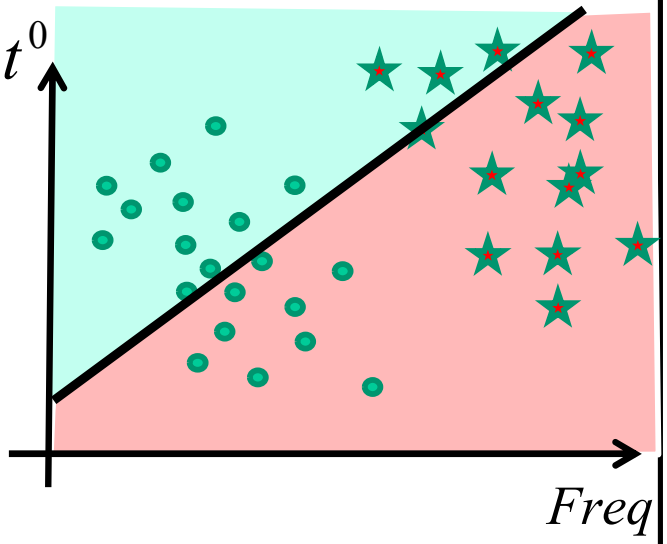
Remember the loss function?



$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) \approx 0$

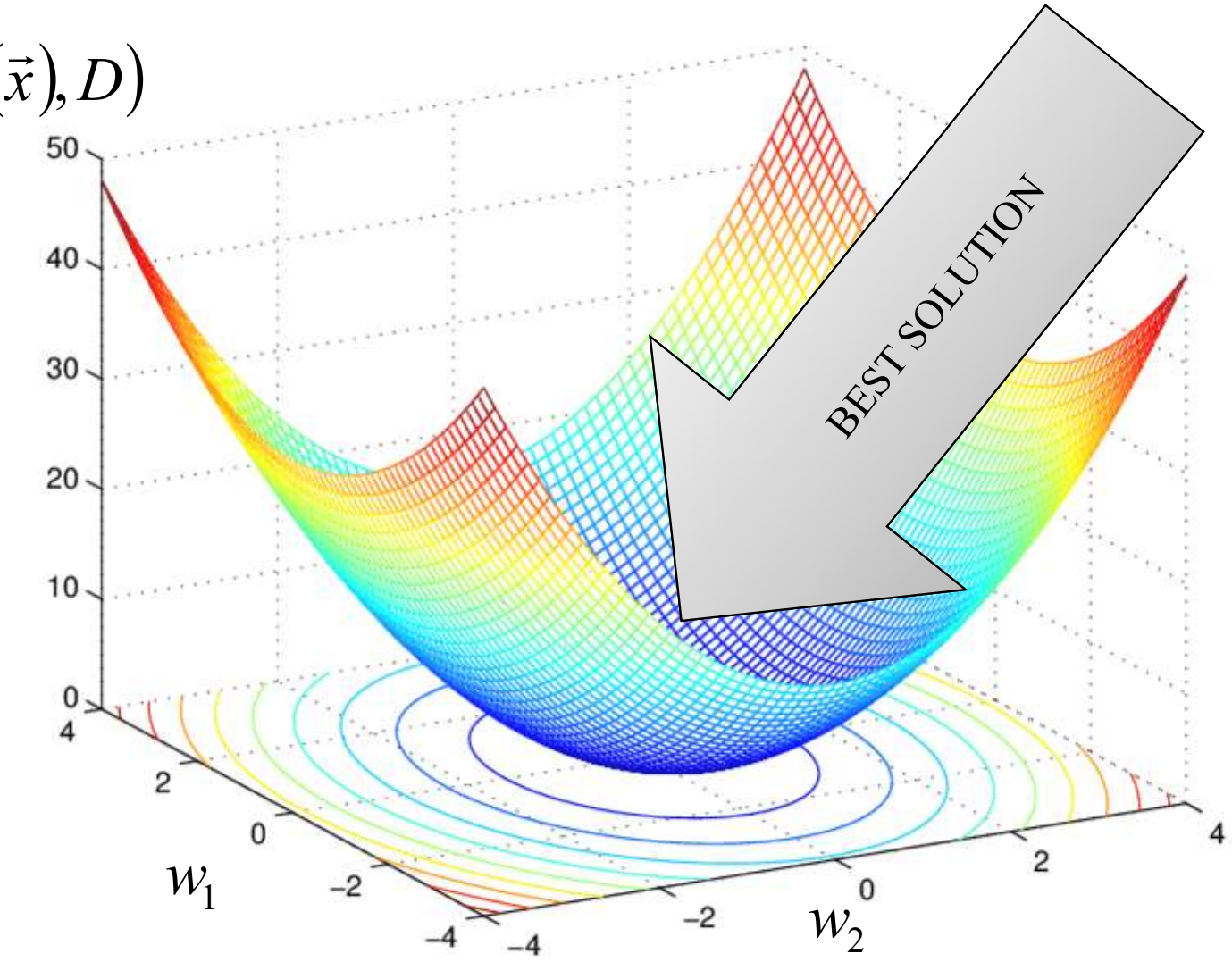


$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) > 0$



$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) >> 0$

$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right)$

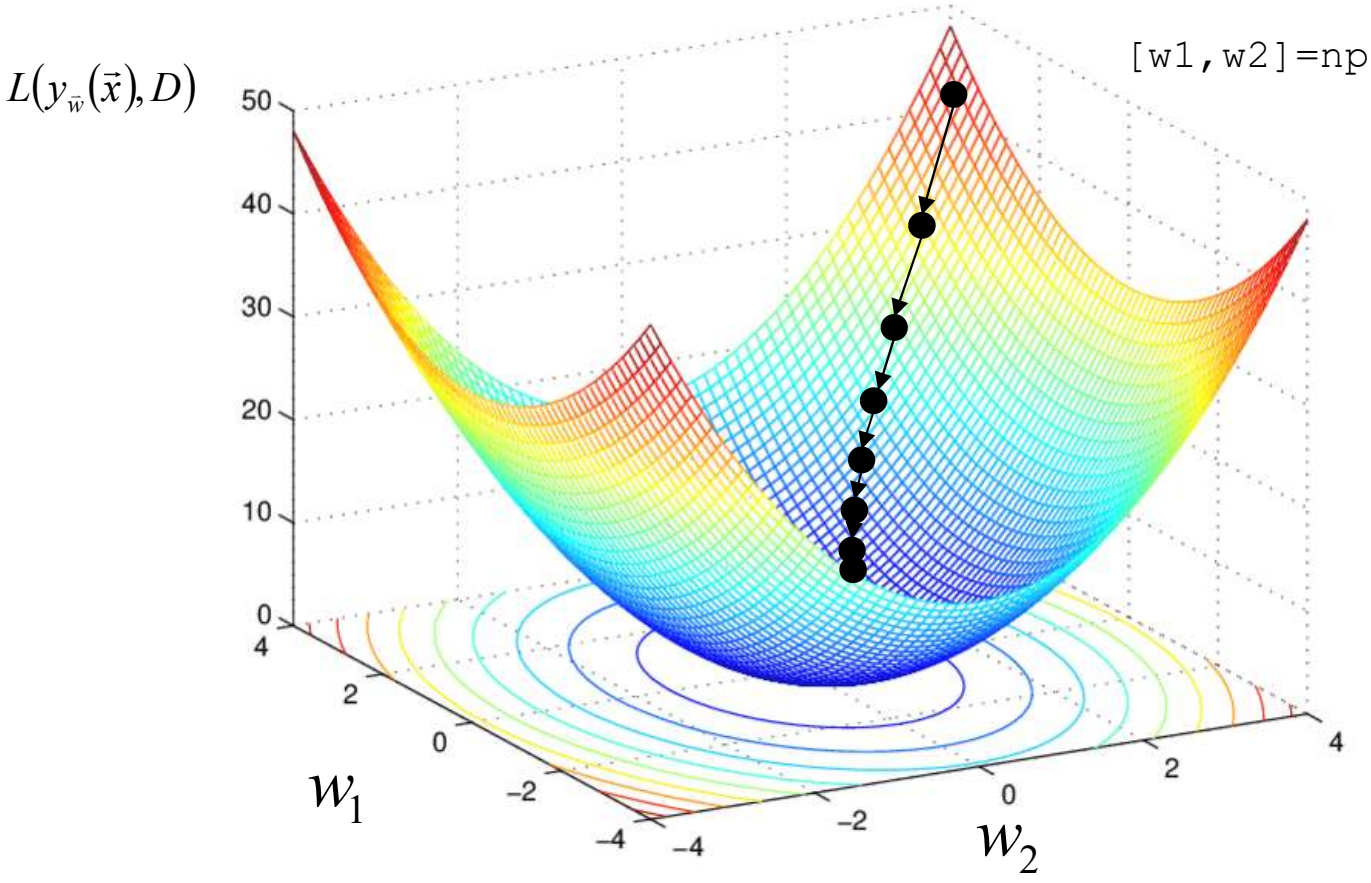


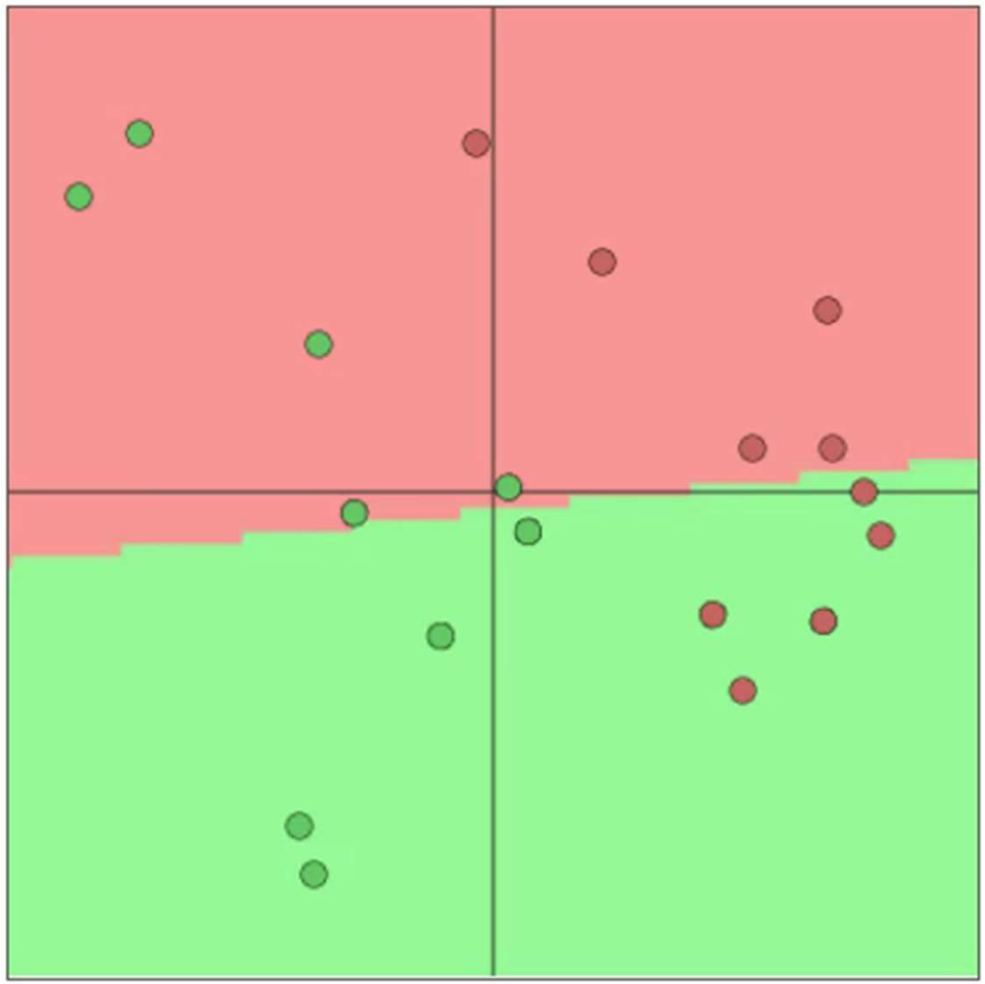
Perceptron

Question: how to find the best solution? $\nabla L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = 0$

Random initialization

```
[w1,w2]=np.random.randn(2)
```





Gradient descent

Question: how to find the best solution? $\nabla L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = 0$

$$\vec{w}^{[k+1]} = \vec{w}^{[k]} - \eta \nabla L\left(y_{\vec{w}^{[k]}}\left(\vec{x}\right), D\right)$$

Gradient of the loss function

Learning rate

Optimization

$$\vec{w}^{[k+1]} = \vec{w}^{[k]} - \eta^{[k]} \nabla L$$

∇L: Gradient of the loss function

$\eta^{[k]}$: learning rate

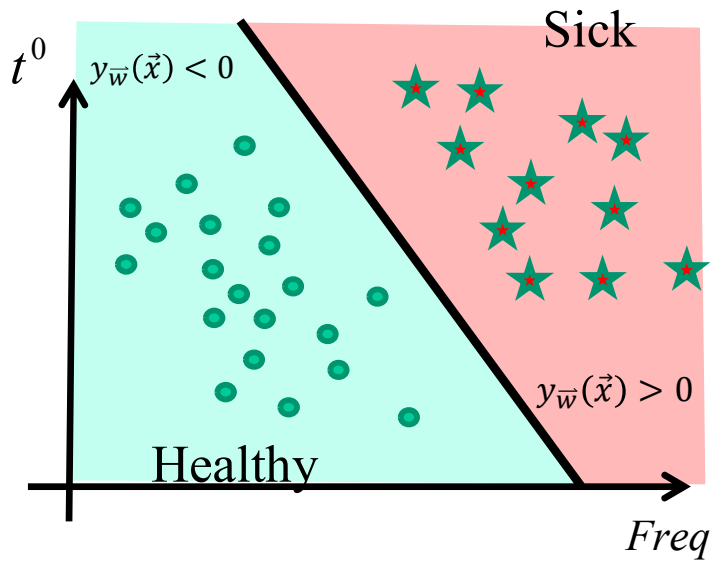
Stochastic gradient descent (SGD)

Init $\vec{w}$
k=0
DO k=k+1
    FOR n = 1 to N
        $\vec{w} = \vec{w} - \eta^{[k]} \nabla L(\vec{x}_n)$

UNTIL every data is well classified or k== MAX_ITER

Perceptron Criterion (loss)

Observation



A **wrongly classified sample** is when

$$\vec{w}^{\mathrm{T}}\vec{x}_n > 0 \text{ and } t_n = -1$$

or

$$\vec{w}^{\mathrm{T}}\vec{x}_n < 0 \text{ and } t_n = +1.$$

Consequently $-\vec{w}^{\mathrm{T}}\vec{x}_n t_n$ is **ALWAYS positive for wrongly classified samples**

Perceptron gradient descent

$$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} -\vec{w}^{\mathrm{T}} \vec{x}_n t_n$$

$$\nabla L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} -\vec{x}_n t_n$$

Stochastic gradient descent (SGD)

Init $\vec{w}$
k=0
DO k=k+1
  FOR n = 1 to N
    IF $\vec{w}^{\mathrm{T}} \vec{x}_n t_n < 0$ THEN /* wrongly classified */
      $\vec{w} = \vec{w} + \eta t_n \vec{x}_n$

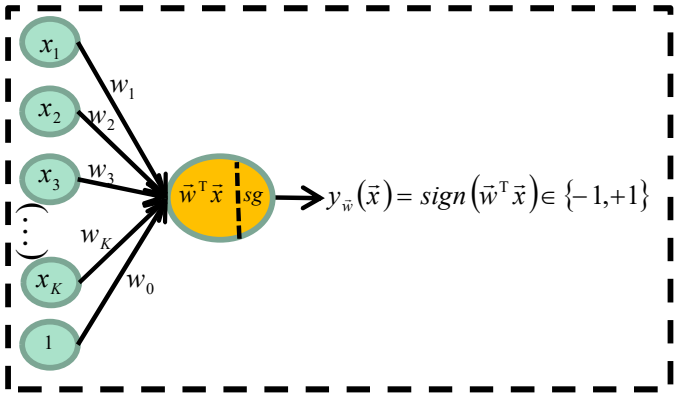
UNTIL every data is well classified OR k=k_MAX

NOTE : **learning rate** η :

- **Too low** => slow convergence
- **Too large** => might not converge (even diverge)
- Can **decrease** at each iteration (e.g. $\eta^{[k]} = cst / k$)

So far…

1. Training dataset: D
2. Linear classification function:
3. Loss function: $L\left(y_{\vec{w}}(\vec{x}), D\right) = \sum_{\vec{x}_n \in V} -\vec{w}^{\mathrm{T}} \vec{x}_n t_n$

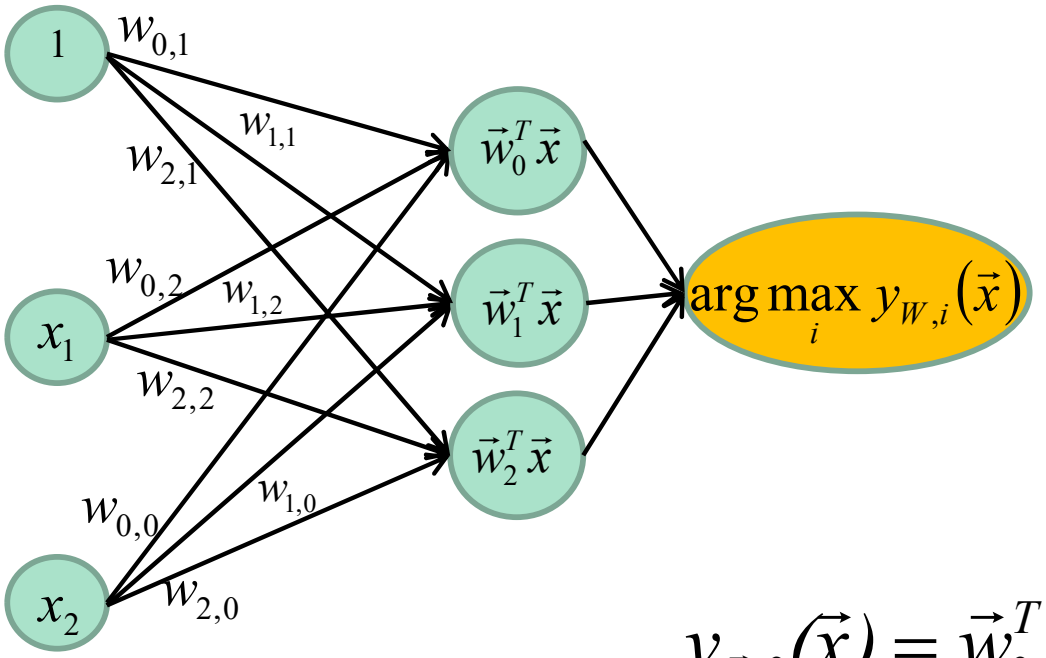


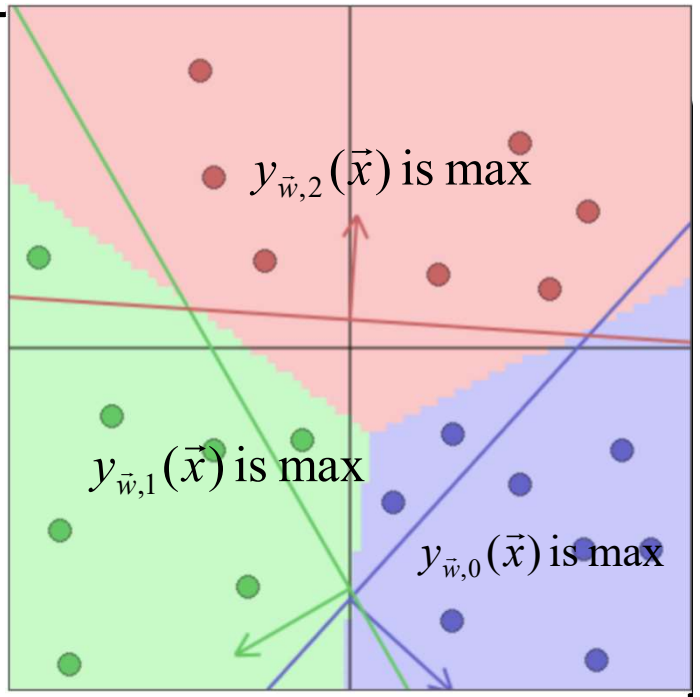
4. Training : find $\vec{w}$ that minimizes $L\left(y_{\vec{w}}(\vec{x}), D\right)$

$$\nabla L\left(y_{\vec{w}}(\vec{x}), D\right) = 0$$

Multiclass Perceptron

(2D and 3 classes)





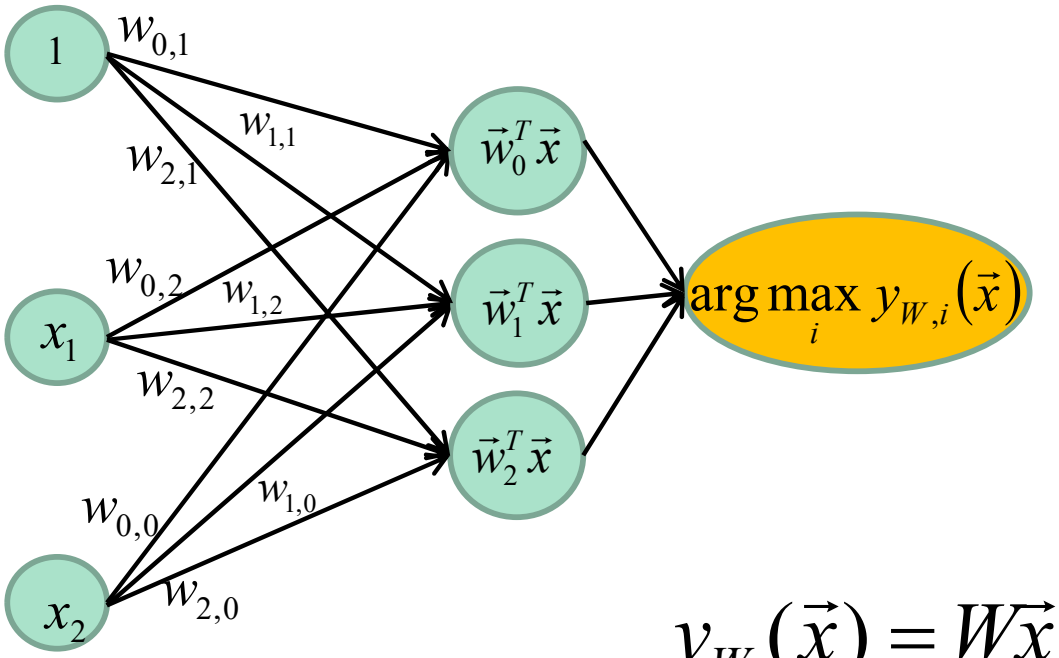
$$y_{\vec{w},0}(\vec{x}) = \vec{w}_0^T\vec{x} = w_{0,0} + w_{0,1}x_1 + w_{0,2}x_2$$

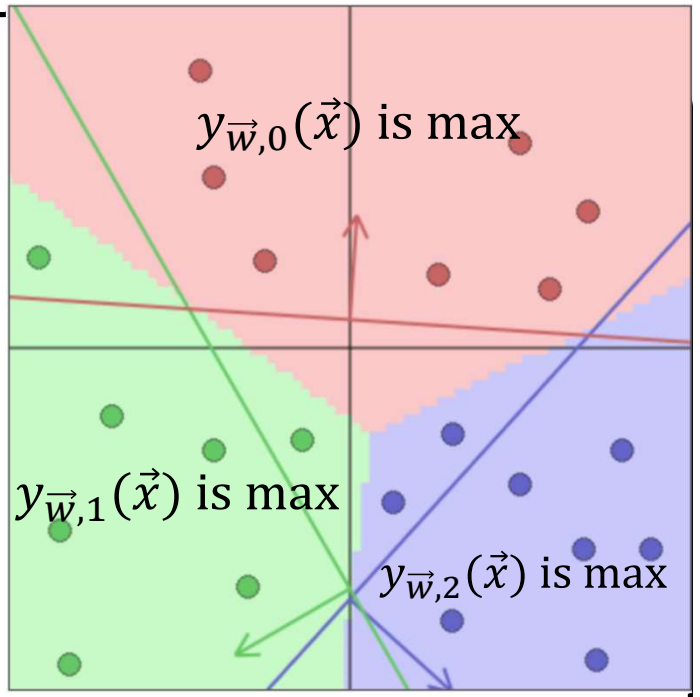
$$y_{\vec{w},1}(\vec{x}) = \vec{w}_1^T\vec{x} = w_{1,0} + w_{1,1}x_1 + w_{1,2}x_2$$

$$y_{\vec{w},2}(\vec{x}) = \vec{w}_2^T\vec{x} = w_{2,0} + w_{2,1}x_1 + w_{2,2}x_2$$

Multiclass Perceptron

(2D and 3 classes)





$$y_W(\vec{x}) = W\vec{x}$$

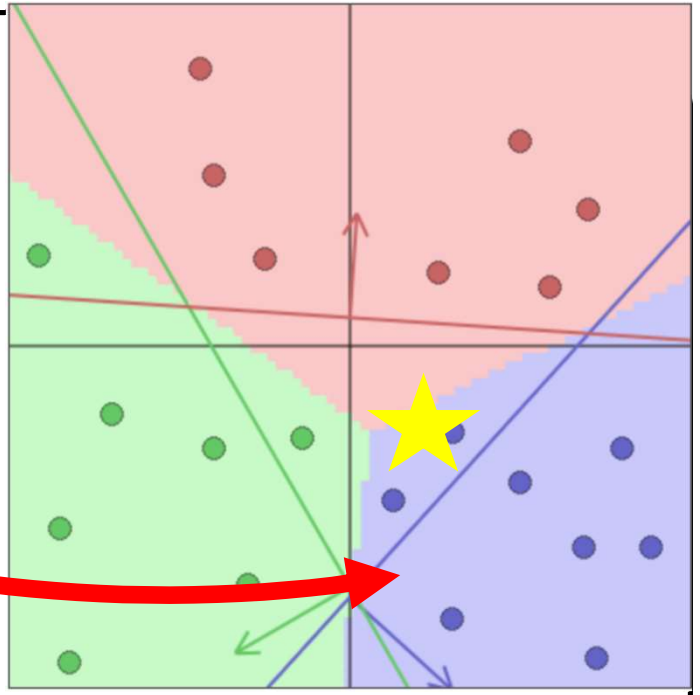
$$y_W(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Multiclass Perceptron

(2D and 3 classes)

Example

(1.1, -2.0)



$$y_W(\vec{x}) = \begin{bmatrix} -2 & -3.6 & 0.5 \\ -4 & 2.4 & 4.1 \\ -6 & 4 & -4.9 \end{bmatrix} \begin{bmatrix} 1 \\ 1.1 \\ -2 \end{bmatrix} = \begin{bmatrix} -6.9 \\ -9.6 \\ 8.2 \end{bmatrix}$$

Class 0

Class 1

Class 2

Multiclass Perceptron

Loss function

$$L\left(y_W\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} \left(\vec{w}_j^T \vec{x}_n - \vec{w}_{t_n}^T \vec{x}_n\right)$$

Sum over all wrongly classified samples

Score of the wrong class

Score of the true class

$$\nabla L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} \vec{x}_n$$

Multiclass Perceptron

Stochastic gradient descent (SGD)

init **W**
k=0, i=0
DO k=k+1
    FOR n = 1 to N
        $j = \arg\max \mathrm{W}^T \vec{x}_n$
        IF $j \neq t_i$ THEN /* wrongly classified sample */

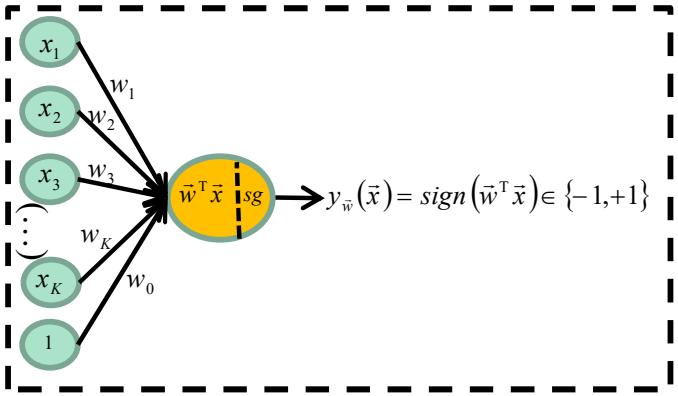
$$\vec{w}_j = \vec{w}_j - \eta\vec{x}_n$$

$$\vec{w}_{t_n} = \vec{w}_{t_n} + \eta\vec{x}_n$$

UNTIL every data is well classified or k > K_MAX.

So far…

1. Training dataset: D
2. Linear classification function:
3. Loss function: $L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} -\vec{w}^{\mathrm{T}} \vec{x}_n t_n$

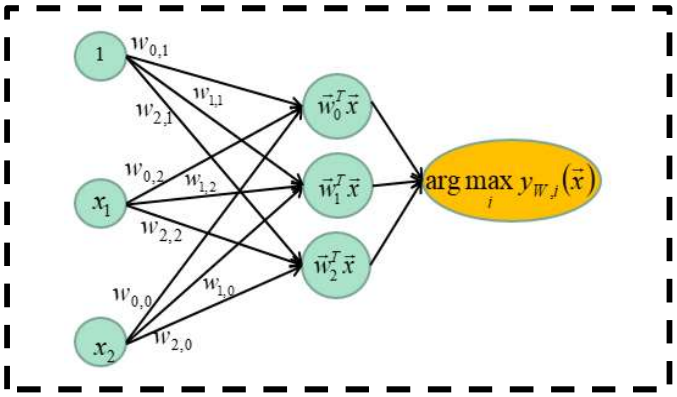


4. Training : find $\vec{w}$ that minimizes $L\left(y_{\vec{w}}\left(\vec{x}\right), D\right)$

$$\nabla L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = 0$$

So far…

1. Training dataset: D
2. Linear classification function:
3. Loss function: $L(y_W(\vec{x}), D) = \sum_{\vec{x}_n \in V} \left(\vec{w}_j^T \vec{x}_n - \vec{w}_{t_n}^T \vec{x}_n \right)$



4. Training : find $\vec{w}$ that minimizes $L(y_{\vec{w}}(\vec{x}), D)$

$$\nabla L(y_{\vec{w}}(\vec{x}), D) = 0$$

Perceptron

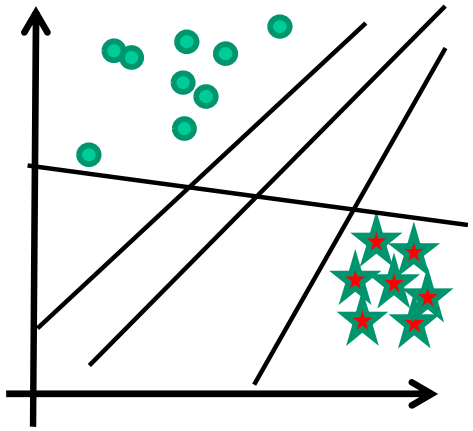
Advantages:

- Very simple
- Does **NOT** assume the data follows a **Gaussian distribution**.
- If data is **linearly separable**, convergence is **guaranteed**.

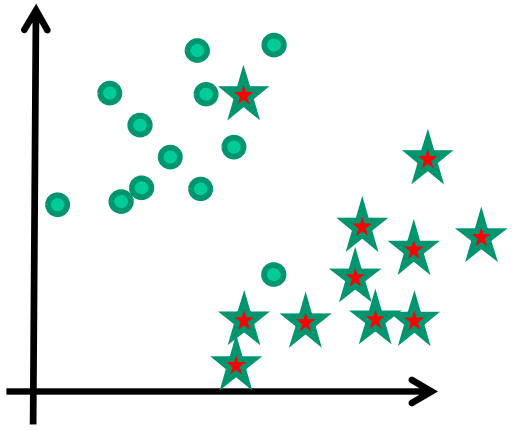
Limitations:

- Zero gradient for many solutions => several “perfect solutions”
- Data must be **linearly separable**

Many “optimal” solutions

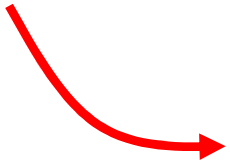


Will never converge



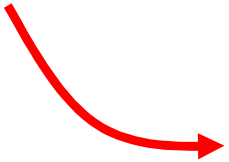
Two famous ways of improving the Perceptron

1. New **activation function** + new **Loss**



Logistic regression

2. **New network** architecture



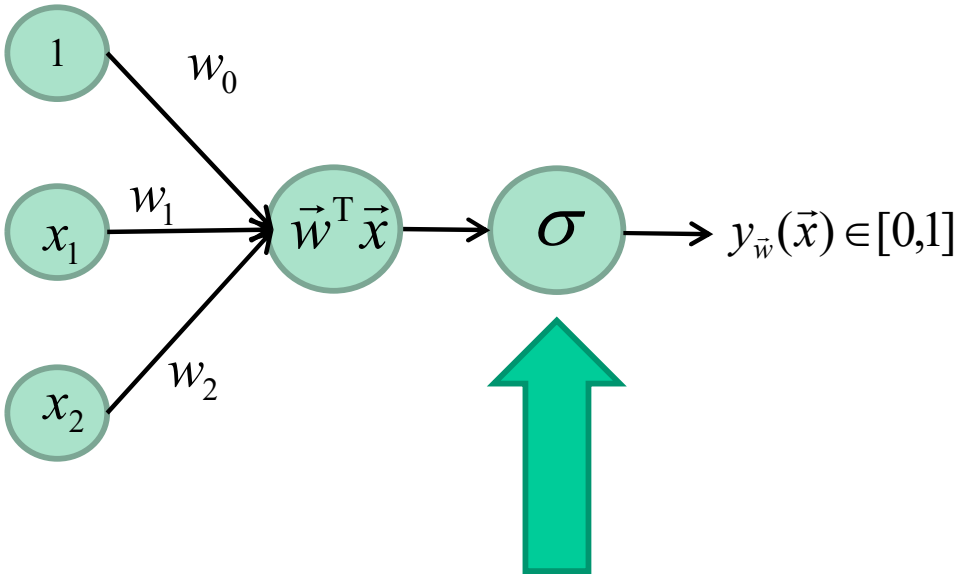
Multilayer Perceptron / CNN

Logistic regression

Logistic regression

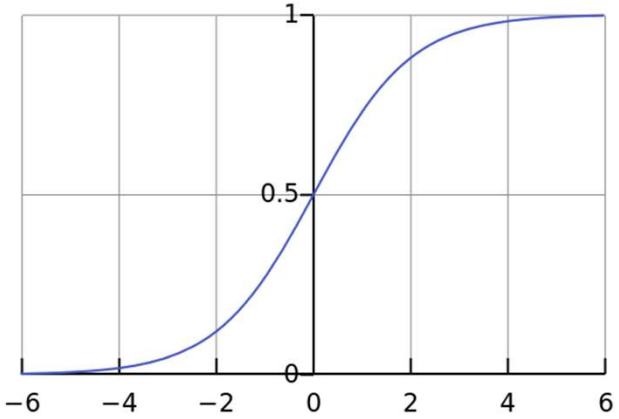
(2D, 2 classes)

New activation function: **sigmoid**



Sigmoid activation function

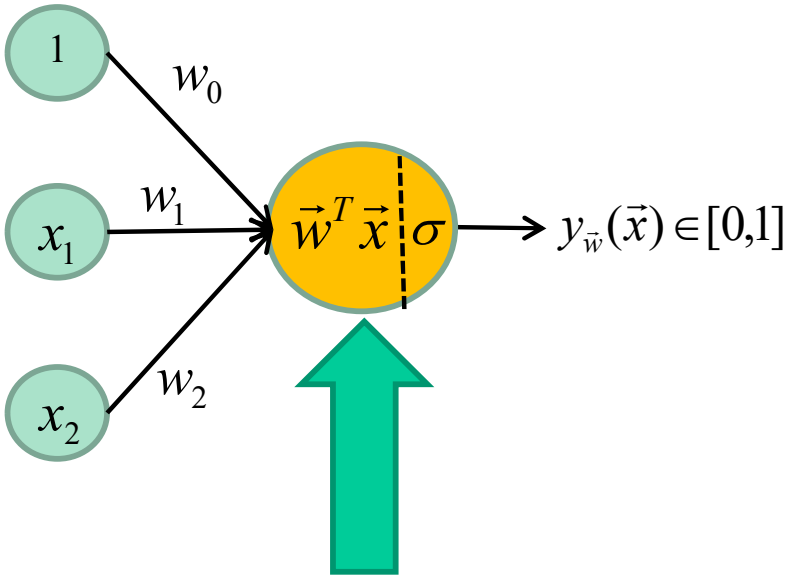
$$\sigma(t) = \frac{1}{1+e^{-t}}$$



Logistic regression

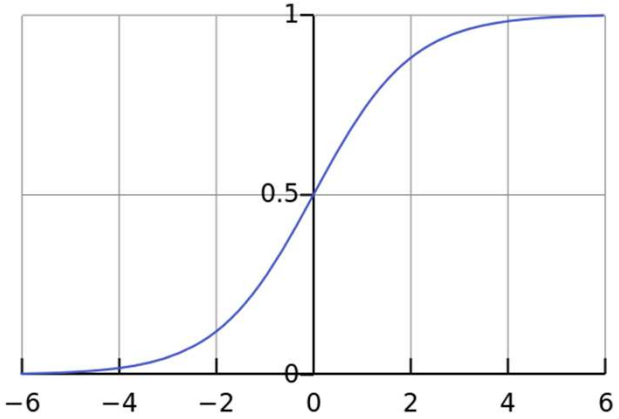
(2D, 2 classes)

New activation function: **sigmoid**



Neuron

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

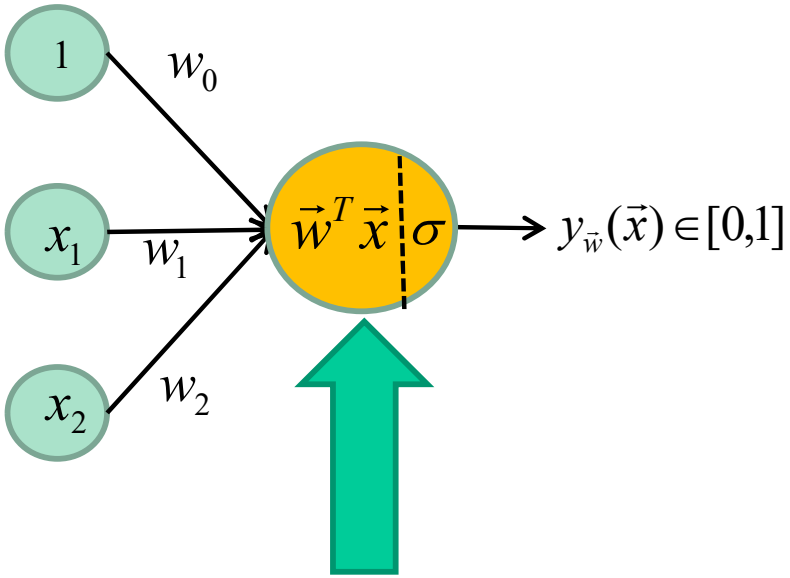


$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T\vec{x}\right) = \frac{1}{1+e^{-\vec{w}^T\vec{x}}}$$

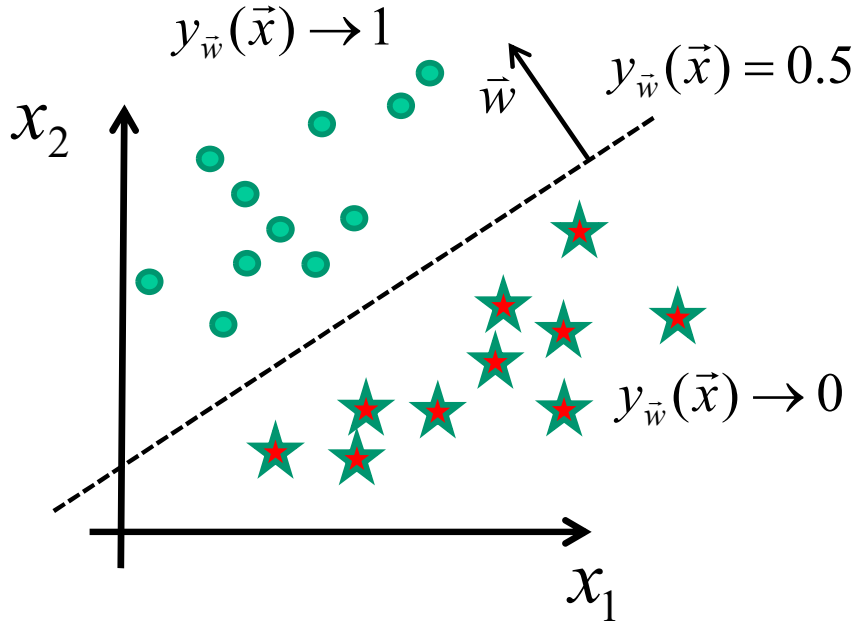
Logistic regression

(2D, 2 classes)

New activation function: **sigmoid**



Neuron



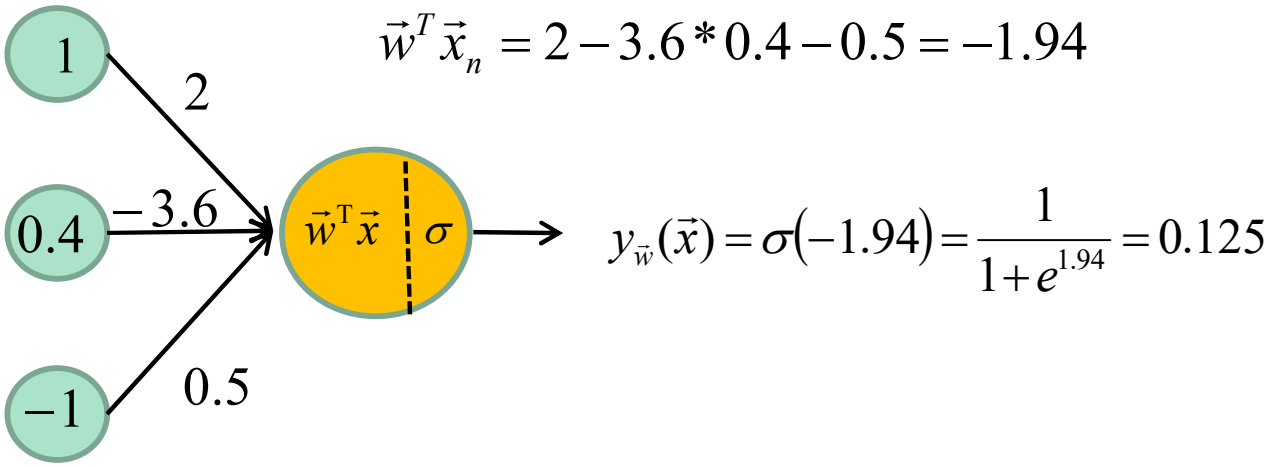
$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T \vec{x}\right)$$

Logistic regression

(2D, 2 classes)

Example

$$\vec{x}_n = (0.4, -1.0), \vec{w} = [2.0, -3.6, 0.5]$$

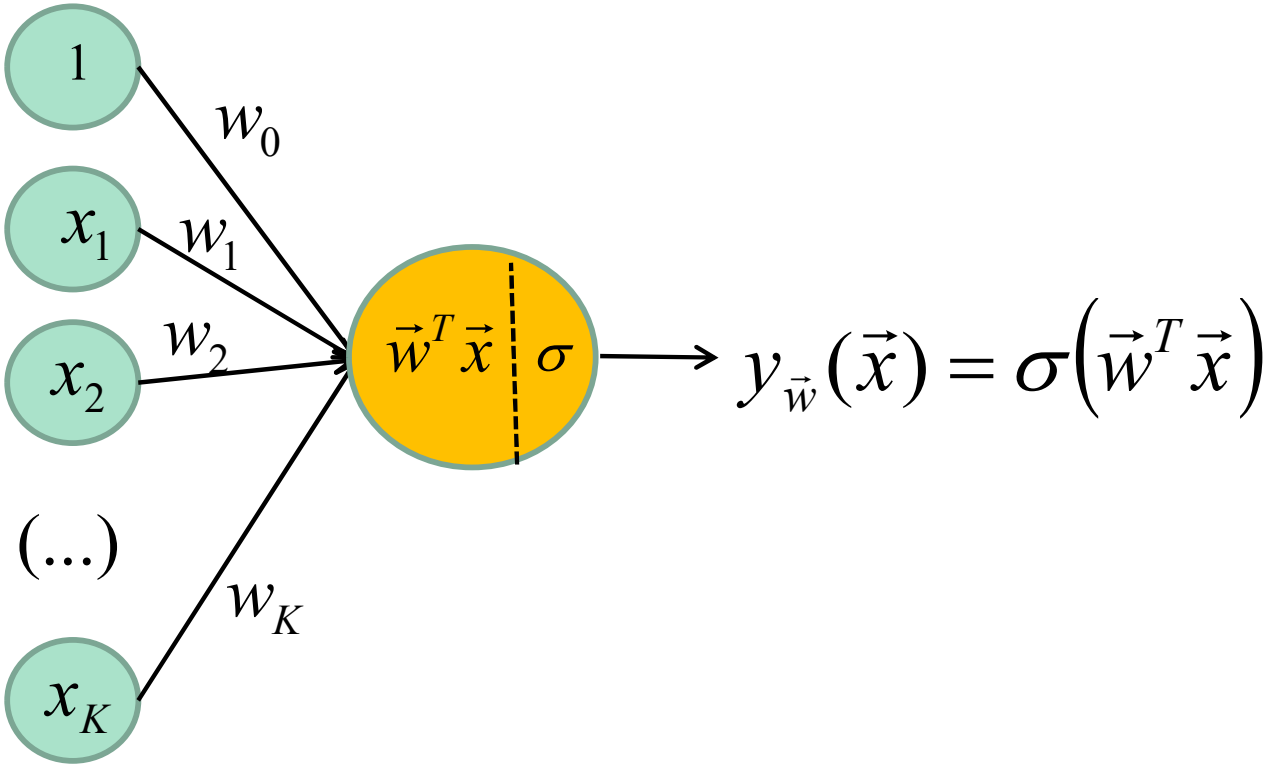


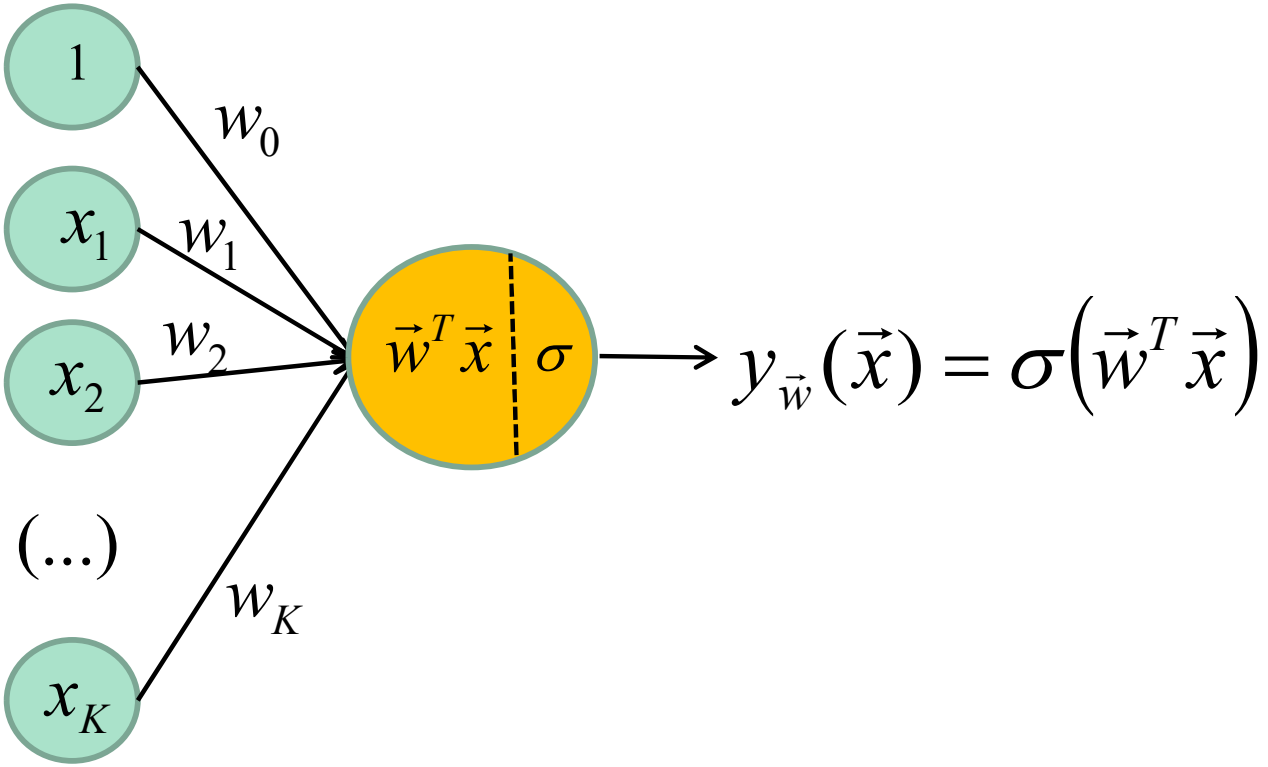
Since 0.125 is lower than 0.5, $\vec{x}_n$ is **behind** the plane.

Logistic regression

(K-D, 2 classes)

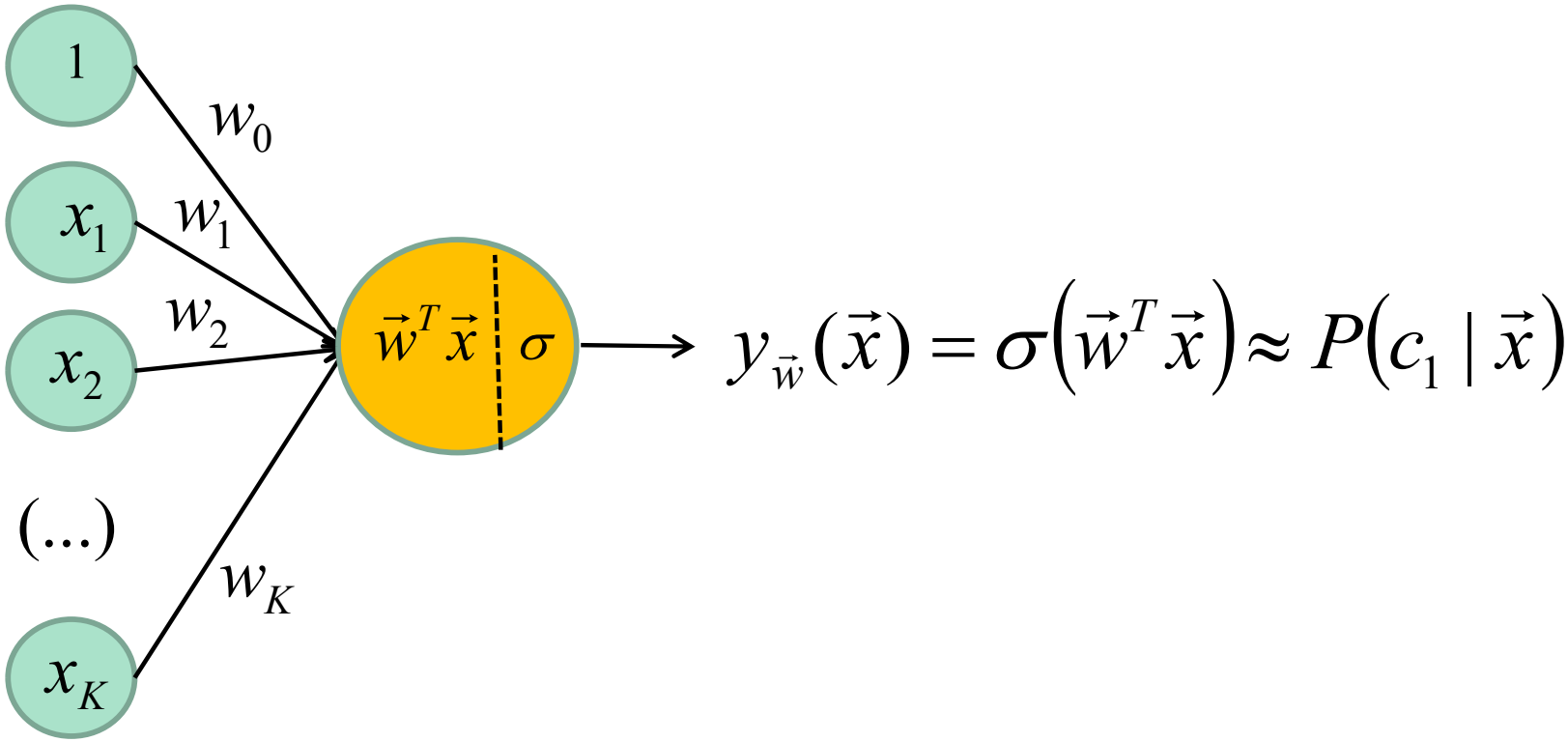
Like the Perceptron, the logistic regression accomodates for K-D vectors



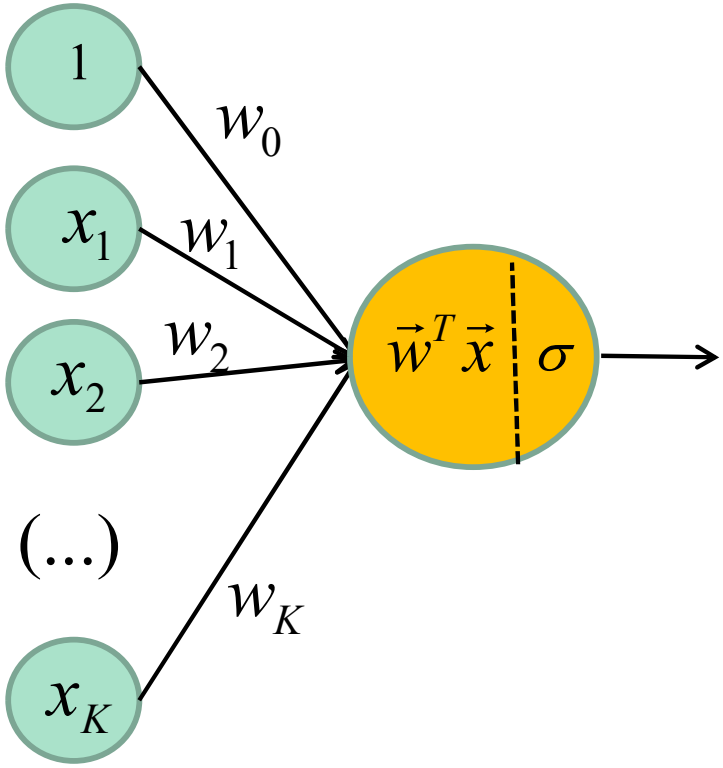


What is the loss function?

With a sigmoid, we can **simulate a conditional probability** of c1 GIVEN $\vec{x}$



With a sigmoid, we can **simulate a conditional probability** of c1 GIVEN $\vec{x}$



$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T\vec{x}\right) \approx P\left(c_1 \mid \vec{x}\right)$$

$$\Rightarrow P\left(c_0 \mid \vec{w}, \vec{x}\right) = 1 - y_{\vec{w}}(\vec{x})$$

Cost function is **–ln of the prediction**

$$L(y_{\vec{w}}(\vec{x}), D) = -\sum_{n=1}^{N} t_n \ln\left(y_{\vec{w}}(\vec{x}_n)\right) + (1 - t_n) \ln\left(1 - y_{\vec{w}}(\vec{x}_n)\right)$$

2 Class Cross entropy

We can also show that

$$\nabla_{\vec{w}} L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^{N} \left(y_{\vec{w}}(\vec{x}_n) - t_n\right) \vec{x}_n$$

As opposed to the Perceptron
the gradient does not depend
on the wrongly classified samples

Optimization of a logistic network

Stochastic gradient descent

Init $\vec{w}$
k=0, i=0
DO k=k+1
    FOR n = 1 to N

$$\nabla L = \sum_{n=1}^{N} \left(y_{\vec{w}}(\vec{x}_n) - t_n \right) \vec{x}_n$$

$$\vec{w} = \vec{w} - \eta \nabla L$$

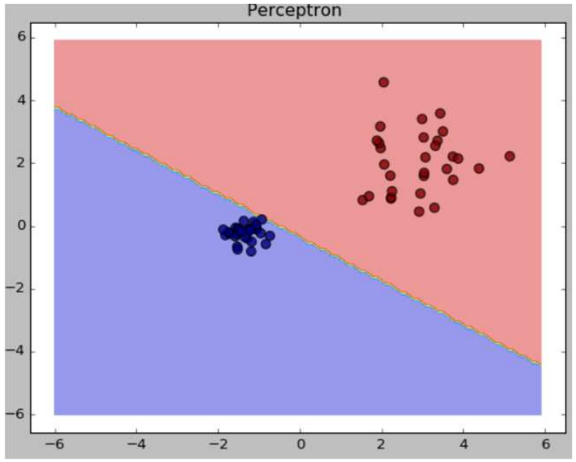
UNTIL K==K_MAX.

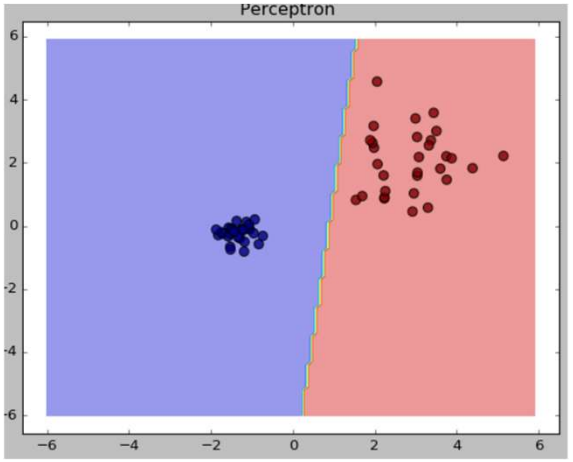
Logistic Network

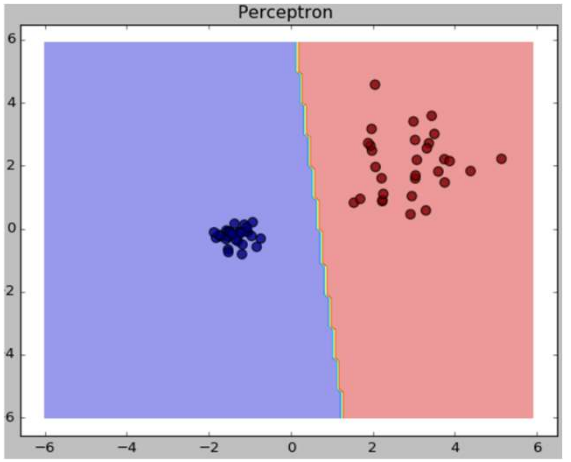
Advantages:

- **More stable than the Perceptron**
- More effective when the data is **non separable**

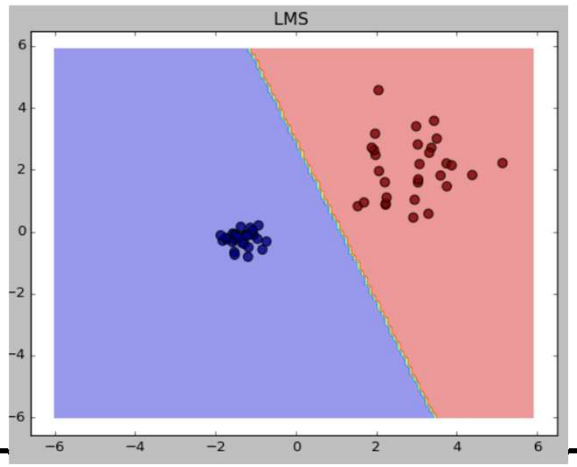
Perceptron

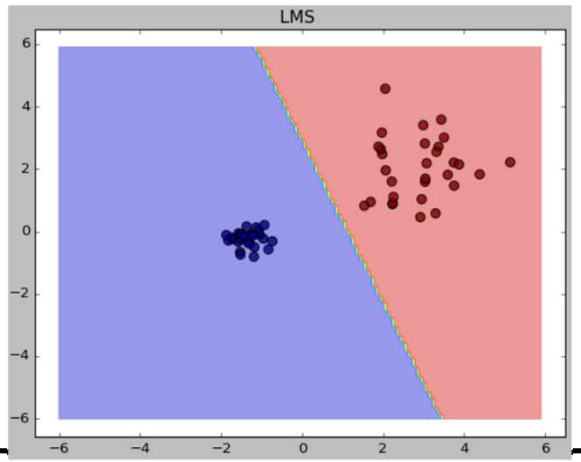


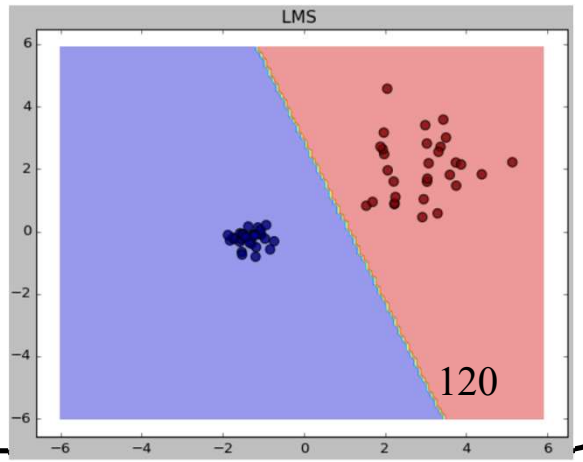




Logistic net

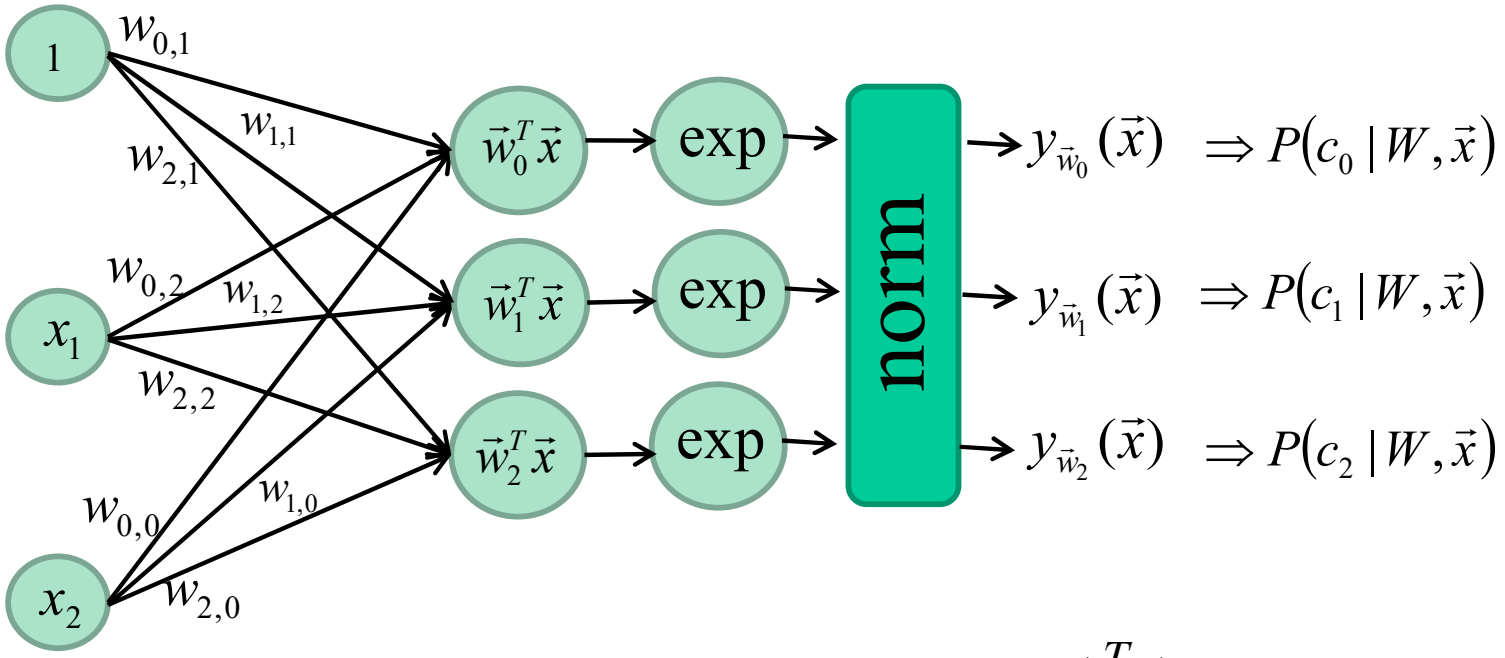






And for K>2 classes?

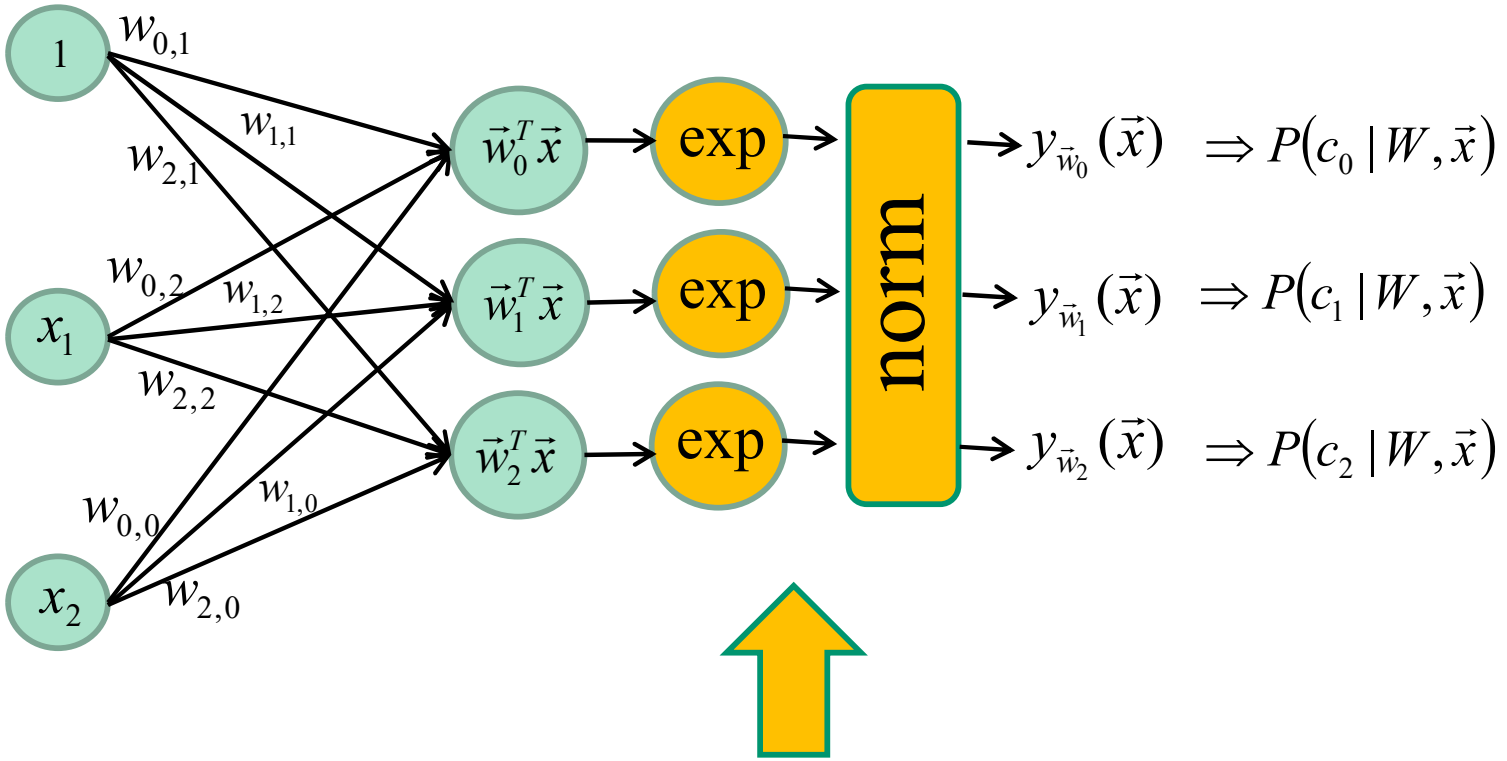
New activation function : **Softmax**



$$y_{\vec{w}_i}(\vec{x}) = \frac{e^{\vec{w}_i^T\vec{x}}}{\sum_c e^{\vec{w}_c^T\vec{x}}}$$

And for K>2 classes?

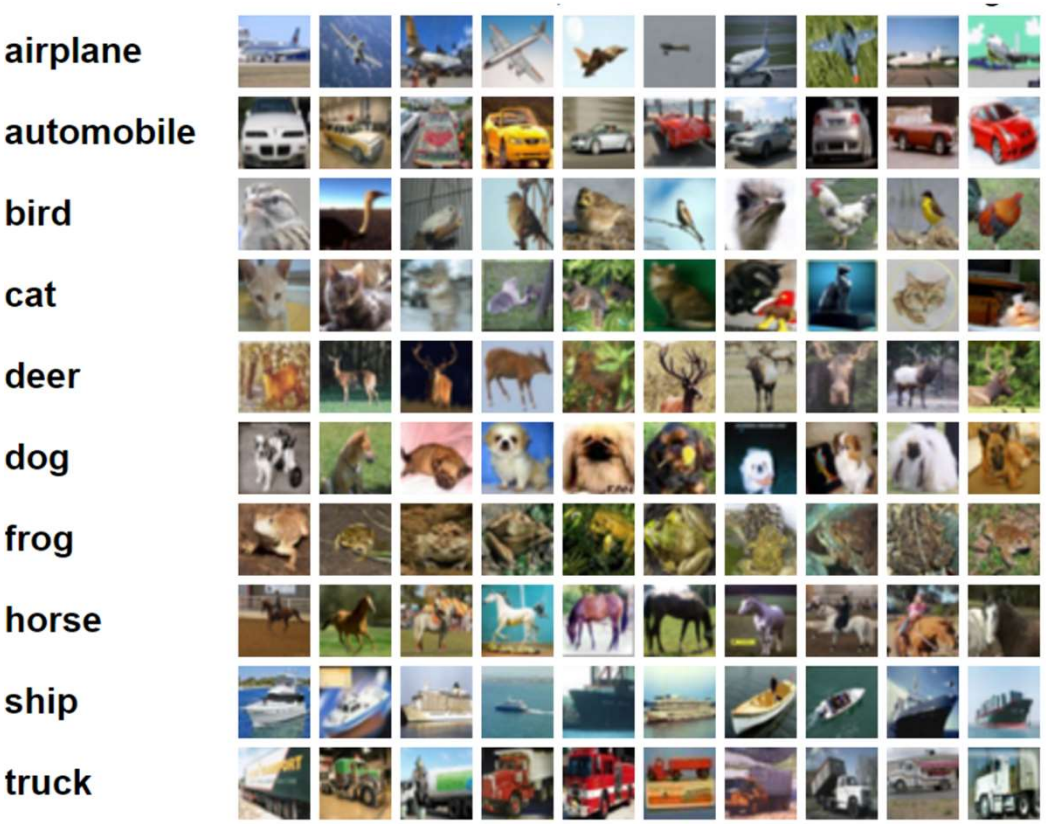
New activation function : **Softmax**



Softmax

$$y_{\vec{w}_i}(\vec{x}) = \frac{e^{\vec{w}_i^T\vec{x}}}{\sum_c e^{\vec{w}_c^T\vec{x}}}$$

And for K>2 classes?



Cifar10

$$\text{'airplane'} \Rightarrow t = [1000000000]$$
$$\text{'automobile'} \Rightarrow t = [0100000000]$$
$$\text{'bird'} \Rightarrow t = [0010000000]$$
$$\text{'cat'} \Rightarrow t = [0001000000]$$
$$\text{'deer'} \Rightarrow t = [0000100000]$$
$$\text{'dog'} \Rightarrow t = [0000010000]$$
$$\text{'frog'} \Rightarrow t = [0000001000]$$
$$\text{'horse'} \Rightarrow t = [0000000100]$$
$$\text{'ship'} \Rightarrow t = [0000000010]$$
$$\text{'truck'} \Rightarrow t = [0000000001]$$

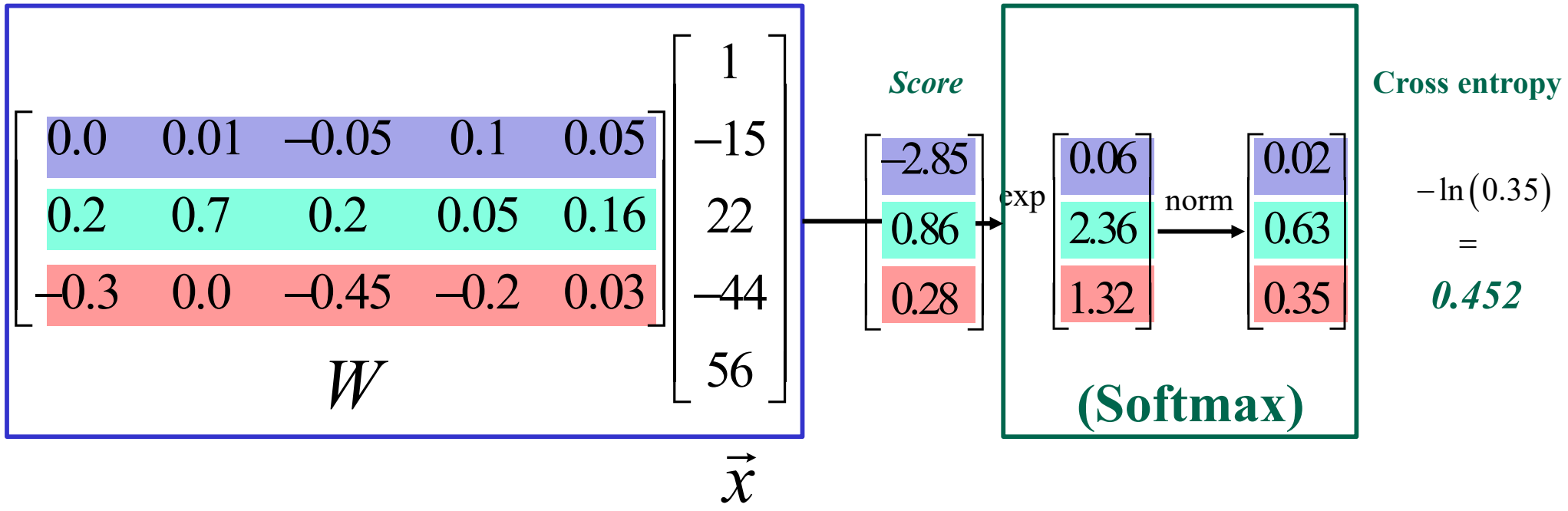
Class labels : **one-hot vectors**

K>2 classes

Cross entropy Loss

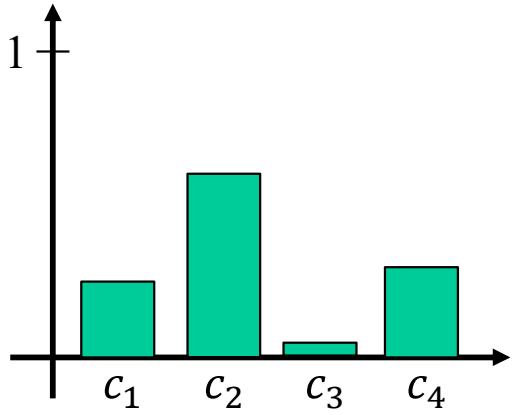
$$L\left(y_W\left(\vec{x}\right), D\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_{W_k}\left(\vec{x}_n\right)$$

$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 3$$

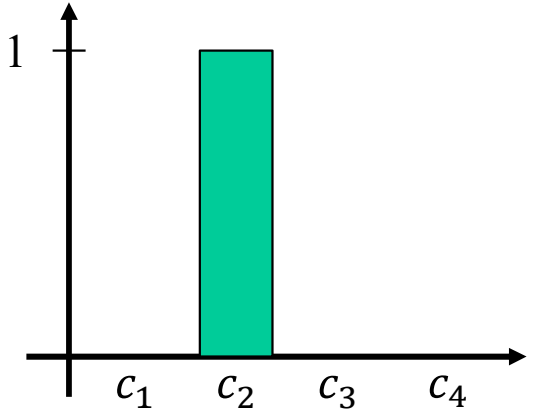


Intuition behind the cross-entropy

Softmax output

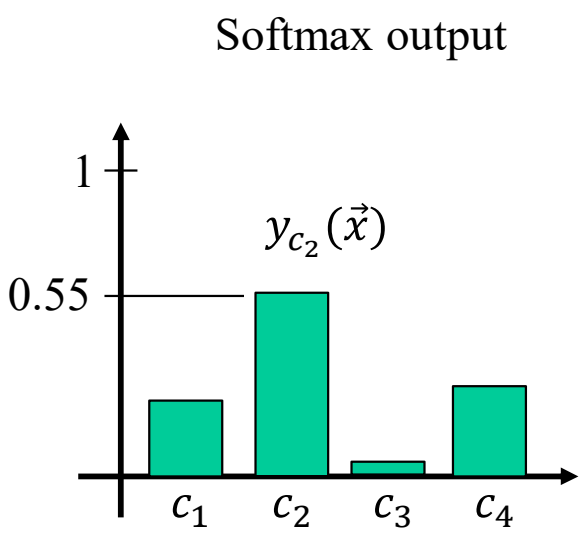


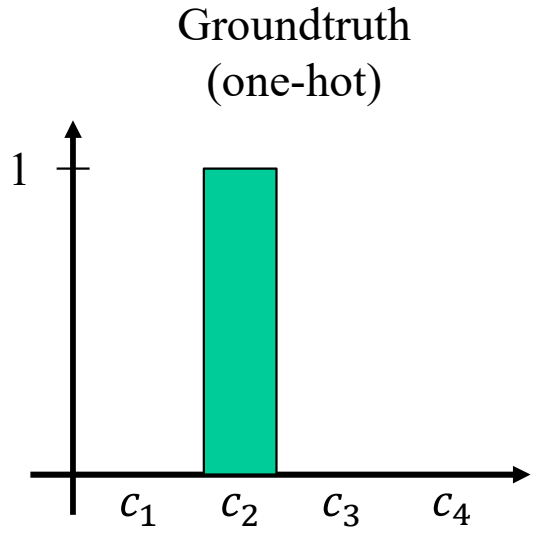
Groundtruth
(one-hot)



We can prove that the CrossEntropy is a measure of **distance between these 2 distributions**

Intuition behind the cross-entropy

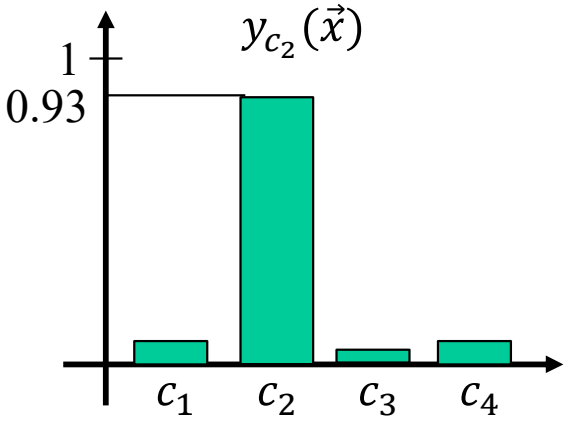




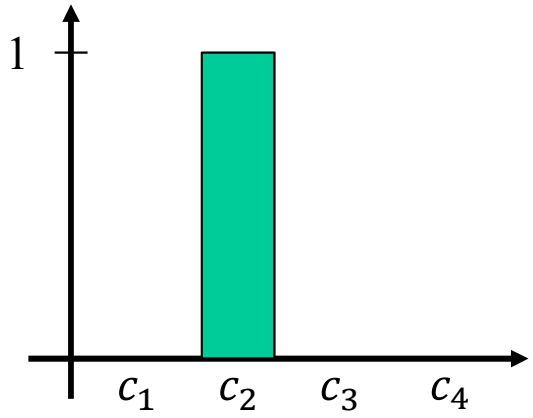
$$CE = -\ln\left(y_{c_2}\right)(\vec{x})$$
$$= -\ln(0.55)$$
$$= 0.6$$

Intuition behind the cross-entropy

Softmax output

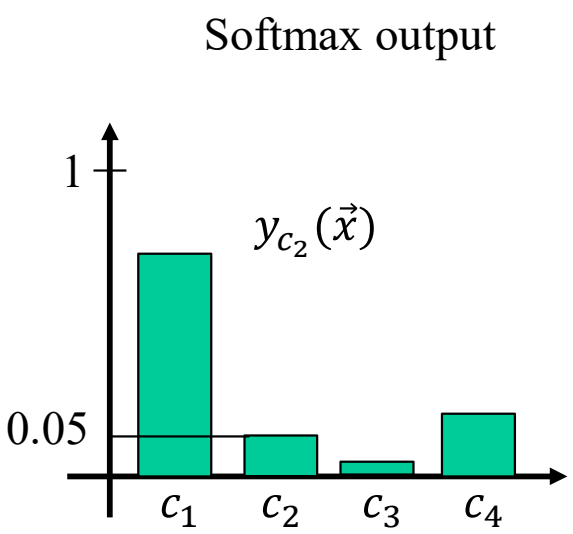


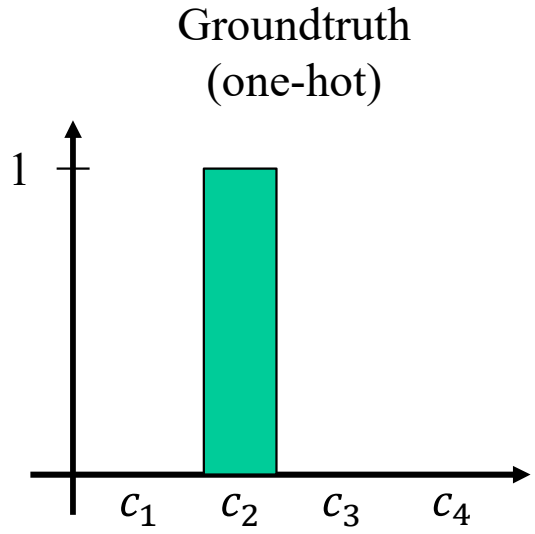
Groundtruth
(one-hot)



$$CE = -\ln\left(y_{c_2}\right)(\vec{x})$$
$$= -\ln(0.94)$$
$$= 0.07$$

Intuition behind the cross-entropy





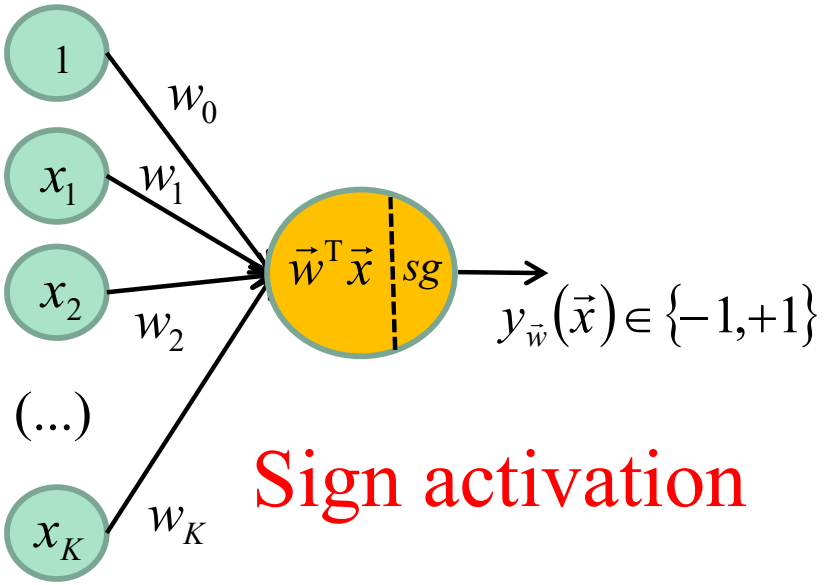
$$CE = -\ln\left(y_{c_2}\right)(\vec{x})$$
$$= -\ln(0.05)$$
$$= 3.1$$

Wow! Loooots of information!

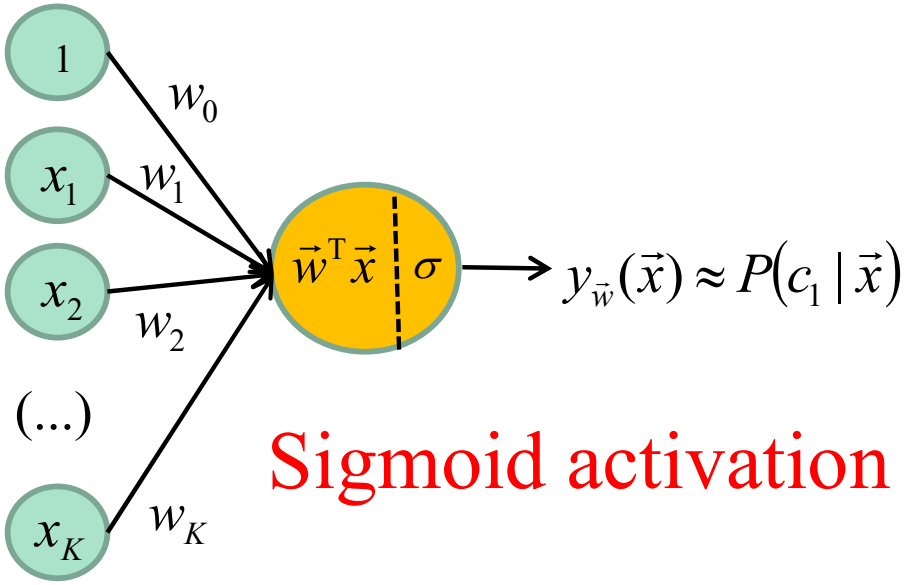
Lets recap…

Neural networks

2 classes



Sign activation

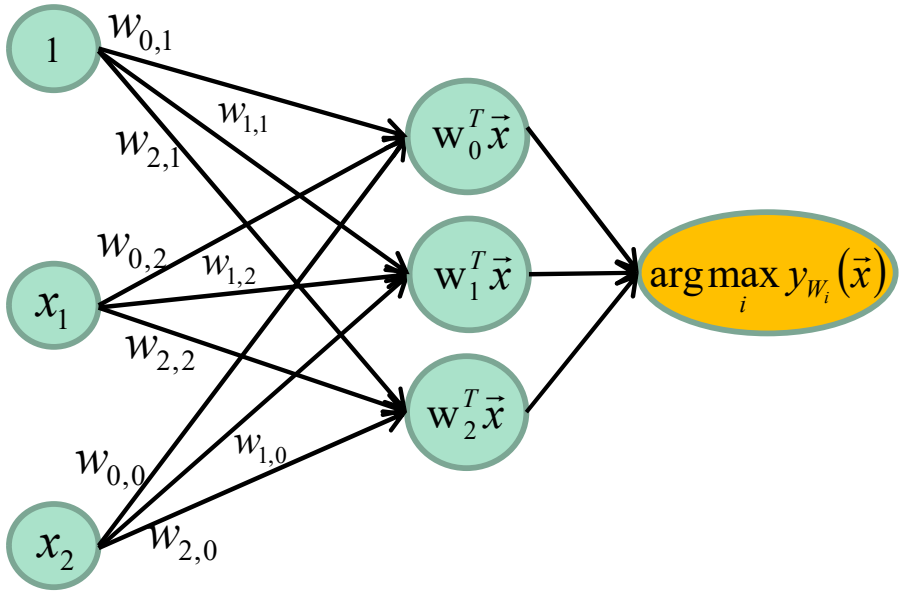


Sigmoid activation

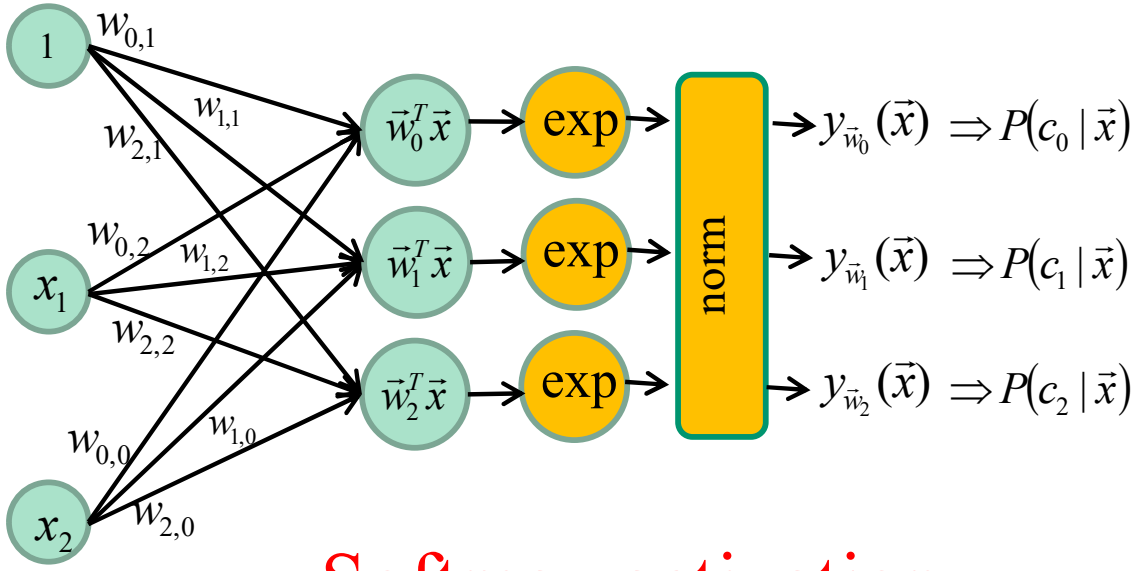
Perceptron

Logistic regression

K-Class Neural networks



Perceptron



Logistic regression

Softmax activation

Optimization

Stochastic gradient descent

Init $\vec{w}$
k=0, i=0
DO k=k+1
    FOR n = 1 to N

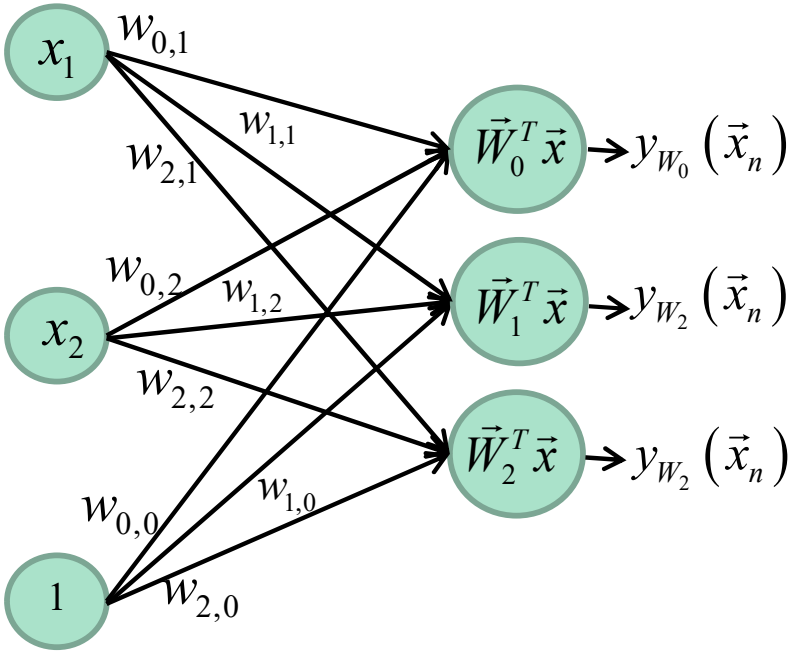
$$\vec{w} = \vec{w} - \eta \nabla L$$

UNTIL K==K_MAX.

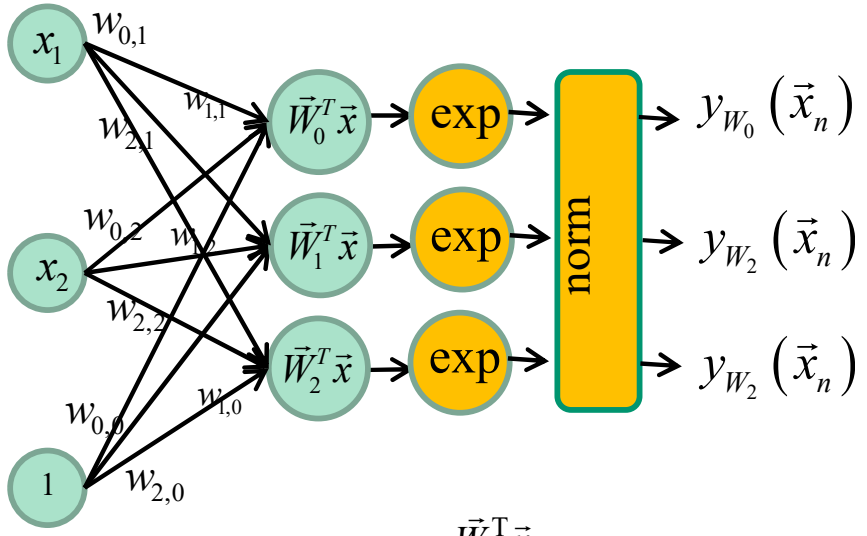
Cross entropy vs Hinge loss

Different *loss* = different **network output**

- Perceptron loss : output = vector
- Cross entropy: output = softmax



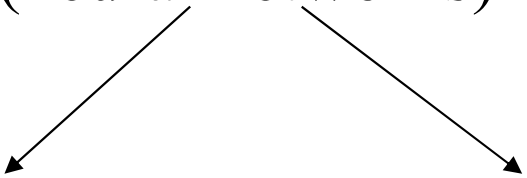
$$y_{W_i}(\vec{x}_n) = \vec{W}_i^{\mathrm{T}}\vec{x}$$



$$y_{W_i}(\vec{x}_n) = \frac{e^{\vec{W}_i^{\mathrm{T}}\vec{x}}}{\sum_j e^{\vec{W}_j^{\mathrm{T}}\vec{x}}}$$

Linear models

(neural networks)



Classification

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

Dot product

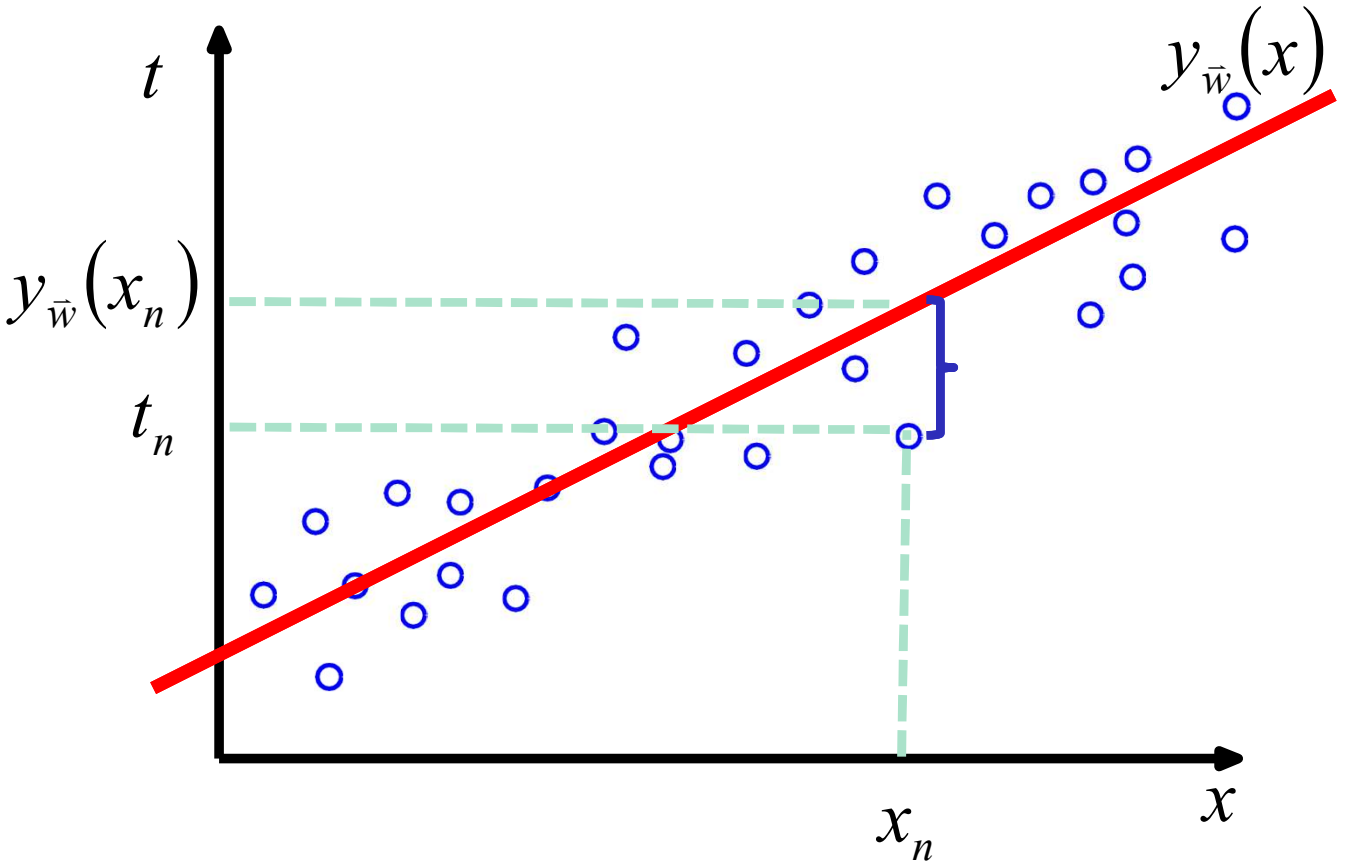
Regression

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T\vec{x} = t$$

Dot product

Problem to solve

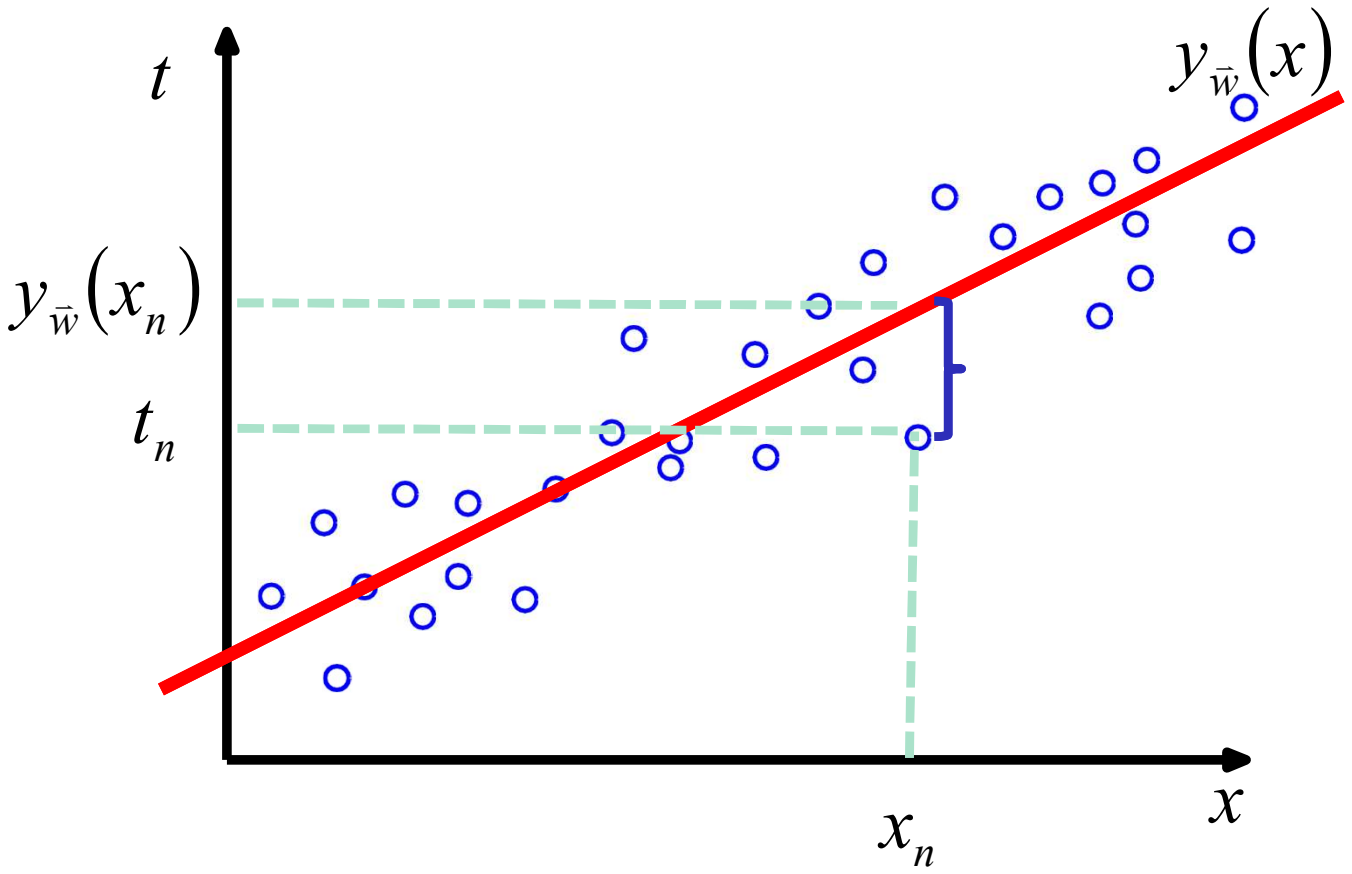
Real data are **noisy**



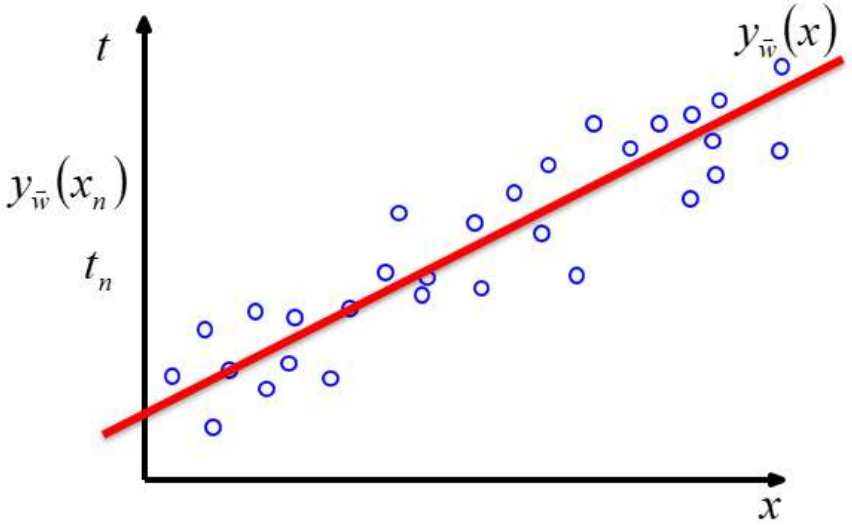
Here the goal is to make **small mistakes**.

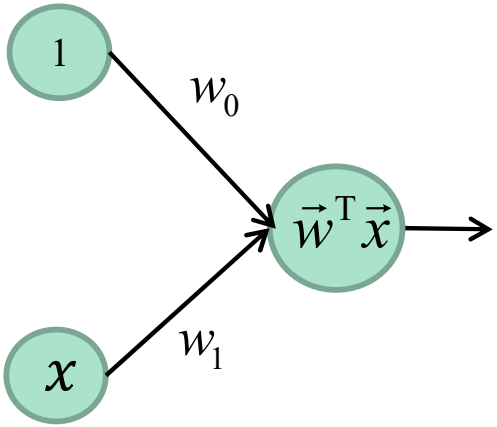
Problem to solve

$$\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} (y_{\vec{w}}(x_n) - t_n)^2$$



Solution, same as for classification



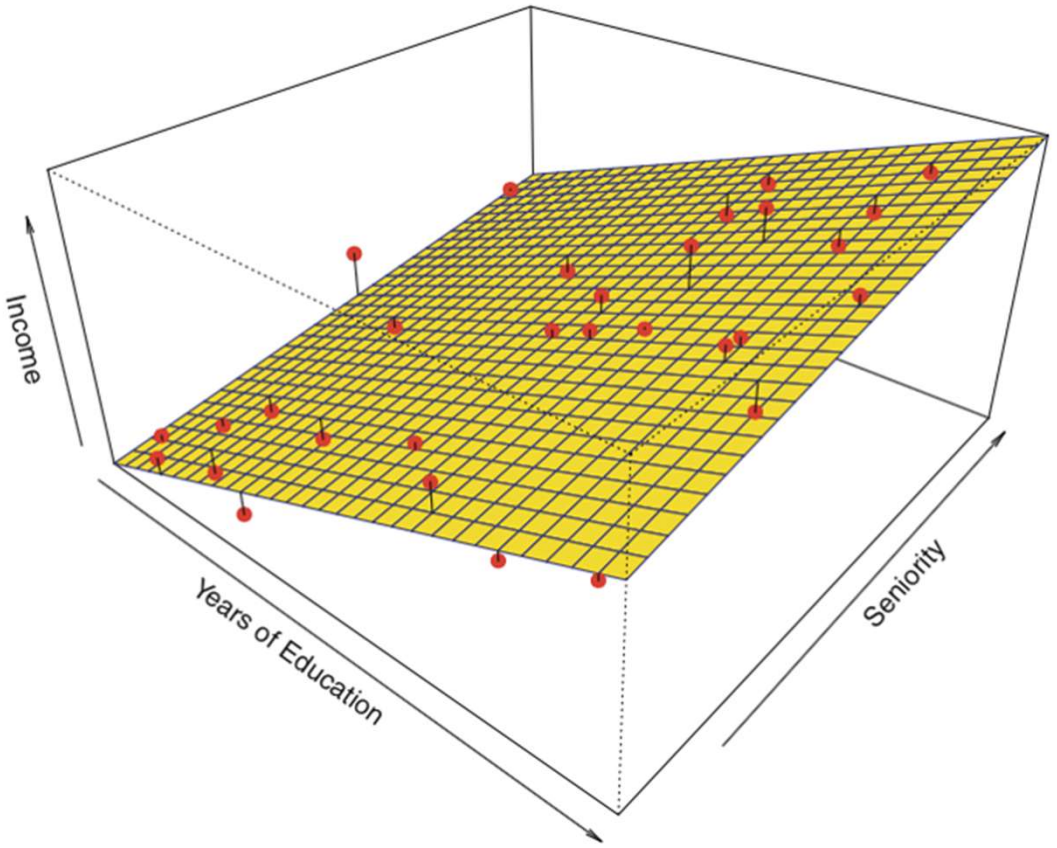


$$L = \sum_{n=1}^{N} (y_{\vec{w}}(x_n) - t_n)^2$$

Works with more than 1 variable

Example with 2 variables

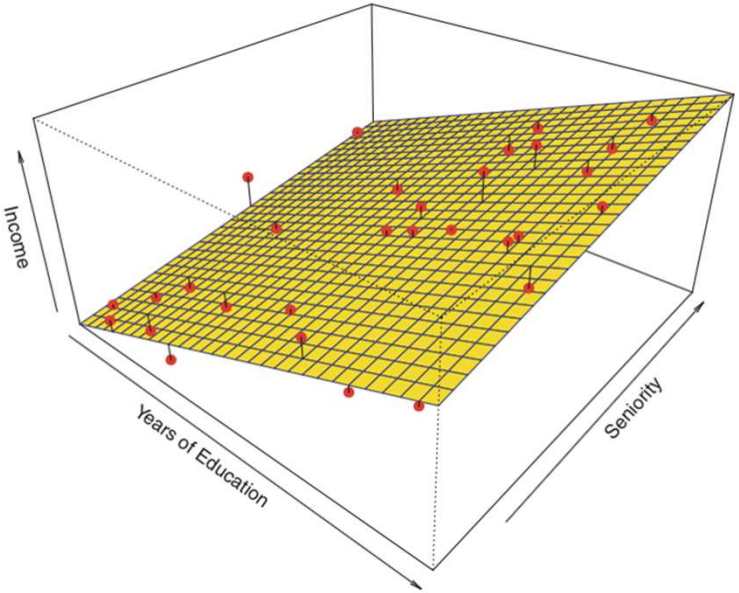
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$

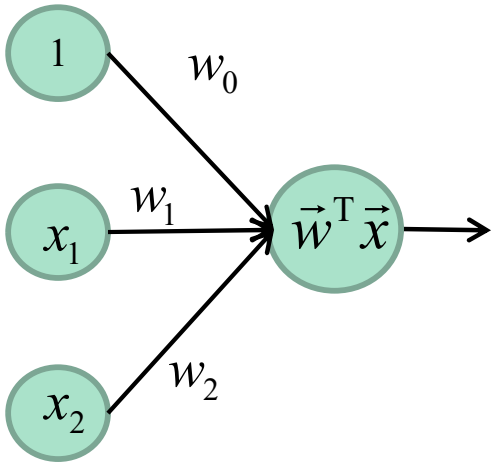


Credit : sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html

Works with more than 1 variable

Example with 2 variables





$$L = \sum_{n=1}^{N} (y_{\vec{w}}(x_n) - t_n)^2$$

Same optimization algorithm than for classification

Stochastic gradient descent

Init $\vec{w}$
k=0, i=0
DO k=k+1
  FOR n = 1 to N

$$\nabla L = 2(y_{\vec{w}}(x_n) - t_n)\vec{w}$$

$$\vec{w} = \vec{w} - \eta \nabla L$$

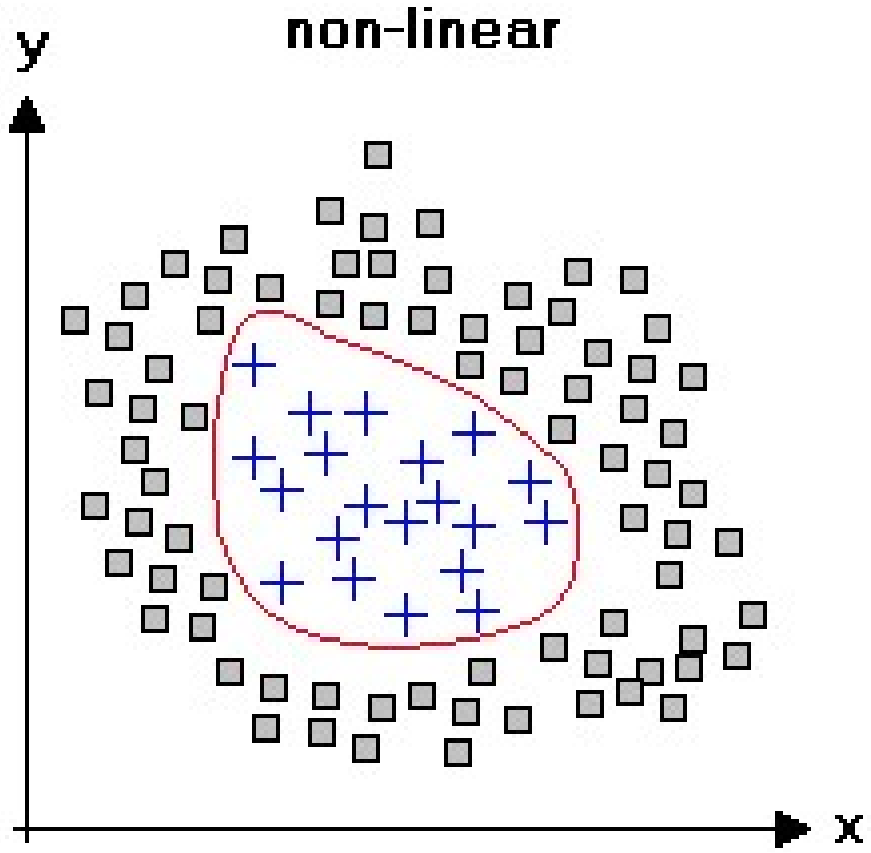
UNTIL K==K_MAX.

Now, lets go

DEEPER

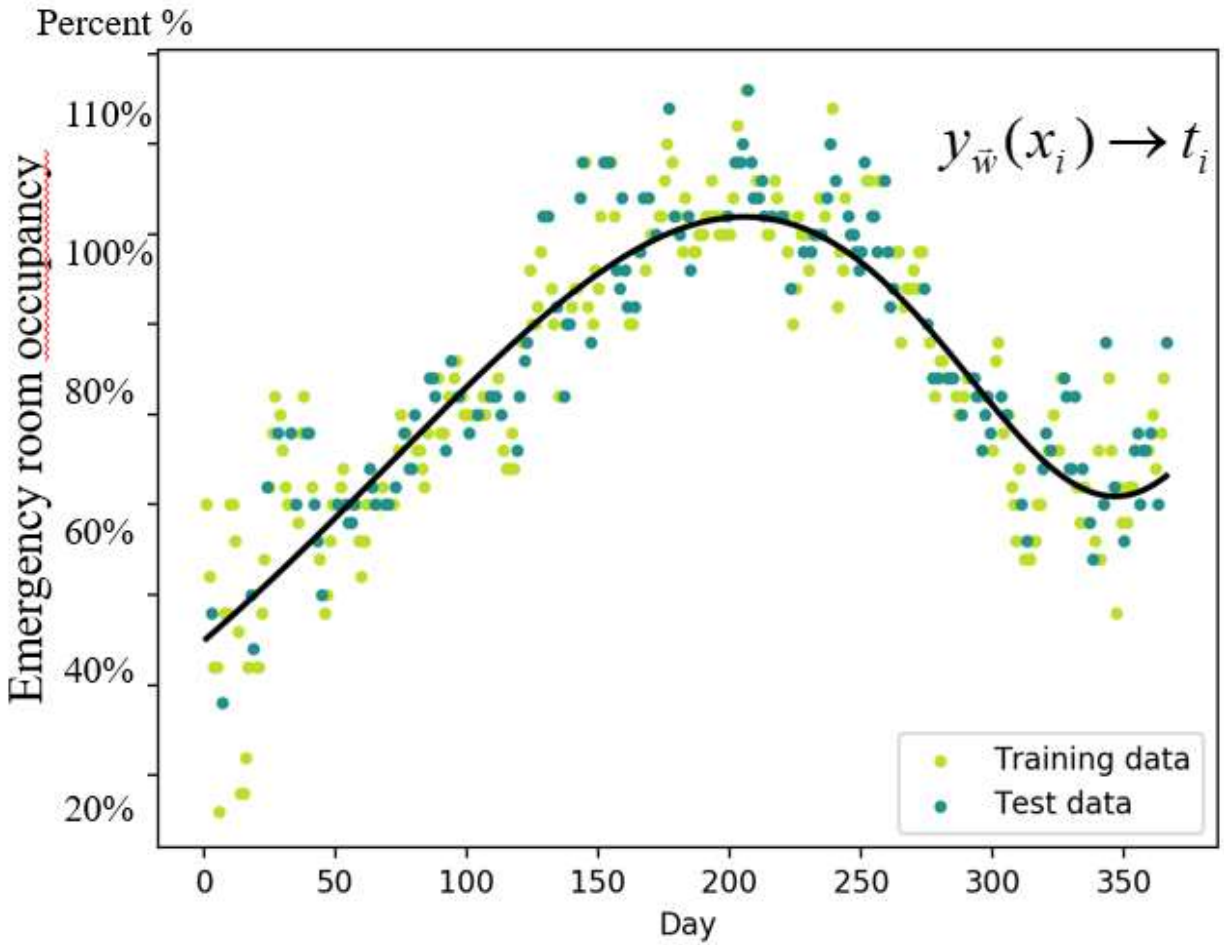
Non-linearly separable training data

Classification (2 classes)



Linear classifier = **large error rate**

Non-linear training data



Linear regression = **large error**

Non-linearly separable training data

Three classical solutions

1. Acquire more data
2. Use a non-linear classifier
3. Transform the data

Non-linearly separable training data

Three classical solutions

1. Acquire more data
2. Use a non-linear classifier
3. **Transform the data**

Linear models

(neural networks)

Classification

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

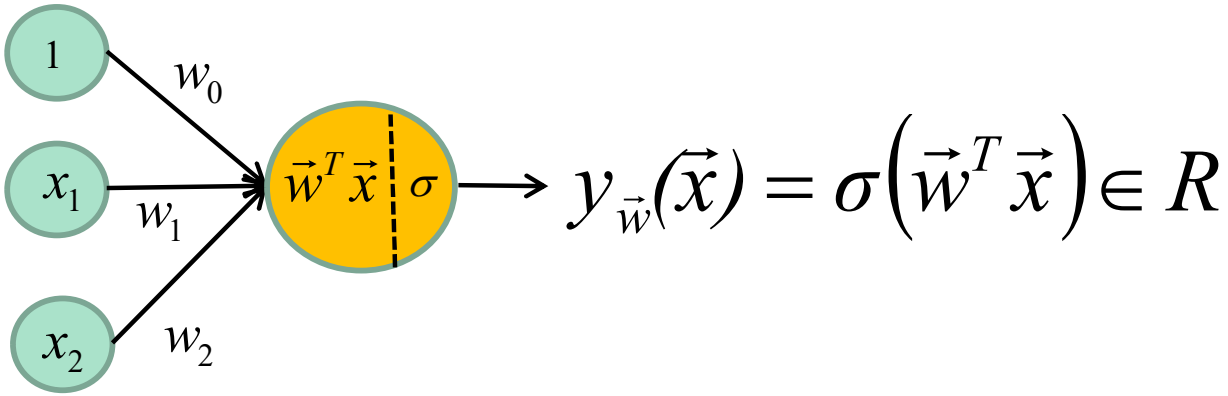
Dot product

Regression

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = t$$

Dot product

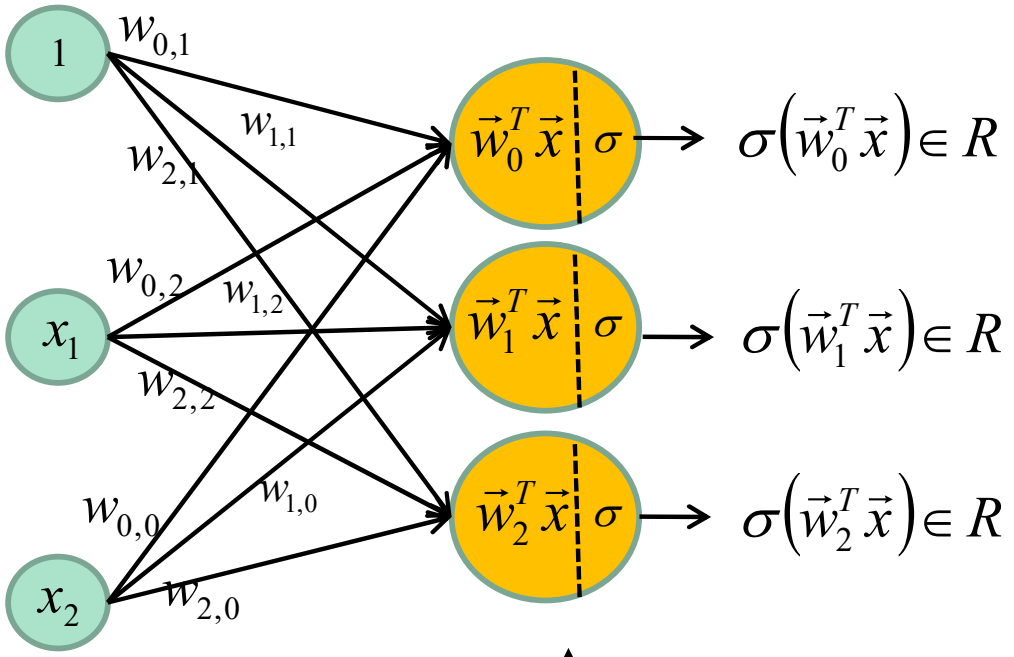
2D, 2Classes, Linear logistic regression



Input layer
(3 "neurons")

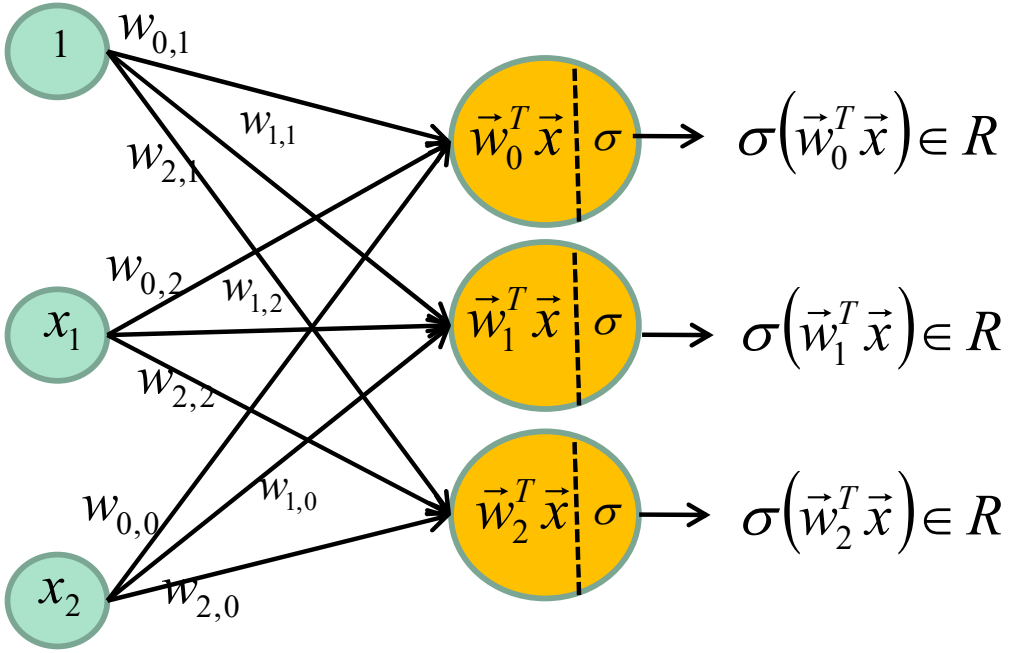
Output layer
(1 neuron with sigmoid)

Let's add 3 neurons



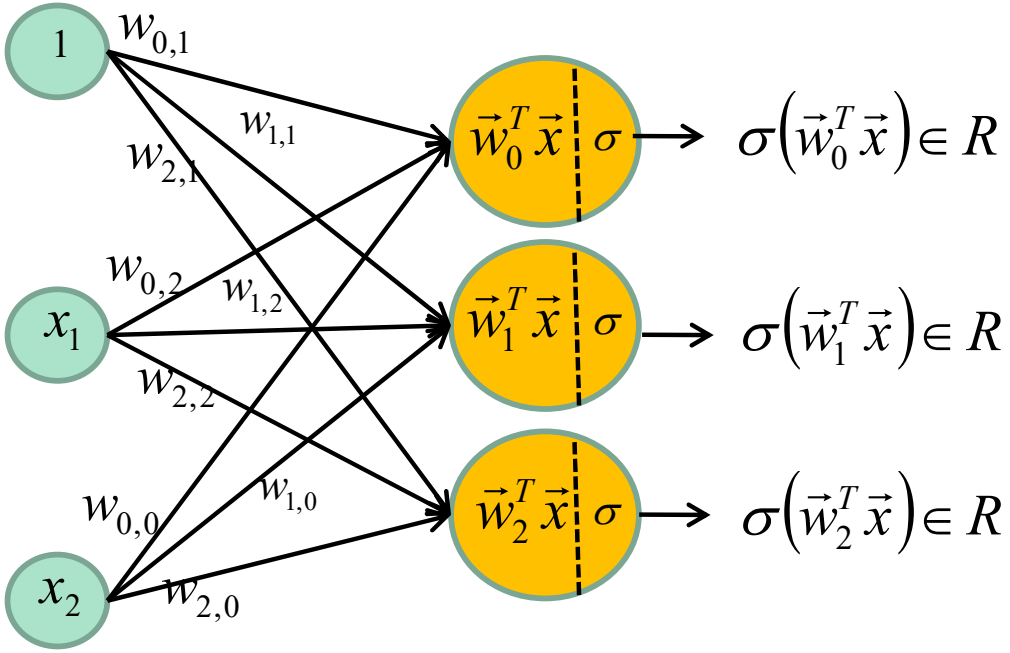
Input layer
(3 "neurons")

First layer
(3 neurons)



NOTE: The output of the first layer is a vector of **3 real** values

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}\right) \in R^3$$

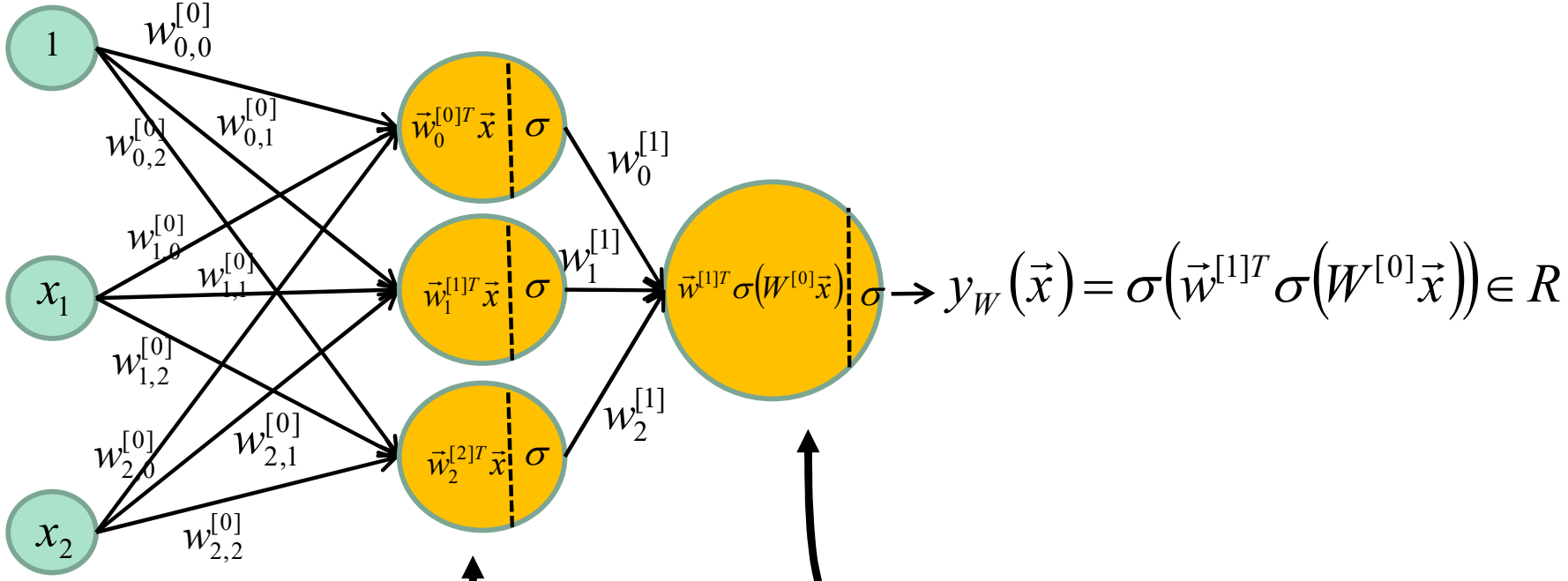


NOTE: The output of the first layer is a vector of **3 real** values

$$\sigma\left(W^{[0]}\vec{x}\right)$$

2-D, 2-Class, 1 hidden layer

If we want a **2-class Classification** via a **logistic regression** (a **cross entropy loss**) we must add an **output neuron.**

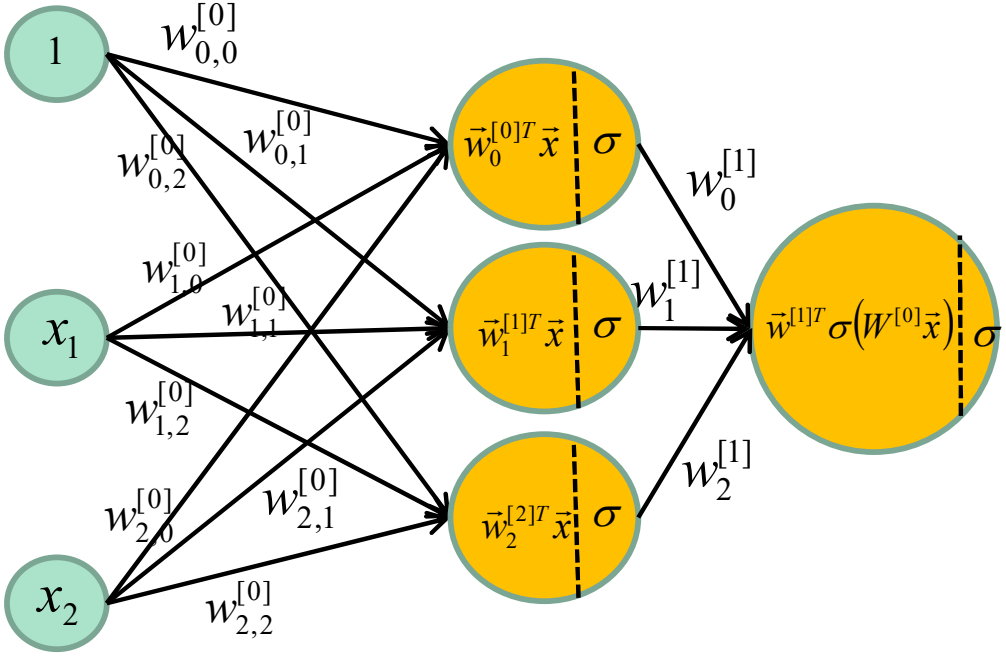


Input layer
(3 « neurons »)

hidden layer
(3 neurons)

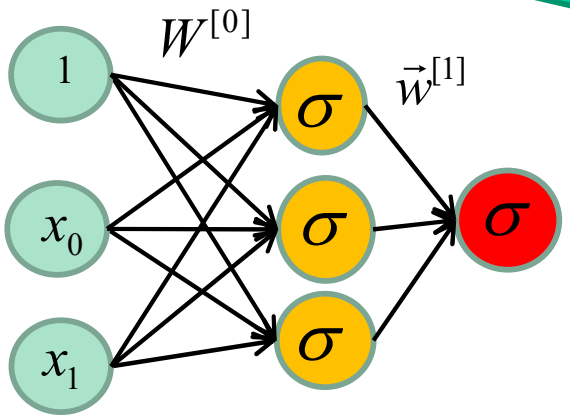
Output layer
(1 neuron)

2-D, 2-Class, 1 hidden layer



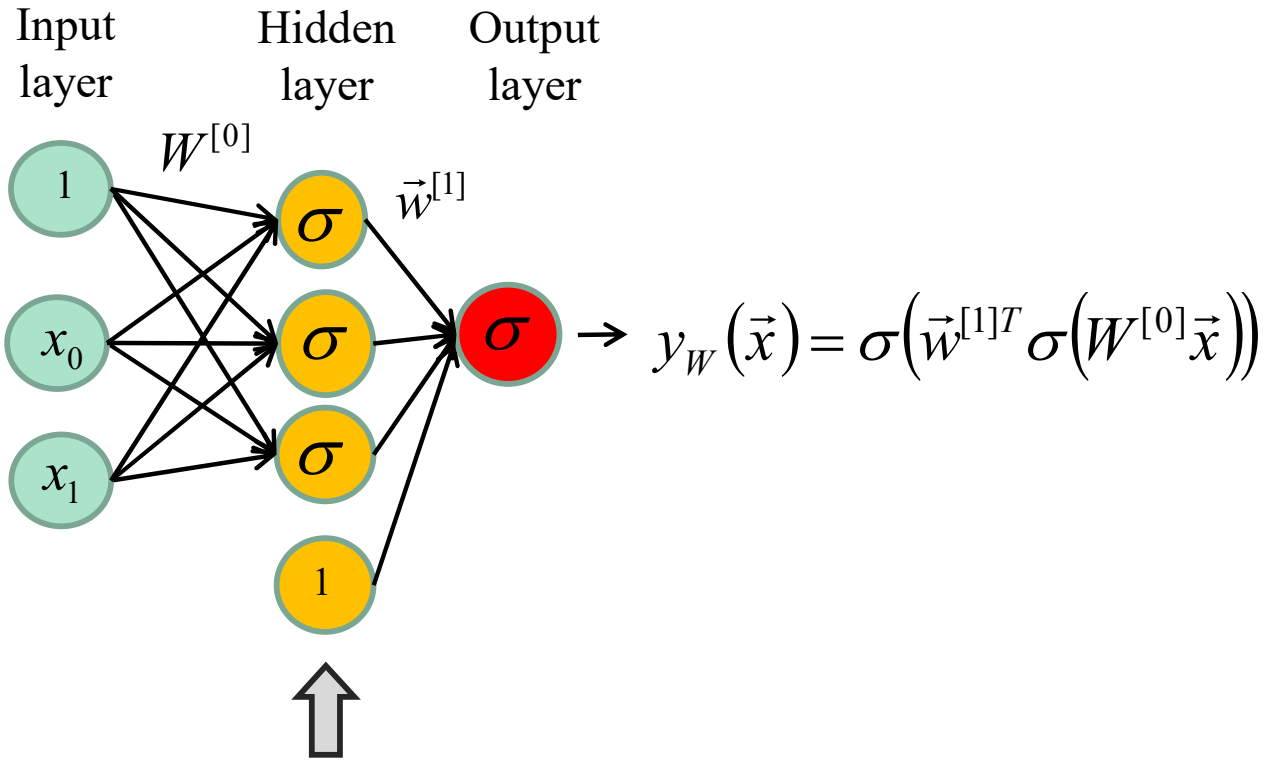
$$\rightarrow y_W(\vec{x}) = \sigma\left(\vec{w}^{[1]T}\sigma\left(W^{[0]}\vec{x}\right)\right) \in R$$

Visual simplification



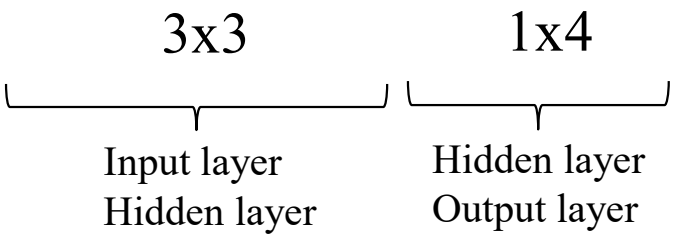
$$\rightarrow y_W(\vec{x}) = \sigma\left(\vec{w}^{[1]T}\sigma\left(W^{[0]}\vec{x}\right)\right)$$

2-D, 2-Class, 1 hidden layer

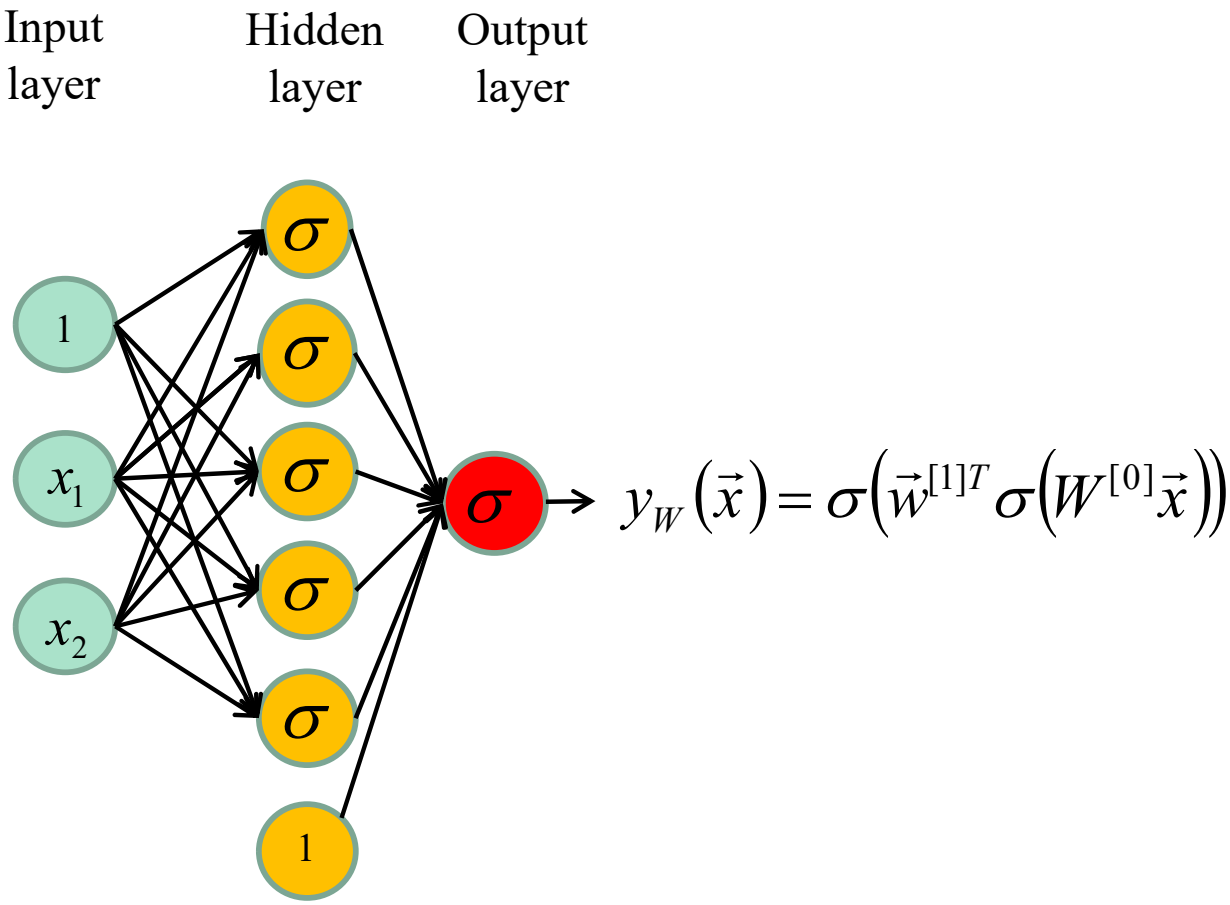


bias neuron.

This network contains a total of **13 parameters**

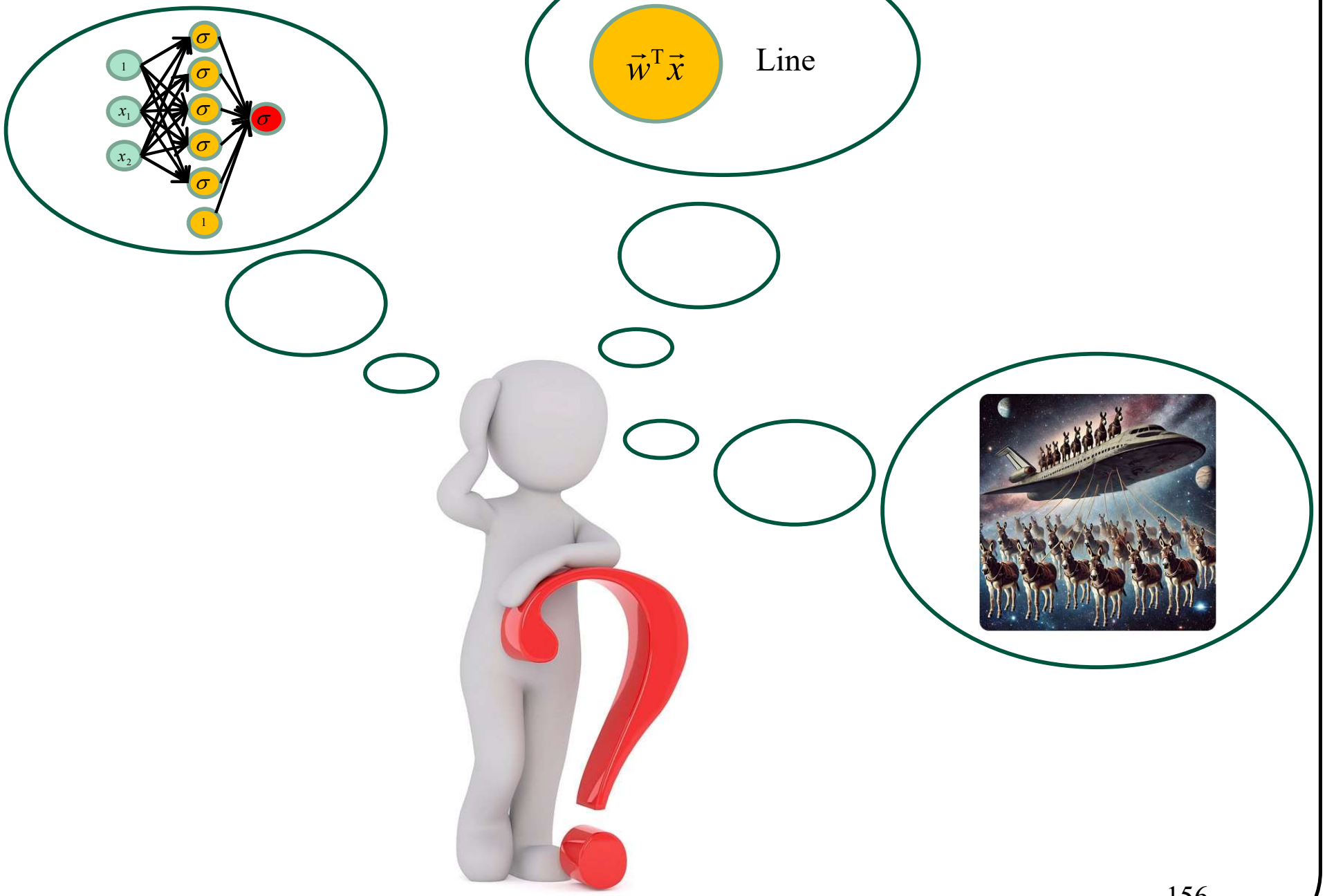


2-D, 2-Class, 1 hidden layer



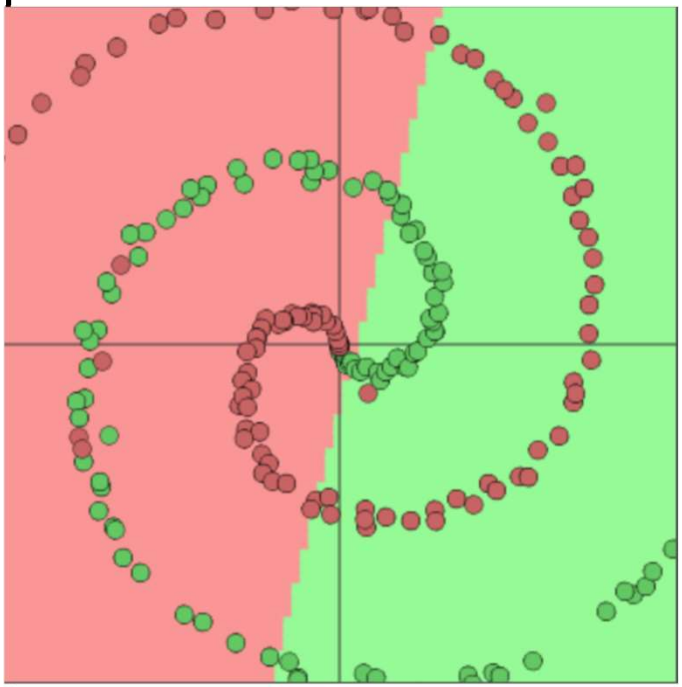
Increasing the number of neurons = increasing the **capacity of the model**

This network has 5x3+1x6=**21 parameters**


1
x_1
x_2
σ
σ
σ
σ
σ
σ
1
$\vec{w}^T\vec{x}$
Line

Nb neurons VS Capacity

No hidden neuron

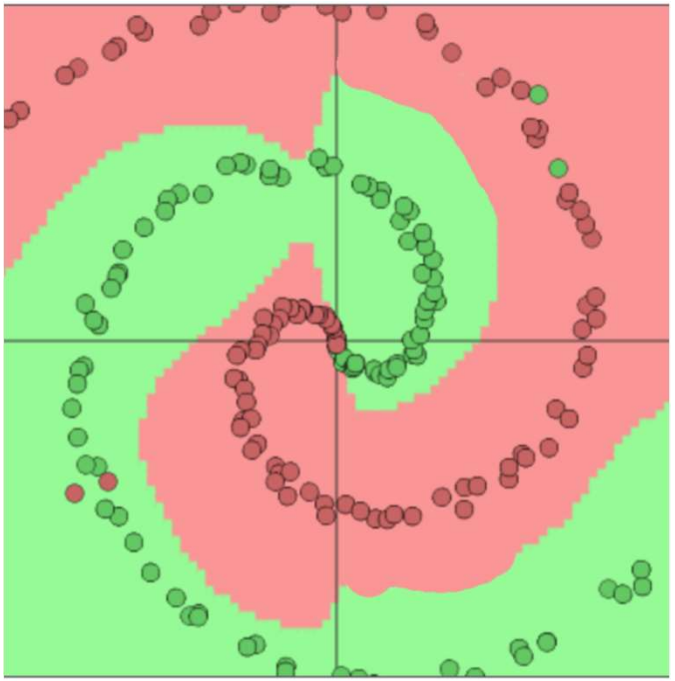


Linear classification

Underfitting

(low capacity)

12 hidden neurons

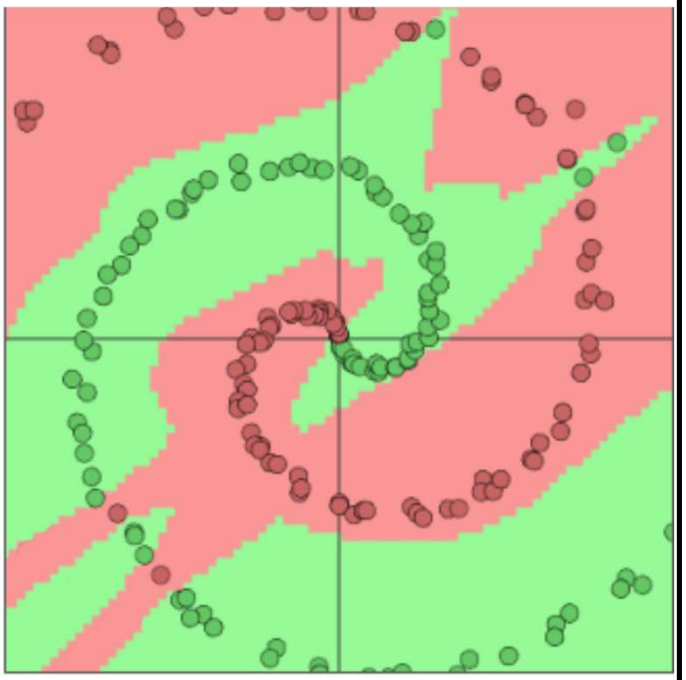


Non linear classification

Good result

(good capacity)

60 hidden neurons



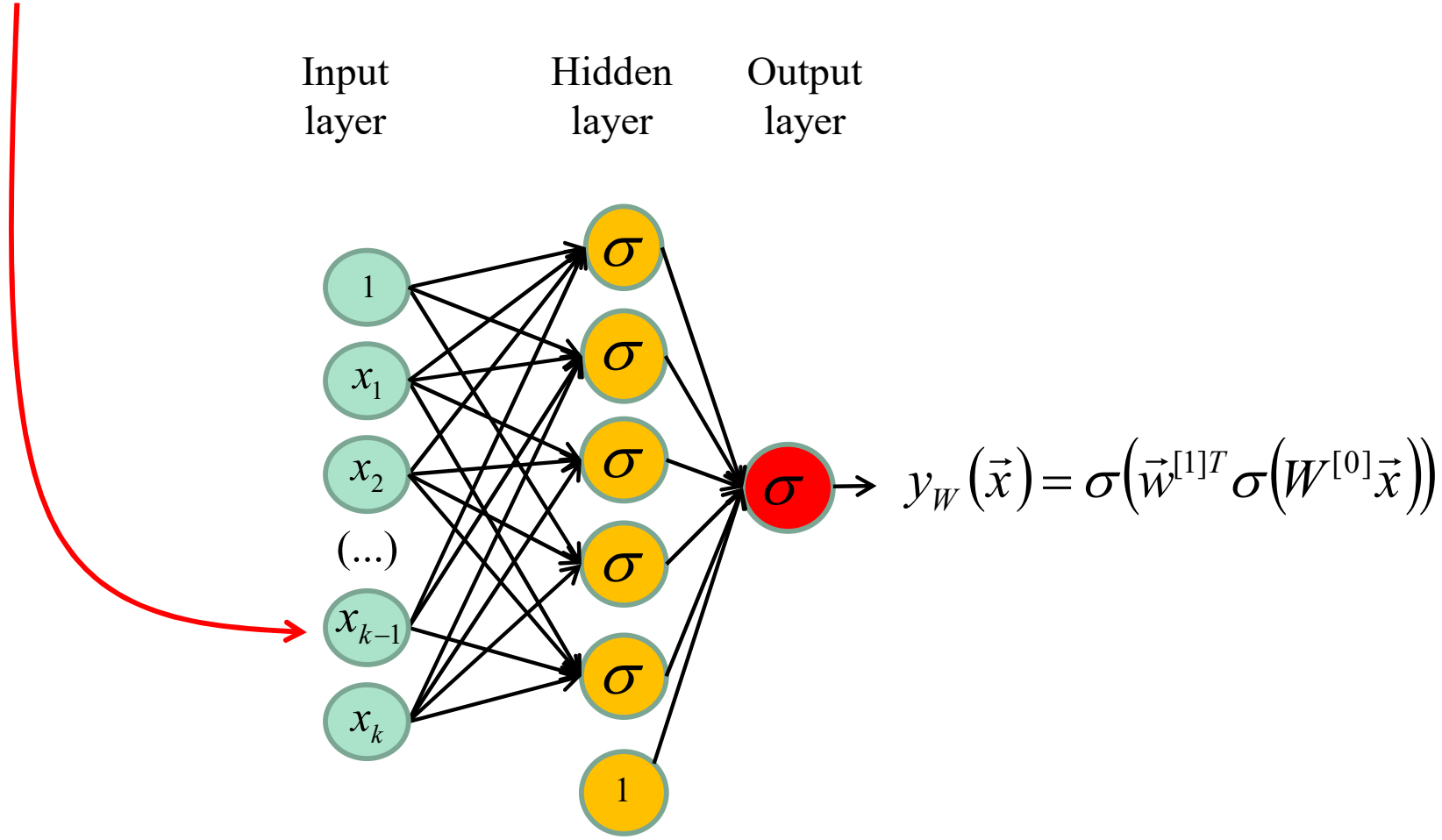
Non linear classification

Over fitting

(too large capacity)

http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

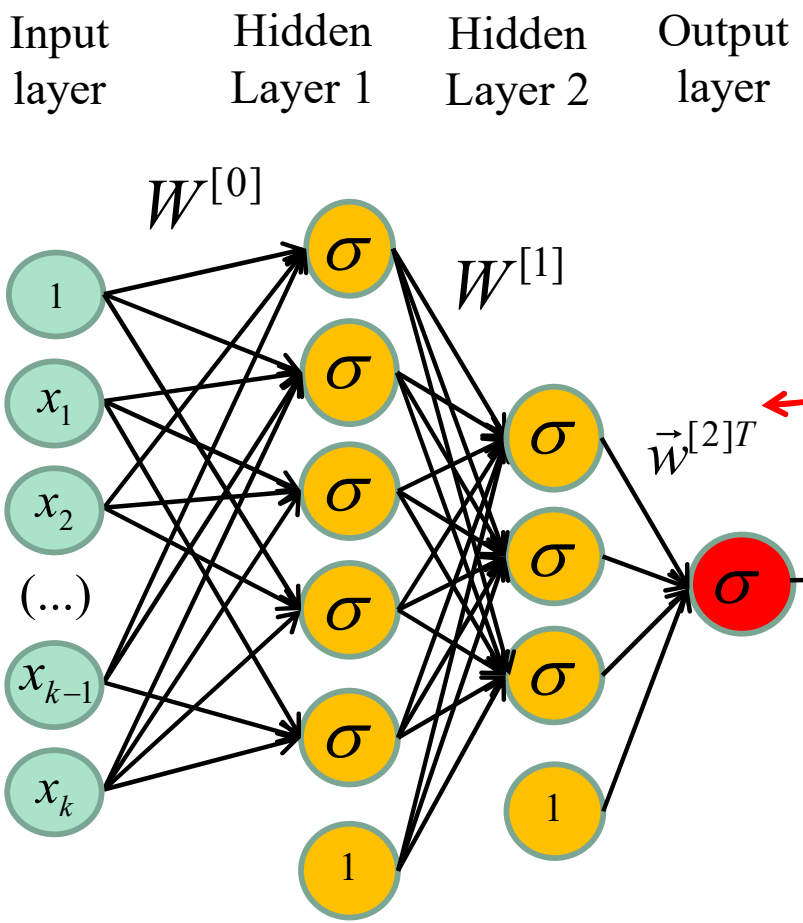
kD, 2Classes, 1 hidden layer



Increasing the dimensionality of the data = **more weights**

This network has 5x(k+1)+1x6 **parameters**

kD, 2Classes, **2 hidden layers**



$$W^{[0]} \in R^{5\times k+1}$$

$$W^{[1]} \in R^{3\times 6}$$

$$\vec{w}^{[2]} \in R^{4}$$

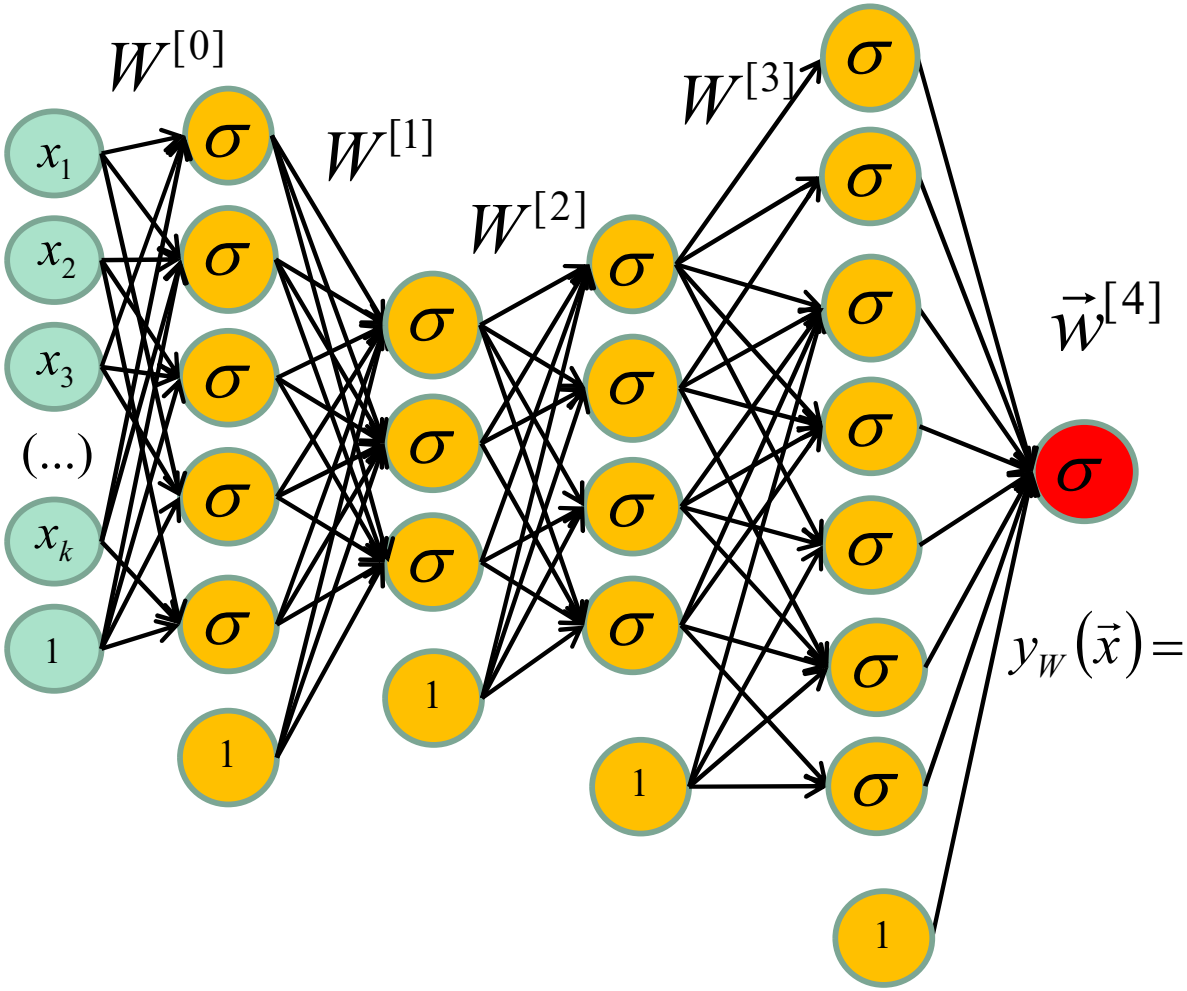
$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[2]T}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)$$

Adding a hidden layer = Adding a matrix multiplication

This network has 5x(k+1)+6x3 + 1x4 **parameters**

kD, 2 Classes, 4 hidden layer network

Input layer | Hidden Layer 1 | Hidden Layer 2 | Hidden Layer 3 | Hidden Layer 4 | Output layer



$$W^{[0]} \in R^{5\times k+1}$$

$$W^{[1]} \in R^{3\times 6}$$

$$W^{[2]} \in R^{4\times 4}$$

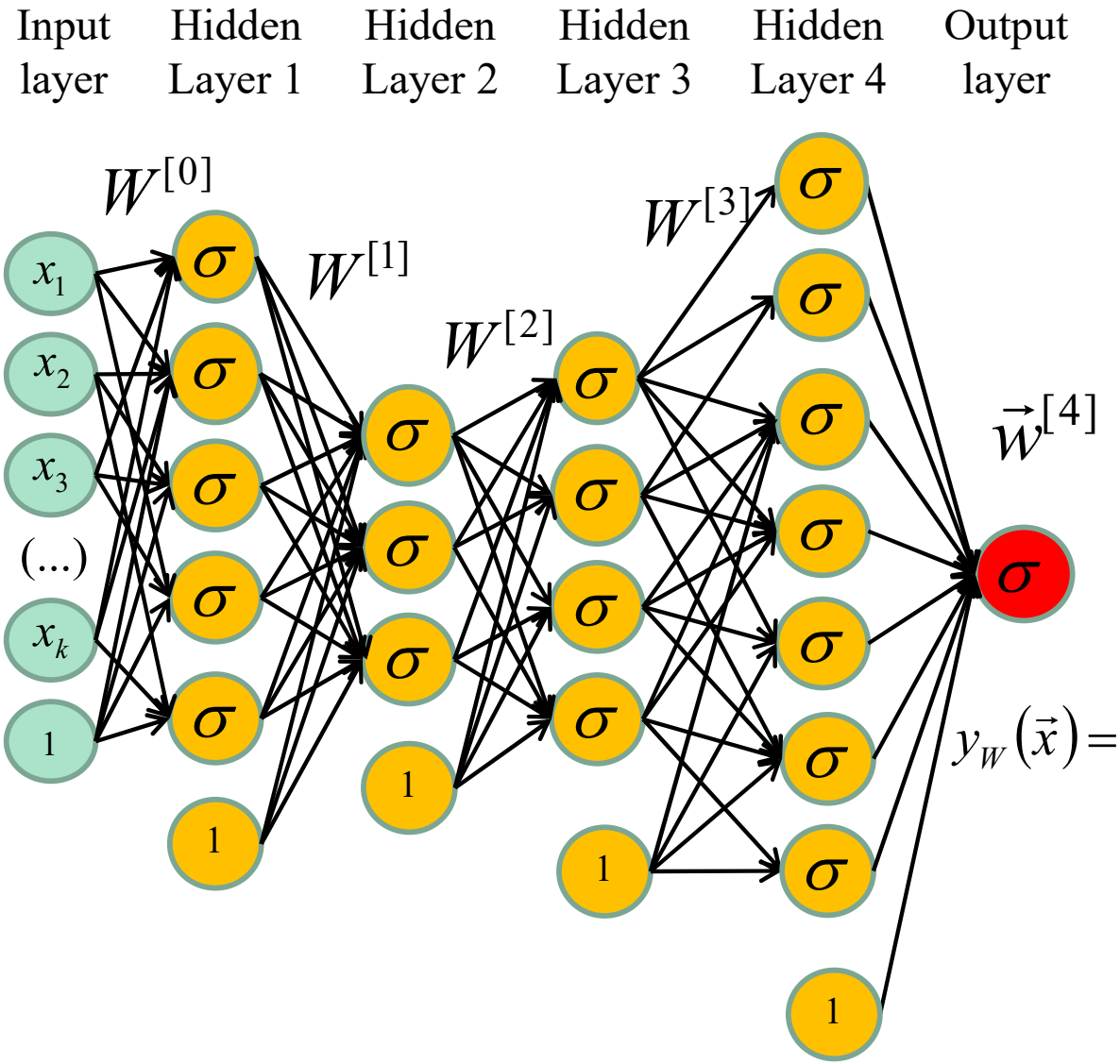
$$W^{[3]} \in R^{7\times 5}$$

$$\vec{w}^{[4]} \in R^{8}$$

$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[4]T}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)\right)$$

This network has 5x(k+1)+6x3 + 4x4+7x5+1x8 **parameters**

kD, 2 Classes, 4 hidden layer network



$$W^{[0]} \in R^{5\times k+1}$$

$$W^{[1]} \in R^{3\times 6}$$

$$W^{[2]} \in R^{4\times 4}$$

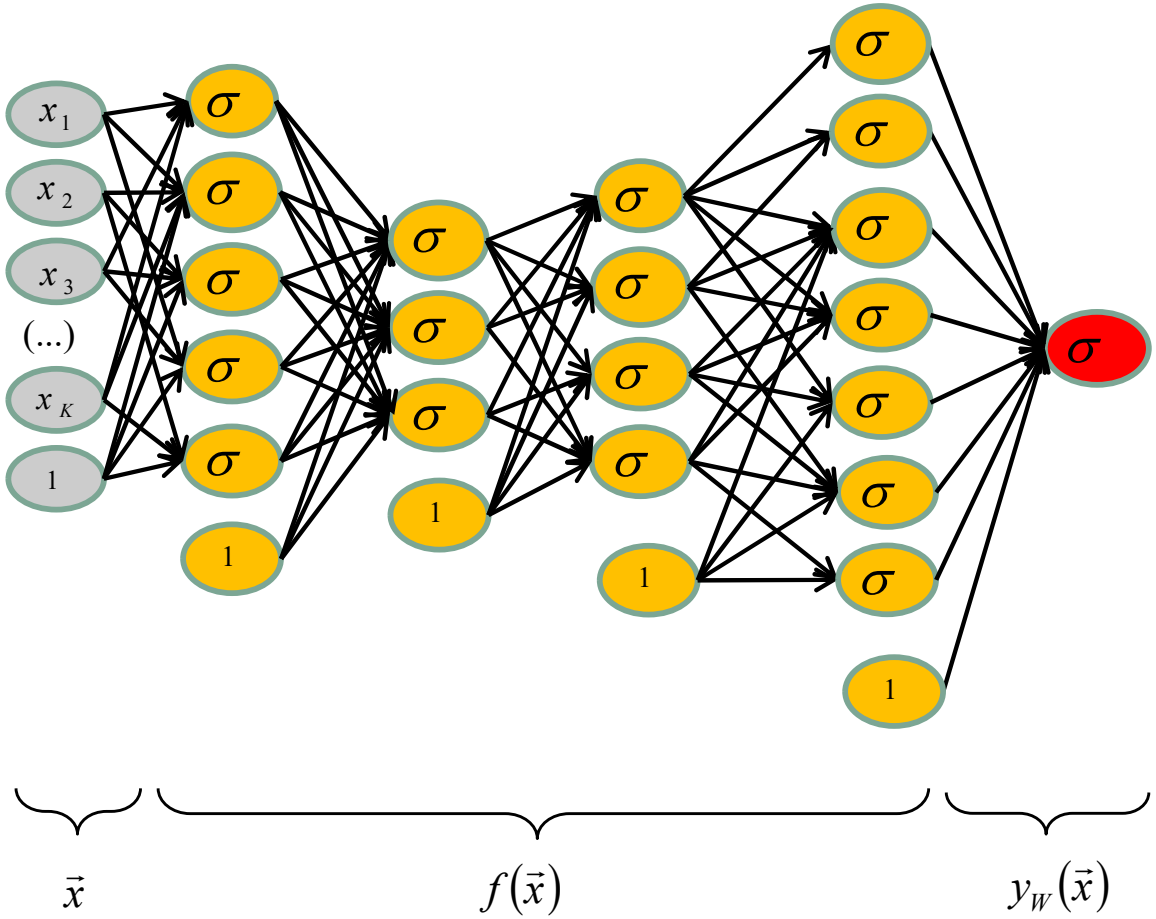
$$W^{[3]} \in R^{7\times 5}$$

$$\vec{w}^{[4]} \in R^{8}$$

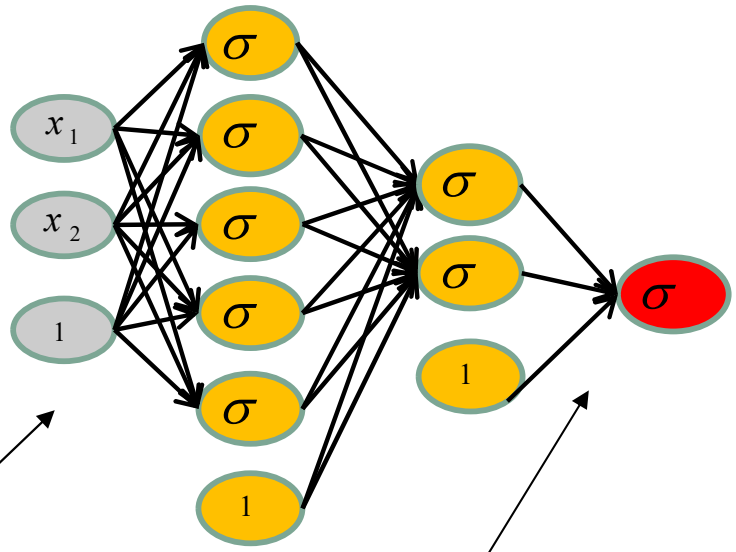
$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[4]T}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)\right)$$

NOTE : More hidden layers = **deeper** network = **more capacity**.

Multilayer Perceptron

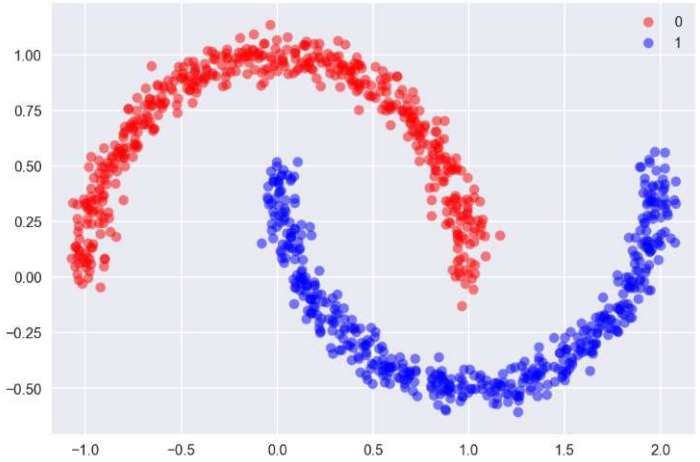


Example

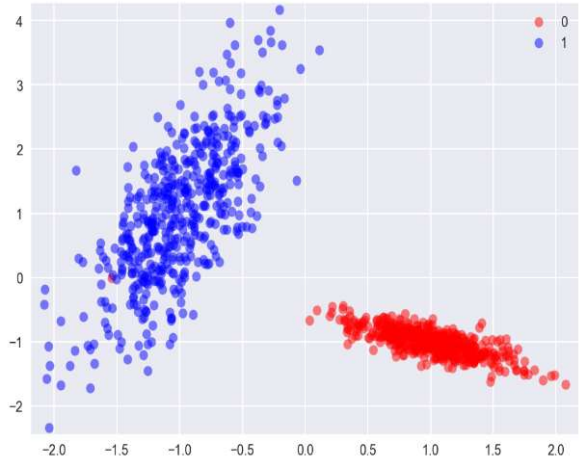


$\vec{x}$

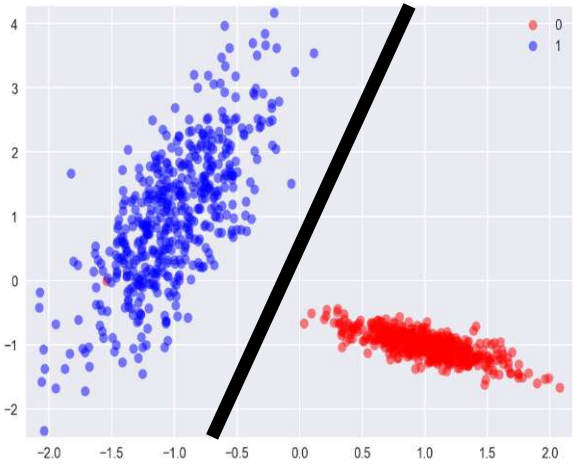
$y_W(\vec{x})$



Input data

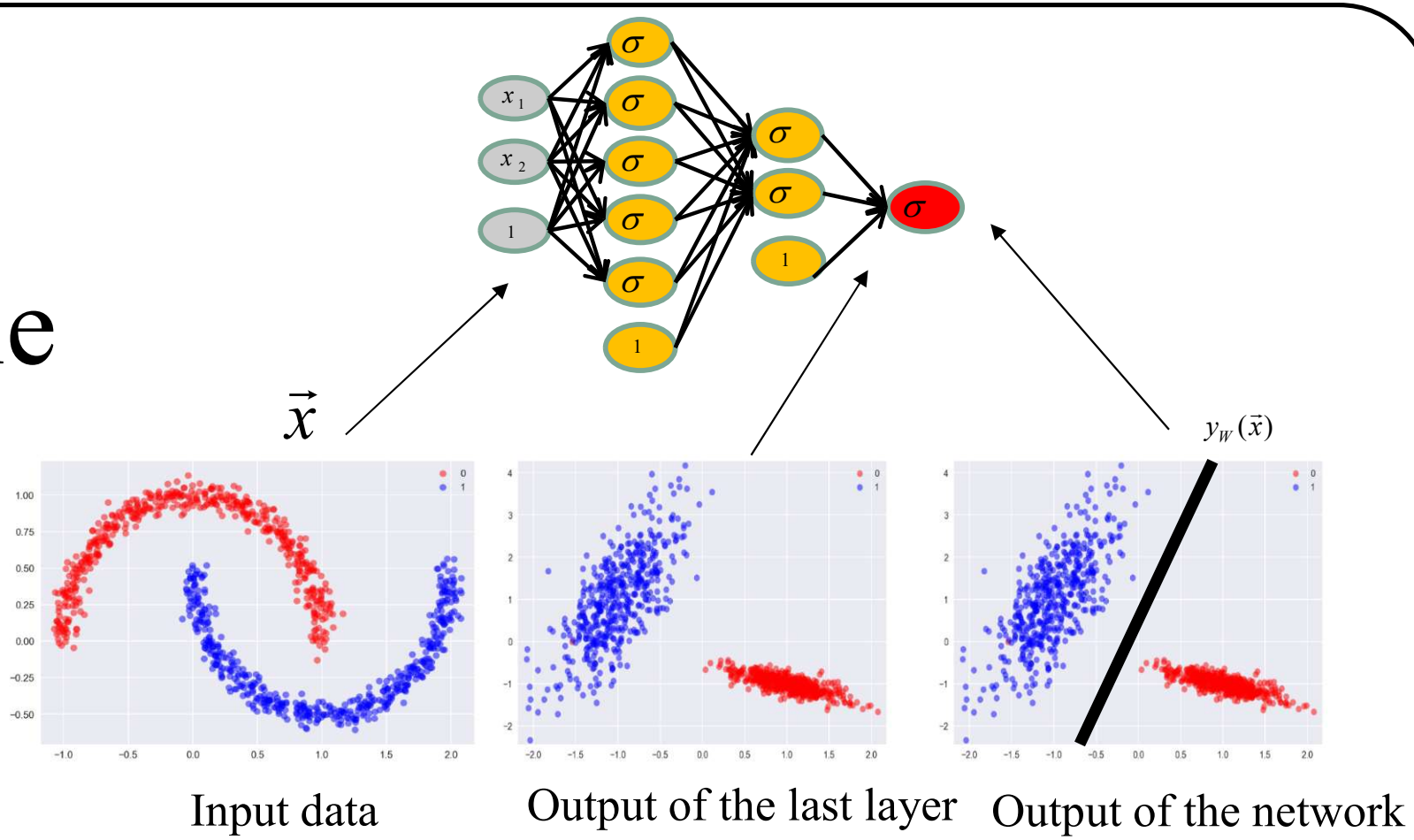


Output of the last layer



Output of the network

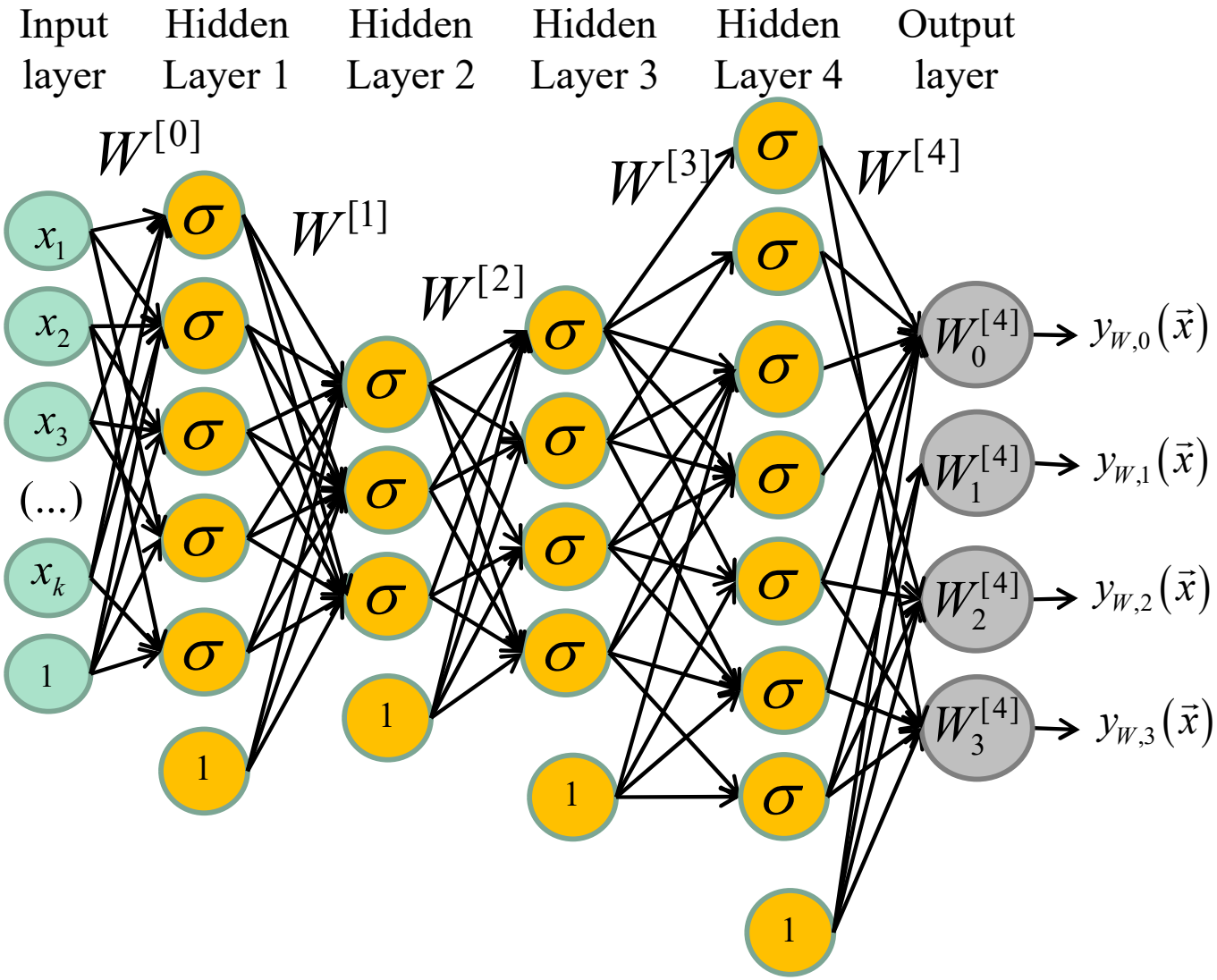
Example



A classification neural network is a **linear classifier** with a bunch of neurons that act as a **basis function**.

A **K-Class** neural network has **K output** neurons.

kD, **4 Classes**, 4 hidden layer network



$$W^{[0]} \in R^{5\times k+1}$$

$$W^{[1]} \in R^{3\times 6}$$

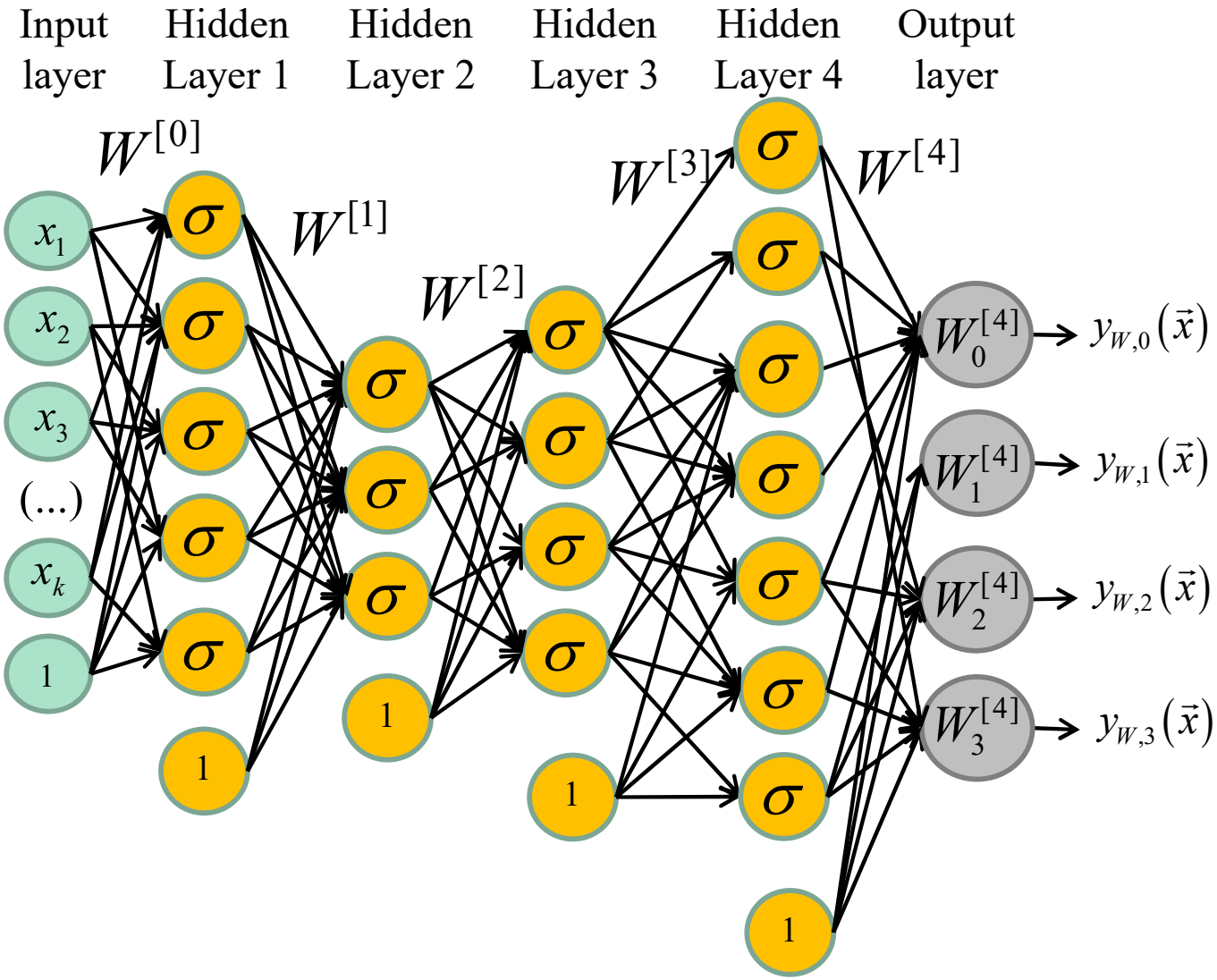
$$W^{[2]} \in R^{4\times 4}$$

$$W^{[3]} \in R^{7\times 5}$$

$$W^{[4]} \in R^{8\times 4}$$

$$y_W(\vec{x}) = W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)$$

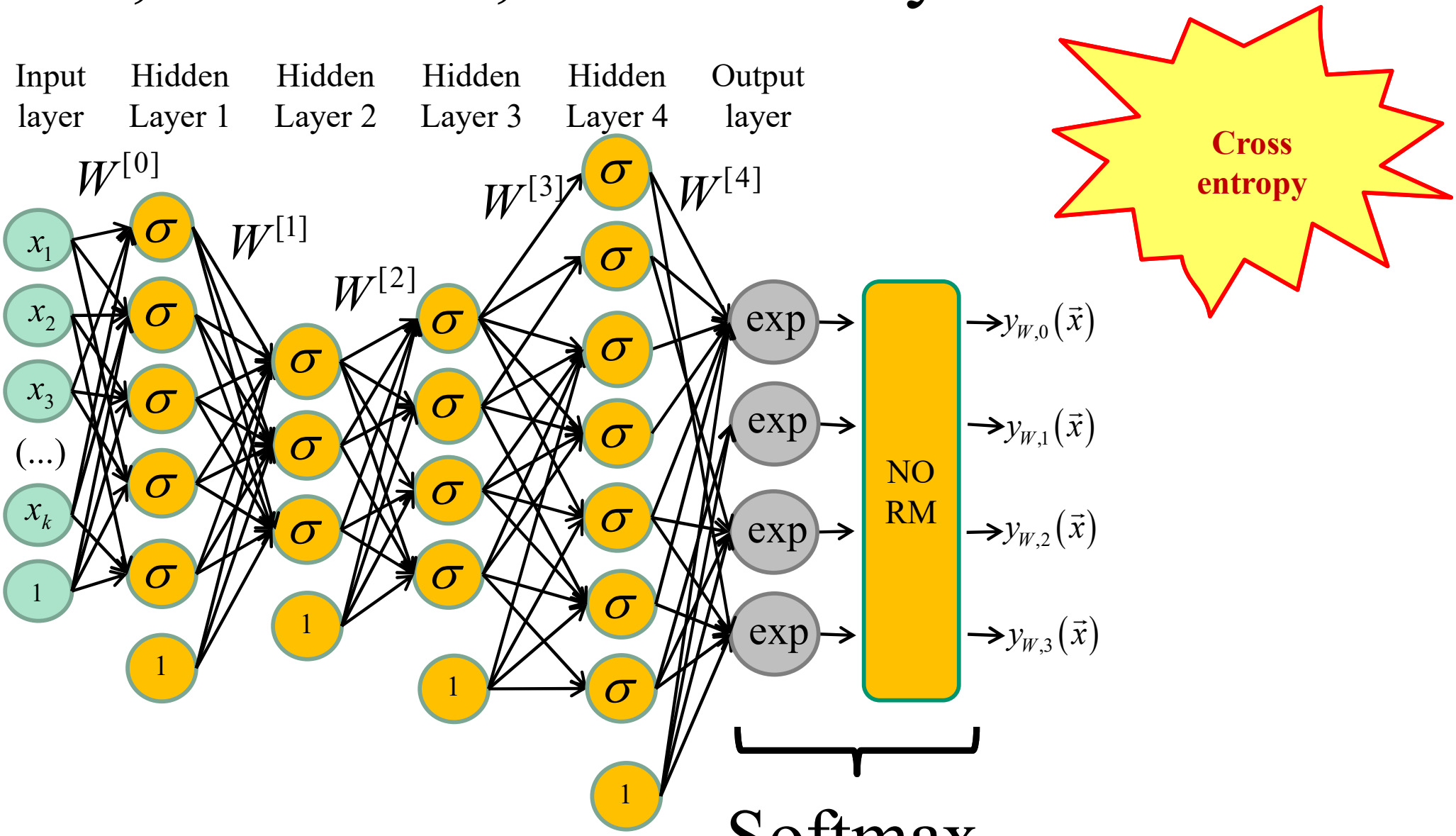
kD, **4 Classes**, 4 hidden layer network



Perceptron loss

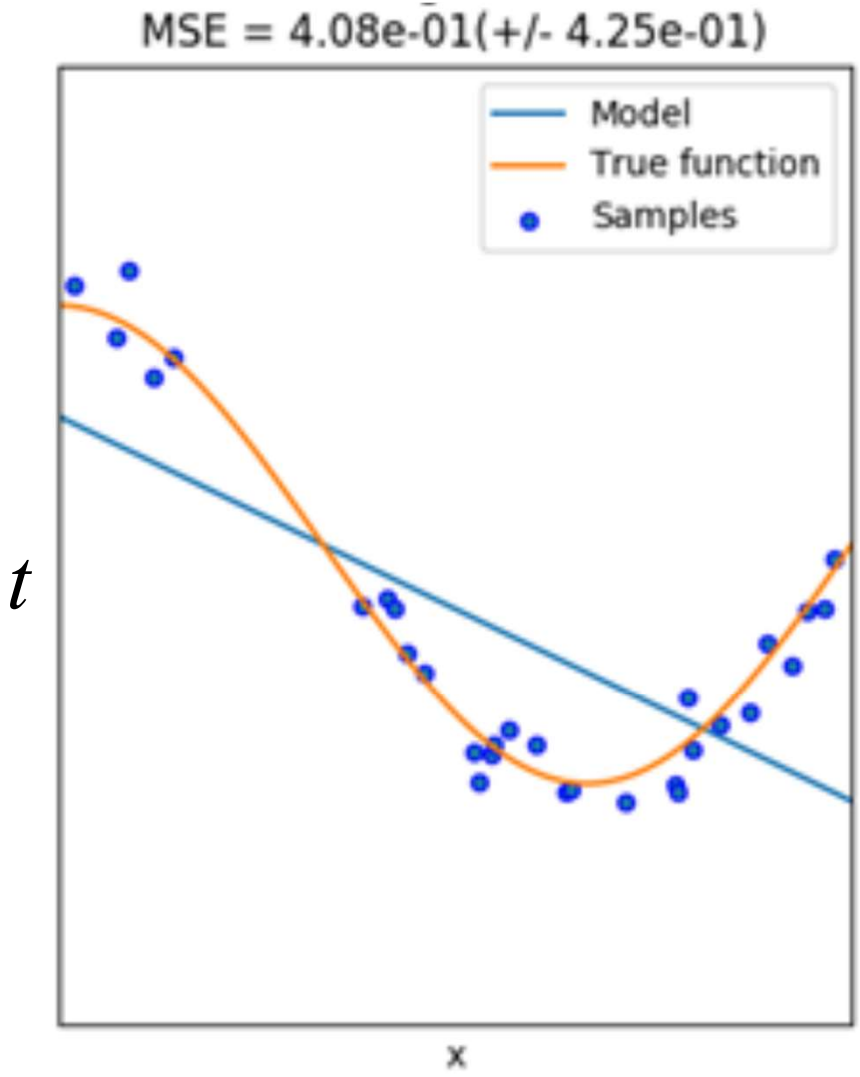
$$y_W(\vec{x}) = W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)$$

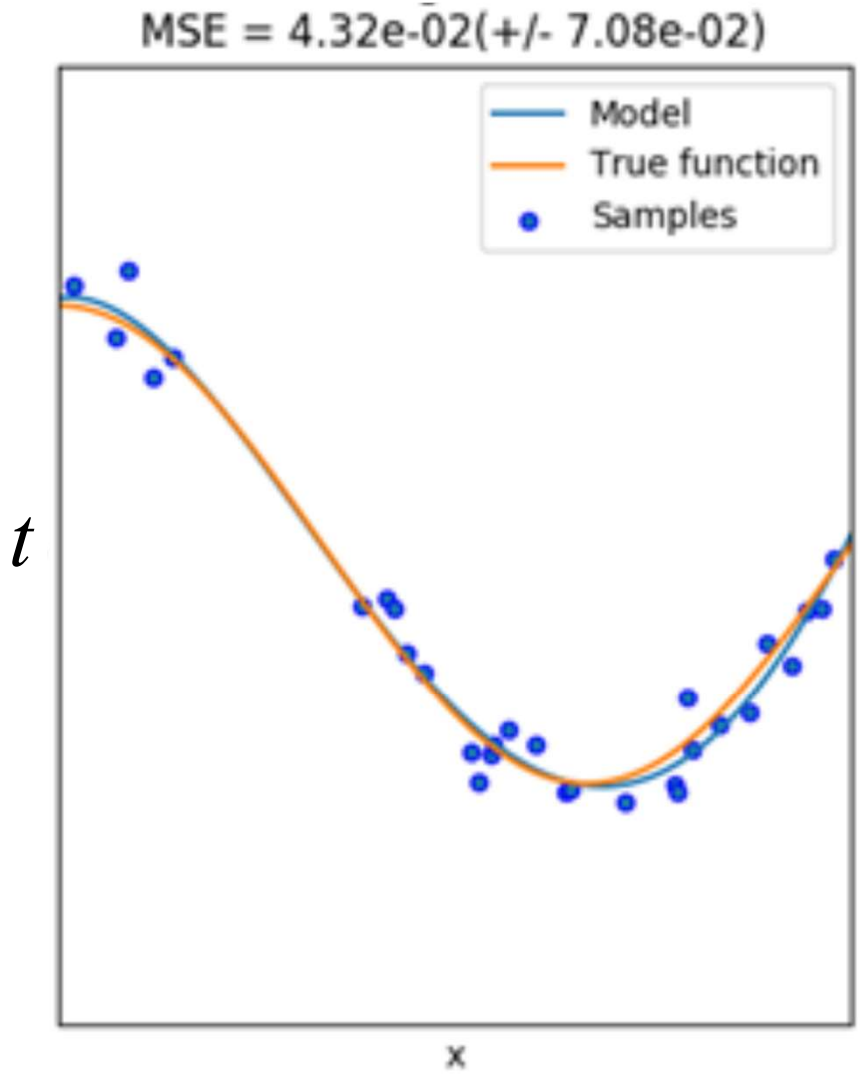
kD, **4 Classes**, 4 hidden layer network



$$y_W(\vec{x}) = \text{softmax}\left(W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)\right)$$

Also works with regression!!





$$y_{\vec{w}}(x) = w_0 + w_1 x$$

Multilayer perceptron

Source : stats.stackexchange.com/questions/305619/artificial-neural-networks-equivalent-to-linear-regression-with-polynomial-featu

Over- and under-fitting

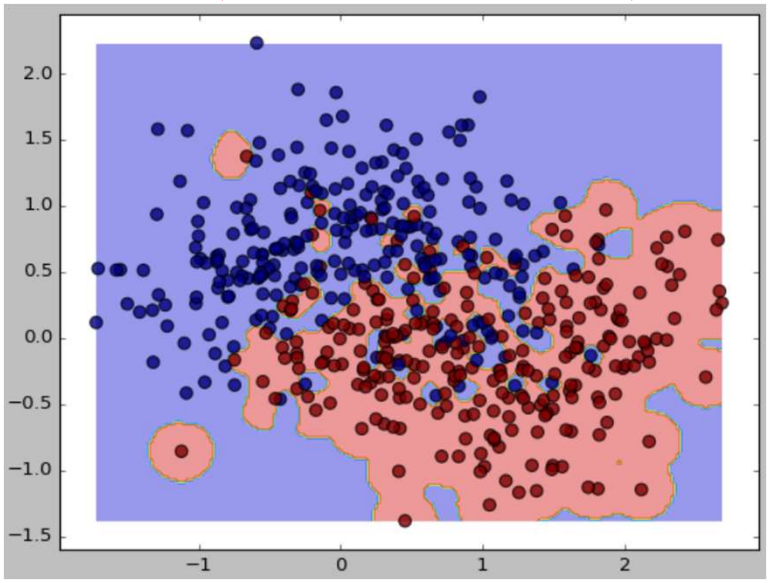
Increase the number of neurons

Increase the number of layers

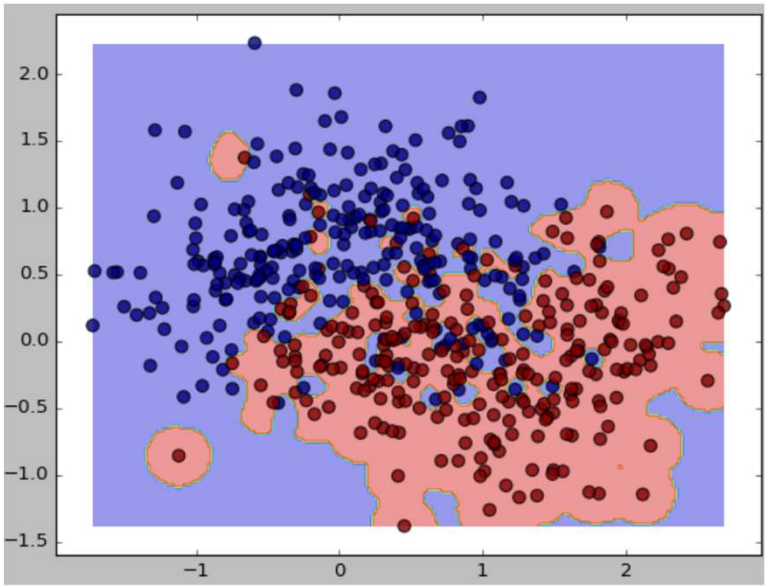
Increase the capacity of the network

Increasing the capacity of the network can lead to **over-fitting**

Overfitting (Classification)

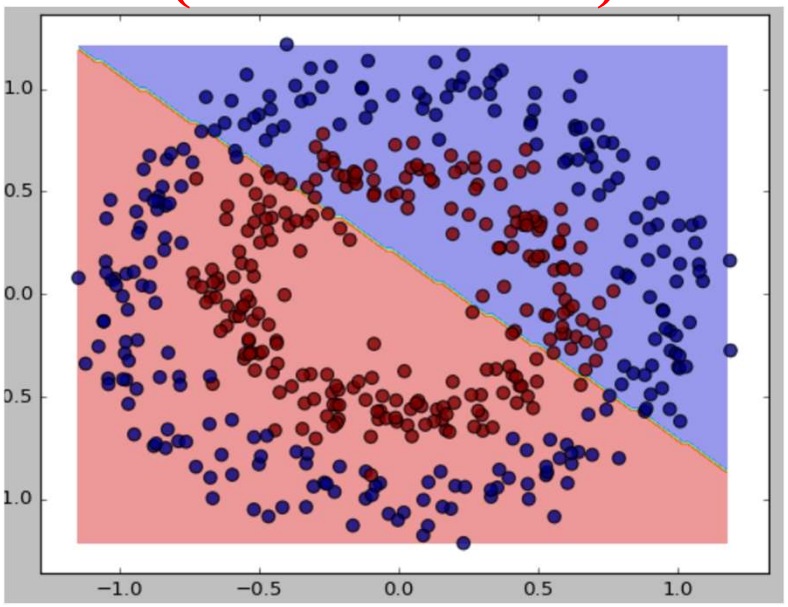


Training accuracy = 99.6%

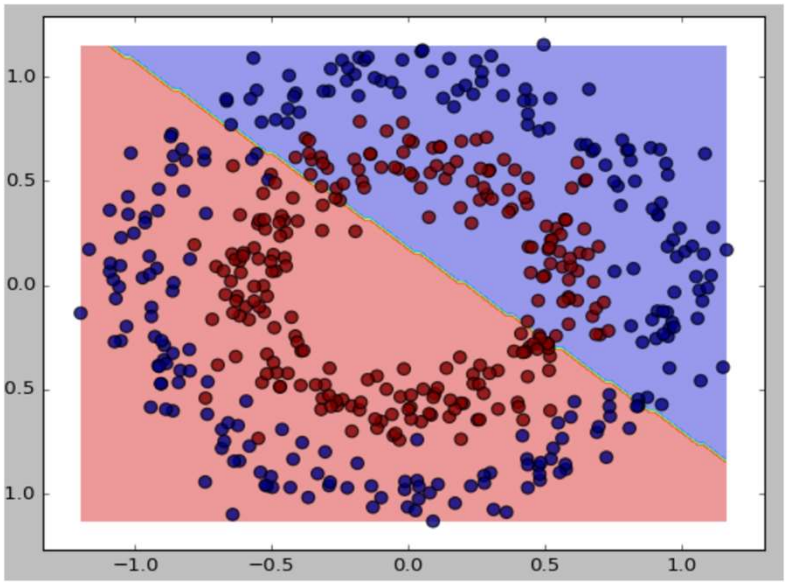


Testing accuracy = 78%

Underfitting (Classification)

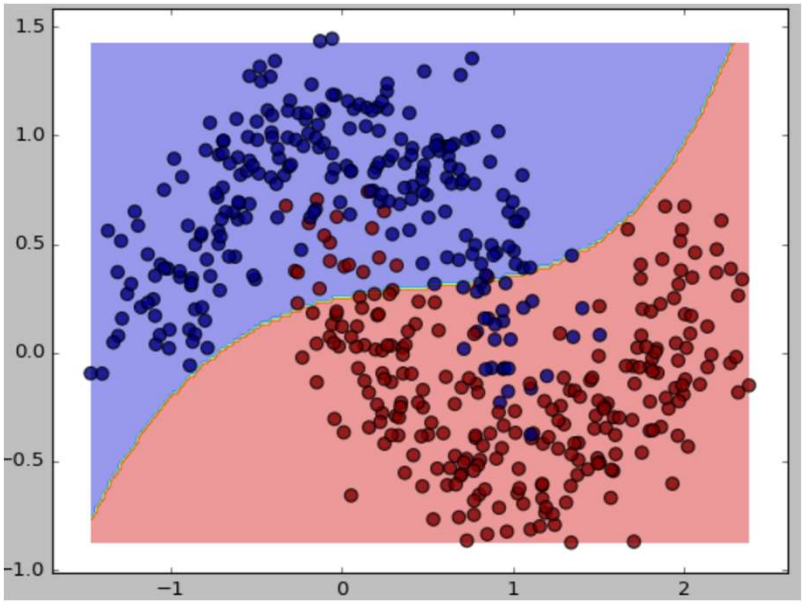


Training accuracy =52.2%

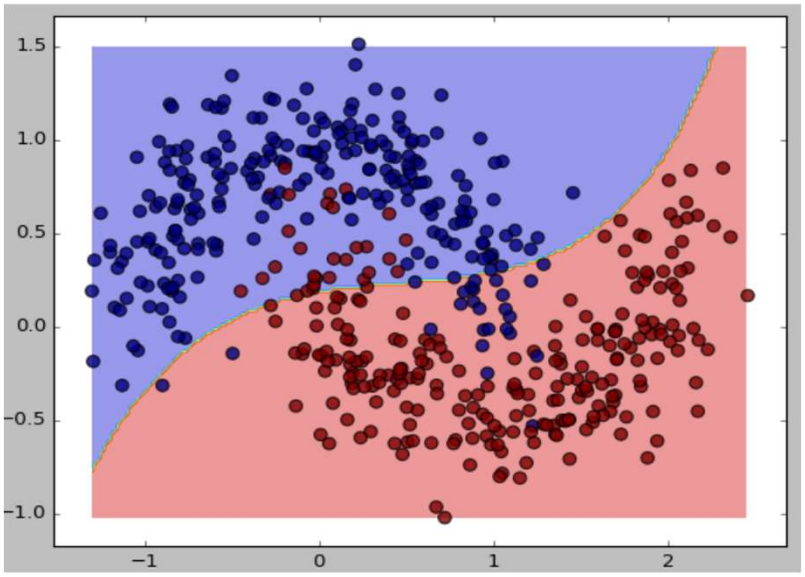


Testing accuracy = 51.2%

Could be better…

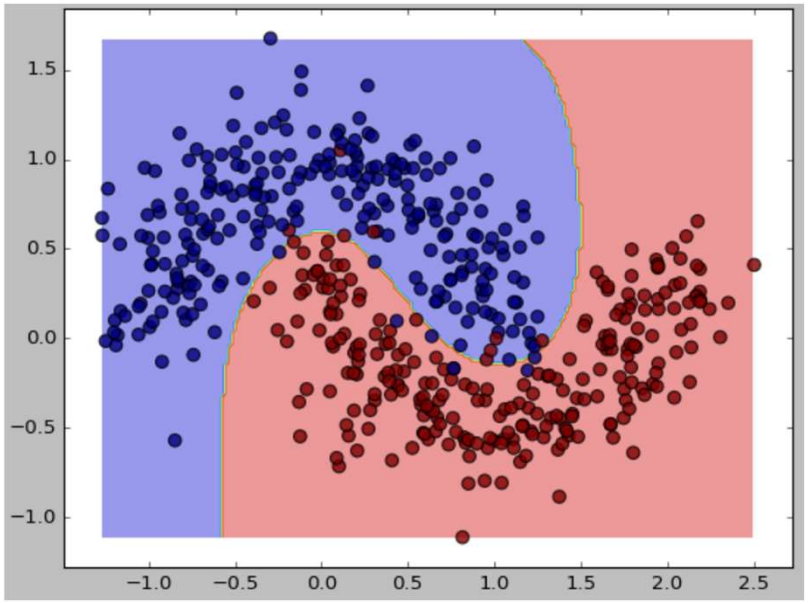


Training accuracy =82%

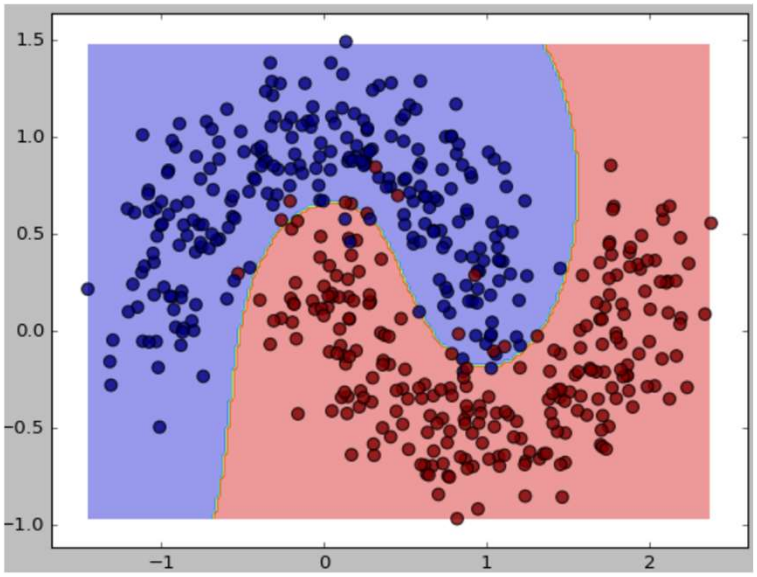


Testing accuracy = 80%

Wonderful !!!

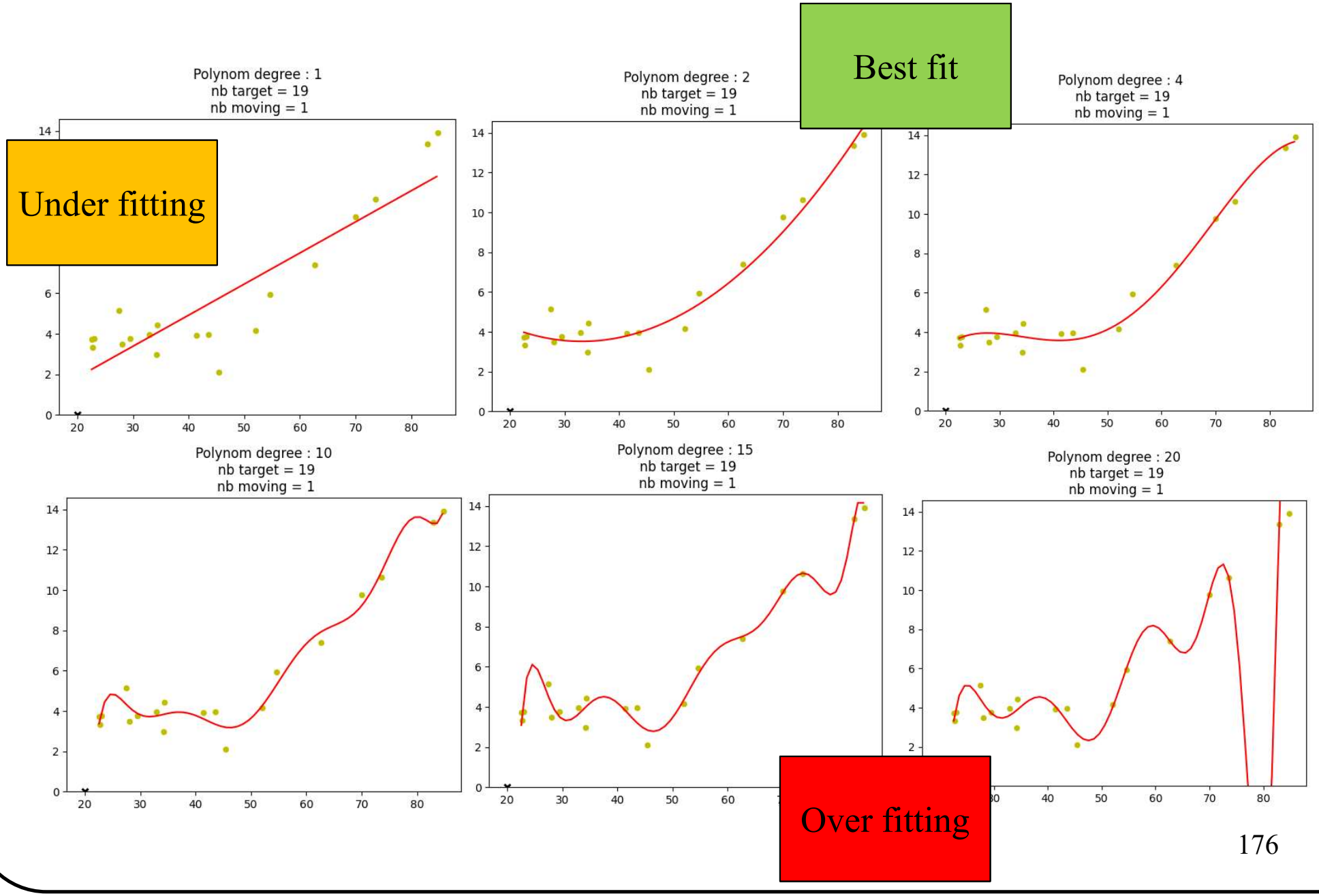


Training accuracy =97.8%

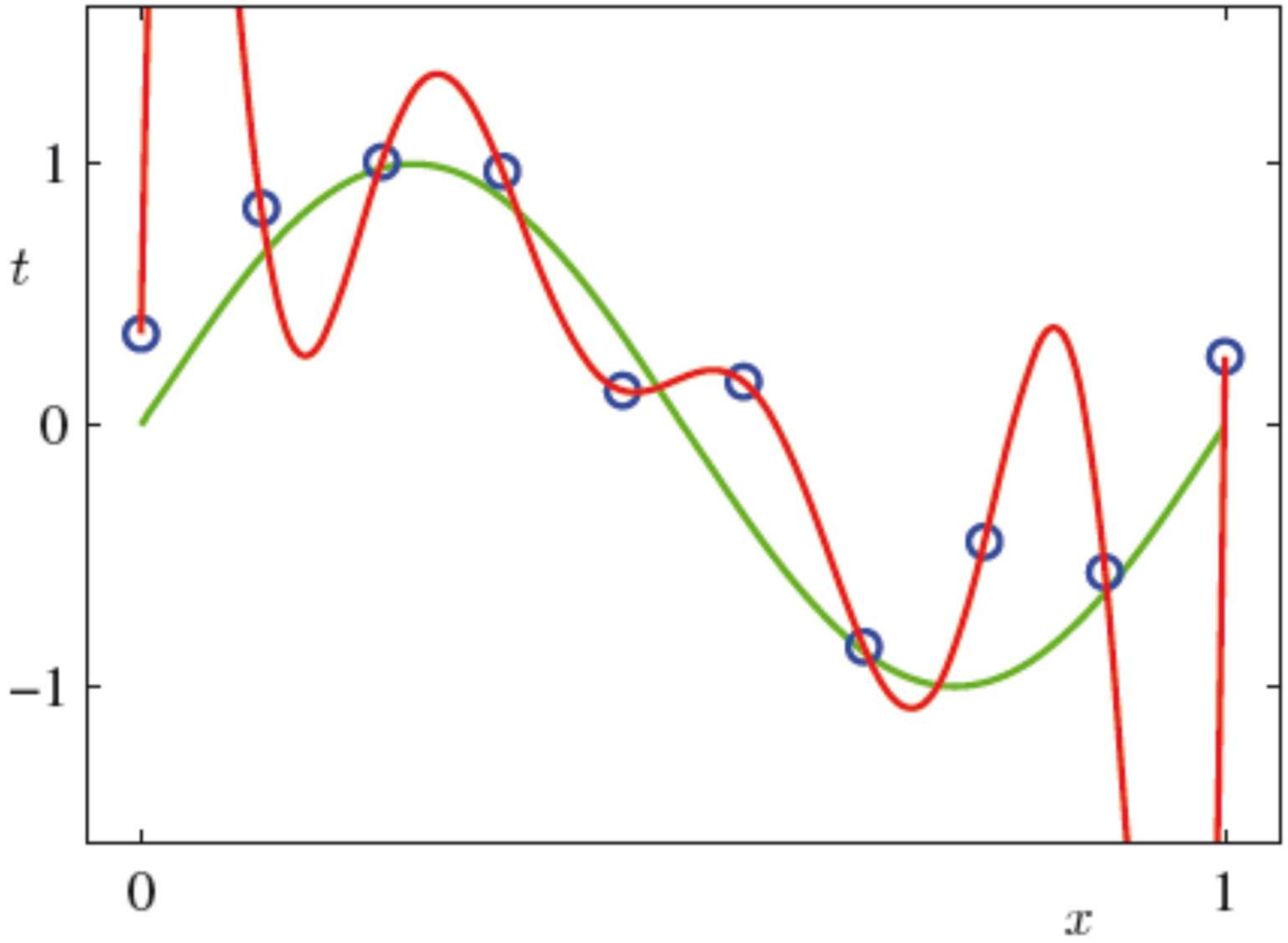


Testing accuracy = 96.2%

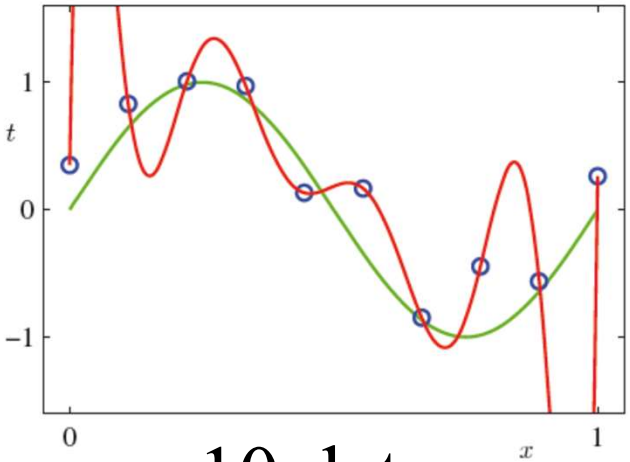
Regression


Polynom degree : 1
nb target = 19
nb moving = 1
Polynom degree : 2
nb target = 19
nb moving = 1
Polynom degree : 4
nb target = 19
nb moving = 1
Polynom degree : 10
nb target = 19
nb moving = 1
Polynom degree : 15
nb target = 19
nb moving = 1
Polynom degree : 20
nb target = 19
nb moving = 1
Under fitting
Best fit
Over fitting

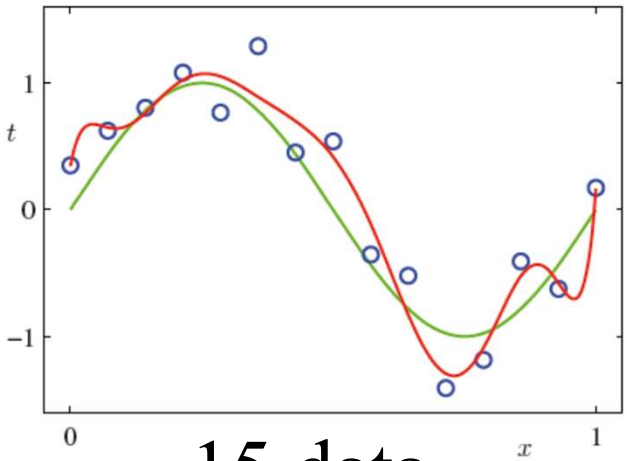
Overfitting



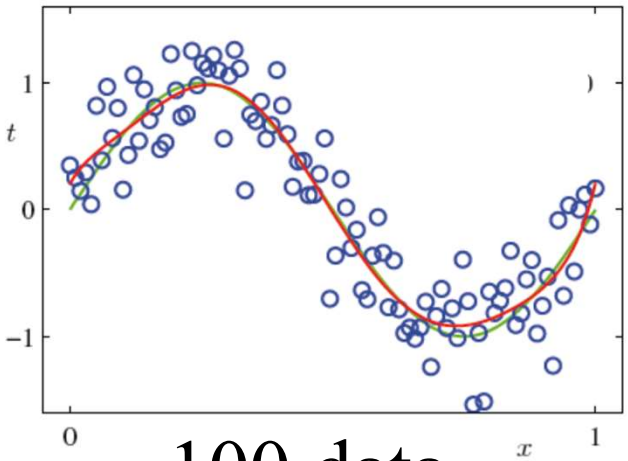
The **more data**, the better a **high capacity model generalizes**



10 data



15 data



100 data

How do we prevent our model from under- and overfitting?

Regularization

Parameter values $\vec{w}$ for different d **without** regularization

	$d = 0$	$d = 1$	$d = 3$	$d = 9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

hyperparameter

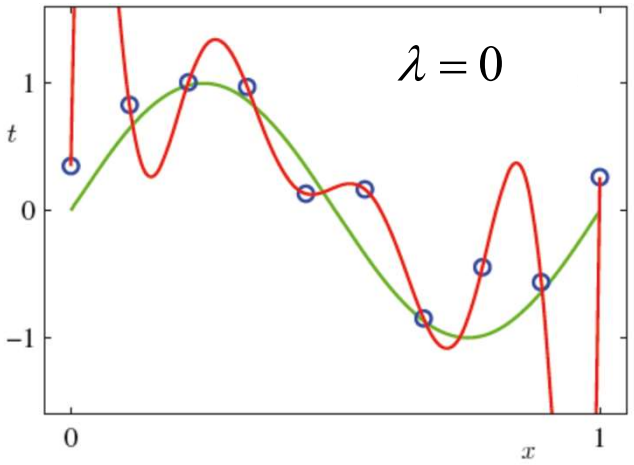
$$\arg\min_{W} = L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) + \lambda R\left(W\right)$$

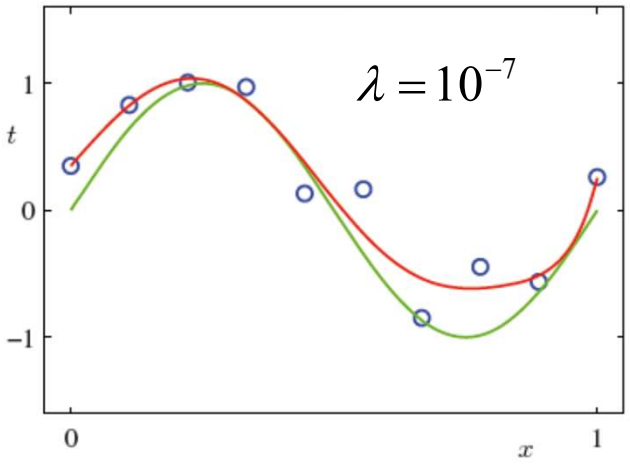
Loss function

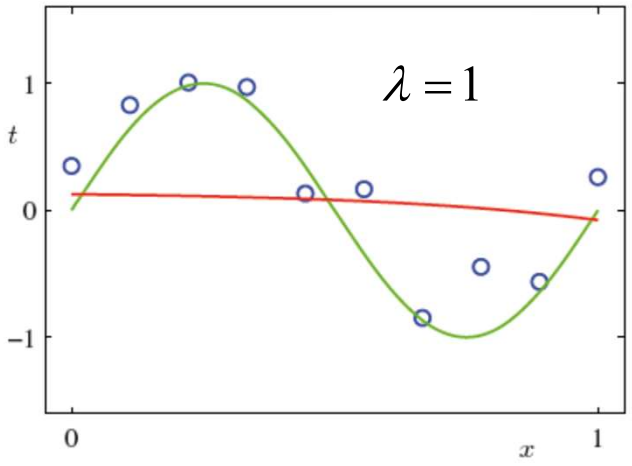
Regularization

$$R(W) = \|\mathrm{W}\|_1 \text{ or } \|\mathrm{W}\|_2$$

Strong regularization= less capacity







In short we have seen

- Supervised vs unsupervised learning
- Regression vs Classification
- Linear vs non-linear models
- Perceptron, MLP
- Loss function and gradient descent.
- Over vs Underfitting
- (Cross-validation)
- (Metrics)