Montreal, July 8-12

DLMI2024

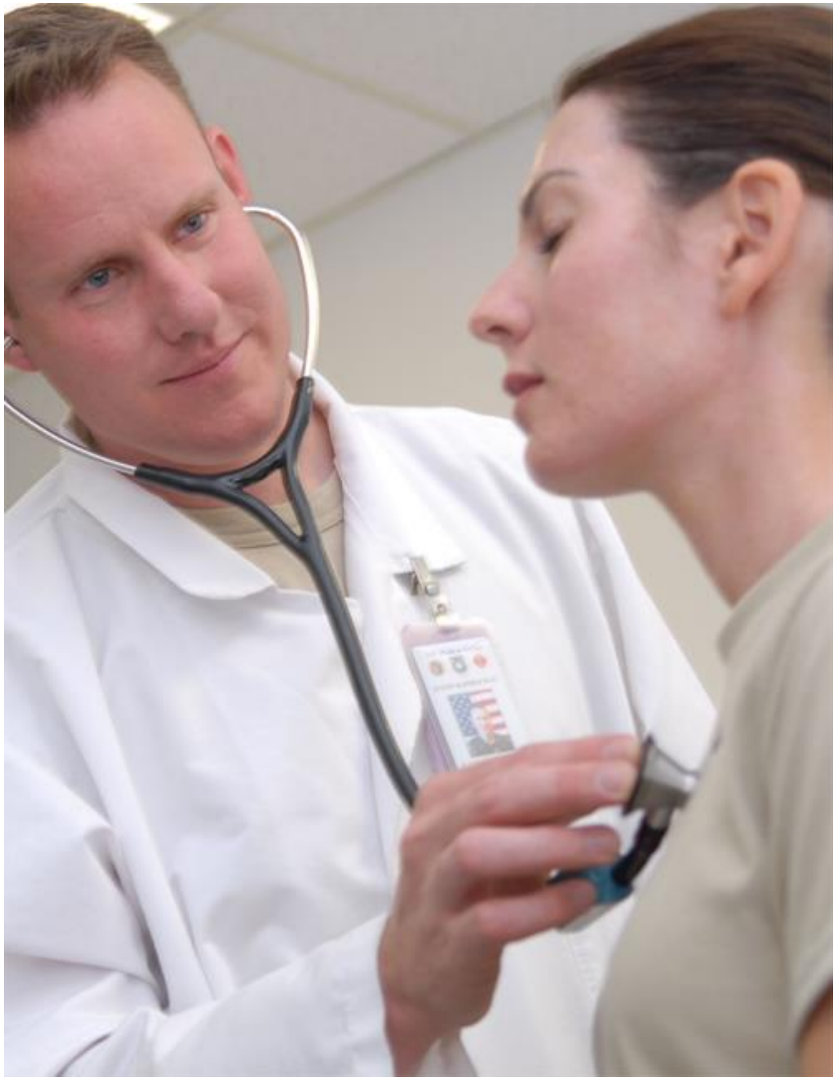
Basics of deep learning part 2

By

Pierre-Marc Jodoin

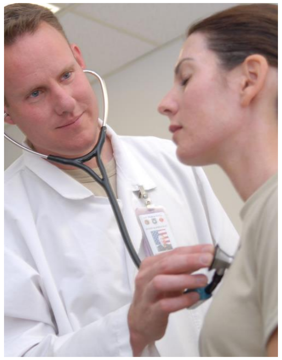
UNIVERSITÉ DE SHERBROOKE

Lets start with a simple example



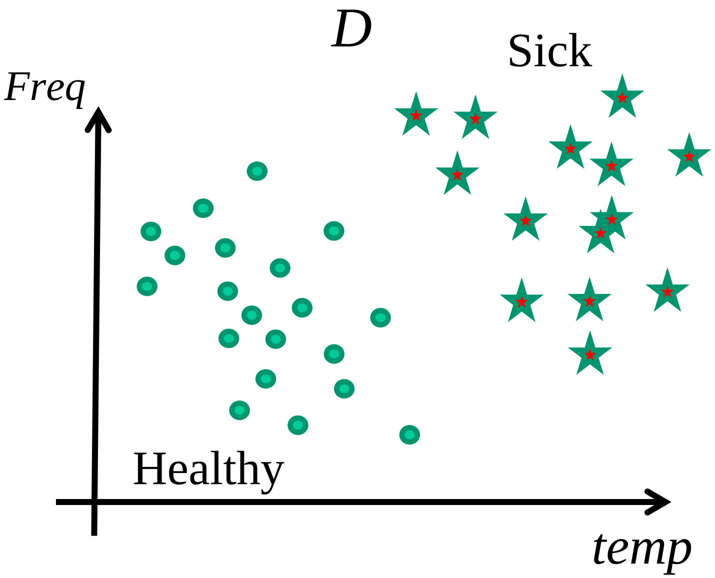
From Wikimedia Commons
the free media repository

Lets start with a simple example



D

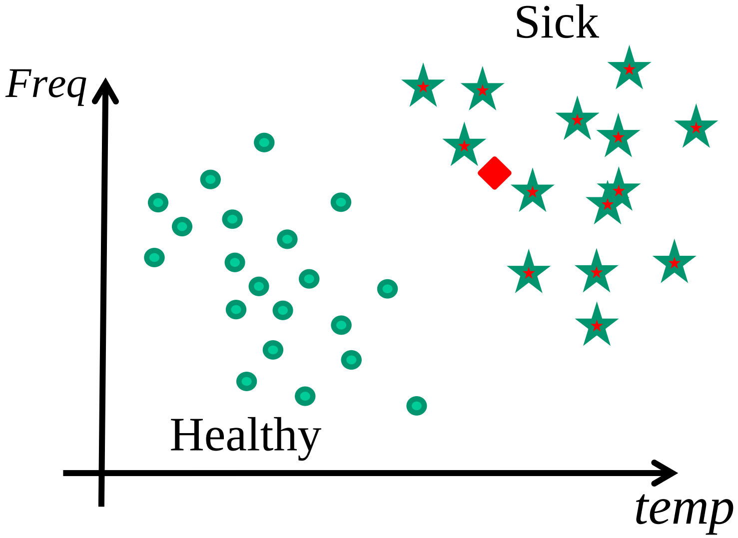
	(temp, freq)	diagnostic
Patient 1	(37.5, 72)	Healthy
Patient 2	(39.1, 103)	Sick
Patient 3	(38.3, 100)	Sick
	(…)	…
Patient N	(36.7, 88)	Healthy
	$\vec{x}$	t



Lets start with a simple example

A new patient comes to the hospital
How can we determine if he is sick or not?

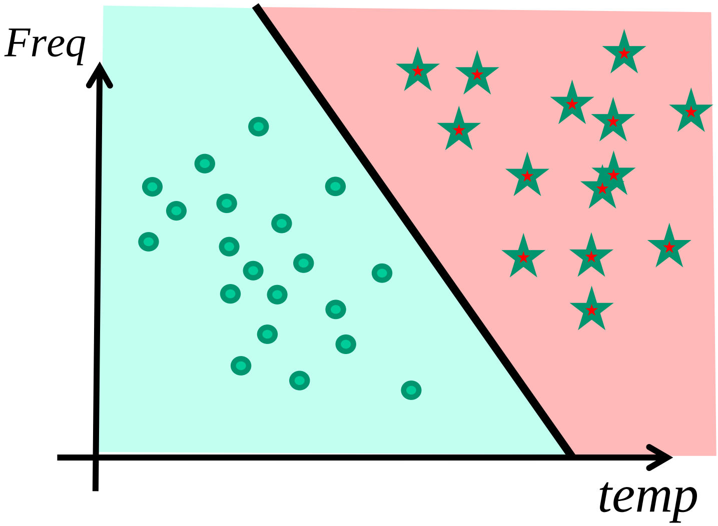
From Wikimedia Commons
the free media repository



Solution


From Wikimedia Commons
the free media repository

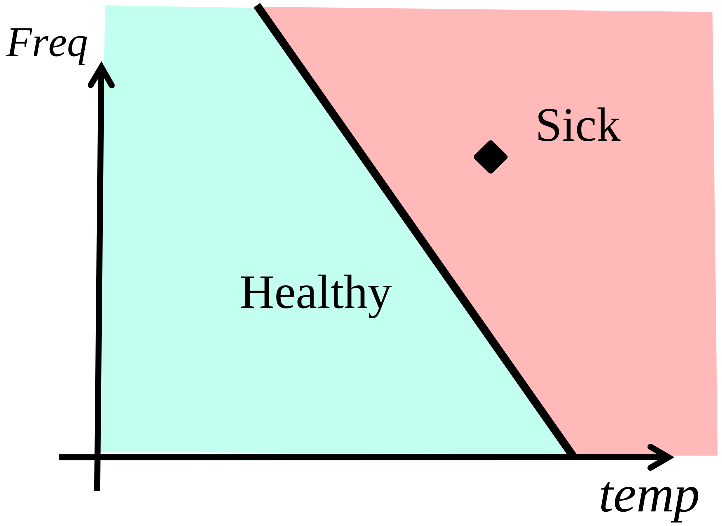
Divide the feature space in 2 regions : sick and healthy



Solution

From Wikimedia Commons
the free media repository

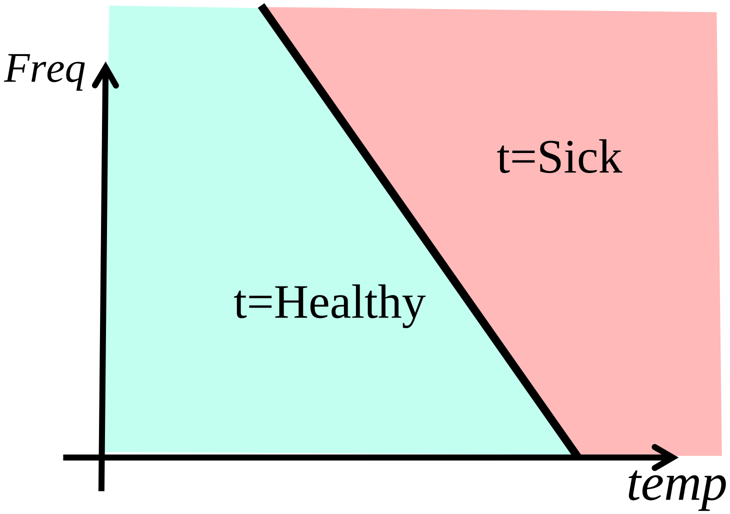
Divide the feature space in 2 regions : sick and healthy



More formally

From Wikimedia Commons
the free media repository

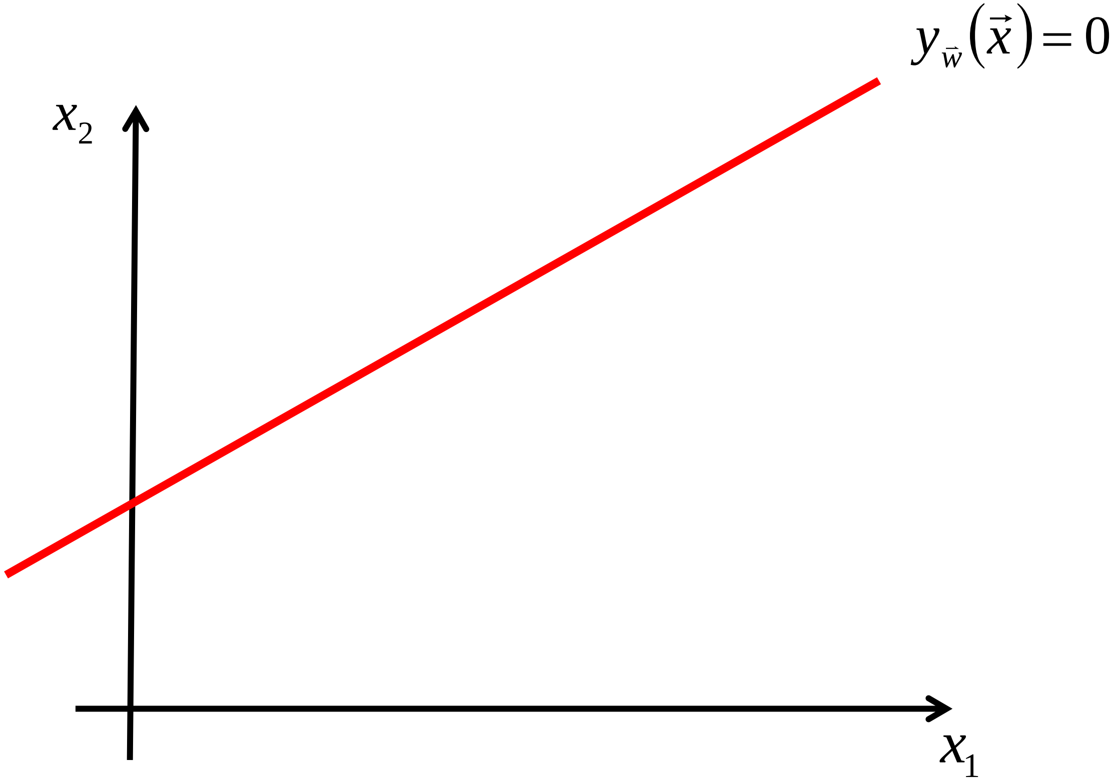
$$y(\vec{x}) = \begin{cases} \text{Healthy if } \vec{x} \text{ is in the green region} \\ \text{Sick otherwise} \end{cases}$$



How to split the feature space?

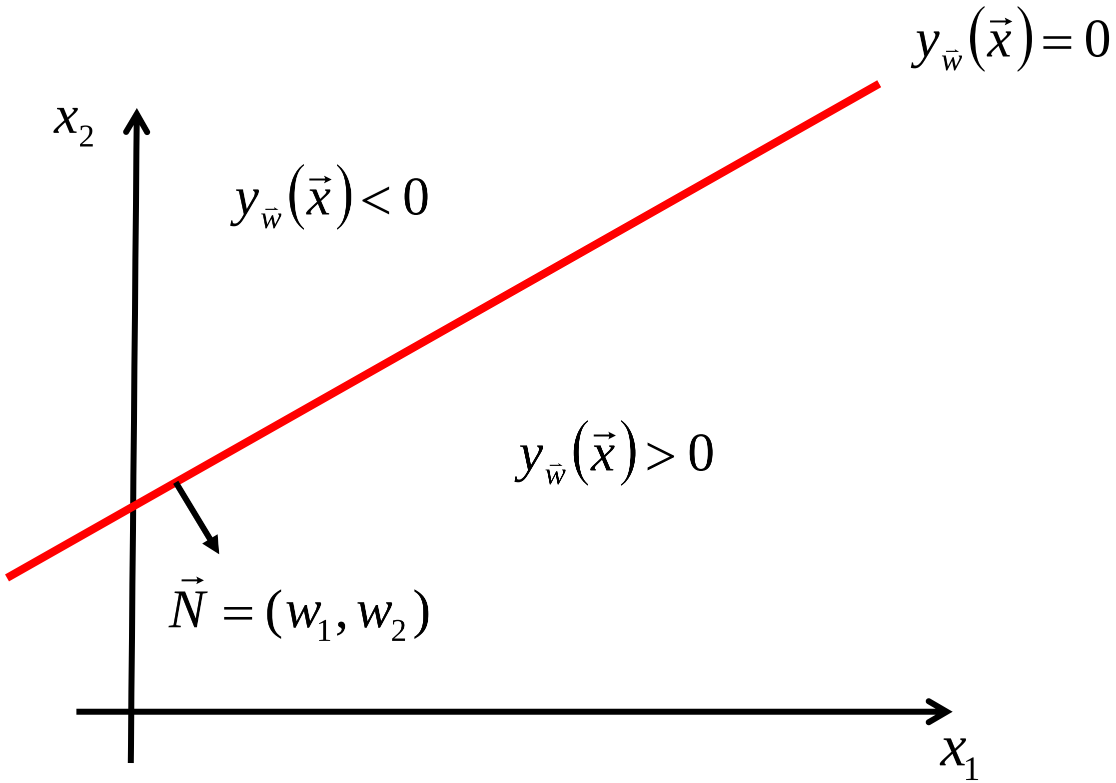
Linear function

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



Classification function

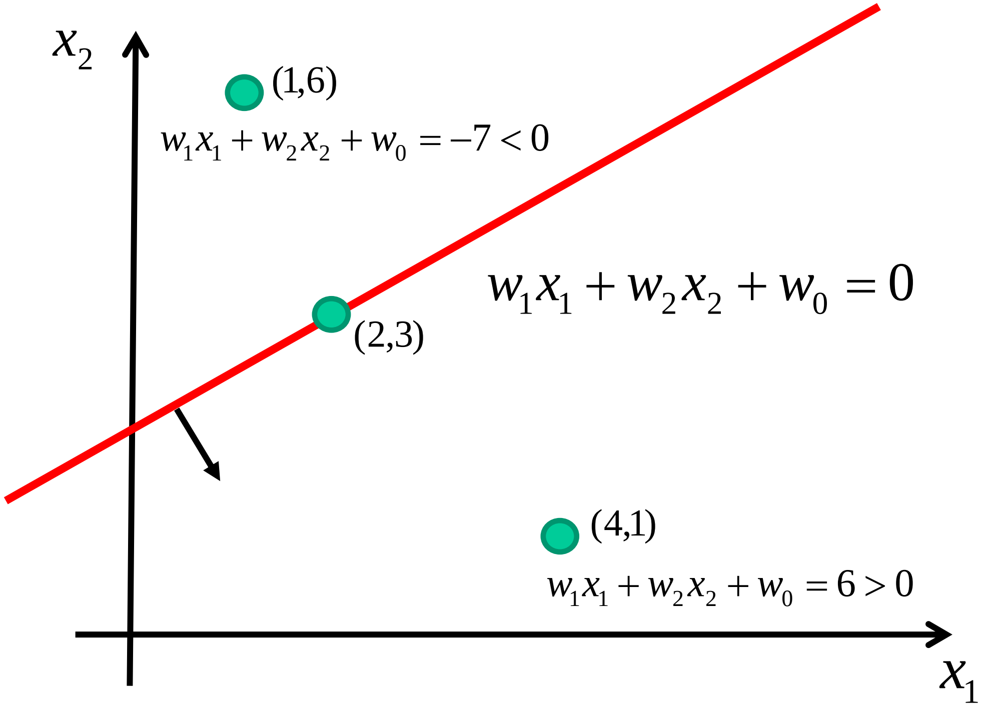
$$y_{\vec{w}}\left(\vec{x}\right) = w_1 x_1 + w_2 x_2 + w_0$$



Classification function

$$w_1 = 1.0$$
$$w_2 = -2.0$$
$$w_0 = 4.0$$

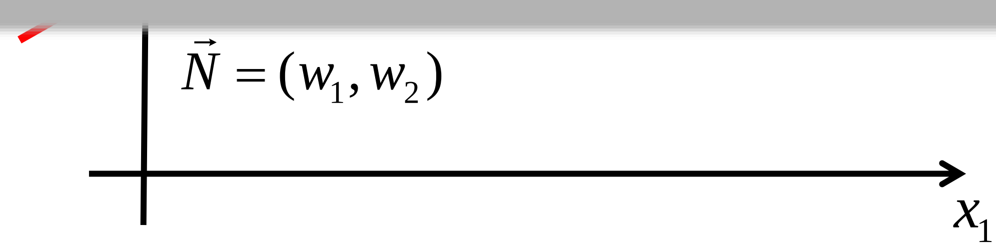
$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$



Classification function

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

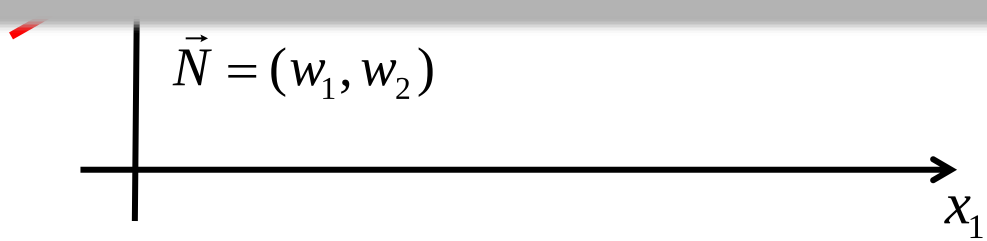
DOT product



Classification function

$$
\begin{aligned}
y_{\vec{w}}(\vec{x}) &= w_1 x_1 + w_2 x_2 + w_0 \\
&= (w_0, w_1, w_2) \cdot (1, x_1, x_2) \\
&= \vec{w}^T \vec{x}'
\end{aligned}
$$

DOT product



linear classification = dot product with bias included

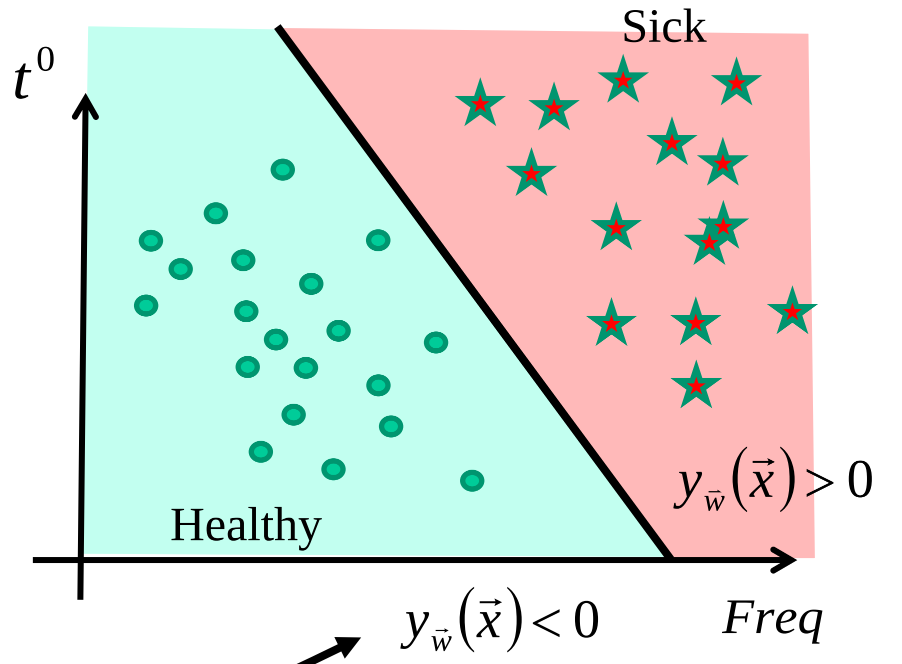
$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x}$$

Learning

With the **training dataset** D

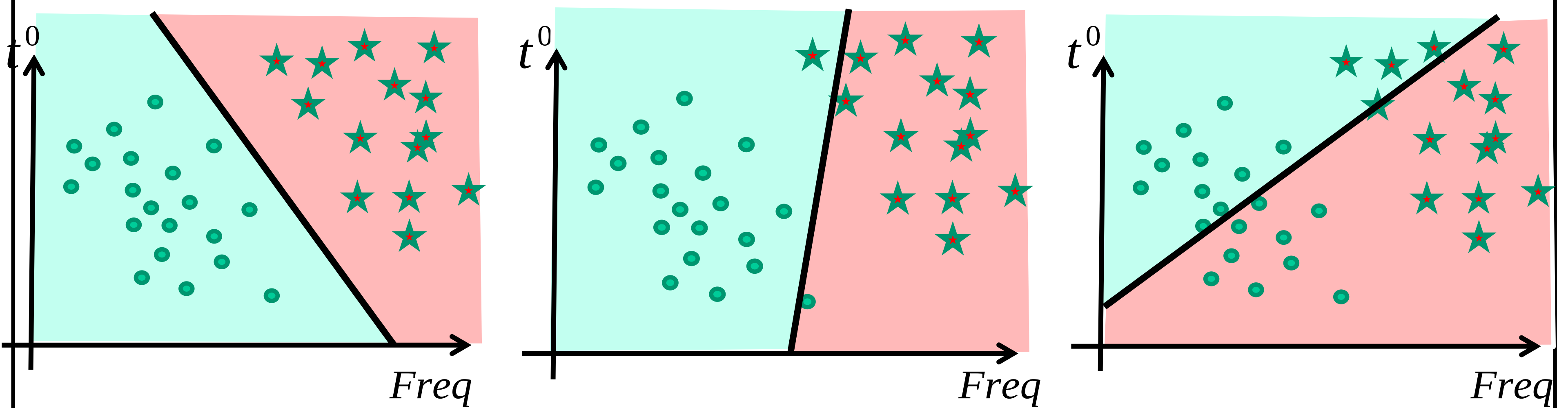
the GOAL is to

find the parameters (w_0, w_1, w_2) that would best separate the two classes.



How do we know if a model is good?

Loss function



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

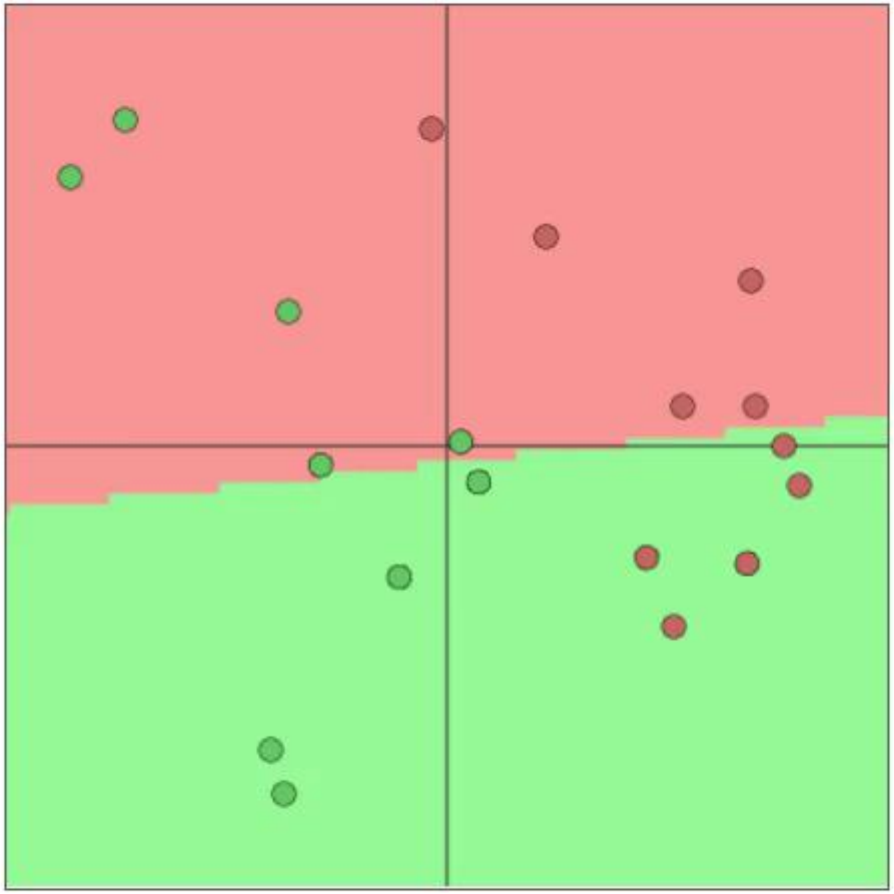
Good! **Medium** **BAD!**

Training a machine

Finding the right parameters w0,w1,w2 such that

PATIENTS are WELL CLASSIFIED

SMALL LOSS



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

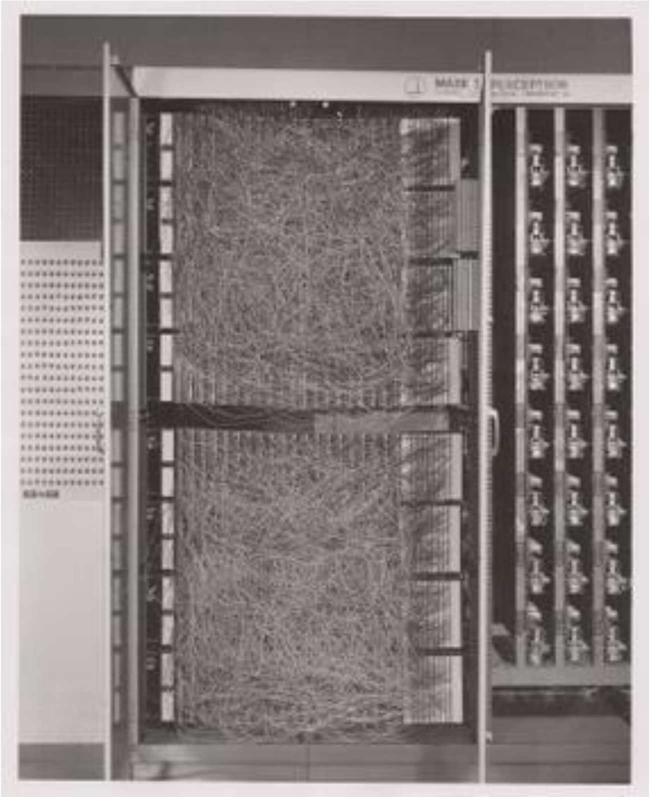
4. Training : find (w_0, w_1, w_2) that minimize $L(y_{\vec{w}}(\vec{x}), D)$

Before deep neural nets were … **linear models**

In this session…

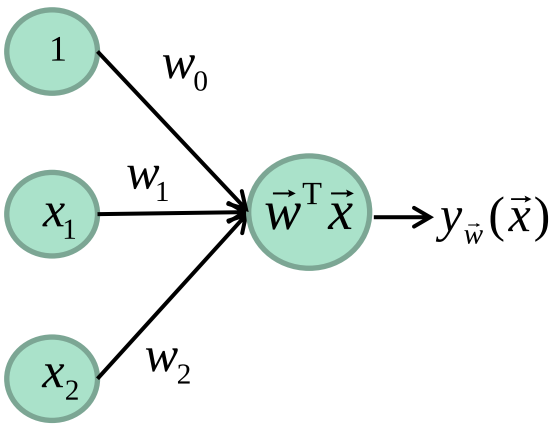
Perceptron
Logistic regression
Multi-layer perceptron

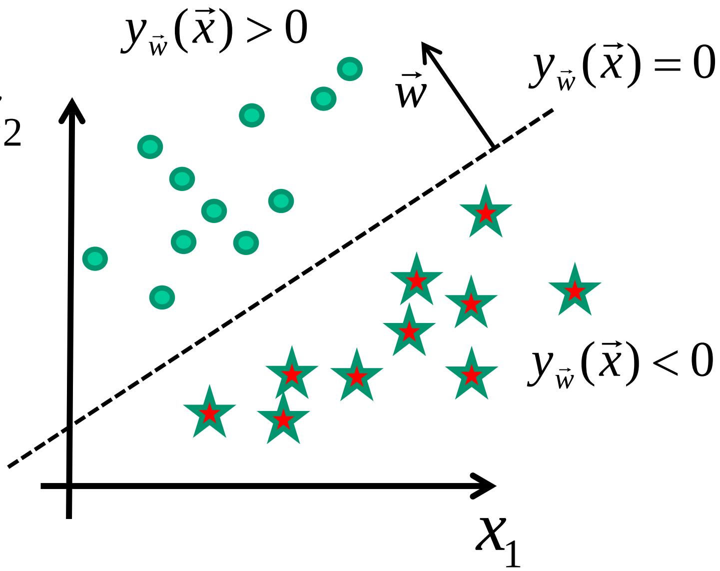
Perceptron



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

Perceptron
(2D and 2 classes)

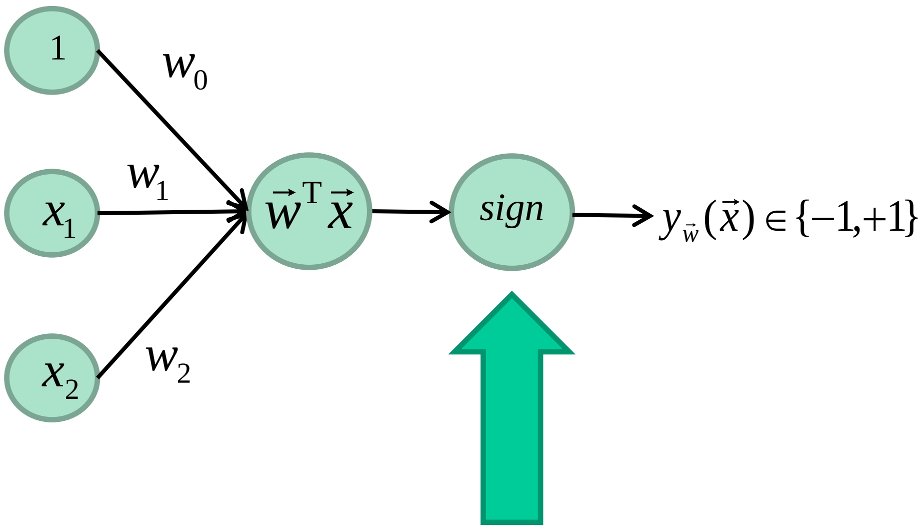




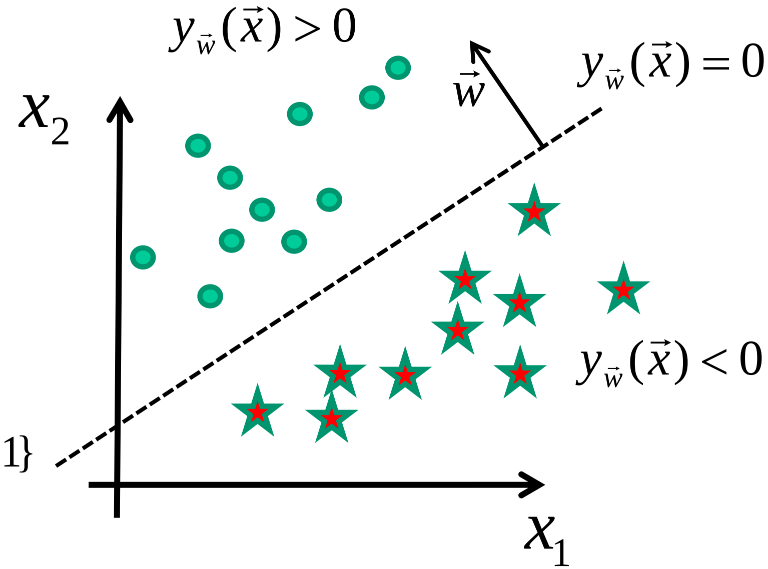
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_1$$

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

Perceptron
(2D and 2 classes)

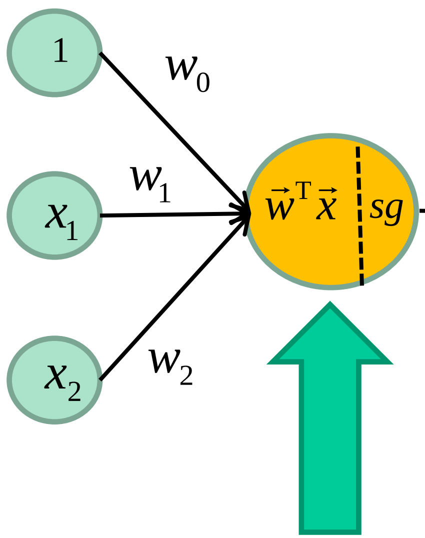


Activation function

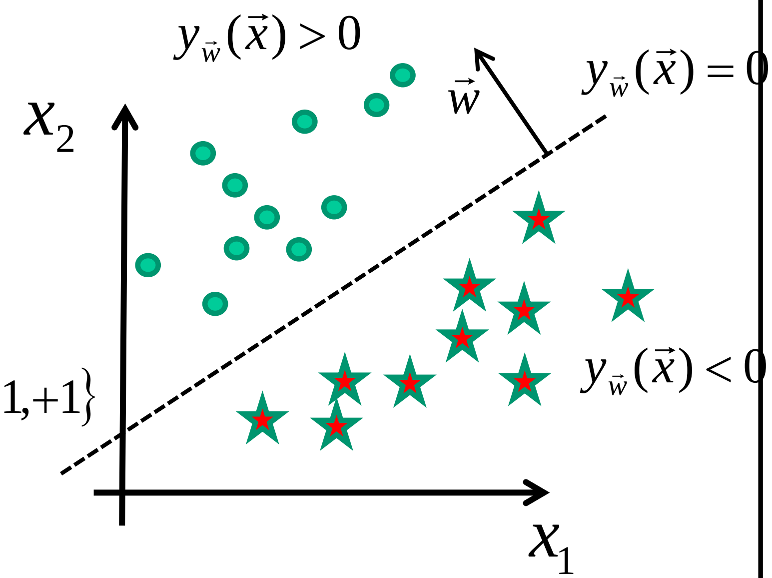


$$y_{\vec{w}}(\vec{x}) = sign\left(\vec{w}^{\mathrm{T}}\vec{x}\right)$$

Perceptron
(2D and 2 classes)



$$y_{\vec{w}}(\vec{x}) = sign\left(\vec{w}^{\mathrm{T}}\vec{x}\right) \in \{-1,+1\}$$



Neuron

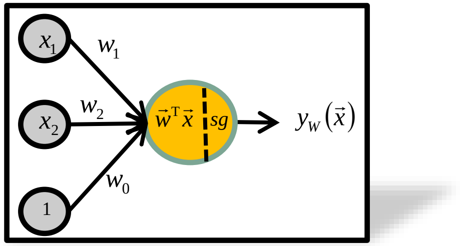
Dot product + activation function

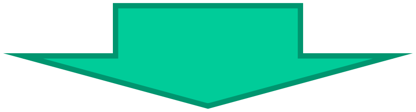
So far…

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

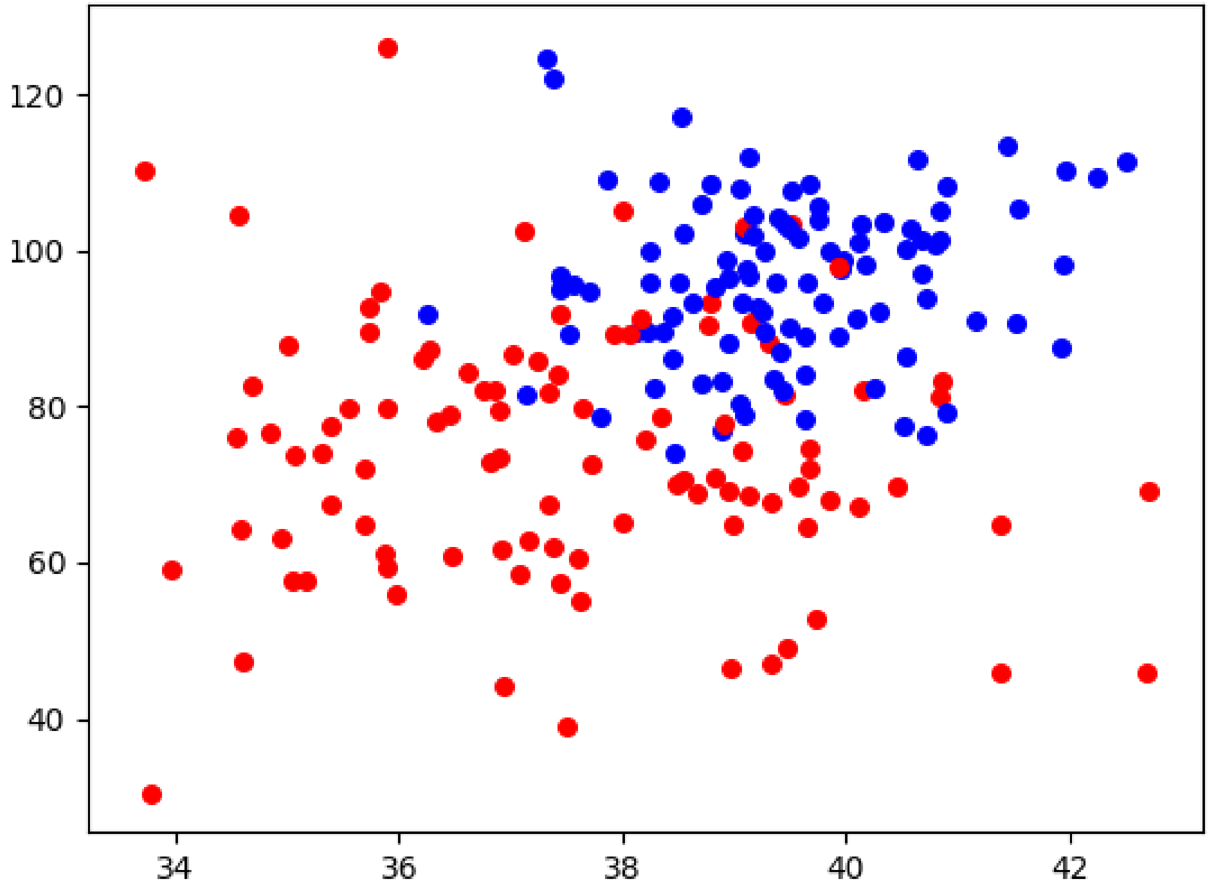




4. Training : find (w_0, w_1, w_2) that minimize $L\left(y_{\vec{w}}(\vec{x}), D\right)$

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

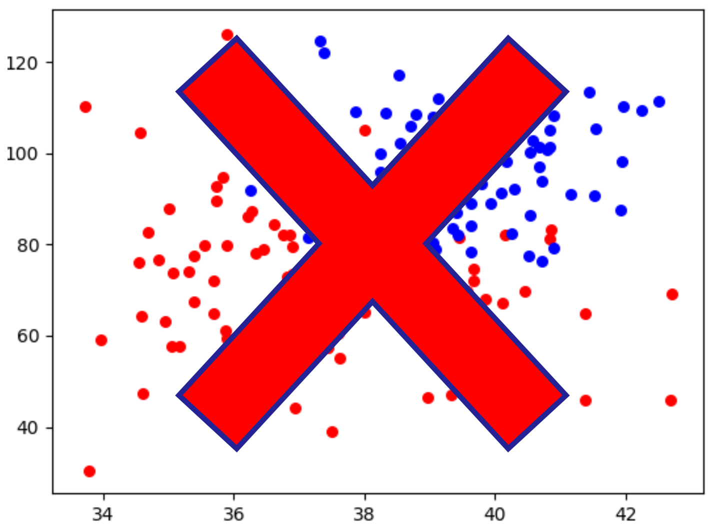
$\vec{x}$ t

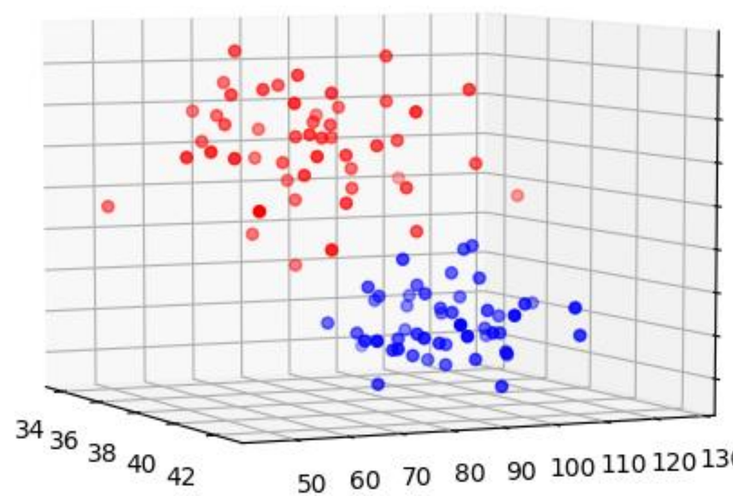
D

	(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





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

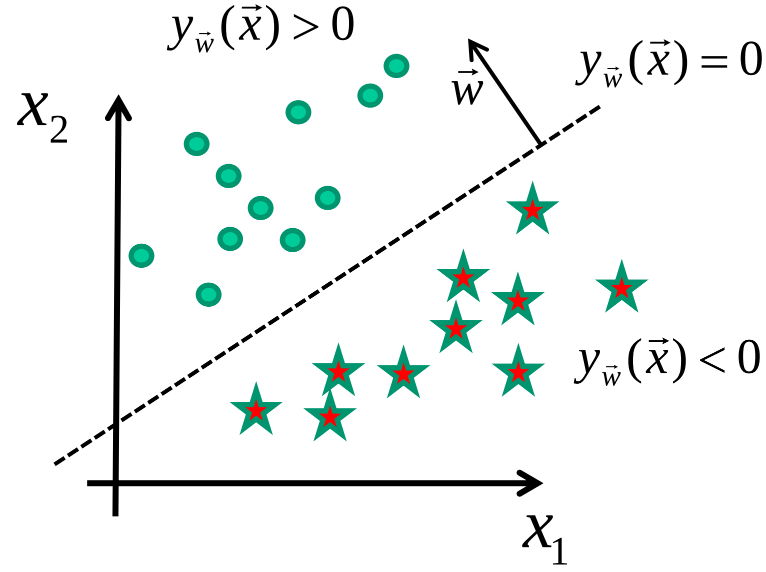
~~(line)~~

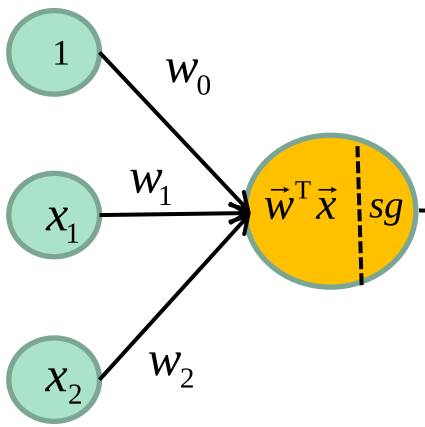
$$y_{\vec{w}}\left(\vec{x}\right) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0$$

(plane)

Perceptron

(2D and 2 classes)





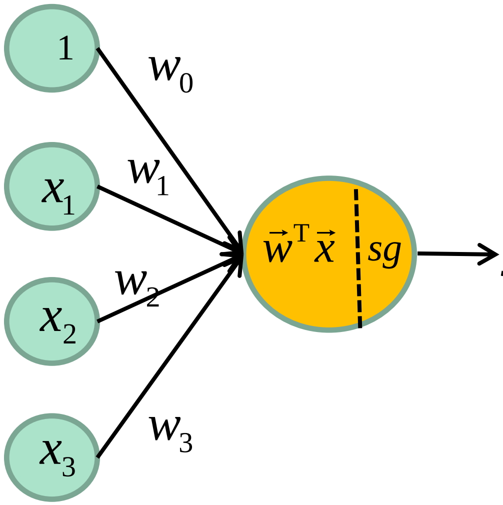
$y_{\vec{w}}(\vec{x}) = sign(\vec{w}^{\mathrm{T}}\vec{x}) \in \{-1,+1\}$

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

(line)

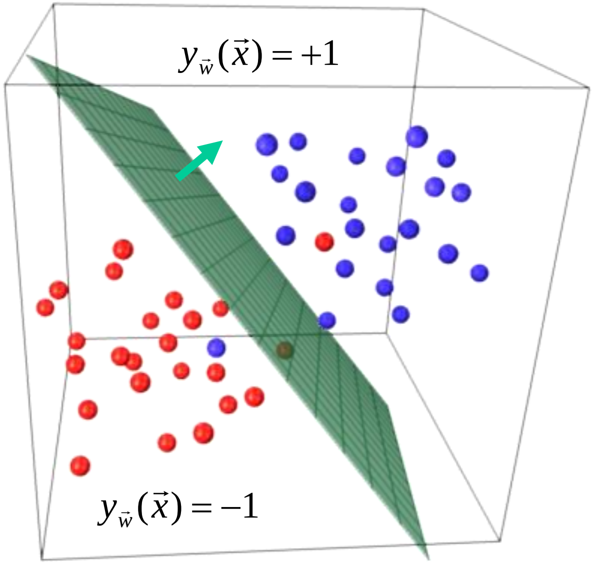
Perceptron

(3D and 2 classes)



$$y_{\vec{w}}(\vec{x}) = sign(\vec{w}^{\mathrm{T}}\vec{x}) \in \{-1,+1\}$$

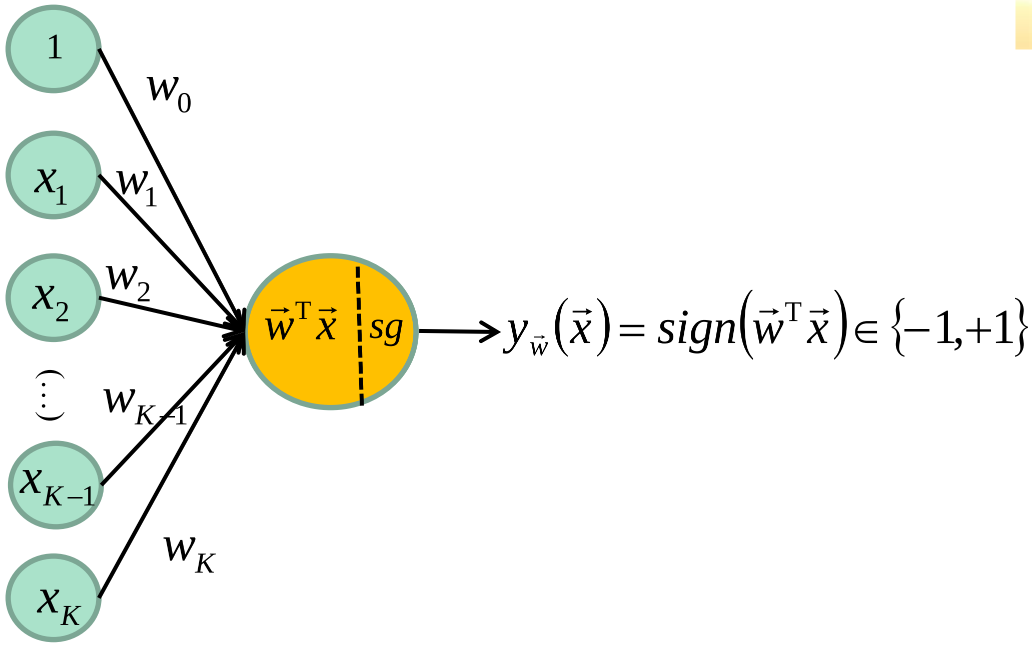
Example 3D



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

(plane)

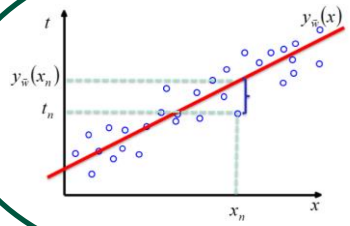
Perceptron
(K-D and 2 classes)



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

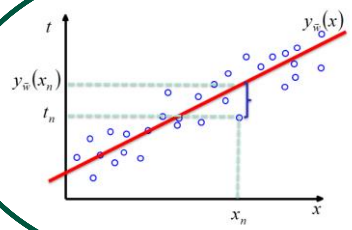
(hyperplane)

How do we train a Perceptron?



$$\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} \left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right)^2$$

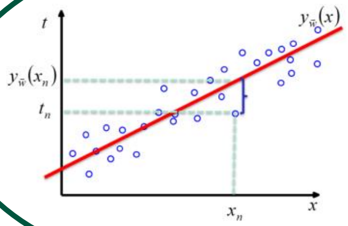
$$\vec{w} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T$$



$$\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} \left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right)^2$$

$$\vec{w} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T$$

YES! If the data follows a Gaussian distribution

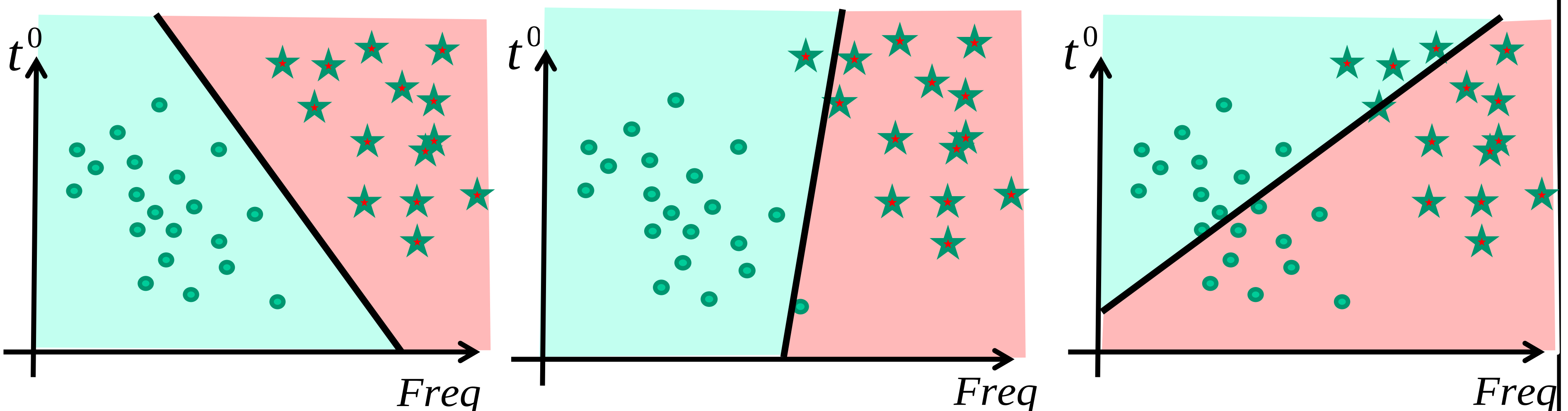


$$\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} \left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right)^2$$

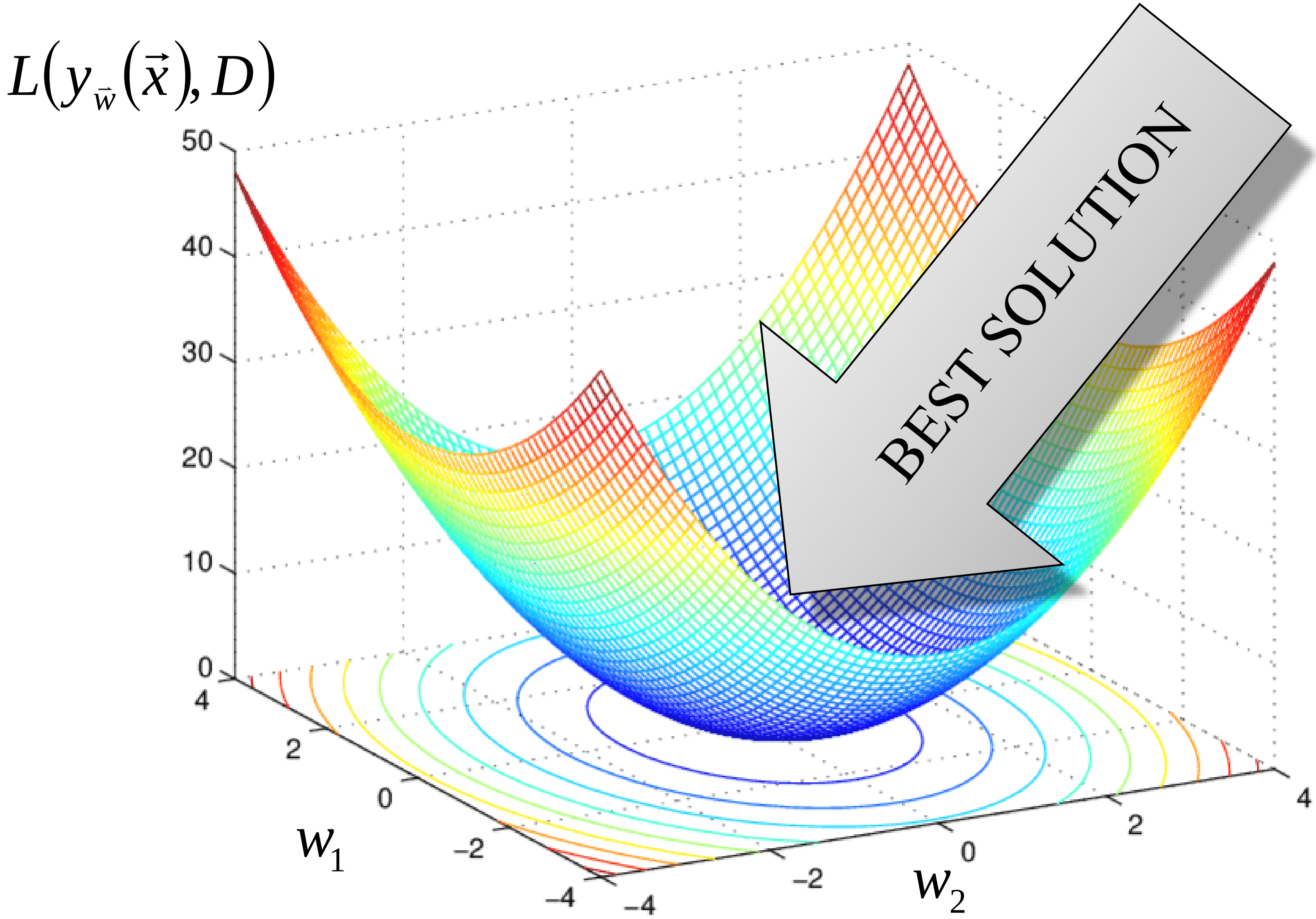
$$\vec{w} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T$$

Otherwise, we need another solution

Loss function



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

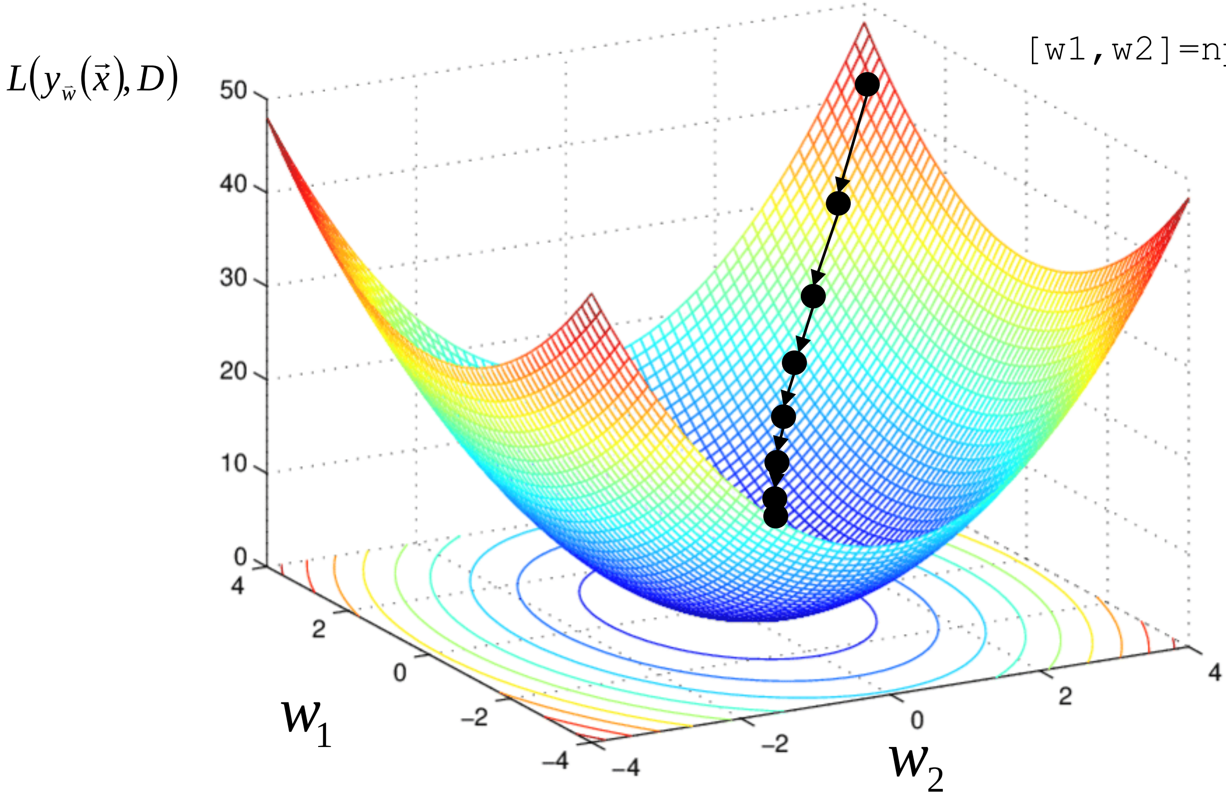

$L(y_{\vec{w}}(\vec{x}), D)$
50
40
30
20
10
0
4
2
0
−2
−4
w_1
−4
−2
0
2
4
w_2
BEST SOLUTION

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

Optimisation

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

Gradient of the loss function

learning rate

Stochastic gradient descent (SGD)

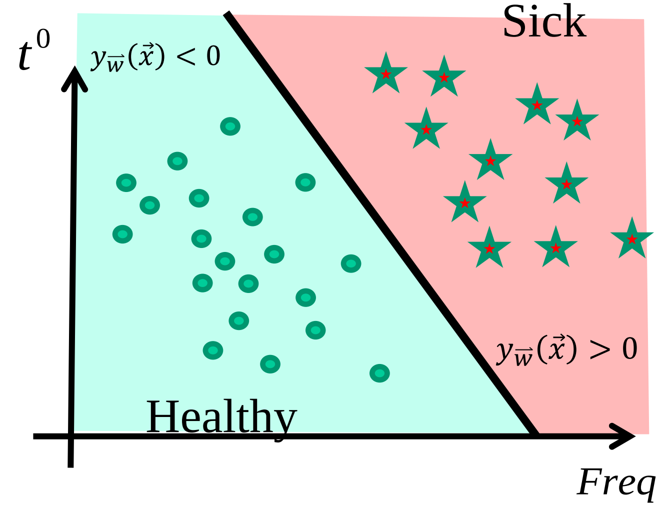
Init $\vec{\mathrm{w}}$
k=0
DO k=k+1
FOR n = 1 to N

$$\vec{w} = \vec{w} - \eta^{[k]} \nabla L(\vec{x}_n)$$

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

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

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

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 :

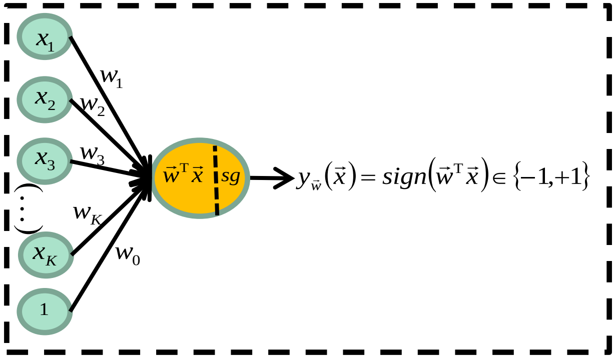
- **Too low** => slow convergence
- **Too large** => might not converge (even diverge)
- Can **decrease** at each iteration (e.g. $\eta^{[k]} = cst / k$)

So far…

1. Training dataset: D
2. Linear classification function: $y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_M x_M + w_0$
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D) = \sum_{\vec{x}_n \in V} -\vec{w}^{\mathrm{T}} \vec{x}_n t_n$

So far…

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

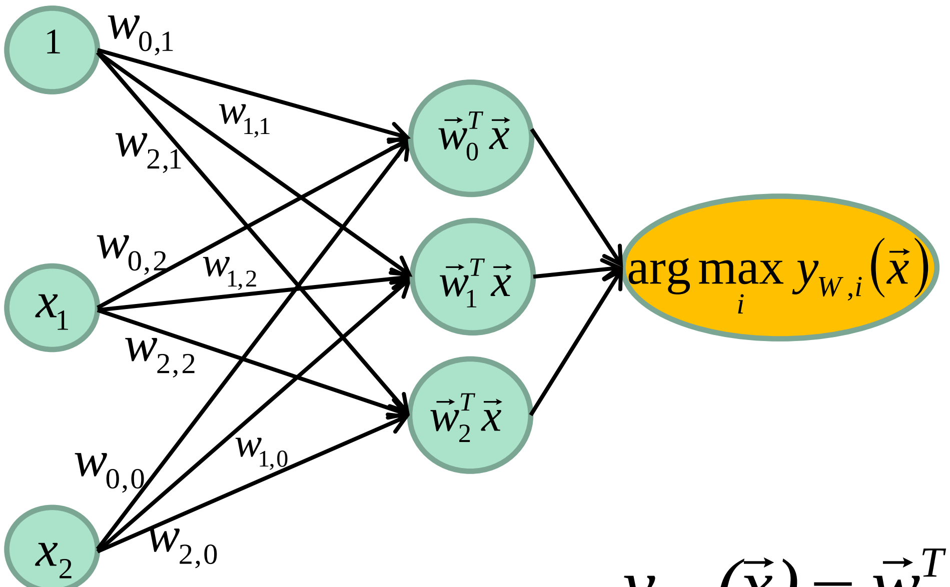


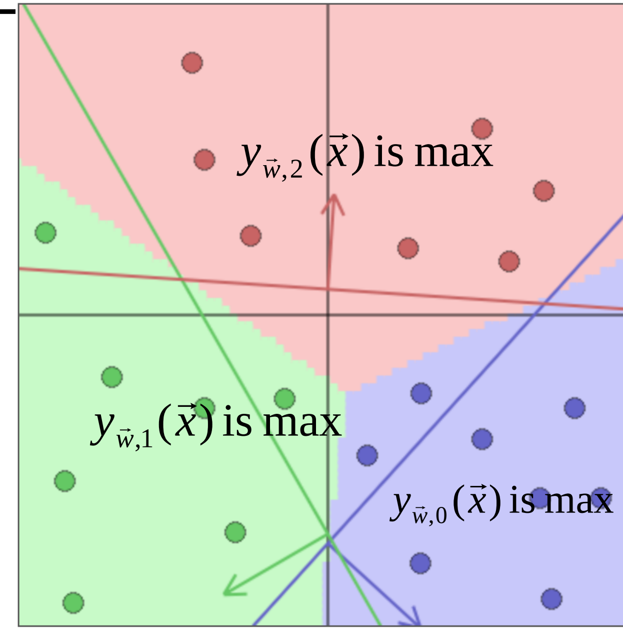
4. Training : find $\vec{w}$ that minimizes $L\left(y_{\vec{w}}(\vec{x}), D\right)$

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

Multiclass Perceptron

(2D and 3 classes)





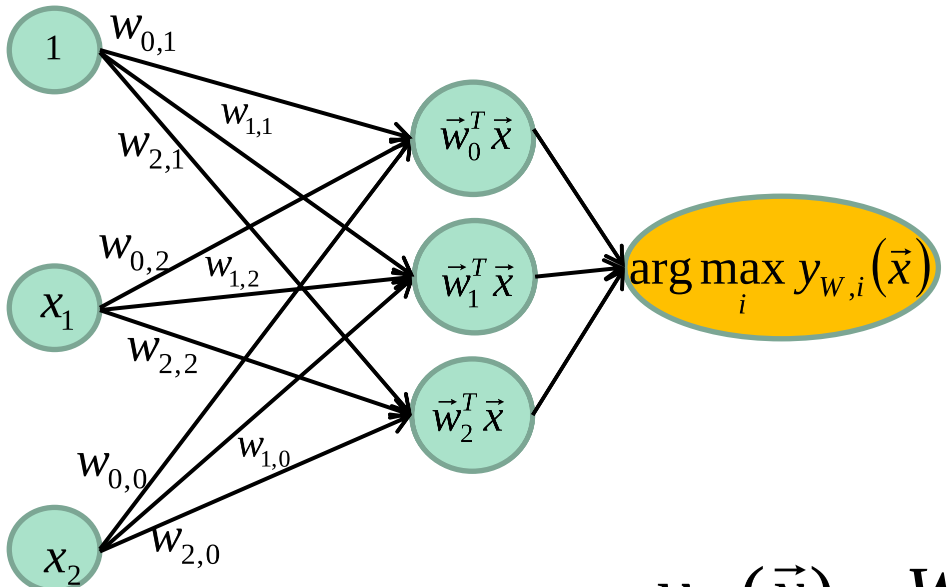
$$y_{\vec{w},0}(\vec{x}) = \vec{w}_0^T\vec{x} = w_{0,0} + w_{0,1}x_1 + w_{0,2}x_2$$

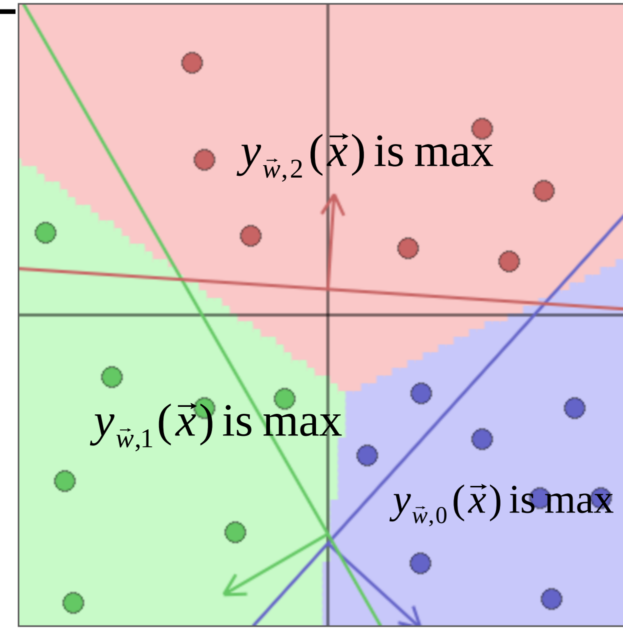
$$y_{\vec{w},1}(\vec{x}) = \vec{w}_1^T\vec{x} = w_{1,0} + w_{1,1}x_1 + w_{1,2}x_2$$

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

Multiclass Perceptron

(2D and 3 classes)





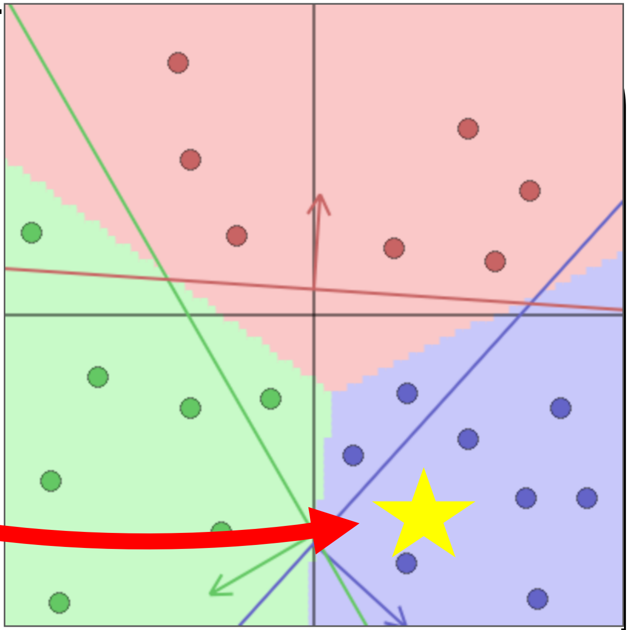
$$y_W(\vec{x}) = W\vec{x}$$

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

Multiclass Perceptron
(2D and 3 classes)

Example

(1.1, -2.0)



$$y_W(\vec{x}) = \begin{bmatrix} -2 & -3.6 & 0.5 \\ -4 & 2.4 & 4.1 \\ -6 & 4 & -4.9 \end{bmatrix} \begin{bmatrix} 1 \\ 1.1 \\ -2 \end{bmatrix} = \begin{array}{c c} \begin{bmatrix} -6.9 \\ -9.6 \\ \boxed{8.2} \end{bmatrix} & \begin{array}{l} \text{Class 0} \\ \text{Class 1} \\ \boxed{\text{Class 2}} \end{array} \end{array}$$

Multiclass Perceptron

Loss function

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

Sum over all wrongly classified samples

Score of the wrong class

Score of the true class

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

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 */
            $\vec{w}_j = \vec{w}_j - \eta \vec{x}_n$
            $\vec{w}_{t_n} = \vec{w}_{t_n} + \eta \vec{x}_n$

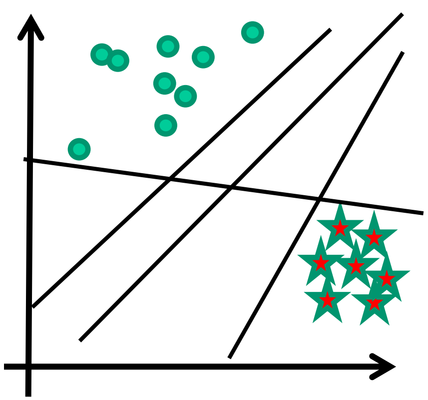
UNTIL every data is well classified or k > K_MAX.

Perceptron

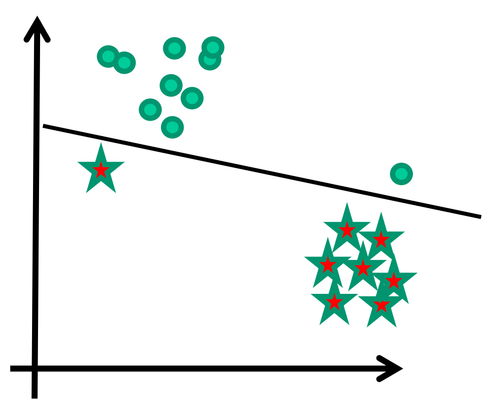
Advantages:
- Very simple
- Does **NOT** assume the data follows a **Gaussian distribution**.
- If data is **linearly separable**, convergence is **guaranteed**.

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

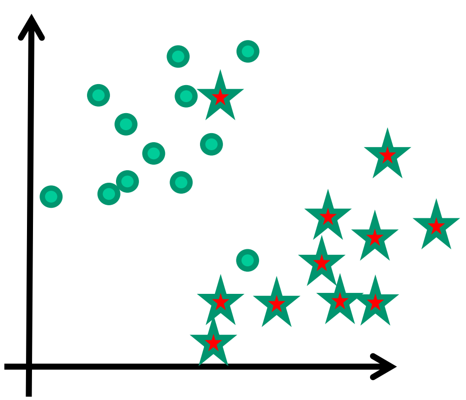
Many “optimal” solutions



Suboptimal solution



Will never converge



Two famous ways of improving the Perceptron

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

Logistic regression

1. New network architecture

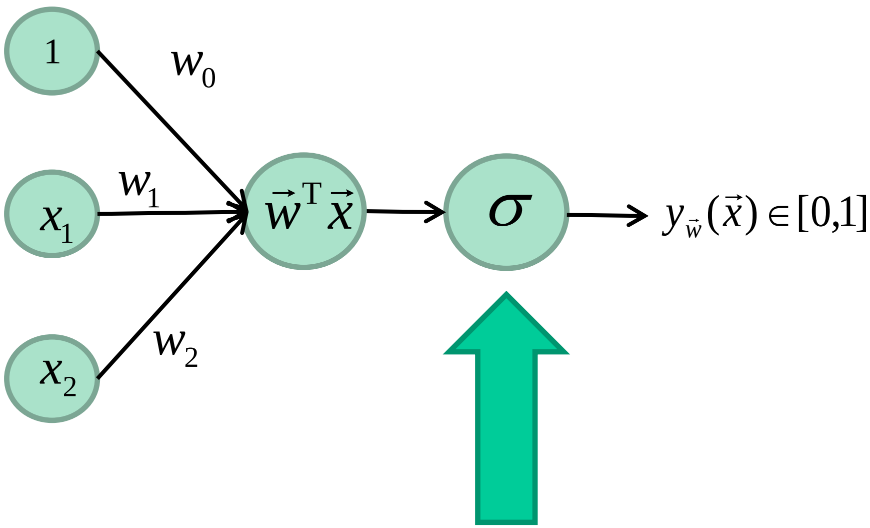
Multilayer Perceptron / CNN

Logistic regression

Logistic regression

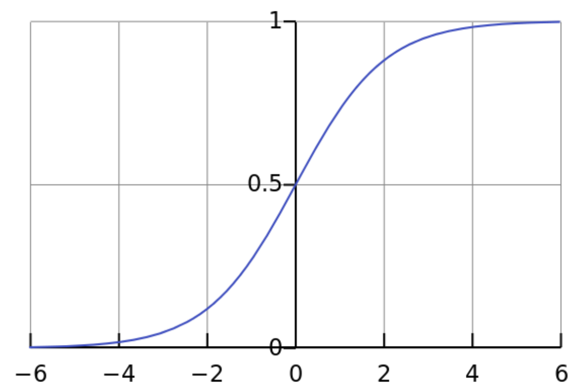
(2D, 2 classes)

New activation function: **sigmoid**



Sigmoid activation function

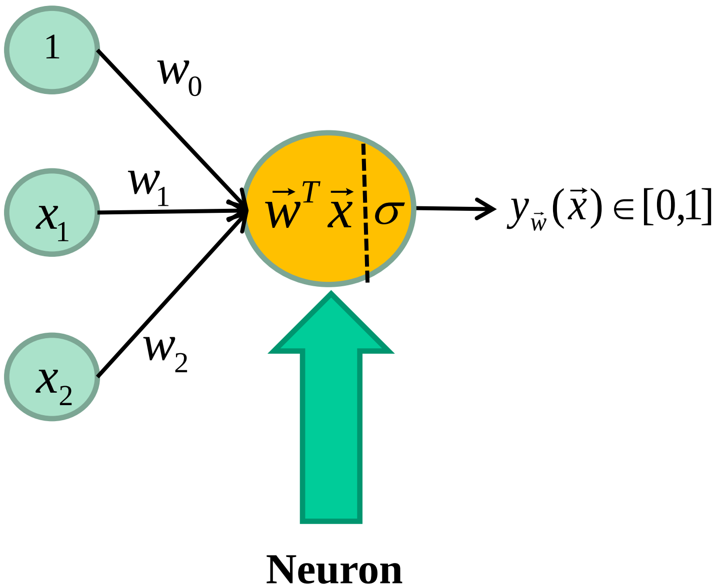
$$\sigma(t) = \frac{1}{1+e^{-t}}$$



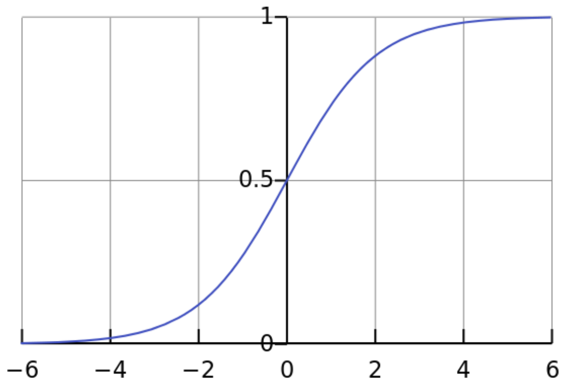
Logistic regression

(2D, 2 classes)

New activation function: **sigmoid**



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

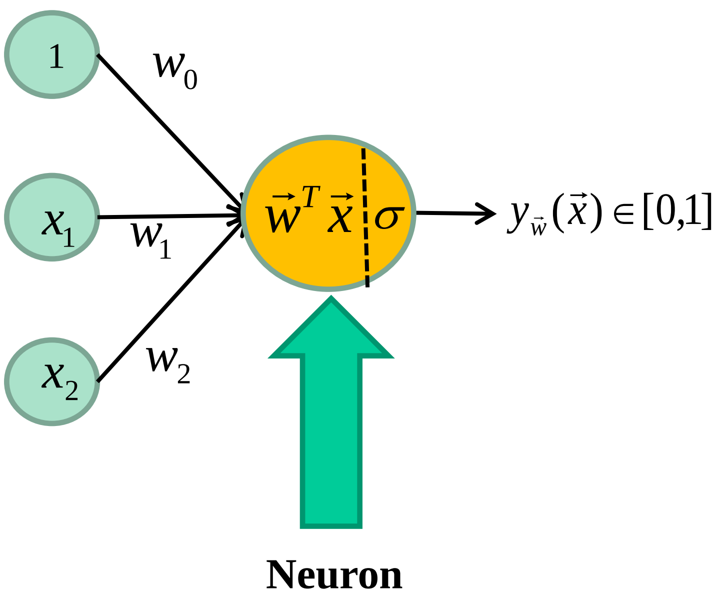


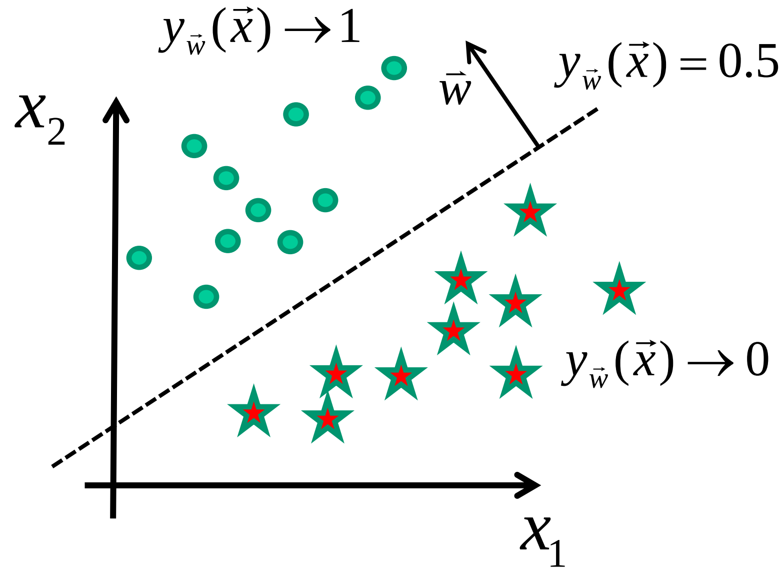
$$\boxed{y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T \vec{x}\right) = \frac{1}{1+e^{-\vec{w}^T \vec{x}}}}$$

Logistic regression

(2D, 2 classes)

New activation function: **sigmoid**





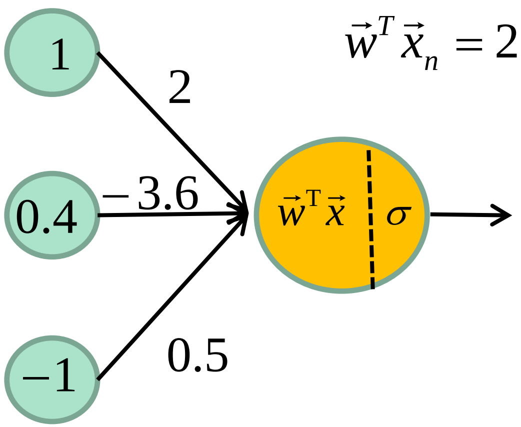
$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T\vec{x}\right)$$

Logistic regression

(2D, 2 classes)

Example

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



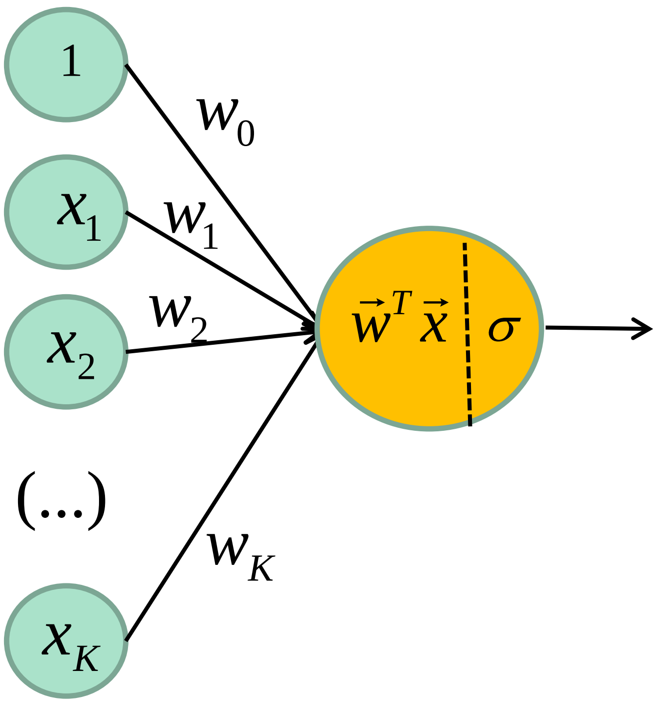
$$\vec{w}^T \vec{x}_n = 2 - 3.6 * 0.4 - 0.5 = -1.94$$

$$y_{\vec{w}}(\vec{x}) = \sigma(-1.94) = \frac{1}{1+e^{1.94}} = 0.125$$

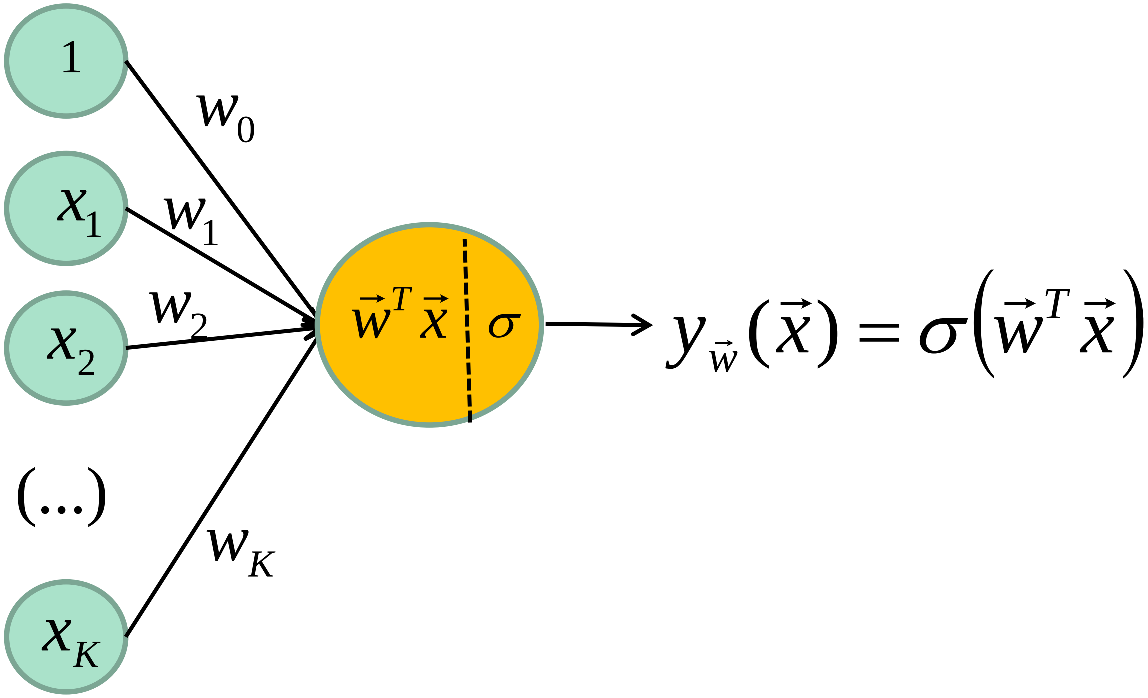
Since 0.125 is lower than 0.5, $\vec{x}_n$ is **behind** the plane.

Logistic regression
(K-D, 2 classes)

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

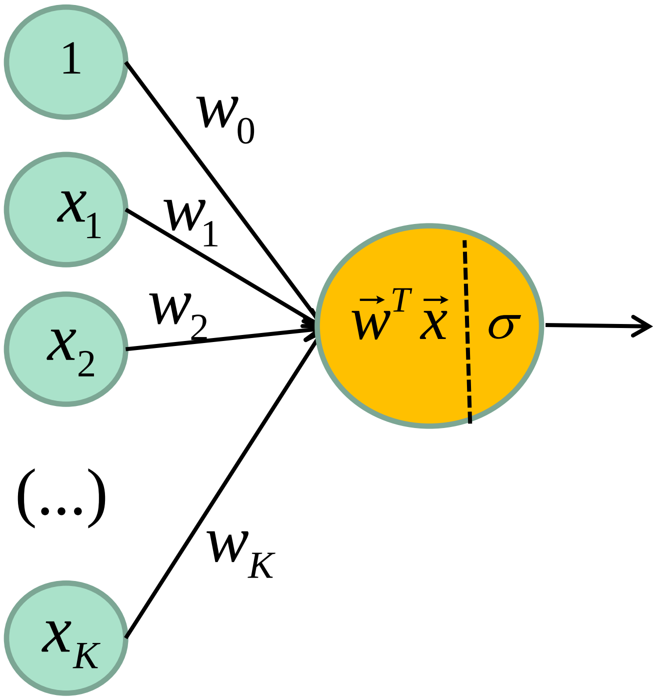


$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T\vec{x}\right)$$



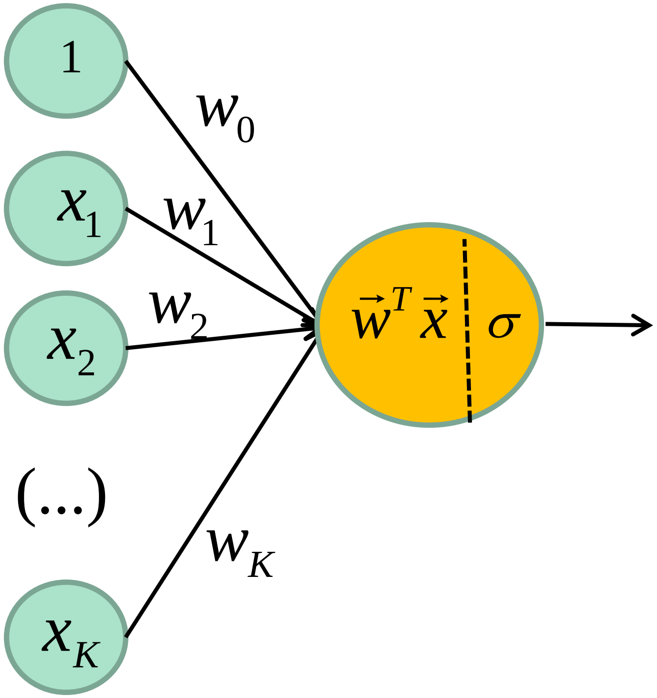
What is the loss function?

With a sigmoid, we can **simulate a conditional probability** of c_1 GIVEN $\vec{x}$



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

With a sigmoid, we can **simulate a conditional probability** of c_1 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**

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

2 Class Cross entropy

We can also show that

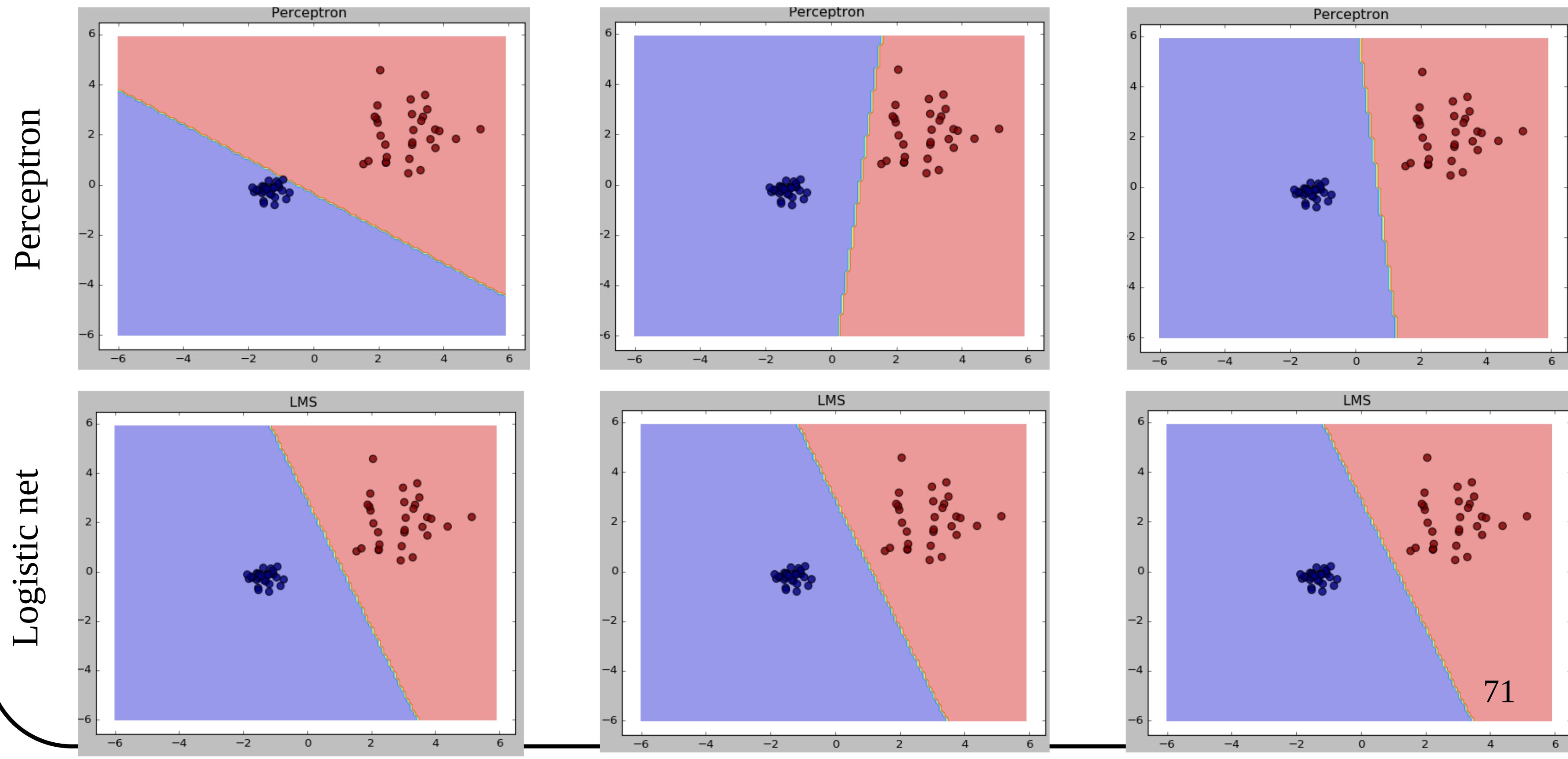
$$\nabla_{\vec{w}} L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^{N} (y_{\vec{w}}(\vec{x}_n) - t_n)\, \vec{x}_n$$

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

Logistic Network

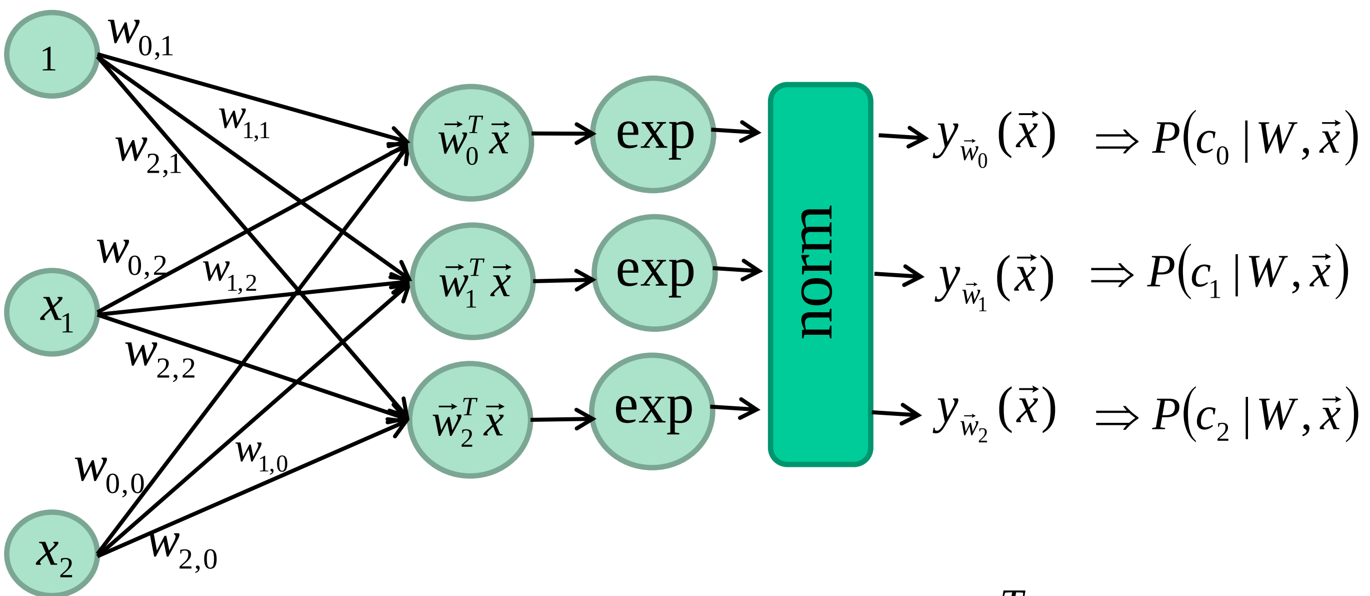
Advantages:

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



And for K>2 classes?

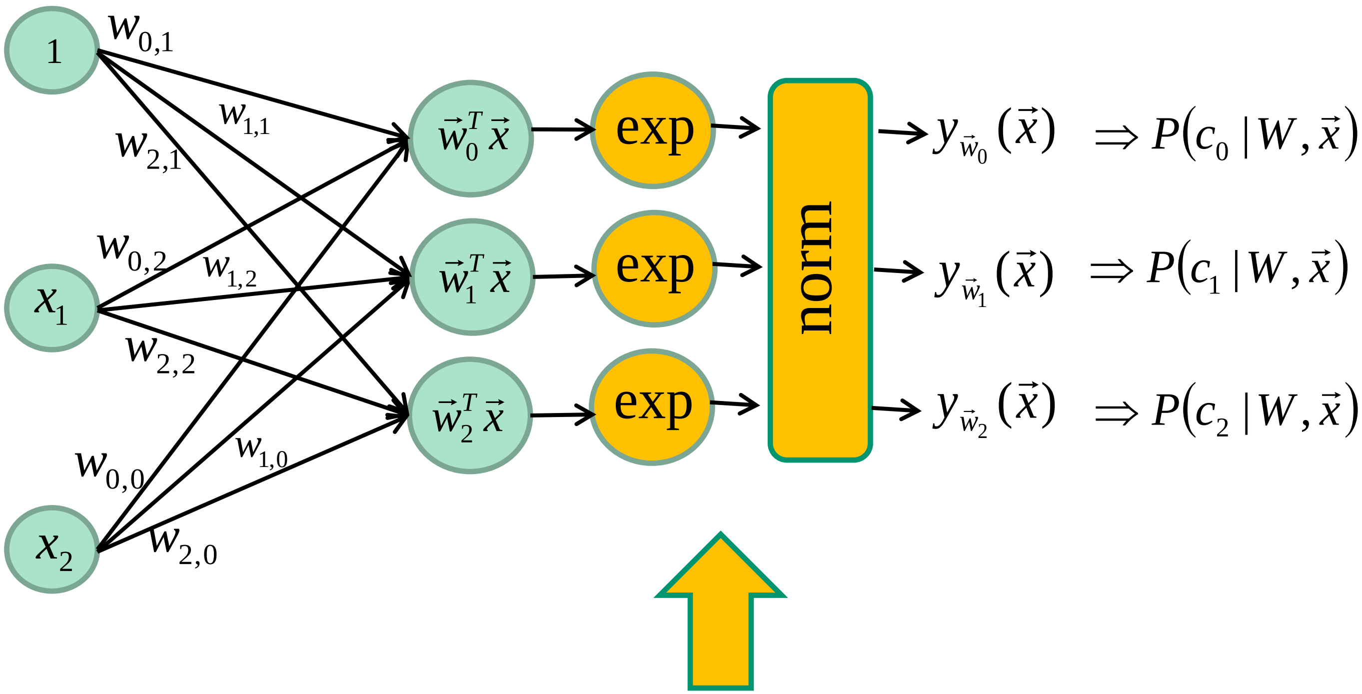
New activation function : **Softmax**



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

And for K>2 classes?

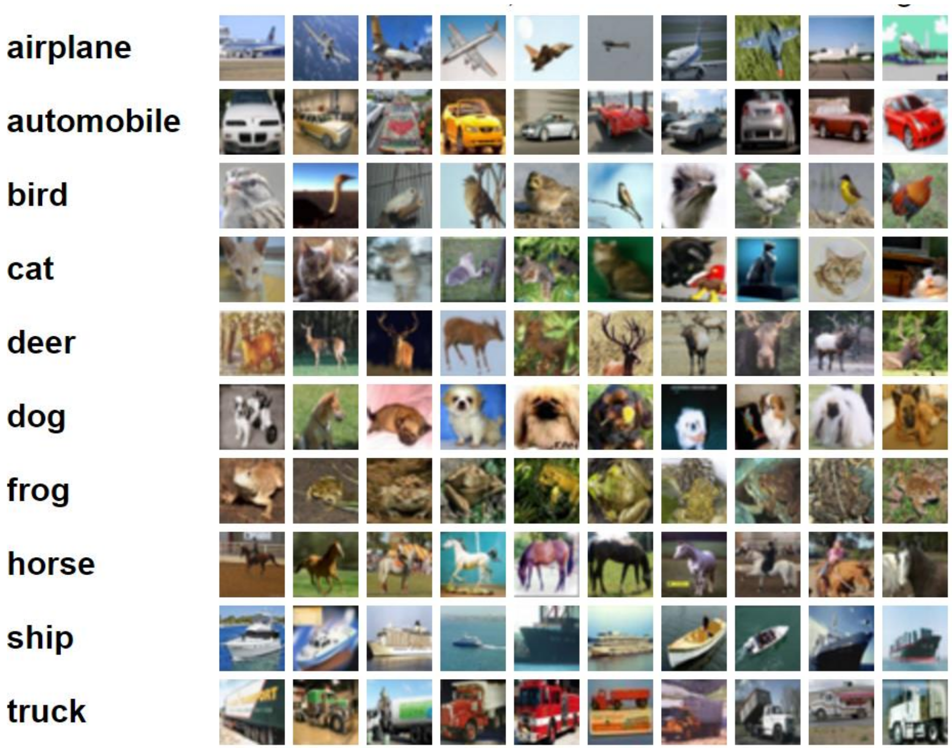
New activation function : **Softmax**



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

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

Regularization

Different weights may give the same score

$$\vec{x} = (1.0, 1.0, 1.0)$$

$$\vec{w}_1^T = [1, 0, 0]$$

$$\vec{w}_2^T = [1/3, 1/3, 1/3]$$

$$\vec{w}_1^T \vec{x} = \vec{w}_2^T \vec{x} = 1$$

Which weights are the best?

Solution: Maximum a posteriori

Maximum *a posteriori*

Regularization

Constant

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

Loss function

Regularization

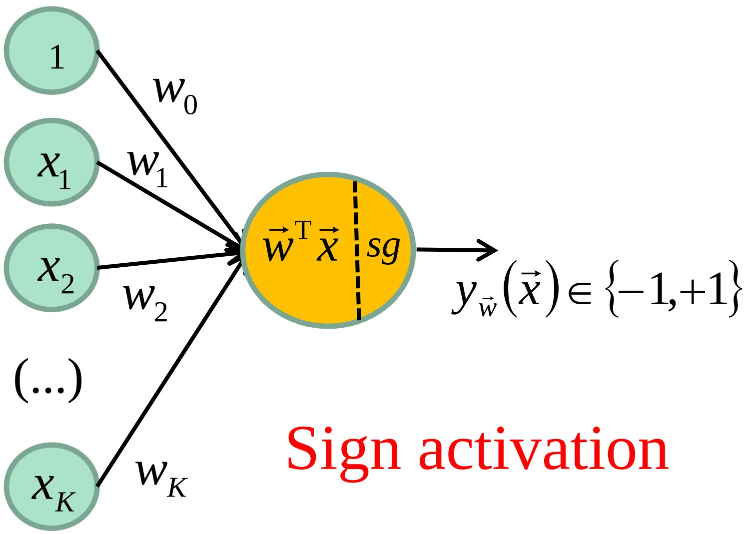
In general L1 or L2 $R(\theta) = \|\mathrm{W}\|_1 \text{ ou } \|\mathrm{W}\|_2$

Wow! Looooots of information!

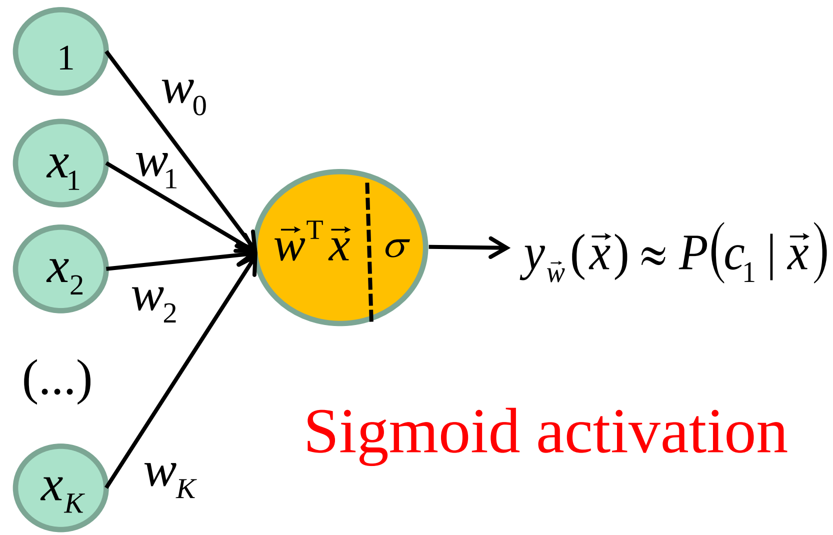
Lets recap…

Neural networks

2 classes

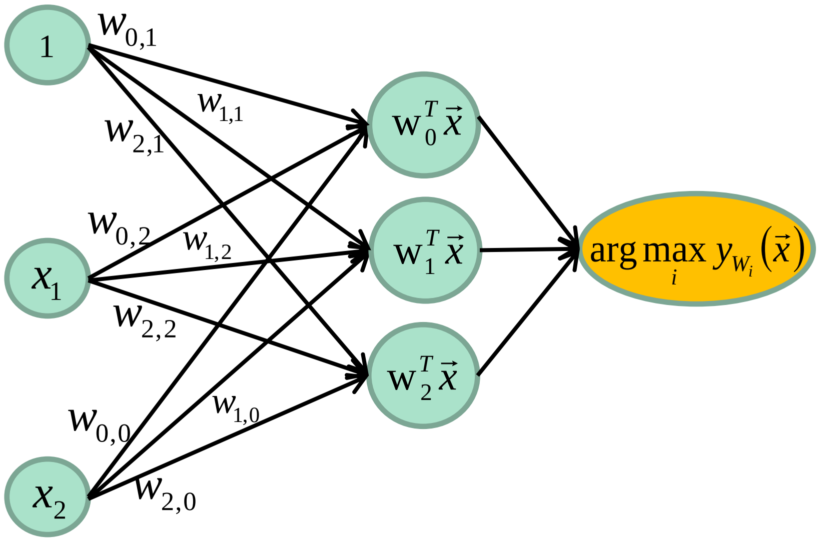


Perceptron

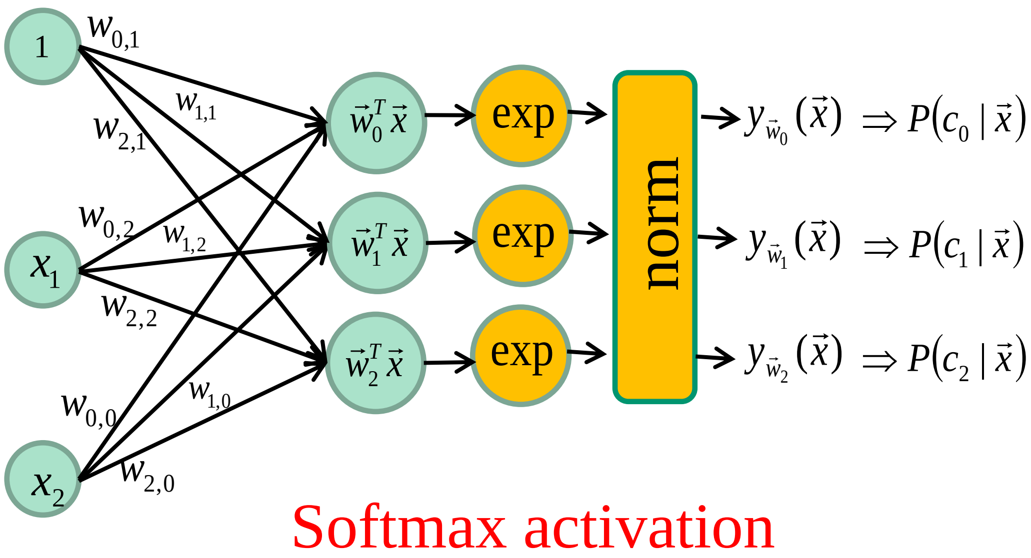


Logistic regression

K-Class Neural networks



Perceptron



Logistic regression

Loss functions

2 classes

$$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} -t_n \vec{w}^{\mathrm{T}} \vec{x}_n \qquad \text{where V is the set of wrongly classified samples}$$

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

Loss functions

K classes

$$L\left(y_{\vec{w}}(\vec{x}), D\right) = \sum_{\vec{x}_n \in V} \left(\vec{w}_j^T \vec{x}_n - \vec{w}_{t_n}^T \vec{x}_n\right)$$ where V *i*s the set of wrongly classified samples

$$L\left(y_{\vec{w}}(\vec{x}), D\right) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_{W_k}\left(\vec{x}_n\right)$$ *Cross entropy* loss with a Softmax

Constant

$$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{n=1}^{N} l\left(y_W\left(\vec{x}_n\right), t_n\right) \ + \lambda R\left(\mathrm{W}\right)$$

Loss function

Regularization

$$R\left(W\right) = \left\|W\right\|_1 \ or \ \left\|W\right\|_2$$

Now, lets go

DEEPER

Non-linearly separable training data

Three classical solutions

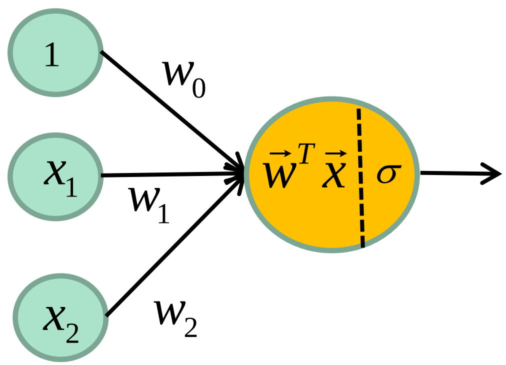
1. Acquire more data
2. Use a non-linear classifier
3. Transform the data

Non-linearly separable training data

Three classical solutions

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

2D, 2Classes, Linear logistic regression

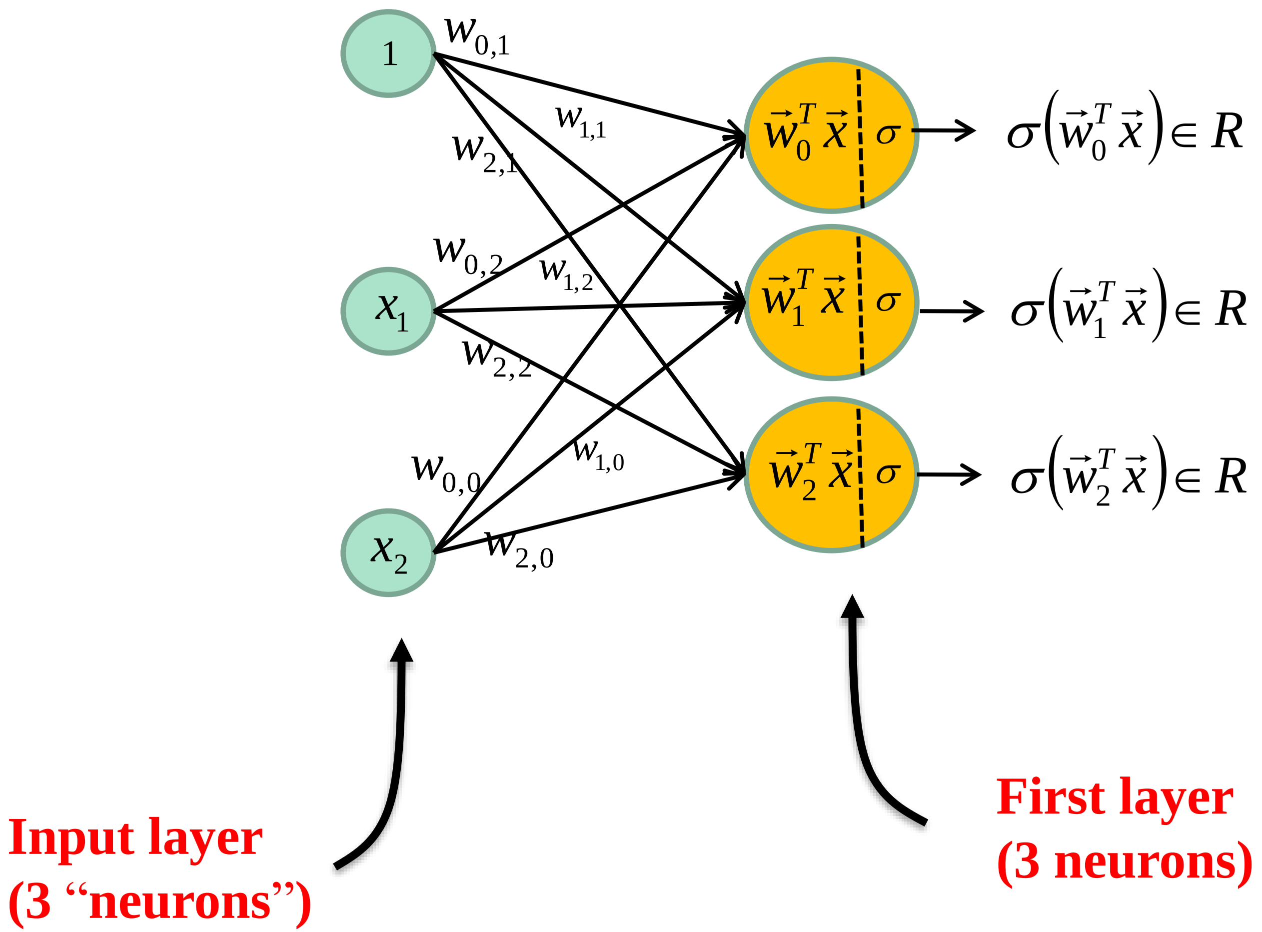


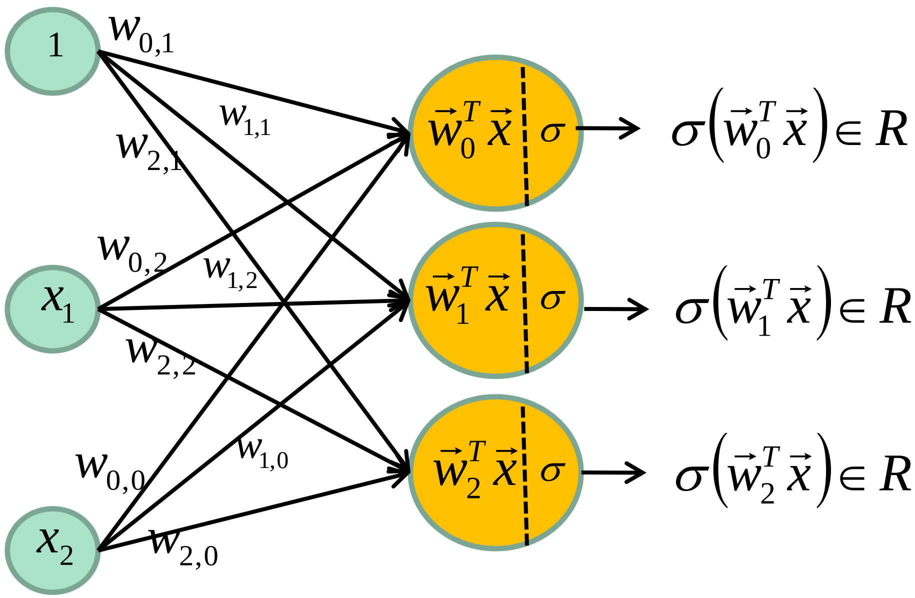
$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T\vec{x}\right) \in R$$

Input layer
(3 "neurons")

Output layer
(1 neuron with sigmoid)

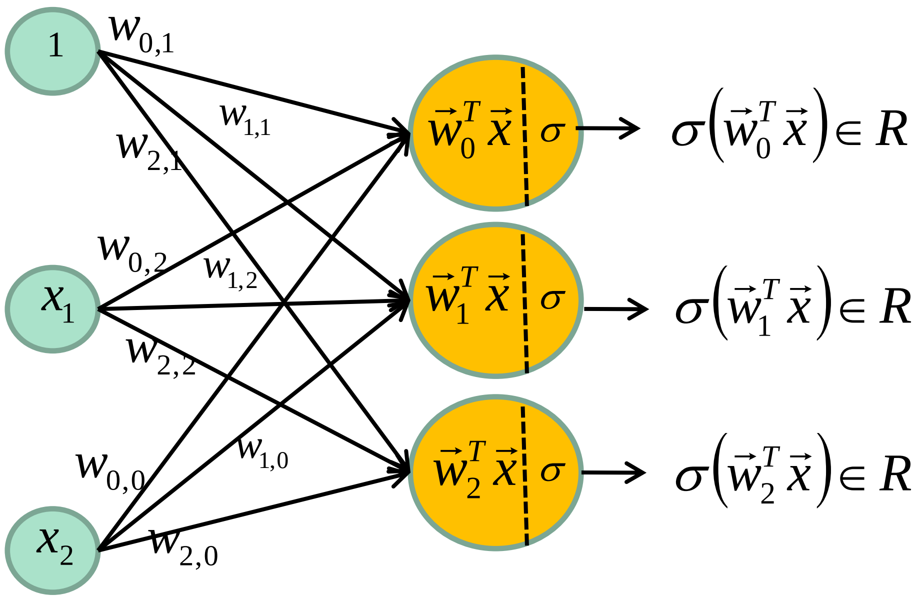
Let's add 3 neurons





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

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

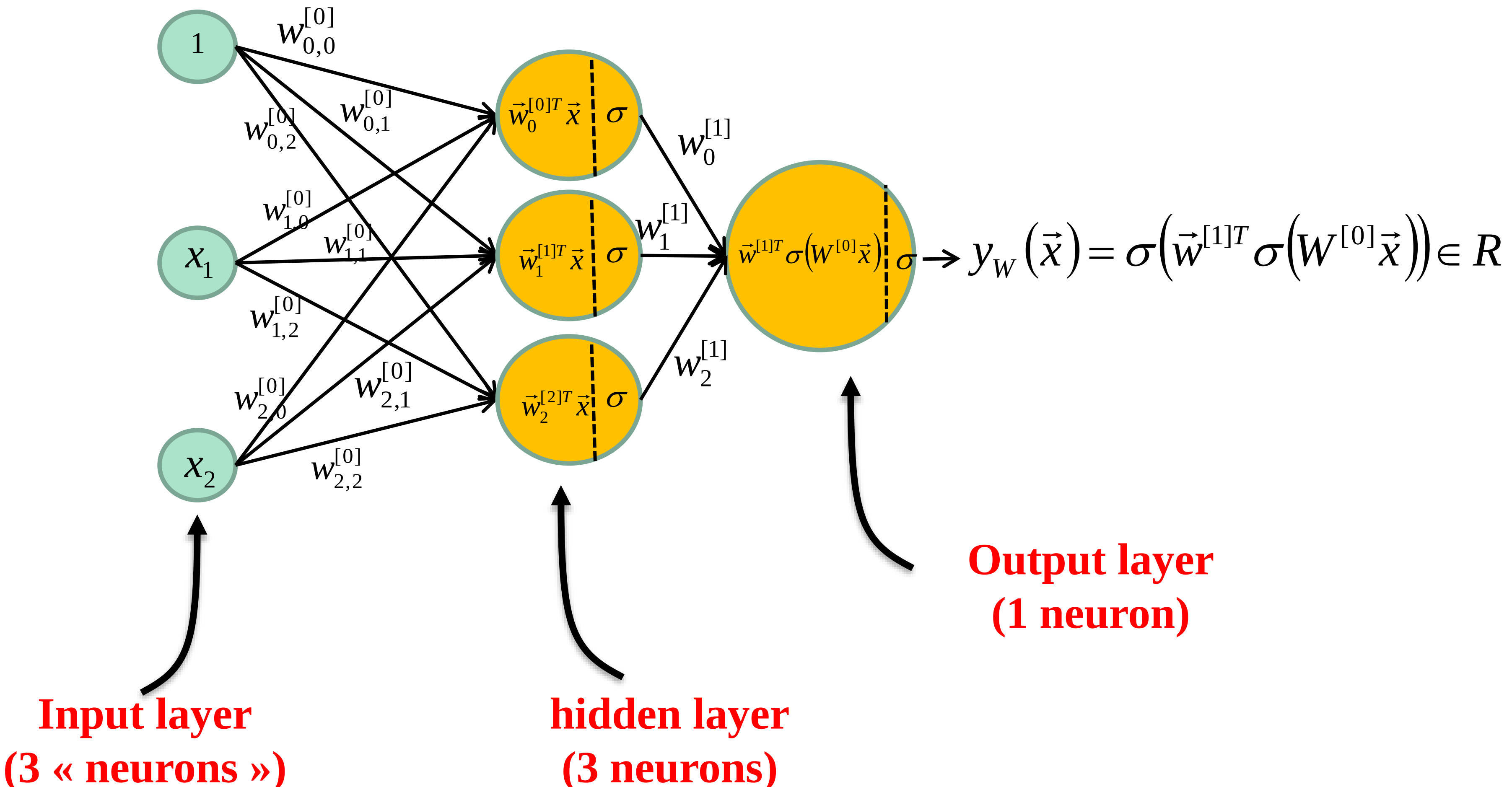


NOTE: The output of the first layer is a vector of **3 real** values

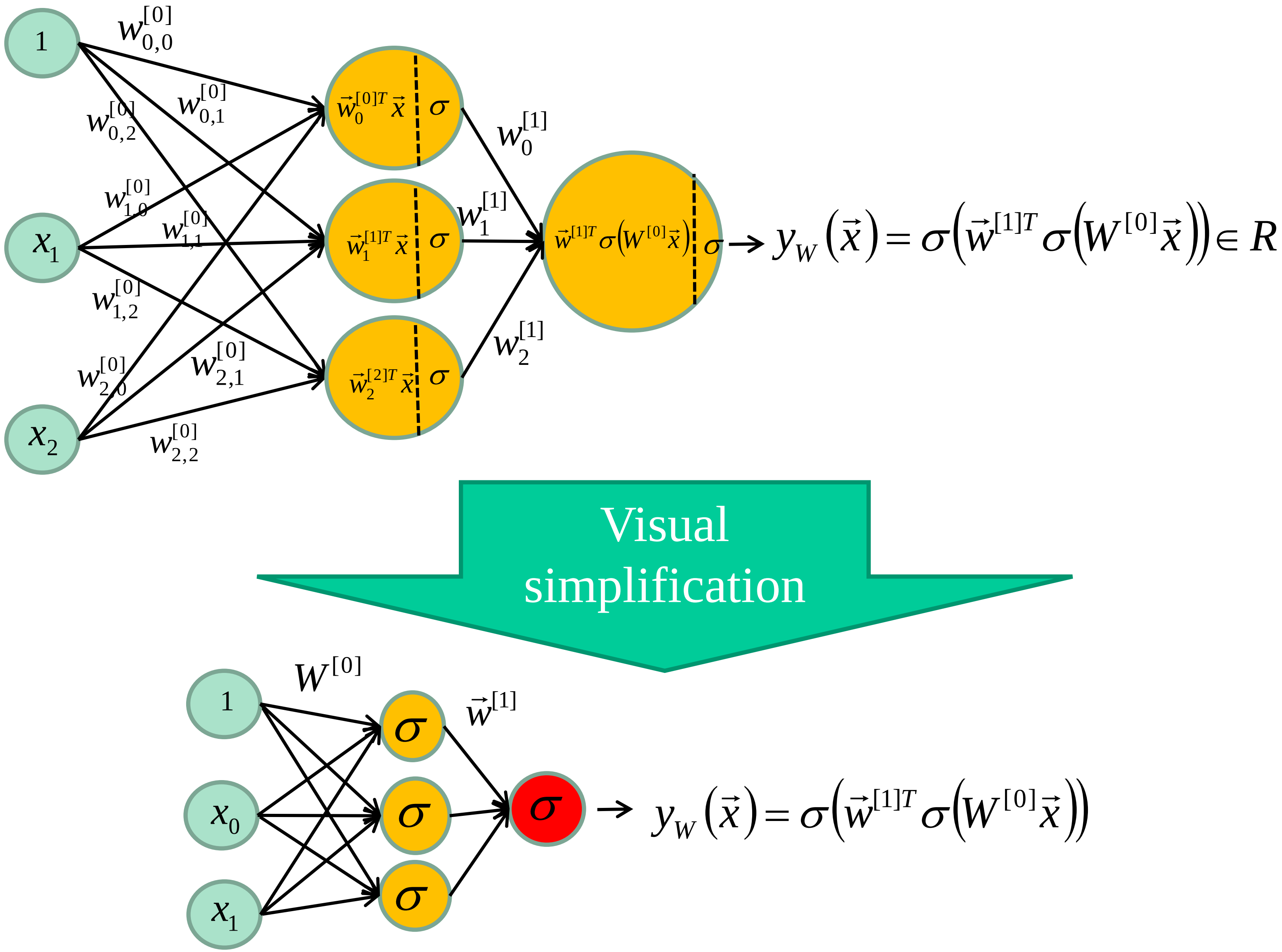
$$\sigma\left(W^{[0]}\vec{x}\right)$$

2-D, 2-Class, 1 hidden layer

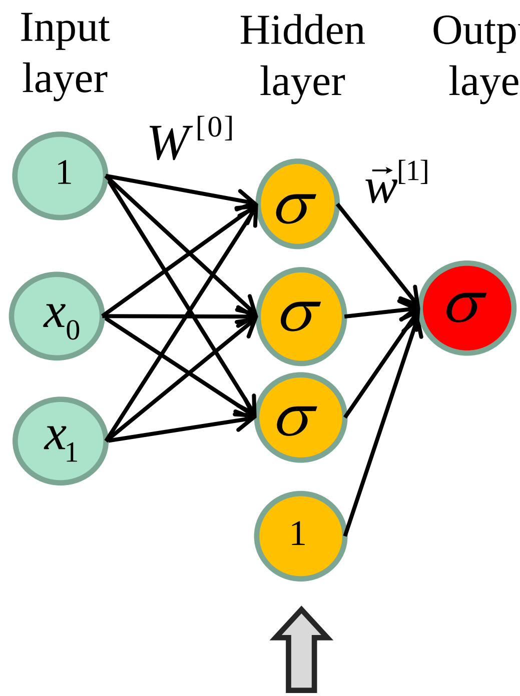
If we want a **2-class Classification** via a **logistic regression** (a **cross entropy loss**) we must add an **output neuron.**



2-D, 2-Class, 1 hidden layer



2-D, 2-Class, 1 hidden layer



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

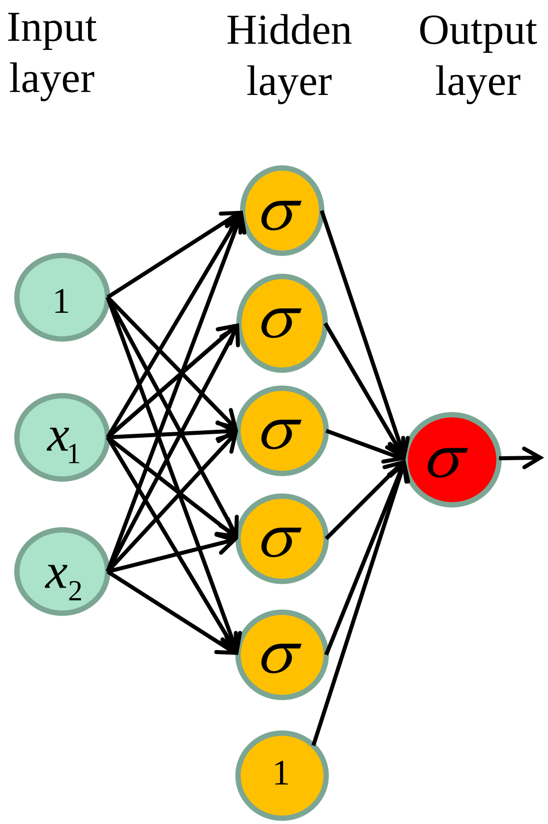
<u>bias</u> neuron.

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

3x3 — Input layer, Hidden layer

1x4 — Hidden layer, Output layer

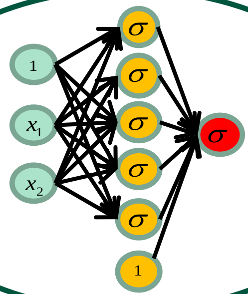
2-D, 2-Class, 1 hidden layer



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

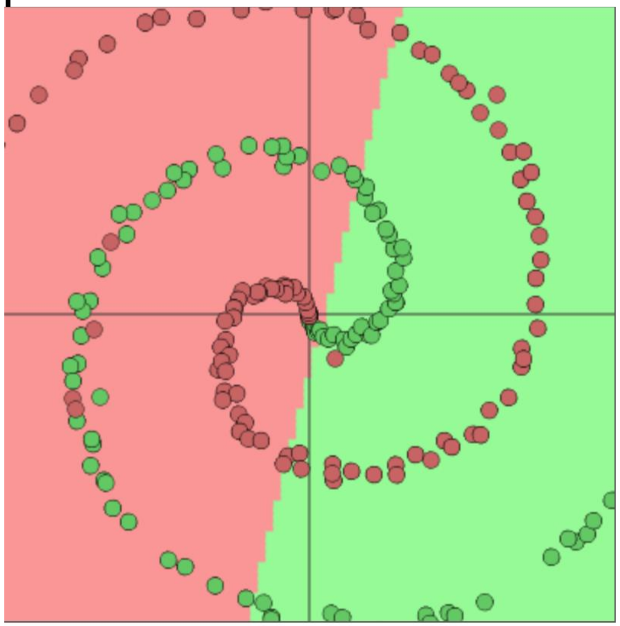
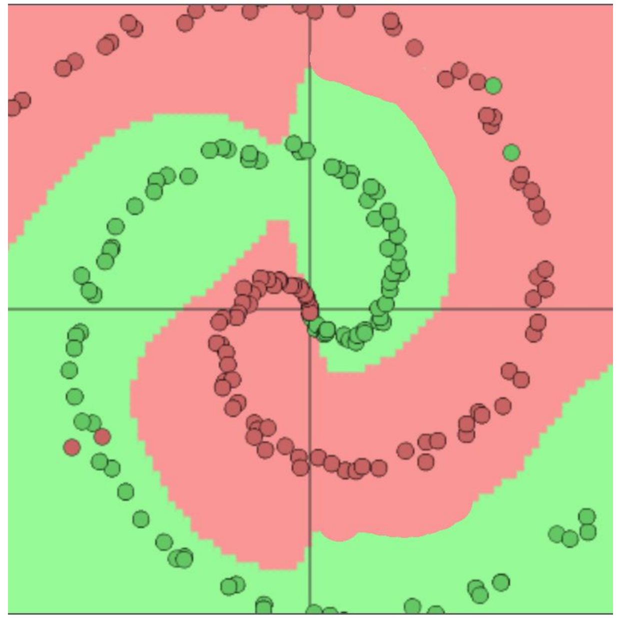
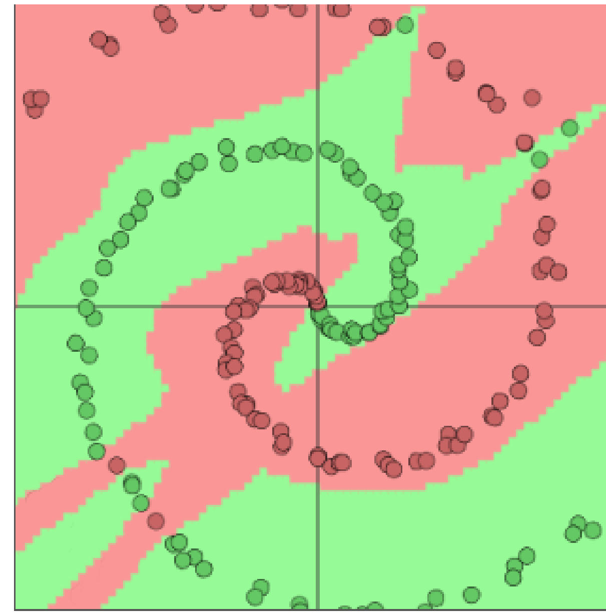
Increasing the number of neurons = increasing the **capacity of the model**

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



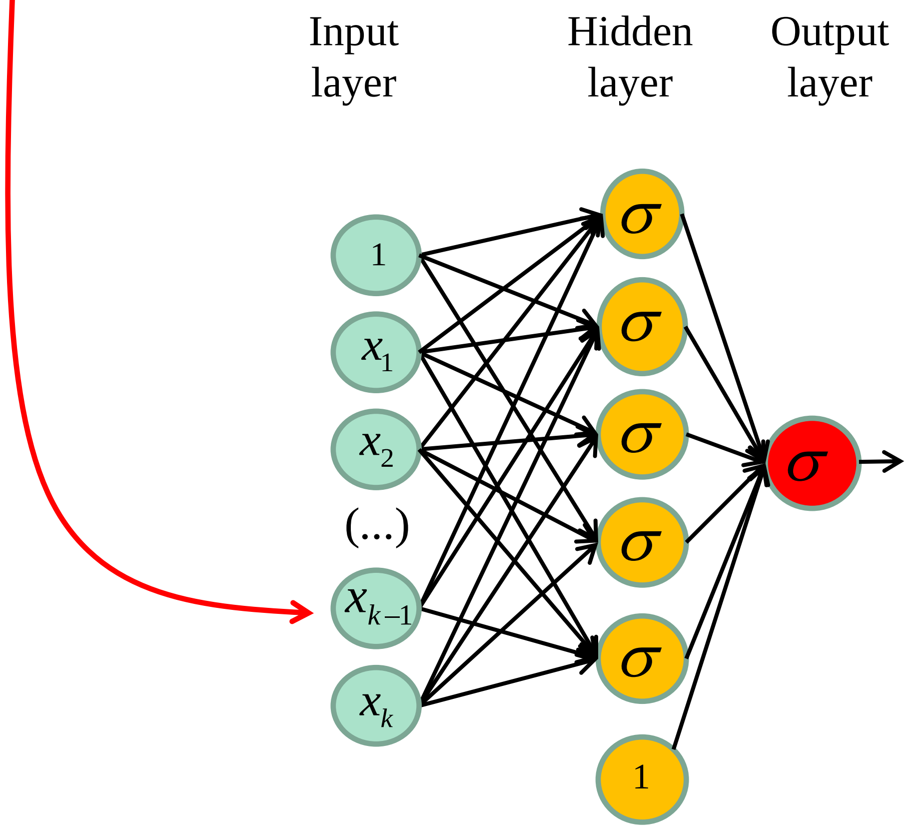
$\vec{w}^{\mathrm{T}}\vec{x}$ Line

Nb neurons VS Capacity

No hidden neuron	12 hidden neurons	60 hidden neurons
		
Linear classification **Underfitting** (low capacity)	Non linear classification **Good result** (good capacity)	Non linear classification **Over fitting** (too large capacity)

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

kD, 2Classes, 1 hidden layer

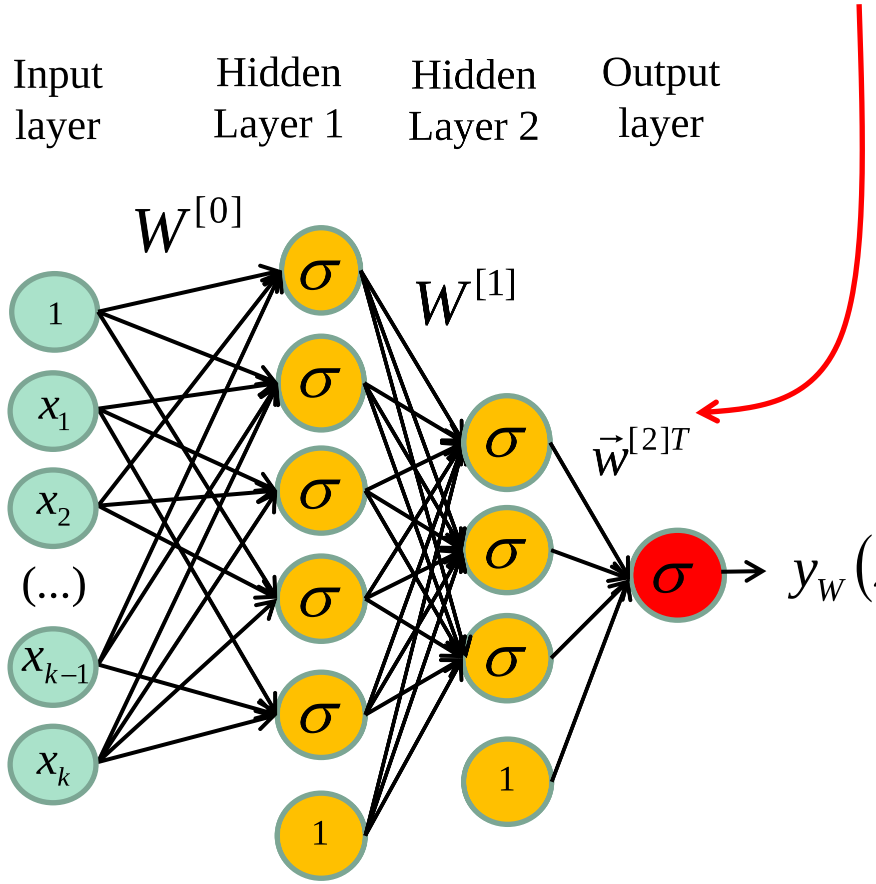


$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[1]T}\sigma\left(W^{[0]}\vec{x}\right)\right)$$

Increasing the dimensionality of the data = **more columns in** $W^{[0]}$

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

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



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

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

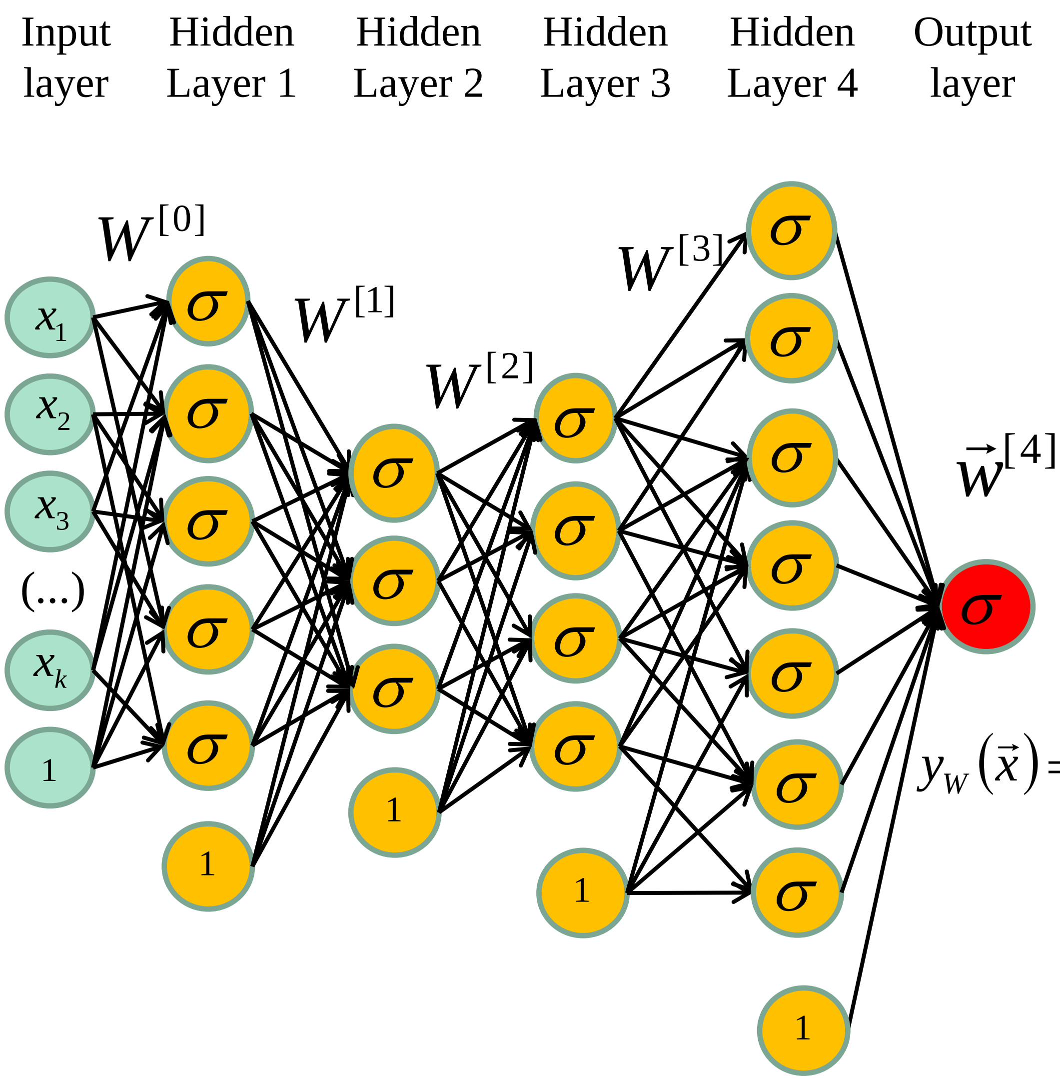
$$\vec{w}^{[2]} \in R^{4}$$

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

Adding an hidden layer = Adding a matrix multiplication

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

kD, 2 Classes, 4 hidden layer network



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

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

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

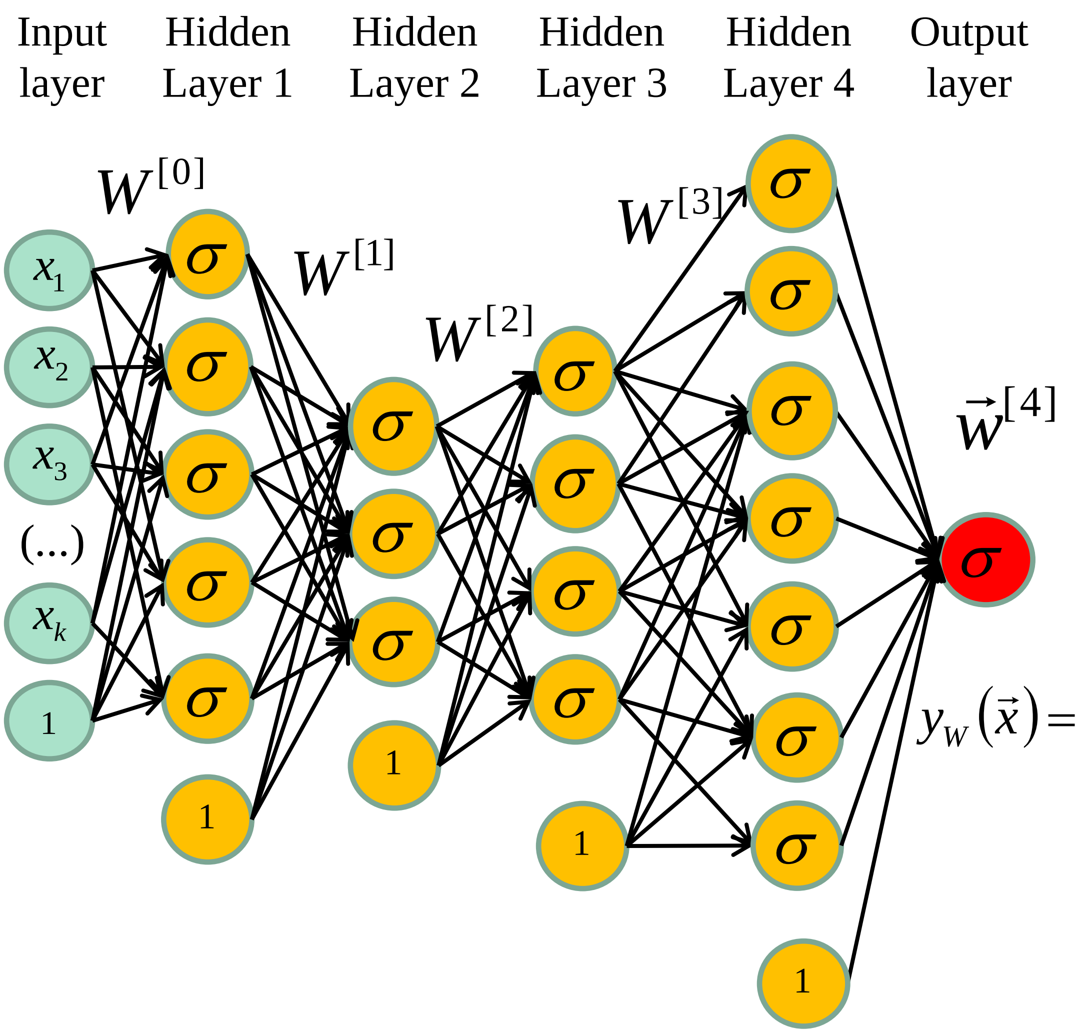
$$W^{[3]} \in R^{7\times 5}$$

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

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

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

kD, 2 Classes, 4 hidden layer network



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

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

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

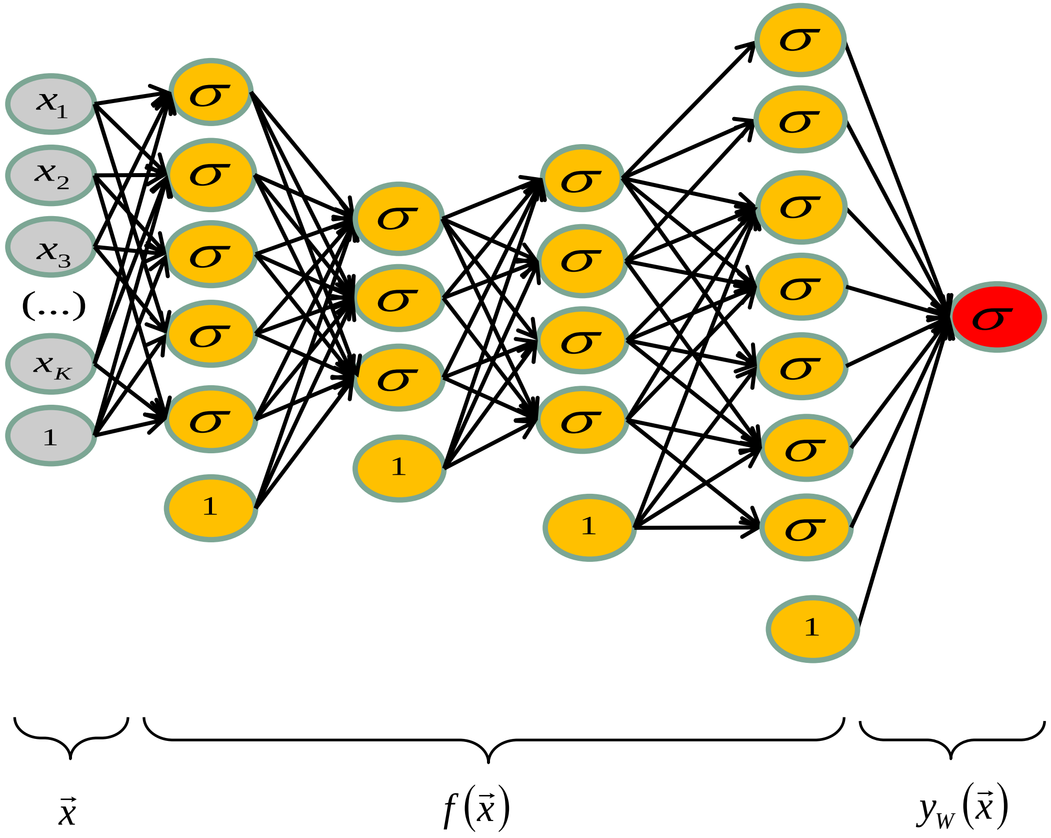
$$W^{[3]} \in R^{7\times 5}$$

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

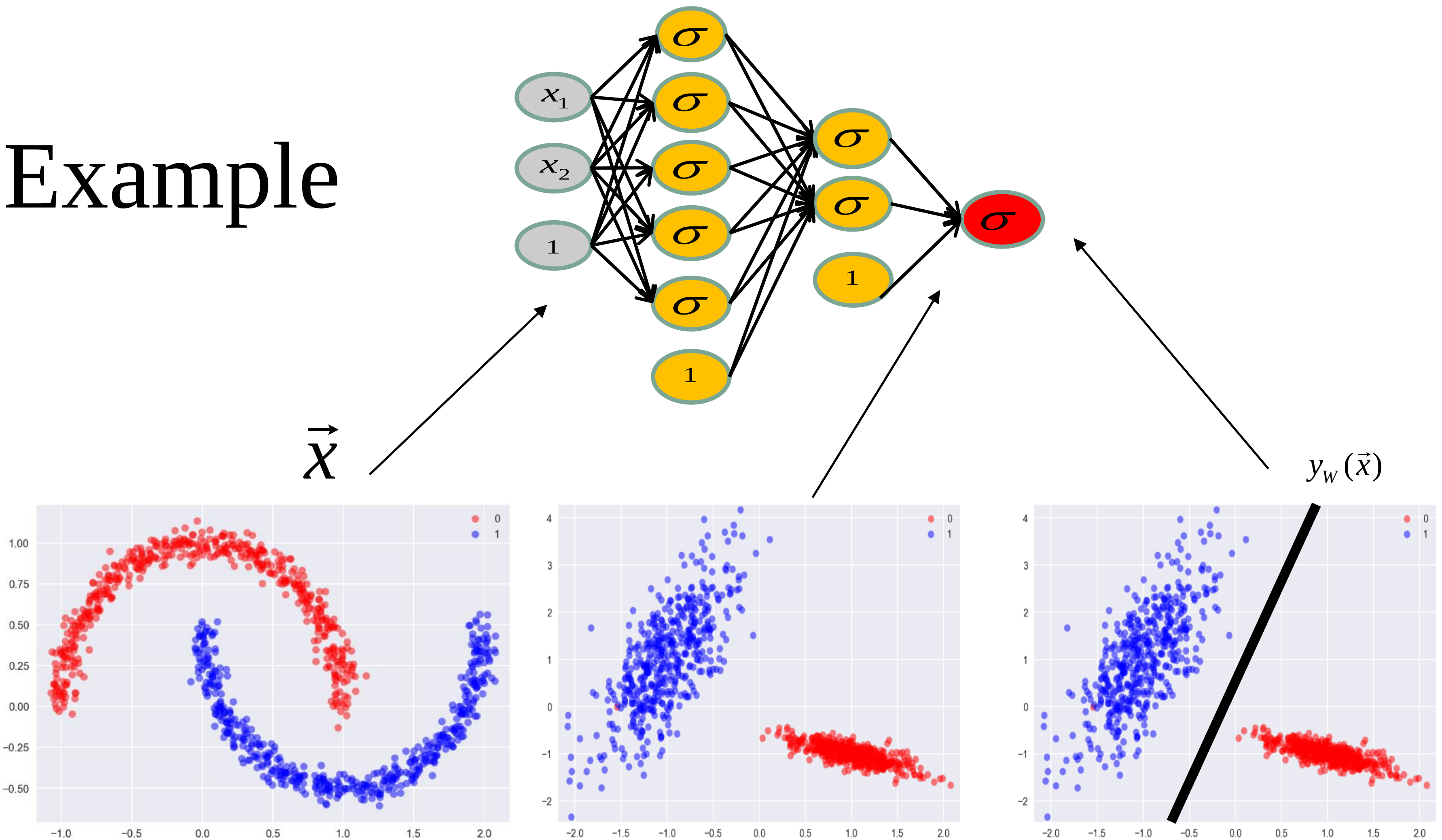
$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[4]T}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)\right)$$

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

Multilayer Perceptron



Example

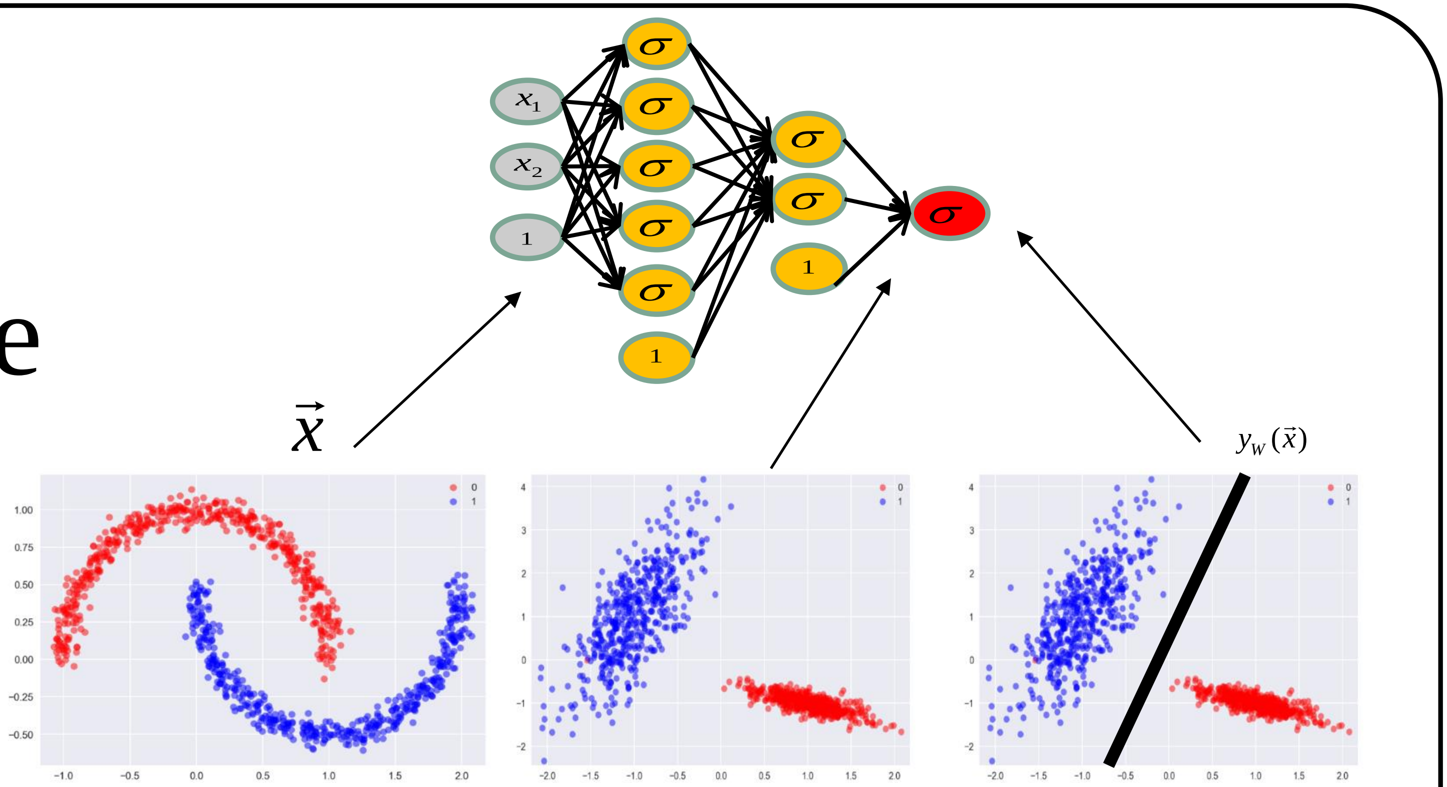


Input data

Output of the last layer

Output of the network

Example



Input data

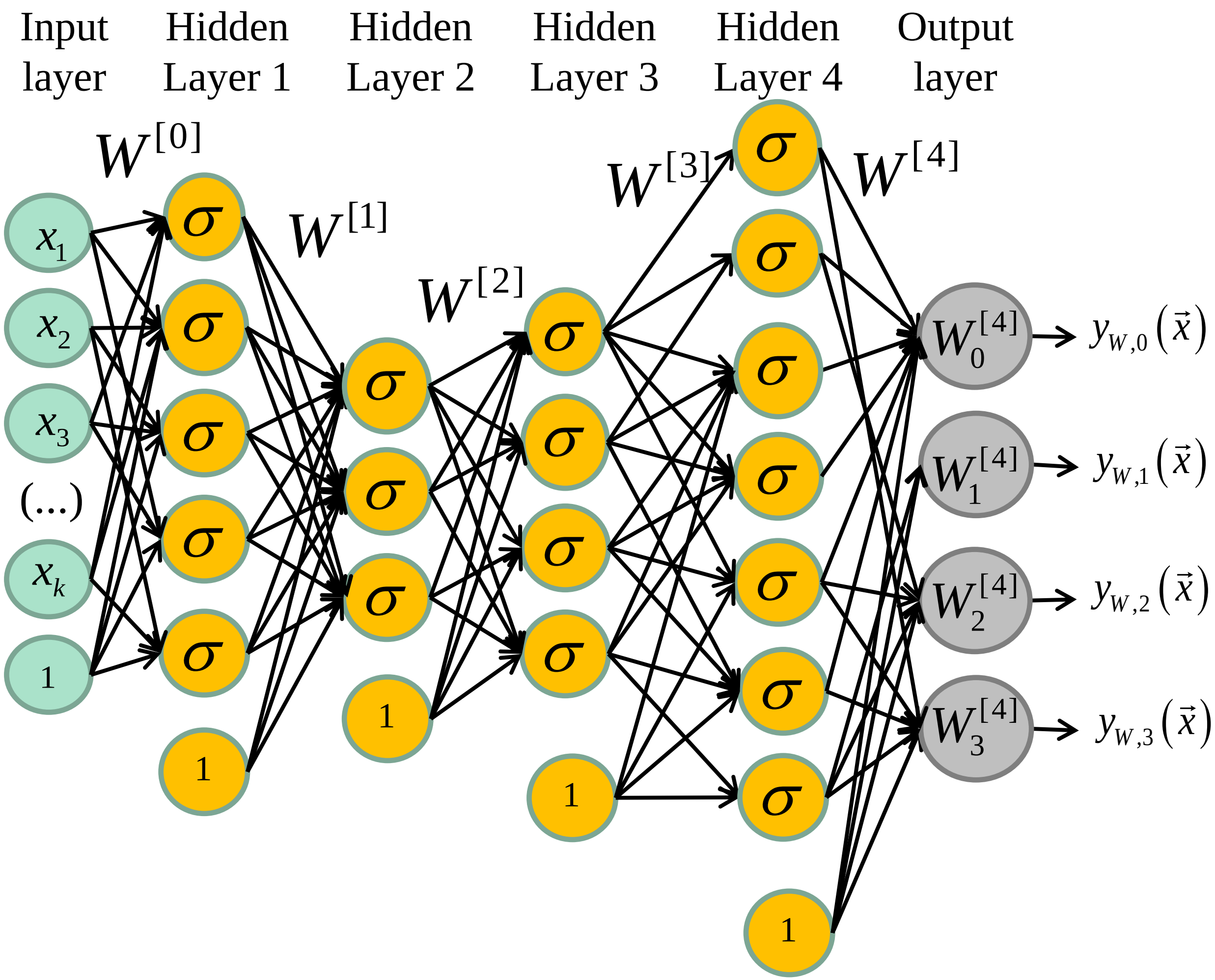
Output of the last layer

Output of the network

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



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

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

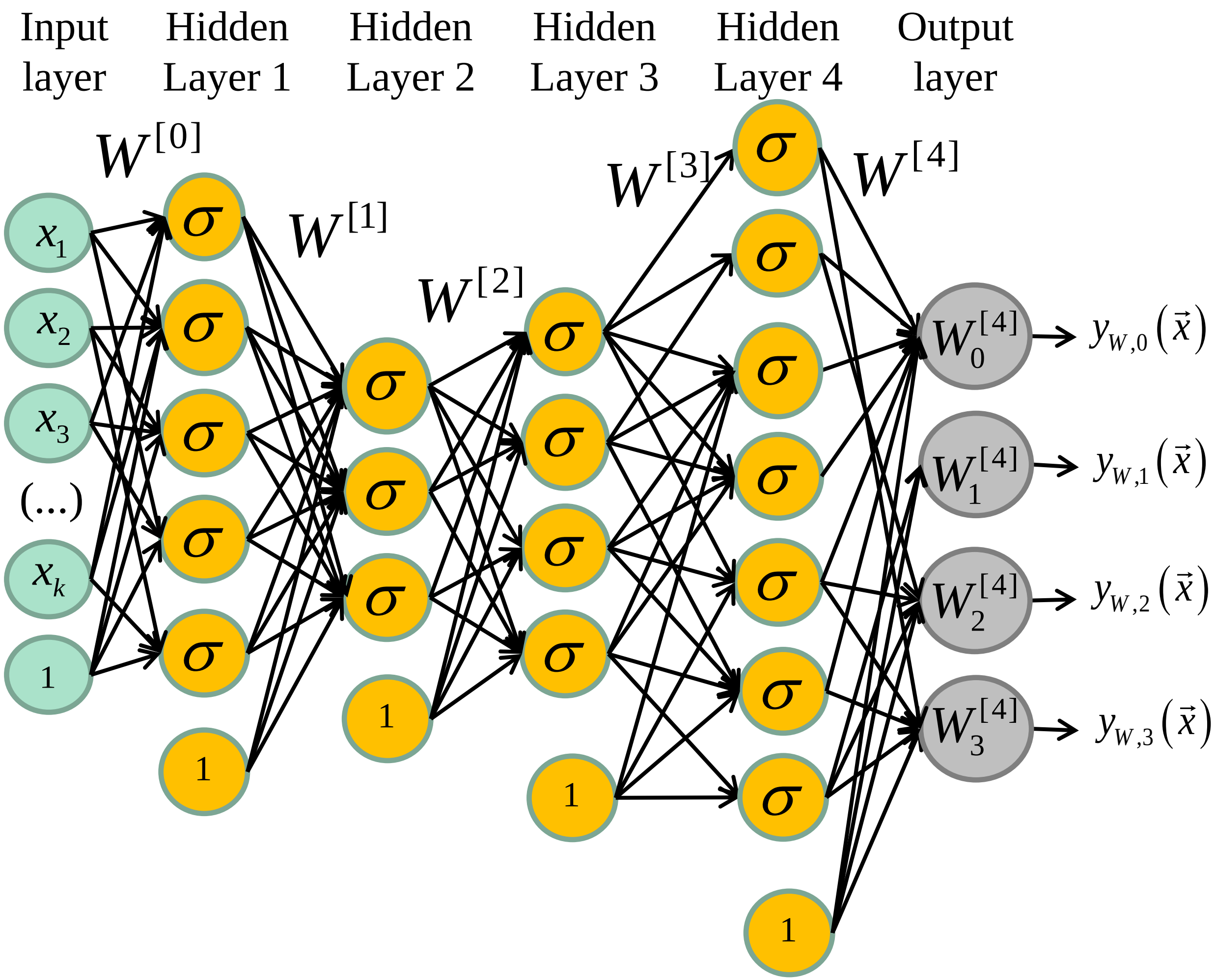
$$W^{[2]} \in R^{4\times 4}$$

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

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

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

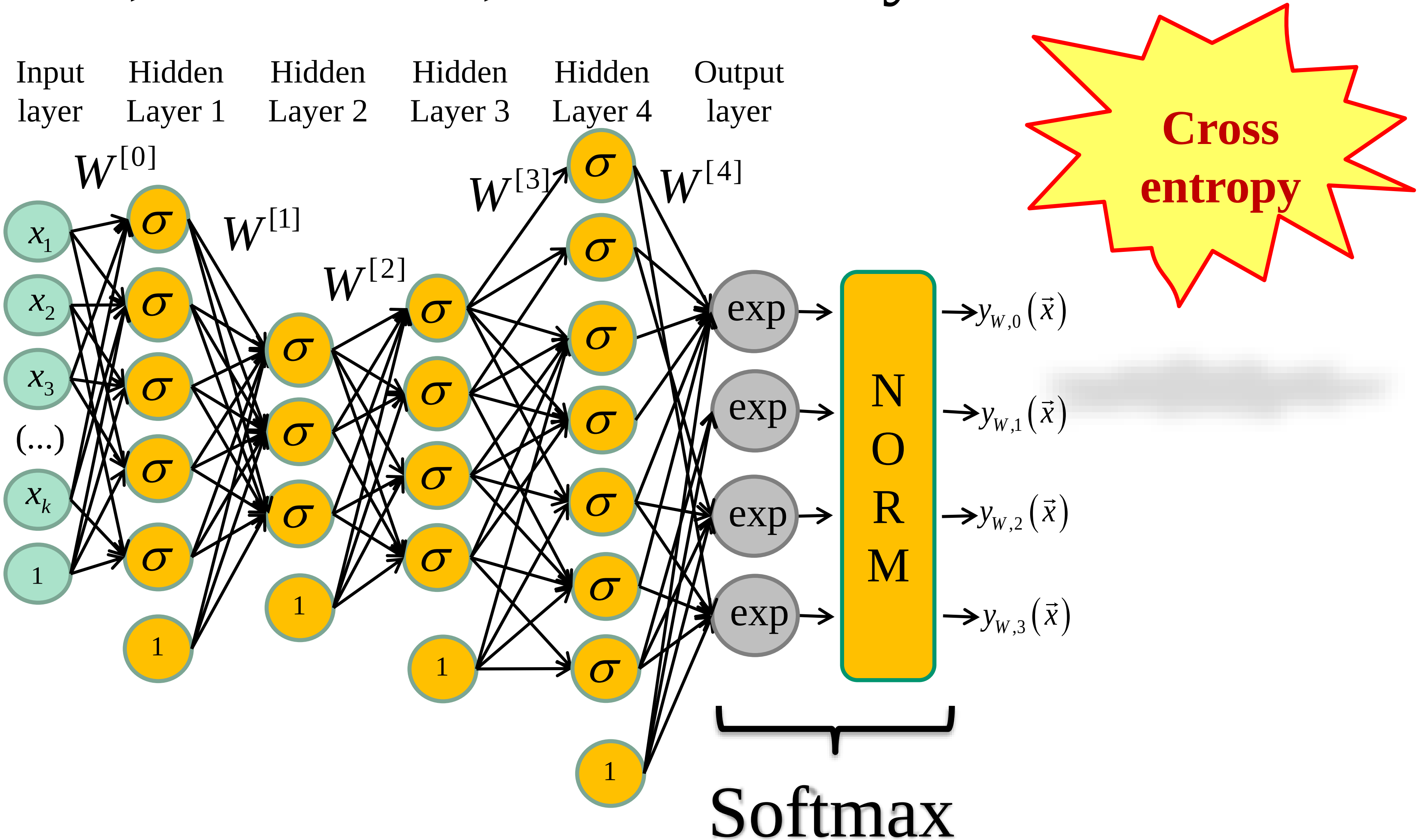
kD, **4 Classes**, 4 hidden layer network



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



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

In conclusion

- Linear classifiers
  - Perceptron
  - Logistic regression
- 2-Class vs K-Class neural nets
- Loss function
  - Hinge Loss
  - Cross-entropy loss
- Gradient descent
- Multi-layer perceptron.

Merci

Extra slides

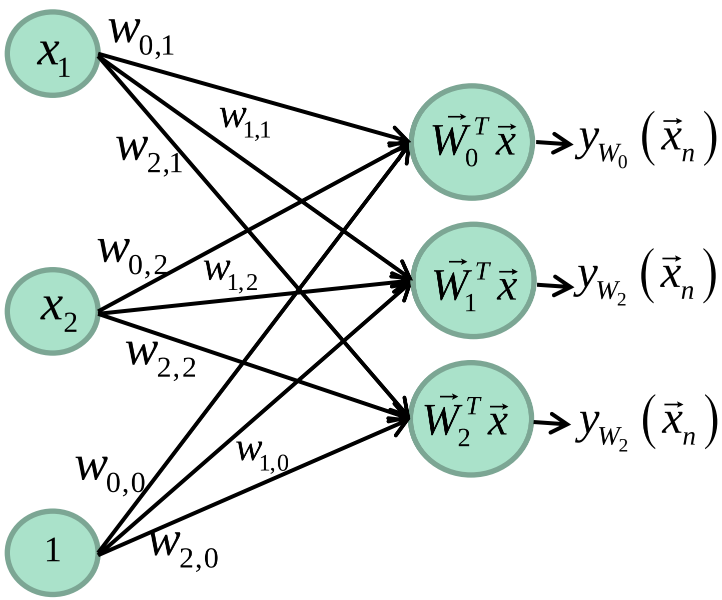
Better understand

Cross entropy vs *Hinge loss*

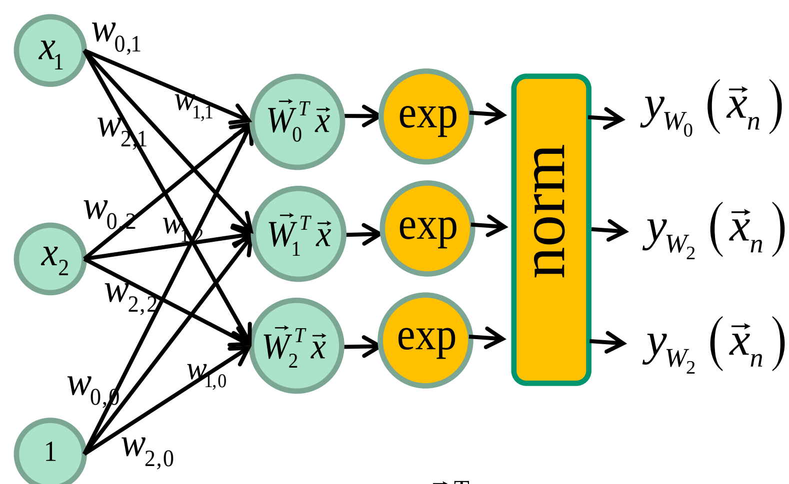
Cross entropy vs Hinge loss

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

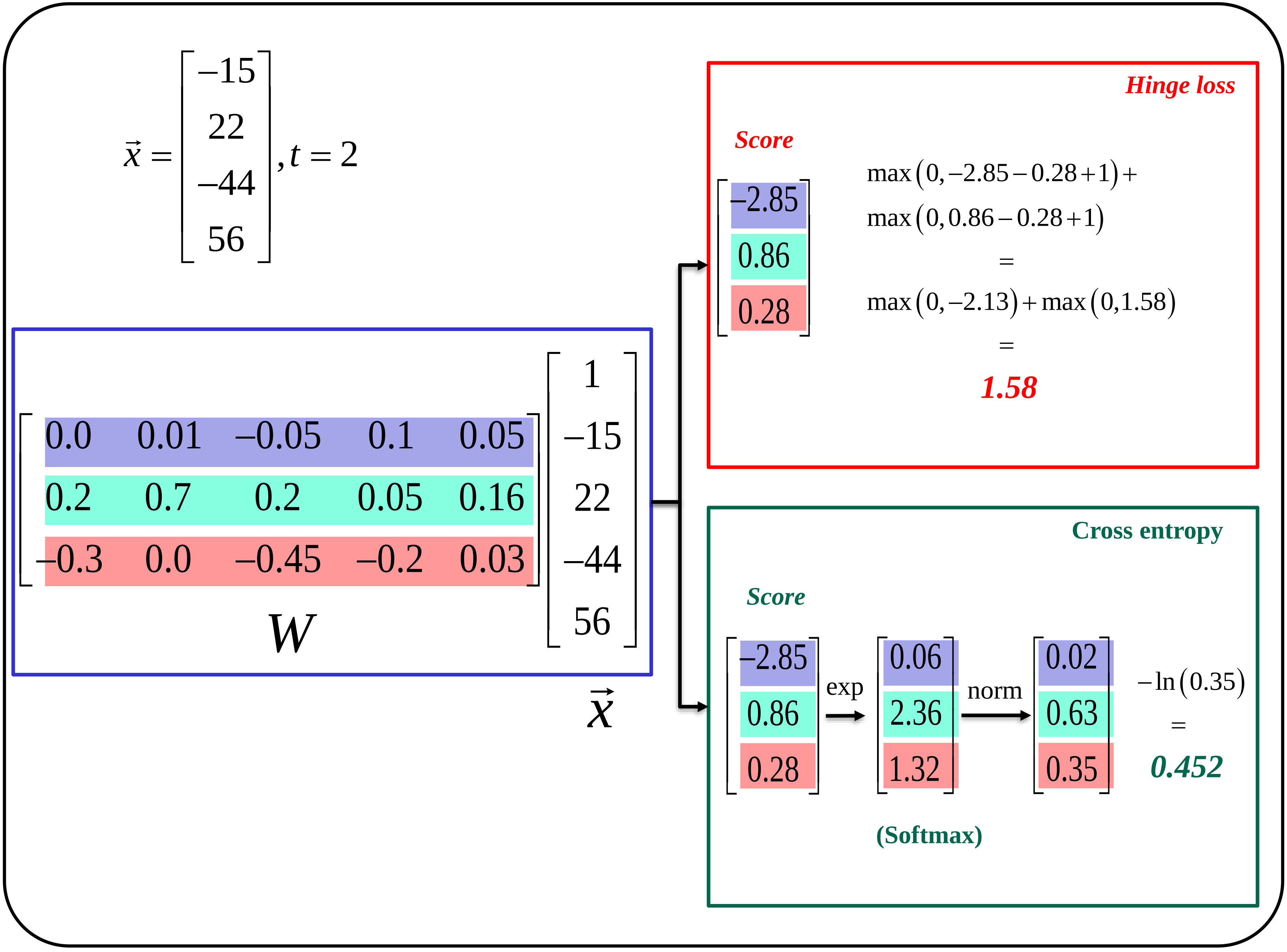
- Hinge loss : output = matrix-vector
- Cross entrpty: sortie = softmax



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



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



$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$
$\begin{bmatrix} 0.0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix} \begin{bmatrix} 1 \\ -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}$
W
$\vec{x}$
Hinge loss
Score
$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$
$\max(0, -2.85 - 0.28 + 1) +$
$\max(0, 0.86 - 0.28 + 1)$
$=$
$\max(0, -2.13) + \max(0, 1.58)$
$=$
1.58
Cross entropy
Score
$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 0.06 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.02 \\ 0.63 \\ 0.35 \end{bmatrix}$
$-\ln(0.35)$
$=$
0.452
(Softmax)

$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

$$\begin{bmatrix} 0.0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix} \begin{bmatrix} 1 \\ -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}$$

Hinge loss

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

$$\max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1) = \max(0, -2.13) + \max(0, 1.58) = \mathbf{1.58}$$

Cross entropy

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 0.06 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.02 \\ 0.63 \\ 0.35 \end{bmatrix}$$

(Softmax)

$$-\ln(0.35) = \mathbf{0.452}$$

Q1: What happens if we increase the score of class 0(-2.85)?

$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

$$\begin{bmatrix} 0.0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix} \begin{bmatrix} 1 \\ -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}$$

Hinge loss

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

$$\max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1) = \max(0, -2.13) + \max(0, 1.58) = \boldsymbol{1.58}$$

Cross entropy

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 0.06 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.02 \\ 0.63 \\ 0.35 \end{bmatrix} \quad -\ln(0.35) = \boldsymbol{0.452}$$

(Softmax)

Q2: What happens if we increase the score of class 1(0.86)?

$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

$$\begin{bmatrix} 0.0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix} \begin{bmatrix} 1 \\ -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}$$

Hinge loss

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

$$\max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1) = \max(0, -2.13) + \max(0, 1.58) = \mathbf{1.58}$$

Cross entropy

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 0.06 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.02 \\ 0.63 \\ 0.35 \end{bmatrix}$$

(Softmax)

$$-\ln(0.35) = \mathbf{0.452}$$

Q3: what is the MIN/MAX values of those two losses?

$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

$$\begin{bmatrix} 0.0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.7 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix} \begin{bmatrix} 1 \\ -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}$$

Hinge loss

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

$$\max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1) = \max(0, -2.13) + \max(0, 1.58) = \mathbf{1.58}$$

Cross entropy

Score

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 0.06 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.02 \\ 0.63 \\ 0.35 \end{bmatrix}$$

(Softmax)

$$-\ln(0.35) = \mathbf{0.452}$$

Q4: what would happed to the total loss if we were to add an L2 regularization?

Increase the number of neurons

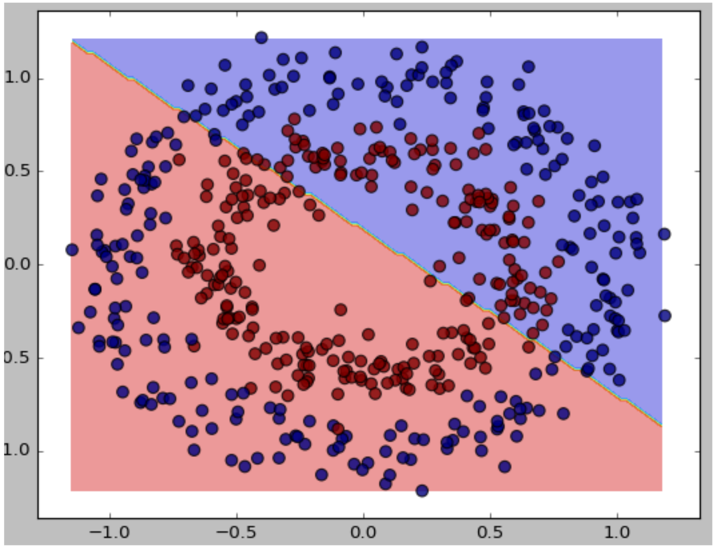
Increase the number of layers

Increase the capacity of the network

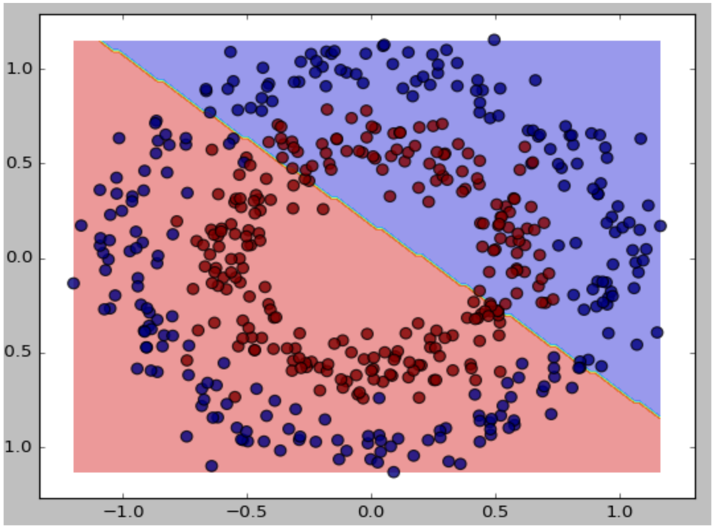
image by www.slon.pics - Freepik.com

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

Under-fitting

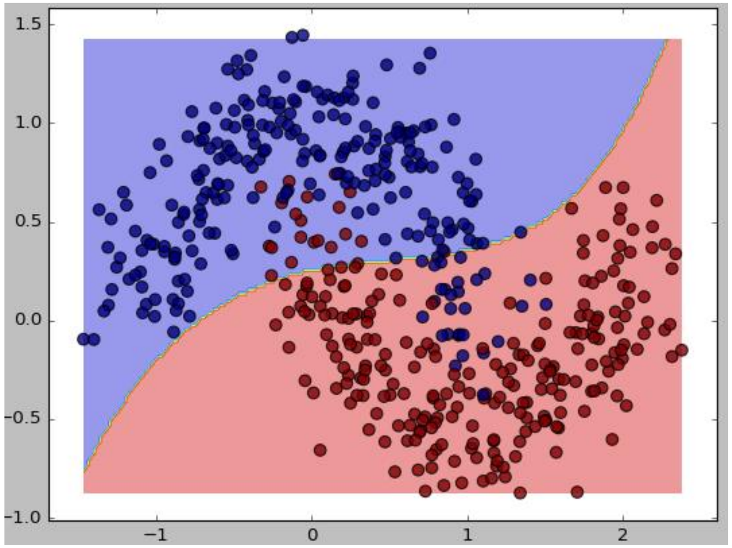


Precision on the training set = 52.2%

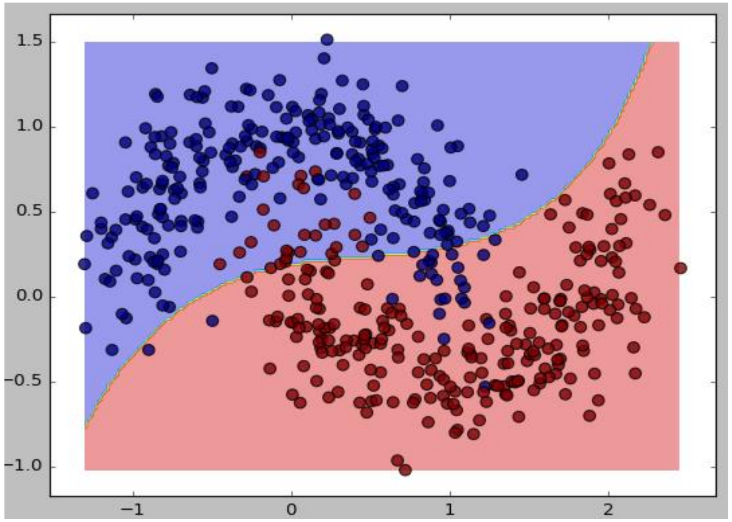


Precision on the test set = 51.2%

Could do better…

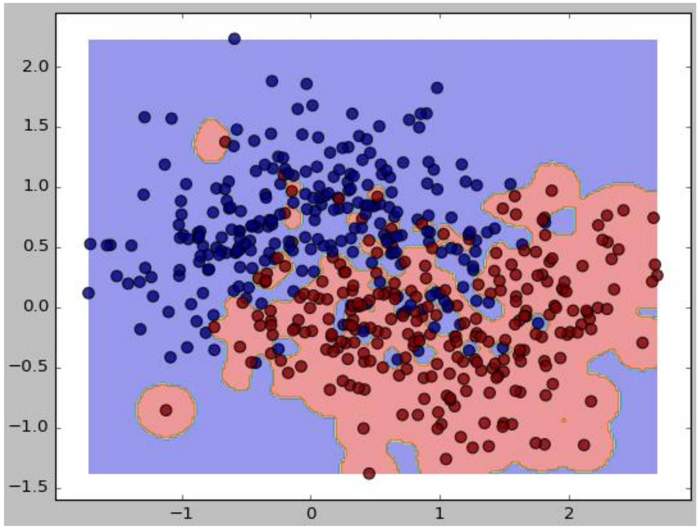


Precision on the training set = 82%

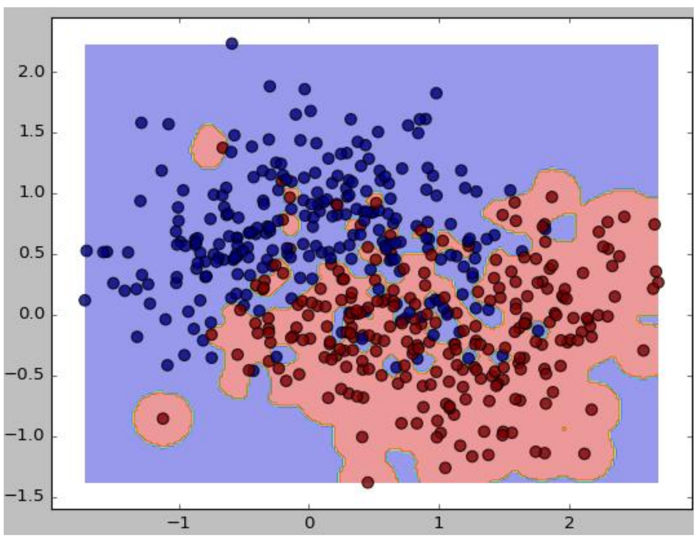


Precision on the test set = 80%

Overfitting

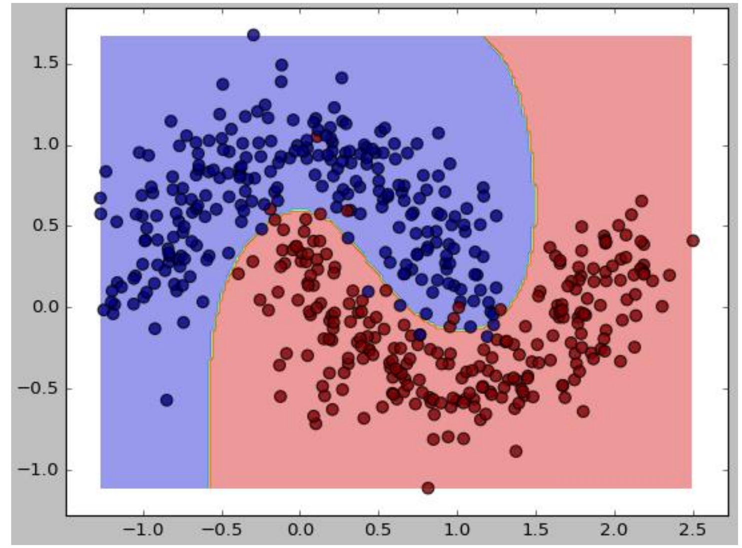


Precision on the training set = 99.6%

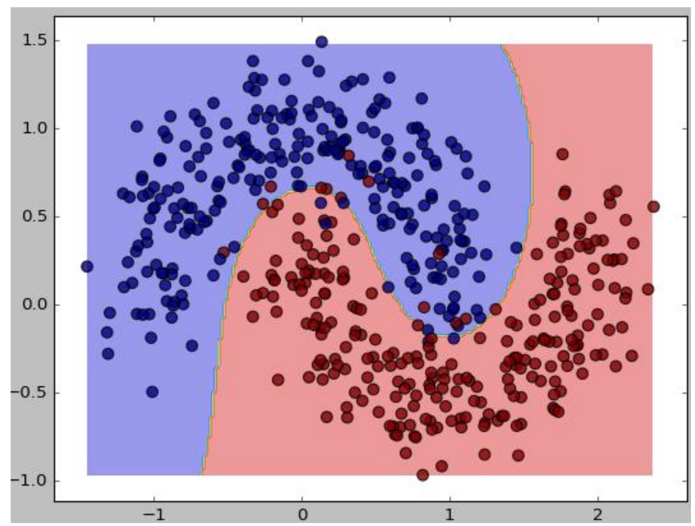


Precision on the test setb= 78%

SUPER !!!



Precision on the training set =97.8%

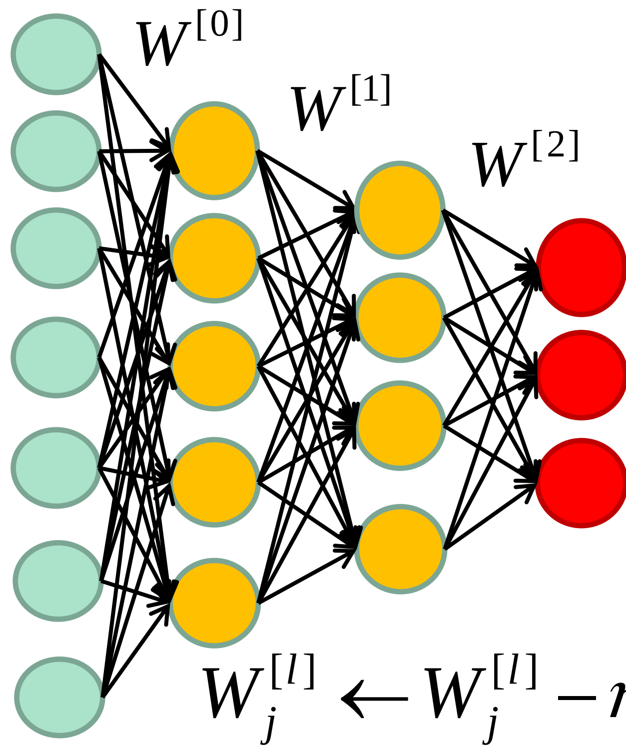


Precision on the test set = 96.2%

kD, 2 Classes, 4 hidden layer network

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

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



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

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

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

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

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

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

$$W_j^{[l]} \leftarrow W_j^{[l]} - \eta \frac{\partial\left(L\left(y_W(\vec{x})\right)\right)}{\partial W_j^{[l]}}$$

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