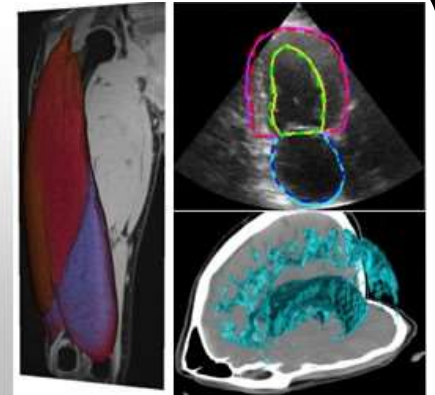


Deep learning for medical imaging school 2021

April 19—24 2021



Virtual edition



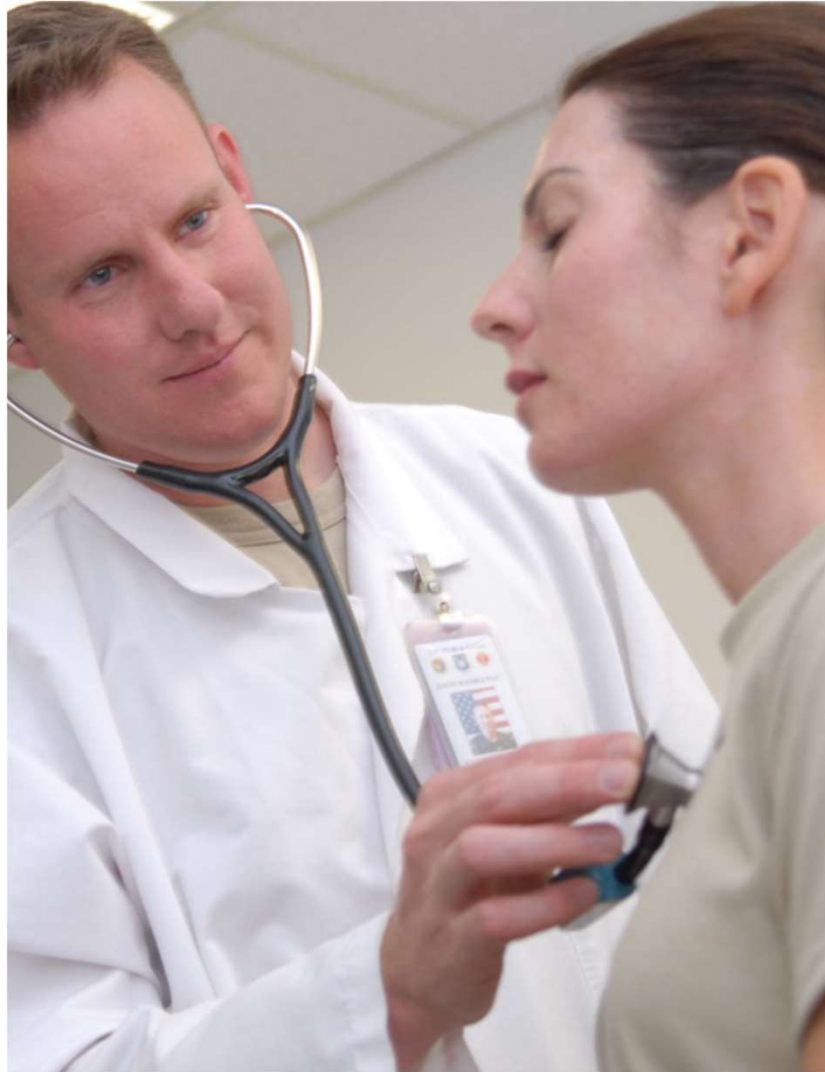
Basics of deep learning part 1

By

Pierre-Marc Jodoin

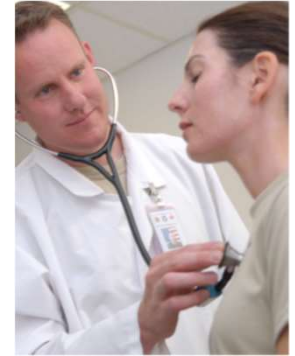


Lets start with a simple example



From Wikimedia Commons
the free media repository

Lets start with a simple example

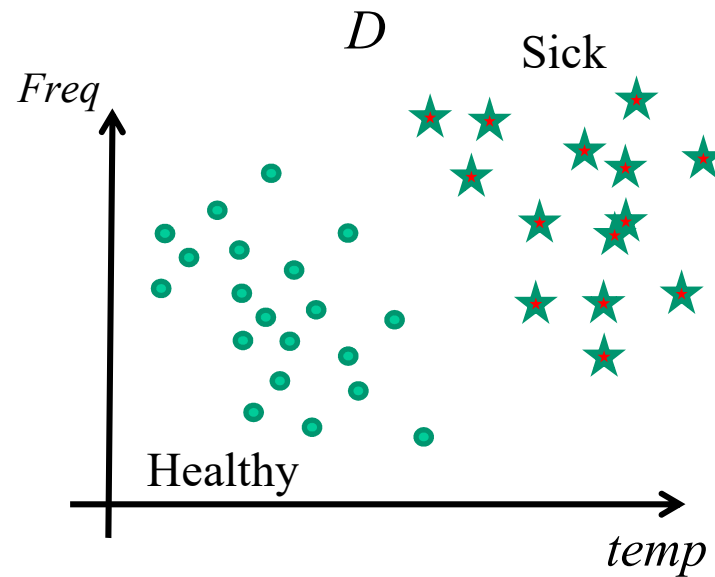


D

	(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

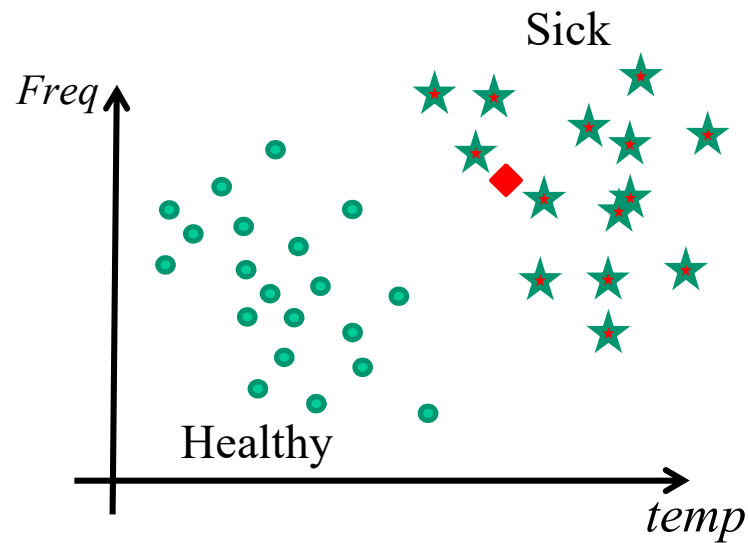


Lets start with a simple example

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



From Wikimedia Commons
the free media repository

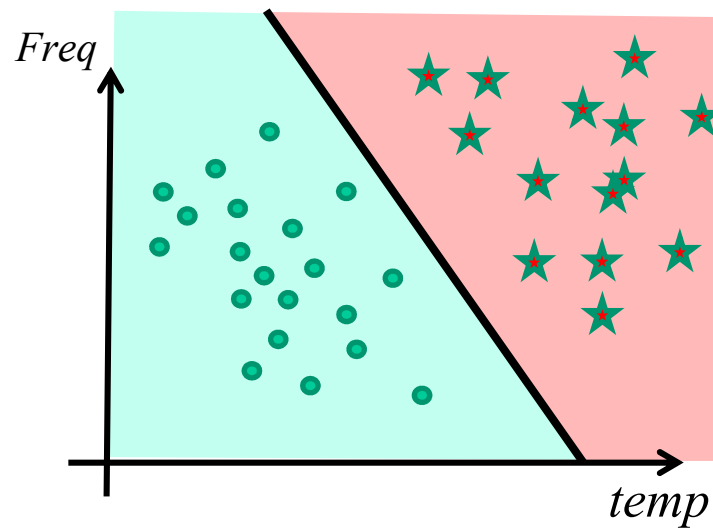


Solution



From Wikimedia Commons
the free media repository

Divide the feature space in 2 regions : **sick** and **healthy**

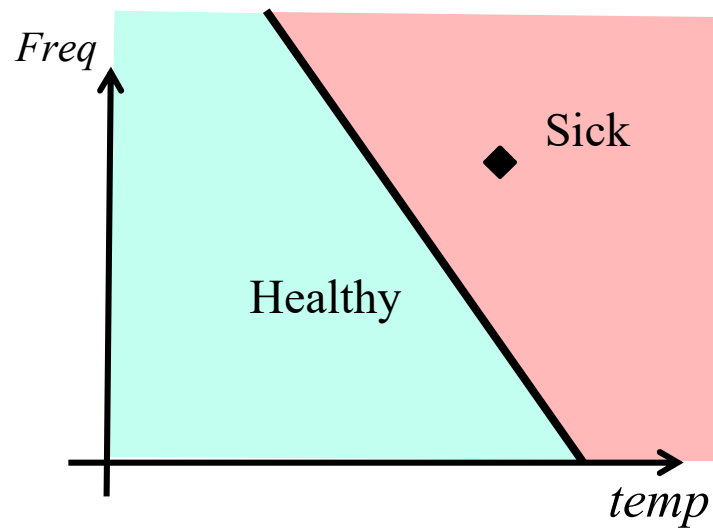


Solution



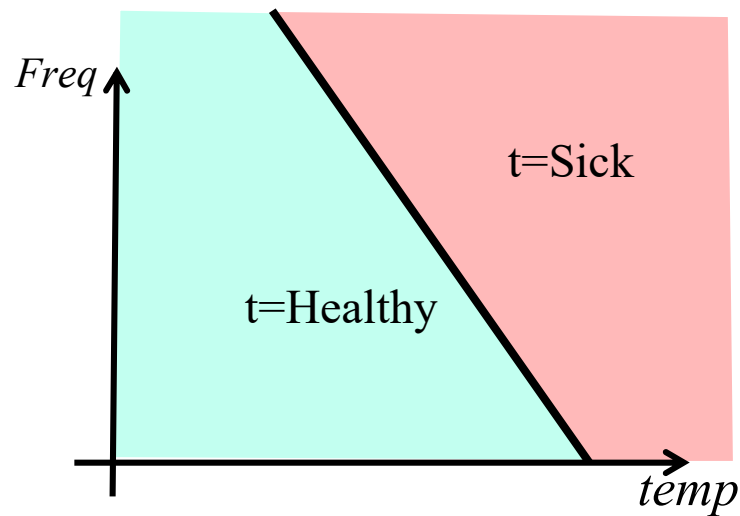
From Wikimedia Commons
the free media repository

Divide the feature space in 2 regions : **sick** and **healthy**



More formally

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

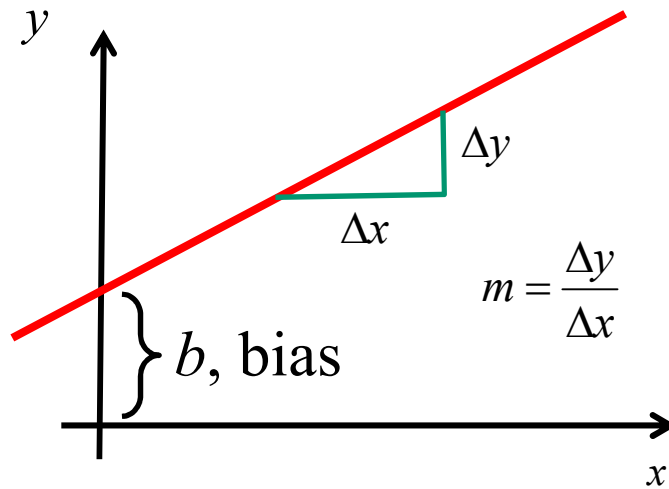


From Wikimedia Commons
the free media repository

How to split
the feature
space?



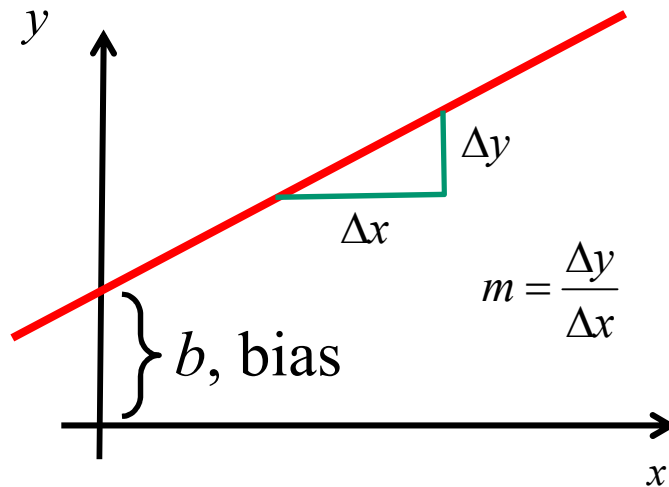
Definition ... a line!



$$y = mx + b$$

slope \uparrow bias \uparrow

Definition ... a line!



$$y = mx + b$$

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

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

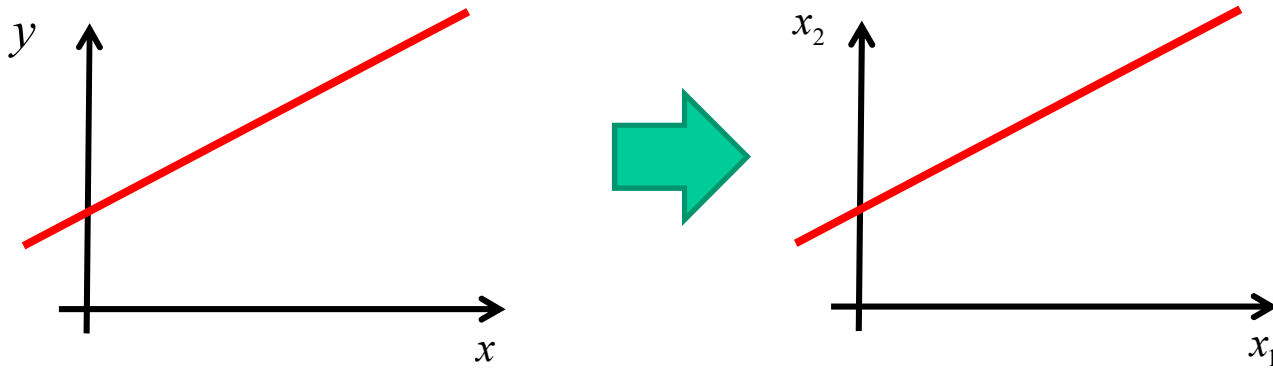
$$0 = \Delta yx - \Delta xy + b\Delta x$$

Rename variables

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

Rename variables

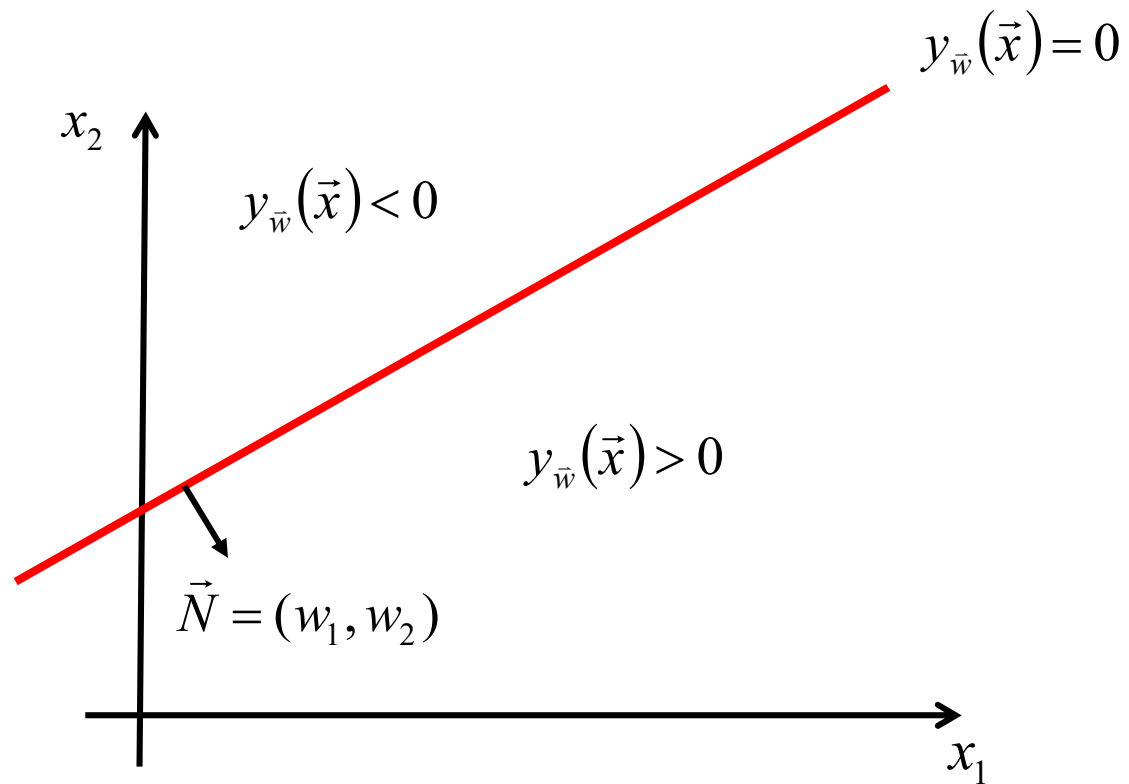
$$0 = w_1x + w_2y + w_0$$



$$0 = w_1x_1 + w_2x_2 + w_0$$

Classification function

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



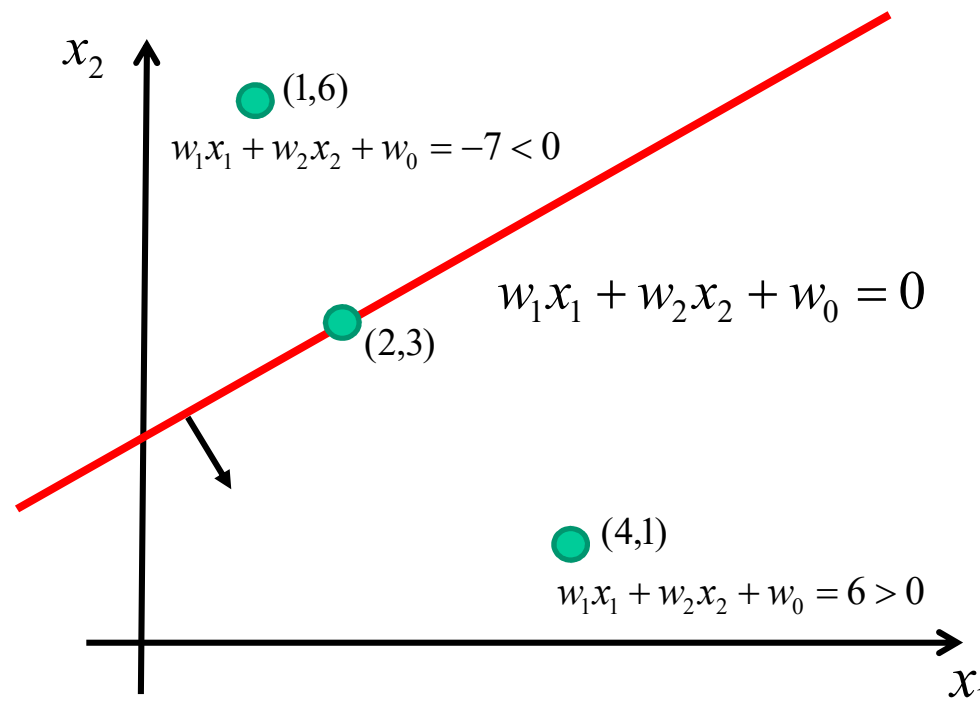
Classification function

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

$$w_1 = 1.0$$

$$w_2 = -2.0$$

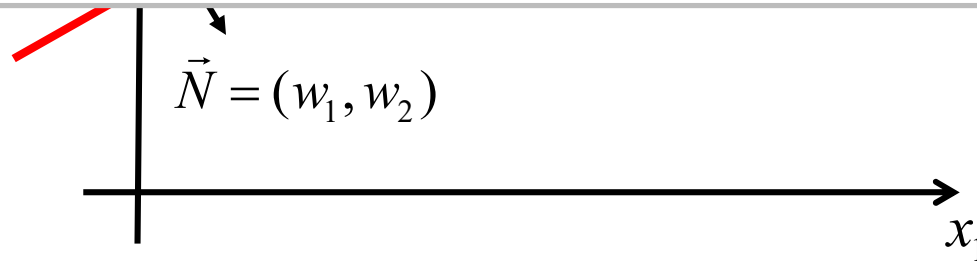
$$w_0 = 4.0$$



Classification function

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

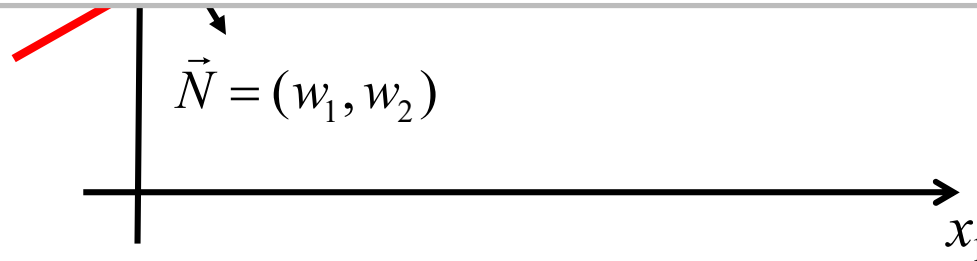
**DOT
product**



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

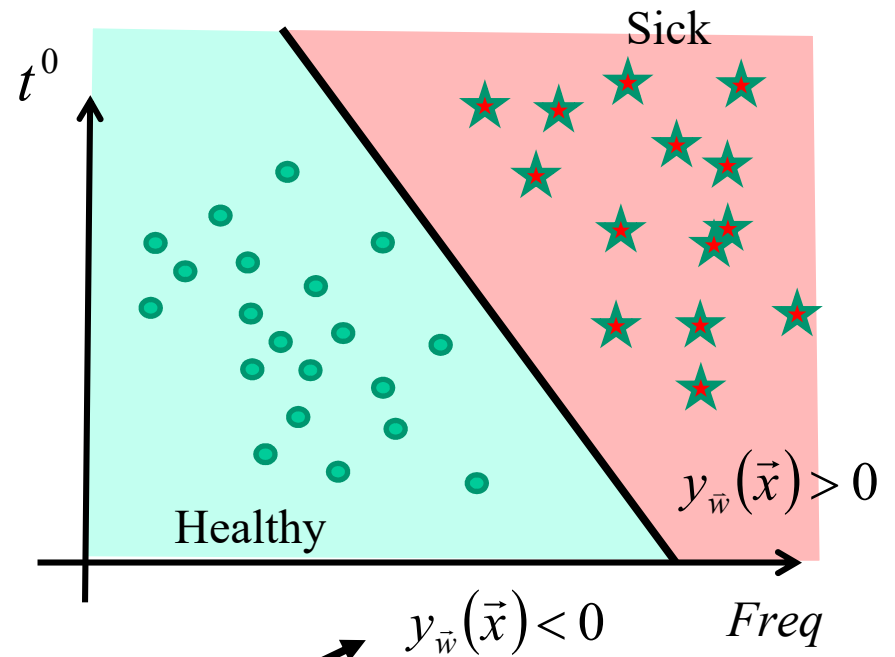
$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x}$$

Learning

With the **training dataset** D

the GOAL is to

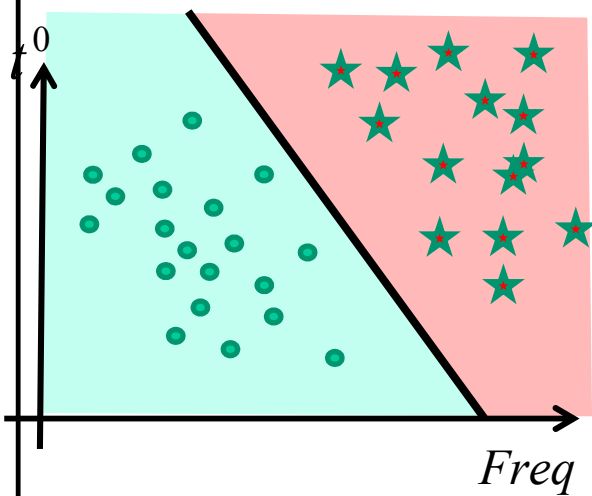
find the parameters (w_0, w_1, w_2) that would best separate the two classes.



**How do we know
if a model is good?**

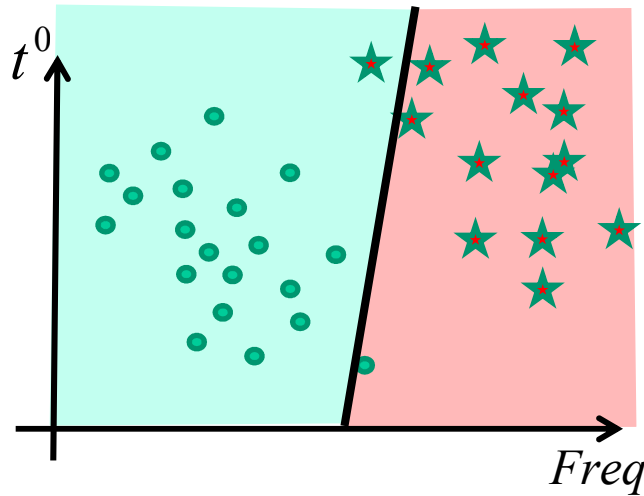


Loss function



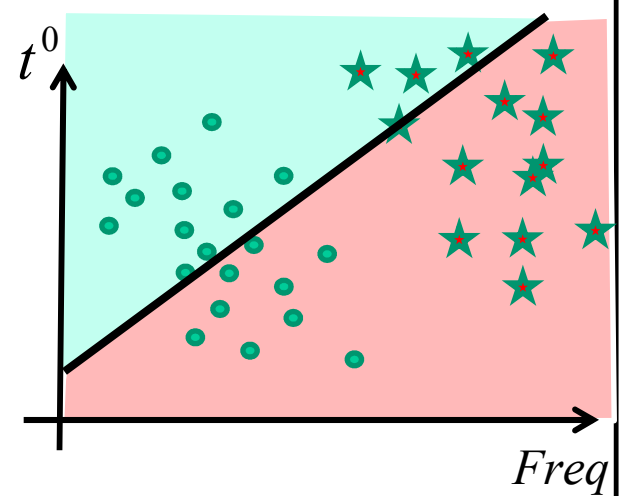
$$L(y_{\bar{w}}(\bar{x}), D) \approx 0$$

Good!



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

Medium



$$L(y_{\bar{w}}(\bar{x}), D) \gg 0$$

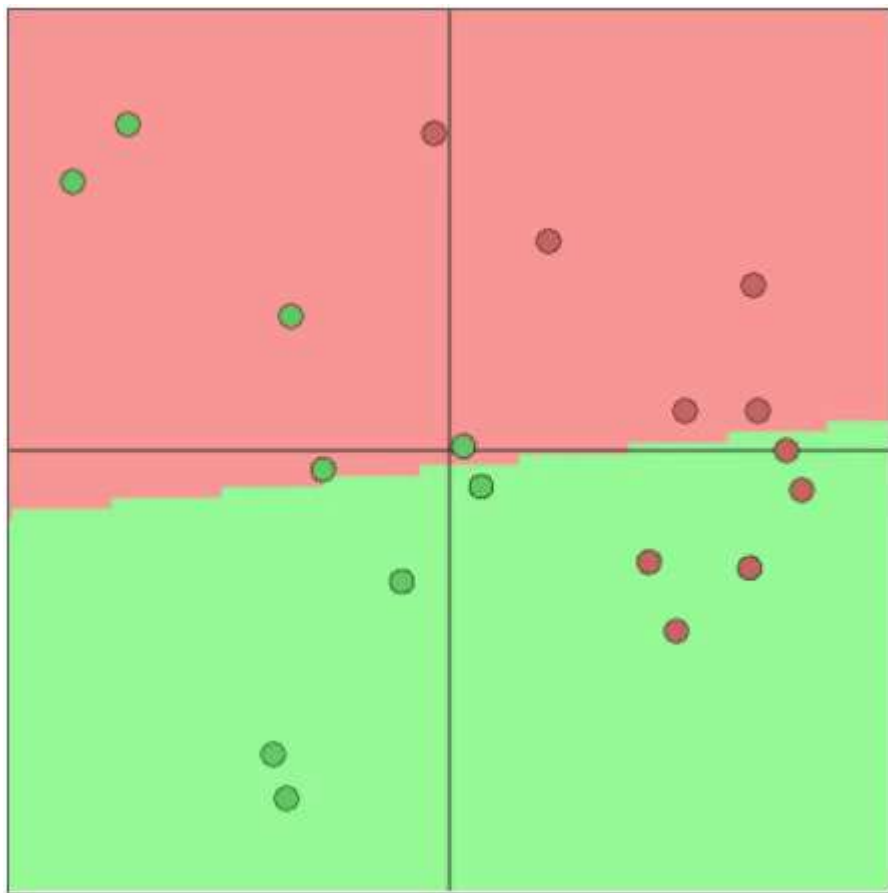
BAD!

Training a machine

Finding the right parameters w_0, w_1, w_2 such that

PATIENTS are WELL CLASSIFIED

SMALL LOSS



So far...

1. Training dataset: D
2. Classification function (a line in 2D) : $y_{\vec{w}}(\vec{x}) = w_1x_1 + w_2x_2 + w_0$
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D)$



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

Today

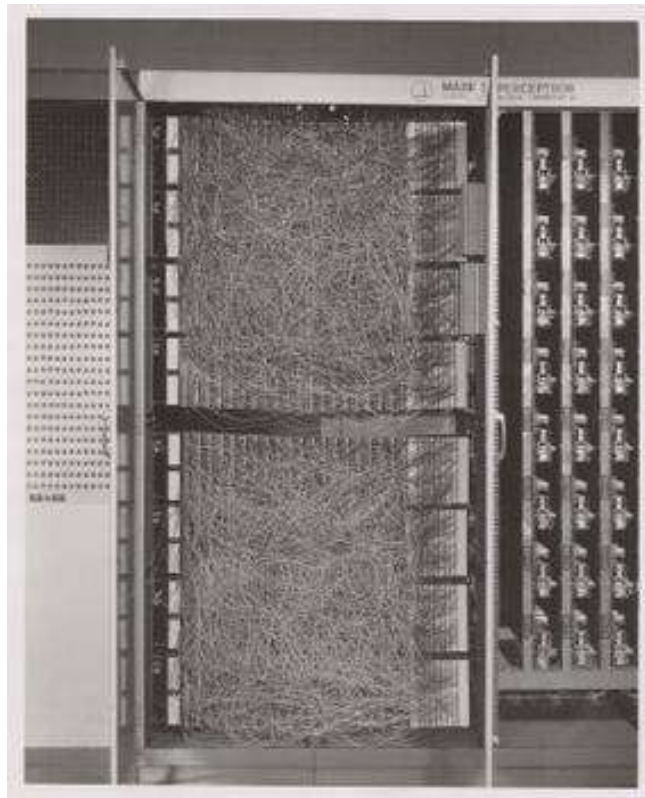


Perceptron

Logistic regression

Multi-layer perceptron

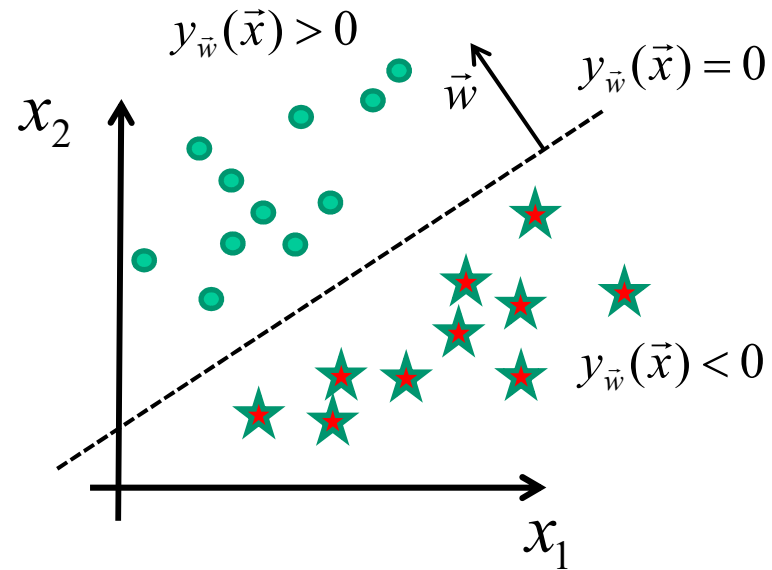
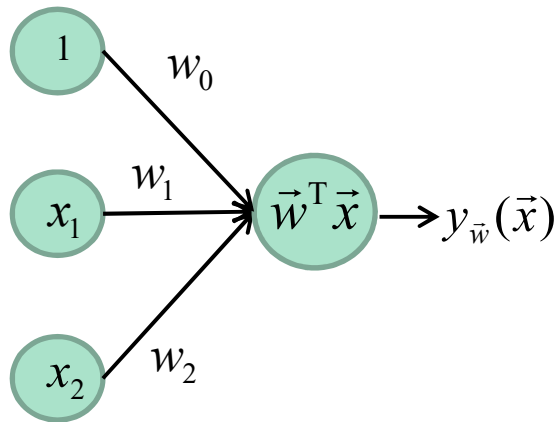
Perceptron



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

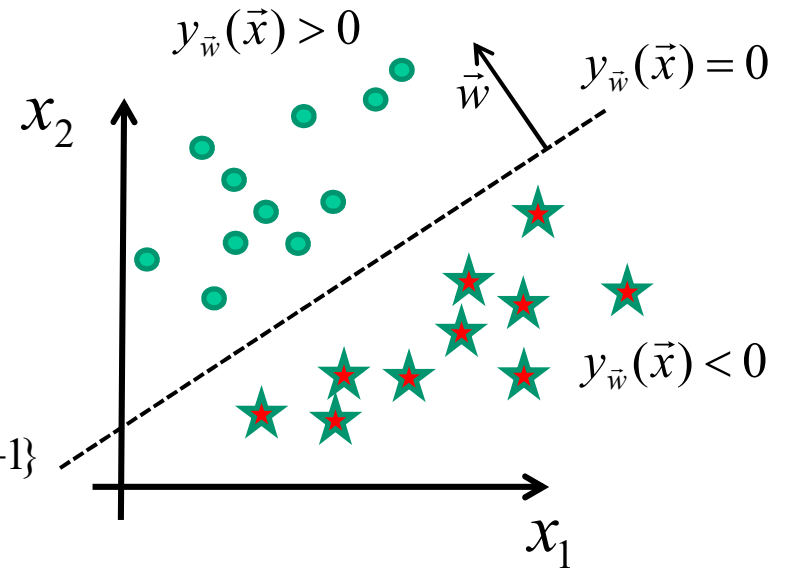
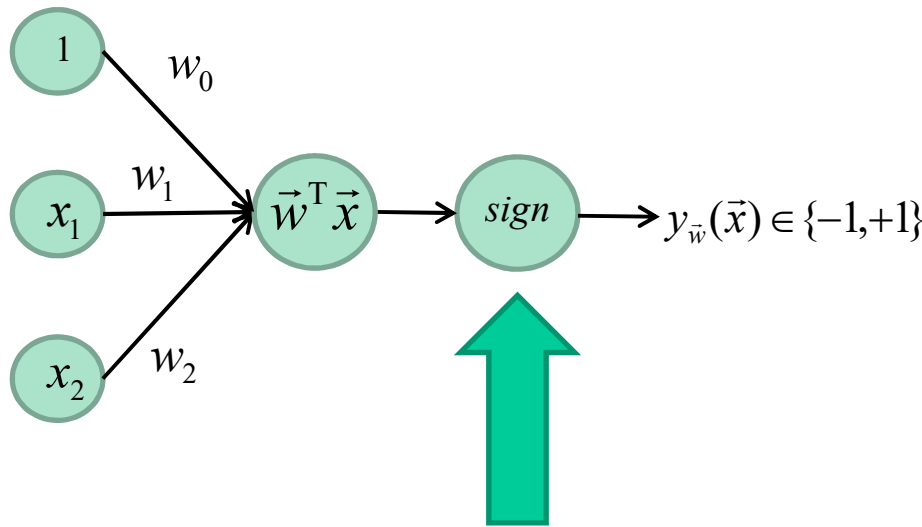
(2D and 2 classes)



$$\begin{aligned} y_{\vec{w}}(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_1 \\ &= \vec{w}^T \vec{x} \end{aligned}$$

Perceptron

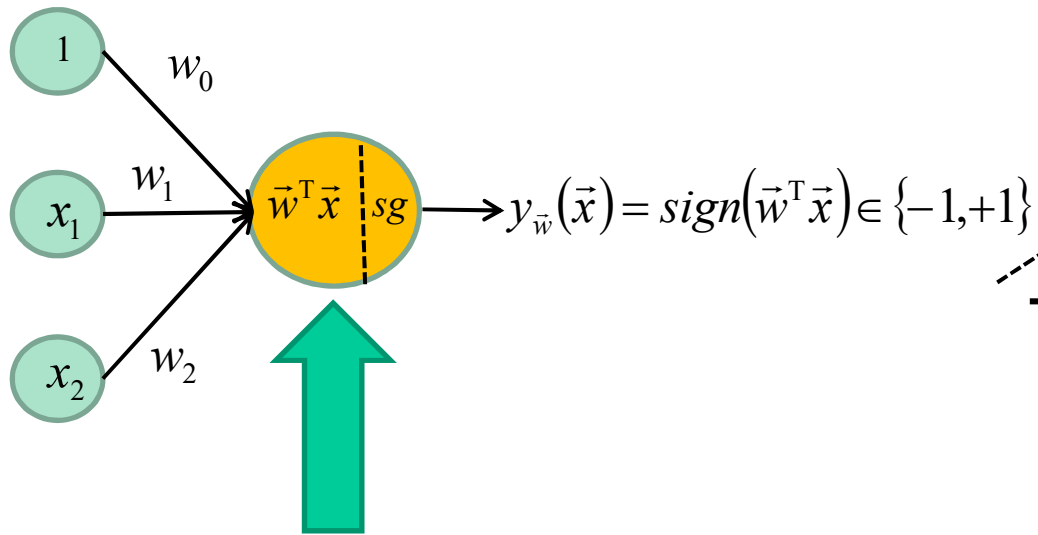
(2D and 2 classes)



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

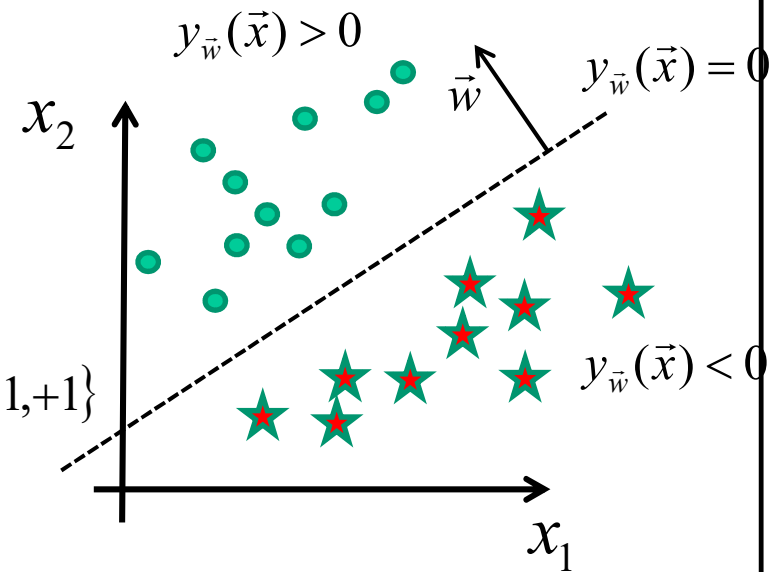
Perceptron

(2D and 2 classes)



Neuron

Dot product + activation function

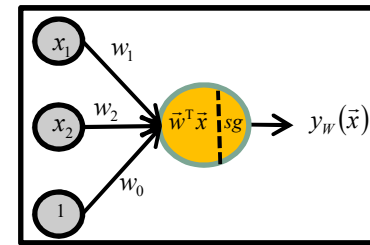


So far...

1. Training dataset: D
2. Classification function (a line in 2D) : $y_{\vec{w}}(\vec{x}) = w_1x_1 + w_2x_2 + w_0$
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: $L(y_{\vec{w}}(\vec{x}), D)$

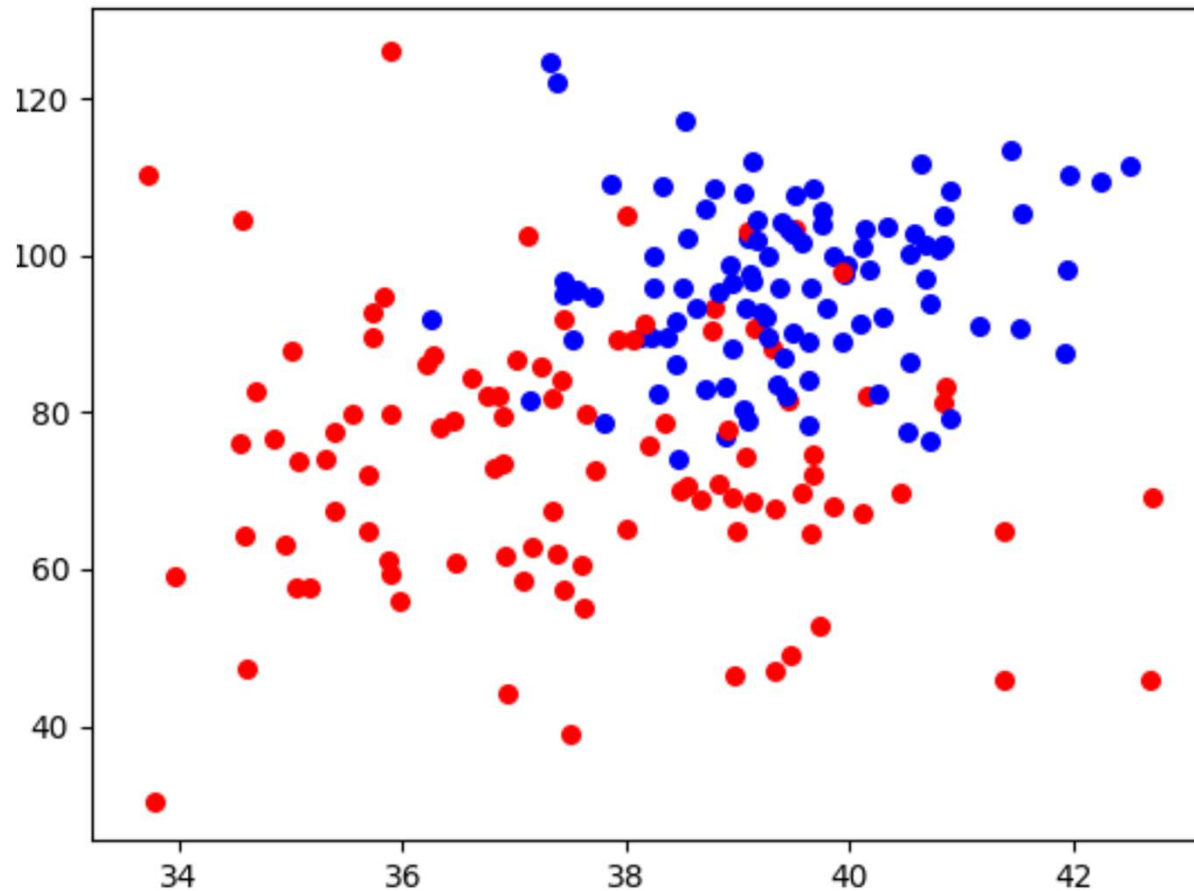


4. Training : find (w_0, w_1, w_2) that minimize $L(y_{\vec{w}}(\vec{x}), D)$

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. Acquire 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

\bar{x}

t



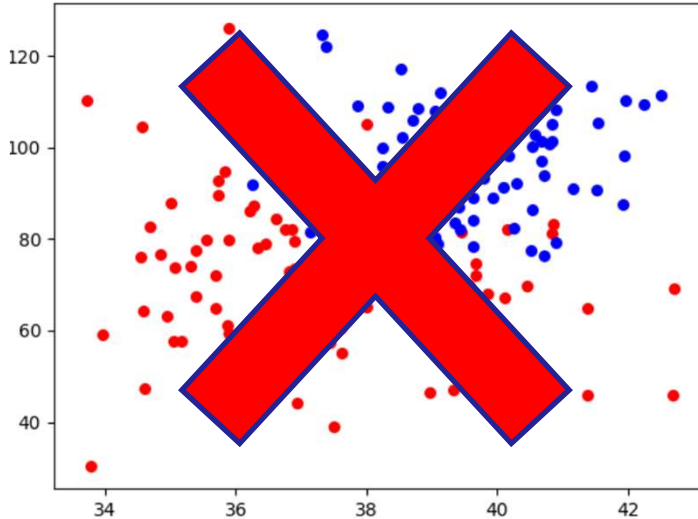
D

	(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

\bar{x}

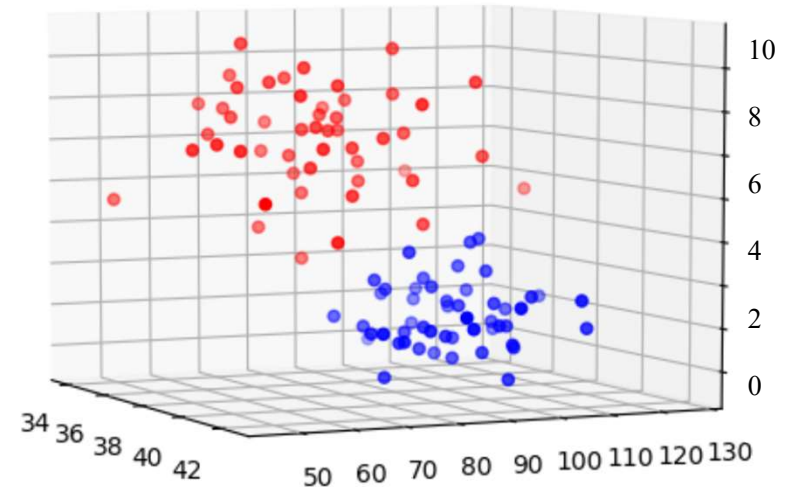
t

Non-linearly separable training data



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

(line)

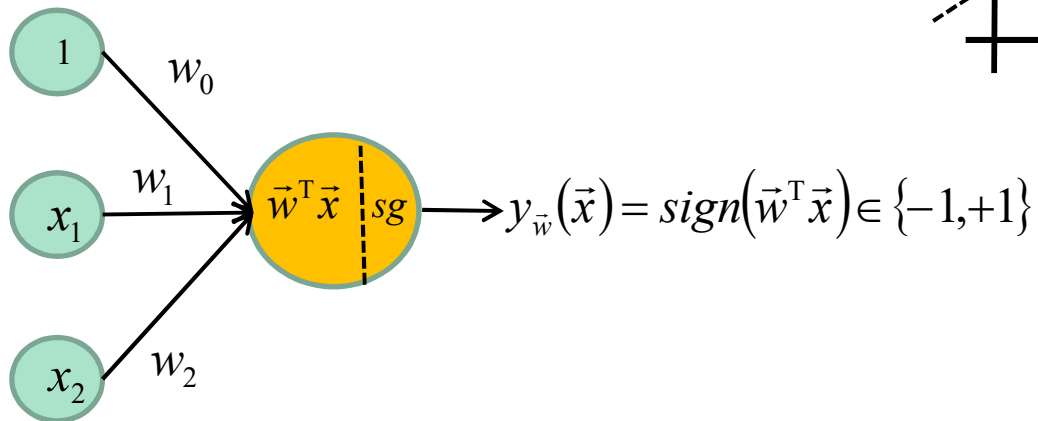
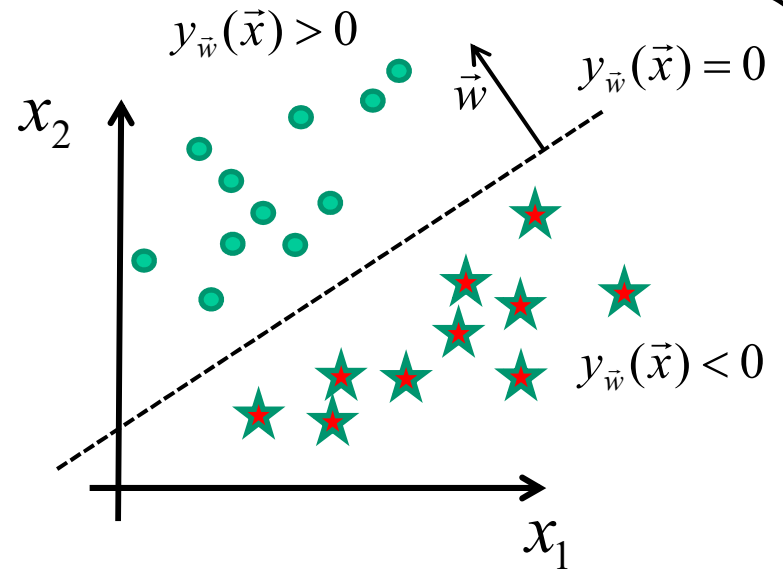


$$y_{\vec{w}}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + w_0$$

(plane)

Perceptron

(2D and 2 classes)

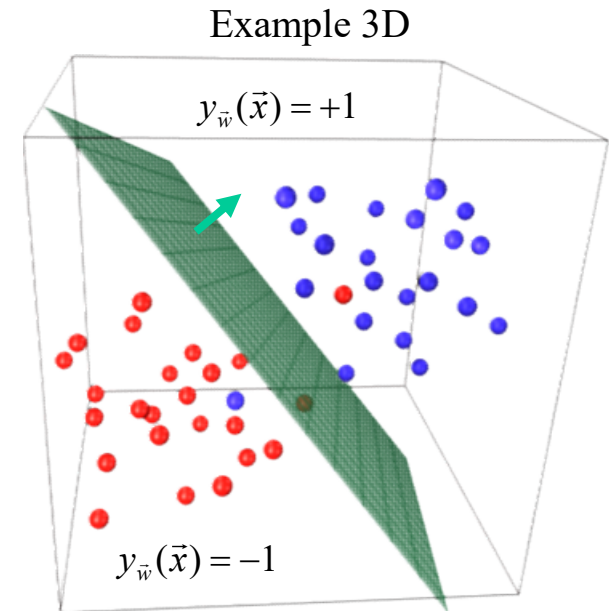
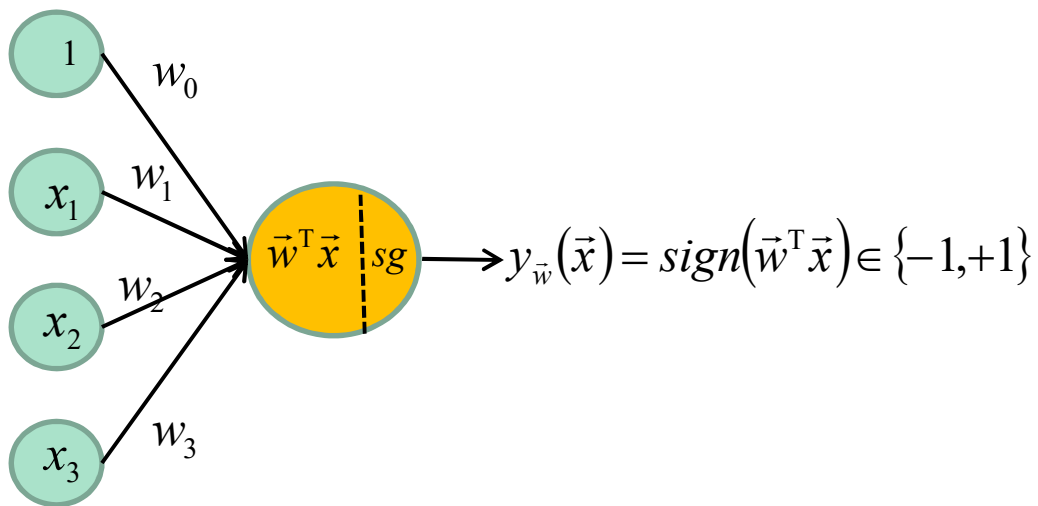


$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

(line)

Perceptron

(3D and 2 classes)

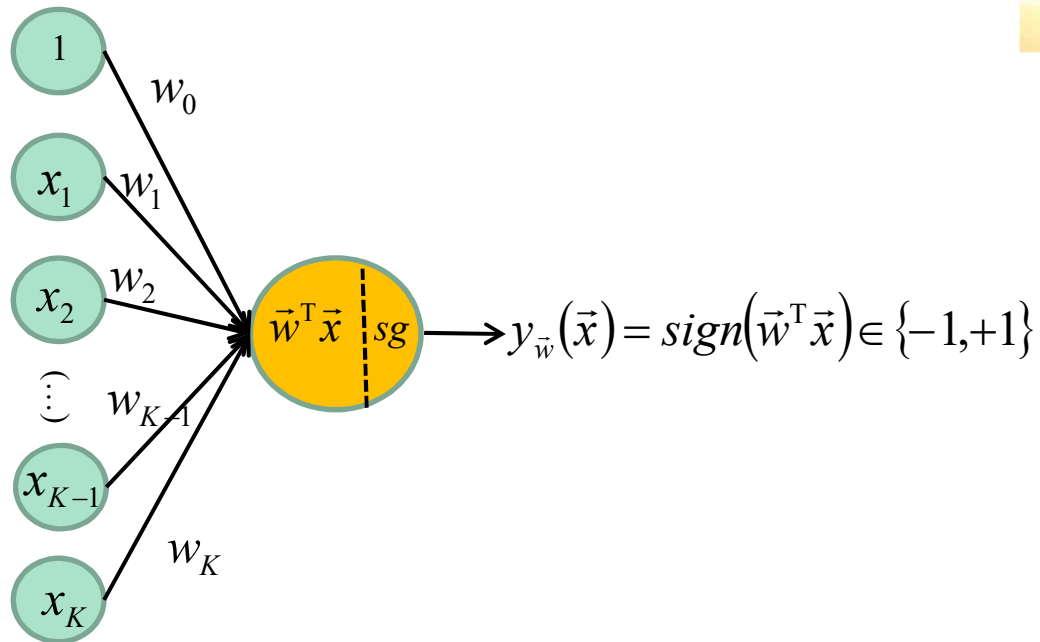


$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0$$

(plane)

Perceptron

(K-D and 2 classes)



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

(hyperplane)

How do we train
a Perceptron?



Learning a machine

The goal: with a set of training data $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \dots, (\vec{x}_N, t_N)\}$, estimate \vec{w} so:

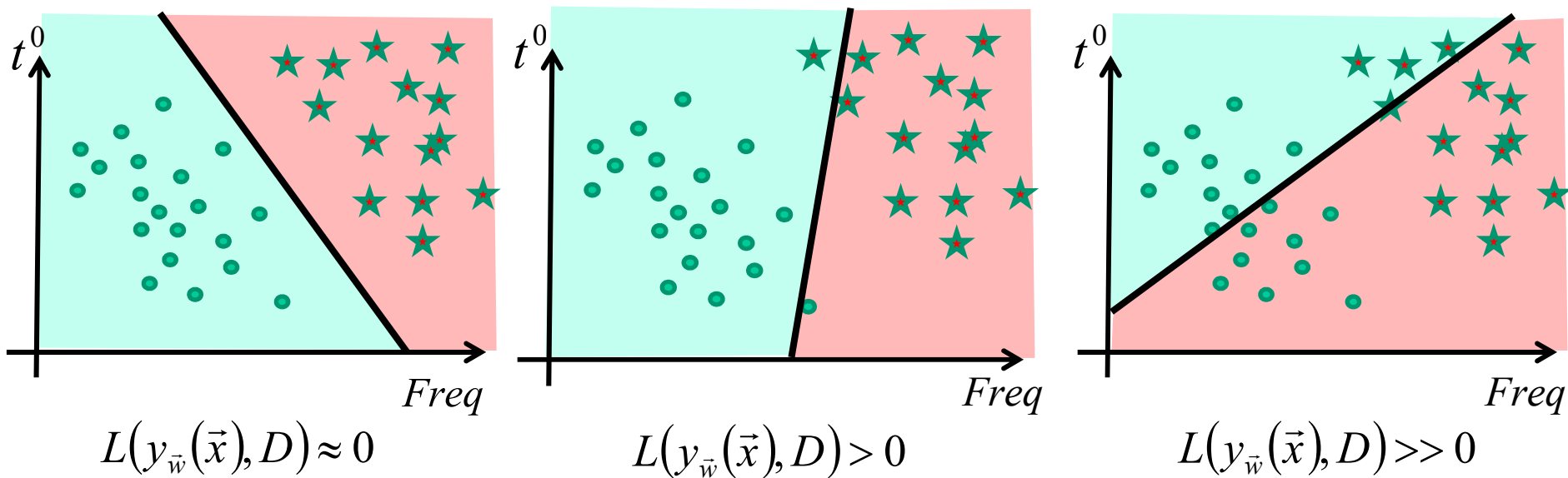
$$y_{\vec{w}}(\vec{x}_n) = t_n \quad \forall n$$

In other words, minimize the **training loss**

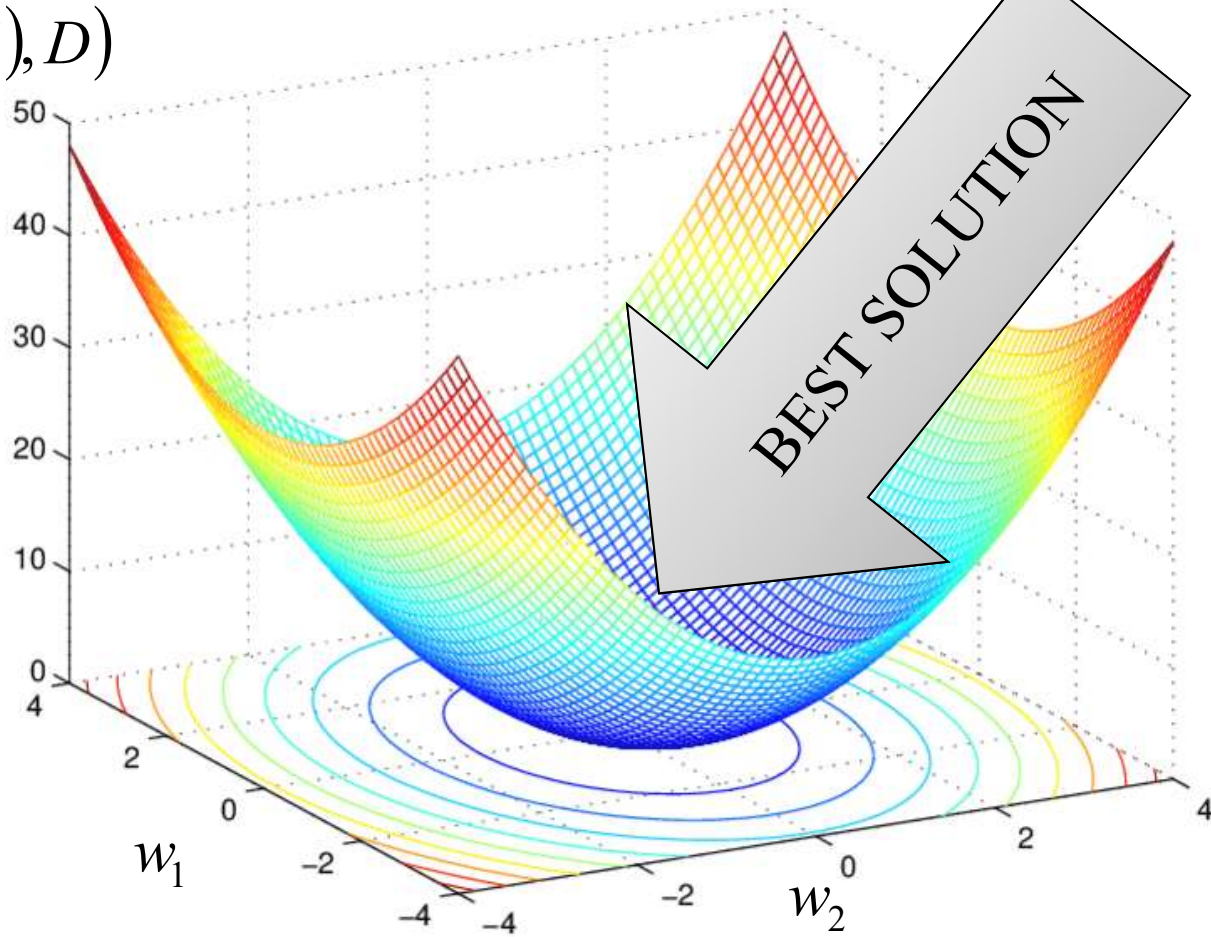
$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N l(y_{\vec{w}}(\vec{x}_n), t_n)$$

Optimization problem

Loss function



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

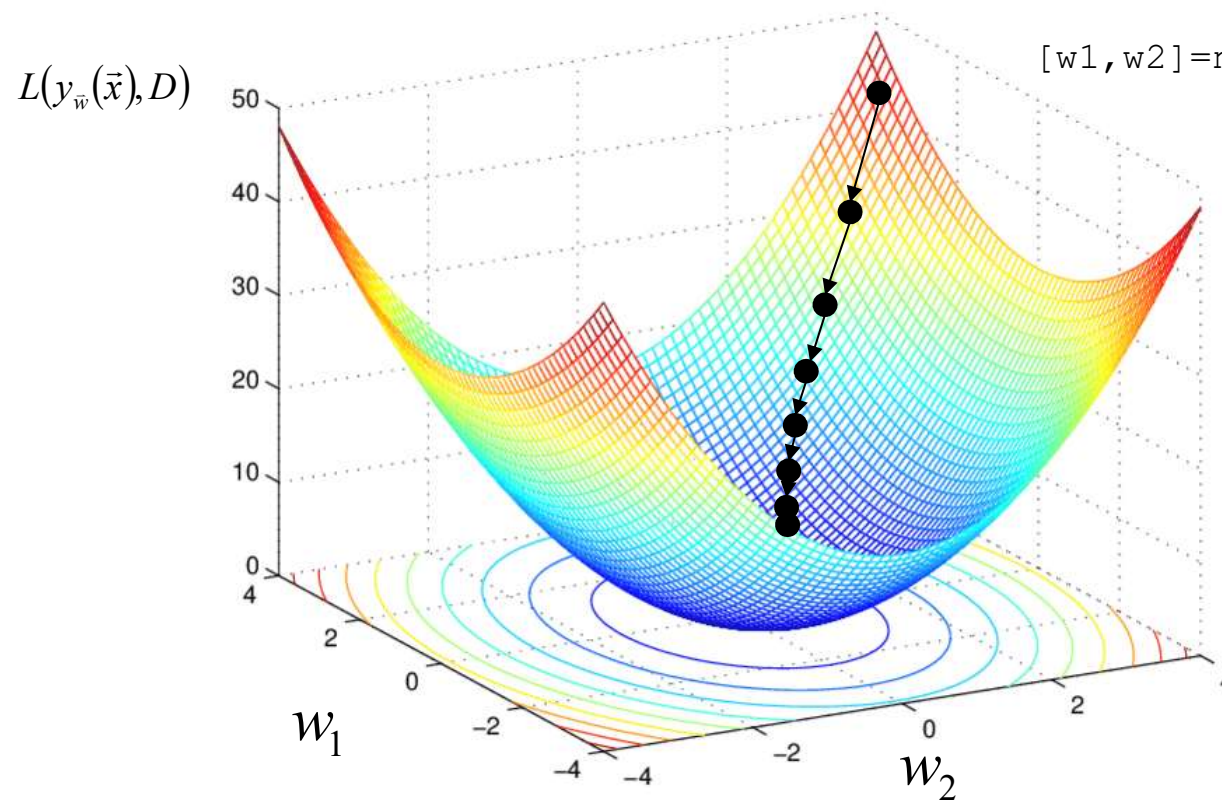


Perceptron

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

Random initialization

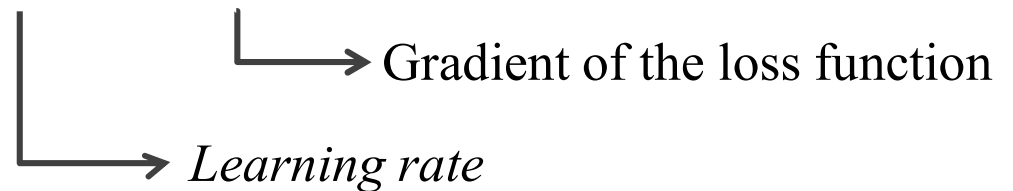
```
[w1, w2] = np.random.randn(2)
```



Gradient descent

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

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

 η Learning rate
 ∇L Gradient of the loss function

Perceptron Criterion (loss)

Observation

A wrongly classified sample is when

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

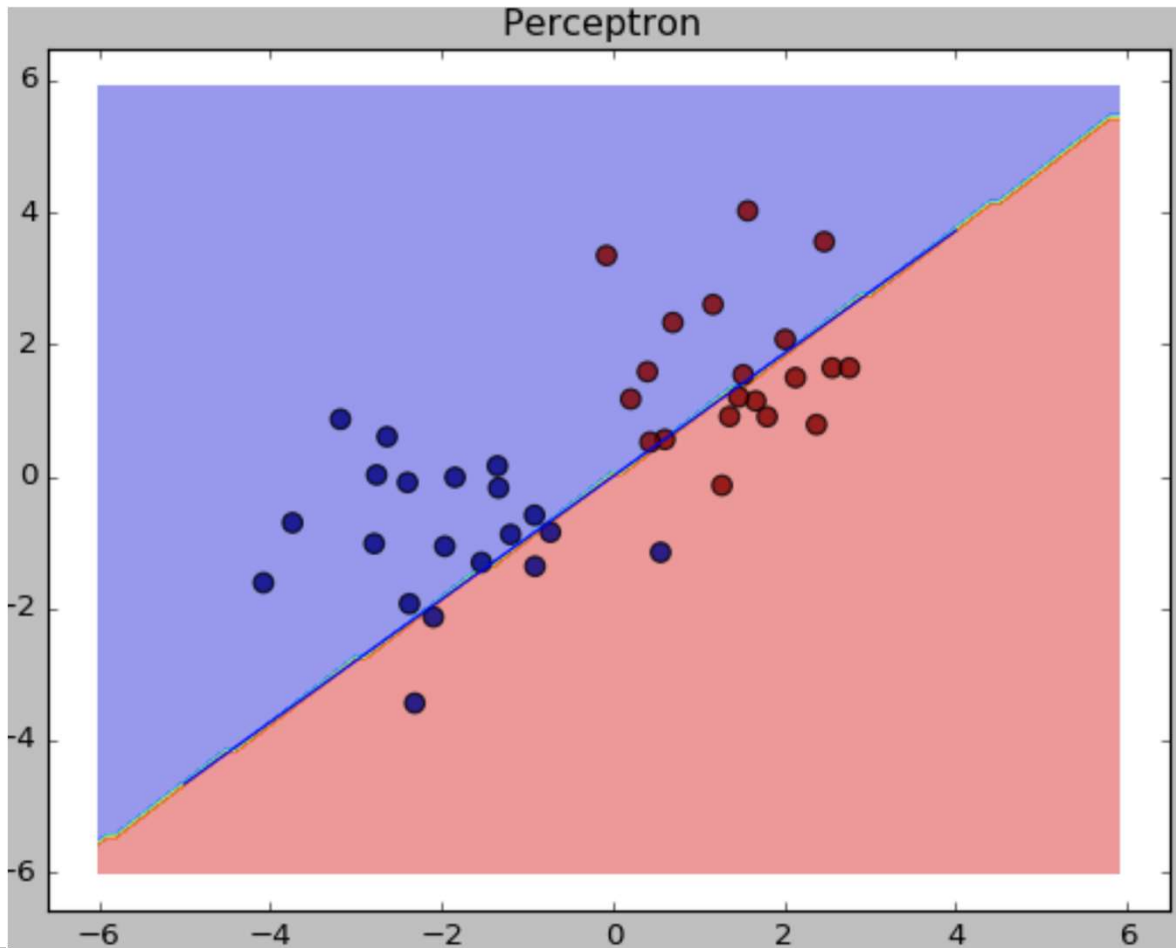
or

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

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

Perceptron Criterion (loss)

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



$$L(y_{\vec{w}}(\vec{x}), D) = 464.15$$

Optimisation

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

learning rate

Gradient of the loss function

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 gradient descent

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

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

Stochastic gradient descent (SGD)

```
Init  $\vec{w}$ 
k=0
DO k=k+1
  FOR n = 1 to N
    IF  $\vec{w}^T \vec{x}_n t_n < 0$  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. $\eta^{[k]} = cst / k$)

Similar loss functions

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

$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N \max(0, -t_n \vec{w}^T \vec{x}_n)$$

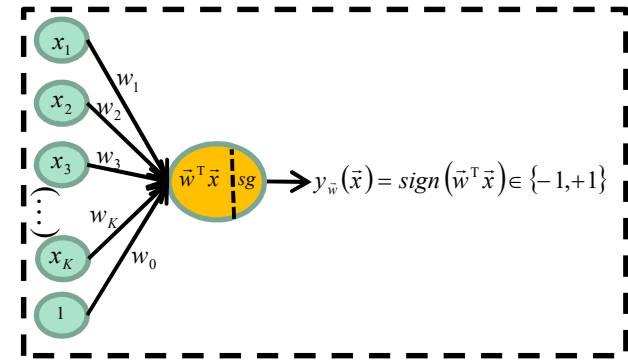
$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N \max(0, 1 - t_n \vec{w}^T \vec{x}_n) \quad \text{“Hinge Loss” or “SVM” Loss}$$

So far...

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

So far...

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

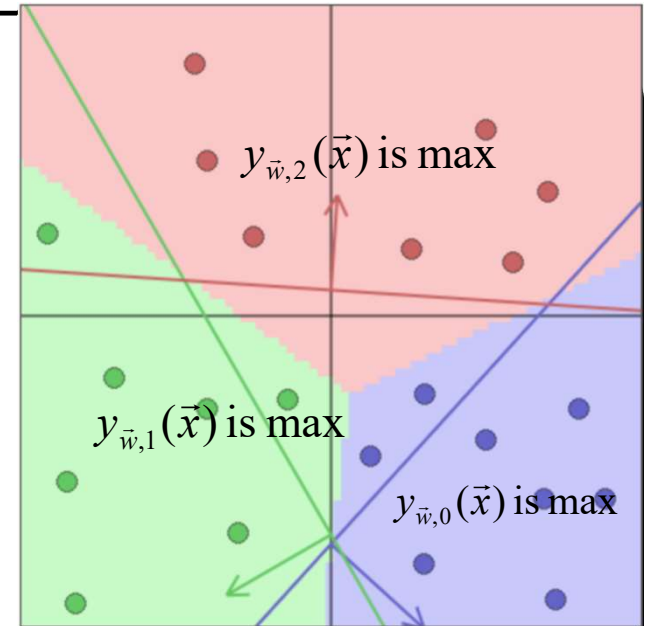
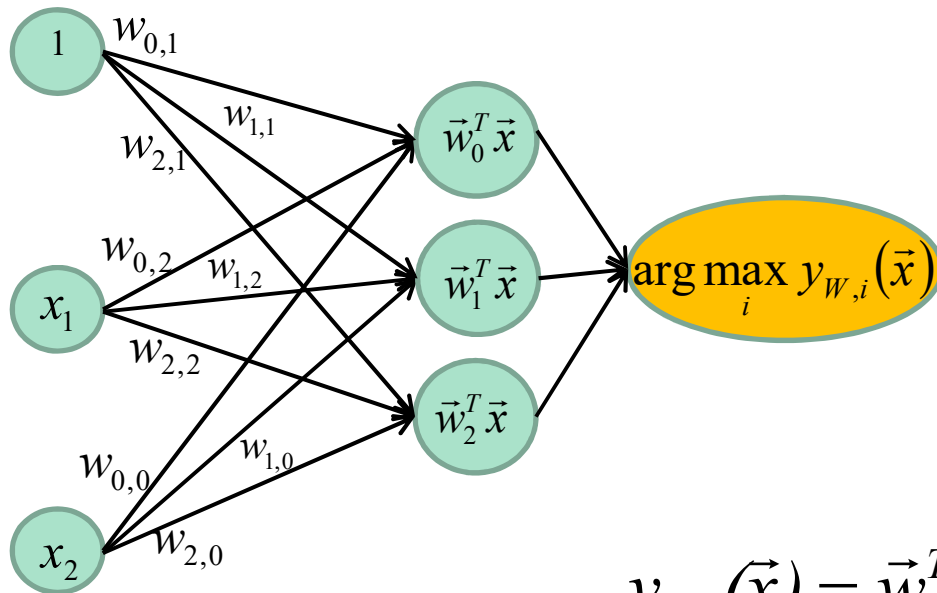


4. Training : find \vec{w} that minimizes $L(y_{\vec{w}}(\vec{x}), D)$

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

Multiclass Perceptron

(2D and 3 classes)



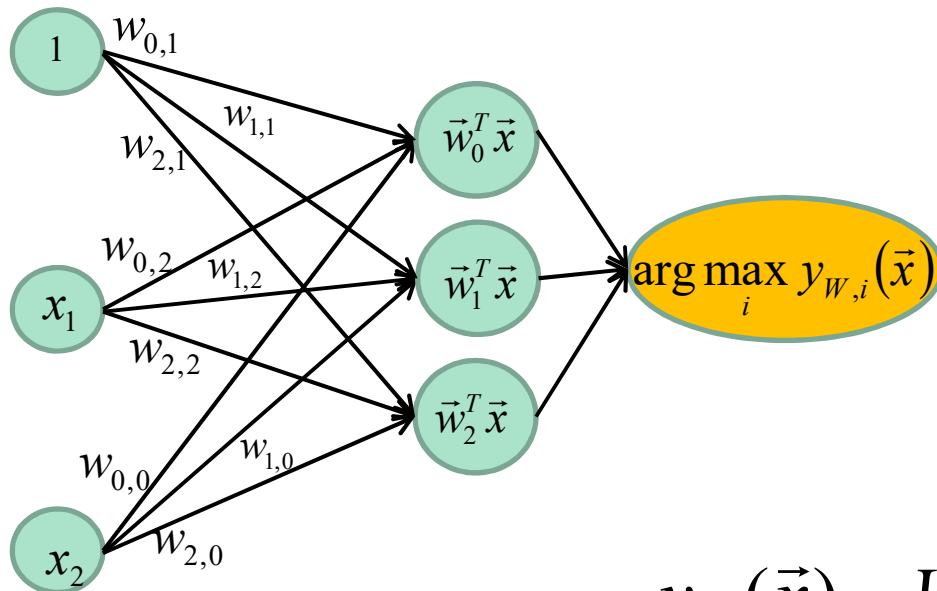
$$y_{\vec{w},0}(\vec{x}) = \vec{w}_0^T \vec{x} = w_{0,0} + w_{0,1}x_1 + w_{0,2}x_2$$

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

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

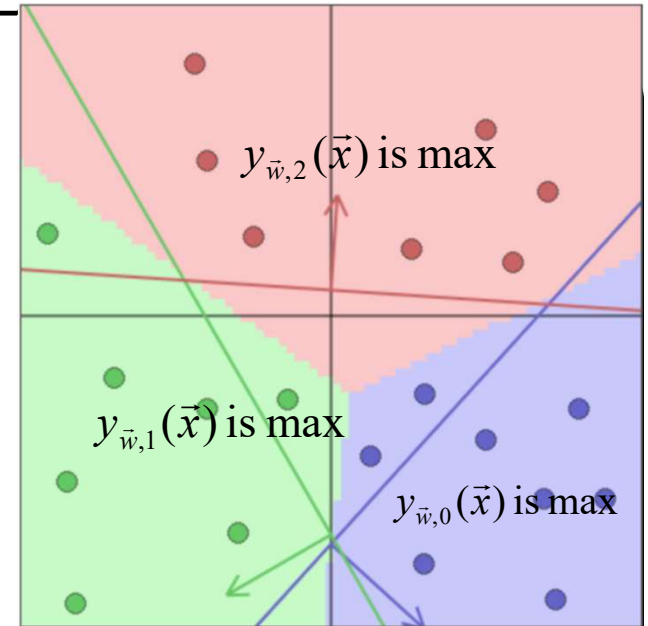
Multiclass Perceptron

(2D and 3 classes)



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

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

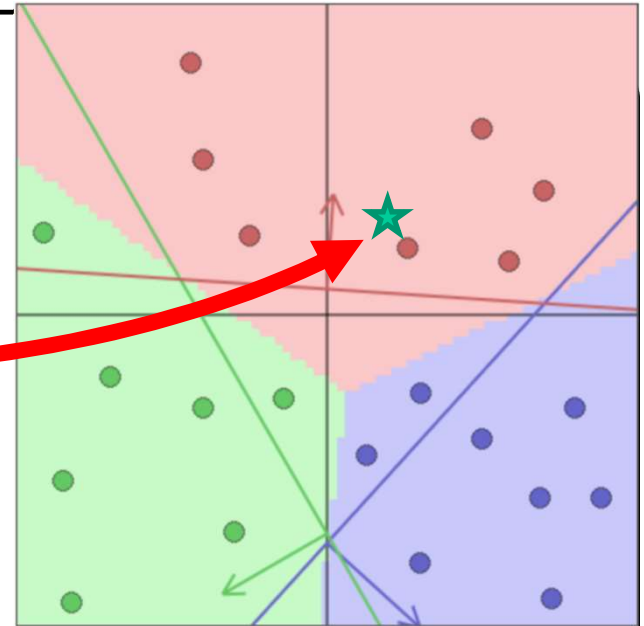


Multiclass Perceptron

(2D and 3 classes)

Example

★ (1.1, -2.0)



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

Multiclass Perceptron

Loss function

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

Sum over all wrongly
classified samples

Score of the true class

Score of the wrong class

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

Multiclass Perceptron

Stochastic gradient descent (SGD)

```
init W  
k=0, i=0  
DO k=k+1  
  FOR n = 1 to N  
     $j = \arg \max \mathbf{W}^T \vec{x}_n$   
    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

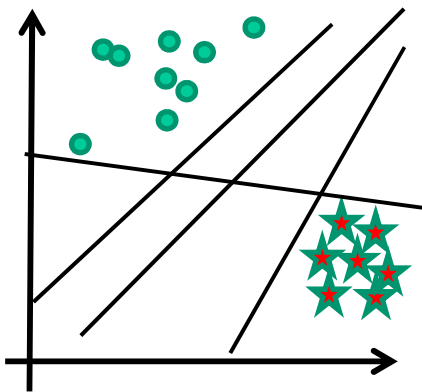
Advantages:

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

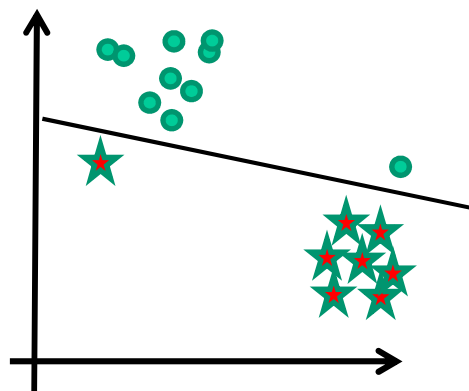
Limitations:

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

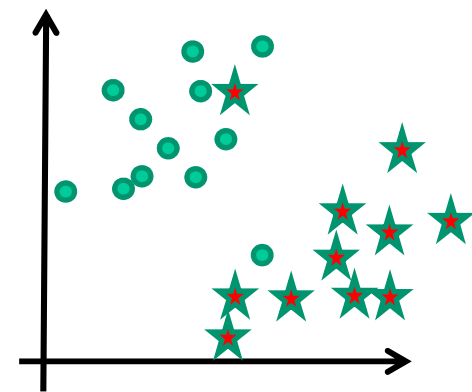
Many “optimal”
solutions



Suboptimal solution



Will never converge



Two famous ways of improving the Perceptron

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



Logistic regression

1. **New network** architecture



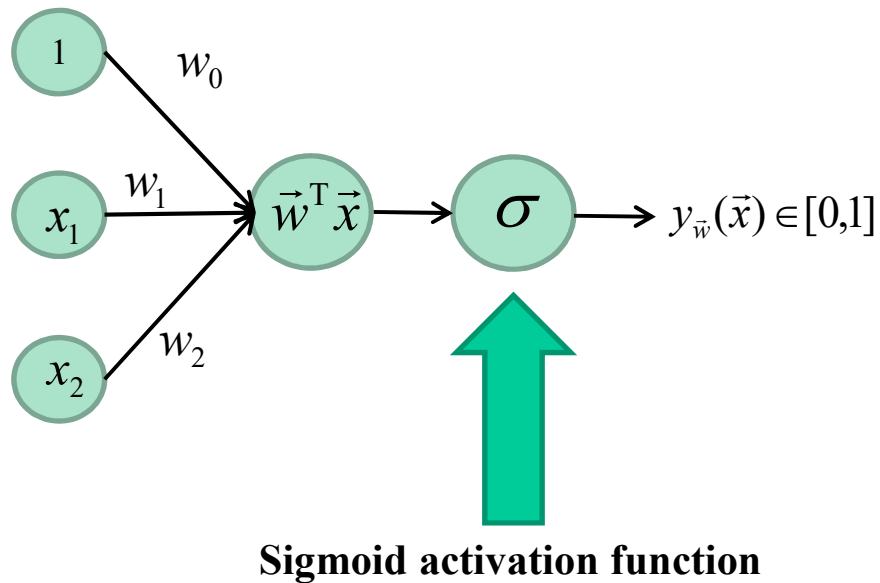
Multilayer Perceptron / CNN

Logistic regression

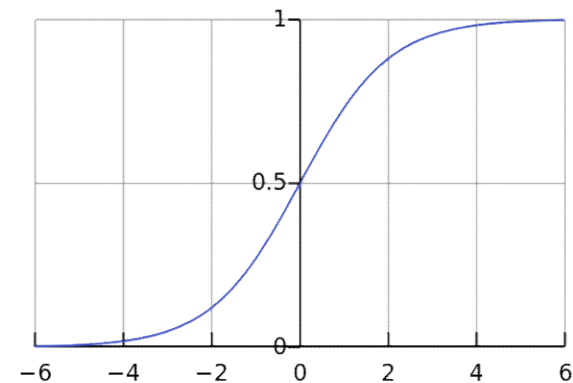
Logistic regression

(2D, 2 classes)

New activation function: **sigmoid**



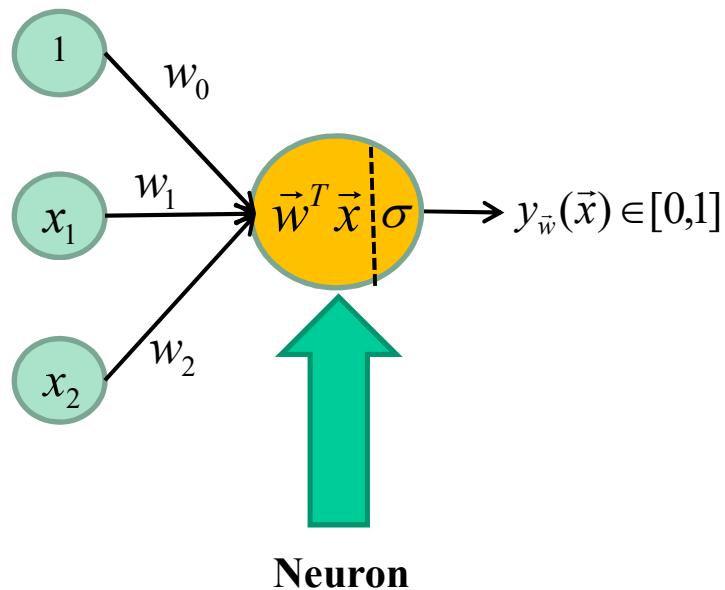
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



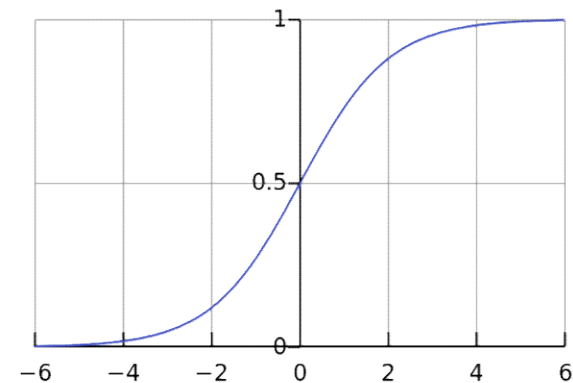
Logistic regression

(2D, 2 classes)

New activation function: **sigmoid**



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

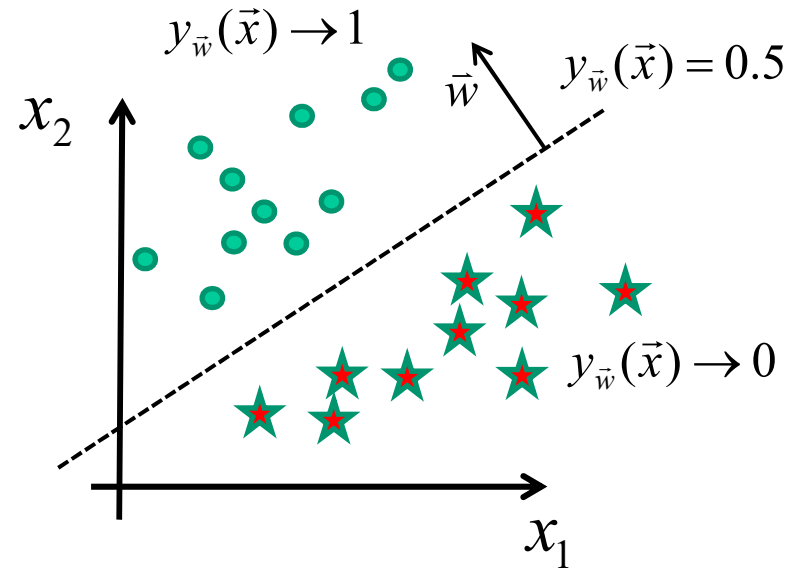
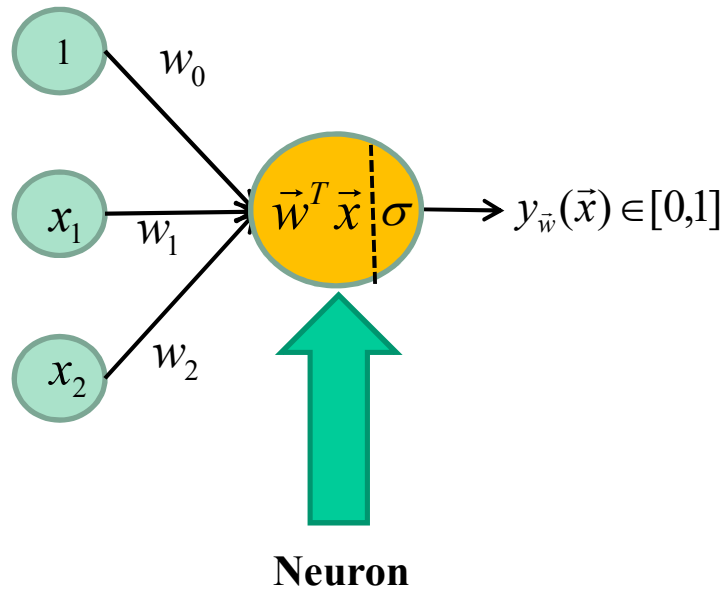


$$y_{\vec{w}}(\vec{x}) = \sigma(\vec{w}^T \vec{x}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}$$

Logistic regression

(2D, 2 classes)

New activation function: **sigmoid**



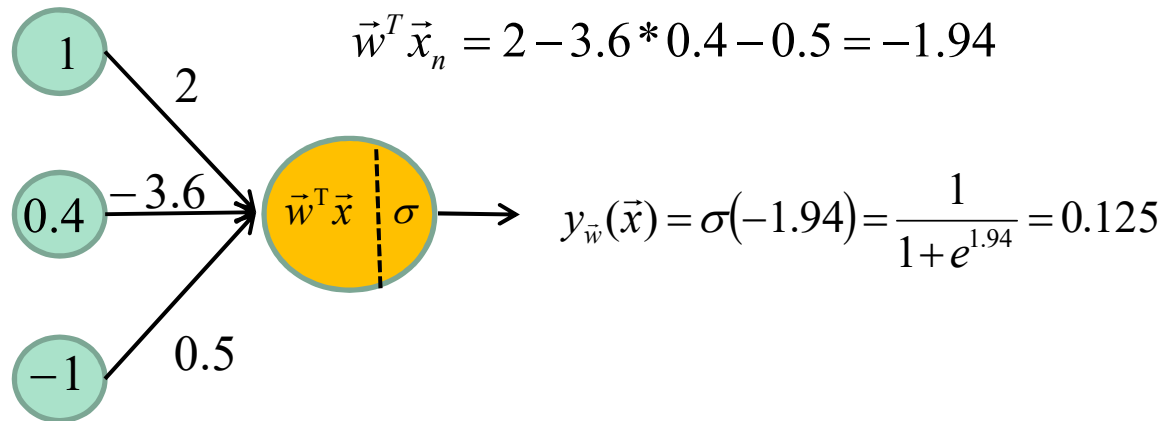
$$y_{\vec{w}}(\vec{x}) = \sigma(\vec{w}^T \vec{x})$$

Logistic regression

(2D, 2 classes)

Example

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

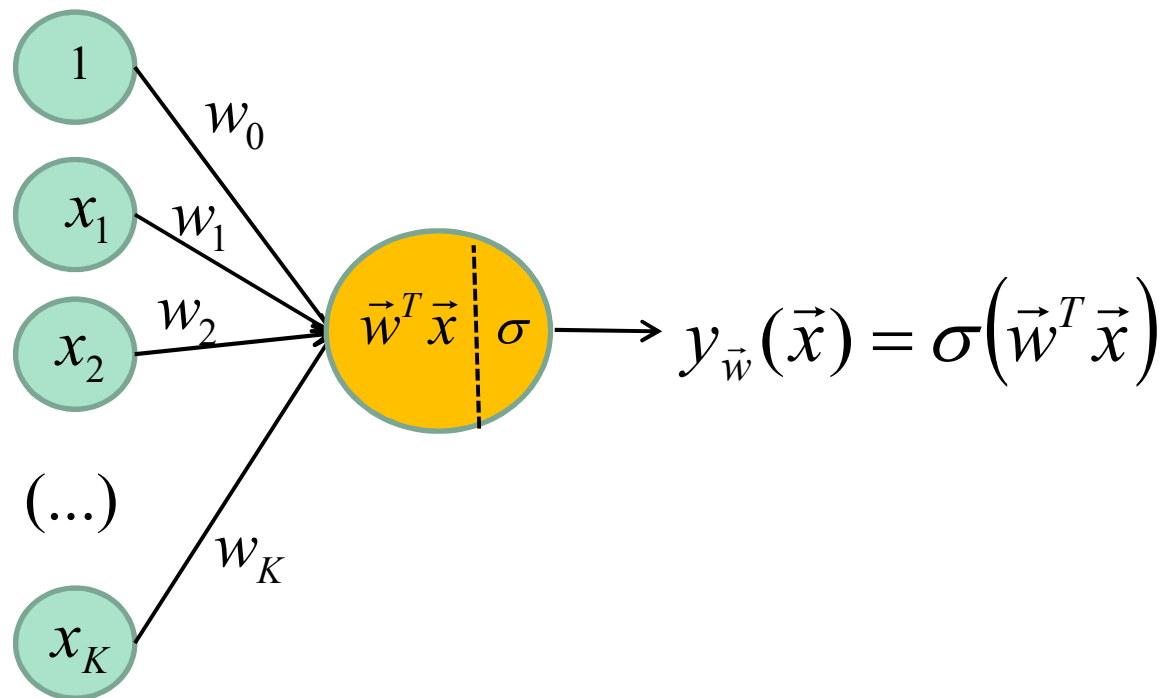


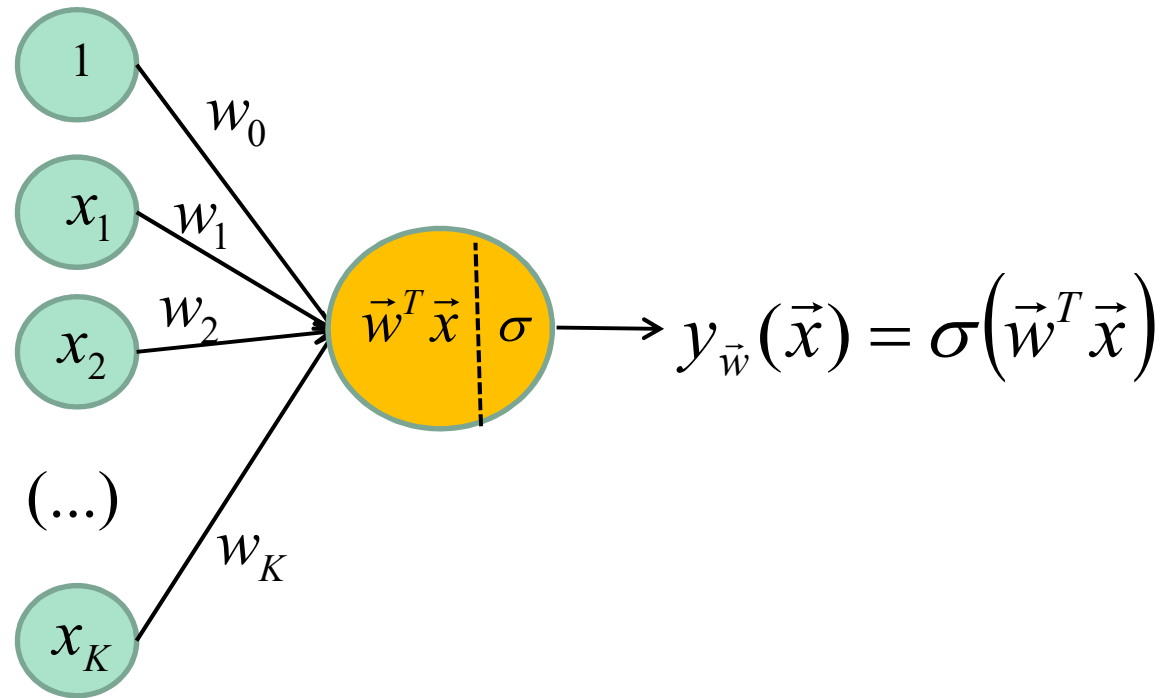
Since 0.125 is lower than 0.5, \vec{x}_n is **behind** the plan.

Logistic regression

(K-D, 2 classes)

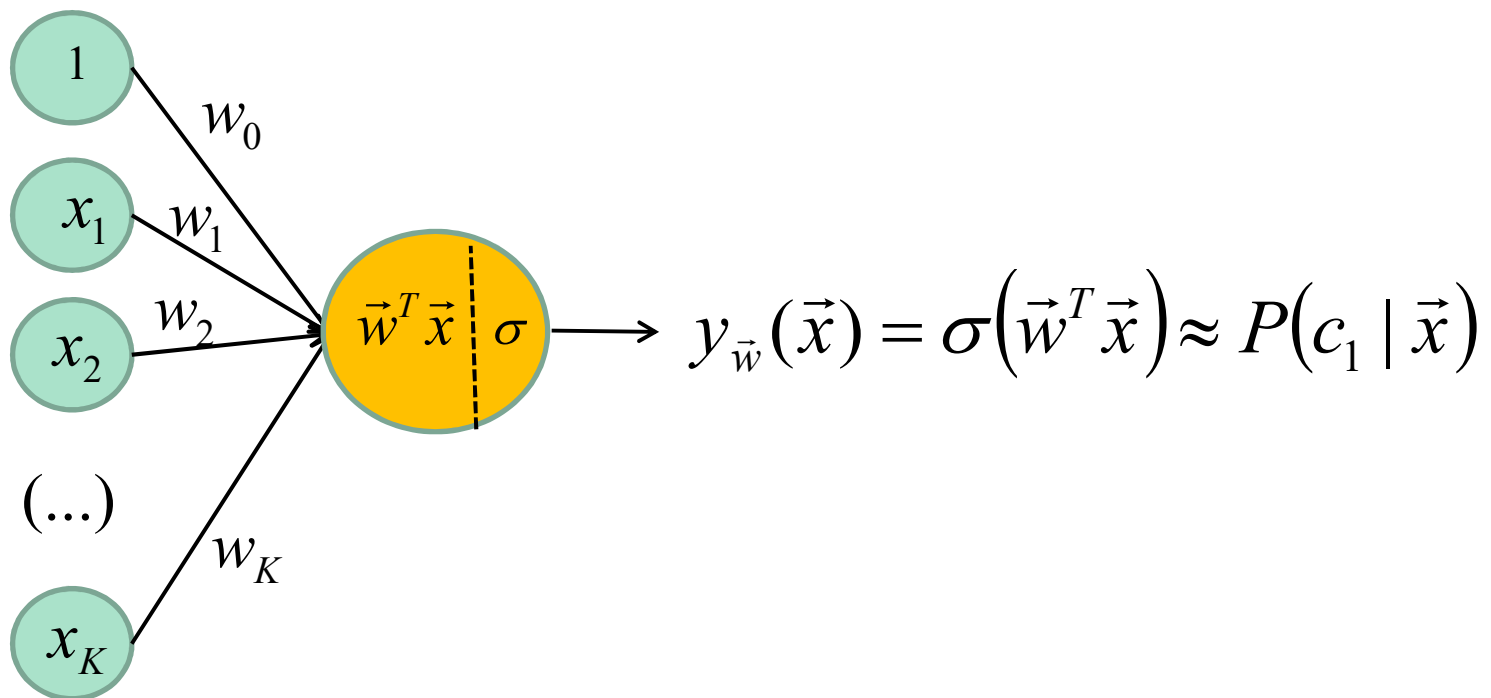
Like the Perceptron the logistic regression accomodates for K-D vectors



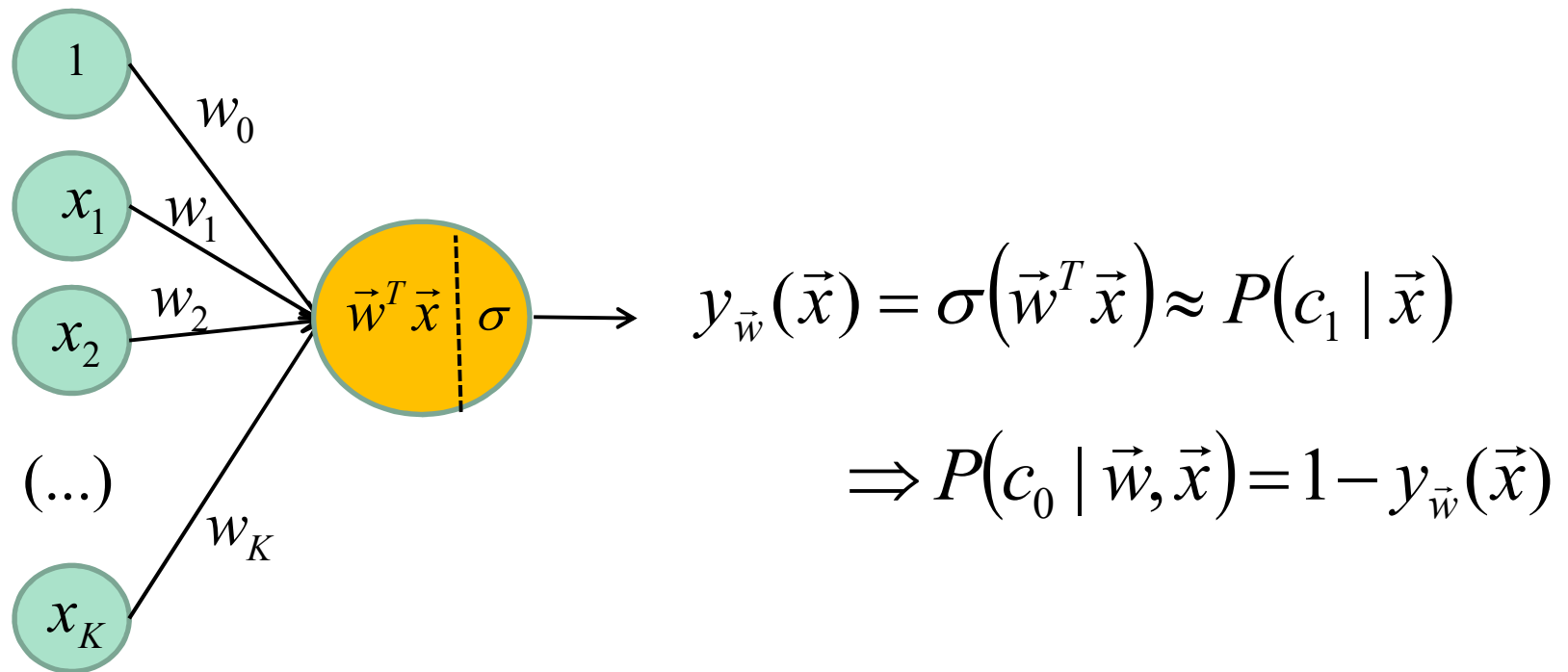


What is the loss function?

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



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



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

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

2 Class Cross entropy

We can also show that

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

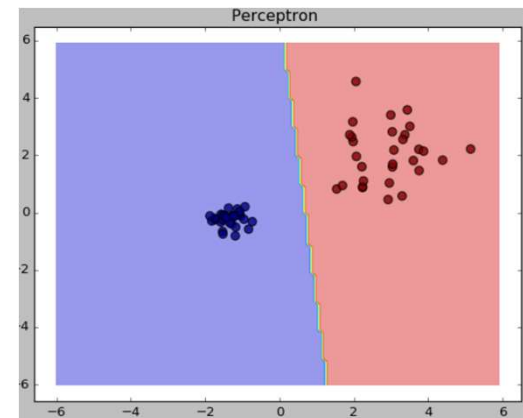
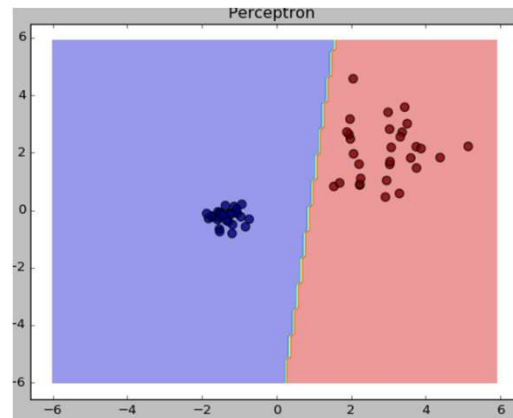
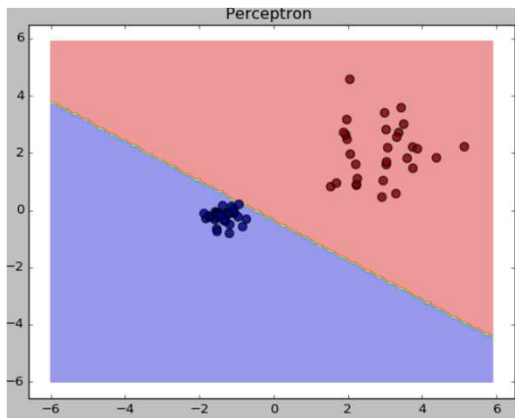
As opposed to the Perceptron
the gradient does not depend
on the wrongly classified samples

Logistic Network

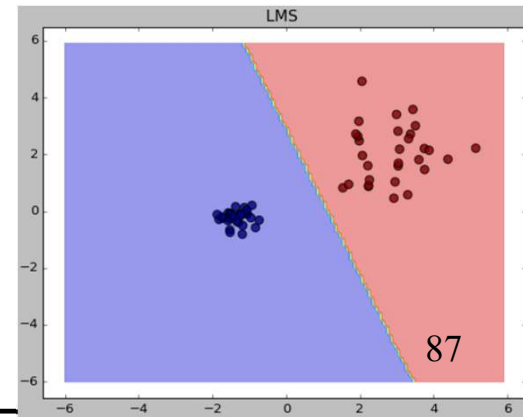
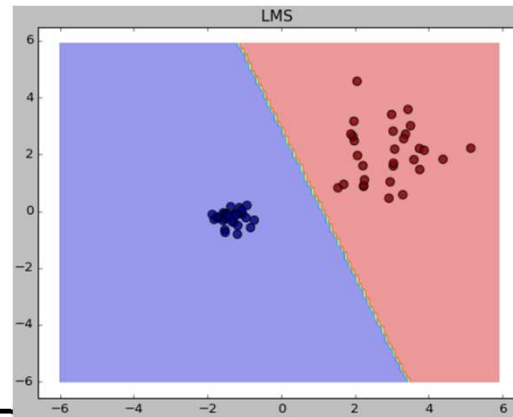
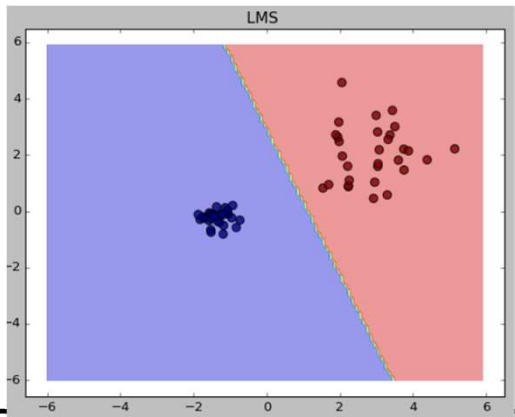
Advantages:

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

Perceptron

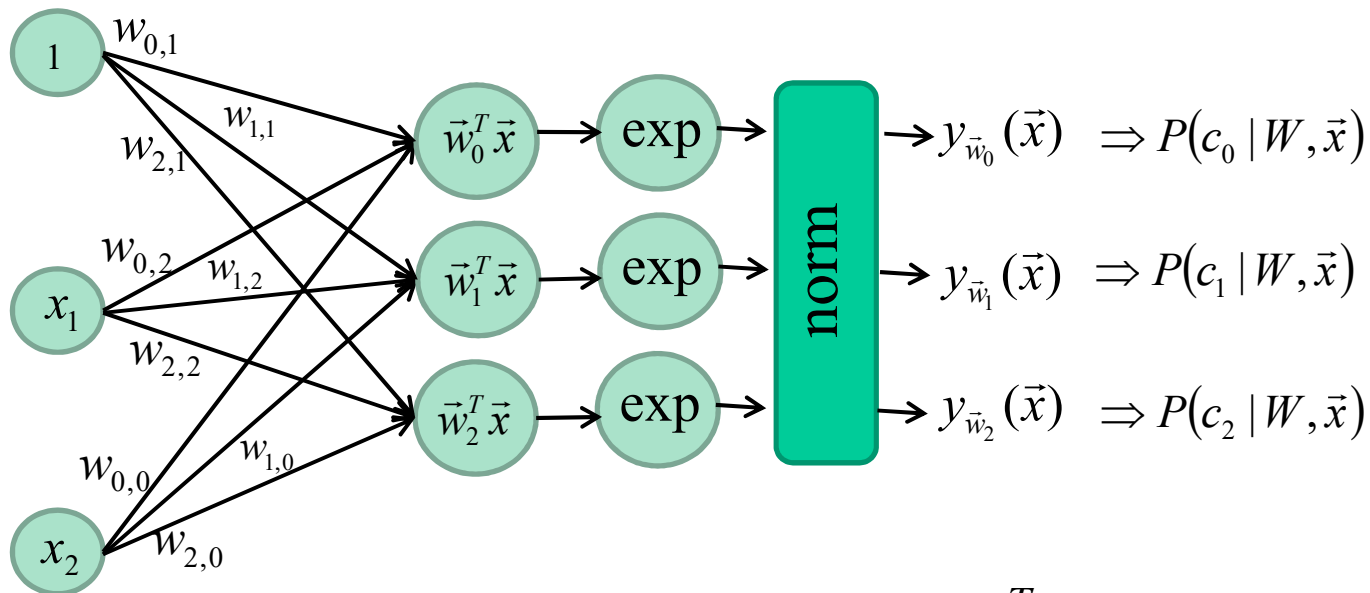


Logistic net



And for $K > 2$ classes?

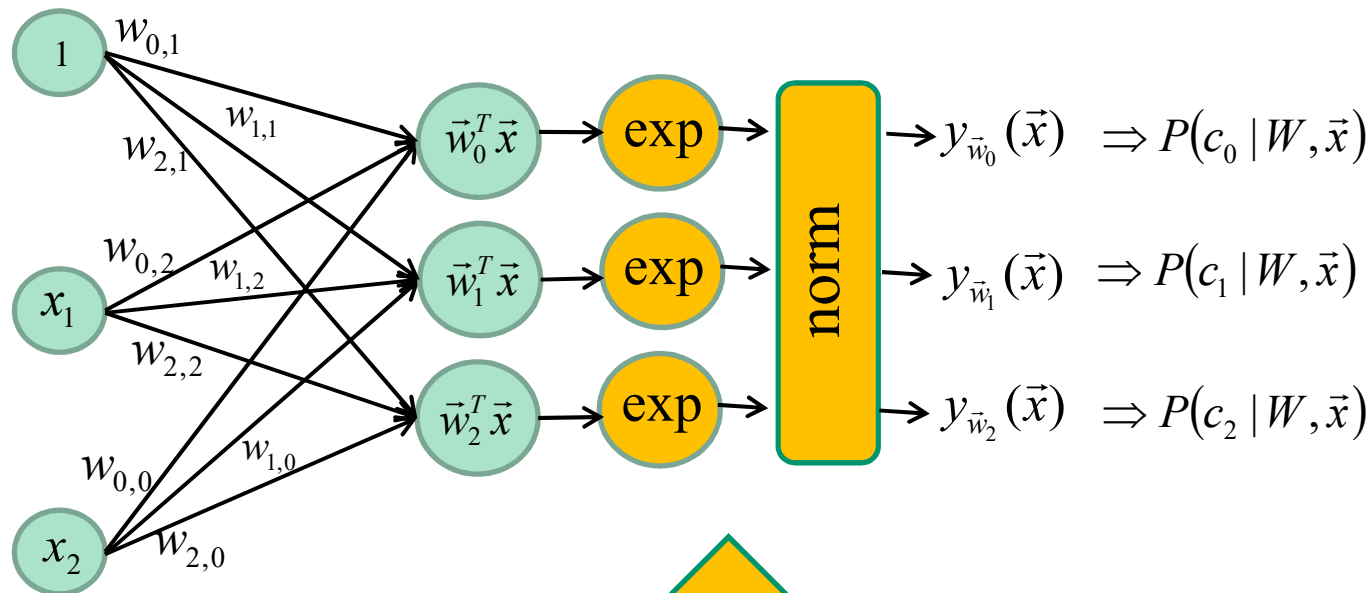
New activation function : **Softmax**



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

And for $K > 2$ classes?

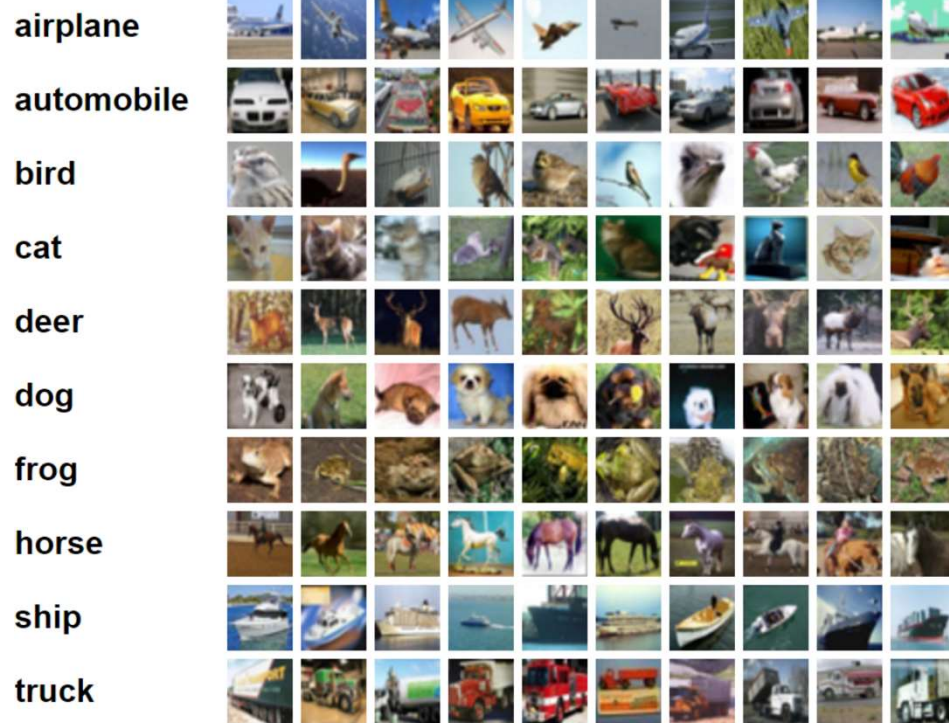
New activation function : **Softmax**



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

'airplane' $\Rightarrow t = [1000000000]$

'automobile' $\Rightarrow t = [0100000000]$

'bird' $\Rightarrow t = [0010000000]$

'cat' $\Rightarrow t = [0001000000]$

'deer' $\Rightarrow t = [0000100000]$

'dog' $\Rightarrow t = [0000010000]$

'frog' $\Rightarrow t = [0000001000]$

'horse' $\Rightarrow t = [0000000100]$

'ship' $\Rightarrow t = [0000000010]$

'truck' $\Rightarrow t = [0000000001]$

Class labels : **one-hot vectors**

$K > 2$ classes

Cross entropy Loss

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

K-Class *cross entropy* loss

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

$$\nabla L = \sum_{n=1}^N \vec{x}_n (y_W(\vec{x}_n) - t_{kn})$$

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

$$\arg \min_W = L(y_{\bar{w}}(\vec{x}), D) + \lambda R(W)$$

Constant

Loss function

Regularization

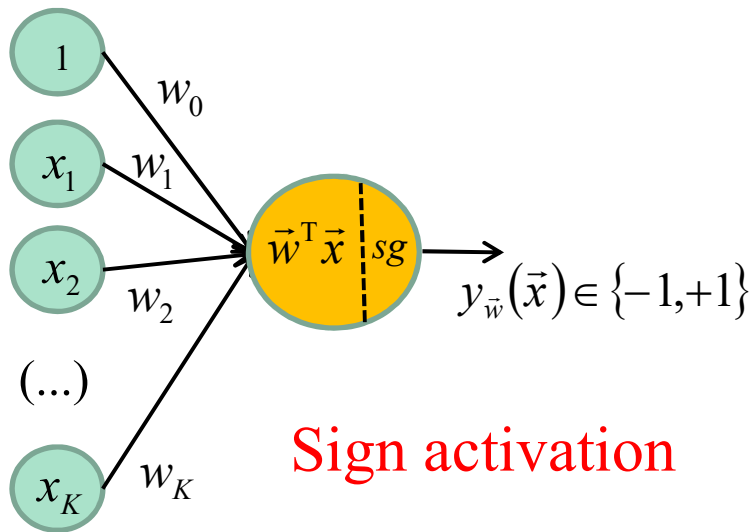
In general L1 or L2 $R(\theta) = \|\mathbf{W}\|_1$ ou $\|\mathbf{W}\|_2$

Wow! Loooots of information!

Lets recap...

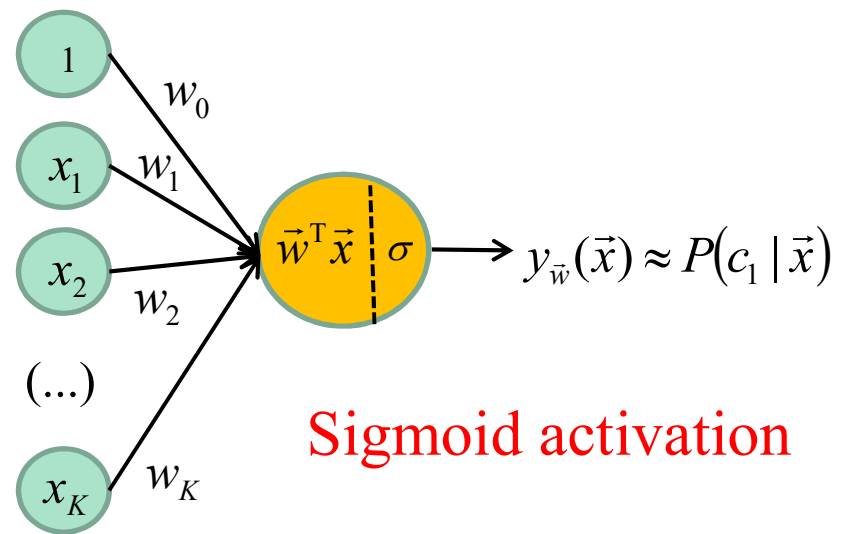
Neural networks

2 classes



Sign activation

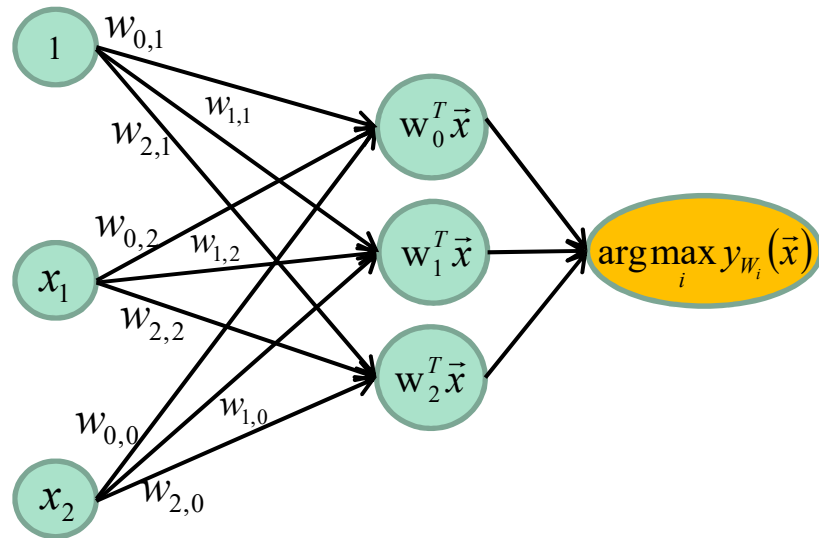
Perceptron



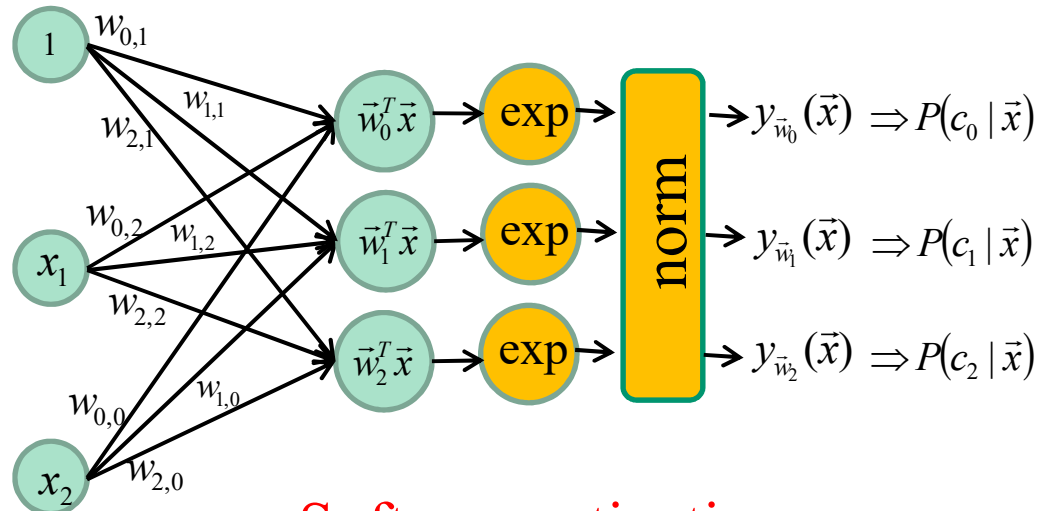
Sigmoid activation

Logistic regression

K-Class Neural networks



Perceptron



Softmax activation

Logistic regression

Loss functions

2 classes

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

$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N \max(0, -t_n \vec{w}^T \vec{x}_n)$$

$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N \max(0, 1 - t_n \vec{w}^T \vec{x}_n) \quad \text{“Hinge Loss” or “SVM” Loss}$$

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

Loss functions

K classes

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

$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N \sum_j \max(0, \vec{w}_j^T \vec{x}_n - \vec{w}_{t_n}^T \vec{x}_n)$$

$$L(y_{\vec{w}}(\vec{x}), D) = \sum_{n=1}^N \sum_j \max(0, 1 + \vec{w}_j^T \vec{x}_n - \vec{w}_{t_n}^T \vec{x}_n) \quad \text{“Hinge Loss” or “SVM” Loss}$$

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

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

Constante



Loss function



Regularization



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

Optimisation

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

learning rate

Gradient of the loss function

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

Now, lets go

DEEPER

DEEPEK

Now, lets go

Non-linearly separable training data

Three classical solutions

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



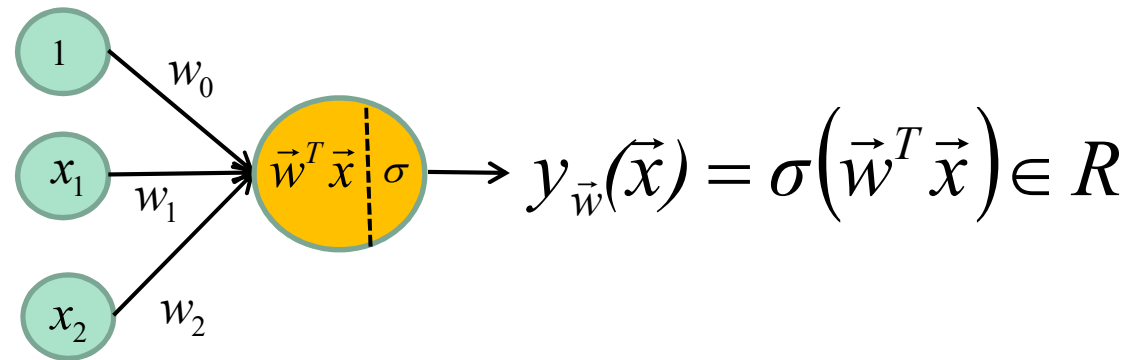
Non-linearly separable training data

Three classical solutions

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



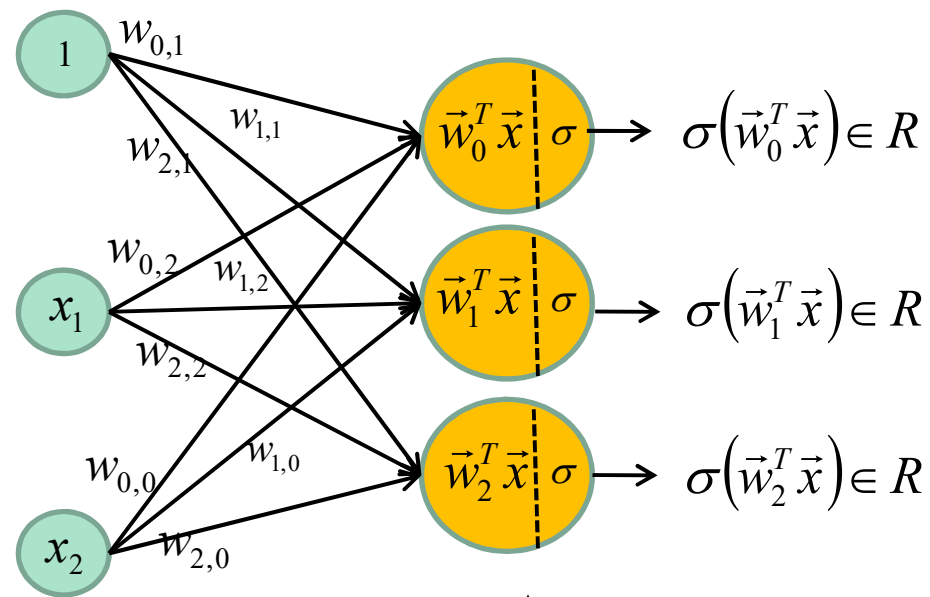
2D, 2Classes, Linear logistic regression



Input layer
(3 “neurons”)

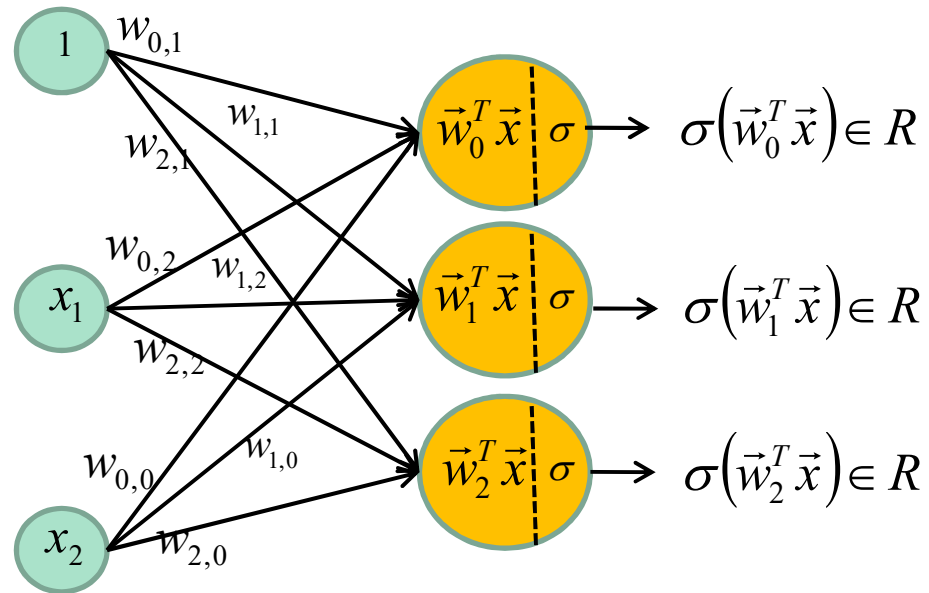
Output layer
(1 neuron with sigmoid)

Let's add 3 neurons



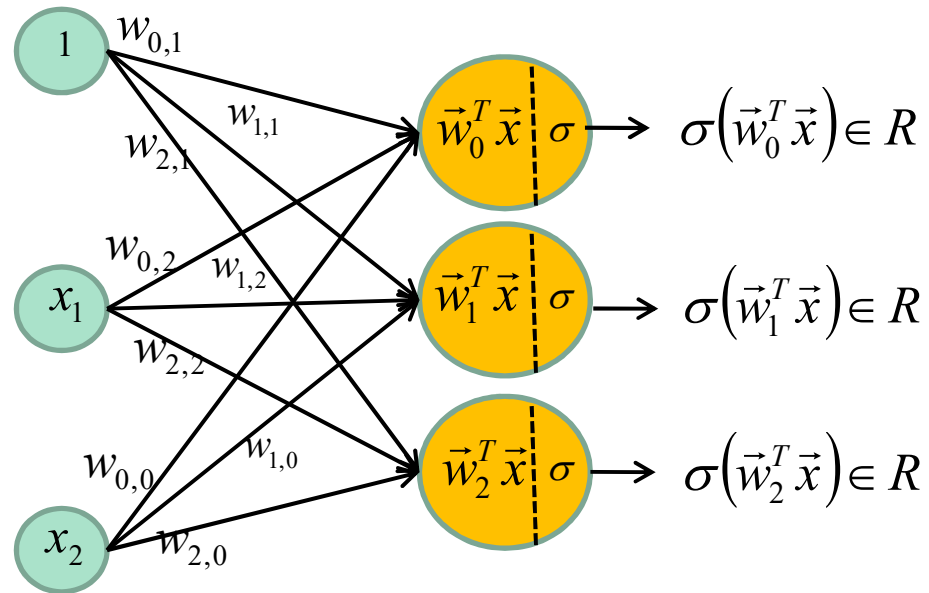
**Input layer
(3 “neurons”)**

**First layer
(3 neurons)**



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

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

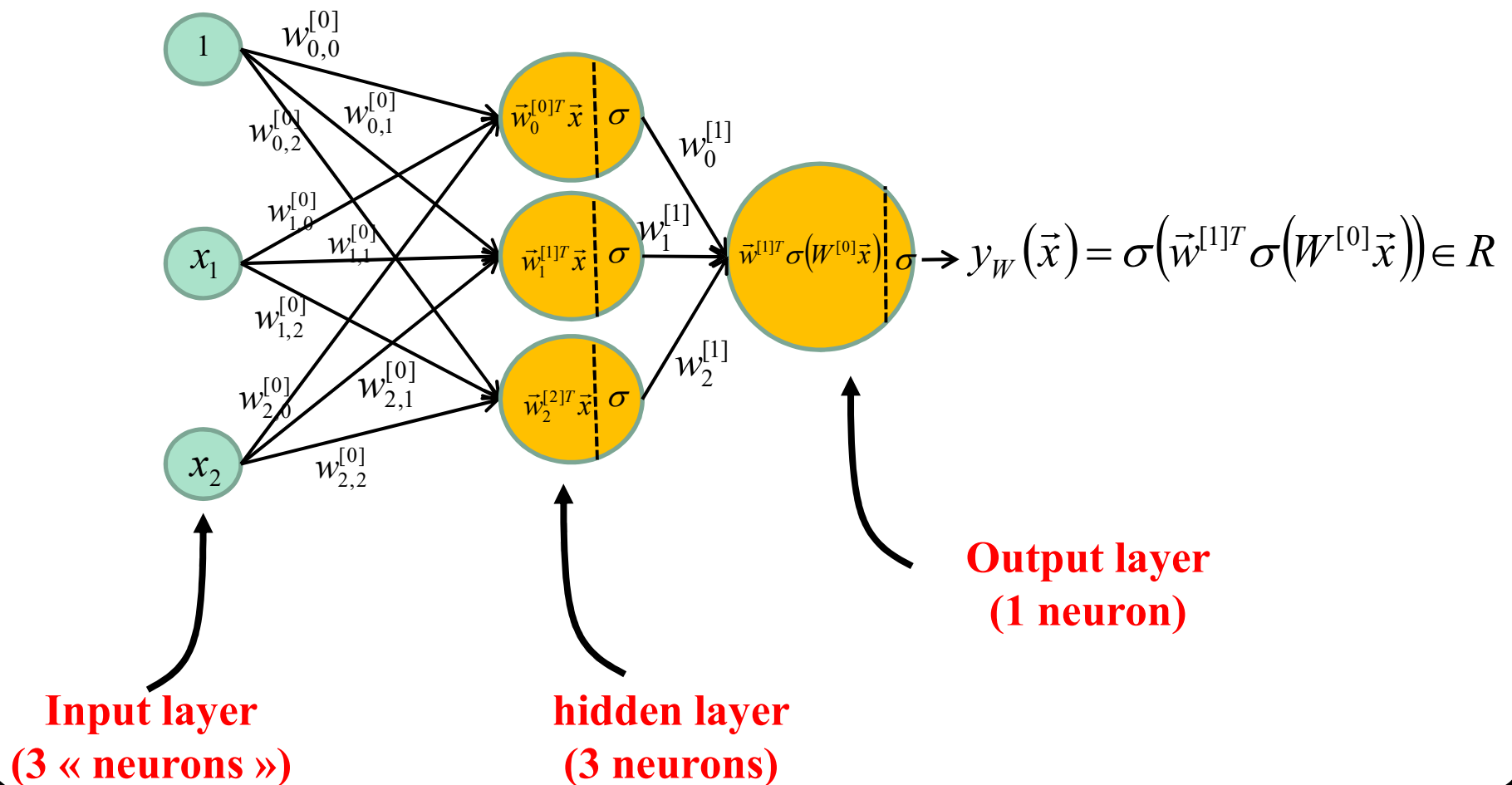


NOTE: The output of the first layer is a vector of **3 real** values

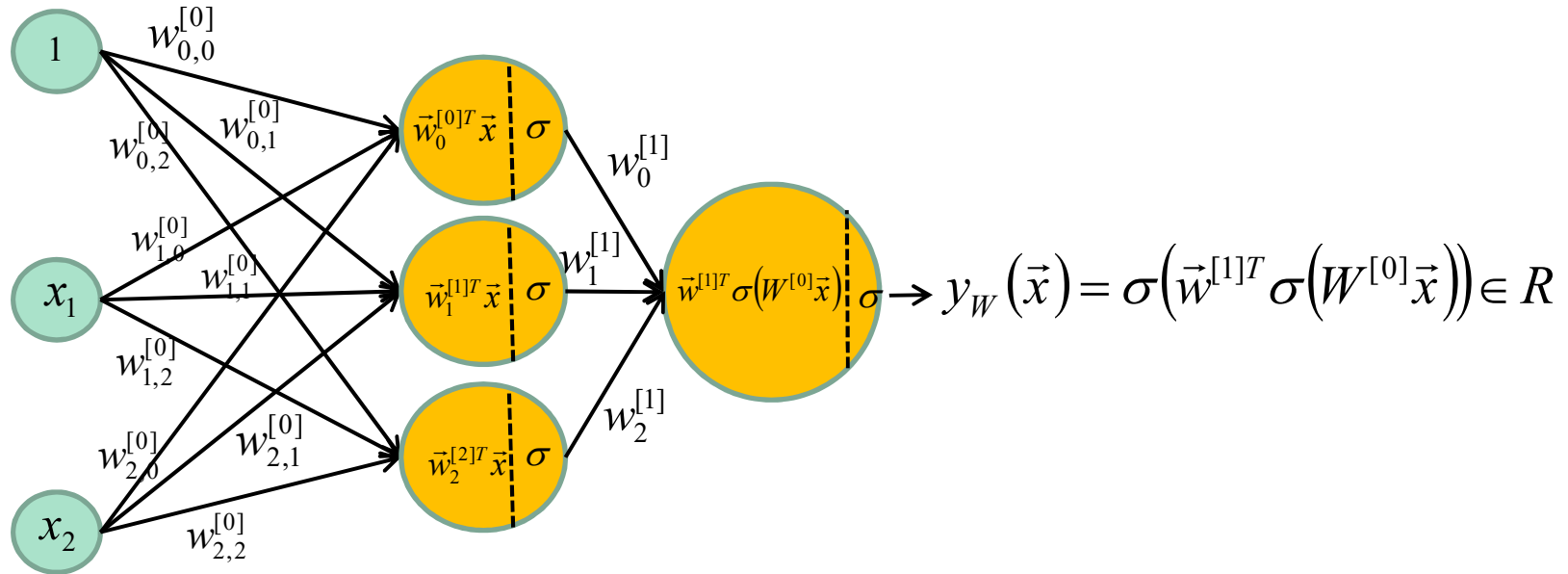
$$\sigma \left(W^{[0]} \vec{x} \right)$$

2-D, 2-Class, 1 hidden layer

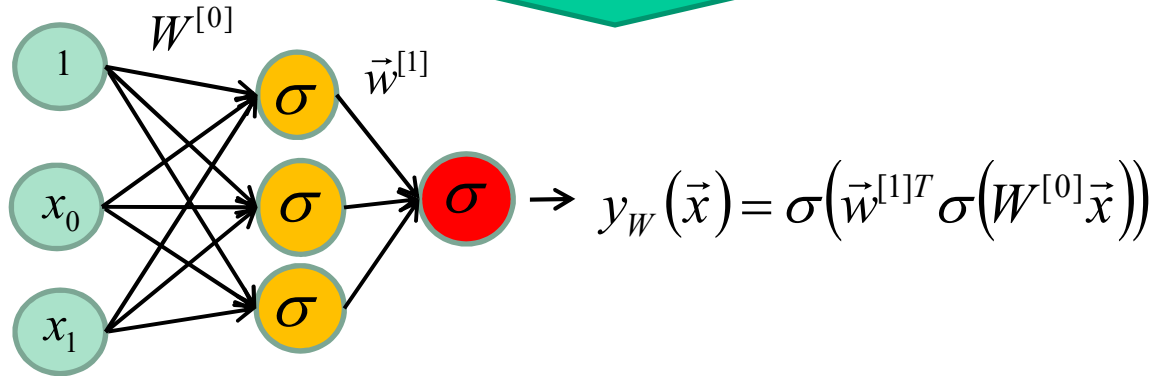
If we want a **2-class Classification** via a **logistic regression** (a **cross entropy loss**) we must add an **output neuron**.



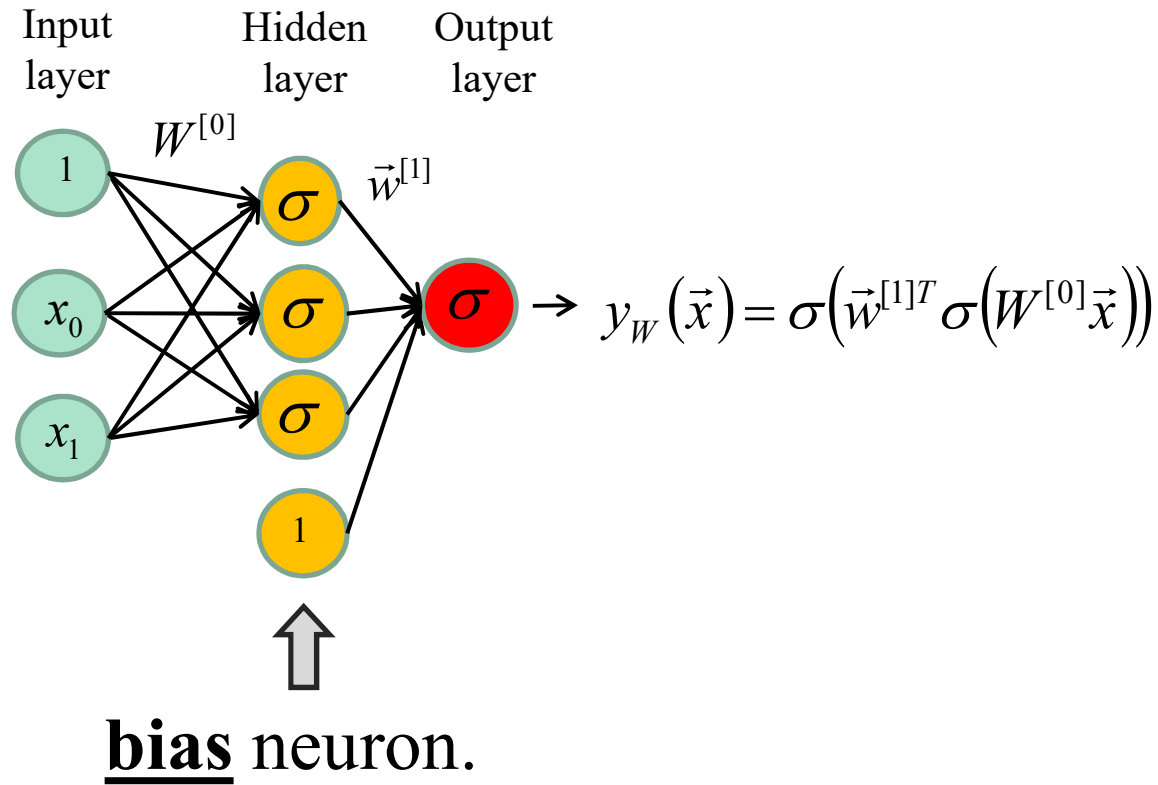
2-D, 2-Class, 1 hidden layer



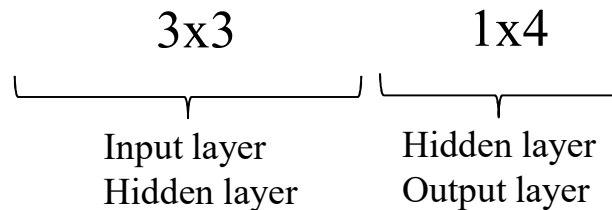
Visual simplification



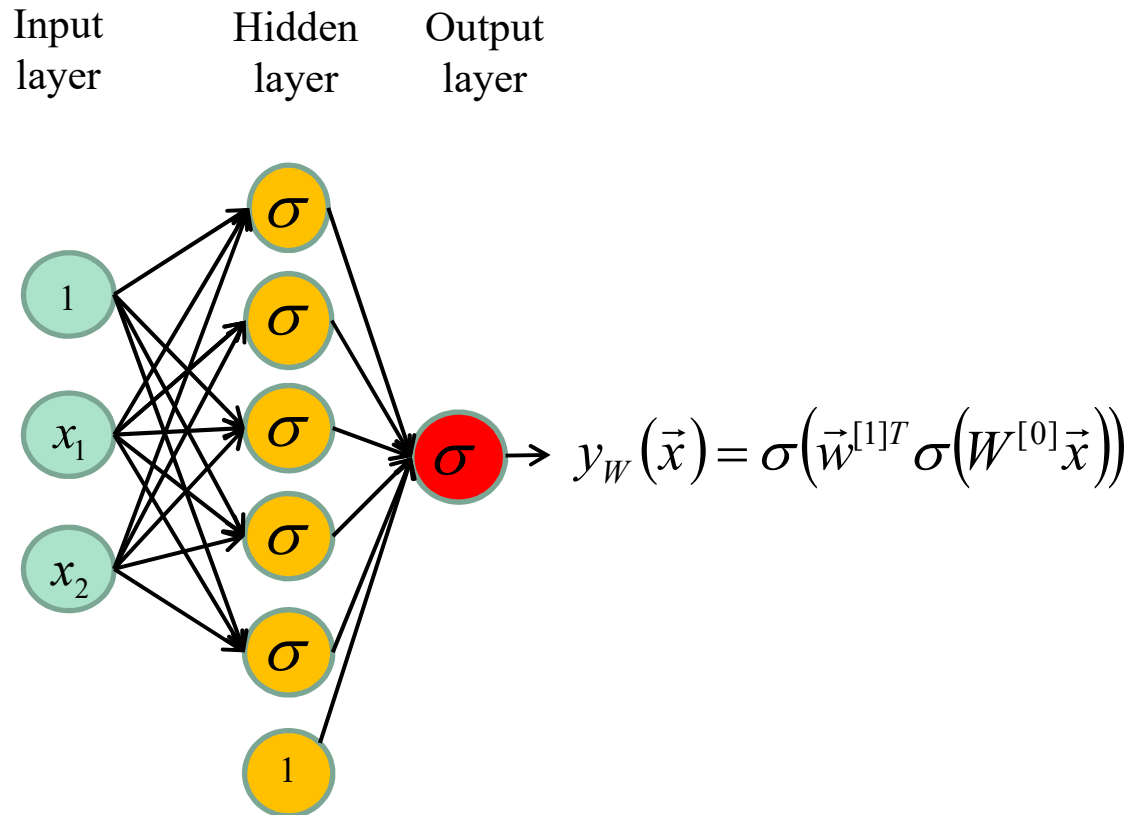
2-D, 2-Class, 1 hidden layer



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



2-D, 2-Class, 1 hidden layer

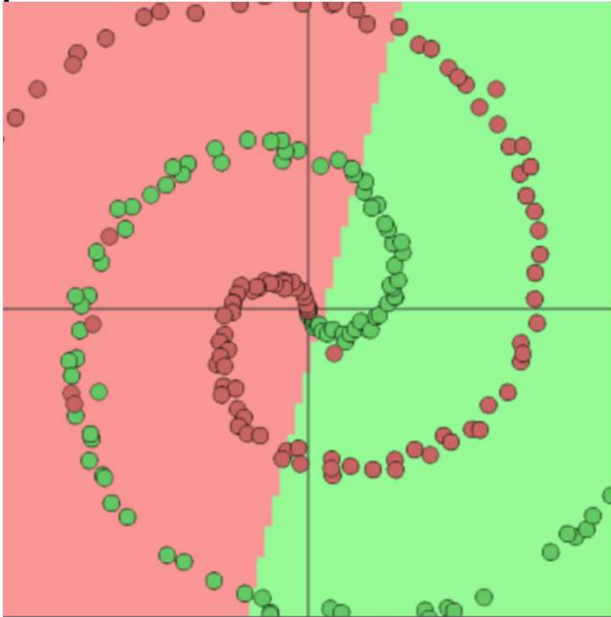


Increasing the number of neurons = increasing the **capacity of the model**

This network has $5 \times 3 + 1 \times 6 = \mathbf{21}$ parameters

Nb neurons VS Capacity

No hidden neuron

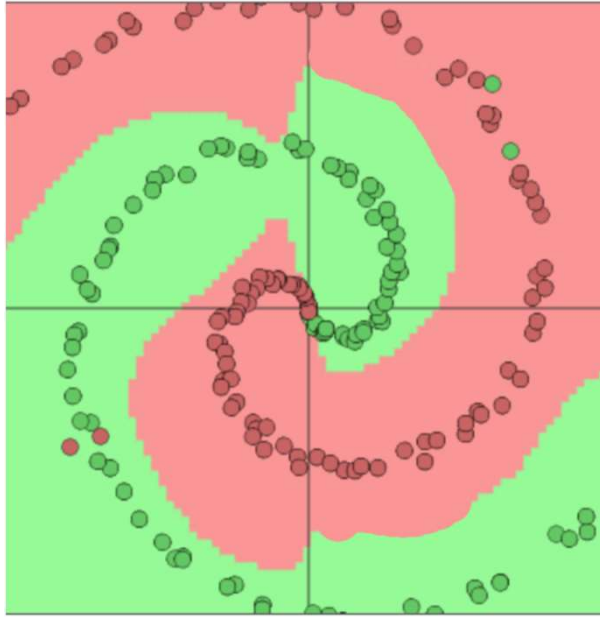


Linear classification

Underfitting

(low capacity)

12 hidden neurons

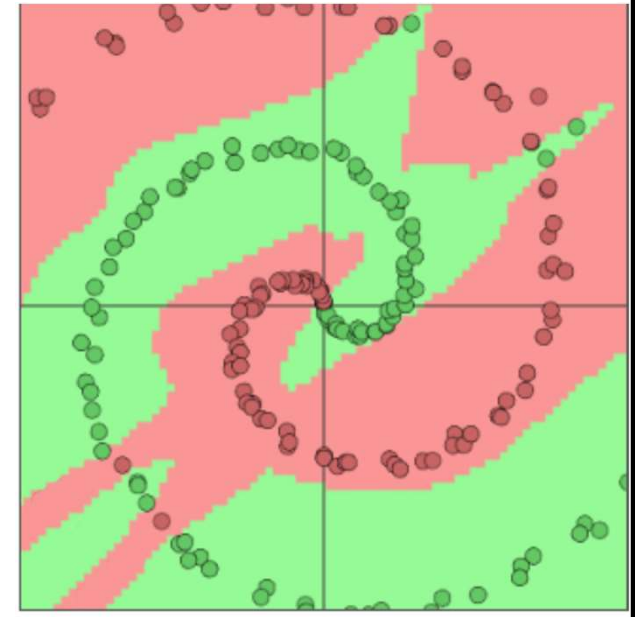


Non linear classification

Good result

(good capacity)

60 hidden neurons

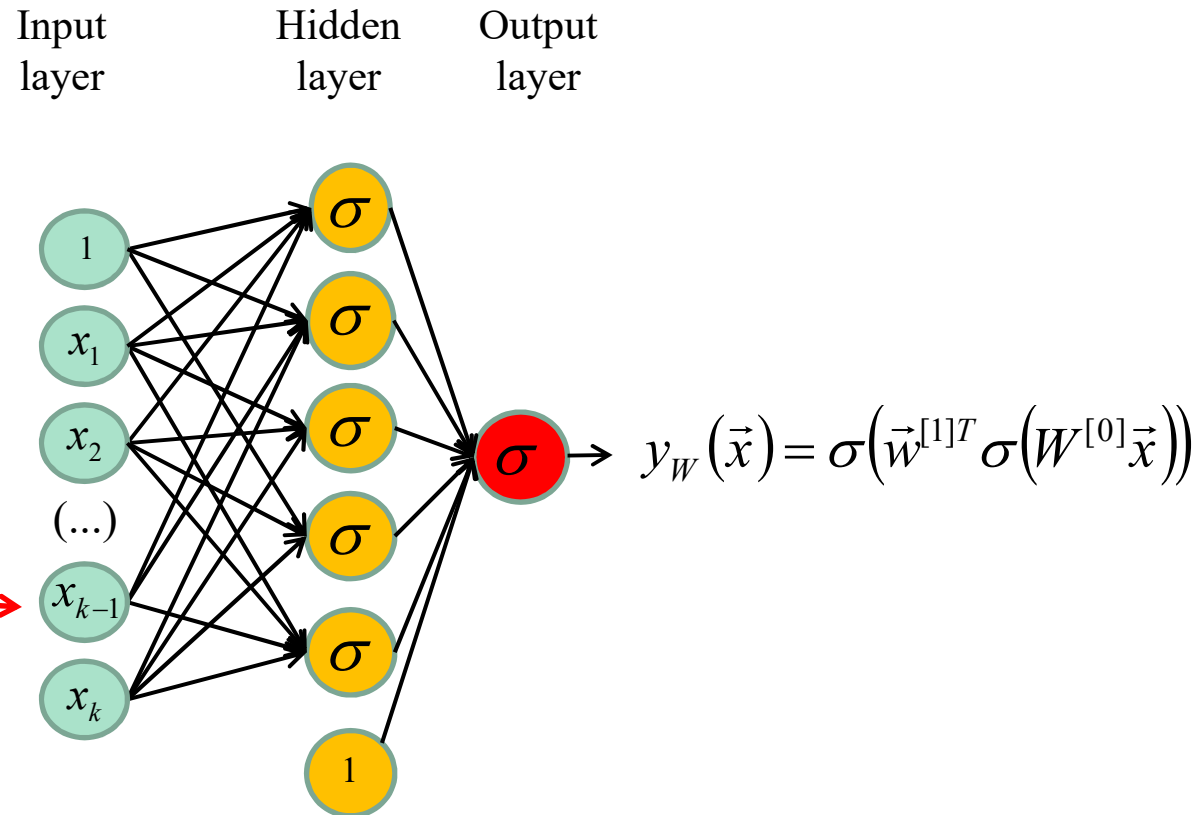


Non linear classification

Over fitting

(too large capacity)

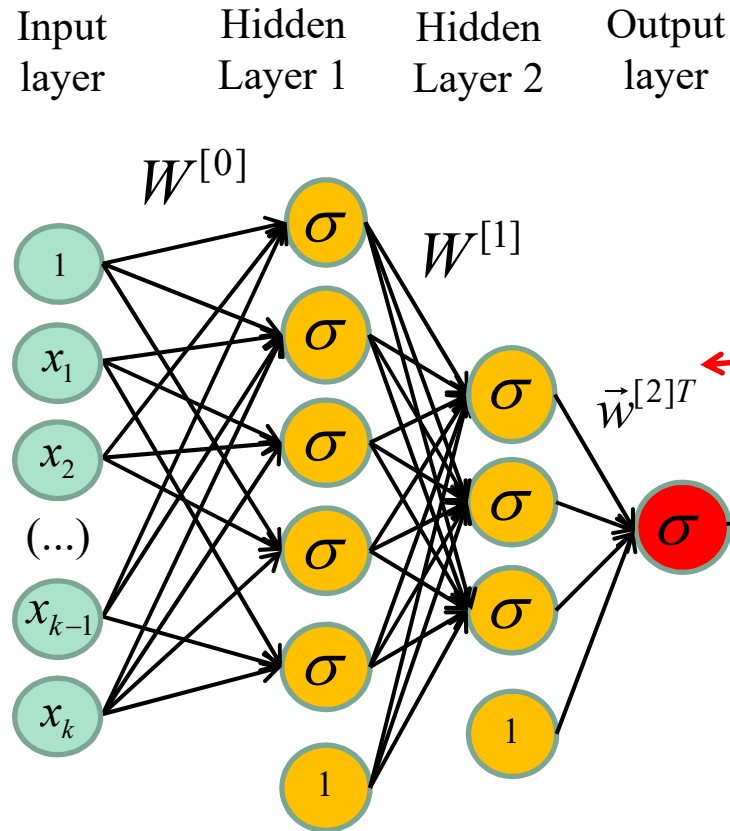
kD, 2Classes, 1 hidden layer



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

This network has $5 \times (k+1) + 1 \times 6$ **parameters**

kD, 2Classes, 2 hidden layers



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

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

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

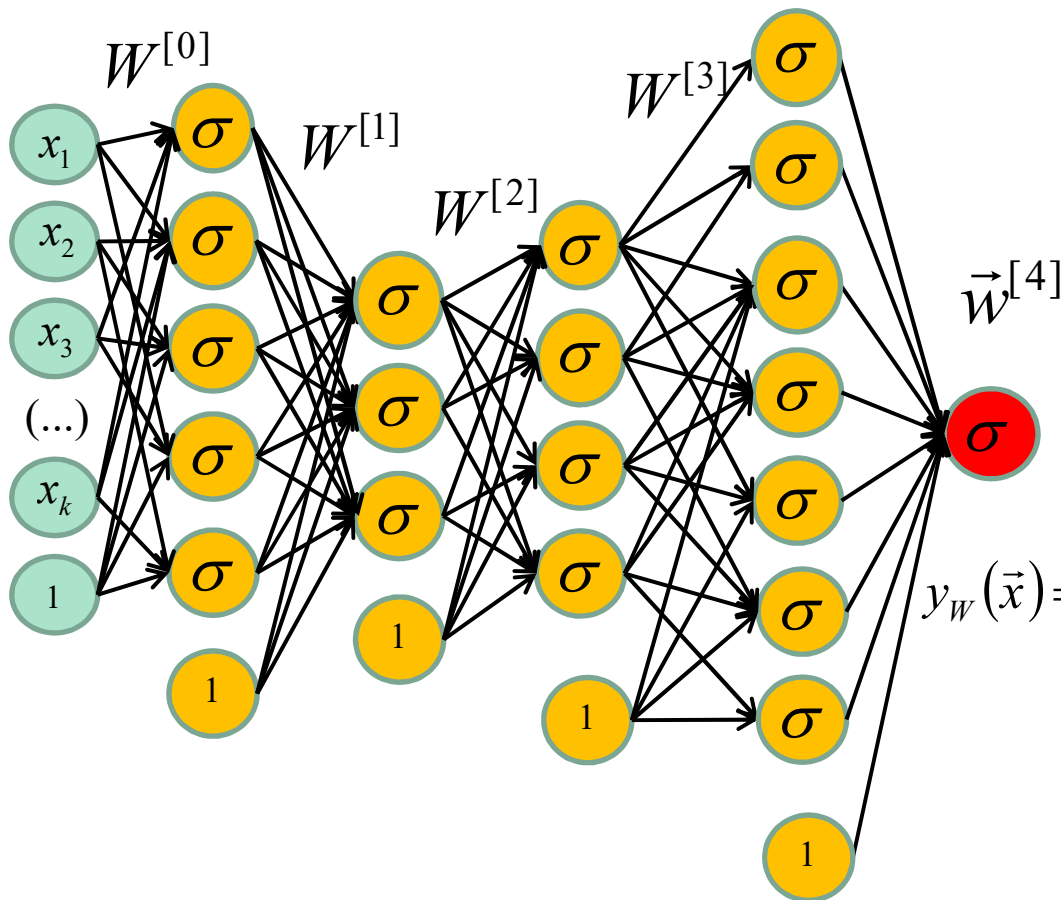
$$y_W(\vec{x}) = \sigma(\vec{w}^{[2]T} \sigma(W^{[1]} \sigma(W^{[0]} \vec{x})))$$

Adding an hidden layer = Adding a matrix multiplication

This network has $5 \times (k+1) + 6 \times 3 + 1 \times 4$ **parameters**

kD, 2 Classes, 4 hidden layer network

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



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

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

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

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

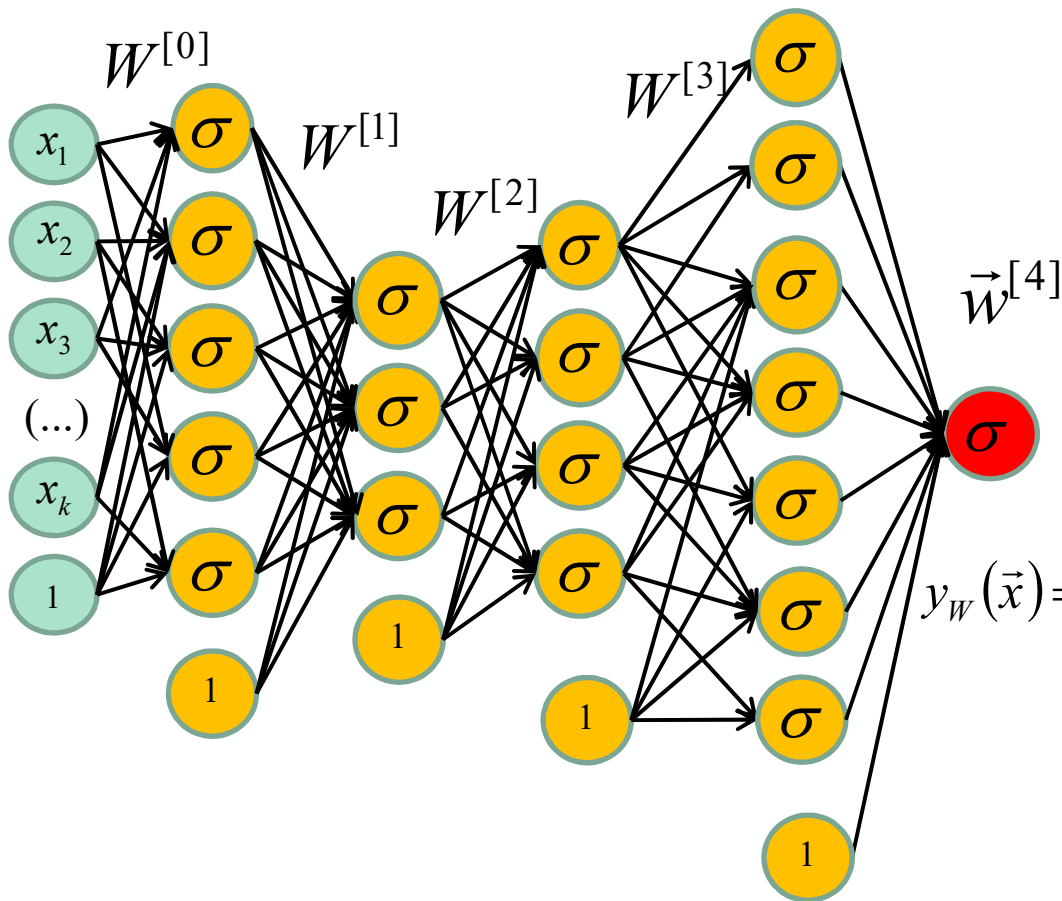
$$\vec{w}^{[4]} \in \mathbb{R}^8$$

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

This network has $5 \times (k+1) + 6 \times 3 + 4 \times 4 + 7 \times 5 + 1 \times 8$ **parameters**

kD, 2 Classes, 4 hidden layer network

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



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

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

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

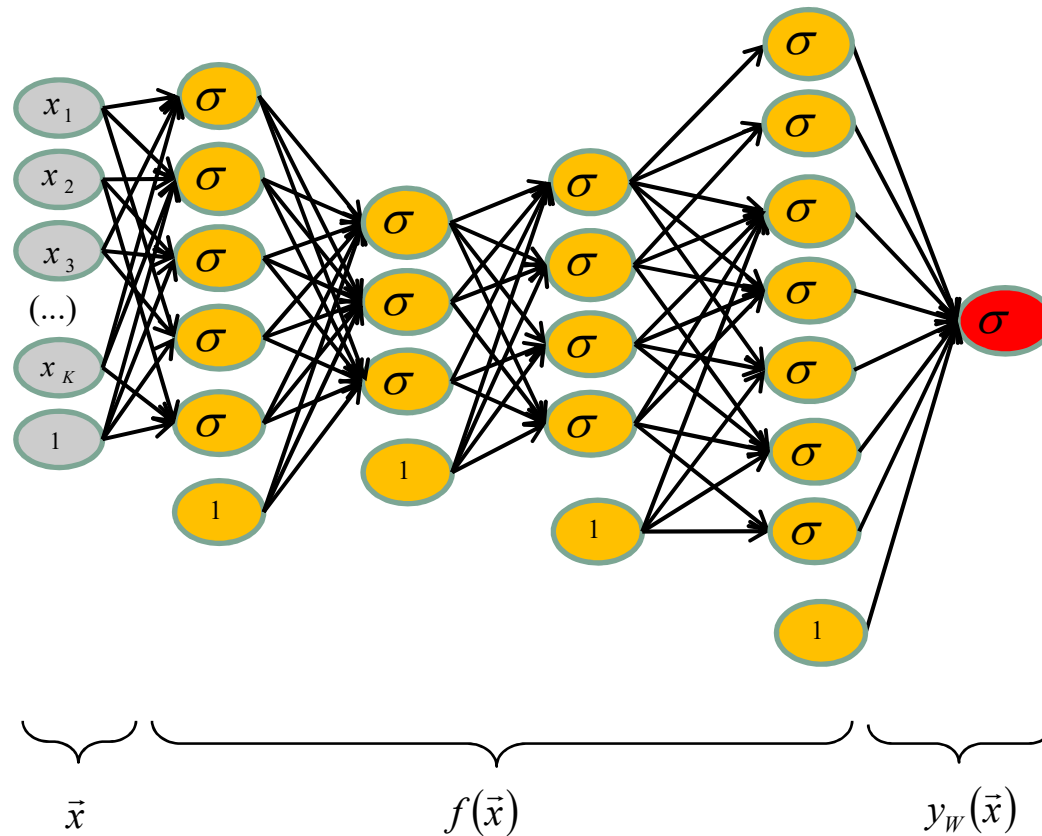
$$W^{[3]} \in \mathbb{R}^{7 \times 5}$$

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

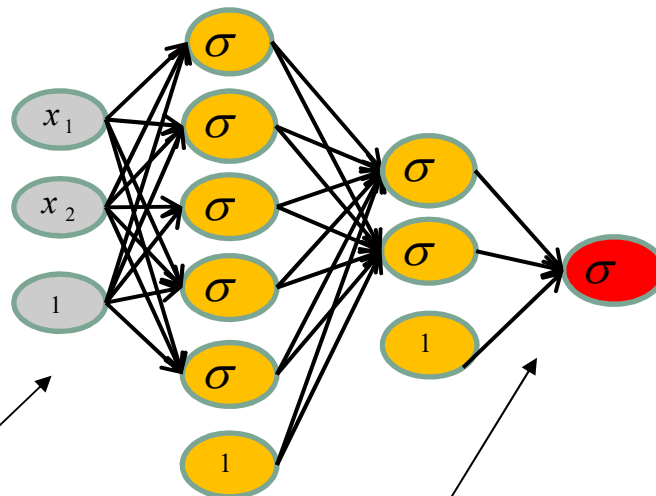
$$y_w(\vec{x}) = \sigma(\vec{w}^{[4]T} \sigma(W^{[3]} \sigma(W^{[2]} \sigma(W^{[1]} \sigma(W^{[0]} \vec{x}))))))$$

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

Multilayer Perceptron

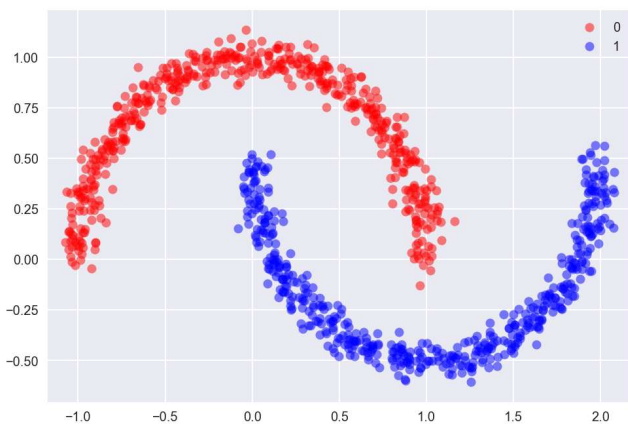


Example

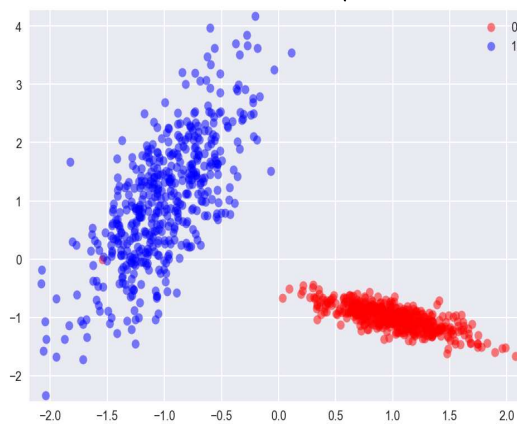


\vec{x}

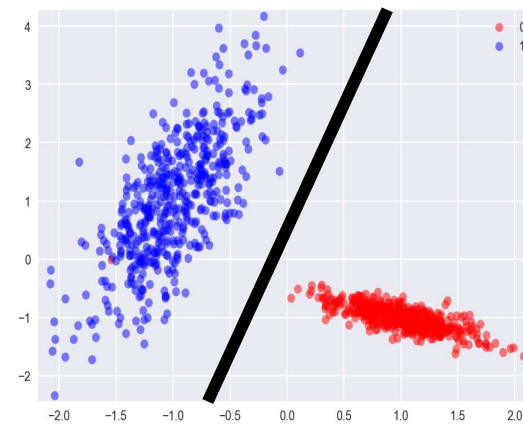
$y_W(\vec{x})$



Input data



Output of the last layer

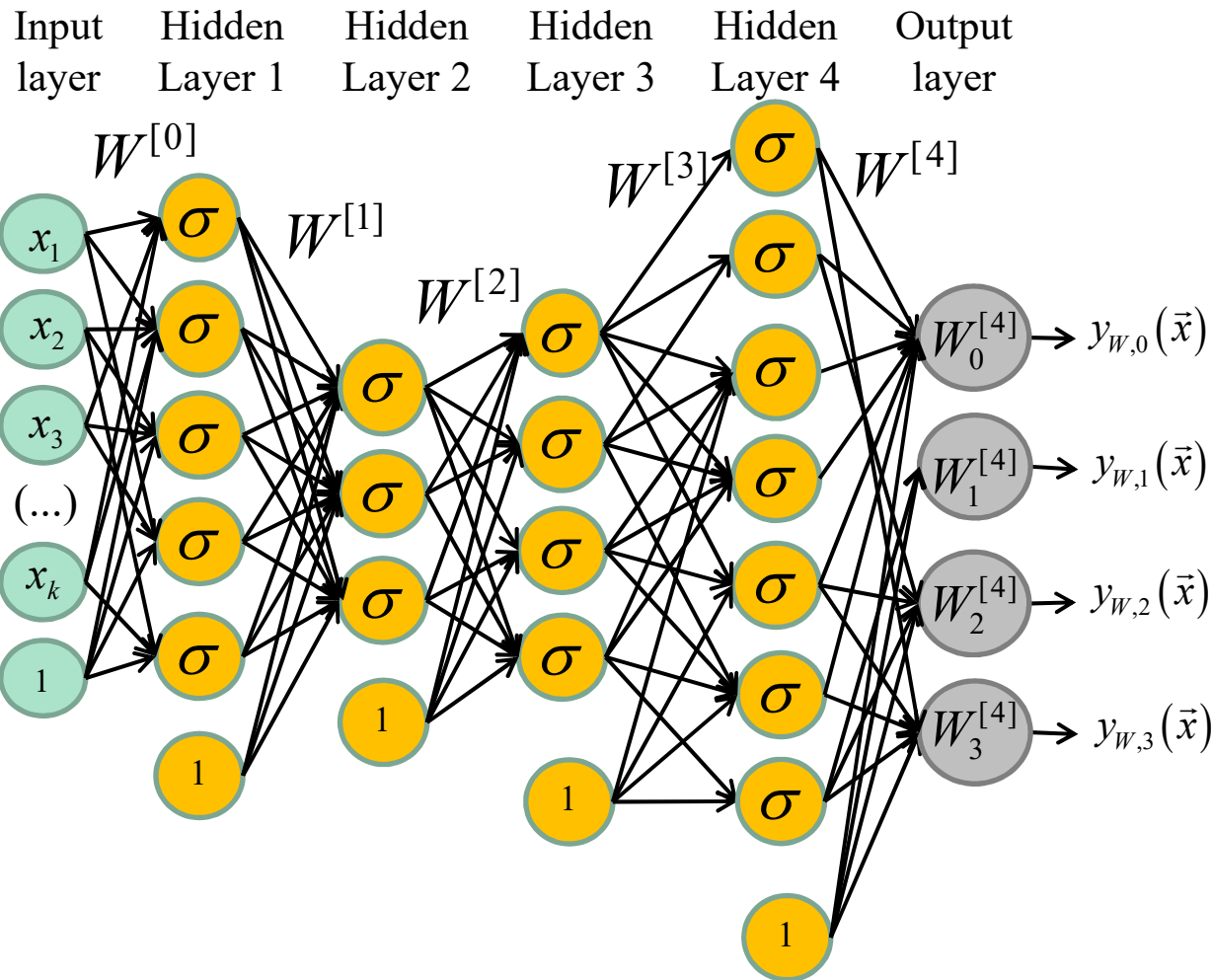


Output of the network



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

kD, 4 Classes, 4 hidden layer network



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

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

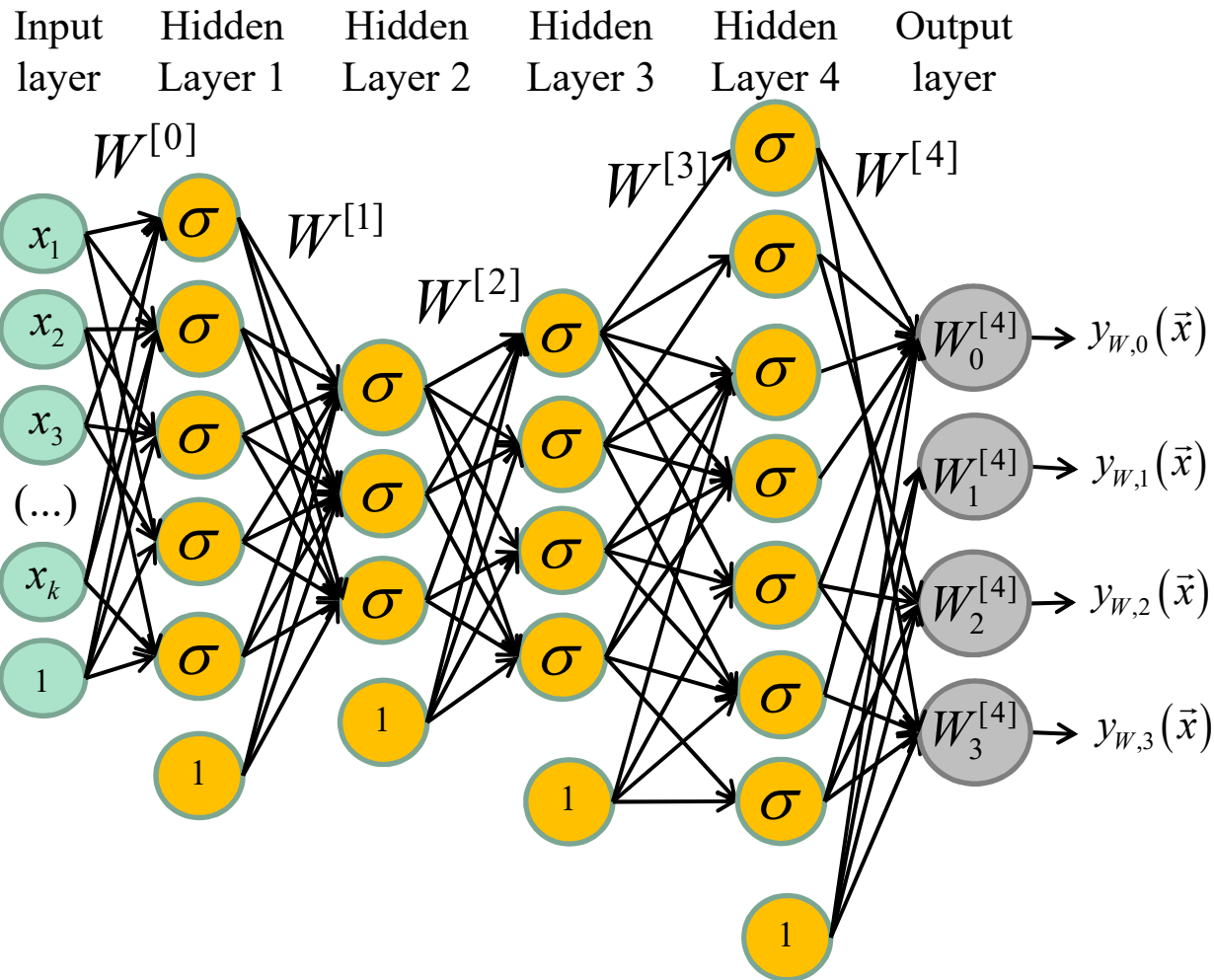
$$W^{[2]} \in R^{4 \times 4}$$

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

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

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

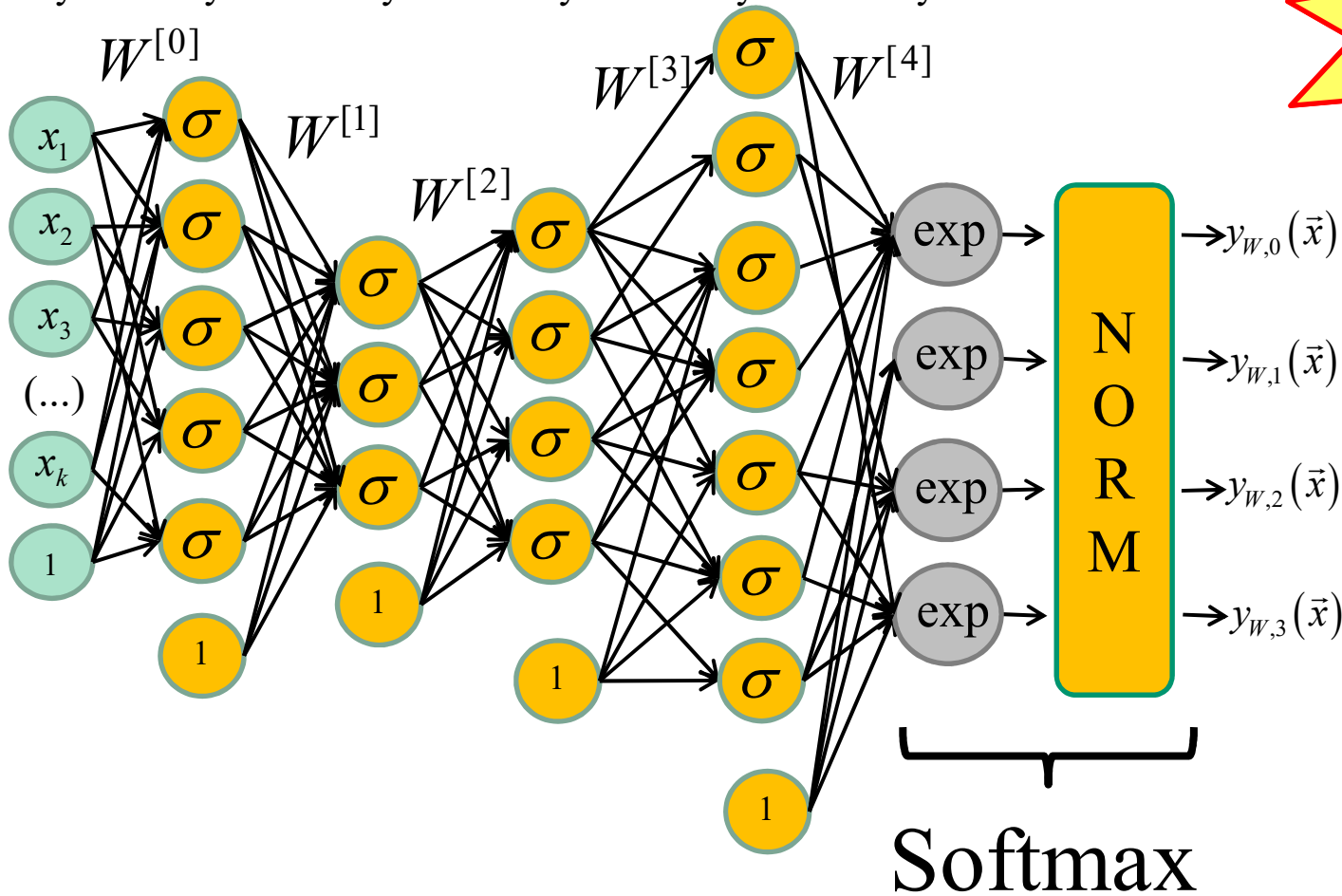
kD, 4 Classes, 4 hidden layer network



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

kD, 4 Classes, 4 hidden layer network

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



Cross entropy

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

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.

The word "Merci" is written in a cursive, hand-drawn style. It is surrounded by several short, radiating lines that emanate from the top and left sides of the letter 'M', giving it a sunburst or glowing appearance. The entire drawing is contained within a rounded rectangular border.

Merci

Extra slides

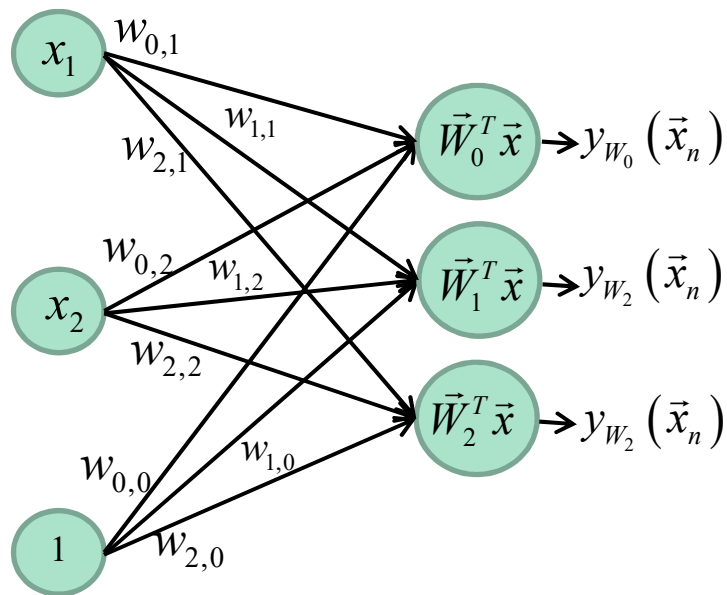
Better understand

Cross entropy vs Hinge loss

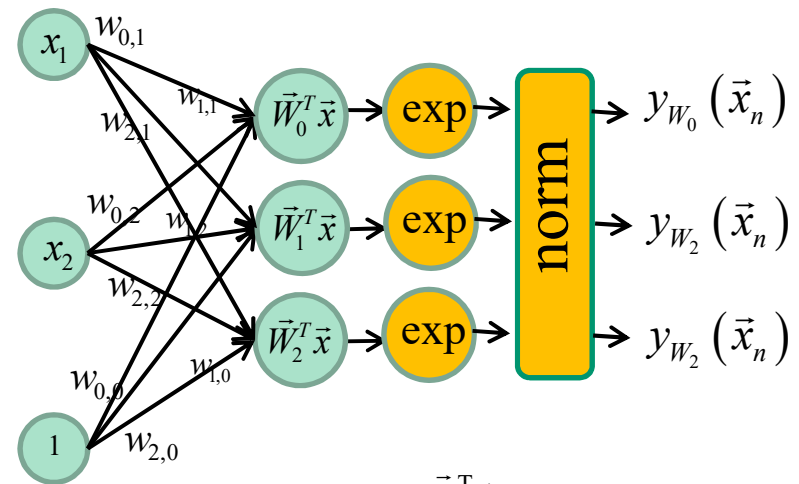
Cross entropy vs Hinge loss

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

- **Hinge loss** : output = matrix-vector
- **Cross entropy**: sortie = softmax



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



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

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

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

Hinge loss

Score

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

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

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)

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

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

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

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

Hinge loss

Score

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

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

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)

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

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

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

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

Hinge loss

Score

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

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

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)

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

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

0.0	0.01	-0.05	0.1	0.05	1
0.2	0.7	0.2	0.05	0.16	-15
-0.3	0.0	-0.45	-0.2	0.03	22
					-44
					56

Hinge loss

Score

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

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

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?

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

0.0	0.01	-0.05	0.1	0.05	1
0.2	0.7	0.2	0.05	0.16	-15
-0.3	0.0	-0.45	-0.2	0.03	22
					-44
					56

Hinge loss

Score

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

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

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)

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

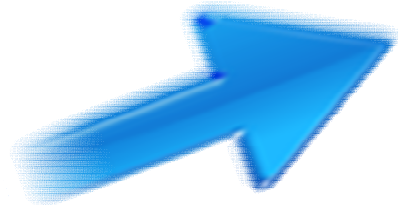
Q4: what would happen to the total loss if we were to add an L2 regularization?



Increase the number
of neurons



Increase the number
of layers

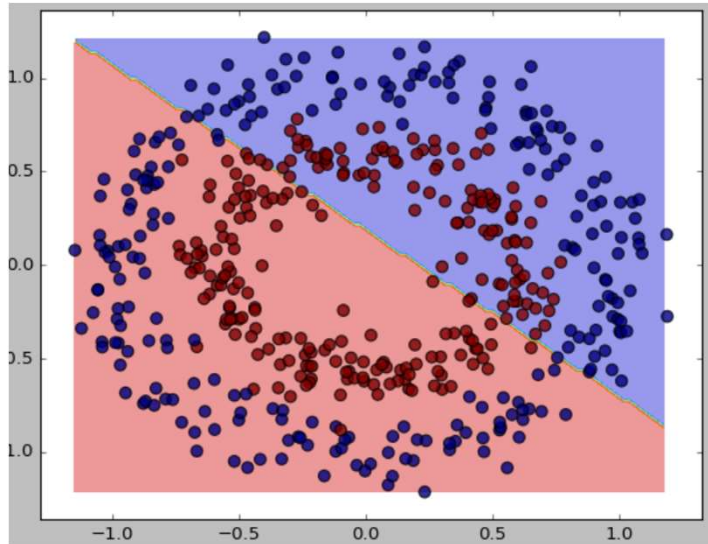


**Increase the
capacity of the network**

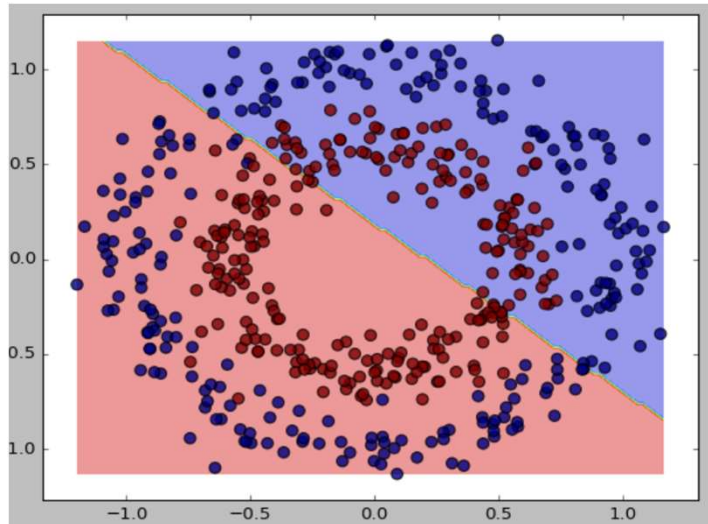


Increasing the capacity of the network
can lead to **over-fitting**

Under-fitting

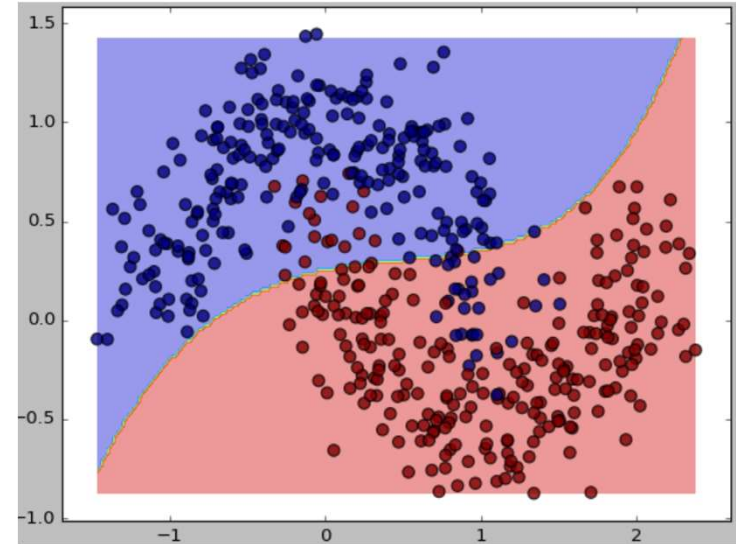


Precision on the training set = 52.2%

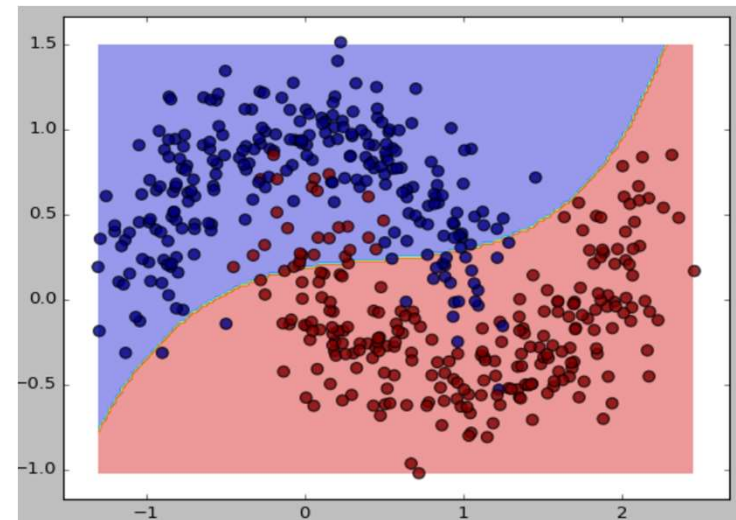


Precision on the test set = 51.2%

Could do better...

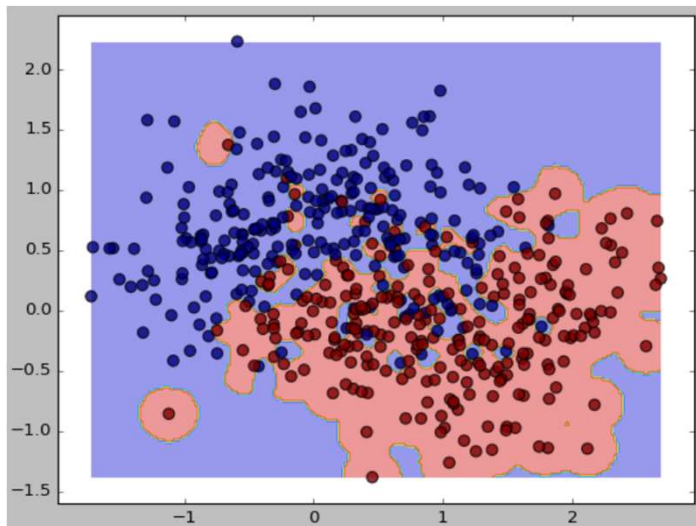


Precision on the training set = 82%

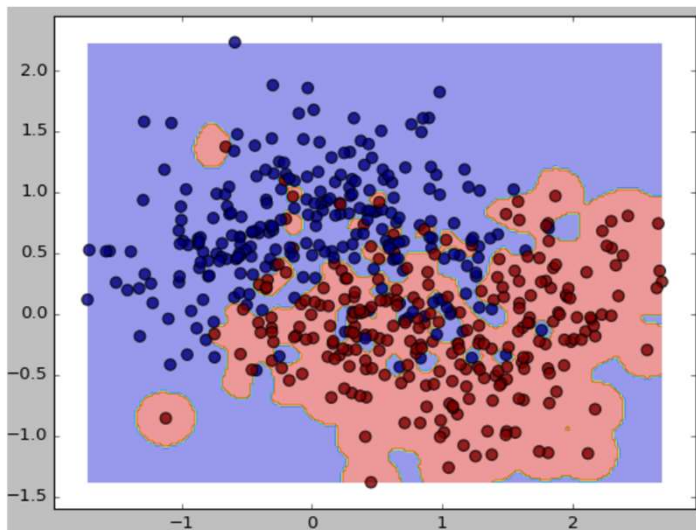


Precision on the test set = 80%

Overfitting

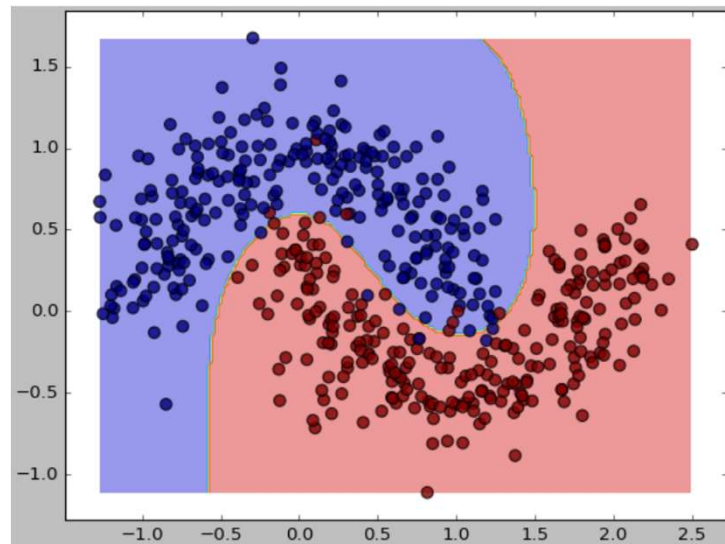


Precision on the training set = 99.6%

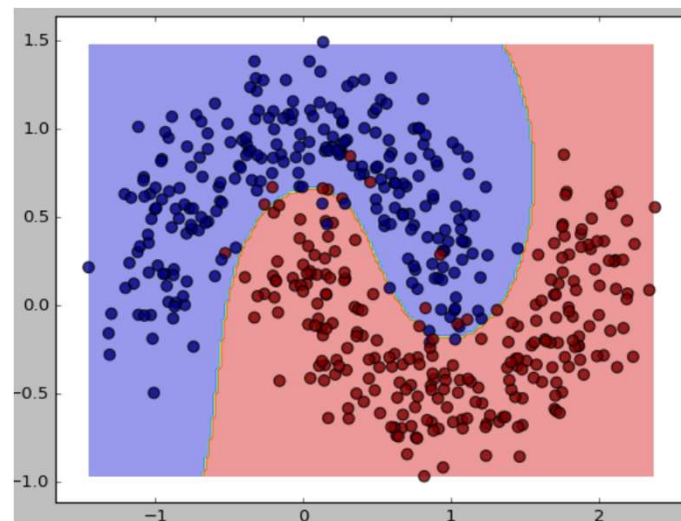


Precision on the test set = 78%

SUPER !!!



Precision on the training set = 97.8%



Precision on the test set = 96.2%