Spring School on

Deep Learning for Medical Imaging (DLMI2023)

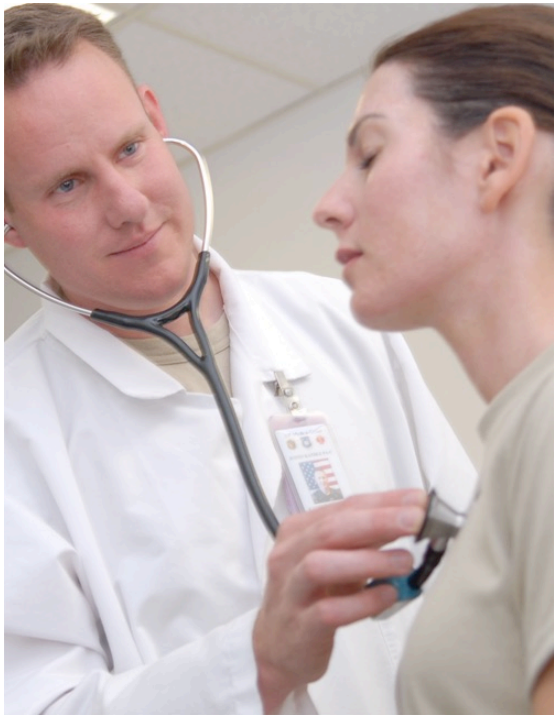
Lyon, France | April 17, 2023

Basics of deep learning : Part I

Christian Desrosiers

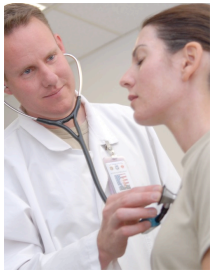
ÉTS Montreal, Canada

Let’s start with a simple example



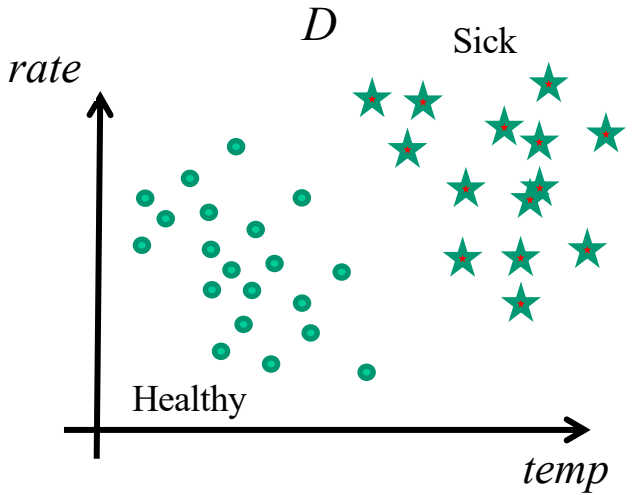
From Wikimedia Commons
the free media repository

Let's start with a simple example



D

	(*temp*, *rate*)	*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

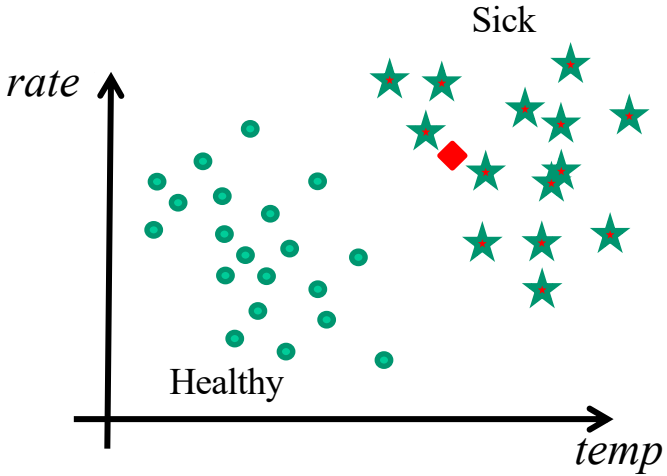


Let's start with a simple example

A new patient comes to the hospital

How can we determine if he is sick or not?

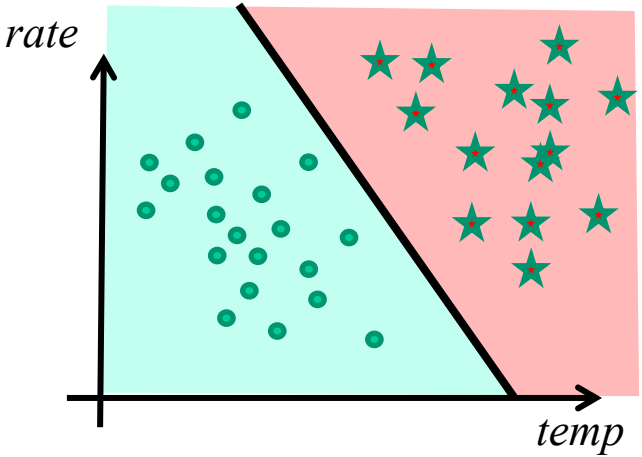
From Wikimedia Commons
the free media repository



Solution

From Wikimedia Commons
the free media repository

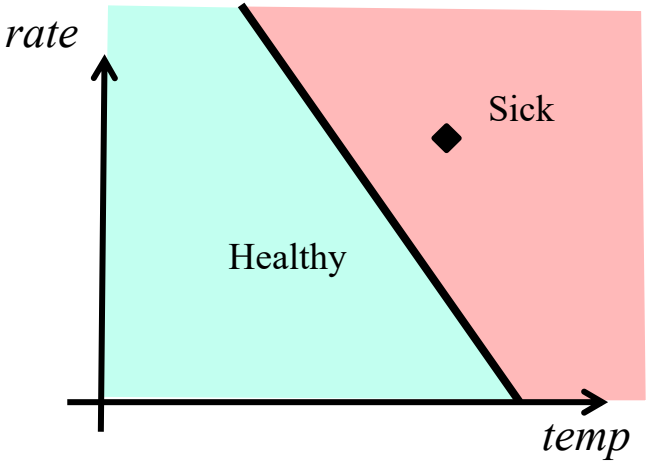
Divide the feature space in 2 regions : Sick and Healthy



Solution

From Wikimedia Commons
the free media repository

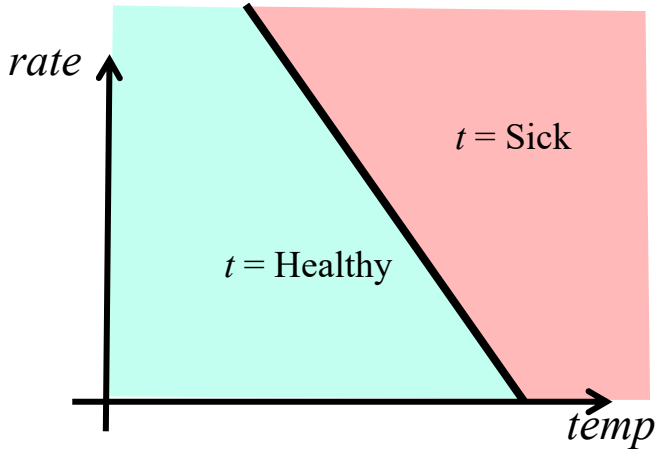
Divide the feature space in 2 regions : Sick and Healthy



More formally

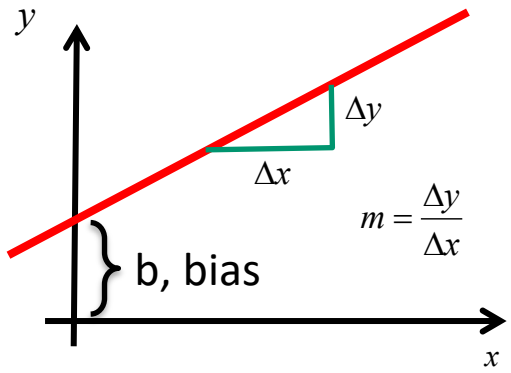
From Wikimedia Commons
the free media repository

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



How to split the feature space?

Definition … a line!

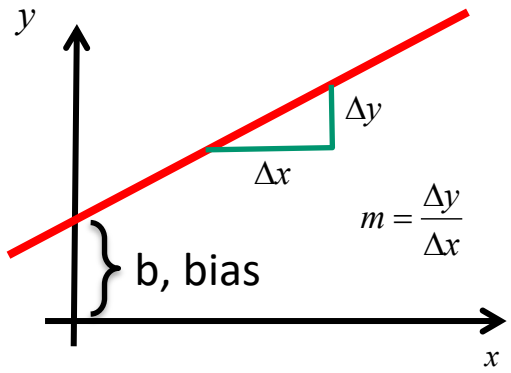


$$y = mx + b$$

slope

bias

Definition … a line!



$$y = mx + b$$

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

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

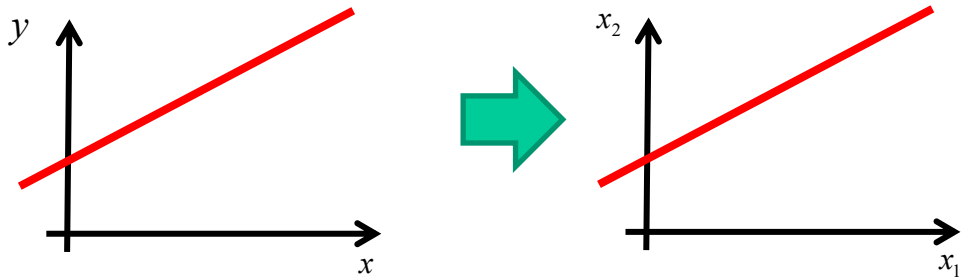
$$0 = \Delta y x - \Delta x y + b\Delta x$$

Rename variables

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

Rename variables

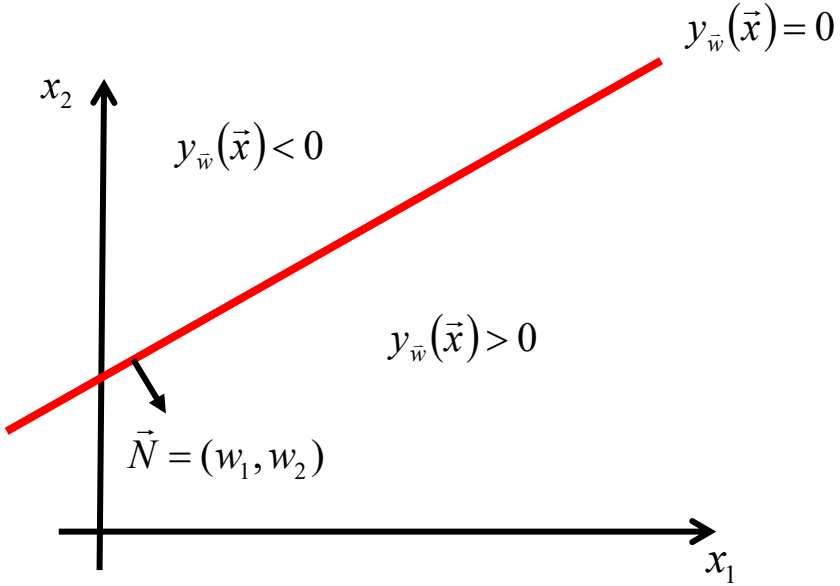
$$0 = w_1 x + w_2 y + w_0$$



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

Classification function

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



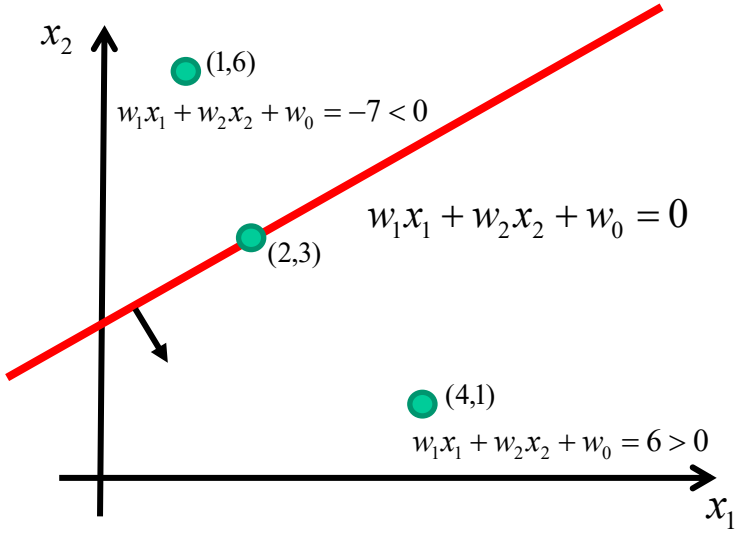
Classification function

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

$w_1 = 1.0$

$w_2 = -2.0$

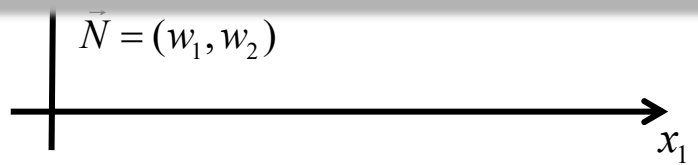
$w_0 = 4.0$



Classification function

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

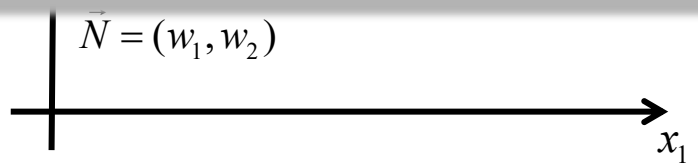
DOT product



Classification function

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

DOT product



Linear classification = Dot product with bias included

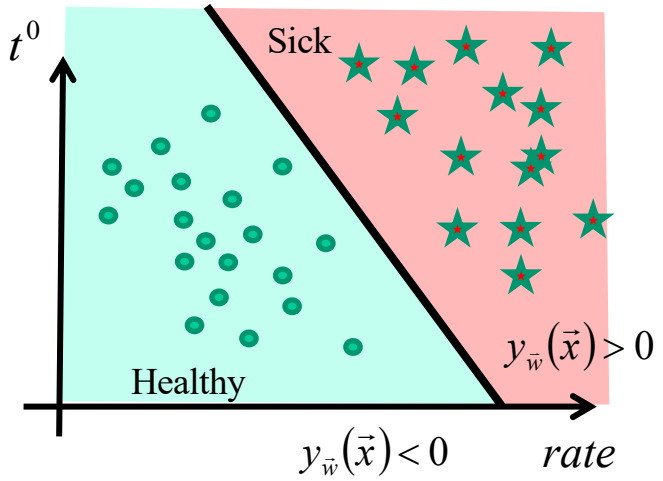
$$y_{\vec{w}}\left(\vec{x}\right) = \vec{w}^{T}\vec{x}$$

Learning

With the training dataset D

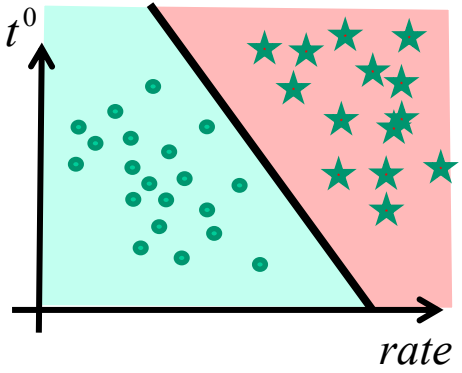
the GOAL is to

Find the parameters (w_0, w_1, w_2) that best separates the two classes



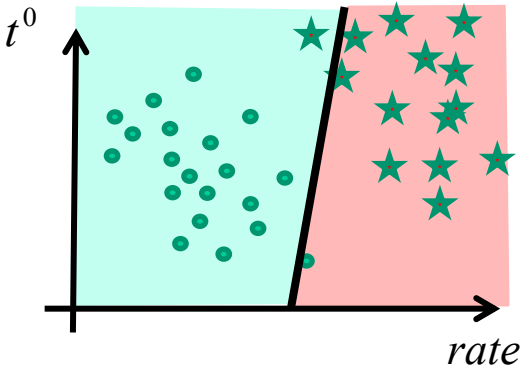
How do we know
if a model is good?

Loss function



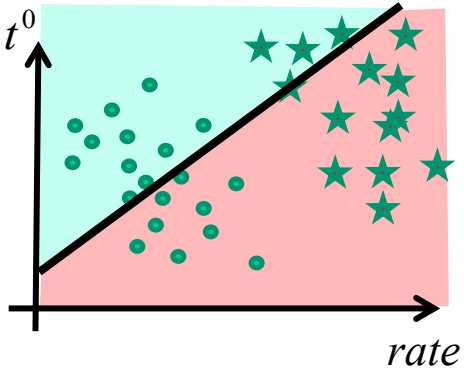
$L(y_{\vec{w}}(\vec{x}), D) \approx 0$

Good!



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

Medium



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

BAD!

Training a model

Finding the right parameters w_0, w_1, w_2 such that

PATIENTS are WELL CLASSIFIED

SMALL LOSS

So far...

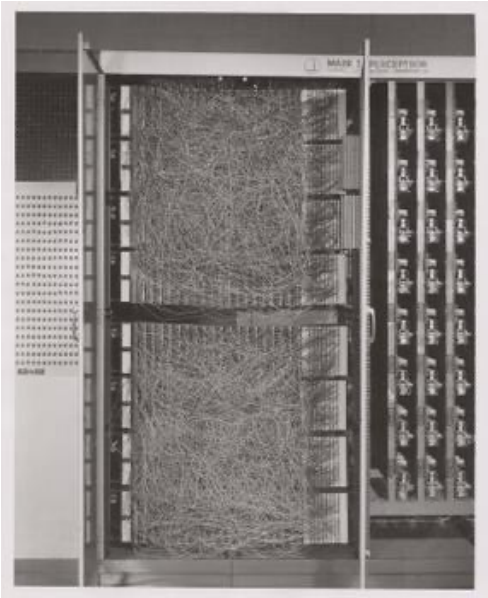
1. Training dataset: D
2. Classification function (a line in 2D) : $y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D)$

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

In this session…

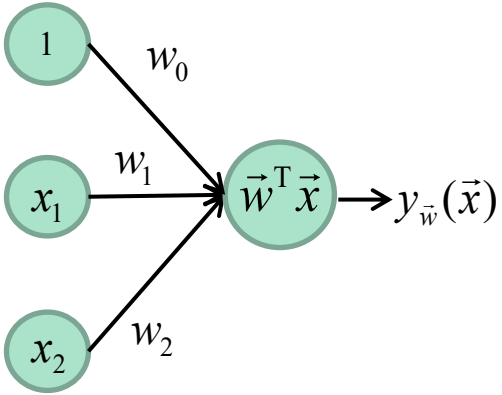
Perceptron
Logistic regression
Multi-layer perceptron

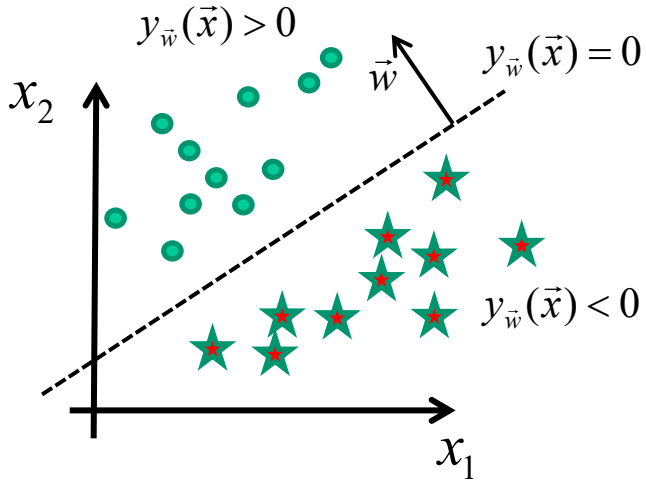
Perceptron



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

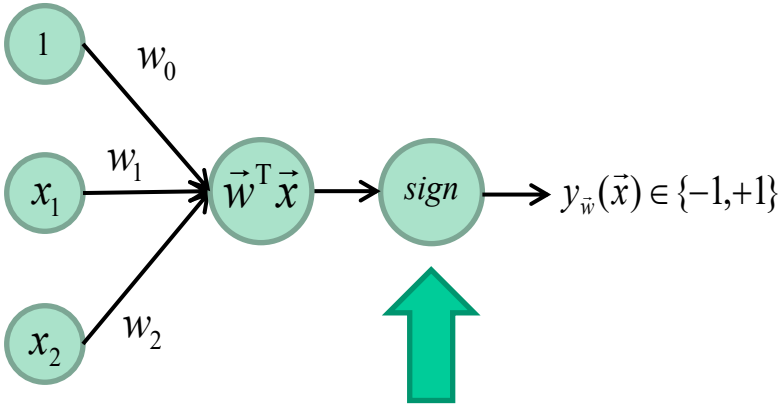
Perceptron
(2D and 2 classes)



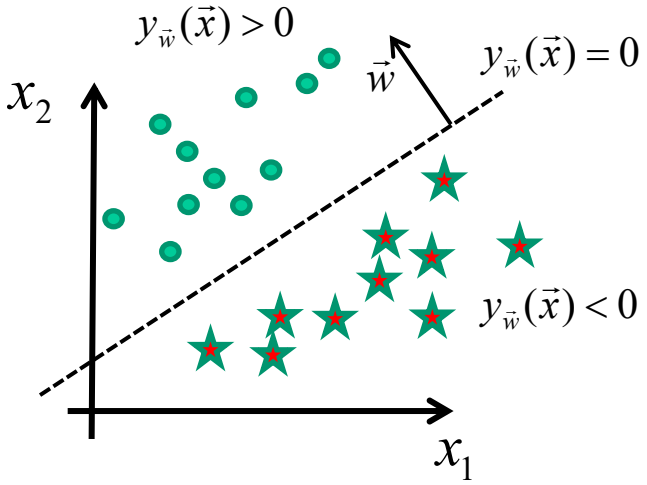


$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_1$$
$$= \vec{w}^{\mathrm{T}}\vec{x}$$

Perceptron
(2D and 2 classes)

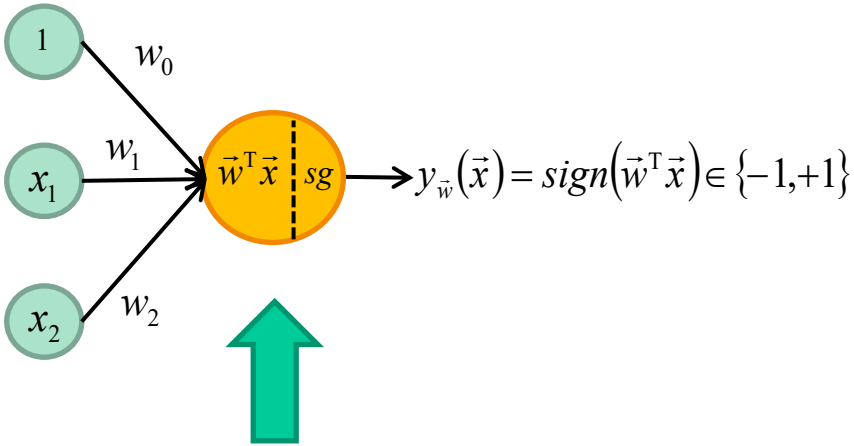


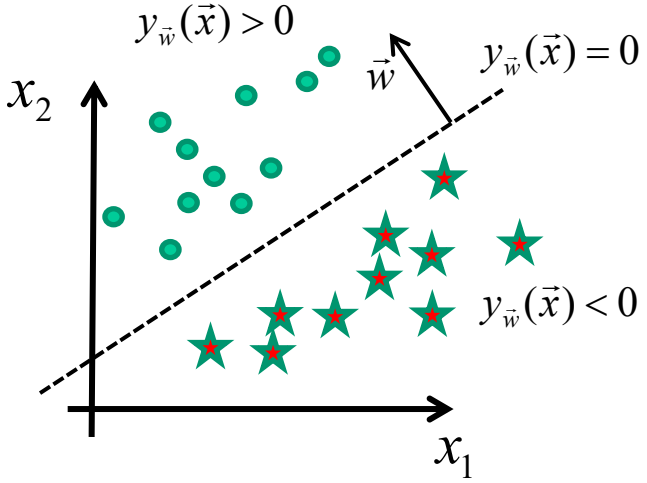
Activation function



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

Perceptron
(2D and 2 classes)





Neuron

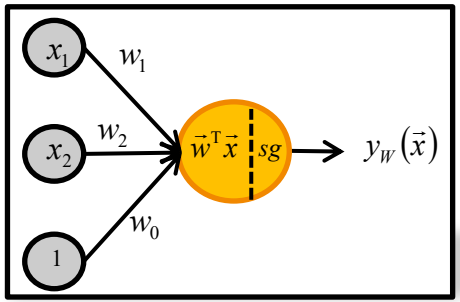
Dot product + activation function

So far...

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

So far...

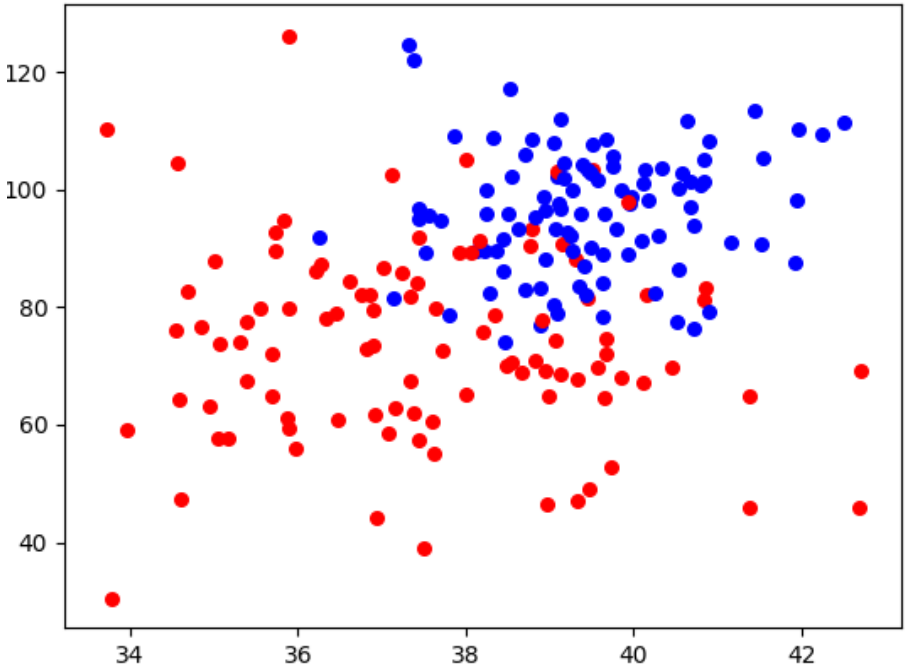
1. Training dataset: D
2. Classification function (a line in 2D) :
3. Loss function: $L(y_{\vec{w}}(\vec{x}), D)$



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

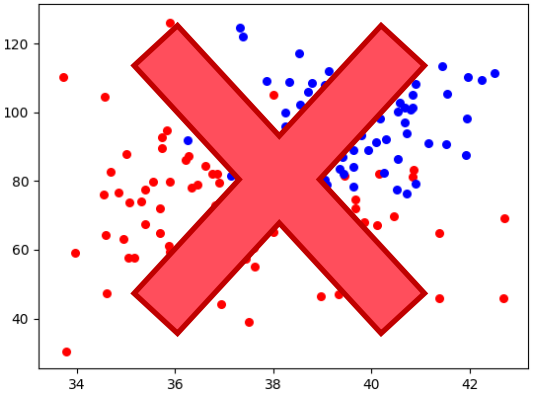
D

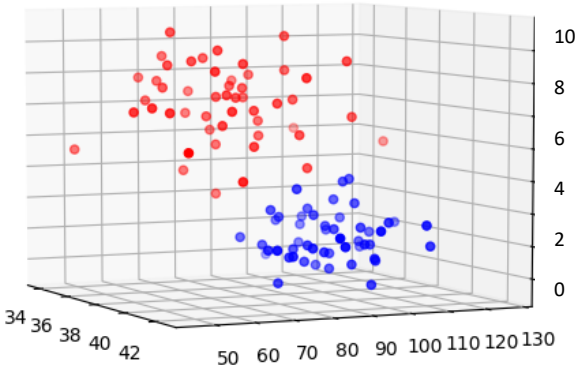
	(temp, rate)	diagnostic
Patient 1	(37.5, 72)	healthy
Patient 2	(39.1, 103)	sick
Patient 3	(38.3, 100)	sick
	(…)	…
Patient N	(36.7, 88)	healthy
	$\vec{x}$	t

D

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

Non-linearly separable training data





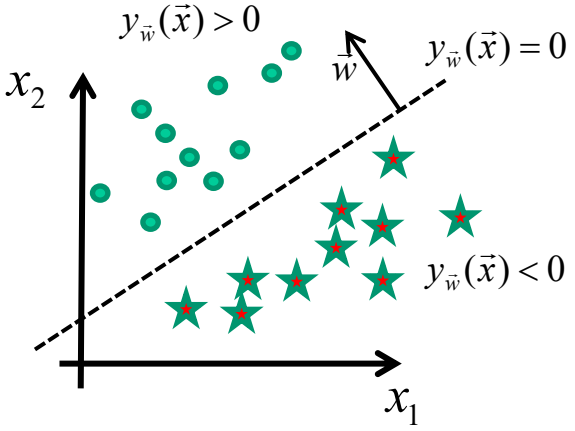
$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

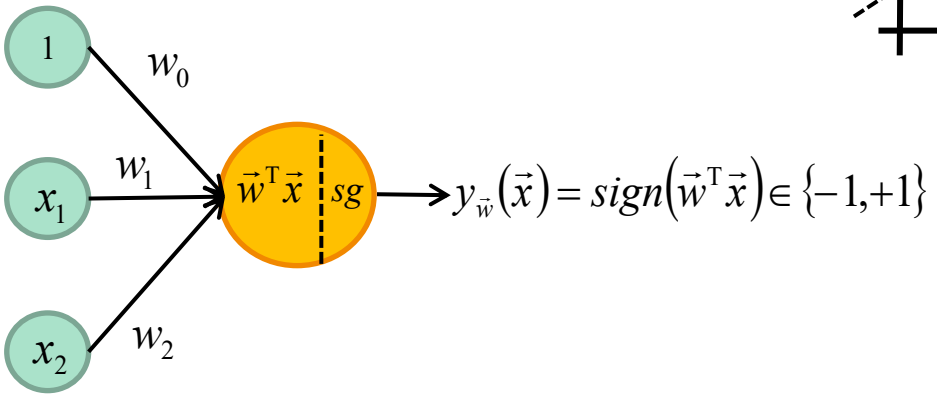
(line)

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

(plane)

Perceptron
(2D and 2 classes)



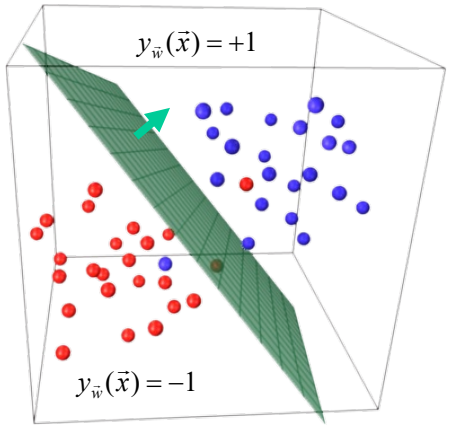


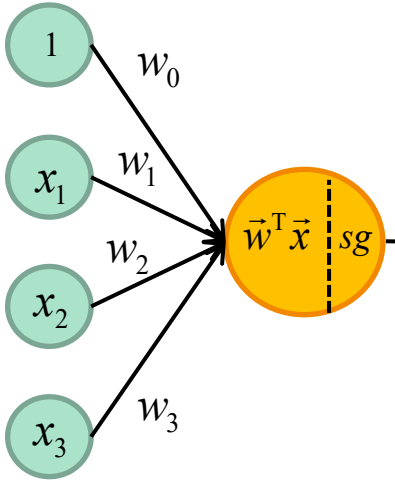
$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

(line)

Perceptron
(3D and 2 classes)

3D example



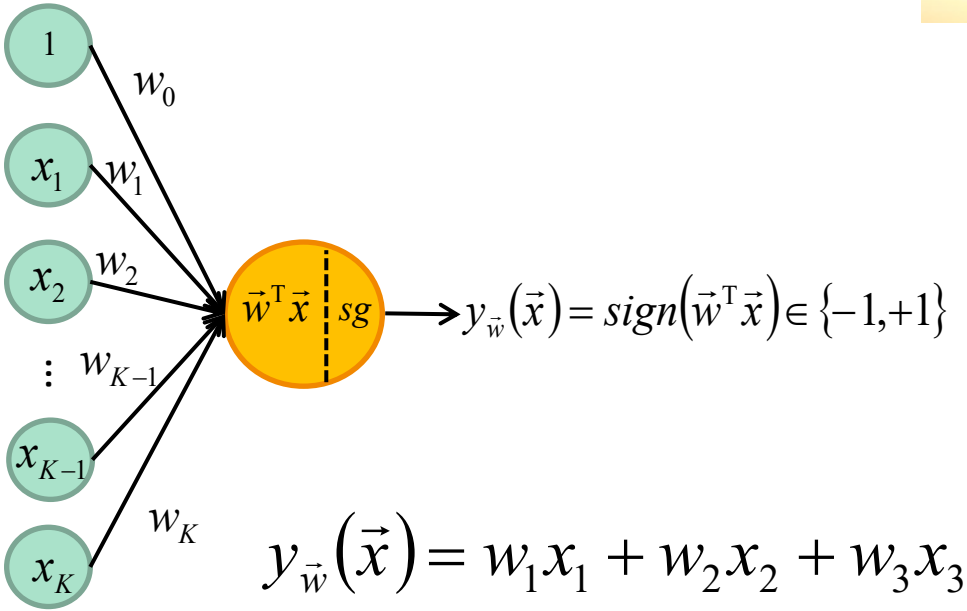


$$y_{\vec{w}}(\vec{x}) = sign(\vec{w}^T\vec{x}) \in \{-1,+1\}$$

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

(plane)

Perceptron
(K-D and 2 classes)

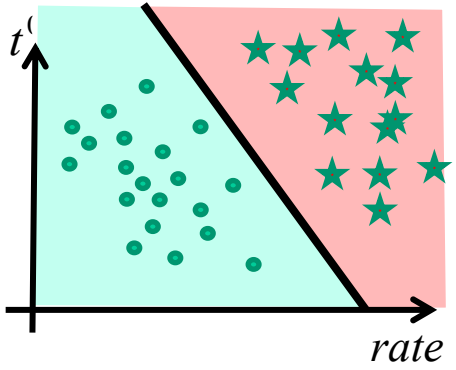


$$y_{\vec{w}}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_Kx_K + w_0$$

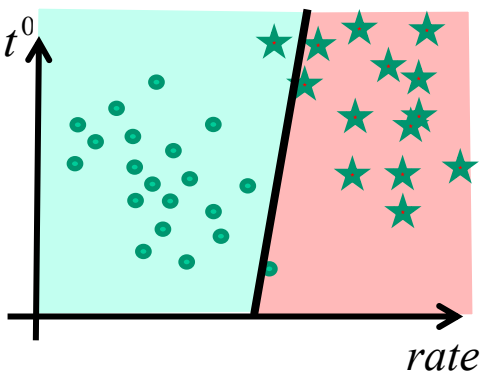
(hyperplane)

How do we train a Perceptron?

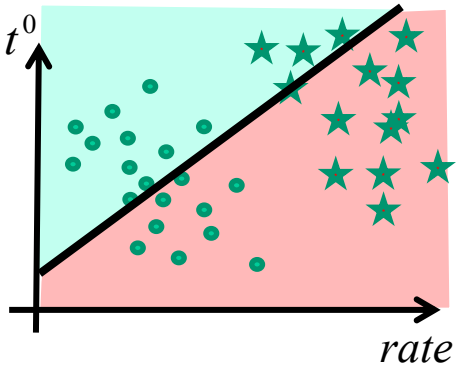
Loss function



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



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



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

Learning a model

Goal: with a set of training data $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$

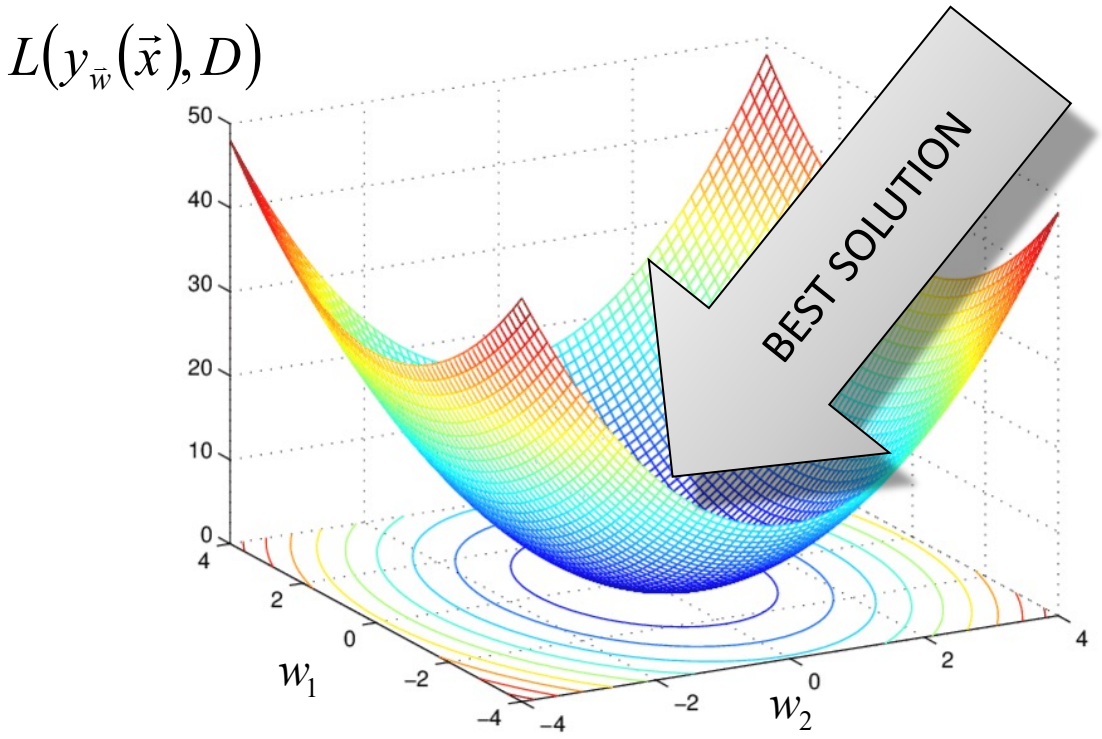
We want to estimate the function y so that

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

Minimizes the **training loss**

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

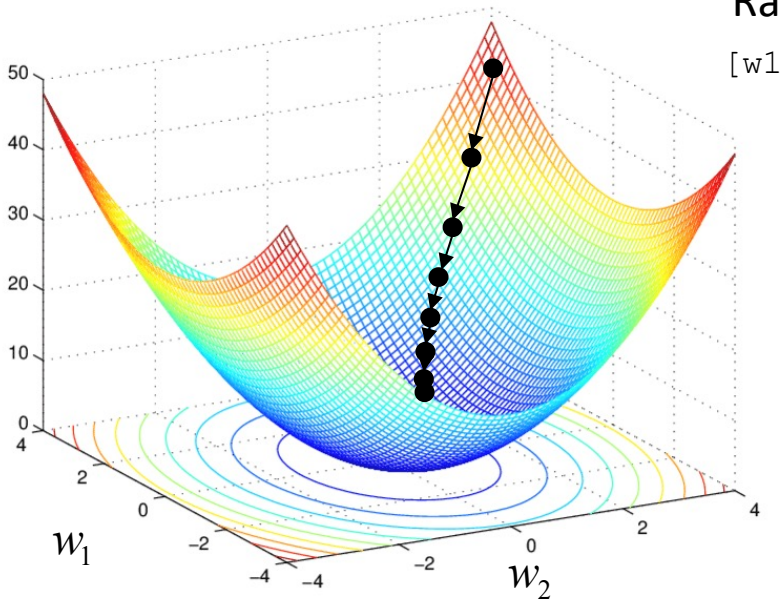
Optimization problem


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

Perceptron

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

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



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

Perceptron Criterion (loss)

Observation

A wrongly classified sample is when

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

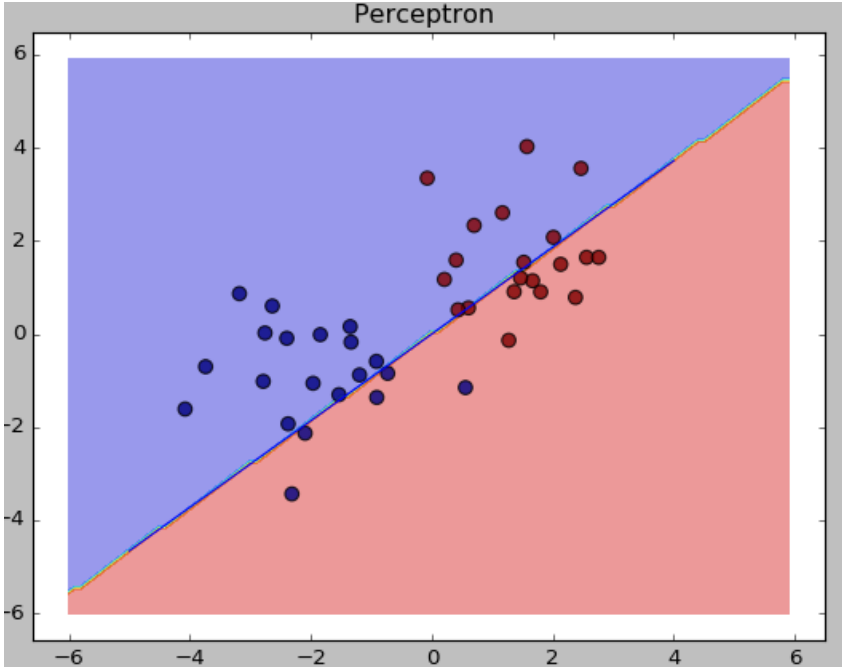
or

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

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

Perceptron Criterion (loss)

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



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

Optimization

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

Gradient of the loss function

Learning rate

Stochastic gradient descent (SGD)

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

UNTIL every data is well classified or k== MAX_ITER

Perceptron 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

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

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

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

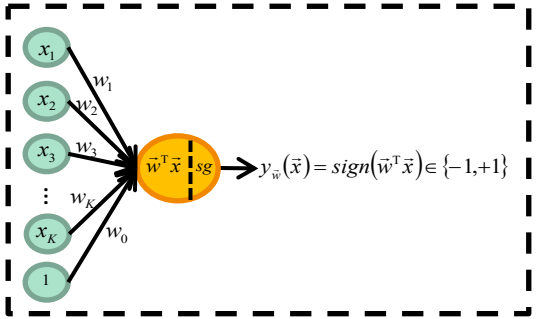
$$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{n=1}^{N} \max\left(0, 1 - t_n \vec{w}^T \vec{x}_n\right)$$ "Hinge Loss" or "SVM" Loss

So far…

1. Training dataset: D
2. Linear classification function: $y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_M x_M + w_0$
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…

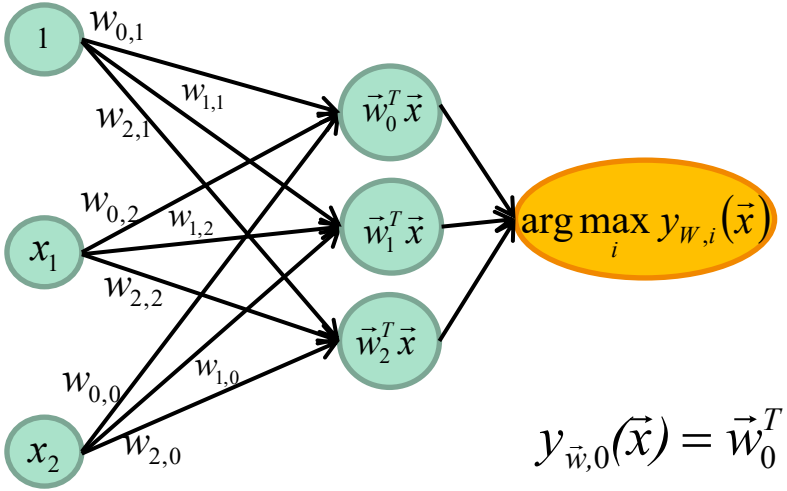
1. Training dataset: D
2. Linear classification function:
3. Loss function: $L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{\vec{x}_n \in V} -\vec{w}^{\mathrm{T}} \vec{x}_n t_n$

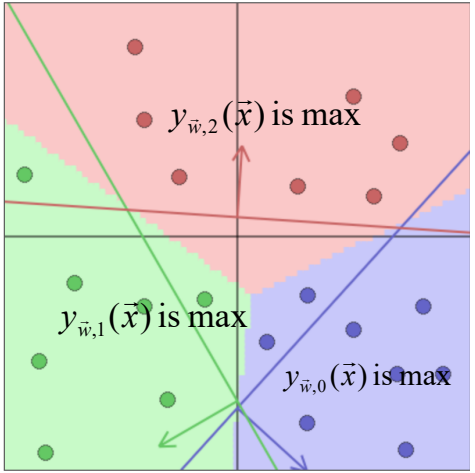


4. Training : find $\vec{w}$ that minimizes $L\left(y_{\vec{w}}\left(\vec{x}\right), D\right)$

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

Multiclass Perceptron
(2D and 3 classes)



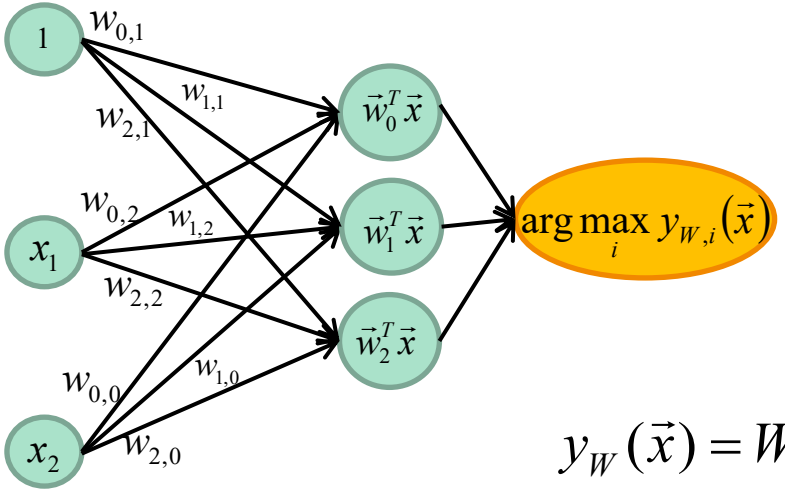


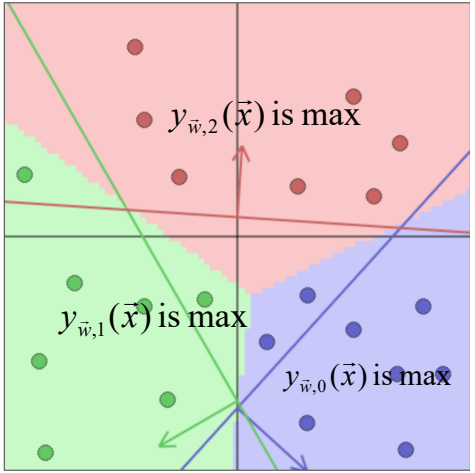
$$y_{\vec{w},0}(\vec{x}) = \vec{w}_0^T\vec{x} = w_{0,0} + w_{0,1}x_1 + w_{0,2}x_2$$

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

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

Multiclass Perceptron
(2D and 3 classes)



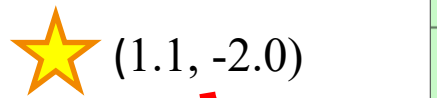


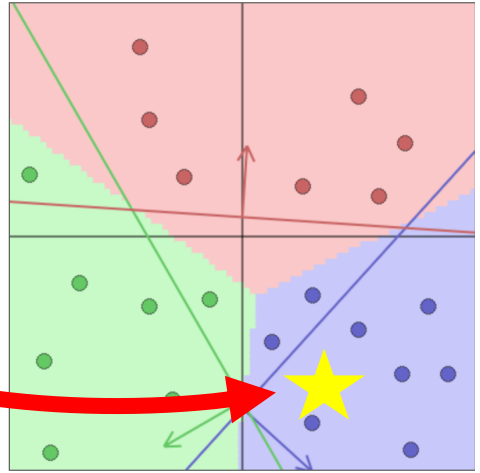
$$y_W(\vec{x}) = W\vec{x}$$

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

Multiclass Perceptron
(2D and 3 classes)

Example





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

Class 0

Class 1

Class 2

Multiclass Perceptron

Loss function

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

Sum over all wrongly classified samples

Score of the wrong class

Score of the true class

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

Multiclass Perceptron

Stochastic gradient descent (SGD)

init W

k=0, i=0

DO k=k+1

FOR n = 1 to N

$j = \arg\max \mathrm{W}^T \vec{x}_n$

IF $j \neq t_i$ THEN /* wrongly classified sample */

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

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

UNTIL every data is well classified or k > K_MAX.

Perceptron

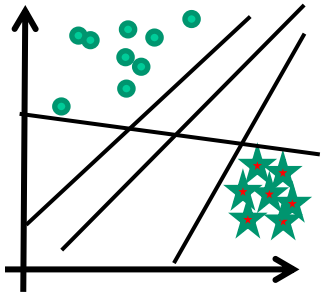
Advantages

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

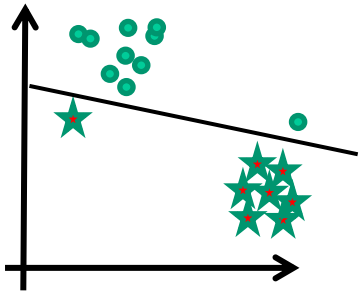
Limitations

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

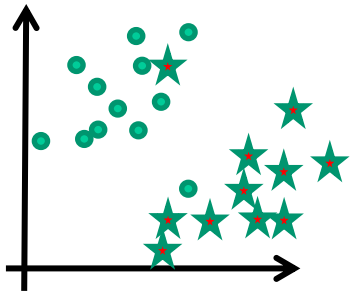
Many “optimal” solutions



Suboptimal solution



Will never converge



Two famous ways of improving the Perceptron

1. New activation function + new Loss

Logistic regression

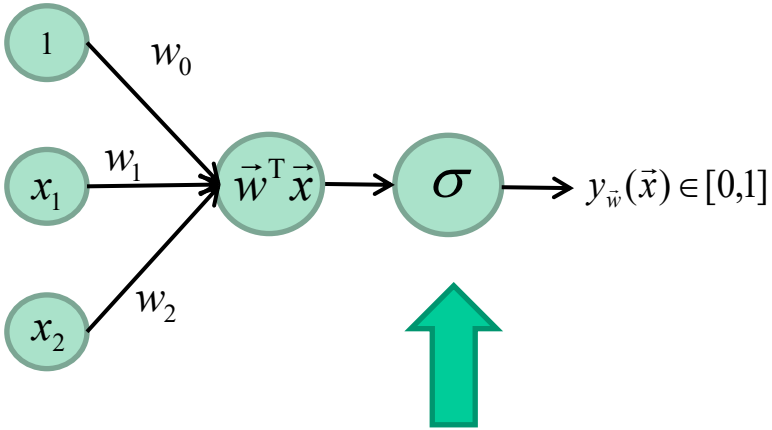
2. New network architecture

Multilayer Perceptron / CNN

Logistic regression

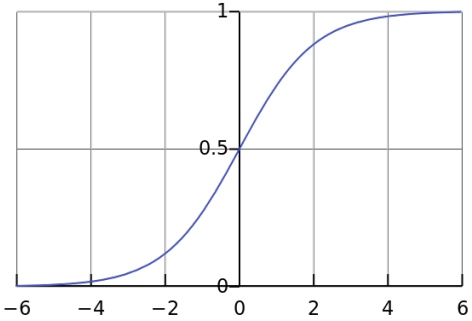
Logistic regression
(2D, 2 classes)

New activation function: sigmoid



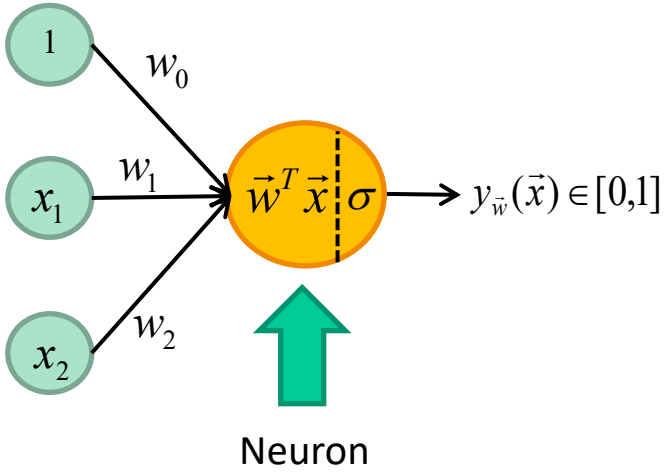
Sigmoid activation function

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

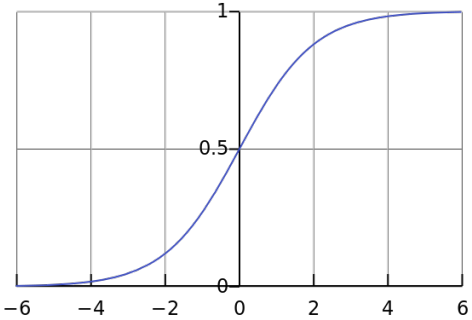


Logistic regression
(2D, 2 classes)

New activation function: sigmoid



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

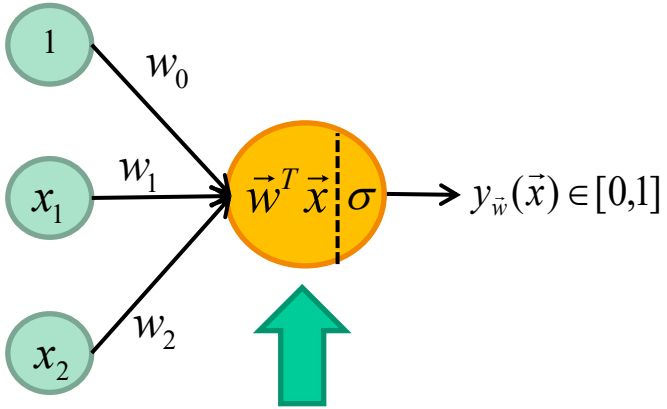


$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T\vec{x}\right) = \frac{1}{1+e^{-\vec{w}^T\vec{x}}}$$

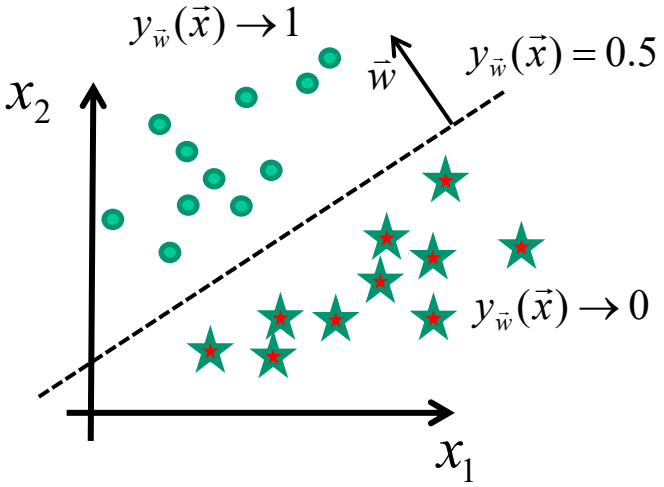
Logistic regression

(2D, 2 classes)

New activation function: sigmoid



Neuron



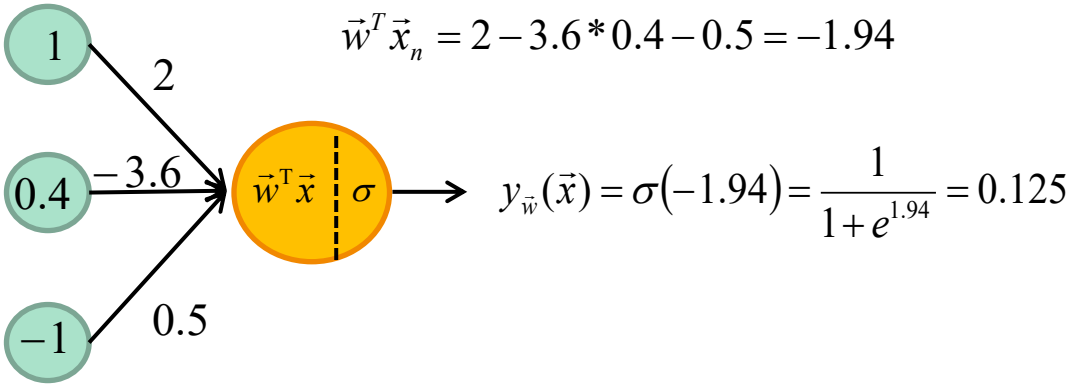
$$y_{\vec{w}}(\vec{x}) = \sigma\left(\vec{w}^T \vec{x}\right)$$

Logistic regression

(2D, 2 classes)

Example

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

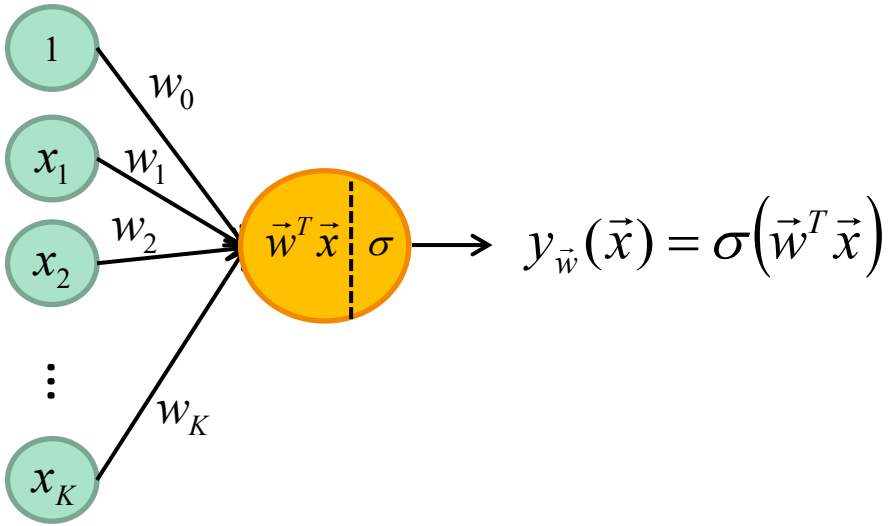


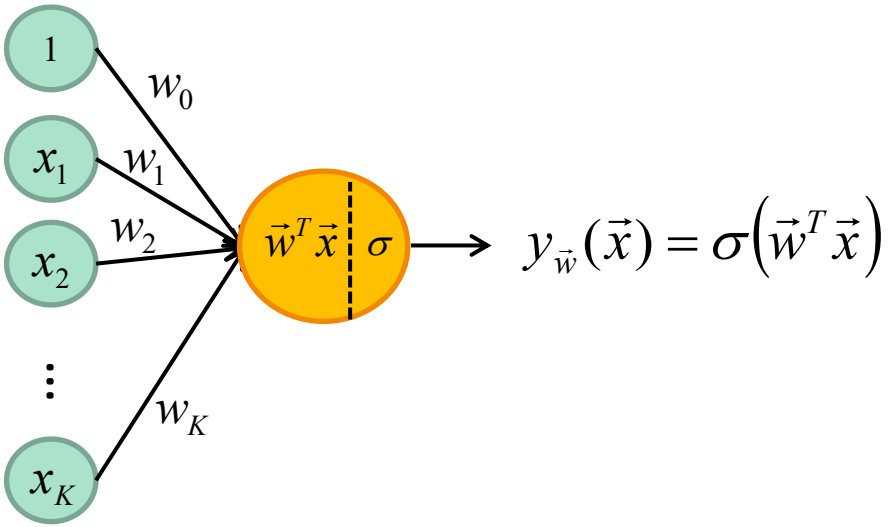
Since 0.125 is lower than 0.5, $\vec{x}_n$ is <u>behind</u> the plane.

Logistic regression

(K-D, 2 classes)

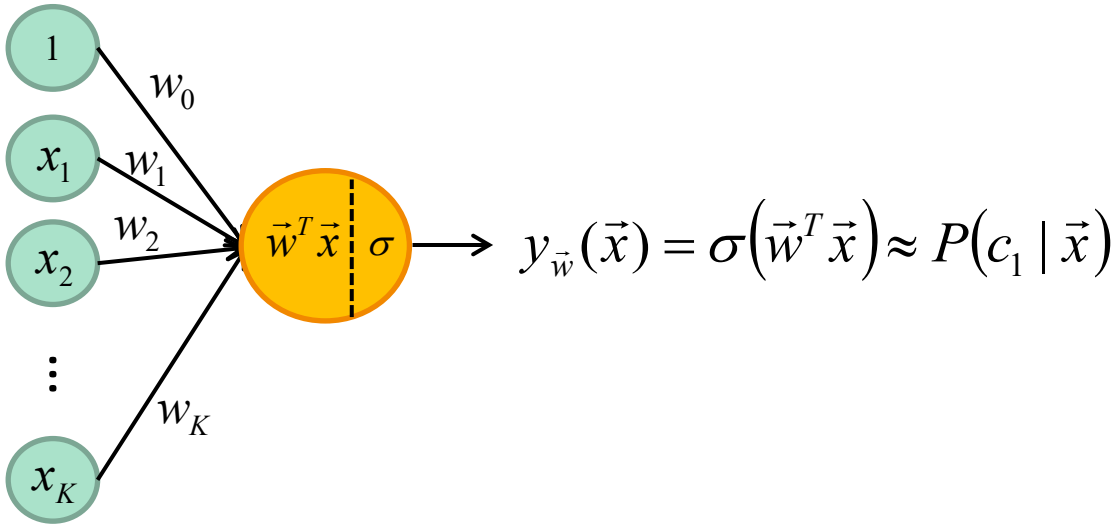
Like the Perceptron the logistic regression accomodates for K-D vectors



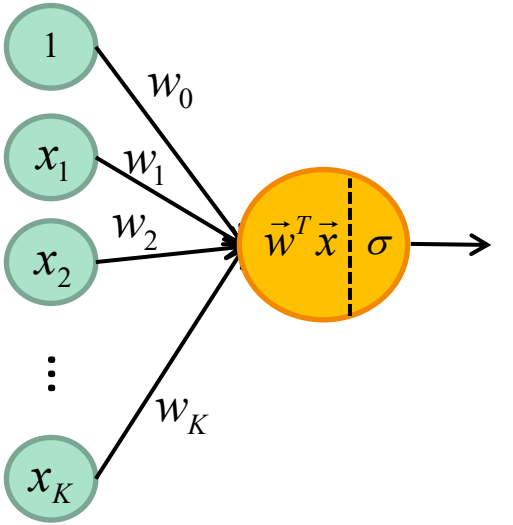


What is the loss function?

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



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 simply the negative log likelihood

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

2 Class Cross entropy

We can also show that

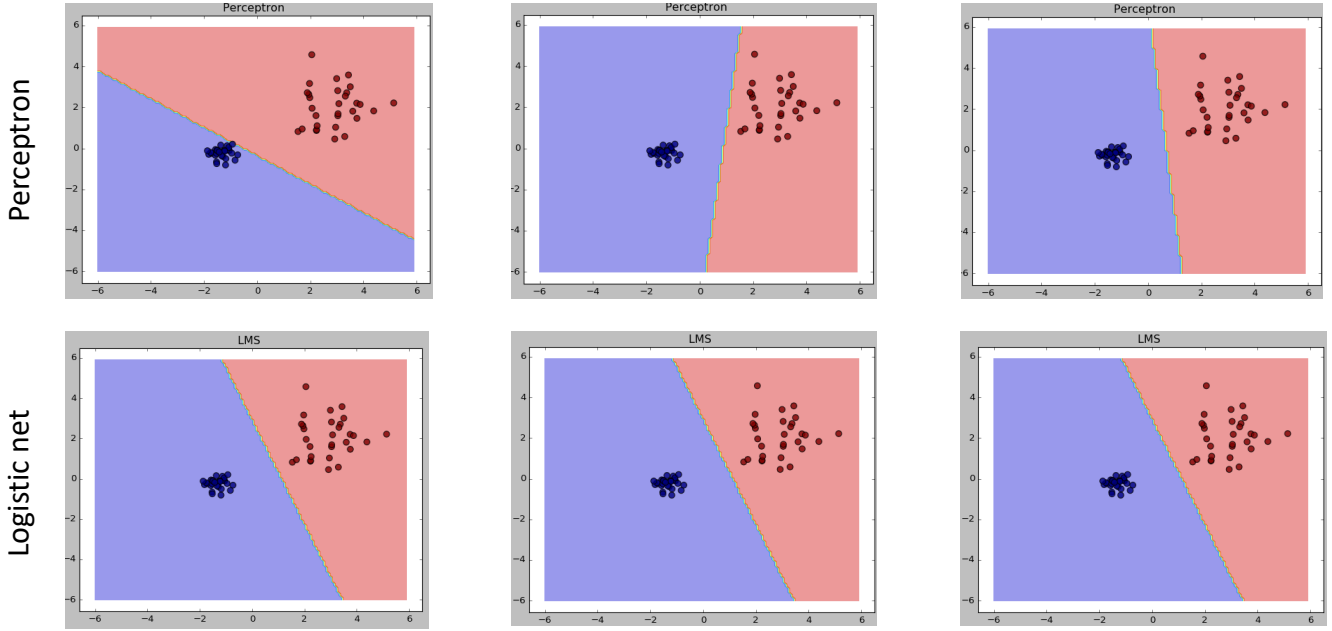
$$\frac{dL\left(y_{\vec{w}}\left(\vec{x}\right), D\right)}{d\vec{w}} = \sum_{n=1}^{N}\left(y_{\vec{w}}\left(\vec{x}_n\right) - t_n\right)\vec{x}_n$$

Unlike for the Perceptron,
the gradient does not depend
on the wrongly classified samples

Logistic Network

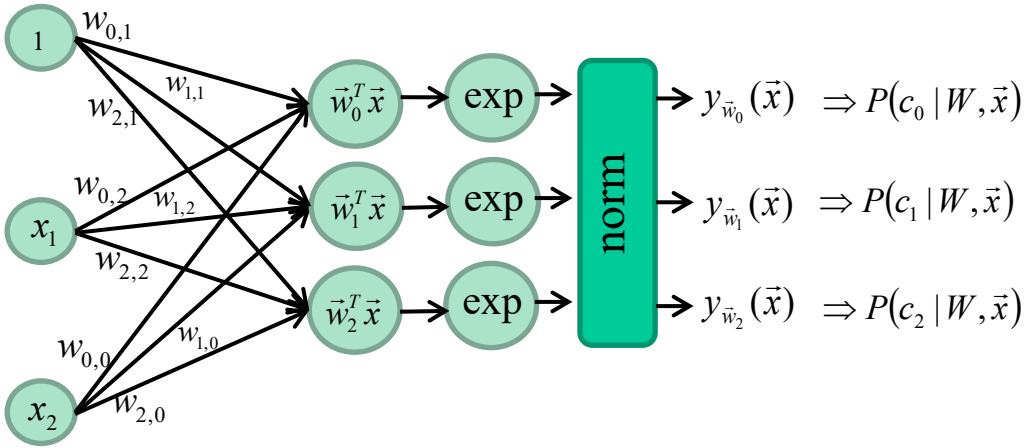
Advantages:

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



And for K>2 classes?

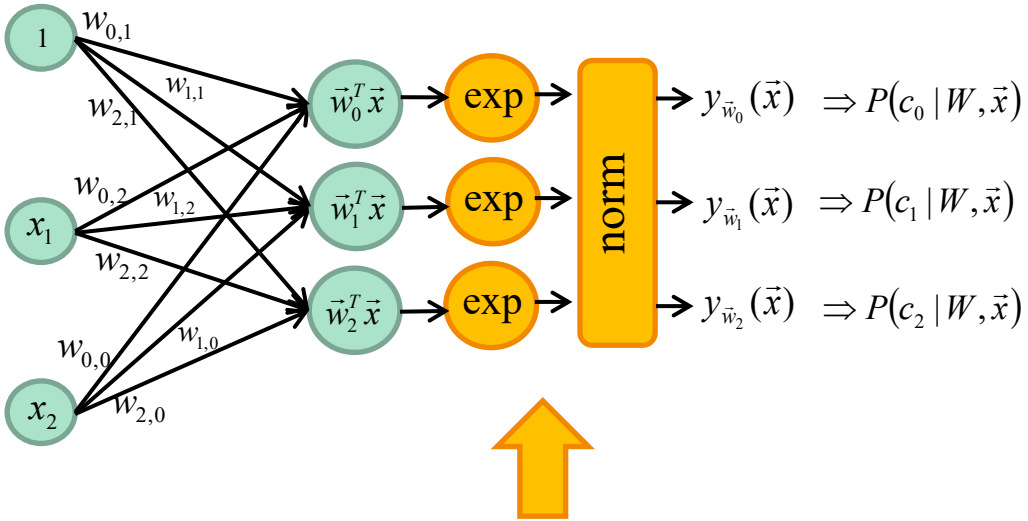
New activation function : Softmax



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

And for K>2 classes?

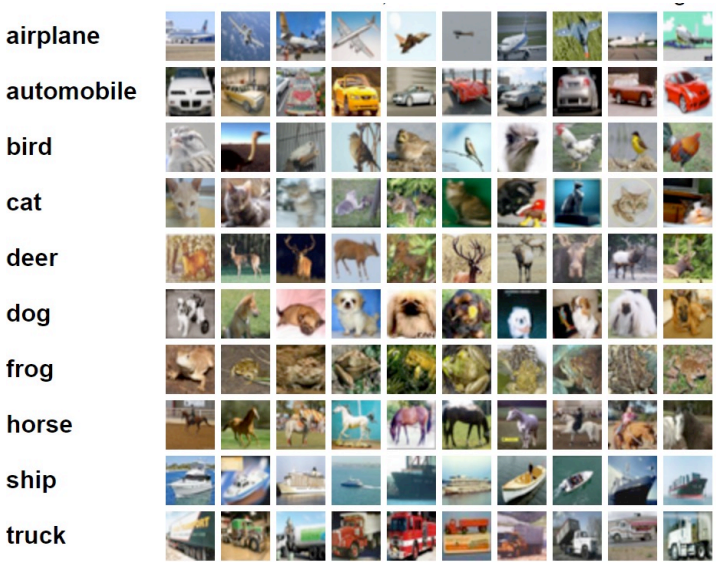
New activation function : Softmax



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

'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 : <u>one-hot vectors</u>

K>2 classes

Cross entropy Loss

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

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 best?

Solution: Maximum a posteriori

Maximum *a posteriori*

Regularization

Constant (hyper-parameter)

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

Loss function

Regularization

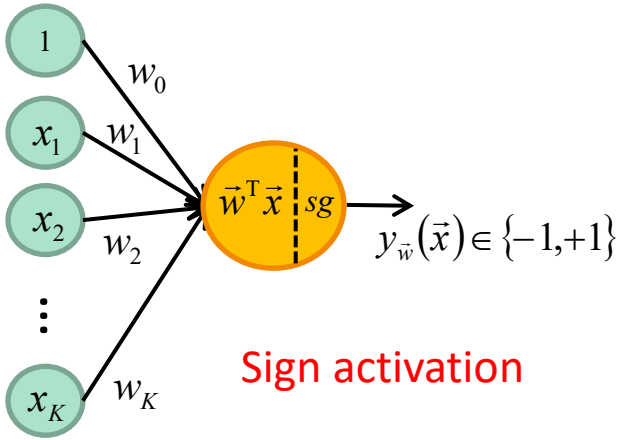
Usually L1 or L2: $R(\theta) = \|\mathrm{W}\|_1$ or $\|\mathrm{W}\|_2$

Wow! Loooots of information!

Lets recap…

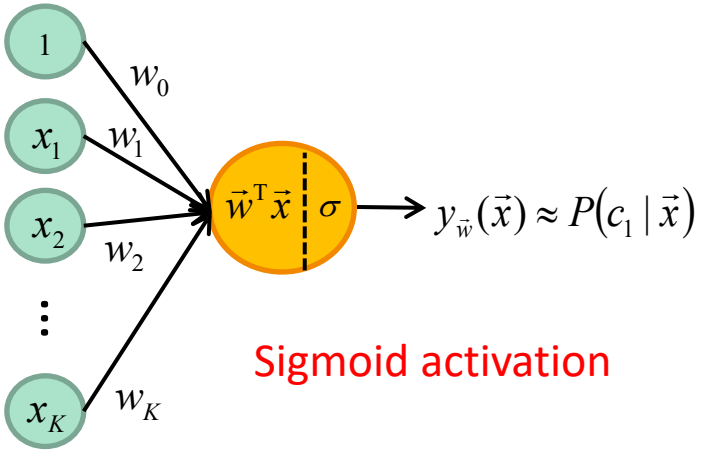
Neural networks

2 classes



Sign activation

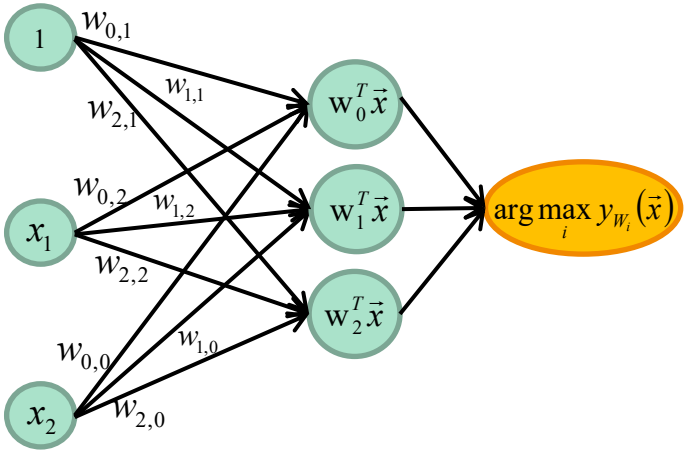
Perceptron



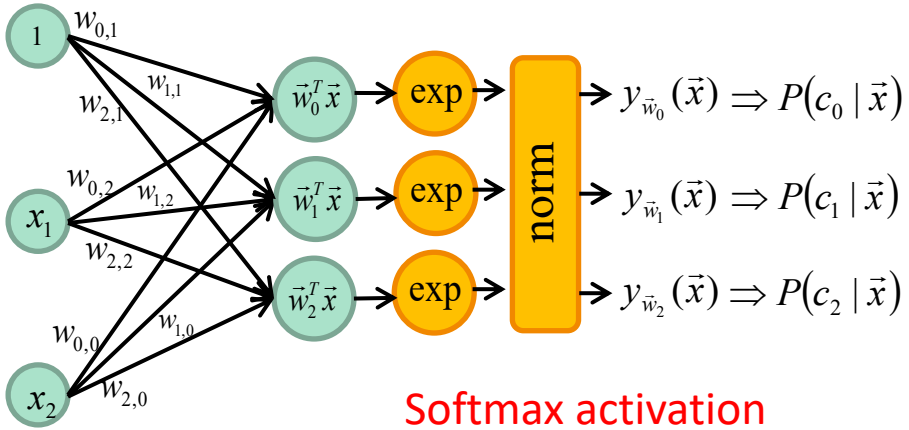
Sigmoid activation

Logistic regression

K-Class Neural networks



Perceptron



Softmax activation

Logistic regression

Loss functions

2 classes

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

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

$$L\left(y_{\vec{w}}\left(\vec{x}\right), D\right) = \sum_{n=1}^{N} \max\left(0, 1 - t_n \vec{w}^{\mathrm{T}} \vec{x}_n\right)$$ "Hinge Loss" or "SVM" Loss

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

Loss functions

K classes

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

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

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

"Hinge Loss" or "SVM" Loss

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

Cross entropy loss with a Softmax

Constant (hyper-parameter)

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

Loss function

Regularization

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

Optimization

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

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 sample is well classified or k== MAX_ITER

Now, lets go

DEEPER

Non-linearly separable training data

Three classical solutions

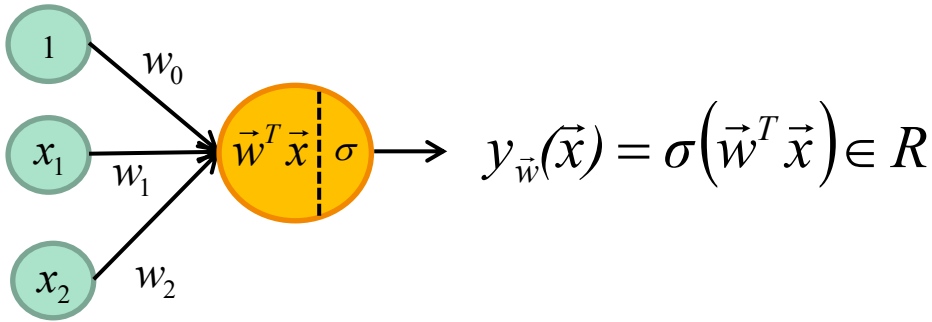
1. Acquire more data
2. Use a non-linear classifier
3. Transform the data

Non-linearly separable training data

Three classical solutions

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

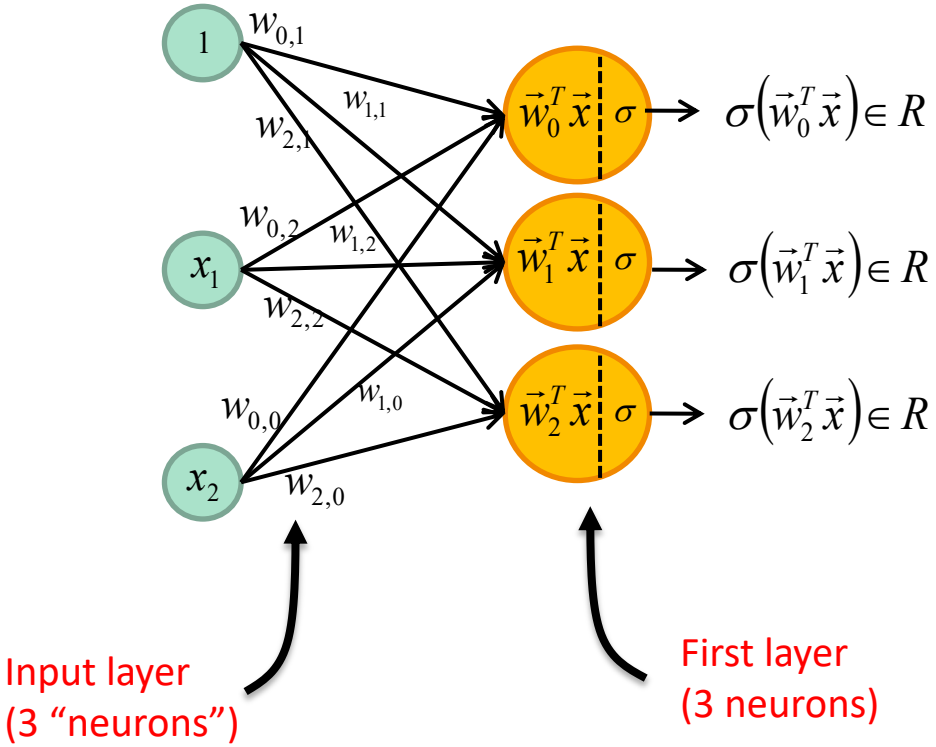
2D, 2 Classes, Linear logistic regression

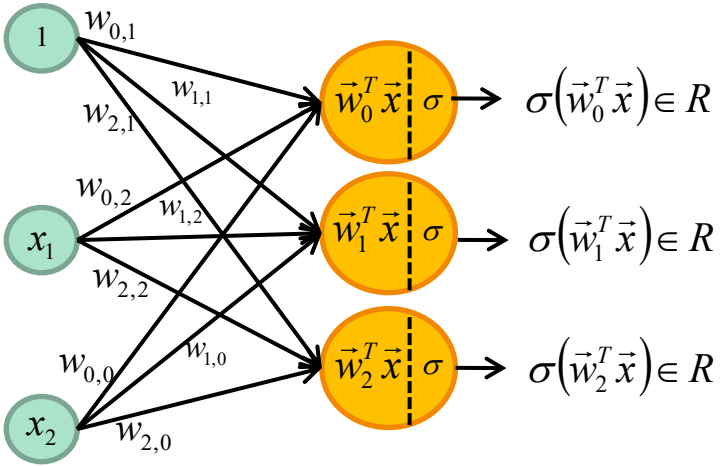


Input layer
(3 “neurons”)

Output layer
(1 neuron with sigmoid)

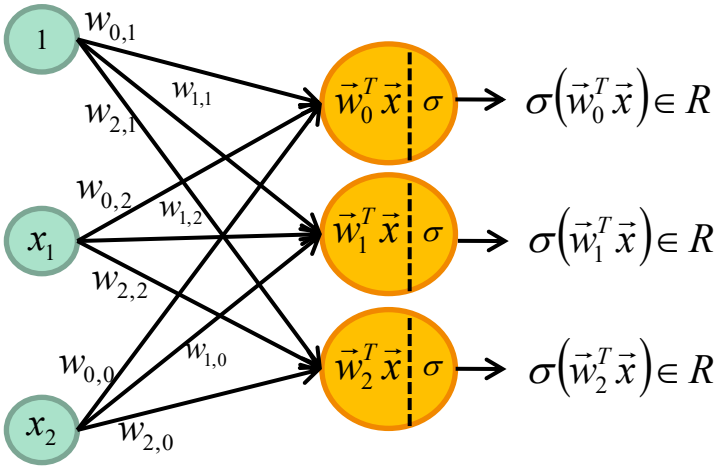
Let's add 3 neurons





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

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

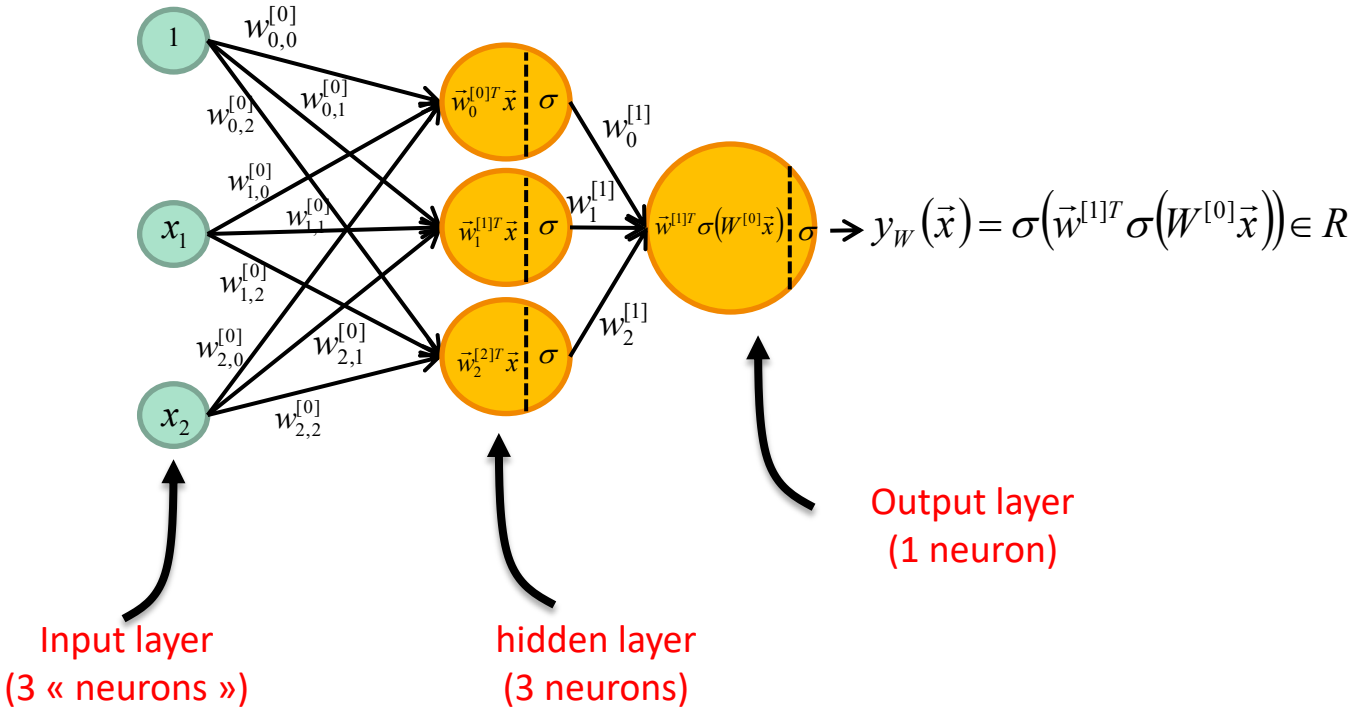


NOTE: The output of the first layer is a vector of 3 real values

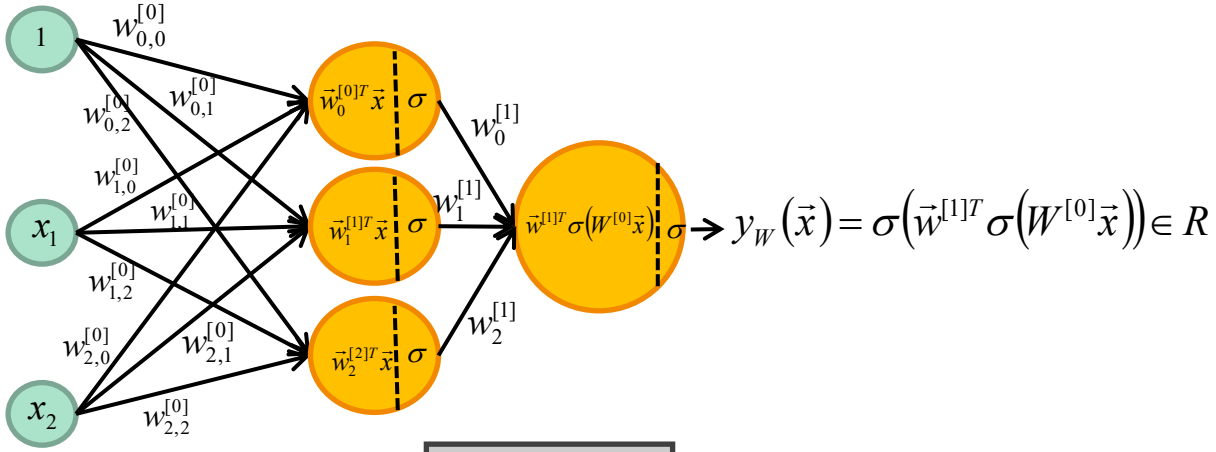
$$\sigma\left(W^{[0]}\vec{x}\right)$$

2-D, 2-Class, 1 hidden layer

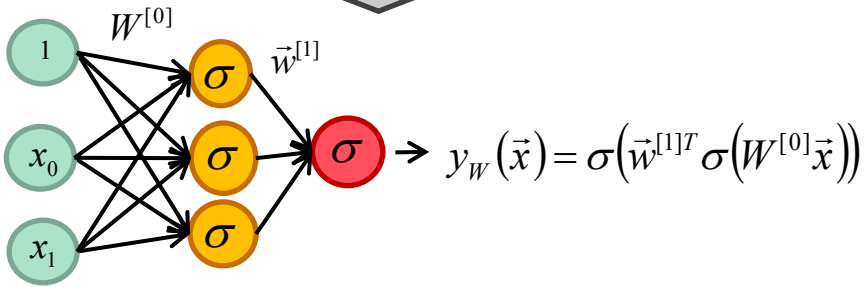
To have a 2-class Classification using logistic regression (cross entropy loss) we must add an output neuron.



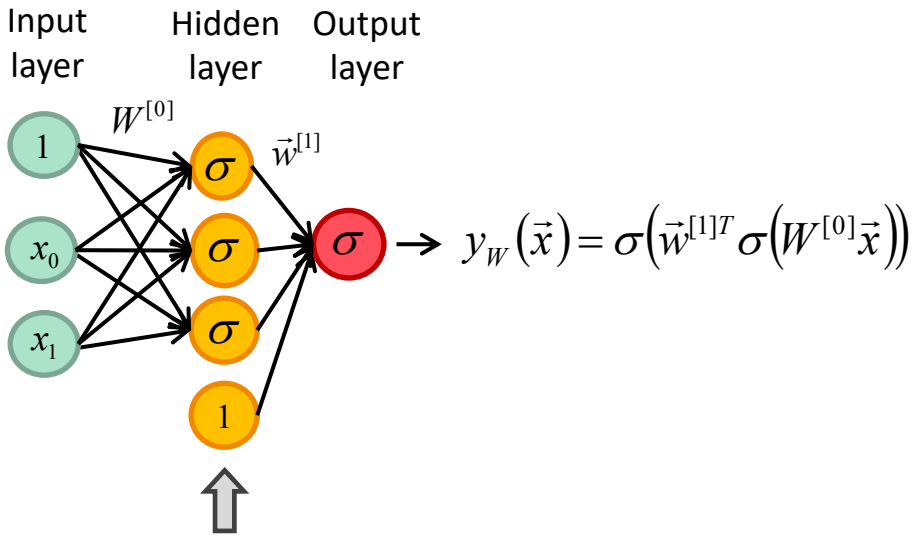
2-D, 2-Class, 1 hidden layer



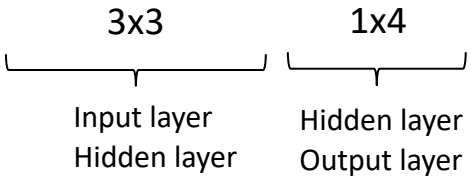
Visual simplification



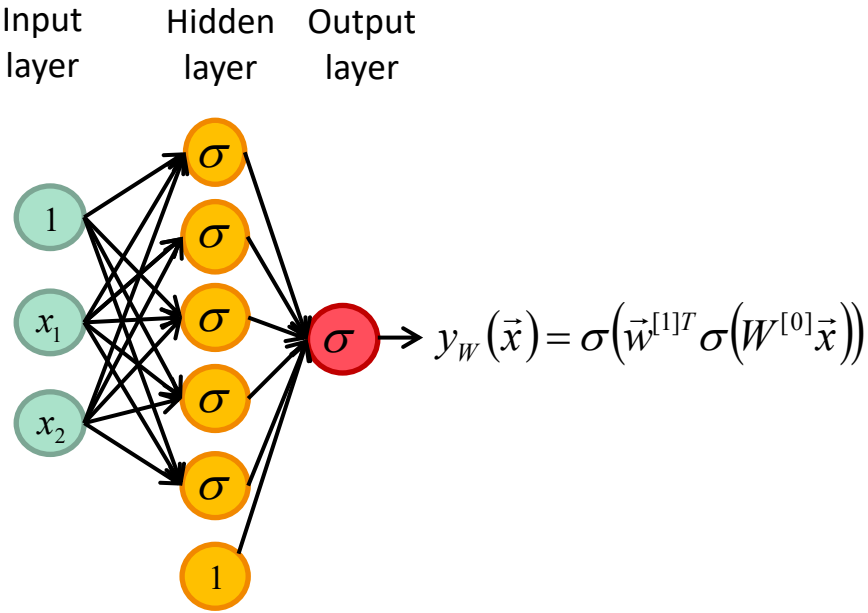
2-D, 2-Class, 1 hidden layer



This network contains a total of 13 parameters



2-D, 2-Class, 1 hidden layer

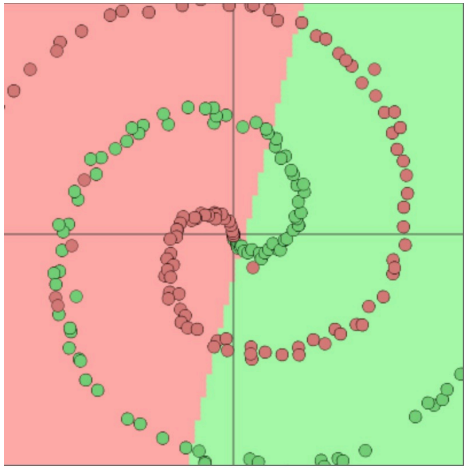


Increasing the number of neurons = increasing the capacity of the model

This network has (5 x 3) + (1 x 6) = 21 parameters

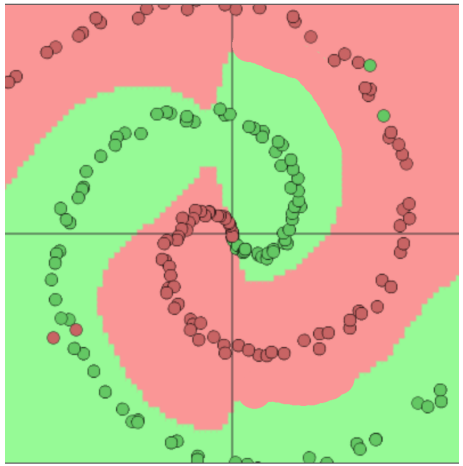
Nb neurons VS Capacity

No hidden neuron



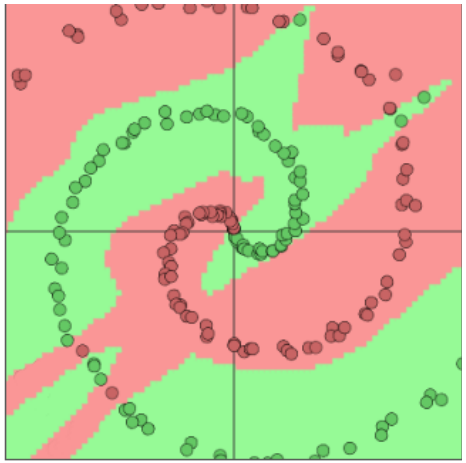
Linear classification
Underfitting
(low capacity)

12 hidden neurons



Non linear classification
Good result
(good capacity)

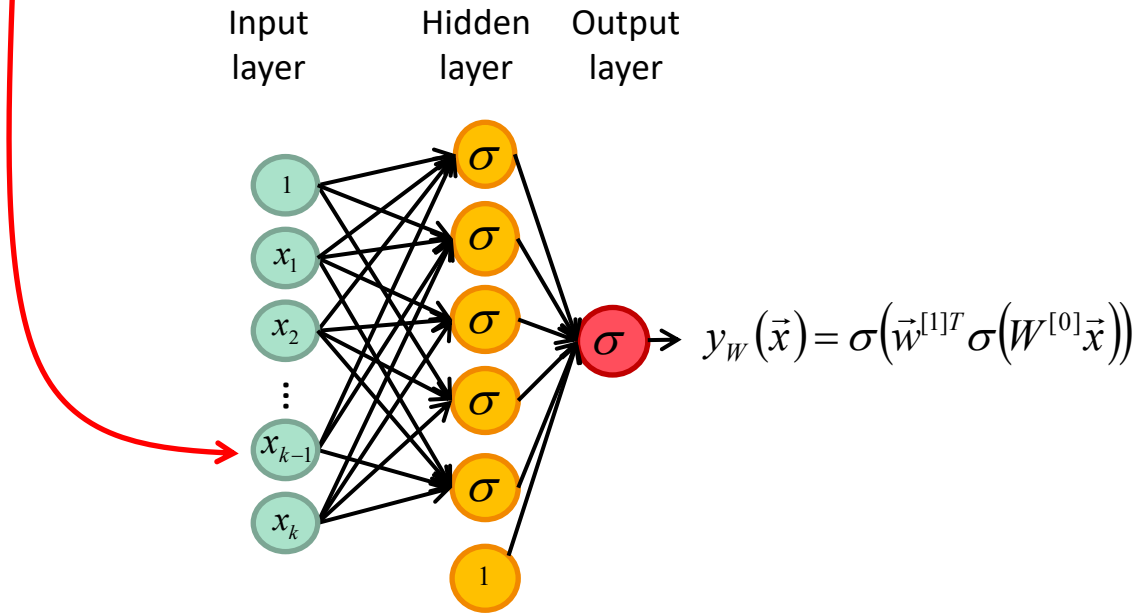
60 hidden neurons



Non linear classification
Over fitting
(too large capacity)

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

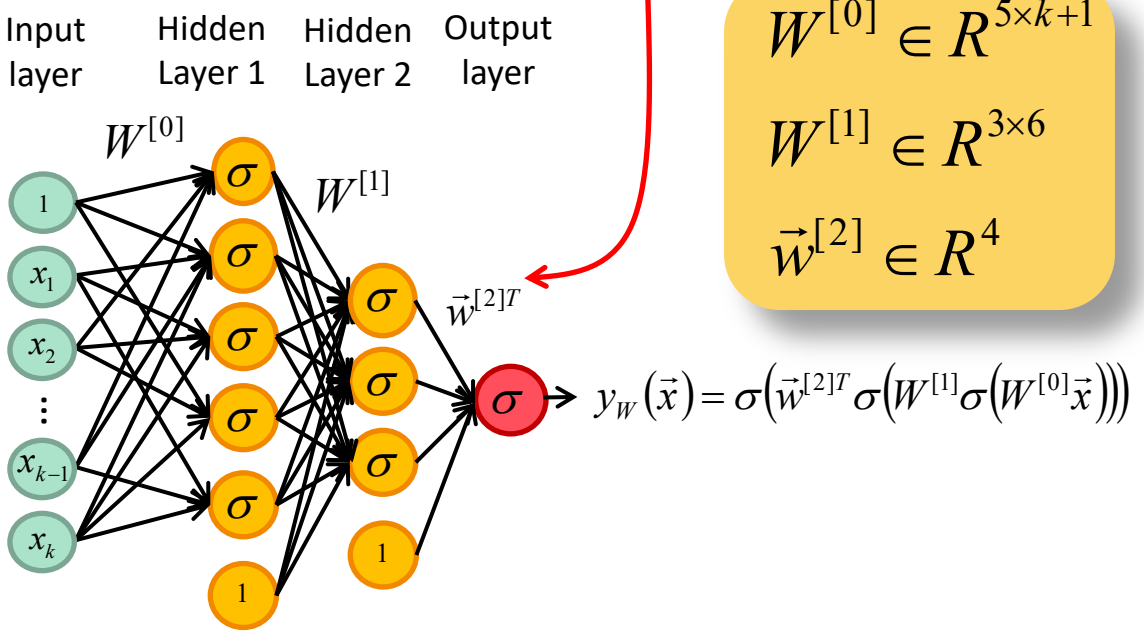
k-D, 2 Classes, 1 hidden layer



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

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

k-D, 2 Classes, 2 hidden layers



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

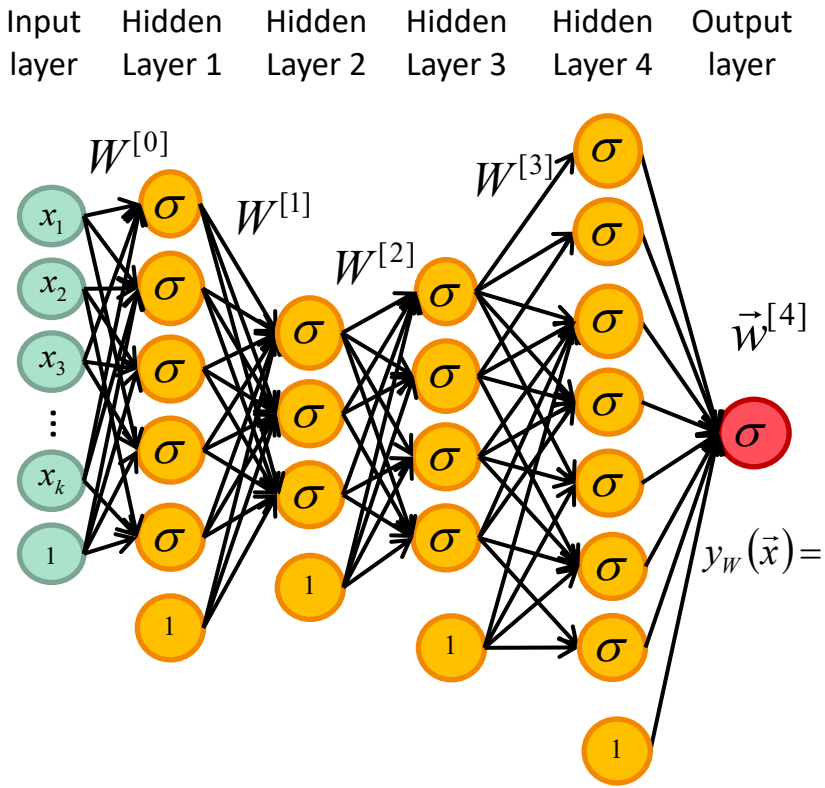
$$W^{[1]} \in R^{3\times 6}$$

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

Adding an hidden layer = Adding a matrix multiplication

This network has (5 x (k+1)) + (6 x 3) + (1 x 4) parameters

k-D, 2 Classes, 4 hidden layer network



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

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

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

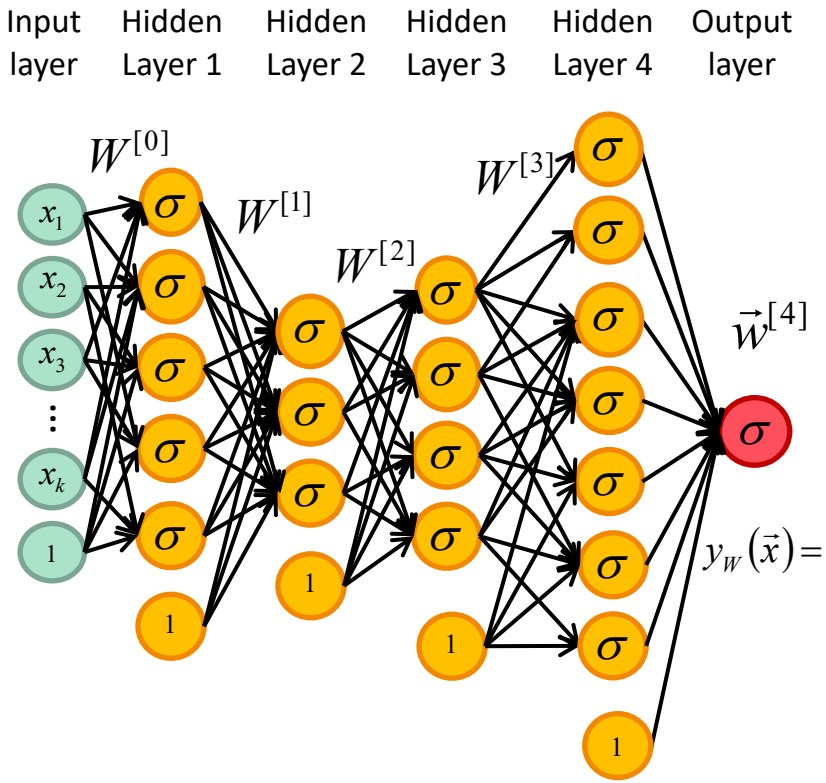
$$W^{[3]} \in R^{7\times 5}$$

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

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

This network has (5 x (k+1)) + (6 x 3) + (4 x 4) + (7 x 5) + (1 x 8) parameters

k-D, 2 Classes, 4 hidden layer network



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

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

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

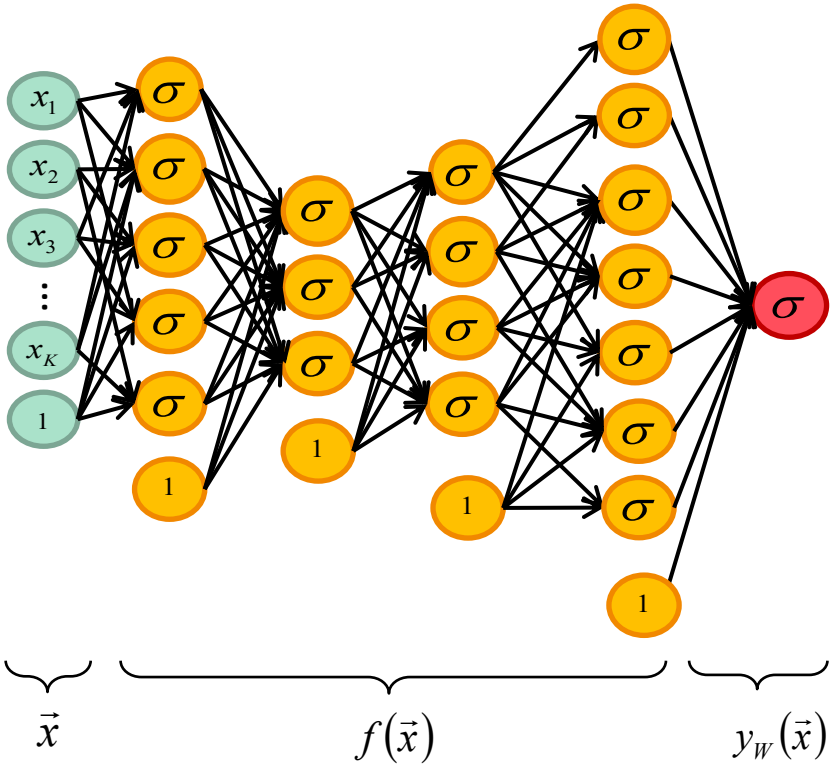
$$W^{[3]} \in R^{7\times 5}$$

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

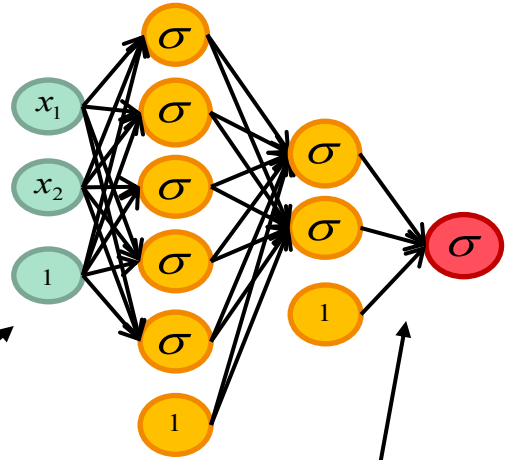
$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[4]T}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)\right)$$

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

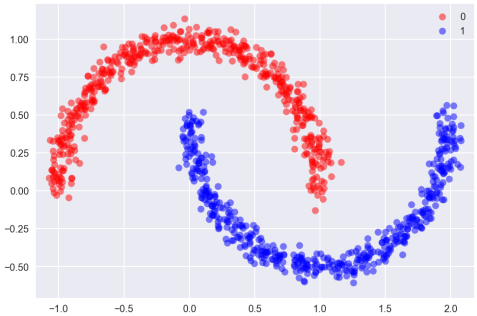
Multilayer Perceptron



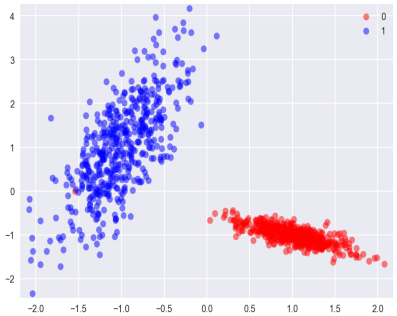
Example



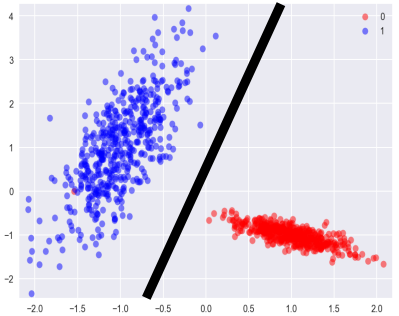
Input data $\vec{x}$



Output of last layer

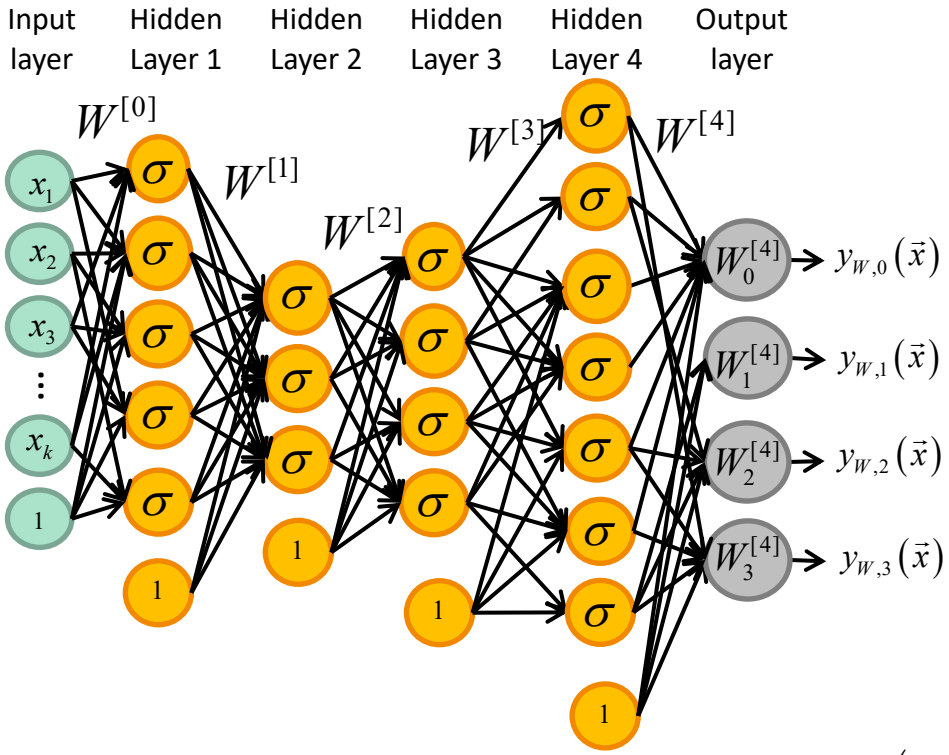


Output of network $y_W(\vec{x})$



A K-Class neural network has K output neurons.

k-D, 4 Classes, 4 hidden layer network



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

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

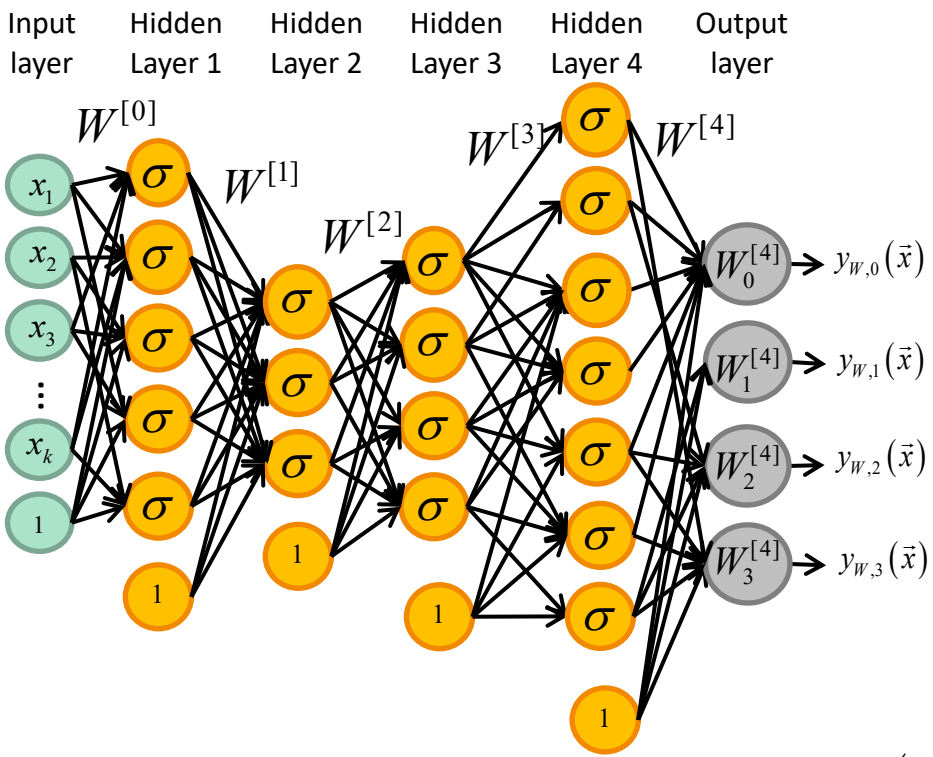
$$W^{[2]} \in R^{4\times 4}$$

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

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

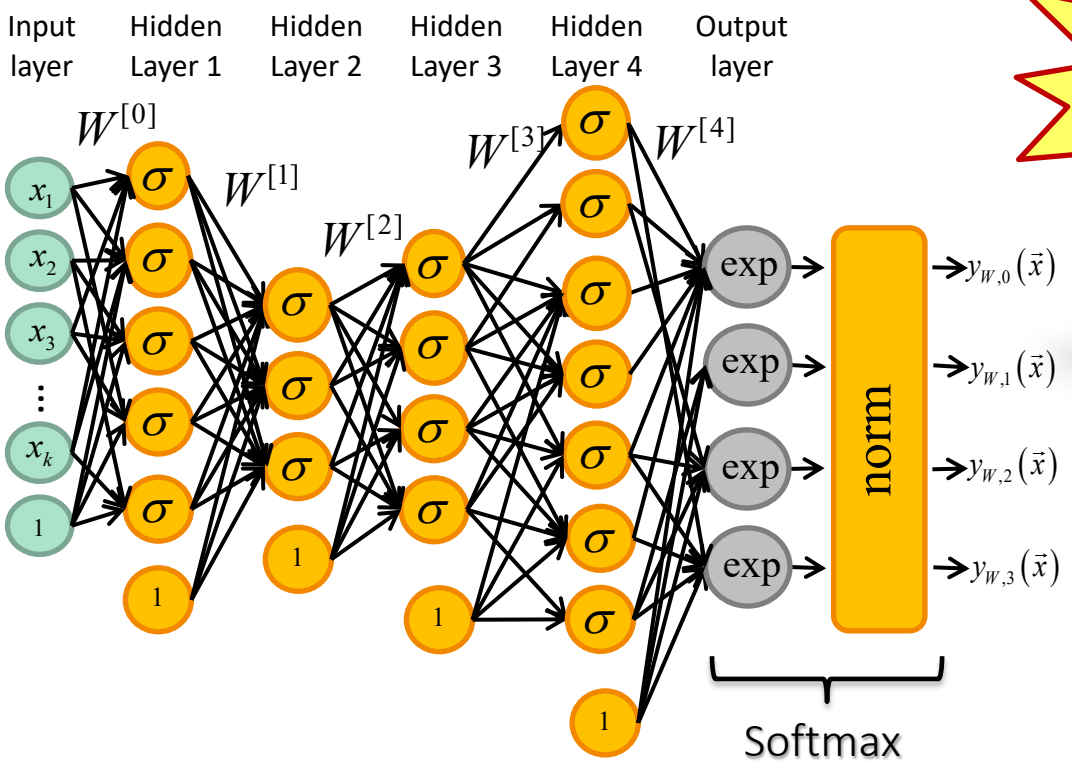
$$y_W(\vec{x}) = W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)\right)$$

k-D, 4 Classes, 4 hidden layer network



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

k-D, 4 Classes, 4 hidden layer network



Cross entropy

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

Summary...

- 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