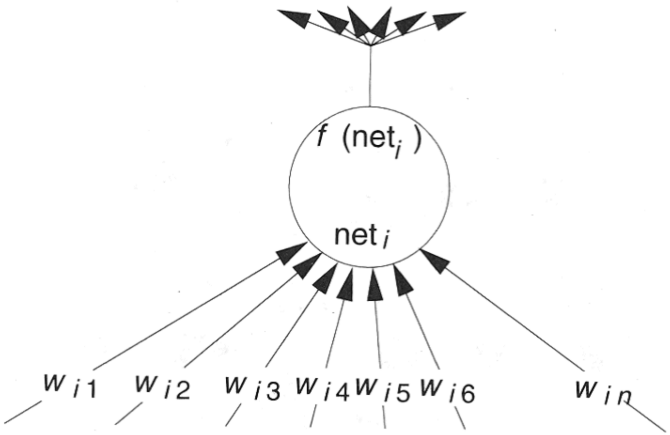
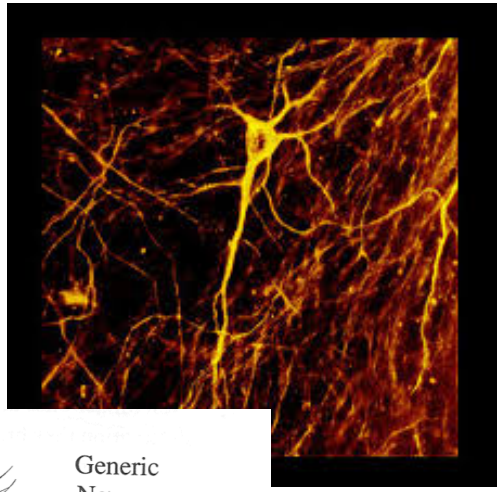
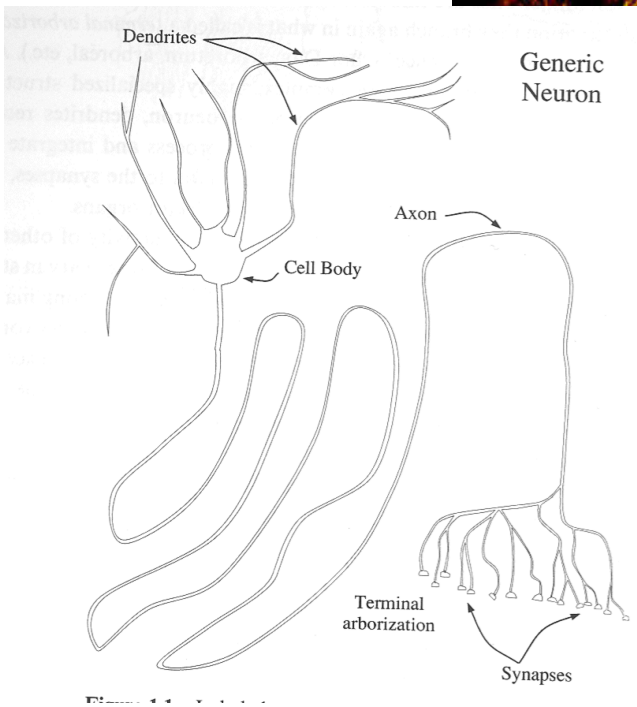
# Lecture 6: Perceptron & Logistic Regression

Andreas Wichert
Department of Computer Science and Engineering
Técnico Lisboa

# Similarity to real neurons...





- The dot product is a linear representation represented by the value *net*

$$y = net := \langle \mathbf{x} | \mathbf{w} \rangle = \sum_{j=1}^{D} w_j \cdot x_j,$$

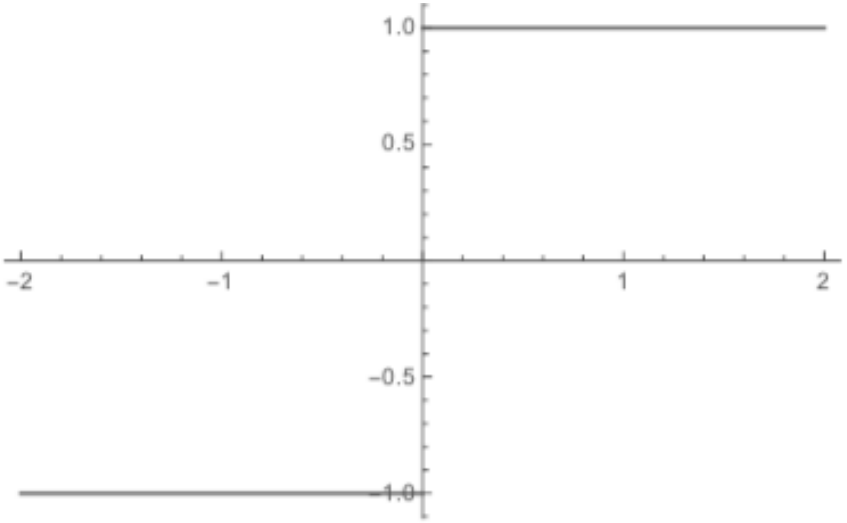
- *Non linearity* can be achieved be a non liner activation (transfer) function *φ()* with

$$o = \phi(net) = \phi(\langle \mathbf{x} | \mathbf{w} \rangle) = \phi \left( \sum_{j=1}^{D} w_j \cdot x_j \right)$$

# *Sgn*

- Examples of nonlinear transfer functions are the *sgn* function

$$\phi(net) := sgn(net) = \begin{cases} 1 & \text{if} \quad net \geq 0 \\ -1 & \text{if} \quad net < 0 \end{cases}$$

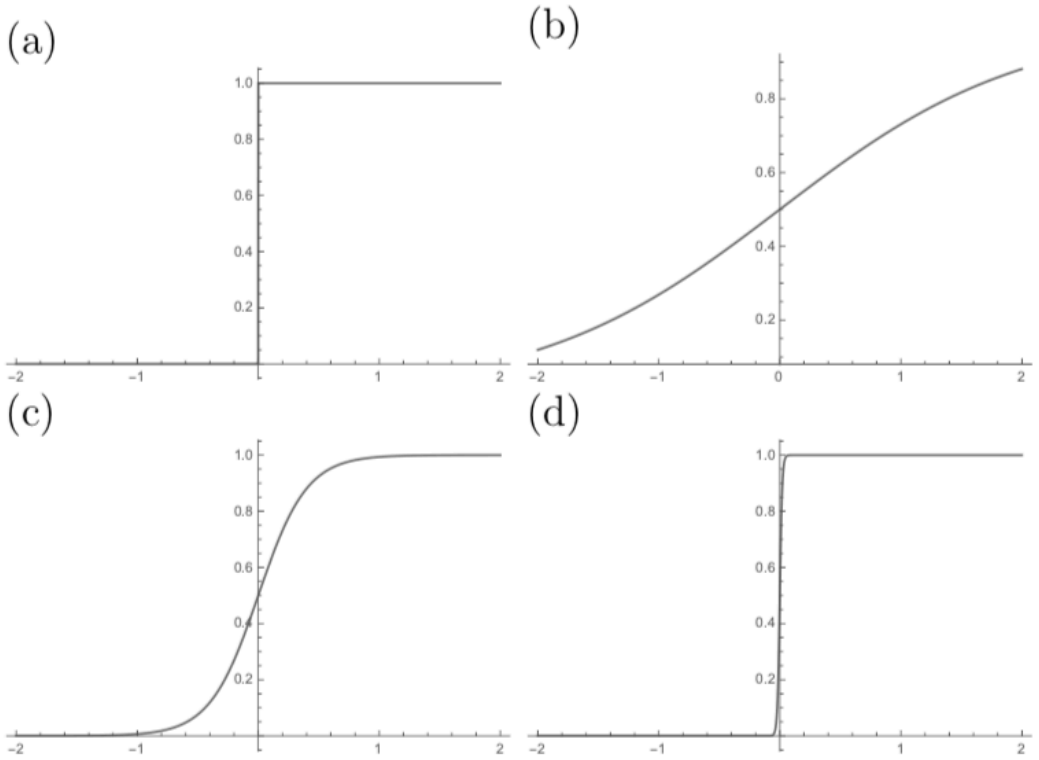


- The *sgn* activation function can be scaled to the two values 0 or 1 for not firing and firing,

$$\phi(net) := sgn_0(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ 0 & \text{if } net < 0 \end{cases}$$

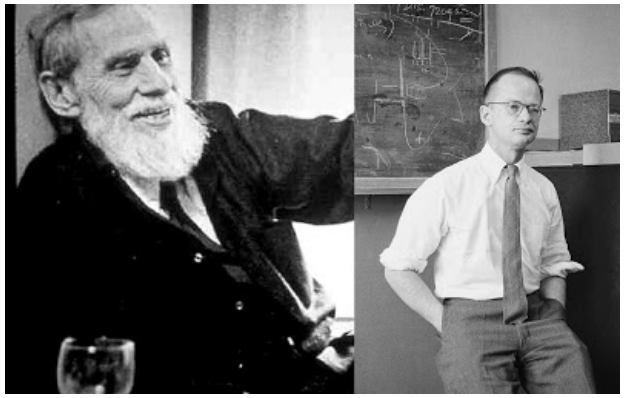
- Both non linear function *sgn(net)* and *sgn0(net)* are non continuous. The activation function *sgn0(net)* can be approximated by the non linear continuous function *σ(net)*

$$\phi(net) := \sigma(net) = \frac{1}{1 + e^{(-\alpha \cdot net)}}$$

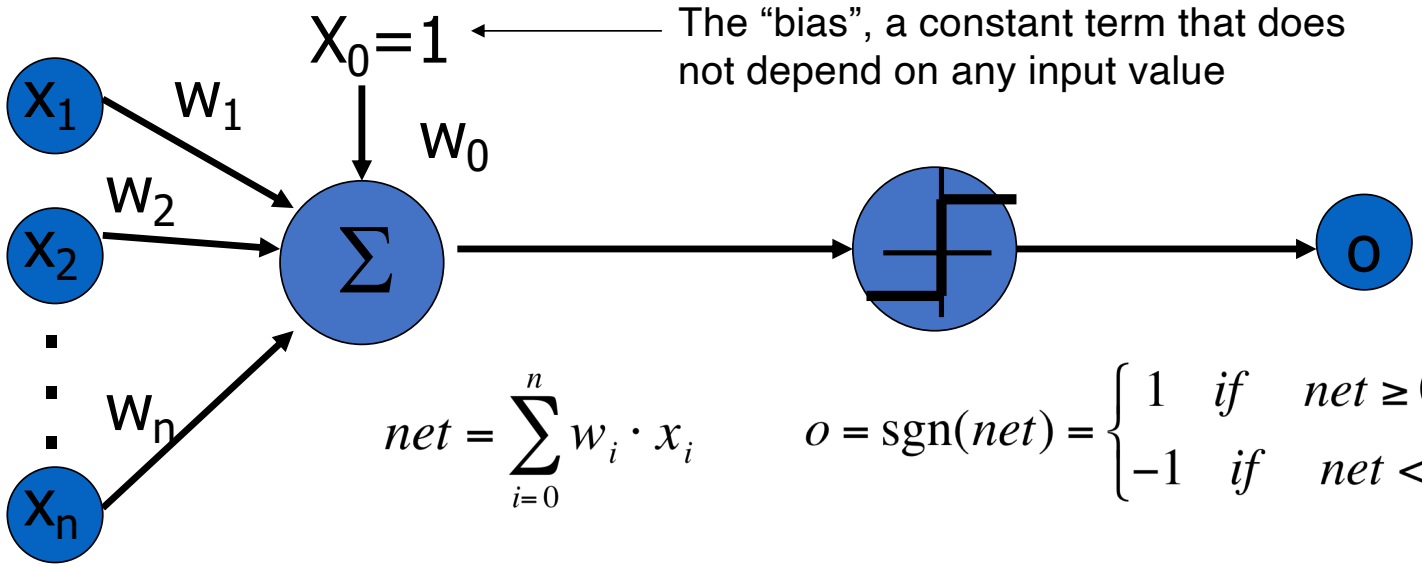



(a) The activation function *sgn0(net).* (b) The function σ(net) with *α = 1*. (c) The function σ*(net) with α = 5*. (d) The function σ(net) with *α = 10* is very similar to *sgn0(net),* bigger *α* make it even more similar

# Perceptron (1957)



- Linear threshold unit (LTU)

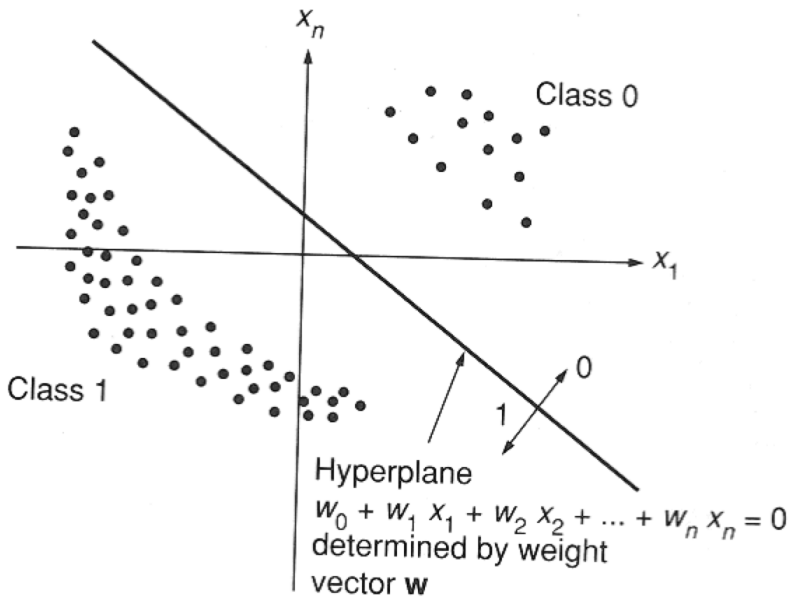



$$net = \sum_{i=0}^{n} w_i \cdot x_i \qquad o = \mathrm{sgn}(net) = \begin{cases} 1 & if \quad net \geq 0 \\ -1 & if \quad net < 0 \end{cases}$$

McCulloch-Pitts model of a neuron (1943)

# Linearly separable patterns

$$o = \mathrm{sgn}(\sum_{i=0}^{n} w_i x_i)$$



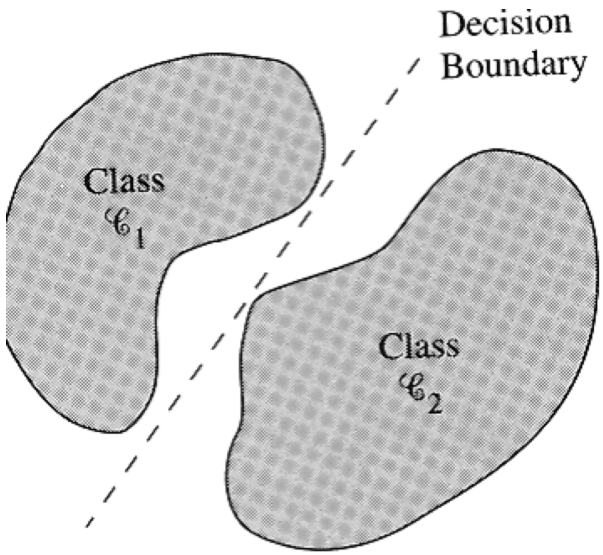

$$\sum_{i=0}^{n} w_i x_i > 0 \quad for \quad C_0$$

$$\sum_{i=0}^{n} w_i x_i \leq 0 \quad for \quad C_1$$

$X_0=1$, bias...

- The goal of a perceptron is to correctly classify the set of pattern $\boldsymbol{D}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,..\boldsymbol{x}_m\}$ into one of the classes $C_1$ and $C_2$
- The output for class $C_1$ is $o=1$ and for $C_2$ is $o=-1$

- For $n=2$ ➔



# Perceptron learning rule

- Consider linearly separable problems
- How to find appropriate weights
  - Initialize each vector ***w*** to some small *random* values
- Look if the output pattern *o* belongs to the desired class, has the desired value *d*

$$w^{new} = w^{old} + \Delta w \qquad \Delta w = \eta \cdot (d - o) \cdot x$$

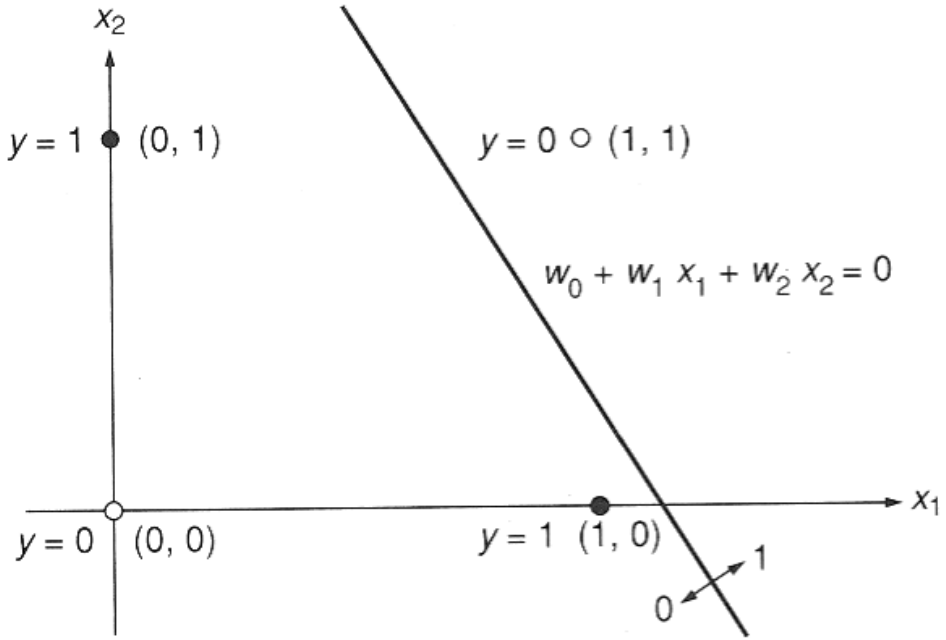
- $\eta$ is called the **learning rate**
- $0 < \eta \leq 1$

- In supervised learning the network has its output compared with known correct answers
  - Supervised learning
  - Learning with a teacher
- *(d-o)* plays the role of the error signal

- The algorithm converges to the correct classification
  - if the training data is linearly separable
  - and $\eta$ is sufficiently small

# XOR problem and Perceptron

- By Minsky and Papert in mid 1960



*Psychological Review*
Vol. 65, No. 6, 1958

# THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

*Cornell Aeronautical Laboratory*

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

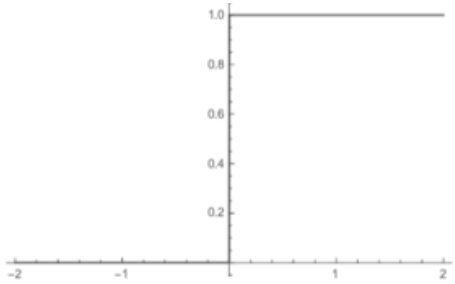
1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain

$$net_0 := \sum_{j=0}^{D} w_j \cdot x_j = \sum_{i=1}^{D} w_j \cdot x_j + w_0 = \langle \mathbf{x} | \mathbf{w} \rangle + w_0 \cdot x_0$$

$$o = sgn_0 \left( \sum_{j=0}^{D} w_j \cdot x_j \right) = sgn_0 \left( \langle \mathbf{x} | \mathbf{w} \rangle + w_0 \cdot x_0 \right)$$

- and $x_0 = 1$



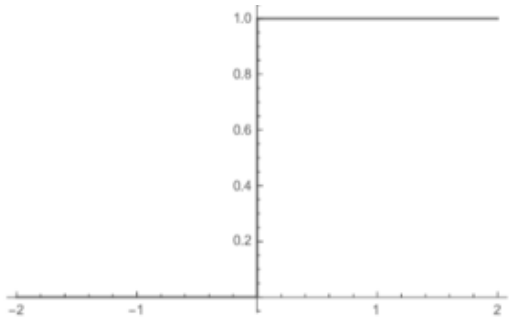
# Discriminant Functions

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \langle \mathbf{x} | \mathbf{w} \rangle + b \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

$$b := w_0$$

- The hyperplane is described by ***w*** and *b*

$$0 = \langle \mathbf{x} | \mathbf{w} \rangle + b.$$



# Learning

$$w_j^{new} = w_j^{old} + \Delta w_j$$

$$\Delta w_j = \eta \cdot (t_k - o_k) \cdot x_j,$$

- $\delta_k = (t_k - o_k)$ plays the role of the error signal with being either zero, *1* or *−1*

$$\delta_k = \begin{cases} 1 & \text{if } t_k = 1 \text{ and } o_k = 0 \\ -1 & \text{if } t_k = 0 \text{ and } o_k = 1 \end{cases}$$

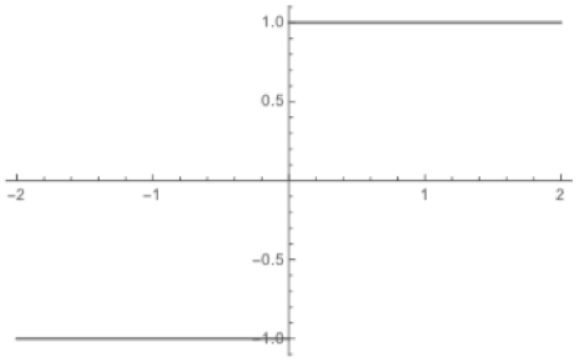
# Learning

- In original Perceptron with *sgn* it is

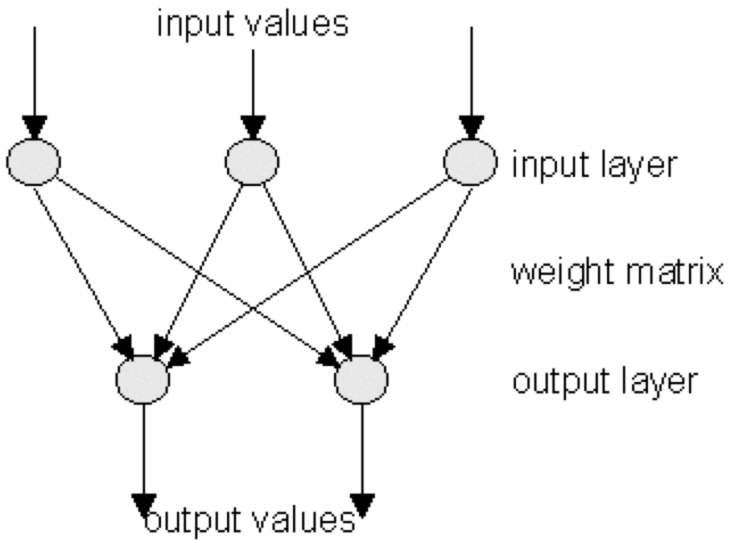
$$\delta_k = \begin{cases} 2 & \text{if } t_k = 1 \text{ and } o_k = -1 \\ -2 & \text{if } t_k = -1 \text{ and } o_k = 1 \end{cases}$$

- Gives the same results if we scale $\eta$ correspondingly

$$\Delta w_j = \eta \cdot (t_k - o_k) \cdot x_j,$$



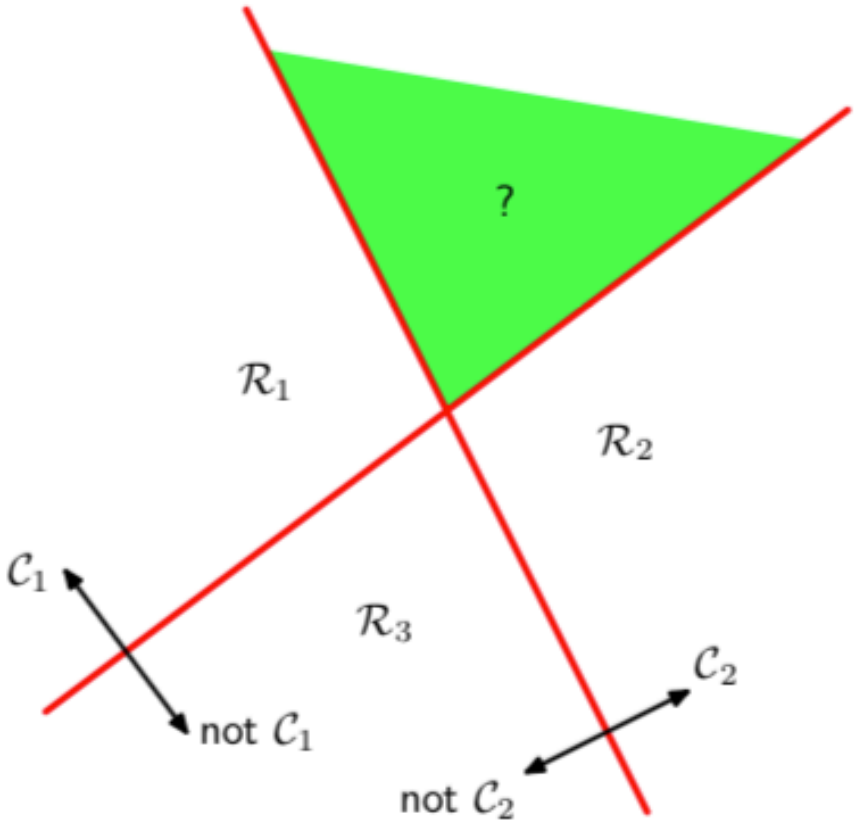
# Multiclass linear discriminant

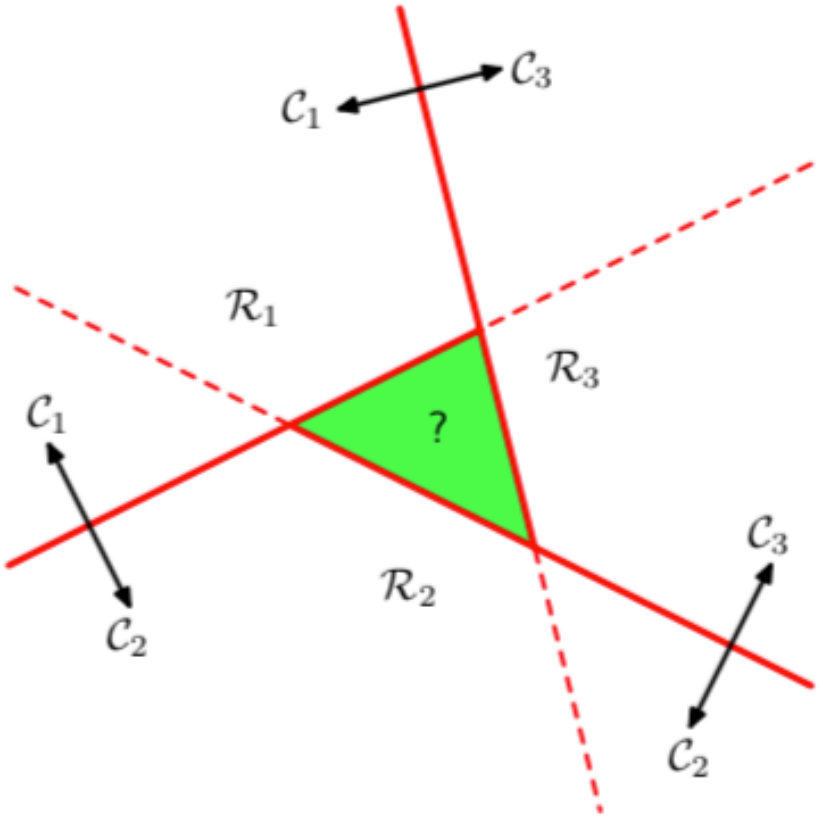



$$o_1 = \mathrm{sgn}(\sum_{i=0}^{n} w_{1i} x_i)$$

$$o_2 = \mathrm{sgn}(\sum_{i=0}^{n} w_{2i} x_i)$$

# Problem


?
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathcal{R}_3$
$\mathcal{C}_1$
not $\mathcal{C}_1$
$\mathcal{C}_2$
not $\mathcal{C}_2$


$\mathcal{C}_1$
$\mathcal{C}_3$
$\mathcal{R}_1$
$\mathcal{R}_3$
?
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{R}_2$
$\mathcal{C}_3$
$\mathcal{C}_2$

# Gradient Decent

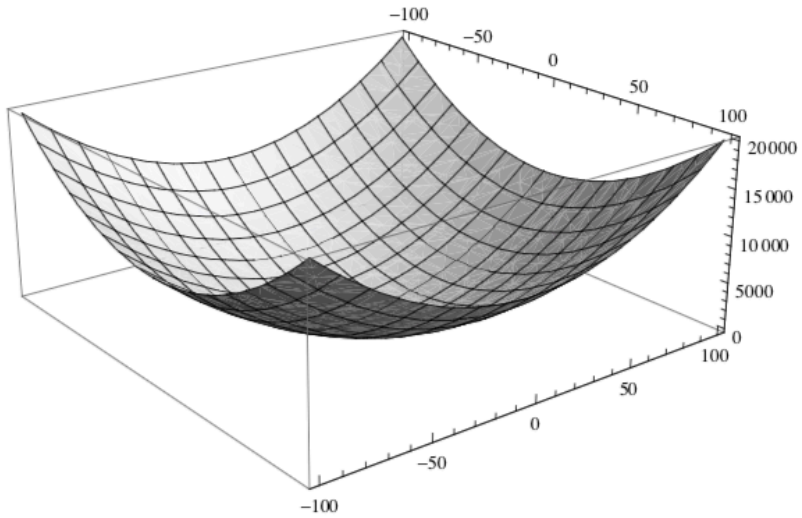
- If the training data is not linearly separable, the perceptron learning algorithm can fail to converge
- However the delta rule that is based on the gradient descent overcomes these difficulties
- Consider simpler linear unit with a linear activation function

$$o = \sum_{j=0}^{D} w_j \cdot x_j = net$$

- We can define the training error for a training data set $D_t$ of $N$ elements with

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (t_k - o_k)^2 = \frac{1}{2} \cdot \sum_{k=1}^{N} \left( t_k - \sum_{j=0}^{D} w_j \cdot x_{k,j} \right)^2$$

- The training error is function of $\boldsymbol{w}$, with $\boldsymbol{w} = (w_0, w_1)$ we can represent all possible values as a two dimensional function




- The error surface as defined by $E(\boldsymbol{w})$ contains only one global minimum since it is convex.

- To find a local minimum of a function *E(**w**)* using gradient descent, one takes steps proportional to the negative of the gradient

$$-\nabla E(\mathbf{w}) = -\left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right)^T$$

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \Delta\mathbf{w}$$

$$w_j^{new} = w_j^{old} + \Delta w_j$$

$$\Delta\mathbf{w} = \eta \cdot (-\nabla E(\mathbf{w}))$$

$$\Delta w_j = -\eta \cdot \frac{\partial E}{\partial w_j}.$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\cdot\sum_{k=1}^{N}(t_k - o_k)^2 = \frac{1}{2}\cdot\sum_{k=1}^{N}\frac{\partial}{\partial w_j}(t_k - o_k)^2$$

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\cdot\sum_{k=1}^{N} 2\cdot(t_k - o_k)\cdot\frac{\partial}{\partial w_j}(t_k - o_k)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}(t_k - o_k)\cdot\frac{\partial}{\partial w_j}\left(t_k - \sum_{j=0}^{D} w_j\cdot x_{k,j}\right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}(t_k - o_k)\cdot(-x_{k,j})$$

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k)\cdot x_{k,j}.$$

- The update rule for gradient decent is given by

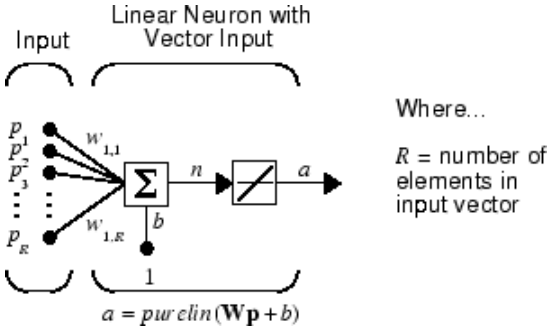
$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

$$w_j^{new} = w_j^{old} + \Delta w_j.$$

# Stochastic gradient descent

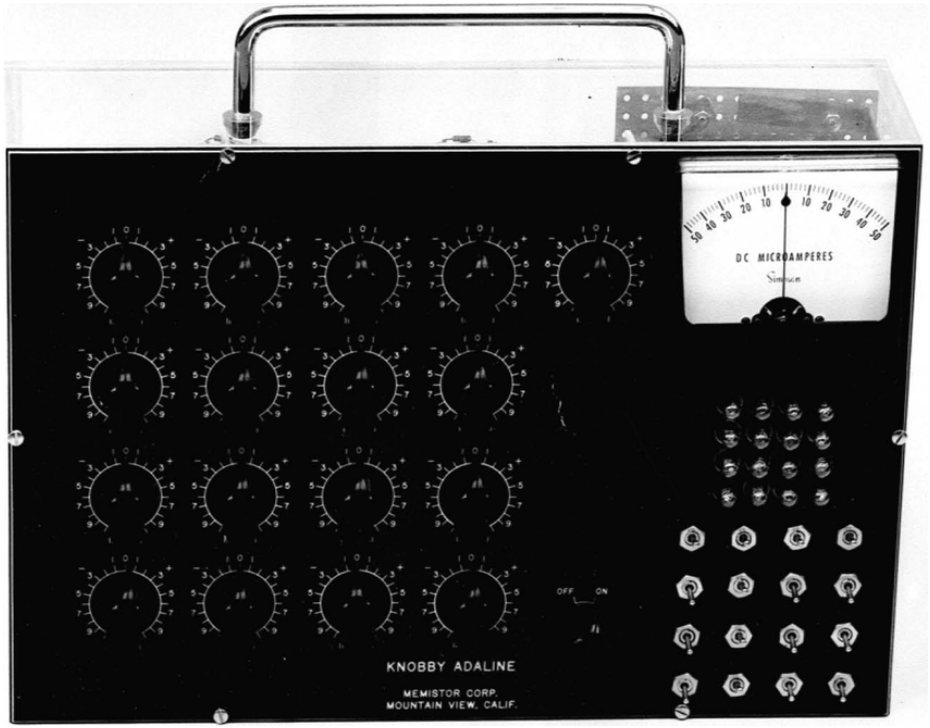
- The gradient decent training rule updates summing over all N training examples.
- Stochastic gradient descent (SGD) approximates gradient decent by updating weights incrementally and calculating the error for each example according to the update rule by

$$\Delta w_j = \eta \cdot (t_k - o_k) \cdot x_{k,j}$$




$$\Delta w_j = \eta \cdot (t_k - o_k) \cdot x_{k,j}$$

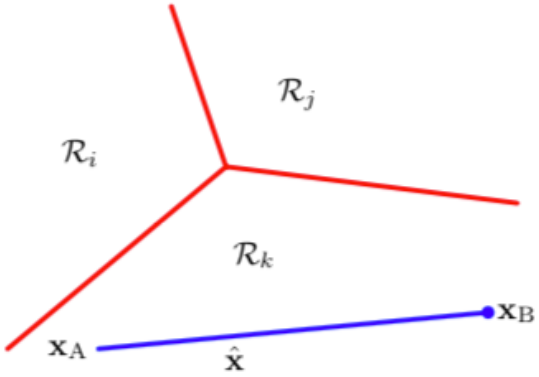
- This rule is known as delta-rule or LMS (last mean-square) weight update.
- It  is used in Adaline (Adaptive Linear Element) for adaptive filters and was developed by Widroff and Hoff




- ADALINE. An adaptive linear neuron. Manually adapted synapses. Designed and built by Ted Hoff in 1960. Demonstrates the least mean square LMS learning algorithm. (Courtesy of Professor Bernard Widrow.)

# Multiclass linear discriminant

- For $K$ artificial neurons an index k is used with $k \in \{1,2,\cdots,K\}$
- Identify the weight vector $w_k$ and the output $o_k$ of the neuron.




with the prediction

$$o_k = net_k = \sum_{j=1}^{D} w_{k,j} \cdot x_j$$

$$\arg\max_{\kappa} \left(\mathbf{w}_\kappa^T \cdot \mathbf{x}\right)$$

# REGULARISATION

With $l_2$ regularisation we get (see linear regression, for simplicity without 1/2)

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k) \cdot x_{k,j} + \lambda \cdot w_j.$$

and the stochastic gradient descent (SGD) rule is

$$\Delta w_j = \eta \cdot (t_k - o_k) \cdot x_{k,j} - \eta \cdot \lambda \cdot w_j$$

With $l_1$ regularisation we get (see linear regression), there is no derivative for $|w_j|$, we use a subderivative with *sign* function

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k) \cdot x_{k,j} + \lambda \cdot sign(w_j)$$

and the stochastic gradient descent (SGD) rule is

$$\Delta w_j = \eta \cdot (t_k - o_k) \cdot x_{k,j} - \eta \cdot \lambda \cdot sign(w_j)$$

# sign

- $|w_j|$ is convex
  - The subdifferential at the origin is the interval *[−1, 1].*
  - The subdifferential at any point $x_0<0$ is the singleton set *{−1}*
  - The subdifferential at any point $x_0>0$ is the singleton set *{1}*.
  - This is similar to the sign function, but is not a single-valued function at *0*
    - instead including all possible subderivatives.

$$sign(w_j) = \begin{cases} -1 & if \quad w_j < 0 \\ 0 & if \quad w_j = 0 \\ 1 & if \quad w_j > 0 \end{cases}$$

# Continuous activation functions

For continuous activation function $\phi()$

$$o_k = \phi\left(\sum_{j=0}^{D} w_j \cdot x_{k,j}\right).$$

we can define as well the update rule for gradient decent with the differential

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}(t_k - o_k) \cdot \frac{\partial}{\partial w_j}\left(t_k - \phi\left(\sum_{j=0}^{D} w_j \cdot x_{k,j}\right)\right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}(t_k - o_k) \cdot \left(-\phi'\left(\sum_{j=0}^{D} w_j \cdot x_{k,j}\right) \cdot x_{k,j}\right)$$

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k) \cdot \phi'\left(\sum_{j=0}^{D} w_j \cdot x_{k,j}\right) \cdot x_{k,j}.$$

# Sigmoid Activation

For the non linear continuous activation function $\sigma()$

$$o_k = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_{k,j}\right)$$

we can define as well the update rule for gradient decent with the differential

$$\sigma(x)' = \alpha \cdot \sigma(x) \cdot (1 - \sigma(x))$$

we get

$$\frac{\partial E}{\partial w_j} = -\alpha \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot \sigma\left(net_{k,j}\right) \cdot \left(1 - \sigma\left(net_{k,j}\right)\right) \cdot x_{k,j}.$$

what does this mean?

# Logistic Regression

Since

$$p(C_1|\mathbf{x}) = \frac{1}{1 + e^{(-net)}} = \frac{e^{(net)}}{1 + e^{(net)}}$$

and

$$\frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})} = e^{(net)}$$

We can define odds

$$odds = \frac{p(\mathbf{x}|C_1) \cdot p(C_1)}{p(\mathbf{x}|C_2) \cdot p(C_2)} = \frac{p(C_1|\mathbf{x})}{p(C_2|\mathbf{x})} = \frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})}$$

and we have the *logit* function that is the inverse of sigmoid $\sigma$ function

$$\log(odds) = logit(\sigma(\mathbf{w}^T \cdot \mathbf{x})) = \log\left(\frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})}\right) = \mathbf{w}^T \cdot \mathbf{x}$$

# Probabilistic Generative Models

We can write using Bayes and the law of total probability

$$p(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1) \cdot p(C_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_1) \cdot p(C_1)}{p(\mathbf{x}|C_1) \cdot p(C_1) + p(\mathbf{x}|C_2) \cdot p(C_2)}$$

$$p(C_1|\mathbf{x}) = \frac{1}{1 + \frac{p(\mathbf{x}|C_2) \cdot p(C_2)}{p(\mathbf{x}|C_1) \cdot p(C_1)}}$$

with

$$net = \log\left(\frac{p(\mathbf{x}|C_1) \cdot p(C_1)}{p(\mathbf{x}|C_2) \cdot p(C_2)}\right) = \log\left(\frac{p(C_1|\mathbf{x})}{p(C_2|\mathbf{x})}\right) = logit(\sigma(net))$$

we arrive at

$$p(C_1|\mathbf{x}) = \sigma(net) = \frac{1}{1 + e^{(-net)}} = \frac{e^{(net)}}{1 + e^{(net)}}$$

the sigmoid activation function, means S-shaped.

It followos

$$o = \sigma(net) = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_j\right) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}\right)$$

$$\sigma'(net) = \sigma(net) \cdot (1 - \sigma(net))$$

and

$$\sigma(-net) = 1 - \sigma(net)$$

We can think of the sigmoid output unit as using the linear input *net* and using the sigmoid activation function to convert *net* into a probability. It can be interpreted as logistic regression with two classes $C_1$ and $C_2$ with

$$p(C_1|\mathbf{x}) = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_j\right) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}\right)$$

and

$$p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$$

# Training

How to train

$$Data = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \cdots, , (\mathbf{x}_N, t_N)\}, \quad t_k \in \{0, 1\}$$

The likelihood function can now be written

$$p(\mathbf{t}|\mathbf{w}) = \prod_{k=1}^{N} p(C_1|\mathbf{x}_k)^{t_k} \cdot (1 - p(C_1|\mathbf{x}_k))^{1-t_k}$$

with

$$o_k = p(C_1|\mathbf{x}_k) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}_k\right) = \sigma(net_k)$$

$$p(\mathbf{t}|\mathbf{w}) = \prod_{k=1}^{N} o_k^{t_k} \cdot (1 - o_k)^{1-t_k}$$

# Error function

- Error function is defined by negative logarithm of the likelihood

$$E(\mathbf{w}) = -\log(p(\mathbf{t}|\mathbf{w})) = -\sum_{k=1}^{N} (t_k \log o_k + (1 - t_k) \log(1 - o_k))$$

which gives the cross entropy error with

$$\neg t_k = (1 - t_k)$$

$$H(t, p) = -\sum_{k=1}^{N} (t_k \cdot \log(p(C_1|\mathbf{x}_k)) + \neg t_k \cdot \log(p(C_2|\mathbf{x}_k)))$$

$$\neg t_k = (1 - t_k)$$

$$H(t, p) = -\sum_{k=1}^{N} (t_k \cdot \log(p(C_1|\mathbf{x}_k)) + \neg t_k \cdot \log(p(C_2|\mathbf{x}_k)))$$

The gradient that minimizes the error (loss) function is given by

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \left( -\sum_{k=1}^{N} (t_k \log o_k + (1 - t_k) \log(1 - o_k)) \right)$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j}\left(-\sum_{k=1}^{N}\left(t_k \log o_k + (1-t_k)\log(1-o_k)\right)\right)$$

with

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}\left(\frac{t_k}{o_k} + -\frac{(1-t_k)}{(1-o_k)}\right)\cdot\frac{\partial}{\partial w_j}(o_k)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}\left(\frac{(1-t_k)}{(1-o_k)} - \frac{t_k}{o_k}\right)\cdot\frac{\partial}{\partial w_j}(o_k)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}\left(\frac{(1-t_k)}{(1-o_k)} - \frac{t_k}{o_k}\right)\cdot(o_k\cdot(1-o_k))\cdot\frac{\partial}{\partial w_j}\left(\sum_{j=0}^{D} w_j\cdot x_{k,j}\right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}((1-t_k)\cdot(o_k)) - (t_k)\cdot(1-o_k))\cdot\frac{\partial}{\partial w_j}\left(\sum_{j=0}^{D} w_j\cdot x_{k,j}\right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N} (o_k - t_k) \cdot \frac{\partial}{\partial w_j} \left( \sum_{j=0}^{D} w_j \cdot x_{k,j} \right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N} (o_k - t_k) \cdot x_{k,j}$$

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

The update rule for gradient decent is given by

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

# Multiclass logistic regression

For $K > 2$ classes

$$p(C_\kappa|\mathbf{x}) = \frac{p(\mathbf{x}|C_\kappa) \cdot p(C_\kappa)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_\kappa) \cdot p(C_\kappa)}{\sum_{j=1}^{K} p(\mathbf{x}|C_j) \cdot p(C_j)}$$

$$p(C_\kappa|\mathbf{x}) = \sigma(\mathbf{net})_\kappa = \frac{p(\mathbf{x}|C_\kappa) \cdot p(C_\kappa)}{\sum_{j=1}^{K} p(\mathbf{x}|C_j) \cdot p(C_j)} = \frac{\exp(net_k)}{\sum_{j=1}^{K} \exp(net_j)}$$

which is called the normalised exponential or softmax function with

$$net_\kappa = \log(p(\mathbf{x}|C_\kappa) \cdot p(C_\kappa))$$

It is the generalisation of the logistic sigmoid with the activation $net_\kappa$

$$net_\kappa = \mathbf{w}_\kappa^T \cdot \mathbf{x}$$

$$odds_s = \frac{p(C_{s,1}|\mathbf{x})}{p(C_{s,2}|\mathbf{x})} = \frac{p(C_{s,1}|\mathbf{x})}{1 - p(C_{1,s}|\mathbf{x})} = e^{(net_s)}.$$

$$\log(odds) = logit(\sigma(\mathbf{w}^T \cdot \mathbf{x})) = \log\left(\frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})}\right) = \mathbf{w}^T \cdot \mathbf{x}$$

$$o_s = \frac{odd_s}{\sum_{t=1}^{K} odd_t} = \frac{\exp(net_s)}{\sum_{t=1}^{K} \exp(net_t)}$$

$$p(C_s|\mathbf{x}) = \sigma(\mathbf{net})_s = \frac{\exp(net_s)}{\sum_{t=1}^{K} \exp(net_t)}$$

$$p(C_\kappa|\mathbf{x}) = \sigma(\mathbf{net})_\kappa = \frac{p(\mathbf{x}|C_\kappa) \cdot p(C_\kappa)}{\sum_{j=1}^{K} p(\mathbf{x}|C_j) \cdot p(C_j)} = \frac{\exp(net_k)}{\sum_{j=1}^{K} \exp(net_j)}$$

- The **softmax function** is used in various multi class classification methods, such as multinomial logistic regression (also known as softmax regression) with the prediction

$$\arg\max_\kappa \left(\sigma(\mathbf{net})_\kappa\right) = \arg\max_\kappa \left(\mathbf{w}_\kappa^T \cdot \mathbf{x}\right)$$

# Cross Entropy Loss Function for softmax

- Taking the error of the gradient function with respect to the vector $\boldsymbol{w}_j$ we obtain

$$\nabla_{\mathbf{w}_s} E(\mathbf{w}_1, \cdots, \mathbf{w}_K) = \sum_{k=1}^{N} (o_{ks} - y_{ks}) \cdot \mathbf{x}_k = -\sum_{k=1}^{N} (y_{ks} - o_{ks}) \cdot \mathbf{x}_k$$

$$\sigma(net_{ks}) = \frac{\exp(net_{ks})}{\sum_{t=1}^{K} \exp(net_{kt})}$$

which is called the normalized exponential or softmax function. For simplification of the derivative function we drop the index $k$ that indicates the $k$-th training pattern and we we get

$$\frac{\partial \sigma(net_s)}{\partial net_t} = \sigma(net_s) \cdot (I_{st} - \sigma(net_t)) = o_s \cdot (I_{st} - o_t)$$

means if $t = s$

$$\frac{\partial \sigma(net_s)}{\partial net_t} = \sigma(net_s) \cdot (1 - \sigma(net_s)) = o_s \cdot (1 - o_s)$$

if $t \neq s$

$$\frac{\partial \sigma(net_s)}{\partial net_t} = -\sigma(net_t) \cdot \sigma(net_s)) = -o_t \cdot o_s$$

# Kronecker Function

with the Kronecker Function

$$I_{st} = \delta_{st} = \begin{cases} 1, & \text{if } s = t \\ 0, & \text{otherwise.} \end{cases}$$

- Taking the error of the gradient function with respect to $net_j$ over all training patterns we get

$$E(\mathbf{w}) = -\sum_{k=1}^{N}\sum_{t=1}^{K} y_{kt} \cdot \log o_{kt}$$

$$\frac{\partial E}{\partial o_{kt}} = -\sum_{k=1}^{N}\sum_{t=1}^{K} \frac{y_{kt}}{o_{kt}}, \qquad \frac{\partial o_{kt}}{\partial net_s} = o_{kt} \cdot (I_{ts} - o_{ks}),$$

$$\frac{\partial E}{\partial net_s} = -\sum_{k=1}^{N}\sum_{t=1}^{K}\frac{y_{kt}}{o_{kt}} \cdot o_{kt} \cdot (I_{ts} - o_{ks})$$

$$\frac{\partial E}{\partial net_s} = -\sum_{k=1}^{N}\sum_{t=1}^{K}(y_{kt} \cdot (I_{ts} - o_{ks}),$$

since

$$\sum_{t=1}^{K} y_{kt} = 1,$$

$$\frac{\partial E}{\partial net_s} = -\sum_{k=1}^{N}(y_{ks} - o_{ks})$$

Since

$$net_{ks} = \sum_{j=1}^{D} w_{js} x_{kj}$$

we get for all $N$ training patterns

$$\frac{\partial E}{\partial w_{js}} = -\sum_{k=1}^{N}(y_{ks} - o_{ks}) \cdot x_{kj}$$

$$\frac{\partial E}{\partial w_{js}} = -\sum_{k=1}^{N}(y_{ks} - o_{ks}) \cdot x_{kj}$$

with the learning rule

$$\Delta w_{js} = \eta \cdot \sum_{k=1}^{N}(y_{ks} - o_{ks}) \cdot x_{kj}$$

$$w_{js}^{new} = w_{js}^{old} + \Delta w_{js}.$$

This algorithm is called the logistic regression.

**Algorithm**

1. iterations=0;

2. $\eta \in (0, 1]$;

3. FOR t=1 TO K
   {
   Initialise all the weights $w_{0t}, w_{1t}, \cdots, w_{Dt}$ to some random values;
   }

4. Choose a pattern $\mathbf{x}_k$ out of the training set;

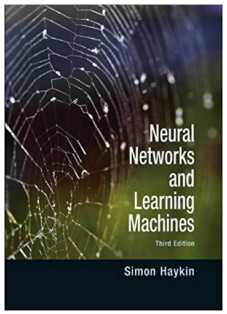
5. FOR t=1 TO K
   {
   Compute $net_{kt} = \sum_{i=1}^{D} w_{jt} \cdot x_{kj} + w_{0t} = \langle \mathbf{x}_k | \mathbf{w}_t \rangle + w_{0t} \cdot x_0$;
   Compute $odds_{kt} = \exp(net_{kt})$;
   }

6. FOR t=1 TO K
   {
   Compute $o_{kt} = \frac{odds_{kt}}{\sum_t odds_{kt}}$;
   Compute $\Delta w_{jt} = \eta \cdot (y_{kt} - o_{kt}) \cdot x_{k,j}$;
   Update the weights $w_{jt} = w_{jt} + \Delta w_{jt}$;
   }

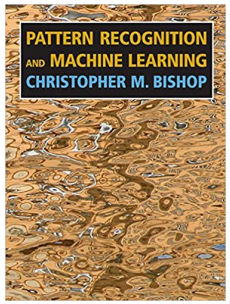
7. iterations++;

8. If no change in weights for all training set or maximum number of iteration THEN STOP ELSE GOTO 4;

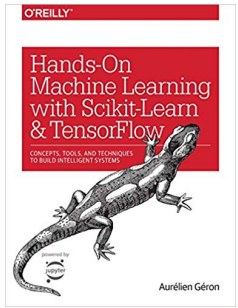
# Literature




- Simon O. Haykin, Neural Networks and Learning Machine, (3rd Edition), Pearson 2008
  - Chapter 1

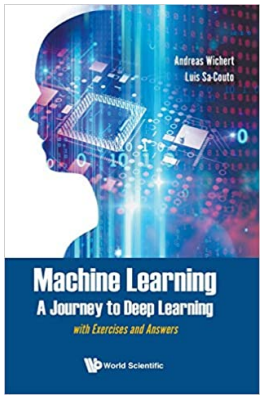



- Christopher M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer 2006
  - Chapter 4



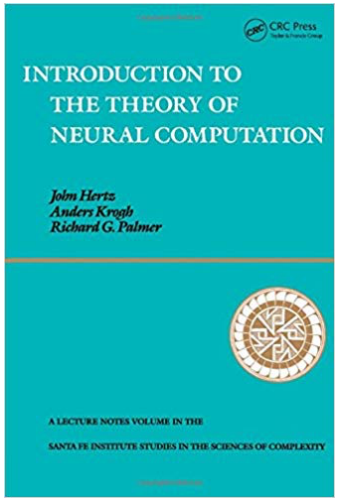

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Aurélien Géron , O'Reilly Media; 1 edition, 2017
  - Chapter 4

# Literature




- Machine Learning - A Journey to Deep Learning, A. Wichert, Luis Sa-Couto, World Scientific, 2021
  - Chapter 5

# Literature (Additional)




- *Introduction To The Theory Of Neural Computation (Santa Fe Institute Series Book 1), John A. Hertz, Anders S. Krogh, Richard G. Palmer, Addison-Wesley Pub. Co, Redwood City, CA; 1 edition (January 1, 1991)*
  - *Chapter 5*