Neural Networks

Two major lines of research

1. understand human brain
   - devise "realistic" computational models (of parts) of the human brain
   - reproduce neurophysiological phenomena
   - assess empirically if the computational model reproduces biological data
2. understand the general computational principles used by human brain
   - it does not matter to reproduce the human brain, but only to infer what are the fundamental computational principles it uses
   - simplifications and abstractions of brain functions/structures are the main working tools
   - device an artificial system that is eventually different from the human brain, but that reproduces some of its functions, perhaps in a faster and more efficient fashion

Neural Networks

We are interested into the second line of research.
There are very many different neural networks models, which answer to different computational needs.
E.g.

- supervised learning (Classification, Regression, Temporal series prediction, ...);
- unsupervised learning (Clustering, Data Mining, Self-Organization maps,...);
- associative memories;
- ...

All these models differ for

- network topology
- function computed by a single neuron
- training algorithm
- how training proceeds (how training data are used)

Neural Networks

When to use a Neural Network ?

- large input size vectors (discrete and/or real valued)
- discrete (classification) or real valued (regression) output
- output is a real valued vector
- noisy data
- unknown target function (no apriori information)
- learning solution does not need to be understood by an human expert ("black-box problem")

Examples of applications

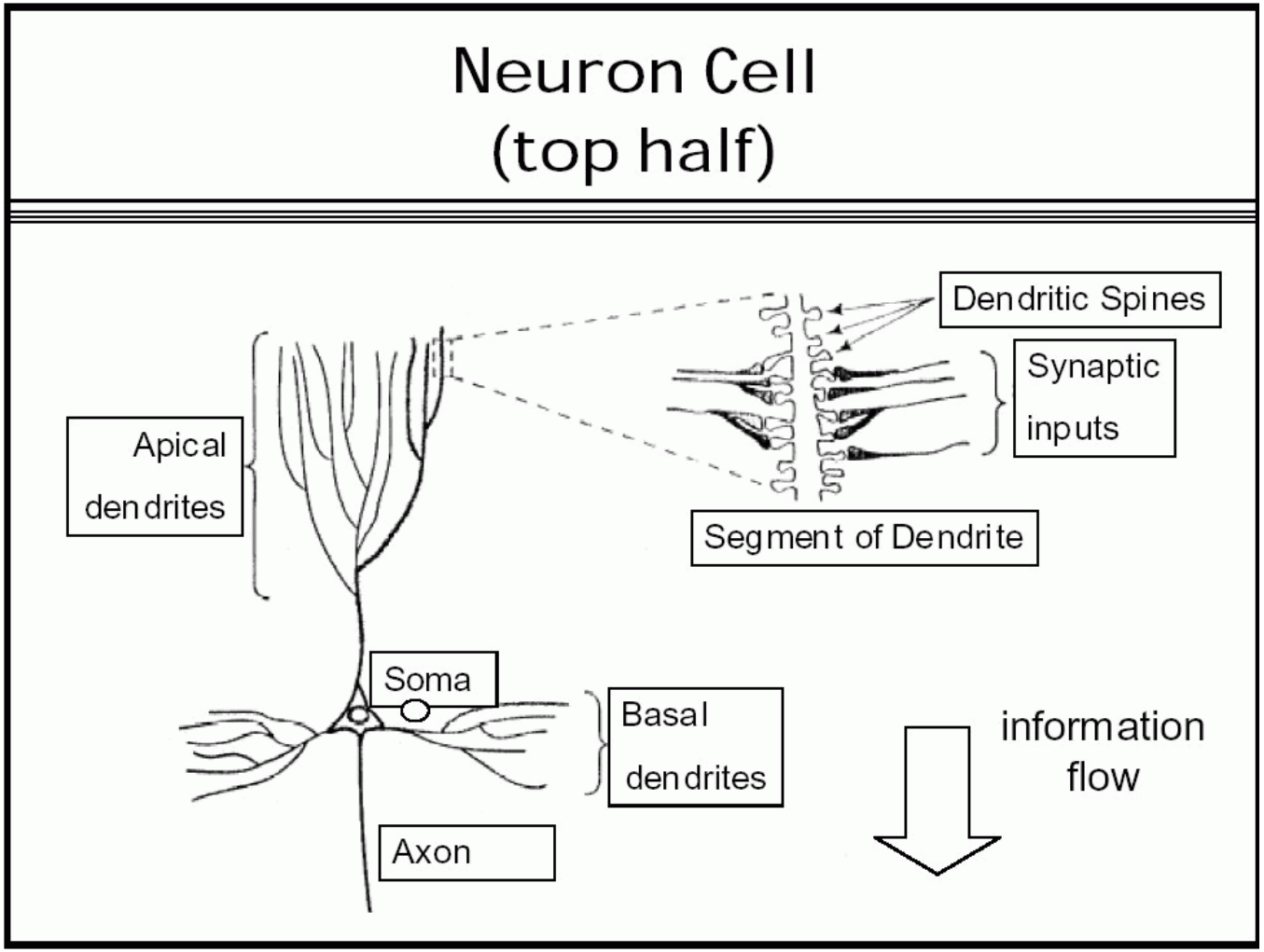
- speech recognition
- image classification
- stock market time series prediction
- control of industrial processes

Neural Networks

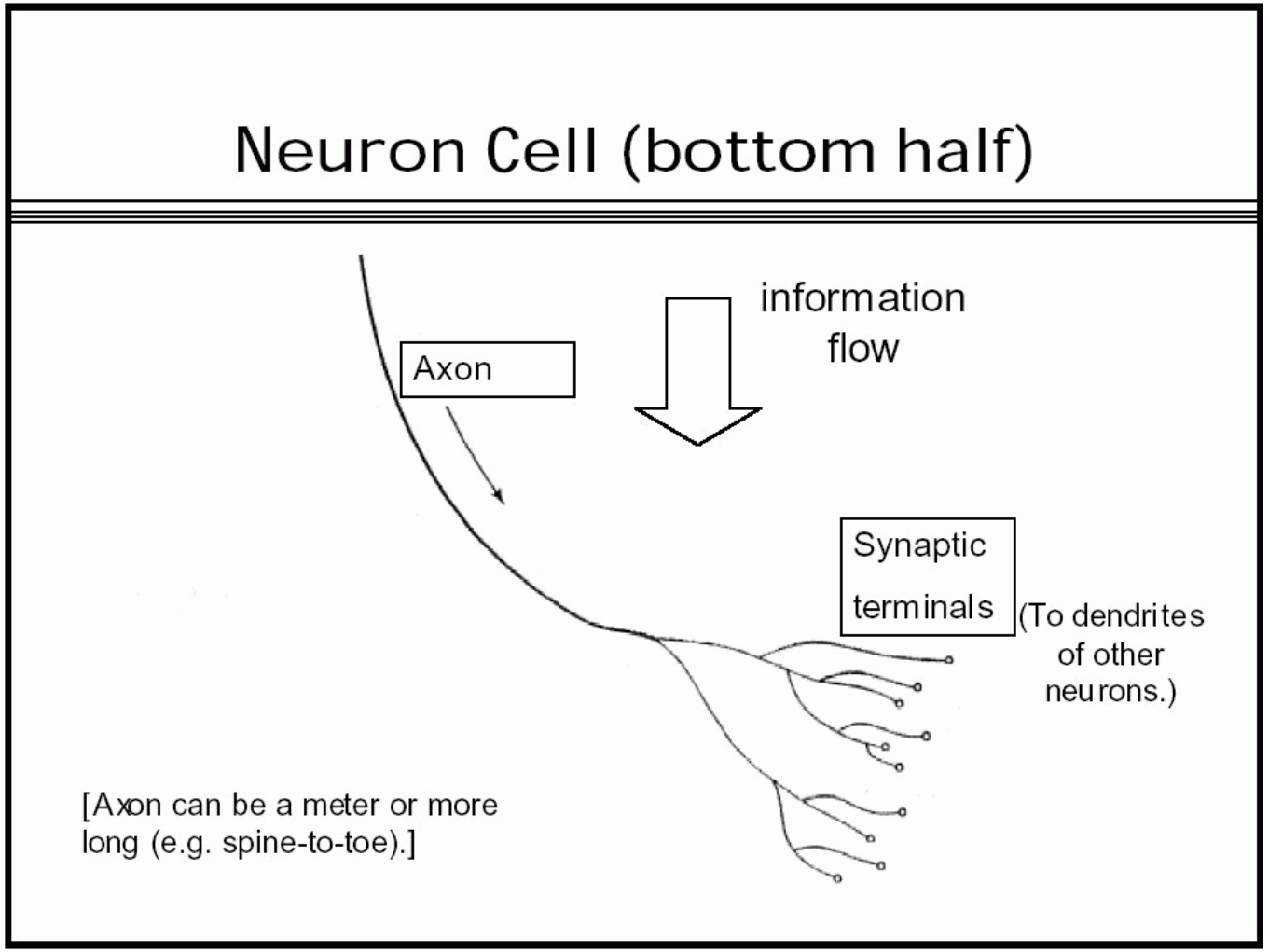
Inspired by human brain:

- Human brain constituted by about 10^{10} strongly interconnected neurons;
- Each neuron possesses a number of connections that goes from about 10^4 to about 10^5;
- The response time of a neuron is about 0.001 seconds;
- Considering that for recognizing the content of a scene a human takes about 0.1 seconds, it follows that the human brain heavily exploits parallel computing: in fact, it can not make more than 100 serial calculations [0.1/0.001=100].

Biological Neuron

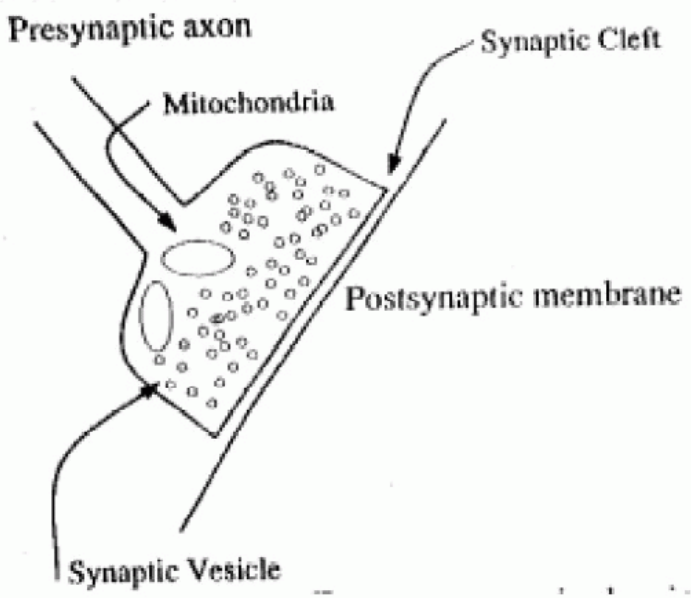


Biological Neuron



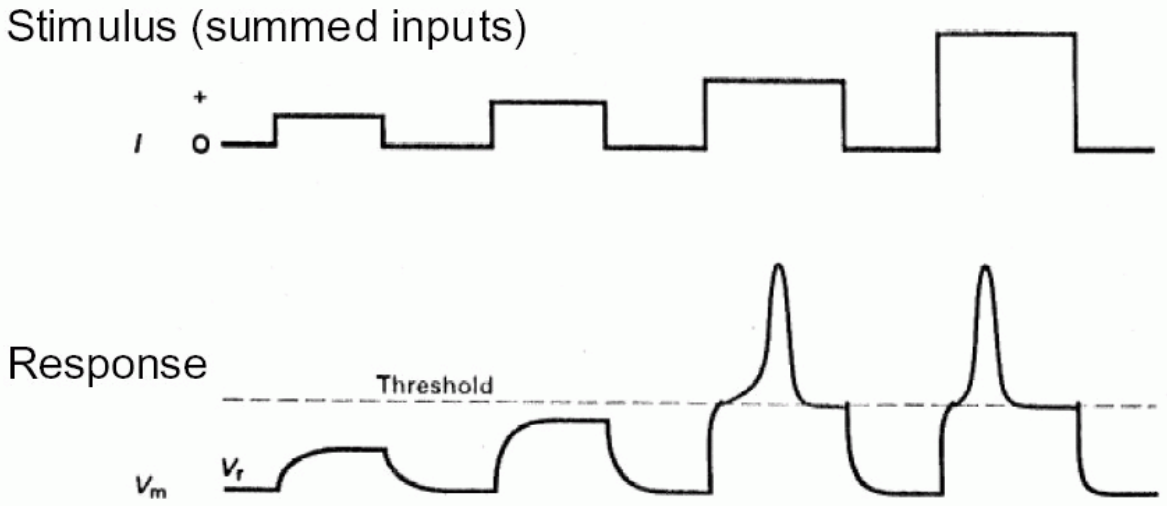
Biological Neuron

Chemical Synapse



reference: James A. Anderson, An Introduction to Neural Networks, MIT Press, 1955.

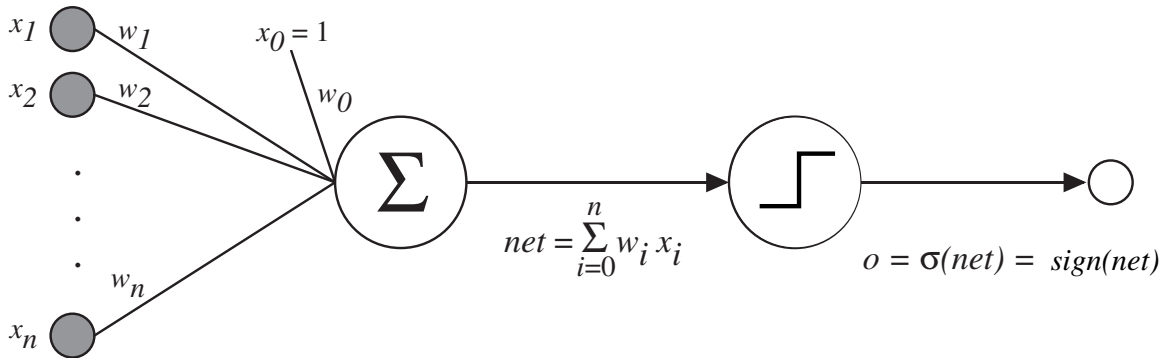
Triggering phenomenon



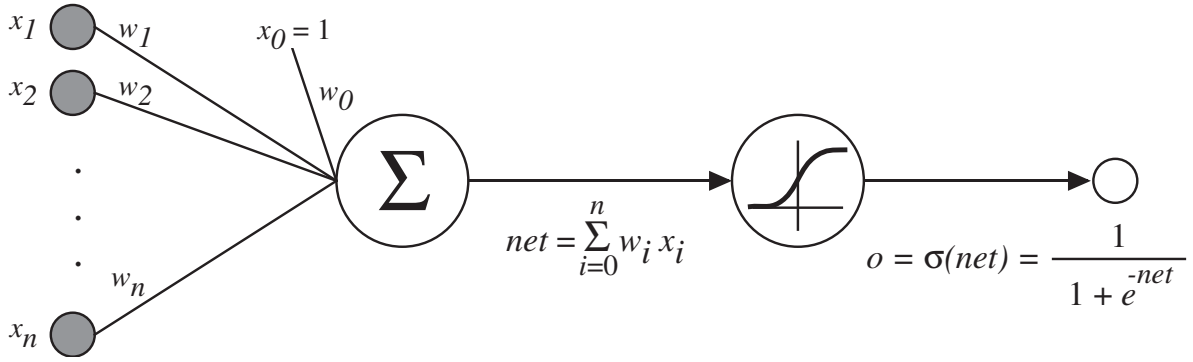
reference: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

Artificial Neuron

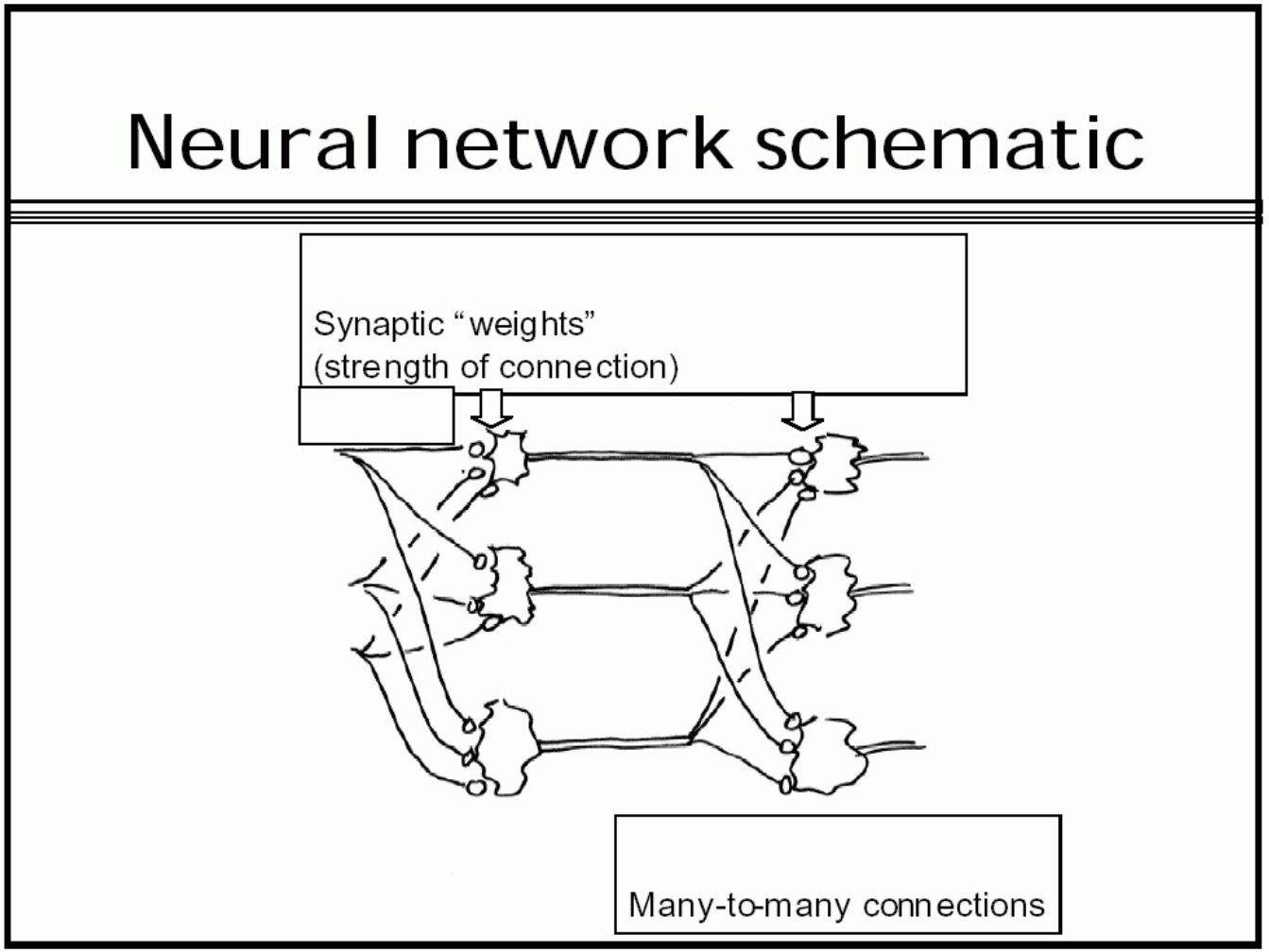
Alternative 1: hard-threshold $\rightarrow$ hyperplane!!



Alternative 2: sigmoidal neuron $\rightarrow$ derivable function



Biological Neural Networks

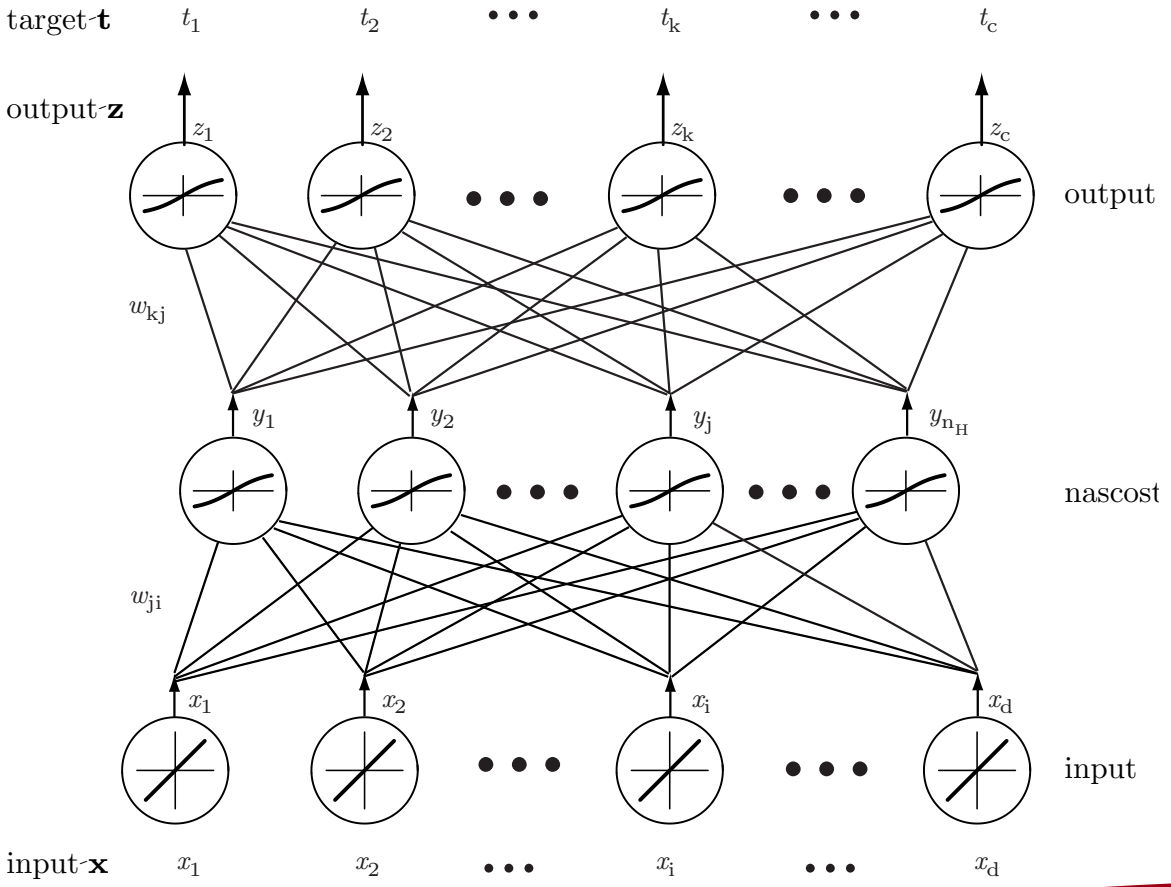


Artificial Neural Network

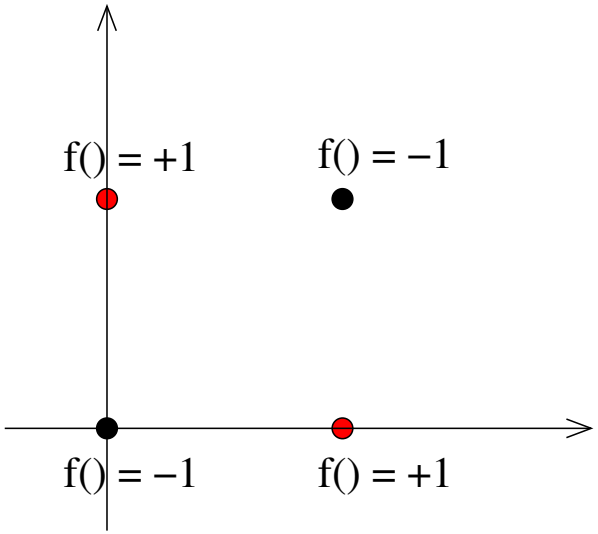
An Artificial Neural Network is a system consisting of interconnected units that compute nonlinear (numerical) functions:

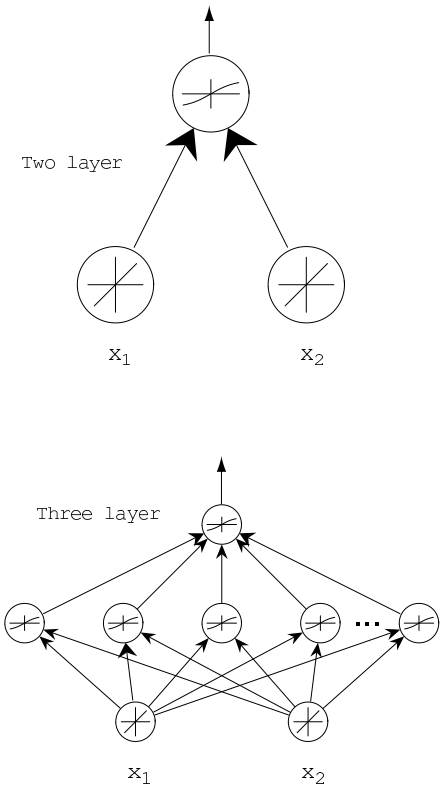
- *input* units represent input variables;
- *output* units represent output variables;
- *hidden* units (if present) represent internal variables that codify (after learning) correlations among input and desired output variables.
- adjustable *weights* are associated to connections among units.

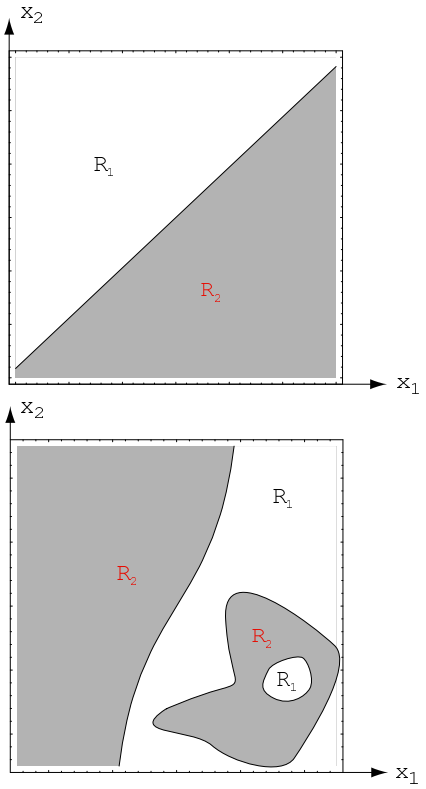
Feed-forward neural network (with a single hidden layer)



Feed-forward Neural Networks and decision surfaces







Single Neuron with Hard-Threshold

Single Neuron with Hard-Threshold: hyperplanes!

$$\mathcal{H} = \{f_{(\vec{w},b)}(\vec{y}) | f_{(\vec{w},b)}(\vec{y}) = sign(\vec{w} \cdot \vec{y} + b), \vec{w}, \vec{y} \in \mathbb{R}^n, b \in \mathbb{R}\}$$

which can be re-written as

$$\mathcal{H} = \{f_{(\vec{w}')}(\vec{y}') | f_{(\vec{w}')}(\vec{y}') = sign(\vec{w}' \cdot \vec{y}'), \vec{w}', \vec{y}' \in \mathbb{R}^{n+1}\}$$

with the following transformations

$$\vec{w}' = [b, \vec{w}]^t \qquad \vec{y}' = [1, \vec{y}]^t$$

We will refer to this neuron (and to the associated training algorithm) as Perceptron

Problem: how to train a Perceptron?

Learning linearly separable functions

Let assume to have linearly separable examples.

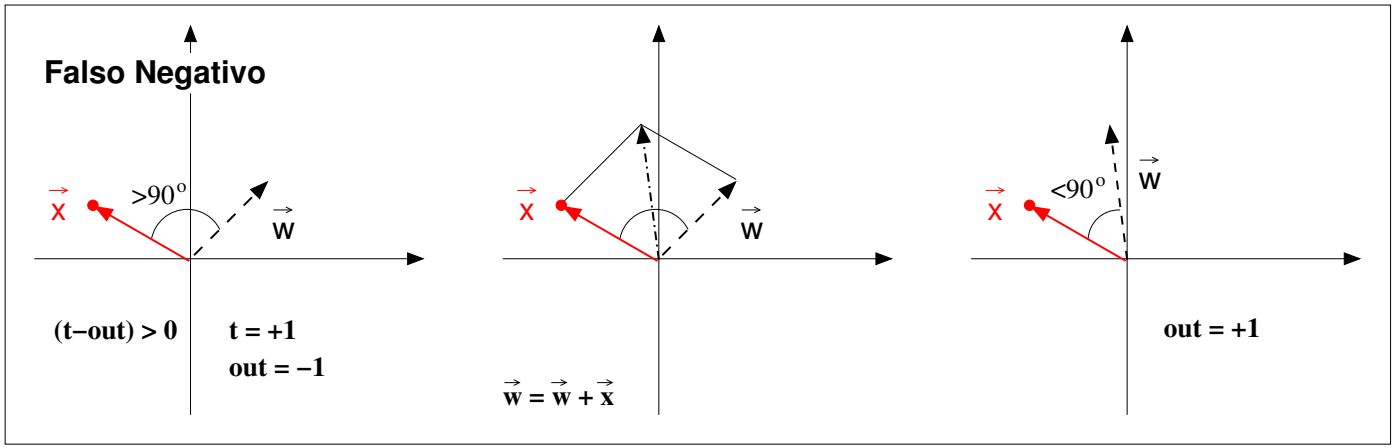
Training algorithm for a Perceptron
input: training set $Tr = \{(\vec{x}, t)\}$, where $t \in \{-1, +1\}$

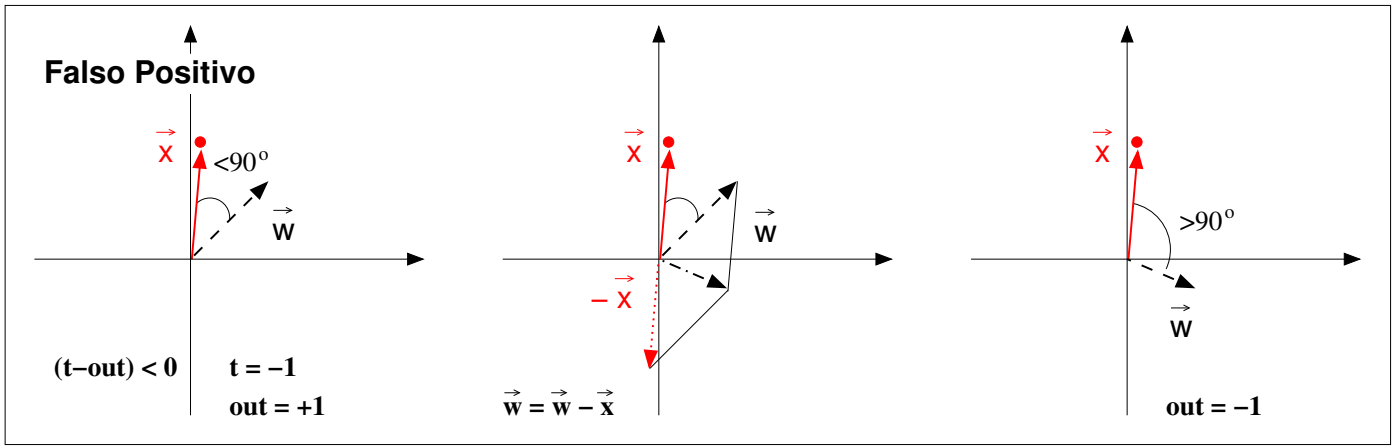
1. inizialize the weight vector $\vec{w}$ to the null vector (all components equal to 0);
2. repeat
    1. select (at random) one training example $(\vec{x}, t)$
    2. if $out = sign(\vec{w} \cdot \vec{x}) \neq t$ then

$$\vec{w} \leftarrow \vec{w} + (t - out)\vec{x}$$

Training a Perceptron

Geometric interpretation





Training a Perceptron

Not necessarily a single learning step 2(2) is capable to change the sign of the output: many steps may be needed.

To make learning more stable a learning coefficient η is introduced

$$\vec{w} \leftarrow \vec{w} + \eta(t - out)\vec{x}$$

with $\eta > 0$ and preferably $\eta < 1$

This prevents the weight vector to undergo too "sharp" changes whenever step 2(2) is performed. This could avoid previously well classified examples to become misclassified because of the strong variation of the weight vector.

If the training set is linearly SEPARABLE, it can be shown that the training algorithm for the Perceptron terminates in a finite number of steps with a solution, otherwise $\vec{w}$ LOOPS into a set of not necessarily optimal weight vectors (i.e, which commit the minimum possible number of errors)

Training a Perceptron: an example

Tr:

ex.	$\vec{x}$	target
1	(4,5)	1
2	(6,1)	1
3	(4,1)	-1
4	(1,2)	-1

$\xrightarrow{\text{threshold}}$

ex.	$\vec{x}'$	target
1'	(1,4,5)	1
2'	(1,6,1)	1
3'	(1,4,1)	-1
4'	(1,1,2)	-1

weights vector: $\vec{w} = (w_0, w_1, w_2)$

assume to start with "random" weights: $\vec{w} = (0, 1, -1)$ and $\eta = \frac{1}{2}$

weights	input	target	out	error	new weights
(0,1,-1)	(1,4,5)	1			
	(1,6,1)	1			
	(1,4,1)	-1			
	(1,1,2)	-1			

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(0,1,-1)	(1,4,5)	1	-1	2	(1,5,4)
(1,5,4)	(1,6,1)	1	1	0	no change
(1,5,4)	(1,4,1)	-1	1	-2	(0,1,3)
(0,1,3)	(1,1,2)	-1	1	-2	(-1,0,1)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-1,0,1)	(1,4,5)	1	1	0	no change
(-1,0,1)	(1,6,1)	1	? (-1)	2	(0,6,2)
(0,6,2)	(1,4,1)	-1	1	-2	(-1,2,1)
(-1,2,1)	(1,1,2)	-1	1	-2	(-2,1,-1)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-2,1,-1)	(1,4,5)	1	-1	2	(-1,5,4)
(-1,5,4)	(1,6,1)	1	1	0	no change
(-1,5,4)	(1,4,1)	-1	1	-2	(-2,1,3)
(-2,1,3)	(1,1,2)	-1	1	-2	(-3,0,1)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-3,0,1)	(1,4,5)	1	1	0	no change
(-3,0,1)	(1,6,1)	1	-1	2	(-2,6,2)
(-2,6,2)	(1,4,1)	-1	1	-2	(-3,2,1)
(-3,2,1)	(1,1,2)	-1	1	-2	(-4,1,-1)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-4,1,-1)	(1,4,5)	1	-1	2	(-3,5,4)
(-3,5,4)	(1,6,1)	1	1	0	no change
(-3,5,4)	(1,4,1)	-1	1	-2	(-4,1,3)
(-4,1,3)	(1,1,2)	-1	1	-2	(-5,0,1)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-5,0,1)	(1,4,5)	1	? (-1)	2	(-4,4,6)
(-4,4,6)	(1,6,1)	1	1	0	no change
(-4,4,6)	(1,4,1)	-1	1	-2	(-5,0,5)
(-5,0,5)	(1,1,2)	-1	1	-2	(-6,-1,3)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-6,-1,3)	(1,4,5)	1	1	0	no change
(-6,-1,3)	(1,6,1)	1	-1	2	(-5,5,4)
(-5,5,4)	(1,4,1)	-1	1	-2	(-6,1,3)
(-6,1,3)	(1,1,2)	-1	1	-2	(-7,0,1)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-7,0,1)	(1,4,5)	1	-1	2	(-6,4,6)
(-6,4,6)	(1,6,1)	1	1	0	no change
(-6,4,6)	(1,4,1)	-1	1	-2	(-7,0,5)
(-7,0,5)	(1,1,2)	-1	1	-2	(-8,-1,3)

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-8,-1,3)	(1,4,5)	1	1	0	no change
(-8,-1,3)	(1,6,1)	1	-1	2	(-7,5,2)
(-7,5,2)	(1,4,1)	-1	1	-2	(-8,1,3)
(-8,1,3)	(1,1,2)	-1	-1	0	no change

Training a Perceptron: an example

weights $\vec{w}$	input $\vec{x}'$	target t	out out	error $(t - out)$	new weights $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t - out)\vec{x}'$
(-8,1,3)	(1,4,5)	1	1	0	no change
(-8,1,3)	(1,6,1)	1	1	0	no change
(-8,1,3)	(1,4,1)	-1	-1	0	no change
(-8,1,3)	(1,1,2)	-1	-1	0	no change

Learning in a network of Perceptrons

A single Perceptron in not able to learn all possibile boolean functions (e.g., XOR)
However a network of Perceptrons can implement any boolean function (via AND, OR, NOT).
Problem: how to train a network of Perceptrons ?
It is not clear how to assign "credit" or "blame" to the hidden units:

CREDIT ASSIGNMENT PROBLEM

A possibile solution is to make a single neuron derivable and exploit the Gradient Descent technique to learn the "right" weights.

Let see how the Gradient Descent technique can be applied to a "simplified" Perceptron.

Gradient descent

Let consider a Perceptron WITHOUT hard-threshold:

$$out(\vec{x}) = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$

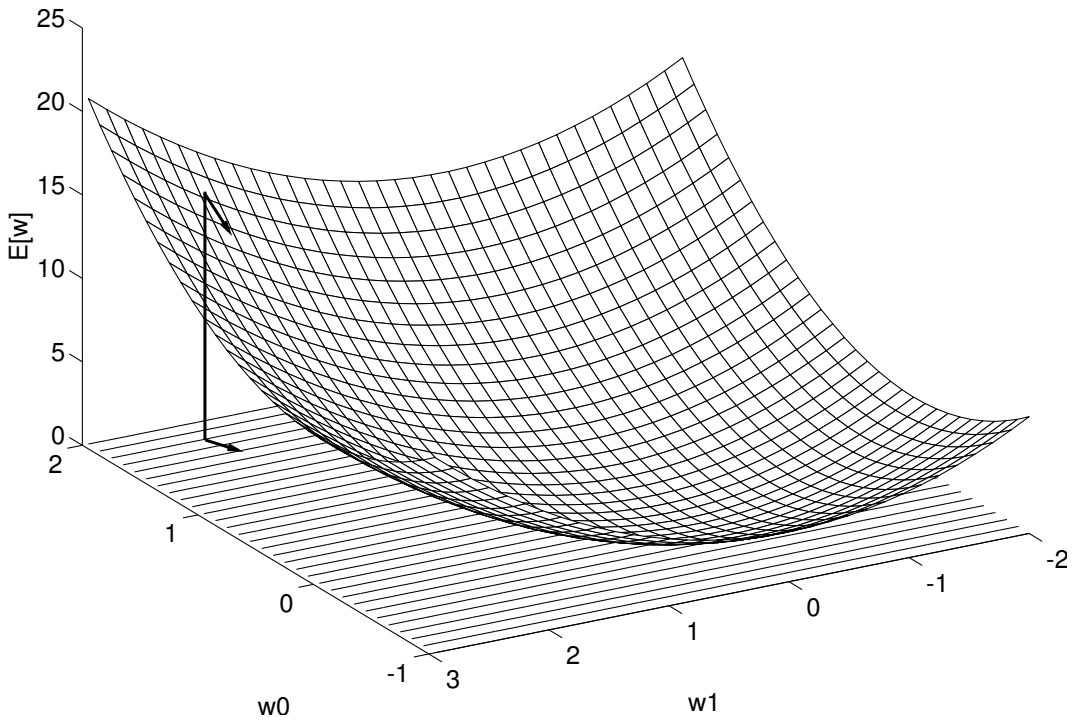
and let define a loss function for a weight vector

$$\textbf{Error function: } E[\vec{w}] = \frac{1}{2N_{Tr}} \sum_{(\vec{x}^{(i)}, t^{(i)}) \in Tr} \left(t^{(i)} - out(\vec{x}^{(i)}) \right)^2$$

where N_{Tr} is the cardinality of the training set Tr.
The error function measures the mean squared error of the target value with respect to the neuron output (out).
Of course, if $\forall (\vec{x}^{(i)}, t^{(i)}) \in Tr$ then $out(\vec{x}^{(i)}) = t^{(i)} \rightarrow E[\vec{w}] = 0$

Thus $E[\vec{w}]$ should be MINIMISED with respect to $\vec{w}$

Gradient descent



Basic idea: start from a random $\vec{w}$ and adjust it along the opposite of the gradient direction (the gradient points in the direction of the greatest rate of increase of $E[\vec{w}]$)

$$\underbrace{\nabla E[\vec{w}]}_{\text{gradient}} \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right], \quad \Delta\vec{w} = -\eta \nabla E[\vec{w}], \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient computation

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2$$

Gradient computation

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})^2
\end{aligned}
$$

Gradient computation

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} 2(t^{(d)} - out^{(d)}) \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})
\end{aligned}
$$

Gradient computation

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} 2(t^{(d)} - out^{(d)}) \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)}) \\
&= \frac{1}{N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)}) \frac{\partial}{\partial w_i} (t^{(d)} - \vec{w} \cdot \vec{x}^{(d)})
\end{aligned}
$$

Gradient computation

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\vec{w}\cdot\vec{x}^{(d)}) \\
\frac{\partial E}{\partial w_i} &= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(-x_i^{(d)}) \\
&= -\frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(x_i^{(d)})
\end{aligned}
$$

Gradient descent

Gradient-Descent(Tr, η)

each training example is a couple $(\vec{x}, t)$, where $\vec{x}$ is the input vector, and t is the output desired value (target). η is the learning parameter (which includes $\frac{1}{N_{Tr}}$).

- Set w_i with small random values
- Until stop condition is false, do
  - $\Delta w_i \leftarrow 0$
  - For all $(\vec{x}, t)$ in Tr, do
    - Feed $\vec{x}$ to the neuron and compute output *out*
    - For all w_i, do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - out)x_i$$

  - For all w_i, do

$$w_i \leftarrow w_i + \Delta w_i$$

Gradient descent with sigmoidal units

Let consider a Perceptron with sigmoidal function:

$$out(\vec{x}) = \sigma(\sum_{i=0}^{n} w_i x_i) = \sigma(\vec{w} \cdot \vec{x})$$

where we recall that $\sigma(net) = \frac{1}{1+e^{-net}}$
Notice that for $\sigma()$ the following relation holds

$$\sigma'(net) = \frac{d\ \sigma(net)}{d\ net} = \sigma(net)(1 - \sigma(net))$$

and recall that (derivative of composition of functions)

$$\frac{d\ f(g(x))}{d\ x} = \frac{d\ f(g(x))}{d\ g(x)} \frac{d\ g(x)}{d\ x}$$

Gradient descent with sigmoidal units

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} 2(t^{(d)} - out^{(d)}) \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})
\end{aligned}
$$

Gradient descent with sigmoidal units

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}} \sum_{d \in Tr} 2(t^{(d)} - out^{(d)}) \frac{\partial}{\partial w_i} (t^{(d)} - out^{(d)}) \\
&= \frac{1}{N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)}) \frac{\partial}{\partial w_i} (t^{(d)} - \sigma(\vec{w} \cdot \vec{x}^{(d)}))
\end{aligned}
$$

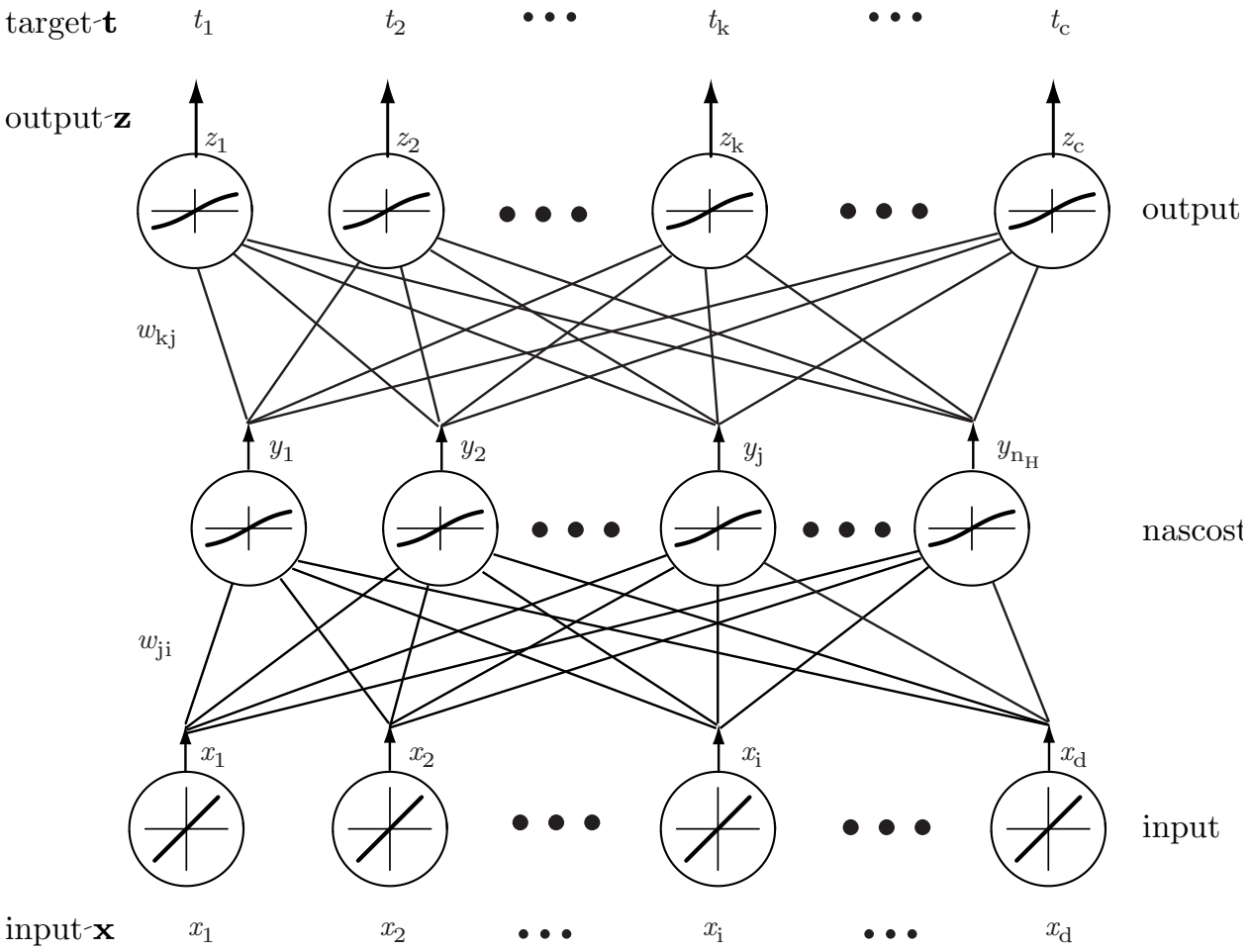
Gradient descent with sigmoidal units

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)} - out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)} - \sigma(\vec{w}\cdot\vec{x}^{(d)})) \\
\frac{\partial E}{\partial w_i} &= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})(-\frac{\partial\sigma(\vec{w}\cdot\vec{x}^{(d)})}{\partial\vec{w}\cdot\vec{x}^{(d)}}\frac{\partial\vec{w}\cdot\vec{x}^{(d)}}{\partial w_i})
\end{aligned}
$$

Gradient descent with sigmoidal units

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\sigma(\vec{w}\cdot\vec{x}^{(d)})) \\
\frac{\partial E}{\partial w_i} &= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(-\frac{\partial\sigma(\vec{w}\cdot\vec{x}^{(d)})}{\partial\vec{w}\cdot\vec{x}^{(d)}}\frac{\partial\vec{w}\cdot\vec{x}^{(d)}}{\partial w_i}) \\
&= -\frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\sigma(\vec{w}\cdot\vec{x}^{(d)})(1-\sigma(\vec{w}\cdot\vec{x}^{(d)}))x_i^{(d)}
\end{aligned}
$$

Feed-forward neural networks: notation



Feed-forward neural networks: notation

- d input units, size input $\vec{x} \equiv (x_1, \ldots, x_d)$
  ($d+1$ if the threshold is included into the weight vector $\vec{x}' \equiv (x_0, x_1, \ldots, x_d)$)
- N_H hidden units (with output $\vec{y} \equiv (y_1, \ldots, y_{N_H})$)
- c output units, output size $\vec{z} \equiv (z_1, \ldots, z_c)$
- c, size target vectors $\vec{t} \equiv (t_1, \ldots, t_c)$
- w_{ji} weight on the connection from input unit i to hidden unit j
- w_{kj} weight on the connection from hidden unit j to output unit k

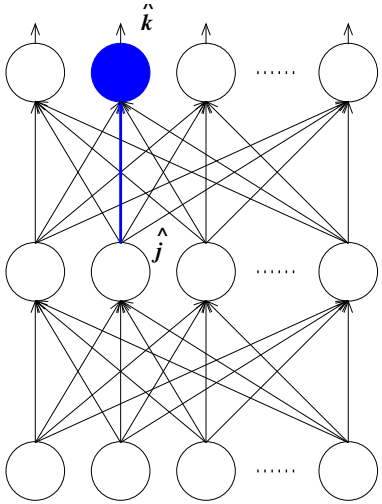
Function error, with c output units:

$$E[\vec{w}] = \frac{1}{2cN_{Tr}} \sum_{(\vec{x}^{(p)}, \vec{t}^{(p)}) \in Tr} \sum_{k=1}^{c} \left(t_k^{(p)} - z_k(\vec{x}^{(p)}) \right)^2$$

Gradient computation for output units

Let fix $\hat{k}$ and $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

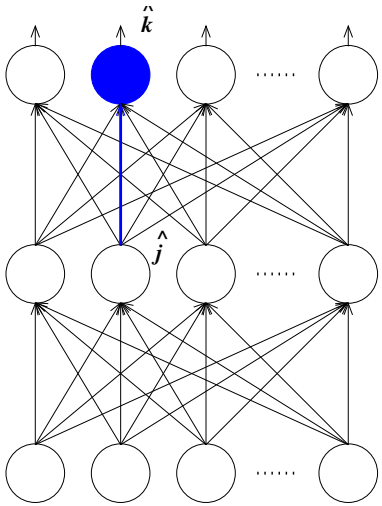


Gradient computation for output units

Let fix $\hat{k}$ and $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

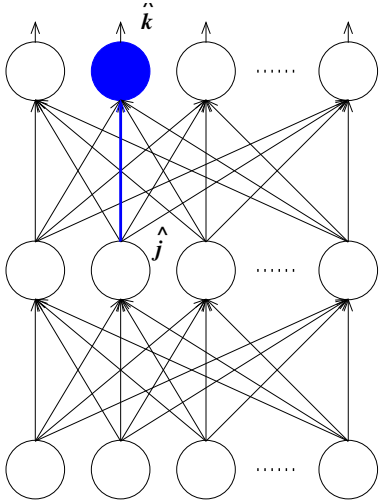
$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$



Gradient computation for output units

Let fix $\hat{k}$ and $\hat{j}$:

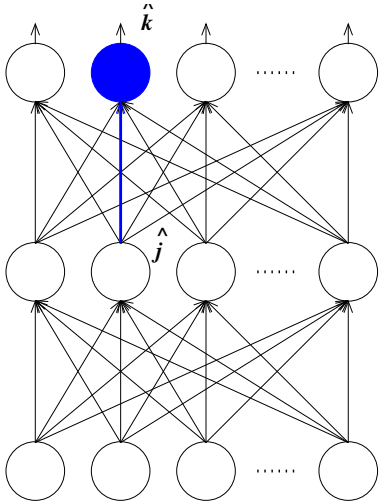
$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} &= \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})
\end{aligned}
$$



Gradient computation for output units

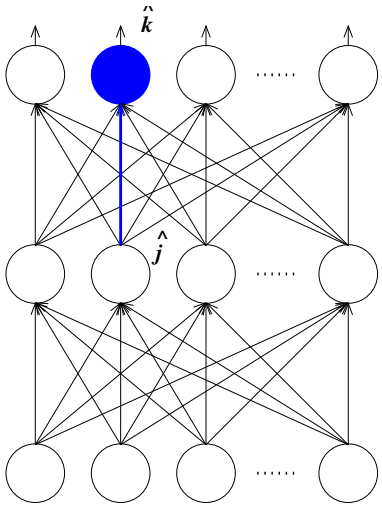
Let fix $\hat{k}$ and $\hat{j}$:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} &= \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \\
&= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}))
\end{aligned}
$$



Gradient computation for output units

Let fix $\hat{k}$ and $\hat{j}$:

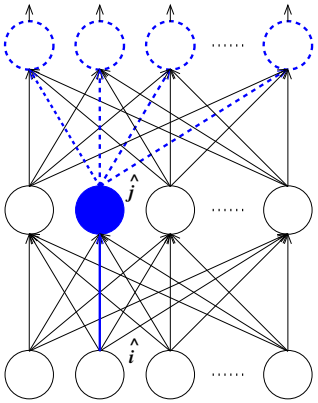


$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} &= \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \\
&= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)})) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \sigma'(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}) y_{\hat{j}}^{(p)}
\end{aligned}
$$

Gradient computation for hidden units

Let fix $\hat{j}$ and $\hat{i}$:

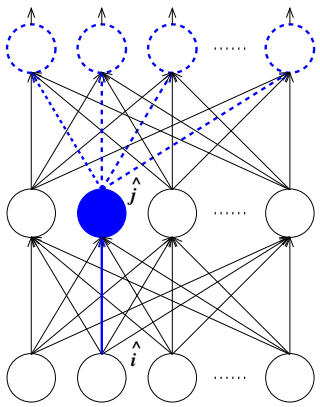
$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$



Gradient computation for hidden units

Let fix $\hat{j}$ and $\hat{i}$:

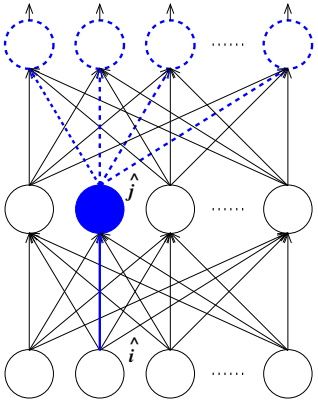
$$\begin{aligned}\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\ &= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2\end{aligned}$$



Gradient computation for hidden units

Let fix $\hat{j}$ and $\hat{i}$:

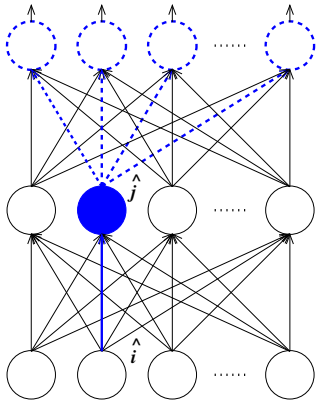
$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})
\end{aligned}
$$



Gradient computation for hidden units

Let fix $\hat{j}$ and $\hat{i}$:

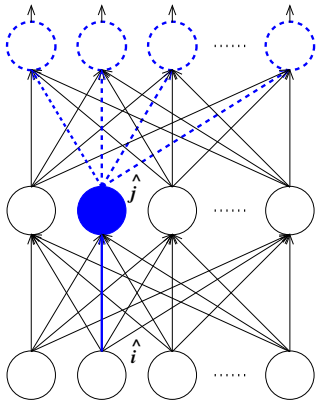
$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)}) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \vec{w}_k \cdot \vec{y}^{(p)}
\end{aligned}
$$



Gradient computation for hidden units

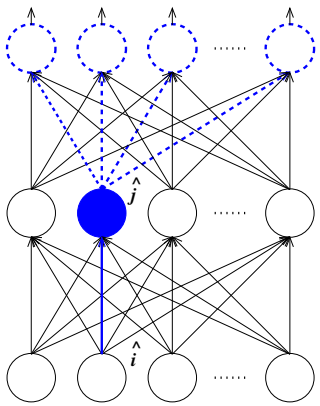
Let fix $\hat{j}$ and $\hat{i}$:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)}) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}
\end{aligned}
$$



Gradient computation for hidden units

Let fix $\hat{j}$ and $\hat{i}$:



$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)}) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)} \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}
\end{aligned}
$$

Gradient computation for hidden units

Let fix $\hat{j}$ and $\hat{i}$:

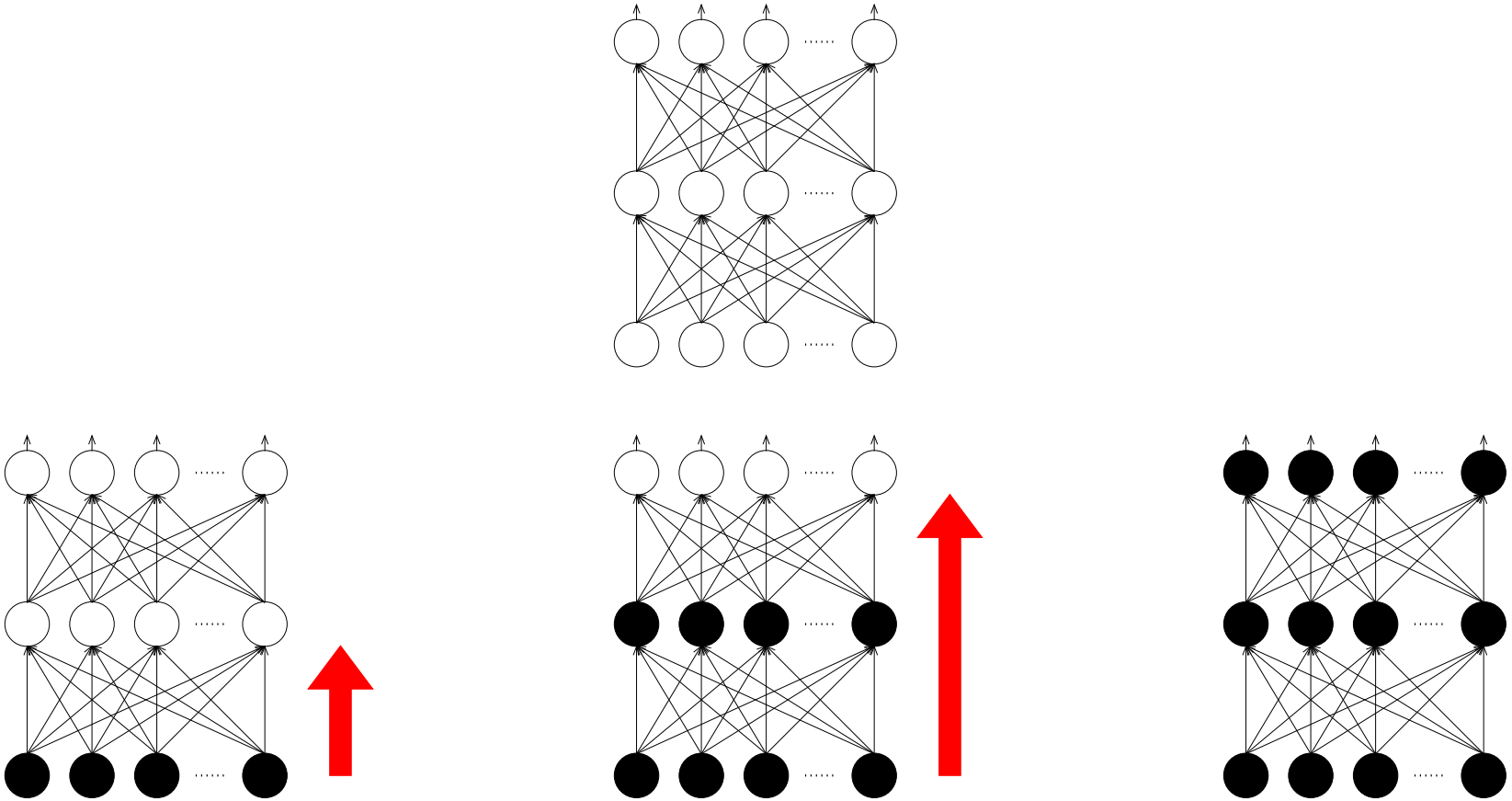
$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)}) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)} \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)} \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})
\end{aligned}
$$

Gradient computation for hidden units

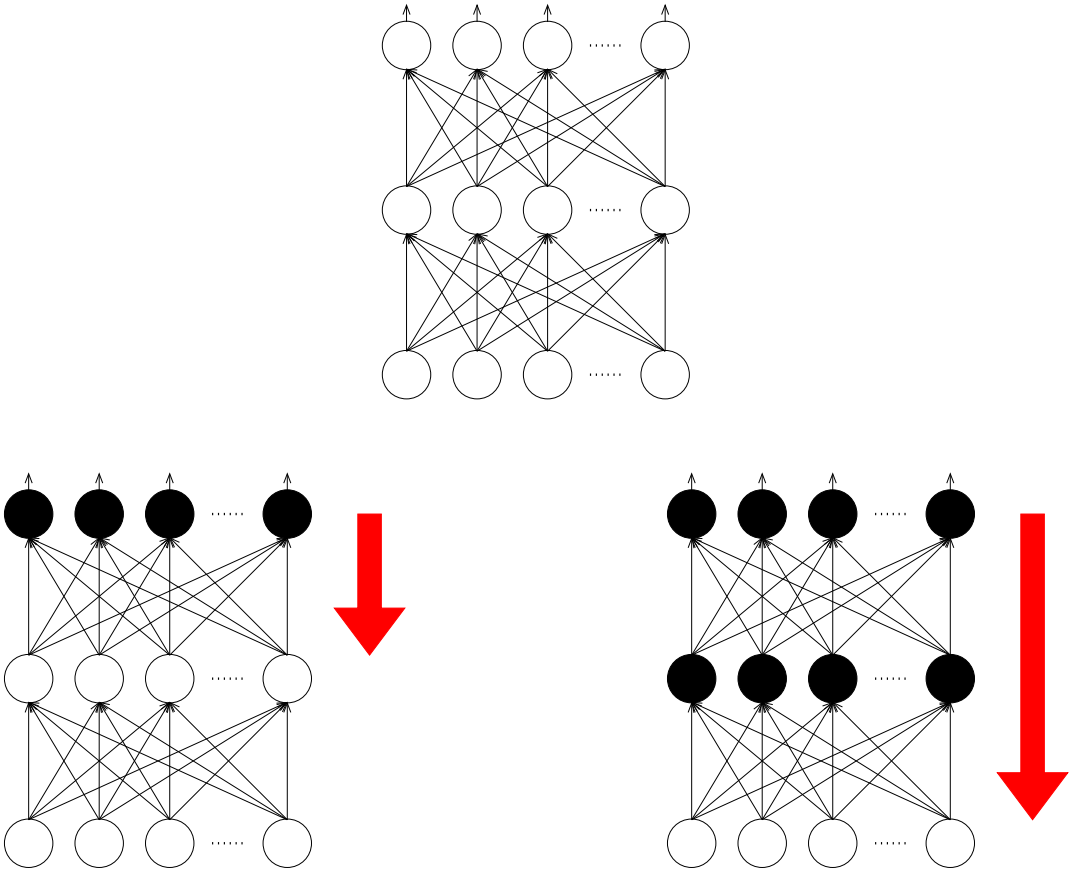
Let fix $\hat{j}$ and $\hat{i}$:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} &= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)}) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)} \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)} \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) x_{\hat{i}}^{(p)}
\end{aligned}
$$

Forward Phase



Backward Phase



Back-Propagation (single hidden layer, stochastic)

Back-Propagation-1hl-stochastic(Tr, η, *network topology*)

- Initialize all weights to small random values
- Until stop condition is false, do
  - For all $(\vec{x}, \vec{t})$ in Tr, do
    1. feed $\vec{x}$ to the network and compute the associated output
    2. For all output units k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

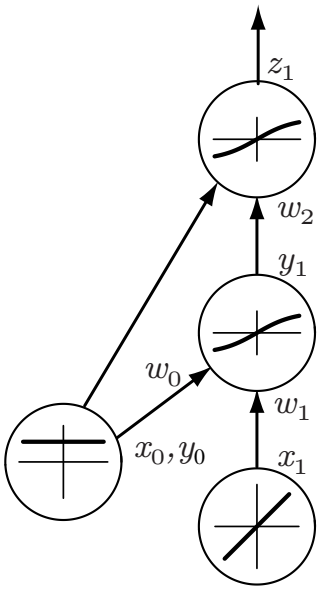
    3. For all hidden units j

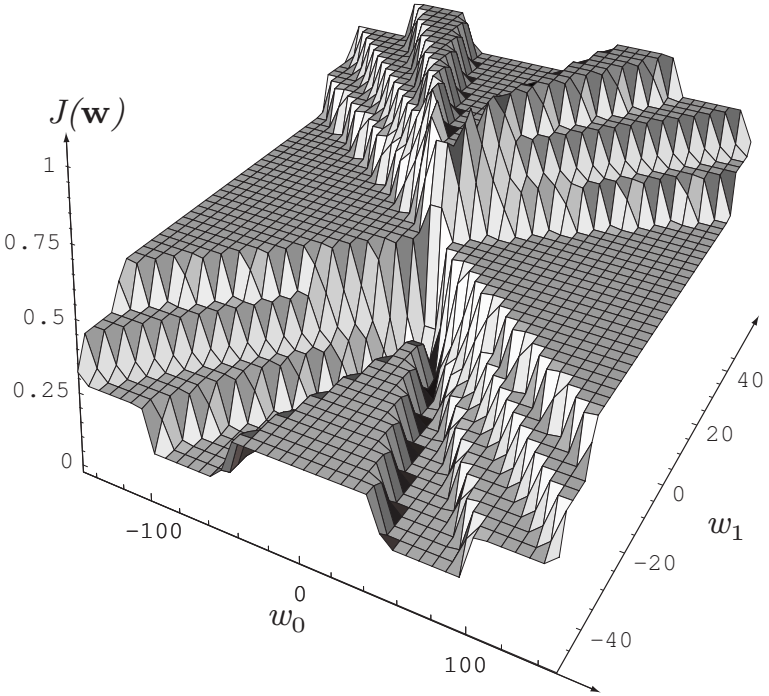
$$\delta_j \leftarrow o_j(1 - o_j) \sum_{k \in outputs} w_{k,j}\delta_k$$

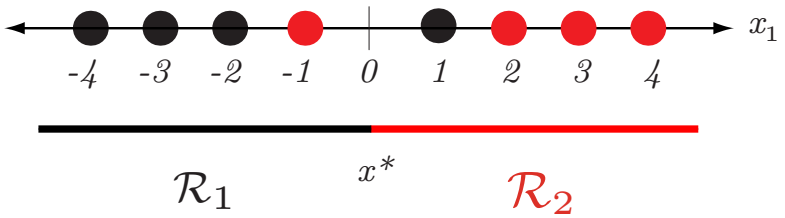
    4. update all weights $w_{p,q}$ of the networks

$$w_{s,q} \leftarrow w_{s,q} + \eta \Delta w_{s,q} \quad \text{where} \quad \Delta w_{s,q} = \begin{cases} \delta_s x_q & \text{if } s \in hidden \\ \delta_s y_q & \text{if } s \in outputs \end{cases}$$

Example of Error Function







Batch and Stochastic Gradient Descent

Batch:
Do until stop condition is false

1. compute $\nabla E_{Tr}[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{Tr}[\vec{w}]$

Stochastic (Incremental):
Do until stop condition is false

- For all training example p in Tr
  1. compute $\nabla E_{p \in Tr}[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{p \in Tr}[\vec{w}]$

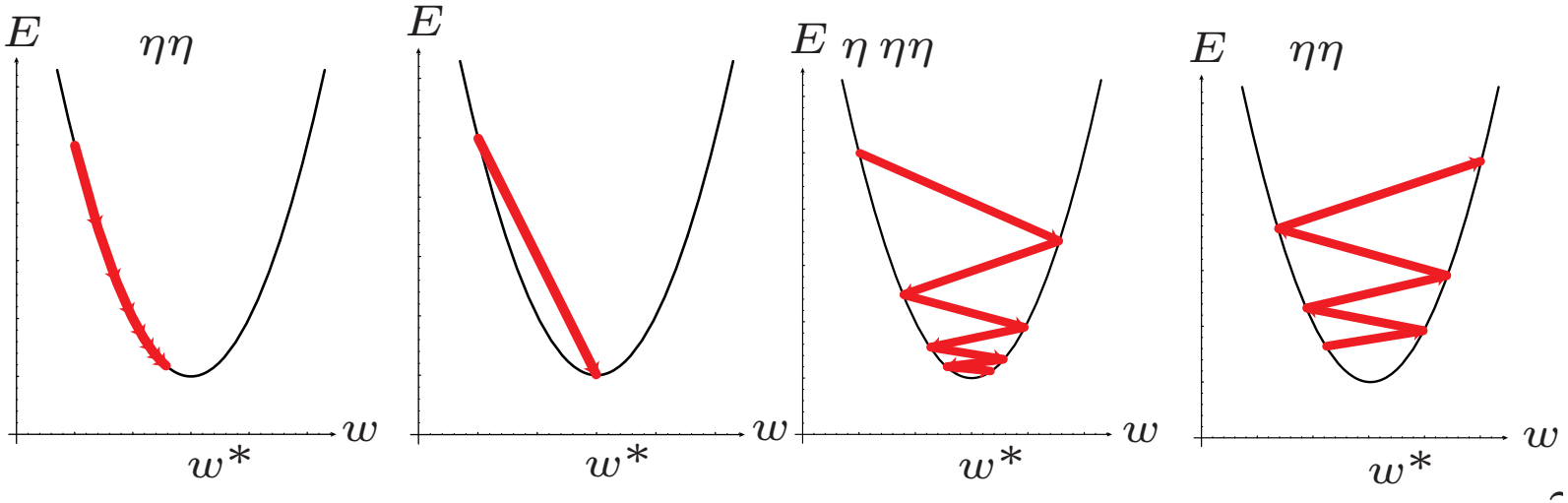
where

$$E_{Tr}[\vec{w}] \equiv \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \qquad E_{p \in Tr}[\vec{w}] \equiv \frac{1}{2c} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

Stochastic gradient descent (instantaneous gradient) can approximate *Batch* gradient descent (exact gradient) with arbitrary precision if η is sufficiently small

Some problems ...

- Choice of network topology $\rightarrow$ determines Hypothesis Space;
- Choice of step descent size (value of η):



- slow learning..., but fast output computation
- LOCAL MINIMA!!

Inductive Bias: both in representation and in learning

Computational power

The following theorem establishes universality of feed-forward neural networks as approximators of continuos functions.

Theorem Let $\varphi(\cdot)$ a monotonically increasing continuous, bounded and non constant function. Denote by I_n the n-dimensional hypercube $[0,1]^n$ and let $C(I_n)$ denote the space of continuos function defined over it. Given any function $f \in C(I_n)$ and $\varepsilon > 0$, there exists an integer M and sets of real valued constants α_i, θ_i, and w_{ij}, where $i = 1, \ldots, M$ and $j = 1, \ldots, n$ such that $f(\cdot)$ can be approximated by

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{M} \alpha_i \varphi(\sum_{j=1}^{n} w_{ij} x_j - \theta_i) \tag{1}$$

in such a way that

$$|F(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)| < \varepsilon \tag{2}$$

for all points $[x_1, \ldots, x_n] \in I_n$.

Computational power

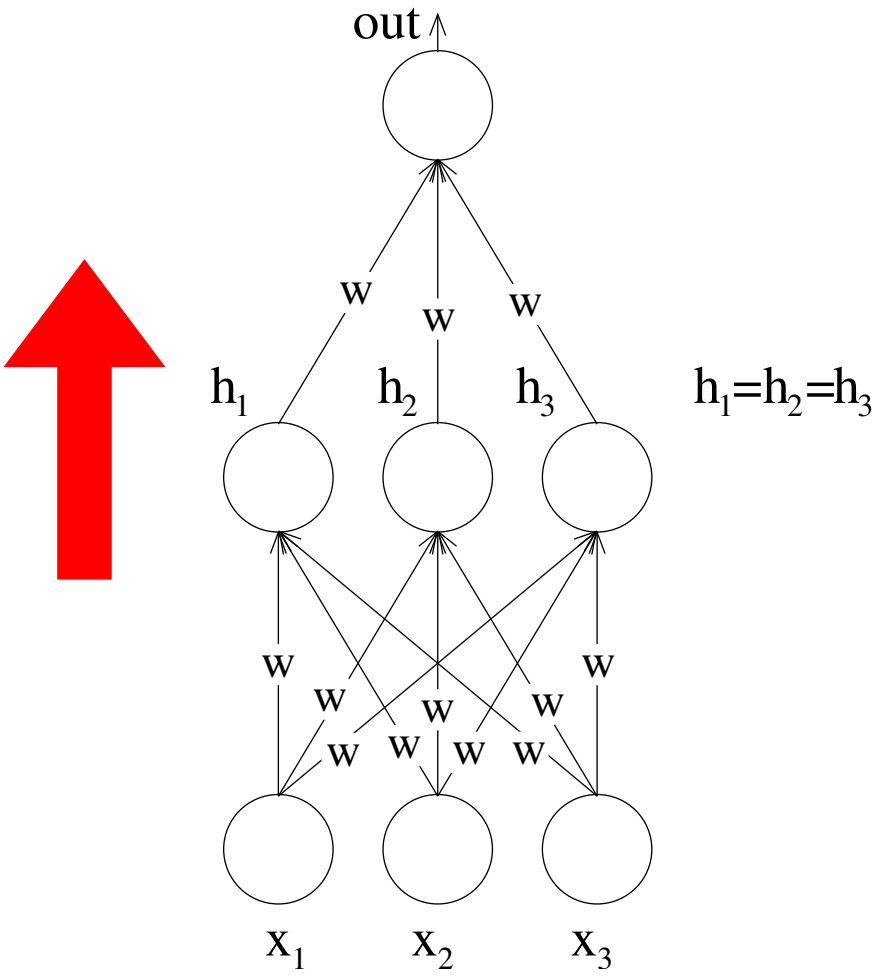
Notice that any sigmoidal function satisfies the conditions required by the theorem on$\varphi(\cdot)$. Moreover, equation (1) represents the output of a multilayer network described as follows

1. the network has n input units and a single hidden layer with M units; inputs are denoted as $x_1, \ldots, x_n$.
2. the i-th unit has associated weights $w_{i1}, \ldots, w_{in}$ and threshold θ_i.
3. the network output is a linear combination of the hidden units outputs, where the coefficients of the combination are given by $\alpha_1, \ldots, \alpha_M$.

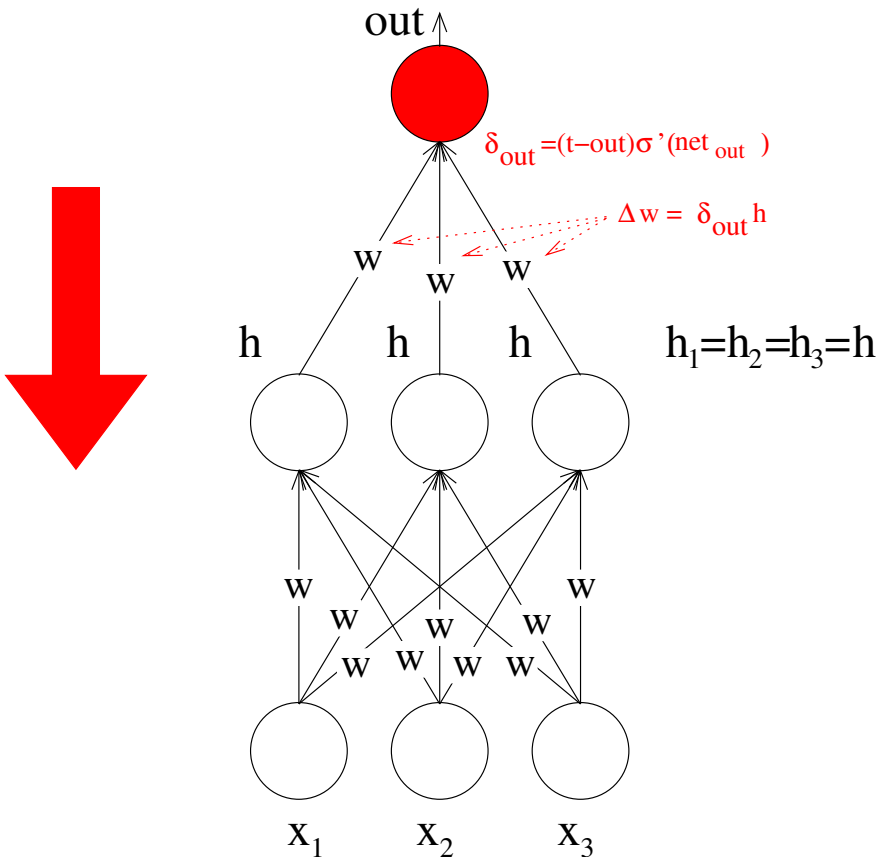
Thus, given a tolerance ε, a network with a single hidden layer can approximate any function in $C(I_n)$.

Notice that the theorem only states the existence of a network, however it does not give any formula for the computation of M (number of hidden units needed to approximate the target function with the desired tolerance.)

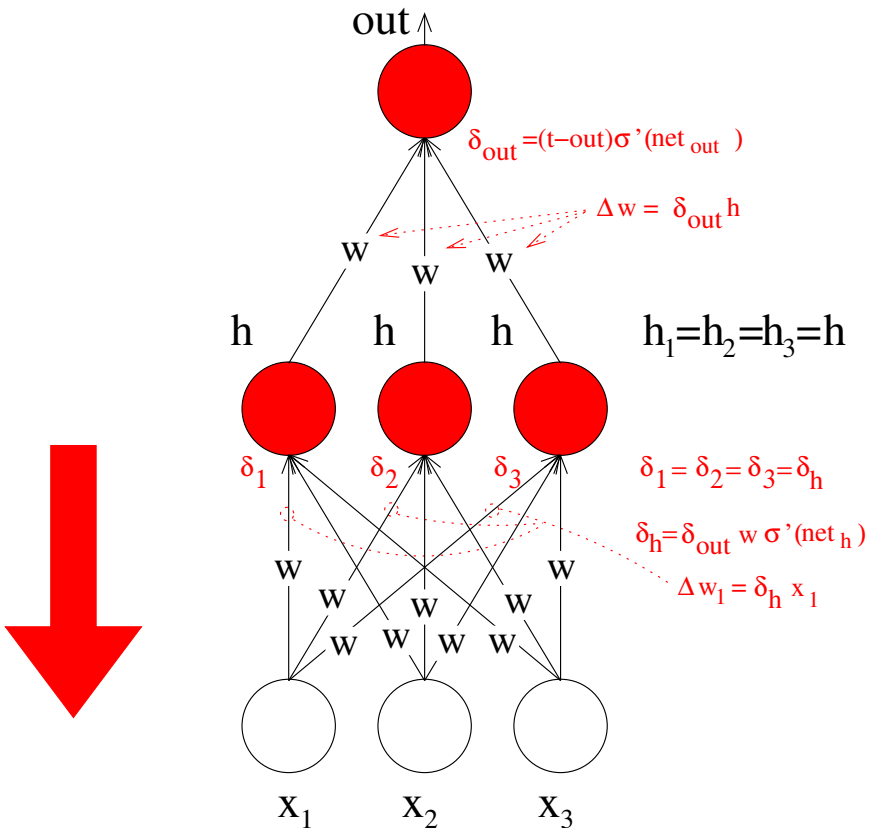
Symmetries



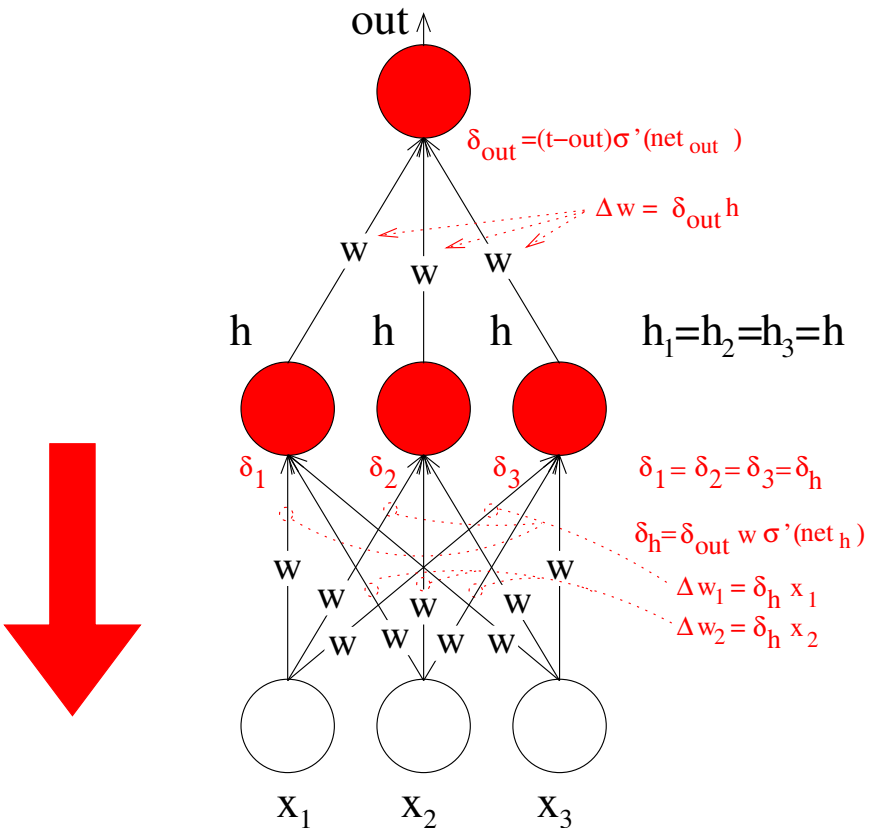
Symmetries


out
δout =(t−out)σ'(netout)
Δw = δout h
w
w
w
h
h
h
h1=h2=h3=h
w
w
w
w
w
w
w
w
x1
x2
x3

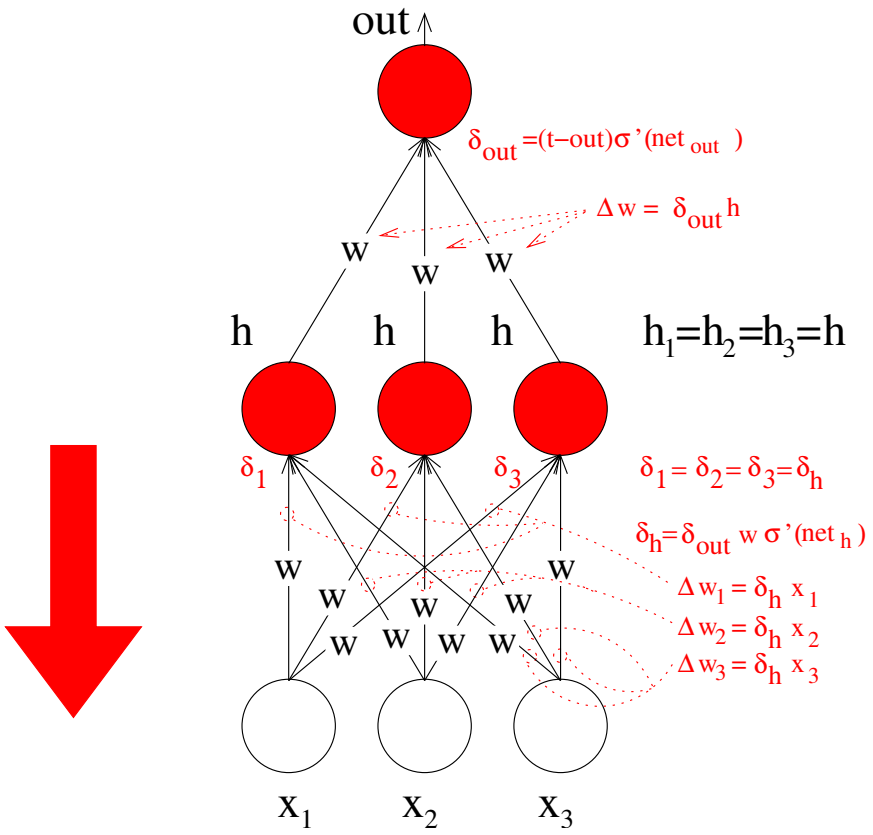
Symmetries


out
$\delta_{out} = (t - out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
w
w
h
h
h
$h_1 = h_2 = h_3 = h$
δ_1
δ_2
δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out}\, w\, \sigma'(net_h)$
$\Delta w_1 = \delta_h\, x_1$
w
w
w
w
w
w
w
w
x_1
x_2
x_3

Symmetries


out
$\delta_{out} = (t - out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
δ_1
δ_2
δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out}\, w\, \sigma'(net_h)$
$\Delta w_1 = \delta_h\, x_1$
$\Delta w_2 = \delta_h\, x_2$
x_1
x_2
x_3

Symmetries


out
$\delta_{out} = (t - out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
δ_1
δ_2
δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out}\, w\, \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
$\Delta w_2 = \delta_h x_2$
$\Delta w_3 = \delta_h x_3$
x_1
x_2
x_3

New trend: Deep Learning

1. Neural Networks with many hidden layers (*deep networks*)
   1. Insufficient depth can hurt
   2. The brain has a deep architecture
   3. Cognitive processes seem deep
2. Networks with probabilistic interpretation (Boltzmann Machines)
3. Use of unsupervised learning (autoencoders) for incremental training (one layer at a time)
4. Brute force training using GPUs