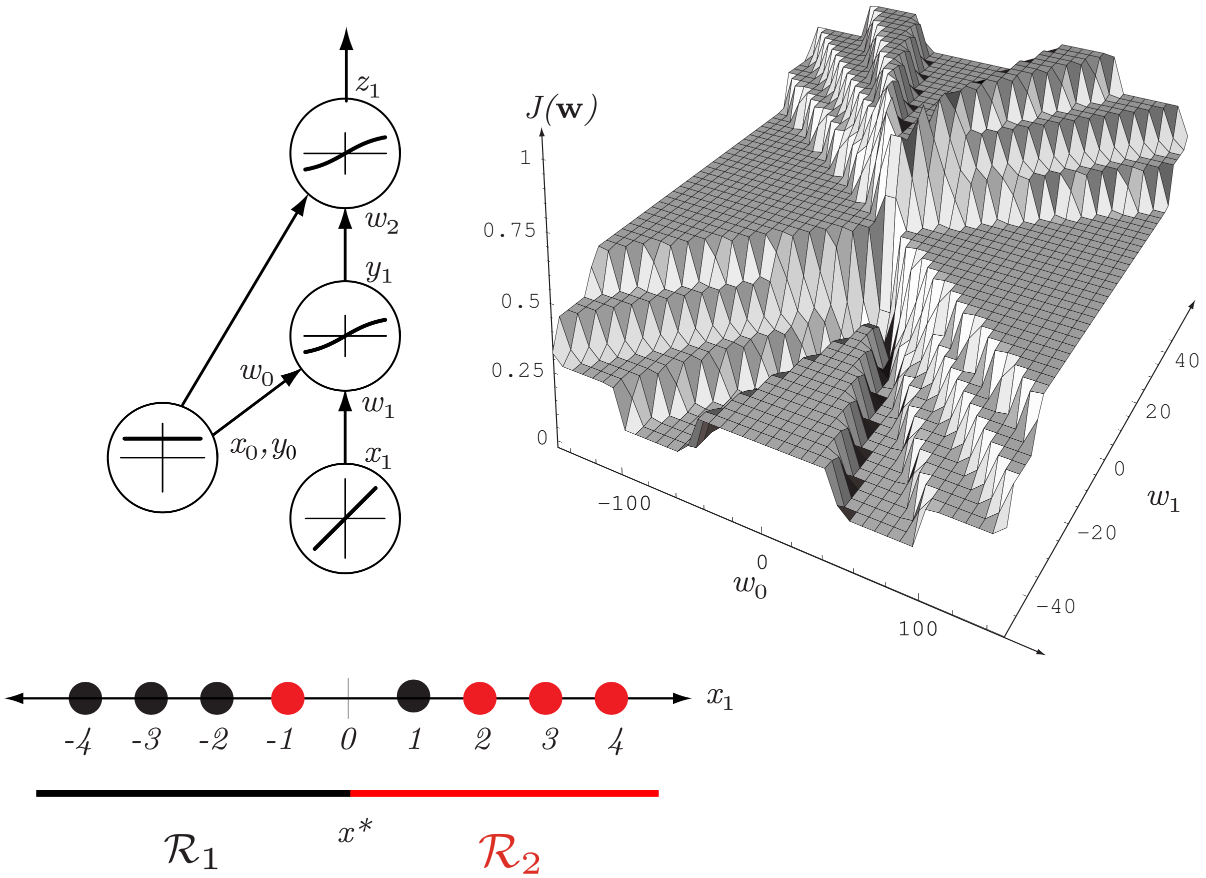
Example of Error Function



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. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{p \in Tr}[\vec{w}]$

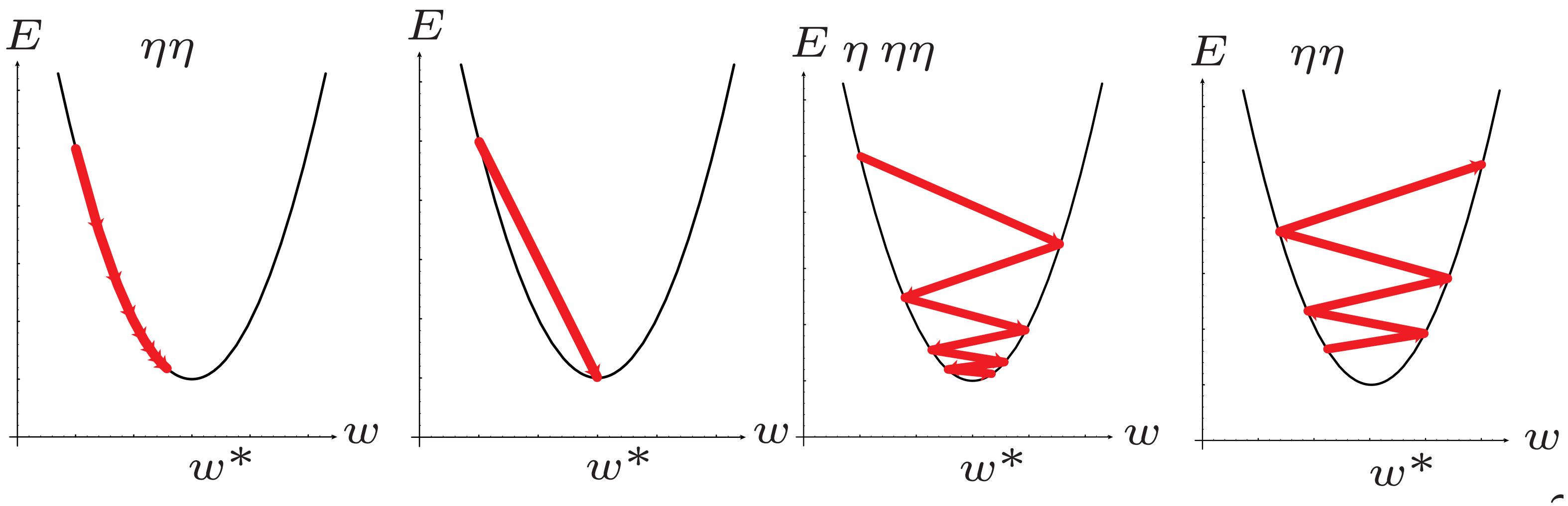
where

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

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
- $\boxed{\text{LOCAL MINIMA!!}}$

Inductive Bias: both in representation and in learning

Some Problems ...

Vanishing of the gradient with network depth

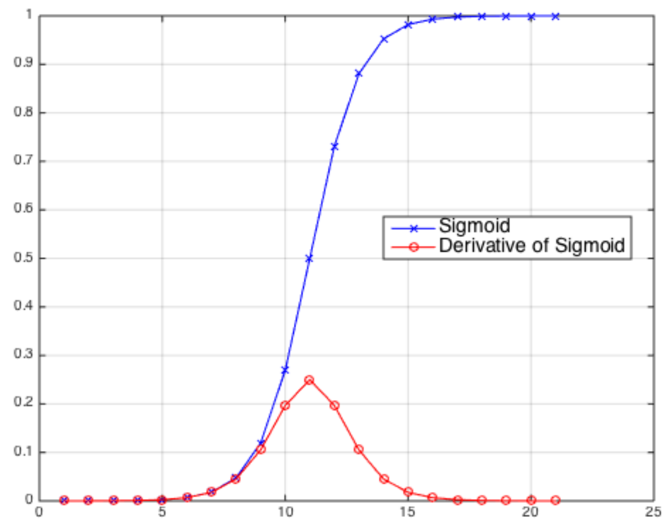
- The gradient of error for hidden weights $\mathbf{W}_l$ at layer $l < L$ in a network with L layers is given by

$$\frac{\partial E}{\partial \mathbf{W}_l} = \frac{\partial E}{\partial \mathbf{h}_L}\frac{\partial \mathbf{h}_L}{\partial \mathbf{W}_l} = \frac{\partial E}{\partial \mathbf{h}_L}\frac{\partial \mathbf{h}_L}{\partial \mathbf{h}_{L-1}}\frac{\partial \mathbf{h}_{L-1}}{\partial \mathbf{W}_l} = \frac{\partial E}{\partial \mathbf{h}_L}\left(\prod_{\tau=0}^{L-l-1}\frac{\partial \mathbf{h}_{L-\tau}}{\partial \mathbf{h}_{L-\tau-1}}\right)\frac{\partial \mathbf{h}_l}{\partial \mathbf{W}_l}$$

- the term $\frac{\partial \mathbf{h}_{L-\tau}}{\partial \mathbf{h}_{L-\tau-1}}$ depends on
  - hidden-to-hidden weights $\mathbf{W}_{L-\tau}$ and sigmoid(-like) first derivative

$$\frac{\partial \mathbf{h}_{L-\tau}}{\partial \mathbf{h}_{L-\tau-1}} = \mathbf{W}_{L-\tau}^{\mathsf{T}} \underbrace{\begin{bmatrix} \bullet & 0 & 0 & 0 & 0 \\ 0 & \bullet & 0 & 0 & 0 \\ 0 & 0 & \bullet & 0 & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & 0 & 0 & 0 & \bullet \end{bmatrix}}_{Jacobian\ of\ sigmoidal\ unit}$$

where $\bullet < \frac{1}{4}$ (sigmoid unit)



Some Problems ...

$$\frac{\partial \mathbf{h}_L}{\partial \mathbf{h}_{L-3}} = \mathbf{W}_L^\top \begin{bmatrix} \bullet & 0 & 0 & 0 & 0 \\ 0 & \bullet & 0 & 0 & 0 \\ 0 & 0 & \bullet & 0 & 0 \\ .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & \bullet \end{bmatrix} \mathbf{W}_{L-1}^\top \begin{bmatrix} \bullet & 0 & 0 & 0 & 0 \\ 0 & \bullet & 0 & 0 & 0 \\ 0 & 0 & \bullet & 0 & 0 \\ .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & \bullet \end{bmatrix} \mathbf{W}_{L-2}^\top \begin{bmatrix} \bullet & 0 & 0 & 0 & 0 \\ 0 & \bullet & 0 & 0 & 0 \\ 0 & 0 & \bullet & 0 & 0 \\ .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & \bullet \end{bmatrix}$$

- if the module of the hidden weight matrices is large enough, then $\|\frac{\partial \mathbf{h}_L}{\partial \mathbf{h}_{L-\tau}}\|$ will increase for increasing values of τ (*exploding gradient* $\Rightarrow$ hard to decide the "right" descent step size)
- if the module of the hidden weight matrices is small, then *gradients vanish* going backward in the network
  $\Rightarrow$ no learning with gradient descent...

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

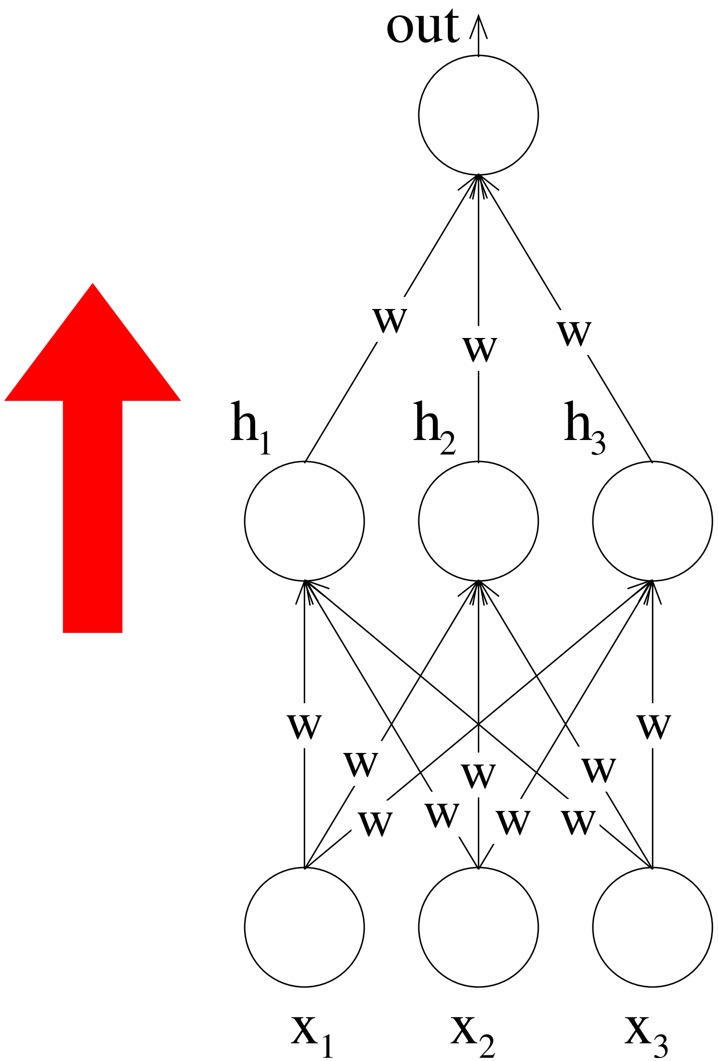
Notice that any sigmoidal function satisfies the conditions required by the theorem on$\varphi(\cdot)$. Moreover, equation (1) represents the output of a multilayer network described as follows

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 $C(I_n)$.

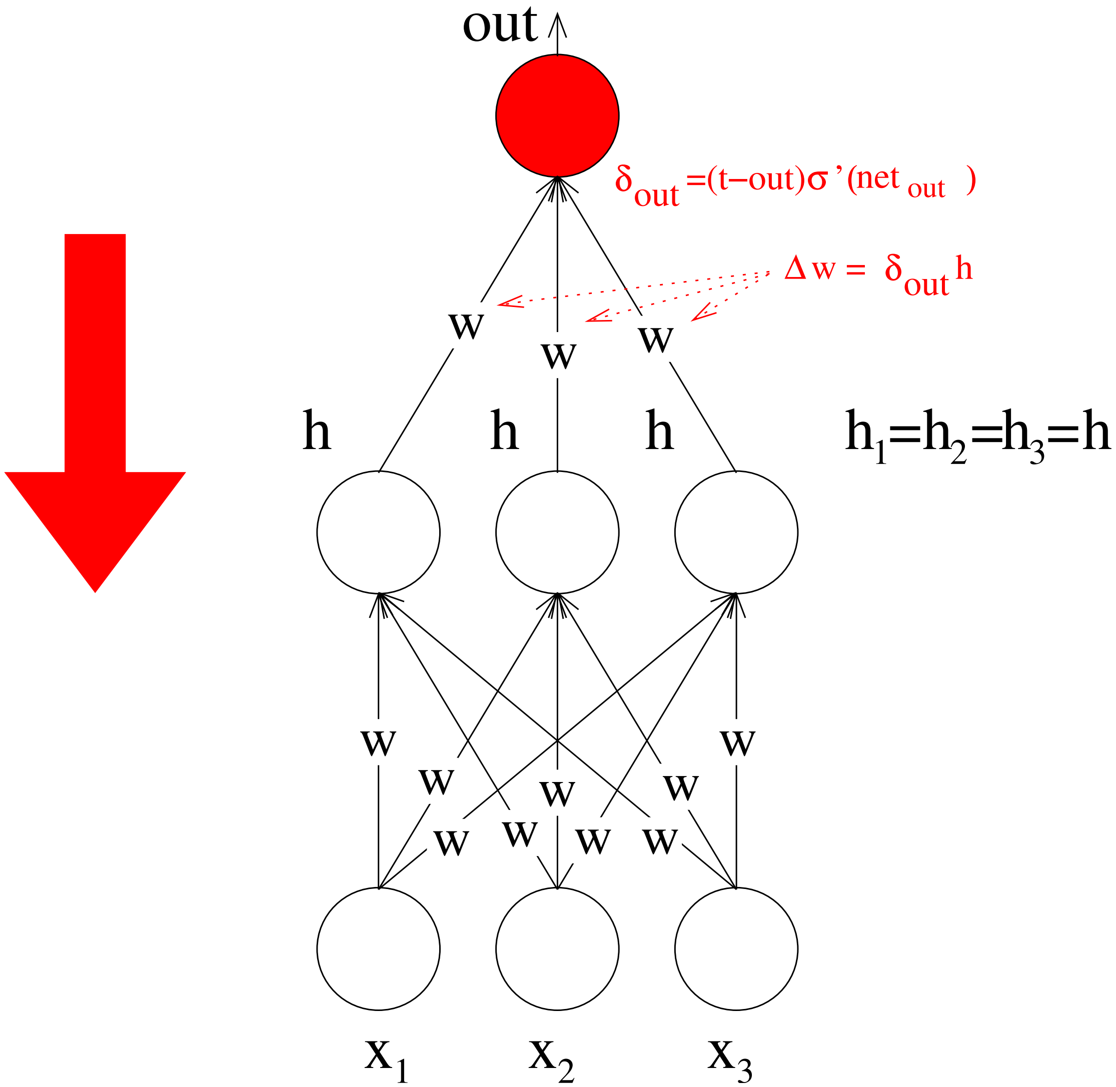
Notice that the theorem only states the existence of a network, however it does not give any formula for the computation of M (number of hidden units needed to approximate the target function with the desired tolerance.)

Symmetries

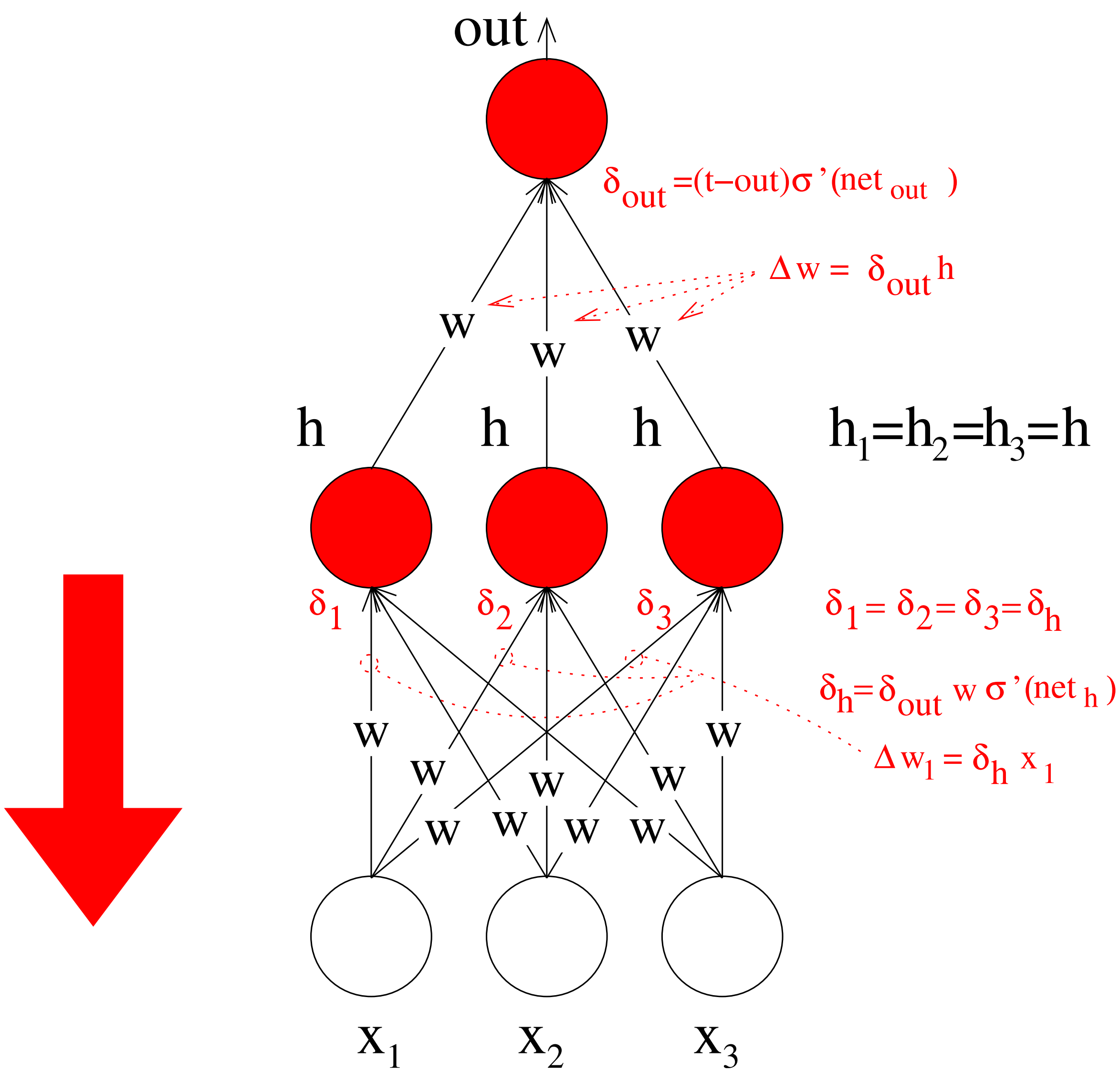


$h_1 = h_2 = h_3$

Symmetries

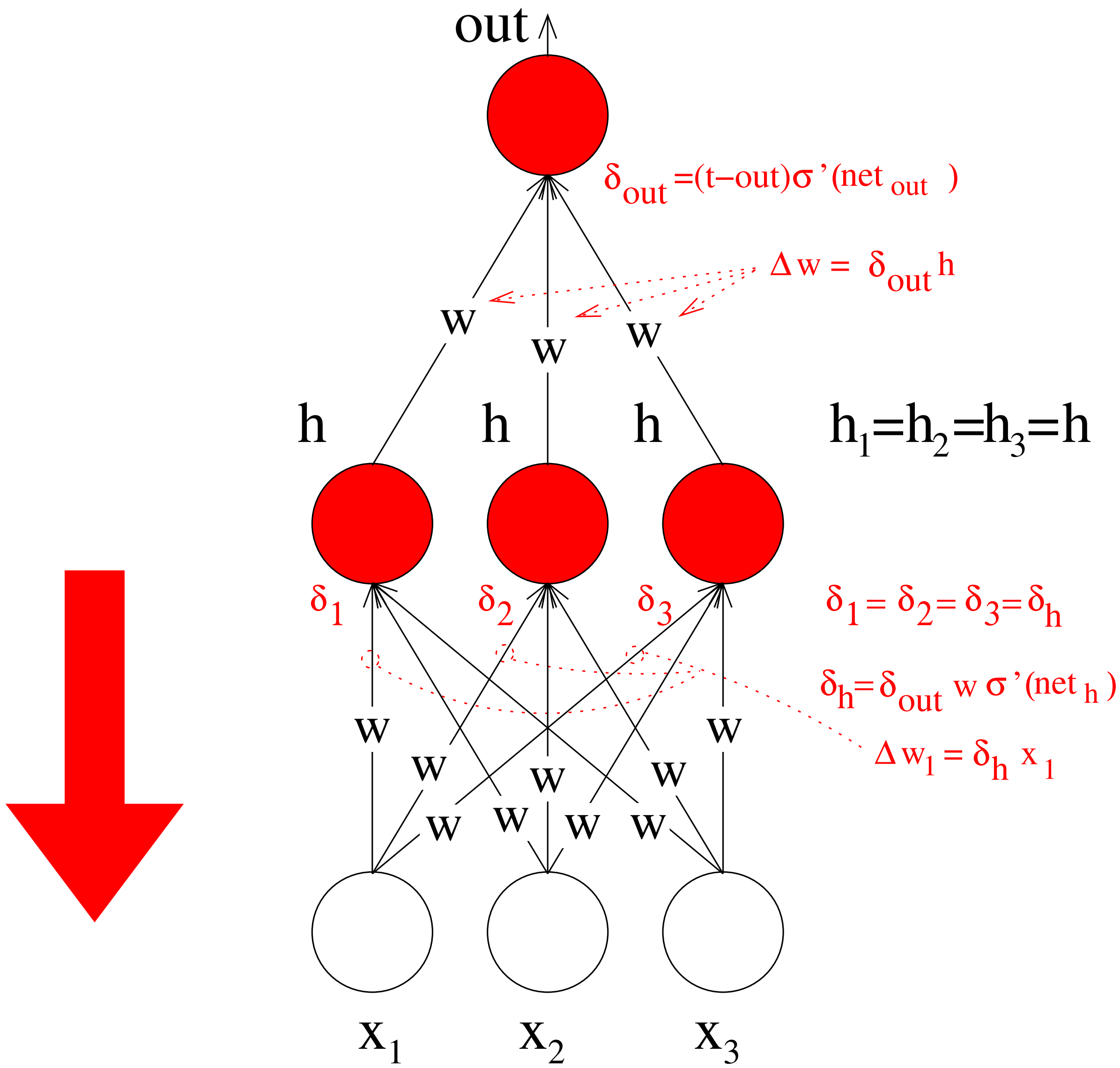

out
δout =(t−out)σ'(net out)
Δw = δout h
w
w
w
h
h
h
h1=h2=h3=h
w
w
w
w
w
w
w
w
w
x1
x2
x3

Symmetries

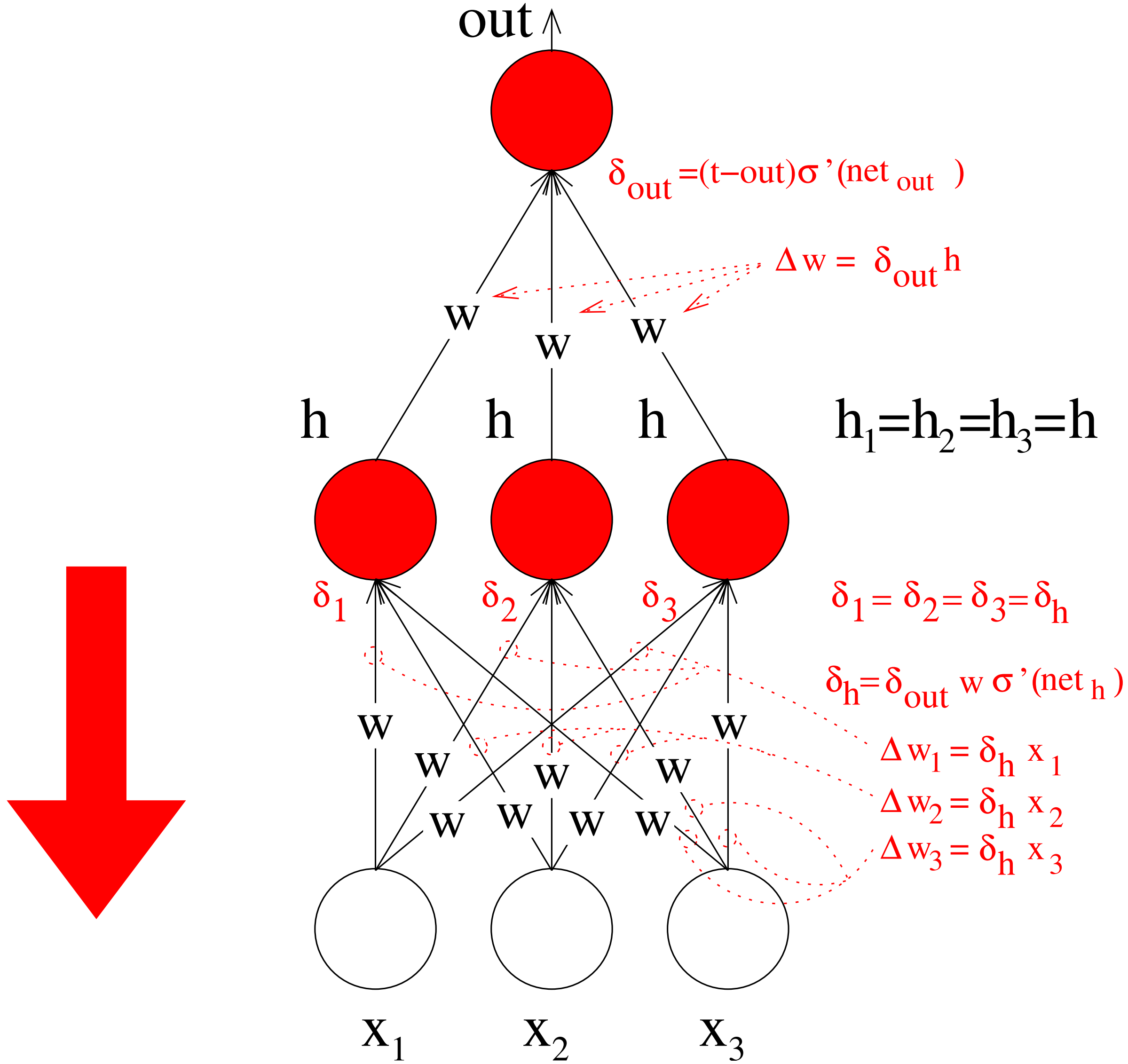


out
δ_{out} =(t−out)σ ’(net out)
Δ w = δ_{out} h
w
w
w
h
h
h
h_1=h_2=h_3=h
δ_1
δ_2
δ_3
δ_1= δ_2= δ_3=δ_h
δ_h=δ_{out} w σ ’(net h)
Δ w_1 = δ_h x 1
x_1
x_2
x_3

Symmetries


out
δ_{out} =(t−out)σ'(net $_{out}$)
Δw = δ_{out} h
w
w
w
h
h
h
h_1=h_2=h_3=h
δ_1
δ_2
δ_3
δ_1= δ_2= δ_3=δ_h
δ_h=δ_{out} w σ'(net $_h$)
Δw_1 = δ_h x_1
x_1
x_2
x_3

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
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out}\, w\, \sigma'(net_h)$
$\Delta w_1 = \delta_h\, x_1$
$\Delta w_2 = \delta_h\, x_2$
$\Delta w_3 = \delta_h\, x_3$
x_1
x_2
x_3

Speeding up learning with momentum

The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction

Gradient Descent

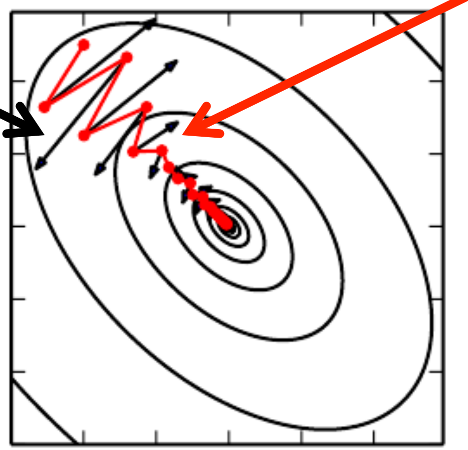
$$\vec{w} = \vec{w} + \Delta\vec{w}$$

$$\Delta\vec{w} = -\eta\nabla E[\vec{w}]$$

Gradient Descent with momentum

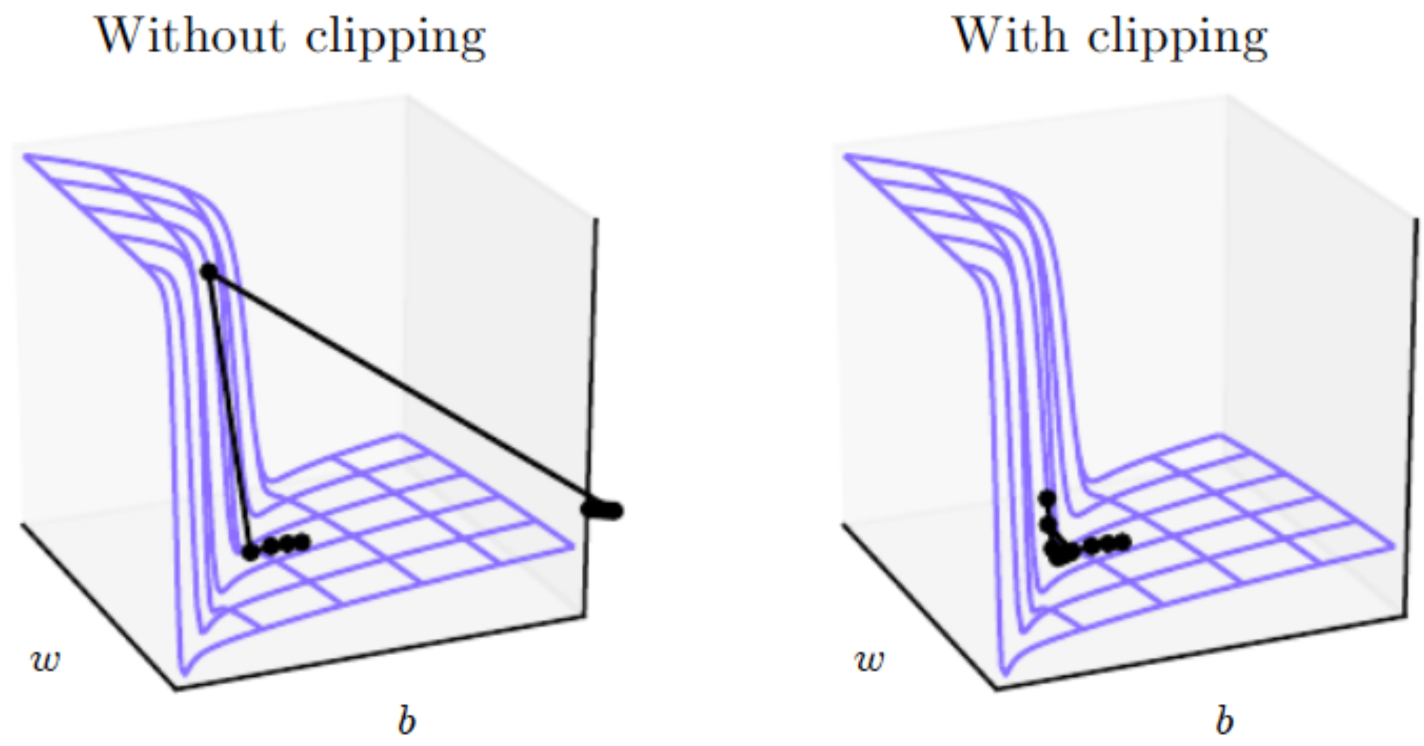
$$\vec{w} = \vec{w} + \vec{v}$$

$$\vec{v} = \alpha\vec{v} - \eta\nabla E[\vec{w}]$$



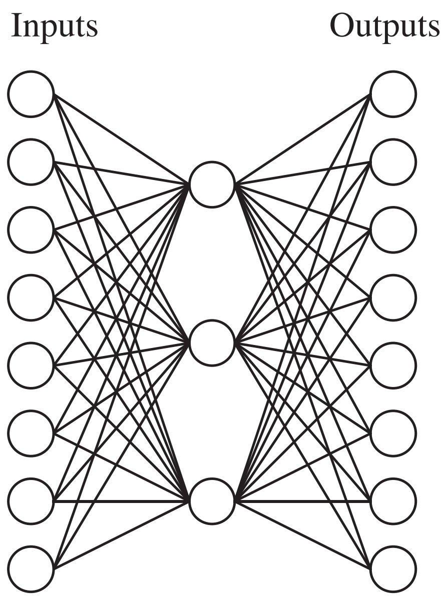
Clipping Gradients: avoiding overshooting

$$\text{if } \|\nabla_{\mathbf{W}} \overbrace{E(\mathbf{W})}^{Loss}\| > \overset{threshold}{\overset{\downarrow}{v}} \Rightarrow \nabla_{\mathbf{W}} E(\mathbf{W}) \leftarrow v \frac{\nabla_{\mathbf{W}} E(\mathbf{W})}{\|\nabla_{\mathbf{W}} E(\mathbf{W})\|}$$



Special Case of Feed-forward Network: Autoencoder

Learning the identity function:
(lossy) compression of data

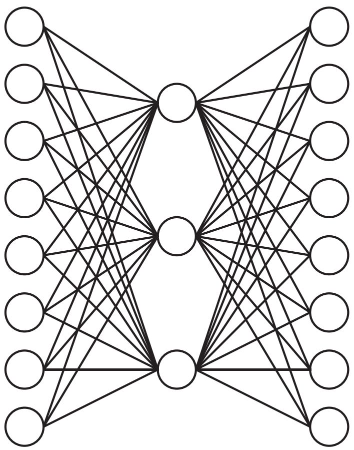


Supervised learning for..
Unsupervised learning!!

Input	Output
00000001	00000001
00000010	00000010
00000100	00000100
00001000	00001000
00010000	00010000
00100000	00100000
01000000	01000000
10000000	10000000

Special Case of Feed-forward Network: Autoencoder

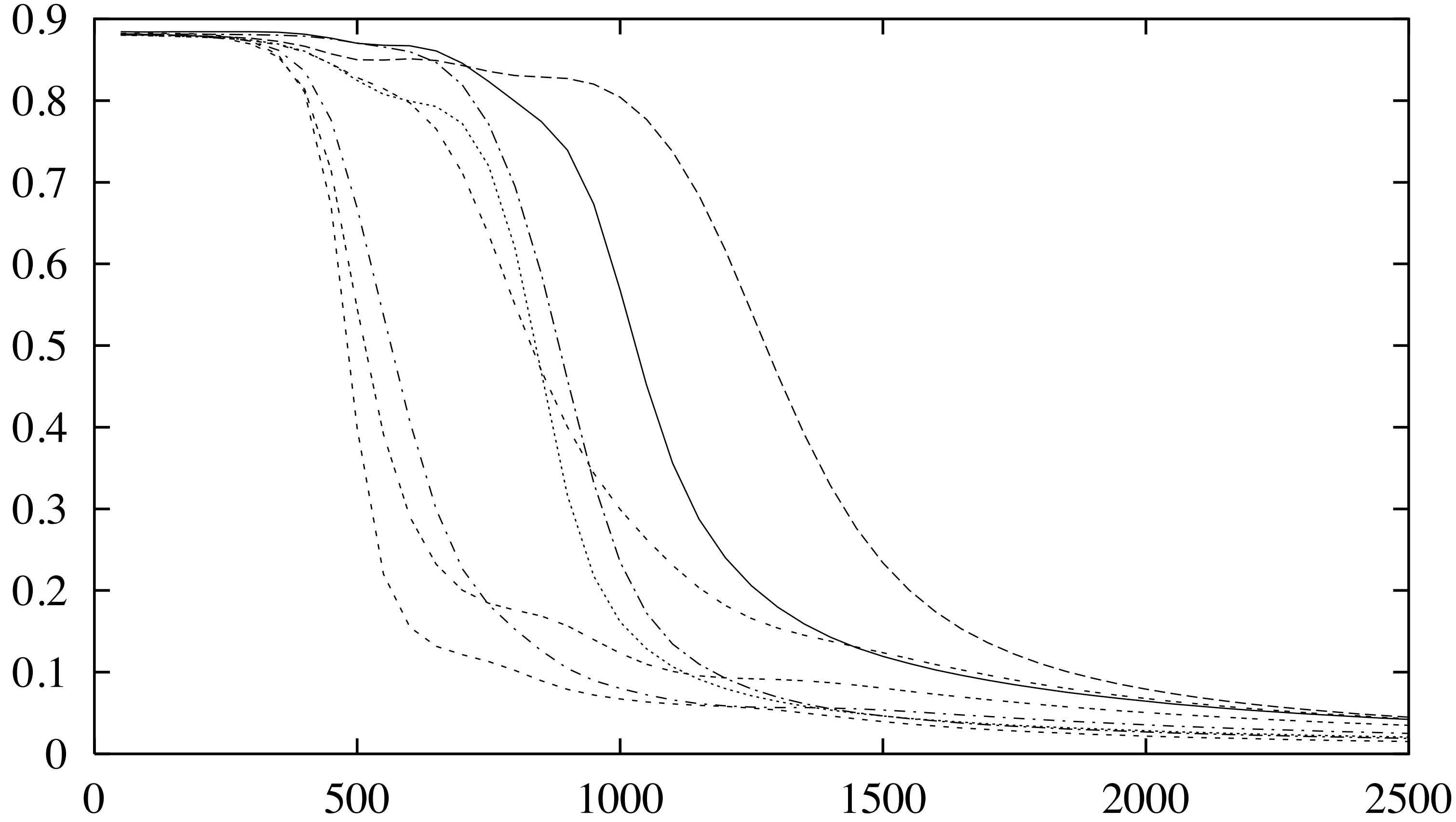
After learning…



Input		Hidden values		Output
10000000	$\rightarrow$	0.89 0.04 0.08	$\rightarrow$	10000000
01000000	$\rightarrow$	0.01 0.11 0.88	$\rightarrow$	01000000
00100000	$\rightarrow$	0.01 0.97 0.27	$\rightarrow$	00100000
00010000	$\rightarrow$	0.99 0.97 0.71	$\rightarrow$	00010000
00001000	$\rightarrow$	0.03 0.05 0.02	$\rightarrow$	00001000
00000100	$\rightarrow$	0.22 0.99 0.99	$\rightarrow$	00000100
00000010	$\rightarrow$	0.80 0.01 0.98	$\rightarrow$	00000010
00000001	$\rightarrow$	0.60 0.94 0.01	$\rightarrow$	00000001

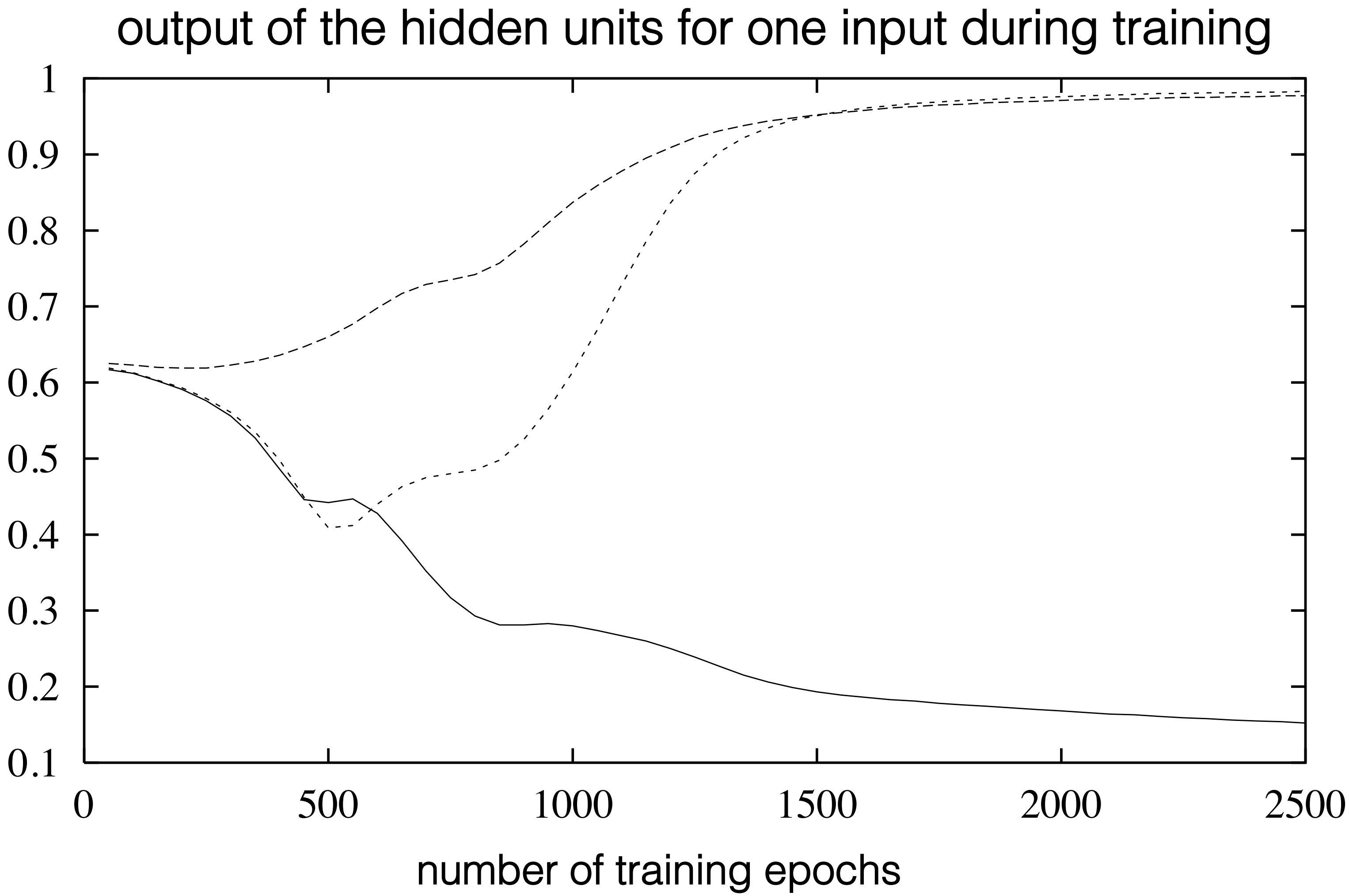
Special Case of Feed-forward Network: Autoencoder

error curves for each output unit during training



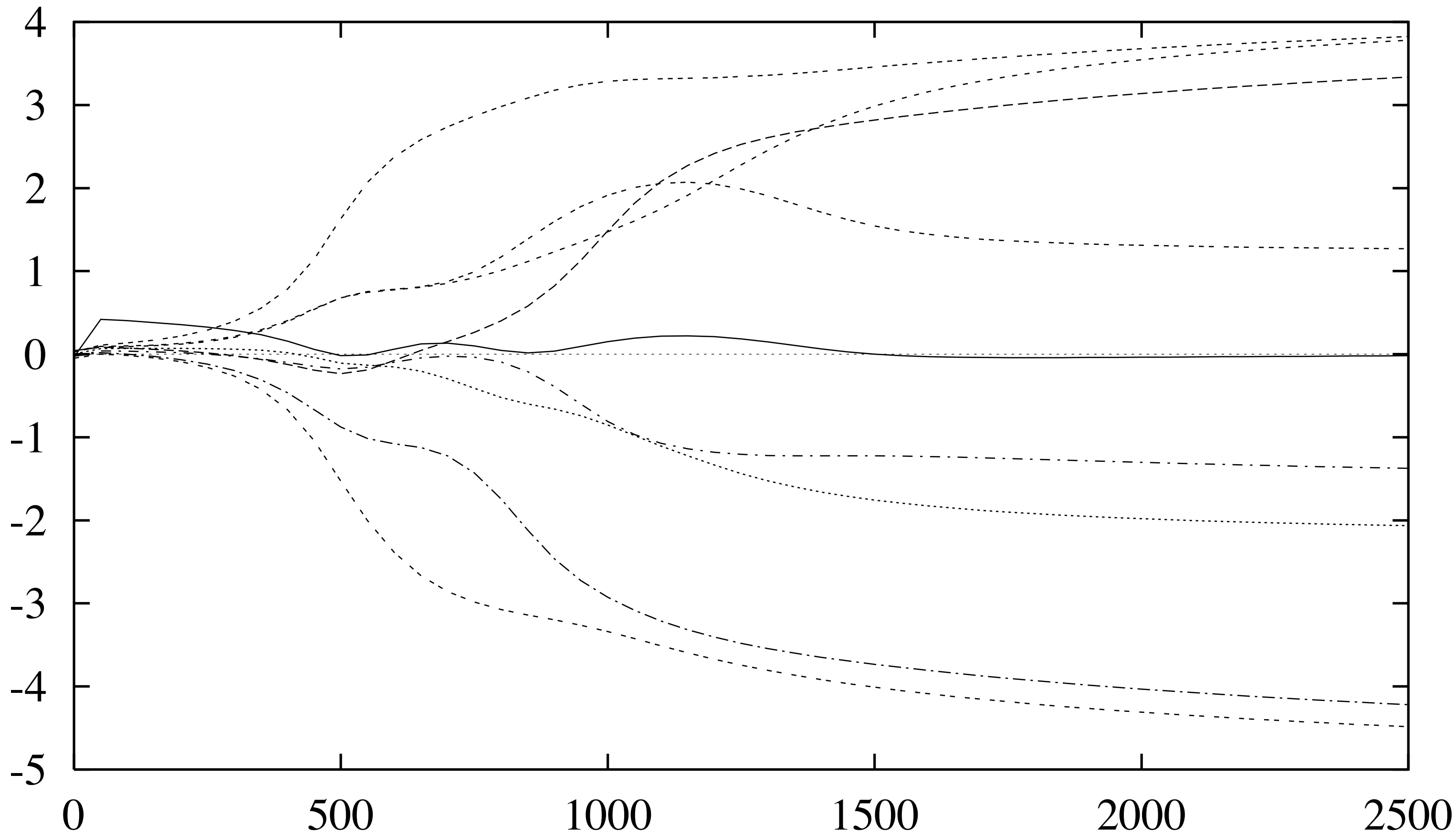
number of training epochs

Special Case of Feed-forward Network: Autoencoder



Special Case of Feed-forward Network: Autoencoder

evolution of weights of one hidden unit during training



number of training epochs

Special Case of Feed-forward Network: Autoencoder