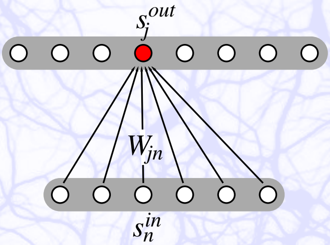
# The Multi-Layer Perceptron (MLP)
## *a short introduction*

Cornelius Weber
Knowledge Technology Research Group
Informatik, Universität Hamburg
Email: weber@informatik.uni-hamburg.de

October 7, 2014

# 1-Layer Perceptron = a layer of neurons




activate a neuron:

$$h_j^{out} \;=\; \sum_n w_{jn} s_n^{in} \;=\; \vec{w}_j \cdot \vec{s}^{in}$$

$\rightarrow$ dot product (scalar product)
between weight vector and input vector

activate all neurons:

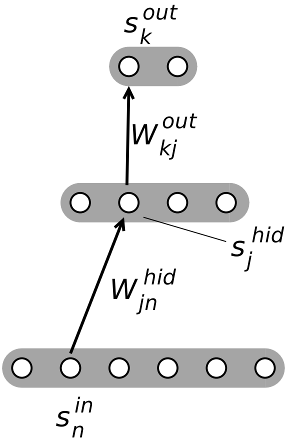
$$\vec{h}^{out} \;=\; W \vec{s}^{in}$$

$\rightarrow$ matrix product with weight matrix

in Python: `h_out = numpy.dot(W,s_in)`
in C: two nested `for`-loops
(outer loop over output neurons, inner loop does scalar product)

transfer function applied, e.g. $s_j^{out} = \tanh(h_j^{out})$

# Feedforward Activation of the MLP




inner activation of neuron $i$ in layer $\lambda$:

$$h_i^{\lambda}(\vec{w}_i^{\lambda\,\lambda-1}) = \sum_j w_{ij}^{\lambda\,\lambda-1} s_j^{\lambda-1} + \underbrace{b_i^{\lambda}}_{\text{bias}}$$

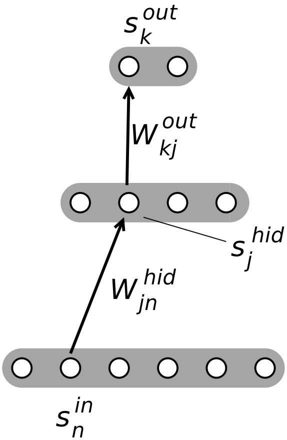
neuron output:

$$s_i^{\lambda}(h_i) = \varphi_i^{\lambda}(h_i^{\lambda}) \quad \stackrel{e.g.}{=} \tanh(h_i^{\lambda})$$

in vector notation:

$$\vec{s}^{\lambda} = \varphi^{\lambda}(\underbrace{W^{\lambda\,\lambda-1}\vec{s}^{\lambda-1}}_{\vec{h}^{\lambda-1}})$$

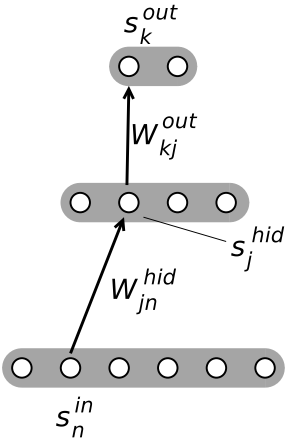
# Feedforward Activation of the MLP




$\Leftrightarrow$

$$\vec{s}^{out} = \varphi^{out}(W^{out}\,\underbrace{\varphi^{hid}(W^{hid}\vec{s}^{in})}_{\vec{s}^{hid}})$$

# Feedforward Activation of the MLP




inner activation of neuron $i$ in layer $\lambda$:

$$h_i^\lambda(\vec{w}_i^{\lambda\,\lambda-1}) = \sum_j w_{ij}^{\lambda\,\lambda-1} s_j^{\lambda-1} + \underbrace{b_i^\lambda}_{\text{bias}}$$

neuron output:

$$s_i^\lambda(h_i) = \varphi_i^\lambda(h_i^\lambda) \quad \stackrel{e.g.}{=} \tanh(h_i^\lambda)$$

local error on output layer:

$$e_i(s_i) = s_i^{\text{Teach}} - s_i^{out}$$

total error (cost function):

$$E(\vec{e}) = \frac{1}{2}\sum^{data}\sum_i e_i^2$$

Dependencies: $E(e(s(h(w))))$. Chain rule $\Rightarrow$ derivatives for training

# Possible Transfer Functions

- linear transfer function — typically used for output units

$$\varphi_i = h_i = \sum_j w_{ij} s_j^{in} + b_i \qquad \varphi_i' = \frac{\partial \varphi_i}{\partial h_i} = 1$$

- logistic/sigmoid function — similar shape as tanh but in $]0,1[$

$$\varphi_i = \frac{1}{1+e^{-h_i}} = \frac{1}{1+e^{-\sum_j w_{ij} s_j^{in} + b_i}} \qquad \varphi_i' = \varphi_i \cdot (1 - \varphi_i)$$

- hyperbolic tangent

$$\varphi_i = \tanh(h_i) \qquad \varphi_i' = 1 - (\varphi_i)^2$$

- Radial Basis Functions — only used for $\vec{s}^{in}$ on input layer

$$\varphi_i = e^{-\frac{\sum_j (s_j^{in} - w_{ij})^2}{2\sigma_i^2}}$$

- max-like operation (Riesenhuber & Poggio's "HMAX model")

$$\varphi_i = \max_j w_{ij} s_j^{in}$$

*not differentiable*

# Non-Local "Layer" Transfer Functions

let $h_i = \sum_j w_{ij} s_j^{in} + b_i$

- softmax (inverse temperature $\beta$)

$$\varphi_i \;=\; \frac{e^{\beta h_i}}{\sum_k e^{\beta h_k}}$$

- winner-take-all (not differentiable)

$$\varphi_{i^*} \;=\; \begin{cases} 1, & h_{i^*} > h_k \; \forall k \neq i^* \\ 0, & \text{else} \end{cases}$$

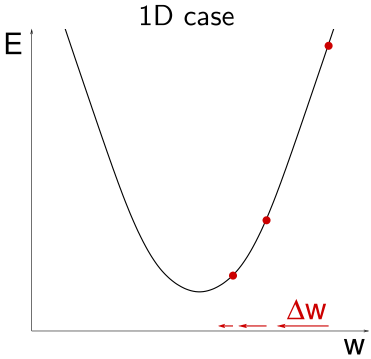
- competitive topographic (given $i^*$ is winning node)

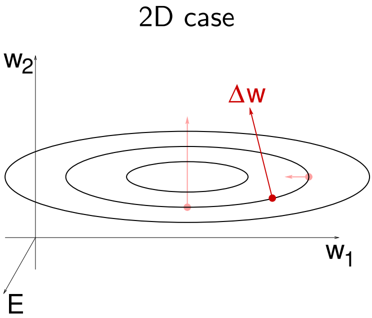
$$\varphi_i \;=\; e^{-\frac{(i-i^*)^2}{\sigma^2}}$$

note: non-local transfer functions lead to non-local update rules from backpropagation

# Gradient Descent

$$\Delta w = -\frac{\partial E}{\partial w}$$




for quadratic functions, update size scales very well

update size in one direction does not match update size in another

# Natural Gradient

$$\Delta w = -\, G^{-1} \frac{\partial E}{\partial w}$$

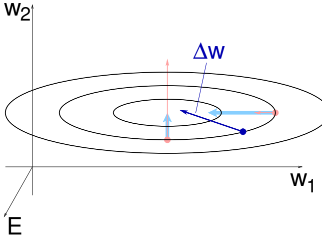
Compensates for "distorted" coordinates. For affine distortions:

$$G = J^T J \;=\; \begin{pmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \dots \\ \frac{\partial^2 E}{\partial w_1 \partial w_2} & \frac{\partial^2 E}{\partial w_2^2} & \dots \\ \dots & \dots & \dots \end{pmatrix} = H \text{ (Hessian)}$$

$$J = \left( \frac{\partial E}{\partial w_1}, \;\; \frac{\partial E}{\partial w_2}, \;\; \dots \right) = \text{Jacobian (here, a vector)}$$

Approximation, neglecting off-diagonals:

$$G^{-1} \;\approx\; \begin{pmatrix} \frac{1}{\frac{\partial^2 E}{\partial w_1^2}} & 0 \\ 0 & \frac{1}{\frac{\partial^2 E}{\partial w_2^2}} \end{pmatrix}$$




Amari Douglas. Why natural gradient? 1998.

# Error Backpropagation: Recursive $\delta$

last layer (using index $i$):

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial e_i} \frac{\partial e_i}{\partial s_i} \frac{\partial s_i}{\partial h_i} \frac{\partial h_i}{\partial w_{ij}}$$

$$= \underbrace{e_i \quad -1 \quad \varphi'(h_i)}_{\delta_i} \quad s_j$$

$2^{nd}$-last layer (using index $j$):

$$\frac{\partial E}{\partial w_{jk}} = \underbrace{\overbrace{\sum_i \underbrace{\underbrace{\frac{\partial E}{\partial e_i}}_{e_i} \cdot \overbrace{\frac{\partial e_i}{\partial h_i}}^{-\varphi_i'(h_i)}}_{\delta_i \text{ (compare with above)}} \overbrace{\frac{\partial h_i}{\partial s_j}}^{w_{ij}}}^{\frac{\partial E}{\partial s_j}} \overbrace{\overbrace{\frac{\partial s_j}{\partial h_j}}^{\varphi_j'(h_j)} \overbrace{\frac{\partial h_j}{\partial w_{jk}}}^{s_k}}^{\frac{\partial s_j}{\partial w_{jk}}}}_{\delta_j \text{ (defined as } \delta_j = \varphi_j' \sum_i \delta_i w_{ij})}$$

where $\cdot \frac{\partial e_i}{\partial s_j} = \frac{\partial e_i}{\partial h_i} \frac{\partial h_i}{\partial s_j}$

# One MLP Learning Step

Given input activities $s_j^0$ on layer $\lambda = 0$.

Forward propagation of activities to hidden- and output layers:

$$s_i^\lambda \; = \; \varphi(\sum_j w_{ij}^{\lambda\,\lambda-1}\, s_j^{\lambda-1} + b_i^\lambda)$$

Error on output layer: *(see next 5 slides)*

$$\delta_i^{\text{out}} \; = \; \varphi_i'(h_i^{\text{out}}) \cdot (s_i^{\text{Teach}} - s_i^{\text{out}}) \overset{usually}{=} s_i^{\text{Teach}} - s_i^{\text{out}}$$

Error backpropagation to all hidden layers:

$$\delta_j^\lambda \; = \; \varphi_j'(h_j^\lambda) \cdot \sum_i \delta_i^{\lambda+1} w_{ij}^{\lambda+1\,\lambda}$$

Learning rule, with small learning rate $\epsilon$:

$$\left.\begin{array}{rcccl} \Delta w_{ij}^{\lambda\,\lambda-1} & = & -\epsilon \frac{\partial E}{\partial w_{ij}^{\lambda\,\lambda-1}} & = & \epsilon \delta_i^\lambda s_j^{\lambda-1} \\ \Delta b_i^\lambda & = & -\epsilon \frac{\partial E}{\partial b_i^\lambda} & = & \epsilon \delta_i^\lambda \end{array}\right\}$$

Many learning steps with different data samples must be repeated ...

# Interpretation of the Squared Error

likelihood of the data $s^{\text{Teach}}$ being generated
given input $s^{in}$ and model parameters $W$

$$P(s^{\text{Teach}}|W, s^{in}) \approx e^{-\frac{1}{2}(s^{\text{Teach}} - s^{out})^2} = e^{-E}$$

assumption: Gaussian noise on output

- $E = -\ln P$
- $P^a\, P^b = e^{-E^a}\, e^{-E^b} = e^{-E^a - E^b}$
  multiplying probabilities $\Leftrightarrow$ adding cost terms

note: Gaussian noise unbounded $\Rightarrow$ unbounded output range assumed $\Rightarrow$ linear transfer function on output units appropriate, but not sigmoid function

# Derivative for Square Error & Linear Function

given: $s^{out} := \varphi(wx) = wx$ linear transfer function

$$\begin{aligned}
-\frac{\partial E}{\partial w} &= -\frac{\partial}{\partial w}\left(\frac{1}{2}(s^{\text{Teach}} - s^{out})^2\right) \\
&= (s^{\text{Teach}} - s^{out})\frac{\partial s^{out}}{\partial w} \\
&= (s^{\text{Teach}} - s^{out})\, x
\end{aligned}$$

$\rightarrow$ squared error & linear function $\Rightarrow$ simple learning rule

# Cross-Entropy Error

outputs considered as class probabilities:

$$P(s^{\text{Teach}}|W, s^{in}) = \begin{cases} s^{out}, & s^{\text{Teach}} = 1 \quad \text{it's this class} \\ 1 - s^{out}, & s^{\text{Teach}} = 0 \quad \text{not this class} \end{cases}$$
$$= (s^{out})^{s^{\text{Teach}}} \cdot (1 - s^{out})^{(1 - s^{\text{Teach}})}$$

cross-entropy error:

$$E = -\ln P$$
$$= -s^{\text{Teach}} \ln s^{out} - (1 - s^{\text{Teach}}) \ln(1 - s^{out})$$

note: for 2 classes, sigmoid/logistic transfer function is natural

www.willamette.edu/∼gorr/classes/cs449/classify.html

# Derivative for Cross Entropy & Logistic Function

given: $s^{out} := \varphi(wx) = \frac{1}{1+e^{-wx}}$ logistic transfer function

for the logistic function it is: $\frac{\partial s^{out}}{\partial w} = x\, s^{out}\,(1 - s^{out})$

$$
\begin{aligned}
-\frac{\partial E}{\partial w} &= -\frac{\partial}{\partial w}\Big( - s^{\text{Teach}} \ln s^{out} \; - \; (1 - s^{\text{Teach}}) \ln(1 - s^{out}) \Big) \\
&= s^{\text{Teach}} \frac{1}{s^{out}} \frac{\partial s^{out}}{\partial w} \; - \; (1 - s^{\text{Teach}}) \frac{1}{1 - s^{out}} \frac{\partial s^{out}}{\partial w} \\
&= s^{\text{Teach}} \frac{1}{s^{out}} x\, s^{out}\,(1 - s^{out}) \; - \; (1 - s^{\text{Teach}}) \frac{1}{1 - s^{out}} x\, s^{out}\,(1 - s^{out}) \\
&= s^{\text{Teach}} x\,(1 - s^{out}) \; - \; (1 - s^{\text{Teach}}) x\, s^{out} \\
&= \begin{cases} s^{\text{Teach}} x\,(1 - s^{out}), & s^{\text{Teach}} = 1 \\ -(1 - s^{\text{Teach}}) x\, s^{out}, & s^{\text{Teach}} = 0 \end{cases} \\
&= x\,(s^{\text{Teach}} - s^{out})
\end{aligned}
$$

$\rightarrow$ cross-entropy & logistic function $\Rightarrow$ simple learning rule

# Derivative for Cross Entropy & Softmax Function

given: $s_k^{out} := \varphi(\{w_{k'}\}, x) = \frac{e^{w_k x}}{\sum_{k'} e^{w_{k'} x}}$ softmax transfer function

useful for 1-of-K classification; output units' activations dependent

for softmax it is: $\frac{\partial s^{out}}{\partial w_k} = x\, s_k^{out}\,(1 - s_k^{out})$ (as for logistic)

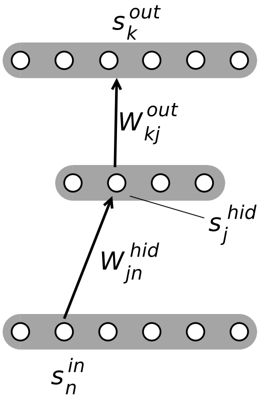
$$
\begin{aligned}
-\frac{\partial E}{\partial w_k} &= -\frac{\partial}{\partial w_k}\Big(-\sum_k s_k^{\text{Teach}} \ln s_k^{out}\Big) \\
&= \ldots \\
&= x\,(s_k^{\text{Teach}} - s_k^{out})
\end{aligned}
$$

$\rightarrow$ cross-entropy & softmax function $\Rightarrow$ simple learning rule

note, for 2 classes ($K = 2$), softmax becomes logistic function:

$$
\begin{aligned}
s_1^{out} &= \frac{e^{w_1 x}}{e^{w_1 x} + e^{w_2 x}} = \frac{1}{1 + e^h} \qquad \text{with} \qquad h = (w_2 - w_1)x \\
s_2^{out} &= \frac{e^{w_2 x}}{e^{w_1 x} + e^{w_2 x}} = \frac{1}{1 + e^{-h}}
\end{aligned}
$$

# Error Backpropagation – Comments



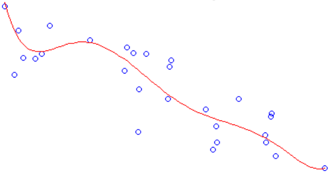

- non-linearity on middle layer important
- more than linear separation – complex transformations possible
- 3-layer network can in theory represent any input-output function
- more layers possible $\rightarrow$ "deep learning" (requires tricks, e.g. initial unsupervised learning, convolutional kernel, max-pooling, ...)
- backpropagation considered unbiological
- overfitting possible


Krizhevsky Sutskever Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012

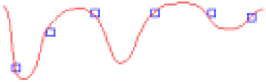
Ciresan Meier .. Schmidhuber. Multi-Column Deep Neural Network for Traffic Sign Classification. Neur Netw 2013

# Early Stopping

overfitting?



overfitting!



- use network with less parameters
- stop when test error increases