Multi-layer Perceptrons

Lecture 6, COMP / ELEC / STAT 502
© E. Merényi

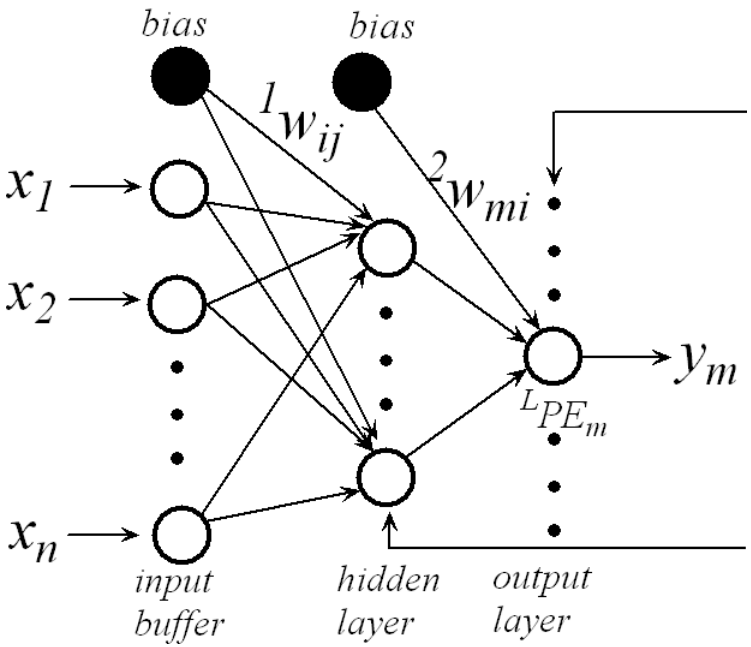
Inception: February 2, 2012 Last updated: February 2, 2017[1]

[1] Sources:
Colin Fyfe, Artificial Neural Networks and Informaton Theory. Course notes, Dept. of Computing and Information Systems, The University of Paisley, 2000
Frederick Ham and Ivica Kostanic: Principles of Neurocomputing for Science & Engineering. McGraw-Hill, 2001
Simon Haykin: Neural Networks. A Comprehensive Foundation. McMillan, New Jersey, 1999

Outline

Preliminaries

We saw earlier that simple perceptrons (single-layer perceptrons) can only solve simple (linearly separable, two-class) problems. It was suspected as early as in the 60's that multilayer perpceptrons (MLPs) should be able to solve more complex tasks — but it was not known how to train an MLP.



error here easily calculated:

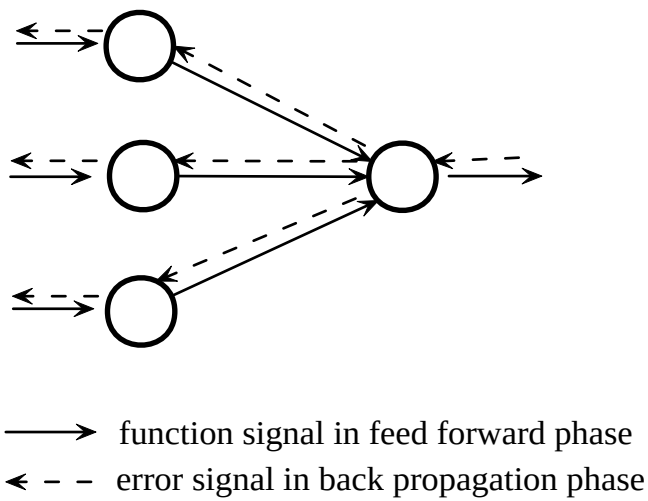
$$\delta_m = (D_m - y_m) f'(NET_m)$$
$$^L\Delta w_{mi} = \gamma\, ^L\delta_m\, ^{L-1}y_i$$
$$= \gamma^2 \delta_m\, ^1y_i \quad (for\ \ L = 2)$$

credit assignement needed to distribute the error

for $l < L$ we have no $^l\delta$:

no ^lD to compute $^l\delta = (^lD - ^ly) f'(^lNET)$

Key Idea of Back Propagation



After the feed forward phase, propagate the error back through the present weights and calculate the (estimate of the) local gradient of the error. I.e., distribute the total error obtained at the output, in proportion to each PE's contribution to the error.

History and Terms

Paul Werbos derived the Back Propagation learning rule in 1974 — a landmark in ANN history, but not popularized until 1986 (in the book "Parallel Distributed Processing" by Jay McClelland & David Rummelhart).
(*http* : *//hincapie.psych.purdue.edu/PDP_Primer/index.html*).
The BP learning rule put to rest the concerns raised by Minsky & Papert in 1969.
Historically, the BP is the first serious ANN.

Other names for Back Propagation:

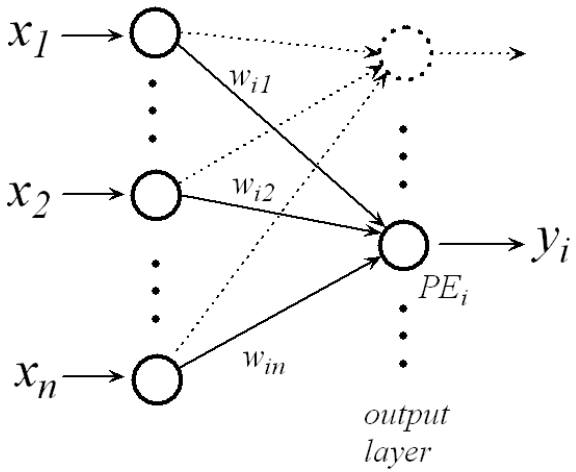
- Backpropagation
- Error backpropagation
- backprop
- BP

Reminder

The Delta rule for weight update in the simple perceptron, for multiple output PEs:

$$\Delta w_{ij} = -\gamma \frac{\partial E^k}{\partial w_{ij}} = -\gamma \frac{\partial E^k}{\partial y_i^k} \frac{\partial y_i^k}{\partial NET_i^k} \frac{\partial NET_i^k}{\partial w_{ij}}$$

$$= -\gamma(-1) \underbrace{(D_i^k - f(\sum_{r=0}^{N} w_{ir} x_r^k)) f'(\sum_{r=0}^{N} w_{ir} x_r^k)}_{\delta_i^k} x_j^k = \gamma \delta_i^k x_j^k$$



$y_i = f(\sum_{j=0}^{N} w_{ij} x_j)$

In a Perceptron with multiple outputs the Delta rule is applied to each output PE, separately.

In figure, bias neuron is not shown (N = n+1).

For Multilayer Perceptron (MLP) we derive the *Generalized Delta Rule*, or BP rule.

Note: CF gives a good recipe and a concrete example for a 2-layer perceptron with BP rule. Go through it!

Definitions, Notation

L	index of the output layer
${}^{l}NPE$	number of PEs in layer $l, 0 \leq l \leq L$
${}^{L}e_i^k = D_i^k - {}^{L}y_i^k$	*error signal* at ${}^{L}PE_i$ for input pattern k
$\frac{1}{2}({}^{L}e_i^k)^2$	*instantaneous error* at ${}^{L}PE_i$ for pattern k
${}^{L}E^k = \frac{1}{2}\sum_{i=1}^{{}^{L}NPE}({}^{L}e_i^k)^2$	*total error* from all PEs in layer L, for pattern k.
$E = \frac{1}{2}\sum_{k=1}^{P} {}^{L}E^k$	total error over all training patterns at layer L
${}^{l}y_i^k = f({}^{l}NET_i^k)$	output of PE_i in layer l in response to pattern k
${}^{l}NET_i^k = \sum_{j=0}^{{}^{l-1}NPE}({}^{l}w_{ij}{}^{l-1}y_j^k)$	net input of ${}^{l}PE_i, 1 \leq l \leq L \quad ({}^{0}y_j^k = x_j^k)$

If an error expression has no layer index L (the output layer) is assumed.
Beware of potential confusion between an upper left index and upper right index of the preceding variable!

The Delta Rule - Slightly Transformed

Since the error in a simple perceptron (with multiple output PEs), in response to $\mathbf{x}^k$, is

$$E^k = \frac{1}{2}\sum_{i=1}^{L_{NPE}} \underbrace{(D_i^k - y_i^k)}_{e_i^k}{}^2 = \frac{1}{2}\sum_{i=1}^{L_{NPE}} (D_i^k - f(\sum_{r=0}^{N} w_{ir} x_r^k))^2$$

the error gradient in the Delta rule for weight update

$$\frac{\partial E^k}{\partial w_{ij}} = \frac{\partial E^k}{\partial y_i^k}\frac{\partial y_i^k}{\partial NET_i^k}\frac{\partial NET_i^k}{\partial w_{ij}} = (-1)\underbrace{(D_i^k - f(\sum_{r=0}^{N} w_{ir} x_r^k)) f'(\sum_{r=0}^{N} w_{ir} x_r^k)}_{\delta_i^k = e_i^k f'(NET_i^k)} x_j^k = -\delta_i^k x_j^k$$

can also be written as

$$\frac{\partial E^k}{\partial w_{ij}} = \underbrace{\frac{\partial E^k}{\partial e_i^k}}_{e_i^k}\underbrace{\frac{\partial e_i^k}{\partial y_i^k}}_{-1}\underbrace{\frac{\partial y_i^k}{\partial NET_i^k}}_{f'(NET_i^k)}\underbrace{\frac{\partial NET_i^k}{\partial w_{ij}}}_{x_j^k}$$

Derivation Of the BP Rule

For a MLP, let's examine the local gradient of the error wrt w_{ij} similarly, but *now at a general location, in layer l:*

$$\frac{\partial E^k}{\partial {}^l w_{ij}} = \underbrace{\frac{\partial E^k}{\partial {}^l e_i^k}}_{{}^l e_i^k} \underbrace{\frac{\partial {}^l e_i^k}{\partial {}^l y_i^k}}_{-1} \underbrace{\frac{\partial {}^l y_i^k}{\partial {}^l NET_i^k}}_{f'({}^l NET_i^k)} \underbrace{\frac{\partial {}^l NET_i^k}{\partial {}^l w_{ij}}}_{{}^{l-1} y_j^k} \tag{6.1}$$

$$\frac{\partial E^k}{\partial {}^l w_{ij}} = - \underbrace{{}^l e_i^k \; f'({}^l NET_i^k)}_{{}^l \delta_i^k} \; {}^{l-1} y_j^k \tag{6.2}$$

The correction to ${}^l w_{ij}$ will be (once we know ${}^l \delta_i^k$, unknown at this point)

$${}^l \Delta w_{ij} = {}^l w_{ij}(t+1) - {}^l w_{ij}(t) = -\gamma \frac{\partial E^k}{\partial {}^l w_{ij}} = \gamma \; {}^l \delta_i^k \; {}^{l-1} y_j^k \tag{6.3}$$

How do we compute ${}^l \delta_i^k$?

Derivation Of the BP Rule

$$^{l}\delta_i^k = {}^{l}e_i^k \;\; f'({}^{l}NET_i^k) = ??$$

For the output layer, $l = L$,

$$^{L}\delta_i^k = \underbrace{(D_i^k - {}^{L}y_i^k)}_{{}^{L}e_i^k,\ \text{known}} \;\; \underbrace{f'({}^{L}NET_i^k)}_{\text{known}} \tag{6.4}$$

For hidden layers $1 \leq l < L$, $f'({}^{l}NET_i^k)$ is known but the error signal ${}^{l}e_i^k$ is unknown ($\because {}^{l}D_i^k$ is unknown). ${}^{l}\delta_i^k$ needs to be determined recursively. Since ${}^{l}e_i^k = -\frac{\partial E^k}{\partial {}^{l}y_i^k}$ from eq. (6.1)

$$^{l}\delta_i^k = {}^{l}e_i^k \;\; f'({}^{l}NET_i^k) = -\frac{\partial E^k}{\partial {}^{l}y_i^k} \;\; f'({}^{l}NET_i^k) \tag{6.5}$$

Can we compute $\frac{\partial E^k}{\partial {}^{l}y_i^k}$ ... recursively?

Derivation Of the BP Rule

Repeating eq.(6.5), for $l < L$:

$$ {}^{l}\delta_i^k = {}^{l}e_i^k \; f'({}^{l}NET_i^k) \qquad = -\underbrace{\frac{\partial E^k}{\partial {}^{l}y_i^k}}_{?} \; f'({}^{l}NET_i^k) $$

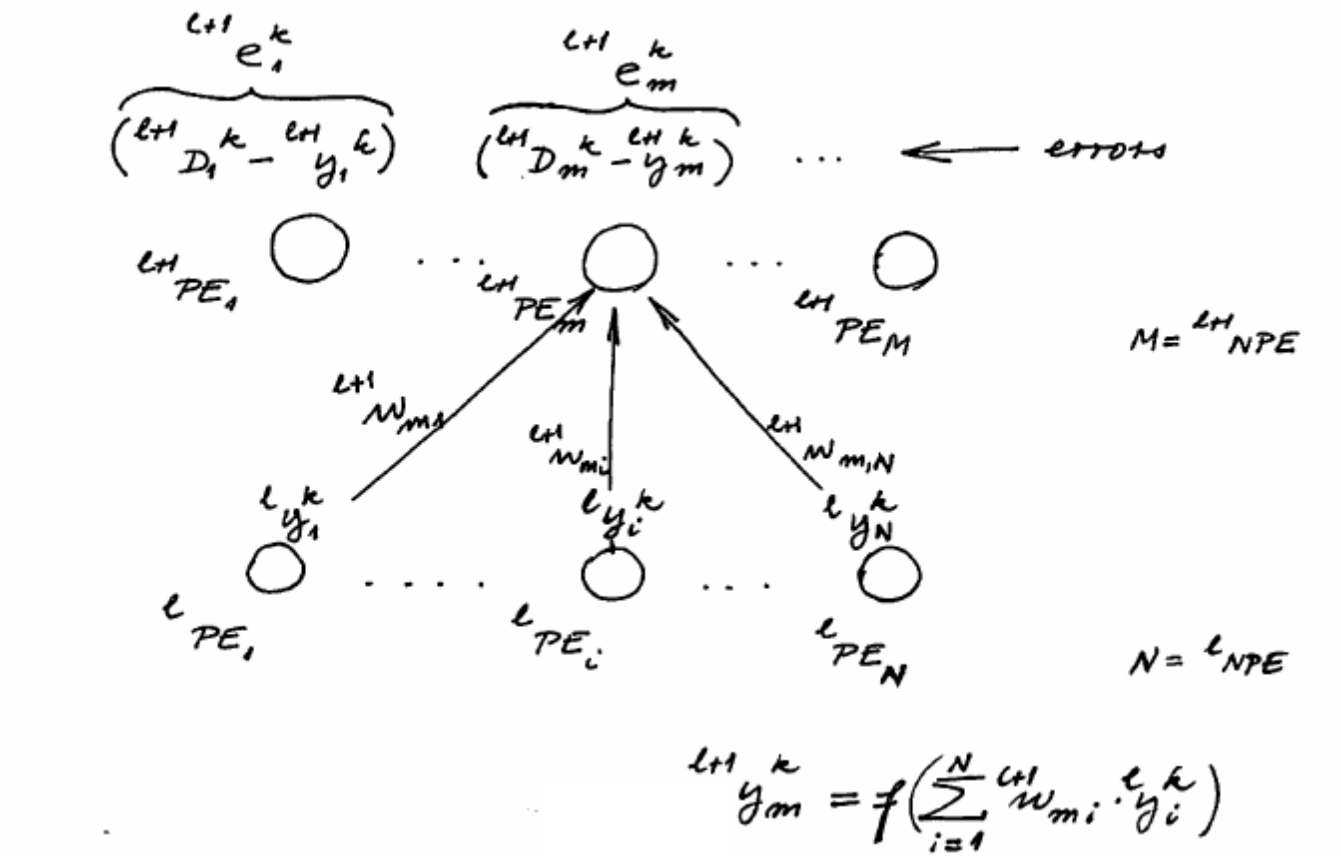
Since ${}^{l}y_i^k, i = 1, \ldots, {}^{l}NPE$, all feed into ${}^{l+1}y_m^k$, through

$$ {}^{l+1}y_m^k = f\left(\sum_{i=0}^{{}^{l}NPE} {}^{l+1}w_{mi} \, {}^{l}y_i^k \right) \quad \text{i.e., } {}^{l+1}y_m^k \text{ depend on } {}^{l}y_i^k \tag{6.6} $$

the local gradient can be expressed as (a total derivative of E^k)

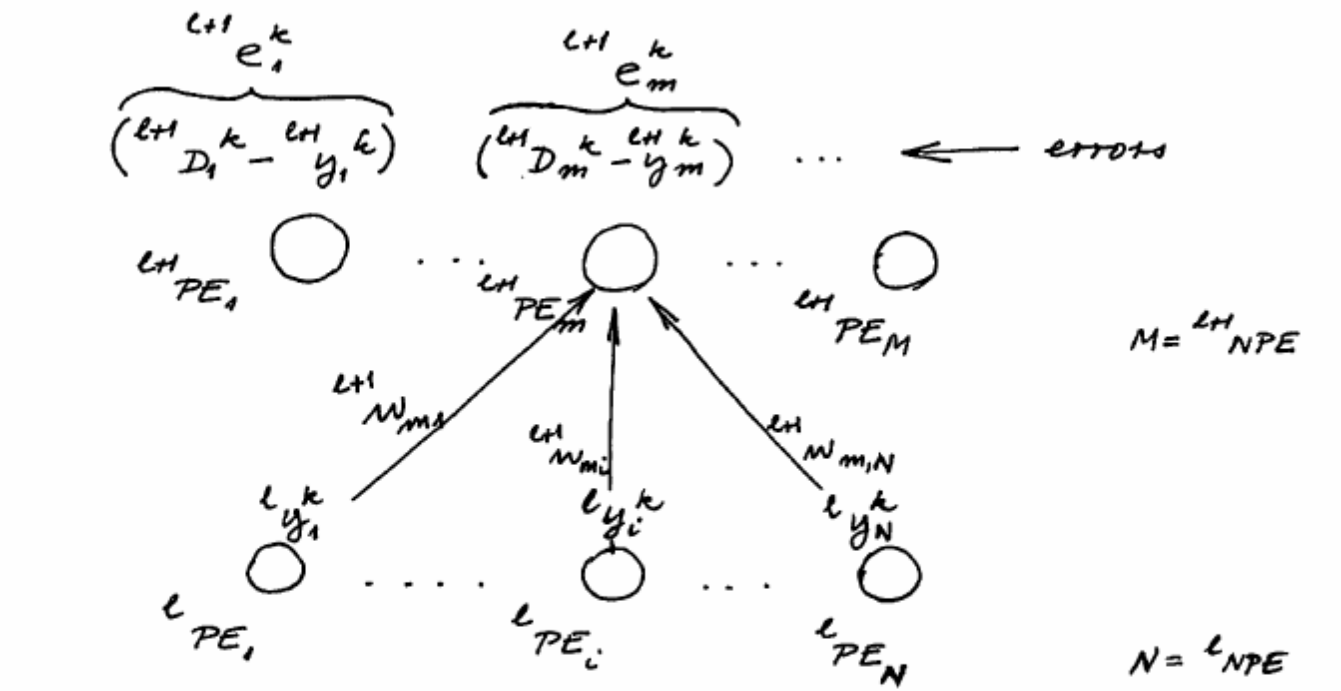
$$ \frac{\partial E^k}{\partial {}^{l}y_i^k} = \sum_{m=0}^{{}^{l+1}NPE} \frac{\partial E^k}{\partial {}^{l+1}y_m^k} \underbrace{\frac{\partial {}^{l+1}y_m^k}{\partial {}^{l}y_i^k}}_{f'({}^{l+1}NET_m^k) \, {}^{l+1}w_{mi}} = \sum_{m=0}^{{}^{l+1}NPE} -[\; {}^{l+1}\delta_m^k] \; {}^{l+1}w_{mi} \tag{6.7} $$

Derivation Of the BP Rule - Illustration



$$\frac{\partial E^k}{\partial {}^l y_i^k} = \frac{\partial}{\partial {}^l y_i^k} \underbrace{\frac{1}{2} \sum_{m=0}^{M} {}^{l+1}(e_m^k)^2}_{E^k \text{ at layer } l+1} = 2\frac{1}{2} \sum_{m=0}^{M} {}^{l+1}e_m^k \frac{\partial {}^{l+1}e_m^k}{\partial {}^l y_i^k} = \sum_{m=0}^{M} \frac{\partial E^k}{\partial {}^{l+1}e_m^k} \frac{\partial {}^{l+1}e_m^k}{\partial {}^{l+1}y_m^k} \frac{\partial {}^{l+1}y_m^k}{\partial {}^l y_i^k}$$

Derivation Of the BP Rule - Illustration



$$\frac{\partial E^k}{\partial {}^l y_i^k} = \frac{\partial}{\partial {}^l y_i^k} \underbrace{\frac{1}{2}\sum_{m=0}^{M} {}^{l+1}(e_m^k)^2}_{E^k \text{ at layer } l+1} = 2\frac{1}{2}\sum_{m=0}^{M} {}^{l+1}e_m^k \frac{\partial {}^{l+1}e_m^k}{\partial {}^l y_i^k} = \sum_{m=0}^{M} \frac{\partial E^k}{\partial {}^{l+1}e_m^k} \frac{\partial {}^{l+1}e_m^k}{\partial {}^{l+1}y_m^k} \frac{\partial {}^{l+1}y_m^k}{\partial {}^l y_i^k}$$

$$= \sum_{m=0}^{M} \frac{\partial E^k}{\partial {}^{l+1}y_m^k} \frac{\partial {}^{l+1}y_m^k}{\partial {}^l y_i^k} = \sum_{m=0}^{M} \frac{\partial E^k}{\partial {}^{l+1}y_m^k} f'({}^{l+1}NET_m^k) {}^{l+1}w_{mi} = \sum_{m=0}^{M} -[\ {}^{l+1}\delta_m^k]\ {}^{l+1}w_{mi}$$

Derivation Of the BP Rule

So, the recursive formula for computing the local ${}^{l}\delta_i^k$ is

$$ {}^{l}\delta_i^k = f'({}^{l}NET_i^k) \sum_{m=0}^{{}^{l+1}NPE} {}^{l+1}\delta_m^k \, {}^{l+1}w_{mi} \tag{6.8} $$

Substituting into the weight update rule (6.3):

$$ {}^{l}\Delta w_{ij} = \gamma \, {}^{l}\delta_i^k \; {}^{l-1}y_j^k = \gamma f'({}^{l}NET_i^k) \left(\sum_{m=0}^{{}^{l+1}NPE} {}^{l+1}\delta_m^k \, {}^{l+1}w_{mi} \right) \; {}^{l-1}y_j^k \tag{6.9} $$

For $l = L$, output layer, by (6.4)

$$ {}^{L}\delta_m^k = ({}^{}D_m^k - {}^{L}y_m^k) \; f'({}^{L}NET_m^k) \tag{6.10} $$

For $l = L - 1$

$$ {}^{L-1}\delta_i^k = f'({}^{L-1}NET_i^k) \sum_{m=0}^{{}^{L}NPE} ({}^{}D_m^k - {}^{L}y_m^k) \; f'({}^{L}NET_m^k) \; {}^{L}w_{mi} \tag{6.11} $$

and so on ...

Derivation Of the BP Rule

To summarize the weight updates:

For the output layer, $l = L$,

$$ {}^{L}\Delta w_{ij} = \gamma({}^{L}D_i^k - {}^{L}y_i^k) f'({}^{L}NET_i^k) \ {}^{L-1}y_j^k \qquad \text{(I)} $$

For $l = L - 1$, by (6.9),

$$ {}^{L-1}\Delta w_{ij} = \gamma f'({}^{L-1}NET_i^k) \ \left(\sum_{m=0}^{{}^{L}NPE} {}^{L}\delta_m^k \ {}^{L}w_{mi} \right) {}^{L-2}y_j^k $$

In general, for any hidden layer $1 \leq l < L$,

$$ {}^{l}\Delta w_{ij} = \gamma f'({}^{l}NET_i^k) \ \left(\sum_{m=0}^{{}^{l+1}NPE} {}^{l+1}\delta_m^k \ {}^{l+1}w_{mi} \right) {}^{l-1}y_j^k \qquad \text{(II)} $$

Batch BP Learning Rule

In *batch* training mode, one weight update is done after the presentation of all those training samples that comprise an *epoch*. The # of training samples in an epoch, K, is a parameter of the learning.
For K training samples ($= 1$ epoch), the total error is defined as

$$E = \frac{1}{2}\sum_{k=1}^{K} E^k = \sum_{k=1}^{K}\sum_{i=1}^{{}^{L}NPE} ({}^{L}e_i^k)^2 \tag{6.12}$$

and consequently, the weight update (6.3) is modified as

$$\begin{aligned} {}^{l}\Delta w_{ij} &= -\gamma\frac{\partial E}{\partial {}^{l}w_{ij}} = -\gamma\sum_{k=1}^{K}\frac{\partial E^k}{\partial {}^{l}w_{ij}} = \gamma\sum_{k=1}^{K} {}^{l}\delta_i^k \;\; {}^{l-1}y_j^k \\ &= \gamma\sum_{k=1}^{K} f'({}^{l}NET_i^k)\left(\sum_{m=0}^{{}^{l+1}NPE} {}^{l+1}\delta_m^k \, {}^{l+1}w_{mi}\right) {}^{l-1}y_j^k \end{aligned} \tag{III}$$

Recipe For BP Training

- 1. Initialize weights: ${}^{l}w_{ij} \in [-a, a], 0 < a < 1$ (usually small).

  Note: ${}^{l}w_{ij}, {}^{l}y_i, {}^{l}NET_i$ all vary with t (or k) $\Longrightarrow$ everything computed with these variables varies with t (or k).

- 2. Initialize time step, $t = 0$, set (or initialize) learning parameters $(\gamma, K, \ldots)$.

  Time-decreasing learning parameters are used commonly except for very simple learning tasks. We will discuss decay schedules, and you will practice in homeworks.

- 3. Present a randomly drawn training vector $\mathbf{x}^k$ to the network, $k \in \{s_1, s_2, \ldots, s_K\}$ (random permutation of the epoch index set), propagate through the current weights until activation reaches the output layer.

$$ {}^{l}y_i^k = f\left(\sum_{j=0}^{{}^{l-1}NPE} {}^{l}w_{ij} \ {}^{l-1}y_j^k \right), l = 1, \ldots, L $$

- 4. Compute ${}^{L}\delta_i^k$ for the output layer (6.10):

$$ {}^{L}\delta_i^k = f'({}^{L}NET_i^k)(D_i^k - {}^{L}y_i^k) $$

Recipe For BP

- 5. Compute ${}^{l}\delta_i^k$ for the hidden layers, $l = L-1, \ldots, 1$ (6.8):

$$ {}^{l}\delta_i^k = f'({}^{l}NET_i^k) \sum_{m=0}^{{}^{l+1}NPE} {}^{l+1}\delta_m^k \ {}^{l+1}w_{mi} $$

- 6. Do 3) - 5) for all traning samples in the epoch. (The length of the epoch, K, can be 1, in which case we are doing *on-line* training.)
- 7. Update all the weights in the network, according to (I), (II), or (III) (above and after (6.12)).
- 8. Increment time, $t = t + 1$. If training met some stopping criterion (convergence, small error, max allowed number of learn steps, ...), stop. Otherwise, go to 3.

> Use at least two simultaneous stopping criteria, a threshold for some convergence measure, and a max allowed number of learning steps. This will prevent a divergent code from running forever.
> Define convergence criteria meaningfully. A "small error" should be small in the context of your goals for your task, not an arbitrary small numerical value. To choice of error measure (MSE, absolute difference, ...) is also important.
>
> You will practice these in homeworks.

Assessing Network Performance

- The quality of learning is measured through the generalization capability: how well the network predicts for test (unseen) data.

  Compare it to the network's performance on training data! Large difference is suspicious. Better performance on testing data than on training data is suspicious.

- To monitor the progress of learning: plot the *learning history* for both training and test data (two graphs in the same plot window).

  The learning history is a curve of the errors recorded at regular intervals of the learning steps (time). To do this, stop the learning at each interval, do a recall with all training samples, compute and save the *training errors*. Do a recall with all test samples, compute and save the *test errors*. Then continue the learning. Plot the recorded errors (any informative error measure computed from the recorded error signals).

- To have a detailed view of the network's performance at a given state: plot the actual outputs of the network against the desired outputs (for all training samples, as well as for all test samples).

To be continued . . . many more issues . . .

Homework Exercise

Implement the simple 2-layer BP ANN in Colin Fyfe's Chapter 1/4, pp. 44 – 45. This can be a hardwired code for the specific architecture and data (the XOR patterns). Run it with the parameters that Colin recommends.
This is the starting point in the next homework, and subsequent problems build on it.