Artificial Neural Networks
Lecture Notes

Stephen Lucci, PhD

Part 6

About this file:

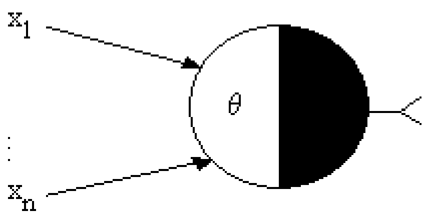
- If you have trouble reading the contents of this file, or in case of transcription errors, email gi0062@bcmail.brooklyn.cuny.edu
- Acknowledgments:
  Background image is from http://www.anatomy.usyd.edu.au/online/neuroanatomy/tutorial1/tutorial1.html (edited) at the University of Sydney Neuroanatomy web page. Mathematics symbols images are from metamath.org's GIF images for Math Symbols web page. Other image credits are given where noted, the remainder are native to this file.

Contents

Neural Net Paradigms - Review and Clarification

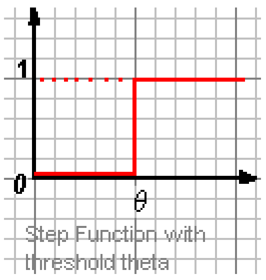
McCulloch-Pitts Networks

- Binary signals - both input and output
- Directed, unweighted edges of excitatory or inhibitory (marked with a small circle) inputs.
- Threshold value θ.
- inputs $x_1, x_2, \dots, x_n$ through n excitatory edges and $y_1, y_2, \dots, y_m$ through m inhibitory edges.
- if m >= 1 and at least one of the signals $y_1, y_2, \dots, y_m$ is the 1, the unit is inhibited and the output is 0.



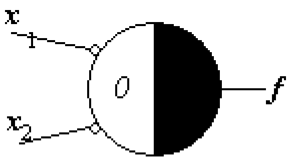
- Otherwise, the total excitation $\bar{x} = x_1 + x_2 + \dots + x_n$ is computed and compared with the threshold θ.

  If $\bar{x} >= \theta$, then unit fires a 1
  If $\bar{x} < \theta$, then output = 1



- Example: The NOR Function

x_1	x_2	$\sum x_i$	NOR
0	0	0	1
0	1	-	0
1	0	-	0
1	1	-	0



NOR TLU

Perceptron Learning Rule

- Weight Updates:

$\overline{w}' = \overline{w} + \alpha(t - y)\,\overline{v}$, or
$\Delta\overline{w} = \alpha(t - y)\overline{v}$

where α is the learning rate, t is the target, y is the output, and $\overline{v}$ is the input vector (equivalent to $\overline{x}$.)

$\Delta w_i = \alpha(t - y)v_i$

with i = 1 to n+1 where $w_{n+1} = \theta$ and $v_{n+1} = -1$, always.

x_1	x_2	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- Example:
  Compute NOR function using the Perceptron Learning Rule

Let initial weights be $w_1 = 0.1$, $w_2 = 0.0$, $\theta = 0.2$, and learning rate $\alpha = 0.5$.

x_1	x_2	w_1	w_2	θ	a	y	t	α(t-y)	Δw_1	Δw_2	$\Delta\theta$
0	0	0.1	0.0	0.2	0	0	1	0.5(1-0)=0.5	0.0	0	-0.5
0	1	0.1	0.0	-0.3	0	1	0	0.5(0-1)=-0.5	0.0	-0.5	0.5
1	0	0.1	-0.5	0.2	0.1	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	1	0.1	-0.5	0.2	-0.4	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
0	0	0.1	-0.5	0.2	0	0	1	0.5(1-0)=0.5	0.0	0.0	-0.5
0	1	0.1	-0.5	-0.3	-0.5	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	0	0.1	-0.5	-0.3	0.1	1	0	0.5(0-1)=-0.5	-0.5	0.0	0.5
1	1	-0.4	-0.5	0.2	-0.9	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
0	0	-0.4	-0.5	0.2	0.0	0	1	0.5(1-0)=0.5	0.0	0.0	-0.5
0	1	-0.4	-0.5	-0.3	-0.5	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	0	-0.4	-0.5	-0.3	-0.4	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	1	-0.4	-0.5	-0.3	-0.9	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
0	0	-0.4	-0.5	-0.3	0.0	1	1	0.0	0.0	0.0	0.0
0	1	-0.4	-0.5	-0.3	-0.5	0	0	0.0	0.0	0.0	0.0
1	0	-0.4	-0.5	-0.3	-0.4	0	0	0.0	0.0	0.0	0.0
1	1	-0.4	-0.5	-0.3	-0.9	0	0	0.0	0.0	0.0	0.0

No changes in the last epoch.

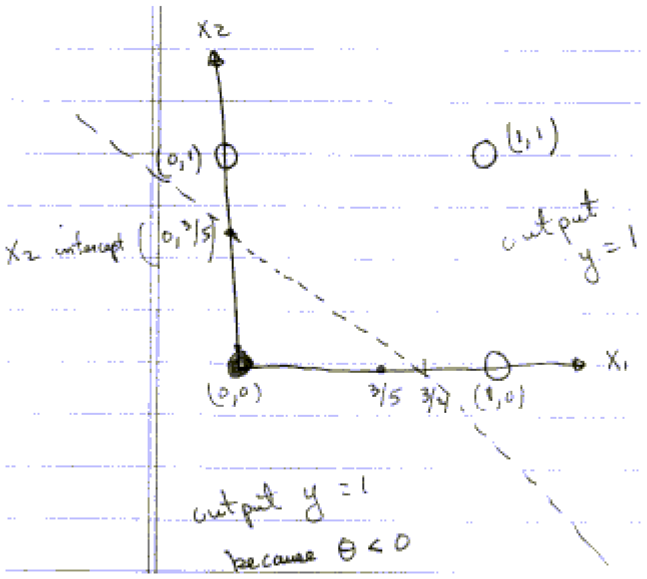
- Final weights:

$w_1 = -0.4$
$w_2 = -0.5$
$\theta = -0.3$

- Hence

x_1	x_2
0	3/5
3/4	0

slope $m = -w_1/w_2 = -(-0.4 / -0.5) = -4/5$
y-intercept $b = \theta/w_2 = -0.3 / -0.5 = 3/5$
$y = mx + b$ (i.e., $x_2 = mx_1 + b$)
$y = -4/5\ x + 3/5$
$x_2 = -4/5\ x_1 + 3/5$



Delta Rule

- Error Function:
  $E = E(w_1, w_2, \ldots, w_{n+1})$
  (where $\overline{w}$ is the augmented weight vector.)
- The optimal weight vector is found by minimizing this function by gradient descent
  $\Delta w_i = -\alpha\ (\partial E/\partial w_i)$
- One attempt at a suitable error function:
  $e^p = 1/2\ (t^p - y^p)^2$
  a^p is smooth and continuous. However, output depends on a^p via the discontinuous step function.
- One remedy:
  $e^p = 1/2\ (t^p - a^p)^2$
  When using the augmented weighted vector, the output changes as the activation changes sign. i.e.,
  $a >= 0 \Rightarrow y = 1$
  One choice that works for target values {-1, 1} - bipolar outputs.
- If the whole training set is presented, we can obtain the true gradient
  $\partial E/\partial w_i$
  (batch training).
- This is computationally intensive.
- Therefore, we adapt the weights based on the presentation of each pattern individually.

- i.e., we present the net with a pattern p, evaluate $\partial e^p/\partial w_i$, and we take this as an <u>estimate</u> of the true gradient $\partial E/\partial w_i$

  $e^p = 1/2\ (t^p - a^p)^2$

  where

  $a^p = w_1x^p_1 + w_2x^p_2 + \ldots + w_{n+1}x^p_{n+1}$

  $\partial e^p/\partial w_i = -(t^p - a^p)x_i^p$
- Hence,

  $\Delta w_i = -\ \alpha(\partial E/\partial w_i)$

  $\approx= -\ \alpha(\partial e^p/\partial w_i)$

  $\Delta w_i = \alpha(t^p - a^p)x_i^p$

  <u>Careful with the signs !</u>
- <u>Pattern training regime</u>:
  Weight changes are made after each vector presentation.
- **Partial Derivatives**
  - Function of two variables.
  - Let f be a function of x and y;

    $f(x,y) = 3x^2y - 5x\cos\pi y$
  - The partial derivative of f with respect to x is the function $\partial f/\partial x$ obtained by differentiating f with respect to x, treating y as constant.

    $\partial f(x,y)/\partial x = 6xy - 5\cos\pi y$

    Alternate notation , $f_x(x,y)$.
  - The partial derivative of f with respect to y is the function $\partial f/\partial y$ obtained by differentiating f with respect to y, treating x as constant.

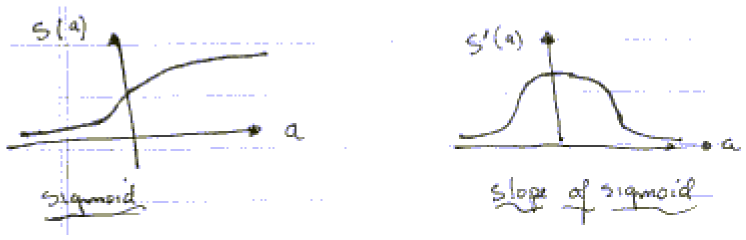
    $f_y(x,y)$ or $\partial f(x,y)/\partial y = 3x^2 + 5\pi x\sin\pi y$
- **Geometric Interpretation**

  $\partial f(x,y)/\partial x$ is the slope of the surface $f(x,y) = 3x^2y - 5x\cos\pi y$ at the point $P(xy, f(x,y))$ in the x-direction.
- **An Example Using the NOR Function**

  Train a two-input TLU with initial weights $w_1 = 0.1$, $w_2 = 0$, $\theta= 0.2$, and learning rate $\alpha= 0.25$

  In the table below note that x_3 is always -1 and is input to $\theta$and is used in the activation column a.
  Note also that $(t^p - a^p)$ is refered to as the delta, or δ, so the the term $\alpha(t^p - a^p)$ is denoted in the table as $\alpha\delta$

x_1	x_2	x_3	w_1	w_2	θ	a	t	$\alpha\delta$	δw_1	δw_2	$\delta\theta$
0	0	-1	0.1	0.0	0.2	-0.2	1	1.2/4 = 0.3	0.0	0	-0.3
0	1	-1	0.1	0.0	-0.1	0.1	-1	-1.1/4 =-0.275	0.0	-0.275	0.275
1	0	-1	0.1	-0.275	+0.175	-0.075	-1	-0.925/4 =-0.231	-0.231	0.0	0.231
1	1	-1	-0.131	-0.275	0.406	-0.812	-1	-0.47	-0.47	-0.47	0.47
0	0	-1	-0.601	-0.745	0.876	-0.876	1	1.876/4 = 0.469	0	0	

- 

- Suppose first that the activation is very large (or small) so that the output is close to 1 or 0, respectively.
  Graph is flat and hence gradient s'(a) is very small.
  Activation close to threshold implies that s'(a) is large.
- alternately - we may use y^p instead of a^p.
  Then,
  $\Delta w_i = \alpha s'(a)(t^p - a^p)x_i^p$.
- Not surprising that s'(a) appears here - Any changes in the weights alter the output (and hence the error) via the activation.
  The effect of any such changes depends thus on the sensitivity of the output with respect to ...(illegible)

Backpropagation

- Layered Network

  n input sites
  k hidden units
  m output units

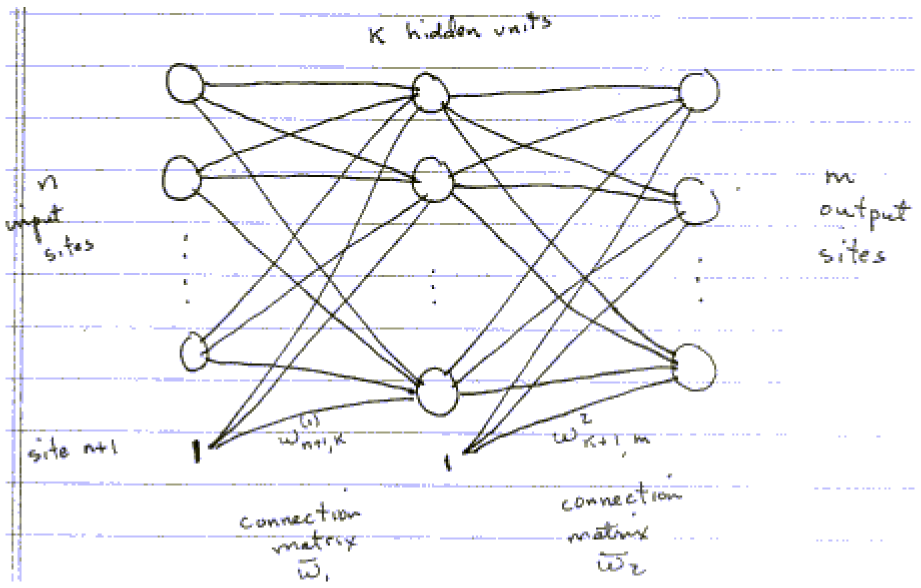


fig 7.17 Notation for the "three-layer" network

- The weight between input site i and hidden unit j: $w_{ij}^{(1)}$
- The weight between hidden unit i and output unit j: $w_{ij}^{(2)}$
- The bias, $-\theta$ is implemented as the weight of an additional edge.

- There are (n+1) * k weights between input sites and hidden units and (k+1) * m between hidden and output units. Thus,

  $\overline{w}_1$ the (n+1) * k matrix with component $w_{ij}^{(1)}$ at the ith row and jthcolumn.

  $\overline{w}_2$ the (k+1) * m matrix with components $w_{ij}^{(2)}$.
- The n-dimensional input vector $\bar{o}$= (o_1, ... , o_n) is extended to: $\hat{o}$= (o_1, ... , o_n, 1) .
- The excitation net_j of the jth hidden unit is given by

$$net_j = \sum_{i=1}^{n+1} w_{ij}^{(1)} \hat{o}_i$$

- The activation function is a sigmoid and the output $o_j^{(1)}$ is

$$o_j^{(1)} = s\left(\sum_{i=1}^{n+1} w_{ij}^{(1)} \hat{o}_i\right)$$

- Excitation of all units in the hidden layer = $\hat{o}\overline{w}_1$
  $\bar{o}^{(1)}$ the vector whose components are the outputs of hidden units

$$\bar{o}^{(1)} = s(\hat{o}\overline{w}_1) .$$

- Excitation of units in the output layer is computed using $\hat{o}^{(1)}$ = ($o_1^{(1)}$, ... , $o_k^{(1)}$, 1) .
- The output of the network is the m-dimensional vector
  $\bar{o}^{(2)}$ = s(IMG src="img06/ohat.gif">$^{(1)}\overline{w}_2$) .

Steps of the Algorithm

- The following schematic is for a single input-output pair ($\bar{o}$,$\bar{t}$)

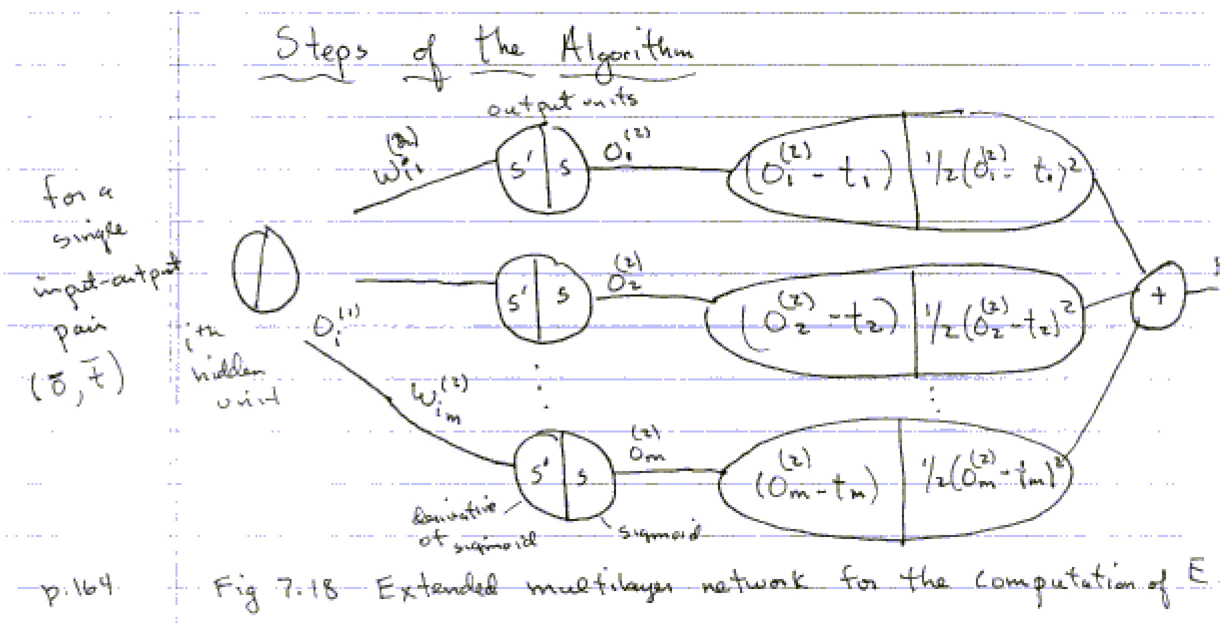


fig 7.18 Extended multilayer network for the computation of E (p.164)

- The error function for p input-output examples - we would need p copies of the above network.

- The weights for the network are chosen randomly.

The Backpropagation Algorithm

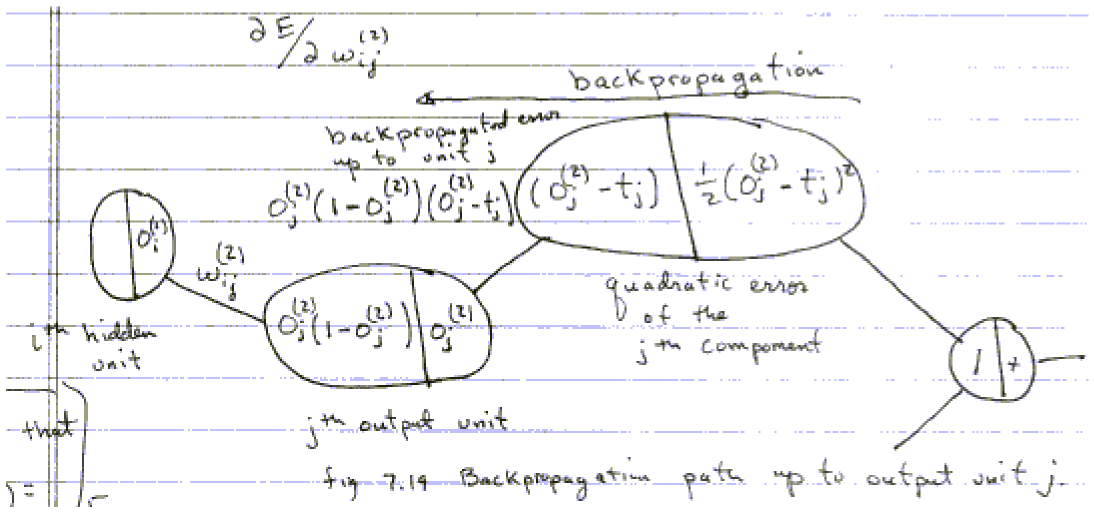
- i) Feed-forward Computation
- ii) Backpropagation to the output layer
- iii) Backpropagation to the hidden layer
- iv) Weight Updates.

- The algorithm is stopped when the value of the error function has become sufficiently small.
- First Step: *Feedforward Computation*

  The vector $\bar{o}$ is presented to the network.
  The vectors $\bar{o}^{(1)}$ and $\bar{o}^{(2)}$ are computed and stored.
  The evaluated derivatives of the activation functions are also stored at each unit.

- Second Step: *Backpropagation to the Output Layer*

  We are looking for the first set of partial derivatives $\partial E / \partial w_{ij}^{(2)}$



$\partial E / \partial w_{ij}^{(2)}$
backpropagation
backpropagated error up to unit j
$o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j)$
$(o_j^{(2)}-t_j)$
$\frac{1}{2}(o_j^{(2)}-t_j)^2$
$o_i^{(1)}$
$w_{ij}^{(2)}$
$o_j^{(2)}(1-o_j^{(2)})$
$o_j^{(2)}$
quadratic error of the jth component
ith hidden unit
jth output unit
fig 7.19 Backpropagation path up to output unit j.

fig 7.19 Backpropagation path up to output unit j

From this path, we can collect all the multiplicative terms which define the backpropagated error $\delta_j^{(2)}$

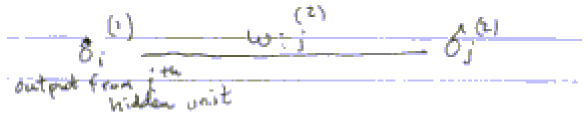
Recall that $d/dx\, s(x) = e^{-x} / (1+e^{-x})^2 = s(x)(1 - s(x))$. Thus, ...

$$\delta_j^{(2)} = o_j^{(2)}(1 - o_j^{(2)})(o_j^{(2)} - t_j)$$

and the partial derivative we are looking for is

$$\partial E / \partial w_{ij}^{(2)} = [\, o_j^{(2)}(1 - o_j^{(2)})(o_j^{(2)} - t_j)\,] o_i^{(1)}$$
$$= \delta_j^{(2)} o_i^{(1)}$$

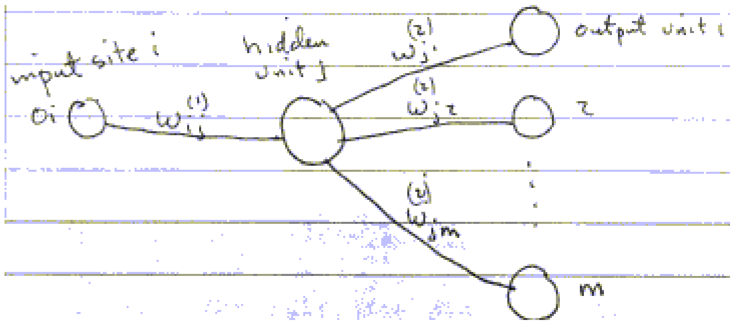
- N.B. For this last step we consider $w_{ij}^{(1)}$ to be a variable and its input $o_i^{(1)}$ a constant.
- The general situation during the backpropagation algorithm:



At the input side of the edge with weight w_{ij} we have $o_i^{(1)}$ and at the output side , the backpropagated error $\delta_j^{(2)}$.

- Third Step: *Backpropagation to the Hidden Layer*

  We want to compute $\partial E / \partial w_{ij}^{(1)}$.
  Each unit j in the hidden layer is connected to each unit q in the output layer with an edge of weight $w_{jq}^{(2)}$ for q = 1, ... , m.



The backpropagated error up to unit j in the hidden layer must be computed taking into account all possible backward paths as shown:

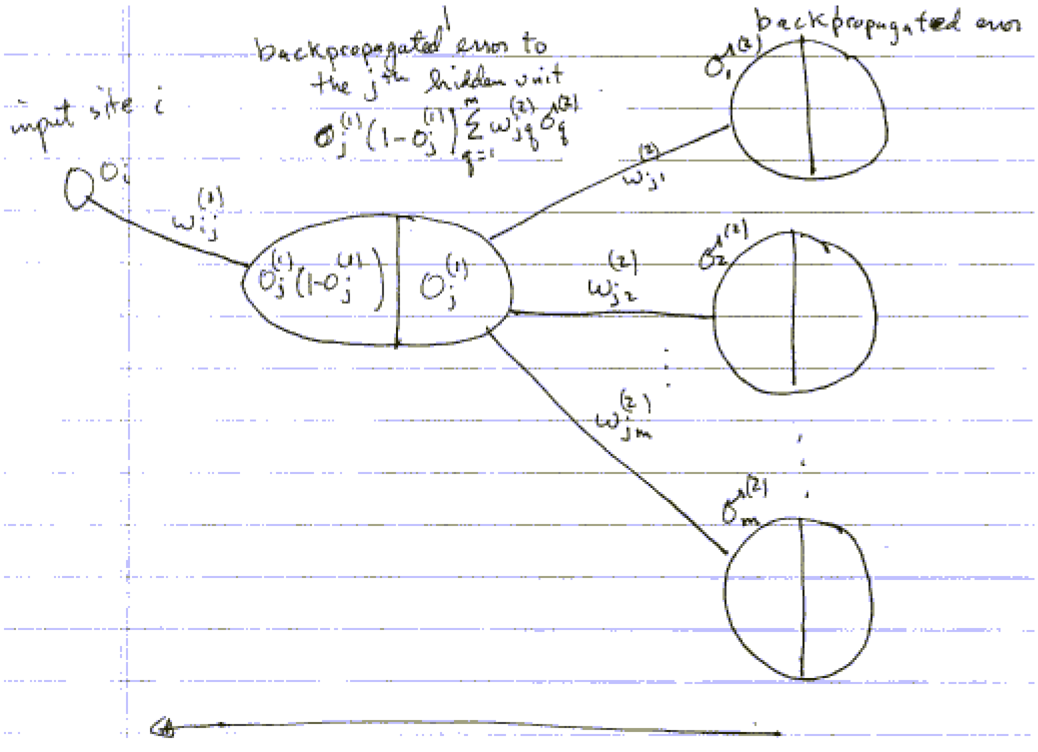


fig 7.21 All paths up to input site i

The backpropagated error is

$$\delta_j^{(1)} = o_j^{(1)}(1 - o_j^{(1)}) \sum_{q=1}^{m} w_{jq}^{(2)}\, \delta_q^{(2)}$$

The partial derivative we are looking for:

$$\partial E/\partial w_{ij}^{(1)} = \delta_j^{(2)} o_i.$$

- Fourth Step: *Weight Updates*
  - After computing all partial derivatives, the network weights are updated in the negative gradient direction.
  - A learning constant y defines the step length of the correction.
  - The corrections for the weights are given by:

    $\Delta w_{ij}^{(2)} = -y o_i^{(1)} \delta^{(2)}$, for i = 1, ... , k+1 and j = 1, ... , m

    and

    $\Delta w_{ij}^{(1)} = -y o_i \delta_j^{(2)}$, for i = 1, ... , n+1 and j = 1, ... , k

    with the convention that:

    $o_{n+1} = o_{k+1}^{(1)} = 1.$
  - Important: Corrections are made to the weights only after the backpropagated error has been computed for all units in the network.

More Than One Training Pattern

- In the case p > 1 input-output patterns, an extended network is used to compute the error function for each of them separately.
- The weight corrections are computed for each pattern yielding for $w_{ij}^{(1)}$ the corrections:

  $\Delta_1 w_{ij}^{(1)} , \Delta_2 w_{ij}^{(1)} , \dots , \Delta_p w_{ij}^{(1)}$
- The necessary updates in the gradient direction is then:

  $\Delta w_{ij}^{(1)} = \Delta_1 w_{ij}^{(1)} + \Delta_2 w_{ij}^{(1)} + \dots + \Delta_p w_{ij}^{(1)}$
- (Batch or offline updates are rather expensive as a training set may consist of thousands of patterns.)
- Often, however, the weight updates are made sequentially after each pattern presentation
- (online training) more efficient
- Here, the corrections do not exactly follow the negative gradient direction. This adds some noise to the gradient direction - can help to prevent falling into shallow local minima of the error function.
- If the training patterns are selected randomly, the search direction oscillates around the exact gradient direction, and, on average, the algorithm implements a form of descent in the error function.