CI & A

ARTIFICIAL NEURAL NETWORKS - ANNs

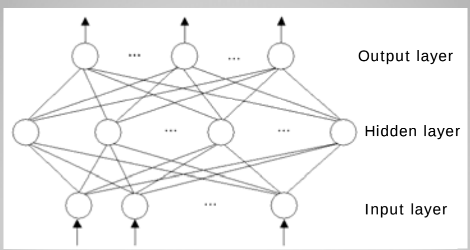
THE MULTILAYER PERCEPTRON

General context

- Classical neural models that used formal neurons were not provided with an automatic learning algorithm.
- The proposal of using **hidden** units / neurons and learning through **error back-propagation** led to the **Multilayer Perceptron** - **MLP**.

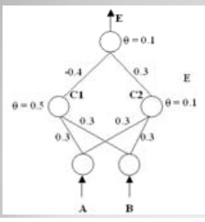
General context

MLP architecture



General context

Learning = creation of internal representations associated to input information.



A	0	0	1	1	
B	0	1	0	1	
C1	0	0	0	1	C1 = A & B
C2	0	1	1	1	C2 = A ∨ B
E	0	1	1	0	E = not (C1) & C2

How? By weights adjustment.

Generalized Delta Rule

The error back-propagation algorithm proposed by Rumelhart and McClelland in 1986 is sometimes called Generalized Delta Rule (notation "Delta" comes from the Greek letter Δ).

Generalized Delta Rule

Learning / Training Data Set

Variables			$f(x,y,z)$
x	y	z	
......	......	......	......
......	......	......	......
....			
......	......	......	......

A table used to define the learning data set for the MLP; the case for 3 inputs – x, y and z – and 1 output – $f(x,y,z)$.

Weights initialization

Weights are initialized with random values, usually chosen in the range (-1, 1).

Generalized Delta Rule

Hypothesis used to apply the algorithm

(i) the MLP-type neural network uses hidden units / neurons;

(ii) activation functions of hidden and output units are considered continuous and differentiable;

(iii) if applicable, output values are scaled within appropriate limits with respect to the activation function.

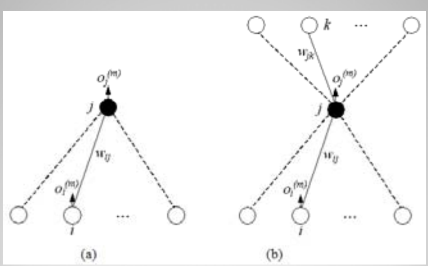
Generalized Delta Rule

2 main stages

- Forward propagation of input pattern $x^{(m)}$ to calculate the actual output $o^{(m)}$.
- Error back-propagation: actual output $o^{(m)}$ is compared to the desired one $d^{(m)}$ and the error term $e^{(m)} = o^{(m)} - d^{(m)}$ is propagated back into the network – from the output layer to the input layer – by adjusting weights with quantity $\Delta w^{(m)}$, based on the least square error principle.

Generalized Delta Rule

Explanation: (a) unit *j* is in the output layer
or (b) unit *j* is in the hidden layer



Generalized Delta Rule

Clause 1

For each input – output pattern *m* of the learning data set, the correction of weights w_{ij} - noted $\Delta^{(m)}w_{ij}$ – for connection between unit *j* and unit *i* in the lower layer is proportional with an error term $\delta_j^{(m)}$ associated to unit *j*:

$$\Delta^{(m)}w_{ij} = \eta \cdot \delta_j^{(m)} \cdot o_i^{(m)}$$

where η is a coefficient called *learning rate*.

Generalized Delta Rule

Clause 2

If unit *j* is in the output layer, the error term $\delta_j^{(m)}$ is calculated based on the deviation between the actual $o_j^{(m)}$ and the desired $d_j^{(m)}$ output values and the derivative of the activation function *f* of unit *j* with respect to the net input for pattern *m*, denoted $net_j^{(m)}$:

$$\delta_j^{(m)} = \left(d_j^{(m)} - o_j^{(m)}\right) \cdot f'\left(net_j^{(m)}\right)$$

Generalized Delta Rule

Clause 2 - continued

If unit *j* is in the hidden layer, being linked with synaptic connections to units *k* in the output layer, the error term $\delta_j^{(m)}$ is proportional to the sum of all of error terms associated to output units *k* , modified by the weights of those connections w_{jk} and the activation function derivative with respect to net input $net_j^{(m)}$:

$$\delta_j^{(m)} = \left(\sum_k \delta_k^{(m)} \cdot w_{jk}\right) \cdot f'\left(net_j^{(m)}\right)$$

Generalized Delta Rule

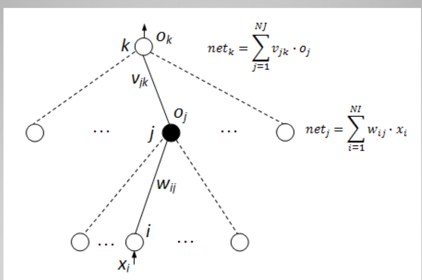
Clause 3

The Generalized Delta Rule is based on the principle of square error minimization; this error describes the square deviation between actual and desired values at the output of the network:

$$E^{(m)} = \frac{1}{2}\sum_{j=1}^{J}\left(d_j^{(m)} - o_j^{(m)}\right)^2$$

Generalized Delta Rule

Architecture



Generalized Delta Rule

Principle

The error back-propagation by Generalized Delta Rule corresponds to a minimization of error E by a gradient method:

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \cdot \nabla E\,(\mathbf{w}^{t}) = \mathbf{w}^{t} - \eta \cdot \Delta \mathbf{w}^{t}$$

i.e.:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \cdot \frac{\partial E}{\partial w_{ij}}\bigg|_{w_{ij}^{t}} = w_{ij}^{t} - \Delta w_{ij}^{t}$$

Generalized Delta Rule

Principle - continued

If you drop the index *m* that shows the number of the pattern in the learning data set, and consider the general case of a network with NK units on the output layer, the error for one of the learning pattern is:

$$E = \frac{1}{2}\sum_{k=1}^{NK}(o_k - d_k)^2$$

Generalized Delta Rule

Weights updating - v_{jk}

$$\frac{\partial E}{\partial v_{jk}} = (o_k - d_k)\cdot\frac{\partial o_k}{\partial v_{jk}} = (o_k - d_k)\cdot f'(net_k)\cdot\frac{\partial net_k}{\partial v_{jk}} =$$

$$= \underbrace{(o_k - d_k)\cdot f'(net_k)}_{\text{error term } \delta_k}\cdot o_j$$

$$\Rightarrow \quad \Delta v_{jk} = \eta\cdot\delta_k\cdot o_j$$

Generalized Delta Rule

Weights updating - w_{ij}

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{NK}(o_k - d_k)\cdot\frac{\partial o_k}{\partial w_{ij}} = \sum_{k=1}^{NK}(o_k - d_k)\cdot f'(net_k)\cdot\frac{\partial net_k}{\partial w_{ij}} =$$

$$= \sum_{k=1}^{NK}\underbrace{(o_k - d_k)\cdot f'(net_k)}_{\text{error term } \delta_k}\cdot v_{jk}\cdot\frac{\partial o_j}{\partial w_{ij}} =$$

$$= \left(\sum_{k=1}^{NK}\delta_k\cdot v_{jk}\right)\cdot\frac{\partial o_j}{\partial w_{ij}} = \left(\sum_{k=1}^{NK}\delta_k\cdot v_{jk}\right)\cdot f'(net_j)\frac{\partial net_j}{\partial w_{ij}} =$$

$$= \underbrace{\left(\sum_{k=1}^{NK}\delta_k\cdot v_{jk}\right)\cdot f'(net_j)}_{\text{error term } \delta_j}\cdot x_i \quad \Rightarrow \quad \Delta w_{ij} = \eta\cdot\delta_j\cdot x_i$$

Generalized Delta Rule

Logistic Sigmoid function

Activation function: $f(x) = \frac{1}{1+e^{(-x+b)}}$

... and its derivative:

$$f'(x) = \frac{-(-1)\cdot e^{(-x+b)}}{\left[1+e^{(-x+b)}\right]^2} = \frac{e^{(-x+b)}}{\left[1+e^{(-x+b)}\right]^2} = \frac{1+e^{(-x+b)}-1}{\left[1+e(-x+b)\right]^2} =$$

$$= \frac{1}{1+e^{(-x+b)}} - \frac{1}{\left[1+e^{(-x+b)}\right]^2} = f(x) - \left[f(x)\right]^2 = f(x)\cdot\left[1-f(x)\right]$$

Generalized Delta Rule

Logistic Sigmoid function - continued

Hypothesis: it is considered that the MLP uses only logistic sigmoid – type activation functions.

$$\frac{\partial E}{\partial v_{jk}} = (o_k - d_k)\cdot o_k \cdot (1-o_k)\cdot o_j \qquad v_{jk}^{t+1} = v_{jk}^{t} - \eta\cdot(o_k - d_k)\cdot o_k\cdot(1-o_k)\cdot o_j$$

$$\frac{\partial E}{\partial w_{ij}} = \left[\sum_{k=1}^{NK}(o_k - d_k)\cdot o_k\cdot(1-o_k)\cdot v_{jk}\right]\cdot o_j\cdot(1-o_j)\cdot x_i$$

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta\cdot\left[\sum_{k=1}^{NK}(o_k - d_k)\cdot o_k\cdot(1-o_k)\cdot v_{jk}\right]\cdot o_j\cdot(1-o_j)\cdot x_i$$

Error back-propagation algorithm – the basic form

1. Definition of MLP network architecture: number of units in each layer (*I, J, K*) and the learning data set $\{x^{(m)}, d^{(m)}\}$ $m = 1, \ldots, M$. Definition of the number of training cycles: C_{max}.
2. Definition of network parameters: learning rates for weights *v* and *w*, denoted η_1 and η_2.
3. Initialization of network weights with random values in the range (-1, 1):

$$v_{jk} = 2\cdot random(\,) - 1;$$
$$w_{ij} = 2\cdot random(\,) - 1;$$
$$(i = 1,\ldots,I;\; j = 1,\ldots,J;\; k = 1,\ldots,K).$$

Error back-propagation algorithm – the basic form

4. Weights updating:

for c = 1 to C_{max} do.
for m = 1 to M do.
// Forward propagation in the first layer
for j = 1 to J do
$y_j = 0$;
for i = 1 to I do $y_j = y_j + w_{ji} \cdot x_i$
// Forward propagation in the second layer
for k = 1 to K do
$o_k = 0$;
for j = 1 to J do $o_k = o_k + v_{kj} \cdot y_j$

Error back-propagation algorithm – the basic form

for j = 1 to J do
// Weight adjustment for the 2-nd layer
for k = 1 to K do

$$v_{jk} = v_{jk} + \eta_1 \cdot \left(d_k^{(m)} - o_k\right) \cdot o_k \cdot (1 - o_k) \cdot y_j$$

// Weight adjustment for the 1-st layer
for i = 1 to I do

$$w_{ij} = w_{ij} + \eta_2 \cdot \sum_{k=1}^{K} \left[\left(d_k^{(m)} - o_k\right) \cdot o_k \cdot (1 - o_k) \cdot v_{jk}\right] \cdot o_k \cdot (1 - o_k) \cdot x_i^{(m)}$$

5. The network was trained on the M patterns in C_{max} cycles; its characteristics were embedded in the weights v_{jk} and w_{ij}.

Convergence acceleration procedures

(i) optimization of the network weights initialization,
(ii) stabilization of the weights adjusting process,
(iii) accelerate the convergence by applying more efficient optimization techniques and
(iv) selecting a network architecture to ensure best performance.

Convergence acceleration– Weights initialization

(i) Standard procedure: initialize weights with random, small values in the range (-1, 1) or (-0.5 , 0.5);

(ii) Russo's rule:

$$-\frac{2.4}{I} \leq w_{ij} \leq \frac{2.4}{I}$$

where **I** – number of input connections of the unit.

Convergence acceleration– Weights initialization

(iii) Nguyen – Widrow procedure :

define parameter: $\beta = 0.7 \cdot J^{\frac{1}{I}}$

then initialize weights using: $w_{ij} = \frac{\beta}{\|w^*\|} \cdot w^*_{ij}$

where: $[w^*] = \{w^*_{11}, \ldots, w^*_{IJ}\}$

Convergence acceleration– Adding a momentum term

Aim: damping trajectory oscillations on the surface error.

Solution: introduction within the weights adjustment formula of a "momentum" term proportional to movement speed (the correction value from the previous iteration).

$$\mathbf{z}^{t+1} = \mathbf{z}^{t} - \eta \cdot \nabla E(\mathbf{z}^{t}) + \mu \cdot (\mathbf{z}^{t} - \mathbf{z}^{t-1})$$

or: $z_{pq}^{t+1} = z_{pq}^{t} - \eta \cdot \frac{\partial E}{\partial z_{pq}}\Big|_{z_{pq}^{t}} + \mu \cdot (z_{pq}^{t} - z_{pq}^{t-1})$

Convergence acceleration– Adding a momentum term

Efects :

•at the beginning of training, when weights corrections are relatively large, ensures moving in the general direction of error decreasing, avoiding "capture" in local minima;
•the momentum term contributes to damping oscillations and smoothing trajectory of successive approximations on the error surface.

Convergence acceleration– Learning rate

Progressive reduction of the learning rate :

•In the initial stage, a great learning rate is recommended: the movement on the error surface occurs with large steps, which allows overcoming local minima.
•After getting close to the minimum: reducing the value of the learning rate allows the stabilization of the searching process around this minimum, reducing the risk of surpassing it.

Convergence acceleration– Learning rate

Principles:

•If in two successive iterations derived $\partial E/\partial w$ retains the sign (i.e., the error E is still falling), the learning rate should be increased to accelerate the approach to the minimum;
•If in two successive iterations derived $\partial E/\partial w$ changes its sign (i.e., the error E is starting to grow), the learning rate should be decreased to return to the decreasing slope.

Convergence acceleration– Learning rate

Learning rate adaptation:

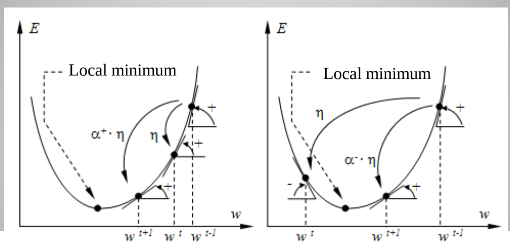
$$\eta^{t+1} = \alpha^{+} \cdot \eta^{t} \quad daca \quad \left(\frac{\partial E}{\partial w}\right)^{t-1} \cdot \left(\frac{\partial E}{\partial w}\right)^{t} > 0$$

$$\eta^{t+1} = \alpha^{-} \cdot \eta^{t} \quad daca \quad \left(\frac{\partial E}{\partial w}\right)^{t-1} \cdot \left(\frac{\partial E}{\partial w}\right)^{t} < 0$$

$$\eta^{t+1} = \eta^{t} \quad daca \quad \left(\frac{\partial E}{\partial w}\right)^{t-1} \cdot \left(\frac{\partial E}{\partial w}\right)^{t} = 0$$

Convergence acceleration– Learning rate

Learning rate adaptation:



Convergence acceleration– Rprop learning function

RProp – Resilient Propagation

Principles:

RProp algorithm does not use values of derivatives $\partial E/\partial z_{ps}$ but only their signs. It uses one coefficient δ_{ps} for each weight z_{ps} which changes its value, based on the evolution of the signs error function derivatives.

Convergence acceleration– Rprop learning function

Principles:

$$\delta_{ps}^{(t+1)} = \begin{cases} \eta^{+} \cdot \delta_{ps}^{(t)} & daca \quad \left(\frac{\partial E}{\partial z_{ps}}\right)^{t+1} \cdot \left(\frac{\partial E}{\partial z_{ps}}\right)^{t} > 0 \\ \eta^{-} \cdot \delta_{ps}^{(t)} & daca \quad \left(\frac{\partial E}{\partial z_{ps}}\right)^{t+1} \cdot \left(\frac{\partial E}{\partial z_{ps}}\right)^{t} < 0 \\ \delta_{ps}^{(t)} & daca \quad \left(\frac{\partial E}{\partial z_{ps}}\right)^{t+1} \cdot \left(\frac{\partial E}{\partial z_{ps}}\right)^{t} = 0 \end{cases}$$

Convergence acceleration– Rprop learning function

Weights updating:

$$\Delta z_{ps}^{(t+1)} = \begin{cases} -sign\left[\left(\frac{\partial E}{\partial z_{ps}}\right)^{t+1}\right] \cdot \delta_{ps}^{(t+1)} & daca \quad \left(\frac{\partial E}{\partial z_{ps}}\right)^{t+1} \cdot \left(\frac{\partial E}{\partial z_{ps}}\right)^{t} \geq 0 \\ -sign\left[\Delta z_{ps}^{(t)}\right] \cdot \delta_{ps}^{(t+1)} & daca \quad \left(\frac{\partial E}{\partial z_{ps}}\right)^{t+1} \cdot \left(\frac{\partial E}{\partial z_{ps}}\right)^{t} < 0 \end{cases}$$

$$z_{ps}^{(t+1)} = z_{ps}^{(t)} + \Delta z_{ps}^{(t+1)}$$

Stopping criteria

Criterion of maximum number of learning cycles

- T_{max} too low: capture in locala minima;
- T_{max} too high: network specializing on the learning data set (over-training or over-learning).
- Consequence: modest values for T_{max} and off-line tests.

Stopping criteria

Criterion of the test data set

The initial learning data set is divided in:
- The training data set
- The test data set

The learning stage uses the training data set and learning is stopped when, after a fixed number of consecutive of learning cycles, the error on the test data set begins to increase.