function BACK-PROP-LEARNING(*examples, network*)
returns a neural network

inputs:
examples, a set of examples, each with input vector **x** and output vector **y**.
network, a multilayer network with L layers, weights $W_{j,i}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

for each weight $w_{i,j}$ in *network* do
$w_{i,j} \leftarrow$ a small random number
repeat
for each example **(x,y) in** *examples* **do**
/* Propagate the inputs forward to compute the outputs. */
1a **for each** node i in the input layer **do** // Simply copy the input values.
$a_i \leftarrow x_i$
1b **for** $l = 2$ to L **do** // Feed the values forward.
for each node j in layer l **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
2a **for each** node j in the output layer **do** // Compute the error at the output.
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
2b /* Propagate the deltas backward from output layer to input layer */
for $l = L - 1$ **to** 1 **do**
for each node i in layer l **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$ // "Blame" a node as much as its weight.
2c /* Update every weight in network using deltas. */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$ // Adjust the weights.
until some stopping criterion is satisfied

return *network*