Clase 8
Redes Neuronales Feed-Forward

Edgardo Ferretti[1]

[1]Universidad Nacional de San Luis, Argentina
e-mails: ferretti@unsl.edu.ar, edgardo.ferretti@gmail.com

Universidad Nacional de San Luis
DEPARTAMENTO DE INFORMATICA
FACULTAD DE CIENCIAS FISICO MATEMATICAS Y NATURALES

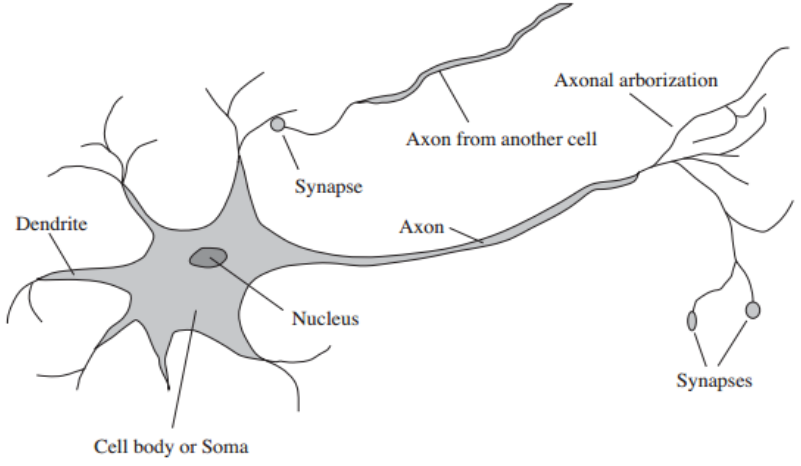
Curso: Minería de Datos (14 de diciembre de 2018)

Redes Neuronales (NN)

- Una red neuronal es un modelo matemático simple de cómo se produce la actividad cerebral en los seres humanos; es decir, modeliza la hipótesis de que la actividad cerebral consiste primariamente de actividad electroquímica entre células cerebrales (neuronas).
- Una neurona tiene como función principal recolectar, procesar y diseminar señales eléctricas.

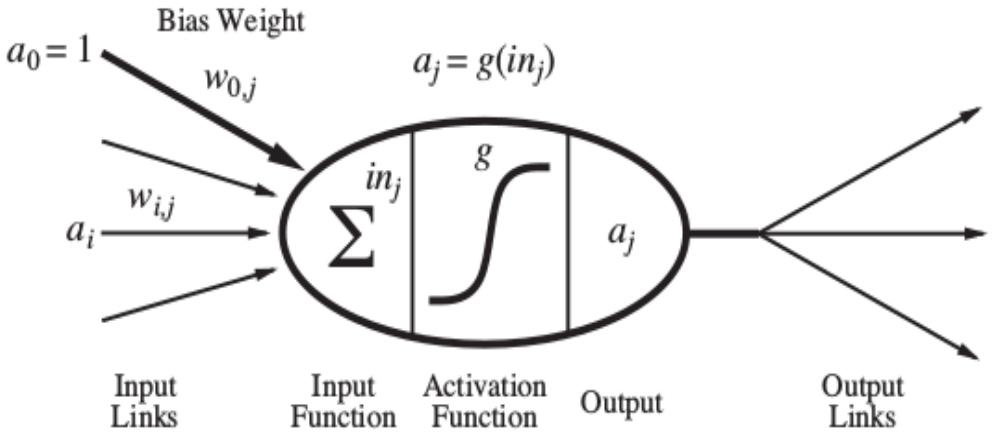
Redes Neuronales (NN)

- Una red neuronal es un modelo matemático simple de cómo se produce la actividad cerebral en los seres humanos; es decir, modeliza la hipótesis de que la actividad cerebral consiste primariamente de actividad electroquímica entre células cerebrales (neuronas).
- Una neurona tiene como función principal recolectar, procesar y diseminar señales eléctricas.



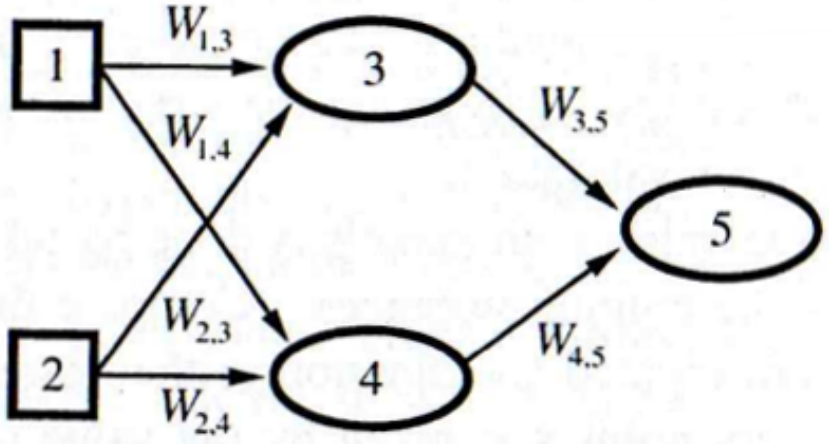
Redes Neuronales (NN)

- Una red neuronal es un modelo matemático simple de cómo se produce la actividad cerebral en los seres humanos; es decir, modeliza la hipótesis de que la actividad cerebral consiste primariamente de actividad electroquímica entre células cerebrales (neuronas).
- Una neurona tiene como función principal recolectar, procesar y diseminar señales eléctricas.



Redes Neuronales (NN)

- Las redes neuronales están compuestas de nodos / unidades (neurona artificial) que se conectan entre ellas por medio de arcos (links).
- Un link entre la unidad i y unidad j sirve para propagar la activación a_i desde i a j.
- Cada link tiene asociado un peso numérico $w_{i,j}$, que determina la fuerza y el signo de la conexión.



Redes Neuronales (NN)

- Cada unidad j computa una suma ponderada sobre sus entradas: $in_j = \sum_{i=0}^{n} w_{i,j} a_i$
- Luego, a esta suma se le aplica una función de activación g para obtener la salida: $a_j = g(in_j)$
- Un impulso es disparado cuando una combinación lineal de las entradas de la neurona excede cierto umbral.

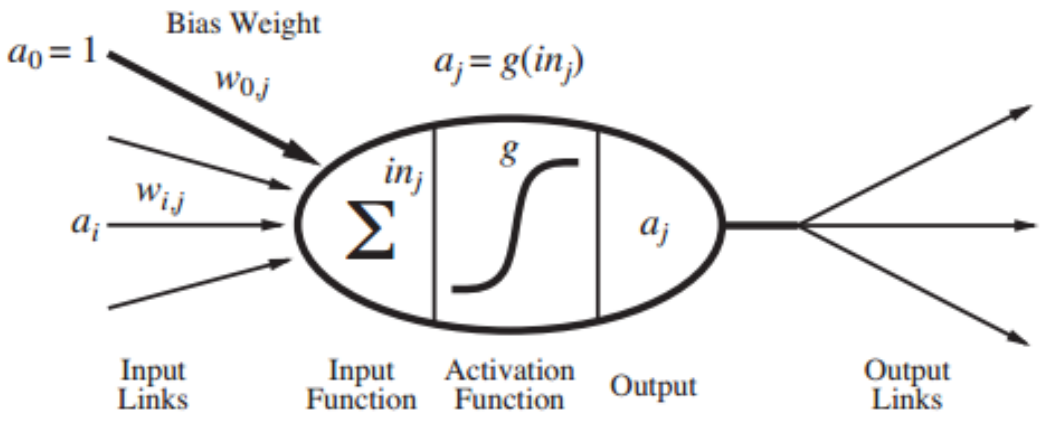
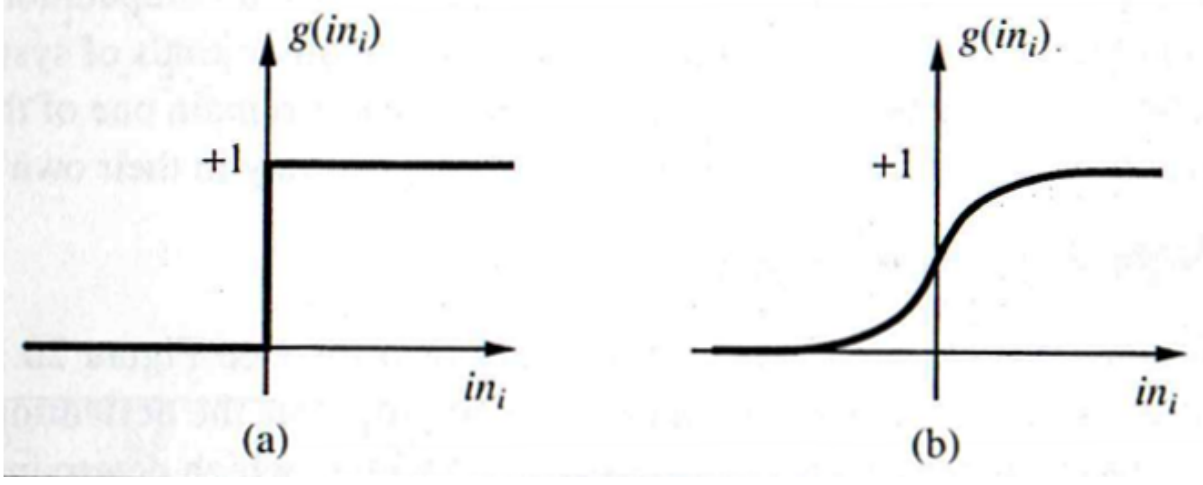


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

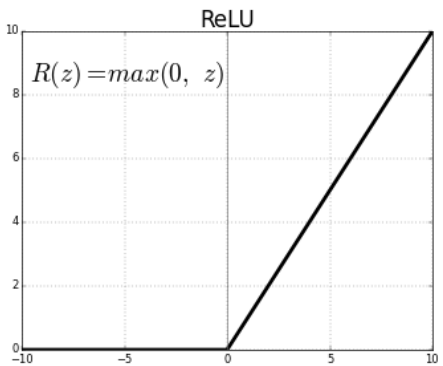
Redes Neuronales (NN)

- La función de activación g típicamente es un umbral duro, como el que se muestra en la figura (a), y en cuyo caso la unidad se llama perceptron; o bien una función logística, como la que se muestra en la figura (b), y que suele llamarse perceptron sigmoidal.
- Ambas funciones de activación garantizan una propiedad muy importante, y es que la NN puede representar funciones no lineales.



Redes Neuronales (NN): la función de activación ReLU

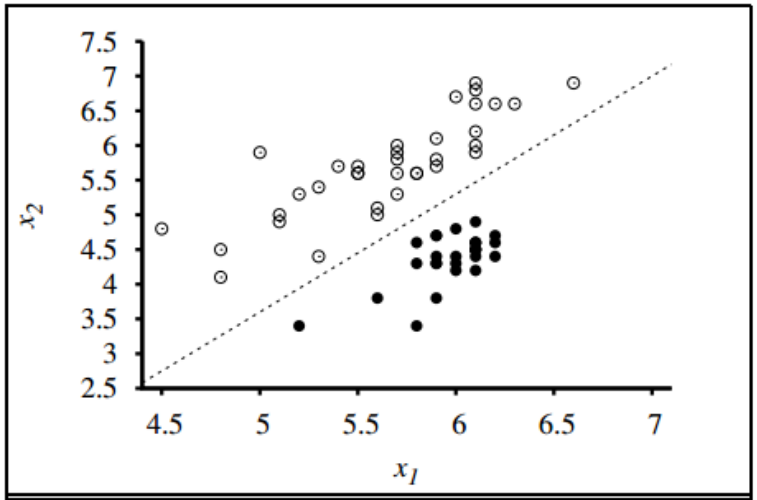
- La función de activación `ReLU` (rectified linear unit) es una de las más populares a ser utilizadas en las capas ocultas de las NN profundas. Se define como $g(z) = \max\{0, z\}$:



- Las unidades ReLU son ampliamente utilizadas en las redes neuronales convolucionales con el fin de reconocer objetos en imágenes.
- Existen algoritmos específicos para que estas unidades puedan aprender vía métodos de descenso del gradiente.[1]

[1] Diederik P. Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization* (2014) `https://arxiv.org/abs/1412.6980`

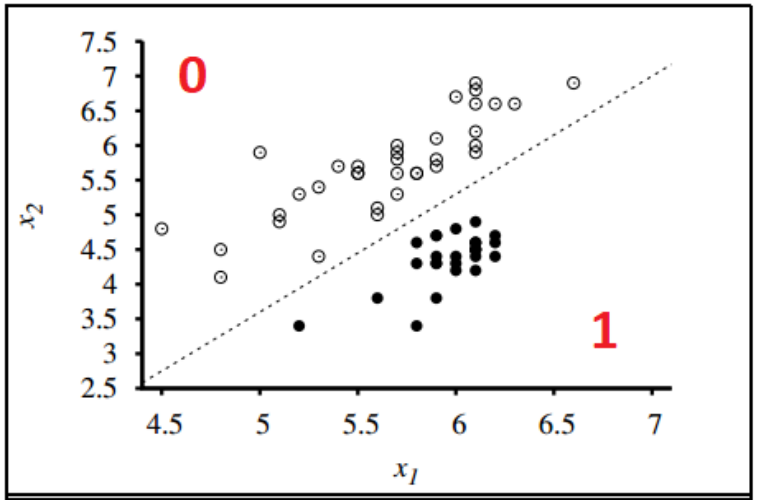
Redes Neuronales (NN)



Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998).

- Los perceptrones pueden representar solamente funciones linealmente separables.
- Un perceptron retorna 1, sólo si la suma ponderada de sus entradas (incluyendo el bias) es positiva: $\sum_{j=0}^{n} w_j x_j > 0$ o $\mathbf{w} \cdot \mathbf{x} > 0$.

Redes Neuronales (NN)

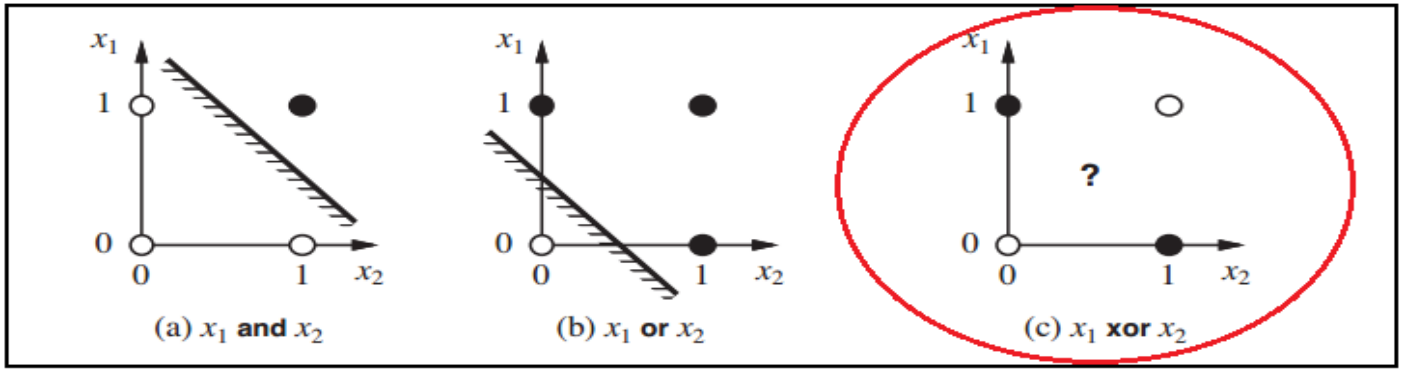


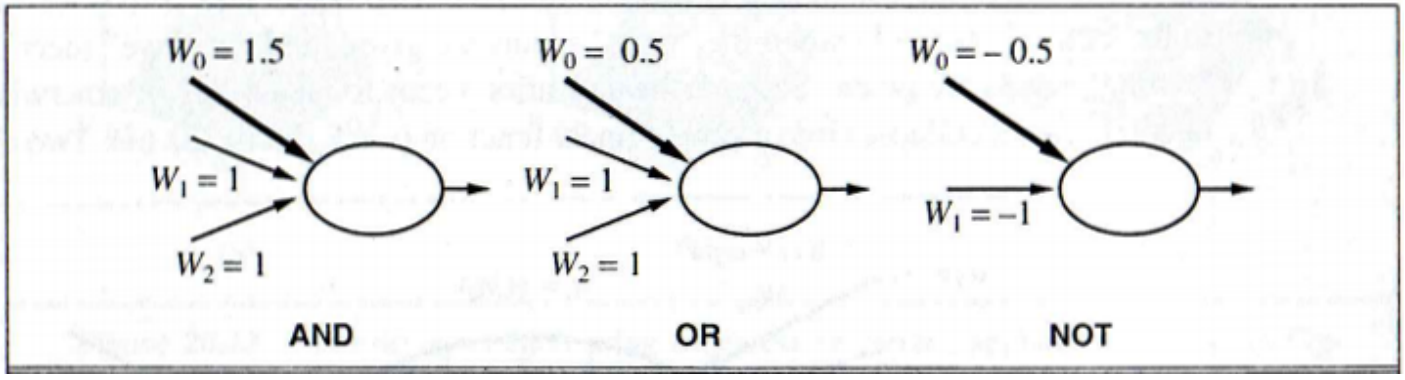
Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998).

- Así la ecuación $\mathbf{w} \cdot \mathbf{x} = 0$ define un hiperplano en el espacio de entrada, y el perceptron retorna 1 sólo si la entrada está de un lado del hiperplano.
- Explosiones nucleares: $-4.9 + 1.7x_1 - x_2 > 0$
- Terremotos: $-4.9 + 1.7x_1 - x_2 < 0$

Redes Neuronales (NN)

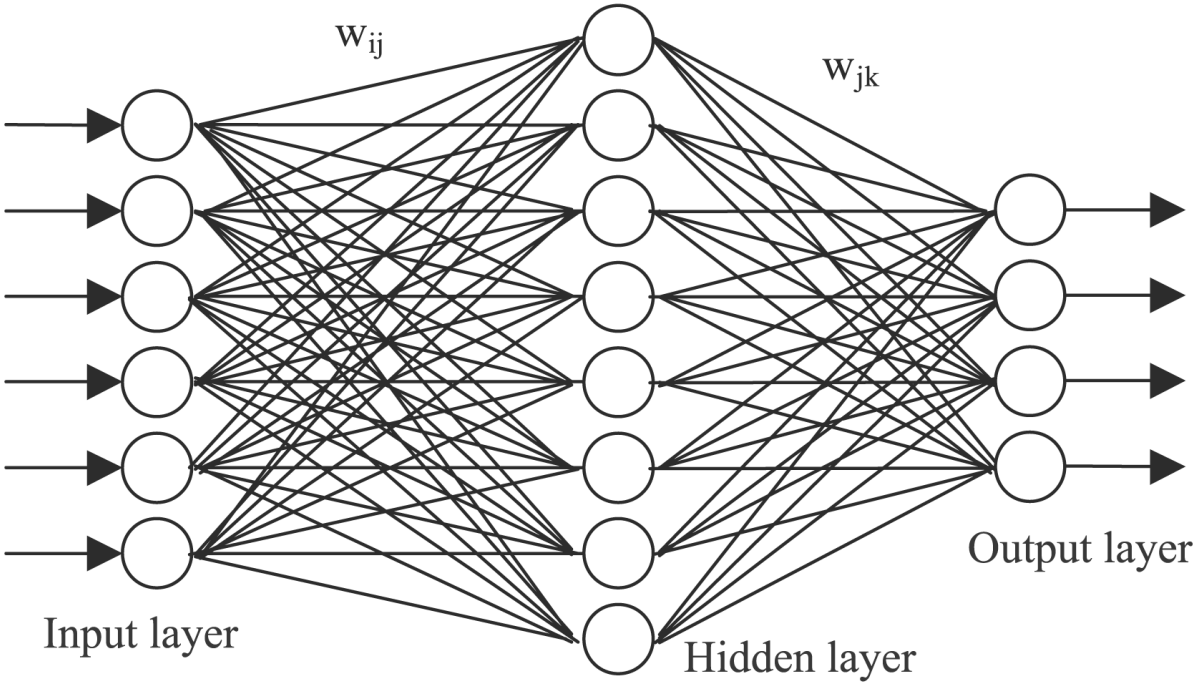
- Por esta razón los perceptrones son llamados separadores lineales.





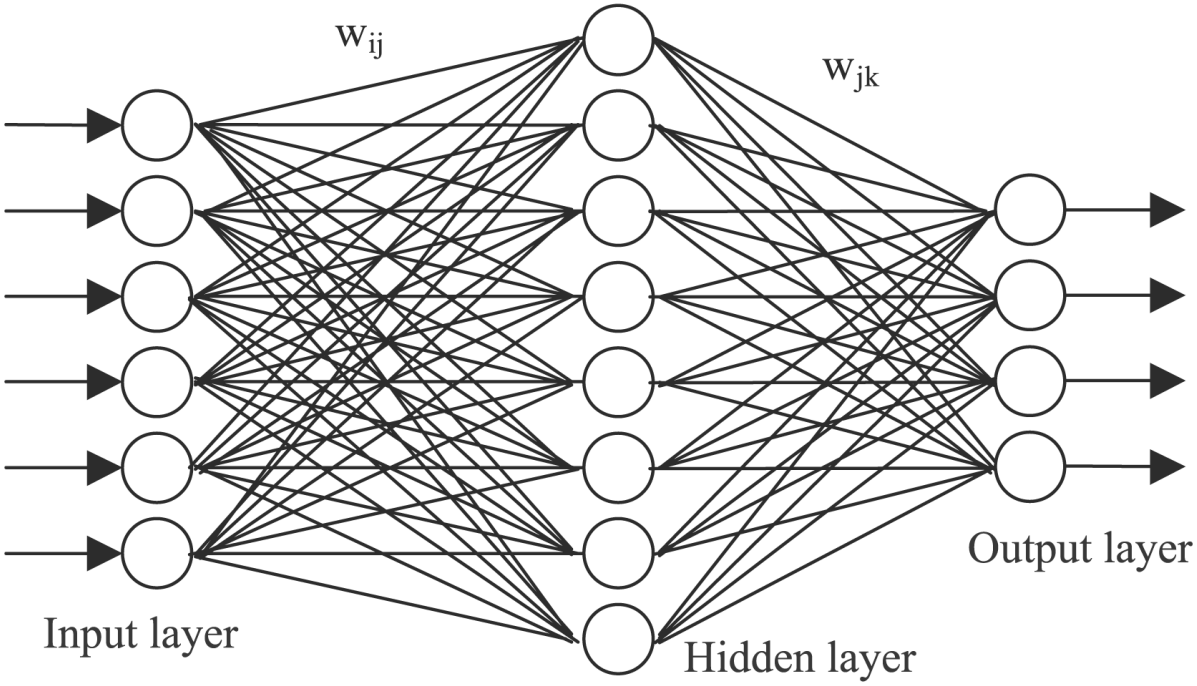
Redes Neuronales (NN)

- Una red feed-forward tiene conexiones en una sola dirección, es decir que es un grafo dirigido acíclico.

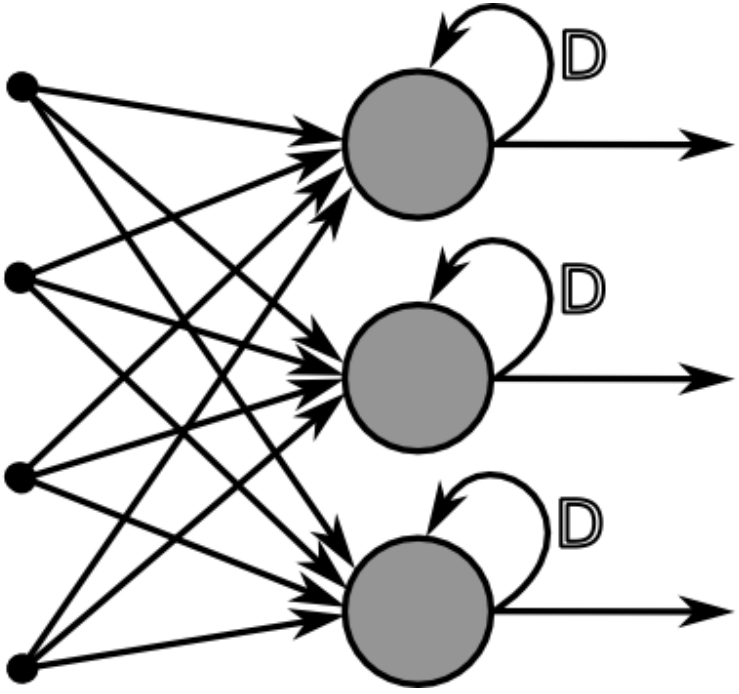


Redes Neuronales (NN)

- Las NN feed-forward se organizan en capas, donde las unidades reciben sus entradas sólo de la capa precedente.



Redes Neuronales (NN)



- Las NN feed-forward representan funciones en base a sus entradas; es decir que no tienen un estado interno, más que los pesos de los links.
- Una NN recurrente además conecta sus salidas a sus entradas, por lo que pueden soportar memoria de corto plazo.

Estructura de una NN

- Una neurona de entrada por cada componente x_i del vector de entrada $\mathbf{x}$.
- Es usual tener sólo una capa oculta. La ventaja de agregar capas ocultas es que se agranda el espacio de hipótesis que la NN puede representar.
- De hecho, con una sola capa oculta, lo suficientemente grande, es posible representar cualquier función continua arbitraria de las entradas de la NN.
- No existe una regla general para determinar la cantidad de unidades ocultas en una capa.

Estructura de una NN

- Si la NN es demasiado grande, memorizará todos los ejemplos (sobre-ajuste) y no generalizará bien para entradas que no haya visto en el conjunto de entrenamiento.
- Dado conocimiento del dominio que se tenga, conviene probar diferentes topologías y utilizar validación cruzada para obtener aquella configuración con menor error de predicción sobre los conjuntos de validación.
- Se pueden usar modelos donde las unidades no estén totalmente conectadas entre sí: e.g. algoritmo optimal-brain damage.

Tareas para las que se puede usar una NN

- Una NN puede utilizarse para clasificación o regresión.
- Para aprendizaje de conceptos es usual tener una unidad de salida e interpretar una clase como los valores mayores a 0.5 y la otra como los valores menores a 0.5.
- Para clasificación multiclase, se pueden dividir los valores de una única neurona de salida en k rangos, aunque es más común tener una neurona de salida por cada clase e interpretar cada valor de salida de las mismas como la probabilidad de pertenencia del elemento clasificado a la clase.

Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j} a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j} a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

([0.35,0.9], 1)

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

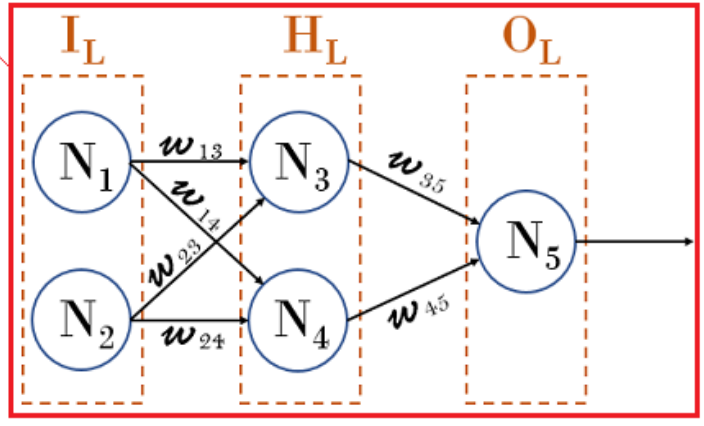
  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

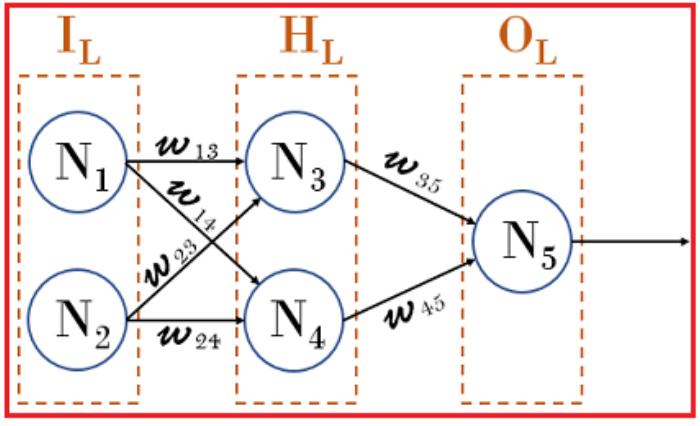
$$Logistic(z) = \frac{1}{1 + e^{-z}}$$



Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

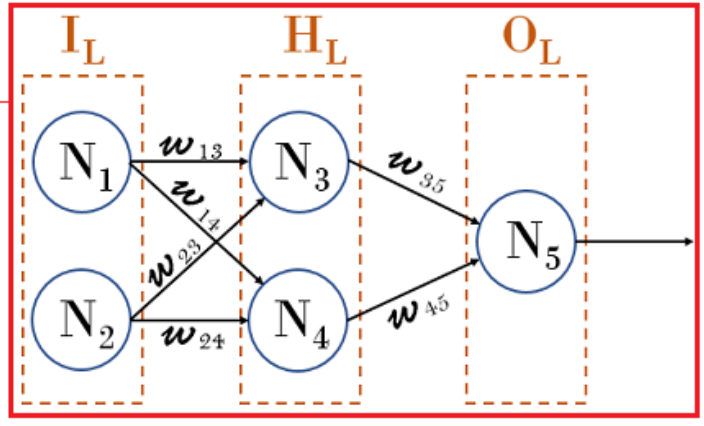
repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*



Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

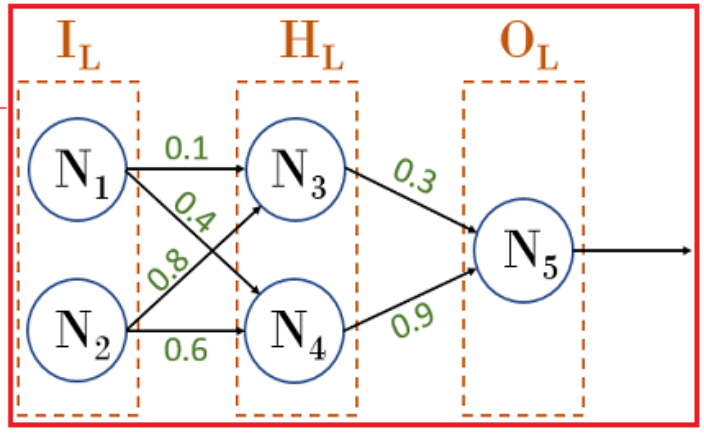
  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number



Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

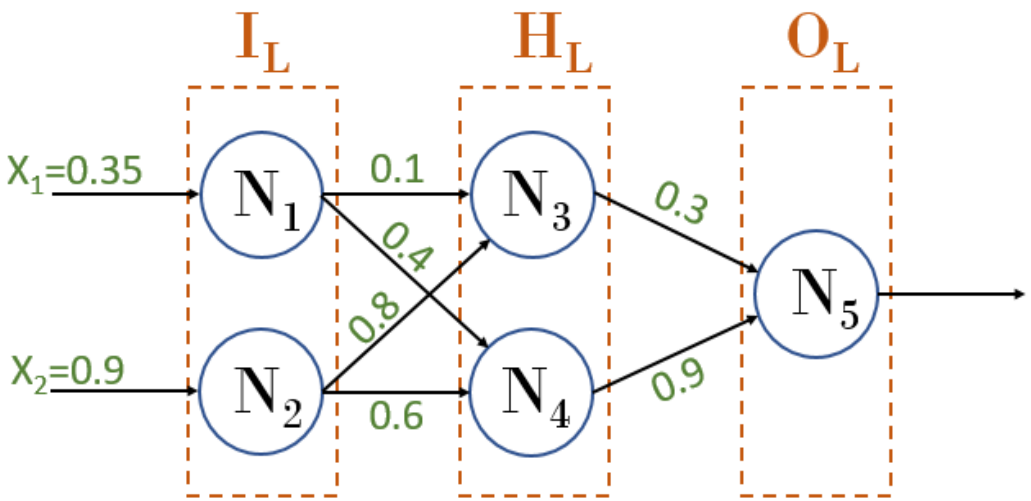
  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number



Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

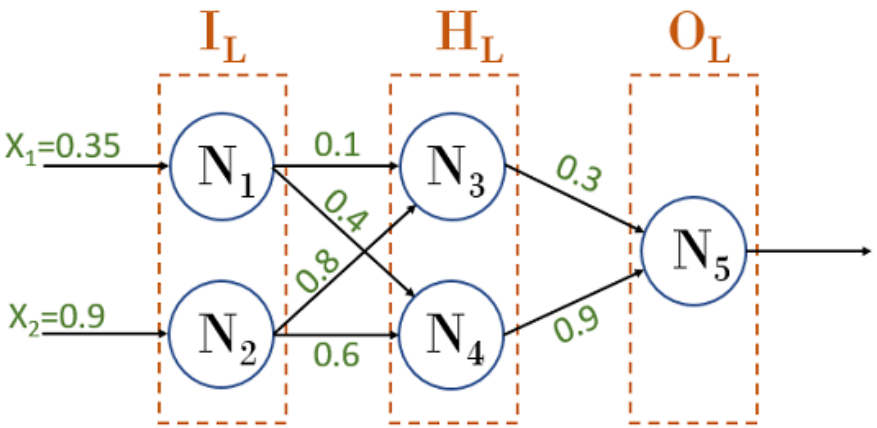
repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$



Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\ a_i$
          $a_j \leftarrow g(in_j)$



Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$

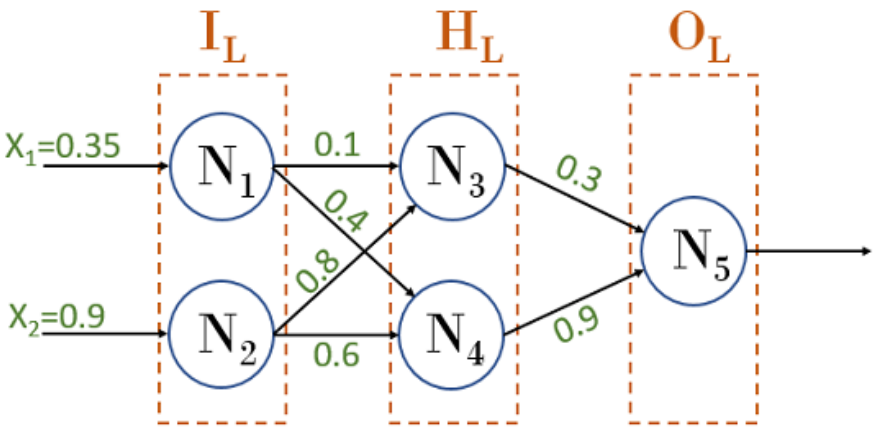
$\ell = 2$

$in_3 = (0.1 \times 0.35) + (0.8 \times 0.9) = 0.755$

$a_3 = g(in_3) = 1 / (1 + e^{-0.755}) = 0.68$

$in_4 = (0.4 \times 0.35) + (0.6 \times 0.9) = 0.68$

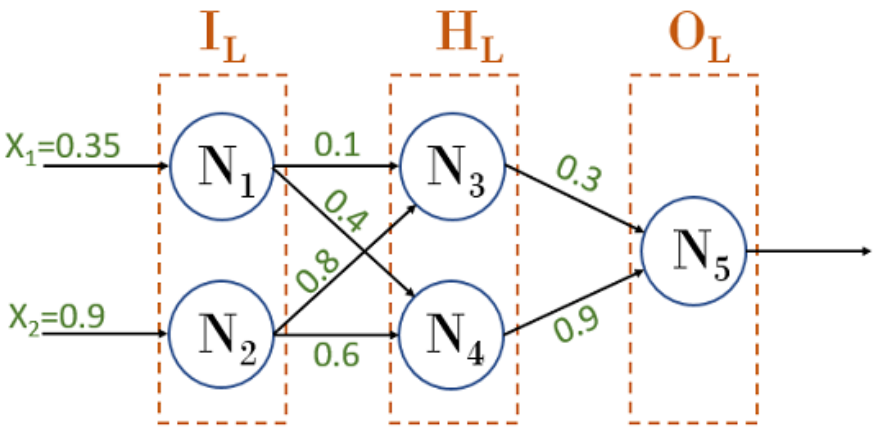
$a_4 = g(in_4) = 1 / (1 + e^{-0.68}) = 0.664$



Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$



$\ell = 2$

$in_3 = (0.1 \times 0.35) + (0.8 \times 0.9) = 0.755$

$a_3 = g(in_3) = 1 / (1 + e^{-0.755}) = 0.68$

$in_4 = (0.4 \times 0.35) + (0.6 \times 0.9) = 0.68$

$a_4 = g(in_4) = 1 / (1 + e^{-0.68}) = 0.664$

$\ell = 3$

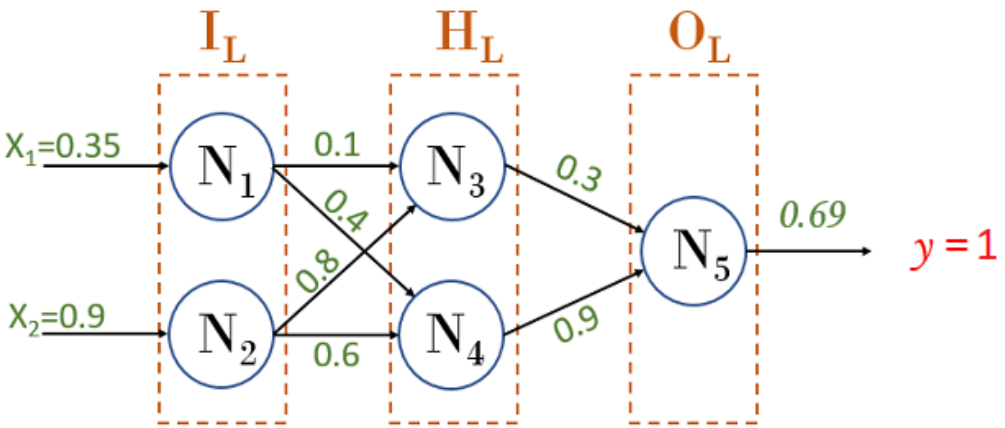
$in_5 = (0.3 \times 0.68) + (0.9 \times 0.664) = 0.802$

$a_5 = g(in_5) = 1 / (1 + e^{-0.802}) = 0.69$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\ a_i$
          $a_j \leftarrow g(in_j)$



Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j} a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

$$Logistic(z) = \frac{1}{1 + e^{-z}}$$

$g'(z) = g(z)(1-g(z))$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

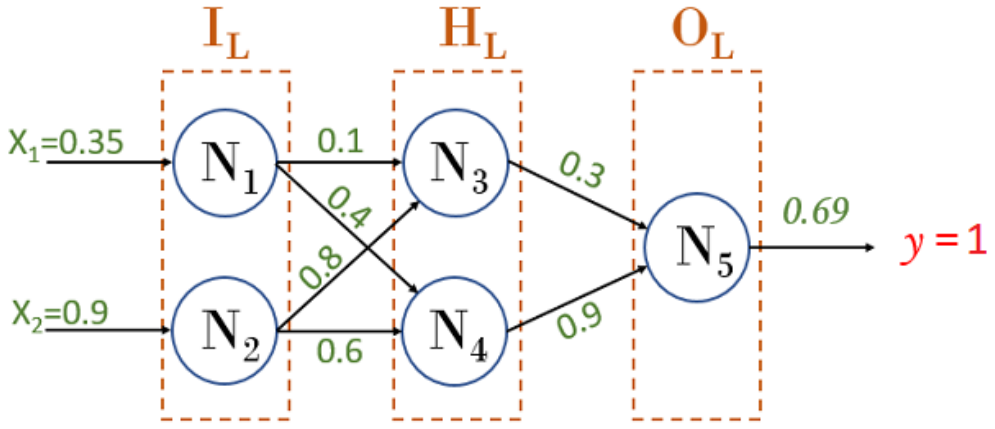
$$g'(z) = g(z)(1-g(z))$$

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

$in_5 = (0.3 \times 0.68) + (0.9 \times 0.664) = 0.802$

$\Delta[5] = g'(in_5) \times (1-0.69)$

$\Delta[5] = 0.69 \times (1 - 0.69) \times (1-0.69) = 0.066$



Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

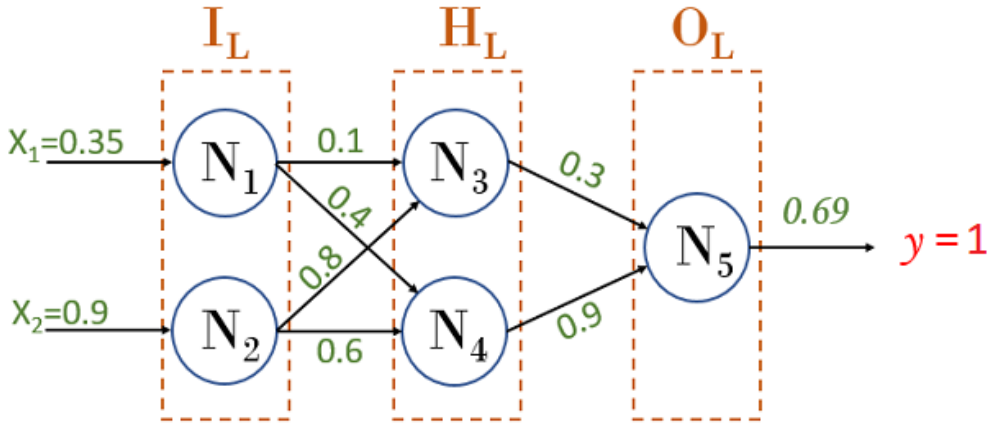
$$g'(z) = g(z)(1-g(z))$$

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$

$in_5 = (0.3 \times 0.68) + (0.9 \times 0.664) = 0.802$

$\Delta[5] = g'(in_5) \times (1 - 0.69)$

$\Delta[5] = 0.69 \times (1 - 0.69) \times (1 - 0.69) = 0.066$

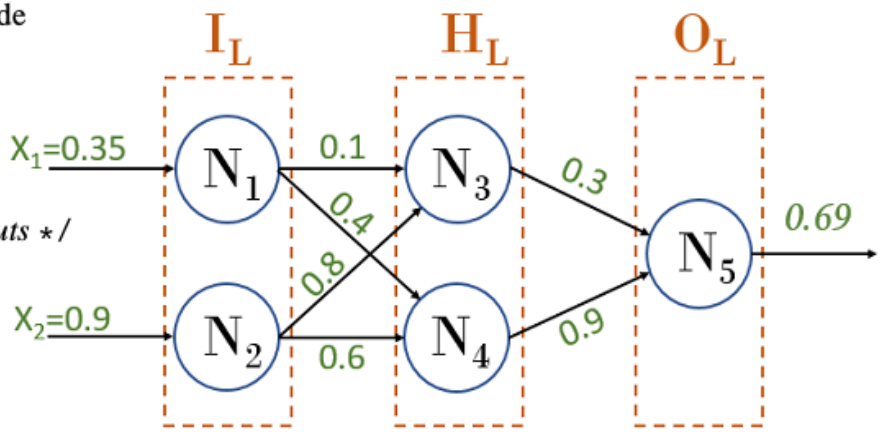


Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$

$\Delta[5] = 0.066$



$\ell = 2$

$in_3 = (0.1 \times 0.35) + (0.8 \times 0.9) = 0.755$

$a_3 = g(in_3) = 1 / (1 + e^{-0.755}) = 0.68$

$in_4 = (0.4 \times 0.35) + (0.6 \times 0.9) = 0.68$

$a_4 = g(in_4) = 1 / (1 + e^{-0.68}) = 0.664$

$\Delta[3] = g'(in_3) \times (0.3 \times (0.066))$

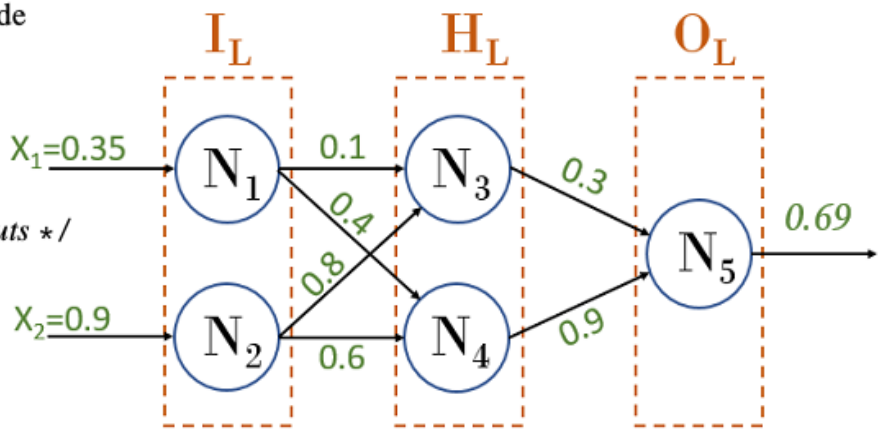
$\Delta[3] = 0.68 \times (1 - 0.68) \times (0.3 \times (0.066)) = 0.004$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$

$\Delta[5] = 0.066$



$\ell = 2$

$in_3 = (0.1 \times 0.35) + (0.8 \times 0.9) = 0.755$

$a_3 = g(in_3) = 1 / (1 + e^{-0.755}) = 0.68$

$in_4 = (0.4 \times 0.35) + (0.6 \times 0.9) = 0.68$

$a_4 = g(in_4) = 1 / (1 + e^{-0.68}) = 0.664$

$\Delta[3] = g'(in_3) \times (0.3 \times (0.066))$

$\Delta[3] = 0.68 \times (1 - 0.68) \times (0.3 \times (0.066)) = 0.004$

$\Delta[4] = g'(in_4) \times (0.9 \times (0.066))$

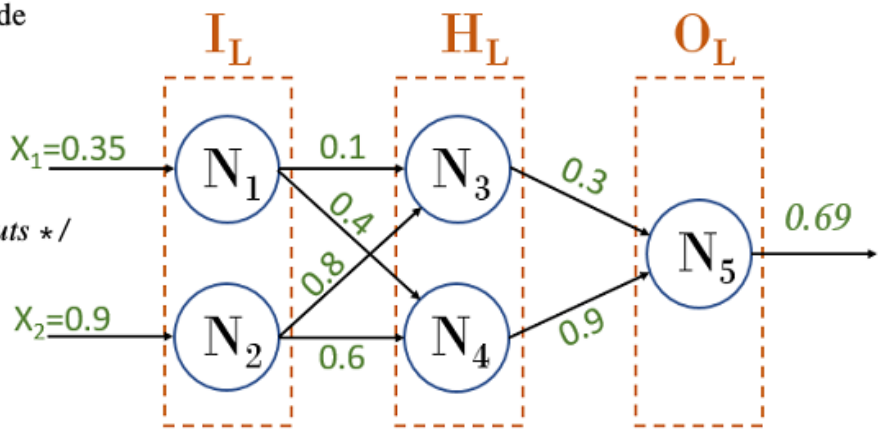
$\Delta[4] = 0.664 \times (1 - 0.664) \times (0.9 \times (0.066)) = 0.013$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$\Delta[5] = 0.066$



$\Delta[3] = g'(in_3) \times (0.3 \times (0.066))$

$\Delta[3] = 0.68 \times (1 - 0.68) \times (0.3 \times (0.066)) = 0.004$

$\Delta[4] = g'(in_4) \times (0.9 \times (0.066))$

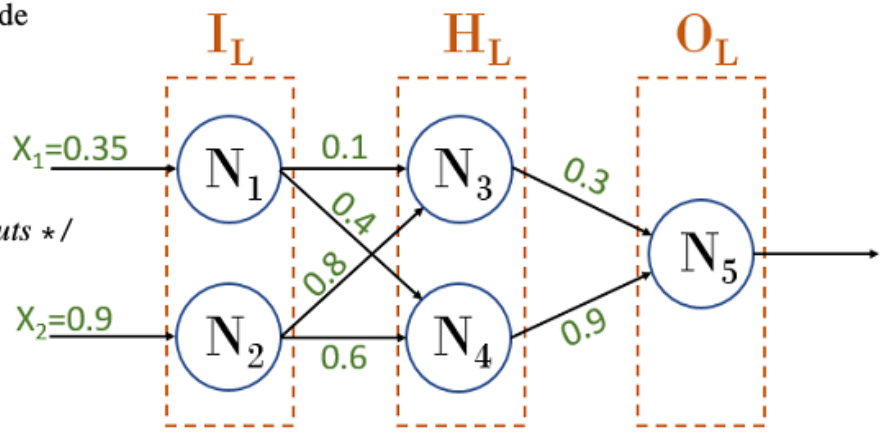
$\Delta[4] = 0.664 \times (1 - 0.664) \times (0.9 \times (0.066)) = 0.013$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$\Delta[4] = 0.013$
$\Delta[3] = 0.004$
$\Delta[5] = 0.066$



$a_1 = 0.35$ $a_2 = 0.9$ $a_3 = 0.68$ $a_4 = 0.664$

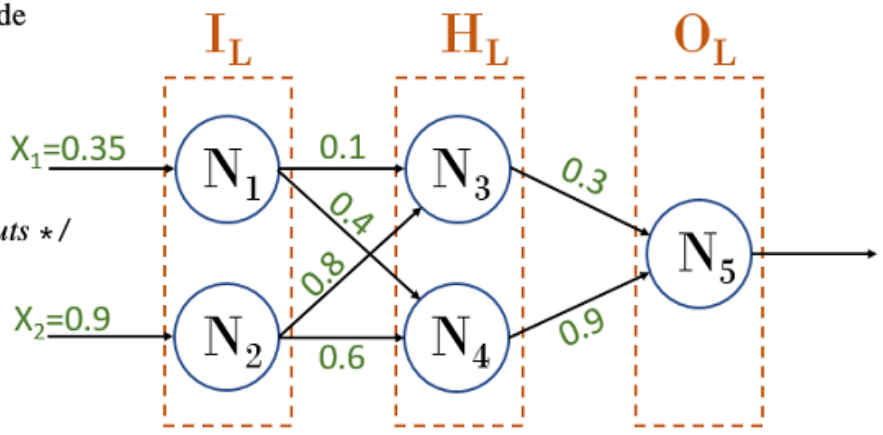
$w_{1,3} = 0.1 + 1 \times 0.35 \times 0.004 = 0.1014$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

$\Delta[4] = 0.013$
$\Delta[3] = 0.004$
$\Delta[5] = 0.066$



$a_1 = 0.35$ $a_2 = 0.9$ $a_3 = 0.68$ $a_4 = 0.664$

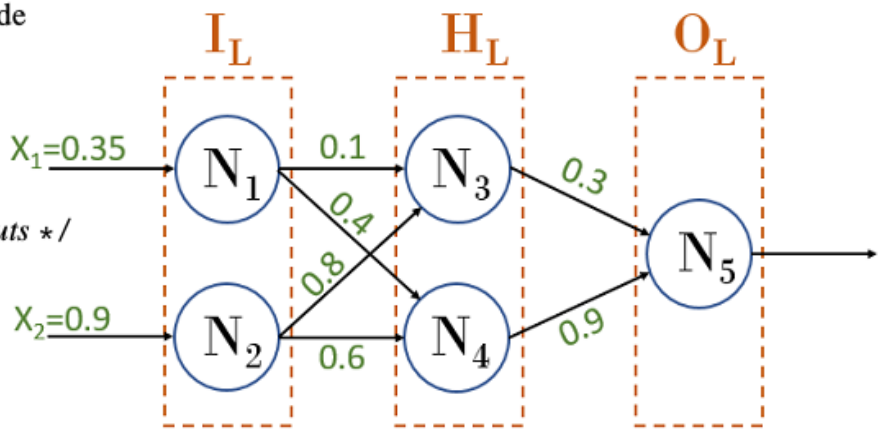
$w_{1,3} = 0.1 + 1 \times 0.35 \times 0.004 = 0.1014$
$w_{1,4} = 0.4 + 1 \times 0.35 \times 0.013 = 0.405$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

$\Delta[4] = 0.013$
$\Delta[3] = 0.004$
$\Delta[5] = 0.066$



$a_1 = 0.35$ $a_2 = 0.9$ $a_3 = 0.68$ $a_4 = 0.664$

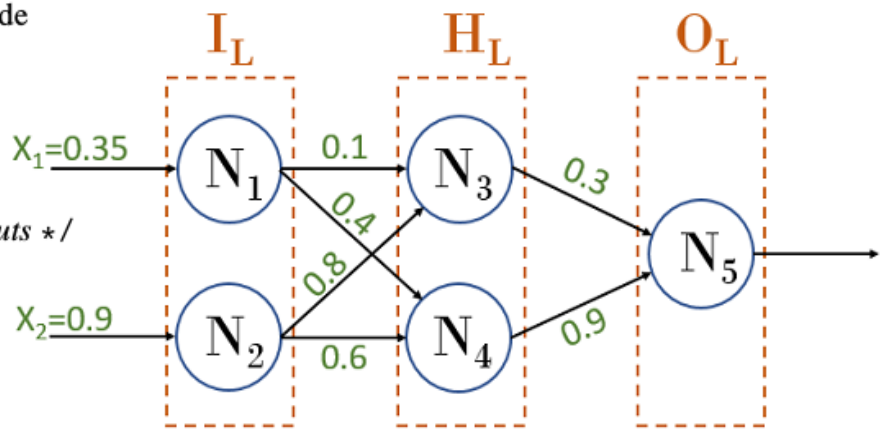
$w_{1,3} = 0.1 + 1 \times 0.35 \times 0.004 = 0.1014$
$w_{1,4} = 0.4 + 1 \times 0.35 \times 0.013 = 0.405$
$w_{2,3} = 0.8 + 1 \times 0.9 \times 0.004 = 0.803$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

$\Delta[4] = 0.013$
$\Delta[3] = 0.004$
$\Delta[5] = 0.066$



$a_1 = 0.35$ $a_2 = 0.9$ $a_3 = 0.68$ $a_4 = 0.664$

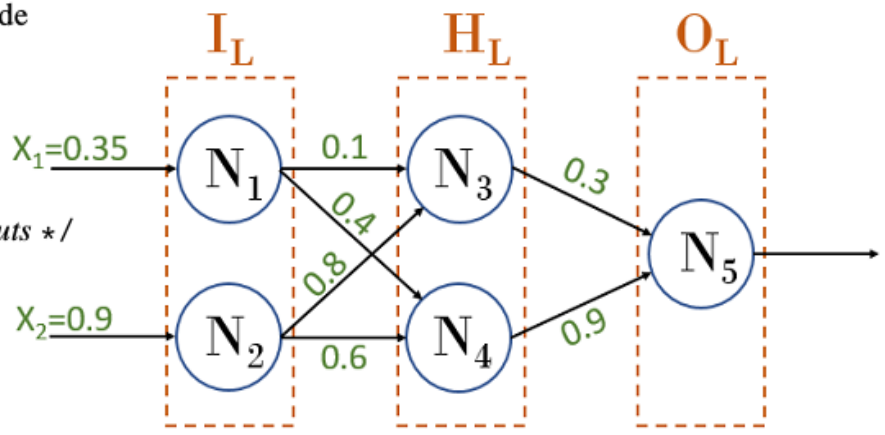
$w_{1,3} = 0.1 + 1 \times 0.35 \times 0.004 = 0.1014$
$w_{1,4} = 0.4 + 1 \times 0.35 \times 0.013 = 0.405$
$w_{2,3} = 0.8 + 1 \times 0.9 \times 0.004 = 0.803$
$w_{2,4} = 0.6 + 1 \times 0.9 \times 0.013 = 0.611$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

$\Delta[4] = 0.013$
$\Delta[3] = 0.004$
$\Delta[5] = 0.066$



$a_1 = 0.35$ $a_2 = 0.9$ $a_3 = 0.68$ $a_4 = 0.664$

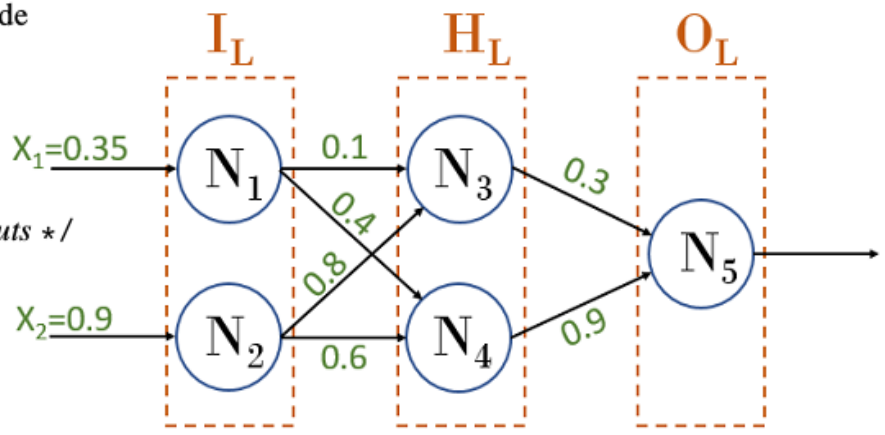
$w_{1,3} = 0.1 + 1 \times 0.35 \times 0.004 = 0.1014$
$w_{1,4} = 0.4 + 1 \times 0.35 \times 0.013 = 0.405$
$w_{2,3} = 0.8 + 1 \times 0.9 \times 0.004 = 0.803$
$w_{2,4} = 0.6 + 1 \times 0.9 \times 0.013 = 0.611$
$w_{3,5} = 0.3 + 1 \times 0.68 \times 0.066 = 0.35$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$\Delta[4] = 0.013$
$\Delta[3] = 0.004$
$\Delta[5] = 0.066$



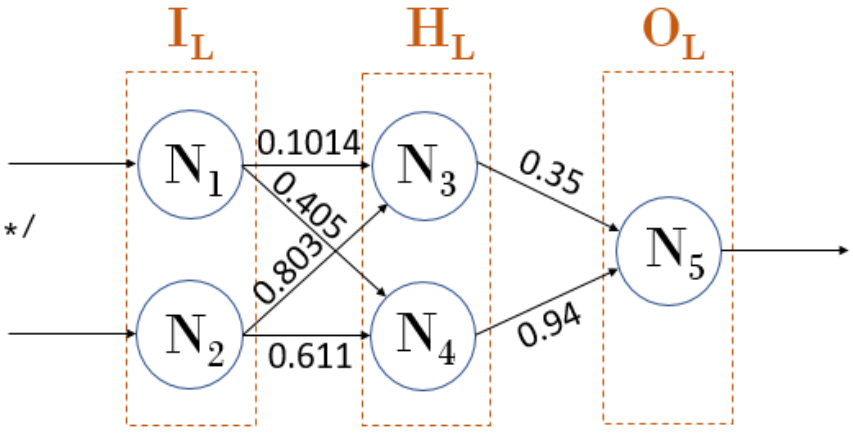
$a_1 = 0.35$ $a_2 = 0.9$ $a_3 = 0.68$ $a_4 = 0.664$

$w_{1,3} = 0.1 + 1 \times 0.35 \times 0.004 = 0.1014$
$w_{1,4} = 0.4 + 1 \times 0.35 \times 0.013 = 0.405$
$w_{2,3} = 0.8 + 1 \times 0.9 \times 0.004 = 0.803$
$w_{2,4} = 0.6 + 1 \times 0.9 \times 0.013 = 0.611$
$w_{3,5} = 0.3 + 1 \times 0.68 \times 0.066 = 0.35$
$w_{4,5} = 0.9 + 1 \times 0.664 \times 0.066 = 0.94$

Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*



Algoritmo “Back-Propagation”

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$$w_{i,j} \leftarrow w_{i,j} + \delta_{i,j}$$

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

Adding Momentum

$$w_{i,j} \leftarrow w_{i,j} + \delta_{i,j}(n)$$

$n =$ iteration number

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$$\delta_{i,j}(n) = \alpha \times a_i \times \Delta[j] + \eta \delta_{i,j}(n-1)$$

Adding Momentum

$$w_{i,j} \leftarrow w_{i,j} + \delta_{i,j}(n)$$

$n =$ iteration number

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\ a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\ \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$$0 \leq \eta < 1$$

$$\delta_{i,j}(n) = \alpha \times a_i \times \Delta[j] + \eta \delta_{i,j}(n-1)$$

Adding Momentum

$$w_{i,j} \leftarrow w_{i,j} + \delta_{i,j}(n)$$

$n =$ iteration number

Algoritmo "Back-Propagation"

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 $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\ a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\ \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$$\alpha = \frac{1}{e}$$

$$e = \text{epoch number}$$

Referencias

- **Jurafsky D. and Martin J.H.** *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2018) 3rd Edition. Capítulo 7: “Neural Networks and Neural Language Models”.
- **Müller A.C. and Guido S.** *Introduction to Machine Learning with Python: A Guide for Data Scientists* (2016) 1st Edition. Capítulo 2: “Supervised Learning”, pp. 104-118.
- **Russell S. and Norvig P.** *Artificial Intelligence: A Modern Approach* (2010) 3rd Edition. Sección 18.7: “Artificial Neural Networks”.
- **Mitchel T.** *Machine Learning* (1997) 1st Edition. Capítulo 4: “Artificial Neural Networks”.
- `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`
- `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`