CURSOS PIT

CTIC UNI

# Fundamentos de Inteligencia Artificial

CURSOS PIT

CTIC UNI

# PhD Wester Edison Zela Moraya

PhD en Computer Science – Inteligencia Artificial por la Universidad Politécnica de Madrid. Master en Ingeniería de Software por la Universidad de Oxford. Master en Análisis Financiero y Económico por la Universidad Complutense de Madrid. Ingeniero de Sistemas de la UNI.

Amplia experiencia profesional en Transformación Digital, Machine Learning, RPAs, Data Science, Metodologías Ágiles, Microservices, gestión económica de proyectos. Docente de Inteligencia Artificial en la Universidad Nacional de Ingeniería.

Director de TI en empresas en Peru y Europa

Consultor de IA y Datos en la SGTD en la PCM

Miembro del AI Connect Program (US Department y Atlantic Council)

Creador de Troomes.com

# Temas – Sesion 5

- Redes Neuronales Artificiales
- Simple Perceptron
- MultiLayer Perceptron

# Algoritmos de Aprendizaje Supervisado

Algunos algoritmos en el aprendizaje supervisado:

–Arboles de Decisión

–Random Forest

–Redes Bayesianas

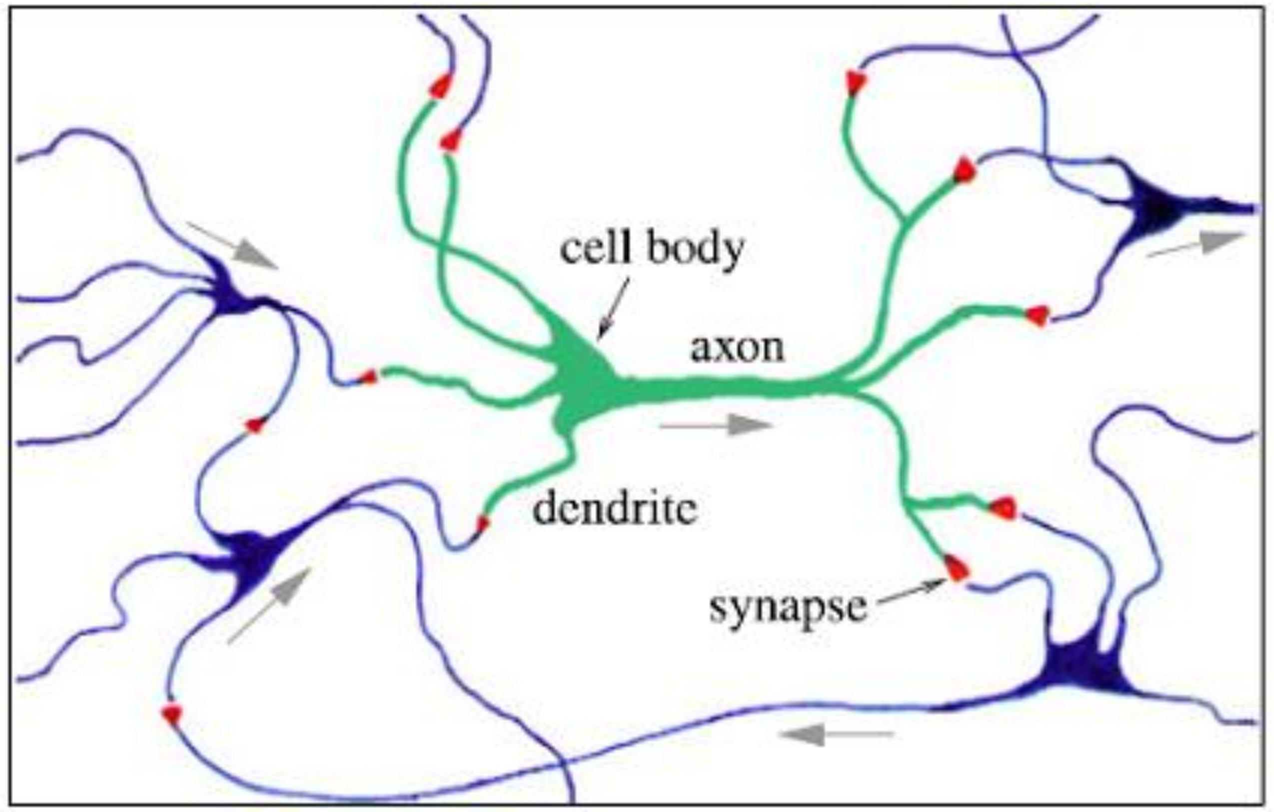
–Máquinas de Vectores de Soporte (SVM)

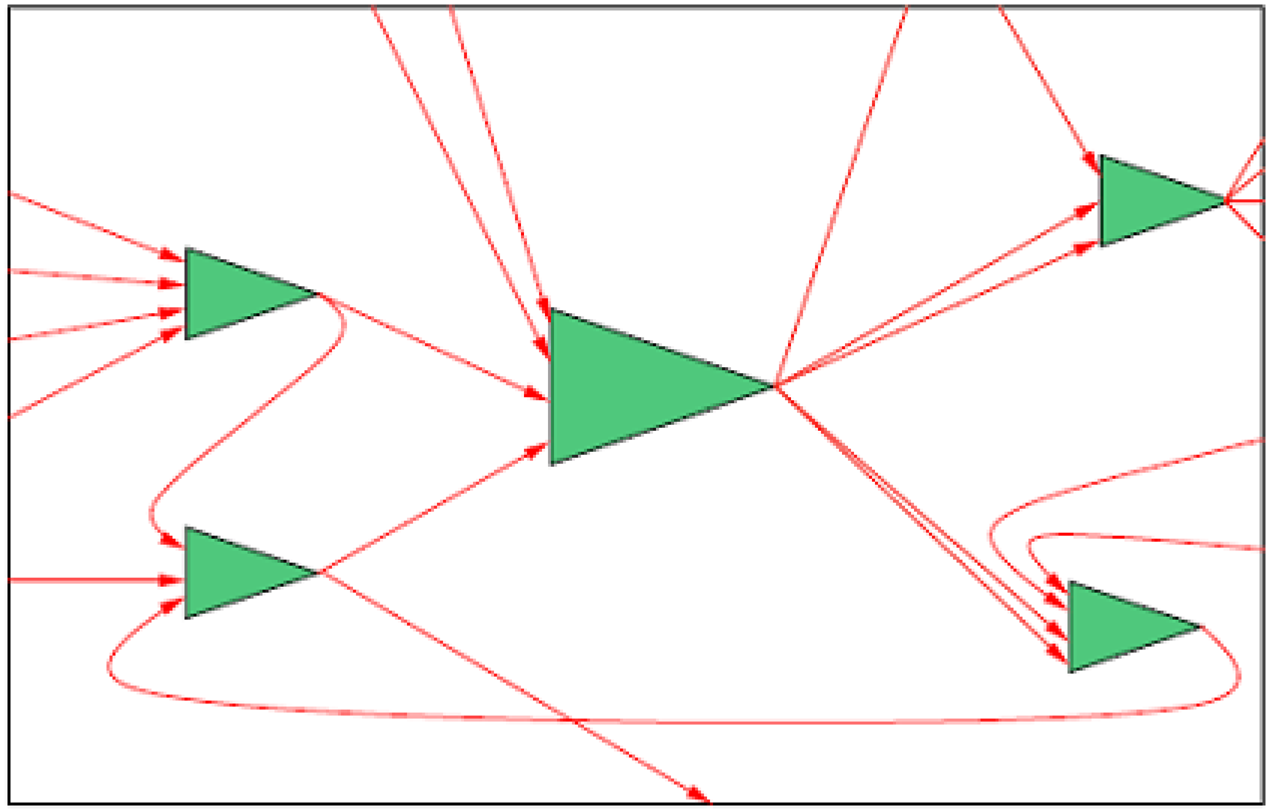
–**Red Neuronal Artificial**

–Aprendizaje profundo (Deep Learning)

# Artificial Neural Network

- *Artificial neural networks* are essentially modeled on the parallel architecture of animal brains, not necessarily human ones. The network is based on a simple form of inputs and outputs.
- From Biology to Simulation



# Artificial Neural Network

- The first model of a simplified brain cell was published in 1943 and is known as the McCullock-Pitts (MCP) neuron.
- Essentially, this is a basic logic gate with binary outputs (‘0’ or ‘1’). A ‘1’ is produced if the sum of inputs arriving at the neuron exceed a given threshold, where each input is multiplied by a corresponding weight coefficient to produce this sum.
- Frank Rosenblatt took the basic MCP neuron concept soon after its publication and produced the **Perceptron Rule Algorithm**

# Artificial Neural Network

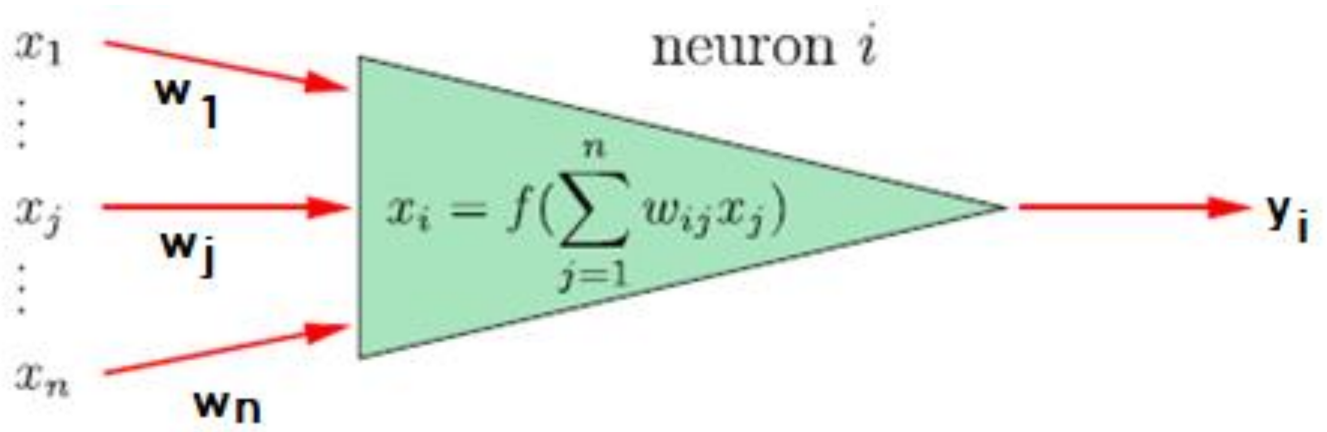
- General and practical method for supervised learning.
- Learn a function $f : X \rightarrow Y$, where it is not necessary to know its "form" a priori.
- They do not have a clear interpretability, they work like black boxes.
- It requires a lot of computational resources to be trained.
- The application of the model is efficient and requires few computational resources.
- The structure of a neural network is parallel, so if this is implemented on a cluster of computers, responses can be obtained in real time.

# Artificial Neural Network

- Some Applications of Artificial Neural Networks
  - Regression, Classification
  - Text mining classifications
  - Pattern Recognition ( i.e: handwriting recognition)
  - Images Classifications (CNN)
  - NLP
  - DeepLearning
  - Clustering

# Artificial Neural Network

- The Mathematical Model
  - The neuron *i* apply the activitation funtion to all the inputs $x_1$ … $x_j$ … $x_n$




  - The sumation before apply the activation funtion (propagation rule)

$$h_i(x_1, \dots, x_n, w_{i,1}, \dots, w_{i,n})$$

  - Some propagation rules :

$$h_i(x_1, \dots, x_n, w_{i,1}, \dots, w_{i,n}) = \sum_{j=1}^{n} w_{ij}x_j.$$

# Artificial Neural Network

- The Mathematical Model
  - Sometime use an aditional parameter $\theta_i$, which is a threshold.

$$h_i(x_1, \dots, x_n, w_{i,1}, \dots, w_{i,n}) = \sum_{j=1}^{n} w_{i,j}\, x_j - \theta_i$$

  - The output of the neuron is $x_i$ after apply activation funtion ***f*** :

$$y_i = f(h_i) = f\left(\sum_{j=1}^{n} w_{i,j}\, x_j - \theta_i\right)$$

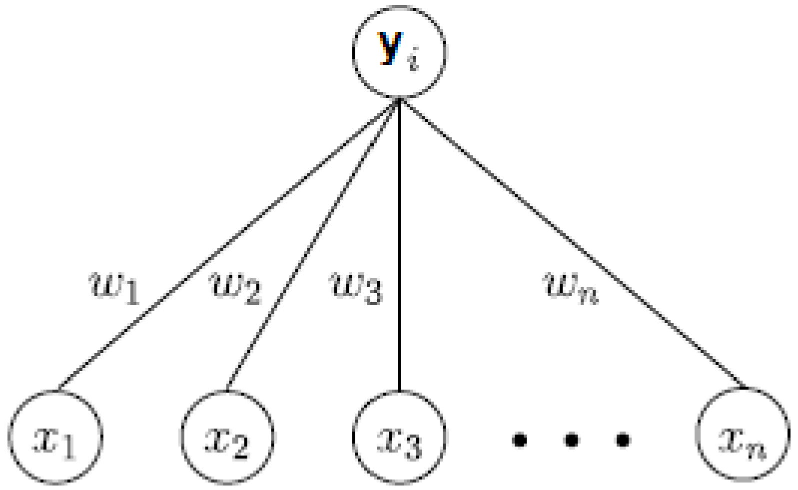
# Artificial Neural Network

- Output based on function using a threshold : $\Theta$

$$H_{\Theta}(x) = \begin{cases} 0 & \text{if } x < \Theta, \\ 1 & \text{else.} \end{cases}$$

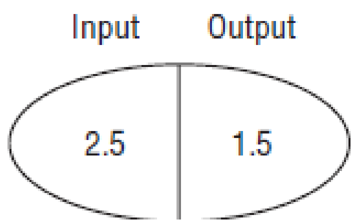
- Including the all input neuron

$$\mathbf{y}_i = \begin{cases} 0 & \text{if } \sum_{j=1}^{n} w_{ij} x_j < \Theta, \\ 1 & \text{else.} \end{cases}$$

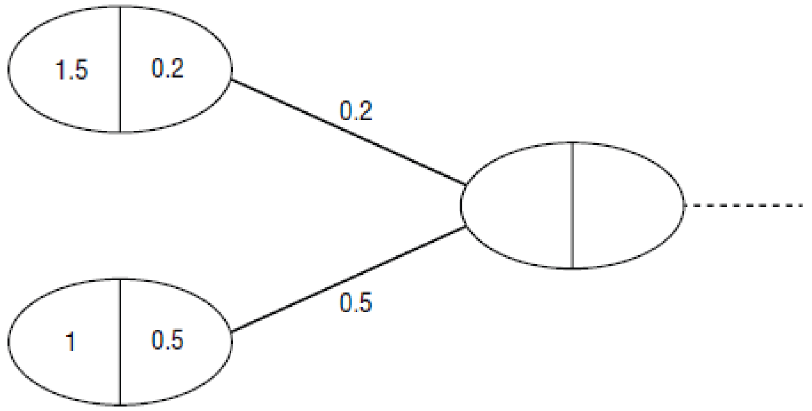


# Artificial Neural Network

- Simple Perceptron:




- Perceptron with two inputs:




- The sumation is Where $w_i$ is weigth, and $Z_i$ input values
  - Example: ( 1.5x 0.2 + 1x0.5)

- Output using threshold: $\sum_i wiZi$

$$
\begin{aligned}
&if \sum_i wiZi \geq t \ then \ y = 1 \\
&else \ y = 0
\end{aligned}
$$

# Artificial Neural Network

- Activation functions:

  - Linear Function :

$$F(\tau) = \beta\tau$$

  - Sigmoid Function :

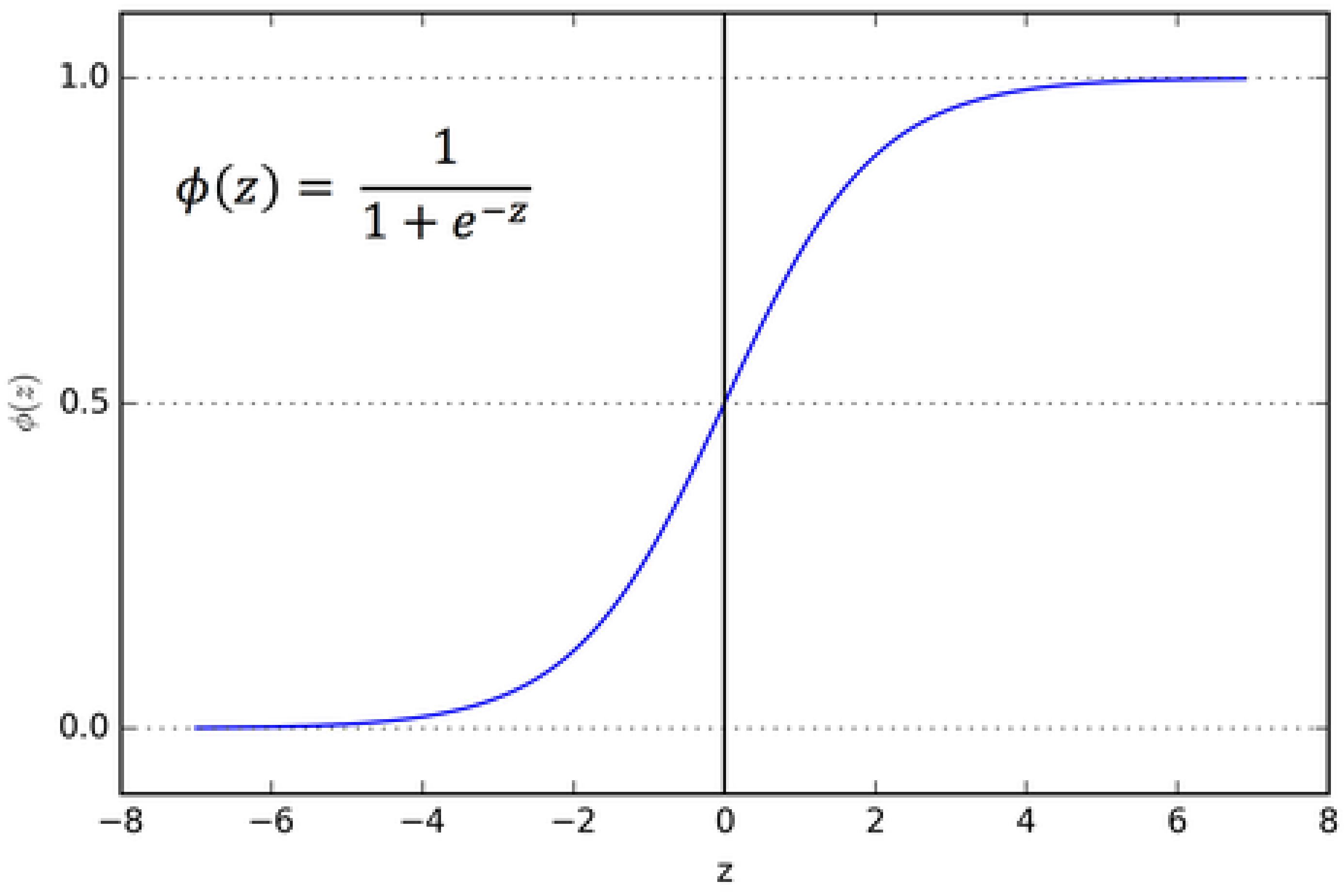
$$F(\tau) = \frac{1}{1 + e^{-\lambda\tau}}$$

  - Hyperbolic tangent Function :

$$F(\tau) = \frac{2}{1 + e^{-\lambda\tau}} - 1$$

  - Gaussian Function :

$$F(\mu) = e^{-\mu^2/\sigma^2}$$

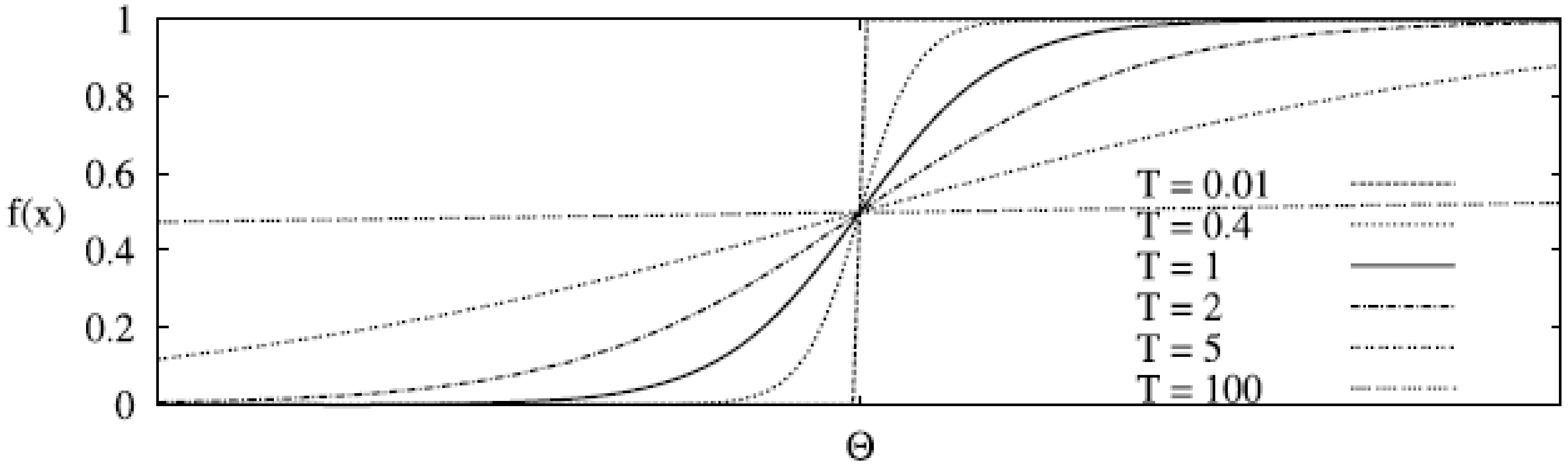


# Artificial Neural Network

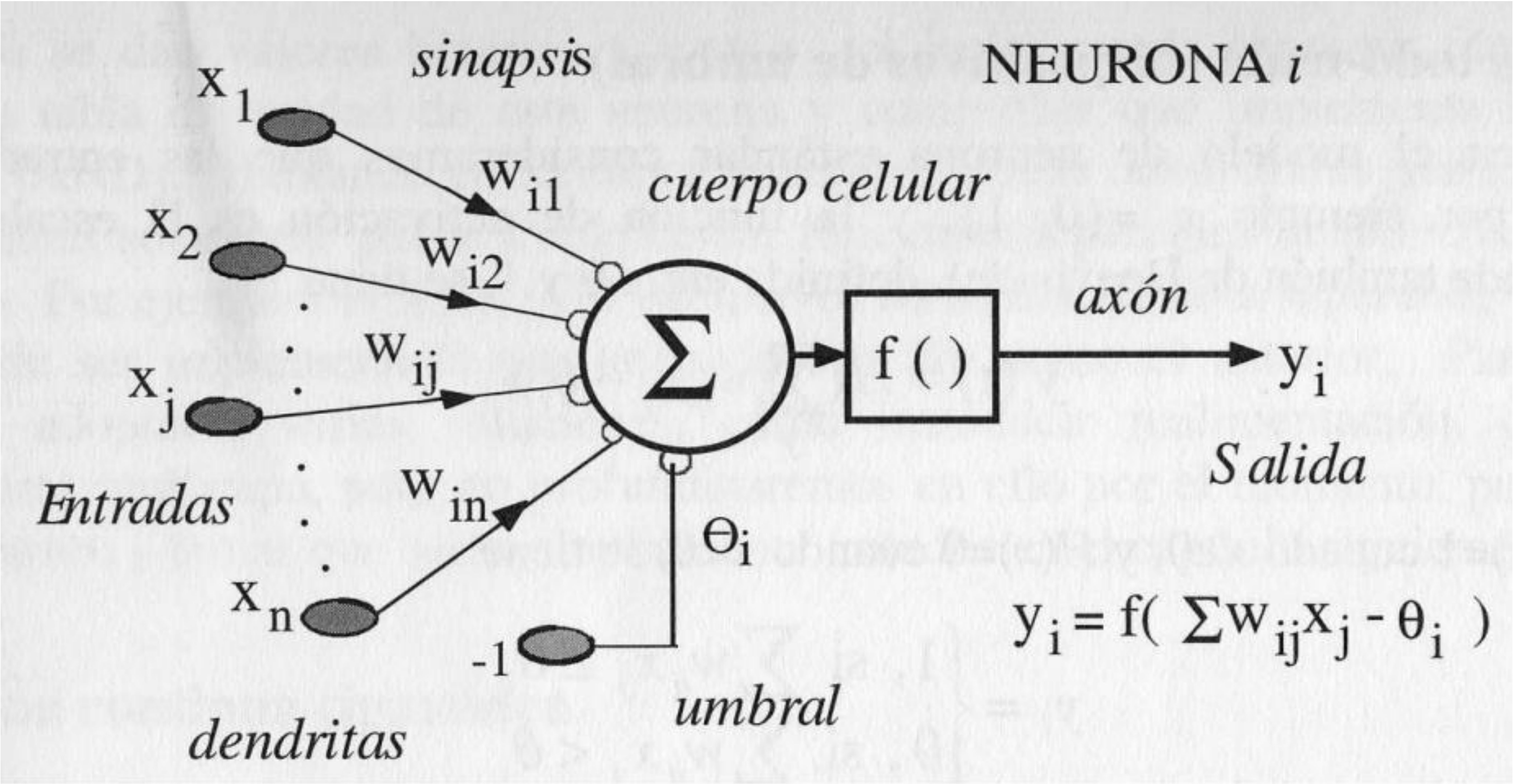
- Sigmoid Function:

$$f(x) = \frac{1}{1 + e^{-\frac{x-\Theta}{T}}}$$

- The Result using a threshold



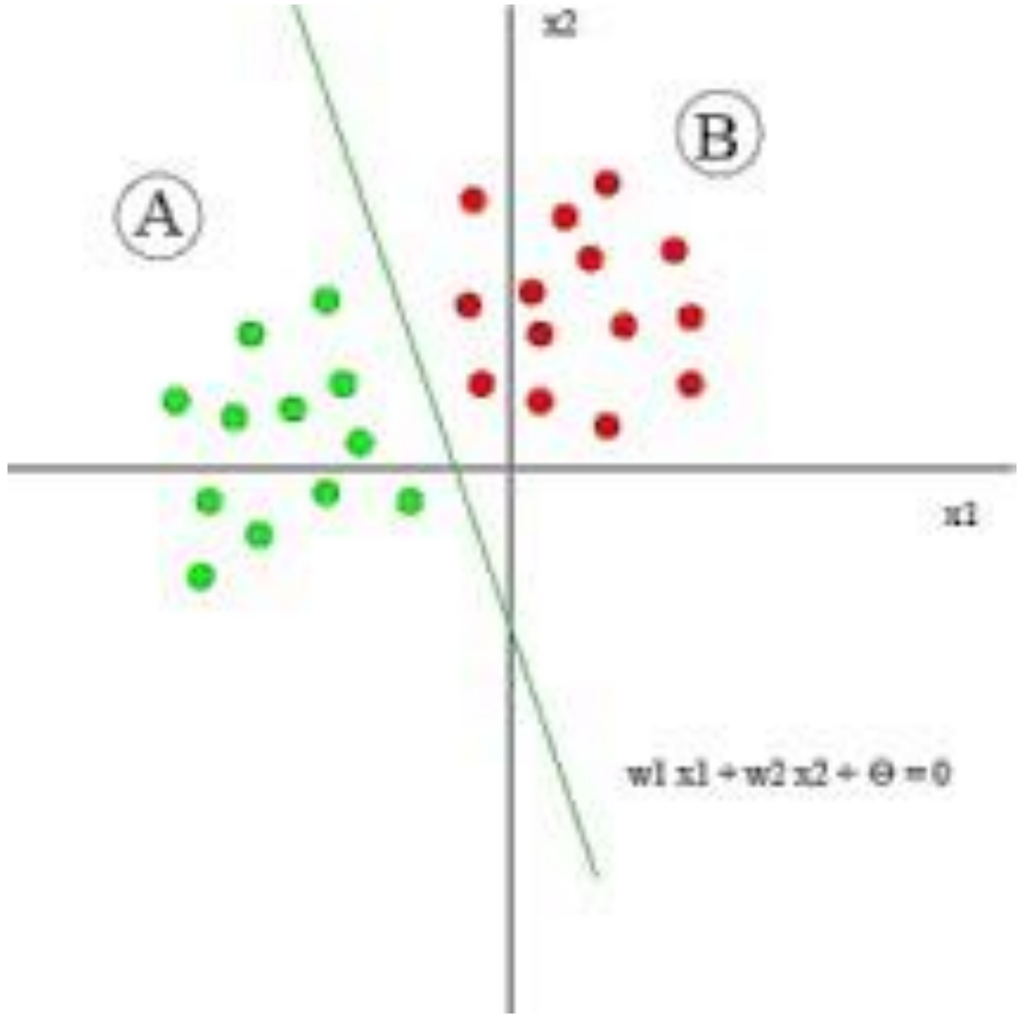
# Artificial Neural Network



# Perceptron Simple

- Fue introducido por Rosenblatt en 1962.
- Se concibió como un sistema capaz de realizar tareas de clasificación de forma automática.

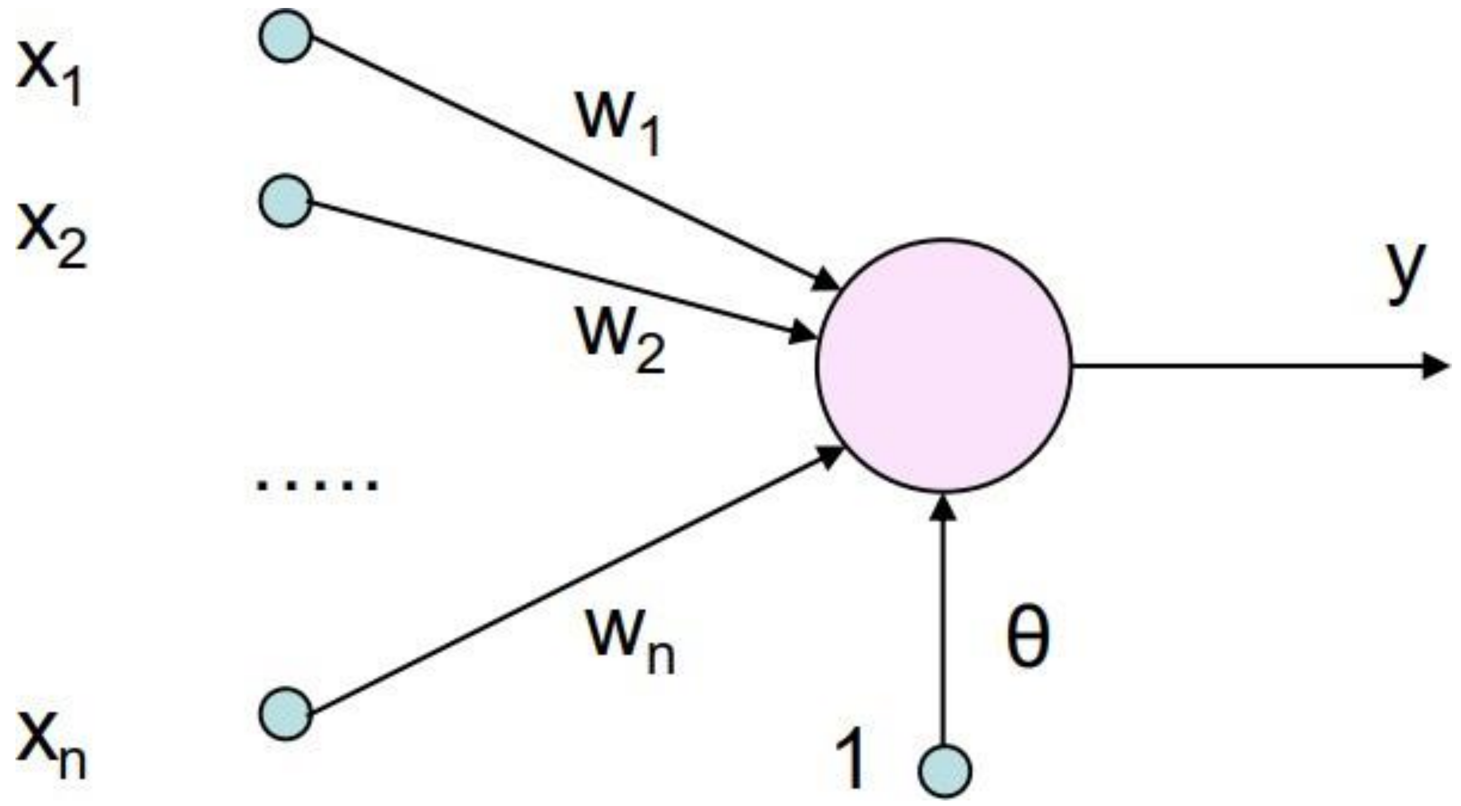
A partir de un número de elementos etiquetados, el sistema determina la ecuación del hiperplano discriminante.




$$w_1x_1 + w_2x_2 + \theta = 0$$

# Perceptron Simple: Arquitectura

Es un modelo unidireccional compuesto por dos capas de neuronas, una de entrada y otra de salida.



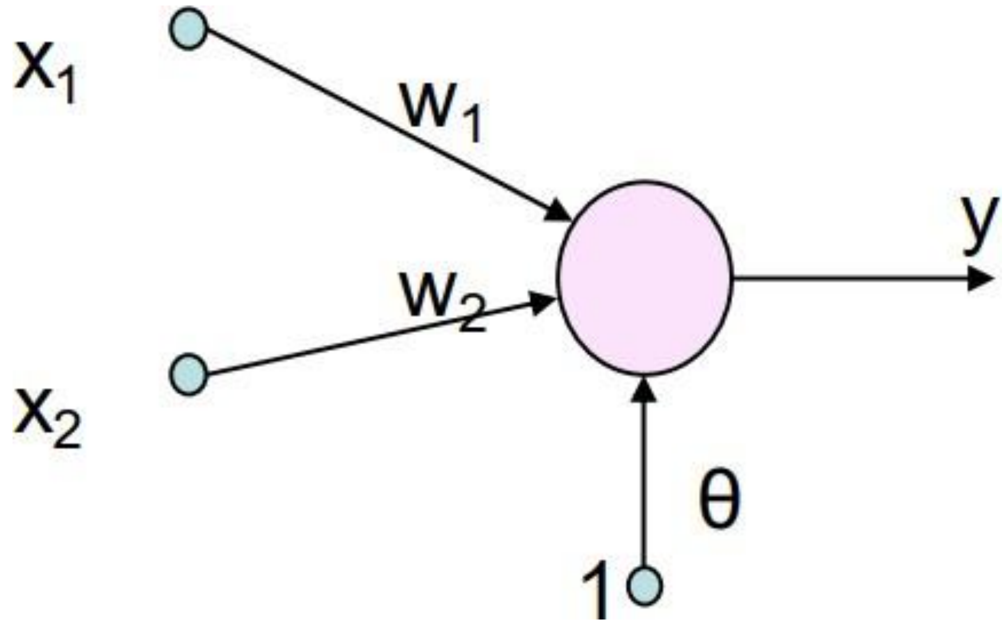

$$y = \begin{cases} +1, & si\ \ w_1x_1 + \cdots + w_nx_n + \theta > 0 \\ -1, & si\ \ w_1x_1 + \cdots + w_nx_n + \theta \leq 0 \end{cases}$$

# Perceptron Simple: Arquitectura

- El perceptron equivale a un hiperplano de dimensión $n - 1$ capaz de separar las clases
  - Si la salida del perceptron es +1, la entrada pertenecerá a una clase (estará situada a un lado del hiperplano)
  - Si la salida es -1, la entrada pertenecerá a la clase contraria (estará situada al otro lado del hiperplano)
- La ecuación del hiperplano es:

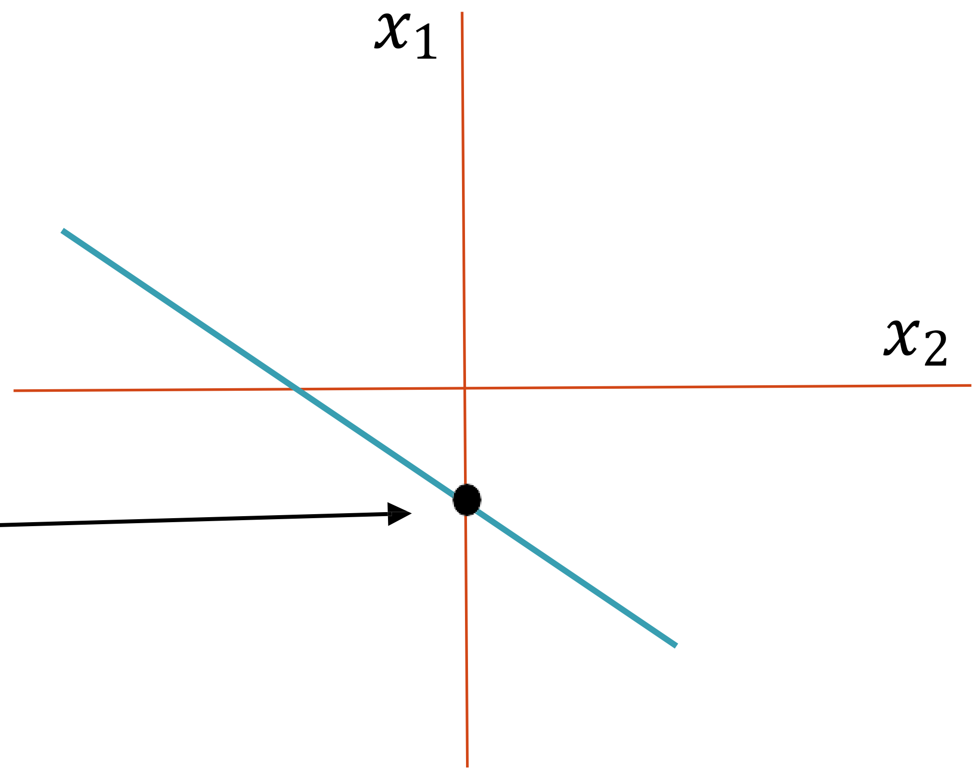
$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \theta = 0$$

# Perceptron Simple: Ejemplo 2 dimensiones




$$y = \begin{cases} +1, & si \;\; w_1x_1 + \cdots + w_nx_n + \theta > 0 \\ -1, & si \;\; w_1x_1 + \cdots + w_nx_n + \theta \leq 0 \end{cases}$$

La ecuación del hiperplano es: $w_1x_1 + \; w_2x_2 + \theta = 0$




$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{\theta}{w_2}$$

Punto de corte

Pendiente de la recta

# Perceptron Simple: Aprendizaje

- Se dispone de un conjunto de observaciones (patrones, ejemplos, datos) de los que se sabe su categoría o clase.
- Los ejemplos o datos son puntos en un espacio multidimensional:

$$\Re^n: (x_1, \dots, x_n)$$

- Hay que determinar la ecuación del hiperplano que deja a un lado los ejemplos de una clase y a otro lado los de la otra clase.
- La ecuación del hiperplano se deduce a partir de los ejemplos o datos.
- Proceso iterativo supervisado.

# Perceptron Simple: Aprendizaje

- Dado:
  - Conjunto de patrones
  - Vector de entrada: $(x_1, \ldots, x_n)$
  - Salida: $d(x)$

$$d(x) = +1 \quad si\ x \in A$$

$$d(x) = -1 \quad si\ x \in B$$

- Hiperplano discriminante:

$w( _1, \ldots, w_n, \theta \ )$ tales que

$$w_1 x_1 + \quad w_2 x_2 + \cdots + w_n x_n + \theta = 0$$

Separe las clases $A$ y $B$.

# Perceptron Simple: Aprendizaje

1. Comenzar con valores aleatorios para pesos y umbral
2. Modificación de los pesos y umbral hasta encontrar el hiperplano discriminante
   a. Seleccionar un ejemplo $x$ del conjunto de entrenamiento
   b. Se calcula la salida de la red: y = $f(w_1 x_1 + w_2 x_2, \dots, w_n x_n + \theta)$
   c. Si $y \neq d(x)$ (clasificación incorrecta) se modifican los pesos y el umbral:

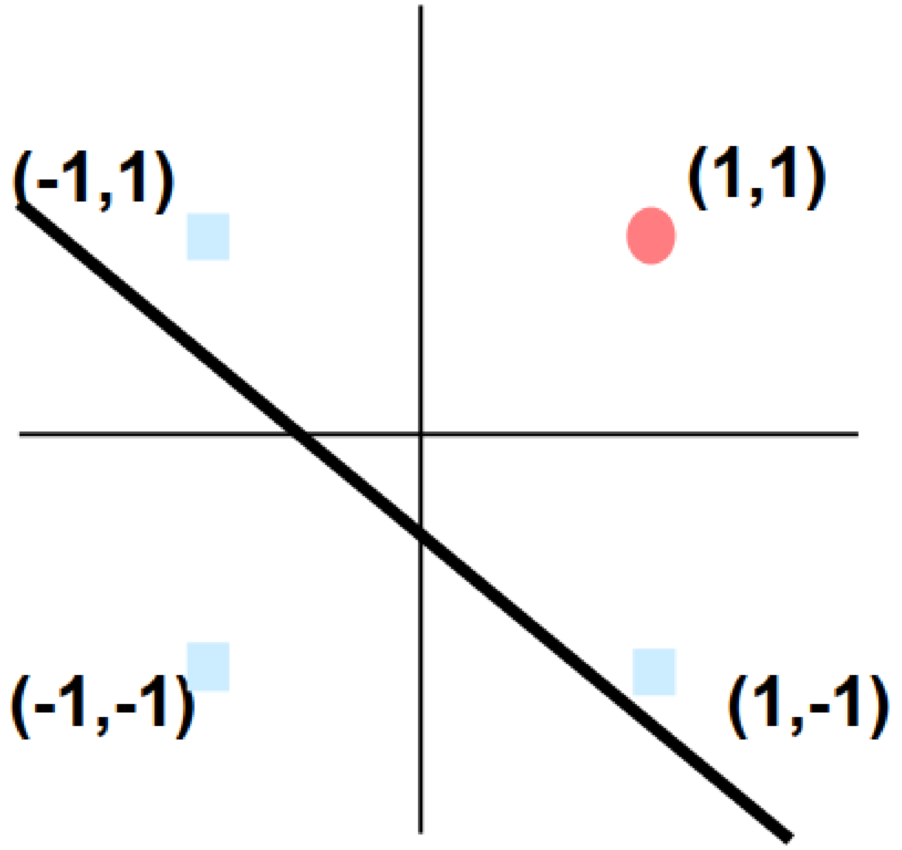
$$w_i(t+1) = w_i(t) + d(x) * x_i$$
$$\theta(t+1) = \theta(t) + d(x)$$

   d. Repetir desde el paso 2.a hasta completar el conjunto de patrones de entrenamiento o hasta alcanzar el criterio de parada.

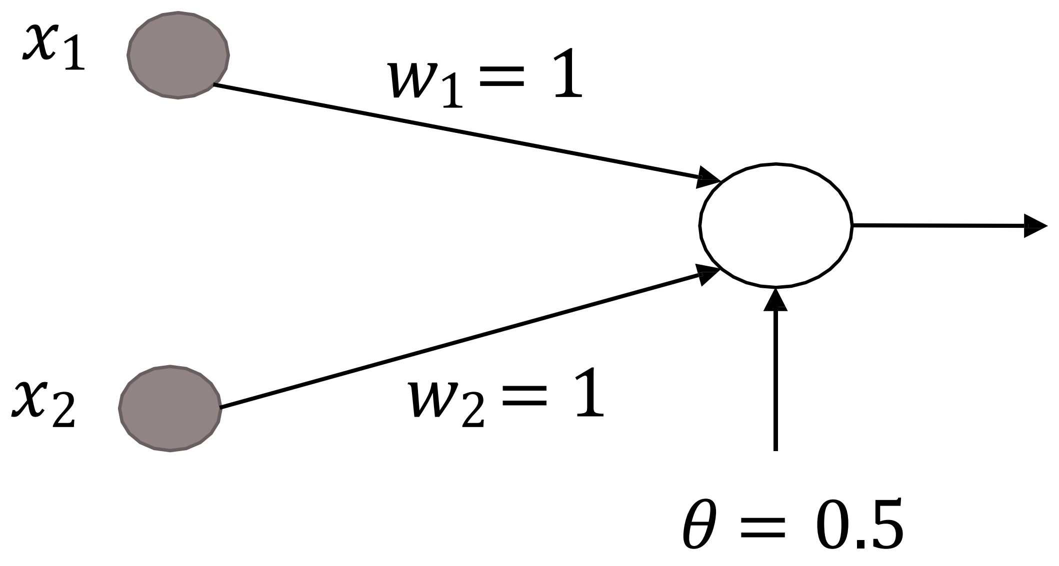
# Perceptron Simple: Ejemplo

- Función lógica AND

| $X_1$ | $X_2$ | AND |
|---|---|---|
| -1 | -1 | -1 |
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| 1 | 1 | 1 |



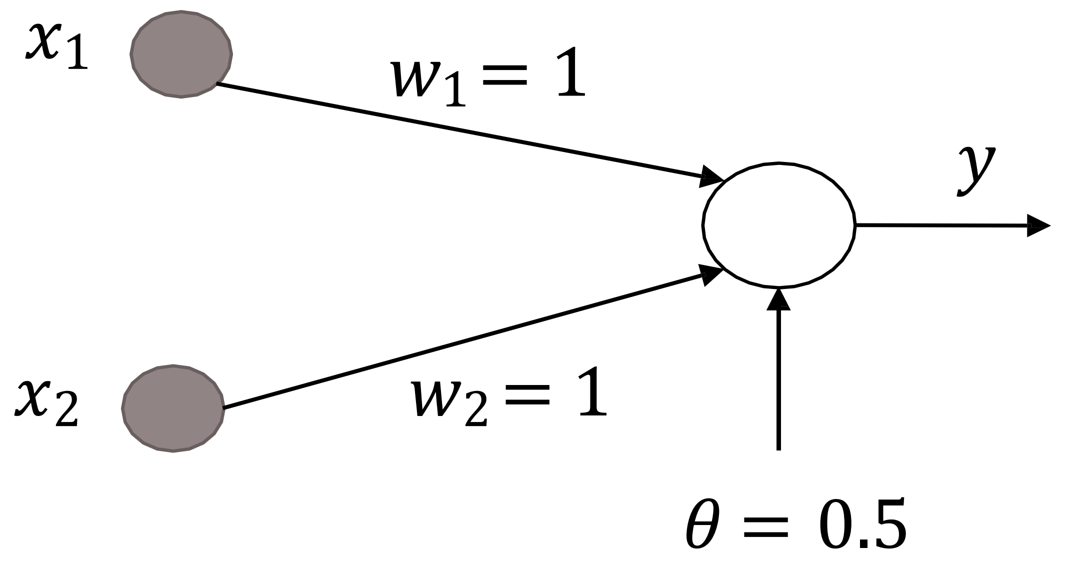

Inicialmente al azar



# Perceptron Simple: Ejemplo

- Función lógica AND

| $x_1$ | $x_2$ | AND |
|---|---|---|
| -1 | -1 | -1 |
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| 1 | 1 | 1 |




$$y = \begin{cases} +1, & si \;\; w_1 x_1 + \cdots + w_n x_n + \theta > 0 \\ -1, & si \;\; w_1 x_1 + \cdots + w_n x_n + \theta \leq 0 \end{cases}$$

$\boldsymbol{x = (-1, -1), \;\; d(x) = -1}$ → $\boldsymbol{y = f(-1.5) = -1}$ → **Bien clasificado**

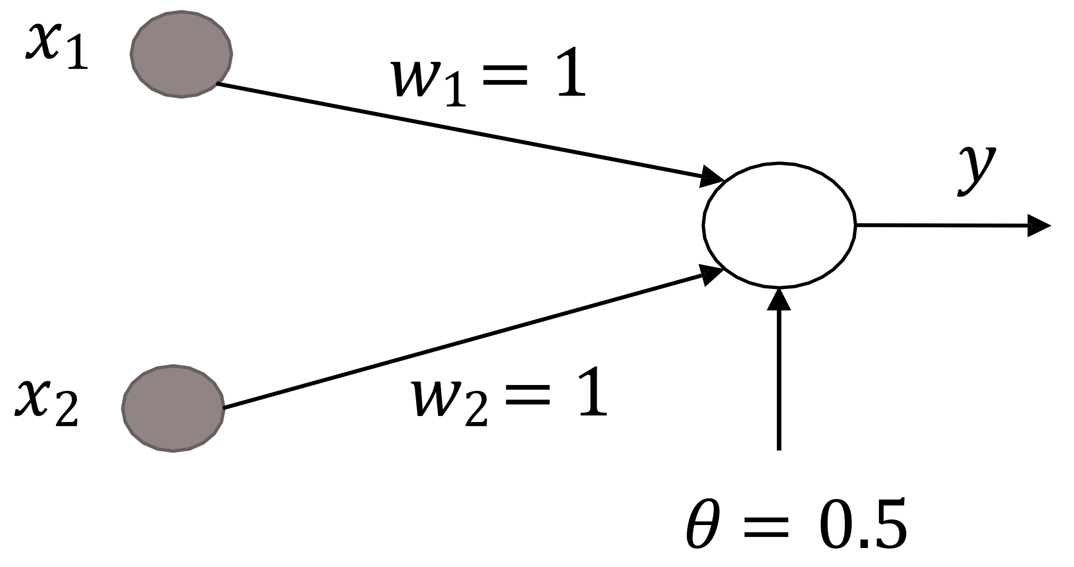
$w_1 * x_1 + w_2 * x_2 + \theta = 1 * -1 + 1 * -1 + 0.5 = -1.5$

Then y = -1

# Perceptron Simple: Ejemplo

- Función lógica AND

| $x_1$ | $x_2$ | AND |
|---|---|---|
| -1 | -1 | -1 |
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| 1 | 1 | 1 |




$$y = \begin{cases} +1, & si\ \ w_1x_1 + \cdots + w_nx_n + \theta > 0 \\ -1, & si\ \ w_1x_1 + \cdots + w_nx_n + \theta \le 0 \end{cases}$$

$\boldsymbol{x = (1, -1), \quad d(x) = -1}$ → $\boldsymbol{y = f(0.5) = 1}$ → **Mal clasificado (nuevos pesos)**

$w_1 = w_1 + d(x) * x_1 = 1 - 1 = 0$

$w_2 = w_2 + d(x) * x_2 = 1 + 1 = 2$

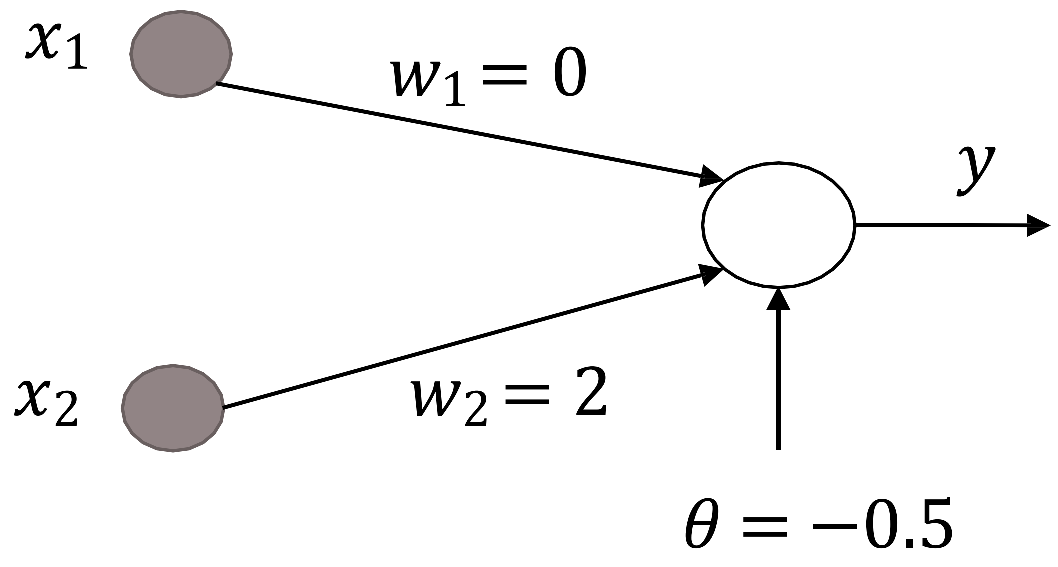
$\theta = \theta + d(x) = 0.5 - 1 = -0.5$

→ $\boldsymbol{y = f(-2.5) = -1}$ → **Bien clasificado**

# Perceptron Simple: Ejemplo

- Función lógica AND

| $x_1$ | $x_2$ | AND |
|---|---|---|
| -1 | -1 | -1 |
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| 1 | 1 | 1 |




$$y = \begin{cases} +1, & si\ \ w_1x_1 + \cdots + w_nx_n + \theta > 0 \\ -1, & si\ \ w_1x_1 + \cdots + w_nx_n + \theta \leq 0 \end{cases}$$

$\boldsymbol{x = (-1, 1), \quad d(x) = -1}$ → $\boldsymbol{y = f(1.5) = 1}$ → **Mal clasificado (nuevos pesos)**

$$w_1 = w_1 + d(x) * x_1 = 0 + 1 = 1$$

$$w_2 = w_2 + d(x) * x_2 = 2 - 1 = 1$$

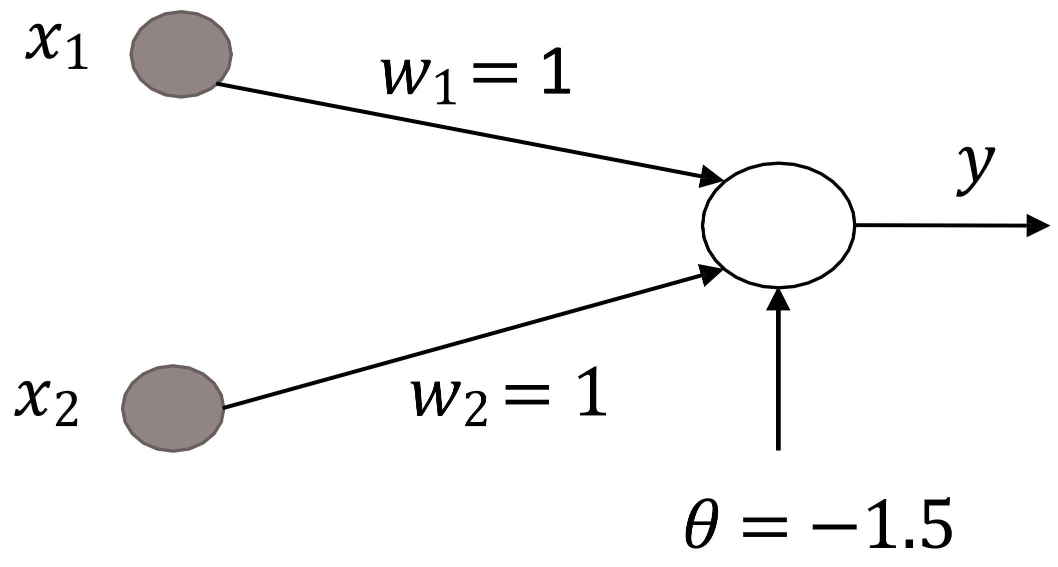
$$\theta = \theta + d(x) = -0.5 - 1 = -1.5$$

→ $\boldsymbol{y = f(-1.5) = -1}$ → **Bien clasificado**

# Perceptron Simple: Ejemplo

- Función lógica AND

| $x_1$ | $x_2$ | AND |
|---|---|---|
| -1 | -1 | -1 |
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| 1 | 1 | 1 |



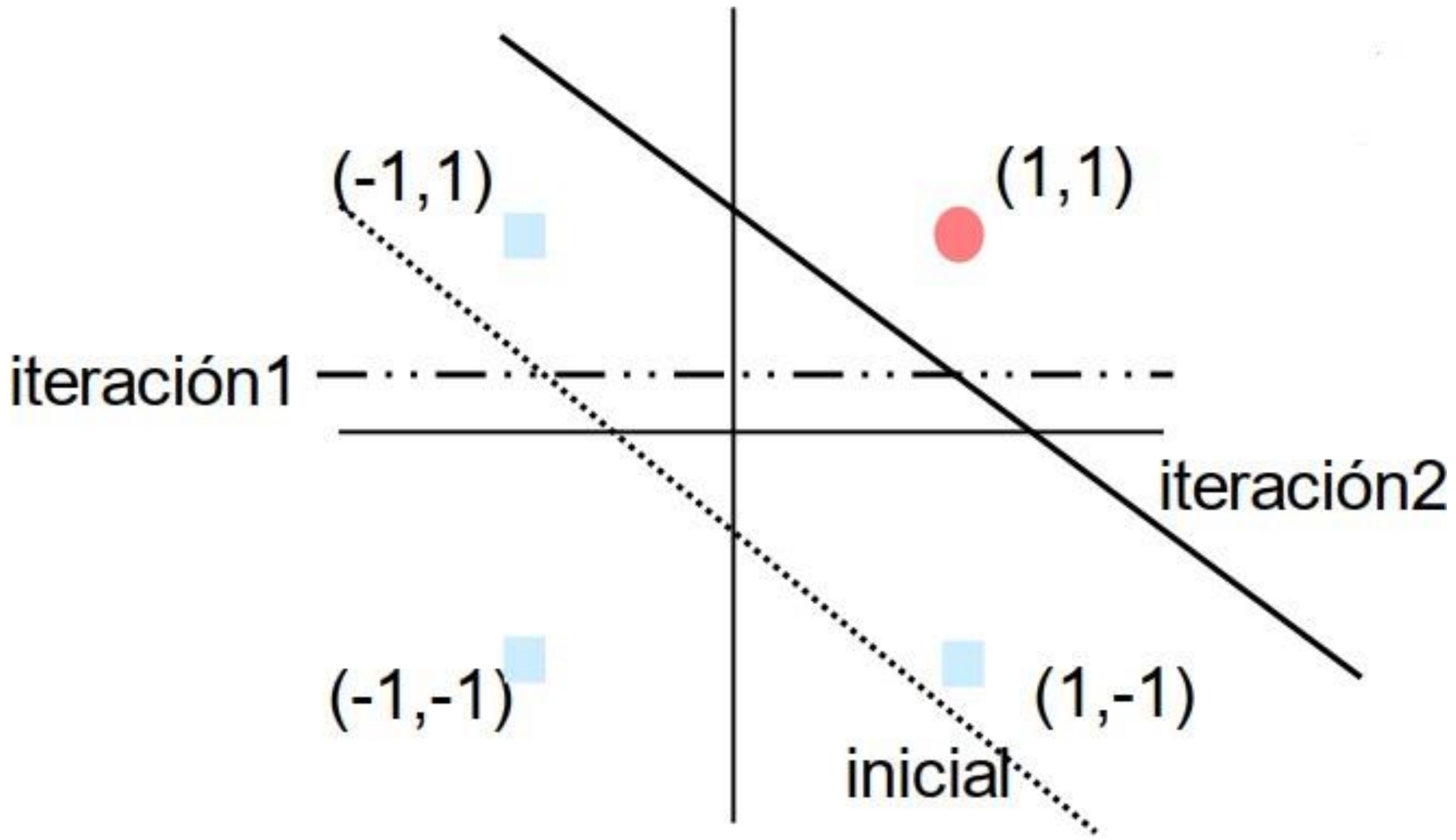

$$y = \begin{cases} +1, & si\ \ w_1 x_1 + \cdots + w_n x_n + \theta > 0 \\ -1, & si\ \ w_1 x_1 + \cdots + w_n x_n + \theta \leq 0 \end{cases}$$

$\boldsymbol{x = (1, 1), \quad d(x) = 1}$ → $\boldsymbol{y = f(2) = 1}$ → **Bien clasificado**

**Un hiperplano solución es:** $\boldsymbol{x_1 + x_2 - 1.5 = 0}$

# Perceptron Simple: Ejemplo

- Función lógica AND



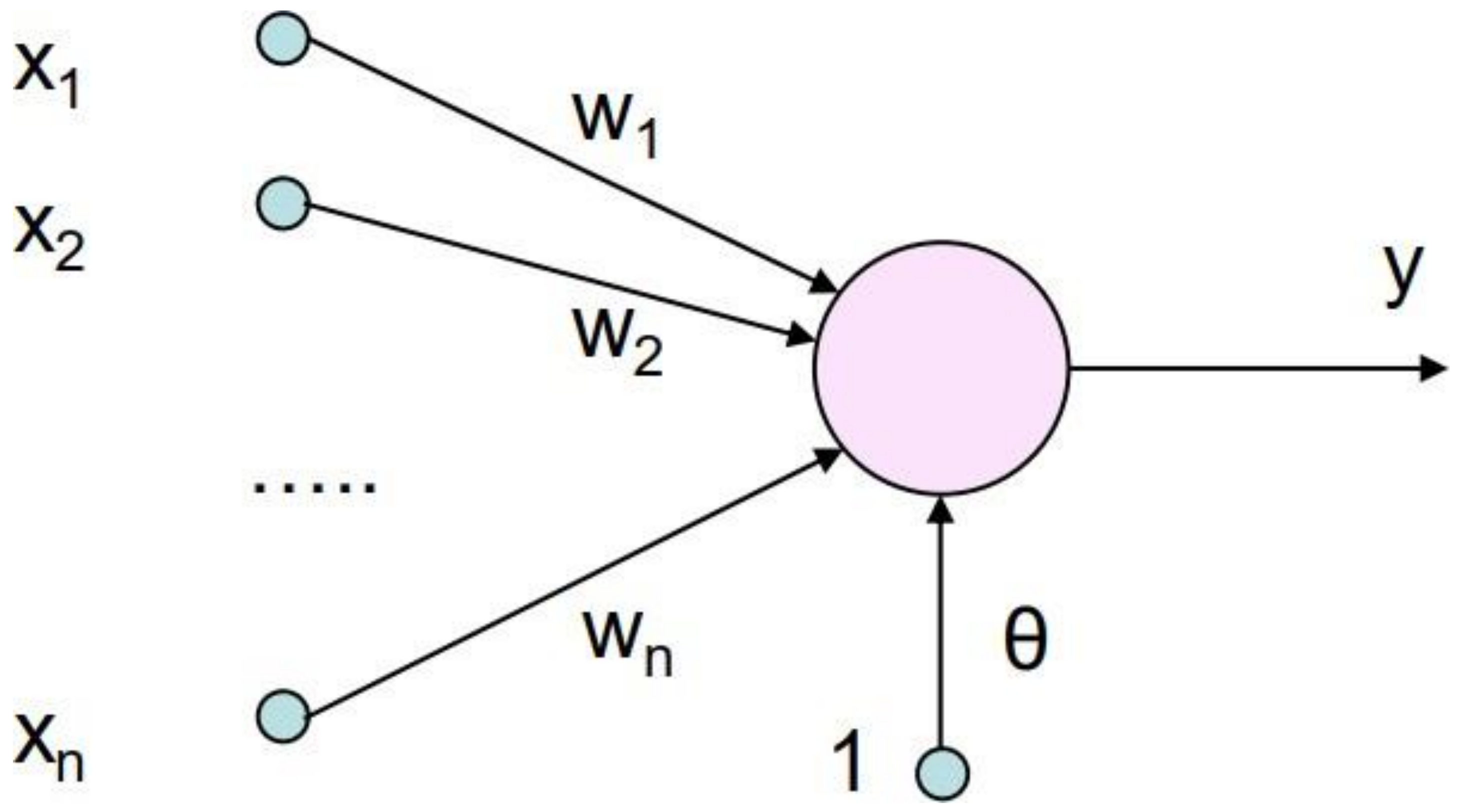

El hiperplano se mueve de una iteración a otra para clasificar correctamente los patrones.

# Tipos de Redes Neuronales

## REDES ADALINE

# ADALINE

- ADALINE (ADAptive LInear NEuron): Desarrollado en 1960 por Widrow y Hoff
- Las entradas pueden ser continuas y se utiliza una neurona similar a la del Perceptron Simple, pero en este caso de respuesta lineal.




$$y = \quad f(w_1 x_1 + \cdots + w_n x_n + \theta)$$

# ADALINE

- La diferencia con el perceptron es la manera de utilizar la salida en la regla de aprendizaje.
  - El perceptron utiliza la salida de la función umbral (binaria) para el aprendizaje. Sólo se tiene en cuenta si se ha equivocado o no.
  - En Adaline se utiliza directamente la salida de la red (real) teniendo en cuenta **cuánto se ha equivocado**.
- La regla de aprendizaje de ADALINE considera el error entre la salida lograda $\boldsymbol{y}$ versus la salida deseada $\boldsymbol{d}$: $|d^p - y^p|$
- Esta regla se conoce como REGLA DELTA. La constante α se denomina TASA DE APRENDIZAJE. $\Delta w_i = \alpha|d^p - y^p|x_i$
- Se busca minimizar la desviación de la red para todos los patrones de entrada, eligiendo una medida del error global. Normalmente se utiliza el error cuadrático medio:

$$E = \frac{1}{2}\sum_{p=1}^{m}(d^p - y^p)^2$$

# ADALINE : Aprendizaje

1. Inicializar los pesos y umbral de forma aleatoria
2. Seleccionar un ejemplo $x$ del conjunto de entrenamiento
3. Calcular la salida de la red: $y = f(w_1 x_1 + \cdots + w_n x_n + \theta)$

   y obtener la diferencia $|d^p - y^p|$
4. Para todos los pesos y para el umbral, calcular

$$\Delta w_i = \alpha \; |d^p - y^p| \, x_i \qquad \Delta\theta_i = \alpha |d^p - y^p|$$

5. Modificar los pesos y el umbral del siguiente modo

$$w_i(t+1) = w_i(t) + \Delta w_i$$

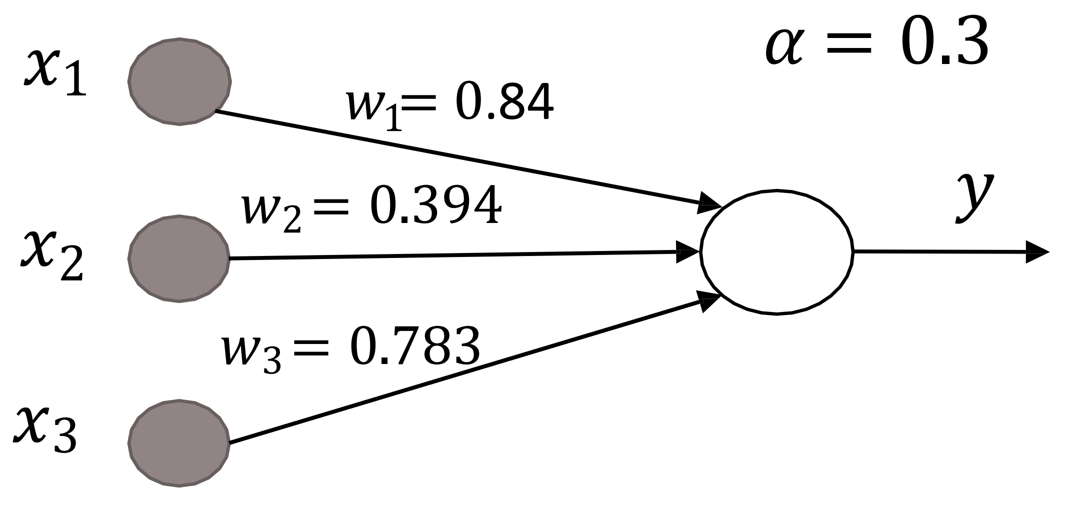
$$\theta(t+1) = \theta(t) + \Delta\theta_i$$

6. Repetir para todos los patrones de entrenamiento hasta cumplir el criterio de parada.

# ADALINE : Ejemplo

❖ Decodificador binario a decimal

| $x_1$ | $x_2$ | $x_3$ | d(X) |
|---|---|---|---|
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **2** |
| 0 | 1 | 1 | **3** |
| 1 | 0 | 0 | **4** |
| 1 | 0 | 1 | **5** |
| 1 | 1 | 0 | **6** |
| 1 | 1 | 1 | **7** |




$$\boldsymbol{y = 0.84 * 0 + 0.394 * 0 + 0.783 * 1 = 0.783}$$

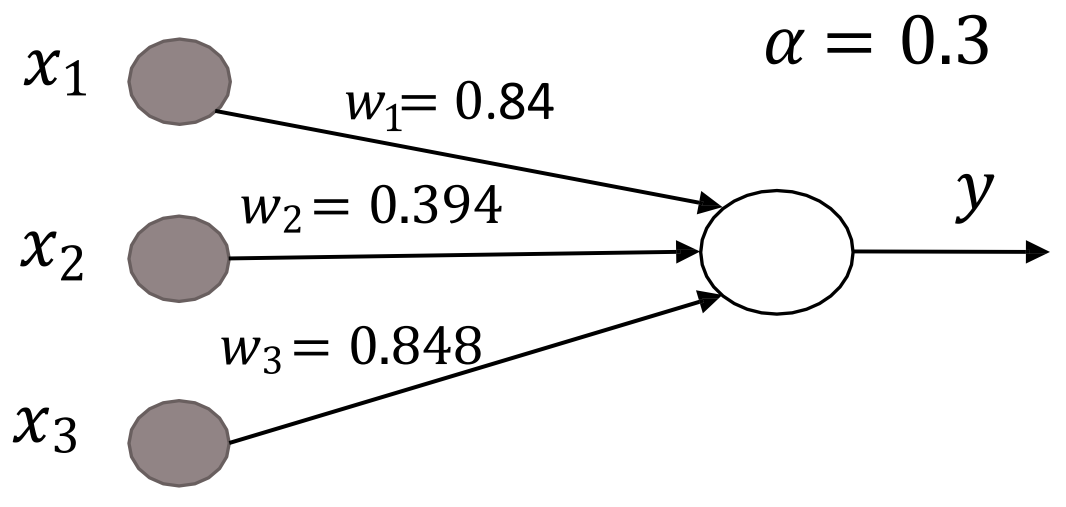
$$E = |d^p - y^p| = |1 - 0.783| = 0.217$$

$$w_1 = w_1 + \alpha * E * x_1 = 0.84 + 0.3 * 0.217 * 0 = \mathbf{0.840}$$
$$w_2 = w_2 + \alpha * E * x_2 = \mathbf{0.394}$$
$$w_3 = w_3 + \alpha * E * x_3 = \mathbf{0.848}$$

# ADALINE : Ejemplo

- Decodificador binario a decimal

| $x_1$ | $x_2$ | $x_3$ | d(X) |
|---|---|---|---|
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **2** |
| 0 | 1 | 1 | **3** |
| 1 | 0 | 0 | **4** |
| 1 | 0 | 1 | **5** |
| 1 | 1 | 0 | **6** |
| 1 | 1 | 1 | **7** |




$$\boldsymbol{y = 0.84 * 0 + 0.394 * 1 + 0.848 * 0 = 0.394}$$

$$E = \ |d^p - y^p\ |= \ |2 - 0.394\ | \ = 1.61$$

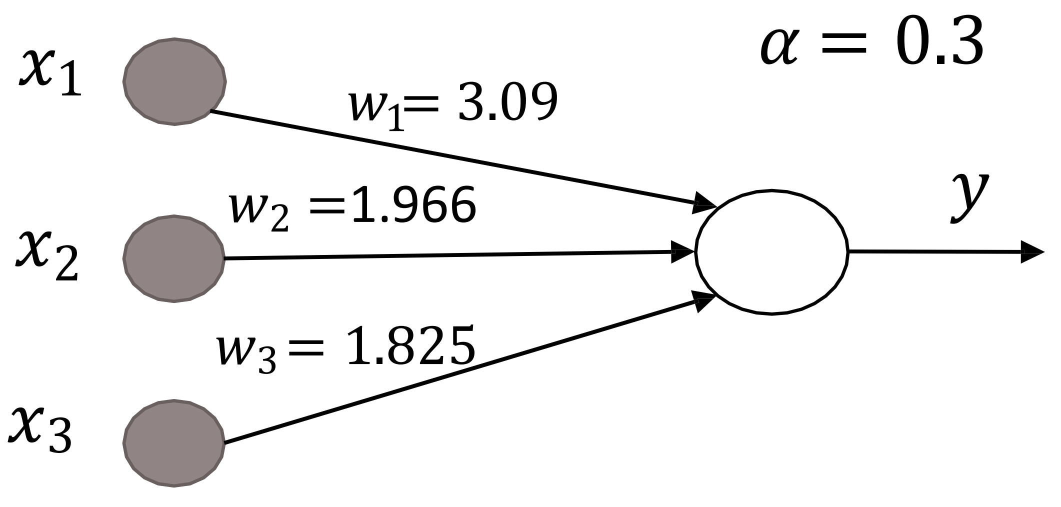
$$w_1 = w_1 + \alpha * E * x_1 = 0.84 + 0.3 * 0.161 * 0 = \mathbf{0.840}$$
$$w_2 \ = w_2 \ + \alpha * E * x_2 = \mathbf{0.876}$$
$$w_3 \ = w_3 \ + \alpha * E * x_3 = \mathbf{0.848}$$

# ADALINE : Ejemplo

- Decodificador binario a decimal

| $x_1$ | $x_2$ | $x_3$ | d(X) |
|---|---|---|---|
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **2** |
| 0 | 1 | 1 | **3** |
| 1 | 0 | 0 | **4** |
| 1 | 0 | 1 | **5** |
| 1 | 1 | 0 | **6** |
| 1 | 1 | 1 | **7** |

Resultado después de la primera iteración del entrenamiento




$$\boldsymbol{y = 3.09 * 1 + 1.966 * 1 + 1.825 * 1 = 6.881}$$

$$E = |d^p - y^p| = |7 - 6.881| = 0.12$$

$$w_1 = w_1 + \alpha * E * x_1 = 3.09 + 0.3 * 0.12 * 1 = \mathbf{3.126}$$
$$w_2 = w_2 + \alpha * E * x_2 = \mathbf{2.002}$$
$$w_3 = w_3 + \alpha * E * x_3 = \mathbf{1.861}$$

# ADALINE : Ejemplo

❖ Decodificador binario a decimal. Visualización de los pesos según iteraciones.

| Iteración | w1 | w2 | w3 |
|---|---|---|---|
| 1 | 3.12 | 2.00 | 1.86 |
| 2 | 3.61 | 1.98 | 1.42 |
| 3 | 3.82 | 1.98 | 1.2 |
| 4 | 3.92 | 1.98 | 1.1 |
| 5 | 3.96 | 1.99 | 1.02 |
| 6 | 3.99 | 2.00 | 1.01 |
| 7 | 4.00 | 2.00 | 1.00 |
| 8 | 4.00 | 2.00 | 1.00 |
| 9 | 4.00 | 2.00 | 1.00 |
| 10 | 4.00 | 2.00 | 1.00 |

>La tasa de aprendizaje $\alpha$ también puede ser adaptativa.

>Por ejemplo al inicio el valor puede ser alto, para dar "grandes pasos" de corrección del error y para salir de mínimos locales.

>Sin embargo al final del entrenamiento debe disminuir para hacer correcciones finas.

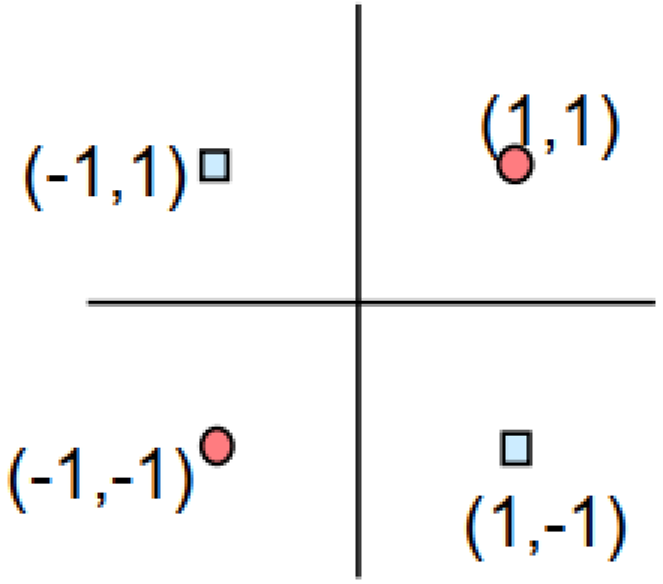
## Conclusión

- El uso del Perceptrón o de las redes ADALINE permite aproximar de manera fácil, cualquier tipo de función  o sistemas, sólo conociendo un conjunto de ejemplos.
- De esta manera cualquier sistema (caja negra), se puede representar por una red.
- Sin embargo, también se demostró que estas técnicas poseen grandes limitaciones.
  - Un ejemplo clásico es el OR Exclusivo
- En conclusión: éstas técnicas sólo pueden resolver sistemas donde los ejemplos son linealmente separables.

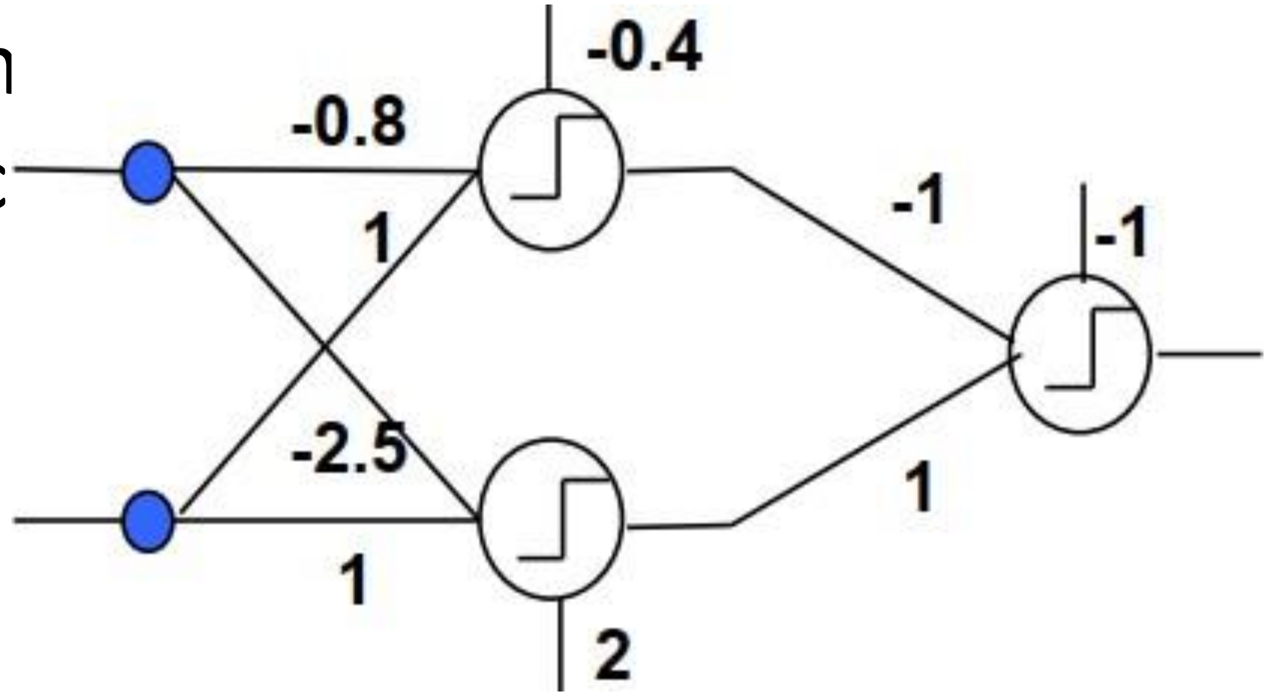
# Conclusión

- Problemas no linealmente separables
- Ejemplo XOR: No existe un hiperplano.

| $x_1$ | $x_2$ | $d(x)$ |
|---|---|---|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |
| 1 | 1 | 1 |



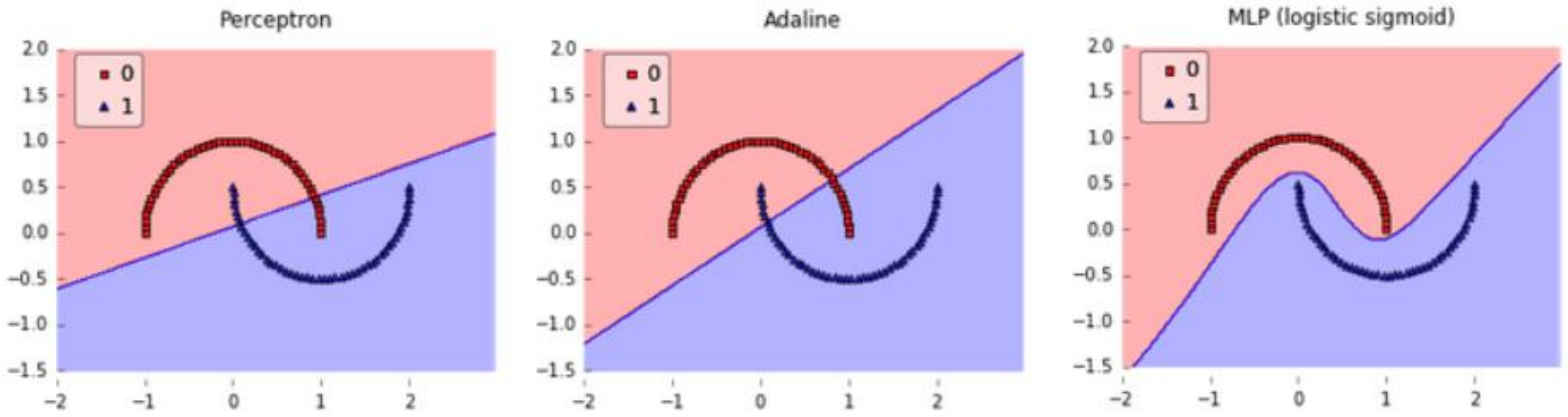

- Solución:
  Com
  Perc



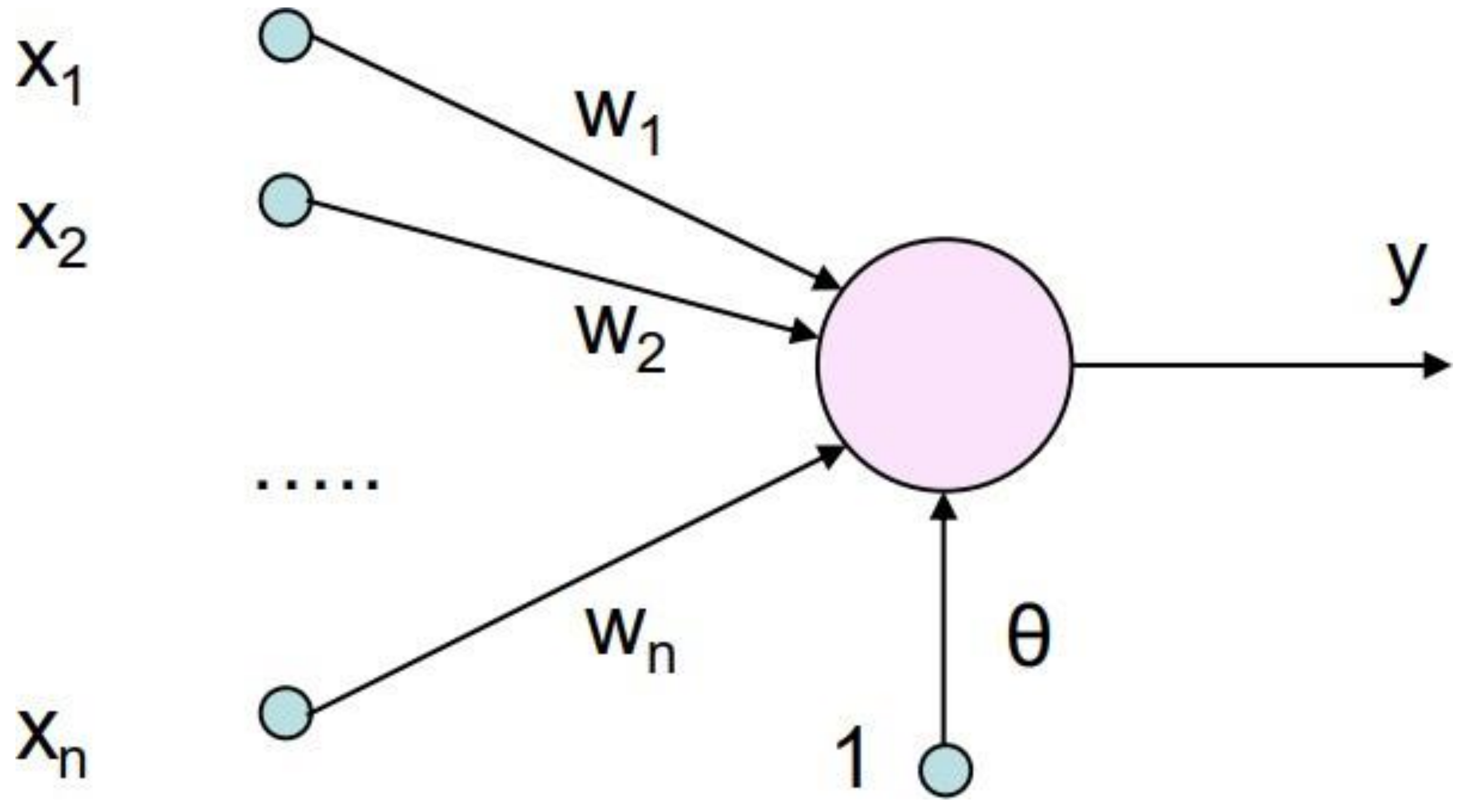

Multicapas

# Conclusión

- Problemas no linealmente separables



# Recordar la red ADALINE




$$y = w_1 x_1 + \cdots + w_n x_n + \theta$$

Si queremos aplicar la fuerza bruta para hallar los mejores pesos W, ¿Cómo seria el procedimiento?:

- Definir una iteración máxima
- En cada iteración elegir aleatoriamente valores para W
- Evaluar la salida de la Red $y$
- Al final, nos quedamos con los pesos que minimizan el error.

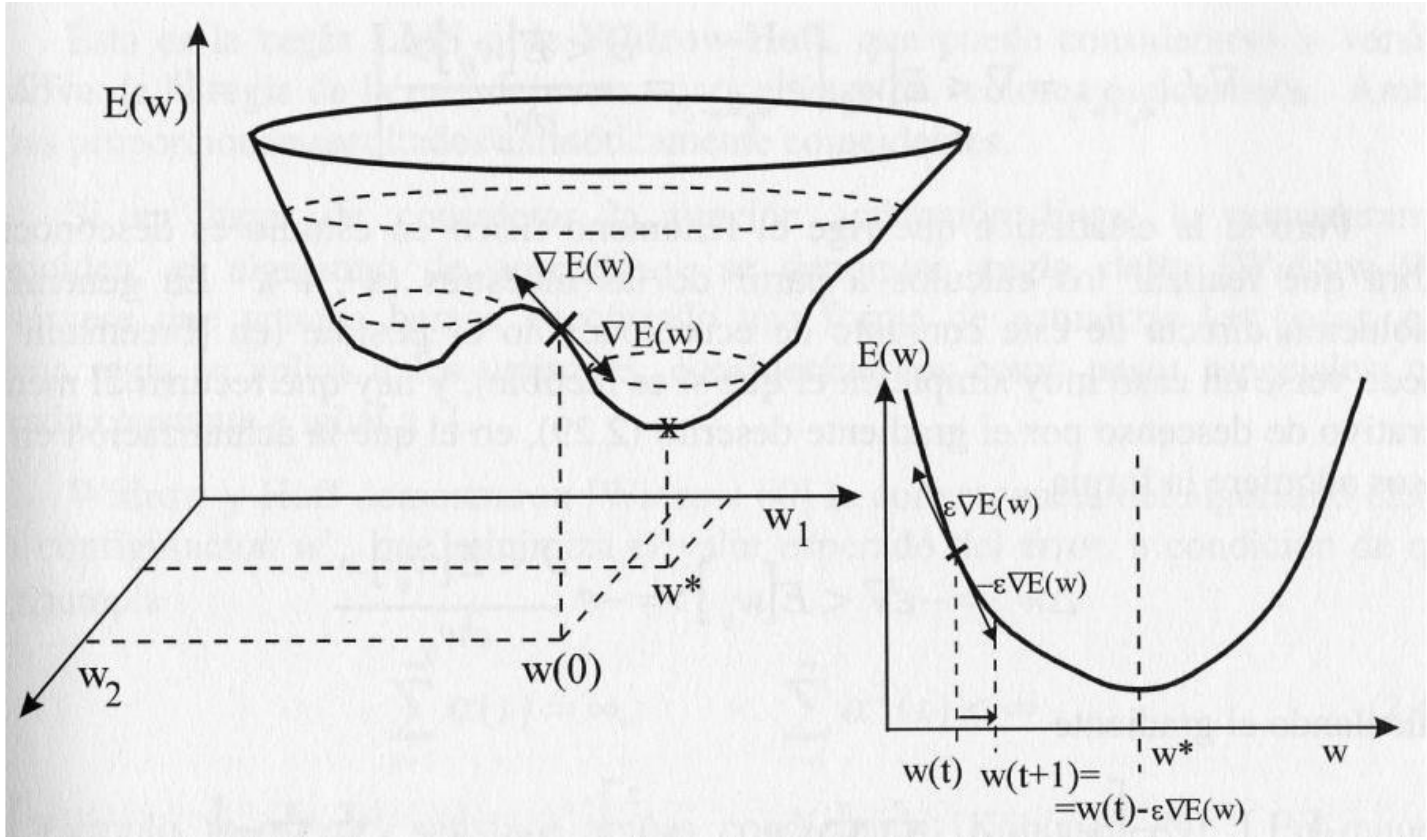
Problema:

- Asignación ciega de los pesos
- Demasiadas iteraciones
- Difícil convergencia

# Recordar la red ADALINE

**¿Cómo lo mejoramos?**

Usaremos el error cuadrático para poder aplicar el método del gradiente descendiente.



# Gradiente descendiente

En la ADALINE, la salida puede representarse de la siguiente manera:

$$y = \sum_{j=1}^{N+1} w_j x_j$$

En donde:
$N$ es la cantidad de entradas
$N + 1$ representa el umbral

El error cuadrático:

$$E = \frac{1}{2}\sum_{k=1}^{p}\left(z^k - y(k)\right)^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left(z^k - \sum_{j=1}^{N+1} w_j(k) x_j^k\right)^2$$

En donde:
$p$ es la cantidad total de ejemplos
$z^k$ es la salida deseada para el ejemplo $k$

# Gradiente descendiente

Vamos a seguir el método del gradiente descendiente, es decir, los pesos se actualizarán por la dirección opuesta a la dirección del gradiente del error.

$$w_j(k+1) = w_j(k) + \Delta w_j(k),$$

En donde: $$\Delta w_j(k) = -\eta \frac{\partial E}{\partial w_j(k)}$$ $j$=1,2,…,$N$

$$= \eta\left[z^k - y(k)\right]x_j^k$$

η representa la taza de aprendizaje, controla la longitud del paso que vamos a dar en la dirección opuesta del gradiente. Conforme mayor sea η mayor será la cantidad por la que se modificarán los pesos sinápticos. Dicho parámetro debe ser un valor pequeño para evitar dar pasos demasiado largos, es decir, que nos lleven a soluciones peores que la que teníamos.

# Gradiente descendiente

En el caso anterior se aplicó el método de gradiente sobre la función de activación de identidad. Como seria la derivación para las funciones sigmoidales.

a) Función logística:

$$g(x) \equiv \frac{1}{1+\exp(-2\beta x)} \longrightarrow g'(x) = 2\beta g(x)[1-g(x)]$$

b) Función tangente hiperbólica:

$$g(x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}} \longrightarrow g'(x) = \beta\left[1-g(x)^2\right]$$

# Gradiente descendiente

Resolución del método del gradiente descendiente para cualquier función de activación.

$$E = \frac{1}{2}\sum_{k=1}^{p}\left(z^k - y(k)\right)^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left(z^k - g(\sum_{j=1}^{N+1} w_j(k)x_j^k)\right)^2$$

Entonces:

$$\Delta w_j(k) = -\eta \frac{\partial E}{\partial w_j(k)}$$

$$= \eta\left[z^k - y(k)\right]g'(h)x_j^k,$$

Donde:

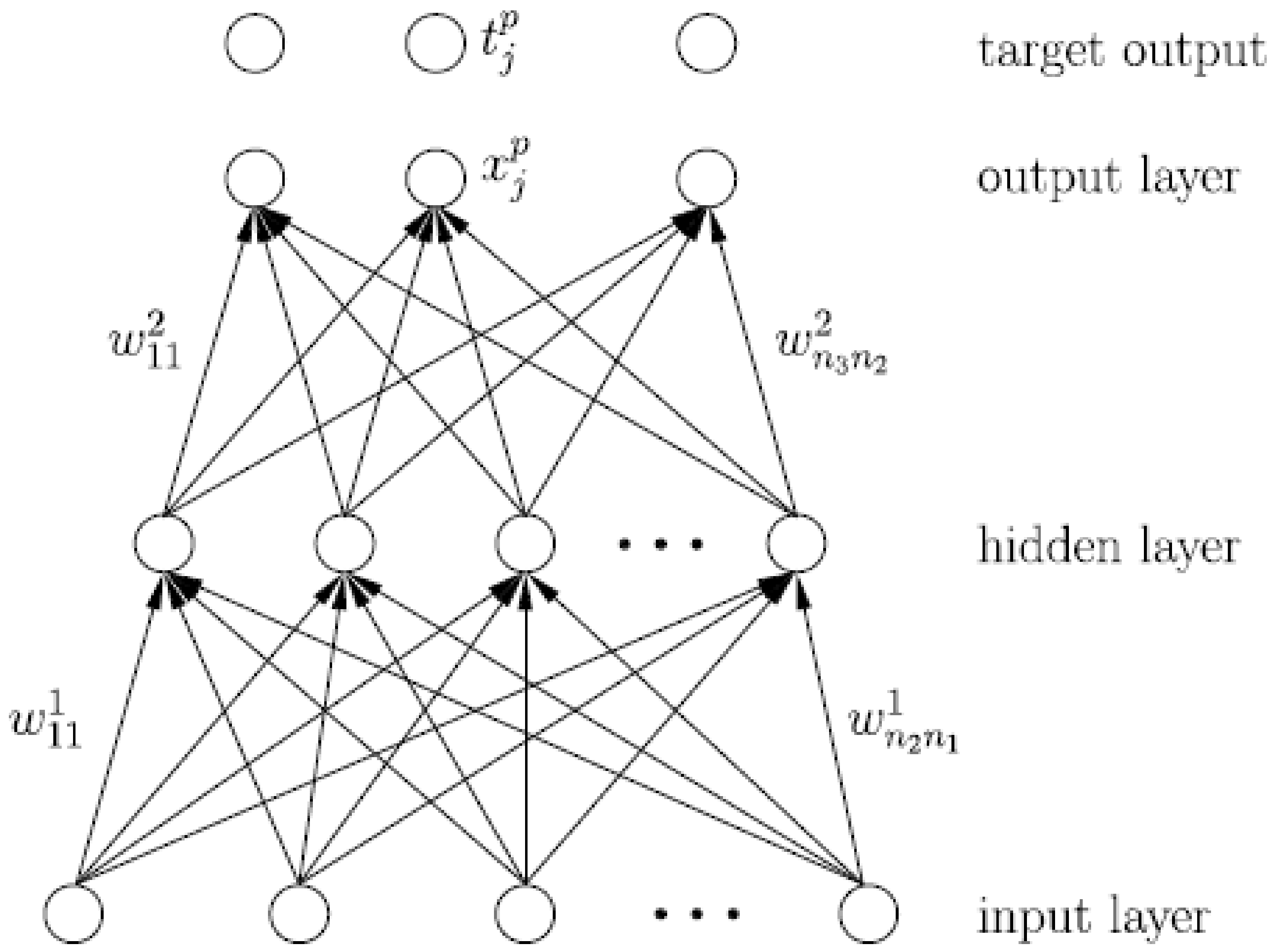
$$h = \sum_{j=1}^{N} w_j x_j$$

# Tipos de Redes Neuronales

Multilayer Percepeton

# BackPropagation Algorithm

- Back propagation calculates the gradients and maps the correct inputs to the correct outputs.
- There are two steps to back propagation: the propagation phase and the updating of the weight



# BackPropagation Algorithm

- Neurons calculate the current value $x_j = f\left(\sum_{i=1}^{n} w_{ji} x_i\right)$ where n is the number of neuron in previous layer.
- It is used sigmoid function: $f(x) = \frac{1}{1+e^{-x}}.$
- 
- Weights are changed based on: $E_p(\boldsymbol{w}) = \frac{1}{2} \sum_{k \in output} (t_k^p - x_k^p)^2$ and $\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}.$

$$\Delta_p w_{ji} = \eta \delta_j^p x_i^p,$$

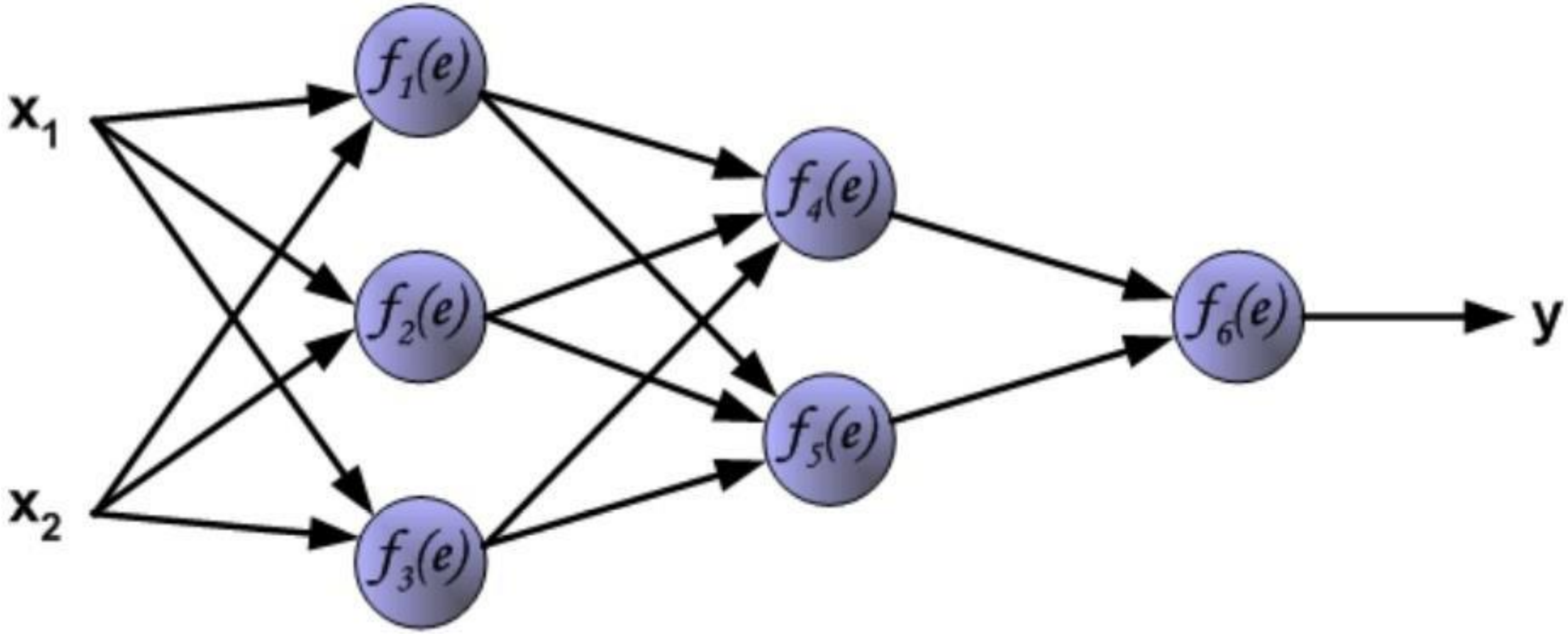
with

$$\delta_j^p = \begin{cases} x_j^p(1-x_j^p)(t_j^p - x_j^p) & \text{if } j \text{ is an output neuron,} \\ x_j^p(1-x_j^p)\sum_k \delta_k^p w_{kj} & \text{if } j \text{ is a hidden neuron,} \end{cases}$$

# BackPropagation Algorithm

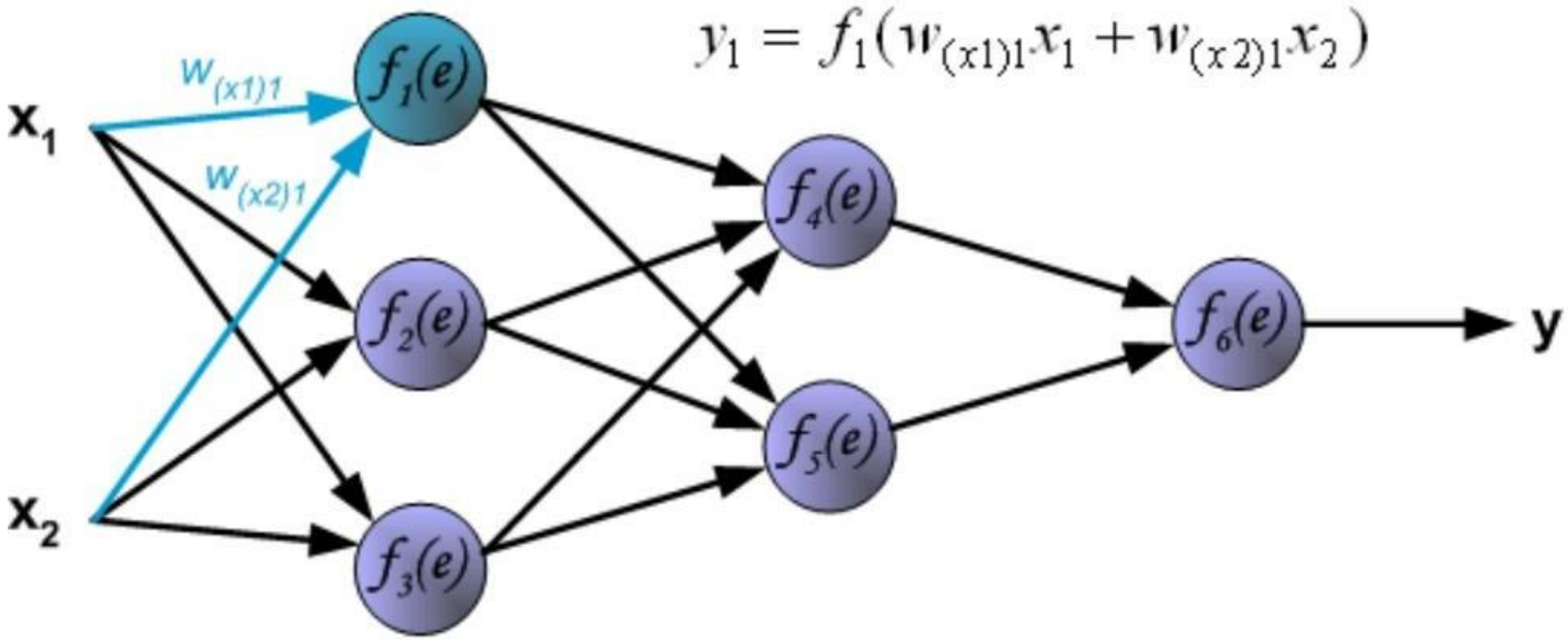
BACKPROPAGATION(*TrainingExamples*, $\eta$)
Initialize all weights $w_j$ to random values
**Repeat**
  **For all** $(q^p, t^p) \in TrainingExamples$
    1. **Apply the query vector** $q^p$ to the input layer
    2. **Forward propagation**:
      For all layers from the first hidden layer upward
        For each neuron of the layer
          Calculate activation $x_j = f(\sum_{i=1}^{n} w_{ji} x_i)$
    3. **Calculation of the square error** $E_p(w)$
    4. **Backward propagation**:
      For all levels of weights from the last downward
        For each weight $w_{ji}$
          $w_{ji} = w_{ji} + \eta \delta_j^p x_i^p$
**Until** $w$ converges or time limit is reached

# Algoritmo Backpropagation: ejemplo gráfico




Ejemplo de un MLP de 3 capas

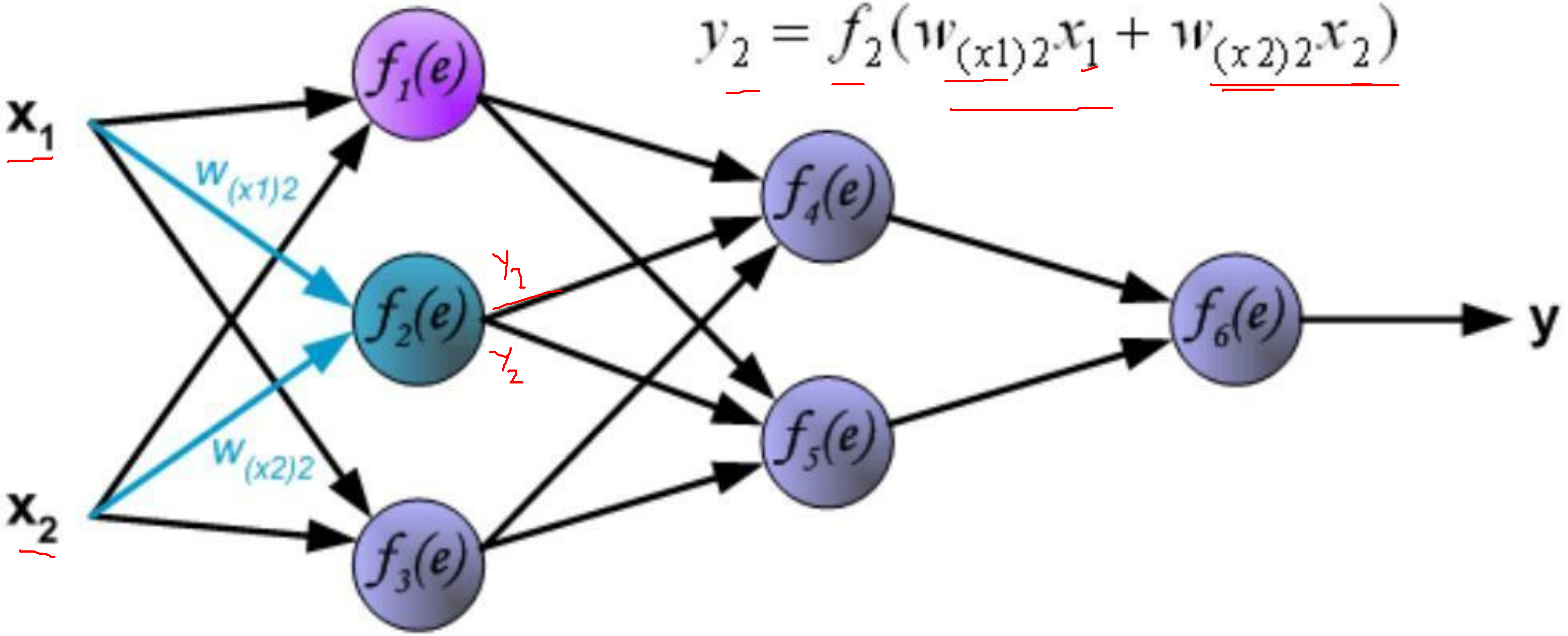
# Backpropagation: Cálculo de las salidas en cada capa




Cálculo salida capa I, neurona 1.

# Backpropagation: Cálculo de las salidas en cada capa

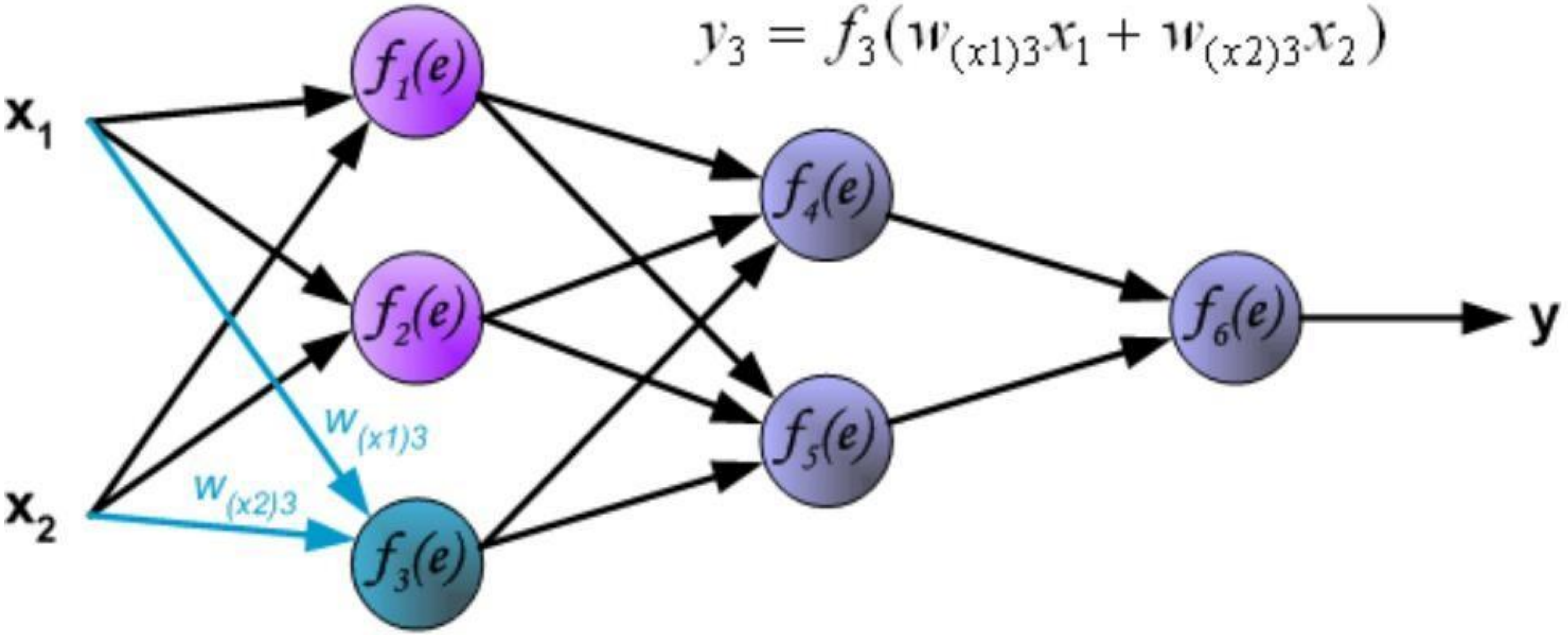
$$y_2 = f_2(w_{(x1)2} x_1 + w_{(x2)2} x_2)$$




Cálculo salida capa I, neurona 2.

# Backpropagation: Cálculo de las salidas en cada capa

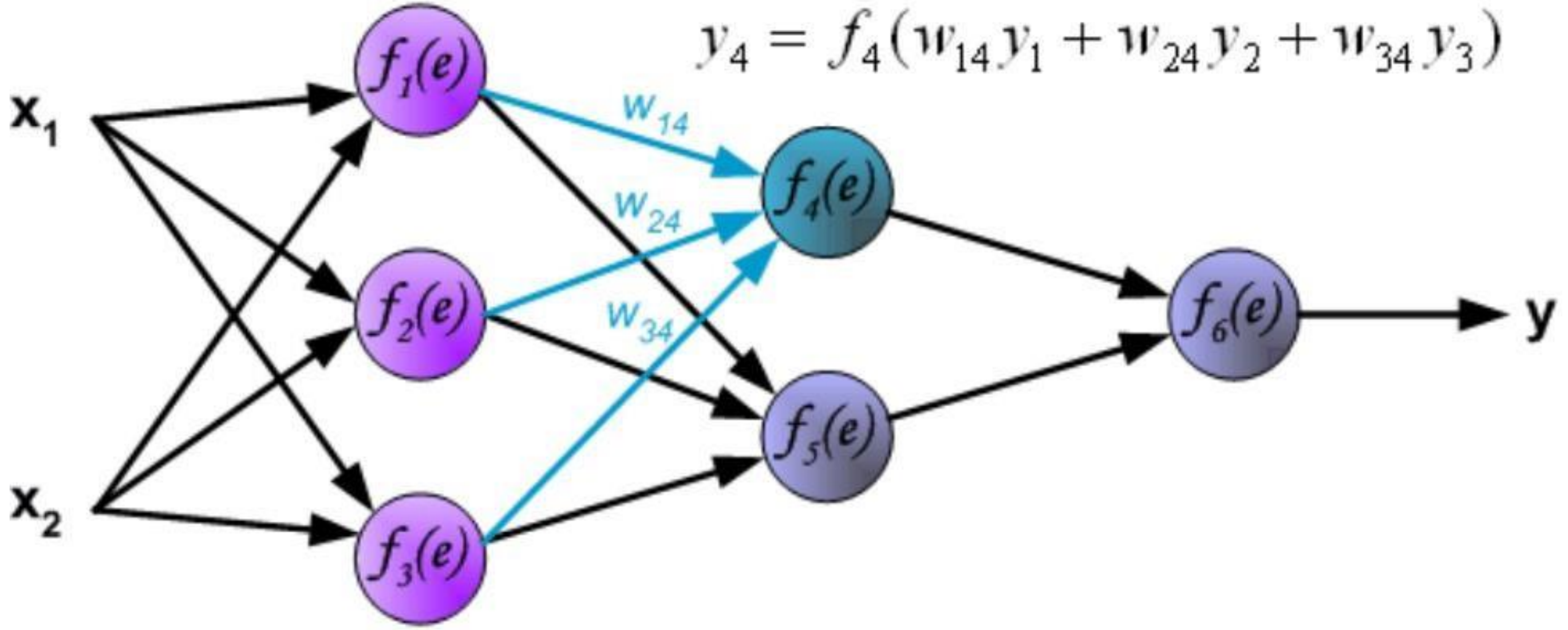
$$y_3 = f_3(w_{(x1)3} x_1 + w_{(x2)3} x_2)$$




Cálculo salida capa I, neurona 3.

# Backpropagation: Cálculo de las salidas en cada capa

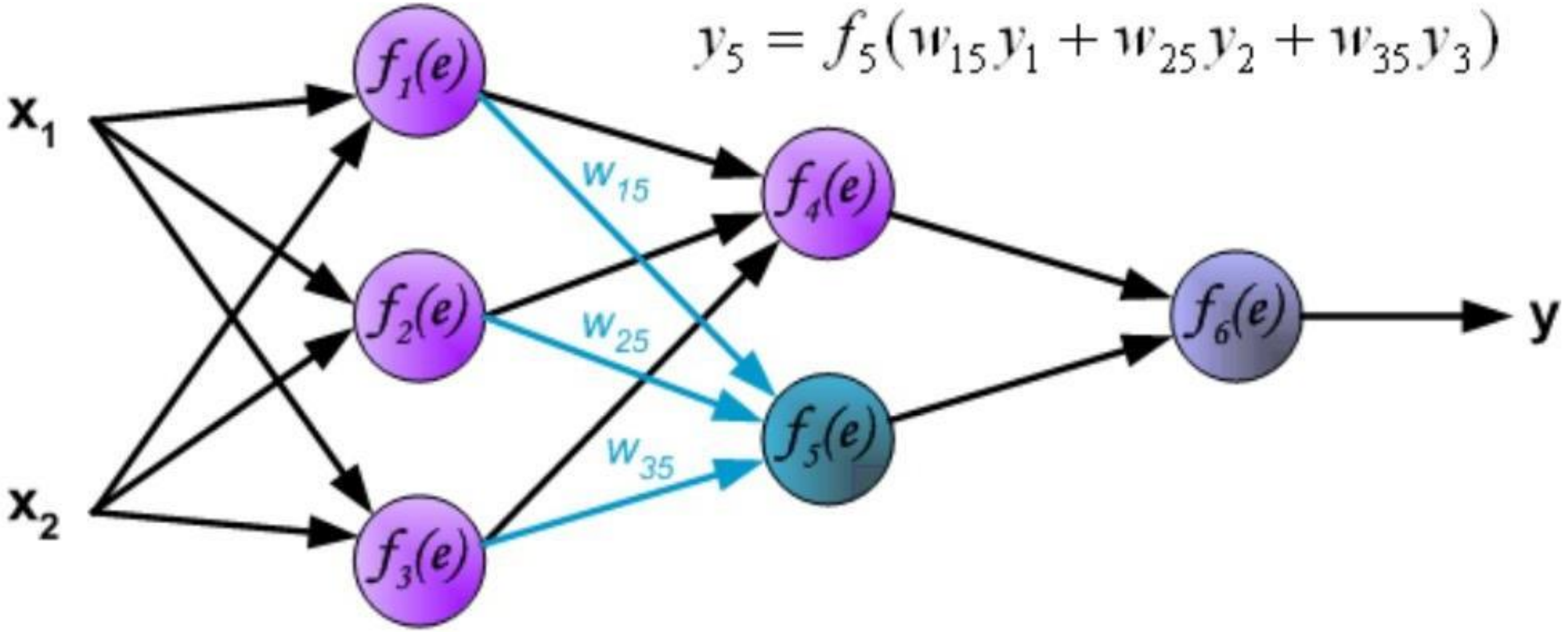
$$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$$




Cálculo salida capa II, neurona 1.

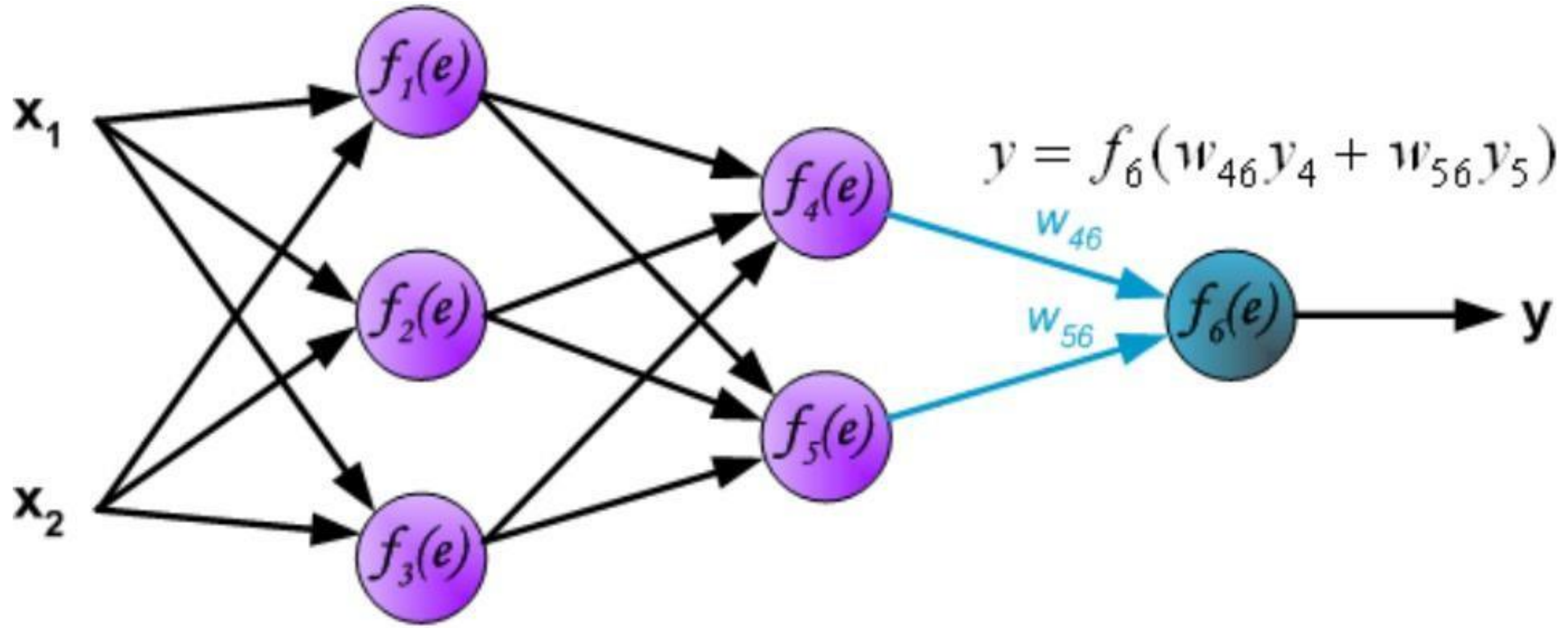
# Backpropagation: Cálculo de las salidas en cada capa

$$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3)$$



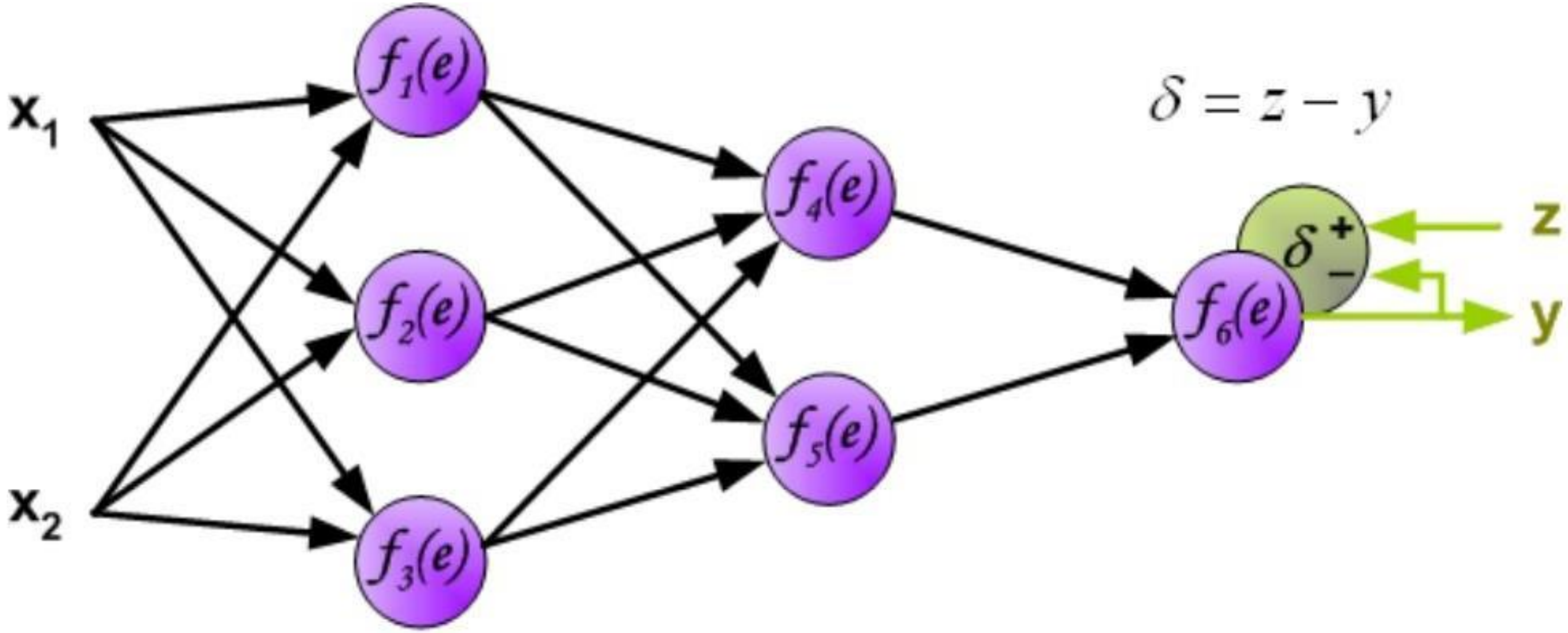

Cálculo salida capa II, neurona 2.

# Backpropagation: Cálculo de las salidas en cada capa




Cálculo salida capa III, neurona 1.

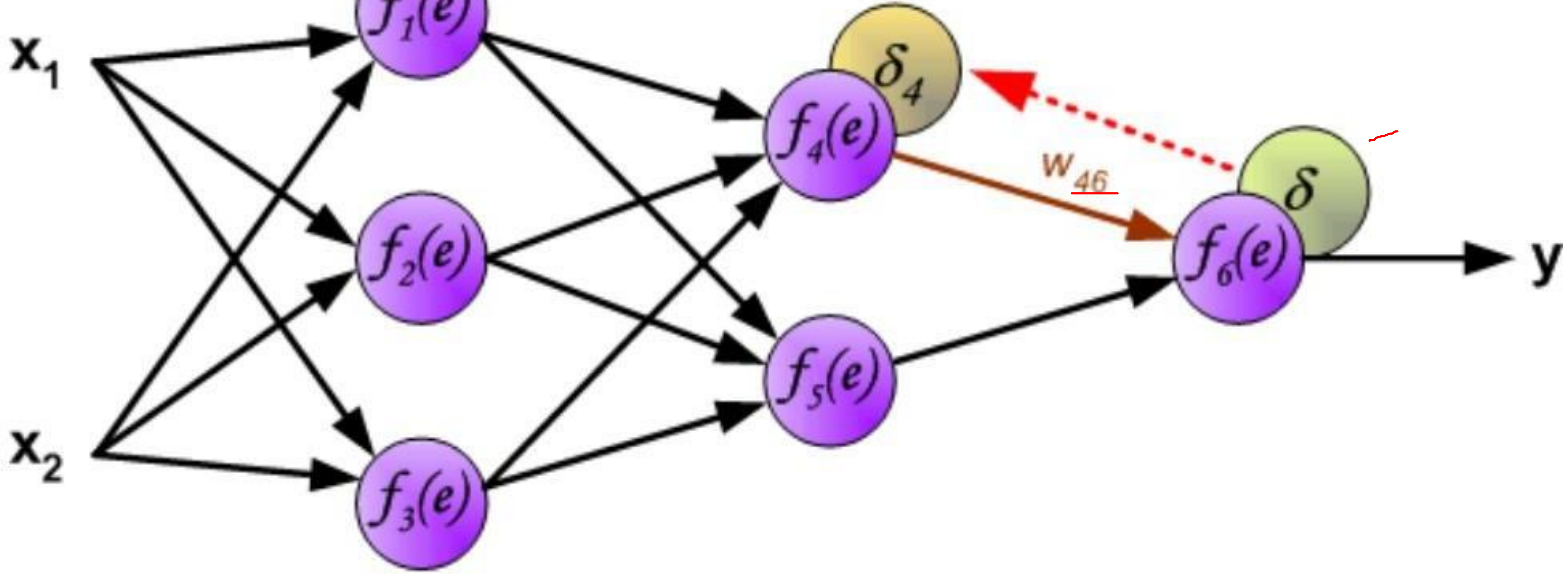
# Backpropagation: Retropropagación en la capa III




Cálculo del error en capa III, neurona 1.

# Backpropagation: Retropropagación en la capa III

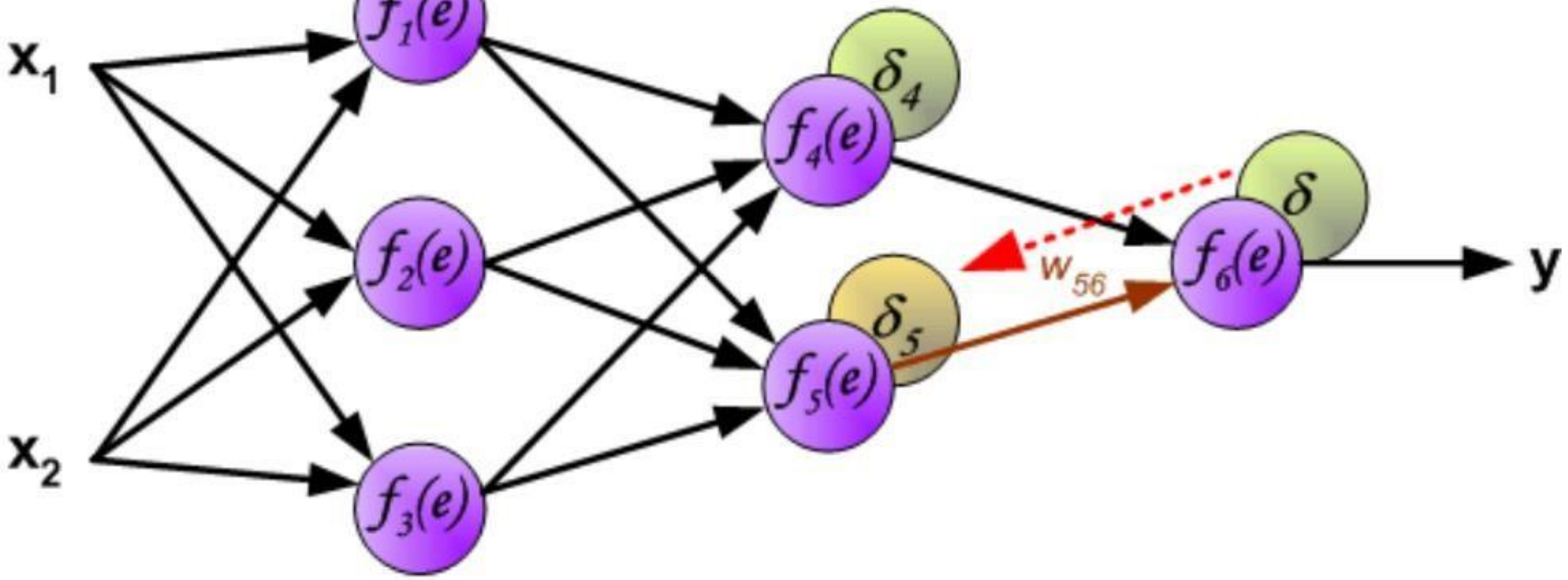
$$\delta_4 = w_{46}\delta$$




Propagación del error a la capa II, neurona 1.

# Backpropagation: Retropropagación en la capa III

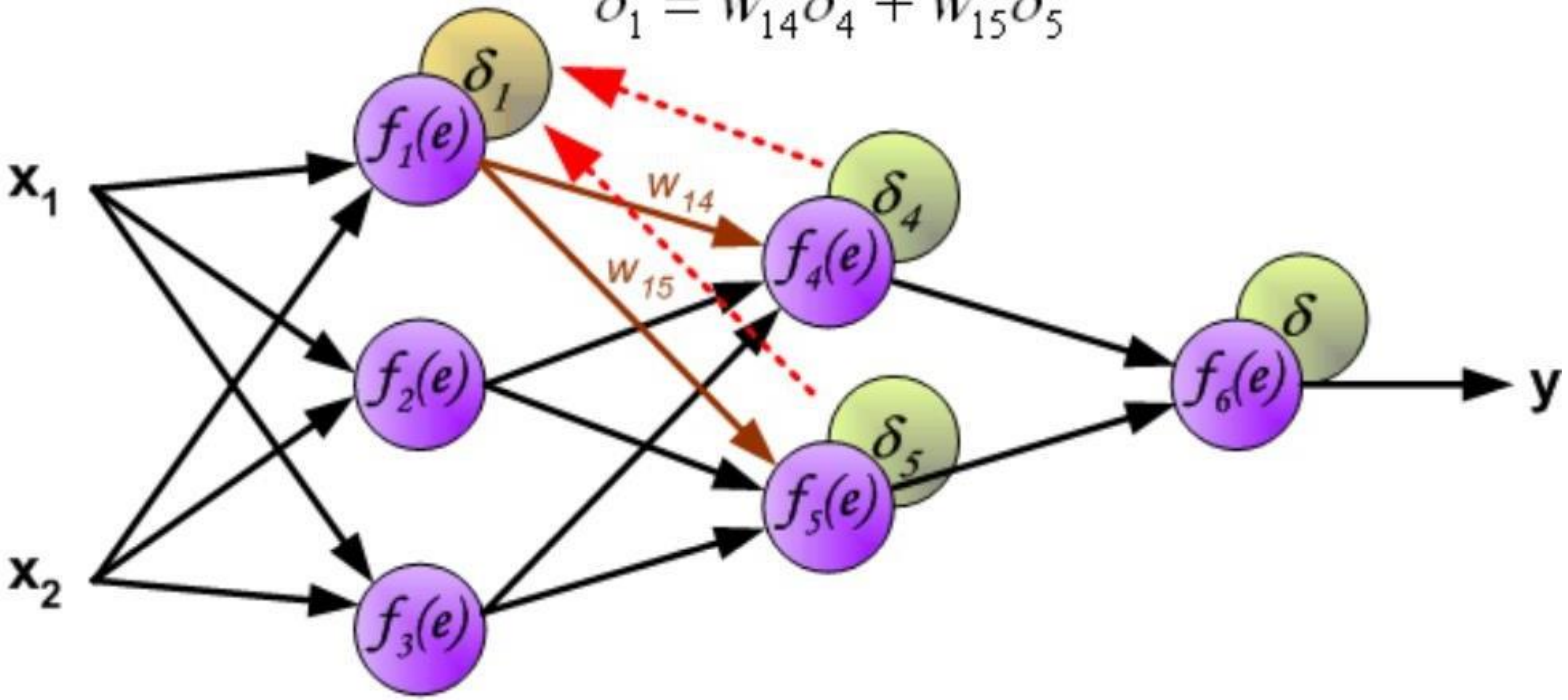
$\delta_5 = w_{56}\delta$




Propagación del error a la capa II, neurona 2

# Backpropagation: Retropropagación en la capa III

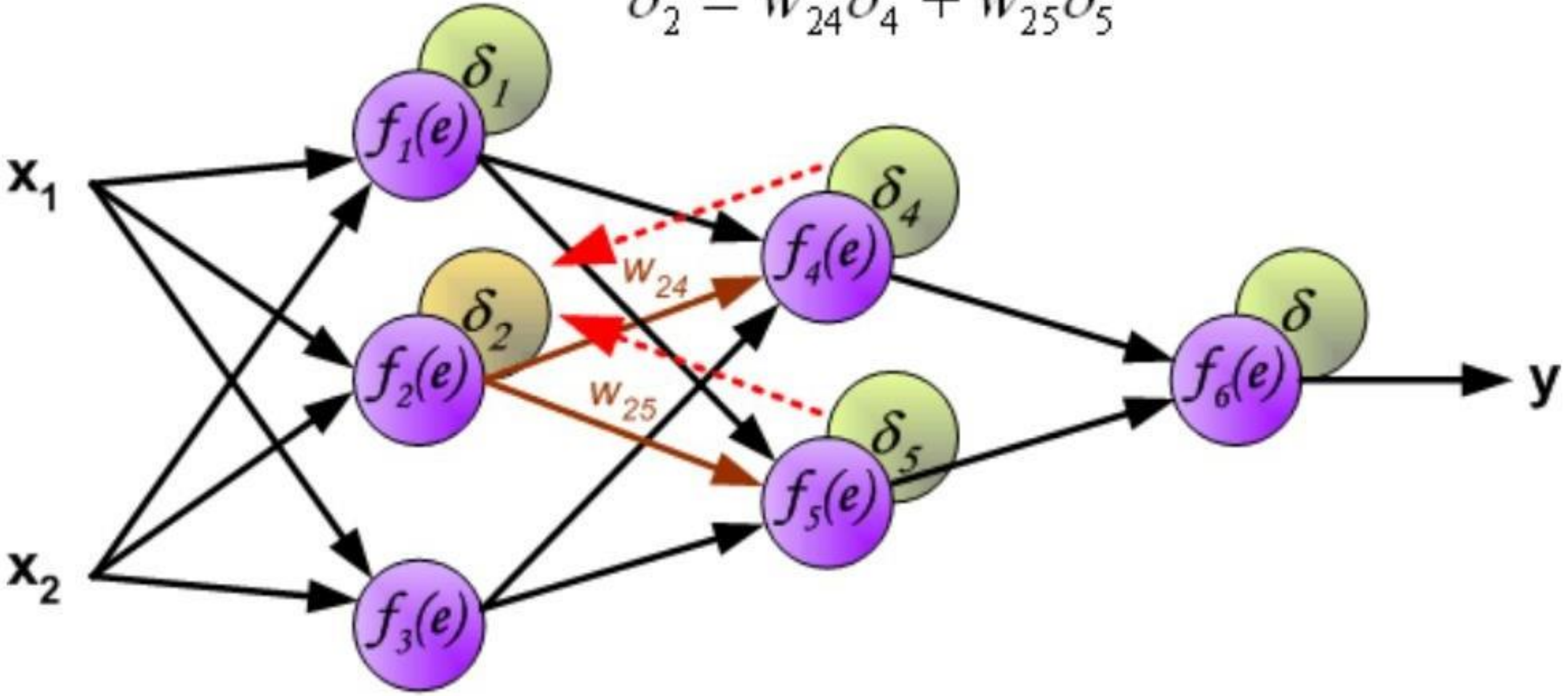
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$




Propagación del error a la capa I, neurona 1

# Backpropagation: Retropropagación en la capa III

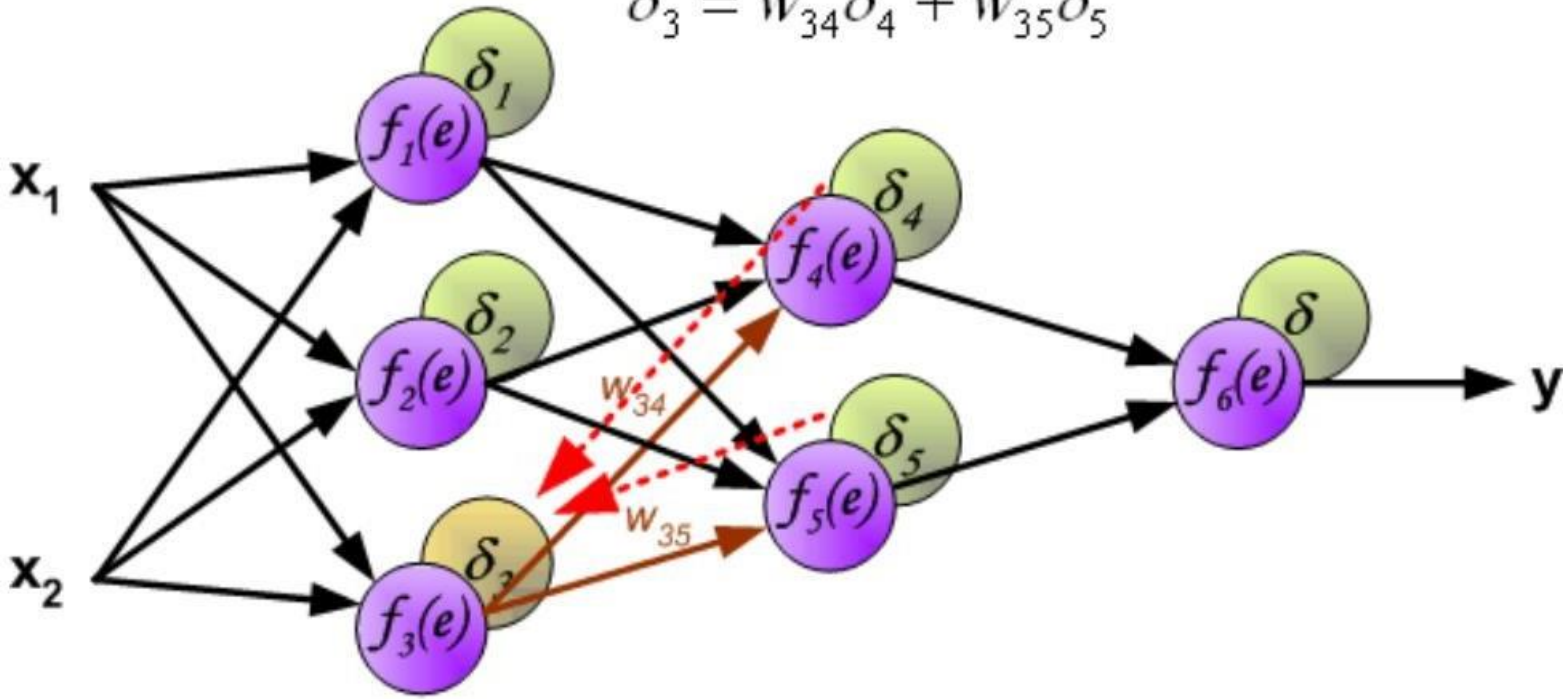
$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$




Propagacion del error a la capa I, neurona 2

# Backpropagation: Retropropagación en la capa III

$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$

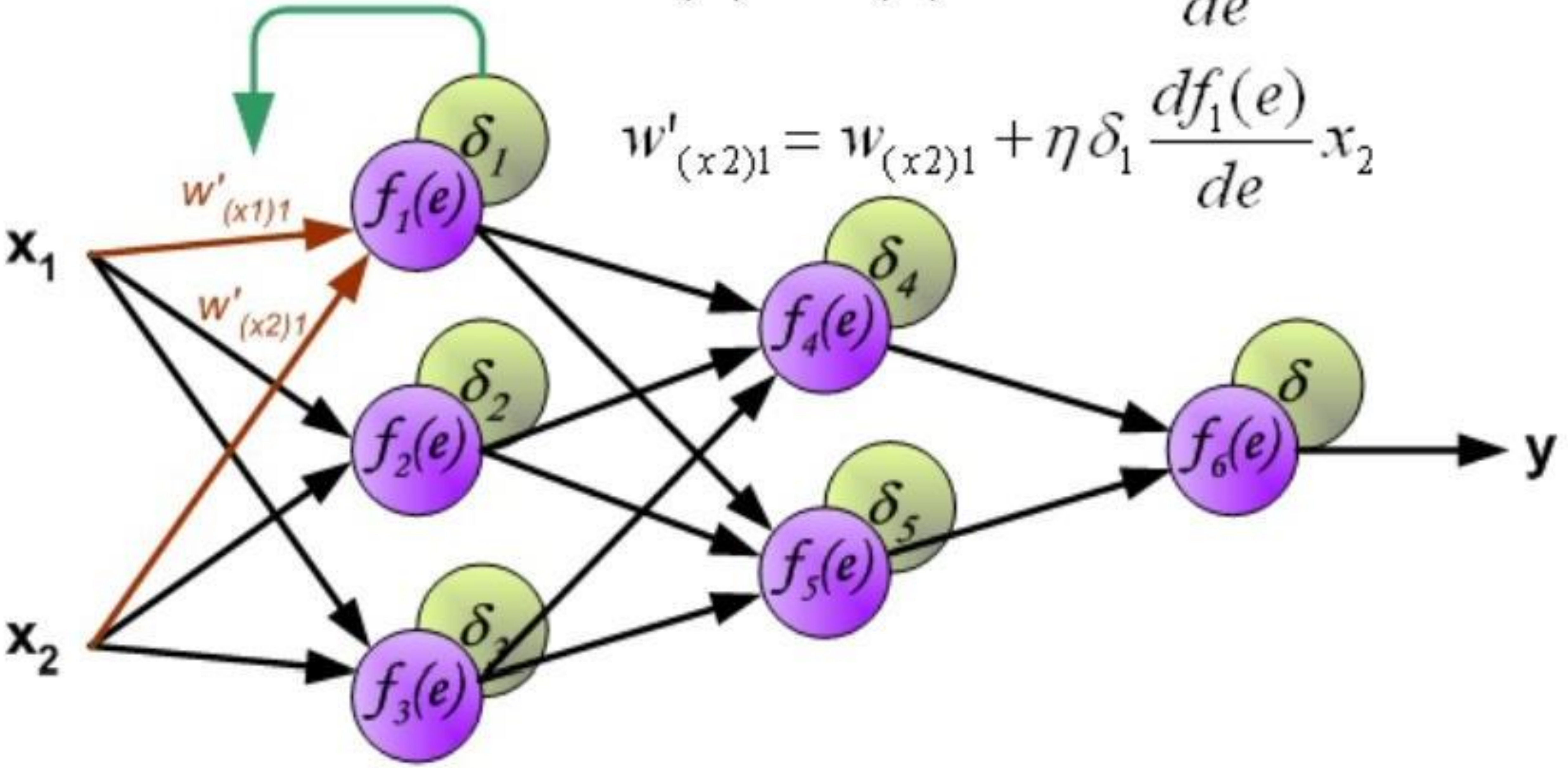



Propagación del error a la capa I, neurona 3

# Backpropagation: Actualizando los pesos de la red

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

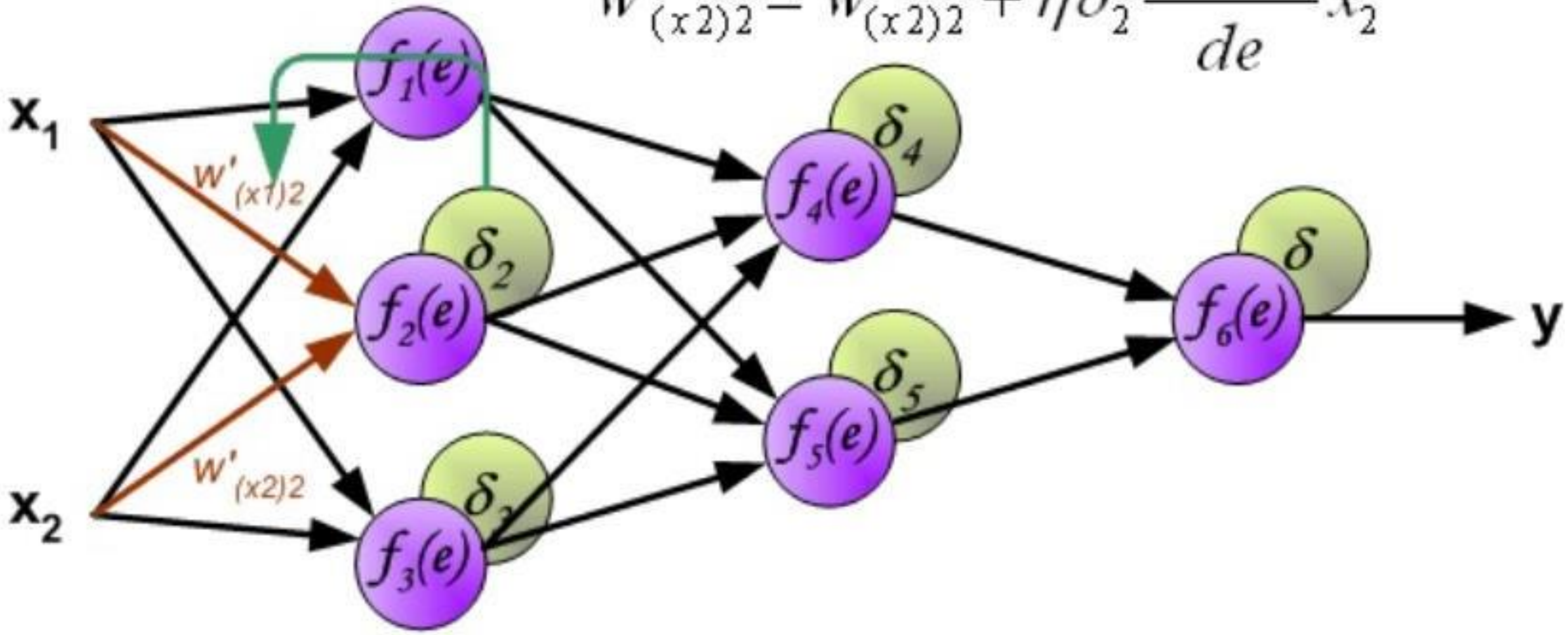



Actualización de pesos capa I, neurona 1

# Backpropagation: Actualizando los pesos de la red

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

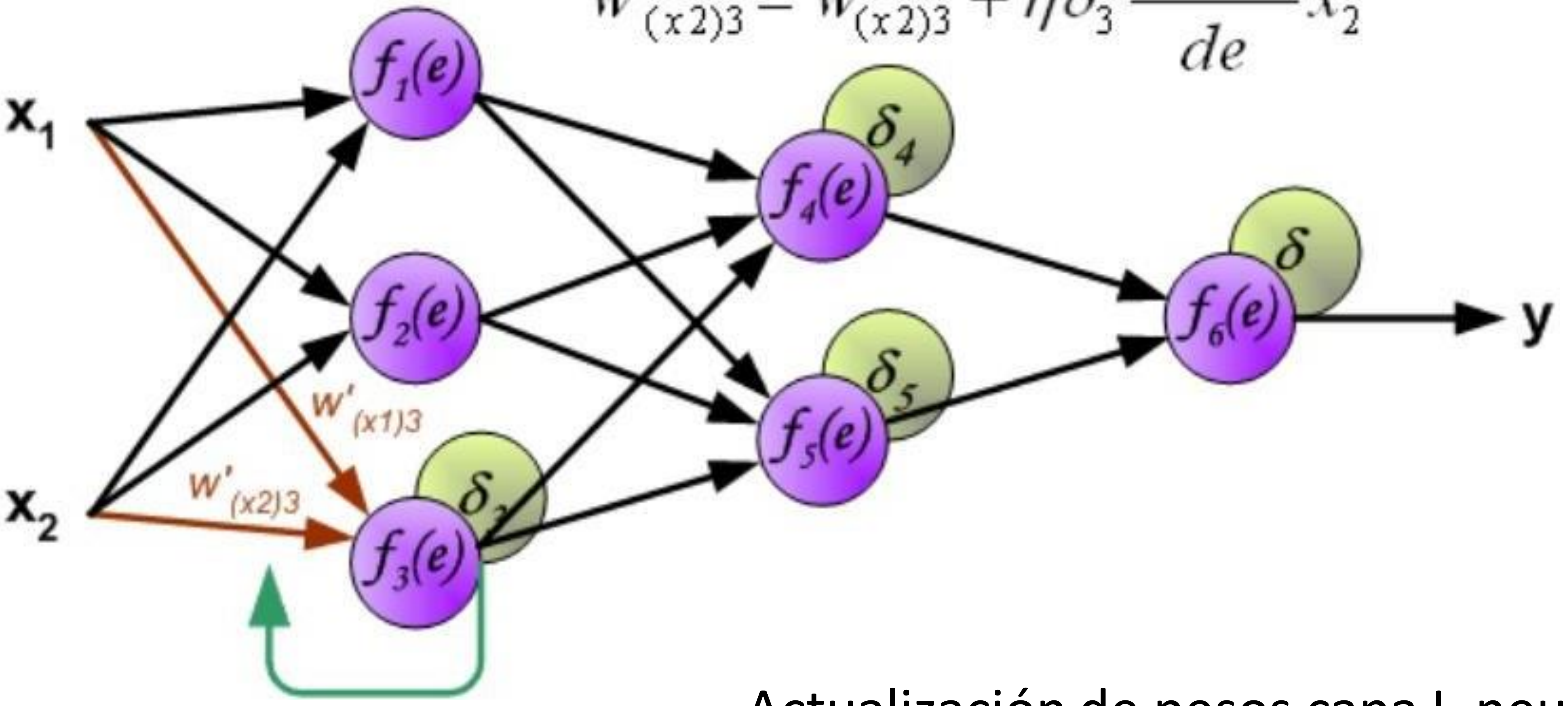



Actualización de pesos capa I, neurona 2.

# Backpropagation: Actualizando los pesos de la red

$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$



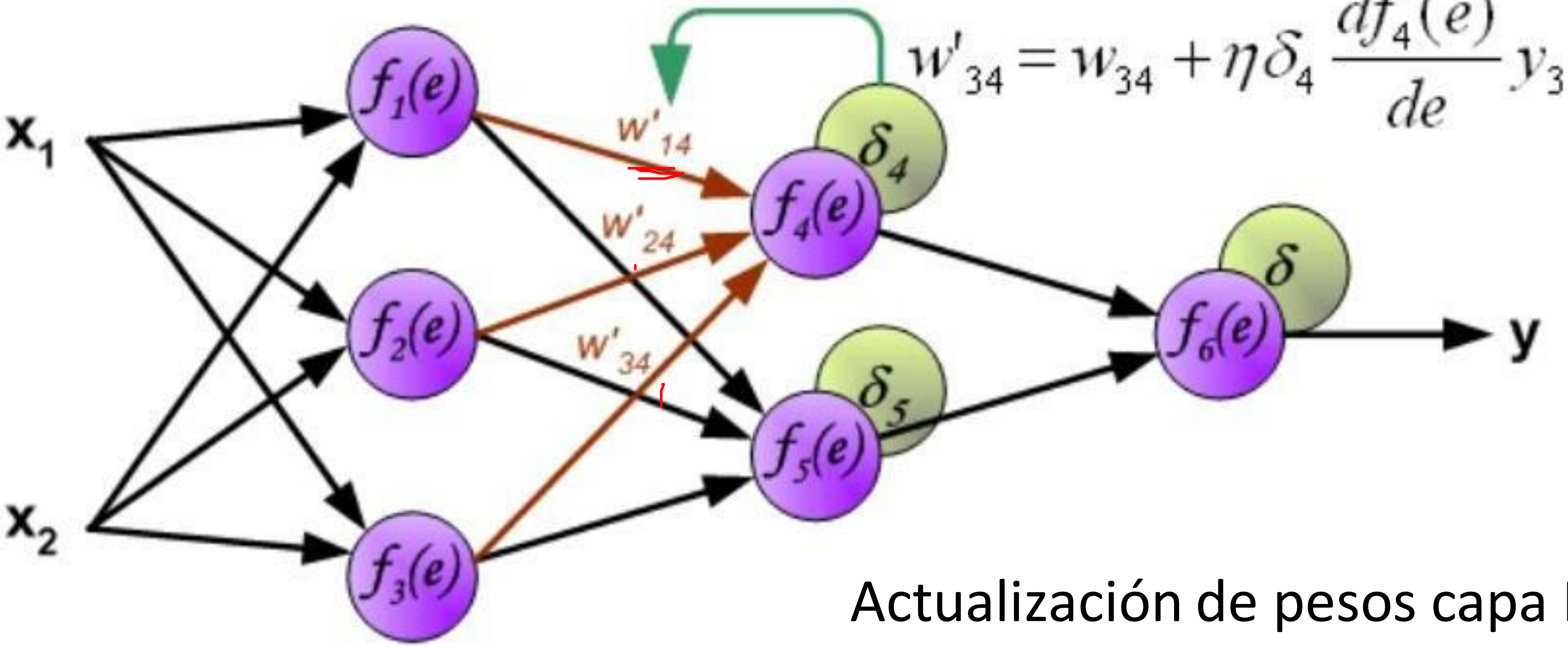

Actualización de pesos capa I, neurona 3.

# Backpropagation: Actualizando los pesos de la red

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



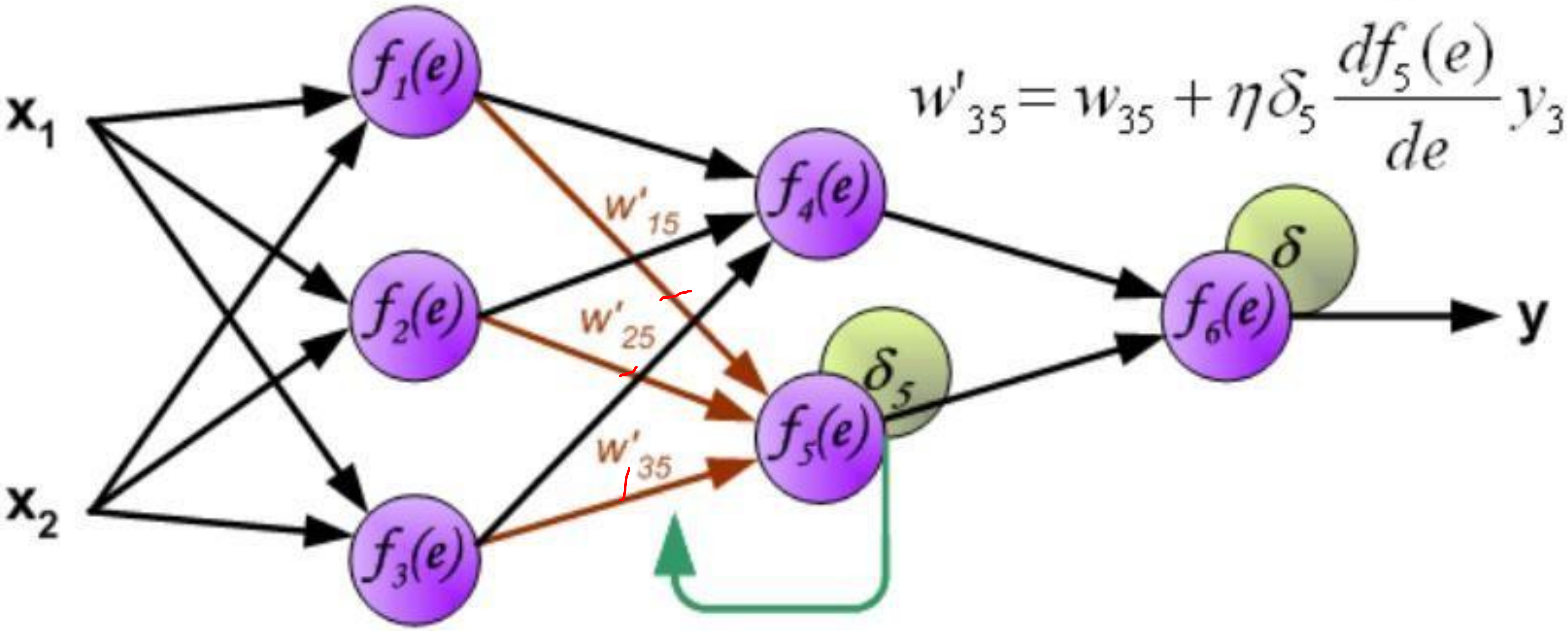

Actualización de pesos capa II, neurona 1.

# Backpropagation: Actualizando los pesos de la red

$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$

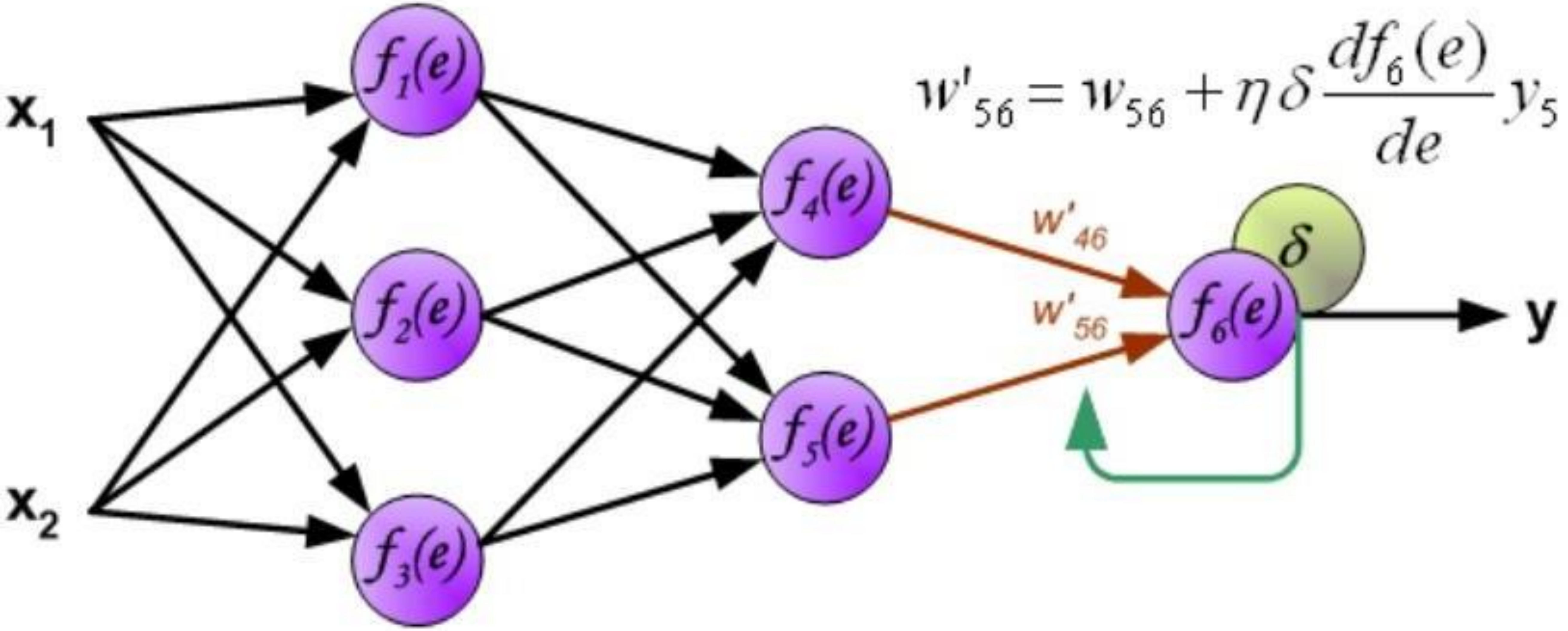



Actualización de pesos capa II, neurona 2.

# Backpropagation: Actualizando los pesos de la red

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



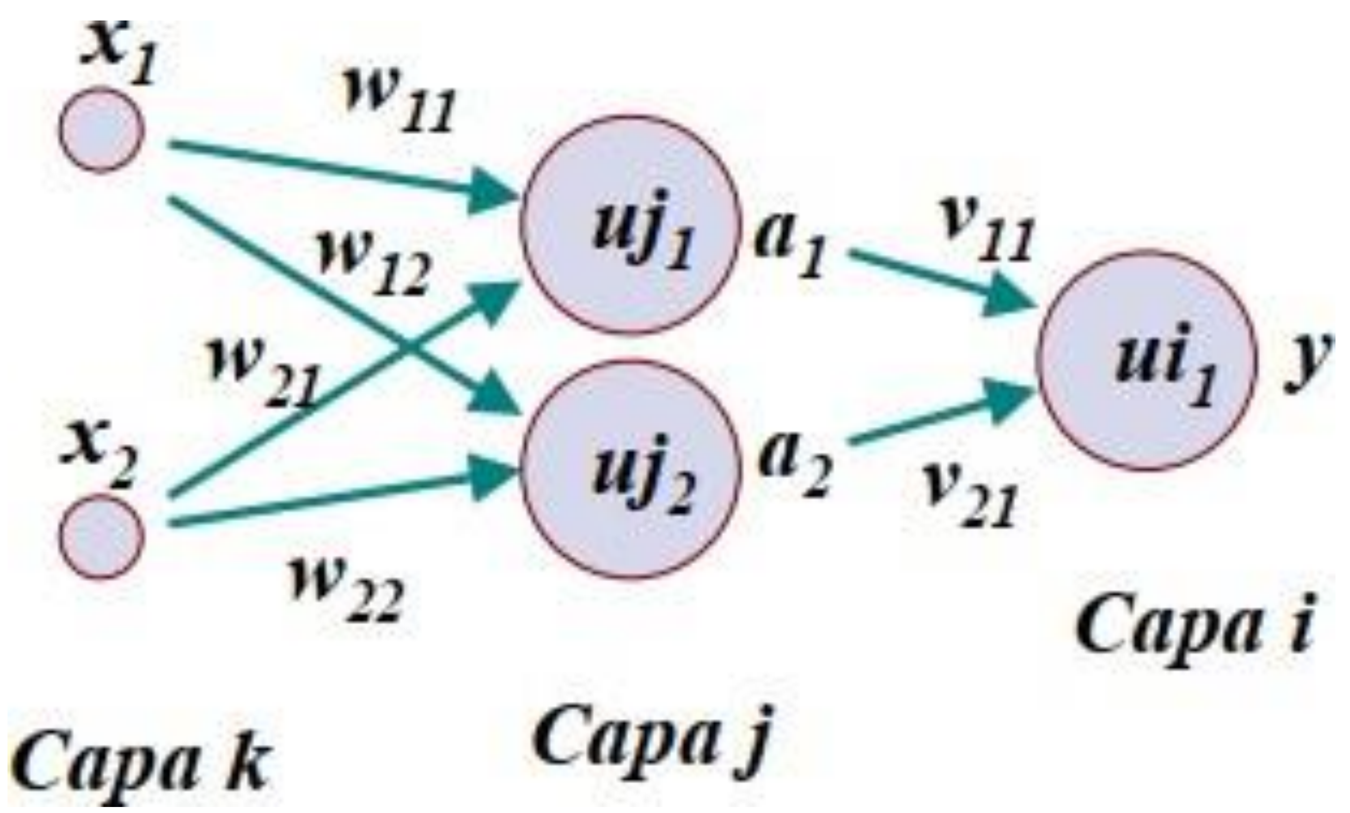

Actualización de pesos capa III, neurona 1.

# Backpropagation: Consideraciones

- Se debe comenzar siempre con pesos iniciales aleatorios pequeños, tanto positivos como negativos.
- Éste es un método de aprendizaje general que presenta como ventaja principal el hecho de que se puede aplicar a gran número de problemas distintos, proporcionando buenas soluciones con no demasiado tiempo de desarrollo.
- Sin embargo si se pretende afinar más y obtener una solución más óptima habría que ser cuidadoso en no caer en el sobreajuste del modelo.

# Perceptrón Multicapa: Ejemplo XOR

El perceptrón para modelar la función XOR presenta la siguiente estructura:




Donde:
$u$ es el umbral,
$a$ son las salidas de la capa $j$ y
$v$ son los pesos para la capa $i$.

# Perceptrón Multicapa: Ejemplo XOR

La actualización de pesos y umbrales se realiza de la siguiente manera:

**Capa $i$**

$$v_{11} = v'_{11} + \alpha \cdot \delta^{i} \cdot a_1$$

$$v_{12} = v'_{12} + \alpha \cdot \delta^{i} \cdot a_2$$

$$ui_1 = ui'_1 + \alpha \cdot \delta^{i}$$

$$\delta^{i} = (s - y) \cdot y \cdot (1 - y)$$

*s es la salida deseada*

**Capa $j$**

$$w_{11} = w'_{11} + \alpha \cdot \delta_1^{j} \cdot x_1$$

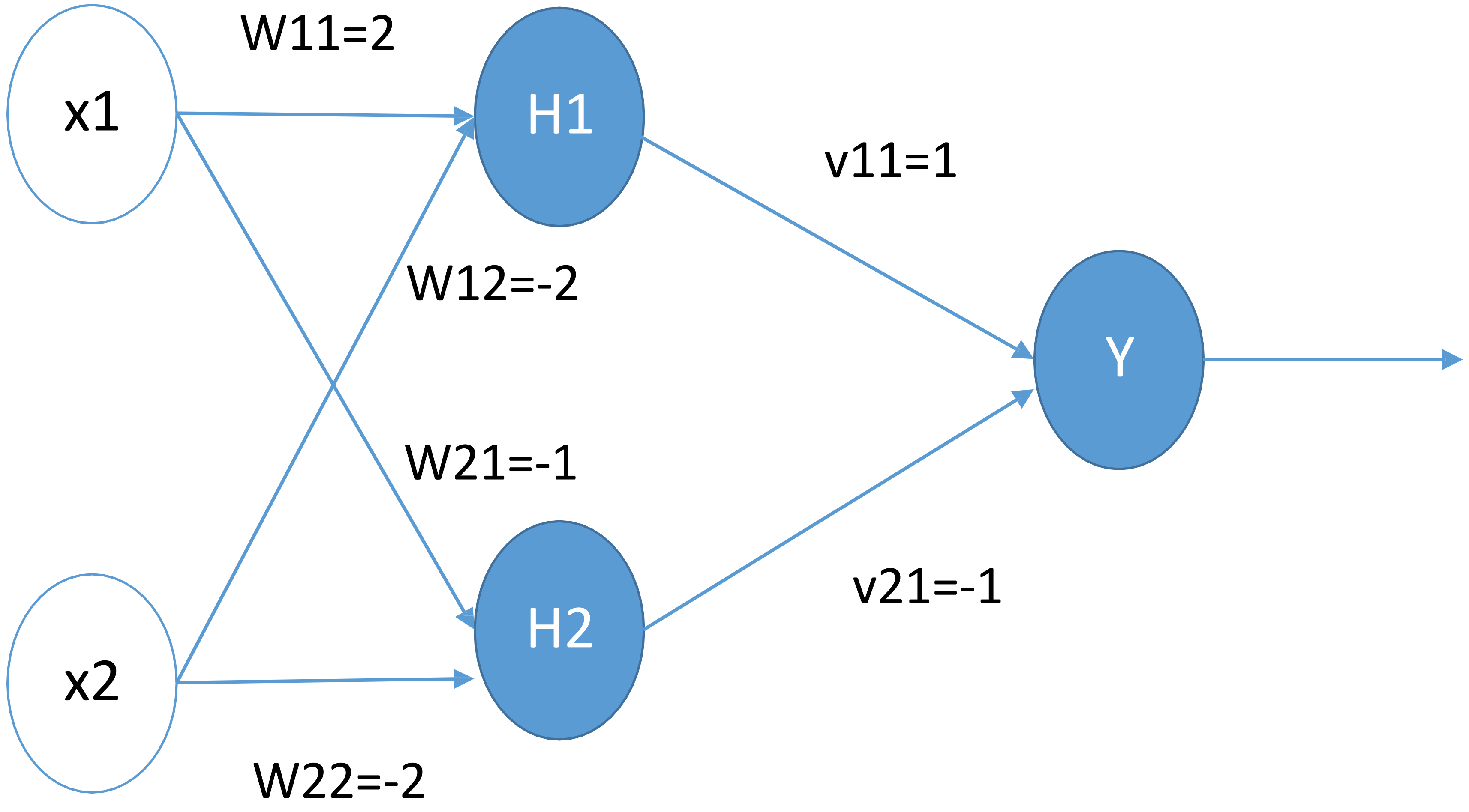
$$\cdots$$

$$uj_1 = uj'_1 + \alpha \cdot \delta_1^{j}$$

$$\delta_1^{j} = a_1(1 - a_1) \sum_{\forall i} v_{1i} \delta^{i}$$

# Ejemplo: Red Multicapa para XOR

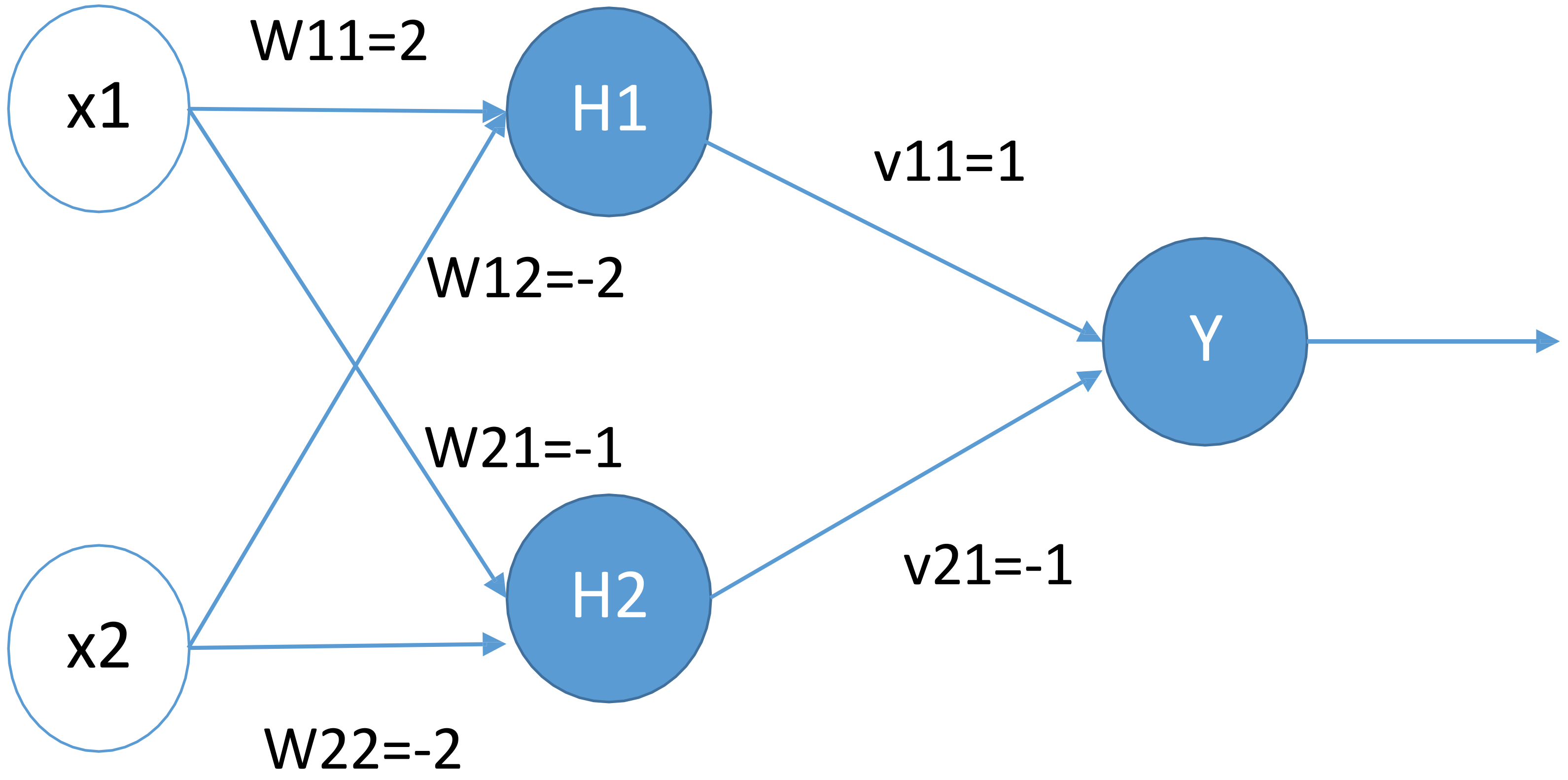
| $X_1$ | $X_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |




Para ejemplificar usaremos la función identidad como función de activación.

$f(x) = x$

| $X_1$ | $X_2$ | XOR |
| --- | --- | --- |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |




H1 = W11 * X1 + W12*X2
H1 = 2*0 -2*0 = 0
f(H1) = 0

H2 = W21 * X1 + W22*X2
H2 = -1*0 – 2*0 = 0
f(H2) = 0

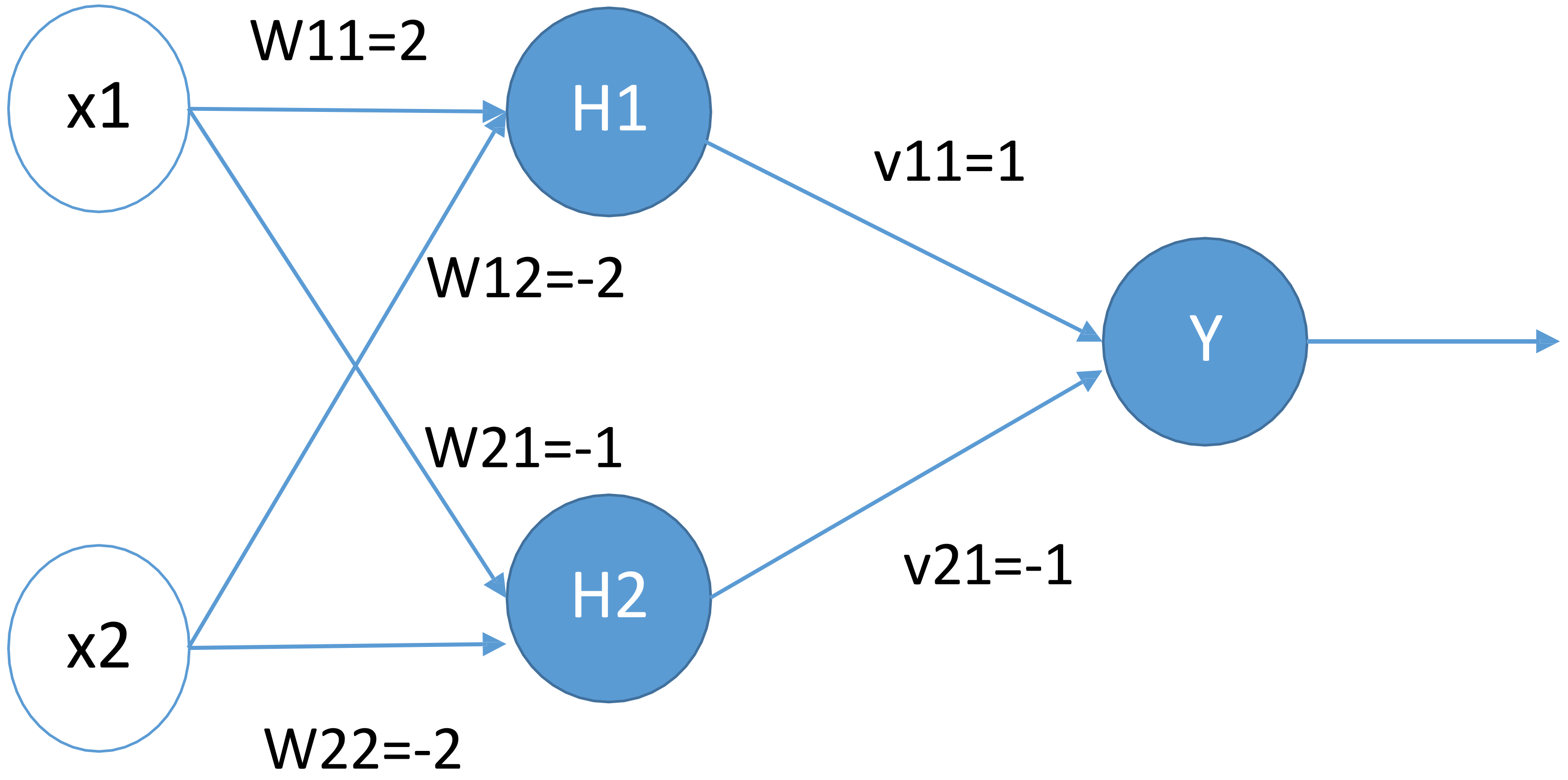
Y = v11*H1 + V21*H2
Y = 1*0 -1*0 = 0
f(Y) = 0

delta_y = z – y
delta_y = 0 - 0 = 0

delta_H1 = v11*delta_y
delta_H1 = 1 * 0 = 0

delta_H2 = v21*delta_y
delta_H2 = -1 * 0 = 0

| $X_1$ | $X_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |




delta_y = z – y
delta_y = 0  - 0  = 0

delta_H1 =  v11*delta_y
delta_H1 = 1 * 0 = 0

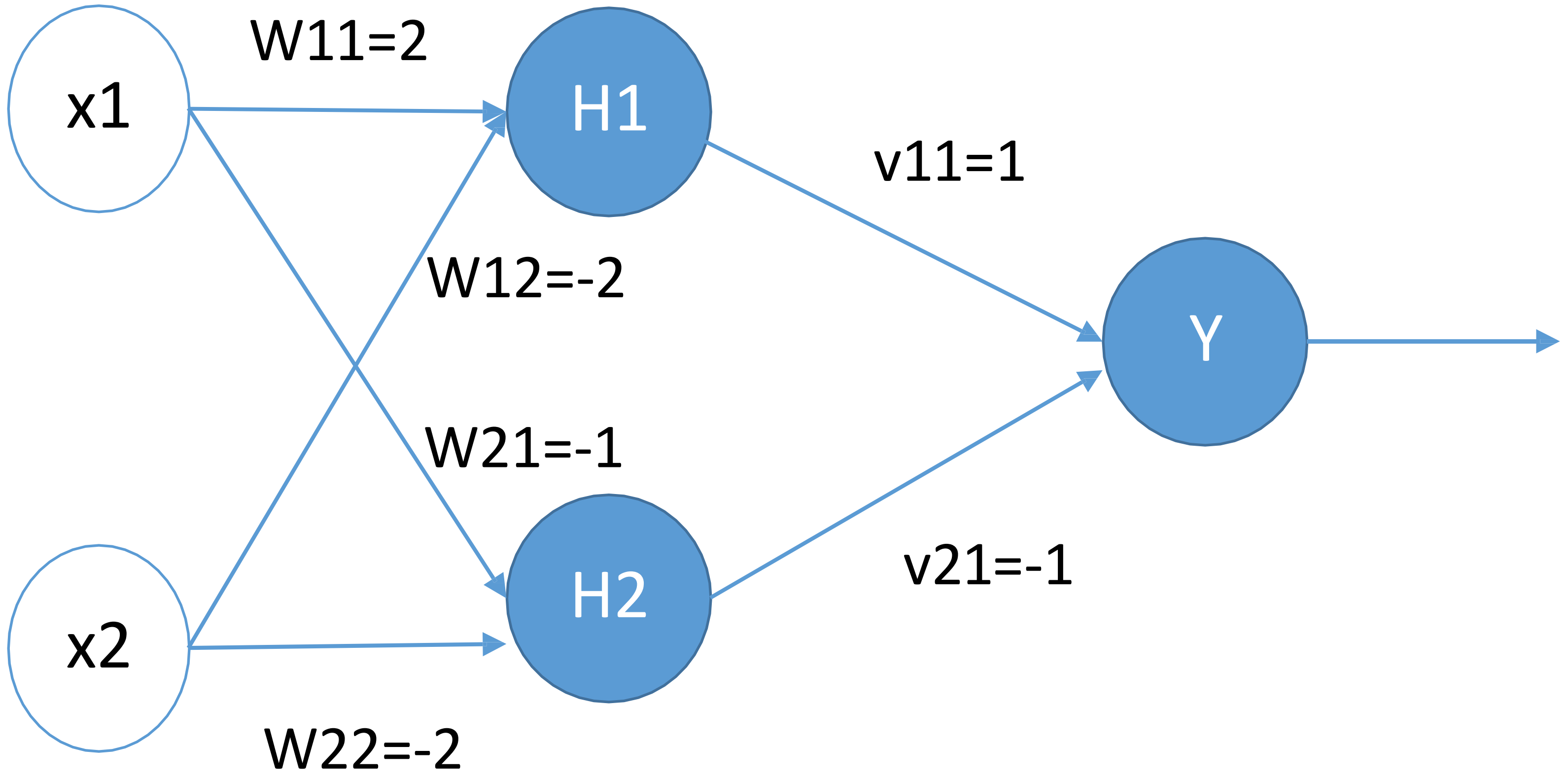
delta_H2 =  v21*delta_y
delta_H2 = -1 * 0 = 0

W11 = w11 + 0.3*delta_H1*X1
W11 = 2 + 0.3*0*0 = 2

W12 = w12 + 0.3*delta_H1*X2
W12 = -2 + 0.3*0*0 = -2

W21 = w21 + 0.3*delta_H2*X1
W21 = -1 + 0 = -1
W22 =  w22 + 0.3*delta_H2*X2
W22 = -2 + 0 = -2

| $X_1$ | $X_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |




delta_y = z – y
delta_y = 0  - 0  = 0

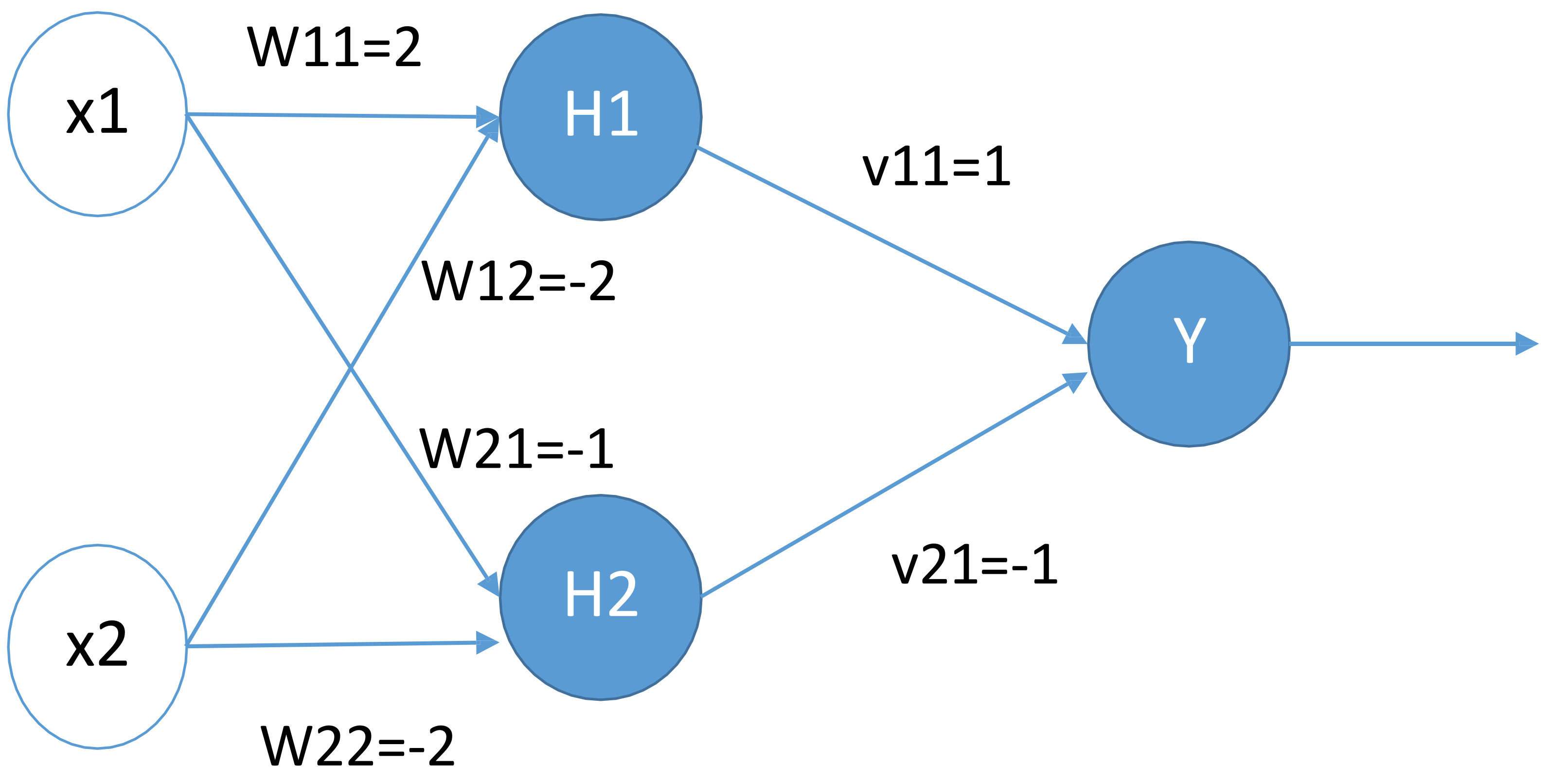
delta_H1 =  v11*delta_y
delta_H1 = 1 * 0 = 0

delta_H2 =  v21*delta_y
delta_H2 = -1 * 0 = 0

v11 = v11 + 0.3*delta_Y*H1
v11 = 1 + 0 = 1

v21 = v21 + 0.3*delta_Y*H2
V21 = -1 + 0 = -1

| $X_1$ | $X_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |




H1 = W11 * X1 + W12*X2
H1 = 2*1 - 2*0 = 2
f(H1) = 2

H2 = W21 * X1 + W22*X2
H2 = -1*1 - 2 *0 = -1
f(H2) = -1

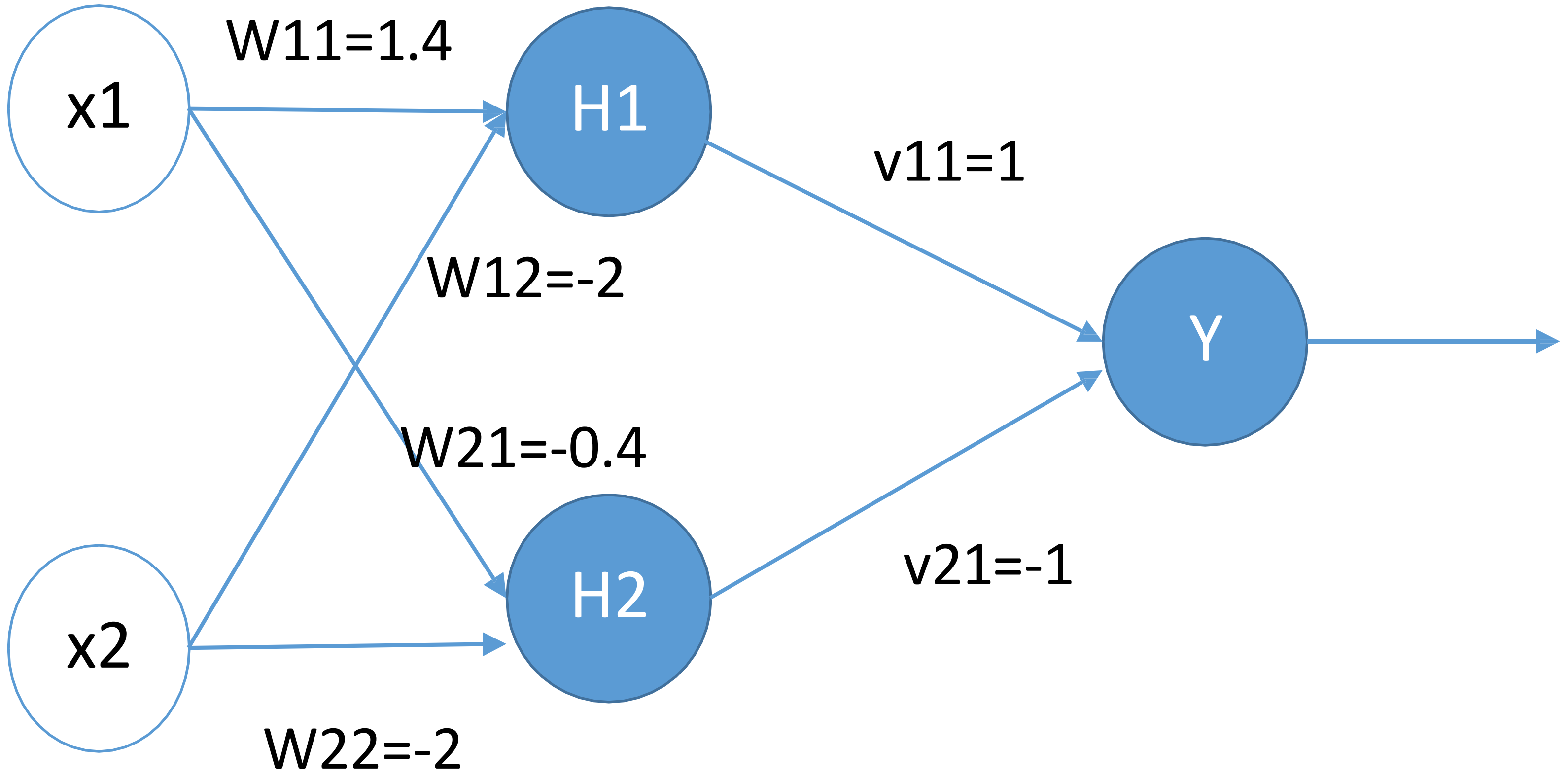
Y = v11*H1 + V21*H2
Y = 1*2 - 1 * -1 = 3
f(Y) = 3

delta_y = z – y
delta_y = 1 – 3 = -2

delta_H1 = v11*delta_y
delta_H1 = 1*-2 = -2

delta_H2 = v21*delta_y
delta_H2 = -1 * -2 = 2

| $X_1$ | $X_2$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |




delta_y = z – y

delta_y = 1 – 3 = -2

delta_H1 = v11*delta_y

delta_H1 = 1*-2 = -2

delta_H2 = v21*delta_y

delta_H2 = -1 * -2 = 2

W11 = w11 + 0.3*delta_H1*X1

W11 = 2 + 0.3*-2*1 = 1.4

W12 = w12 + 0.3*delta_H1*X2

W12 = -2 + 0.3*-2*0 = -2

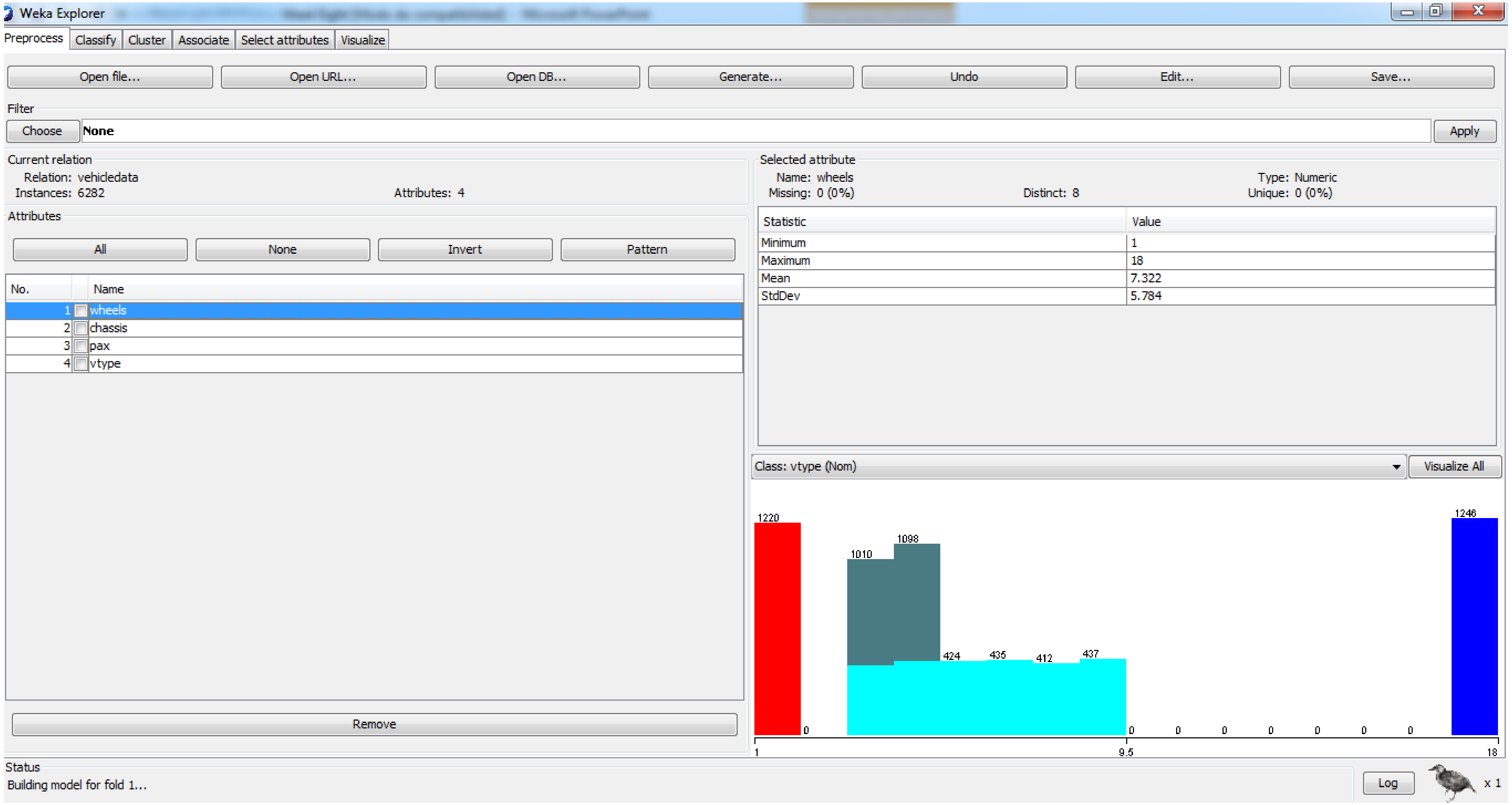
W21 = w21 + 0.3*delta_H2*X1

W21 = -1 + 0.3*2*1 = -0.4

W22 = w22 + 0.3*delta_H2*X2

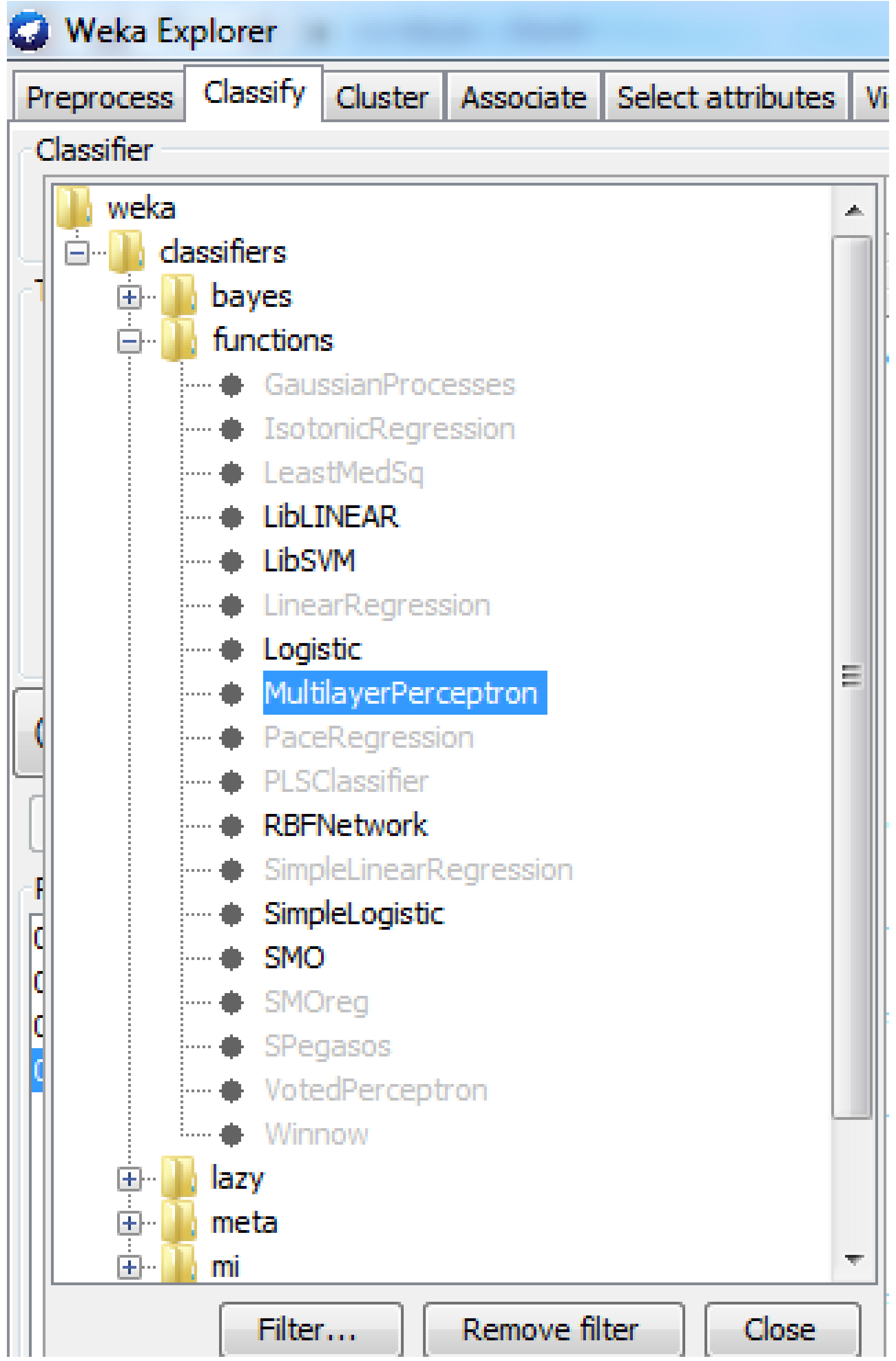
W22 = -2

# Artificial Neural Network in Weka



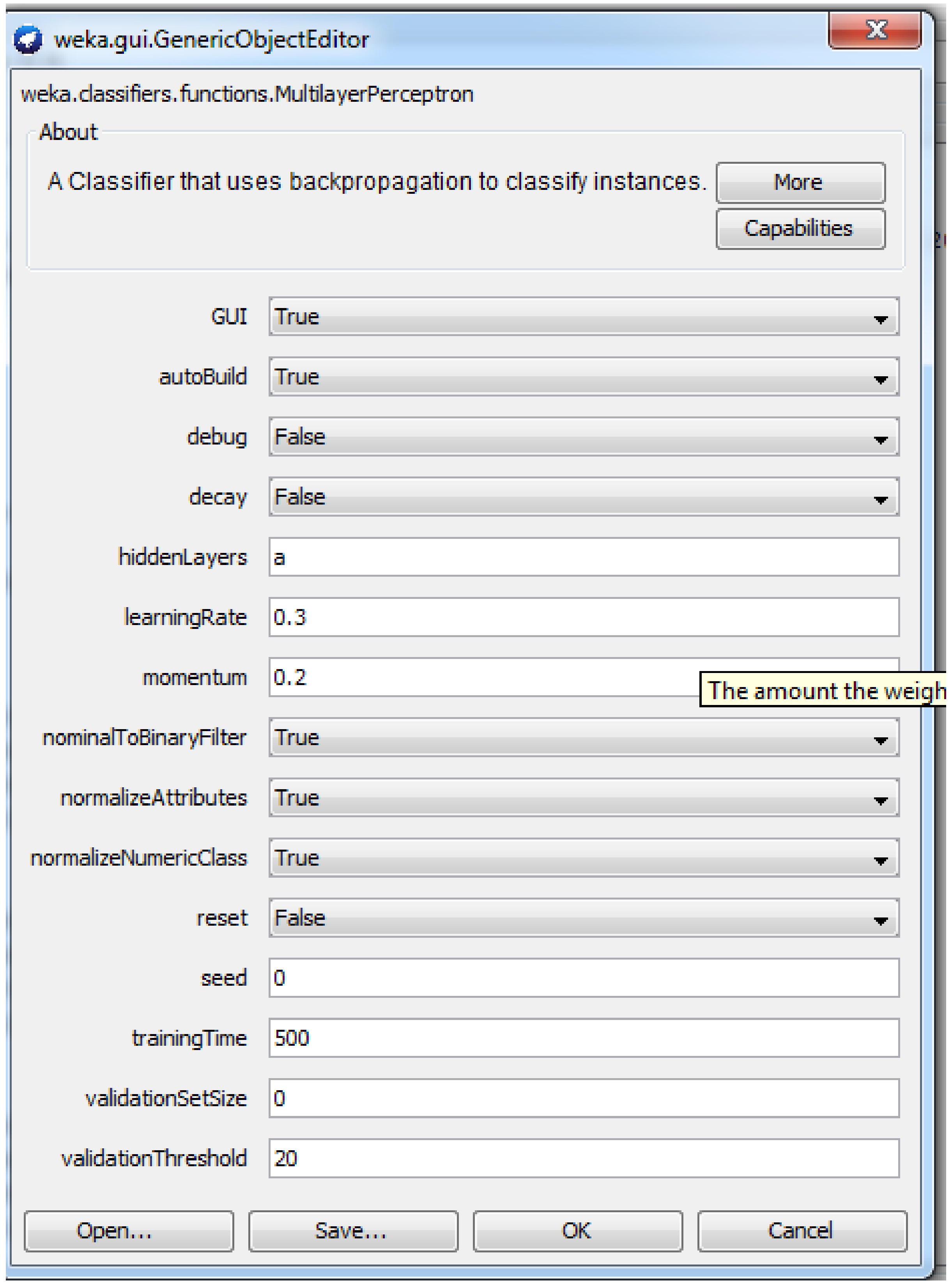
# Artificial Neural Network in Weka

- Go to Classify
- Click in the Choose
- Select Functions -> MultilayerPerceptron
- Selection Options del Classifier



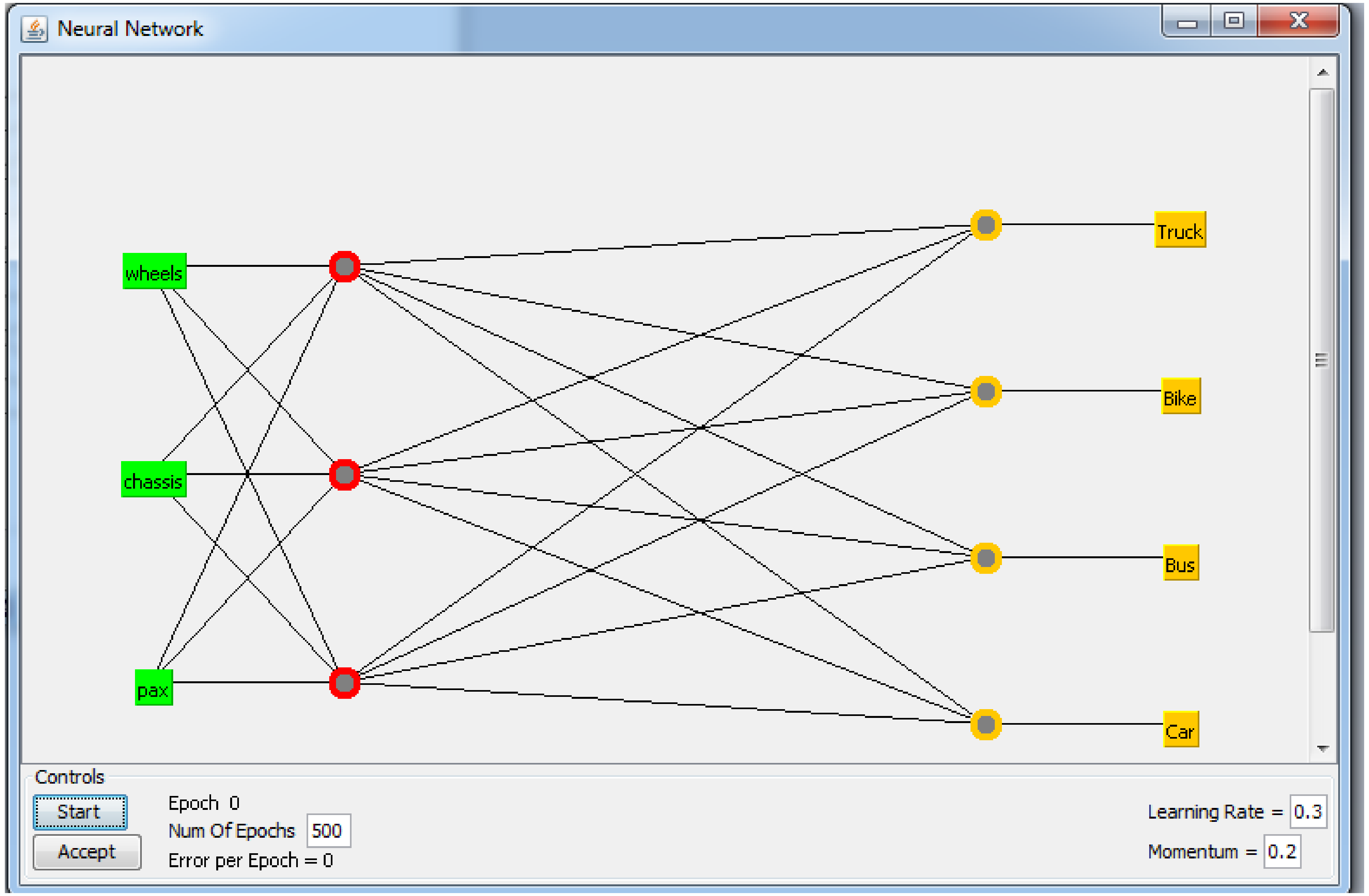
# Artificial Neural Network in Weka

- Select The Options for the Classifier
  - GUI: True
  - HiddenLayer:
    - a: (attribute + classes)/2
    - i: attributes
    - o: classes
    - t: attributes + classes
  - Autobuild
  - Training Time



# Artificial Neural Network in Weka

- Visualize the Neural Network



# Artificial Neural Network in Weka

- Neural Network Trained:
  - Nodes and weigths in the model

```
=== Classifier model (full training set) ===

Sigmoid Node 0
    Inputs    Weights
    Threshold    -5.128060528849861
    Node 4    11.140896690859105
    Node 5    -6.330205998540699
    Node 6    -11.827001525955668
Sigmoid Node 1
    Inputs    Weights
    Threshold    -6.049562146318793
    Node 4    -8.640816687012341
    Node 5    13.837737532993078
    Node 6    -5.371280219663195
Sigmoid Node 2
    Inputs    Weights
    Threshold    -8.430632294334599
    Node 4    2.065964962127525
    Node 5    0.07454198943977772
    Node 6    13.254811024674849
Sigmoid Node 3
    Inputs    Weights
    Threshold    6.820438517413495
    Node 4    -13.104424703205783
    Node 5    -13.83318708732965
    Node 6    -1.7079933680239345
```

# Artificial Neural Network in Weka

- Results of training

```
=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances        6282              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0016
Root mean squared error                  0.0023
Relative absolute error                  0.4508 %
Root relative squared error              0.5439 %
Total Number of Instances             6282
```

```
=== Confusion Matrix ===

    a    b    c    d   <-- classified as
 1246    0    0    0 |    a = Truck
    0 1220    0    0 |    b = Bike
    0    0 2534    0 |    c = Bus
    0    0    0 1282 |    d = Car
```