Redes Multicapa
y
Algoritmos con Retro-propagación

Redes Multicapa
y
Algoritmos Backpropagation

Backpropagation es la generalización de la regla de aprendizaje Widrow-Hoff para **redes multi-capa** and **funciones de transferencia diferenciables no lineales.**

Los vectores de entrada y los correspondientes vectores deseados se usan para entrenar la red hasta que esta pueda **aproximar una funcion**, asocia los vectores de entrada con los vectores de salida específicos, o **clasifican** los vectores de entrada en una forma aproximada a como la define usted.

Arquitectura

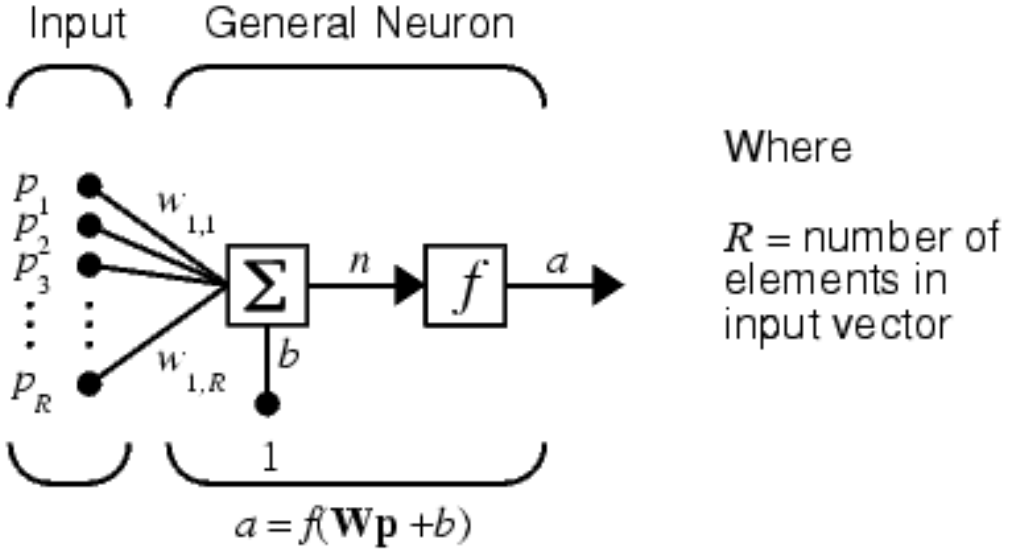
Esta sección presenta la arquitectura de la red que es más comúnmente usada con el algoritmo backpropagation **–**

La red multicapa feedforward (con alimentación hacia adelante)

Arquitectura

Modelo de Neurona

Una neurona elemental con R entradas es mostrada debajo. Cada entrada es ponderada con un w apropiado. La suma de las entradas ponderadas y el bias (sesgo, umbral) forman la entrada a la función de transferencia f. Las neuronas pueden usar cualquier **función de transferencia diferenciable** f para generar su salida.



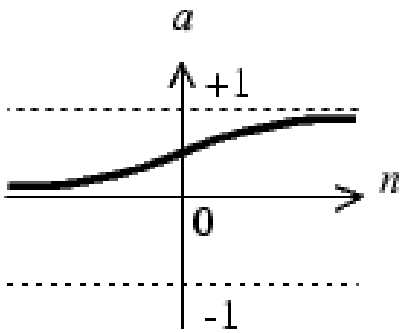
Arquitectura

Modelo Neuronal

Función de transferencia (Función de activación)

Las redes multicapa frecuentemente usan la función de transferencia **log-sigmoid (logsig)**

La función logsig genera las salidas entre **0** y **1** de tal forma que la entrada de la red de neuronas va de negativo a infinito positive.

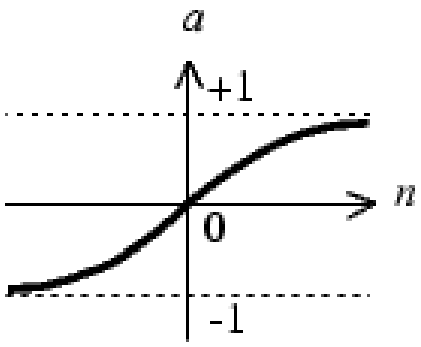


$a = logsig(n)$

Log-Sigmoid Transfer Function

Arquitectura
Modelo Neuronal
Función de transferencia (Función de activación)

Alternativamente, las redes multi-capa pueden usar la función de tranferencia **tan-sigmoid** (**tansig)**.
La función tansig genera salidas entre **-1 y +1** de tal forma que la entrada de la red de neuronas va de negativo a infinito positive.



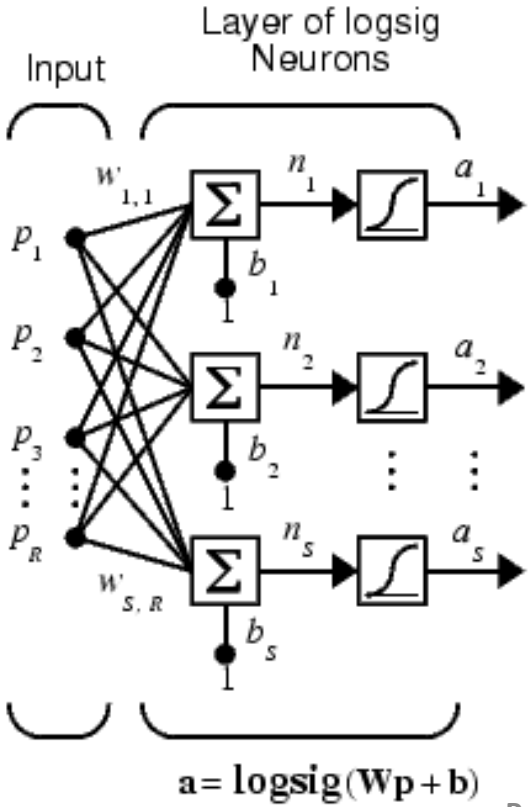
$a = tansig(n)$

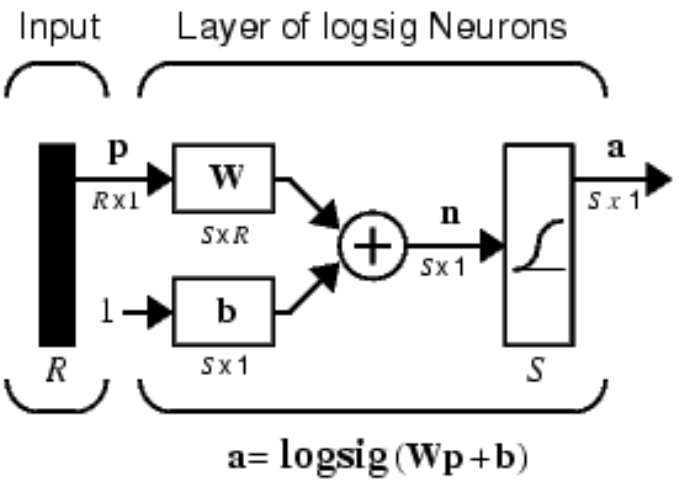
Tan-Sigmoid Transfer Function

Arquitectura

Red Feedforward

Una red con una sola capa de S neuronas logsig con R entradas se muestra debajo con detalles a la izquierda y con un diagrama de capas a la derecha.





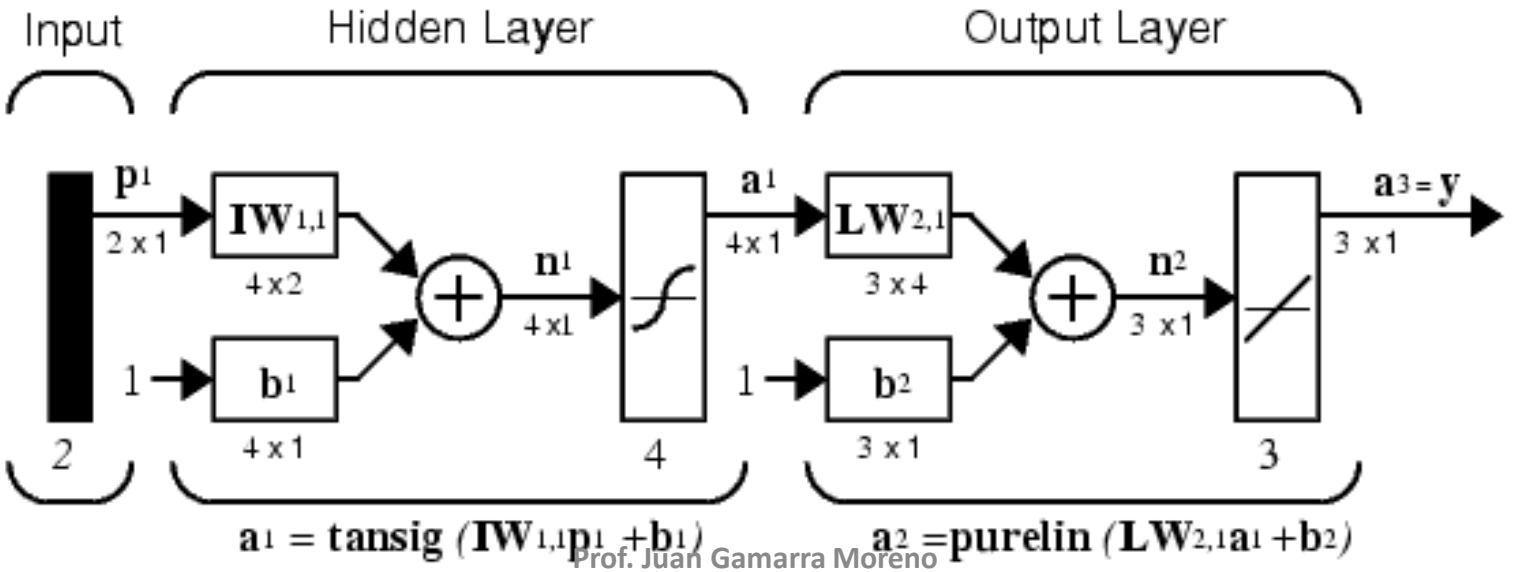
Where...

R = number of elements in input vector

S = number of neurons in layer

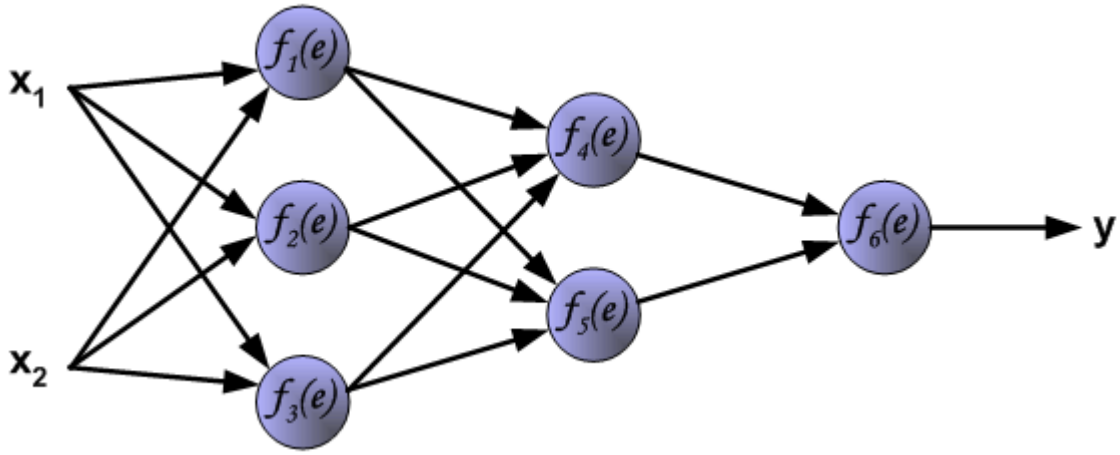
Arquitectura
Red Feedforward

Las redes Feedforward frecuentemente tienen una o más capas ocultas de neuronas sigmoidales seguidas por una capa de neuronas lineales.
Múltiples capas de neuronas con funciones de transferencia no lineales y no lineales permiten relacionar vectores de entrada con los de salida.
La capa de salida lineal permite que la red produzca valores en el rango de -1 a 1. Y además, si quiere restringir las salidad de una red (puede ser entre 0 y 1), la capa de salida debería usar una función de transferencia sigmoidal (tal como logsig).



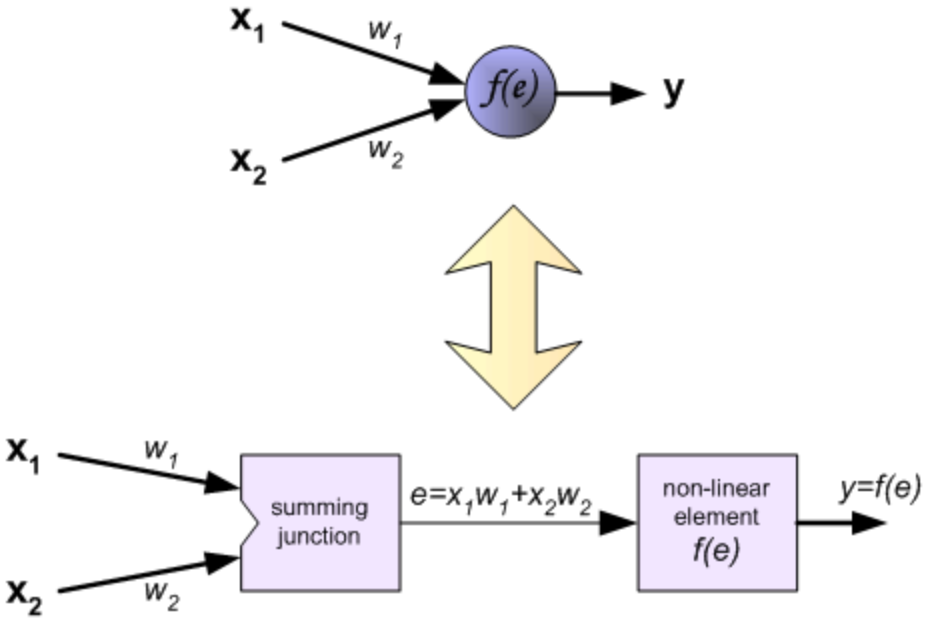
Algoritmo de Aprendizaje: Backpropagation

Aquí se describe el proceso de enseñanza de una red neuronal multicapa usando el algoritmo **backpropagation**. Para ilustrar este proceso se tiene la red neuronal de tres capas con dos entradas y una salida, como se muestra abajo:



Algoritmo de aprendizaje: Backpropagation

Cada neurona se compone de dos unidades. La primera unidad suma los productos de los pesos con la entrada. La segunda unidad ejecuta la función no lineal, llamada función de transferencia (función de activación). El símbolo *e* se agrega a la señal de salida, and *y = f(e)* es la salida del elemento no lineal. La señal *y* es también la señal de salida de una neurona.



Algoritmo de aprendizaje: Backpropagation

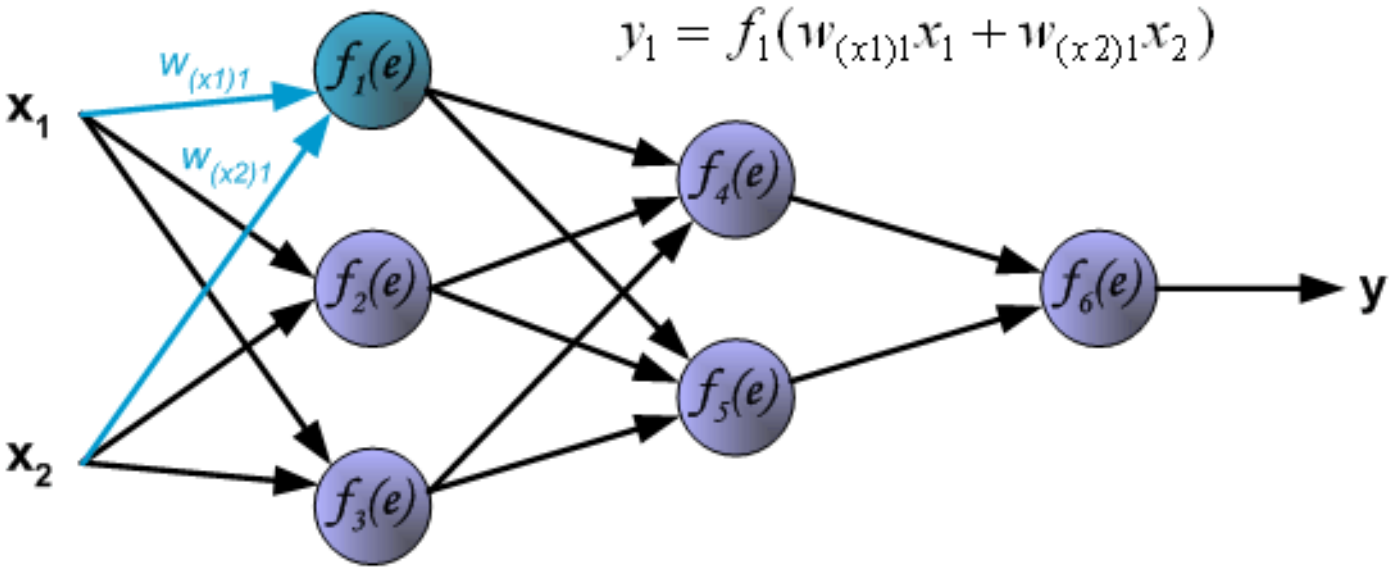
Para enseñar a la red neuronal necesitamos entrenarla con el conjunto de datos de entrenamiento. El conjunto de datos de entrenamiento consiste de señales de entrada (x_1 and x_2) asignados a la meta correspondiente (salida deseada) z.

El entrenamiento de la red es un proceso iterativo. En cada iteración los coeficentes de los pesos de los nodos se modifican usando nuevos datos del conjunto de datos de entrenamiento. La modificación se calcula usando el algoritmo descrito.

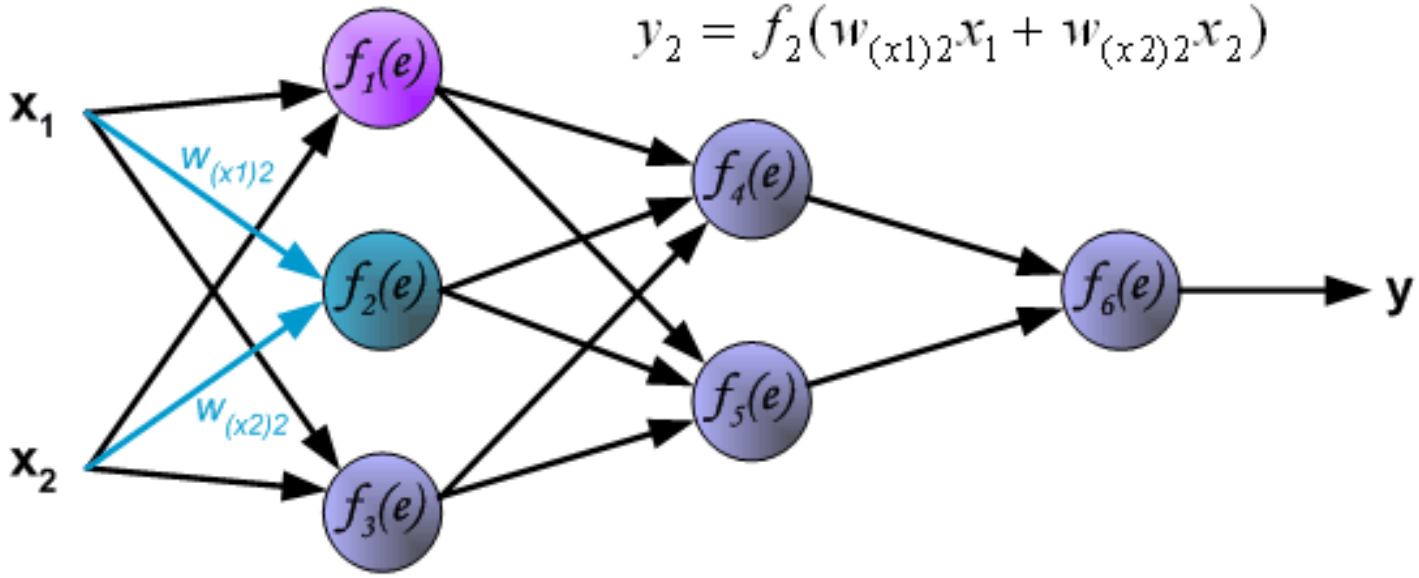
Cada paso de enseñanza comienza forzando ambas señales de entrada del conjunto de entrenamiento. Después de cada etapa podemos determinar los valores de salida para cada neurona en cada capa de la red.

Algoritmo de aprendizaje: Backpropagation

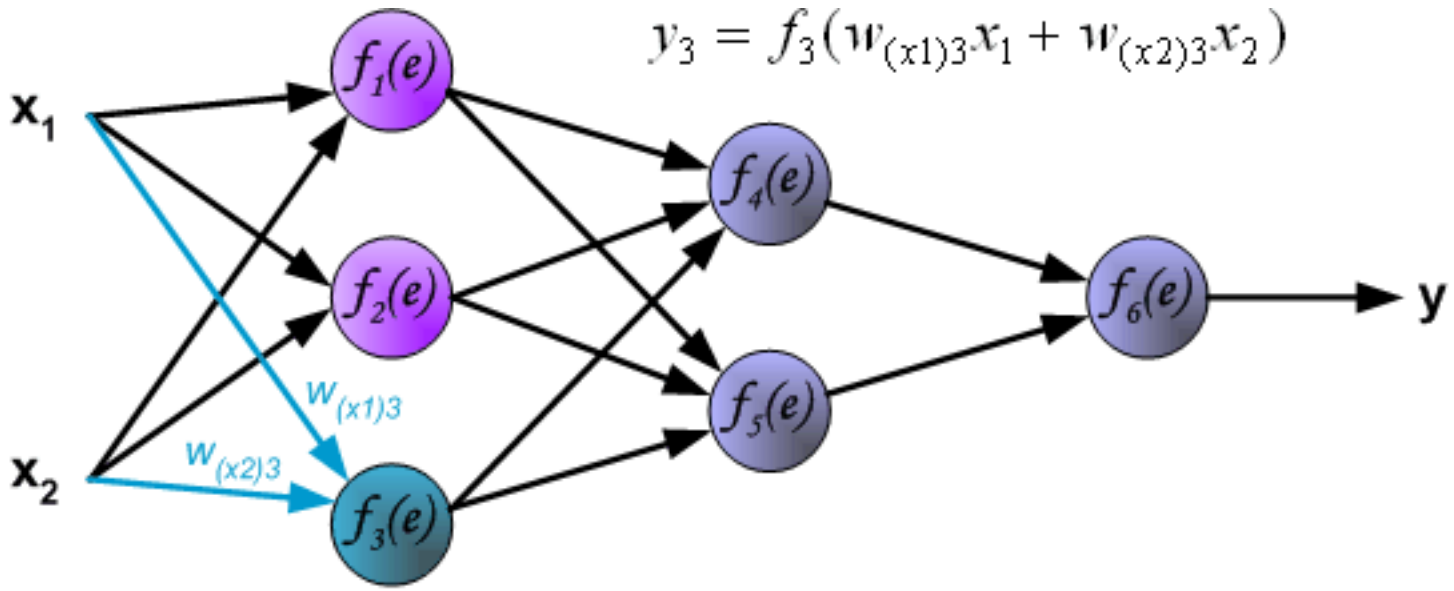
La figura ilustra como se propaga la señal a través de la red, Los Simbolos $w_{(xm)n}$ representan los pesos de las conexiones entre la entrada de la red x_m y la neurona *n* en la capa de entrada. El Símbolo y_n representa la señal de salida de la neurona *n*.



Algoritmo de aprendizaje: Backpropagation

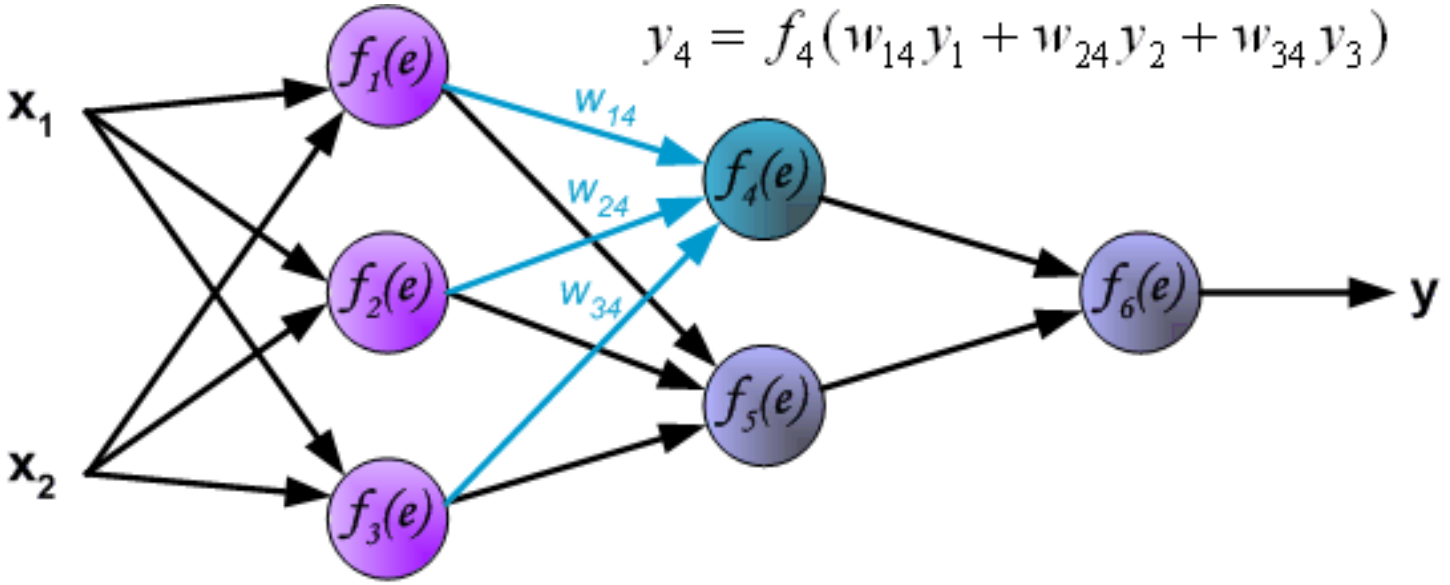


Algoritmo de aprendizaje: Backpropagation

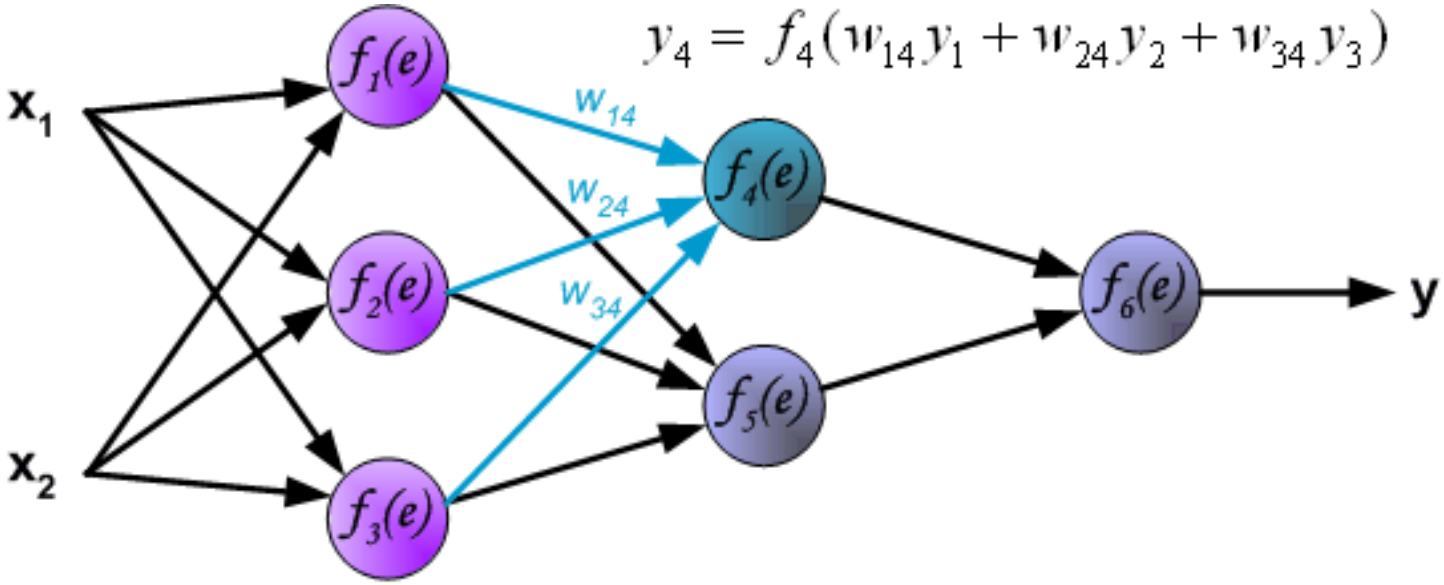


Algoritmo de aprendizaje: Backpropagation

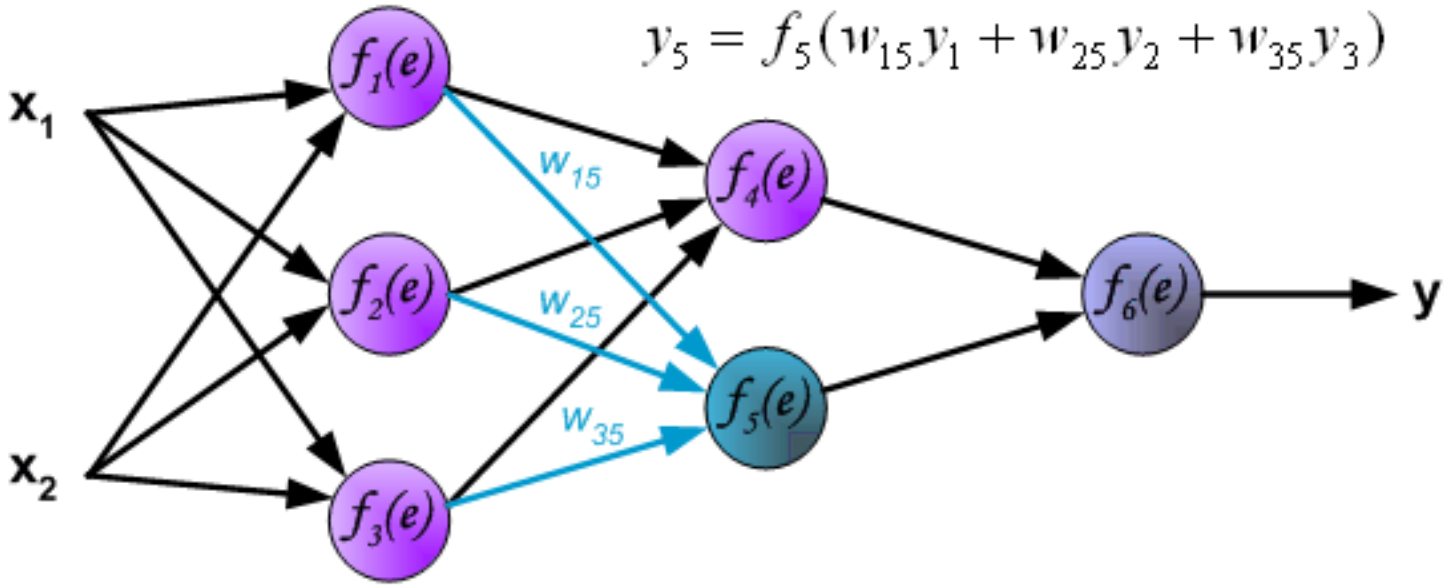
Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.



Algoritmo de aprendizaje: Backpropagation

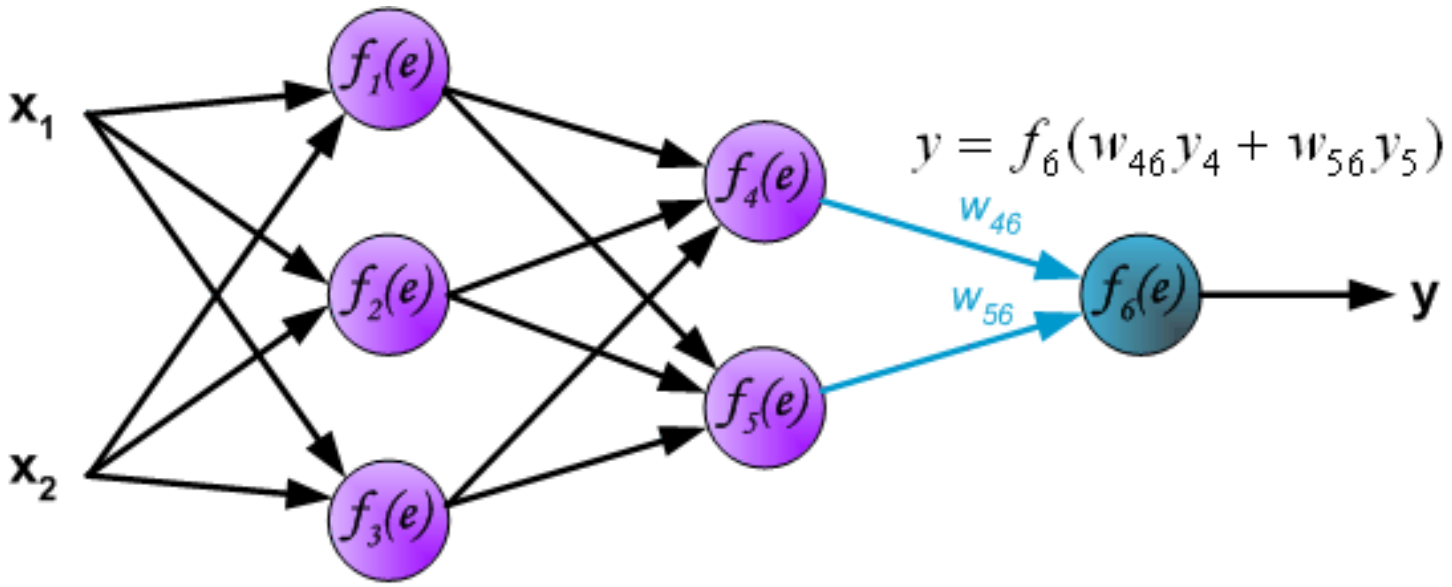


Algoritmo de aprendizaje: Backpropagation



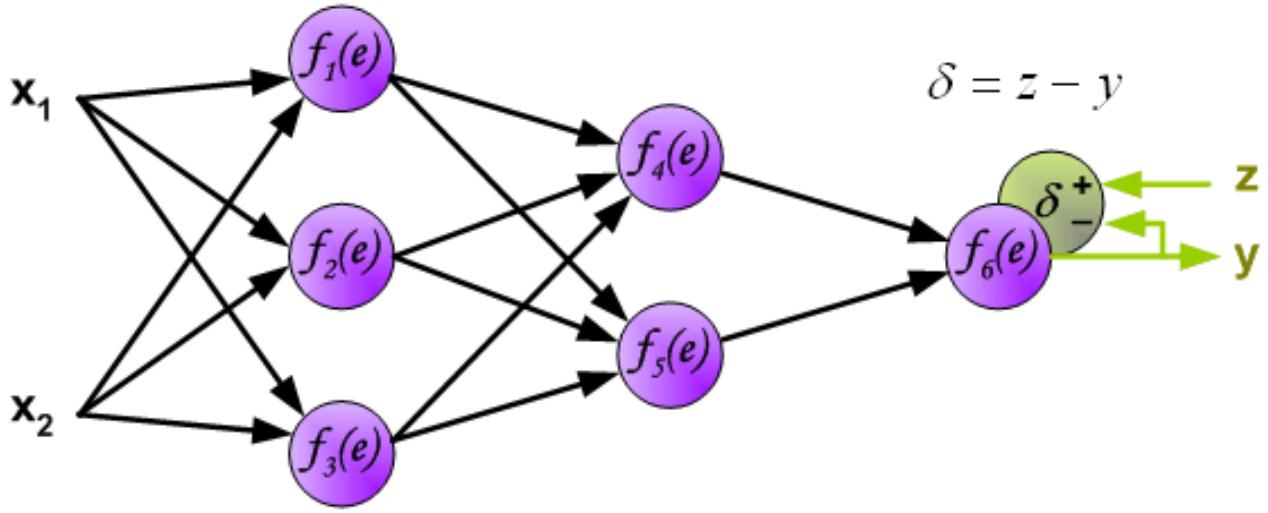
Algoritmo de aprendizaje: Backpropagation

Propagation of signals through the output layer.



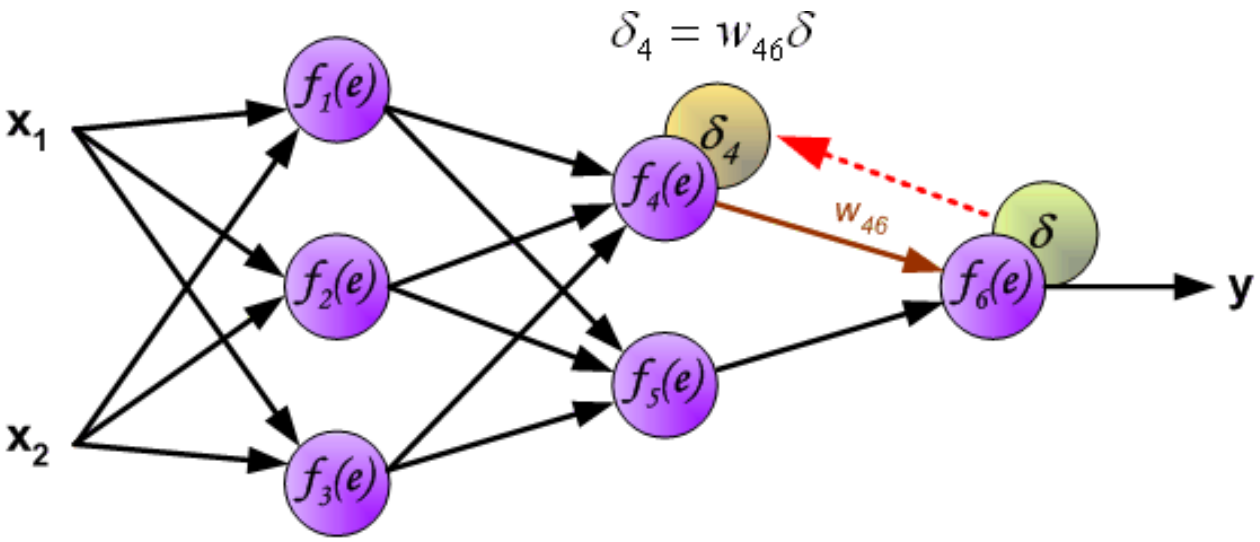
Algoritmo de aprendizaje: Backpropagation

En el próximo paso la señal de salida *y* de la red se compara con el valor de salida deseado (la meta), la que se encuentra en el conjunto de datos de entrenamiento. La diferencia es el error de la neurona de la capa de salida.

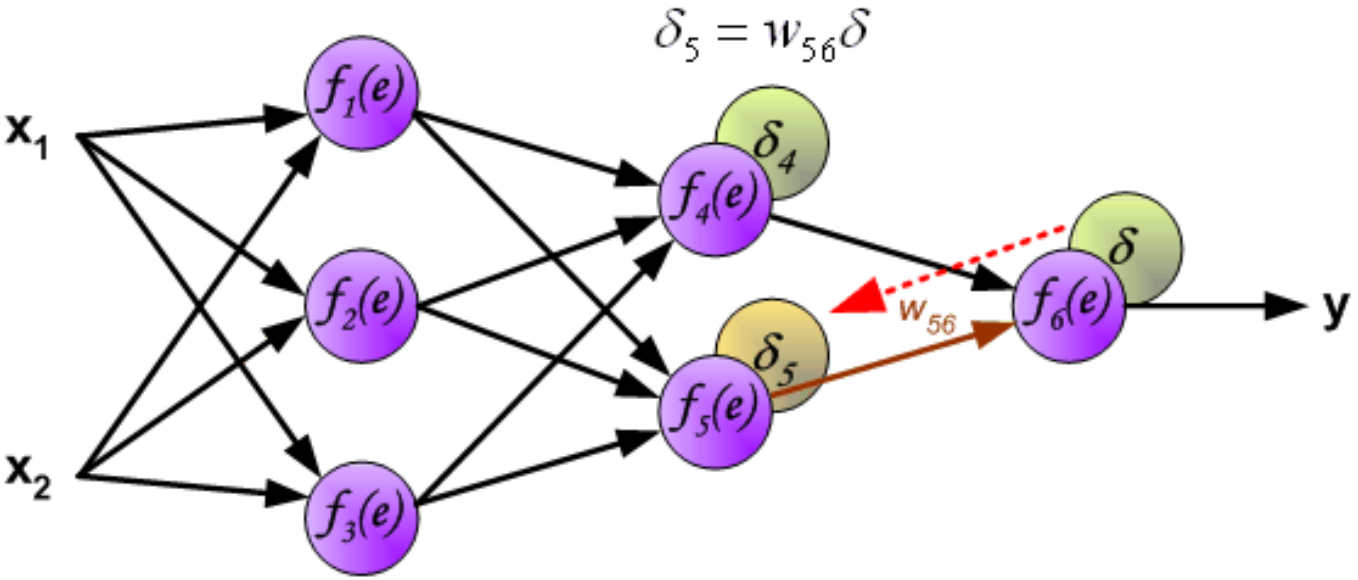


Algoritmo de aprendizaje: Backpropagation

La idea es propagar el error δ hacia atrás a todas las neuronas las cuales fueron entradas para la neurona en discusión.

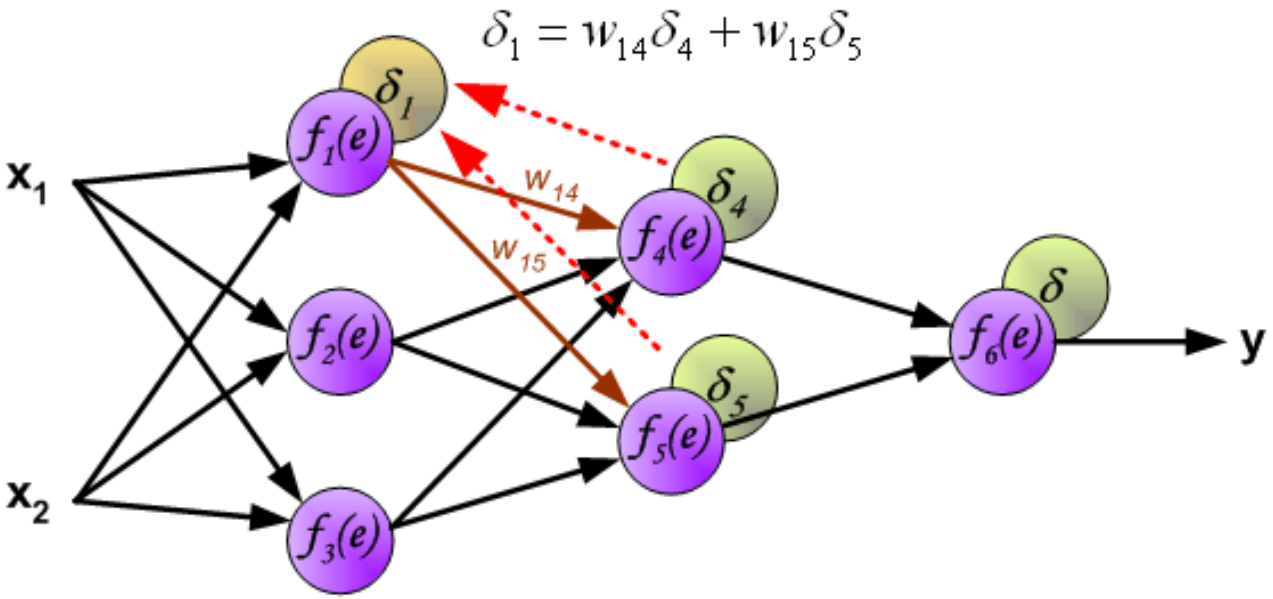


Algoritmo de aprendizaje: Backpropagation



Algoritmo de aprendizaje: Backpropagation

Los coeficientes de los pesos w_{mn} usados para propagar los errores hacia atrás son iguales a los usados durante el calculo del valor de salida. Solamente la dirección del dato de salida del flujo de datos cambia. Esta técnica se usa para todas las capas de salida.

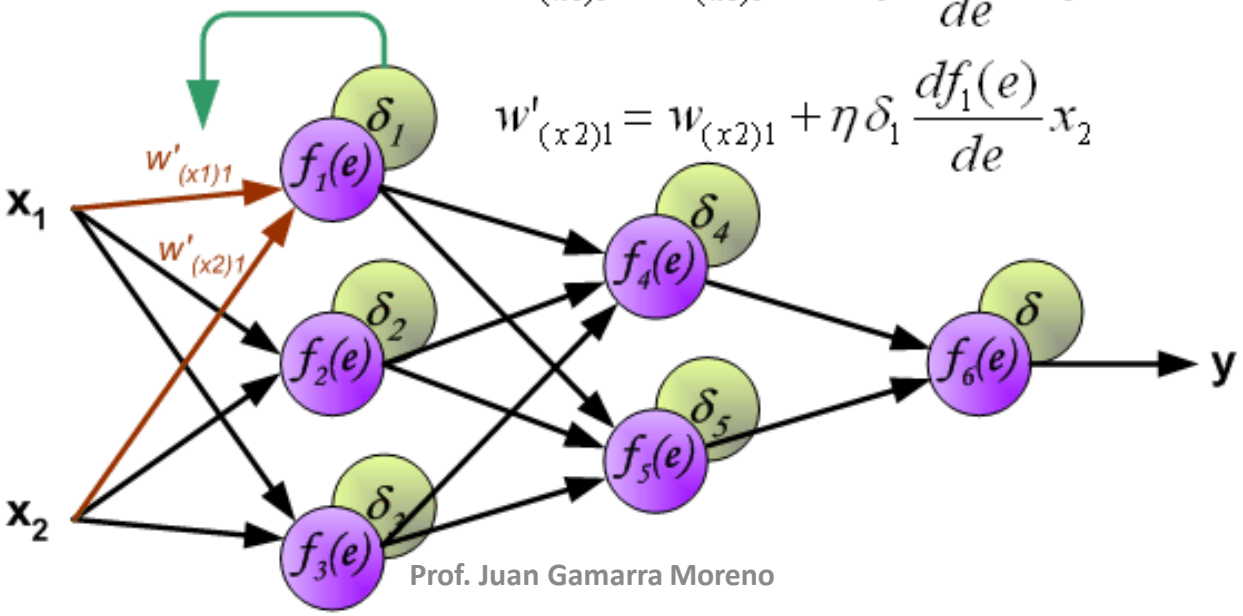


Algoritmo de aprendizaje: Backpropagation

Cuando el error para cada neurona se calcula, los coeficentes de los pesos de cada neurona pueden modificarse. En las formulas debajo *df(e)/de* representa la derivada de la función de activación (cuyos pesos son modificados)

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

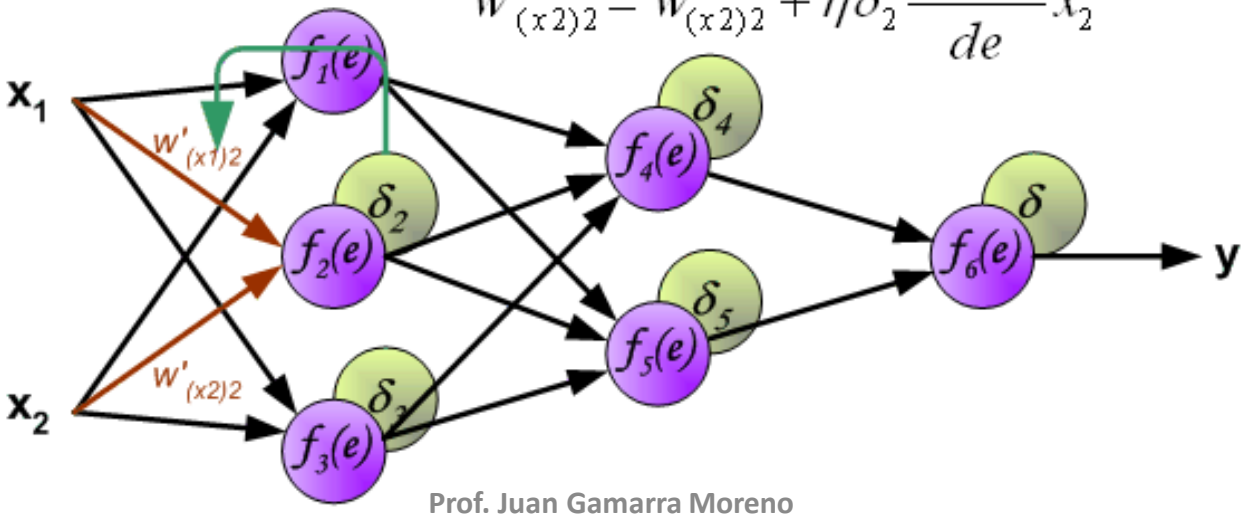
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Algoritmo de aprendizaje: Backpropagation

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

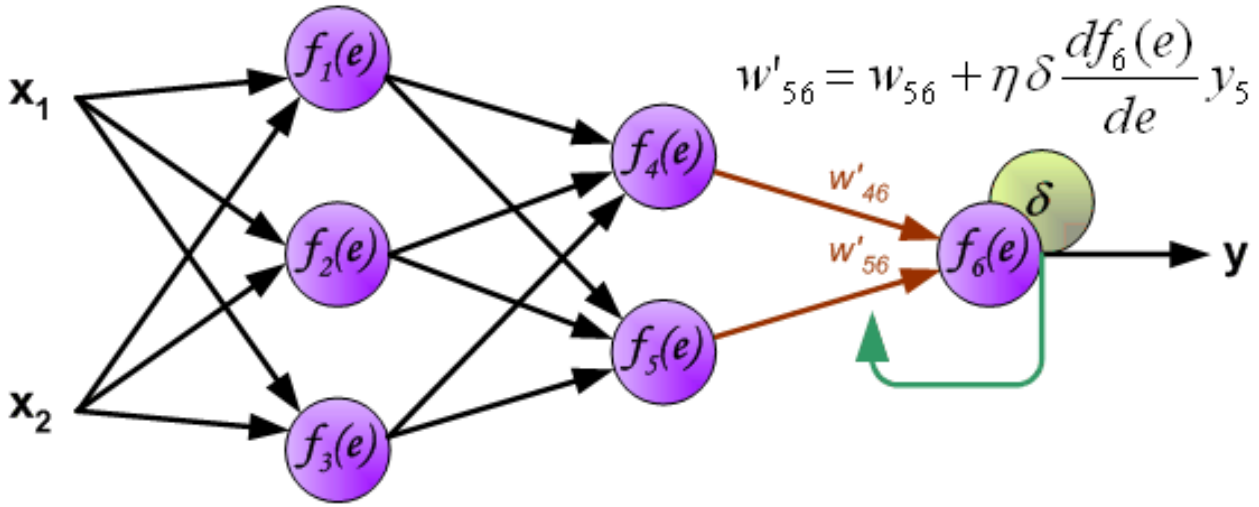
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Algoritmo de aprendizaje: Backpropagation

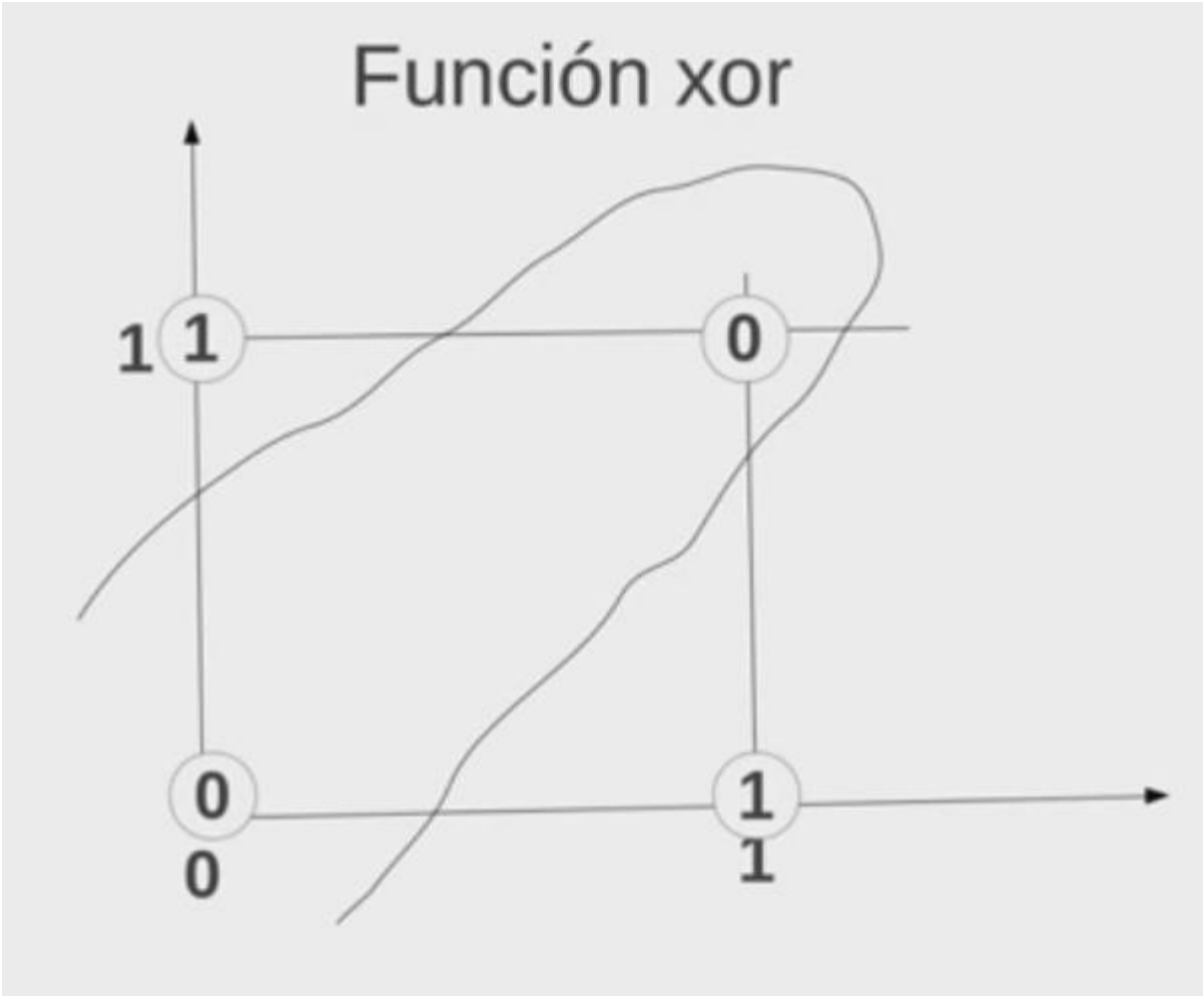
$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

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

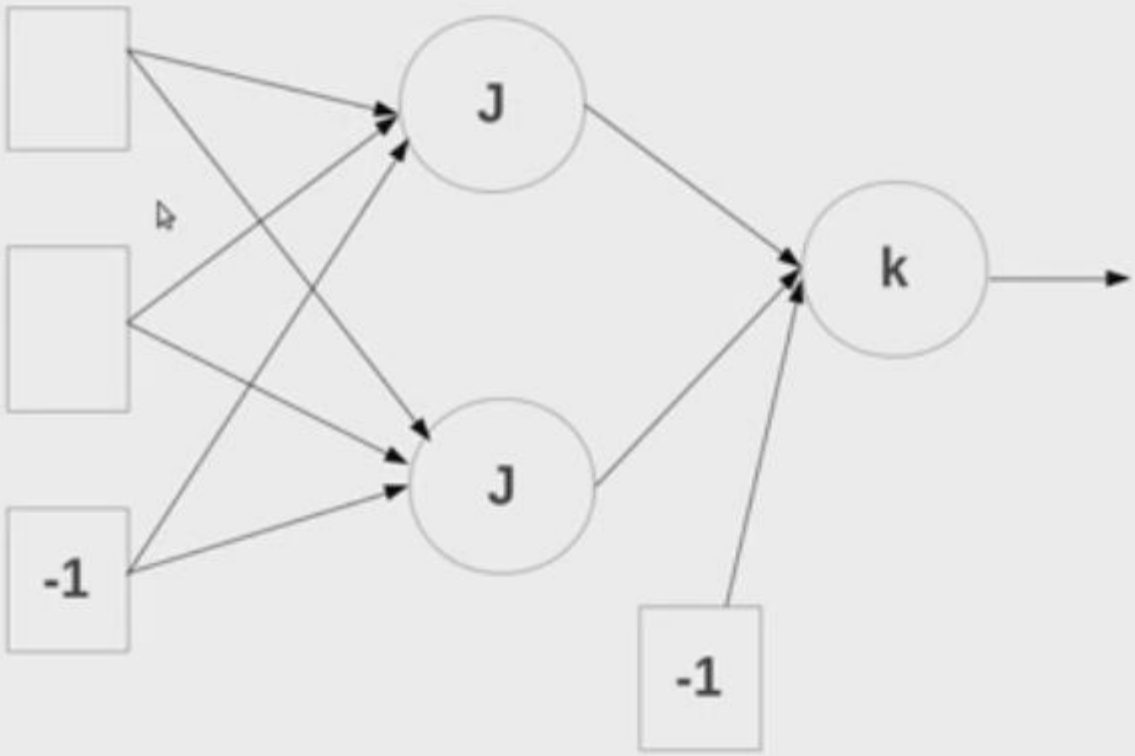


Función xor

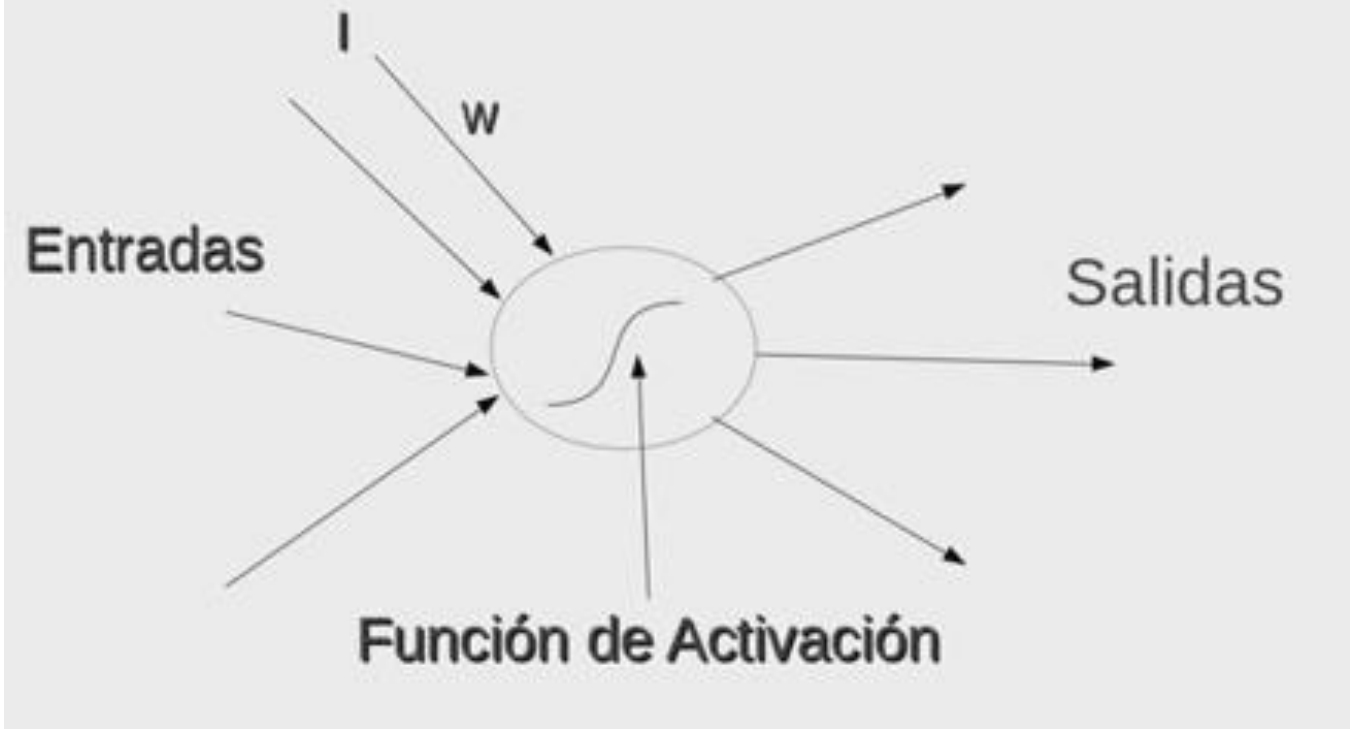
A	B	xor
1	1	0
1	0	1
0	1	1
0	0	0


Función xor
1
1
0
0
0
1
1

Entrenamiento de Red Neuronal



Entrenamiento de Neurona



```
****** epoca 0 ********Iteración 126567 ***********
Entrada (1.0,1.0) aprender 0.0
activacion 0.49477735847635496
Activación 0.0
****** epoca 1 ********Iteración 126568 ***********
Entrada (0.0,0.0) aprender 0.0
activacion 0.49982317464083287
Activación 0.0
****** epoca 2 ********Iteración 126569 ***********
Entrada (0.0,1.0) aprender 1.0
activacion 0.5004683591411158
****** epoca 3 ********Iteración 126570 ***********
Entrada (1.0,0.0) aprender 1.0
activacion 0.5000536891679682
Activación 1.0
*********actualiza Pesos_IJ*******
*******Pesos**************
-0.43580654549466213
-0.37874738838597927
0.11952746307612956
*******Pesos**************
1.3087287592719743
1.143629139353848
-0.039968235017526485
*********actualiza Pesos_JK*******
*******Pesos**************
0.6270592455927838
0.23630486342754753
0.41603479025475104
BUILD SUCCESSFUL (total time: 53 seconds)
```

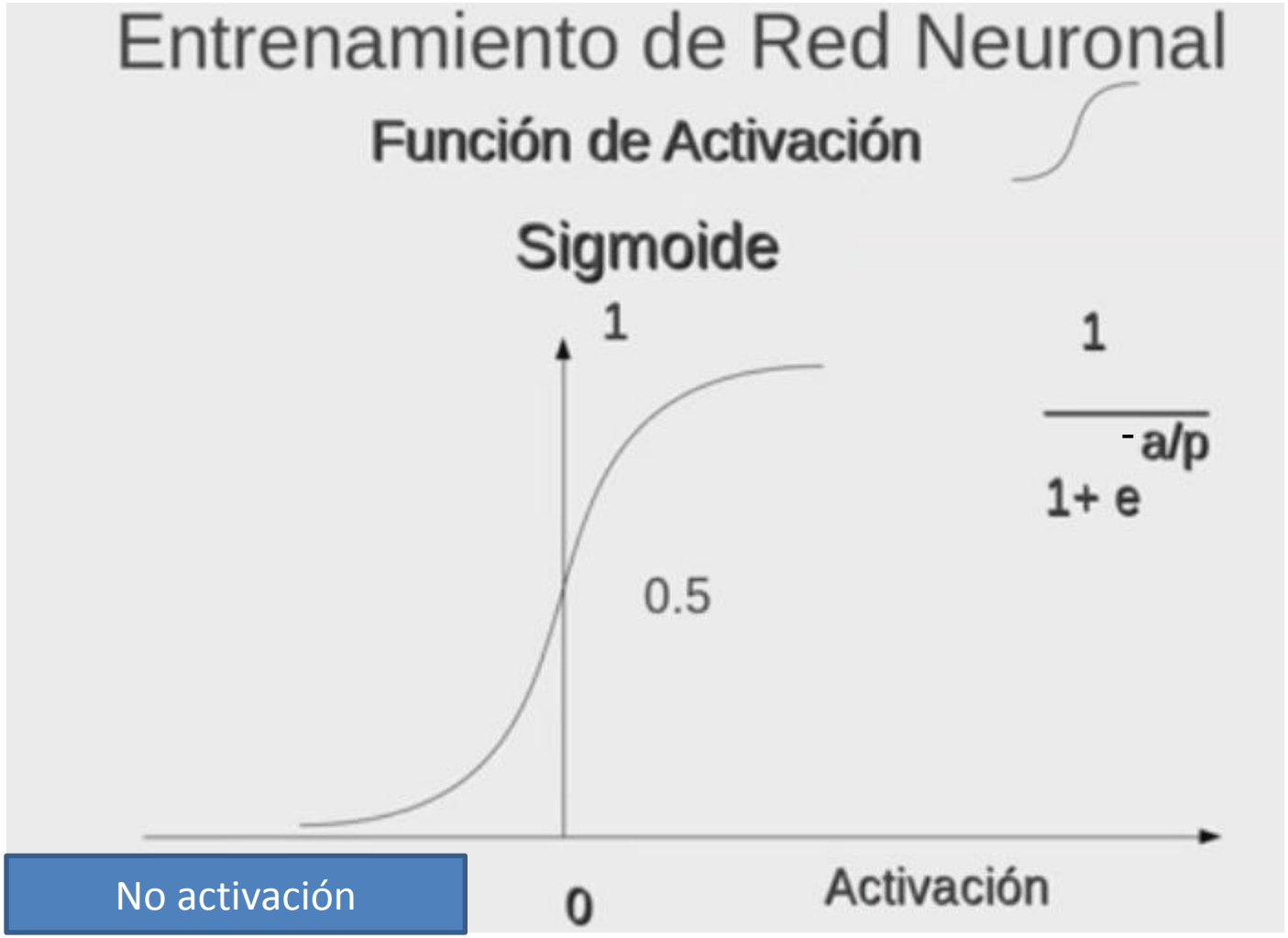
```
****** epoca 0 ********Iteración 57398 ***********
Entrada (1.0,1.0) aprender 0.0
activacion 0.4961999055605986
Activación 0.0
****** epoca 1 ********Iteración 57399 ***********
Entrada (0.0,0.0) aprender 0.0
activacion 0.4996866748718371
Activación 0.0
****** epoca 2 ********Iteración 57400 ***********
Entrada (0.0,1.0) aprender 1.0
activacion 0.49998982101300804
Activación 0.0
```

```
****** epoca 0 ********Iteración 57401 ***********
Entrada (1.0,1.0) aprender 0.0
activacion 0.5001297570533132
Activación 1.0
****** epoca 0 ********Iteración 57402 ***********
Entrada (1.0,1.0) aprender 0.0
activacion 0.4962101961603383
Activación 0.0
****** epoca 1 ********Iteración 57403 ***********
Entrada (0.0,0.0) aprender 0.0
activacion 0.49975633236012706
Activación 0.0
****** epoca 2 ********Iteración 57404 ***********
Entrada (0.0,1.0) aprender 1.0
activacion 0.500142999195403
Activación 1.0
****** epoca 3 ********Iteración 57405 ***********
Entrada (1.0,0.0) aprender 1.0
activacion 0.5000047693320956
Activación 1.0
```

```
*********actualiza Pesos_IJ*******
*******Pesos**************
-0.5672214005550219
0.8016244274849591
0.9935036526490338
*******Pesos**************
0.19733588486173123
-0.3945596772580417
0.7951695880484915
```

```
*********actualiza Pesos_JK*******
*******Pesos**************
0.3228551681857045
0.7356539449210586
0.317049109997351
BUILD SUCCESSFUL (total time: 22 seconds)
```

Entrenamiento de Red Neuronal



No activación

Entrenamiento de Red Neuronal

Función de Activación

Sigmoide

$$\sum Xi * Wi$$

$$\frac{1}{1 + e^{-a/p}}$$

a ➡

P forma de la Función

e=2.7183....

Entrenamiento de Neurona

Sigmoide

Source History

```
 7
 8  /*** @author victor**/
 9  public class Sigmoide {
10      double e;
11      double p;
12      public Sigmoide(){
13        e = Math.E;  p=1;
14      }
15      public double funcion(double a){
16         return(1/(1+ Math.pow(e, -a/p)));
17      }
18
19  }
```

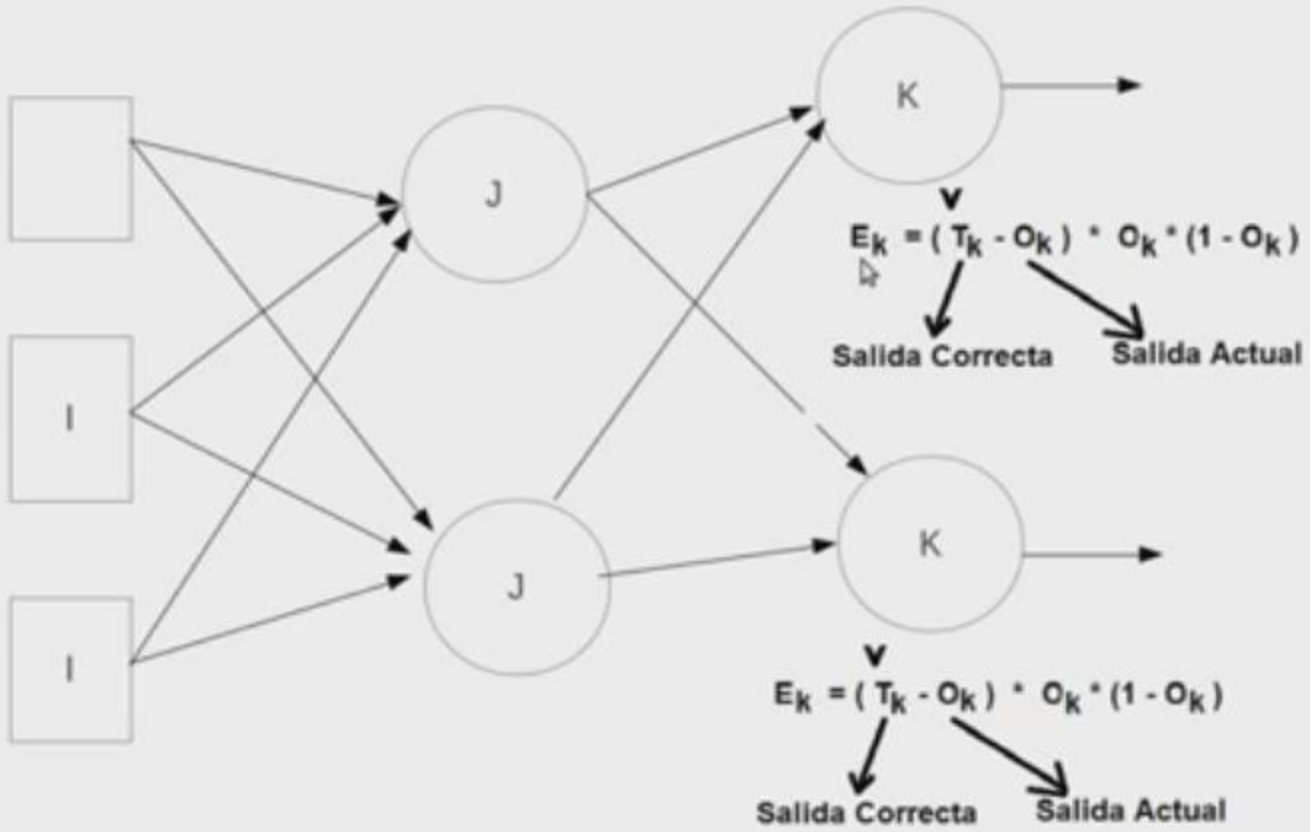
Entrenamiento de Red Neuronal

Función de Activación

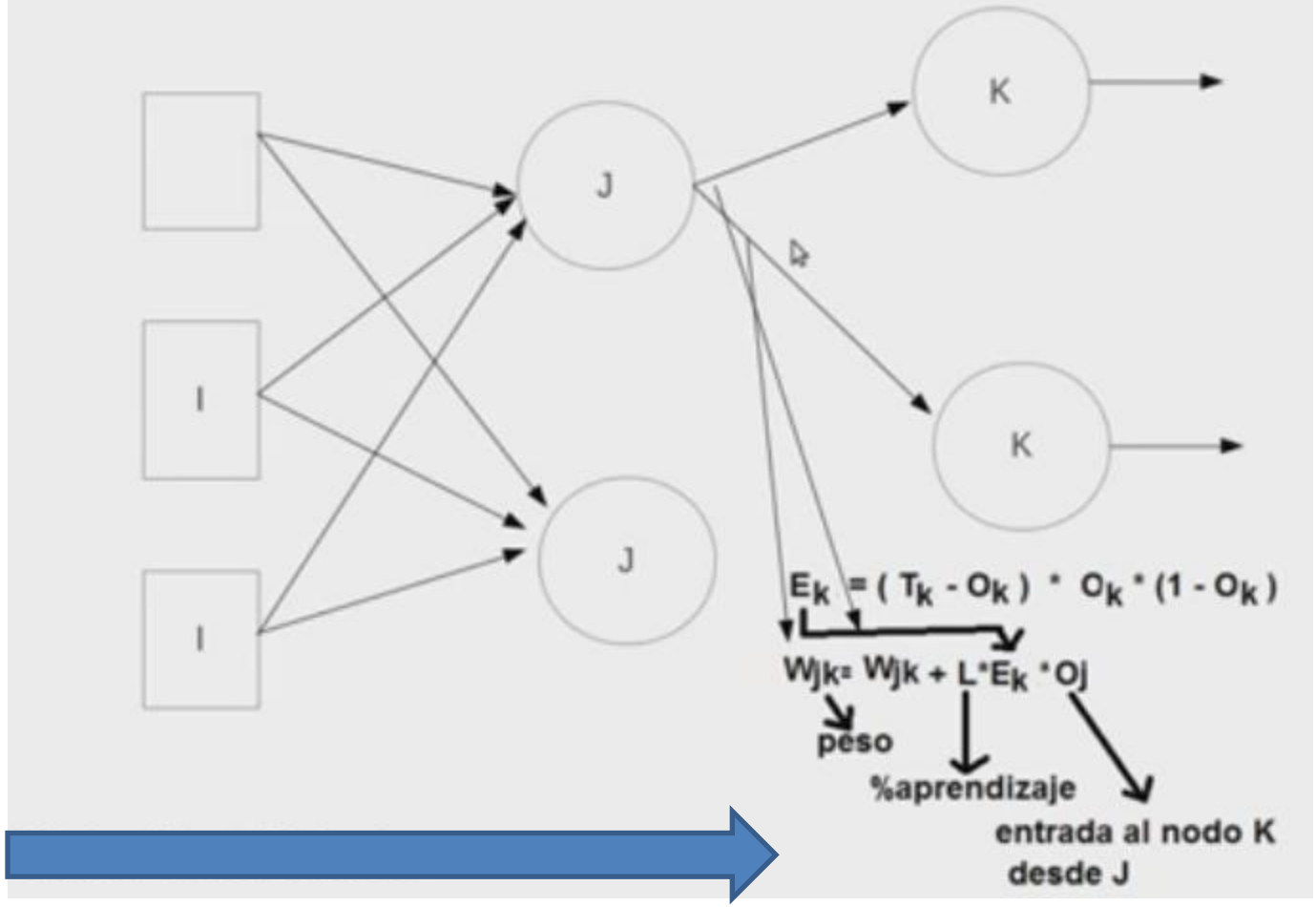
Neurona

```
public void activacion(){
    double suma=0;
   for(int i=0;i<entradas.length;i++){
     suma+=entradas[i]*pesos[i];
   }
   activacionu= sigmoide.funcion(suma);
   llenaSalidas();
}
```

BackPropagation

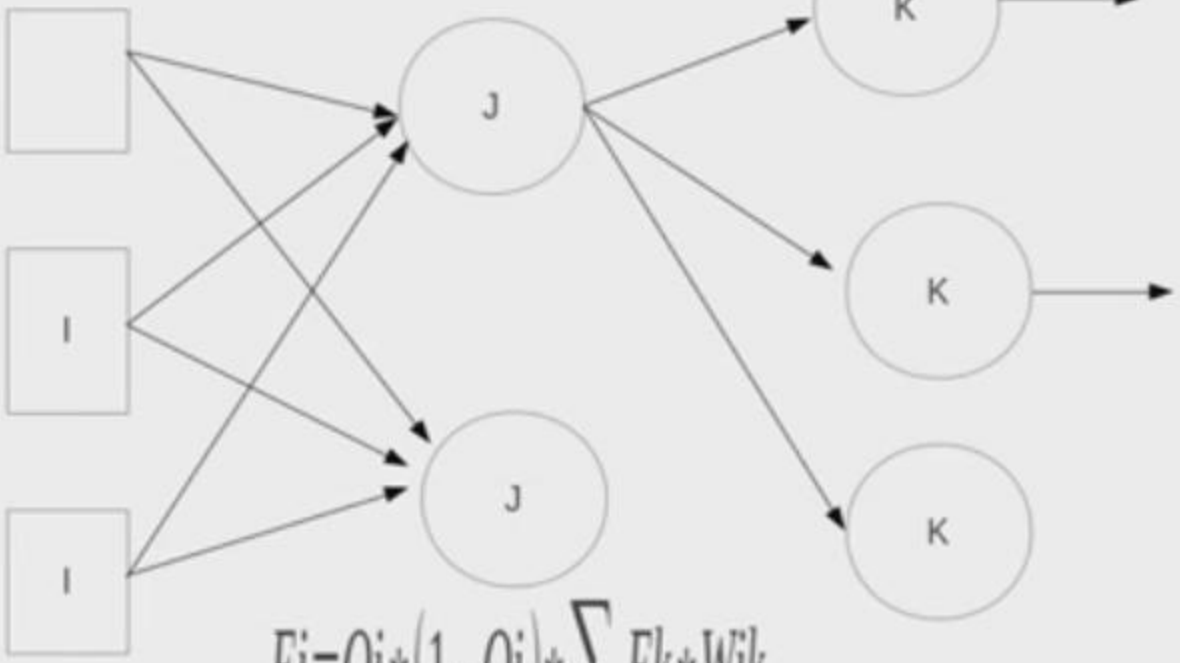


BackPropagation

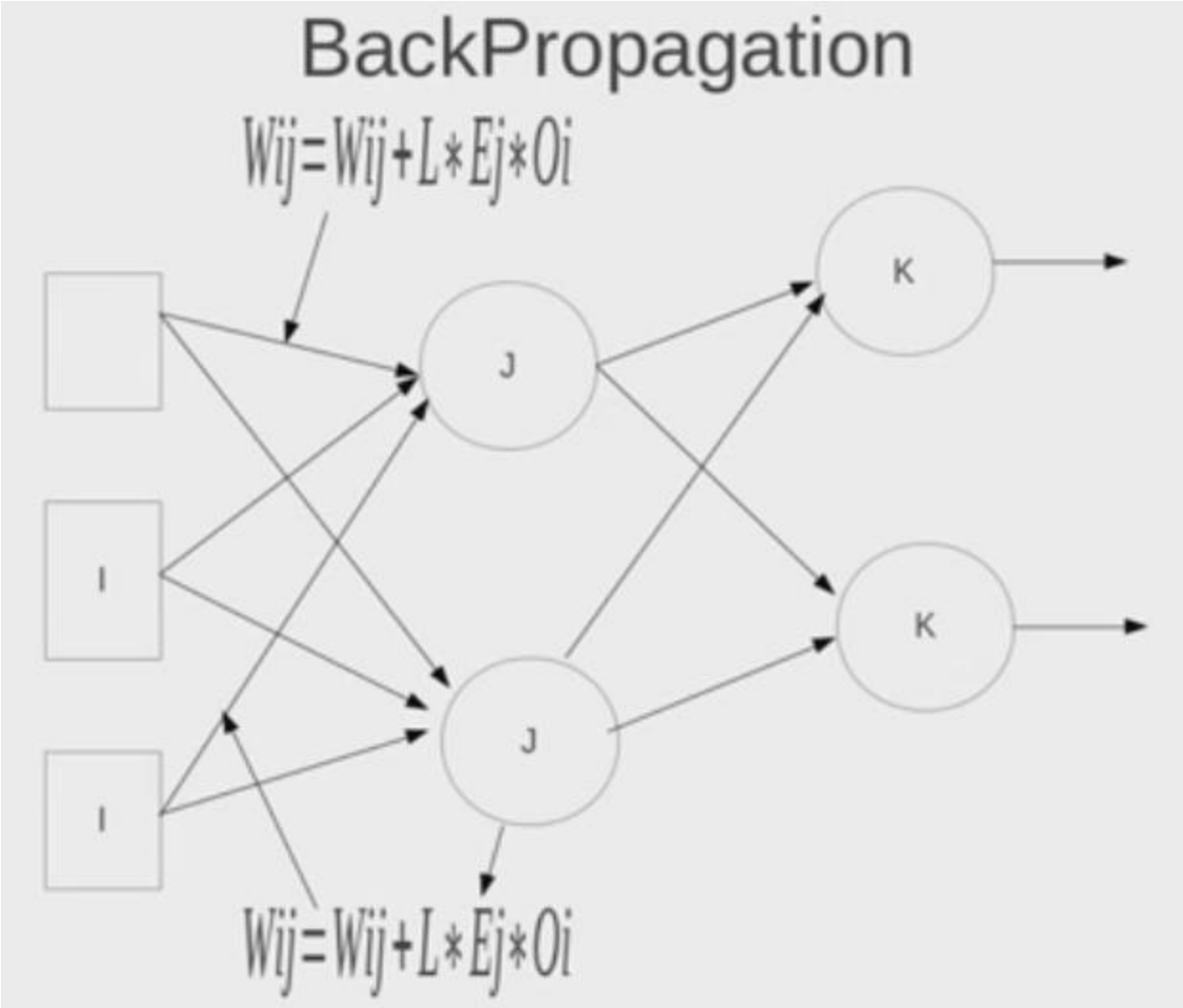


BackPropagation

$$Ej = Oj * (1 - Oj) * \sum Ek * Wjk$$



$$Ej = Oj * (1 - Oj) * \sum Ek * Wjk$$


BackPropagation
Wij=Wij+L*Ej*Oi
I
I
J
J
K
K
Wij=Wij+L*Ej*Oi

Entrenamiento de Red Neuronal