



# Inteligencia Artificial

Unidad 3

## Redes Neuronales

---

PERCEPTRON MULTICAPA

- Hugo David Calderón
- Heider Sanchez Enriquez

# Perceptrón Multicapa





# Perceptrón Multicapa

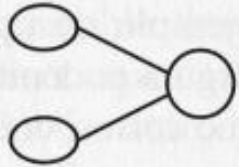
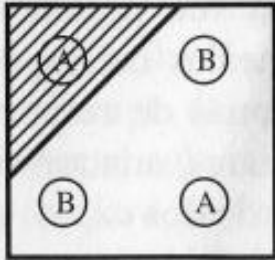
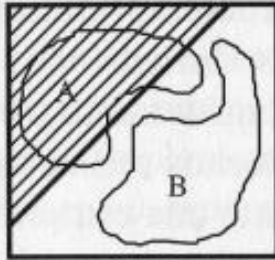
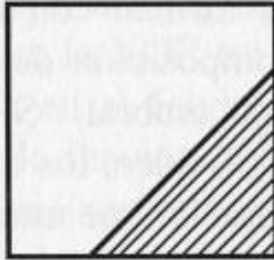
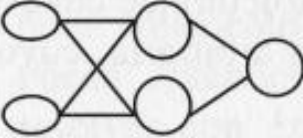
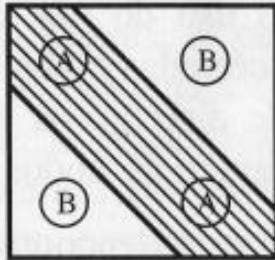
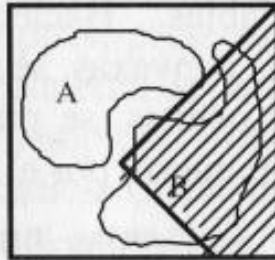
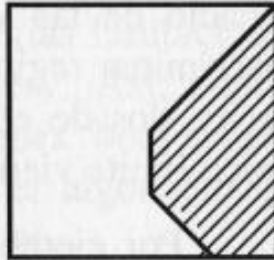
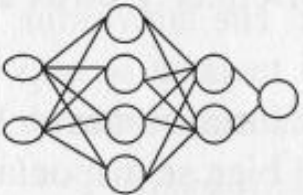
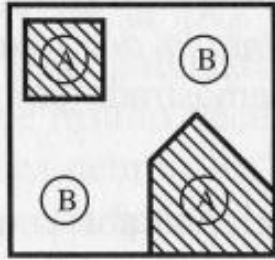
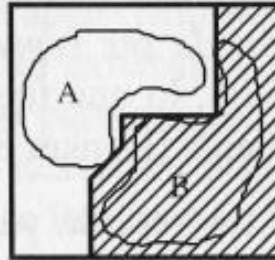
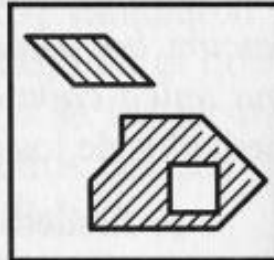
- ❖ En la sesión anterior hemos visto las limitaciones del perceptrón simple, ya que con él tan sólo podemos discriminar patrones que pueden ser separados por un hiperplano, una recta en el caso de dos neuronas de entrada.
- ❖ Una manera de solventar estas limitaciones del perceptrón simple es por medio de la inclusión de **capas ocultas**, obteniendo de esta forma una red neuronal que se denomina **perceptrón multicapa**.
- ❖ Cybenko Hornik en 1989 demostró que el perceptrón multicapa es un **aproximador universal**. Cualquier función continua sobre  $\mathbb{R}^n$ , puede ser aproximada, con al menos una capa oculta.



# Perceptrón Multicapa

La imagen muestra las regiones de decisión que se obtienen para distintas arquitecturas de redes neuronales considerando dos neuronas en la capa inicial.

Se puede observar que perceptrón multicapa con dos capas de neuronas ocultas es capaz de discriminar regiones de forma arbitraria.

Arquitectura	Región de decisión	Ejemplo 1: XOR	Ejemplo 2: clasificación	Regiones más generales
Sin capa oculta 	Hiperplano (dos regiones)			
Una capa oculta 	Regiones polinomiales convexas			
Dos capas ocultas 	Regiones arbitrarias			



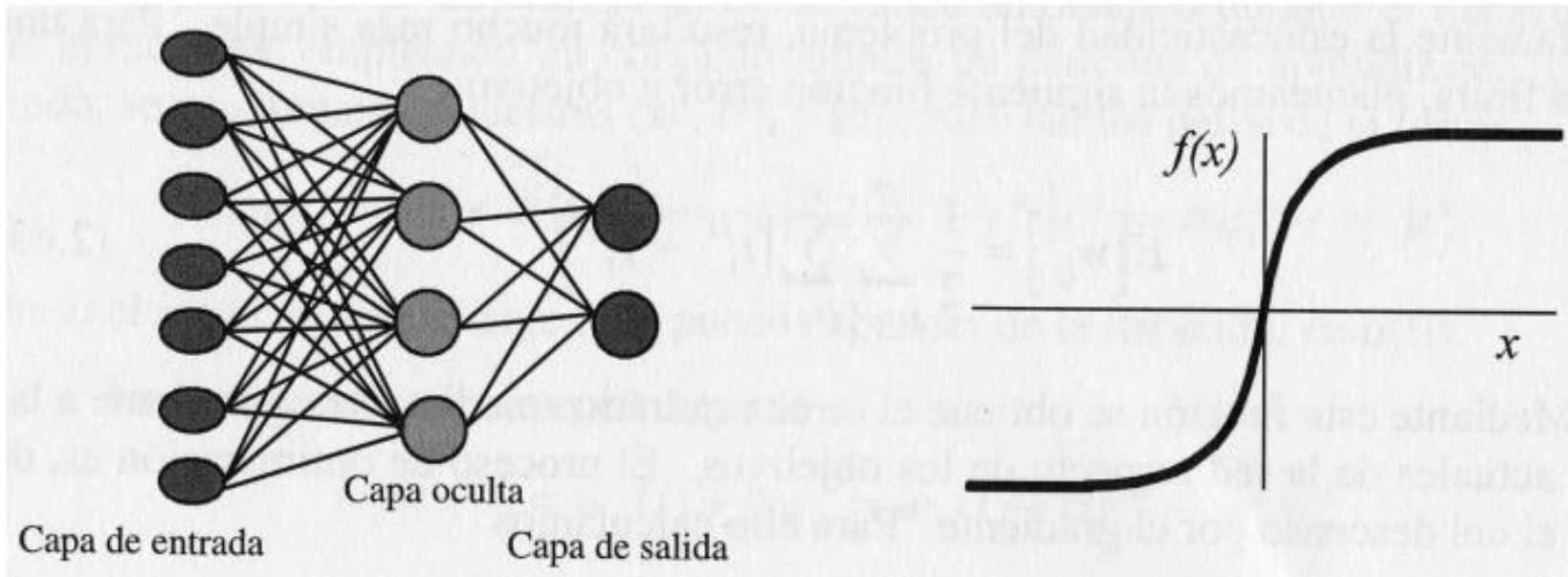
# Perceptrón Multicapa

- ❖ El perceptrón multicapa o MLP (Multi-Layer Perceptron) se suele entrenar por medio de un algoritmo de retropropagación de errores o BP (**Back Propagation**) de ahí que dicha arquitectura se conozca también bajo el nombre de red de retropropagación.



# Perceptrón Multicapa: Arquitectura

- ❖ La estructura del MLP con una única capa oculta se muestra en la siguiente imagen:



**Función de  
activación sigmoideal**





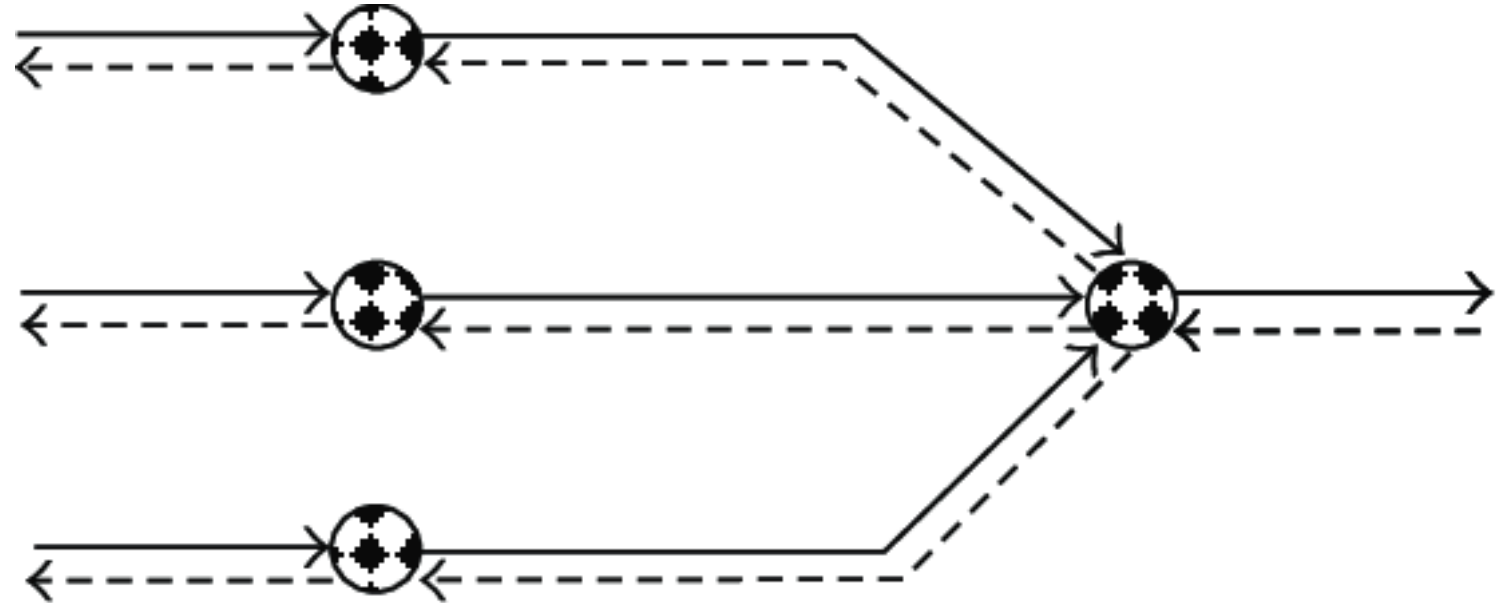
# Perceptrón Multicapa: Arquitectura

- ❖ Todas las neuronas transmiten información hacia delante: se denominan redes feedforward.
- ❖ Cada neurona posee un umbral independiente. Se considera como una entrada más cuya entrada es 1.
- ❖ Generalmente se utilizan redes completamente conectadas.

# Perceptrón Multicapa: Arquitectura



Ilustración de las dos direcciones básicas de flujos de señal en un MLP.



—————> Function signals

<----- Error signals

**Feed forward**

**Back propagation**





# Perceptrón de tres capas

Cálculo de las salidas en cada capa

- Capa I

$$v_j^I = \langle \mathbf{w}^I, \mathbf{x} \rangle = \sum_{i=0}^N w_{ji}^I x_i \quad (\text{completo } \mathbf{v}^I = \mathbf{W}\mathbf{x})$$
$$y_j^I = \phi(v_j^I) = \frac{2}{1 + e^{-bv_j^I}} - 1 \quad (\text{simétrica } \pm 1)$$



# Perceptrón de tres capas

Cálculo de las salidas en cada capa

- Capa II

$$v_j^{II} = \langle \mathbf{w}^{II}, \mathbf{y}^I \rangle \rightarrow y_j^{II} = \phi(v_j^{II})$$

- Capa III

$$v_j^{III} = \langle \mathbf{w}^{III}, \mathbf{y}^{II} \rangle \rightarrow y_j^{III} = \phi(v_j^{III}) = y_j$$

# Perceptrón de tres capas



**Cálculo del error:** Suma del error cuadrático instantáneo

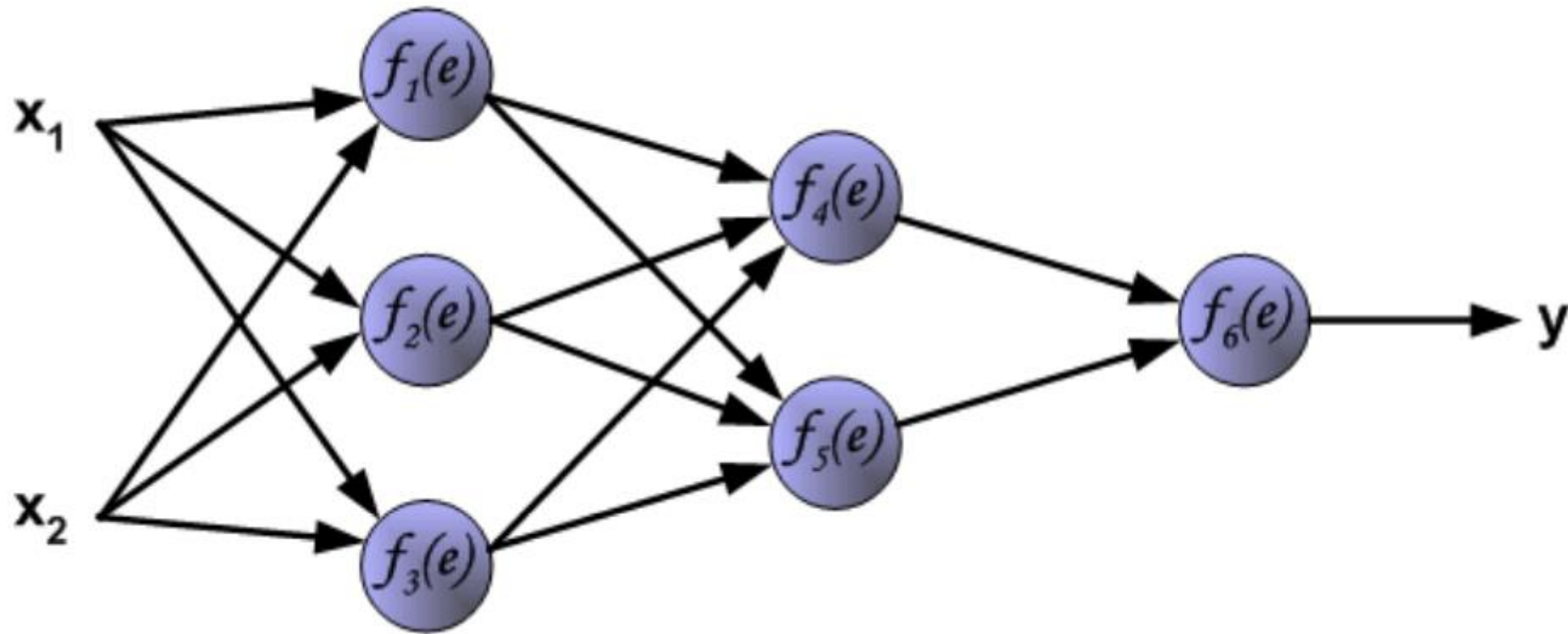
$$\xi(n) = \frac{1}{2} \sum_{j=1}^M e_j^2(n)$$



# Algoritmo Backpropagation

1. Inicialización aleatoria
2. Propagación hacia adelante de la entrada
3. Propagación hacia atrás del error  $\delta$
4. Adaptación de los pesos
5. Repetir desde el 2 hasta convergencia o finalización

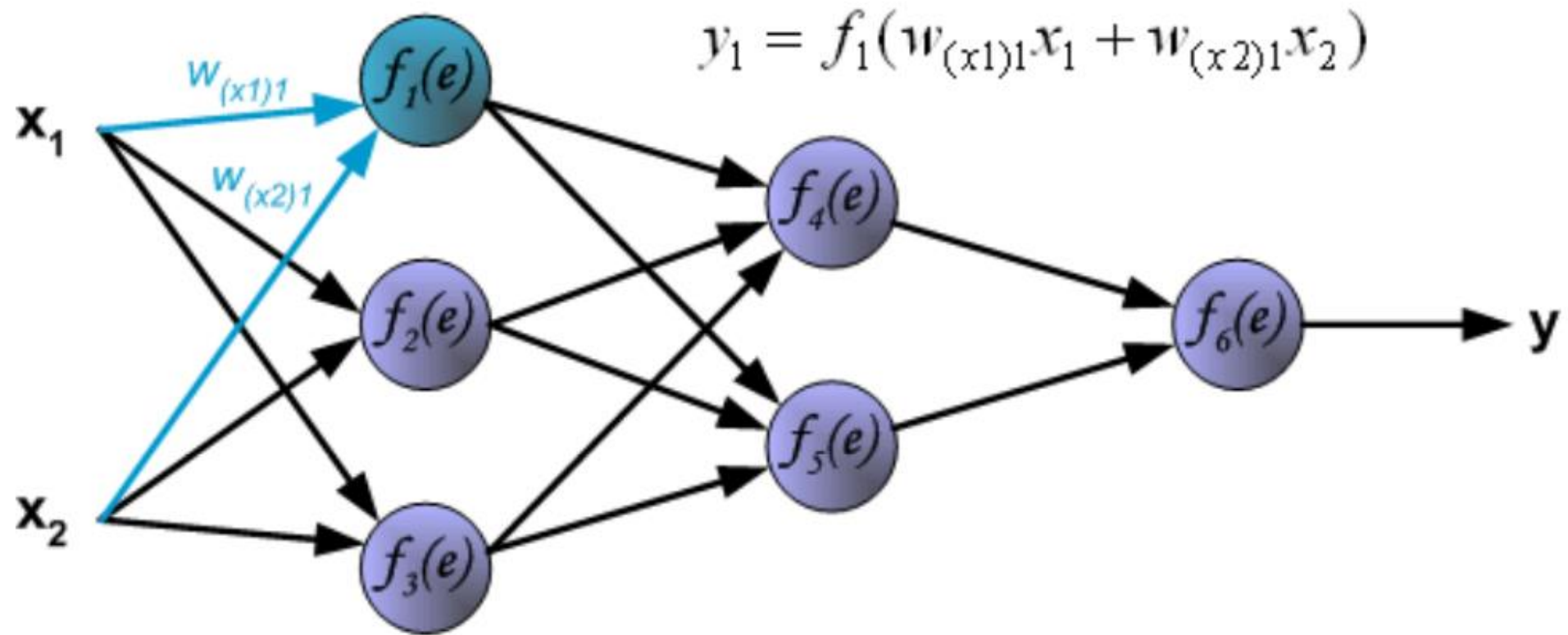
# Algoritmo Backpropagation: ejemplo gráfico



Ejemplo de un MLP de 3 capas



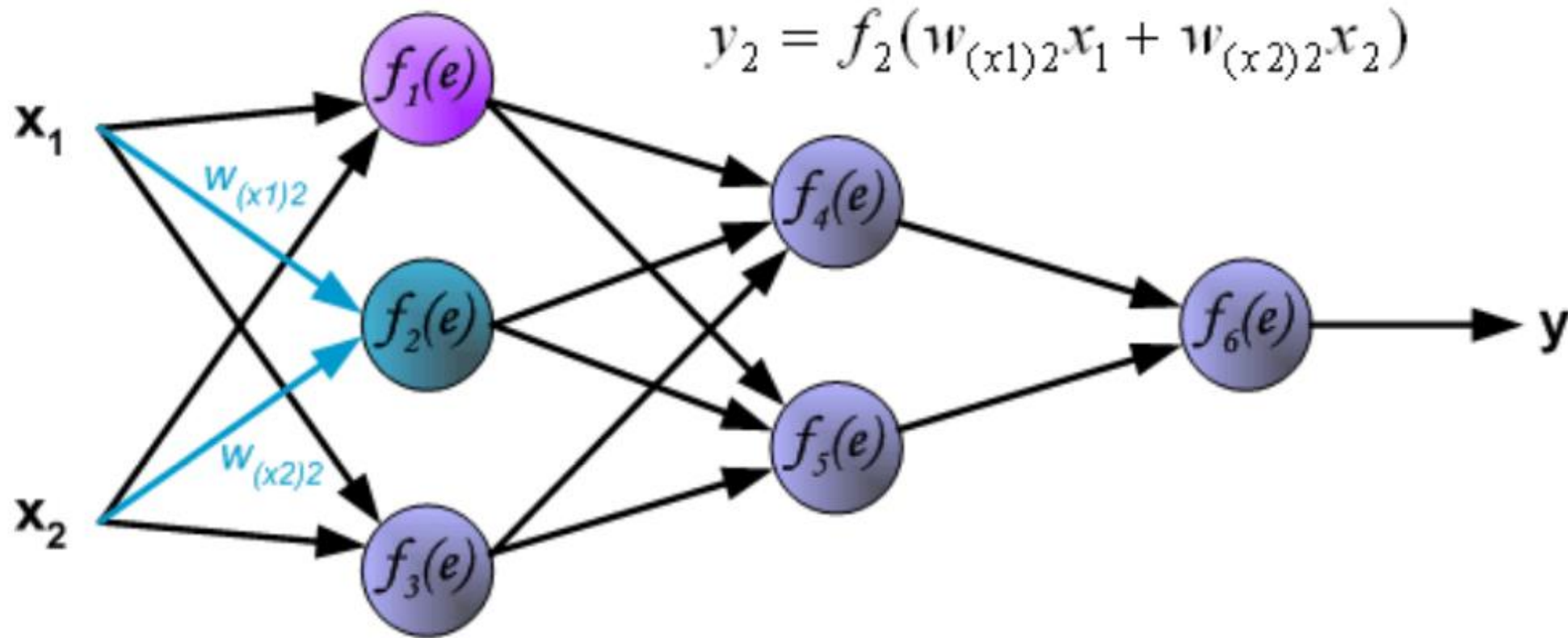
# Backpropagation: Cálculo de las salidas en cada capa



Cálculo salida capa I, neurona 1.



# Backpropagation: Cálculo de las salidas en cada capa

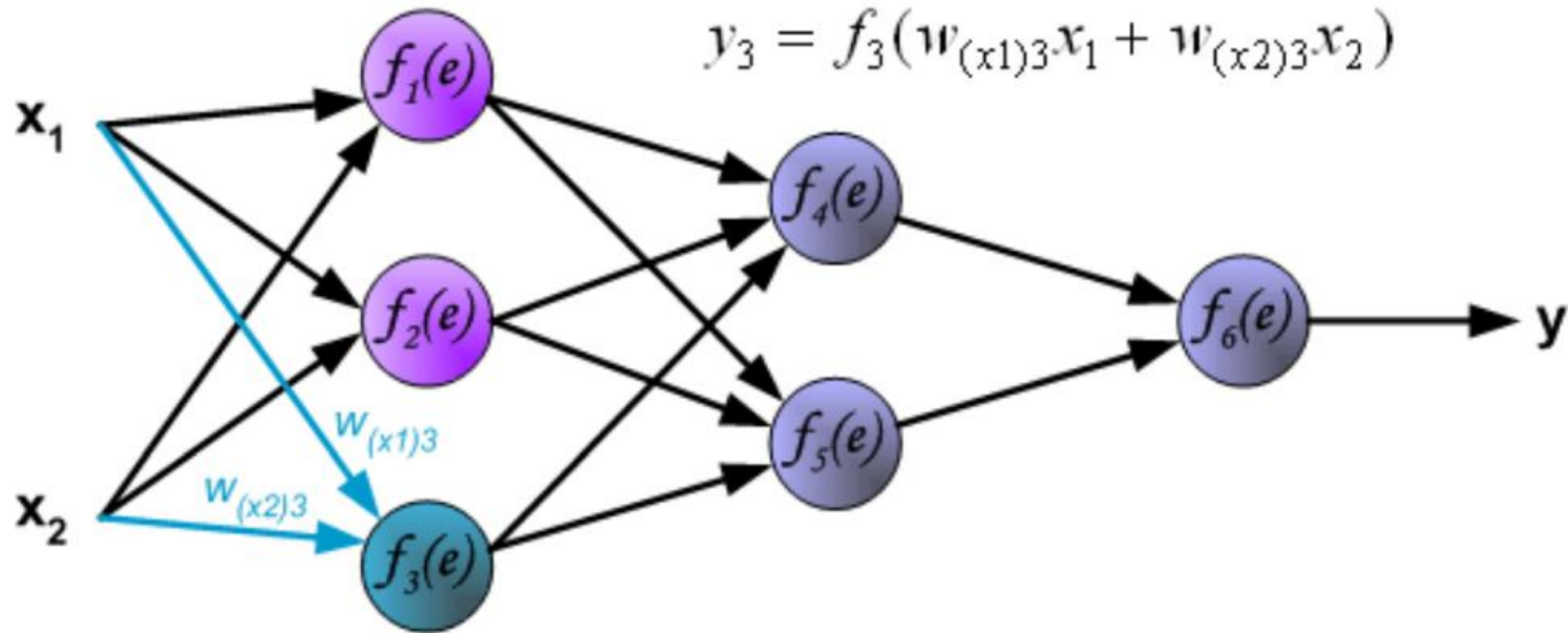


Cálculo salida capa I, neurona 2.





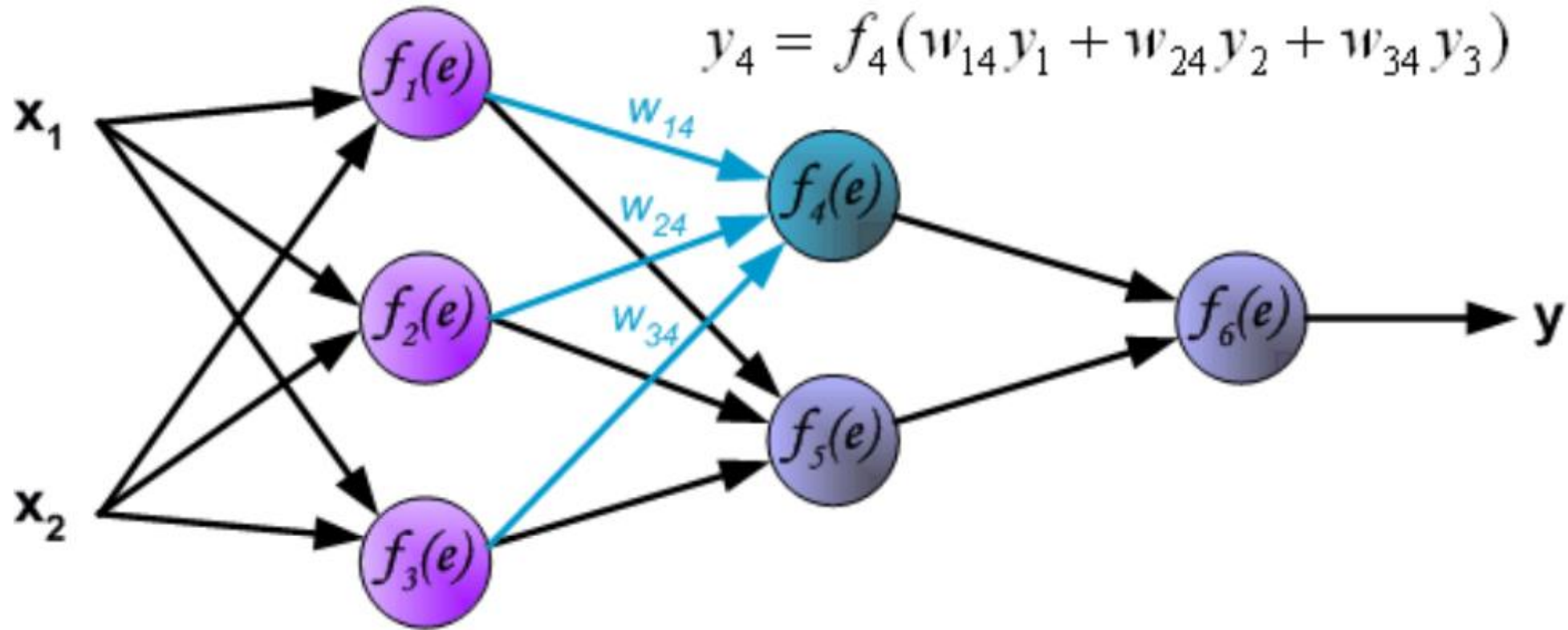
# Backpropagation: Cálculo de las salidas en cada capa



Cálculo salida capa I, neurona 3.



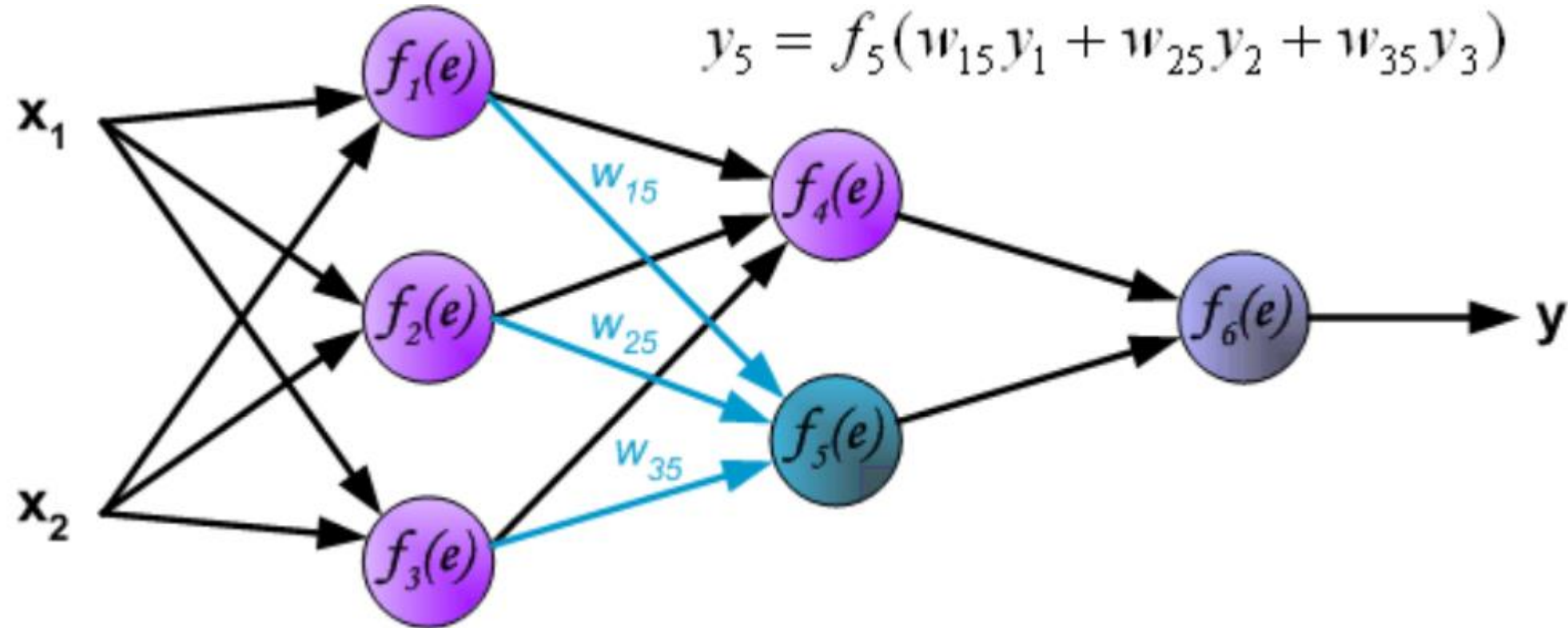
# Backpropagation: Cálculo de las salidas en cada capa



Cálculo salida capa II, neurona 1.



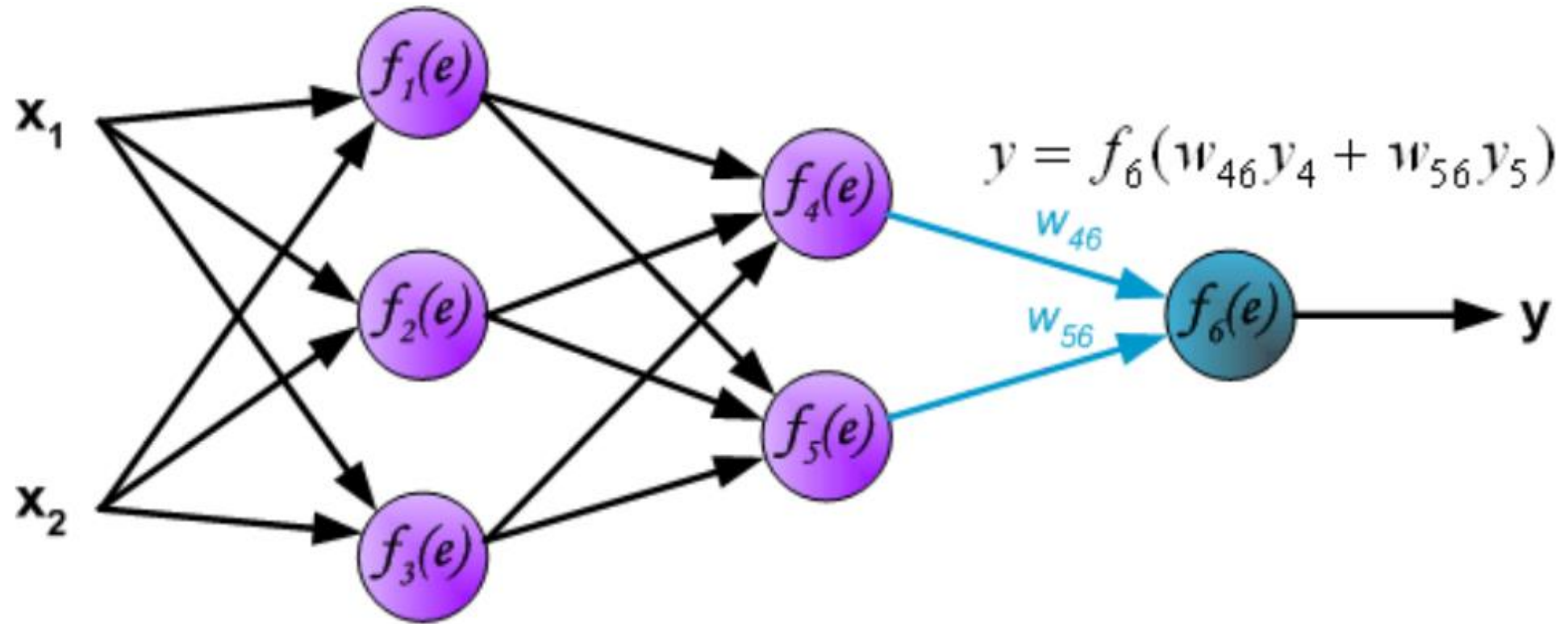
# Backpropagation: Cálculo de las salidas en cada capa



Cálculo salida capa II, neurona 2.



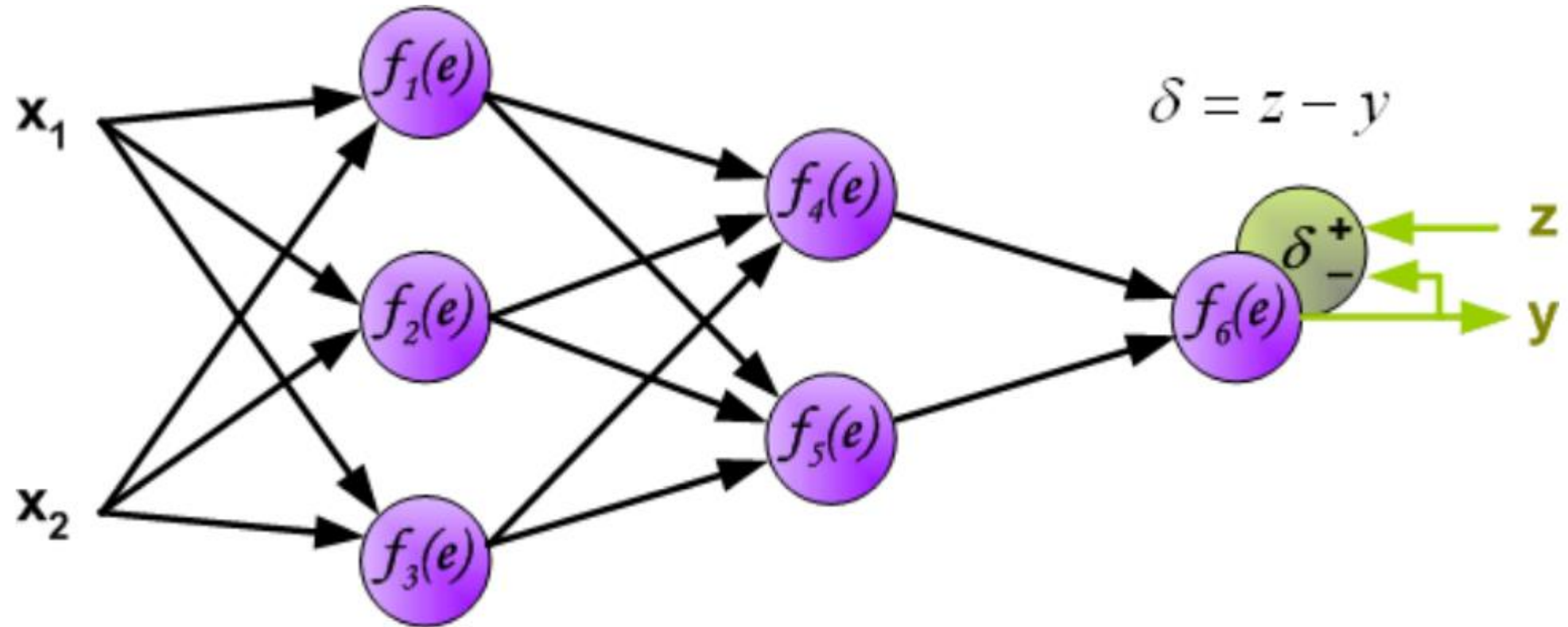
# Backpropagation: Cálculo de las salidas en cada capa



Cálculo salida capa III, neurona 1.



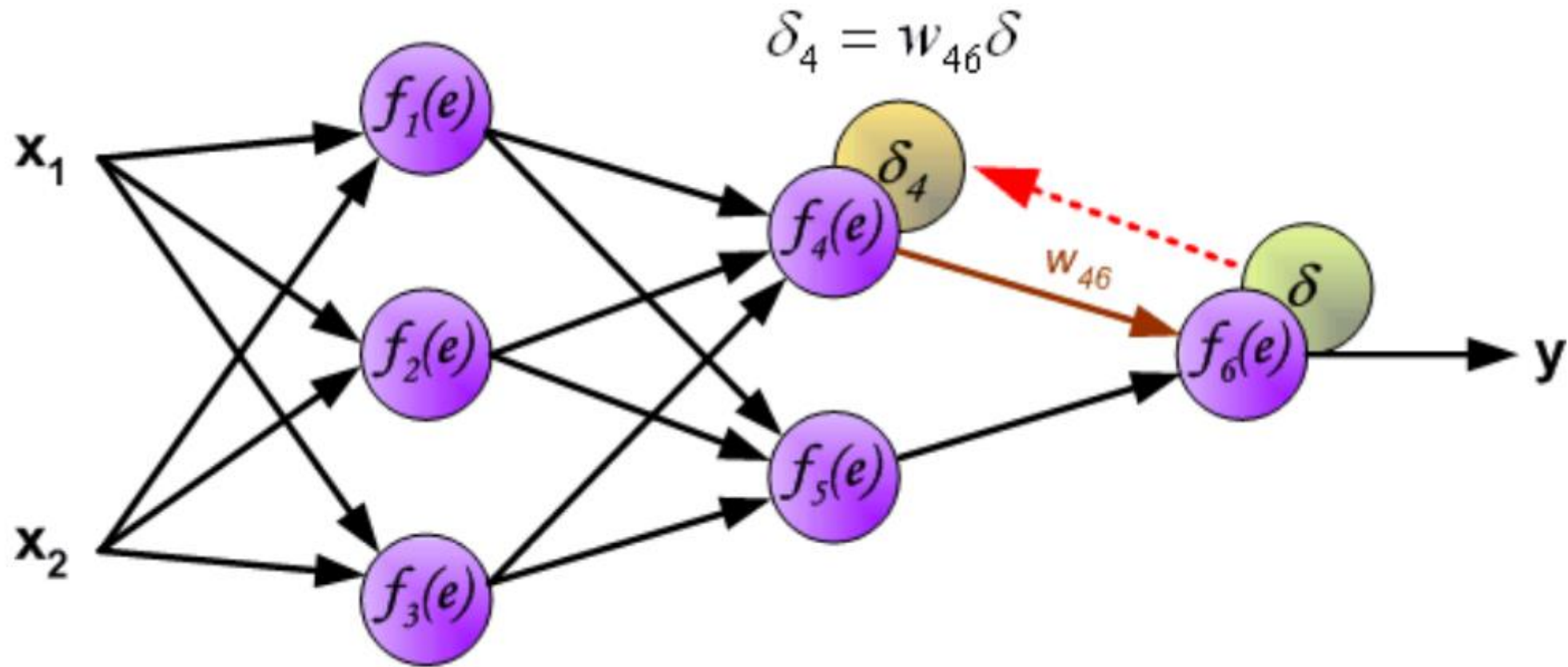
# Backpropagation: Retropropagación en la capa III



Cálculo del error en capa III, neurona 1.



# Backpropagation: Retropropagación en la capa III

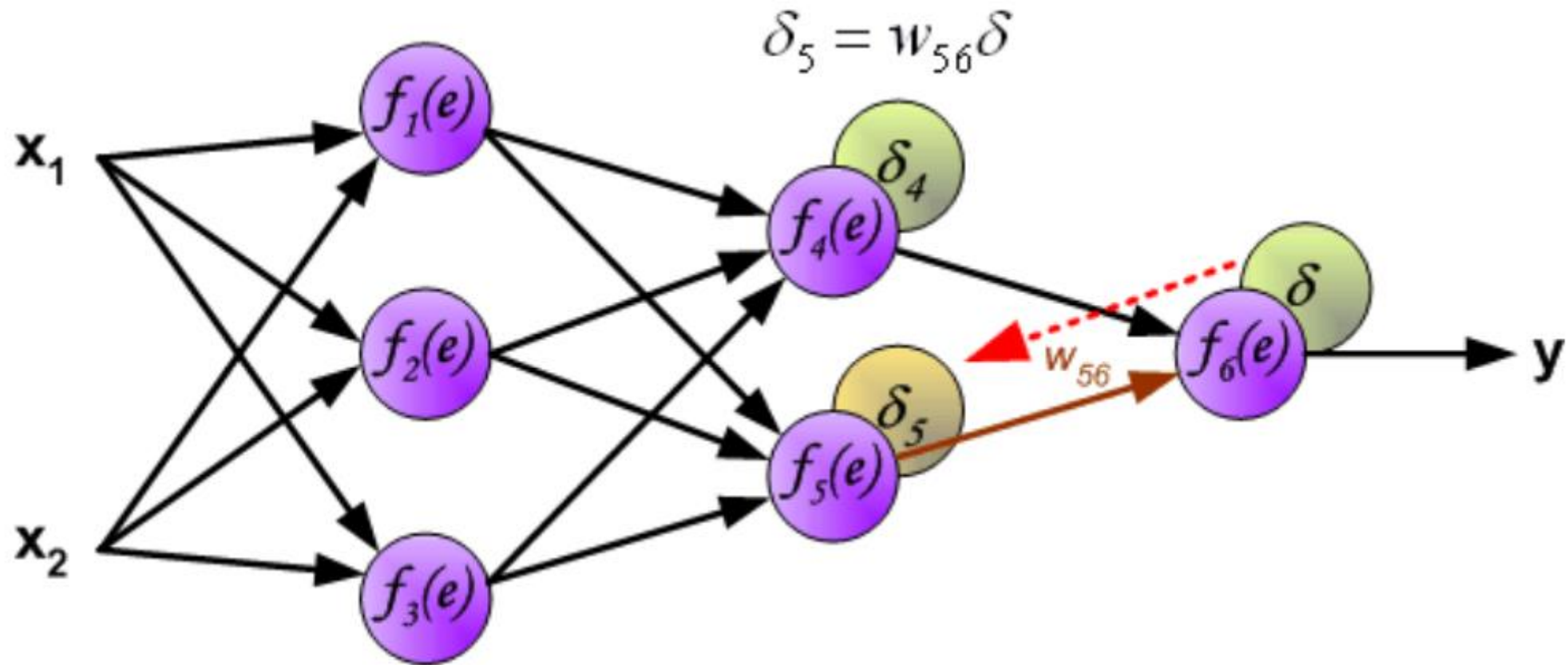


Propagación del error a la capa II, neurona 1.





# Backpropagation: Retropropagación en la capa III

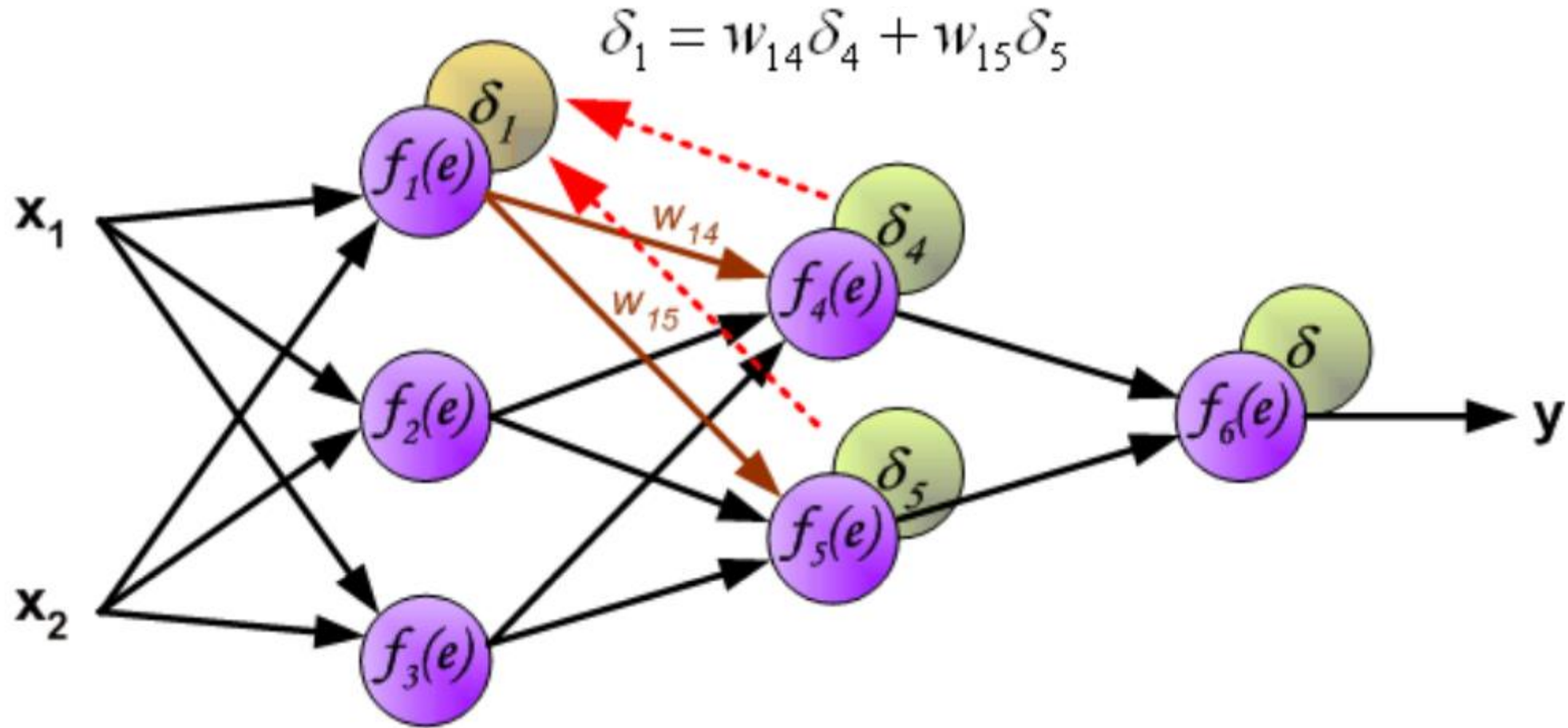


Propagación del error a la capa II, neurona 2





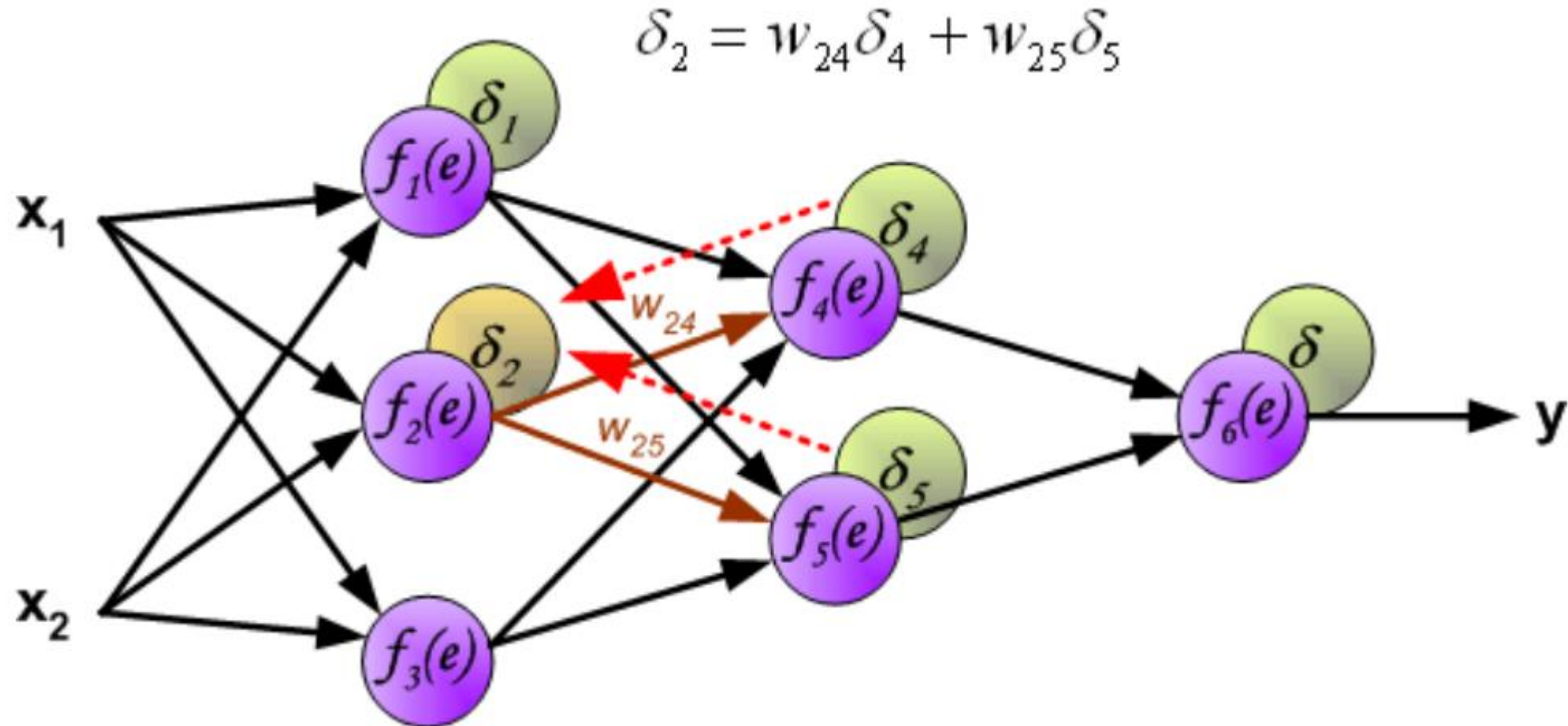
# Backpropagation: Retropropagación en la capa III



Propagación del error a la capa I, neurona 1



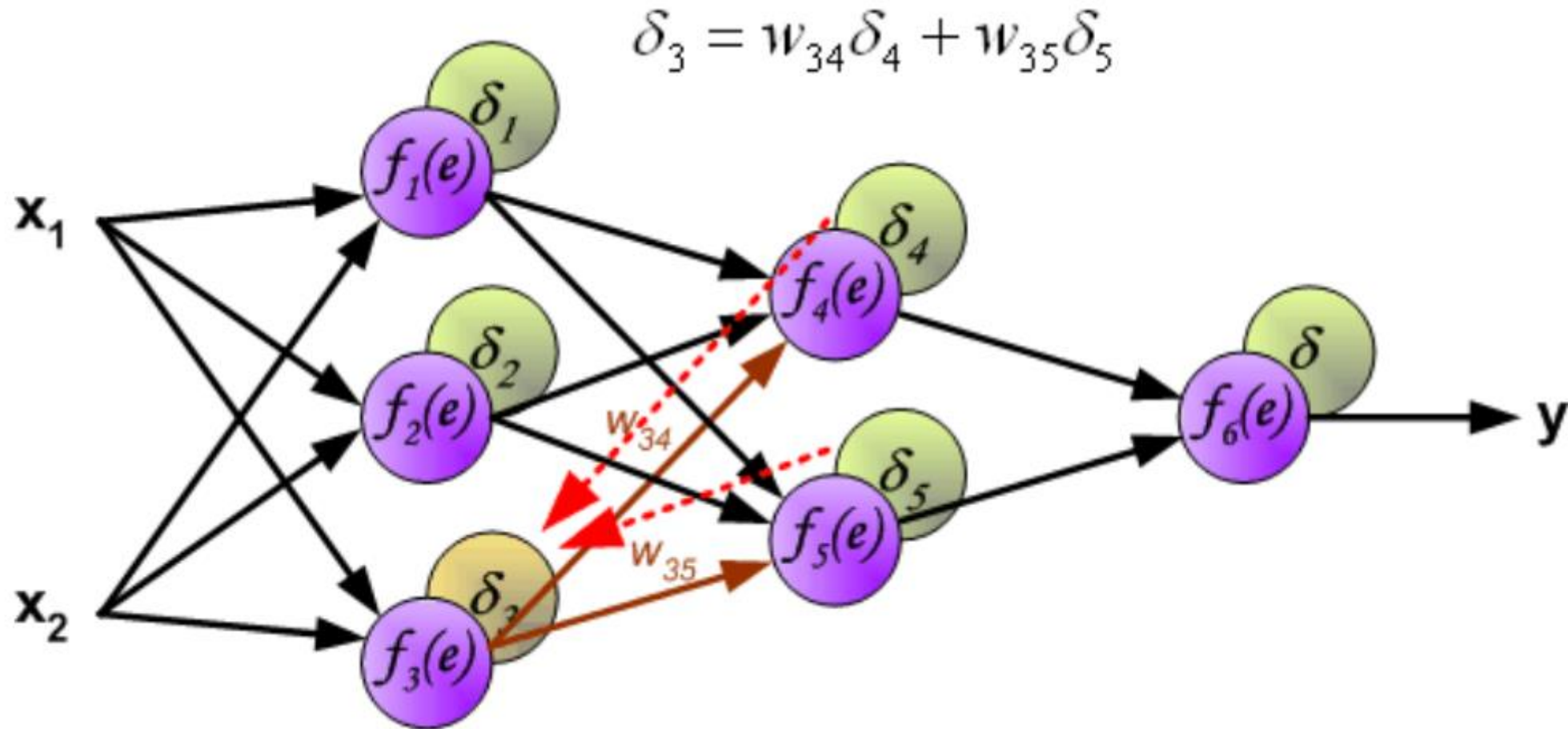
# Backpropagation: Retropropagación en la capa III



Propagacion del error a la capa I, neurona 2



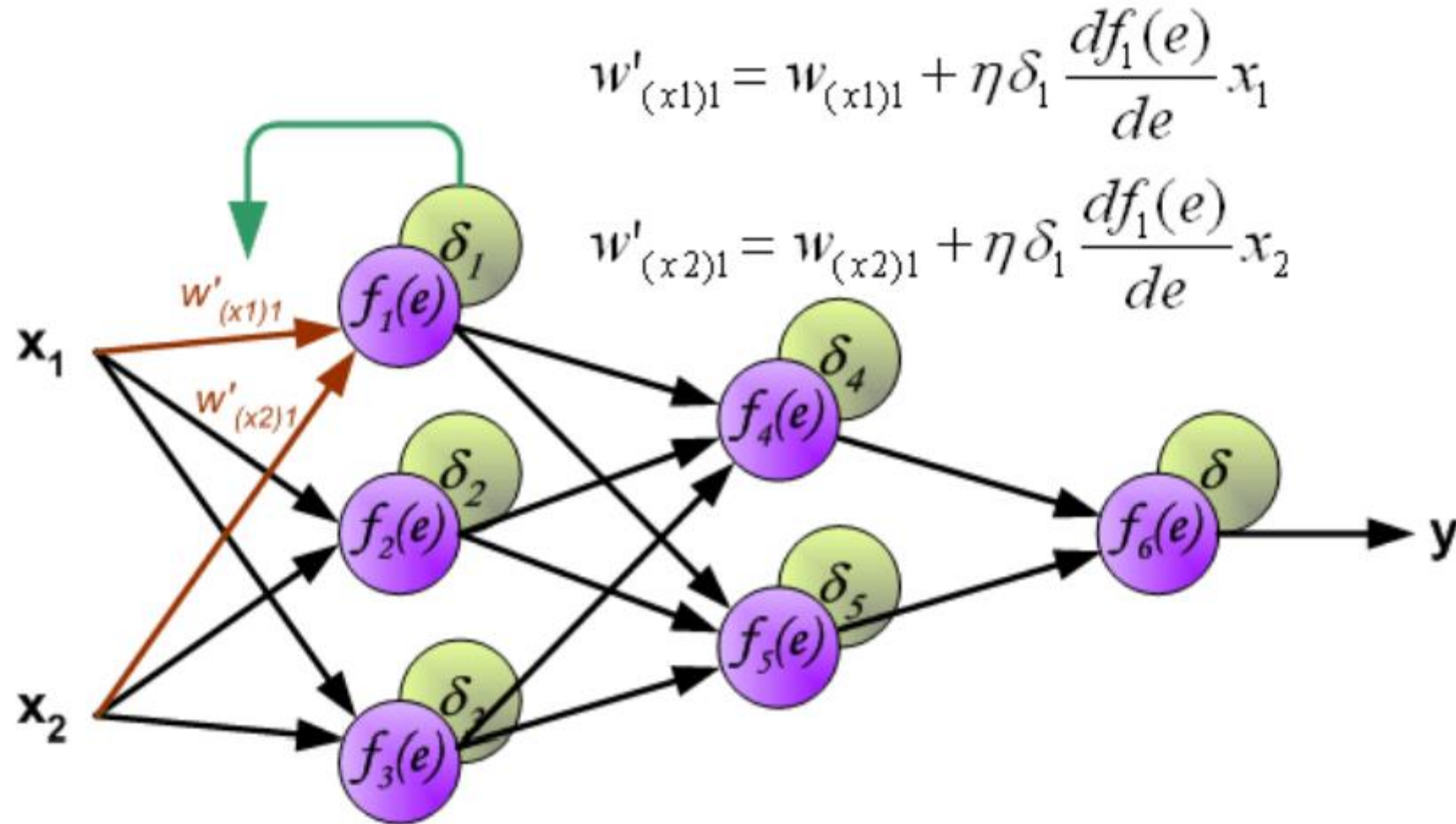
# Backpropagation: Retropropagación en la capa III



Propagación del error a la capa I, neurona 3



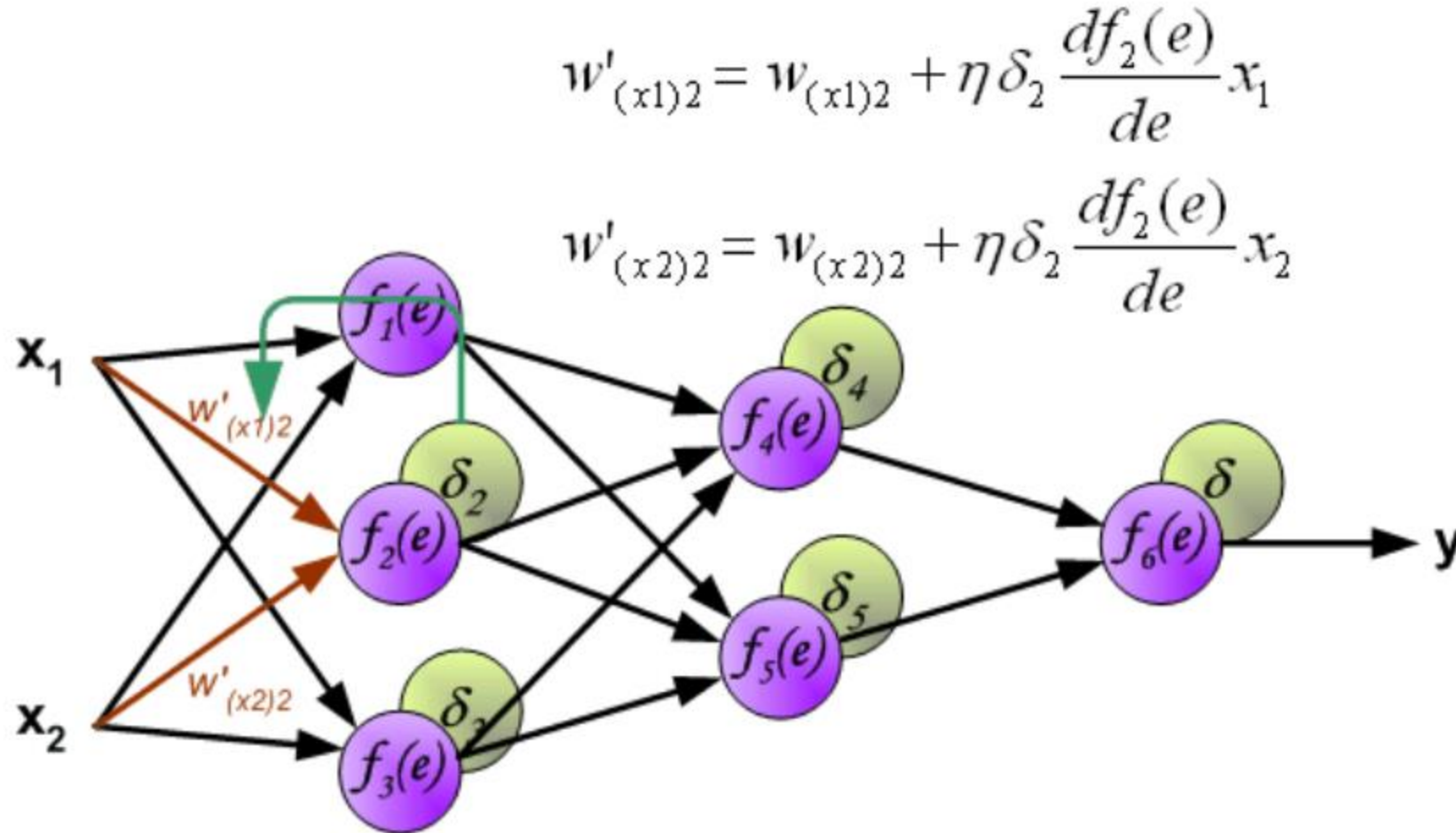
# Backpropagation: Actualizando los pesos de la red



Actualización de pesos capa I, neurona 1



# Backpropagation: Actualizando los pesos de la red

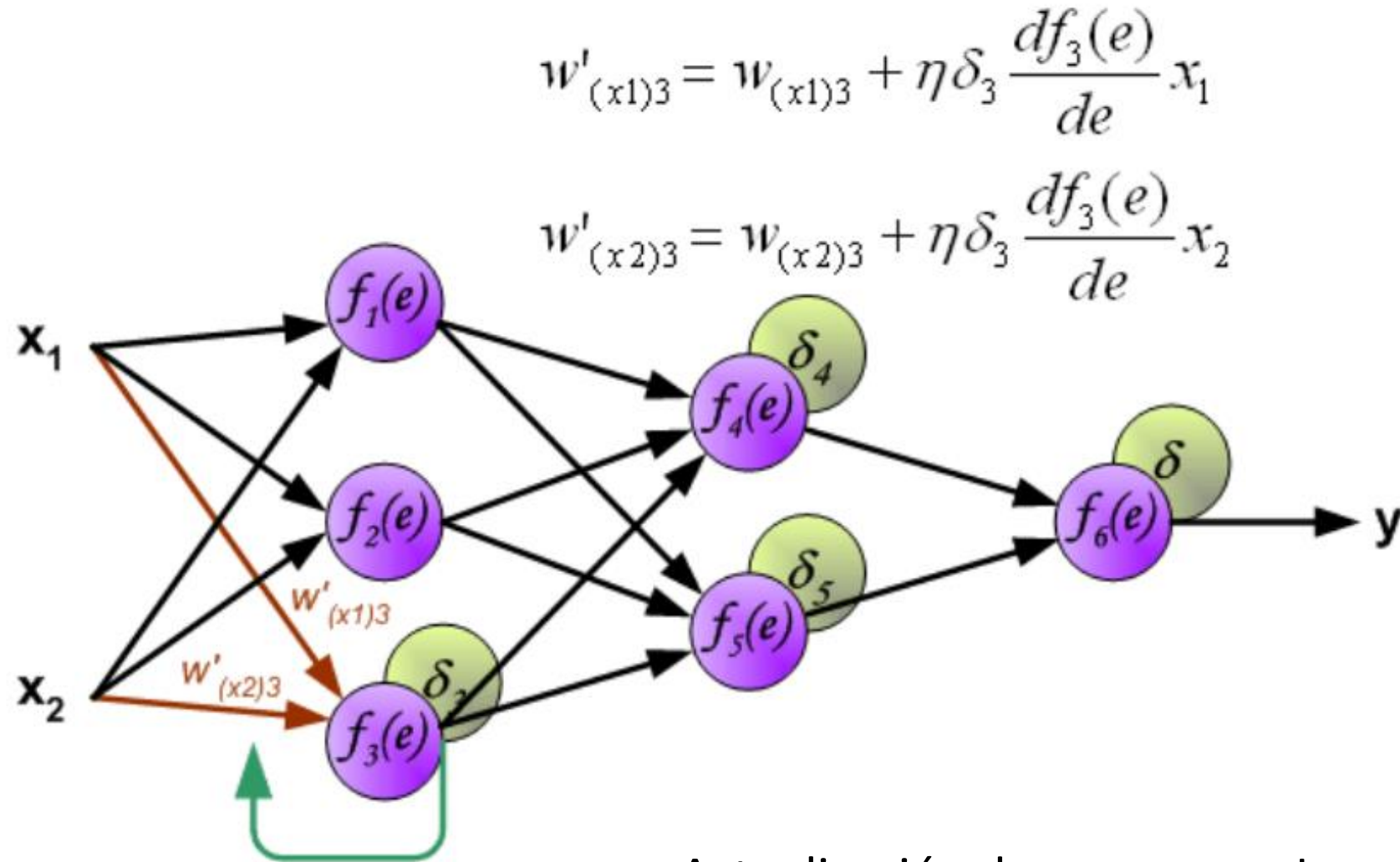


Actualización de pesos capa I, neurona 2.





# Backpropagation: Actualizando los pesos de la red



Actualización de pesos capa I, neurona 3.

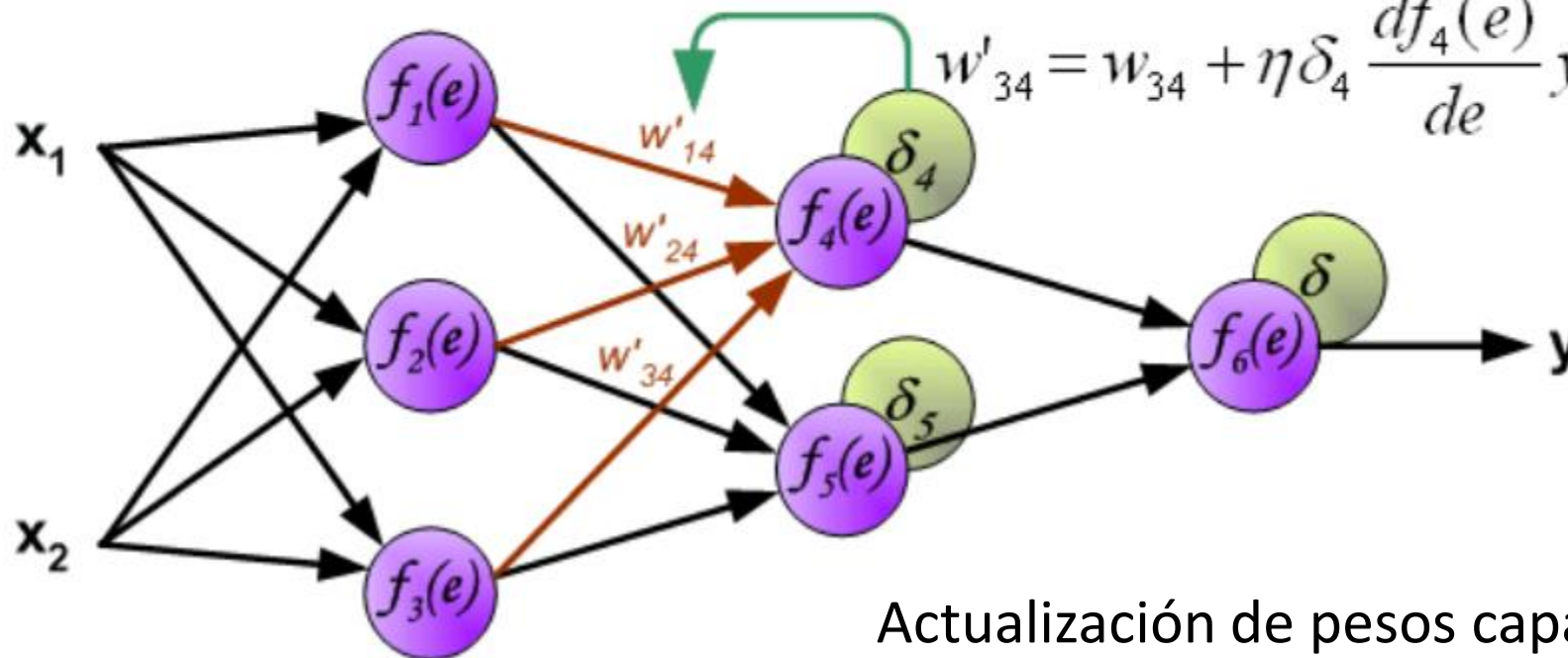


# Backpropagation: Actualizando los pesos de la red

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

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

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



Actualización de pesos capa II, neurona 1.



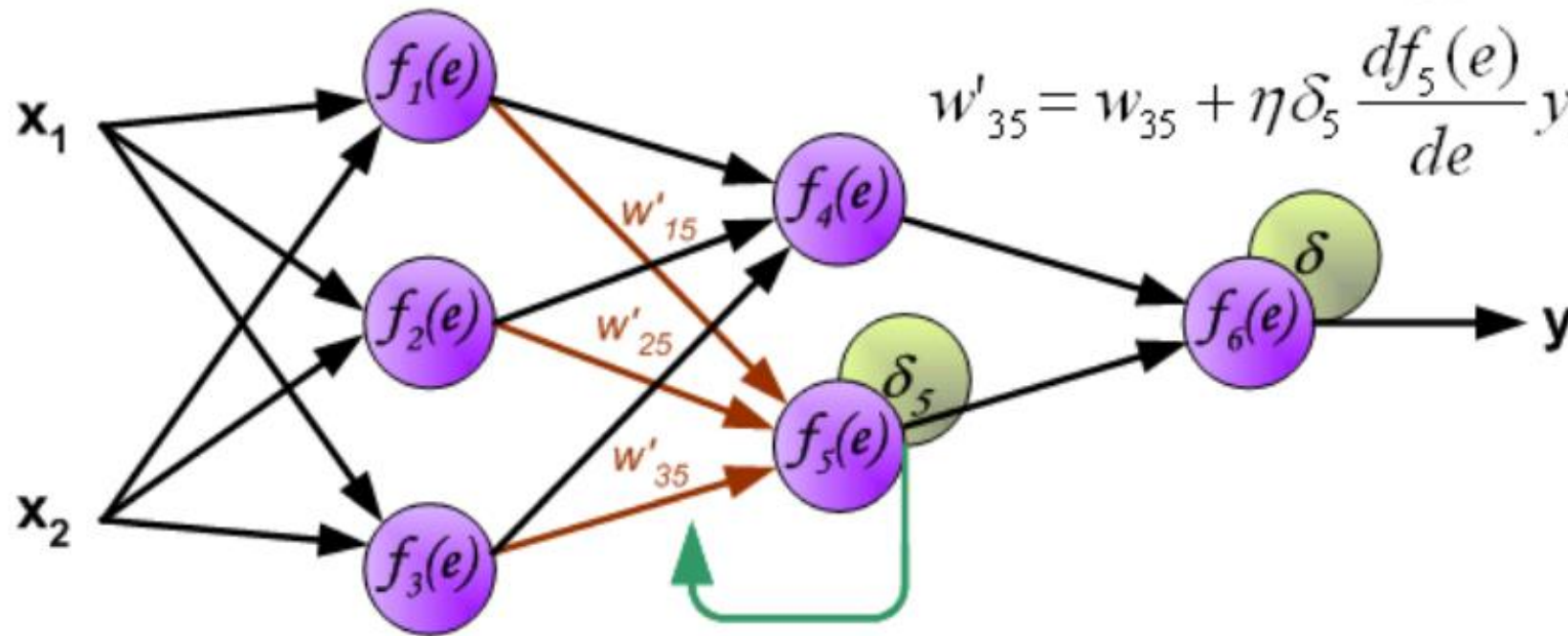


# Backpropagation: Actualizando los pesos de la red

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

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

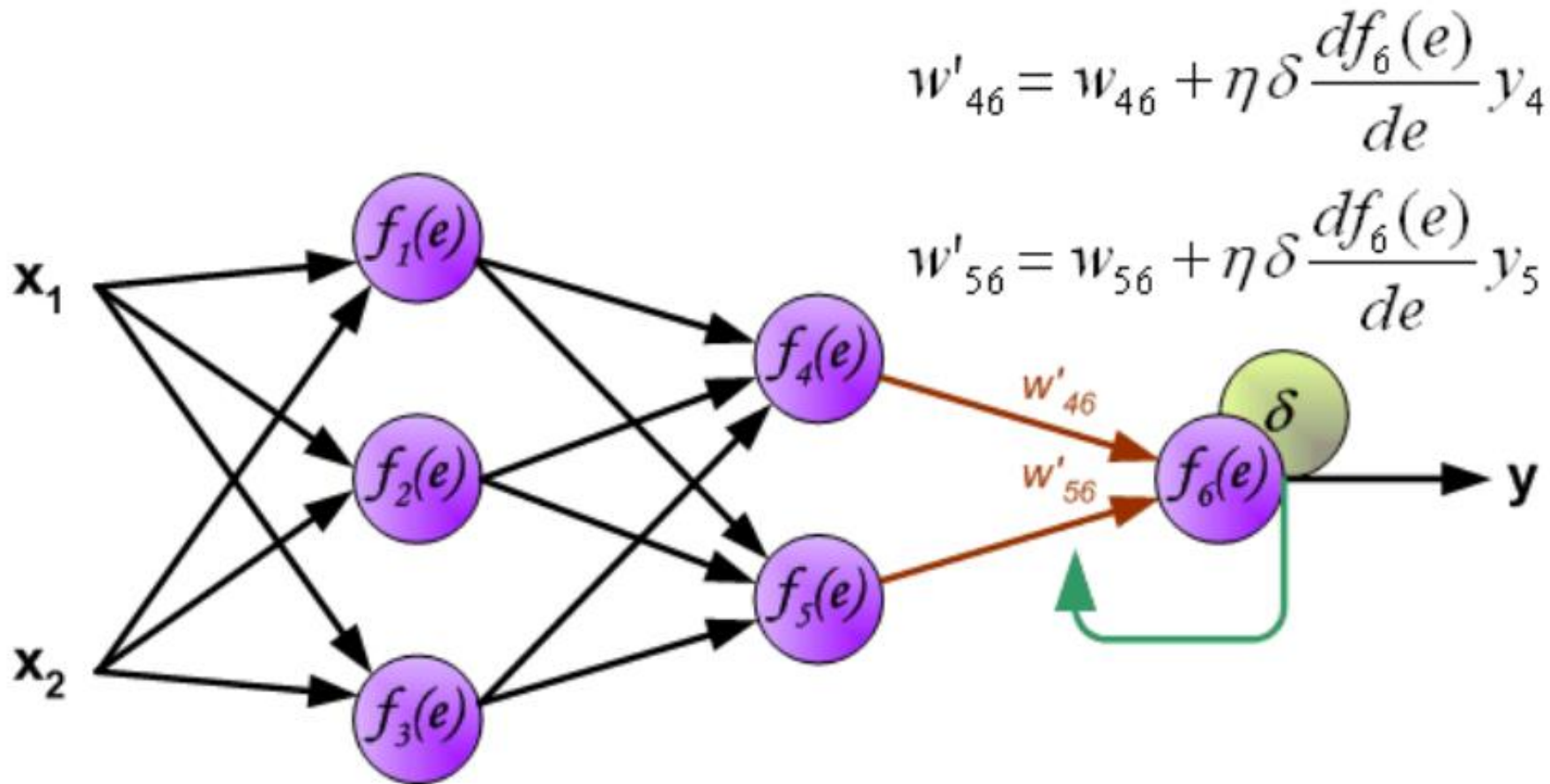
$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



Actualización de pesos capa II, neurona 2.



# Backpropagation: Actualizando los pesos de la red



Actualización de pesos capa III, neurona 1.



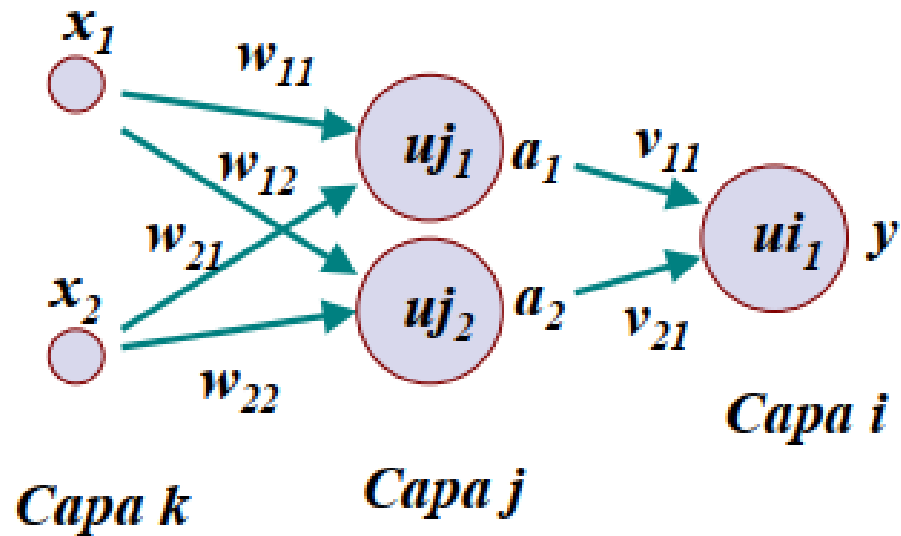
# Backpropagation: Consideraciones

- ❖ 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  $j$**

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

...

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

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