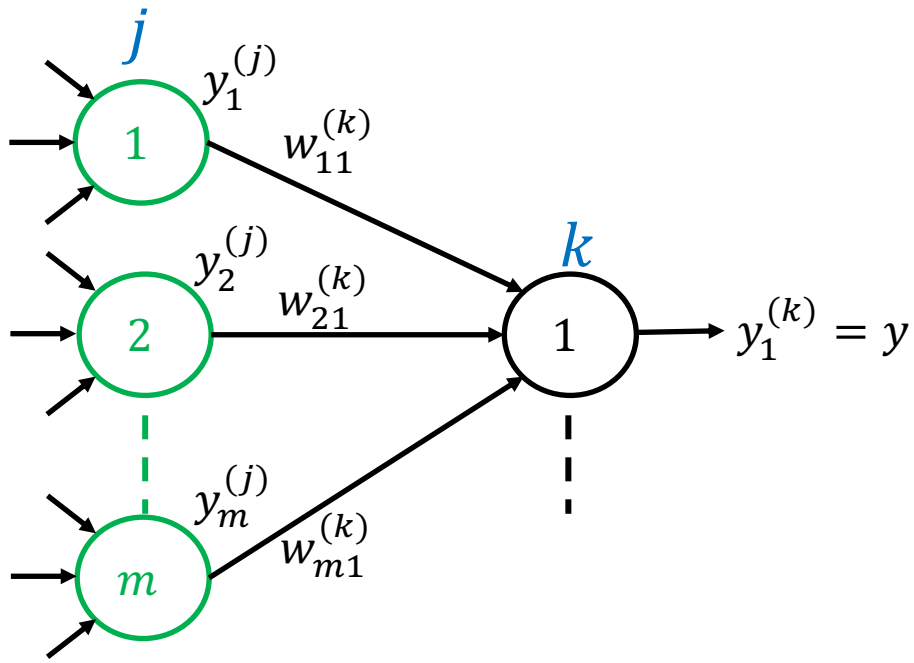
Tema 5: Algoritmo back-propagation

- δ para función de activación sigmoide.
- Cálculo de los δ's en las diferentes capas.
- Cálculo de los pesos, propagación hacia atrás.

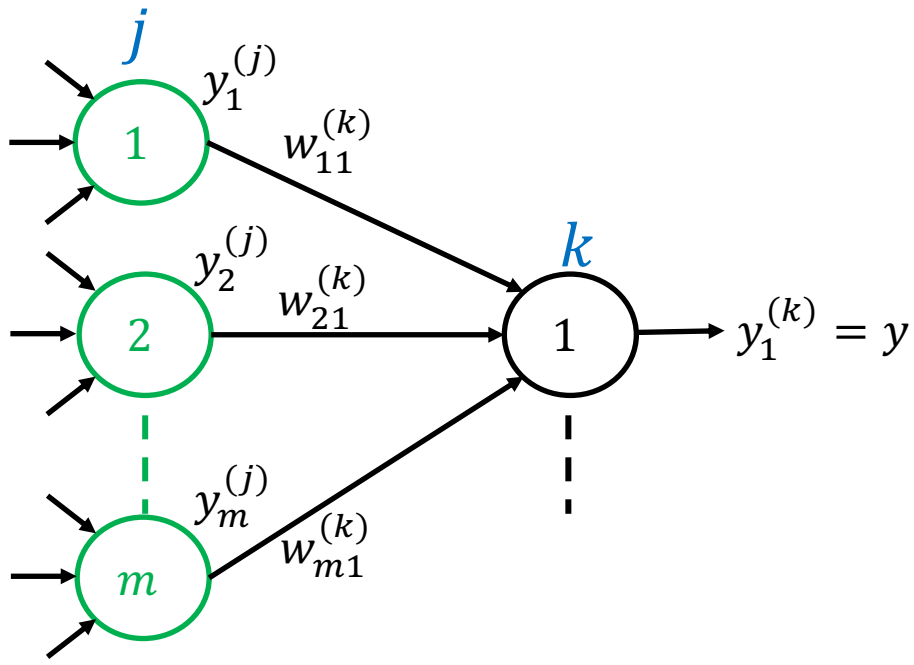
Algoritmo back-propagation:

- Cálculo de los pesos de la red multicapa, entrenamiento.



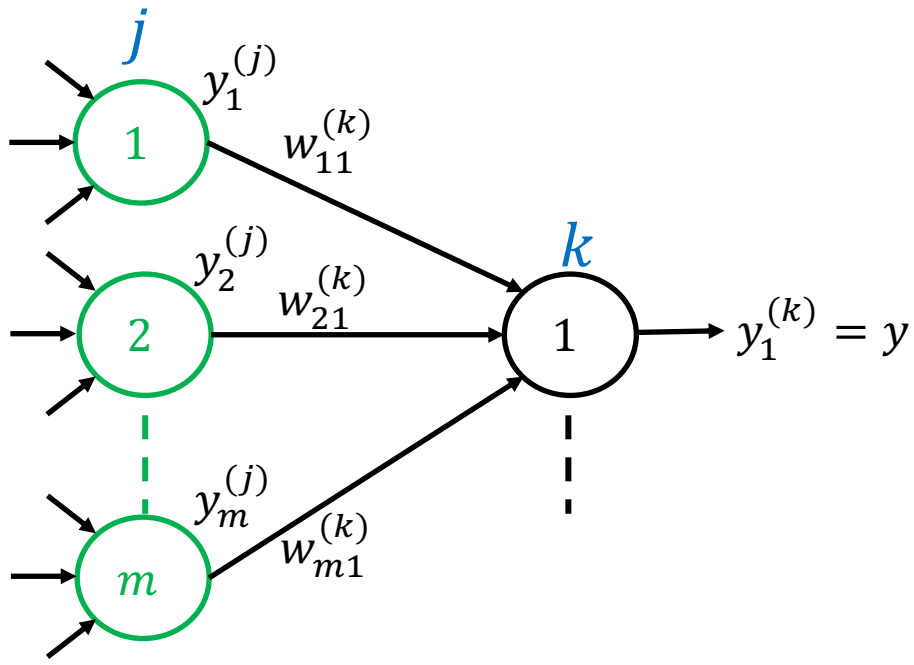
Algoritmo back-propagation:

- Cálculo de los pesos de la red multicapa, entrenamiento.
  1. Aplicamos el algoritmo del descenso de la pendiente en cada neurona de salida.
     (1)Tenemos m neuronas en la penúltima capa:
     - $f_1 = Loss(w_{11}, \cdots, w_{m1}) = (y_d - y)^2 \Rightarrow$ para la neurona $1^{(k)}$.



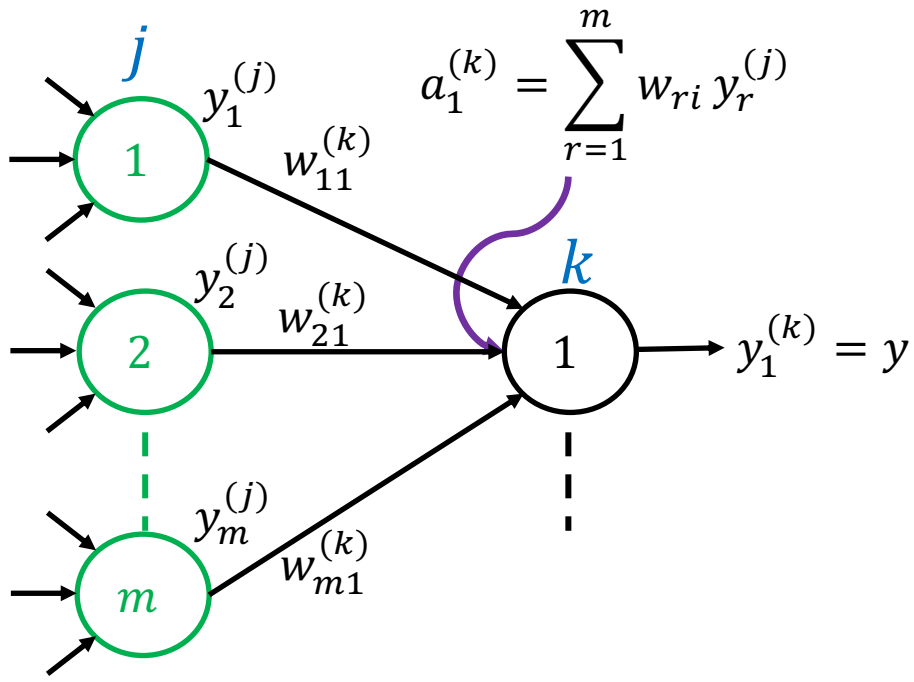
Algoritmo back-propagation:

- Cálculo de los pesos de la red multicapa, entrenamiento.
  1. Aplicamos el algoritmo del descenso de la pendiente en cada neurona de salida.
     (1)Tenemos m neuronas en la penúltima capa:
     - $f_1 = Loss(w_{11}, \cdots, w_{m1}) = (y_d - y)^2 \Rightarrow$ para la neurona $1^{(k)}$.
     - $\frac{\partial f_1}{\partial \mathbf{w}_1^{(k)}} = \left[\frac{\partial f_1}{\partial w_{11}^{(k)}}, \cdots, \frac{\partial f_1}{\partial w_{m1}^{(k)}}\right]$.



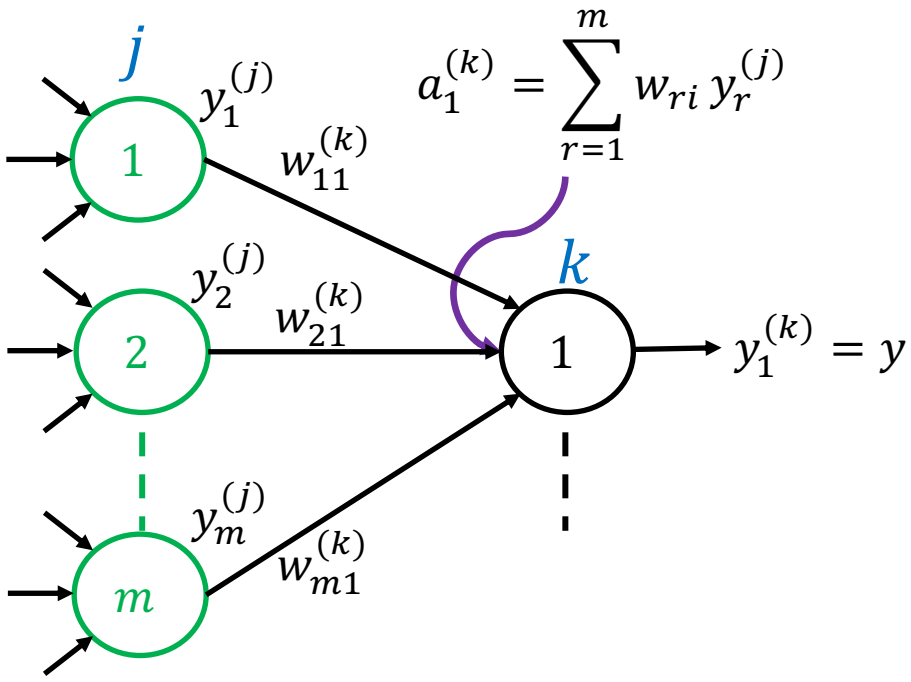
Algoritmo back-propagation:

- Cálculo de los pesos de la red multicapa, entrenamiento.
  1. Aplicamos el algoritmo del descenso de la pendiente en cada neurona de salida.
     (1)Tenemos m neuronas en la penúltima capa:
     - $f_1 = Loss(w_{11}, \cdots, w_{m1}) = (y_d - y)^2 \Rightarrow$ para la neurona $1^{(k)}$.
     - $\frac{\partial f_1}{\partial \mathbf{W}_1^{(k)}} = \left[\frac{\partial f_1}{\partial w_{11}^{(k)}}, \cdots, \frac{\partial f_1}{\partial w_{m1}^{(k)}}\right]$.
     - $\frac{\partial f_1}{\partial \mathbf{W}_1^{(k)}} = \frac{\partial f_1}{\partial a_1^{(k)}} \frac{\partial a_1^{(k)}}{\partial \mathbf{W}_1^{(k)}} = -2(y_d - y) \frac{\partial y}{\partial a_1^{(k)}} \mathbf{Y}^{(j)}$.
       - $\mathbf{W}_1^{(k)} = \left[w_{11}^{(k)}, \cdots, w_{m1}^{(k)}\right]$.
       - $\mathbf{Y}^{(j)} = \left[y_1^{(j)}, \cdots, y_m^{(j)}\right]$.



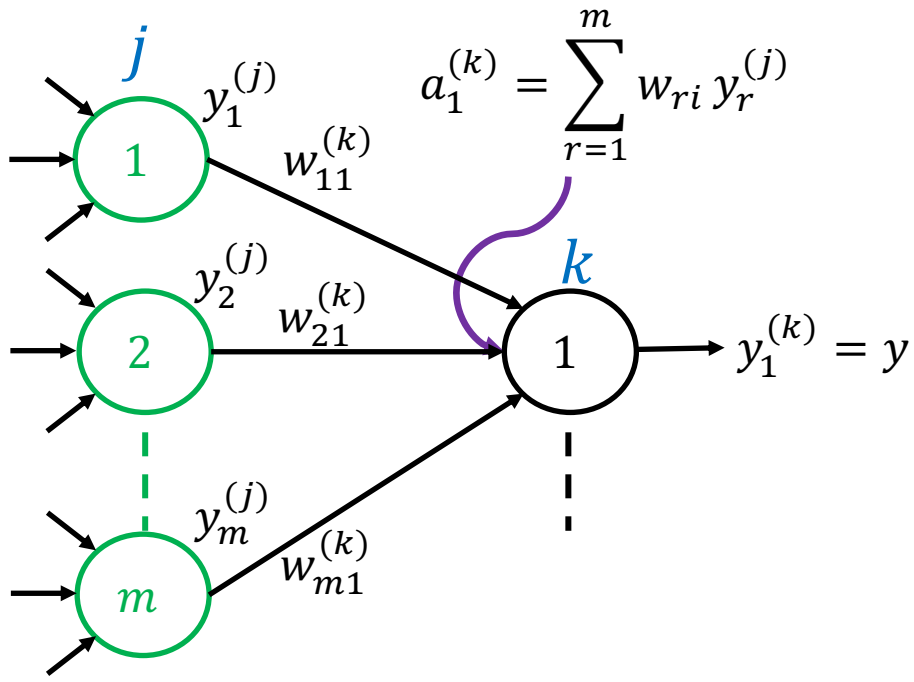
Algoritmo back-propagation:

- Cálculo de los pesos de la red multicapa, entrenamiento.
  1. Aplicamos el algoritmo del descenso de la pendiente en cada neurona de salida.
     (1)Tenemos m neuronas en la penúltima capa:
     - $f_1 = Loss(w_{11}, \cdots, w_{m1}) = (y_d - y)^2 \Rightarrow$ para la neurona $1^{(k)}$.
     - $\frac{\partial f_1}{\partial \mathbf{w}_1^{(k)}} = \left[\frac{\partial f_1}{\partial w_{11}^{(k)}}, \cdots, \frac{\partial f_1}{\partial w_{m1}^{(k)}}\right]$.
     - $\frac{\partial f_1}{\partial \mathbf{w}_1^{(k)}} = \frac{\partial f_1}{\partial a_1^{(k)}} \frac{\partial a_1^{(k)}}{\partial \mathbf{w}_1^{(k)}} = -2(y_d - y) \frac{\partial y}{\partial a_1^{(k)}} \mathbf{Y}^{(j)}$.
       - $\mathbf{W}_1^{(k)} = \left[w_{11}^{(k)}, \cdots, w_{m1}^{(k)}\right]$.
       - $\mathbf{Y}^{(j)} = \left[y_1^{(j)}, \cdots, y_m^{(j)}\right]$.
     - $\delta_1^k = (y_d - y) \frac{\partial y}{\partial a_1^{(k)}}$.



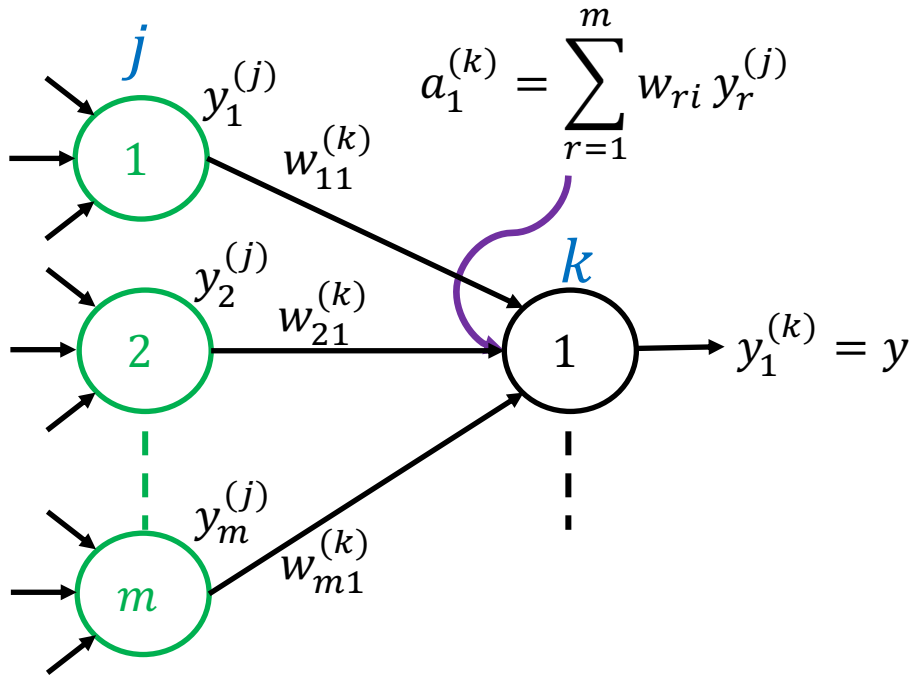
Algoritmo back-propagation:

- Cálculo de los pesos de la red multicapa, entrenamiento.
  1. Aplicamos el algoritmo del descenso de la pendiente en cada neurona de salida.
     (1)Tenemos m neuronas en la penúltima capa:
     - $f_1 = Loss(w_{11}, \cdots, w_{m1}) = (y_d - y)^2 \Rightarrow$ para la neurona $1^{(k)}$.
     - $\frac{\partial f_1}{\partial \mathbf{W}_1^{(k)}} = \left[\frac{\partial f_1}{\partial w_{11}^{(k)}}, \cdots, \frac{\partial f_1}{\partial w_{m1}^{(k)}}\right]$.
     - $\frac{\partial f_1}{\partial \mathbf{W}_1^{(k)}} = \frac{\partial f_1}{\partial a_1^{(k)}} \frac{\partial a_1^{(k)}}{\partial \mathbf{W}_1^{(k)}} = -2(y_d - y) \frac{\partial y}{\partial a_1^{(k)}} \mathbf{Y}^{(j)}$.
       - $\mathbf{W}_1^{(k)} = \left[w_{11}^{(k)}, \cdots, w_{m1}^{(k)}\right]$.
       - $\mathbf{Y}^{(j)} = \left[y_1^{(j)}, \cdots, y_m^{(j)}\right]$.
     - $\delta_1^k = (y_d - y) \frac{\partial y}{\partial a_1^{(k)}}$.
  2. Actualización de los pesos:
     - $\mathbf{W}_1^{(k)} = \mathbf{W}_1^{(k)} + c_1^{(k)} \delta_1^{(k)} \mathbf{Y}^{(j)}$.
     - $c_1^{(k)}$ es el ritmo de aprendizaje para el vector $\mathbf{W}_1^{(k)}$.



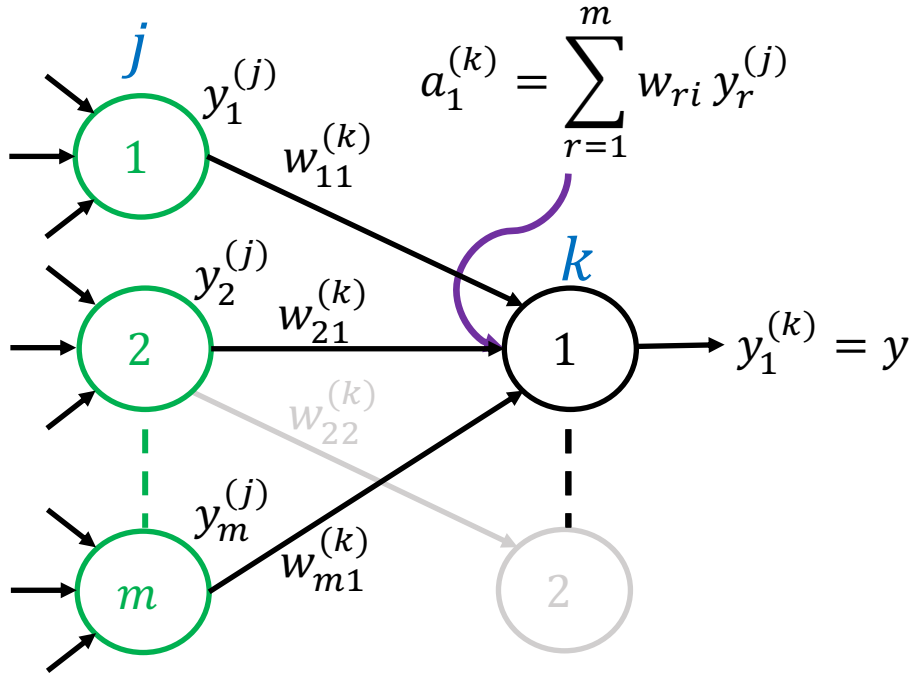
Algoritmo back-propagation:

- Para $y = \text{sigmoid}\left(a_1^{(k)}\right)$:
  - $\delta_1^k = (y_d - y)\frac{\partial y}{\partial a_1^{(k)}} = (y_d - y)y(1-y)$.
  - $\mathbf{W}_1^{(k)} = \mathbf{W}_1^{(k)} + c_1^{(k)}(y_d - y)y(1-y)\mathbf{Y}^{(j)}$.



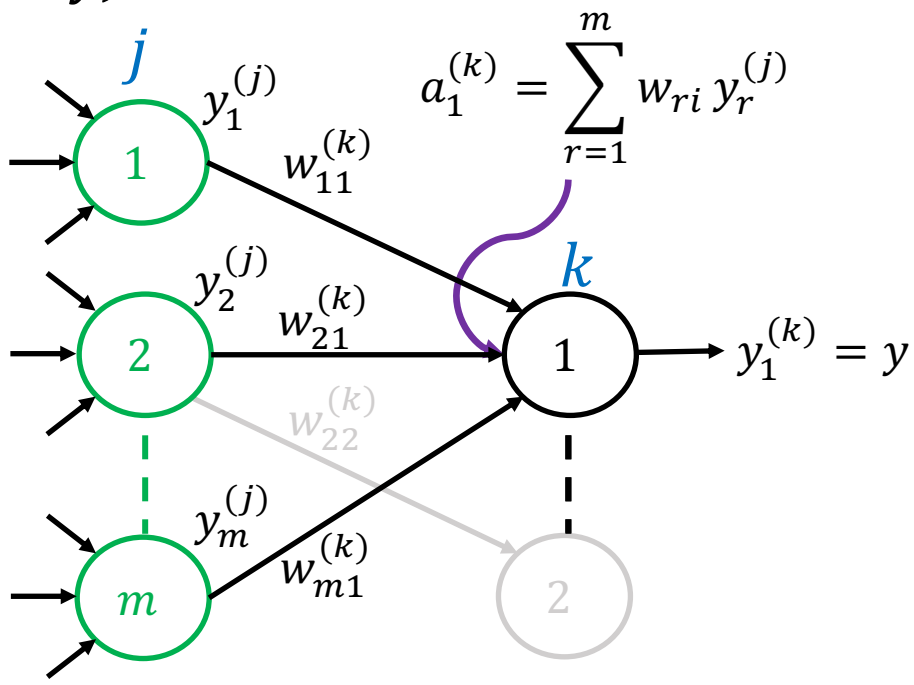
Algoritmo back-propagation:

- Cálculo de los pesos de las capas intermedias (sigmoide).
  - Tenemos que calcular los $\delta_l^{(j)}$ de las neuronas de la capa intermedia:
    - $\boldsymbol{\delta_1^{(k)}} = (y_d - y)\frac{\partial y}{\partial a_1^{(k)}} = (y_d - y)y(1-y) = \boldsymbol{\varepsilon_k y(1-y)}$.
    - El $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)\left(y_{d2} - y_2^{(j)}\right)$.
    - $y_{d2} - y_2^{(j)}$, no lo conocemos.



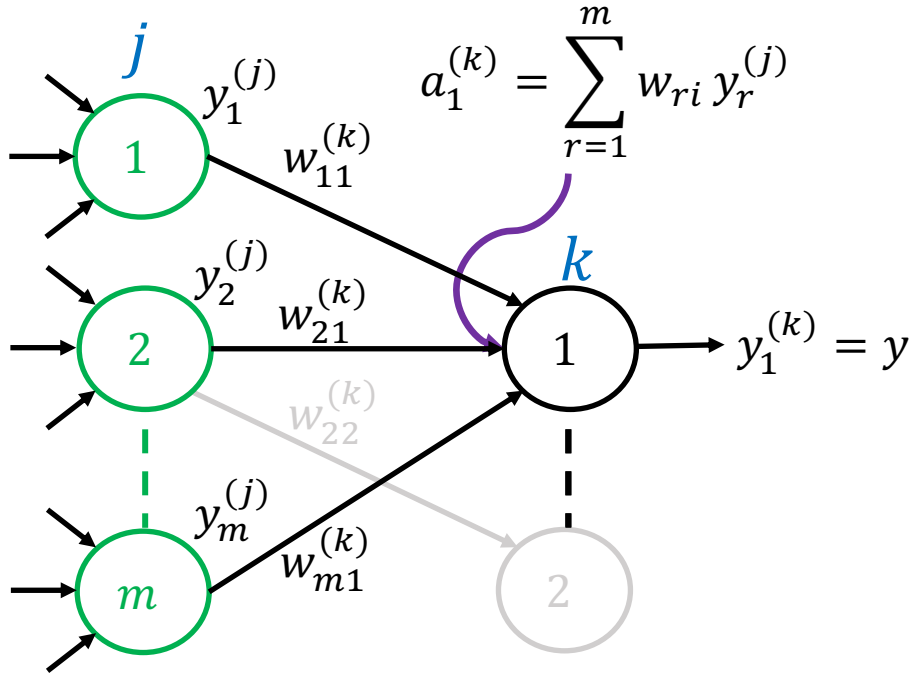
Algoritmo back-propagation:

- Cálculo de los pesos de las capas intermedias (sigmoide).
  - Tenemos que calcular los $\delta_l^{(j)}$ de las neuronas de la capa intermedia:
    - $\boldsymbol{\delta_1^{(k)}} = (y_d - y)\frac{\partial y}{\partial a_1^{(k)}} = (y_d - y)y(1-y) = \boldsymbol{\varepsilon_k y(1-y)}$.
    - El $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)\left(y_{d2} - y_2^{(j)}\right)$.
    - $y_{d2} - y_2^{(j)}$, no lo conocemos.
  - Hacemos una propagación hacia atrás del δ:
    - $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)\sum_{i=1}^{n} w_{2i}^{(k)}\delta_i^{(k)}$.
    - En la figura solo tenemos una neurona de salida:
      - $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)w_{21}^{(k)}\delta_1^{(k)}$.



Algoritmo back-propagation:

- Cálculo de los pesos de las capas intermedias (sigmoide).
  - Tenemos que calcular los $\delta_l^{(j)}$ de las neuronas de la capa intermedia:
    - $\boldsymbol{\delta_1^{(k)}} = (y_d - y)\frac{\partial y}{\partial a_1^{(k)}} = (y_d - y)y(1-y) = \boldsymbol{\varepsilon_k y(1-y)}$.
    - El $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)\left(y_{d2} - y_2^{(j)}\right)$.
    - $y_{d2} - y_2^{(j)}$, no lo conocemos.
  - Hacemos una propagación hacia atrás del δ:
    - $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)\sum_{i=1}^{n} w_{2i}^{(k)}\delta_i^{(k)}$.
    - En la figura solo tenemos una neurona de salida:
      - $\delta_2^{(j)} = y_2^{(j)}\left(1 - y_2^{(j)}\right)w_{21}^{(k)}\delta_1^{(k)}$.
  - Una vez tenemos todos los δ's:
    - Actualizamos los pesos de la red.



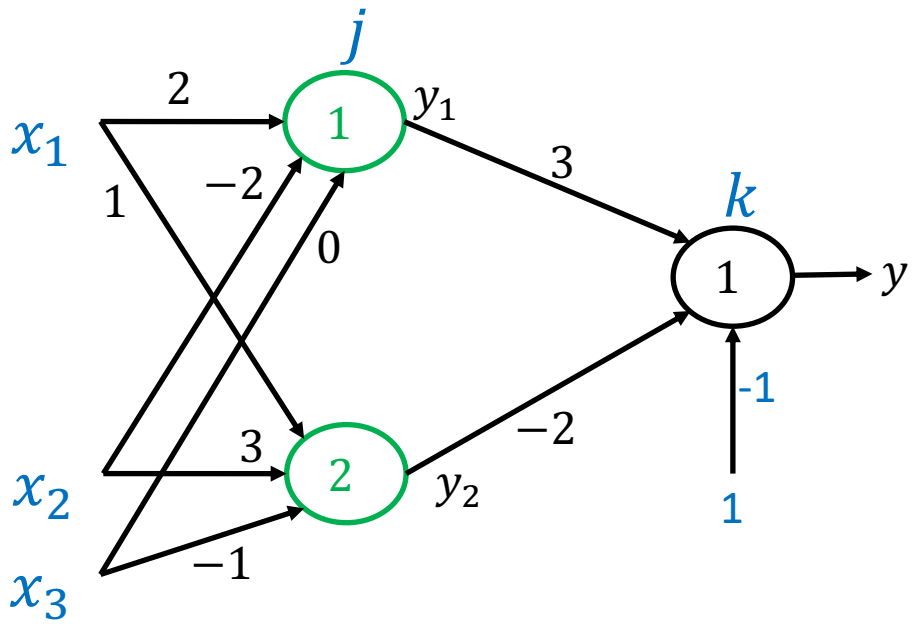
Algoritmo back-propagation (resumen):

1. Propagamos hacia adelante.
2. Calculamos los δ's salida.
3. Calculamos los δ's de las neuronas de las capas intermedias.
4. Actualizamos pesos.
5. Repetimos el proceso para todos los ejemplos.
6. Entrenamos a la red con los ejemplos hasta que converja.

Algoritmo back-propagation (ejemplo):

- Dada la red neuronal de la figura y los datos de entrenamiento de la tabla, calcular los nuevos valores de los pesos después de una iteración, considerar un ritmo de aprendizaje de 1 y función de activación sigmoide.

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

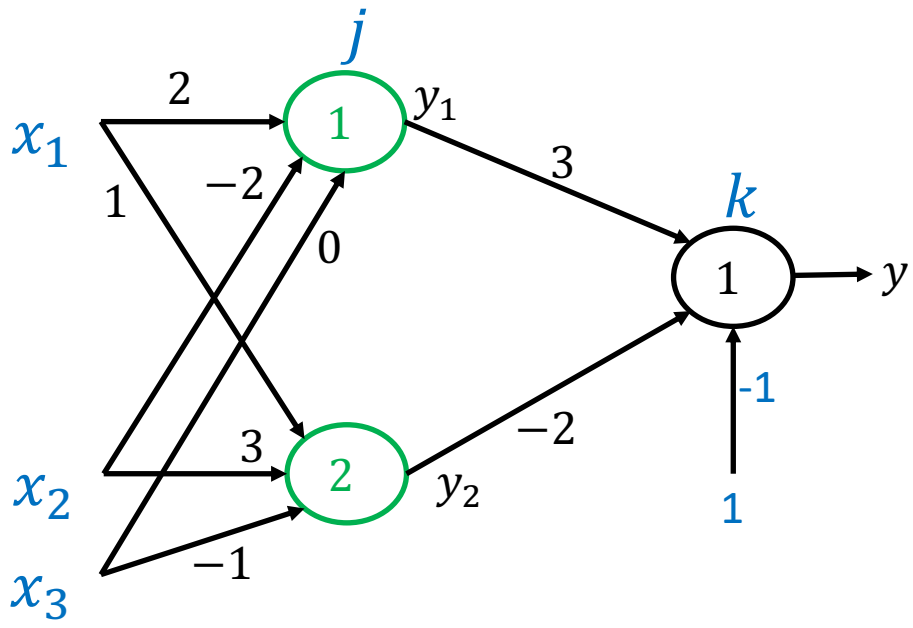


Algoritmo back-propagation (ejemplo):

$$\frac{1}{1+e^{-\sum w_i x_i}}$$

1. Propagamos hacia adelante:
   - $y_1 = \text{sigmoid}(2 \cdot 1 - 2 \cdot 0 + 0 \cdot 1) = 0.881$
   - $y_2 = \text{sigmoid}(1 \cdot 1 - 3 \cdot 0 - 1 \cdot 1) = 0.5$
   - $y = \text{sigmoid}(3 \cdot 0.881 - 2 \cdot 0.5 - 1) = 0.655$

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

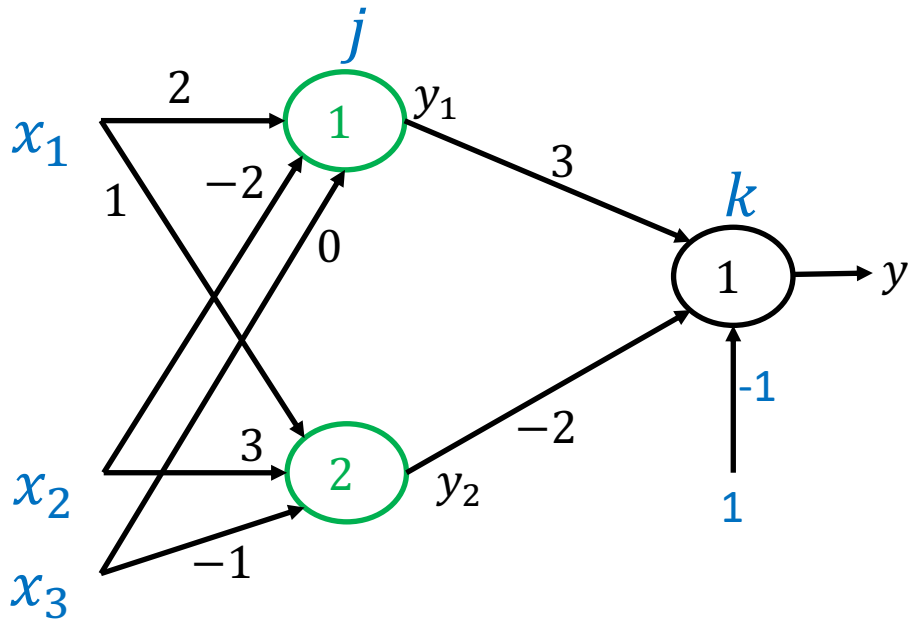


Algoritmo back-propagation (ejemplo):

$$\frac{1}{1+e^{-\sum w_i x_i}}$$

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

1. Propagamos hacia adelante:
   - $y_1 = \text{sigmoid}(2 \cdot 1 - 2 \cdot 0 + 0 \cdot 1) = 0.881$
   - $y_2 = \text{sigmoid}(1 \cdot 1 - 3 \cdot 0 - 1 \cdot 1) = 0.5$
   - $y = \text{sigmoid}(3 \cdot 0.881 - 2 \cdot 0.5 - 1) = 0.655$
2. Calculamos el delta de salida:
   - $\delta_1^k = y(1-y)(y_d - y) =$
     $= 0.655(1 - 0.655)(0 - 0.655) = -0.148.$

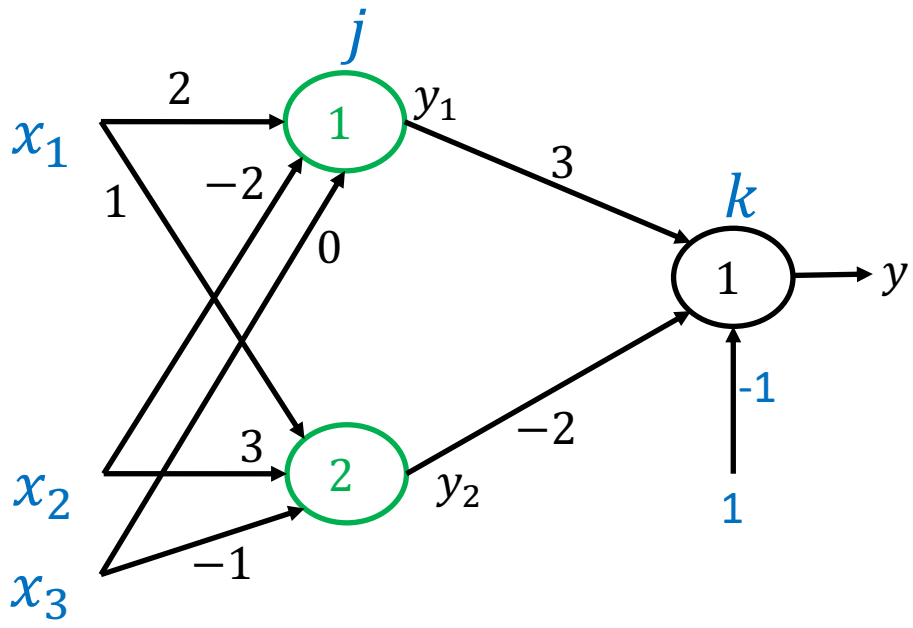


Algoritmo back-propagation (ejemplo):

$$\frac{1}{1+e^{-\sum w_i x_i}}$$

3. Calculamos los δ's intermedios:
   - $\delta_1^{(j)} = y_1^{(j)}\left(1-y_1^{(j)}\right) w_{11}^{(k)} \delta_1^k =$
     $0.881 \cdot (1-0.881) \cdot 3 \cdot (-0.148) = -0.0467$
   - $\delta_2^{(j)} = y_2^{(j)}\left(1-y_2^{(j)}\right) w_{21}^{(k)} \delta_2^k =$
     $0.5 \cdot (1-0.5) \cdot (-2) \cdot (-0.148) = 0.074$

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

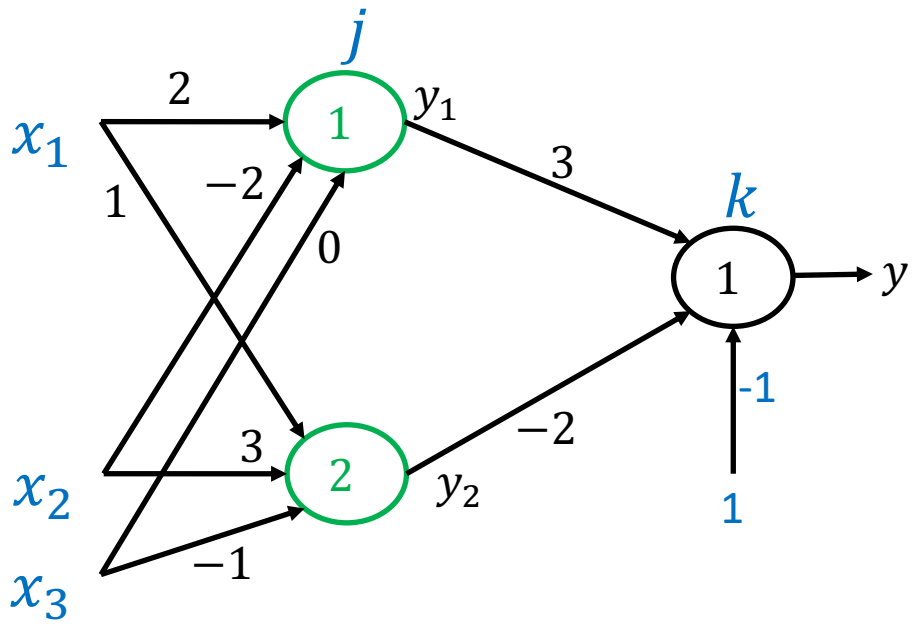


Algoritmo back-propagation (ejemplo):

$$\frac{1}{1+e^{-\sum w_i x_i}}$$

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

3. Calculamos los δ's intermedios:
   - $\delta_1^{(j)} = y_1^{(j)}\left(1-y_1^{(j)}\right)w_{11}^{(k)}\delta_1^k =$
     $0.881 \cdot (1-0.881) \cdot 3 \cdot (-0.148) = -0.0467$
   - $\delta_2^{(j)} = y_2^{(j)}\left(1-y_2^{(j)}\right)w_{21}^{(k)}\delta_2^k =$
     $0.5 \cdot (1-0.5) \cdot (-2) \cdot (-0.148) = 0.074$
4. Actualizamos pesos:
   - Capa de salida (k):
     - $\mathbf{W}^{(k)} = \mathbf{W}^{(k)} + c^{(k)}\delta_1^k\mathbf{Y}^{(j)} =$
       $= \begin{bmatrix}3\\-2\\-1\end{bmatrix} + 1 \cdot (-0.148) \cdot \begin{bmatrix}0.881\\0.5\\1\end{bmatrix} = \begin{bmatrix}2.87\\-2.074\\-1.148\end{bmatrix}.$



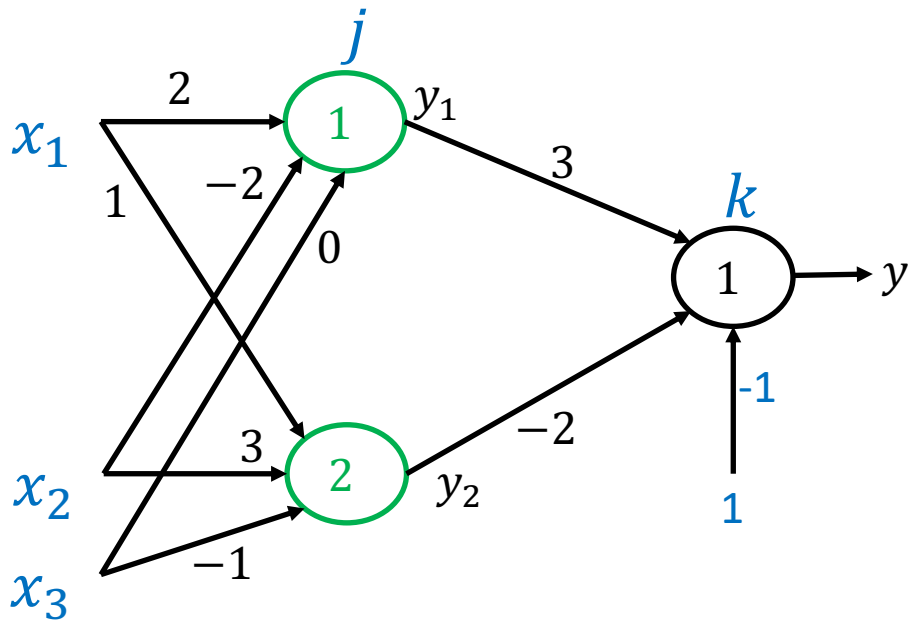
Algoritmo back-propagation (ejemplo):

$$\frac{1}{1+e^{-\sum w_i x_i}}$$

4. Actualizamos pesos:
   - Capa intermedia (j):
     - $\mathbf{W}_1^{(j)} = \mathbf{W}_1^{(j)} + c^{(j)}\delta_1^j \mathbf{X}^{(j)} =$

$$= \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix} + 1 \cdot (-0.0467) \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.953 \\ -2 \\ -0.0467 \end{bmatrix}.$$

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

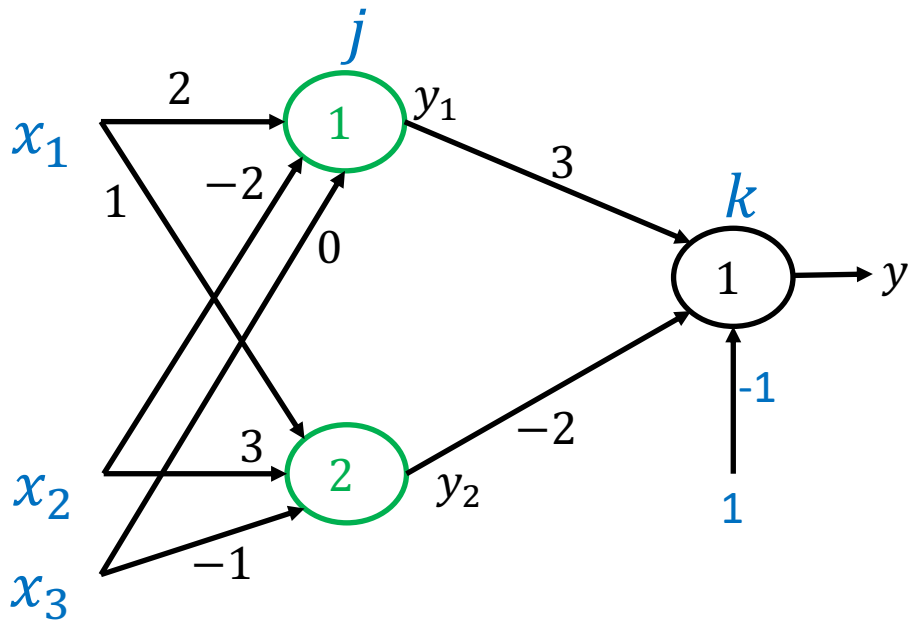


Algoritmo back-propagation (ejemplo):

$$\frac{1}{1+e^{-\sum w_i x_i}}$$

4. Actualizamos pesos:
   - Capa intermedia (j):
     - $\mathbf{W}_1^{(j)} = \mathbf{W}_1^{(j)} + c^{(j)}\delta_1^j \mathbf{X}^{(j)} =$
       $= \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix} + 1 \cdot (-0.0467) \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.953 \\ -2 \\ -0.0467 \end{bmatrix}.$
     - $\mathbf{W}_2^{(j)} = \mathbf{W}_2^{(j)} + c^{(j)}\delta_2^j \mathbf{X}^{(j)} =$
       $= \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix} + 1 \cdot (0.074) \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.074 \\ 3 \\ -0.926 \end{bmatrix}.$

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1



Algoritmo back-propagation (ejemplo):

- Dada la red neuronal de la figura y los datos de entrenamiento de la tabla, calcular los nuevos valores de los pesos después de una iteración, considerar un ritmo de aprendizaje de 1 y función de activación sigmoide.

x_1	x_2	x_3	y_d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1