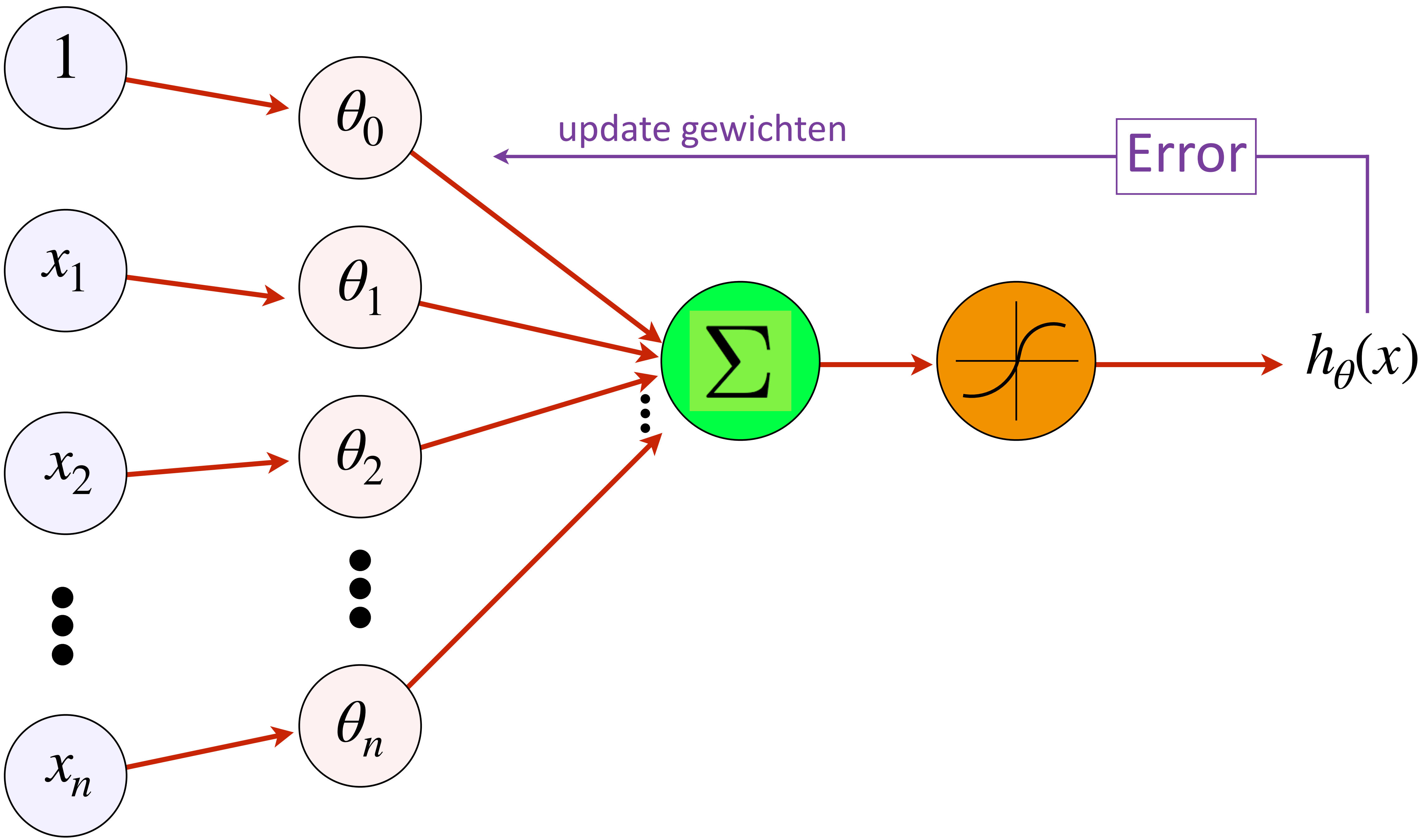
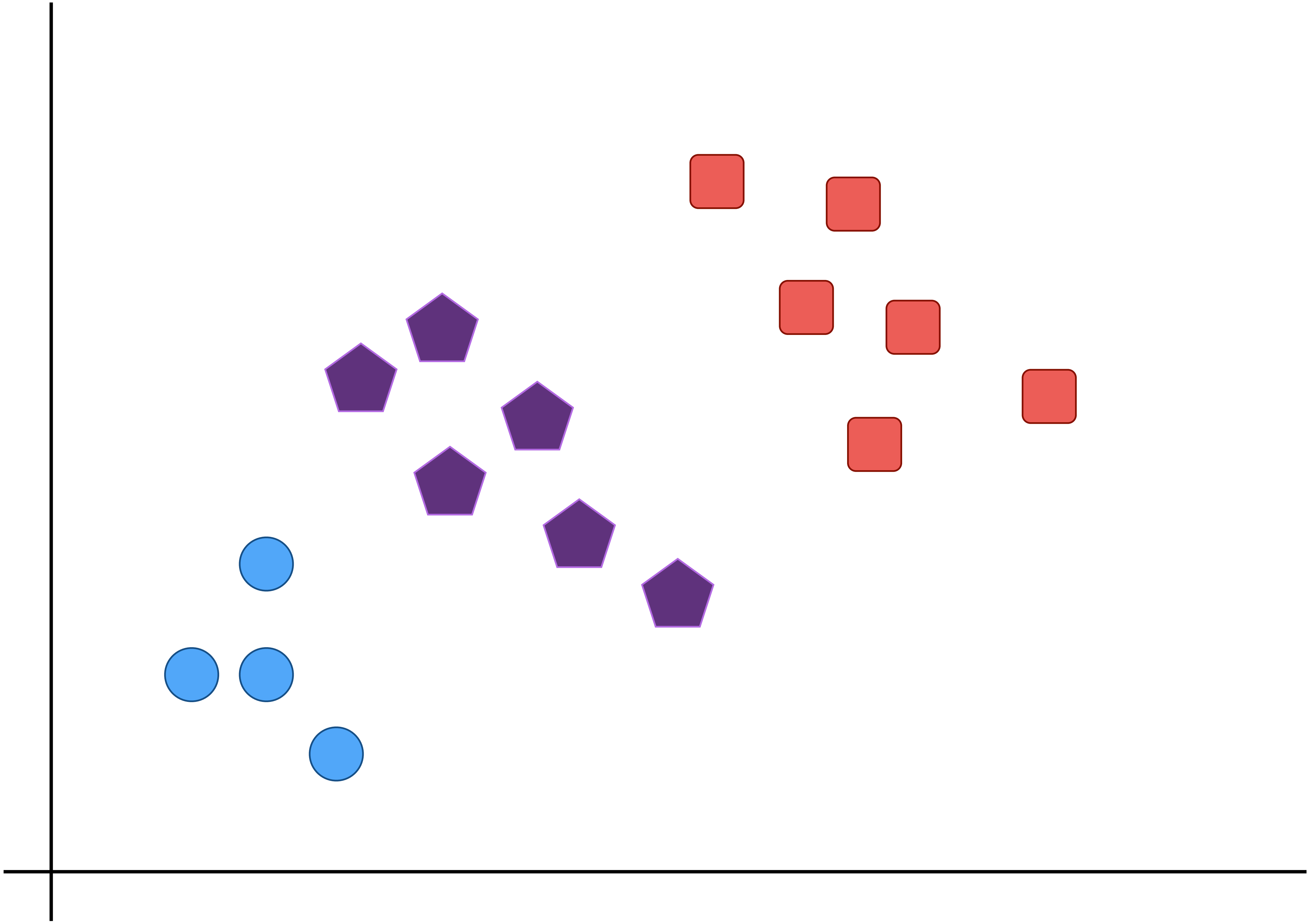
Machine Learning

4. neurale netwerken

ml:neurale netwerken



https://nl.wikipedia.org/wiki/Perceptron

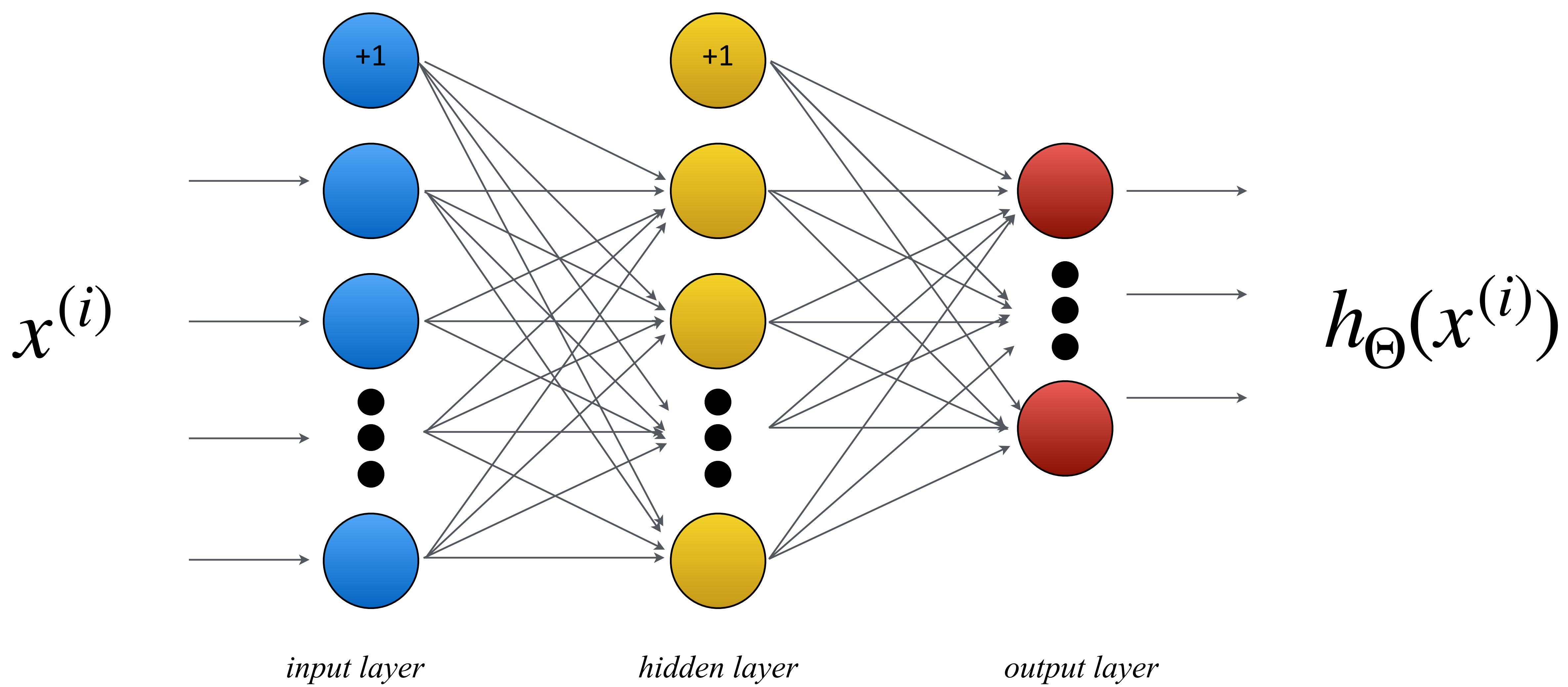


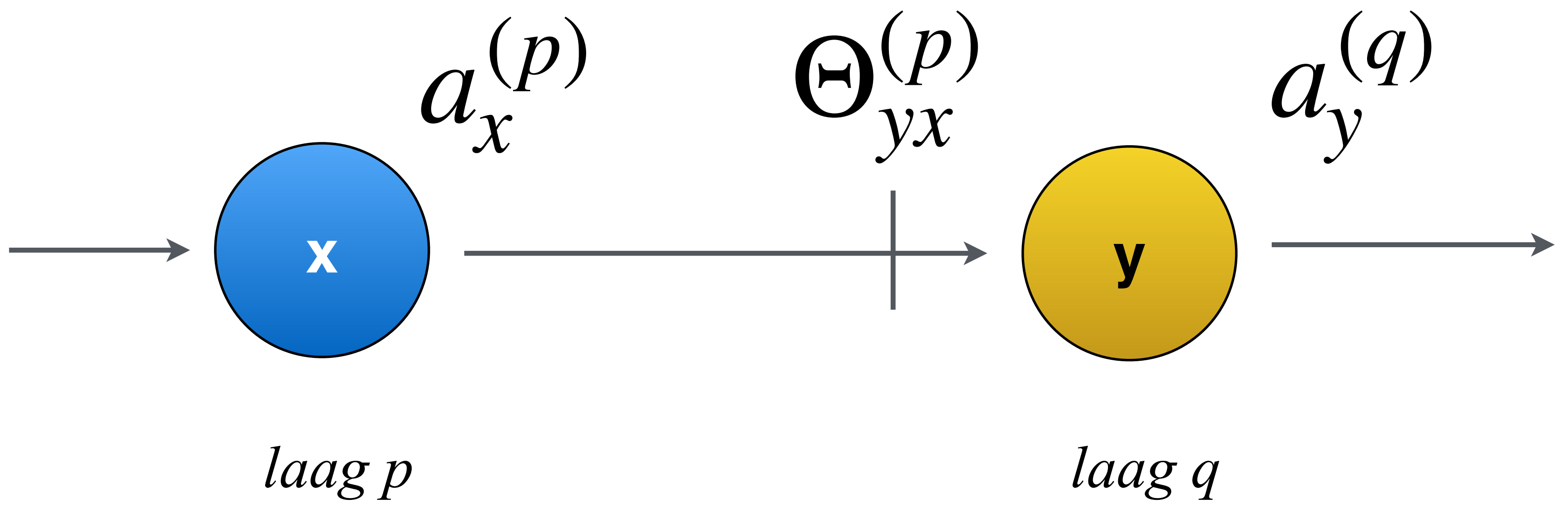
It is observed that non-linear problems can not be solved using single layer network with conventional type of neuron activation function

http://hsmazumdar.net/single_layer_neural_net.htm wat wij zien

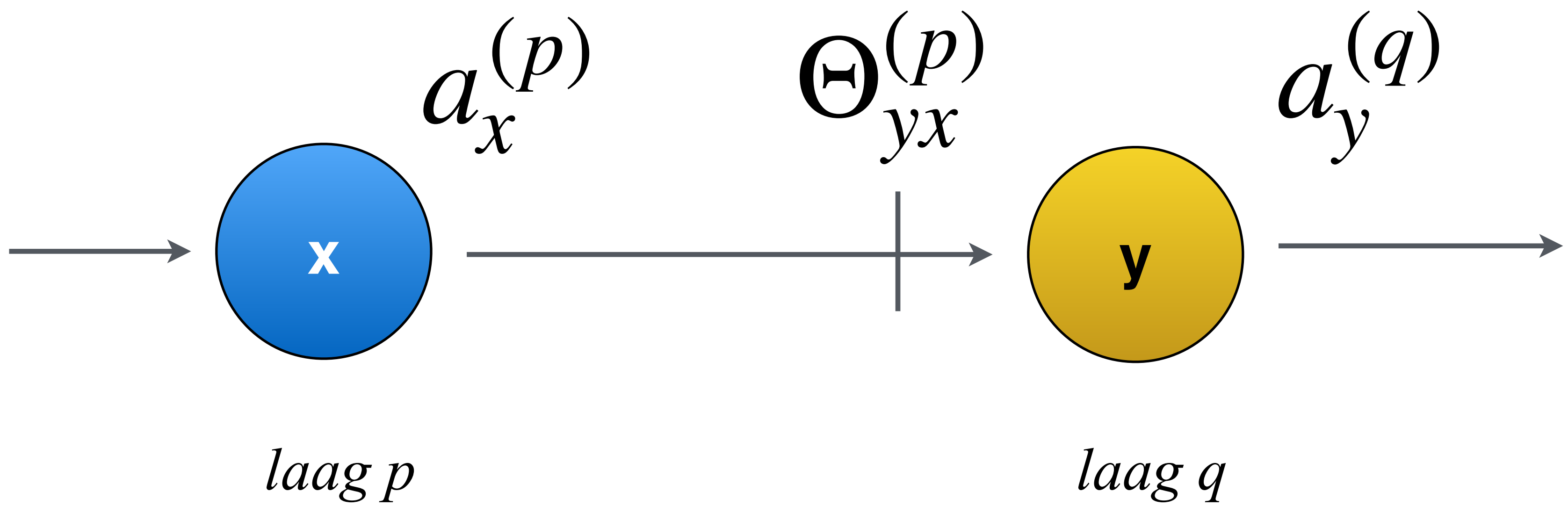
wat de computer ‘ziet’

```
[       [ 4.9,  3.0,  1.4],
        [ 4.7,  3.2,  1.3],
        [ 4.6,  3.1,  1.5],
        [ 5.0,  3.6,  1.4],
        [ 5.4,  3.9,  1.7],
        [ 4.6,  3.4,  1.4],
        [ 5.0,  3.4,  1.5],
        [ 4.4,  2.9,  1.4],
        [ 4.9,  3.1,  1.5],
        [ 5.4,  3.7,  1.5],
        [ 4.8,  3.4,  1.6],
        [ 4.8,  3.0,  1.4],
        [ 4.3,  3.0,  1.1],
        [ 5.8,  4.0,  1.2],
        [ 5.7,  4.4,  1.5],
        [ 5.4,  3.9,  1.3],
        [ 5.1,  3.5,  1.4],
        [ 5.7,  3.8,  1.7],
        [ 5.1,  3.8,  1.5],
…
        [ 5.4,  3.4,  1.7],
]
```


+1
+1
$x^{(i)}$
$h_\Theta(x^{(i)})$
input layer
hidden layer
output layer

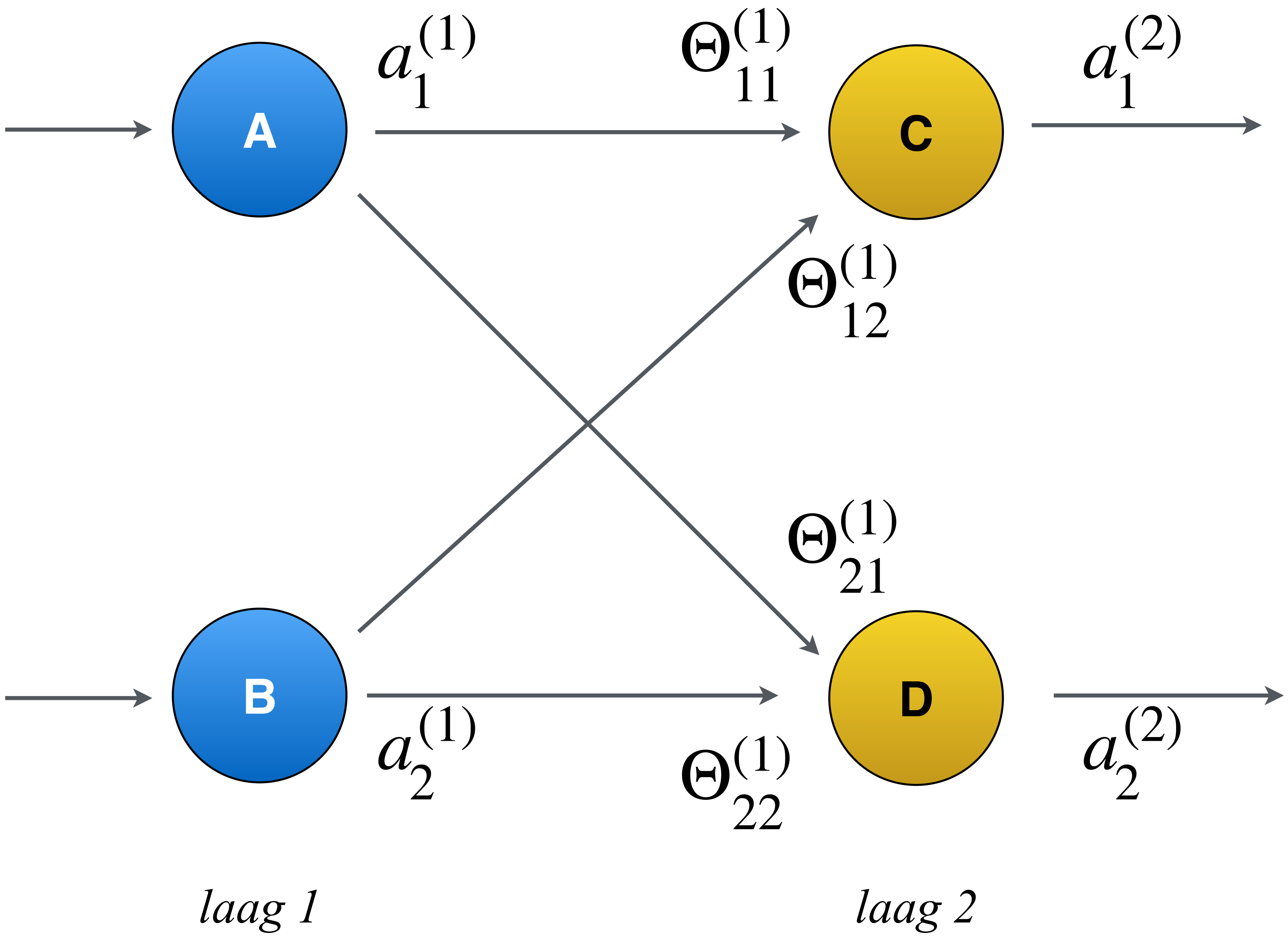


$a_x^{(p)}$
$\Theta_{yx}^{(p)}$
$a_y^{(q)}$
x
y
laag p
laag q



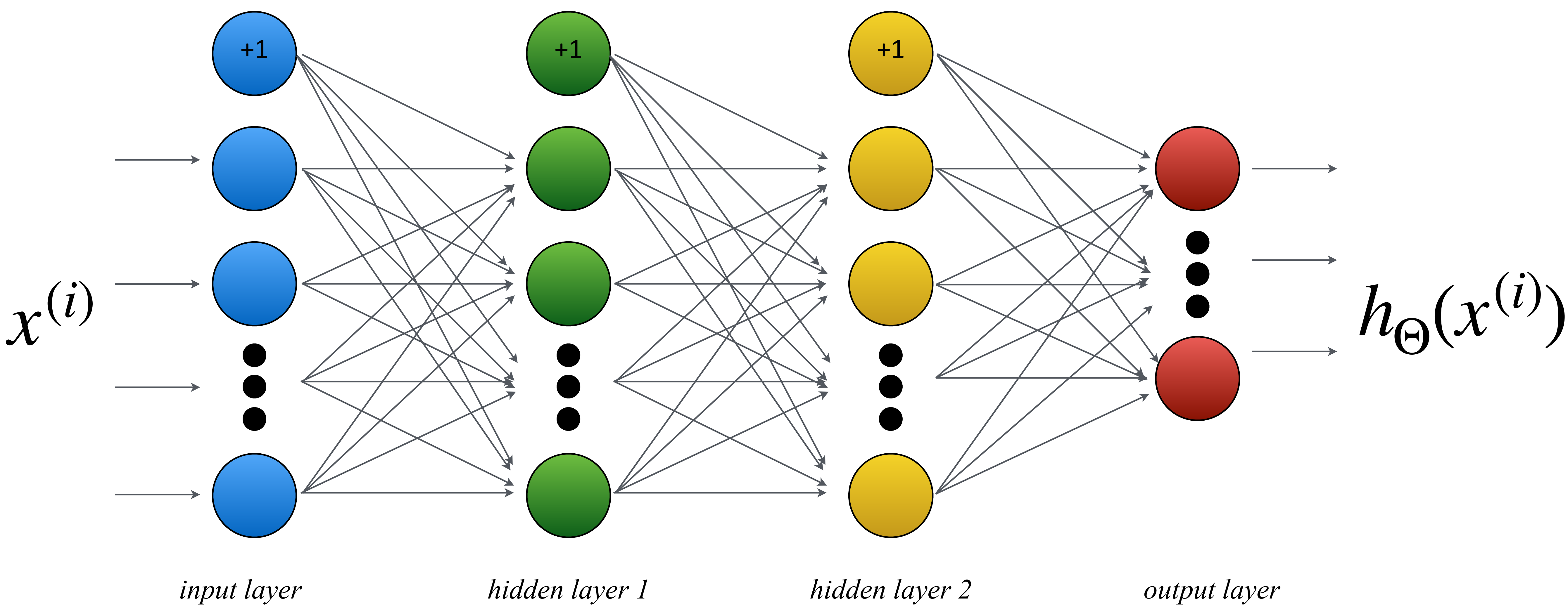
gewicht / belang dat neuron y aan
de input van neuron x geeft / hecht.

$$a_1^{(2)} = g(\Theta_{11}^{(1)} a_1^{(1)} + \Theta_{12}^{(1)} a_2^{(1)}$$



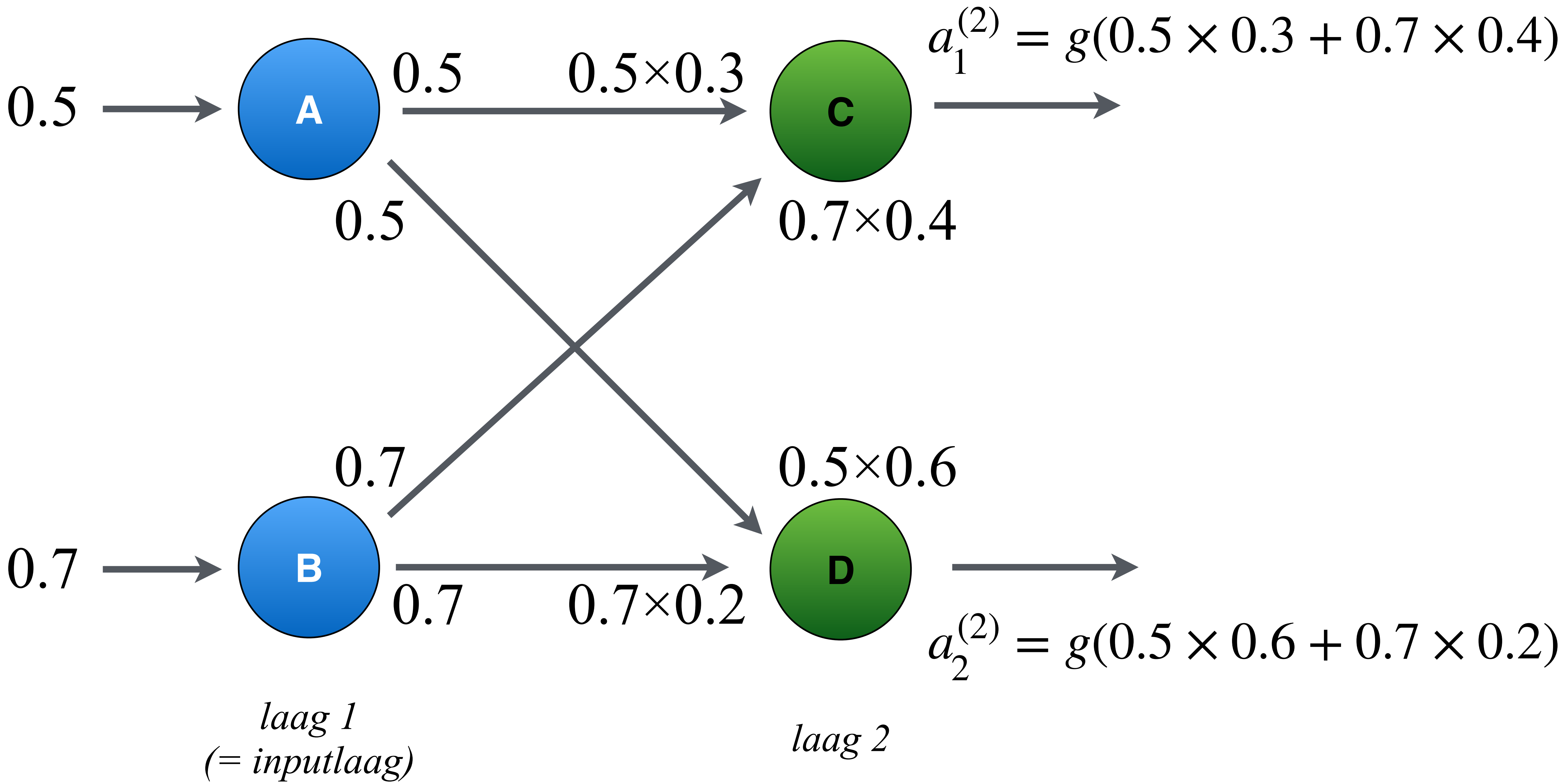
$$\Theta^{(1)} = \begin{bmatrix} CA & CB \\ DA & DB \end{bmatrix}$$

nn:forward propagation

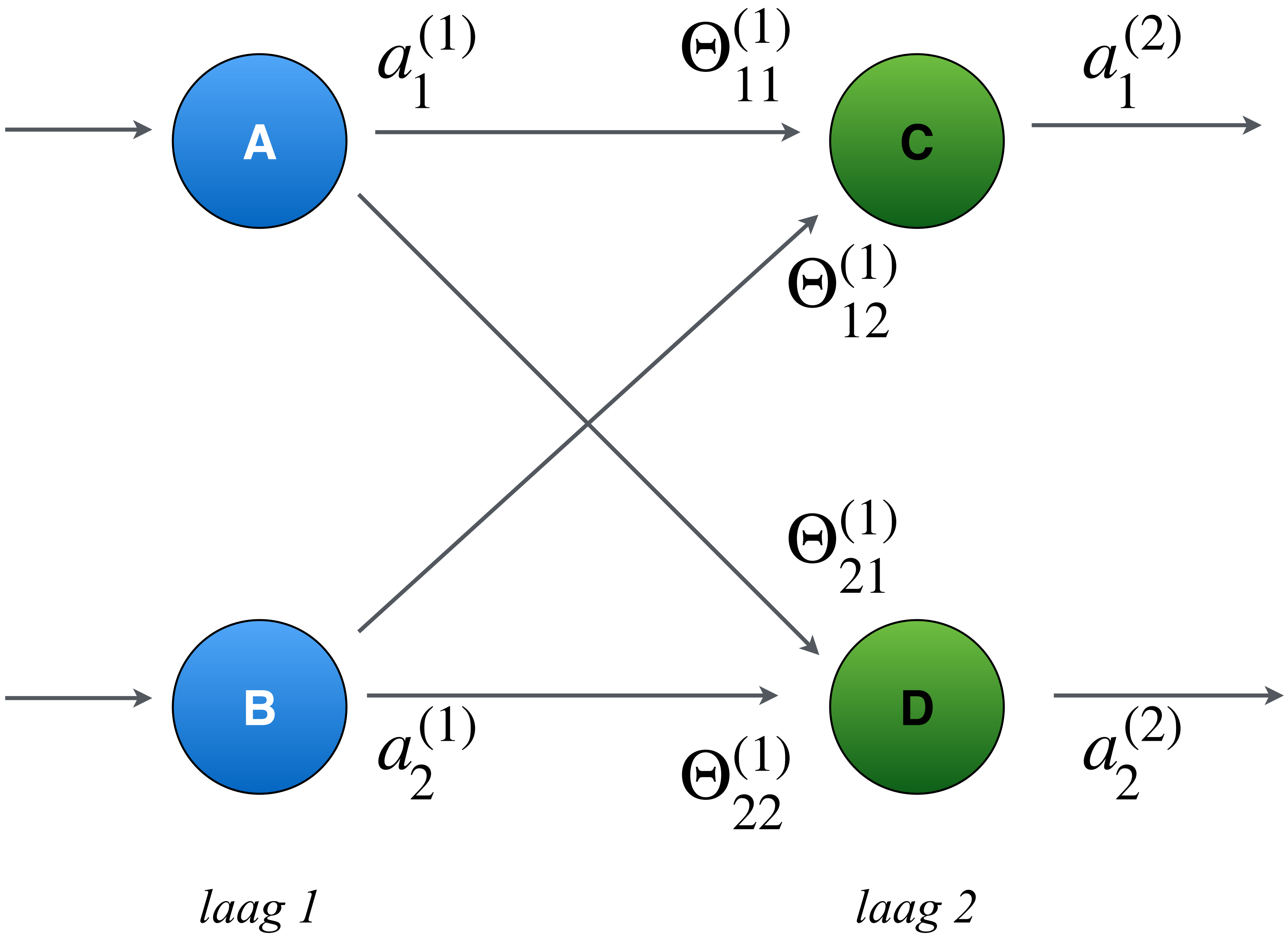


+1
+1
+1
$x^{(i)}$
$h_\Theta(x^{(i)})$
input layer
hidden layer 1
hidden layer 2
output layer

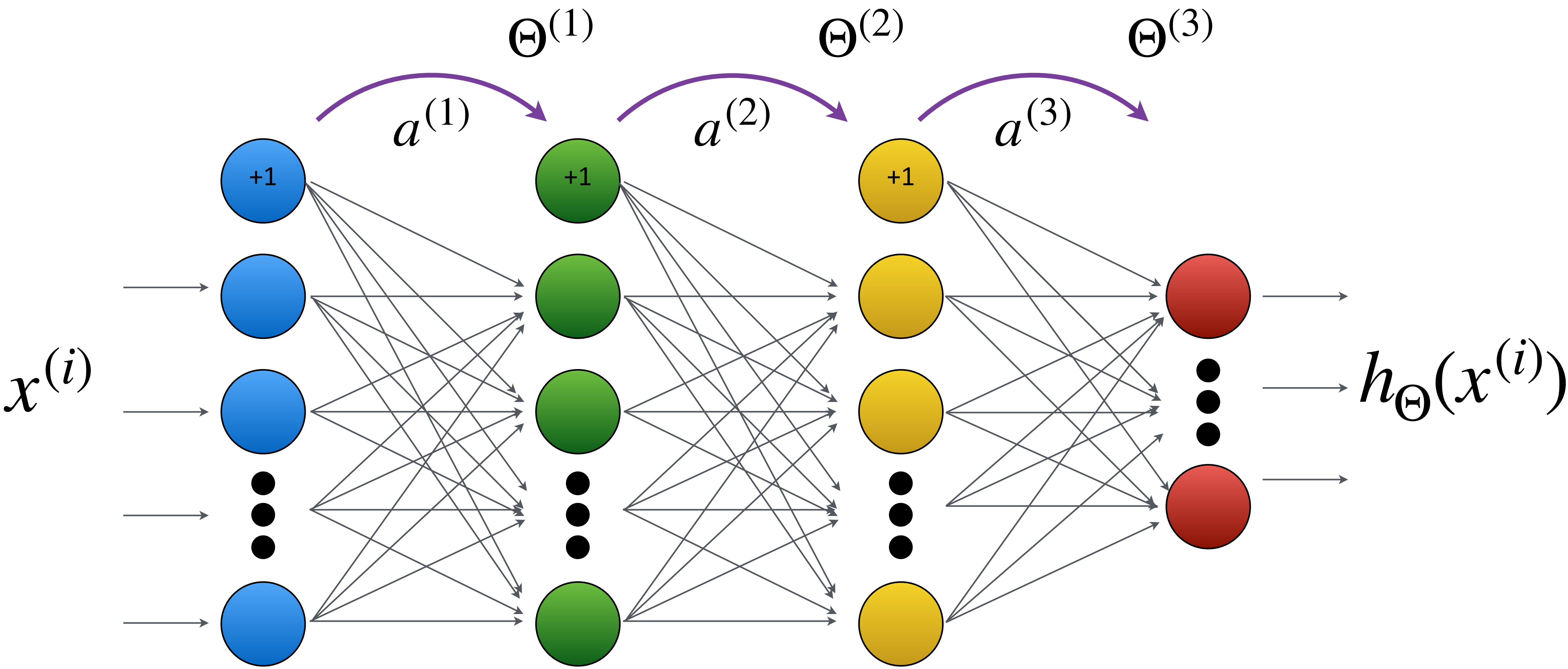
$$a^{(1)} = \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix} \qquad \Theta^{(1)} = \begin{bmatrix} 0.3 & 0.4 \\ 0.6 & 0.2 \end{bmatrix}$$



$$a_1^{(2)} = g(\Theta_{11}^{(1)} a_1^{(1)} + \Theta_{12}^{(1)} a_2^{(1)})$$



$$\Theta^{(1)} = \begin{bmatrix} CA & CB \\ DA & DB \end{bmatrix}$$



input layer

$$a^{(1)} = x$$
$$a_0^{(1)} = 1$$

hidden layer 1

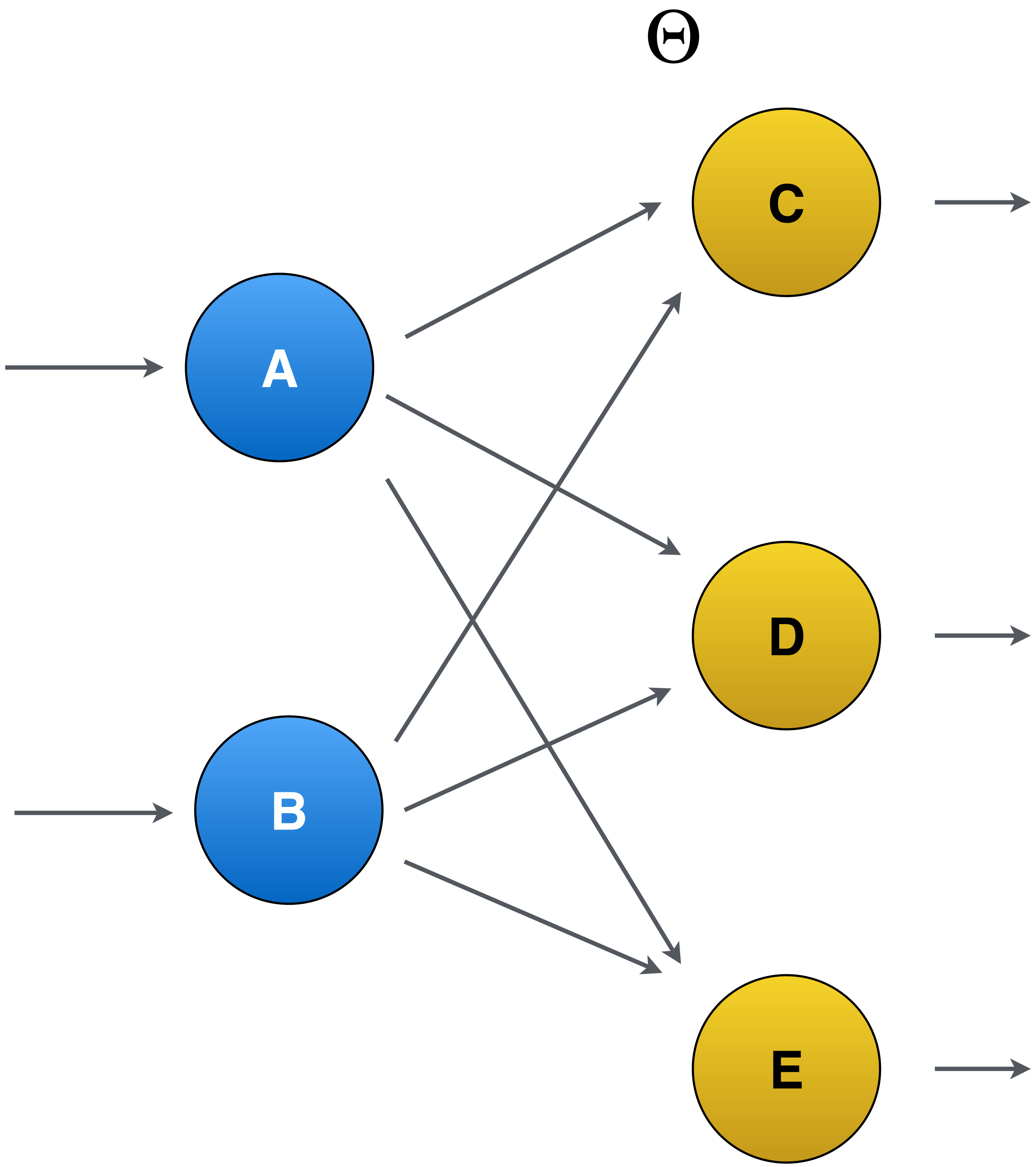
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$a_0^{(2)} = 1$$

hidden layer 2

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$a_0^{(3)} = 1$$

output layer

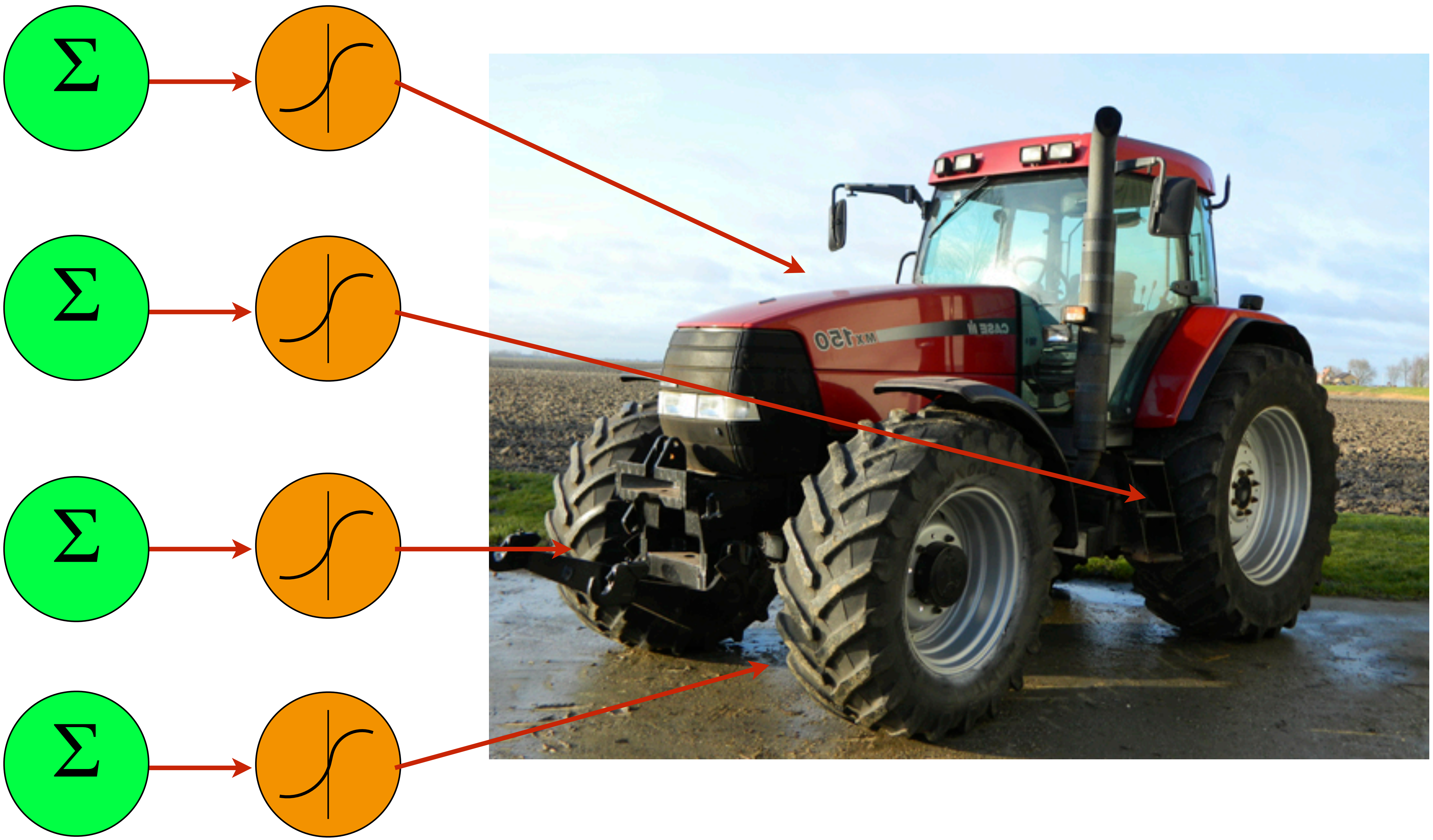
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = g(z^{(4)})$$
$$= h_\Theta(x^{(i)})$$
$$(= \hat{y}^{(i)})$$



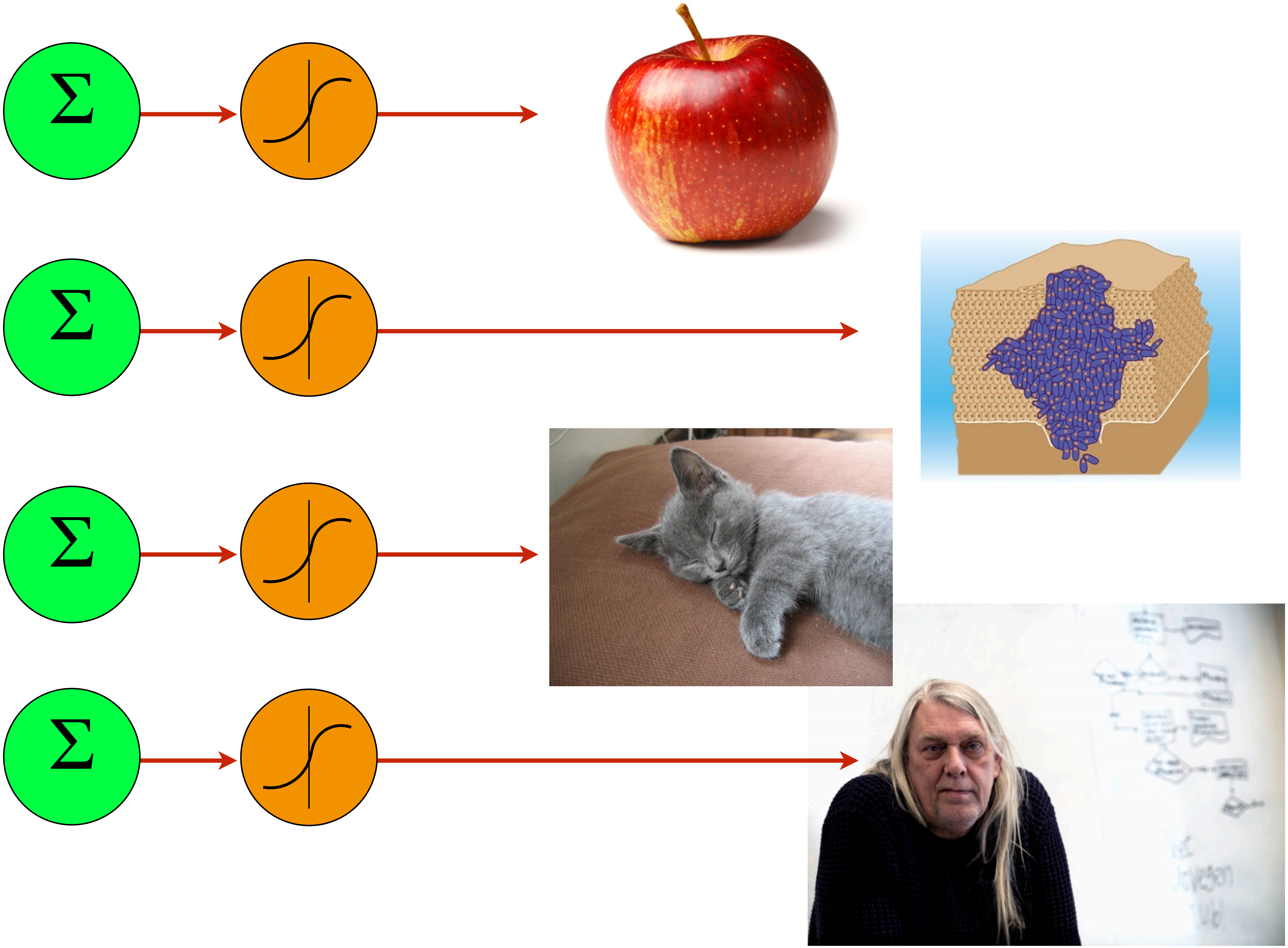
$$\Theta = \begin{bmatrix} CA & CB \\ DA & DB \\ EA & EB \end{bmatrix}$$

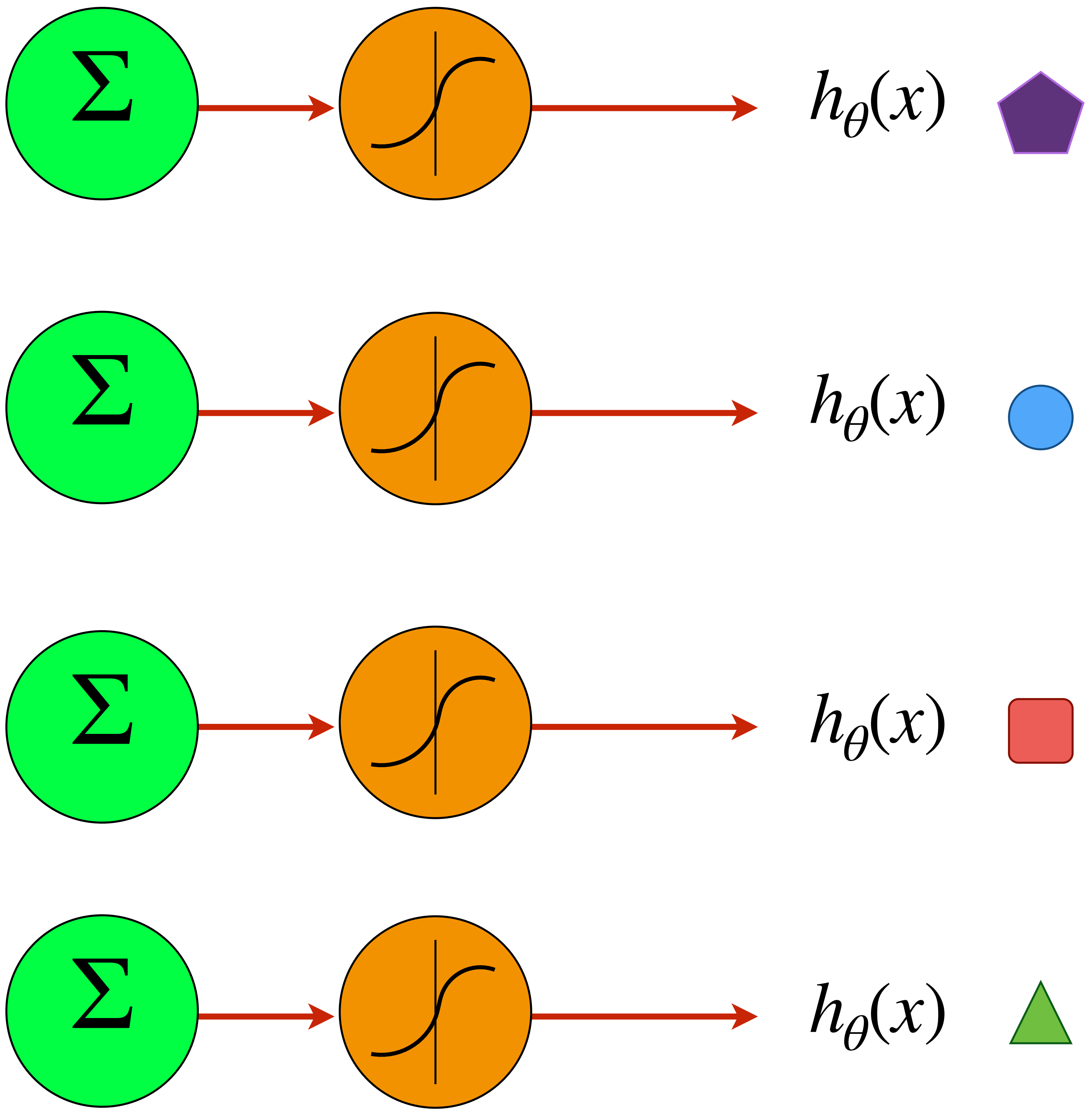
nn:cost function

train network for specific goal

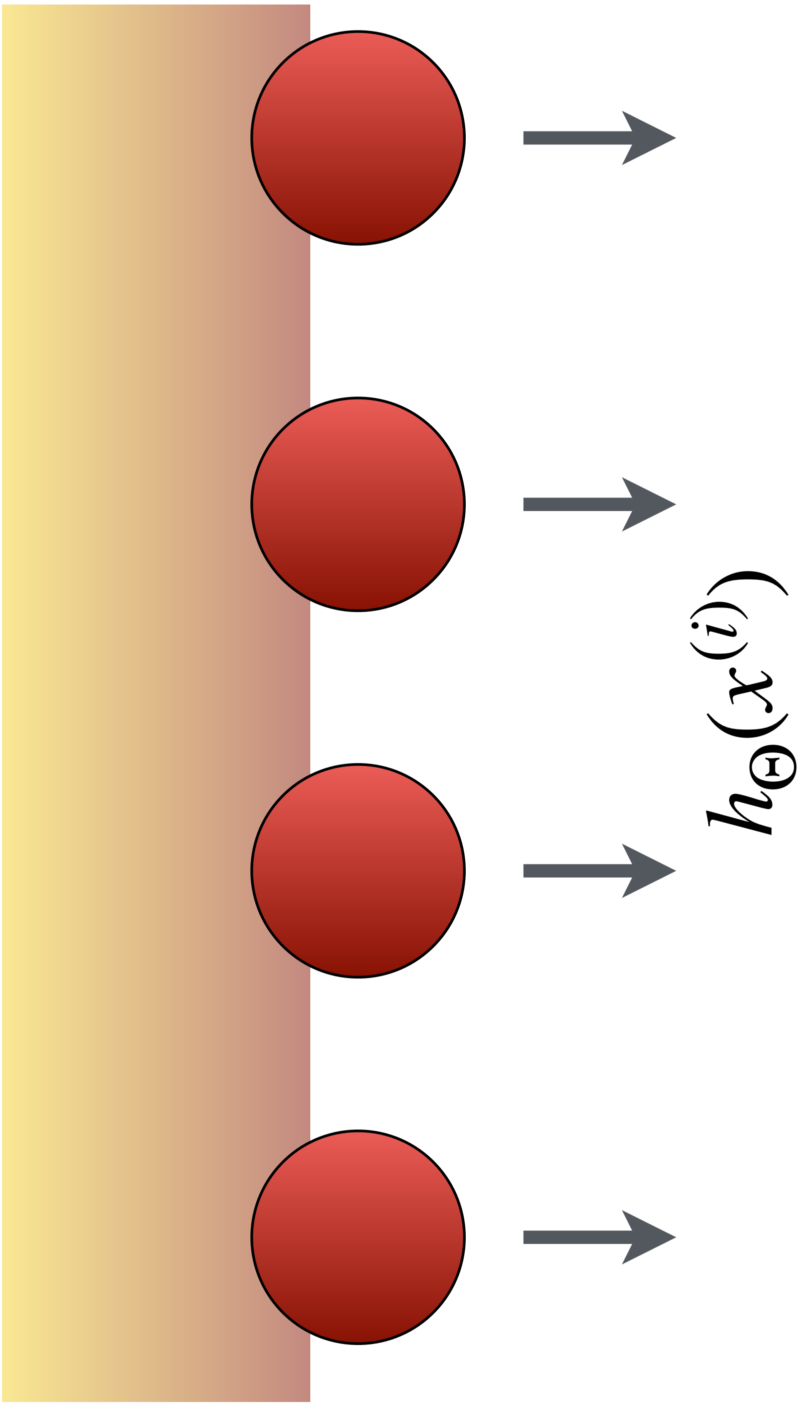


Multi-class classification: one versus all



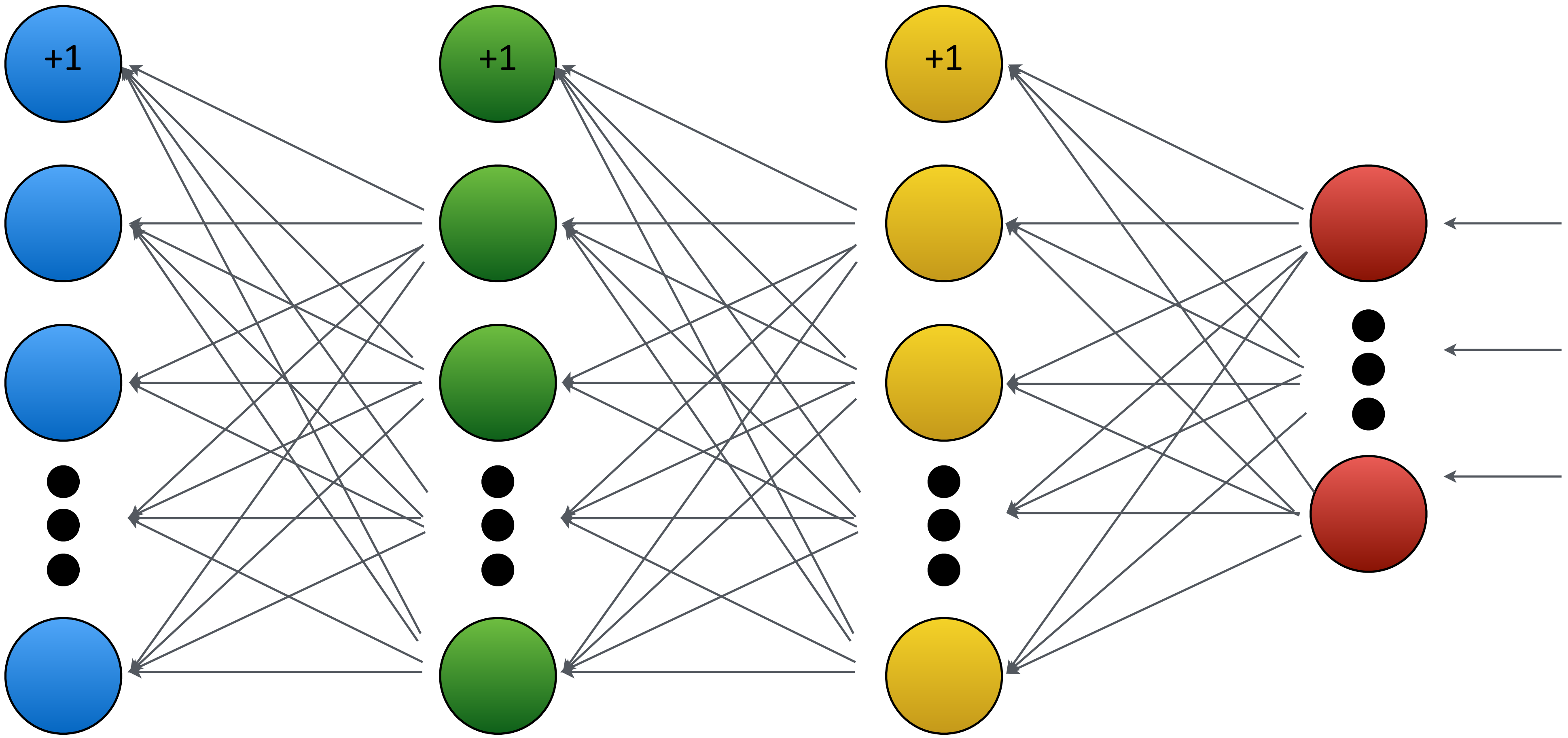


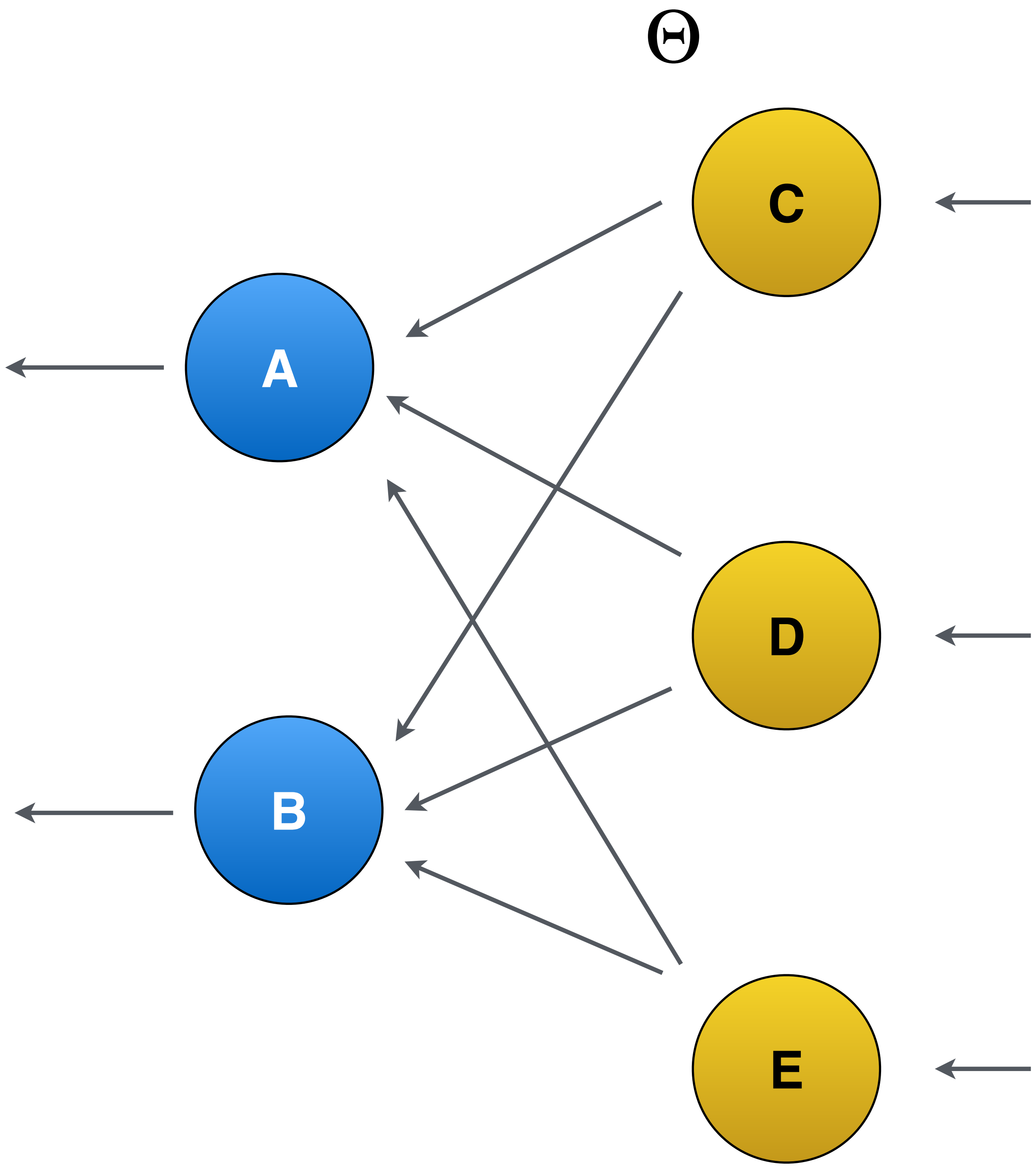
$$\begin{bmatrix} 0.213 \\ 0.423 \\ 0.786 \\ 0.143 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$


$h_\Theta(x^{(i)})$

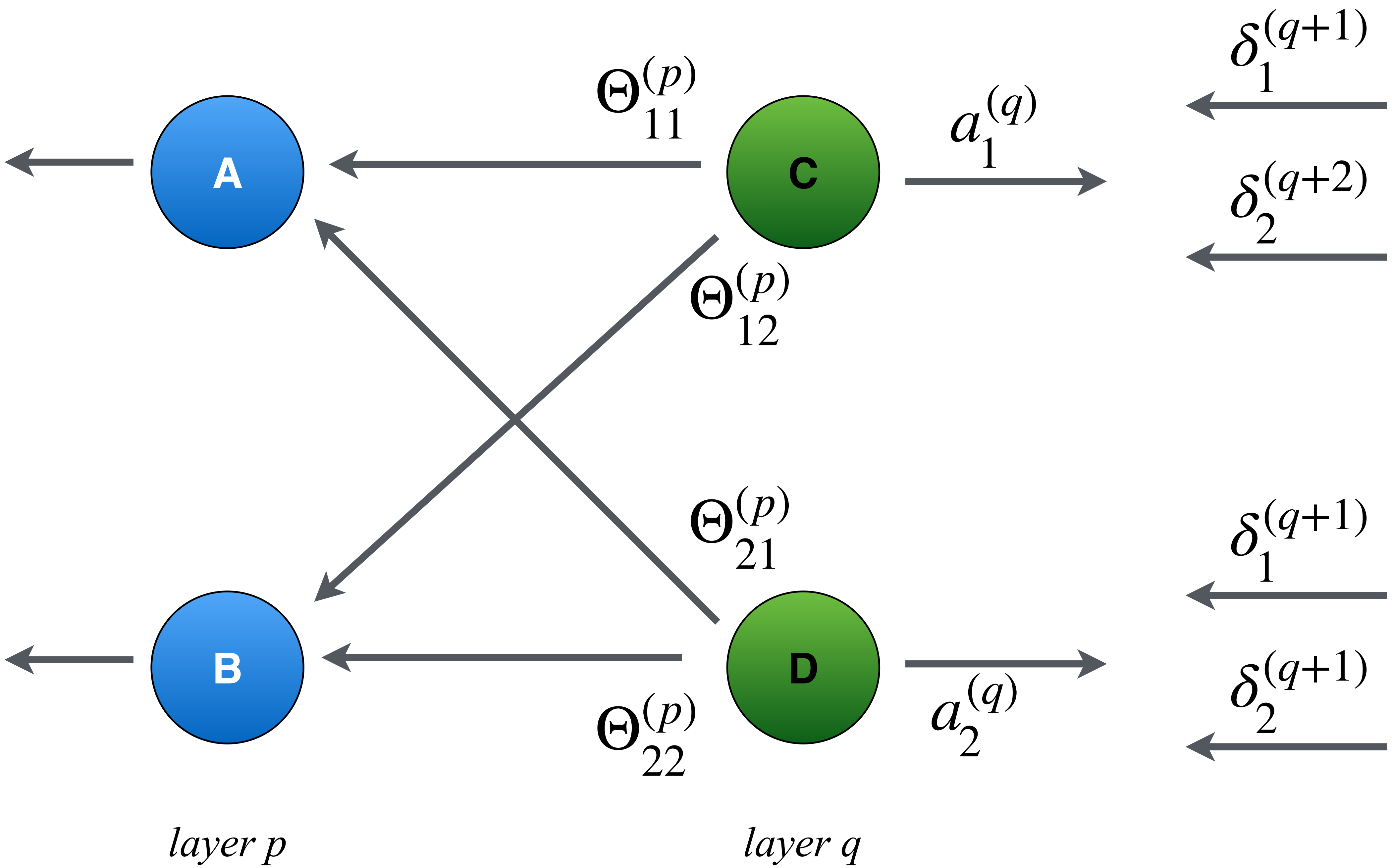
$$J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - h_\Theta(x^{(i)}))_k \right]$$

nn:backpropagation


+1
+1
+1



$$\Theta = \begin{bmatrix} CA & CB \\ DA & DB \\ EA & EB \end{bmatrix}$$



A
B
C
D
$\Theta_{11}^{(p)}$
$\Theta_{12}^{(p)}$
$\Theta_{21}^{(p)}$
$\Theta_{22}^{(p)}$
$a_1^{(q)}$
$a_2^{(q)}$
$\delta_1^{(q+1)}$
$\delta_2^{(q+2)}$
$\delta_1^{(q+1)}$
$\delta_2^{(q+1)}$
layer p
layer q

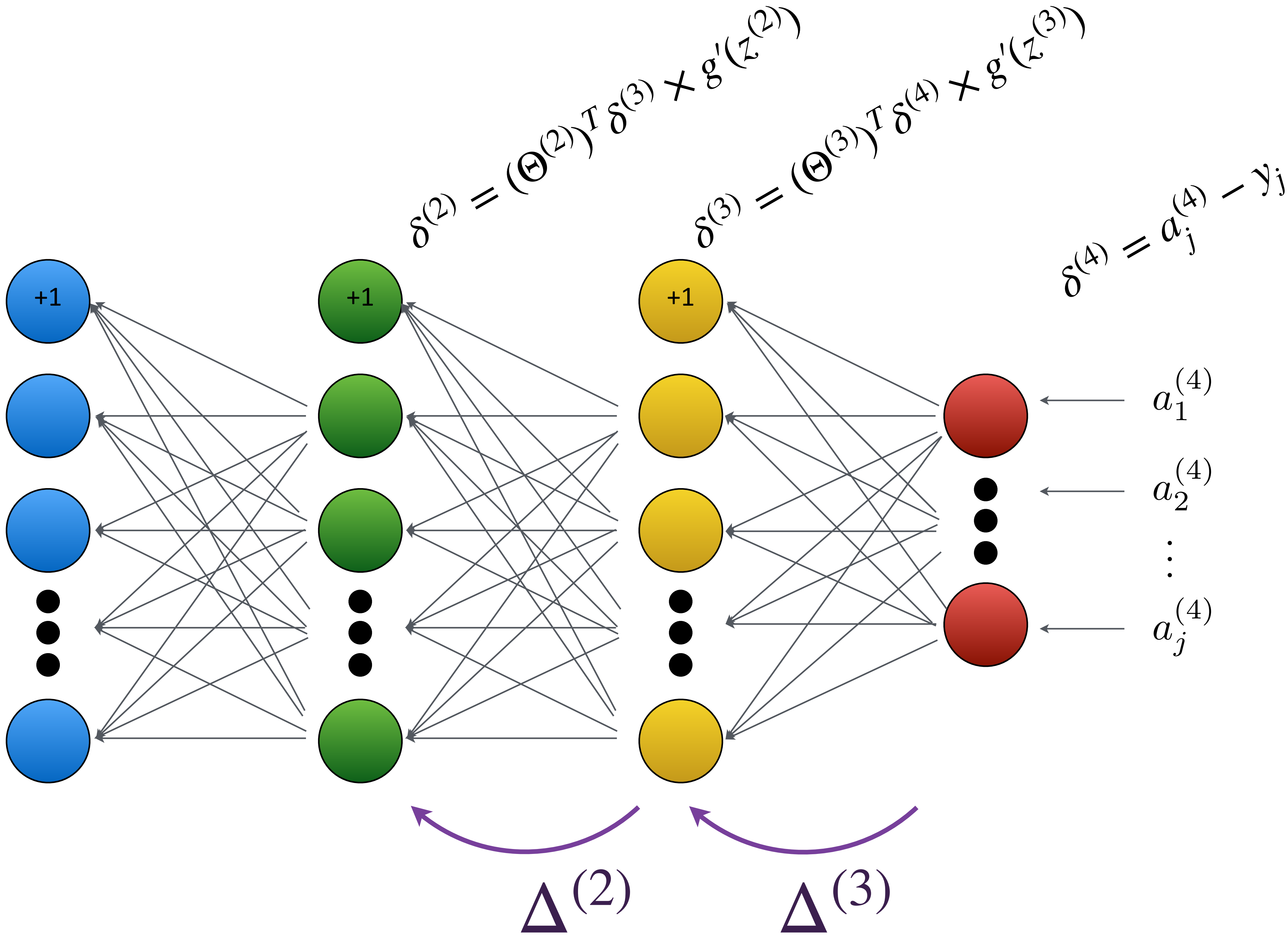
sigmoïdefunctie

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

$$\delta^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \times g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} \times g'(z^{(2)})$$



$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} \times g'(z^{(2)})$
$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \times g'(z^{(3)})$
$\delta^{(4)} = a_j^{(4)} - y_j$
+1
+1
+1
$a_1^{(4)}$
$a_2^{(4)}$
$\vdots$
$a_j^{(4)}$
$\Delta^{(2)}$
$\Delta^{(3)}$

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

backpropagation algorithme

Given a trainings set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})\}$

$\Delta_{ij}^{(l)} = 0$ for all i, j, l

repeat $i = 1 \ldots m$

set $a^{(1)} = x^{(i)}$

calculate $a^{(l)}$ voor $l = 2,3,\ldots,L$

calculate $\delta^{(L)} = a^{(L)} - y^{(L)}$ on basis of $y^{(i)}$

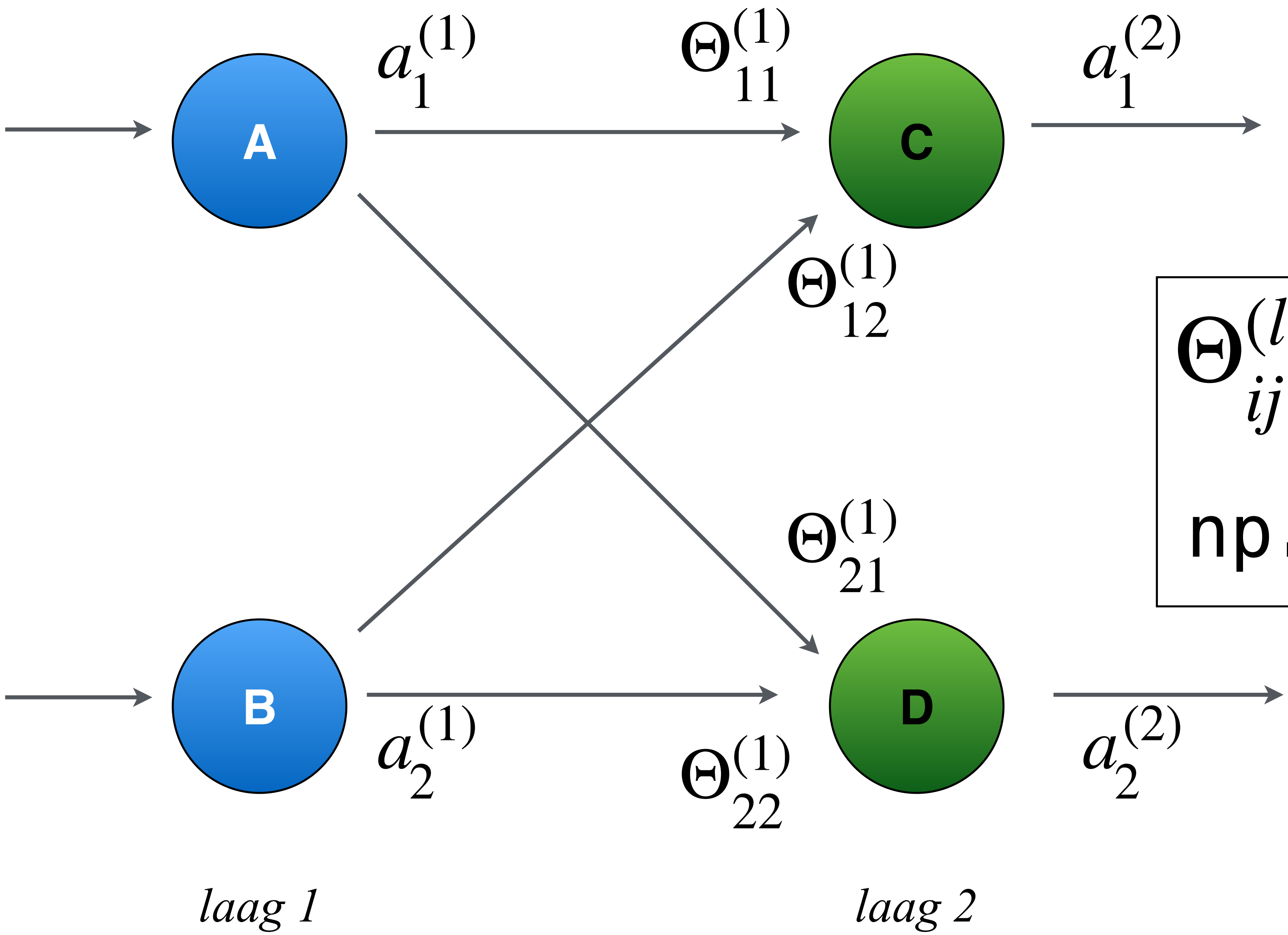
calculate $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$\Delta_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$$

nn::implementatiedetails

Initiële waarden van de Theta's (1/2)



$$\Theta_{ij}^{(l)} = 0$$

```
np.zeros((m,n))
```

Initiële waarden van de Theta's (2/2) *symmetry breaking*

$$\Theta_{ij}^{(l)} = random[-\epsilon, \epsilon]$$

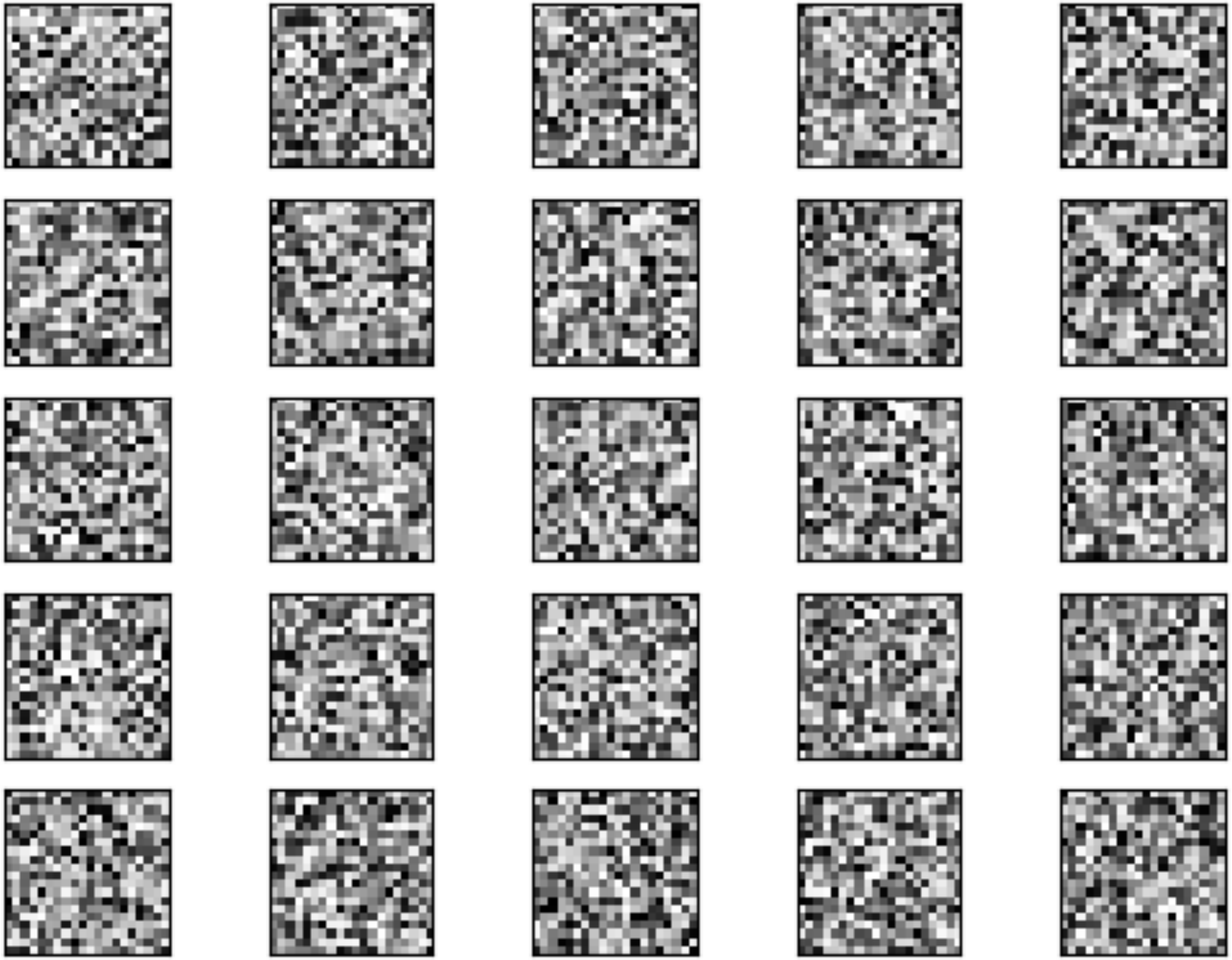
unrolling parameters

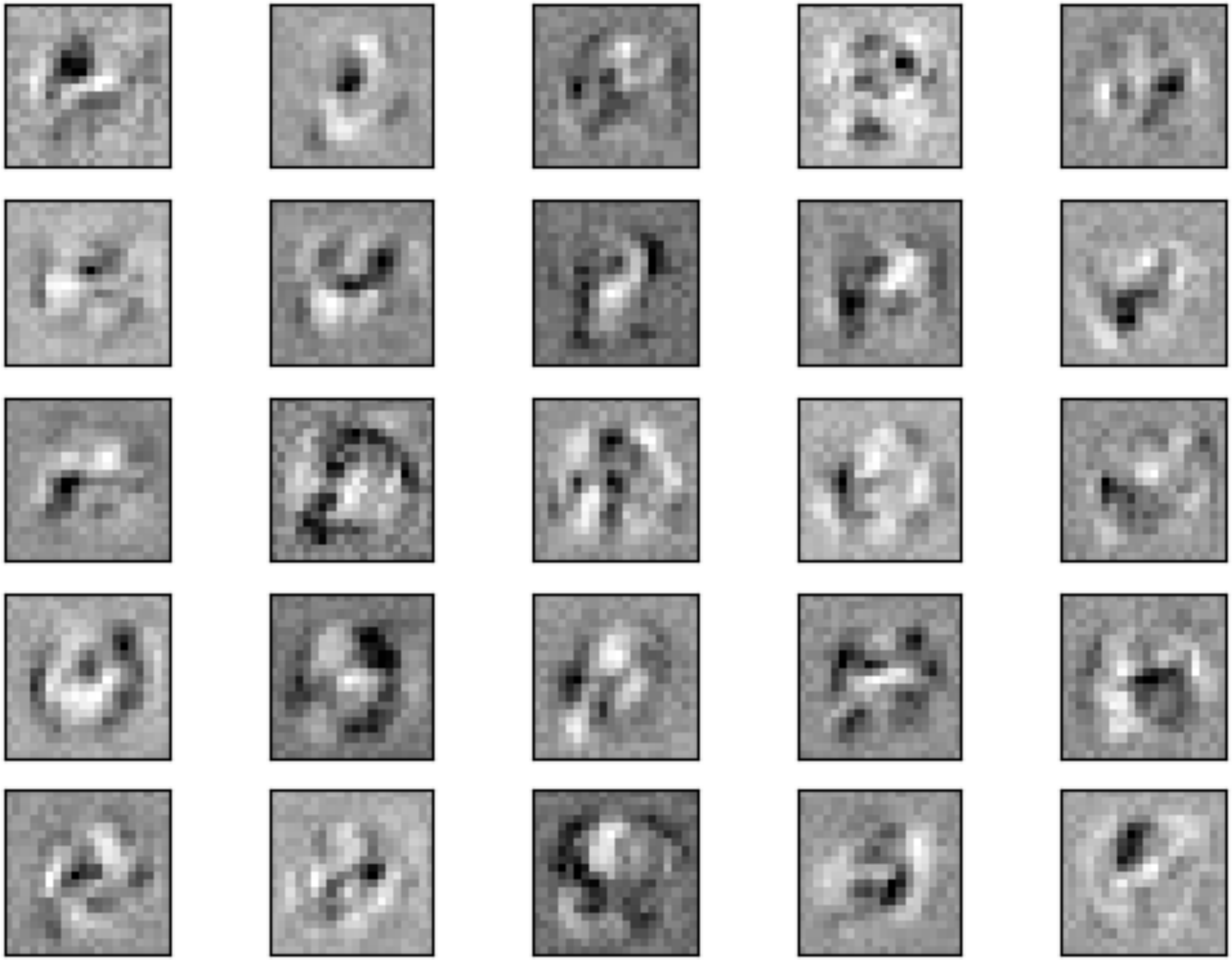
```
scipy.optimize.minimize()
```

```
>>> Theta1.shape
(25, 401)
>>> Theta2.shape
(10, 26)
>>> np.concatenate ( (Theta1.flatten(), Theta2.flatten()) ).shape
(10285,)
>>>
```

```
def nnCostFunction(Thetas, X, y):

    global input_size, hidden_size, num_labels
    size = hidden_size * (1+input_size)

    Theta1 = Thetas[:size].reshape(hidden_size, input_size+1)
    Theta2 = Thetas[size:].reshape(num_labels, hidden_layer_size+1)

    J = computeCost(Theta1, Theta2, X, y)
    grad1, grad2 = nnCheckGradients(Theta1, Theta2, X, y)

    return J, np.concatenate( (grad1.flatten(), grad2.flatten()) )
```

```
res = minimize(nnCostFunction, init_params, args=args,
   method='CG', callback=callbackF,
   options={'maxiter':30,'disp':True})
```






vpro