Práctica 4: entrenamiento de redes neuronales

one_hot y

```python
data = loadmat('ex4data1.mat')
y = data['y'].ravel() # (5000, 1) --> (5000,)
X = data['X']

m = len(y)
input_size = X.shape[1]
num_labels = 10
```

$$
\begin{array}{cccccccc}
 & 1 & & 2 & & 10 & & y \\
y = & \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} & , & \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} & , \ldots \text{ o} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} & & \\
 & 0 & & 1 & & 9 & & y - 1
\end{array}
$$

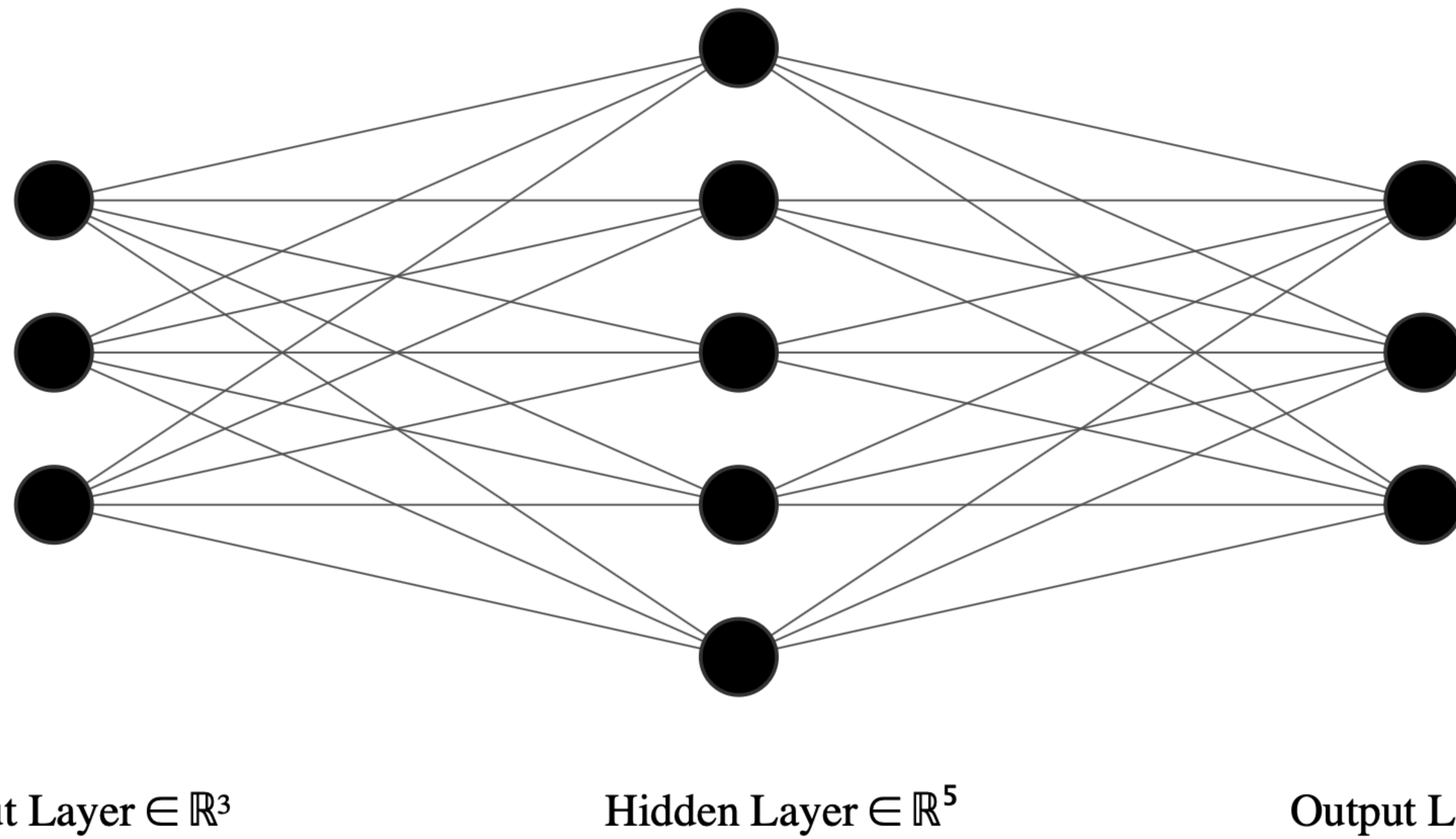
```python
y = (y - 1)
y_onehot = np.zeros((m, num_labels))  # 5000 x 10

for i in range(m):
    y_onehot[i][y[i]] = 1
```

Función de coste

```
input_layer_size = 3
hidden_layer_size = 5
num_labels = 3
m = 100
```

$$X : 100 \times 4$$
$$y : 100 \times 3$$

$$\Theta^{(1)} : 5 \times 4$$
$$\Theta^{(2)} : 3 \times 6$$



http://alexlenail.me/NN-SVG/

```
input_layer_size = 3
hidden_layer_size = 5
num_labels = 3
m = 100
```

$X : 100 \times 4$

$y : 100 \times 3$

$\Theta^{(1)} : 5 \times 4$

$\Theta^{(2)} : 3 \times 6$

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ & \vdots & & \\ 1 & x_1^{(100)} & x_2^{(100)} & x_3^{(100)} \end{pmatrix} \quad y = \begin{pmatrix} y_1^{(1)} & y_2^{(1)} & y_3^{(1)} \\ y_1^{(2)} & y_2^{(2)} & y_3^{(2)} \\ & \vdots & \\ y_1^{(100)} & y_2^{(100)} & y_3^{(100)} \end{pmatrix}$$

$$\Theta^{(1)} = \begin{pmatrix} \Theta_{1,0}^{(1)} & \Theta_{1,1}^{(1)} & \Theta_{1,2}^{(1)} & \Theta_{1,3}^{(1)} \\ \Theta_{2,0}^{(1)} & \Theta_{2,1}^{(1)} & \Theta_{2,2}^{(1)} & \Theta_{2,3}^{(1)} \\ \Theta_{3,0}^{(1)} & \Theta_{3,1}^{(1)} & \Theta_{3,2}^{(1)} & \Theta_{3,3}^{(1)} \\ \Theta_{4,0}^{(1)} & \Theta_{4,1}^{(1)} & \Theta_{4,2}^{(1)} & \Theta_{4,3}^{(1)} \\ \Theta_{5,0}^{(1)} & \Theta_{5,1}^{(1)} & \Theta_{5,2}^{(1)} & \Theta_{5,3}^{(1)} \end{pmatrix} \Theta^{(2)} = \begin{pmatrix} \Theta_{1,0}^{(2)} & \Theta_{1,1}^{(2)} & \Theta_{1,2}^{(2)} & \Theta_{1,3}^{(2)} & \Theta_{1,4}^{(2)} & \Theta_{1,5}^{(2)} \\ \Theta_{2,0}^{(2)} & \Theta_{2,1}^{(2)} & \Theta_{2,2}^{(2)} & \Theta_{2,3}^{(2)} & \Theta_{2,4}^{(2)} & \Theta_{2,5}^{(2)} \\ \Theta_{3,0}^{(2)} & \Theta_{3,1}^{(2)} & \Theta_{3,2}^{(2)} & \Theta_{3,3}^{(2)} & \Theta_{3,4}^{(2)} & \Theta_{3,5}^{(2)} \end{pmatrix}$$

Para cada ejemplo de entrenamiento $(x^{(i)}, y^{(i)})$

```
input_layer_size = 3
hidden_layer_size = 5
num_labels = 3
m = 100
```

$X : 5 \times 4$

$y : 5 \times 3$

$\Theta^{(1)} : 5 \times 4$

$\Theta^{(2)} : 3 \times 6$

forward:

$$a^{(1)} = x^{(1)} : (4,1)$$

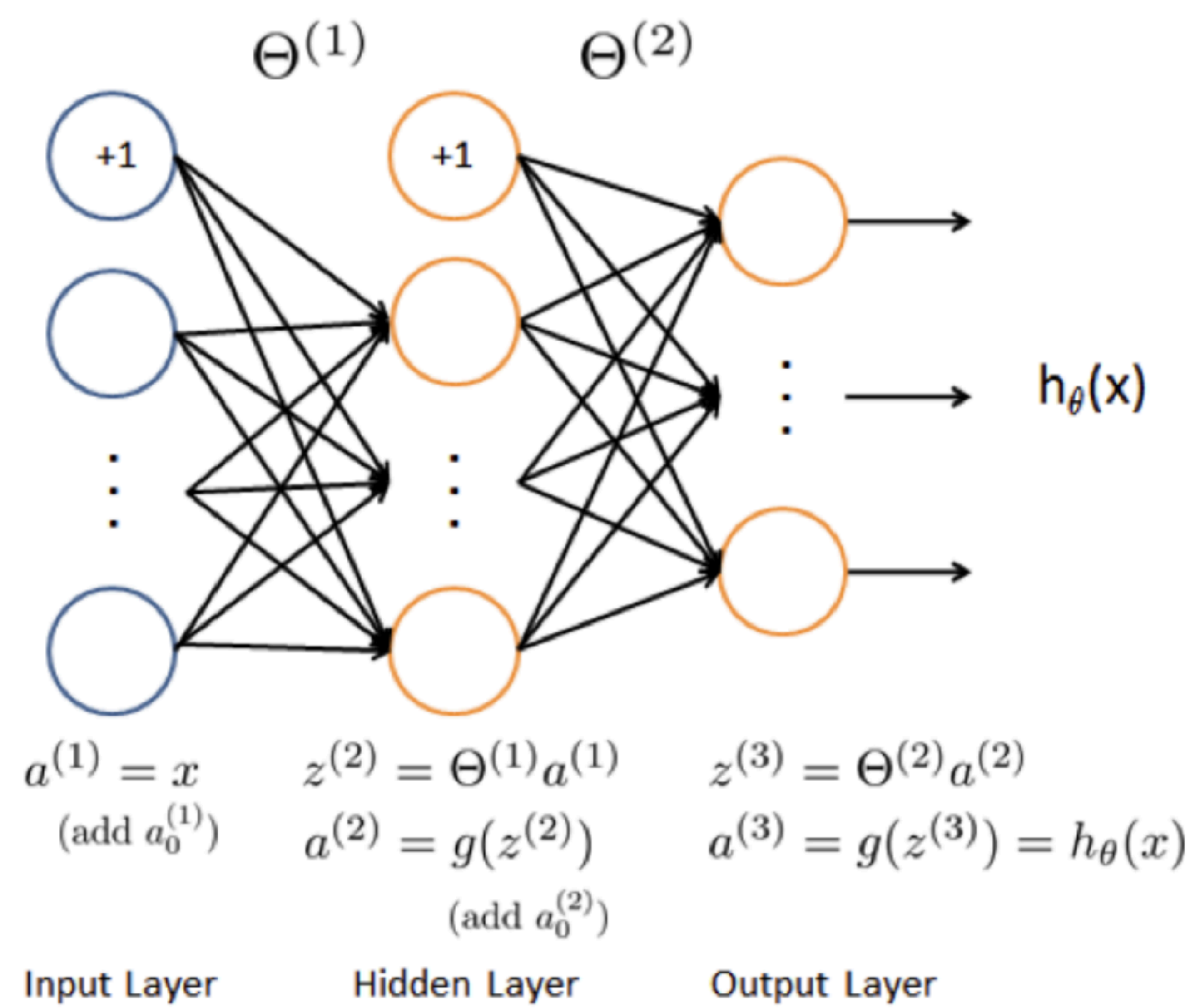
$$z^{(2)} = \Theta^{(1)} a^{(1)} : (5,1)$$

$$a^{(2)} = g(z^{(2)}) : (5,1)$$

$$\text{add } a_0^{(2)} \rightarrow a^{(2)} : (6,1)$$

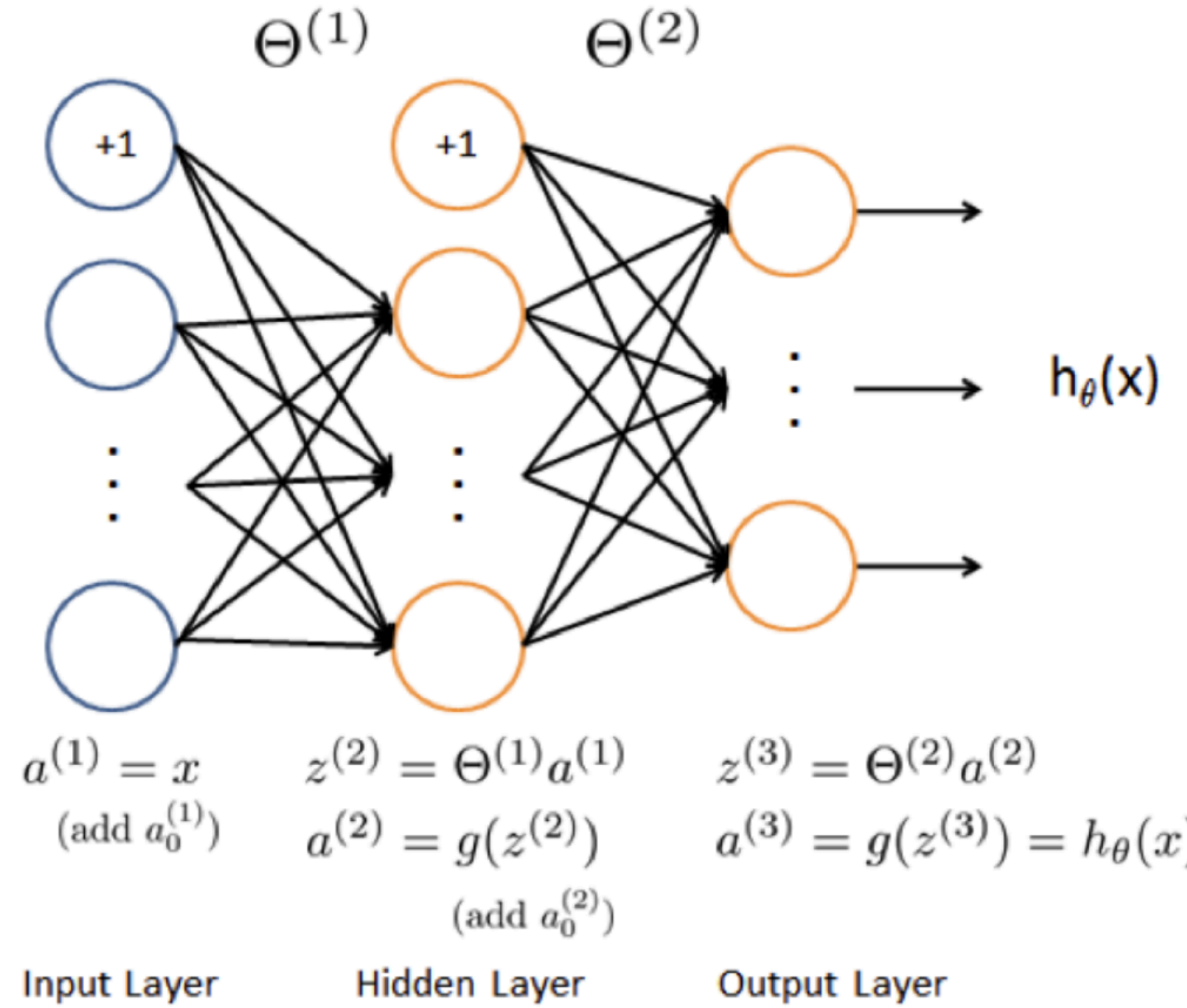
$$z^{(3)} = \Theta^{(2)} a^{(2)} : (3,1)$$

$$a^{(3)} = g(z^{(3)}) : (3,1)$$



Implementación vectorizada

$$X = \begin{bmatrix} — & x^{(1)} & — \\ — & x^{(2)} & — \\ & \vdots & \\ — & x^{(100)} & — \end{bmatrix}$$

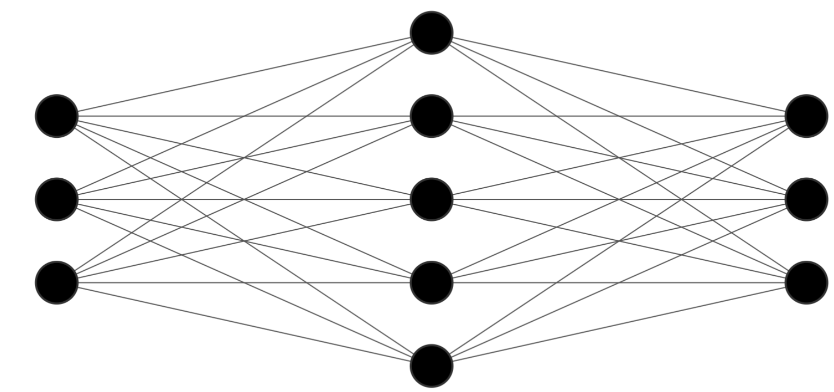


$a^{(i)(j)}$: output of layer i,
$a^{(i)}$, for input $x^{(j)}$, $j \in 1..m$

$$A^{(1)} = \begin{bmatrix} 1 & — & x^{(1)} & — \\ 1 & — & x^{(2)} & — \\ & \vdots & & \\ 1 & — & x^{(100)} & — \end{bmatrix} = \begin{bmatrix} — & a^{(1)(1)} & — \\ — & a^{(1)(2)} & — \\ & \vdots & \\ — & a^{(1)(100)} & — \end{bmatrix}$$

Implementación vectorizada

$$\Theta^{(1)} = \begin{bmatrix} \Theta^{(1)}_{1,0} & \Theta^{(1)}_{1,1} & \Theta^{(1)}_{1,2} & \Theta^{(1)}_{1,3} \\ \Theta^{(1)}_{2,0} & \Theta^{(1)}_{2,1} & \Theta^{(1)}_{2,2} & \Theta^{(1)}_{2,3} \\ \Theta^{(1)}_{3,0} & \Theta^{(1)}_{3,1} & \Theta^{(1)}_{3,2} & \Theta^{(1)}_{3,3} \\ \Theta^{(1)}_{4,0} & \Theta^{(1)}_{4,1} & \Theta^{(1)}_{4,2} & \Theta^{(1)}_{4,3} \\ \Theta^{(1)}_{5,0} & \Theta^{(1)}_{5,1} & \Theta^{(1)}_{5,2} & \Theta^{(1)}_{5,3} \end{bmatrix} = \begin{bmatrix} — & \Theta^{(1)}_{1} & — \\ — & \Theta^{(1)}_{2} & — \\ & \vdots & \\ — & \Theta^{(1)}_{5} & — \end{bmatrix}$$



$$A^{(1)} = \begin{bmatrix} — & a^{(1)(1)} & — \\ — & a^{(1)(2)} & — \\ & \vdots & \\ — & a^{(1)(100)} & — \end{bmatrix}$$

100×4

$$(\Theta^{(1)})^T = \begin{bmatrix} | & | & & | \\ \Theta^{(1)}_{1} & \Theta^{(1)}_{2} & \dots & \Theta^{(1)}_{5} \\ | & | & & | \end{bmatrix}$$

4×5

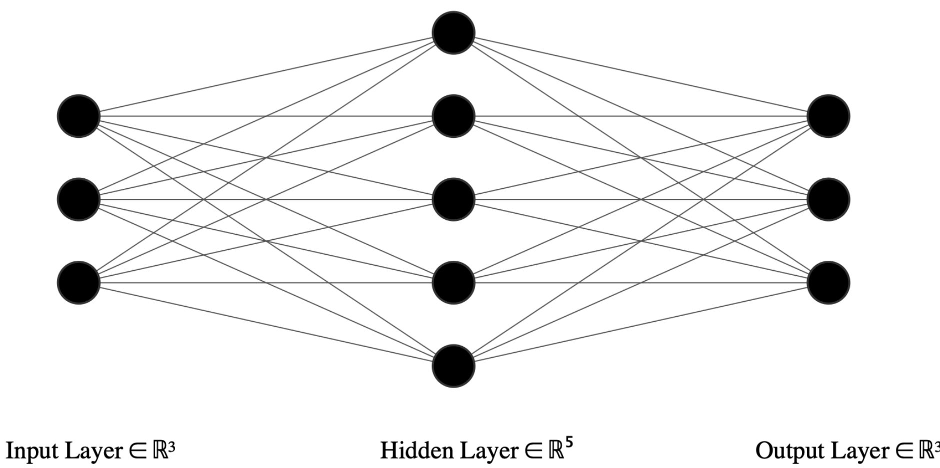
$$A^{(1)}(\Theta^{(1)})^T = \begin{bmatrix} a^{(1)(1)}\Theta^{(1)}_{1} & a^{(1)(1)}\Theta^{(1)}_{2} & \dots & a^{(1)(1)}\Theta^{(1)}_{5} \\ a^{(1)(2)}\Theta^{(1)}_{1} & a^{(1)(2)}\Theta^{(1)}_{2} & \dots & a^{(1)(2)}\Theta^{(1)}_{5} \\ & \vdots & & \\ a^{(1)(100)}\Theta^{(1)}_{1} & a^{(1)(100)}\Theta^{(1)}_{2} & \dots & a^{(1)(100)}\Theta^{(1)}_{5} \end{bmatrix}$$

100×5

Implementación vectorizada

$$A^{(1)}(\Theta^{(1)})^T = \begin{bmatrix} a^{(1)(1)}\Theta_1^{(1)} & a^{(1)(1)}\Theta_2^{(1)} & \dots & a^{(1)(1)}\Theta_5^{(1)} \\ a^{(1)(2)}\Theta_1^{(1)} & a^{(1)(2)}\Theta_2^{(1)} & \dots & a^{(1)(2)}\Theta_5^{(1)} \\ & \vdots & & \\ a^{(1)(100)}\Theta_1^{(1)} & a^{(1)(100)}\Theta_2^{(1)} & \dots & a^{(1)(100)}\Theta_5^{(1)} \end{bmatrix}$$

100×5

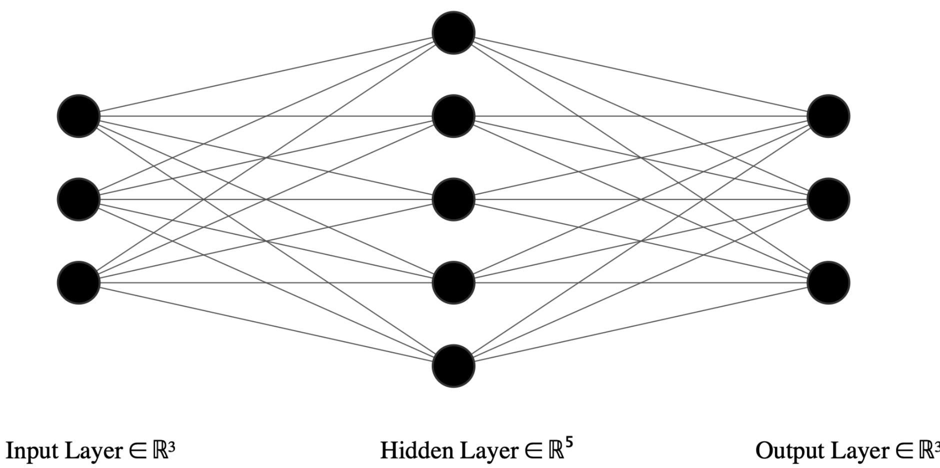


$$g(A^{(1)}(\Theta^{(1)})^T) = \begin{bmatrix} a_1^{(2)(1)} & a_2^{(2)(1)} & \dots & a_5^{(2)(1)} \\ a_1^{(2)(2)} & a_2^{(2)(2)} & \dots & a_5^{(2)(2)} \\ & \vdots & & \\ a_1^{(2)(100)} & a_2^{(2)(100)} & \dots & a_5^{(2)(100)} \end{bmatrix} \quad 100 \times 5$$

$$A^{(2)} = \begin{bmatrix} 1 & a_1^{(2)(1)} & a_2^{(2)(1)} & \dots & a_5^{(2)(1)} \\ 1 & a_1^{(2)(2)} & a_2^{(2)(2)} & \dots & a_5^{(2)(2)} \\ & & \vdots & & \\ 1 & a_1^{(2)(100)} & a_2^{(2)(100)} & \dots & a_5^{(2)(100)} \end{bmatrix} = \begin{bmatrix} — & a^{(2)(1)} & — \\ — & a^{(2)(2)} & — \\ & \vdots & \\ — & a^{(2)(100)} & — \end{bmatrix} \quad 100 \times 6$$

Implementación vectorizada

$$A^{(2)} = \begin{bmatrix} — & a^{(2)(1)} & — \\ — & a^{(2)(2)} & — \\ & \vdots & \\ — & a^{(2)(100)} & — \end{bmatrix} \quad 100 \times 6$$



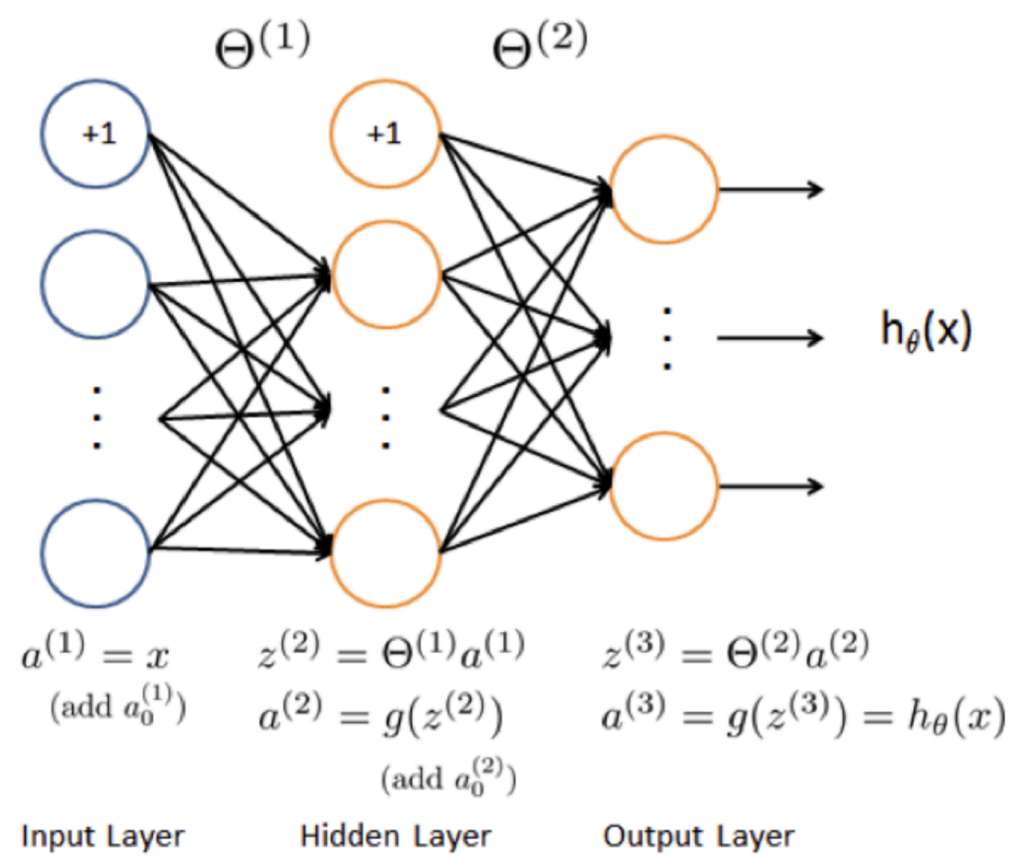
$$\Theta^{(2)} = \begin{bmatrix} \Theta^{(2)}_{1,0} & \Theta^{(2)}_{1,1} & \Theta^{(2)}_{1,2} & \Theta^{(2)}_{1,3} & \Theta^{(2)}_{1,4} & \Theta^{(2)}_{1,5} \\ \Theta^{(2)}_{2,0} & \Theta^{(2)}_{2,1} & \Theta^{(2)}_{2,2} & \Theta^{(2)}_{2,3} & \Theta^{(2)}_{2,4} & \Theta^{(2)}_{2,5} \\ \Theta^{(2)}_{3,0} & \Theta^{(2)}_{3,1} & \Theta^{(2)}_{3,2} & \Theta^{(2)}_{3,3} & \Theta^{(2)}_{3,4} & \Theta^{(2)}_{3,5} \end{bmatrix} = \begin{bmatrix} — & \Theta^{(2)}_1 & — \\ — & \Theta^{(2)}_2 & — \\ — & \Theta^{(2)}_3 & — \end{bmatrix}$$

$$(\Theta^{(2)})^T = \begin{bmatrix} | & | & | \\ \Theta^{(2)}_1 & \Theta^{(2)}_2 & \Theta^{(2)}_3 \\ | & | & | \end{bmatrix} \quad 6 \times 3$$

$$g(A^{(2)}(\Theta^{(2)})^T) = \begin{bmatrix} a_1^{(3)(1)} & a_2^{(3)(1)} & a_3^{(3)(1)} \\ a_1^{(3)(2)} & a_2^{(3)(2)} & a_3^{(3)(2)} \\ & \vdots & \\ a_1^{(3)(100)} & a_2^{(3)(100)} & a_3^{(3)(100)} \end{bmatrix} = H \qquad 100 \times 3$$

Implementación vectorizada

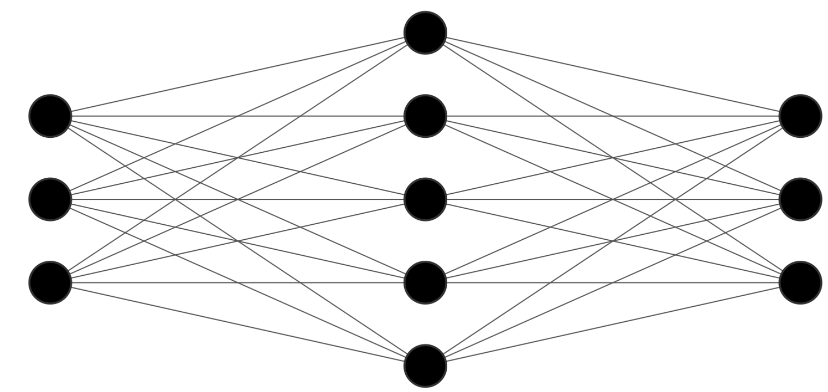
```python
def forward_propagate(X, Theta1, Theta2):
    m = X.shape[0]

    A1 = np.hstack([np.ones([m, 1]), X])
    Z2 = np.dot(A1, Theta1.T)
    A2 = np.hstack([np.ones([m, 1]), sigmoid(Z2)])
    Z3 = np.dot(A2, Theta2.T)
    H = sigmoid(Z3)

    return A1, A2, H
```



Cálculo del coste

$$Y = \begin{bmatrix} y_1^{(1)} & y_2^{(1)} & y_3^{(1)} \\ y_1^{(2)} & y_2^{(2)} & y_3^{(2)} \\ & \vdots & \\ y_1^{(100)} & y_2^{(100)} & y_3^{(100)} \end{bmatrix}$$

100×3

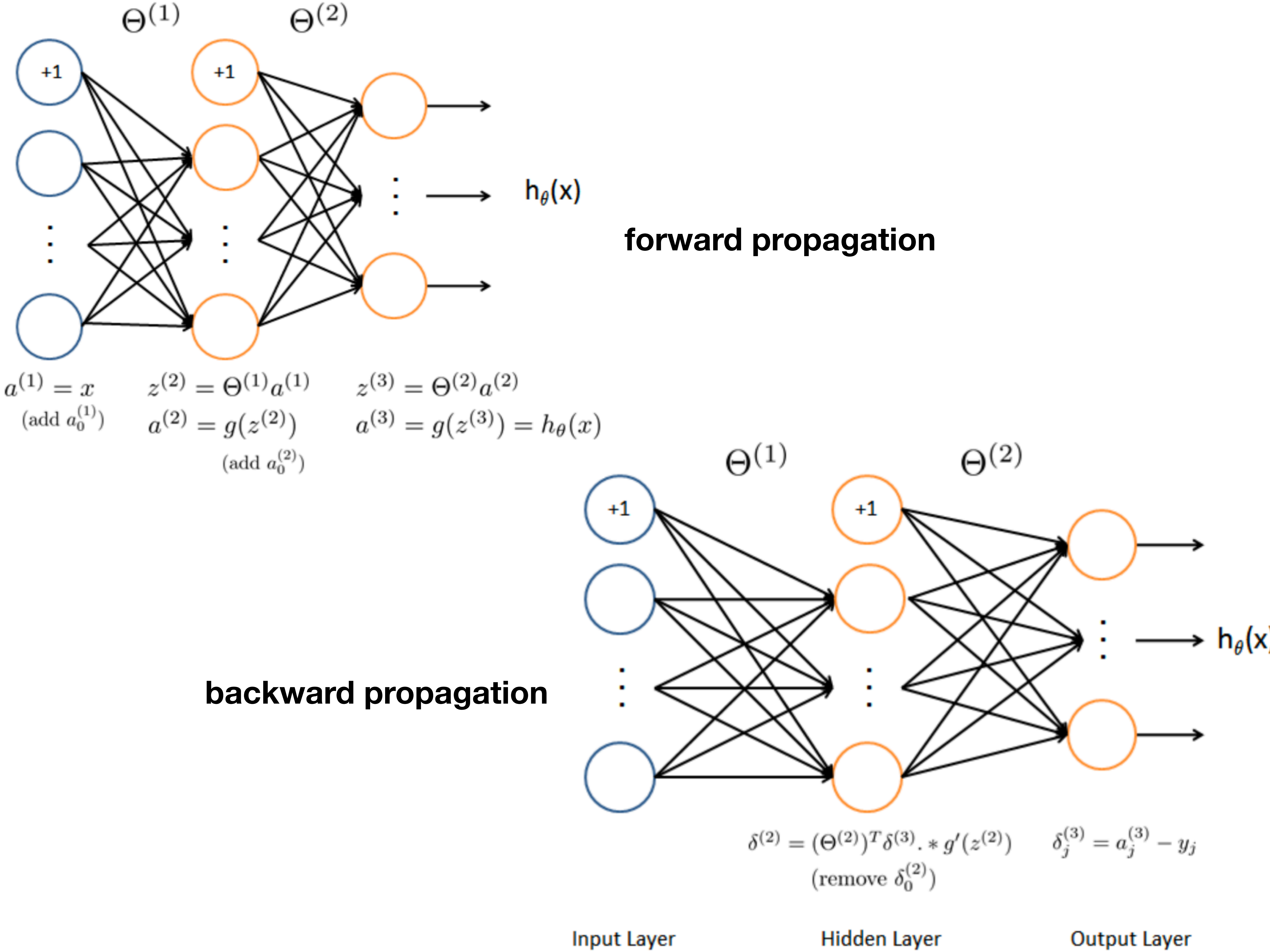


$$\begin{bmatrix} a_1^{(3)(1)} & a_2^{(3)(1)} & a_3^{(3)(1)} \\ a_1^{(3)(2)} & a_2^{(3)(2)} & a_3^{(3)(2)} \\ & \vdots & \\ a_1^{(3)(100)} & a_2^{(3)(100)} & a_3^{(3)(100)} \end{bmatrix} = H = \begin{bmatrix} (h_\theta(x^{(1)}))_1 & (h_\theta(x^{(1)}))_2 & (h_\theta(x^{(1)}))_3 \\ (h_\theta(x^{(2)}))_1 & (h_\theta(x^{(2)}))_2 & (h_\theta(x^{(2)}))_3 \\ & \vdots & \\ (h_\theta(x^{(100)}))_1 & (h_\theta(x^{(100)}))_2 & (h_\theta(x^{(100)}))_3 \end{bmatrix}$$

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

Pista:
```
np.sum( np.array([[1,2],[3,4]]) )
>>>> 10
```

Cálculo del gradiente



$\Theta^{(1)}$
$\Theta^{(2)}$
+1
+1
$h_\theta(x)$
forward propagation
$a^{(1)} = x$
(add $a_0^{(1)}$)
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$
(add $a_0^{(2)}$)
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)}) = h_\theta(x)$
$\Theta^{(1)}$
$\Theta^{(2)}$
+1
+1
$h_\theta(x)$
backward propagation
$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$
(remove $\delta_0^{(2)}$)
$\delta_j^{(3)} = a_j^{(3)} - y_j$
Input Layer
Hidden Layer
Output Layer

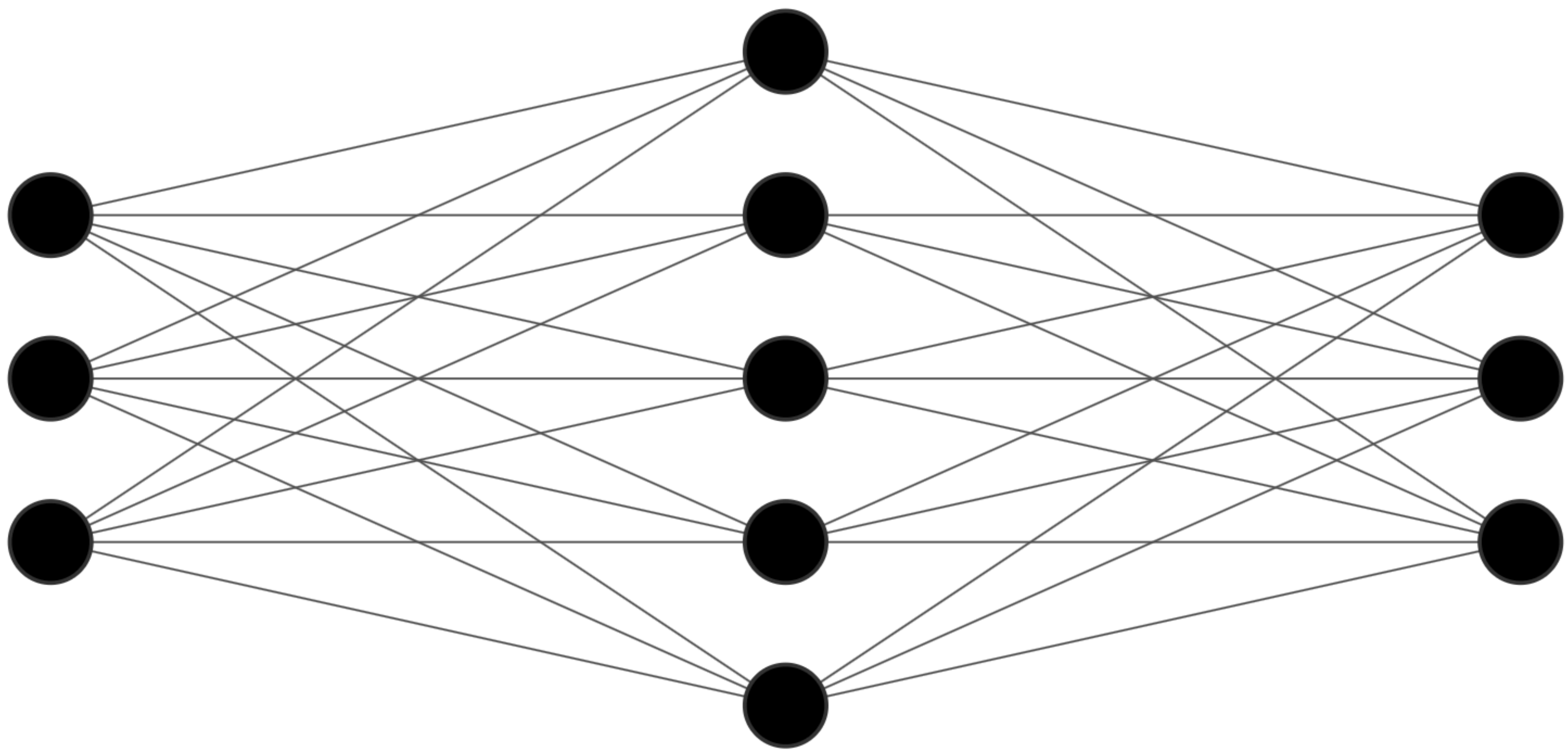
```
input_layer_size = 3
hidden_layer_size = 5
num_labels = 3
m = 100
```

$X : 100 \times 4$

$y : 100 \times 3$

$\Theta^{(1)} : 5 \times 4$

$\Theta^{(2)} : 3 \times 6$



Input Layer $\in \mathbb{R}^3$ Hidden Layer $\in \mathbb{R}^5$ Output Layer $\in \mathbb{R}^3$

http://alexlenail.me/NN-SVG/

```
input_layer_size = 3
hidden_layer_size = 5
num_labels = 3
m = 100
```

$X : 100 \times 4$

$y : 100 \times 3$

$\Theta^{(1)} : 5 \times 4$

$\Theta^{(2)} : 3 \times 6$

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ & \vdots & & \\ 1 & x_1^{(100)} & x_2^{(100)} & x_3^{(100)} \end{pmatrix} \quad y = \begin{pmatrix} y_1^{(1)} & y_2^{(1)} & y_3^{(1)} \\ y_1^{(2)} & y_2^{(2)} & y_3^{(2)} \\ & \vdots & \\ y_1^{(100)} & y_2^{(100)} & y_3^{(100)} \end{pmatrix}$$

$$\Theta^{(1)} = \begin{pmatrix} \Theta^{(1)}_{1,0} & \Theta^{(1)}_{1,1} & \Theta^{(1)}_{1,2} & \Theta^{(1)}_{1,3} \\ \Theta^{(1)}_{2,0} & \Theta^{(1)}_{2,1} & \Theta^{(1)}_{2,2} & \Theta^{(1)}_{2,3} \\ \Theta^{(1)}_{3,0} & \Theta^{(1)}_{3,1} & \Theta^{(1)}_{3,2} & \Theta^{(1)}_{3,3} \\ \Theta^{(1)}_{4,0} & \Theta^{(1)}_{4,1} & \Theta^{(1)}_{4,2} & \Theta^{(1)}_{4,3} \\ \Theta^{(1)}_{5,0} & \Theta^{(1)}_{5,1} & \Theta^{(1)}_{5,2} & \Theta^{(1)}_{5,3} \end{pmatrix} \Theta^{(2)} = \begin{pmatrix} \Theta^{(2)}_{1,0} & \Theta^{(2)}_{1,1} & \Theta^{(2)}_{1,2} & \Theta^{(2)}_{1,3} & \Theta^{(2)}_{1,4} & \Theta^{(2)}_{1,5} \\ \Theta^{(2)}_{2,0} & \Theta^{(2)}_{2,1} & \Theta^{(2)}_{2,2} & \Theta^{(2)}_{2,3} & \Theta^{(2)}_{2,4} & \Theta^{(2)}_{2,5} \\ \Theta^{(2)}_{3,0} & \Theta^{(2)}_{3,1} & \Theta^{(2)}_{3,2} & \Theta^{(2)}_{3,3} & \Theta^{(2)}_{3,4} & \Theta^{(2)}_{3,5} \end{pmatrix}$$

Cálculo del gradiente

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

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

$$\boxed{\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = \frac{1}{m} \sum_{k=1}^{m} (a_j^{(l)(k)} \delta_i^{(l+1)(k)})}$$

For $k = 1$ to m

Set $a^{(1)} = x^{(k)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(k)}$, compute $\delta^{(L)} = a^{(L)} - y^{(k)}$

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

for all l, i, j

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

vectorized, for all l

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)}$ if $j = 0$

$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}}$$

For every training example $(x^{(i)}, y^{(i)})$

```
input_layer_size = 3
hidden_layer_size = 5
num_labels = 3
m = 100
```

$X : 5 \times 4$ $\quad \Theta^{(1)} : 5 \times 4$ $\quad \Delta^{(1)} : 5 \times 4$

$y : 5 \times 3$ $\quad \Theta^{(2)} : 3 \times 6$ $\quad \Delta^{(2)} : 3 \times 6$

forward:

$$a^{(1)} = x^{(1)} : (4,1)$$

$$z^{(2)} = \Theta^{(1)} . a^{(1)} : (5,1)$$

$$a^{(2)} = g(z^{(2)}) : (5,1)$$

$$\text{add } a_0^{(2)} \rightarrow a^{(2)} : (6,1)$$

$$z^{(3)} = \Theta^{(2)} a^{(2)} : (3,1)$$

$$a^{(3)} = g(z^{(3)}) : (3,1)$$

backward:

$$\delta^{(3)} = a^{(3)} - y^{(1)} : (3,1)$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) : (6,1)$$

$$g'(z^{(2)}) = a^{(2)} .* (\overrightarrow{1} - a^{(2)}) : (6,1)$$

$$\text{remove } \delta_0^{(2)} \rightarrow \delta^{(2)} : (5,1)$$

$$\Delta^{(1)} = \Delta^{(1)} + \delta^{(2)} (a^{(1)})^T : (5,4)$$

$$\Delta^{(2)} = \Delta^{(2)} + \delta^{(3)} (a^{(2)})^T : (3,6)$$

1D and 2D arrays in Python

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

```
a = np.array([1,2,3])

a.shape
(3,)

b = np.array([4,5])

b.shape
(2,)

np.matmul(a[:, np.newaxis],b[np.newaxis, :])
array([[ 4,  5],
       [ 8, 10],
       [12, 15]])
```

Back prop

```
m = X.shape[0]
...

A1, A2, H =
  forward_propagate(X, Theta1, Theta2)
...

for t in range(m):
    a1t = A1[t, :]  # (401,)
    a2t = A2[t, :]  # (26,)
    ht = H[t, :]  # (10,)
    yt = y[t]  # (10,)

    d3t = ht - yt  # (10,)
    d2t = np.dot(Theta2.T, d3t) * (a2t * (1 - a2t))  # (26,)

    Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
    Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])
```

$$\delta^{(3)} = a^{(3)} - y^{(1)}$$

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

$$g'(z^{(2)}) = a^{(2)} .* (\overrightarrow{1} - a^{(2)})$$

$$\text{remove } \delta_0^{(2)} \rightarrow \delta^{(2)}$$

$$\Delta^{(1)} = \Delta^{(1)} + \delta^{(2)}(a^{(1)})^T$$

$$\Delta^{(2)} = \Delta^{(2)} + \delta^{(3)}(a^{(2)})^T$$

Cálculo del gradiente

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

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

For $k = 1$ to m

Set $a^{(1)} = x^{(k)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(k)}$, compute $\delta^{(L)} = a^{(L)} - y^{(k)}$

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

for all l, i, j

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

vectorized, for all l

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

$$\boxed{\begin{array}{ll} D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} & \text{if } j = 0 \end{array}}$$

$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}}$$

Comprobación del gradiente

```python
def computeNumericalGradient(J, theta):
    """
    Computes the gradient of J around theta using finite differences and
    yields a numerical estimate of the gradient.
    """

    numgrad = np.zeros_like(theta)
    perturb = np.zeros_like(theta)
    tol = 1e-4

    for p in range(len(theta)):
        # Set perturbation vector
        perturb[p] = tol
        loss1 = J(theta - perturb)
        loss2 = J(theta + perturb)

        # Compute numerical gradient
        numgrad[p] = (loss2 - loss1) / (2 * tol)
        perturb[p] = 0

    return numgrad
```

$$\frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\epsilon} \approx \frac{\partial}{\partial \theta_i} J(\theta)$$

$$\theta^{(i+)} = \theta + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix} \qquad \theta^{(i-)} = \theta - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix}$$

```python
def checkNNGradients(costNN, reg_param):
    """
    Creates a small neural network to check the back propogation gradients.
    Outputs the analytical gradients produced by the back prop code and the
    numerical gradients computed using the computeNumericalGradient function.
    These should result in very similar values.
    """
    # Set up small NN
    input_layer_size = 3
    hidden_layer_size = 5
    num_labels = 3
    m = 5

    # Generate some random test data
    Theta1 = debugInitializeWeights(hidden_layer_size, input_layer_size)
    Theta2 = debugInitializeWeights(num_labels, hidden_layer_size)

    # Reusing debugInitializeWeights to get random X
    X = debugInitializeWeights(input_layer_size - 1, m)

    # Set each element of y to be in [0,num_labels]
    y = [(i % num_labels) for i in range(m)]

    ys = np.zeros((m, num_labels))
    for i in range(m):
        ys[i, y[i]] = 1
```

```python
# Unroll parameters
nn_params = np.append(Theta1, Theta2).reshape(-1)

# Compute Cost
cost, grad = costNN(nn_params,
                    input_layer_size,
                    hidden_layer_size,
                    num_labels,
                    X, ys, reg_param)

def reduced_cost_func(p):
    """ Cheaply decorated nnCostFunction """
    return costNN(p, input_layer_size, hidden_layer_size, num_labels,
                  X, ys, reg_param)[0]

numgrad = computeNumericalGradient(reduced_cost_func, nn_params)

# Check two gradients
print('grad shape: ', grad.shape)
print('num grad shape: ', numgrad.shape)
np.testing.assert_almost_equal(grad, numgrad)
return (grad - numgrad)
```