

Práctica 3:

regresión logística multiclase y redes neuronales

```
from scipy.io import loadmat
```

```
data = loadmat('ex3data1.mat')
```

```
# se pueden consultar las claves con data.keys()
```

```
y = data['y']
```

```
X = data['X']
```

```
# almacena los datos leídos en X, y
```

```
# Selecciona aleatoriamente 10 ejemplos y los pinta
```

```
sample = np.random.choice(X.shape[0], 10)
```

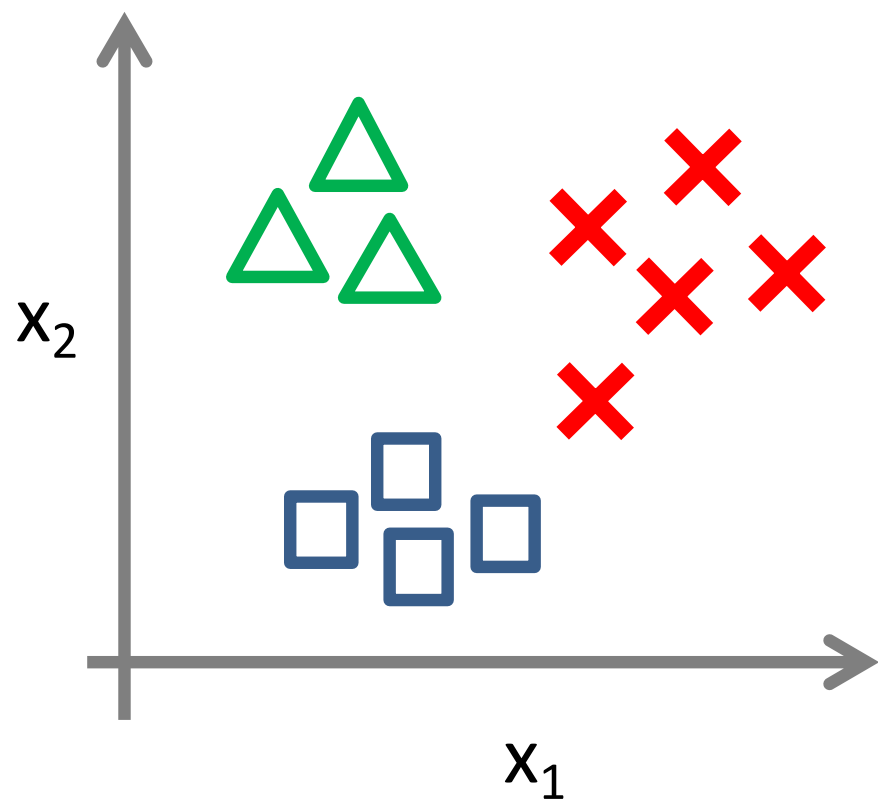
```
plt.imshow(X[sample, :].reshape(-1, 20).T)
```




```
plt.axis('off')
```

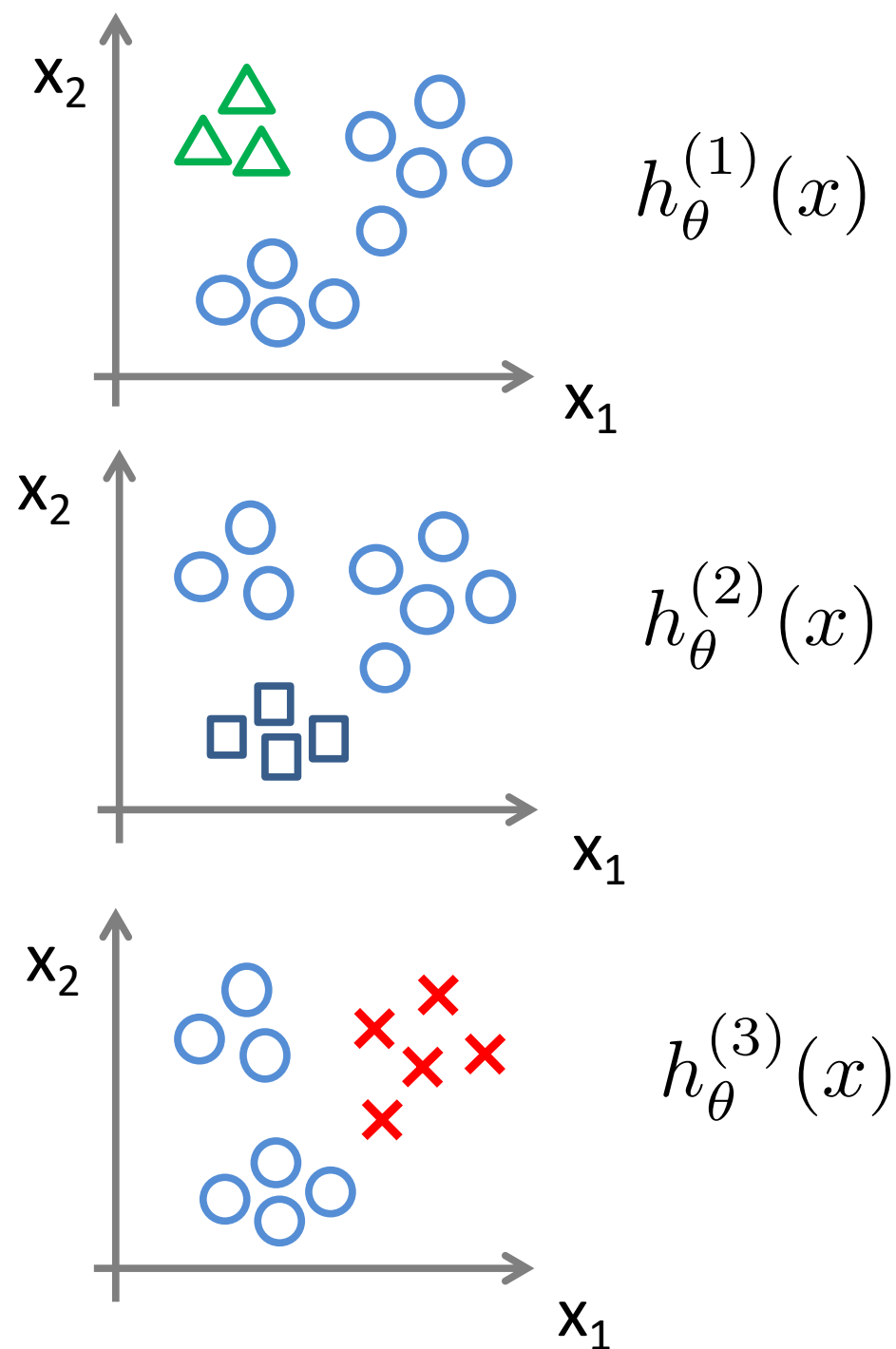


Regresión logística

One-vs-all (one-vs-rest):



Class 1: 
Class 2: 
Class 3: 



$$h_{\theta}^{(i)}(x) = P(y = i|x; \theta) \quad (i = 1, 2, 3)$$

```
def oneVsAll(X, y, n_labels, reg):  
    """
```

```
    oneVsAll entrena varios clasificadores por regresión logística con término  
    de regularización 'reg' y devuelve el resultado en una matriz, donde  
    la fila i-ésima corresponde al clasificador de la etiqueta i-ésima  
    """
```

Recuerda que el argumento y es un vector con etiquetas de 1 a 10, donde el dígito “0” se ha hecho corresponder con la etiqueta 10. Por otra parte, cuando entrenes al clasificador para la clase $k \in \{1, \dots, K\}$, tendrás que obtener un vector m -dimensional de etiquetas y donde $y_j \in \{0, 1\}$ indica si el ejemplo de entrenamiento j -ésimo pertenece a la clase k ($y_j = 1$) o a otra clase ($y_j = 0$). Para ello, te será útil saber que en Python `True * 1` es igual a 1 y `False * 1` es igual a 0.

```
▶ ▶≡ M↓
```

```
y = np.array([10, 10, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9])
```

```
▶ ▶≡ M↓
```

```
y == 10
```

```
array([ True,  True, False, False, False, False, False, False, False,  
       False, False, False, False, False, False, False, False, False,  
       False, False])
```

```
▶ ▶≡ M↓
```

```
(y == 10) * 1
```

```
array([1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Cálculo del coste: $\log(0)$ NAN

```
def sigmoid(x):  
    ## cuidado con  $x > 50$  devuelve 1  
    ##  
    s = 1 / (1 + np.exp(-x))  
  
    return s
```

```
def costReg(theta, reg, XX, Y):  
    m = Y.size  
    h = sigmoid(XX.dot(theta))  
  
    ... np.log(1 - h) ...
```

```
def costReg(theta, reg, XX, Y):  
    m = Y.size  
    h = sigmoid(XX.dot(theta))  
  
    ... np.log(1 - h + 1e-6) ...
```

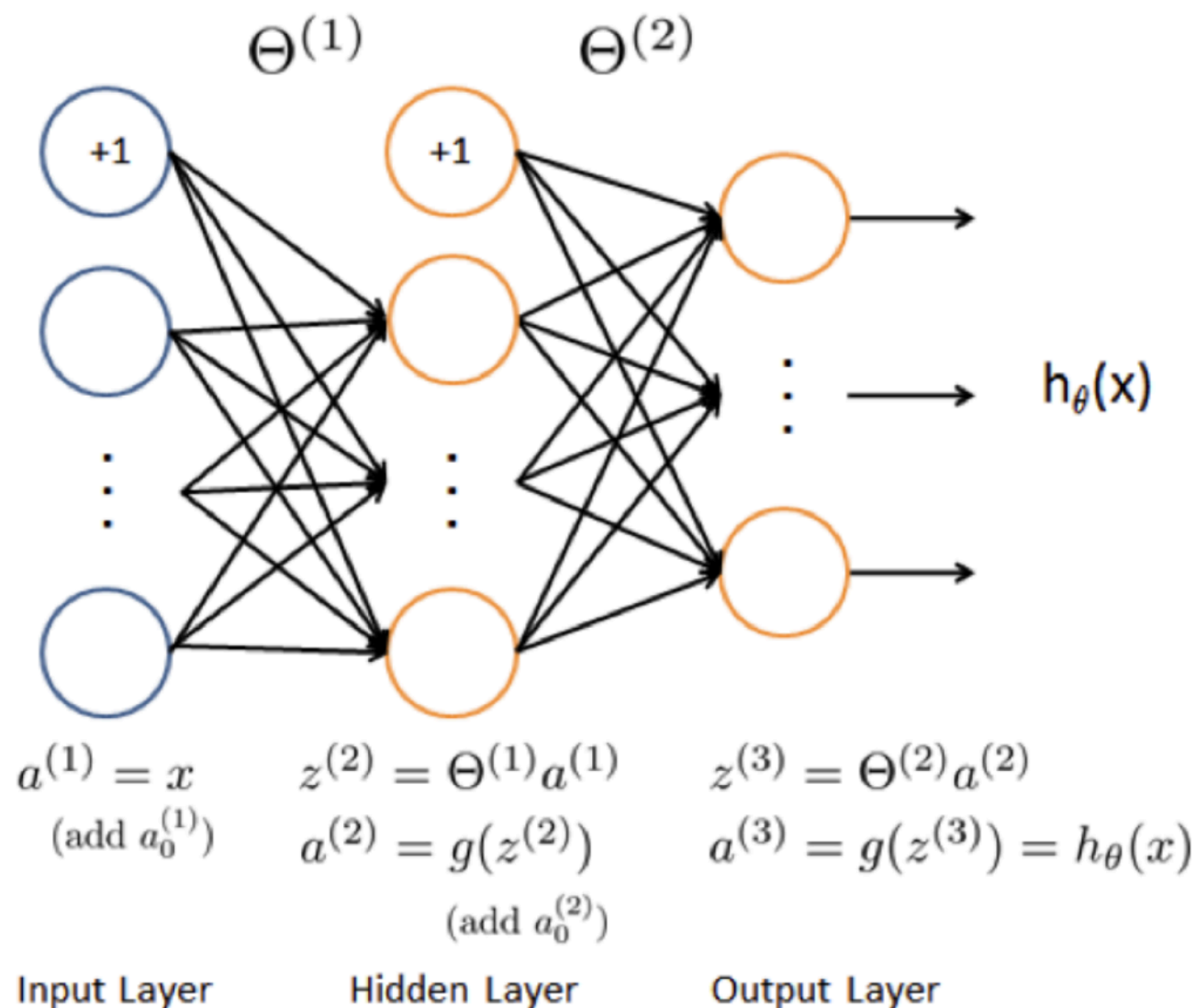


Cálculo del gradiente: 'y' es 2D

```
def ejemplo_error():  
    data = loadmat('ex3data1.mat')  
    y = data['y'] # (5000, 1)  
    X = data['X'] # (5000, 400)  
    m = np.shape(X)[0]  
    X1s = np.hstack([np.ones([m, 1]), X]) # (5000, 401)  
    theta = np.zeros(X1s.shape[1]) # (401, )  
    H = sigmoid(np.matmul(X1s, theta)) # (5000, )  
    error = H - y # (5000, 5000)  
    error = H - np.ravel(y) # (5000)
```



Redes neuronales



La red neuronal está entrenada de forma que la primera neurona de salida se activa cuando reconoce un 1, la segunda cuando reconoce un 2, y así sucesivamente hasta la décima que se activa cuando reconoce un 0

```

weights = loadmat('ex3weights.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']
# Theta1 es de dimensión 25 x 401
# Theta2 es de dimensión 10 x 26

```