Introducción al Aprendizaje Profundo

Modelo de redes neuronales

Francisco Cervantes

Septiembre, 2019

HOY …

- Finalizar la actividad pendiente: Ex02

- Un poco más de notación

- Representación de redes neuronales

- Ex03

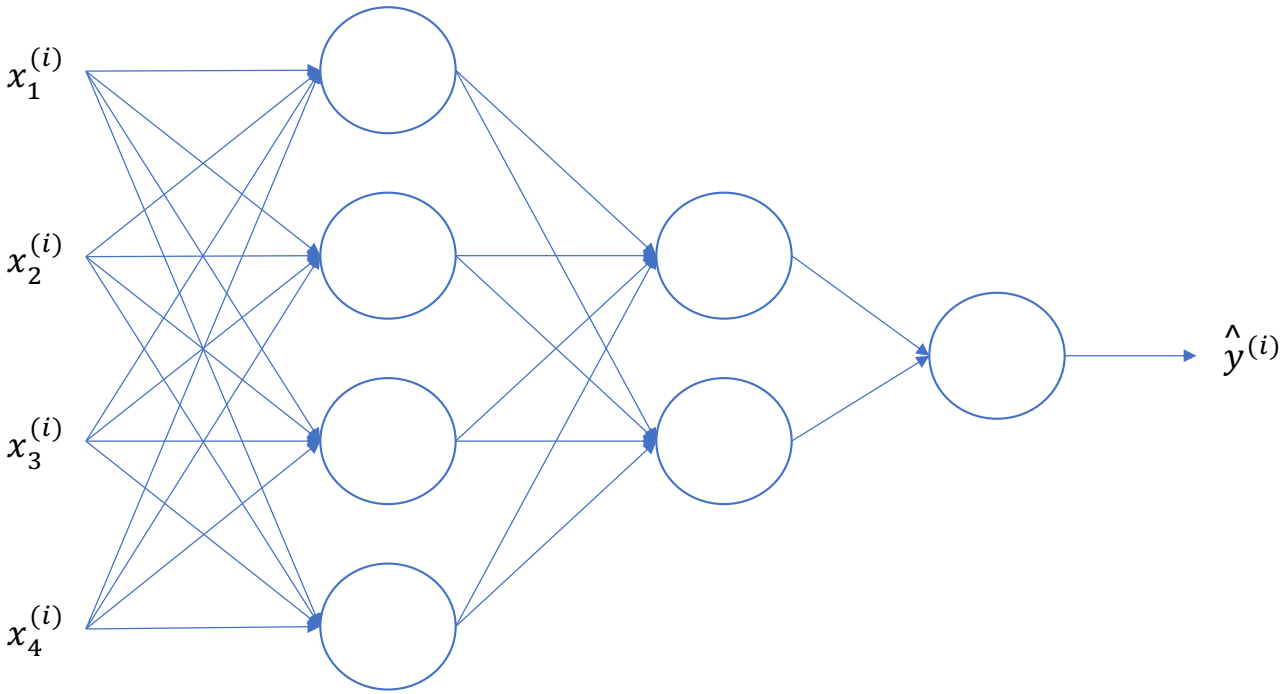
Notación

superíndice (i)	denota el i -ésimo ejemplo de entrenamiento.
superíndice $[i]$	denota la i -ésima capa en una red neuronal.
m	denota el número de ejemplos en el dataset.
n_x	denota el tamaño de los datos de entrada.
n_y	denota el número de clases (tamaño de dato de salida).
$n_h^{[l]}$	denota el número de neuronas ocultas en la capa l-ésima.
l	denota el número de capas en una red neuronal.
$X \in \mathbb{R}^{n_x \times m}$	denota la matriz de entrada.
$x^{(i)} \in \mathbb{R}^{n_x}$	denota el i –ésimo ejemplo representado como una columna en una matriz X.

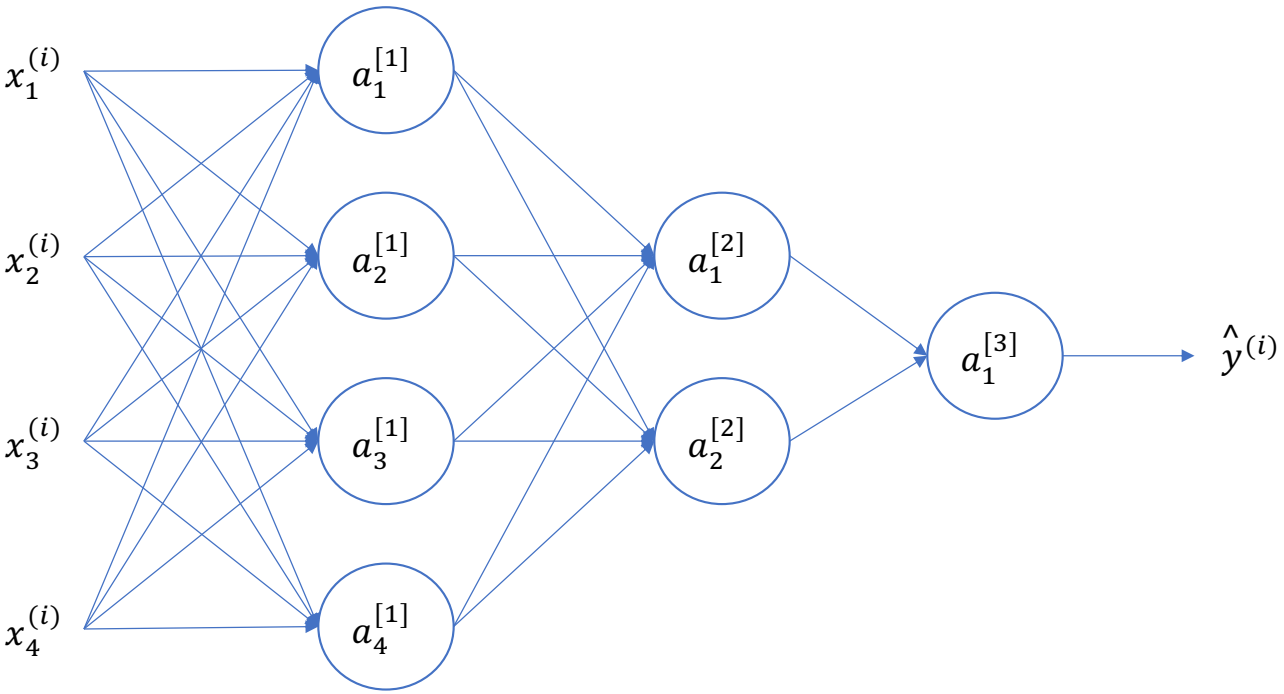
Notación

$Y \in \mathbb{R}^{n_y \times m}$	denota la matriz de etiquetas.
$y^{(i)} \in \mathbb{R}^{n_y}$	denota la etiqueta de salida para el i –ésimo ejemplo.
$W^{[l]} \in \mathbb{R}^{n_h^{[l+1]} \times n_h^{[l-1]}}$	denota la matriz de pesos en la capa l-ésima.
$b^{[l]} \in \mathbb{R}^{n_h^{[l+1]}}$	denota el vector bias en la capa l-ésima.
$\hat{y} \in \mathbb{R}^{n_y}$	es al predicción (vector de salida). También se puede denotar por $a^{[L]}$ donde L es el número de capas en la red neuronal.

Representación de una red neuronal



Representación de una red neuronal

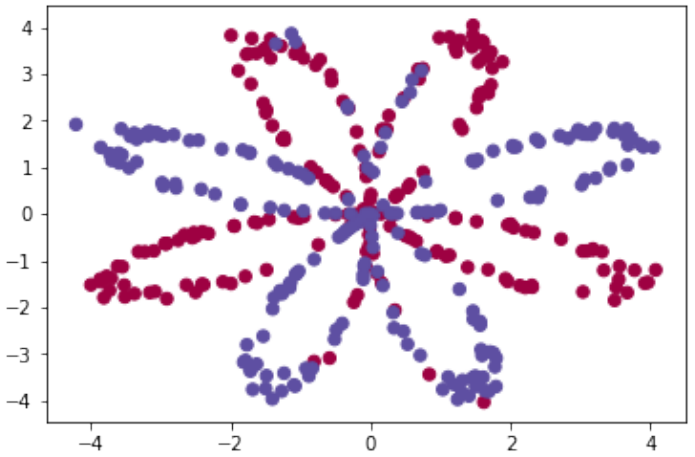


EX03 Clasificación de datos con una red neuronal

Es tiempo de construir tu primer red neuronal, la cual tendrá una capa oculta. En esta actividad observaremos una gran diferencia entre este modelo y el que implementamos en la actividad EX02 (utilizando regresión logística).

Propósitos de aprendizaje

- Implementar una red neuronal para 2-clases (con una capa oculta)
- Utilizar funciones de activación (unidades de activación), tales como `tanh`
- Calcular la perdida de entropia cruzada (cross entropy loss)
- Implementar el forward y backward propagation.



¿Qué tan bueno sería utilizar regresión logística?

Antes de construir nuestra red neuronal, veamos que tan buena es la regresión logística para abordar el problema planteado en esta actividad. Utilicemos `sklearn` para construir nuestro clasificador.

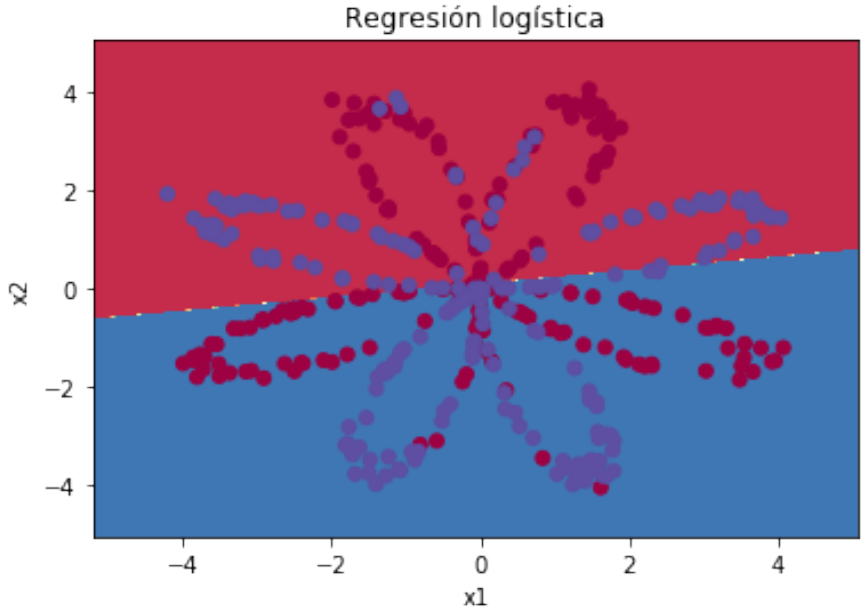
```
# Entrenamiento del clasificar basado en regresión logística

model = sklearn.linear_model.LogisticRegressionCV(cv=3);
model.fit(X.T, Y[0].T)
```

```
# Grafiquemos el límite de decisión generado por la regresión lineal
plot_db(lambda x: model.predict(x), X, Y[0])
plt.title ("Regresión logística")

# Imprimir la exactitud del modelo
predictions = model.predict(X.T)
print(f"Exactitud: {float((np.dot(Y[0],predictions) + np.dot(1-Y[0],1-predictions))/float(Y[0].size)*100) }")
```

¿Qué tan bueno sería utilizar regresión logística?



Modelo de una red neuronal

La regresión logística no realizó una buena clasificación de los datos. Probemos una red neuronal con una capa oculta.

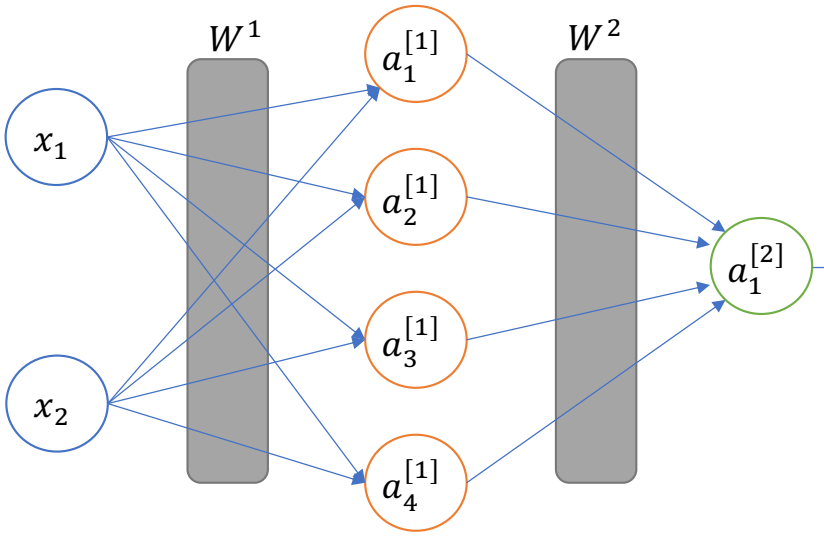
Recordemos la metodología general utilizada para construir un modelo basado en una red neuronal:

1. Definir la estructura de la red neuronal.
    1. Número de unidades de entrada
    2. Número de unidades ocultas
    3. Número de capas
    4. Funciones de activación
    5. etc.

2. Inicializar los parámetros

3. Etapa de aprendizaje
    1. Implementar el forward propagation
    2. Calcular el error (loss function)
    3. Implementar el back propagation (obtener gradientes)
    4. Actualizar los parámetros

Modelo de una red neuronal



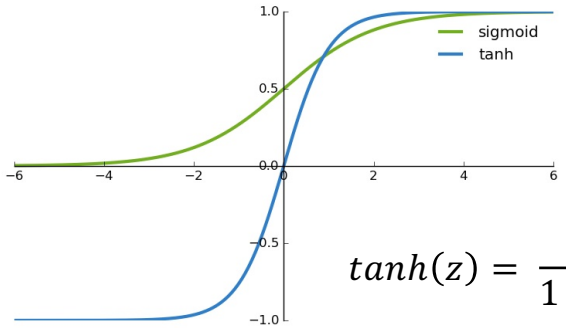
Capa de entrada

Capa Oculta (tanh)

Capa de Salida (sigmoid)

Matemáticamente para un ejemplo $x^{(i)}$:

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1](i)}$$
$$a^{[1](i)} = tanh(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2](i)}$$
$$a^{[2](i)} = \sigma\,(z^{[2](i)})$$
$$y^{(i)} = a^{[2](i)}$$



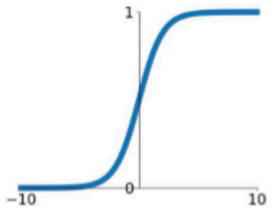
$$tanh(z) = \frac{2}{1+e^{-2z}} - 1$$

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$$

Algunas funciones de activación clásicas

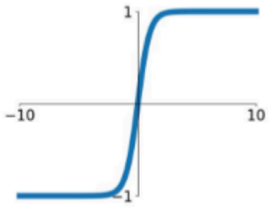
Sigmoid

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



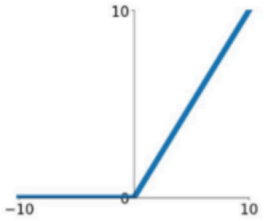
tanh

$$\tanh(z)$$



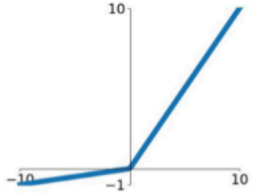
ReLU

$$\max(0, z)$$



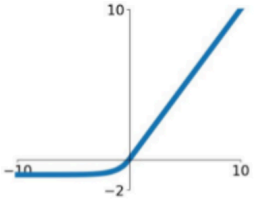
Leaky ReLU

$$\max(0.1z, z)$$



ELU

$$\begin{cases} z & z \geq 0 \\ \sigma(e^z - 1) & z < 0 \end{cases}$$



¿Por qué utilizar funciones derivables?

1.1 Definición de la estructura de la red neuronal

Ejercicio 1: Definamos una función **layer_size()** que permita calcular el tamaño de la capa de entrada y la capa de salida para la red neuronal (a partir de los datasets: **X** y **Y**). Adicionalmente, se debe establecer el número de unidades en la capas ocultas (para esta actividad serán 4 unidades en la capa oculta).

1.2 Inicialización de parámetros

Antes de continuar, verifica que las dimensiones de tus parámetros son correctos acorde a la estructura de la red neuronal propuesta para esta actividad.

Ejercicio 2: Implemetemos la función **init_parameters()**. Esta función debe inicializar:

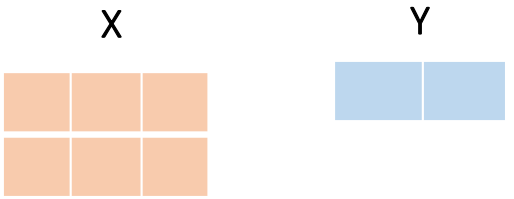
- la matriz de pesos con valores aleatorios.
- los bias en cero (observe que se trata de vectores de bias).

Tips:

• utilice $np.random.randn(a,b)*0.01$ para inicializar una matriz de dimensiones $a \times b$ con valores aleatorios.

• utilice $np.zeros((a,b))$ para inicializar una matriz de dimensiones $a \times b$ con ceros.

Inicialización de parámetros



(2, 3)

(1, 2)

m = 3

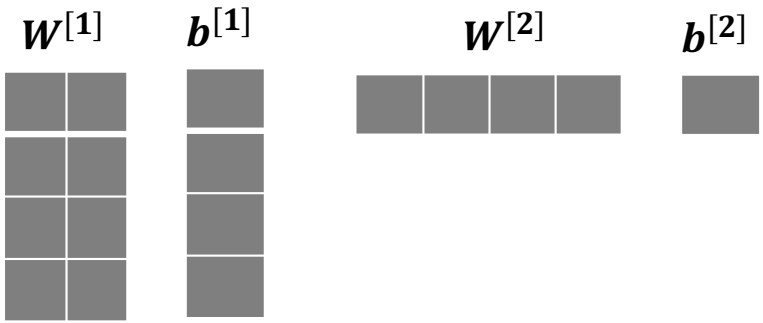
1. ¿cuál es la estructura utilizada para la red neuronal?

   $n_x = 2$

   $n_h = 4$

   $n_y = 1$

2. Inicialicemos los parámetros (pesos y bias de la red).



$W^{[1]} = (n_h, n_x)$

$b^{[1]} = (n_h, 1)$

$W^{[2]} = (n_y, n_h)$

$b^{[2]} = (n_y, 1)$

1.3 Etapa de aprendizaje: forward propagation

Ejercicio 3: Implementemos el forward propagation de nuestro algoritmo. Antes de iniciar consultemos la representación matemática de la red neuronal vista anteriormente. Los pasos a realizar en este ejercicio son los siguientes:

1. Recuperar los parámetros del diccionario parameters utilizando parameters["..."].

2. Implementar el forward propagation. En nuestra red neuronal, esto implica calcular $Z^{[1]}, A^{[1]}, Z^{[2]}$ y $A^{[2]}$. Observe que $A^{[2]}$ es el vector con las predicciones para todos los ejemplos de entrenamiento.

3. Almacenar en cache los valores que serán requeridos en el backpropagation.

***Tip*:** podemos utilizar la función sigmoid(z) definida en ex03_utils y la función np.tanh(z) que es parte de la librería numpy.

1.3 Etapa de aprendizaje: función de costo

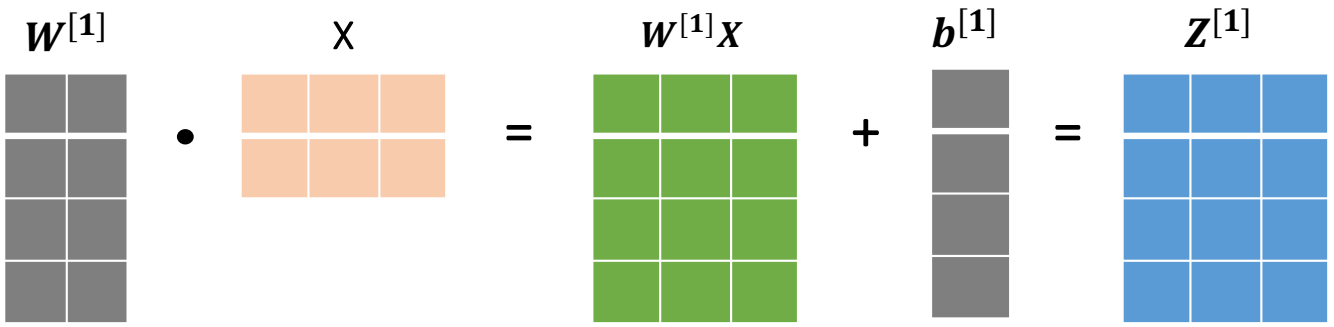
Ahora que hemos calculado el valor de A2, es decir la unidad de activación $A^{[2]}$ de nuestra red neuronal que contiene $a^{[2](i)}$ para cada ejemplo de entrenamiento, podemos calcular la función de costo J. Esto es:

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$$

Ejercicio 4: Implementemos la función **cost_function()** para calcular el costo J.

Forward propagation

$z^{[1]} = W^{[1]}X + b^{[1]}$



$A^{[1]} = \tanh(z^{[1]})$

$z^{[2]} = W^{[2]}X + b^{[2]}$

$A^{[2]} = \text{sigmoid}(z^{[2]})$

$\hat{Y} = A^{[2]}$

1.3 Etapa de aprendizaje: backward propagation

Es momento de implementar el backward propagation. Para esto, utilizaremos el contenido de la variable **cache** que hemos calculado durante el forward propagation.

Ejercicio 4: Para algunos de nosotros, el backpropagation suele ser la parte menos sencilla (por las matemáticas). Implementa la función **bpropagation()**.

Tomemos como base las siguientes ecuaciones :

Backward propagation - gradiente descendiente

$$dZ^{[2]} = A^{[2]} - y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Para calcular $dZ1$ necesitamos calcular $g^{[l]'}(Z[l])$. Debido a que $g^{[l]}()$ es la función de activación, si $a=g^{[1]}(z)$ entonces $g^{[1]'}(z)=1-a^2$. Así que debemos calcular $g^{[1]'}(Z^{[1]})$ utilizando (1 - np.power(A1, 2)).