Redes Neuronales: Entrenamiento y Comportamiento

UNIVERSIDAD COMPLUTENSE
MADRID

Fernando Bueno

Trabajo de Fin de Grado- Grado en Ingeniería Matemática

2019

Tutor: Antonio López Montes

Dto.: Análisis Matemático y Matemática Aplicada

Índice:

1-. Introducción.

La motivación por la cual he escogido este tema es por ser, las Redes Neuronales, un apartado importante dentro de la Inteligencia Artificial, que me parece un tema importante actualmente en tecnología debido a su repercusión y la inversión que hay en ello.

El propósito de este trabajo es estudiar la respuesta de una Red Neuronal según cambios en sus diferentes atributos para el mismo conjunto de datos dado.

Para ello utilizaré dos ejemplos concretos, con diferentes conjuntos de datos reales y diferentes tipos de Redes Neuronales en cada uno de ellos.

En el primero, utilizaré el conjunto de datos "mnist" que consiste en un conjunto de fotografías dadas en las que se ven diferentes imágenes de números escritos a mano y la Red Neuronal intentará reconocer dicho número. Las imágenes tienen una resolución de 28x28 pixeles y los números van del 0 al 9.

En el segundo, utilizaré un conjunto de datos de la web filmográfica IMDB, del cual se extraerán 25000 diferentes críticas de películas etiquetadas según el sentimiento de dicha crítica. Para el buen funcionamiento de la Red Neuronal y la agilidad del estudio, he puesto el límite máximo de palabras a analizar en 500. Para este último caso, utilizare una Red Neuronal Convolucional.

Otra idea que haré ver más adelante es que, aunque ciertos tipos de Redes se usan más para ciertos tipos de conjuntos de datos, cualquier Red puede ser usada para cualquier tipo de datos. Por ejemplo, las Redes Neuronales Convolucionales se usan más a menudo para imágenes y en este trabajo lo haré al contrario.

1.1 Definiciones y ejemplo

Es importante tener presentes algunas definiciones sobre este campo.

Machine Learning: forma parte de las ciencias de la información y es parte de la rama de la inteligencia artificial, siendo su objetivo el de desarrollar técnicas que permitan que las máquinas "aprendan", es decir, programar comportamientos en ellas a partir de ejemplos, y no de forma explícita.

El aprendizaje automático en las máquinas tiene aplicaciones útiles como: motores de búsqueda, corrección automática de textos, detección de fraude en sistemas virtuales de intercambio de información, análisis de valores en mercados financieros, juegos, robótica…

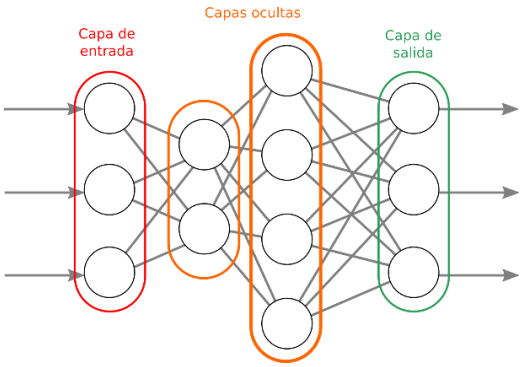
Se debe decir también, que existen dos tipos de aprendizaje en el proceso de *Machine Learning*:

- El aprendizaje no supervisado, en el que un modelo estudia los datos que se quieren tratar e intenta obtener ciertos patrones en ellos sin conocimiento a priori de los resultados del conjunto sometido a estudio (*Clustering*, agrupación de datos).

- El aprendizaje supervisado en el que dado un subconjunto de datos del que sabemos los resultados, se utiliza para entrenar o ajustar el modelo, para que sea capaz de predecir también los resultados del conjunto total, del que desconocemos los resultados (Redes Neuronales, Regresión, Clasificación).

En el caso concreto de este trabajo, nos adentraremos en el ámbito del *Machine Learning* a través de las Redes Neuronales y su comportamiento dependiendo de los diferentes parámetros que hay en ella, así como los datos que procese y demás posibles variaciones. Para ello voy a introducir, en primer lugar, el concepto de Red Neuronal, describiendo brevemente cómo funciona.

Red Neuronal: es un modelo computacional basado en un conjunto de unidades neuronales simples, cada una con una regla o una condición diferente. A dicha condición se le llama "función de activación", y esta se debe cumplir antes de propagarse la información a través del resto de unidades neuronales. Cada unidad neuronal o neurona, está conectada con otras a través de enlaces, que a su vez tienen asociados unos pesos que, activan, aumentan, disminuyen o desactivan el valor de cada neurona de la red.



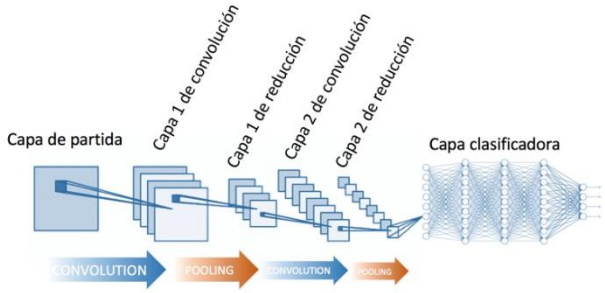
Esquema de una Red Neuronal y sus capas

Dentro de una red neuronal, existen capas, que son conjuntos de neuronas por los cuales pasa la información que se desea clasificar o usar en una regresión. Dichas capas están siempre ocultas en cuanto a que, normalmente, no se ve el resultado del tratamiento de información en las mismas; sólo hay dos capas no ocultas, la capa de entrada de datos, y la de salida.

El propósito de las Redes Neuronales, es que tienen la capacidad de aprender por si solas. Dado un conjunto de entrenamiento, del cual conocemos los resultados, se introduce en la red con un método *backward* (hacia atrás), y de esta manera, las funciones de activación y los pesos, van cambiando hasta adaptarse al conjunto conocido y después ser usado en un conjunto real. Este método se basa en la "función de pérdida", que evalúa la diferencia entre el valor de salida de la red y el valor real del conjunto de entrenamiento. Cambiando el valor de los pesos en los enlaces entre neuronas y el sesgo en cada neurona, se intenta minimizar el valor de la función pérdida.

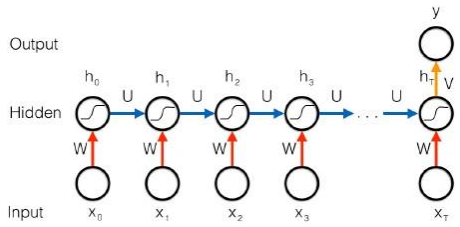
Existen diferentes tipos de Redes Neuronales, pero cabe destacar tres de ellos por encima del resto:

1. Redes Neuronales Convolucionales(CNN): Este tipo de red, como su propio nombre indica, utiliza la operación de convolución como base para el procesamiento de los datos. Dentro de esta red, existen matrices llamadas filtros y cada una de ellas detecta, a su vez, distintos tipos de características dentro de los datos que se quieren procesar. Se usan principalmente para el procesamiento de imágenes, con lo cual, dichos filtros serán capaces de detectar, por ejemplo, bordes. Además, estos filtros se moverán por la imagen original según un parámetro que mide la longitud del salto (stride).
   Por otro lado, para reducir la dimensión de los datos a procesar, lo que hace a estas redes muy interesantes, se utiliza otra operación llamada *Poolling*, que tiene diferentes formas de implementación.



Estructura de una CNN

2. Redes Neuronales Recurrentes: Las Redes Neuronales Recurrentes son aquellas capaces de influir sobre si mismas a la hora de entrenarse. Dada una RNN, cada neurona de la misma devuelve dos parámetros, el habitual output o coste y otro que representa la información obtenida en ella, que a su vez es traspasada a la siguiente neurona para influir sobre ella y así crear la recurrencia.
   La RNN (*Recurrent Neural Network*) más usada en la actualidad es la de Long-Short Term Memory, la cual a diferencia de las RNN tradicionales incluye bloques LSTM, que se encargan de recordar un valor y determinar la duración en el tiempo del mismo. Además, estas unidades deciden qué entradas debe almacenar y si debe recordarlas, borrarlas o enviarlas como salidas de la red. La principal ventaja de las LTSM es que no necesita de mucha memoria para trasladar información entre neuronas lejanas.



Estructura de una RNN

3. Redes Neuronales Densas: Son aquellas redes en las que todas las neuronas de una capa están conectadas a todas las neuronas de la siguiente. Estas redes, obviamente son más eficaces en cuanto al tratamiento de información porque cada neurona dispone de más datos para tratar, pero a su vez, una Red Neuronal Densa, es más lenta a la hora de procesar los datos, por la misma razón, la gran cantidad de información que mueven de una capa a otra.

2-. Ejemplos, fundamentos matemáticos y algoritmos.

A continuación, muestro un ejemplo de una Red Neuronal muy sencilla creada en Python.

Consiste en una red compuesta por tres entradas y una neurona que devuelve un solo valor de salida. A la Red Neuronal, se le dan tres vectores de entrada de dimensión tres, de ceros y unos, y un vector de dimensión cuatro de salida, de ceros y unos también, en el que cada posición corresponde a la respuesta de cada uno de los vectores de entrada. Dicha Red Neuronal intentará predecir si corresponderá un a nuevo vector de entrada un cero o un uno en su salida.

Esta Red Neuronal, compuesta sólo por una neurona, se llama Perceptron y es la Red Neuronal más sencilla posible y la primera estructura de la historia en ser diseñada para esta rama de la Inteligencia Artificial. El Perceptrón Simple de Rosenblatt data del año 1959.

En resumen, dado un vector de dimensión tres de ceros y unos, la Red Neuronal se encarga de asignarle un cero o un uno.

En concreto, le doy los vectores [0,0,1], [1,1,1], [1,0,1], [0,1,1] y un resultado a priori [0,1,1,0] de ceros y unos para cada vector respectivamente. En primer lugar, veremos los pesos aleatorios iniciales para cada vector, después los pesos obtenidos una vez entrenado el modelo, y finalmente, el resultado para un vector que no se encuentra entre los datos de entrenamiento.

```
import numpy as np #Librería que importa funciones para operar con matrices y vectores.

training_set_inputs = np.array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]]) #Entradas del conjunto de entrenamiento pa
ra la Red Neuronal
training_set_outputs = np.array([[0, 1, 1, 0]]).T #Salidas del conjunto de entrenamiento para la Red Neuronal.

class NeuralNetwork():
    def __init__(self):
        np.random.seed(1)#Generador de números aleatorios, con la misma raiz para generar siempre los mismos.
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1 #le damos pesos aleatorios entre -1 y 1 a las diferen
tes entradas, como tenemos tres entradas y una neurona, sera una matriz 3x1

    def __sigmoid(self, x): #Defino la funcion sigmoid en forma de S y dado x que me devuelva f(x). Se pasan la suma
ponderada de las entradas a través de esta función para normalizarlos entre 0 y 1.
        return 1 / (1 + np.exp(-x))

    def __sigmoid_derivative(self, x): #La derivada de la función Sigmoid, el gradiente, que indica la confianza que te
nemos en el paso siquiente
        return x * (1 - x)

    def think(self, inputs): #Pasa las entradas a través de nuestra red neuronal (una neurona), multiplicando inputs p
or sus pesos.
        return self.__sigmoid(np.dot(inputs, self.synaptic_weights))

    # Entrenamos a la red neuronal a través de un proceso de prueba y error
    # Se realiza un ajuste de los pesos sinápticos cada vez.
    def train(self, training_set_inputs, training_set_outputs, number_of_training_iterations):
        for iteration in range(number_of_training_iterations): #Tantas iteraciones como se quiera con cuidado de no s
obrentrenar.
            output = self.think(training_set_inputs)
            error = training_set_outputs - output #Calcular el error entre lo real y lo esperado. Función pérdida.

            # Multiplica el error por la entrada y nuevamente por el gradiente de la curva Sigmoid
            # Esto significa que los pesos menos confiables se están ajustando más
```

```
        # Esto significa que las entradas que son cero, no causan cambios en los pesos.
        adjustment = np.dot(training_set_inputs.T, error * self.__sigmoid_derivative(output))

        # Ajuste de los pesos
        self.synaptic_weights += adjustment


if __name__ == "__main__": #llamada para inicializar el propio programa

    #Iniciar una red neuronal de una neurona
    neural_network = NeuralNetwork()

    print ("Random starting synaptic weights: ")
    print (neural_network.synaptic_weights)

    # El conjunto de pruebas. Tenemos cuatro ejemplos, cada uno consiste en 3 valores de entrada
    # y un valor de salida.
    training_set_inputs = np.array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    training_set_outputs = np.array([[0, 1, 1, 0]]).T

    # Entrenar la red neuronal utilizando el conjunto de entrenamiento.
    # Realizar 10000 veces y realizar un ajuste más pequeño cada vez.
    neural_network.train(training_set_inputs, training_set_outputs, 10000)

    print ("New synaptic weights after training: ")
    print (neural_network.synaptic_weights)

    # Prueba la red neuronal con una nueva situación
    print ("Considering new situation [1, 0, 0] -> ?: ")
    print (neural_network.think(np.array([1, 0, 0])) )
```

```
Random starting synaptic weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
New synaptic weights after training:
[[ 9.67299303]
 [-0.2078435 ]
 [-4.62963669]]
Considering new situation [1, 0, 0] -> ?:
[0.99993704]
```

2.1 Modelo matemático

Para poder explicar el funcionamiento matemático de una red neuronal, hay que empezar por explicar el funcionamiento de una neurona dentro de la Red. Cada neurona se encarga de recibir unos datos, de los que se conoce su comportamiento y clasificarlos de forma binaria, creando así un modelo que pueda ser utilizado para datos futuros de los que no se conoce su comportamiento. Posteriormente, las neuronas van trasladando la información entre sí.

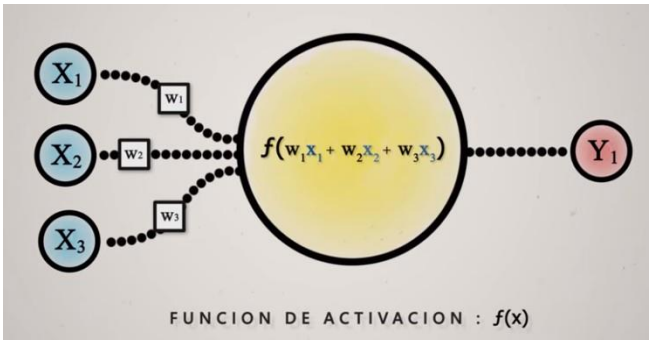
Cada neurona o capa de la Red procesa un tipo o un conjunto diferente de atributos de los datos de entrenamiento, que, puestos en común con las demás neuronas, son capaces de clasificar los datos en sí.

El propósito es, crear un modelo a partir de datos conocidos para después utilizarlo con datos de los que se desconoce su comportamiento. Por ejemplo, crear un modelo que relacione el precio de los paraguas y el mes del año. Conociendo el precio del paraguas en los meses impares, se puede crear un modelo sobre el precio durante todo el año, aunque no se sepa el precio real en los meses pares.

Dicho modelo cambia según los datos de entrada y puede ser modificado para ajustarse lo mejor posible a nuestro conjunto, por ejemplo, con el método de mínimos cuadrados.

En una Red Neuronal, cada neurona, internamente, tiene un funcionamiento para clasificar los datos recibidos. La neurona recibe los valores de entrada multiplicados por sus respectivos pesos, provenientes de otras neuronas, que será la importancia que tendrá dicha variable en el aprendizaje de la Red Neuronal. Además, existe un término independiente, dicho término, en la neurona, viene dado como una variable más de entrada, asociada siempre a la variable 1, y se le llama Sesgo. El sesgo podremos ajustarlo a gusto modificando el peso asociado a él, para un entrenamiento más preciso de la Red Neuronal.

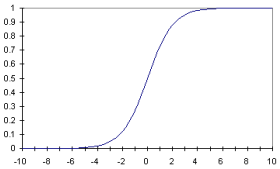
Se puede ver fácilmente que el efecto de concatenar muchas operaciones de lineales o sumar muchas líneas rectas, equivale a solamente haber hecho una línea recta, una única operación de regresión lineal. Ese es un problema que nos limitaría una Red Neuronal, porque no serviría de mucho aplicar las diferentes capas a nuestros datos.



En este momento es donde entra en juego la función de activación, que, en pocas palabras, deslinealiza la salida de cada neurona para que la concatenación de capas tenga sentido al introducir la no linealidad que nos permite que la Red pueda aproximar cualquier función.

Hay muchos tipos diferentes de funciones de activación, a continuación, muestro cuatro ejemplos:

- Función escalonada: $f(x) = \begin{cases} 0 \; si \; x \; \leq U \\ 1 \; si \; x \; > U \end{cases}$, siendo U el umbral deseado. Dicha función no será muy útil para este caso, debido a su curvatura, no es derivable en todos los puntos.
- Función Sigmoide: $f(x) = \sigma(x) = \frac{1}{1+e^x}$, la cual es bastante útil por su curvatura y por su capacidad de representar probabilidades que vienen siempre representadas en un rango de cero a uno.



Función Sigmoide

- Función Tangente Hiperbólica: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, que tiene las mismas ventajas que la función sigmoide, pero cuyo rango varía de menos uno a uno.
- Función ReLU: $f(x) = \begin{cases} 0 \; si \; x \; \leq U \\ x \; si \; x \; > U \end{cases}$, siendo U el umbral deseado.

Ahora, para resumir los valores que manejamos en el fondo matemático de una Red Neuronal:

$$C_i^L = (a_i^L - y_i^L)^2$$

$$z_j^L = \sum_i (w_{ij}^L * a_i^{L-1} + b_j^L)$$

$$a_i^L = \sigma(z_i^L)$$

- $a_i^{(L)}$. Indica el comportamiento de una neurona i en una determinada capa L. Sería el resultado de aplicar la función de activación (σ) a la información recibida de la capa anterior.
- $z_i^{(L)}$. En muchas publicaciones, se llama "valor de entrada neta", es decir, al sumatorio de los anteriores $a_i^{(L-1)}$multiplicados por sus respectivos pesos de conexión con esta neurona más el sesgo. Normalmente, este valor varía entre 0 y 1 o entre 1 y -1, esto dependerá de la función de activación.
- $b_i^{(L)}$. El sesgo, comentado anteriormente, se representa con la letra b y no es más que un peso asociado a la variable x=1.
- $w_{ij}^{(L)}$. Los pesos se representan por capas, con el superíndice de la capa a la que llegan, e indicando las neuronas i, j que conectan entre la capa (L-1) y la capa (L).
- $y_i^{(L)}$. Para determinar el valor esperado o valor real de salida de una neurona i en la capa L se utiliza la letra y Dicho valor se usará para entrenar a la Red Neuronal, y que aprenda.
- $C_i^{(L)}$. Sería el error en cada neurona de la capa en la que el algoritmo *Backpropoagation* está actuando, dicho algoritmo, viene explicado a continuación. Obviamente este es un tipo concreto de error usado como ejemplo en este caso, pero los errores pueden ser calculados de muchas maneras diferentes.

De esta manera, tenemos una concatenación de modelos lineales, modificados por funciones de activación que, en conjunto, serán capaces de procesar información con el objetivo de clasificarla o usarla en una regresión. En definitiva, una Red Neuronal.

2.2 Aprendizaje. Algoritmo de Backpropagation. Algoritmo del descenso del Gradiente.

Las Redes Neuronales que se a usan utilizan el método de aprendizaje supervisado, ya que voy a entrenarlas antes de que funcionen por si solas. Aunque son también muy usadas para semi-supervisado como ocurre en NLP (*Natural language procesing)* o no supervisado como para *clustering*.

Para comenzar el proceso de aprendizaje de mi Red Neuronal, dispongo de un subconjunto de datos de los que se conoce el resultado, el *training set*, para pasarlos por la red y que "aprenda", es decir, que modifique sus valores internos para ser capaz de por sí sola predecir la clasificación de la información. Esto se hace minimizando los errores de la red al compararlos con el valor real, dado un *training set*.

Durante mucho tiempo (desde la primera publicación del "Perceptron" como idea primigenia de las Redes Neuronales en 1969), este proceso de aprendizaje se realizaba "hacia delante", viendo como influían los cambios de la primera capa en las posteriores y hasta la salida, posteriormente la segunda etc, pero el problema de este método es claro: una Red Neuronal densa, con muchas conexiones entre neuronas, hace este proceso largo y complicado, ya que básicamente había que mirar cada camino desde cada neurona hasta el *output* final, siendo el número de caminos diferentes extremadamente alto.

Fue el método del *Backpropagation* (publicado por Rumelhart, Hinton y Williams en 1986) el que revolucionó y reactivó las investigaciones en este sector del *Machine Learning*, que, aunque a posteriori parezca un razonamiento lógico y sencillo, tardó tiempo en desarrollarse.

Gracias a esta idea se recuperó el interés en el campo, acabándose así el llamado "Invierno en la Inteligencia Artificial" en el que el desarrollo de las mismas estuvo detenido, más de 15 años.

La idea base del *Backporpagation* es ir calculando el error hacia atrás, esto es, comparar el valor de salida de la Red con el valor deseado del *training set* e ir reduciendo dichos errores modificando los pesos de cada conexión.

Este proceso se realiza capa por capa, de manera que simplifica el trabajo, ya que, en cada paso, solo se modifican los pesos de una capa, y no todos los pesos de cada camino posible como en el algoritmo hacia delante.

La técnica utilizada para la reducción de errores es la del *Backpropagation*, esto es, calcular las derivadas parciales del Coste en función de los parámetros, es decir, en función de los pesos asignados a la información recibida de cada neurona de la capa anterior y el sesgo. Con esto, seremos capaces de conocer la influencia de cada una de las neuronas en el resultado final a través del vector gradiente, que posteriormente, utilizará el método del Descenso del Gradiente, para el aprendizaje de la Red.

El método del descenso del gradiente es un método muy utilizado en general en *Machine Learning*, mientras que el *Backpropagation* es el que se aplicará de forma más concreta en una Red Neuronal.

2.3 Matemáticas en el método Backpropagation

Como es un método "hacia atrás", se debe comenzar desde la última capa (L):

I. Hallar la derivada del coste respecto a los parámetros en la primera capa:

   a. Respecto a los pesos de cada neurona de la que recibe información:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L}$$

   b. Respecto al sesgo:

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L}$$

   c. Nótese que:

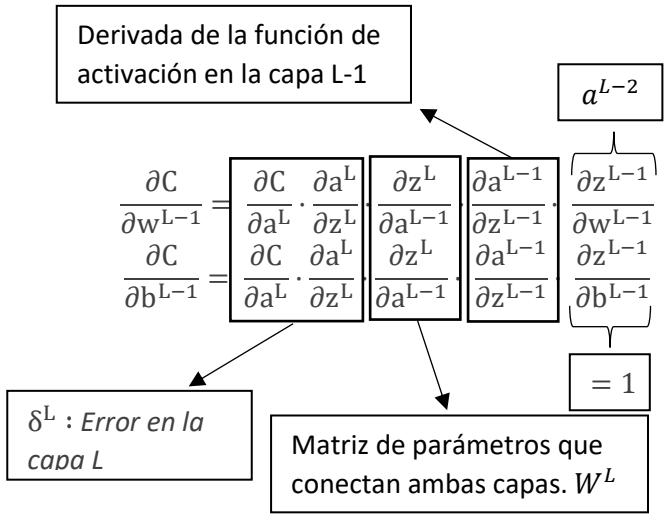
$$\frac{\partial z^L}{\partial w^L} = a_i^{L-1} \; ; \; \frac{\partial z^L}{\partial b^L} = 1$$

Además, el "Error imputado a la neurona", que forma parte de la primera expresión es:

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

que representa cómo influye la neurona en el error final.

II. Hallar la derivada del coste respecto a los parámetros, en el resto de capas(L-1):



Con lo cual, se puede usar la ecuación de forma recurrente, teniendo que calcular solamente una derivada por cada neurona que queramos evaluar:

$$\frac{\partial C}{\partial w^{L-1}} = \delta^L \cdot W^L \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot a^{L-2}$$
$$\frac{\partial C}{\partial b^{L-1}} = \delta^L \cdot W^L \cdot \frac{\partial z^L}{\partial a^{L-1}}$$

A su vez, los tres primeros términos de la ecuación, es decir: el error en la capa L, la matriz de parámetros W^L y la derivada de la función de activación en la capa L-1, conformarían el error en la capa L-1(δ^{L-1})

Por lo tanto, los pasos a seguir en el Backpropagation para obtener el vector gradiente son los siguientes:

1. Computar el error en la última capa:

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

2. Retropropagar el error:

$$\delta^{L-1} = \delta^{L} \cdot W^{L} \cdot \frac{\partial z^{L}}{\partial a^{L-1}}$$

3. Calcular las derivadas de cada capa usando el error retropropagado, y obtener el vector gradiente que posteriormente se utilizará para el método del descenso del gradiente:

$$\frac{\partial C}{\partial b^{L-1}} = \delta^{L-1}$$

$$\frac{\partial C}{\partial w^{L-1}} = \delta^{L-1} \cdot a^{L-2}$$

2.4 Método del descenso del gradiente

Una vez se tiene el vector gradiente, se resta dicho vector al error, tantas veces como sea posible, hasta que dicho error sea mínimo o se supere un número máximo de iteraciones.

En cada neurona, tomo los pesos y el sesgo y le resto el valor correspondiente del vector gradiente para minimizar el error.

Vector gradiente:

$$\nabla f^{L} = \begin{bmatrix} \frac{\partial C}{\partial b^{L}} \\ \frac{\partial C}{\partial w_i^{L}} \end{bmatrix}$$

Vector de parámetros de entrada en dicha neurona:

$$N_i{}^{L} = \begin{bmatrix} b^{L} \\ w_i^{L} \end{bmatrix}$$

Recurrencia para minimizar el error de cada neurona:

$$N_i^{L} = N_i^{L} - \nabla f^{L}$$

Hasta que el error sea menor que una cierta épsilon dada:

$$\mathbf{E} > e_i^L = (a_i^L - y_i^L)^2$$

De esta forma se reduce el error de salida de cada neurona al modificar los valores de entrada en la misma, a partir del vector gradiente previamente calculado con el algoritmo de Backpropagation.

2.5 Ejemplo en Python de una Red Neuronal Simple y sus elementos

A continuación, quiero mostrar un ejemplo en lenguaje Python, que es el que se usa durante todo el trabajo, de una Red Neuronal simple.

En dicha red se podrán ver todos los elementos de la misma como la función de activación y sus derivadas, asignación de pesos, métodos de entrenamiento y reducción de errores etc…

Es un ejemplo bastante ilustrativo para explicar el modelo ya que, las Redes Neuronales se pueden implementar con Python desde librerías predefinidas como Keras y otras herramientas como TensorFlow. El ejemplo a continuación podría reducirse a unas pocas líneas de código, esto es lo que hace a Python un lenguaje muy útil para el desarrollo de Redes Neuronales.

Concretamente, este ejemplo, está implementado en un entorno Jyputer, dentro del paquete Anaconda.

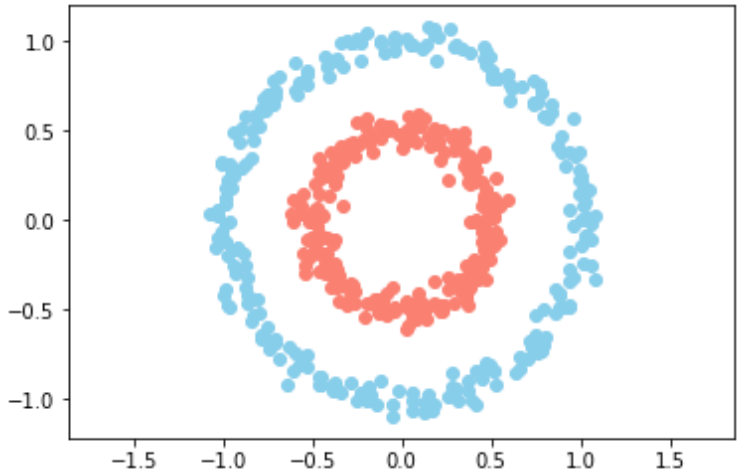
Esta Red toma un conjunto de datos divididos en en dos círculos concéntricos, con más o menos dispersión, dependiendo del ruido que se quiera meter. Este conjunto será el conjunto de entrenamiento.

A continuación, la Red, va desarrollándose y cambiando su umbral en función de los datos para clasificarlos en los dos conjuntos de círculos concéntricos, de tal manera que cuando haya mas datos del mismo tipo (conjunto test), sea capaz de clasificarlos.

Es un ejemplo ilustrativo de cómo una Red Neuronal puede ir cambiano y "entrenandose" para clasificar diferentes conjuntos de datos.

```
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt

from sklearn.datasets import make_circles
```

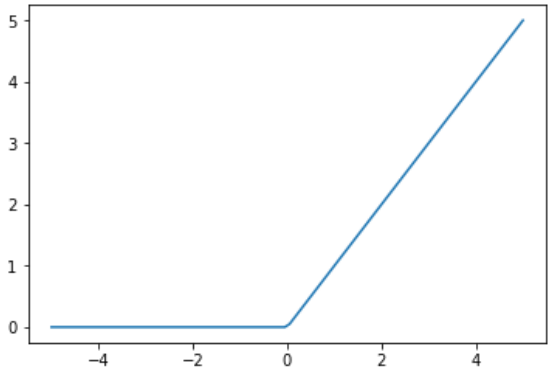
```
# CREAR EL DATASET

n = 500 #número de registros en nuestros datos
p = 2 #número de características de cada entrada de datos

X, Y = make_circles(n_samples=n, factor=0.5, noise=0.05)

Y = Y[:, np.newaxis]

plt.scatter(X[Y[:, 0] == 0, 0], X[Y[:, 0] == 0, 1], c="skyblue")
plt.scatter(X[Y[:, 0] == 1, 0], X[Y[:, 0] == 1, 1], c="salmon")
plt.axis("equal")
plt.show()

#una vez aquí, el objetivo es crear una RN que separe ambos conjun-
tos
```




```
# CLASE DE LA CAPA DE LA RED

class neural_layer():

  def __init__(self, n_conn, n_neur, act_f):
    self.act_f = act_f
    self.b = np.random.rand(1, n_neur)       * 2 - 1
    #para tenerlo  de -1 a 1 y no de 0 a 1
    self.W = np.random.rand(n_conn, n_neur) * 2 - 1
```

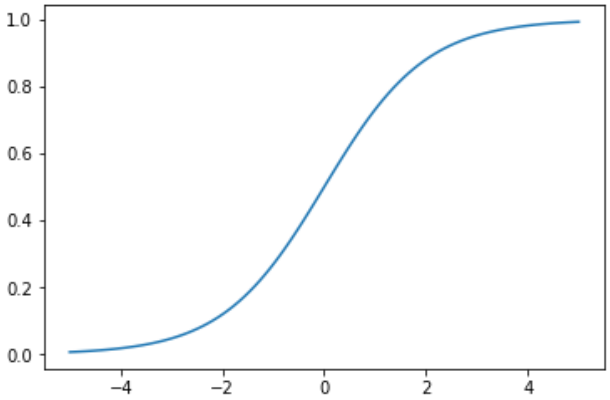
```
# FUNCIONES DE ACTIVACION

sigm = (lambda x: 1 / (1 + np.e ** (-x)),
        lambda x: x * (1 - x))

relu = lambda x: np.maximum(0, x)
```

```
_x = np.linspace(-5, 5, 100)
plt.show()
plt.plot(_x, relu(_x))
plt.show()
plt.plot(_x, sigm[0](_x))
```



<- ReLu

Sigmoide ->

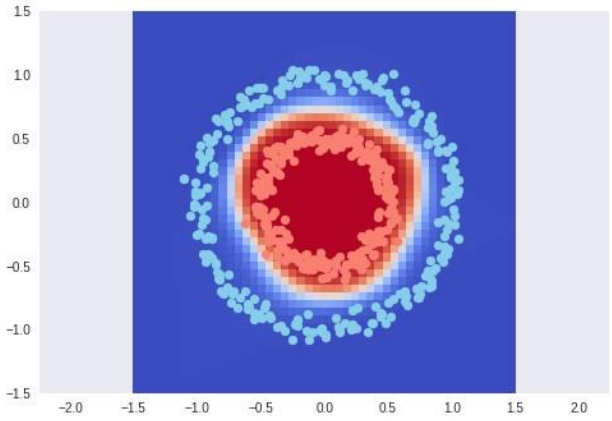



```
# CREAMOS LA RED NEURONAL

l0 = neural_layer(p, 4, sigm)
l1 = neural_layer(4, 8, sigm)
# ...así se podría ir haciendo definiendo capa por capa, pero voy a
hacerlo más elegante

def create_nn(topology, act_f):

  nn = []

  for l, layer in enumerate(topology[:-1]):

    nn.append(neural_layer(topology[l], topology[l+1], act_f))

  return nn
```

```
#  FUNCION DE ENTRENAMIENTO EN 3 PASOS (PRECESAMIENTO HACIA DELANTE
, VALORACION DE ERRORES, PROCESAMIENTO DEL ERROR HACIA DETRAS)

topology = [p, 4, 8, 1]#la última es una sola neurona porque el res
ultado es binario
```

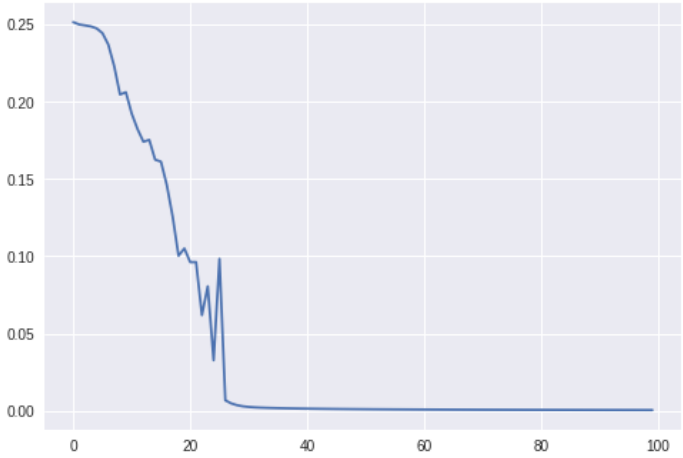
```
neural_net = create_nn(topology, sigm) #ahora mismo tengo la red
neuronal creada, pero sin entrenar, de momento no sirve para nada

l2_cost = (lambda Yp, Yr: np.mean((Yp - Yr) ** 2),
           lambda Yp, Yr: (Yp - Yr))

def train(neural_net, X, Y, l2_cost, lr=0.5, train=True):#ahora
mismo tengo la red neuronal creada, pero sin entrenar, de momento
no sirve para nada

  out = [(None, X)]

  # Forward pass
  for l, layer in enumerate(neural_net):

    z = out[-1][1] @ neural_net[l].W + neural_net[l].b  #suma pond-
erada de la primera capa
    a = neural_net[l].act_f[0](z)

    out.append((z, a))


  if train:    #si train es false, me calcula lo de arriba sólo, el
error, sino activa el entrenamiento

    # Backward pass
    deltas = []

    for l in reversed(range(0, len(neural_net))):

      z = out[l+1][0]
      a = out[l+1][1]

      if l == len(neural_net) - 1:
        #calcular delta de la última capa
        deltas.insert(0, l2_cost[1](a, Y) * neural_net[l].act_f[1](
a))
      else:
        #calcular delta respecto a capa previa
        deltas.insert(0, deltas[0] @ _W.T * neural_net[l].act_f[1](
a))

      _W = neural_net[l].W

      # Gradient descent
      neural_net[l].b = neural_net[l].b - np.mean(deltas[0], axis=0
, keepdims=True) * lr
      neural_net[l].W = neural_net[l].W - out[l][1].T @ deltas[0] *
lr

  return out[-1][1]


train(neural_net, X, Y, l2_cost, 0.5) #ya hemos calculado el error,
es decir, hemos procesado hacia adelante
print("")
```

```
# VISUALIZACIÓN Y TEST

import time
from IPython.display import clear_output

neural_n = create_nn(topology, sigm)
loss = []

for i in range(2500):

  # Entrenemos a la red!
  pY = train(neural_n, X, Y, l2_cost, lr=0.05)

  if i % 25 == 0:

    print(pY)

    loss.append(l2_cost[0](pY, Y))

    res = 50

    _x0 = np.linspace(-1.5, 1.5, res)
    _x1 = np.linspace(-1.5, 1.5, res)

    _Y = np.zeros((res, res))

    for i0, x0 in enumerate(_x0):
      for i1, x1 in enumerate(_x1):
        _Y[i0, i1] = train(neural_n, np.array([[x0, x1]]), Y, l2_co
st, train=False)[0][0]

    plt.pcolormesh(_x0, _x1, _Y, cmap="coolwarm")
    plt.axis("equal")

    plt.scatter(X[Y[:,0] == 0, 0], X[Y[:,0] == 0, 1], c="skyblue")
    plt.scatter(X[Y[:,0] == 1, 0], X[Y[:,0] == 1, 1], c="salmon")

    clear_output(wait=True)
    plt.show()
    plt.plot(range(len(loss)), loss)
    plt.show()
    time.sleep(0.5)
```



<- Conjunto clasificado (fotograma de la última iteración)

Función de pérdida (fotograma de la última iteración) ->



3-. Planteamiento de mis problemas

Como he comentado en la introducción, el propósito de este trabajo es el de comparar el comportamiento y la eficacia de las Redes Neuronales según su función de activación y el número de neuronas por capa. Para ello voy a distinguir dos casos diferentes, cada uno con su propio conjunto de datos, su tipo de Red Neuronal diferente y sus componentes que variaré según se vaya viendo el comportamiento de las mismas.

Caso 1.0

En este primer caso, el conjunto de datos consiste en 70.000 imágenes de números de 0 a 9. Cada una de las imágenes tiene una resolución de 28x28 píxeles y utilizaré como conjunto de entrenamiento 60.000 de dichas imágenes y como conjunto de test, para probar la eficacia de la Red Neuronal según su función de activación, las otras 10.000 imágenes del conjunto.

El problema que se plantea resolver con una Red Neuronal Densa en este caso es, a través de una fotografía de un número escrito a mano, descubrir o predecir qué número es el que se ha escrito a pesar de que, al estar escrito a mano, puede no tener una caligrafía suficientemente clara.

Este problema, aunque a menor escala, es paralelo al de informatizar y mecanografiar de manera automática textos escritos a mano.

Caso 1.1

Análogo al caso 1.0, pero en este caso, cambiando las funciones de activación y poniendo en las tres capas de la Red Neuronal, funciones de activación Sigmoide.

Caso 2.0

En el segundo he tomado como conjunto de datos, las críticas de películas de la página web filmográfica IMDB, dicho conjunto de datos dispone de 25.000 diferentes críticas clasificadas según el sentimiento de dicha crítica en positivo o negativo.

Para mayor comodidad, las palabras se indexan por frecuencia general en el conjunto de datos, de modo que, por ejemplo, el número entero "3" codifica la tercera palabra más frecuente en los datos. Esto permite operaciones de filtrado rápido como: "solo considera las 10 000 palabras más comunes, pero elimina las 20 palabras más comunes". Como convención, "0" no significa una palabra específica, sino que se utiliza para codificar cualquier palabra desconocida.

Para entrenar mi Red Neuronal Convolucional en este caso, usaré las 500 palabras más frecuentes. Tomo las 25000 críticas para entrenar a la Red y su clasificación en positiva o negativa.

Este problema, llevado a una situación real, ayuda a IMDB a clasificar sus películas y puntuarlas según las críticas de los espectadores.

Caso 2.1

Análogo al caso 2.0 pero con una Red Neuronal Densa y no Convolucional. Posteriormente se verá la comparación de resultados.

4-. Resolución y resultados

Caso 1.0

A continuación, muestro el código de mi red neuronal y los resultados paso a paso:

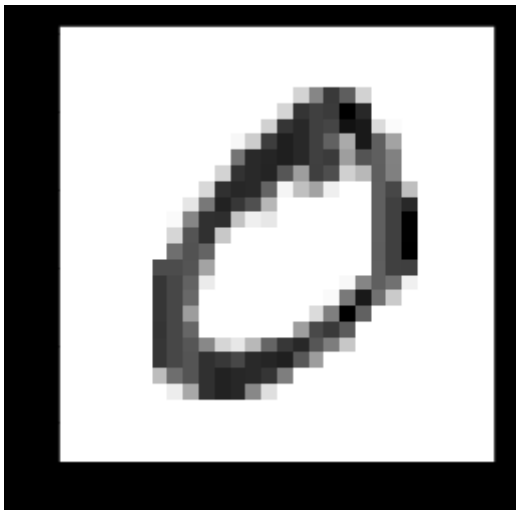
```
import tensorflow as tf
from tensorflow.python.keras.models import save_model

mnist = tf.keras.datasets.mnist #28x28 de resolución en imágenes es
critas a mano de dígitos de 0 a 9
x_train,y_train),(x_test,y_test) = mnist.load_data()
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten()) #crear la capa input
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))
#crear una capa densa de 128 neuronas con función de activacion
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation = tf.nn.sigmoid))
#10 tipos de output sigmoid o softmax

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train,y_train,epochs=3)
# 3 pasadas del conjunto por la red para entrenarla
```

Este sería el primer caso en el que tengo como funciones de activación las ReLu en las dos primeras capas y una sigmoide en la tercera, la capa de salida. Además 128 neuronas en la primera capa y la segunda y 10 en la tercera.

```
Epoch 1/3
60000/60000 [==============================] - 8s 134us/step - loss
: 0.2853 - acc: 0.9193
Epoch 2/3
60000/60000 [==============================] - 7s 111us/step - loss
: 0.1200 - acc: 0.9636
Epoch 3/3
60000/60000 [==============================] - 6s 108us/step - loss
: 0.0832 - acc: 0.9743
```

La red, a la tercera pasada obtiene un porcentaje de acierto superior al 97%, lo cual, es aceptable. Obviamente, dependiendo de los datos que se estén tratando, el porcentaje aceptable varía, pero normalmente siempre se tomará uno superior al 95%.

```
val_loss, val_acc = model.evaluate(x_test,y_test)
print(x_test, y_test)
import matplotlib.pyplot as plt
plt.imshow(x_train[1], cmap = plt.cm.binary)
plt.show()
print(x_train[1])
```



Cada uno de los pixels tiene un valor asignado de 0 (blanco) a 1 (negro), siendo esta imagen, una matriz de 28x28

```
tf.keras.models.save_model(
    model,
    'epic_num_reader.h5',
    overwrite=True,
    include_optimizer=True
)
```

```
import keras
loaded_model = keras.models.load_model('epic_num_reader.h5')
```

```
loaded_model
```

```
predictions = loaded_model.predict([x_test])
```

```
predictions[0]
```

```
array([4.8952447e-08, 3.1080894e-08, 1.5647711e-05, 4.9168413e-04,
       2.7032907e-09, 1.3056349e-07, 5.3248947e-13, 7.2111350e-01,
       1.6750093e-07, 1.4927409e-06], dtype=float32)
```

```
import numpy as np
print (np.argmax(predictions[0]))
```

7

```
plt.imshow(x_test[1])
plt.show()
```

Efectivamente la Red Neuronal ha transcrito bien la primera imagen del conjunto test.

Caso 1.1

```
import tensorflow as tf
from tensorflow.python.keras.models import save_model

mnist = tf.keras.datasets.mnist #28x28 resolution images of hand-wr
itten digits 0-9

(x_train,y_train),(x_test,y_test) = mnist.load_data()
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten()) #crear la capa input
model.add(tf.keras.layers.Dense(128, activation = tf.nn.sigmoid)) #
crear una capa densa de 128 neuronas con funcion de activacion
model.add(tf.keras.layers.Dense(128, activation = tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(10, activation = tf.nn.sigmoid)) #1
0 tipos de output #sigmoid o softmax

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train,y_train,epochs=3)
```

```
Epoch 1/3
60000/60000 [==============================] - 7s 125us/step - loss
: 0.5341 - acc: 0.8542
Epoch 2/3
60000/60000 [==============================] - 7s 118us/step - loss
: 0.2218 - acc: 0.9336
Epoch 3/3
60000/60000 [==============================] - 7s 118us/step - loss
: 0.1663 - acc: 0.9501
```

Caso 2.0

A continuación, muestro el código de mi red neuronal y los resultados paso a paso:

```
# CNN for the IMDB problem
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

En el recuadro anterior, he cargado las herramientas necesarias de las librerías que voy a usar.

Ahora, voy a cargar el conjunto de datos que voy a usar para entrenar la Red, que son 25.000 críticas con sus respectivos 1 o 0 según sean positivas o negativas. También cargo el conjunto de datos que voy a usar para validar que la Red Neuronal Convolucional esté bien entrenada o no, dicho conjunto tiene las mismas dimensiones que el conjunto de entrenamiento, aunque, obviamente, es diferente.

```
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top
_words)
# pad dataset to a maximum review length in words
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
```

A continuación, creo el modelo con las diferentes capas de la Red, el tipo de Red, el tipo de convolución, el número de neuronas y capas y las funciones de activación.

Y posteriormente lo entreno.

```
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activat
ion='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics
=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 500, 32)           3104
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 250, 32)           0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 250)               2000250
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 251
=================================================================
Total params: 2,163,605
Trainable params: 2,163,605
Non-trainable params: 0
```

Una vez entrenado el modelo, lo aplico al conjunto de test para ver si es un buen modelo o no, mostrando como parámetros indicadores la función de pérdida y la precisión con la que el modelo predice los resultados.

```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2, batch_size=128, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/2
 - 42s - loss: 0.4798 - acc: 0.7374 - val_loss: 0.2802 - val_acc: 0.8848
Epoch 2/2
 - 40s - loss: 0.2233 - acc: 0.9114 - val_loss: 0.2734 - val_acc: 0.8856
Accuracy: 88.56%
```

Ahora saco la prediccion sobre la primera crítica, para ver si de verdad coincide con la realidad o no. Veo que ha tenido un valor de 0.2 en una escala de 0 a 1, lo que quiere decir que lo clasifica como una crítica negativa.

```
predictions = model.predict(X_test)
predictions[0]
```

```
array([0.20142166], dtype=float32)
```

Quiero mostrar, el texto y el vector de la segunda crítica.

```
rounded = [round(x[0]) for x in predictions]
rounded_indexed = {k: v for k,v in enumerate(rounded)}
index_from=3
word_to_id = imdb.get_word_index()
word_to_id = {k:(v+index_from) for k,v in word_to_id.items()}
word_to_id["<PAD>"] = 0
```

```
word_to_id["<START>"] = 1
word_to_id["<UNK>"] = 2
id_to_word = {value:key for key,value in word_to_id.items()}

item=1
print('\npredicted: '+str(rounded_indexed[item])+' actual: '+str(y_
test[item]))
sentence = ' '.join(id_to_word[id] for id in X_test[item])
sentence = sentence.replace('<PAD> ','')
sentence = sentence.replace('<START> ', '')
sentence = sentence.replace('<UNK> ', '?')
print('sentence: '+sentence)
print('vector: '+str(X_test[item]))
print(X_test[item].shape)
```

```
predicted: 1.0 actual: 1
sentence: this film requires a lot of ?because it focuses on mood a
nd character development the plot is very simple and many of the sc
enes take place on the same set in ??the ?dennis character apartmen
t but the film builds to a disturbing climax br br the characters c
reate an atmosphere ?with sexual tension and psychological ?it's ve
ry interesting that robert ?directed this considering the style and
structure of his other films still the ??audio style is evident her
e and there i think what really makes this film work is the brillia
nt performance by ?dennis it's definitely one of her darker charact
ers but she plays it so perfectly and convincingly that it's scary
michael burns does a good job as the ?young man regular ?player mic
hael murphy has a small part the ?moody set fits the content of the
story very well in short this movie is a powerful study of ?sexual
?and desperation be patient ?up the atmosphere and pay attention to
the wonderfully written script br br i praise robert ?this is one o
f his many films that deals with ?fascinating subject matter this f
ilm is disturbing but it's sincere and it's sure to ?a strong emoti
onal response from the viewer if you want to see an unusual film so
me might even say bizarre this is worth the time br br unfortunatel
y it's very difficult to find in video ?you may have to buy it off
the internet
vector: [ 0,0 … 0,0,1,14,22,3443,6,176,7,2,88,12,2679,23,1310,5,109
,943,4,114,9,55,606,5,111,7,4,139,193,273,23,4,172,270,11,2,2,4,2,2
,801,109,1603,21,4,22,3861,8,6,1193,1330,10,10,4,105,987,35,841,2,1
9,861,1074,5,1987,2,45,55,221,15,670,2,526,14,1069,4,405,5,2438,7,
27,85 108  131    4    2    2 3884  405    9 3523  133    5   50
13  104   51   66  166   14   22  157    9    4  530  239   34    2
2801   45  407   31    7   41 3778  105   21   59  299   12   38  9
50    5 4521   15   45  629  488 2733  127    6   52  292   17    4
2  185  132 1988    2 1799  488 2693   47    6  392  173    4    2
4378  270 2352    4 1500    7    4   65   55   73   11  346   14
20    9    6  976 2078    7    2  861    2    5 4182   30 3127    2
56    4  841    5  990  692  8    4 1669  398  229   10   10   13 2
822  670    2   14    9   31    7   27  111  108   15 2033   19
2 1429  875  551   14   22    9 1193   21   45 4829    5   45  252
8    2    6  565  921 3639   39    4  529   48   25  181    8   67
35 1732   22   49  238   60  135 1162   14    9  290    4   58   10
10  472   45   55  878    8  169   11  374    2   25  203   28    8
818   12  125    4 3077]   (500,)
```

Caso 2.1

```
# MLP for the IMDB problem
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

```
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top
_words)
```

```
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
```

```
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics
=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
flatten_1 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               4000250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 4,160,501
Trainable params: 4,160,501
Non-trainable params: 0
_________________________________________________________________
None
```

```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3, batch_size=128, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/3
 - 34s - loss: 0.5086 - acc: 0.7129 - val_loss: 0.3330 - val_acc: 0.8530
Epoch 2/3
 - 32s - loss: 0.1909 - acc: 0.9270 - val_loss: 0.3002 - val_acc: 0.8734
Epoch 3/3
 - 33s - loss: 0.0599 - acc: 0.9826 - val_loss: 0.4145 - val_acc: 0.8614
Accuracy: 86.14%
```

5-. Conclusiones

Caso 1.0

En este primer caso se puede ver que, al utilizar una función de activación ReLu en las dos primeras capas y Sigmoide en la tercera, se obtienen unos resultados eficaces, ya que el porcentaje de acierto es alto, 97% pero no es el ideal.

Caso 1.1

En este caso, con funciones de activación Sigmoide en todas las capas de la Red, se ve que el porcentaje de acierto de la Red Neuronal, es menor que el anterior, de un 95.01%, lo que quiere decir, que en este caso, funciona mejor la función de activación ReLu, que, a pesar de ser más sencilla y a priori parecer que pueda dar peor resultado, no es así.

Caso 2.0

Vistos los resultados de test del segundo caso, llego a la conclusión de que es un conjunto de datos, que, puede ser supervisado o resuelto en primera instancia por una Red Neuronal, pero nunca definitivamente ya que el nivel de acierto del 88.56%.

Caso 2.1

El propósito de utilizar una Red Neuronal no Convolucional para este mismo problema era el de ver que no siempre las Redes Neuronales son buenas sólo para el tratamiento de imágenes. En este caso, se ve que también es mejor para el conjunto de críticas a películas de IMDB, ya que, al realizar el mismo problema con una Red Neuronal Densa, el resultado es peor, con un 86.14%.

6-. Bibliografía

Andrew Ng (2012), https://es.coursera.org/specializations/deep-learning

- Sequence model
- Neural network and Deep learning

Tensorflow, https://playground.tensorflow.org/
Keras, https://keras.io/

Canal de Youtube "DotCSV" (2017),
https://www.youtube.com/channel/UCy5znSnfMsDwaLlROnZ7Qbg/featured

Páginas web consultadas:

- https://github.com/kenophobio/keras-example-notebook/blob/master/Vanilla%2BCNN%2Bon%2BMNIST%2Bdataset.ipynb
- http://www.clubdetecnologia.net/blog/2017/python-como-construir-una-red-neuronal-simple/
- http://www.clubdetecnologia.net/blog/2017/python-como-construir-una-red-neuronal-simple/
- https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6
- https://es.wikipedia.org/wiki/Red_neuronal_artificial
- https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
- https://keras.io/optimizers/
- https://medium.com/@nicolas_19145/state-of-the-art-in-compressing-deep-convolutional-neural-networks-cfd8c5404f22
- https://www.nature.com/articles/s41598-018-24271-9#ref-CR14
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- http://blog.echen.me/2017/05/30/exploring-lstms/
- http://colah.github.io/posts/2014-07-Conv-Nets-Modular/
- http://www.diegocalvo.es/red-neuronal-convolucional-cnn/
- https://inteligenciartificiallmca.wordpress.com/2017/06/10/1-3-estado-del-arte/
- https://medium.com/@nicolas_19145/state-of-the-art-in-compressing-deep-convolutional-neural-networks-cfd8c5404f22
- http://ikuz.eu/2016/01/20/deep-learning-theory-history-state-of-the-art-practical-tools/
- https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research
- https://visualgenome.org/api/v0/api_home.html
- http://yann.lecun.com/exdb/mnist/
- http://help.sentiment140.com/for-students
- https://archive.ics.uci.edu/ml/datasets/sms+spam+collection
- http://archive.ics.uci.edu/ml/datasets/Car+Evaluation

- http://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Drugs.com%29
- https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/
- https://machinelearningmastery.com/lstms-with-python/
- https://www.deeplearningbook.org/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://www.quora.com/What-are-hyperparameters-in-machine-learning
- http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
- https://www.tensorflow.org/api_docs/python/tf/nn