Función de error y de optimización en RNAs

U-TAD

Introducción a la función de error

Hemos visto todo el proceso por el cual esas entradas van fluyendo hacia delante (forward propagation) hasta que obtenemos una predicción.

La información va pasando por las distintas neuronas de cada capa hasta que llegan a la última (output layer), y de aquí, obtendremos una predicción.

El Forward Propagation nos servía para dos cosas fundamentales:

La primera de ellas era el proceso de entrenamiento. Y luego el resultado, en base a una función de error o de coste y una función de optimización, íbamos modificando y actualizando el valor de los pesos o parámetros del modelo, hasta encontrar el valor óptimo que se adapta lo mejor posible a la tendencia o a las etiquetas que teníamos en nuestro conjunto de datos.

Y la segunda casuística era, una vez que ya tenemos esos valores óptimos para los pesos, entonces utilizamos Forward Propagation para que, cuando nos llega un nuevo ejemplo, una nueva instancia para la que queremos realizar una predicción, pasa por todas las neuronas hasta que obtenemos la predicción.

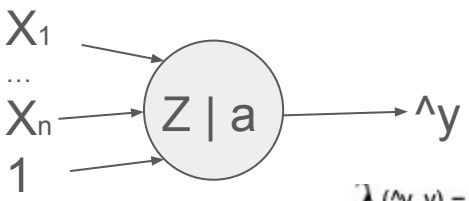
Introducción a la función de error

Nos centramos ahora en la primera casuística, es decir, una vez que tenemos nuestro conjunto de datos de experiencia pasada y tenemos ese valor aleatorio con el que hemos inicializado los pesos o los parámetros del modelo, comienza el proceso de entrenamiento. A través del cual, vamos a utilizar nuestro conjunto de datos para encontrar el valor óptimo, el valor adecuado de esos pesos para que nuestra red neuronal realice buenas predicciones en base a ese conjunto de datos de experiencia pasada, ese conjunto de datos de entrenamiento.

A partir de aquí realizaremos Forward Propagation obteniendo una predicción para cada uno de los ejemplos de nuestro conjunto de datos de entrenamiento y, después, tendremos de alguna manera que comparar esa predicción de nuestra red neuronal con la etiqueta que tenemos en nuestro conjunto de datos. Y, en base a ese resultado, si es una buena o mala predicción, se ajusta el valor de los parámetros de una manera u otra.

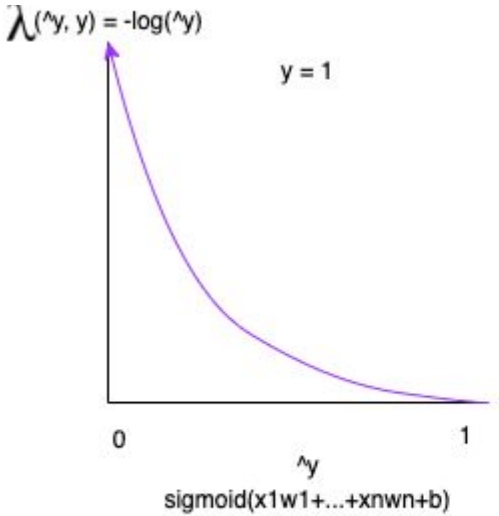
La función que va a realizar esa comparación entre el resultado de nuestra red neuronal (predicción) y la etiqueta que tenemos en nuestro conjunto de datos es la función de error o función de coste.

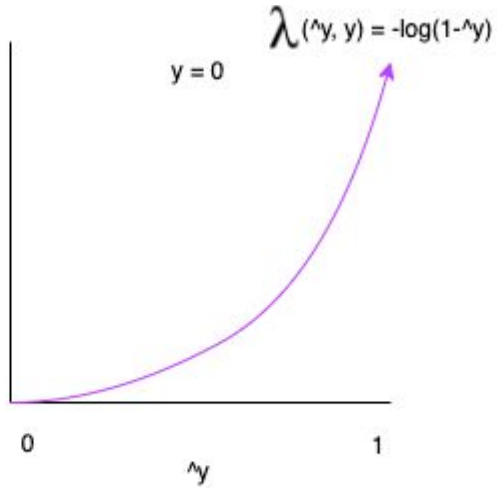
Entendiendo la función de error



λ(^y, y) = núm. alto si ^y != y ó núm. bajo si ^y == y

*Es como el algoritmo de regresión logística





Si la etuiqueta correcta 'y' es 0, entonces la función de coste será 0 si el resultado de la función hipótesis también es 0. Si el resultado de la función hipótiesis se acerca a 1, entonces la función de coste se aproximará al infinito. Lo mismo ocurre si 'y' es 1

Función de error final

y = 1, λ(^y, y) = -log(^y)

y = 0, λ(^y, y) = -log(1-^y)

λ(^y, y) = -(ylog(^y) + (1-y)log(1^y))

λ(^y, y) = -(0log(^y) + (1-0)log(1-^y))

= -log(1-^y)

Con una sola TLU: -log(^y) = -log(sigmoid(X_1W_1+...+X_nW_n+ b))

X_1	X_2	…	X_n	y
1	2	…	3	0
0	10	…	11	1
…	…	…	…	…
$X_{1(m)}$	$X_{2(m)}$	…	$X_{1(m)}$	$y_{(m)}$

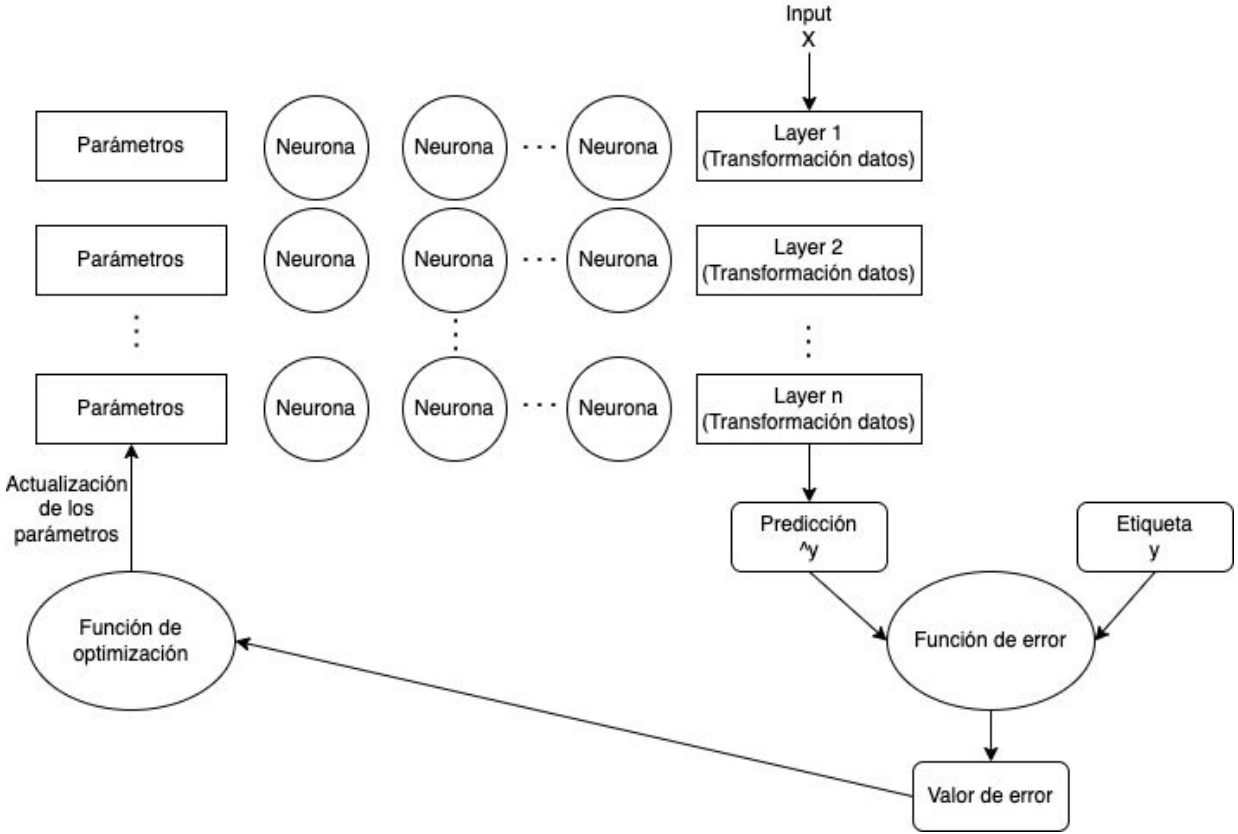
Ahora incorporando los m ejemplos (error global):

J(W, B) =1/m $\sum\lambda$(^$y_{(i)}$, $y_{(i)}$) =

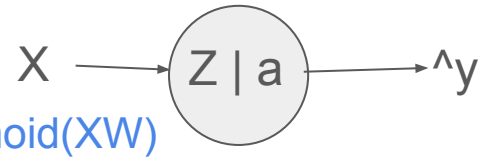
= -1/m$\sum$ $y_{(i)}$log(^$y_{(i)}$) + (1-$y_{(i)}$)log(1-^$y_{(i)}$)

m = # ejemplos del conj. de datos
n = # input features

Introducción a la función de optimización



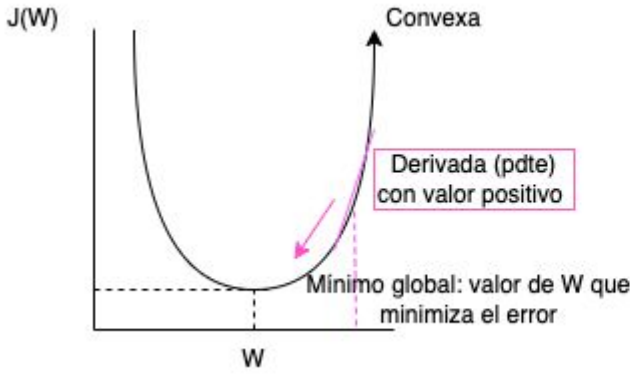
Entendiendo la función de optimización



^y=sigmoid(XW)

Gradient Descent

Función de coste: J(W,B) = = **-1/m $\sum$ y(i)log(^y(i)) + (1-y(i))log(1-^y(i))** (Cross entropy loss)



Inicializa los parámetros aleatoriamente (n=cte)

$W = W - \eta (dJ / dW) = W - \eta (x(\hat{y} - y))$

$W = W - \eta$ (número positivo).
Por lo que iría acercándose al 0, hasta que W = W

De manera análoga para pendientes negativas

Función de optimización final (gradient descent)

$J=0, dw_1 = 0, dw_2 = 0, db = 0$

for i = 1 to m:

$\hat{y}(i) = \text{sigmoid}(X_{1(i)}W_1 + X_{2(i)}W_2 + b)$ (forward prop.)

$J = J + \lambda(\hat{y}, y) = J + (-(y_{(i)}\log(\hat{y}_{(i)} + (1-y_{(i)})\log(1-\hat{y}_{(i)}))$

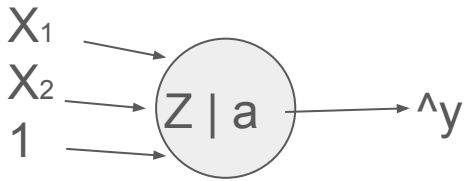
$dW_1 = dW_1 + dJ/dW_1 = dW_1 + X_{1(i)}(\hat{y}_{(i)} - y_{(i)})$

$dW_2 = dW_2 + dJ/dW_2 = dW_2 + X_{2(i)}(\hat{y}_{(i)} - y_{(i)})$

$db = db + dJ/db = db + (\hat{y}_{(i)} - y_{(i)})$

$J=J/m, dW_1=dW_1/m, dw_2=dw_2/m, db=db/m$

$W_1=W_1-ndW_1, W_2=W_2-ndW_2, b=b-ndb$ (n = cte.) //Se repite hasta que W_i y b no varían



X_1	X_2	y
$X_{1(1)}$	$X_{2(1)}$	$y_{(1)}$
$X_{1(2)}$	$X_{2(2)}$	$y_{(2)}$
…	…	…
$X_{1(m)}$	$X_{2(m)}$	$y_{(3)}$

m = # ejemplos

Caso Práctico: Clasificación de imágenes de dígitos

Clasificación de imágenes con el Perceptrón Multicapa.ipynb