Retropropagación a un paso

La retropropagación nos brinda un mecanismo para entrenar redes de más de una capa. Su funcionamiento traslada el error en la predicción hacia atŕas usando la derivada parcial de dicho error con respecto de cada uno de los pesos. Es decir,

$$\frac{\partial E}{\partial w_{ho}} = \frac{\partial E}{\partial h_o}\frac{\partial h_o}{\partial w_{ho}} = \frac{\partial E}{\partial h_o} y_h$$

En este primer programa implementaremos el algoritmo de retropropagación para determinar los incrementos que debe tener cada capa de la red. En un siguiente ejercicio realizaremos todo el proceso de retropropagación.

Open in Colab

Alumno: Trejo Arriaga Rodrigo Gerardo

```
#importamos paquetes
import numpy as np


# definimos funciones útiles

def sigmoid_prime(x):
    return sigmoid(x)*(1-sigmoid(x))

def sigmoid(x):
    """
    Calculate sigmoid
    """
    return 1 / (1 + np.exp(-x))



# Definamos una arquitectura de red 3x2x1

x = np.array([0.5, 0.1, -0.2])
target = 0.6
learnrate = 0.5

weights_input_hidden = np.array([[0.5, -0.6],
                                 [0.1, -0.2],
                                 [0.1, 0.7]])

weights_hidden_output = np.array([0.1, -0.3])
```

Pase frontal

Definimos el comportamiento de la red en el pase frontal.

$$\hat{y} = f(WX)$$

```
hidden_layer_input = np.dot(x, weights_input_hidden)
hidden_layer_output = sigmoid(hidden_layer_input)
output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)
output = sigmoid(output_layer_in)
```

Pase hacia atrás

Calcula el cambio que deben de tener los pesos de acuerdo al error. Recordemos que para la capa de salida el término de error que se aplica es:

$$\delta_o = (y - \hat{y}) f'(h)$$

Y para la capa oculta:

$$\delta_h = \delta_o w_{h,o} f'(h_h)$$

este último término de error es diferente para cada neurona con índice h de la capa oculta. Es decir tendremos tantas h como neuronas tenga la capa oculta. Nota que aunque el ejercicio se puede resolver usando vectores e índices, lo implementaremos usando notación matricial.

```
## Backwards pass
error = target - output
del_err_output = error * sigmoid_prime(output_layer_in)
del_err_hidden = del_err_output * np.multiply(weights_hidden_output, sigmoid_pr
```

Ya que sabemos los términos de error que aplican en cada capa, procedemos a determinar los incrementos en cada capa.

$$\Delta w_{h,o} = \Delta w_{h,o} + \delta_o \hat{y}_h$$

$$\Delta w_{i,h} = \Delta w_{i,h} + \delta_h x_i$$

```
delta_w_h_o = learnrate * del_err_output * hidden_layer_output
delta_w_i_h = learnrate * np.outer(x, del_err_hidden)
```

```
print('Incremento de los pesos oculta a salida:')
print(delta_w_h_o)

print('Incremento de los pesos de entrada a oculta:')
print(delta_w_i_h)
```

```
Incremento de los pesos oculta a salida:
[0.00804047 0.00555918]
Incremento de los pesos de entrada a oculta:
[[ 1.77005547e-04 -5.11178506e-04]
 [ 3.54011093e-05 -1.02235701e-04]
```

```
[-7.08022187e-05  2.04471402e-04]]
```

Conclusión

La implementación de la retropropagación en esta práctica proporciona una comprensión detallada de los fundamentos del aprendizaje en redes neuronales. Este ejercicio manual de ajuste de pesos mediante el gradiente del error ofrece una ventana hacia la mecánica del aprendizaje automático y subraya la importancia de una correcta elección de la tasa de aprendizaje. Al dominar estos principios básicos, se establecen las bases para la exploración avanzada en inteligencia artificial, preparando el terreno para la utilización de herramientas automatizadas que manejan estas operaciones a gran escala.