Algoritmo backpropagation

August 17, 2019

1 Diseño de una red neuronal en python desde cero

1.1 Vamos a definir una red neuronal de 3 capas:

- La capa de entrada va a tener 2 neuronas.
- La capa oculta va a tener 2 neuronas.
- La capa de salida va a tener 1 neurona.

1.1.1 La función de activación de todas las neuronas va a ser la función sigmoide:

$$\text{sigmoide}\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \frac{1}{1 + \mathrm{e}^{-\sum_{i=1}^{n} w_i \cdot x_i}}.$$

El coeficiente de aprendizaje c va a ser de 1 para todas las neuronas.

```
In [1]: from IPython.display import IFrame
        IFrame("./ejemplo_red.pdf", width=800, height=400)

Out[1]: <IPython.lib.display.IFrame at 0x10854d0f0>
```

1.1.2 Importamos los paquetes necesarios:

```
In [2]: import numpy as np

In [4]: c = 1
        xe = np.array([[1, 0, 1]])
        yd = np.array([[1]])
        w1 = np.array([[2, 1],[-2, 3],[0, -1]])
        w2 = np.array([[3, 3],[2, 2]])
        w3 = np.array([[1], [1]])
```

1.1.3 Cálculo de las salidas de la primera capa:

1. Primero calculamos las activaciones: $a_i^{(1)} = \sum_{i=1}^{n} w_i \cdot x_i$ para el primer ejemplo de entrenamiento, la activación de la neurona 1 de la capa 1:

$$\mathrm{a}^{(1)} = \mathrm{x_e} \cdot \mathrm{w}^{(1)} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ -2 & 3 \\ 0 & -1 \end{bmatrix} = [2, 0]\,.$$

1.1.4 Propagamos hacia adelante:

```
In [5]: a1=xe.dot(w1)
        print(a1)
```

```
[[2 0]]
```

1.1.5 Para calcular las salidas de las neuronas a las activaciones les tenemos que aplicar la función sigmoide:

$$\text{sigmoide}\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \frac{1}{1 + e^{-\sum_{i=1}^{n} w_i \cdot x_i}}.$$

```
In [6]: y1 = 1/(1+np.exp(-a1))
        print(y1)
```

```
[[0.88079708 0.5       ]]
```

1.1.6 Repetimos el procedimiento para la capa intermedia y para la capa de salida:

```
In [7]: a2 = y1.dot(w2)
        print(a2)
```

```
[[3.64239123 3.64239123]]
```

```
In [8]: y2 = 1/(1+np.exp(-a2))
        print(y2)
```

```
[[0.97447875 0.97447875]]
```

```
In [9]: a3 = y2.dot(w3)
        print(a3)
```

```
[[1.9489575]]
```

```
In [10]: y3 = 1/(1+np.exp(-a3))
         print(y3)
```

```
[[0.87533292]]
```

1.1.7 Algoritmo backpropagation:

1. Primero hay que calcular el δ de cada una de las neuronas de salida: $\delta = y \cdot (1 - y) \cdot (y_d - y)$
2. Calculamos los δ's de las neuronas de la capa intermedia y de entrada haciendo una propagación hacia atrás (backpropagation) con los pesos de cada una de las neuronas.
3. Actualizamos los pesos.

```
In [11]: delta_s = y3*(1-y3)*(yd-y3)
         print(delta_s)
```

```
[[0.01360432]]
```

1.1.8 Ahora propagamos los δ's de salida hacia atrás:

$$\delta_i^{(j)} = y_i^{(j)} \cdot \left(1 - y_i^{(j)}\right) \cdot \sum_l w_l^{(k+1)} \cdot \delta_l^{(k+1)}.$$

1.1.9 En el caso de la capa de salida, siendo Δ el vector que contiene los δ's de salida:

$$\sum_l w_l^{(j+1)} \cdot \delta_l^{(j+1)} = \Delta \cdot \mathrm{W}^{(\mathrm{j}+1)}.$$

```
In [12]: delta_2=delta_s*w3
         print(delta_2)
```

```
[[0.01360432]
 [0.01360432]]
```

```
In [13]: delta_1 = delta_2.T.dot(w2)
         print(delta_1)
```

```
[[0.0680216 0.0680216]]
```

1.1.10 Actualizamos los valores de los pesos:

$$\mathrm{W}_i^{(k)} = \mathrm{W}_i^{(k)} + c_i^{(k)} \cdot \delta_i^k \cdot \mathrm{Y}^{(\mathrm{k}-1)}.$$

```
In [14]: c=1
```

1.1.11 Capa de salida:

```
In [15]: w3 = w3+c*delta_s*y2.T
         print(w3)
```

```
[[1.01325712]
 [1.01325712]]
```

1.1.12 Capa intermedia:

```
In [16]: w2 = w2+c*delta_2*y1.T
         print(w2)
```

```
[[3.01198264 3.01198264]
 [2.00680216 2.00680216]]
```

1.1.13 Capa de entrada:

```
In [17]: w1 = w1+c*(delta_2*xe).T
         print(w1)
```

```
[[ 2.01360432  1.01360432]
 [-2.          3.        ]
 [ 0.01360432 -0.98639568]]
```

```
In [ ]:
```