Redes neuronales

Aure

December 24, 2021

1 Red Neuronal

Se conoce como red neuronal a un conjunto de capas de neuronas (variables con valores en $[0,1]$) conectadas mediante pesos (coeficientes de ponderación). El valor de las neuronas de la primera capa es el valor del input del programa. Los valores de las neuronas de las siguientes capas provienen de las capas anteriores, su valores son una suma ponderada por los pesos de las neuronas de la capa anterior evaluada en una función de manera que el valor vuelve a estar en $[0,1]$. Además cada neurona tendrá un umbral de activación para controlar donde se quiere establecer el "cero" de la activación.

Finalmente el programa devolverá un output que serán los valores de la última capa de neuronas. Si los valores no son como se esperan entonces tras cada prueba deberán cambiarse los pesos y umbrales. De manera que "aprender" en realidad es encontrar pesos y umbrales más adecuados.

El resultado final será un programa que se auto-refina esos parámetros automáticamente a través de ejemplos (entrenos) mediante ensayo-error.

1.1 Activaciones de las neuronas, función sigmoide

La red tendrá c capas y cada capa tendrá n_k neuronas $1 \leq k \leq c$. Cada neurona está conectada con cada una de las neuronas de la siguiente capa.
A la i-ésima neurona $1 \leq i \leq n_k$ de la capa k la denotaremos por $a_i^{(k)}$. El umbral o peso de dicha neurona la denotaremos por $u_i^{(k)}$. (A veces se denota como $b_i^{(k)}$ por el inglés bias).
La conexión de la i-ésima neurona $1 \leq i \leq n_k$ de la capa k con la neurona j-ésima $1 \leq j \leq n_{k+1}$ de a capa $k+1$ la denotaremos por $\omega_{ji}^{(k)}$.
Todos estos parámetros (pesos y umbrales) se inicializarán con valores aleatorios comprendidos entre 0 y 1.

Consideremos la siguiente función, conocida como sigmoide

$$f(x) = \frac{1}{1+e^{-x}} \in (0,1)$$

que cumple que

$$f'(x) = f(x) \cdot (1 - f(x))$$

Se dice que ahora se usa mejor la función ReLU(x) =max$(0, x)$.
El output de la capa $k = 1$ es simplemente el input del programa y para $k > 1$ tendremos que

$$a_i^{(k)} = f(\sum_{j=1}^{n_{k-1}} w_{ij}^{(k-1)} a_j^{(k-1)} + u_i^{(k-1)})$$

Por lo general podemos pensarlo como
creo k me he colado con la capa de las ω,deberia ser k y no k-1

$$\begin{pmatrix} a_1^{(k)} \\ a_2^{(k)} \\ \vdots \\ a_{n_k}^{(k)} \end{pmatrix} = f\left(\begin{pmatrix} w_{11}^{(k-1)} & w_{12}^{(k-1)} \cdots w_{1n_{k-1}}^{(k-1)} \\ w_{21}^{(k-1)} & w_{22}^{(k-1)} \cdots w_{2n_{k-1}}^{(k-1)} \\ \vdots & \ddots \\ w_{n_k1}^{(k-1)} & w_{n_k2}^{(k-1)} \cdots w_{n_kn_{k-1}}^{(k-1)} \end{pmatrix} \begin{pmatrix} a_1^{(k-1)} \\ a_2^{(k-1)} \\ \vdots \\ a_{n_{k-1}}^{(k-1)} \end{pmatrix} + \begin{pmatrix} u_1^{(k-1)} \\ u_2^{(k-1)} \\ \vdots \\ u_{n_{k-1}}^{(k-1)} \end{pmatrix} \right)$$

y por lo tanto en forma matricial nos quedaría

$$a^{(k)} = f(\omega^{(k-1)} a^{(k-1)} + b^{(k-1)})$$

Observamos que aquí queda justificado el orden de los subíndices ji de la definición de los pesos, ya que si no la matriz anterior tendría que estar traspuesta.

1.2 Última capa de neuronas, función coste y descenso del gradiente

Sea c el número de capas. Los valores de $a_i^{(c)}$ de las n_c neuronas de la última capa son los outputs del algoritmo. A estos outputs también les llamaremos y_l, $1 \leq l \leq n_c$.
A la práctica, en función de los valores de y_l se tendrán que interpretar del alguna manera los resultados, como por ejemplo escogiendo el valor mas grande los y_l o lo que sea.

1.2.1 Función coste

Lo más seguro es que los valores obtenidos y_l no serán exactamente los deseados y, sin embargo, pueden estar muy "cerca" de lo que esperamos.
Cuantificaremos "el error" del algoritmo mediante lo que se conoce como una función de coste, que lo normal es escoger

$$C = E(y) = \frac{1}{2} \sum (y_i - s_i)^2$$

donde s_i son las soluciones/valores deseados al ejemplo de entreno que se esté usando. Observamos que tiene sentido usar esta función como función de coste

ya que equivalente a calcular la mitad de la distancia eucliciana entre dos puntos.
El objetivo será encontrar valores para los pesos y umbrales que minimicen esta función C.

1.2.2 Algoritmo del descenso del gradiente

Inicialmente los pesos y umbrales estarán inicializados con valores aleatorios entre 0 y 1. Para encontrar valores adecuados para estos parámetros lo que se hace es tener "ejemplos/problemas" resueltos que usaremos como entrenamiento. Tras cada resultado de cada entreno trataremos de encontrar nuevos parámetros que minimicen C.

El error/coste del algoritmo tiene que pensarse como una función C que depende de los parámetros ω_i y u_i.
Esa función C dependerá de demasiadas variables como para resolver $\nabla C = 0$. De hecho no sería ni siquiera lo ideal ya que tras cada entreno seguiremos corrigiendo los pesos etc.

La fórmula recursiva que usaremos es

$$\omega_i := \omega_i - \alpha \frac{\partial f}{\partial \omega_i}$$

cada una de los pesos ω_i se modificará de esta forma. El signo $-$ es debido a que un mínimo relativo de una función C esta en el sentido contrario del gradiente ∇C y el coeficiente real α es debido a que si no puede entrar en bucle el algoritmo o que la corrección puede ser demasiado brusca/ridícula, únicamente nos importa que ω_i se modifique en el sentido $-\frac{\partial f}{\partial \omega_i}$.

A la práctica lo que haremos será minimizar media de los errores de unas cuantos ejemplos y modificaremos los pesos y umbrales en función de esa media.
Esto se conoce como descenso del gradiente estocástico. La fórmula del coste quedaría así

$$C = \frac{1}{n} \sum_x C_x$$

donde f_i es el error de caca uno de los n pruebas.

1.3 Algoritmo Backpropagation

Por lo general la red neuronal puede tener en total miles de neuronas, pesos y umbrales por lo que calcular ∇C puede ser demasiado costoso. Hay que tener en cuenta que por ejemplo el peso $\omega_{11}^{(1)}$ influye en los valores de todas las siguientes capas y por lo tanto calcular $\frac{\partial C}{\partial \omega_{11}^{(1)}}$ mediante la regla de la cadena (es decir, como

suma de cada uno de los errores por cada una de "los caminos") es demasiado costoso.
El algoritmo Backpropagation (retroalimentación o retropropagación) es un algoritmo que permite calcular las derivadas parciales respecto los pesos/umbrales de la capa k en función de las de la capa $k+1$, yendo así desde el final hasta atrás hasta el principio.

Para agilizar la notación, denotaremos

$$z_i^{(k)} = \sum_{j=1}^{n_k} \omega_{ji}^{(k)} a^{(k)} + u_j^{(k)}$$

para $1 \leq k \leq c-1$, de manera que simplemente $a_i^{(k+1)} = f(z_i^{(k)})$ y en forma matricial tendremos

$$z^{(k)} = \omega^{(k)} a^{(k)} + b^{(k)} \qquad a^{(k+1)} = f(z^{(k)})$$

observamos que para $z_i^{(k)}$ tenemos que $1 \leq i \leq n_{k+1}$.
Denotaremos el error de $z_j^{(k)}$ como

$$\delta_j^{(k)} = \frac{\partial C}{\partial z_j^{(k)}}$$

para $1 \leq k \leq c-1$, $1 \leq j \leq n_{k+1}$, y como siempre denotaremos por $\delta^{(k)}$ el vector de errores asociados con la capa k.
Observamos que el error $\delta_j^{(c-1)}$ de una red neuronal con c capas cumple

$$\delta_j^{(c-1)} = \frac{\partial C}{\partial a_j^{(c)}} \frac{\partial a_j^{(c)}}{\partial z_j^{(c-1)}} = (y_j - s_j) \cdot f'(z_j^{(c-1)})$$

En notación matricial tendremos

$$\delta^{(c-1)} = \begin{pmatrix} \frac{\partial C}{\partial a_1^{(c)}} \\ \vdots \\ \frac{\partial C}{\partial a_{n_c}^{(c)}} \end{pmatrix} \odot \begin{pmatrix} f'(z_1^{(c-1)}) \\ \vdots \\ f'(z_n^{(c-1)}) \end{pmatrix} = \nabla_{a^{(c)}} C \odot f'(z^{(c-1)}) = (y-s) \odot f'(z^{(c-1)})$$

donde $\odot$ representa el producto de Hadamard, es decir, el producto componente a componente.

Este $\delta^{(c-1)}$ es fácil de calcular. Ahora encontraremos una fórmula para expresar el error de la capa k ($\delta^{(k)}$) en función del error de la capa $k+1$ ($\delta^{(k+1)}$). Como

$$z_i^{(k+1)} = \sum_{j=1}^{n_{k+1}} \omega_{ij}^{(k+1)} a_j^{(k+1)} + u_j^{(k+1)} = \sum_{j=1}^{n_{k+1}} \omega_{ij}^{(k+1)} f(z_j^{(k)}) + u_j^{(k+1)}$$

entonces derivando obtenemos

$$\frac{\partial z_i^{(k+1)}}{\partial z_j^{(k)}} = \omega_{ij}^{(k+1)} f'(z_j^{(k)})$$

$$\delta_j^{(k)} = \frac{\partial C}{\partial z_j^{(k)}} = \sum_{i=1}^{n_{k+1}} \frac{\partial C}{\partial z_i^{(k+1)}} \frac{\partial z_i^{(k+1)}}{\partial z_j^{(k)}} = \sum_{i=1}^{n_{k+1}} \delta_i^{(k+1)} \frac{\partial z_i^{(k+1)}}{\partial z_j^{(k)}} = \sum_{i=1}^{n_{k+1}} \delta_i^{(k+1)} \omega_{ij}^{(k+1)} f'(z_j^{(k)})$$

y en notación matricial

$$\begin{pmatrix} \delta_1^{(k)} \\ \delta_2^{(k)} \\ \vdots \\ \delta_{n_{k+1}}^{(k)} \end{pmatrix} = \left(\begin{pmatrix} w_{11}^{(k+1)} & w_{12}^{(k+1)} \cdots w_{1n_k}^{(k+1)} \\ w_{21}^{(k+1)} & w_{22}^{(k+1)} \cdots w_{2n_k}^{(k+1)} \\ \vdots & \ddots \\ w_{n_{k+1}1}^{(k+1)} & w_{n_{k+1}2}^{(k+1)} \cdots w_{n_{k+1}n_k}^{(k+1)} \end{pmatrix}^T \begin{pmatrix} \delta_1^{(k+1)} \\ \delta_2^{(k+1)} \\ \vdots \\ \delta_{n_k}^{(k+1)} \end{pmatrix} \right) \odot \begin{pmatrix} f'(z_1^{(k)}) \\ f'(z_2^{(k)}) \\ \vdots \\ f'(z_{n_{k+1}}^{(k)}) \end{pmatrix}$$

$$\delta^{(k)} = ((\omega^{(k+1)})^T \delta^{(k+1)}) \odot f'(z^{(k)})$$

para $1 \leq k \leq c-2$.
Finalmente para calcular para calcular las derivadas parciales del coste respecto los pesos y los umbrales tenemos que

$$\frac{\partial C}{\partial u_j^{(k)}} = \frac{\partial C}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial u_j^{(k)}} = \delta_j^{(k)} \cdot 1 = \delta_j^{(k)}, \qquad 1 \leq k \leq c-1$$

$$\frac{\partial C}{\partial \omega_{ij}^{(k)}} = \frac{\partial C}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial \omega_{ij}^{(k)}} = \delta_i^{(k)} \cdot a_j^{(k)}, \qquad 1 \leq k \leq c-1$$

aclarar esto último. uno es j y el otro es i

Función coste del algoritmo

Sea x un entreno individual. Sea $y(x)$ el vector de neuronas de la última capa (output de la red neuronal). El error de l-ésima capa de neuronas $a^l(x)$ lo calcularemos como

$$C_x = \frac{1}{2}||y(x) - a^l(x)||^2 = \frac{1}{2}\sum_j (y_j - a_j^l)^2$$

y para un conjunto de n entrenos haremos su media de manera que el coste será

$$C = \frac{1}{n}\sum_x C_x = \frac{1}{2n}\sum_x ||y(x) - a^l(x)||^2$$

2 Cross-entropy cost function

Recoramos que para la función de coste

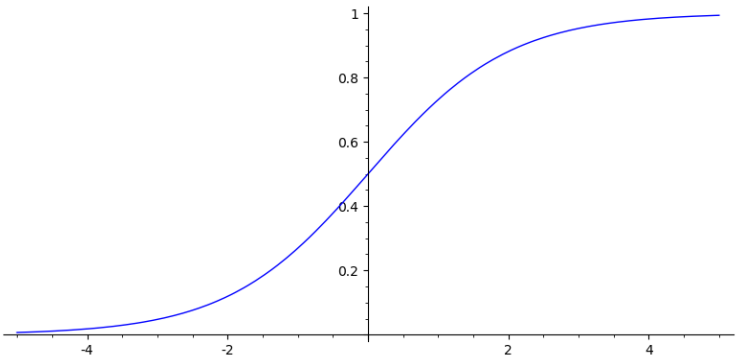
$$C_E = \frac{1}{2n} \sum_x ||y(x) - s(x)||^2$$

teníamos que

$$\frac{\partial C_E}{\partial \omega_{ji}^{(c-1)}} = \delta_j^{(c-1)} \cdot a_i^{(c-1)} = (a_i^{(c)} - s_i) \cdot f'(z_i^{(c-1)}) a_i^{(c-1)}$$

$$\frac{\partial C_E}{\partial u_i^{(c-1)}} = \delta_i^{(c-1)} = (a_i^{(c)} - s_i) \cdot f'(z_i^{(c-1)})$$

donde $f(x) = \frac{1}{1+e^{-x}}$ es la sigmoide. Observemos como es su gráfico



Observamos que para x "grandes" la función es un muy plana y por lo tanto $f'(x)$ será prácticamente nula. Esto hace que para algunos $z_i^{(c-1)}$ tanto $\frac{\partial C_E}{\partial \omega}$ como $\frac{\partial C_E}{\partial b}$ sean muy bajos y por lo tanto la red aprenda muy lento. Por lo tanto un desearía tener una función de coste C tal que

$$\frac{\partial C}{\partial \omega_{ji}^{(c-1)}} = (a_i^{(c)} - s_i) \cdot a_i^{(c-1)}$$

$$\frac{\partial C}{\partial u_i^{(c-1)}} = (a_i^{(c)} - s_i)$$

es decir, sin el factor $f'(z_i)$. Observamos que para una función de coste cualquiera C tenemos, por la regla de la cadena, que

$$\frac{\partial C}{\partial u_i^{(c-1)}} = \frac{\partial C}{\partial a_i^{(c)}} \frac{\partial a_i^{(c)}}{\partial z_i^{(c-1)}} \frac{\partial z_i^{(c-1)}}{\partial u_i^{(c-1)}} = \frac{\partial C}{\partial a_i^{(c)}} f'(z_i^{(c-1)}) \cdot 1$$

Como la sigmoide cumple $f'(x) = f(x)(1 - f(x))$ entonces

$$\frac{\partial C}{\partial u_i^{(c-1)}} = \frac{\partial C}{\partial a_i^{(c)}} f(z_i^{(c-1)})(1 - f(z_i^{(c-1)})) = \frac{\partial C}{\partial a_i^{(c)}} a_i^{(c)}(1 - a_i^{(c)})$$

Por otro lado queríamos que C cumpliera

$$\frac{\partial C}{\partial u_i^{(c-1)}} = (a_i^{(c)} - s_i)$$

por lo tanto

$$(a_i^{(c)} - s_i) = \frac{\partial C}{\partial a_i^{(c)}} a_i^{(c)}(1 - a_i^{(c)})$$

Denotando $a \equiv a_i^{(c)}$ e integrando

$$C = \int \frac{\partial C}{\partial a} da = \int \frac{a - s_i}{a(1-a)} da = -\left(s_i \ln a + (1 - s_i) \ln (1 - a)\right)$$

Observamos que si $a \in (0, 1)$ entonces el logaritmo es negativo y con el signo del principio tendremos que $C > 0$.

Además observamos que si $s_i = 0$ y $a \approx 0$ (lo cual es bueno) entonces $\ln(1-a) \approx 0$ y por lo tanto el "coste" será mínimo. Por otro lado si $s_i = 1$ y $a \approx 1$ entonces $\ln(a) = 0$ y de nuevo el coste es mínimo.

De modo que esta nueva función de coste tiene la ventaja respecto la C_E de que no se ralentiza su aprendizaje debido a $f'(z_i)$.

Por lo general conoceremos a

$$C = -\frac{1}{n} \sum_x \left(y \ln a + (1 - y) \ln (1 - a)\right)$$

como the Cross-entropy cost function.