Algoritmo de Retropropagación

Pedro Almagro Blanco

A continuación presentamos el método de retropropagación, uno de los algoritmos más conocidos para entrenar los pesos de una red neuronal artificial feedforward (las conexiones entre las unidades no forman un ciclo) multi-capa (Figura 1).

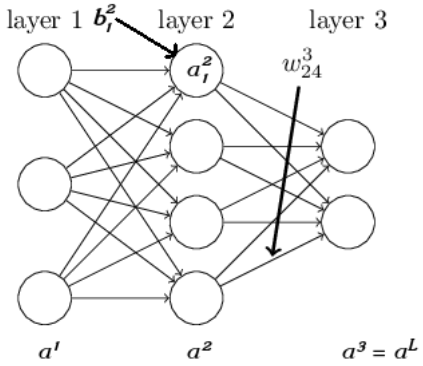

Fig. 1. Representación de una red neuronal artificial feedforward multicapa.

Definiciones Dada una red neuronal artificial prealimentada con $L > 2$ capas, w_{jk}^l representa el peso del enlace que conecta la neurona k-ésima en la capa $l-1$ con la neurona j-ésima en la capa l, b_j^l representa el bias de la neurona j-ésima en la capa l y a_j^l representa la salida (activación) de la neurona j-ésima en la capa l. Con esta notación, obtenemos la ecuación que relaciona la activación de una neurona en la capa l con las activaciones de las neuronas en la capa $l-1$:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Donde σ representa la función de activación neuronal. A continuación definimos la misma relación de una forma matricial, lo cual nos permitirá obtener

ecuaciones más sencillas que las que obtendríamos con las fórmulas elementales. Definimos una matriz w^l que representa los pesos de los enlaces que llegan a las neuronas en la capa l:

$$w^l = \begin{bmatrix} w^l_{11} & w^l_{12} & \dots \\ w^l_{21} & \ddots & \\ \vdots & & \end{bmatrix}$$

De manera similar, definimos un vector b^l que representa los bias de las neuronas en la capa l y un vector a^l que representa las activaciones de las neuronas en la capa l:

$$b^l = \begin{bmatrix} b^l_1 \\ \vdots \end{bmatrix}, a^l = \begin{bmatrix} a^l_1 \\ \vdots \end{bmatrix}$$

Si asumimos la vectorización de la función σ podemos considerar que dicha función se aplica sobre un vector elemento a elemento:

$$\sigma\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} \sigma(x) \\ \sigma(y) \end{bmatrix}$$

Con estos ingredientes podemos describir de forma matricial la activación de las neuronas en la capa l con respecto a las activaciones en la capa $l-1$:

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right)$$

A $\left(w^l a^{l-1} + b^l\right)$ lo representaremos mediante z^l (la entrada ponderada a las neuronas de la capa l) donde:

$$z^l = \begin{bmatrix} z^l_1 \\ \vdots \end{bmatrix}$$

Antes de continuar con la explicación del método de retropropagación, necesitamos definir el producto Hadamard ($\odot$), una operación binaria que toma dos matrices de iguales dimensiones y produce una nueva matriz donde cada elemento ij es el producto de los elementos ij en las dos matrices originales:

$$\begin{bmatrix}2\\4\end{bmatrix} \odot \begin{bmatrix}3\\6\end{bmatrix} = \begin{bmatrix}2*3\\4*6\end{bmatrix} = \begin{bmatrix}6\\24\end{bmatrix}$$

Dada una función de coste C que, dado un ejemplo de entrenamiento ($\boldsymbol{x}$,$\boldsymbol{y}$), evalúa la similitud entre la salida obtenida de la red a^L para la entrada $\boldsymbol{x}$ y la salida esperada $\boldsymbol{y}$, el objetivo del algoritmo de retropropagación es calcular las derivadas parciales $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$ las cuales nos permitirán modificar los pesos y bias de cada neurona para reducir el coste C. Definimos δ_j^l como el error en la neurona j-ésima en la capa l, es decir cuánto varía la función de coste con respecto a las entradas de dicha neurona:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

A continuación haremos una distinción entre el error en las neuronas de la capa de salida L y el error en las neuronas del resto de las capas $L-1, L-2, ..., 1$. El error en las neuronas de la capa de salida vendrá dado por la siguiente ecuación:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

Cuya forma matricial puede ser representada como:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{1}$$

Siendo $\nabla_a C$ el vector compuesto por las derivadas parciales de C con respecto a las diferentes activaciones de las neuronas en la capa de salida. El error en las neuronas de las capas interiores vendrá dado por la siguiente ecuación:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2}$$

Donde $(w^{l+1})^T$ representa la matriz traspuesta de la matriz de pesos de la capa $l+1$, necesaria para *propagar hacia atrás* los errores. Combinando las ecuaciones 1 y 2 podemos obtener el error δ_j^l en cualquier neurona perteneciente a cualquier capa en la red. Comenzaremos utilizando 1 para calcular δ^L, a continuación utilizaremos 2 para calcular δ^{L-1} en función de δ^L y sucesivamente utilizaremos 2 para calcular δ^{l-1} en función de δ^l.

Por último, presentamos dos ecuaciones más, necesarias para poder llevar a cabo el algoritmo de manera adecuada. La ecuación que indica cómo cambia el coste C con respecto a cualqueir bias en la red:

$$\frac{\partial C}{\partial b} = \delta \tag{3}$$

y la ecuación que indica cómo cambia el coste C con respecto a cualquier peso en la red:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{4}$$

Con esto, obtenemos cuatro ecuaciones fundamentales (1,2,3 y 4) del algoritmo de retropropagación que nos servirán para entender de manera más adecuada las modificaciones realizadas sobre los pesos y las implicaciones que conllevan.

Demostrando las ecuaciones fundamentales A continuación demostramos las cuatro ecuaciones fundamentales (1,2,3 y 4) que intervienen en el algoritmo de retropropagación, todas ellas se obtienen utilizando la regla de la cadena [**?**]. Comenzaremos demostrando la ecuación 1 del error en una neurona de la capa L:

$$\delta^L_j = \frac{\partial C}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

Y extendiendo a su forma matricial obtenemos la ecuación 1:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Para obtener la ecuación 2 utilizaremos también la regla de la cadena para obtener δ^l en función de δ^{l+1}:

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} = \sum_k \frac{\partial C}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_k} = \sum_k \frac{\partial z^{l+1}_k}{\partial z^l_j}\delta^{l+1}_k \tag{5}$$

Sabiendo que $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$ podemos deducir:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

Y ahora sustituyendo en la ecuación 5 obtenemos:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

Y extendiendo a su forma matricial obtenemos la ecuación 2:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Utilizando la regla de la cadena deducimos la ecuación 3:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \frac{\partial [\sum_k w_{kj}^l a_k^{l-1} + b_j^l]}{\partial b_j^l} = \delta_j^l$$

De manera similar obtenemos la ecuación 4:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l \frac{\partial [\sum_k w_{kj}^l a_k^{l-1} + b_j^l]}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Algoritmo Una vez demostradas las cuatro ecuaciones fundamentales, pasamos a describir el algoritmo de retropopagación en base a ellas:

1. **Entrada:** Activar las neuronas de la primera capa a^1 con la entrada $\boldsymbol{x}$.
2. **Propagación:** Para cada $l = 2, 3, ..., L$ calcular $z^l = w^l a^{l-1} + b^l$ y $a^l = \sigma(z^l)$
3. **Cálculo de errores en la capa de salida:** Calcular el vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. **Retropropagación del error:** Para cada $l = L-1, L-2, ..., 2$ calcular $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$

5. **Salida:** El gradiente de la función de coste C viene dado por:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_k^l, \frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Con el algoritmo presentado obtenemos el gradiente de la función de coste ∂C con respecto a los parámetros del modelo para una ejemplo determinado ($\boldsymbol{x}$,$\boldsymbol{y}$). En la práctica es común combinar el algoritmo de retropropagación con algún algoritmo de aprendizaje como la regla delta o el método del descenso por el gradiente estocastico. La regla delta actualiza los parámetros del modelo tras una ejecución del algoritmo de retropropagación para un ejemplo determinado de la siguiente forma:

$$w^l \to w^l - \eta\delta^l(a^{l-1})^T$$

$$b^l \to b^l - \eta\delta^l$$

El método de descenso por el gradiente estocástico calcula el gradiente de la función de coste ∂C para varios ejemplos de entrada $((\boldsymbol{x_1}, \boldsymbol{y_1}, ..., \boldsymbol{x_n}, \boldsymbol{y_n})$ como la suma de los gradientes obtenidos para cada ejemplo concreto.En concreto dado un *mini-batch* de m ejemplos, el siguiente algoritmo aplica un paso de aprendizaje del gradiente descendiente basado en dicho *mini-batch*:

1. **Se recibe un conjunto de ejemplos.**
2. **Para cada ejemplo de entrenamiento x:** Activar las neuronas de la primera capa $a^{x,1}$ con la entrada $\boldsymbol{x}$.
   (a) **Propagación:** Para cada $l = 2, 3, ..., L$ calcular $z^{x,l} = w^l a^{x,l-1} + b^l$ y $a^{x,l} = \sigma(z^{x,l})$
   (b) **Cálculo de errores en la capa de salida:** Calcular el vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$
   (c) **Retropropagación del error:** Para cada $l = L-1, L-2, ..., 2$ calcular $\delta^{x,l} = ((w^{l+1})^T\delta^{x,l+1}) \odot \sigma'(z^{x,l})$
3. **Gradiente descendiente:** Para cada $l = L, L-1, ..., 2$ actualizar los pesos deacuerdo a la regla $w^l \to w^l - \frac{\eta}{m}\sum_x \delta^{x,l}(a^{x,l-1})^T$ y los bias de cuardo a la regla $b^l \to b^l - \frac{\eta}{m}\sum_x \delta^{x,l}$.

Para implementar correctamente el gradiente descendiente se debería crear un loop externo que generara los *mini-batches* de manera aleatoria para iterar por las diferentes *épocas* del entrenamiento.

Limitaciones Observando la ecuación 4 se puede deducir que los enlaces que provienen de neuronas con una baja activación aprenden su pesos de una manera más lenta que las que provienen de una neurona con un alto valor de activación. Hasta ahora hemos presentado el algoritmo de retropropagación sin fijar una función de activación determinada. La elección de la función de activación σ es muy influyente a la hora de conseguir que una red aprenda una función determinada. Si utilizamos una función lineal como función de activación no podremos conseguir que la red aprenda una función que no sea lineal, ya que la combinación lineal de funciones lineales da como resultado una función lineal, además una función de este tipo no acota las activaciones neuronales a ningún rango, lo cual puede implicar que los valores propagados por la red tiendan a crecer demasiado o se vuelvan *incontrolables*. Una función muy adecuada para ser utilizada como función de activación en redes neuronales es la función sigmoide:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

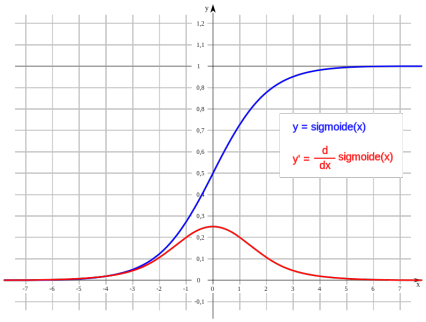

Fig. 2. Representación gráfica de la función sigmoide y su derivada.

La función sigmoide tiene algunas características que la hacen muy interesante para ser utilizada como función activadora de las en una red neuronal: Tiene un rango de salida $(0, 1)$ lo que hace que la propagación de las activaciones se mantenga siempre en ese rango y que no se *descontrole*, además la derivada de la función sigmoide es muy sencilla:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

La simplicidad en la derivada de la sigmoide implica que una vez que se ha calculado $\sigma(x)$ el cálculo de $\sigma'(x)$ sea casi directo y esto permite que los cálculos

en el algoritmo de retropropagación sean más rápidos. Pero una función como la función sigmoide también presenta algunos problemas a la hora de ser utilizada como función de activación neuronal, si utilizamos la función de activación sigmoide, observando la ecuación 2 podemos deducir que si la activación de una neurona en la capa de salida está cercana a 0 o a 1 la derivada de ésta será practicamente cero (Figura 2) y por tanto el aprendizaje de los pesos en los enlaces que llegan hacia ella se producirá de una manera muy lenta, cuando esto ocurre se dice que la neurona en la capa de salida está saturada. Por otra parte, al ser la función sigmoide una función exponencial y por tanto, también serlo su derivada, la transmisión de los errores hacia atrás provoca que éstos sean cada vez más cercanos a 0 y por tanto que el aprendizaje en las capas más alejadas de la capa de salida sea casi nulo, este es el motivo por el que no ha sido posible entrenar redes neuronales con un número de capas elevado.