Algoritmo de Retropropagación

Pedro Almagro Blanco

March 11, 2016

Red Neuronal Artificial Feedforward Multi-capa

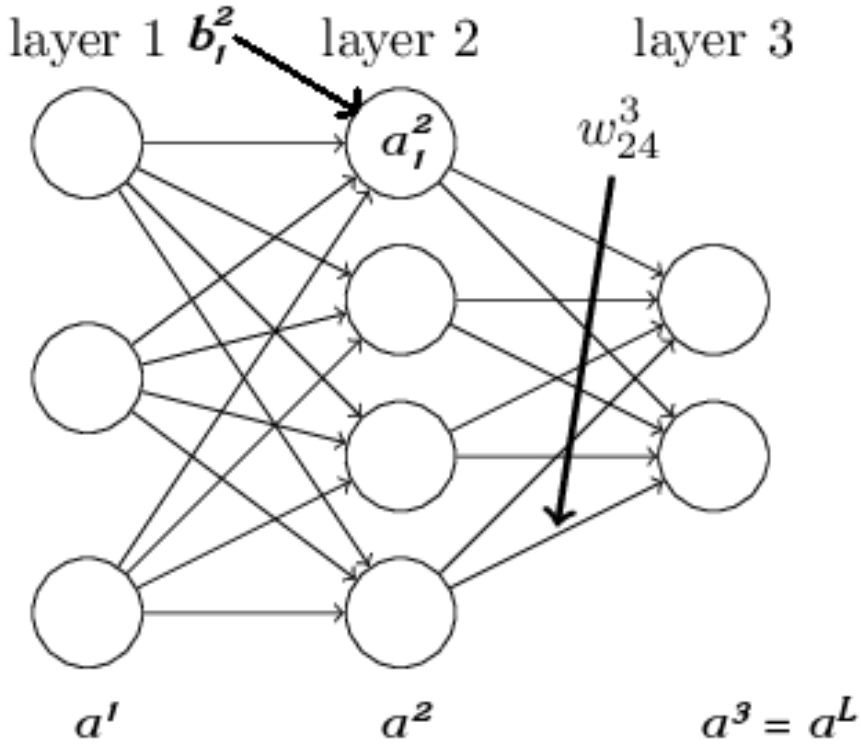

Figure : Representación de una red neuronal artificial feedforward multicapa.

Definiciones

Dada una red neuronal artificial prealimentada con $L > 2$ capas:

1. w^l_{jk} 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
2. b^l_j representa el bias de la neurona j-ésima en la capa l
3. a^l_j 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 $l-1$:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

Donde σ representa la función de activación neuronal.

Definiciones

Definiremos la misma relación de forma matricial. Definimos una matriz w^l que representa los pesos de los enlaces que llegan a las neuronas en la capa l:

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

Así como 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:

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

Definiciones

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 $l-1$:

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

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

Definiciones

El producto Hadamard ($\odot$) es 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:

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

Ecuaciones Fundamentales

δ^L para las neuronas en la capa de salida:

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

δ^l para las neuronas de las capas interiores:

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

Cómo cambia el coste C con respecto a cualqueir bias en la red:

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

Cómo cambia el coste C con respecto a cualquier peso en la red:

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

Algoritmo

1. **Entrada:** Activar las neuronas de la primera capa a^1 con la entrada $\vec{x}$.
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 $\delta^L = \nabla_{a^L} C \odot \sigma'(z^L)$
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:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_k, \frac{\partial C}{\partial b^l_j} = \delta^l_j$$

Descenso por el Gradiente

Con el algoritmo presentado obtenemos el gradiente de la función de coste ∂C con respecto a los parámetros del modelo para una ejemplo determinado $(\vec{x},\vec{y})$. En la práctica es común combinar el algoritmo de retropropagación con algún algoritmo de aprendizaje como el método del descenso por el gradiente:

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

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

Descenso por el Gradiente mini-batch

El método de descenso por el gradiente mini-batch calcula el gradiente de la función de coste ∂C para varios ejemplos de entrada $((\vec{x_1}, \vec{y_1}, ..., \vec{x_n}, \vec{y_n})$ como la suma de los gradientes obtenidos para cada ejemplo concreto:

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

Condición de Parada

Existen varios criterios para detener el algoritmo, algunas:

1. Cuando la cota del error cometido se sitúa por debajo de un umbral (sobre un conjunto de validación).
2. Cuando se alcanza un número de épocas determinadas.
3. Cuando el error cometido sobre el conjunto de validación deja de mejorar.

Función 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*.

Función de Activación Sigmoide

Una función muy adecuada para ser utilizada como función de activación en redes neuronales es la función sigmoide:

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

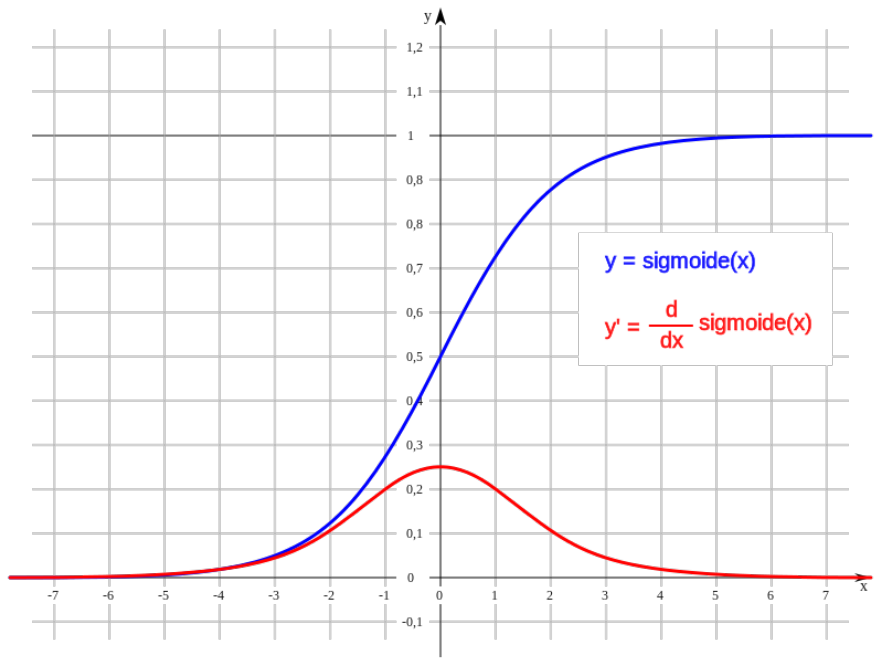

Figure : Representación gráfica de la función sigmoide y su derivada.

Función de Activación Sigmoide: Bondades

La función sigmoide tiene algunas características que la hacen muy interesante para ser utilizada como función activadora en una red neuronal: Tiene un rango de salida en el intervalo $(0, 1)$ 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.

Función de Activación Sigmoide: Limitaciones

Pero una función como la función sigmoide también presenta algunos problemas:

1. Observando la ecuaciones 1 y 2 podemos deducir que si la activación de una neurona 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.
2. Al ser la función sigmoide una función exponencial y por tanto, también serlo su derivada, la transmisión de los cálculos 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.