Redes neuronales Back Propagation

Enrique Calot

4 de octubre de 2009

1. Introducción

Una red neuronal de tipo *back propagation* permite aprender mediante un conjunto de ejemplo (entrada-salida) comunmente denominado *training set*. Al haber aprendido mediante este conjunto, se puede obtener una salida coherente para una entrada dada.

En la figura 1 se puede observar como se obtiene una salida a partir de la entrada. La red neuronal en este caso la vemos como una caja negra. En la figura 2 se observa como es internamente una red neuronal, en este caso solo mostramos dos capas, una de entrada y otra de salida, más adelante incorporaremos más capas intermedias.

Como podemos apreciar, cada neurona de entrada, que posee un valor en el rango [0;1], pasa ese valor a todas las neuronas de salida. Ese valor es multiplicado por el peso $w_{i,j}$ representado en las aristas. El valor de o_j es igual al de una función (denominada de transferencia) aplicada a la sumatoria de todos los productos definidos como $z_j = w_{i,j}x_i$ para el j de esa neurona de salida.

La información que almacena el aprendizaje de una red se encuentra en los pesos $w_{i,j}$ y en ningún lado más. Es muy importante comprender esto, ya que son los pesos los que hay que ajustar en el proceso de entrenamiento.

En la figura 3 podemos observar el proceso de entrenamiento tomando la red en forma de caja negra. Vemos que existen dos salidas: la que obtenemos mediante la red y la deseada. Al comparar ambas podemos observar cuan buena fue la predicción. El objetivo del proceso de entrenamiento es minimizar el error de la predicción y para ello, como se mencionó anteriormente, solo es posible modificar los pesos de la red.

El proceso de entrenamiento es iterativo, se inicializa la red con pesos cargados de manera arbitraria como configuración inicial y luego se tiende a modificarlos de la mejor forma posible. Para ellos se utiliza la propagación del error hacia atrás mediante sus derivadas y es por ello que la red toma el nombre de *back propagation*. Para el error obtenido se encuentra un vector $\Delta\vec{w}$ que sumado al vector de pesos $\vec{w}$ se obtiene una red que arroja un error más pequeño para esa entrada.

Como es de esperar, si se corre para la misma entrada este proceso varias veces, el resultado final sería, siempre y cuando la configuración de la red lo permita, una red con error nulo para ese valor. Ese no es el objetivo, sino lo que se desea es entrenar a la red con varias entradas y luego ver que sucede cuando ingresamos alguna que no estaba en el set de datos de entrenamiento. Es por esta razón que no se itera sobre un mismo elemento del conjunto de datos hasta eliminar el error sino que se realiza un acercamiento con un elemento, luego con otro y así hasta recorrer todo el conjunto de datos. A esta recorrida sobre el conjunto de datos se la suele denominar *epoch*. El error no será bajo, pero la red se habrá acercado hacia una zona donde convergen todos los elementos. Al repetir el proceso varias veces, es decir iterar varios *epoch*, la red comenzará a entrenarse.

2. Demostración formal

Ahora procederemos a definir estos conceptos detalladamente.
Llamaremos

- $\vec{x}$ es nuestro vector de entrada, cuyos n elementos denominaremos x_i.
- $\vec{o}$ es nuestro vector de salida obtenida (por *output*, entrada en inglés), cuyos m elementos denominaremos o_i.
- $\vec{t}$ es nuestro vector de salida deseada (por *target*, objetivo o blanco en inglés), cuyos m elementos denominaremos t_i.
- $\vec{w}$ es nuestro vector pesos (por *weight*, peso en inglés), cuyos $n \times m$ elementos denominaremos $w_{i,j}$. Notemos que es un vector unidimensional que pertenece a $\Re^{nm}$ y no es una matriz perteneciente a $\Re^{n \times m}$.
- $\vec{w}_j$ es el vector de pesos que llegan a un determinado o_j.
- W sí es la matriz de pesos, los cuales son los mismos que $w_{i,j}$ pero ahora sí estarán ordenados de manera matricial en $\Re^{n \times m}$.
- $\vec{z}$ es un vector intermedio previo a $\vec{o}$, cuyos m elementos denominaremos $z_j = \sum_{\forall i} x_i w_{i,j} = \vec{x}\vec{w}_j$.
- $Downstream(j)$ conjunto de unidades cuyas entradas inmediatas son las salidas de j.

Además mencionamos la función de transferencia, la cual definimos como $o_j = f(z_j) = f(\sum_{\forall i} x_i w_{i,j}) = f(\vec{x}\vec{w}_j)$.

El objetivo de la iteración es, dado $\vec{x}$ y $\vec{t}$, obtener un valor, $\Delta\vec{w}$, que sumado a $\vec{w}$ nos permita disponer de un mejor conjunto de pesos que acerque más los valores de $\vec{o}$ a los de $\vec{t}$. Para ellos definiremos una función de error, que será un numero escalar no negativo cuyo objetivo será minimizarlo mediante el ajuste de los pesos.

Una buena medida del error viene dada por la distancia euclidiana de ambos vectores, es decir $\left\|\vec{o} - \vec{t}\right\|$. Debido a que no nos interesa la magnitud del error y que a futuro nos simplificará las cuentas, agregaremos una constante multiplicativa de $1/2$ al principio. Obtenemos así nuestra función del error como $E = \frac{1}{2}\left\|\vec{o} - \vec{t}\right\| = \frac{1}{2}\sum_{j=1}^{m}(o_j - t_j)^2$.

Con el error definido, aplicaremos el operador gradiente para obtener su dirección de máximo crecimiento (siempre derivando con respecto a los $w_{i,j}$ que son nuestras variables independientes. La multiplicaremos por -1 para obtener la dirección de máximo decrecimiento y luego la multiplicaremos por un coeficiente que indicará la velocidad en que el algoritmo avanzará. Un coeficiente alto puede hacer que nos pasemos y que el algoritmo diverja, pero un valor muy bajo puede hacernos tardar mucho en llegar el objetivo. Por lo general se utilizan valores entre $0,6$ y $0,1$. Denominaremos al coeficiente como η.

Finalmente obtenemos la formula que nos permitirá calcular nuestro algoritmo $\Delta\vec{w} = -\eta\nabla E$. Siendo $\vec{w}_{n+1} = \vec{w}_n + \Delta\vec{w}_n$ y $\vec{w}_0$ un vector aleatorio. Notemos que estos coeficientes se refieren al numero de iteración y no a la definición dada anteriormente de $\vec{w}_j$.

2.1. Propiedades previas

Antes de comenzar, debemos repasar propiedades matemáticas que son aisladas de nuestro problema ya que su explayamiento en medio de la explicación del proceso de entrenamiento puede generar confusión.

1. Propiedades de la distribución sigmoidal

   La distribución sigmoidal viene dada por la formula

   $\sigma_k(x) = \frac{1}{e^{-xk}+1}$ (eq 1)

   Podemos observar varias propiedades: La primera es que tiene valores entre 0 y 1 y es biyectiva. Esto se prueba mostrando que su máximo valor es el 1 cuando $lim_{x\,\text{ínf}}\sigma_k(x) = 1$ y su mínimo valor es 0 cuando x tiende a $-$ínf, según el limite $lim_{x-\text{ínf}}\sigma_k(x) = 0$. Al mostrar que la función es monótonamente creciente veremos que es biyectiva con un dominio en todos los números reales y una imagen en el intervalo 0;1.

   Otra propiedad es que $1-\sigma_k(x) = \sigma_k(-x)$ ya que $1-\sigma_k(x) = 1-\frac{1}{e^{-xk}+1} = \frac{e^{-xk}+1}{e^{-xk}+1} - \frac{1}{e^{-xk}+1} = \frac{e^{-xk}+1-1}{e^{-xk}+1} = \frac{e^{-xk}}{e^{-xk}+1} = \frac{1}{1+e^{xk}} = \sigma_k(-x)$
   Esto nos muestra que $\sigma_k(x) + \sigma_k(-x) = 1$, o sea que la función es impar sobre un eje de simetría ubicado en 1/2.

   La derivada de la función sigmoidal se calcula mediante la regla de la cadena como

   $\sigma'_k(x) = -1(\frac{1}{e^{-xk}+1})^{-2}e^{-xk}(-k) = k\frac{1}{e^{-xk}+1}\frac{e^{-xk}}{e^{-xk}+1} = k\frac{1}{e^{-xk}+1}\frac{1}{1+e^{xk}} = k\sigma_k(x)\sigma_k(-x)$

   (eq2)

   Como podemos apreciar es siempre positiva en el intervalo (0;1) y por lo tanto nuestra función es monótonamente creciente.

   Además, por la propiedad de simetría, podemos decir que $\sigma'_k(x) = k\sigma_k(x)\sigma_k(-x) = k\sigma_k(x)(1-\sigma_k(x))$, por lo que si se conoce el resultado, es posible calcular su derivada sin la necesidad de hallar el x que la genera, acelerando los cálculos de derivadas.

   Por simplicidad de cuentas utilizaremos $\sigma(x) = \sigma_1(x) = \frac{1}{e^{-x}+1}$ a la función sigmoidal de $k = 1$.

2. Regla de la cadena

   La regla de la cadena nos permite derivar funciones compuestas de manera independiente una de la otra, en nuestro caso la utilizaremos para derivar funciones con varias variables.

La regla viene dada por la formula: $\frac{\partial E}{\partial w} = \sum_{\forall i} \frac{\partial E}{\partial p_i} \frac{\partial p_i}{\partial w}$

(eq)

3. Derivada de una sumatoria con constantes.

   Es conveniente aclarar esta propiedad matemática antes de empezar el desarrollo de redes neuronales.

   Al derivar en una sumatoria con varias constantes, solo sobreviven las derivadas que dependen de la variable sobre la cual derivamos, es decir $\frac{\partial(x_1+x_2+x_3+x_4)}{\partial x_3} = \frac{\partial x_3}{\partial x_3} = 1$

   Ahora, si generalizamos este planteo es muy facil de ver que

   $\frac{\partial \sum_{\forall i} x_i}{\partial x_j} = \frac{\partial x_j}{\partial x_j} = 1$. Es decir, la única variable que sobrevive es en el caso $i = j$, el resto son constantes y su derivada es nula.

   Agregando una constante multiplicativa (w_i) a cada termino, vemos que solo sobrevive la constante del termino perteneciente a la variable independiente.

   $\frac{\partial \sum_{\forall i} w_i x_i}{\partial x_j} = \frac{\partial (w_j x_j)}{\partial x_j} = w_j$

   Generalmente se utiliza la definición del delta de Kronecker para eliminar sumatorias de este tipo.

2.2. Demostración del algoritmo sin capas intermedias

En la presente sección veremos como hallar $\Delta\vec{w}$ a partir de un vector de entradas $\vec{x}$, uno de salidas deseadas $\vec{t}$, nuestros pesos $\vec{w}$ y el valor del paso (o velocidad de convergencia) η.

Como mencionamos anteriormente, debemos hallar el gradiente de E con respecto a los pesos, es decir $\nabla E = \frac{\partial E}{\partial w_{i,j}}$. Por la regla de la cadena (propiedad 2) sabemos que $\nabla E_u = \frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial z_i} \frac{\partial z_i}{\partial w_{i,j}}$ (siendo u la posición en E de cada par i, j.

Calcularemos ambos términos por partes comenzando por la derivada del error con respecto a z_i y aplicando la propiedad 3 obtenemos que

$\frac{\partial E}{\partial z_j} = \frac{1}{2} \frac{\partial \sum_{n=1}^{m} (o_n - t_n)^2}{\partial z_j} = \frac{1}{2} \frac{\partial (o_j - t_j)^2}{\partial z_j}$

luego por las propiedades de la función sigmoidal (utilizaremos la de $k = 1$) y que $o_j = \sigma(z_j)$ podemos llegar a la siguiente expresión

$\frac{\partial E}{\partial z_j} = \frac{1}{2} \frac{\partial (o_j - t_j)^2}{\partial z_j} = \frac{1}{2} \frac{\partial (\sigma(z_j) - t_j)^2}{\partial z_j} = \frac{1}{2} 2 \frac{\partial (\sigma(z_j) - t_j)}{\partial z_j} (\sigma(z_j) - t_j) = \sigma(z_j)(1 - \sigma(z_j))(\sigma(z_j) - t_j) = o_j(1 - o_j)(o_j - t_j)$

Definiremos $\delta_j = \frac{\partial E}{\partial z_j} = o_j(1 - o_j)(o_j - t_j)$ ya que solo depende de j, con lo cual, al iterar este cálculo puede realizarse una sola vez.

Ahora veremos la segunda parte que es $\frac{\partial z_i}{\partial w_{i,j}}$, la cual, por la propiedad 3, se puede resolver de manera fácil en

$\frac{\partial z_i}{\partial w_{i,j}} = \frac{\partial \sum_{\forall k} x_k w_{k,j}}{\partial w_{i,j}} = x_i$

Finalmente obtuvimos que $\frac{\partial E}{\partial w_{i,j}} = \delta_j x_i = o_j(1-o_j)(o_j - t_j)x_i$.

Por lo tanto, para ajustar un peso $w_{i,j}$ debemos hacer $w_{n+1,i,j} = w_{n,i,j} - \eta o_j(1-o_j)(o_j-t_j)x_i$.

2.3. Ampliación a n capas intermedias

Utiulizando la definición previa de $Downstream(j)$ y aplicando las derivadas parciales se obtiene, mediante la regla de la cadena la siguiente sumatoria $\frac{\partial E}{\partial w_{ji}} = \sum_{\forall k \in Downstream(j)} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial o_j}\frac{\partial o_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} = \sum_{\forall k \in Downstream(j)} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial o_j}\frac{\partial o_j}{\partial z_j}x_{ji} = \delta_j x_{ji}$

Podemos observar que δ_j ahora se calcula mediante una sumatoria, pero todo el proceso se realiza análogamente.

Sustituyendo

$\frac{\partial E}{\partial z_k} = \delta_k$, $\frac{\partial z_k}{\partial o_j} = w_{k,j}$,$\frac{\partial o_j}{\partial z_j} = o_j(1-o_j)$ en nuestra nueva δ_j obtenemos que

$\delta_j = \sum_{\forall k \in Downstream(j)} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial o_j}\frac{\partial o_j}{\partial z_j} = \sum_{\forall k \in Downstream(j)} \delta_k w_{kj} o_j(1-o_j)$.

Como los o_j no dependen de k se pueden quitar de la sumatoria obteniendo

$\delta_j = o_j(1-o_j)\sum_{\forall k \in Downstream(j)} \delta_k w_{kj}$.

Como se puede observar, el entrenamiento es muy similar al de la capa final, salvo que esta vez tenemos una sumatoria.

Para generalizar decimos que en la última capa calculamos δ_{k_0} como $\delta_{k_0} = o_{k_0}(1-o_{k_0})(t_{k_0}-o_{k_0})$ mientras que en el resto lo hacemos con δ_{k_n} como $\delta_{k_n} = o_{k_n}(1-o_{k_n})\sum_{\forall k_{n-1} \in Downstream(k_n)} w_{k_{n-1}kn}\delta k_{n-1}$.

Finalmente los pesos se ajustan mediante $w_{n+1,i,j} = w_{n,i,j} - \eta o_j \delta_j x_i$.