ALGORITMOS DE APRENDIZAJE EN RETROPROPAFACION PARA EL PERCEPTRON MULTICAPA

Existen diferentes algoritmos de aprendizaje que optimizan las conexiones entre las neuronas según el error este cometiendo la red, entendiendo por error o error instantáneo, la diferencia que existe entre la salida ofrecida por la red y la salida deseada.

En el momento de la implementación se deben tener en cuenta las características del problema a resolver, para escoger el algoritmo más adecuado, sin embargo para optimizar el funcionamiento, el algoritmo debe cumplir con las siguientes particularidades [1]:

- Eficacia
- Robustez, para adaptarse a diferentes problemas
- Independencia respecto a las condiciones iniciales
- Alta capacidad de generalización
- Coste computacional bajo
- Sencillez en los razonamientos empleados

Algunos de los algoritmos de aprendizaje existentes para perceptrones multicapa son:

- Retro propagación (Back Propagation)
- Momento
- Resilient BackPropagation (RPROP)
- Algoritmos de segundo orden
    - Levenberg-Maquardt

1. Momento

Esta variante añade un término que controla la velocidad de acercamiento al mínimo acelerándola, como dice [2], cuando se está lejos de este y deteniéndola cuando se está cerca, viene dado por la expresión:

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} + \mu \Delta w(n-1) \qquad (1)$$

Donde $\Delta w(n-1) = w(n-1) - w(n-2)$ es el incremento que sufrio el parámetro w en la iteración anterior y μ es un numero positivo que controla la importancia dada el incremento anterior y se denomina momento, esta regla fue propuesta por [3].

2. Resilient BackPropagation (RPROP)

2.1. Rprop y la gradiente

Muchos algoritmos de aprendizaje automático, incluido Rprop, se basan en un concepto matemático llamado gradiente. Creo que el uso de una imagen es la mejor manera de explicar qué es un gradiente. Eche un vistazo al gráfico en la **figura 2**. La curva representa un error frente al valor de un solo peso. La idea aquí es que debe tener alguna medida de error (hay varios) y que el valor del error cambiará a medida que cambie el valor de un peso, suponiendo que mantenga los valores de los otros pesos y los sesgos iguales. Para una red neuronal con muchos pesos y sesgos, habría gráficos como el de la **Figura 2** para cada peso y sesgo.

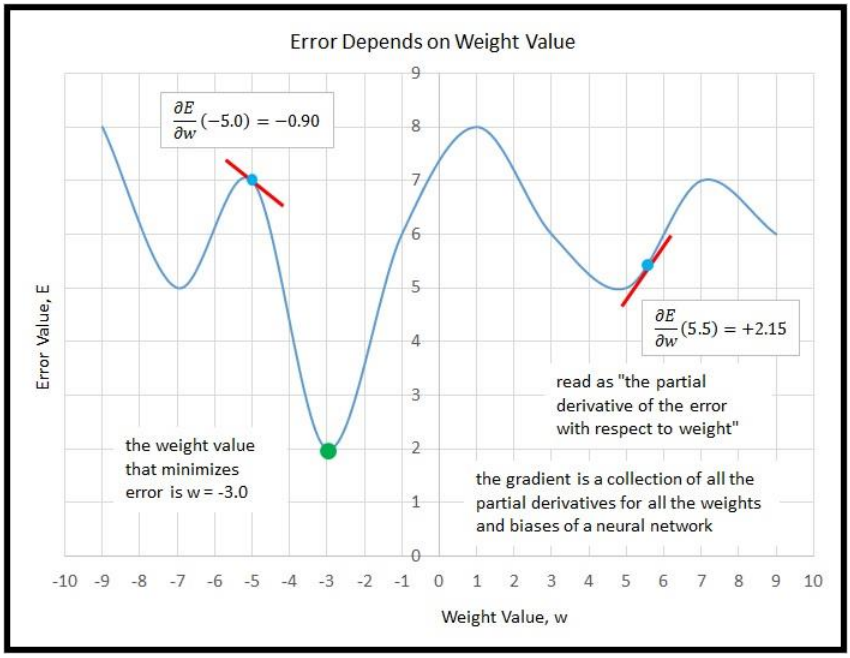

Figura 2. Derivados parciales y gradiente

Un gradiente está compuesto por varios "derivados parciales". Una derivada parcial para un peso se puede considerar como la pendiente de la línea tangente (la pendiente, no la línea tangente en sí) a la función de error para algún valor del peso. Por ejemplo, en la

figura, la "derivada parcial de error con respecto al peso" en peso = -5.0 es -0.90. El signo de la pendiente / derivada parcial indica en qué dirección ir para llegar a un error menor. Una pendiente negativa significa ir en la dirección del peso positivo, y viceversa. La inclinación (magnitud) de la pendiente indica qué tan rápido está cambiando el error y da una idea de qué tan lejos moverse para llegar a un error menor.

Las derivadas parciales se llaman parciales porque solo tienen un peso en cuenta; Se supone que los otros pesos son constantes. Un gradiente es solo una colección de los derivados parciales para todos los pesos y sesgos. Tenga en cuenta que aunque el gradiente de palabra es singular, tiene varios componentes. Además, los términos gradiente y derivado parcial (o simplemente "el parcial", por brevedad) a menudo se usan indistintamente cuando el significado está claro en el contexto.

Durante el entrenamiento, la propagación regular de la espalda usa las magnitudes de las derivadas parciales para determinar cuánto ajustar un valor de peso. Esto parece muy razonable, pero si miras la **Figura 2**, puedes ver un inconveniente para este enfoque. Supongamos que un peso tiene un valor actual de -5.0 y la propagación regular de la espalda ve un gradiente bastante empinado y calcula un delta de peso de +7.0. El nuevo valor de peso será -5.0 + 7.0 = 2.0 y, por lo tanto, el peso ha ido más allá del valor óptimo en -3.0. En la siguiente iteración de entrenamiento, el peso podría girar salvajemente hacia atrás y rebasar nuevamente, pero en la otra dirección. Esta oscilación podría continuar y el peso del error mínimo nunca se encontraría.

Con la propagación regular de la espalda, normalmente usa una pequeña tasa de aprendizaje, que, junto con la magnitud del gradiente, determina el delta del peso en una iteración de entrenamiento. Esto significa que probablemente no rebasará una respuesta óptima, pero significa que el entrenamiento será muy lento a medida que te acercas más y más a un peso que proporciona un error mínimo.

El algoritmo Rprop realiza dos cambios significativos en el algoritmo de propagación posterior. Primero, Rprop no usa la magnitud del gradiente para determinar un delta de peso; en su lugar, solo usa el signo del gradiente. En segundo lugar, en lugar de utilizar una única tasa de aprendizaje para todos los pesos y sesgos, Rprop mantiene deltas de peso por separado para cada peso y sesgo, y adapta estos deltas durante el entrenamiento.

La versión original del algoritmo de Rprop se publicó en un documento de 1993, "Un método directo adaptativo para un aprendizaje de propagación más rápido de la espalda: el algoritmo de Rprop", de M. Riedmiller y H. Braun. Varias variaciones de Rprop se han publicado desde el artículo original. El programa de demostración sigue de cerca la versión original del algoritmo, sin embargo, las descripciones de algunas partes del algoritmo son ambiguas y se pueden interpretar de más de una manera.

2.2. Algoritmo de Rprop

Según [4], sabemos que el RPROP es un esquema de aprendizaje eficiente, que realiza una adaptación directa del paso de peso basado en información de gradiente local. Para lograr esto, presentamos para cada peso su valor de actualización individual Δ_{ji} que determina únicamente el tamaño de la actualización del peso. Este valor de actualización adaptativo evoluciona durante el proceso de aprendizaje basado en la función de error E, de acuerdo con la siguiente regla de aprendizaje:

$$\Delta_{ji}^{(t)} = \begin{cases} n^+ * \Delta_{ji}^{(t-1)}, & si \frac{\partial E}{\partial w_{ji}}^{(t-1)} * \frac{\partial E}{\partial w_{ji}}^{(t)} > 0 \\ n^- * \Delta_{ji}^{(t-1)}, & si \frac{\partial E}{\partial w_{ji}}^{(t-1)} * \frac{\partial E}{\partial w_{ji}}^{(t)} < 0 \\ \Delta_{ji}^{(t-1)}, & otros\ casos \end{cases} \quad (2)$$

$$\boldsymbol{where\ 0 < n^- < 1 < n^+}$$

Verbalizada, la regla de adaptación funciona de la siguiente manera: cada vez que la derivada parcial del peso correspondiente $\boldsymbol{w_{ji}}$cambia su signo, lo que indica que la última actualización fue demasiado grande y el algoritmo saltó por encima de un mínimo local, se reduce el valor de actualización Δ_{ji} por el factor $\boldsymbol{n^-}$.Si la derivada conserva su signo, el valor de actualización se incrementa ligeramente para acelerar la convergencia en regiones poco profundas. Una vez que se adapta el valor de actualización para cada peso, la actualización de peso sigue una regla muy simple: si la derivada es positiva (aumento de error), el peso se reduce por su valor de actualización, si la derivada es negativa, el valor de actualización se agrega:

$$\Delta w_{ji}^{(t)} = \begin{cases} -\Delta_{ji}^{(t)}, & si \ \frac{\partial E}{\partial w_{ji}}^{(t)} > 0 \\ +\Delta_{ji}^{(t)}, & si \ \frac{\partial E}{\partial w_{ji}}^{(t)} < 0 \\ 0 \ , & otros \ casos \end{cases} \quad (3)$$

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)} \quad (4)$$

Sin embargo, hay una excepción: si la derivada parcial cambia de signo, es decir, el paso anterior fue demasiado grande y se omitió el mínimo, se revierte la actualización de peso anterior y no se toma en cuenta (3) y (4) para el $\Delta w_{ji}^{(t)}$:

$$\Delta w_{ji}^{(t)} = -\Delta w_{ji}^{(t-1)} \ , si \frac{\partial E}{\partial w_{ji}}^{(t-1)} * \frac{\partial E}{\partial w_{ji}}^{(t)} < 0 \quad (5)$$

El siguiente fragmento de seudocódigo se basa en las reglas (2), (3), (4) y (5). Donde el operador ***sign*** retorna +1 si el argumento es positivo, -1, si el argumento es negativo, y 0 en otros casos.

Para todos los pesos y bias {

si ($\frac{\partial E}{\partial w_{ji}}^{(t-1)} * \frac{\partial E}{\partial w_{ji}}^{(t)} > 0$) {

$$\Delta_{ji}^{(t)} = n^{+} * \Delta_{ji}^{(t-1)}$$

$$\Delta w_{ji}^{(t)} = -sign\left(\frac{\partial E}{\partial w_{ji}}^{(t)}\right) * \Delta_{ji}^{(t)}$$

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

}

sino si ($\frac{\partial E}{\partial w_{ji}}^{(t-1)} * \frac{\partial E}{\partial w_{ji}}^{(t)} < 0$) {

$$\Delta_{ji}^{(t)} = n^{+} * \Delta_{ji}^{(t-1)}$$

$$\Delta w_{ji}^{(t+1)} = \Delta w_{ji}^{(t)} - \Delta w_{ji}^{(t-1)}$$

$$\frac{\partial E}{\partial w_{ji}}^{(t)} = 0$$

}

sino si ($\frac{\partial E}{\partial w_{ji}}^{(t-1)} * \frac{\partial E}{\partial w_{ji}}^{(t)} = 0$) {

$$\Delta w_{ji}^{(t)} = -sign\left(\frac{\partial E}{\partial w_{ji}}^{(t)}\right) * \Delta_{ji}^{(t)}$$

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

}

}

3. LEVENBERG-MAQUARDT

El Back-Propagation ha demostrado converger muy lentamente en varias aplicaciones, en especial cuando se tiene una gran cantidad de patterns donde suele converger pero a un MSE demasiado grande, lo que se llama mínimo local del MSE y que muchas veces no es útil ya que se busca una convergencia hacia el mínimo absoluto [5].Es por eso que las investigaciones en cuanto a redes neuronales no se detiene, en especial por que han demostrado gran potencial. A la fecha existen diferentes algoritmos de entrenamiento supervisado que han surgido del Back-Propagation y que muestran velocidades muchos más rápidos de convergencias del MSE hacia el mínimo absoluto. Uno de ellos es el algoritmo de Levenberg-Maquardt.

Antes de continuar, mostraremos unas expresiones matemáticas que están relacionadas a este algoritmo de aprendizaje, los cuales son la matriz Jacobiana y la matriz Hessiana.

3.1. Matriz Jacobiana

La matriz Jacobiana es una matriz formada por las derivadas parciales de primer orden de una función, para este caso del algoritmo de Levenberg-Maquardt está dada por la función de error, y entonces la matriz estaría compuesta por las derivadas de los errores respecto a los pesos y umbrales. En la ecuación (6), donde w_{k0} representa a las bias de la capa de oculta.

$$J(E) = \begin{bmatrix} \frac{dE_1}{\partial w_{10}} & \cdots & \frac{dE_1}{\partial w_{1p}} \\ \vdots & \ddots & \vdots \\ \frac{dE_m}{\partial w_{m0}} & \cdots & \frac{dE_m}{\partial w_{mp}} \end{bmatrix} \quad (6)$$

Dónde:

- k : 1…m
- j : 1…p
- E_k : errores de la neuronas de la capa de salida
- w_{kj}: bias y pesos entre las neuronas de la capa oculta y la capa de salida

3.2. Matriz Hessiana

La matriz Hessiana es una matriz formada por las derivadas parciales de segundo orden de una función, para este caso del algoritmo de Levenberg-Maquardt está dada por la

función de error, por lo que la matriz Hessiana estaría compuesta por las derivadas de segundo orden respecto a los pesos y umbrales. En la ecuación (7), donde w_{k0} representa a la bias de la capa oculta.

$$H(E) = \begin{bmatrix} \frac{\partial^2 E_1}{\partial w_{10}} & \cdots & \frac{\partial^2 E_1}{\partial w_{1j}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E_k}{\partial w_{10}} & \cdots & \frac{\partial^2 E_k}{\partial w_{kj}} \end{bmatrix} \quad (6)$$

Dónde:

- k : 1…m
- j : 1…p
- E_k : errores de la neuronas de la capa de salida
- w_{kj}: bias y pesos entre las neuronas de la capa oculta y la capa de salida

Como se sabe el algoritmo de Levenberg- Maquardt es un método de segundo orden, pues hacen uso de la dervidada de segundo orden del error, es decir hacen uso de la matriz Hessiana, pero calcular las derivadas de segundo orden tiene un alto coste computacional, por lo que este método aproxima la matriz Hessiana en base a la matriz Jacobiana. De esta manera el algoritmo fue diseñado para alcanzar una velocidad de entrenamiento de segundo orden sin tener que calcular la matriz Hessiana.Cuando la función de coste empleada es el error cuadrático medio (MSE), la Hessiana puede aproximarse como:

$$H(n) = J^T(n) * J(n) \quad (7)$$

Dónde:

- *H(n)*: Matriz Hessiana
- *J(n)*: Matriz Jacobiana

Y el gradiente puede calcularse como

$$g(n) = J^T(n) * e(n) \quad (7)$$

Dónde:

- *e(n)*: vector de los errores de las neuronas de salida

El algoritmo de Levenberg-Maquardt se aplica principalmente a redes neuronales multicapa con un número de grande patterns ya que tiene la velocidad de convergencia del MSE más rápida hasta ahora, principalmente en problemas de aproximación de funciones a pesar de que su complejidad en cálculos es mayor [6] .La ecuación con la que se actualizan los pesos es la siguiente:

$$w(n+1) = w(n) - \frac{J^T(n).e}{J^T(n) * J(n) + \mu.I} \quad (7)$$

Dónde μ es una constante equivalente a la tasa de aprendizaje, que es decrecida en cada iteración en la que se observa una reducción del MSE, o incrementada y se descartan los pesos actualizados cuando se obtiene un aumento en el MSE.

Además cuando el valor de μ es cero, el algoritmo es simplemente un método de Newton, empleando una aproximación de la hessiana, mientras que cuando μ es grande, este método se convierte en un método de descenso por gradiente con un peso de aprendizaje pequeño. El método de Newton es más rápido y preciso cerca de un mínimo, por tanto, el objetivo es cambiar hacia este método tan pronto como sea posible. El valor de μ se decrementa después de cada iteración en la que se haya reducido el valor de la función objetivo y se incrementa solo cuando el valor de la función de error en la iteración actual crece. De esta forma, la función de error simple decrece en cada iteración del algoritmo.

REFERENCIAS

[1] Serrano, A.;Soria, E.;Martín, J. Redes Neuronales Artificiales, en http://ocw.uv.es/ingenieria-y-arquitectura/1-2/libro_ocw_libro_de_redes.pdf

[2] http://www2.unalmed.edu.co/~pruebasminas/index.php?option=com_docman&task=doc_view&gid=1743&tmpl=component&format=raw&Itemid=285

[3] Rumelhart, D.,Hinto, G., Williams,R.,Parallel Distributed Processing,MITPress,1986.

[4] A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm (1993).http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.1417

[5] http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/rodriguez_p_hu/capitulo3.pdf

[6] http://ruc.udc.es/dspace/bitstream/handle/2183/1042/FontenlaRomero_Oscar_TD_2002.pdf?sequence=1