Perceptrón y propagación hacia atrás

Perceptron and backpropagation

Autores: Cristian Camilo Rodríguez Ríos
Nicolas Amaya Rico
Victor Hugo Betancourth Granada
IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: c.rodriguez2@utp.edu.co
nicolas.amaya@utp.edu.co
victor.betancourth@utp.edu.co

***Resumen*— En este documento se presenta un breve resumen del funcionamiento de un perceptrón simple, perceptrón multicapa, y finalmente el algoritmo backpropagation, aplicándose a un breve problema de clasificación.**

***Palabras clave*— Sistemas, perceptrón, redes, inteligencia artificial, software, computación, búsquedas, industria, perceptrón, redes neuronales, función de activación.**

***Abstract*— In this document present a short summary of working of simple perceptron, multilayer perceptron, and finally backpropagation algorithm, applying to a short classification problem.**

***Keyword*— systems, networks, artificial intelligence, software, computing, research, industry, neural nets, activation function .**

I. INTRODUCCIÓN

Para nadie es un secreto que la inteligencia artificial ha sido unos de los campos que más ha influenciado nuestra vida cotidiana. Siendo una de las ciencias principales en el desarrollo de innovaciones en robótica, medicina, traducción de idiomas inclusive en los juguetes infantiles de nuestros niños.
Por ello en este paper, profundizaremos en el tema que sentó las bases para esta revolución, las redes neuronales. Donde presentaremos los conceptos claves para entender el tema desde el perceptrón hasta el algoritmo backpropagation en una red neuronal.

II. CONTENIDO

II.1 Perceptrón simple

En [1] se define el perceptrón simple, como el modelo más sencillo de redes neuronales artificiales, el cual posee una sola capa de neuronas con una única salida. (Ver figura 1.1)

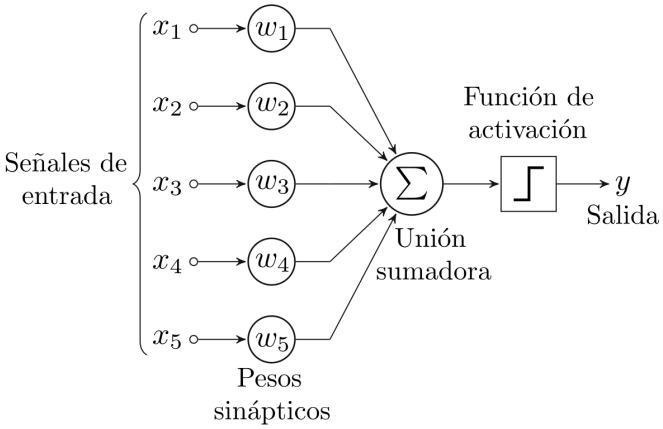

***Figura 1.1**: Ejemplo perceptrón (Fuente: [1])*

En la figura 1.1, encontramos algunas partes importantes de la composición del perceptrón. Algunas de ellas son: Señales de entrada, pesos, unión sumadora, función de activación y finalmente la salida.
La unión sumadora, y la función de activación, forman lo que conocemos como neurona.

Para encontrar el valor de la salida 'Y', deberemos aplicar un algoritmo de aprendizaje, para lo que encontramos en primer lugar una suma ponderada que identificamos bajo la letra 'Z' de las entradas con sus respectivos pesos, de la siguiente manera.

$$Z = b + \sum_i x_i * w_i$$

Donde, xi es la entrada i a la capa de neuronas, wi es el peso que tiene la entrada i con la capa neuronal, b 'bias' o mejor

conocido el sesgo el cual controla que tan predispuesta está la neurona a disparar un 1 o 0 independiente de los pesos. Un sesgo alto hace que la neurona requiera una entrada más alta para generar una salida de 1. Un sesgo bajo lo hace más fácil.
En este punto del algoritmo, 'Z' nos servirá finalmente como entrada para una función de activación, la cual transformará los datos en una salida.
Históricamente la función sigmoide es la función de activación más antigua y popular. Se define como:

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

Donde, 'e' se denota la constante exponencial, que es igual a 2.71828.
Una neurona que utiliza la sigmoide como función de activación se llama neurona sigmoide.
Teniendo nuestra variable z, resta únicamente pasarla a través de la función sigmoide.

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

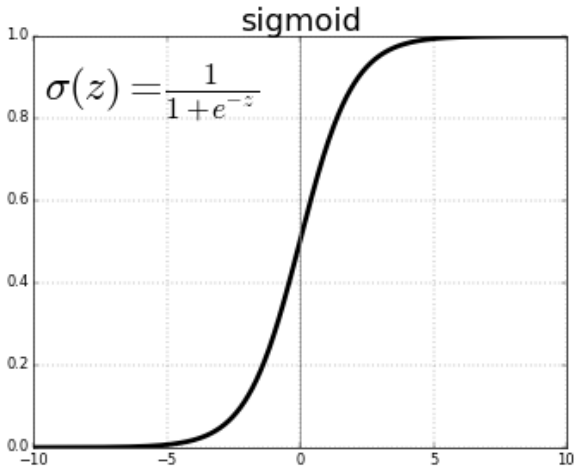

Figura 1.2: *Gráfica función sigmoide (Fuente: [2])*

Podemos ver que la función de activación, comprime nuestra salida a un rango de 0 a 1. (Ver figura 1.2)
Finalmente,

$$Z = b + \sum_i x_i * w_i$$
$$Y = \sigma(z)$$

Después de encontrar el valor de salida que nuestro perceptrón simple género, se da por finalizado el algoritmo.

Es importante tener en cuenta, que en las aplicaciones prácticas un perceptrón simple no tiene mucha cabida, esto debido a su limitada matemática lineal, y es por ello, que estas unidades se suelen agrupar en un conjunto de dos o más para conformar una capa neuronal. Estas capas neuronales se unen entre sí formando lo que conocemos como perceptrón multicapa o red neuronal.

II.2 Perceptrón multicapa

Perceptrón multicapa o red neuronal consiste en una serie de capas de neuronas interconectadas, específicamente, todas las neuronas de una capa se conectan a las neuronas de la siguiente capa.

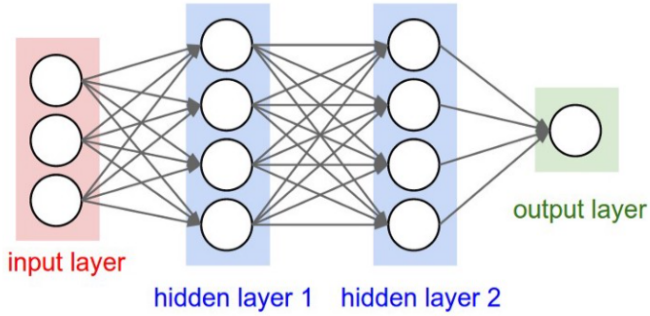

Figura 1.3: *Perceptrón multicapa (Fuente: [3])*

La figura anterior representa una red neuronal de 2 capas con 1 capa oculta. Contiene 3 neuronas de entrada, 2 neuronas en la capa oculta, y 1 neurona de salida. (Ver figura 1.3)

Nuestro cálculo comienza con la capa de entrada a la izquierda, de la cual pasamos valores a la capa oculta, de ahí, la capa oculta envía valores de salida a la última capa, que contiene el valor final.

Aunque pareciera que cada una de las tres neuronas de entrada envía múltiples valores de salida a la capa oculta, en realidad solamente hay un valor de salida por neurona. Las neuronas siempre producen un valor, independientemente de cuántas conexiones de salida tengan.

En una red neuronal nos encontramos con varios procesos (algoritmos), uno de ellos, es la propagación hacia adelante o forward propagation en inglés, mediante el cual una red neuronal envía su entrada a través de sus capas hacia la salida, aplicando en cada capa el algoritmo visto para el perceptrón simple, donde se define una sumatoria ponderada con los pesos y las entradas (salidas de la capa anterior) 'Z', la cual terminará sirviendo como insumo para la funcion de activacion de la capa actual. Las capas que ejecutan dicho algoritmo se conocen como capas ocultas, estas sirven para trazar una conexión directa entre nuestras entradas y nuestras salidas. En la mayoría de los problemas del mundo real, las variables de entrada tienden a ser altamente interdependientes

y afectan la salida de forma combinatoria y compleja. Las neuronas de las capas ocultas nos permiten capturar interacciones sutiles entre nuestras entradas.

Otra manera de interpretar el comportamiento de las capas ocultas, es la representación de características a nivel superior o atributos de nuestros datos. Cada una de las neuronas de una capa oculta evalúa sus entradas de diferente forma, y así, separa finalmente características de los datos.

Nuestra neurona de salida logra almacenar estas características intermedias, y no solo las entradas originales. De esta manera, al aumentar el número de capas ocultas, permitimos que la red neuronal obtenga información de varios niveles de abstracción.

[2] añade que, la encadenación de múltiples transformaciones no lineales a través de las capas, termina aumentando la flexibilidad y capacidad de expresión de la red neuronal sin perder información.

II.3 Backpropagation

En [4] se define Propagación hacia atrás, o backpropagation en inglés, como un tipo de red neuronal basada en el aprendizaje supervisado.
En este tipo de red neuronal, se presentan dos momentos importantes. Un primer momento del cual ya hemos hablado en este documento, la propagación hacia adelante. En el cual, resumiendo, se aplica un algoritmo a la entrada de la red, que se propaga desde la primera capa a través de las superiores hasta generar una salida. Esta es comparada con la salida deseada y se calcula una señal de error, lo que da paso al segundo momento, la propagación hacia atrás.
Partiendo de la capa de salida hacia todas las neuronas de la capa oculta anterior, se propaga una fracción de dicho error dependiendo de la contribución relativa que haya aportado esta neurona a la salida original.
Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describe su impacto en el error total.
Mediante este proceso se busca adaptar los pesos que hay entre las capas ocultas con el fin de aprender relaciones existentes entre las características de las entradas y sus salidas correspondientes.
El algoritmo de entrenamiento, backpropagation, tiende a desarrollar relaciones internas entre neuronas con el fin de organizar mejor los datos en clases

Para aplicar el algoritmo de propagación hacia atrás, se debe realizar primeramente la propagación hacia adelante en la nuestra red neuronal.
Recordemos del perceptrón simple que la salida de una neurona es igual a la aplicación de una función de activación utilizando como componente una suma ponderada de las entradas y sus respectivos pesos a cada neurona.

Como insumo para la explicación, nos basaremos en una red neuronal hipotética de N capas ocultas, con P neuronas cada una, y finalmente una neurona de salida.

$$a_k^{(n)} = f(z_k^{(n)})$$

$$z_k^{(n)} = b + \sum_k w_{ik}^{(n)} * x_k^{(n)}$$

Donde 'n' es la capa actual, 'k' es la neurona a la que se le está aplicando el proceso.
De esta manera $a_k^{(n)}$haría las veces de entrada para la siguiente capa oculta. La cual al aplicarle nuevamente el algoritmo, alimentaria la siguiente, y así, hasta llegar a la capa de salida.

$$Y_1 = a_1^N$$

Teniendo una salida Y_1se debe hacer una comparación junto con la salida Y deseada buscando encontrar el error.

$$Error\colon\ E\ = C(Y_1)$$

Donde C(Y) es una función de coste definida para el problema. La función de coste más común en este tipo de ejercicios es:

$$C(Y) = \frac{1}{2}(Yr - Y)^2$$

Después de encontrar el error actual, debemos actualizar los pesos utilizando la derivada de la función de coste, de la siguiente manera.
Para la última capa de la red neuronal.

$$\frac{\partial C(Y)}{\partial Y} = -(Yr - Y)$$

$$w_k^N = w_k^N + (LR) * \frac{\partial C(Y)}{\partial w_k^N}$$

$$\frac{\partial C(Y)}{\partial w_k^N} = \frac{\partial C(Y)}{\partial a_k^N} * \frac{\partial a_k^N}{\partial z_k^N} * \frac{\partial z_k^N}{\partial w_k^N}$$

$$\delta_1^N = \frac{\partial C(Y)}{\partial a_k^N} * \frac{\partial a_k^N}{\partial z_k^N}$$

$$a_1^{(N-1)} = \frac{\partial z_k^N}{\partial w_k^N}$$

$$\frac{\partial C(Y)}{\partial w_k^N} = \delta_1^N * a_1^{(N-1)}$$

$$w_k^N = w_k^N - (LR)* \delta_1^N * a_1^{(N-1)}$$

$$b^N = b^N - (LR) * \delta_1^N$$

Donde, LR es un factor de aprendizaje que determina la velocidad en la que la red neuronal aprende. El valor de este suele variar bastante de red neuronal a red neuronal, pues define la divergencia o convergencia de la misma. δ^NEs el impacto que tiene esa neurona en el resultado final, y por ende, en el error, es decir, es el error imputado a esa neurona.

Para cualquier capa de la red:

$$w_k^{N-j} = w_k^{N-j} - (LR)(\frac{\partial C(Y)}{\partial w_k^{N-j}})$$

$$\frac{\partial C(Y)}{\partial w_k^{N-j}} = \frac{\partial C(Y)}{\partial a_k^N} * \frac{\partial a_k^N}{\partial z_k^N} * \frac{\partial z_k^N}{\partial a_k^{N-1}} * \frac{\partial a_k^{N-1}}{\partial z_k^{N-1}} * \frac{\partial z_k^{N-1}}{\partial a_k^{N-2}} * \dots * \frac{\partial z_k^{N-j}}{\partial w_k^{N-j}}$$

$$\delta_K^{N-j} = \frac{\partial C(Y)}{\partial a_k^N} * \frac{\partial a_k^N}{\partial z_k^N} * \frac{\partial z_k^N}{\partial a_k^{N-1}} * \frac{\partial a_k^{N-1}}{\partial z_k^{N-1}}$$

$$\delta_K^{N-j} = \delta_1^N * w^N * \frac{\partial a_k^{N-1}}{\partial z_k^{N-1}}$$

$$w_k^{N-j} = w_k^{N-j} - (LR)(\delta_k^{N-j} * a_k^{(N-j-1)})$$

$$b^{N-j} = b^{N-j} - (LR) * \delta_1^{N-j}$$

Es importante tener en cuenta que:

$$x_k = a_k^0$$

Al aplicar el algoritmo de backpropagation, y obtener las derivadas parciales de la función de coste con respecto a los parámetros. Podemos aplicar el descenso del gradiente, con el fin de minimizar el error de la red y ajustar el error de la misma.

III. CONCLUSIONES

- Gracias a la implementación del algoritmo de backpropagation, se permitió la modificación de los pesos de las entradas de cada capa oculta con mayor facilidad, buscando acercar la salida a un resultado esperado.
- El número de capas ocultas que se implementen en una red neuronal, determina el nivel de abstracción que se podrá tener de la información de entrada, lo que se traduce, a la solución de problemas más complejos o de mayor número de características.
- La teoría de Redes Neuronales Artificiales, presenta grandes ventajas con respecto a otros modelos típicos de solución de problemas en Ingeniería, una de ellas tiene su inspiración en modelos biológicos del funcionamiento del cerebro.
  Los modelos matemáticos en que han sido desarrollados los algoritmos para todos los tipos de redes son modelos sencillos, que aunque exigen cierto grado de conocimientos de cálculo diferencial, pueden ser asimilados y desarrollados en cualquier lenguaje de programación.
  La red tipo Perceptrón es una red que puede implementarse exitosamente para resolver problemas de clasificación de patrones que sean linealmente separables, la red responderá mejor entre más sencillos sean los patrones que debe clasificar.
  Una red de Perceptrón multicapa está en capacidad de generar regiones de decisión arbitrariamente complejas; aunque en ciertos problemas se puede simplificar el aprendizaje aumentando capas ocultas, la tendencia es aumentar la extensión de la función de activación, en lugar de la complejidad de la red.

IV. REFERENCIAS

IV.1 Referencias en la Web:

[1]
TAZ-TFG-2018-148.pdf (unizar.es)

[2]
Redes Neuronales (ml4a.github.io)

[3]

Redes neuronales. Programa de Visión… | by Bootcamp AI | Medium

[4]
Quispe Gavino., Giancarlo, Cornejo Ruiz, Daniel. (2018). Aplicación del algoritmo Backpropagation de redes neuronales para determinar los niveles de morosidad en los alumnos de la Universidad Peruana Unión. 1-3