Perceptrón, entrenamiento y el backpropagation
Perceptron, training and backpropagation

Autores: Kevin Mauricio Carmona Loaiza
Jorge Miguel Coronado Marin
IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correos: kevin.carmona@utp.edu.co
miguel.coronado@utp.edu.co

***Resumen*— En este documento se proporciona información de una manera clara y concisa acerca de lo que es el perceptrón, como entrenarlo y por consiguiente la retropropagación, el cual es un método para entrenar dichas neuronas.**
Hay que tener en cuenta que todo esto se efectúa a la base de red neuronal y sus aplicaciones como algoritmos para manipularla.

Palabras clave*— *perceptrón, neurona, retropropagación, redes, algoritmos.

Abstract- This paper provides information in a clear and concise manner about what the perceptron is, how to train it and therefore backpropagation, which is a method to train such neurons.
It should be noted that all this is done on the basis of neural network and its applications as algorithms to manipulate it.

Keywords*— *perceptron, neuron, backpropagation, networks, algorithms.

I. INTRODUCCIÓN

Las redes de neuronas, o redes neuronales (en inteligencia artificial), son un paradigma de procesamiento y aprendizaje automático inspirado en cómo funciona el sistema nervioso de los animales. Es un sistema de conexiones de neuronas que trabajan juntas para producir una salida.

El fin del presente documento es explicar y especificar cada uno de los temas mencionados, Como sabemos, la inteligencia artificial tiene como objetivo imitar la inteligencia natural, por lo que en este caso las redes neuronales tienen como objetivo imitar la "arquitectura" del cerebro (conexiones neuronales) que le dan inteligencia a los humanos. Es cierto que el término inteligencia es bastante ambiguo, pero en este caso donde estamos hablando de redes neuronales estamos buscando inteligencia para aquellas personas que son capaces de conocer un tema en particular y tener una respuesta a un problema en particular dar (cálculo, clasificación, etc.); Por ejemplo, una vez que sabemos algo sobre los animales, los humanos podemos transformar un animal (a través de una fotografía) en un mamífero, pájaro, reptil, etc. es decir; que de alguna manera aprendimos la diferencia entre estos animales y cómo clasificarlos. Este es un ejemplo de lo que puede hacer una red neuronal artificial bien entrenada (o aprendida) al colocar una foto de un animal en ella. Obviamente, esta red neuronal puede fallar al igual que los humanos, pero la idea es que una red neuronal bien entrenada sea capaz de predecir de la misma manera que lo hacen los humanos.

I.1 PERCEPTRON

Primero que todo empezaremos definiendo o dando una definición de lo que significa el perceptrón. En el campo de las redes neuronales, creado por Frank Rosenblatt, se refiere a la neurona artificial o unidad básica de inferencia en forma de discriminador o clasificador lineal, a partir de lo cual se desarrolla un algoritmo capaz de generar un criterio para seleccionar un subgrupo a partir de un grupo de componentes más grandes.

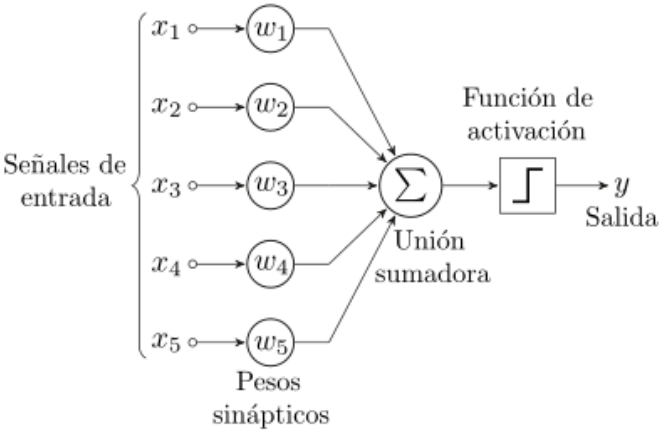

Diagrama de un perceptrón con cinco señales de entrada

El perceptrón puede utilizarse con otros tipos de perceptrones o de neurona artificial para formar una red neuronal artificial más compleja.

Para consolidar lo antes mencionado hacemos la comparación con el área de la biología, el modelo biológico más simple de un perceptor es una neurona y viceversa.

Es decir, el modelo matemático más simple de una neurona es un perceptrón. La neurona es una célula especializada y caracterizada por poseer una cantidad indefinida de canales de entrada llamados dendritas y un canal de salida axón. las dendritas operan como sensores que recogen información de la región donde se hallan y la derivan hacia el cuerpo de la neurona que reacciona mediante una sinapsis que envía una respuesta hacia el cerebro, esto en caso de los seres vivos.

Una neurona sola y aislada carece de razón de ser, su labor especializada se torna valiosa en la medida en que se asocia a otras neuronas, formando una red. Normalmente el axón de una neurona entrega su información como "señal de entrada" a una dendrita de otra neurona y así se repite el proceso sucesivamente. El perceptrón que capta la señal en adelante se extiende formando una red de neuronas, sean éstas biológicas o de sustrato semiconductor (compuertas lógicas).

Dicho perceptrón usa una matriz para representar las redes neuronales y es un discriminador terciario que traza su entrada x (un vector binario) a un único valor de salida f(x) (un solo valor binario) a través de dicha matriz.

$$f(x) = \begin{cases} 1 & \text{si } w \cdot x - u > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Donde w es un vector de pesos reales y w.x es el producto escalar (que computa una suma ponderada). u es el 'umbral', el cual representa el grado de inhibición de la neurona, es un término constante que no depende del valor que tome la entrada.

El valor de f (x) (0 o 1) se usa para clasificar x como un caso positivo o un caso negativo, en el caso de un problema de clasificación binario. El umbral puede entenderse como una manera de compensar la función de activación, o una forma de fijar un nivel mínimo de actividad a la neurona para considerarse como activa. La suma ponderada de las entradas debe producir un valor mayor que u para cambiar la neurona de estado 0 a 1.

I.2 APRENDIZAJE O ENTRENAMIENTO

En el perceptrón, existen dos tipos de aprendizaje, el primero utiliza una tasa de aprendizaje mientras que el segundo no la utiliza. Esta tasa de aprendizaje amortigua el cambio de los valores de los pesos.

El algoritmo de aprendizaje es el mismo para todas las neuronas, todo lo que sigue se aplica a una sola neurona en el aislamiento. Se definen algunas variables primero:

- x(j) denota el elemento en la posición j en el vector de la entrada
- w(j)el elemento en la posición j en el vector de peso
- y denota la salida de la neurona
- δ denota la salida esperada
- α es una constante tal que $0 < \alpha < 1$

Los dos tipos de aprendizaje difieren en este paso. Para el primer tipo de aprendizaje, utilizando tasa de aprendizaje, utilizaremos la siguiente regla de actualización de los pesos:

$w(j)' = w(j) + \alpha(\delta - y)\, x(j)$

Para el segundo tipo de aprendizaje, sin utilizar tasa de aprendizaje, la regla de actualización de los pesos será la siguiente:

$w(j)' = w(j) + (\delta - y)x(j)$

Por lo cual, el aprendizaje es modelado como la actualización del vector de peso después de cada iteración, lo cual sólo tendrá lugar si la salida y difiere de la salida deseada δ . Para considerar una neurona al interactuar en múltiples iteraciones debemos definir algunas variables más:

- x_i denota el vector de entrada para la iteración i
- w_i denota el vector de peso para la iteración i
- y_i denota la salida para la iteración i
- $D_m = \{ (x_1, y_1), \ldots, (x_m, y_m) \}$ denota un periodo de aprendizaje de m iteraciones

En cada iteración el vector de peso es actualizado como sigue:

- Para cada pareja ordenada (x , y) {\displaystyle (x,y)} (x,y) en $D_m = \{ (x_1, y_1), \ldots, (x_m, y_m) \}$ {\displaystyle D_{m}=\{(x_{1},y_{1}),\dots ,(x_{m},y_{m})\}} $D_m = \{(x_1,y_1),\ldots.(x_m,y_m)\}$

Pasar (x_i, w_i, y_i) a la regla de actualización $w(j)' = w(j) + \alpha(\delta - y)x(j)$

El periodo de aprendizaje D_m se dice que es separable linealmente si existe un valor positivo γ, y un vector de peso w tal que: $y_i \cdot (\langle w, x_i \rangle + u) > \gamma$

Novikoff (1962) probó que el algoritmo de aprendizaje converge después de un número finito de iteraciones si los datos son separables linealmente y el número de errores está limitado a: $(2R/\gamma)^2$

Sin embargo, si los datos no son separables linealmente, la línea de algoritmo anterior no se garantiza que converja.

I.3 BACKPROPAGATION

Se define el backpropagation como un método de cálculo del gradiente utilizado en algoritmos de aprendizaje supervisado utilizados para entrenar redes neuronales artificiales.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta sólo reciben una fracción de la señal total del error, basándose

aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

En el corazón de backpropagation hay una expresión para la derivada parcial $\frac{\partial C}{\partial w}$ de la función de coste C con respecto a cualquier peso (o sesgo) en la red. La expresión nos dice qué tan rápido cambia el coste cuando cambiamos los pesos y los sesgos. Y aunque la expresión es algo compleja, también tiene un cierto encanto, ya que cada elemento tiene una interpretación natural e intuitiva. Entonces, backpropagation no es solo un algoritmo rápido para el aprendizaje; en realidad, nos brinda información detallada sobre cómo cambiar los pesos y los sesgos cambia el comportamiento general de la red.

UN ENFOQUE RÁPIDO PARA CALCULAR LA SALIDA DE UNA RED NEURONAL

Antes de analizar en profundidad la backpropagation, hagamos un calentamiento con un algoritmo rápido basado en matriz para calcular la salida de una red neuronal. De hecho, ya vimos brevemente este algoritmo cerca del final del anterior capítulo, pero se describió rápidamente, por lo que vale la pena revisarlo en detalle. En particular, esta es una buena manera de sentirse cómodo con la notación utilizada en backpropagation, en un contexto familiar. Comencemos con una notación que nos permite referirnos a los pesos en la red de una manera no ambigua. Usaremos w^l_{jk} para denotar el peso de la conexión de la neurona k-ésima en la capa a la neurona j en la capa l.

Entonces, por ejemplo, el siguiente diagrama muestra el peso en una conexión desde la cuarta neurona en la segunda capa a la segunda neurona en la tercera capa de una red:

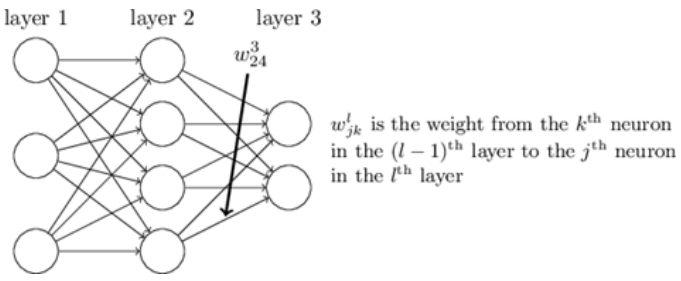

Figura 1. Introducción a la notación de índices de los pesos de una red neuronal.

Una peculiaridad de la notación es el ordenamiento de los índices j y k. Puede pensar que tiene más sentido usar j para referirse a la neurona de entrada, y k a la neurona de salida, no viceversa, como se hace realmente. Explicaremos la razón de este a continuación. Utilizamos una notación similar para los sesgos y activaciones de la red. Explícitamente, usamos b^l_j para el sesgo de la neurona j-ésima en la capa l-ésima. Y usamos a^l_j para la activación de la neurona j-ésima en la capa l-ésima. El siguiente diagrama muestra ejemplos de estas anotaciones en uso:

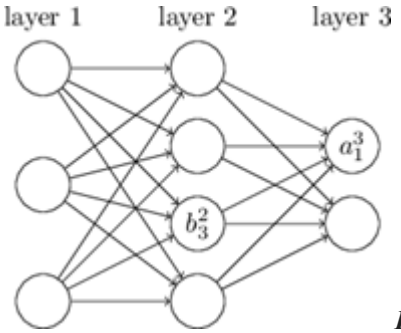

Figura 2: Ejemplo de anotación

Con estas notaciones, la activación a^l de la neurona j-ésima en la capa l-ésima está relacionada con las activaciones en la capa (l-1)-ésima por la ecuación:

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

donde el sumatorio está sobre todas las neuronas k en la capa $(l-1)$. Para reescribir esta expresión en forma de matriz, definimos una matriz de peso w^l para cada capa l. Las entradas de la matriz de peso w^l son sólo los pesos que se conectan a la l-ésima capa de neuronas, es decir, la entrada en la fila j y la columna k es w^l. Del mismo modo, para cada capa l definimos un vector de bias, b^l. Probablemente pueda adivinar cómo funciona esto: los componentes del vector de sesgo son solo los valores b^l, un componente para cada neurona en la capa l-ésima. Y finalmente, definimos un vector de activación a^l cuyos componentes son las activaciones a^l_j. El último ingrediente que necesitamos para reescribir la fórmula anterior en forma de matriz es la idea de vectorizar una función como σ. Queremos aplicar una función como σ a cada elemento en un vector v. Utilizamos la notación obvia $\sigma(v)$ para denotar este tipo de aplicación de elementos de una función. Es decir, los componentes de $\sigma(v)$ son solo $\sigma(v)_j = \sigma(v_j)$. Como ejemplo, si tenemos la función $f(x) = x^2$, entonces la forma vectorizada de f tiene el efecto:

$$f\left(\begin{bmatrix}2\\3\end{bmatrix}\right) = \begin{bmatrix}f(2)\\f(3)\end{bmatrix} = \begin{bmatrix}4\\9\end{bmatrix}$$

Podemos, con esta notación en mente, escribir la ecuación que nos interesa como:

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

Esta expresión nos da una forma mucho más global de pensar acerca de cómo las activaciones en una capa se relacionan con las activaciones en la capa anterior: simplemente aplicamos la matriz de peso a las activaciones, luego agregamos el vector de polarización y finalmente aplicamos la función σ. Esa visión global a menudo es más fácil y más sucinta (¡e implica menos índices!) que la visión de neurona por neurona que hemos visto ahora. Piense en ello como una forma de escapar del infierno del índice, sin dejar de ser preciso sobre lo que está sucediendo.

Las dos suposiciones que tenemos que realizar sobre la función de coste:

El objetivo de backpropagation es calcular las derivadas parciales $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$ de la función de coste C con respecto a cualquier peso w o sesgo b en la red.

Para que el backpropagation funcione, debemos hacer dos suposiciones principales sobre la forma de la función de coste. Sin embargo, antes de exponer esas suposiciones, es útil tener en mente una función de coste de ejemplo. Utilizaremos la función de coste cuadrática:

$$C(w,b) = \frac{1}{2n}\Sigma_x \parallel y(x) - a^L(x) \parallel^2$$

w denota la colección de todos los pesos en la red, b todos los sesgos, n es el número total de entradas de entrenamiento, a^L es el vector de salidas de la red cuando se ingresa x, y la suma es sobre todas las entradas de entrenamiento, x. L Es el número de capas que tiene nuestra red neuronal.

Bien, entonces, ¿qué suposiciones debemos hacer sobre nuestra función de coste, C para poder aplicar backpropagation? La primera suposición que necesitamos es que la función de coste puede escribirse como un promedio $C = \frac{1}{n}\Sigma_x C_x$ sobre las funciones de coste C_x para un ejemplo de entrenamiento individual. Este es el caso de la función de coste cuadrático, donde el coste de un solo ejemplo de entrenamiento es $C_x = 1/2 \parallel y - a^L \parallel^2$. Este supuesto también será válido para todas las demás funciones de coste que encontraremos.

La razón por la que necesitamos esta suposición es porque lo que backpropagation realmente nos permite hacer es calcular las derivadas parciales $\frac{\partial C_x}{\partial w}$ y $\frac{\partial C_x}{\partial b}$ para un solo ejemplo de entrenamiento. Luego recuperamos $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$ promediando los ejemplos de entrenamiento. De hecho, con esta suposición en mente, supondremos que el ejemplo de entrenamiento x ha sido arreglado, y eliminaremos el subíndice x, escribiendo el coste C_x como C. Eventualmente volveremos a colocar la x, pero por ahora es una notación molestia que es mejor dejar implícita.

La segunda suposición que hacemos sobre el coste es que se puede escribir en función de los resultados de la red neuronal:

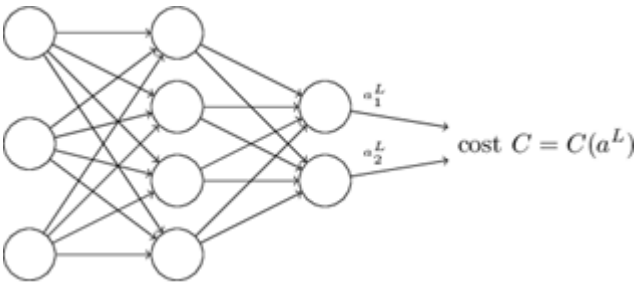

Figura 3. Construcción de la función de coste como función del output de una red neuronal.

Por ejemplo, la función de costo cuadrático cumple este requisito, ya que el coste cuadrático para un solo ejemplo de entrenamiento x puede escribirse como:

$$C_x = \frac{1}{2} \parallel y - a^L(x) \parallel^2 = \frac{1}{2}\Sigma_j(y_j - a_j^L)$$

y por lo tanto es una función de las activaciones de salida. Por supuesto, esta función de costo también depende de la producción deseada y, y puede que se pregunte por qué no consideramos el coste también en función de y. Sin embargo, recuerde que el ejemplo de entrenamiento de entrada x es fijo, por lo que la salida y también es un parámetro fijo. En particular, no es algo que podamos modificar cambiando los pesos y los sesgos de ninguna manera, es decir, no es algo que la red neuronal aprenda. Por lo tanto, tiene sentido considerar C como una función de las activaciones de salida a^L sólo, con y simplemente un parámetro que ayuda a definir esa función.

4 ECUACIONES FUNDAMENTALES

Backpropagation se basa en cuatro ecuaciones fundamentales. Juntas, esas ecuaciones nos dan una manera de calcular tanto el error δ_j^l como el gradiente de la función de coste. A continuación, se declaran las cuatro ecuaciones.

BP1: Una ecuación del error en la capa de salida, δ^L.

$$\delta_j^l = \frac{\partial C}{\partial a_j^L}\sigma'(z - j^L)$$

BP2: Una ecuación del error δ^l en términos del error de la siguiente capa, $\delta(l+1)$.

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

BP3: Una ecuación para calcula la tasa de cambio del coste con respecto a cualquier bias de la red,

$$\delta^l_j = \frac{\partial C}{\partial b^l_j}$$

BP4: una ecuación para la tasa de cambio de la función de coste con respecto a cualquier peso en la red,

$$\frac{\partial C}{\partial w^l_{j\,k}} = \left(a^{l-1}_k - \delta^l_j\right)$$

REFERENCIAS

Referencias en la Web:

1. https://elvex.ugr.es/decsai/computational-intelligence/slides/N2%20Backpropagation.pdf
2. https://empresas.blogthinkbig.com/como-funciona-el-algoritmo-backpropagation-en-una-red-neuronal-parte-ii/
3. https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s
4. https://es.wikipedia.org/wiki/Perceptr%C3%B3n
5. https://jarroba.com/introduccion-a-las-redes-neuronales-el-perceptron-video/