ALGORITMO DE ENTRENAMIENTO PERCEPTRÓN

Nicolás Rodríguez Fernández

Facultad de Ingeniería de Sistemas, Universidad Tecnológica de Pereira, Pereira, Colombia

e-mail: nickoro094@utp.edu.co

RESUMEN: Este documento presenta el método de entrenamiento de redes neurales llamado Perceptrón, se muestra cómo funciona, su arquitectura de diseño y por qué es usado para el entrenamiento de redes neuronales. Éste trabajo está hecho con el objetivo de visualizar el funcionamiento de las partes involucradas en el entrenamiento de inteligencias artificiales basadas en redes neuronales.

PALABRAS CLAVES: **Perceptrón, Red Neuronal, Inteligencia Artificial.**

ABSTRACT: ***This document presents the neural network training method called Perceptron, it shows how it works, its design architecture and why it is used for the training of neural networks. This work is done with the objective of visualizing the functioning of the parties involved in the training of artificial intelligences driven by neural networks.***

KEYWORDS*: Perceptron, Neural Network, Artificial Intelligence.*

1. INTRODUCCIÓN

Las redes neuronales y el aprendizaje profundo son temas importantes en Ciencias de la computación y en la industria de la tecnología; actualmente brindan las mejores soluciones para muchos problemas en el reconocimiento de imágenes, el reconocimiento de voz y el procesamiento del lenguaje natural. En la actualidad se han publicado muchos artículos sobre la IA que pueden aprender a pintar, construir modelos en 3D, crear interfaces de usuario, algunos crear imágenes y hay muchas más cosas increíbles que se hacen todos los días utilizando redes neuronales. Éstas redes neuronales necesitan de un algoritmo de entrenamiento para llegar a la etapa de madurez que les permite ser lo que son el día de hoy y uno de los métodos pioneros y más utilizados es el algoritmo de entrenamiento Perceptrón.

2. ARQUITECTURA

2.1. Qué es Perceptrón

Perceptrón es un clasificador lineal (binario). Además, se utiliza en el aprendizaje supervisado. Ayuda a clasificar los datos de entrada en una red neuronal. El Perceptrón es un algoritmo para aprender un clasificador binario llamado función de umbral: una función que mapea su entrada x (un vector de valor real) a un valor de salida *f(x)* (un solo valor binario).

2.2. Definición

$$f(x) = \begin{cases} 1 & si\ w \cdot x + b > 0, \\ 0 & en\ otro\ caso \end{cases}$$

Donde *w* es un vector de pesos con valores reales, $\sum_{i=1}^{m} w_i x_i$ *w* . *x* es el producto punto , donde *m* es el número de entradas del Perceptrón, y *b* es el *bias.* El *bias* cambia el límite de la decisión moviéndolo lejos del origen y no depende de ningún valor de entrada.

El valor de *f (x)* (0 o 1) se usa para clasificar *x* como una instancia positiva o negativa, en el caso de un problema de clasificación binaria. Si *b* es negativo, entonces la combinación ponderada de entradas debe producir un valor positivo mayor que | b | con el fin de empujar la neurona clasificadora sobre el umbral de 0. Espacialmente, el sesgo altera la posición (aunque no la orientación) del límite de decisión.

El algoritmo de aprendizaje de Perceptrón no termina si el conjunto de aprendizaje no es linealmente separable. Si los vectores no son linealmente separables, el aprendizaje nunca alcanzará un punto donde todos los vectores se clasifiquen correctamente.

2.3. Funcionamiento

El Perceptrón consta de 4 partes.

- Valores de entrada o una capa de entrada
- Pesos y Bias
- Función de activación
- Suma neta

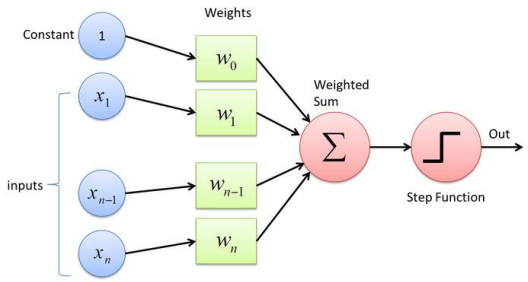


Figura 1. Perceptrón

El Perceptrón funciona siguiendo los pasos:

a) Todas las entradas *x* se multiplican con sus pesos *w*. Llamémoslo *k*.

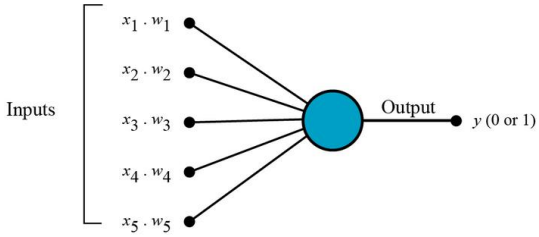


Figura 2. Multiplicando entradas por pesos (5 entradas).

b) Sumar todos los valores obtenidos, a esto lo llamamos Sumatoria de Pesos.

$$\sum_{k=1}^{5} k$$

c) Aplicamos la sumatoria de pesos a la función de activación correcta

$$f(x) = \begin{cases} 0 & si\ 0 > x \\ 1 & si\ x \geq 0 \end{cases}$$

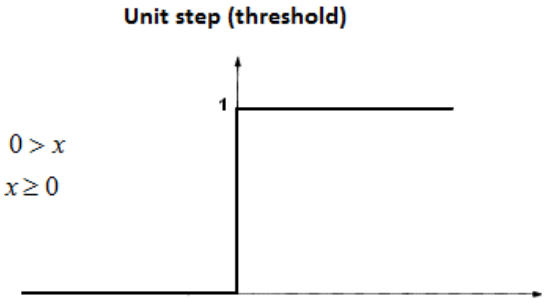


Figura 3. Función de Activación

2.4. Pesos y bias

Los pesos muestran la fuerza del nodo particular.

Un valor de bias le permite cambiar la curva de la función de activación hacia arriba o hacia abajo.

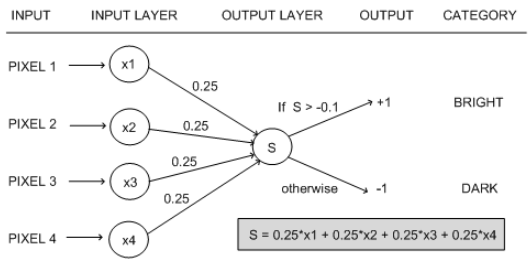


Figura 4. Ejemplo de Perceptron mostrando pesos y bias.

2.5. Función de Activación

La función de activación es usualmente una abstracción representando una tasa de potencial de activación gatillándose en la celda. En su forma simplificada, esta función es binaria, esto es, se activa la neurona o no.

En resumen, las funciones de activación se utilizan para asignar la entrada entre los valores requeridos como (0, 1) o (-1, 1).

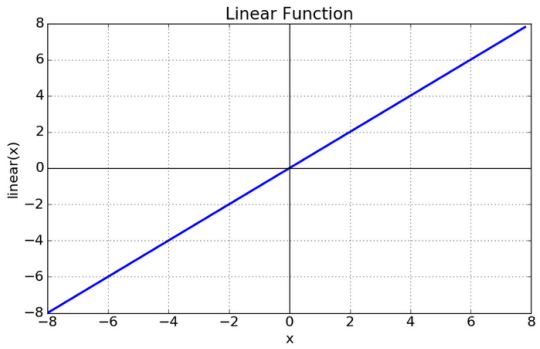


Figura 5. Función de activación lineal

2.6. Convergencia

El Perceptrón es un clasificador lineal, por lo tanto, nunca llegará al estado con todos los vectores de entrada clasificados correctamente si el conjunto de entrenamiento *D* no es linealmente separable, es decir, si los ejemplos positivos no pueden separarse de los ejemplos negativos por un hiperplano. En este caso, ninguna solución "aproximada" se abordará gradualmente bajo el algoritmo de aprendizaje estándar, sino que el aprendizaje fallará por completo. Por lo tanto, si la separabilidad lineal del conjunto de entrenamiento no se conoce a priori, se debe usar una de las variantes de entrenamiento a continuación.

Si el conjunto de entrenamiento es linealmente separable, entonces se garantiza que el Perceptrón converge. Además, hay un límite superior en el número de veces que el Perceptrón ajustará sus pesos durante el entrenamiento.

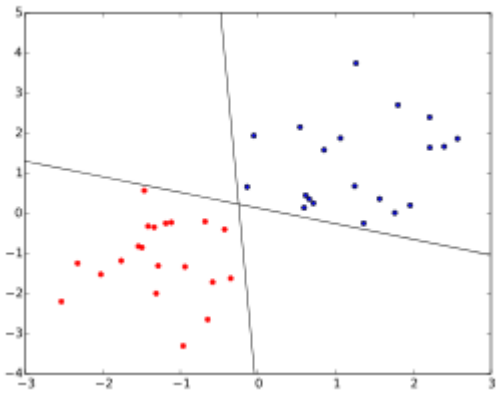


Figura 6. Dos clases de puntos y dos de los infinitos límites lineales que los separan. A pesar de que los límites están casi en ángulo recto entre sí, el algoritmo de percepción no tiene forma de elegir entre ellos.

Si bien se garantiza que el algoritmo de Perceptrón convergerá en alguna solución en el caso de un conjunto de entrenamiento separable linealmente, todavía puede elegir cualquier solución y los problemas pueden admitir muchas soluciones de calidad variable.

3. EJEMPLO

Nuestro objetivo es encontrar el vector que pueda clasificar perfectamente las entradas positivas y negativas en nuestros datos. Para ello tenemos el siguiente algoritmo:

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

Inicializamos *w* con algún vector aleatorio. Luego iteramos sobre todos los ejemplos en los datos, (P U N) ejemplos positivos y negativos. Ahora bien, si una entrada *x* pertenece a *P*, idealmente, ¿cuál debería ser el producto de puntos *w.x*? Podemos decir que es mayor o igual a 0 porque eso es lo único que nuestro perceptrón quiere al final del día, así que démosle eso. Y si *x* pertenece a *N*, el producto punto debe ser menor que 0. Entonces, si miramos las condiciones if en el bucle while:

while $!convergence$ **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end

Caso 1: cuando x pertenece a P y su producto de puntos w.x <0

Caso 2: cuando x pertenece a N y su producto de puntos w.x ≥ 0

Solo para estos casos, estamos actualizando nuestra *w* inicializada al azar. De lo contrario, no tocamos *w* en absoluto porque el Caso 1 y el Caso 2 están violando la misma regla de un perceptrón. Por lo tanto, estamos sumando *x* a *w* (ej., vector de suma) en el Caso 1 y restando *x* de *w* en el Caso 2.

Ya hemos establecido que cuando x pertenece a *P*, queremos *w.x> 0*, la regla de percepción básica. Lo que también queremos decir con esto es que cuando x pertenece a P, el ángulo entre w y x debe ser menor de 90 grados.

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||} \quad \Big| \quad cos\alpha \propto \mathbf{w}^T\mathbf{x}$$

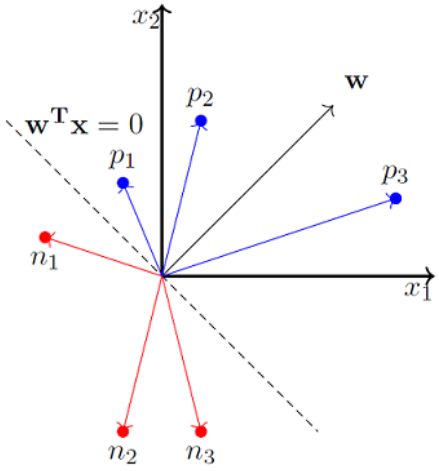
Por lo que, si

$$\mathbf{w}^T\mathbf{x} > 0 \quad \Rightarrow cos\alpha > 0 \quad \Rightarrow \alpha < 90$$

Y similarmente, si

$$\mathbf{w}^T\mathbf{x} < 0 \quad \Rightarrow cos\alpha < 0 \quad \Rightarrow \alpha > 90$$

Entonces, cualquiera que sea el vector w, siempre que forme un ángulo de menos de 90 grados con los vectores de datos de ejemplo positivos (x EP) y un ángulo de más de 90 grados con los vectores de datos de ejemplo negativos (x EN), vamos por buen camino . Así que idealmente, debería verse algo como esto:



Así que ahora creemos firmemente que el ángulo entre w y x debe ser menor que 90 cuando x pertenece a la clase P y el ángulo entre ellos debe ser mayor que 90 cuando x pertenece a la clase N. Aquí es por qué la actualización funciona:

(α_{new}) when $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w}+\mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha + \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &> cos\alpha \end{aligned}$$

(α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w}-\mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} - \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha - \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &< cos\alpha \end{aligned}$$

Entonces, cuando estamos agregando x a w, lo que hacemos cuando *x* pertenece a *P* y *wx <0* (Caso 1), esencialmente estamos aumentando el valor cos (), lo que significa que estamos disminuyendo el valor alfa, el ángulo entre w y x, que es lo que deseamos. Y la intuición similar funciona para el caso cuando *x* pertenece a N y w.x ≥ 0 (Caso 2).

Aquí hay una simulación de cómo podemos terminar aprendiendo con un ángulo de menos de 90 para ejemplos positivos y más de 90 para ejemplos negativos.

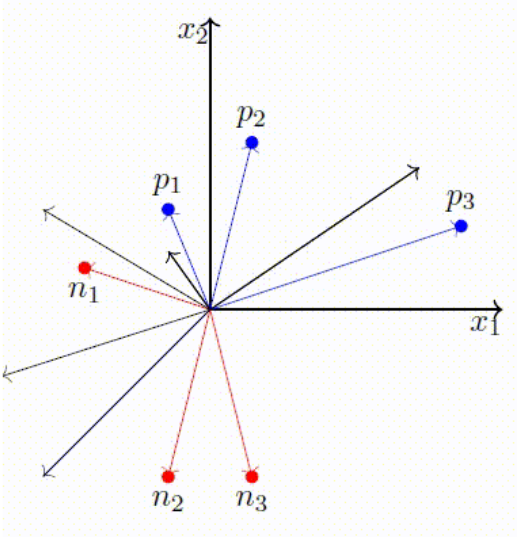


Figura 8. Simulación Perceptrón

4. CONCLUSIONES

- Una neurona artificial es una función matemática concebida como un modelo de neuronas biológicas, es decir, una red neuronal.
- Un Perceptrón es una unidad de red neuronal que realiza ciertos cálculos para detectar características o inteligencia

empresarial en los datos de entrada. Es una función que asigna su entrada "x", que se multiplica por el coeficiente de peso aprendido, y genera un valor de salida "f (x).

- La Regla de aprendizaje de Perceptrón establece que el algoritmo aprenderá automáticamente los coeficientes de peso óptimo.
- Los Perceptrones de una sola capa pueden aprender solo patrones separables linealmente.

5. BIBLIOGRAFÍA

[1] Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory.

[2] Minsky M. L. and Papert S. A. 1969. Perceptrons. Cambridge, MA: MIT Press.

[3] Widrow, B., Lehr, M.A., "30 years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," Proc. IEEE, vol 78, no 9 (1990).

[4] https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975