Redes neuronales Perceptron conceptos

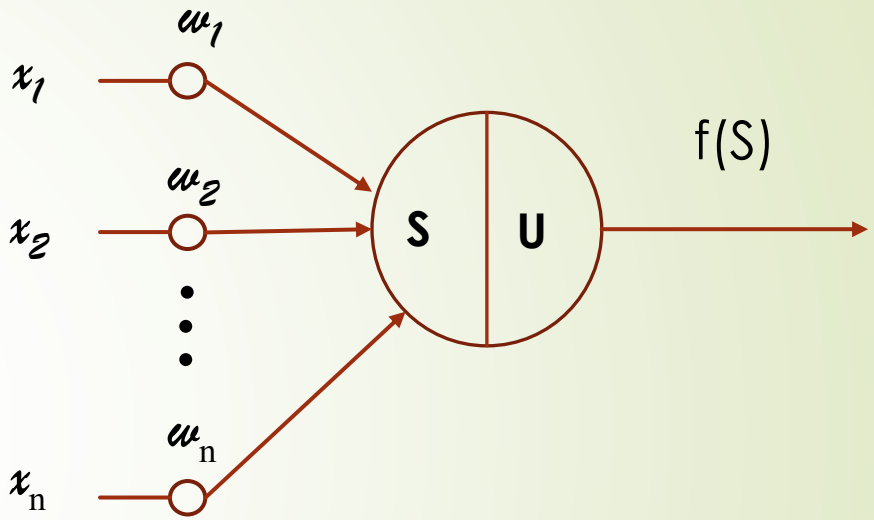
Jesús Alejandro Flores Hernández – enero 2016

Neurona

Una neurona es una célula que recibe un conjunto de entradas **(*x*1,*x*2,…,*xn*)**, multiplica cada entrada por un peso **(*w*1,*w*2,…,*wn*)** con lo que tenemos un conjunto de entradas ponderadas; suma las entradas ponderadas y si superan un Umbral laa neurona transmite uns señal se salida, no la transite en otro caso.

Neurona

$$S(x_1, x_2, x_n) = \sum x_i * w_i$$



Donde x_i son las entradas.
w_i son los pesos sinápticos
U es el umbral

$$f(S) = \text{signo}((\sum a_i * w_i) - U)$$

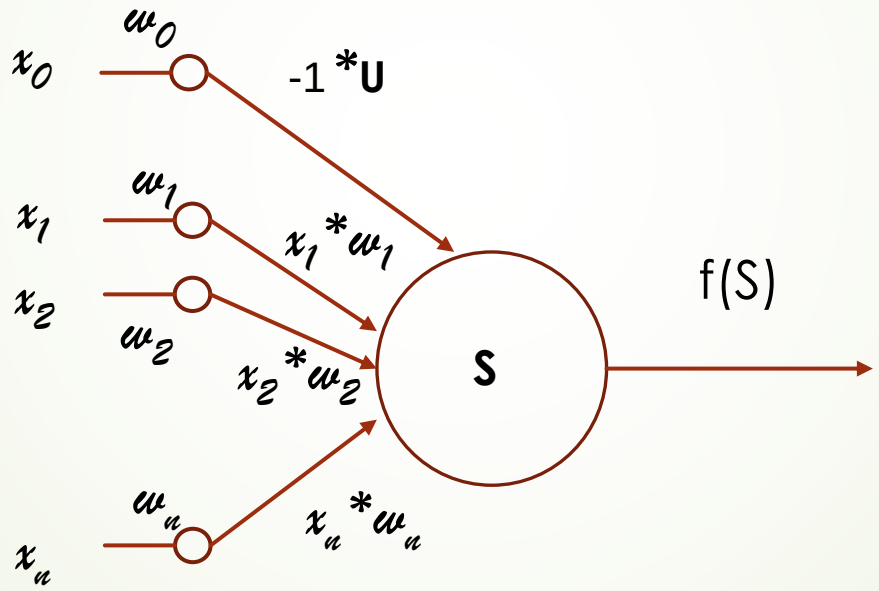
$$f = \begin{cases} 1 \text{ si } a_1*w_1 + a_2*w_2 \ldots + a_n*w_n > U \\ -1 \text{ si } a_1*w_1 + a_2*w_2 \ldots + a_n*w_n \leq U \end{cases}$$

Neurona simplificada

El umbral se puede ver como una entrada mas con un valor x_0=-1 y U= w_0.

x *Son las entradas*

w *Son las pesos de cada entrada*



$$S=\sum x_i * w_i$$

$$f(s)=\begin{cases} 1 & \text{si } S>0 \\ -1 & \text{si } S \leq 0 \end{cases}$$

F puede ser una función que mapea a la función sigmoide [0,1] o a la función tangente hiperbólica [-1,1]

Perceptrón

El **perceptrón** es uno de los algoritmos más simples y fundamentales en el campo del aprendizaje automático y las redes neuronales. Fue introducido por Frank Rosenblatt en 1957 y es la base de modelos más complejos, como las redes neuronales multicapa.

El perceptrón es un algoritmo de clasificación binaria, es decir, se utiliza para separar datos en dos clases. Es un tipo de **red neuronal de una sola capa** (sin capas ocultas) y funciona como un clasificador lineal.

Estructura

El perceptrón toma un conjunto de entradas, las combina linealmente usando pesos y produce una salida binaria (0 o 1). Su estructura es la siguiente:

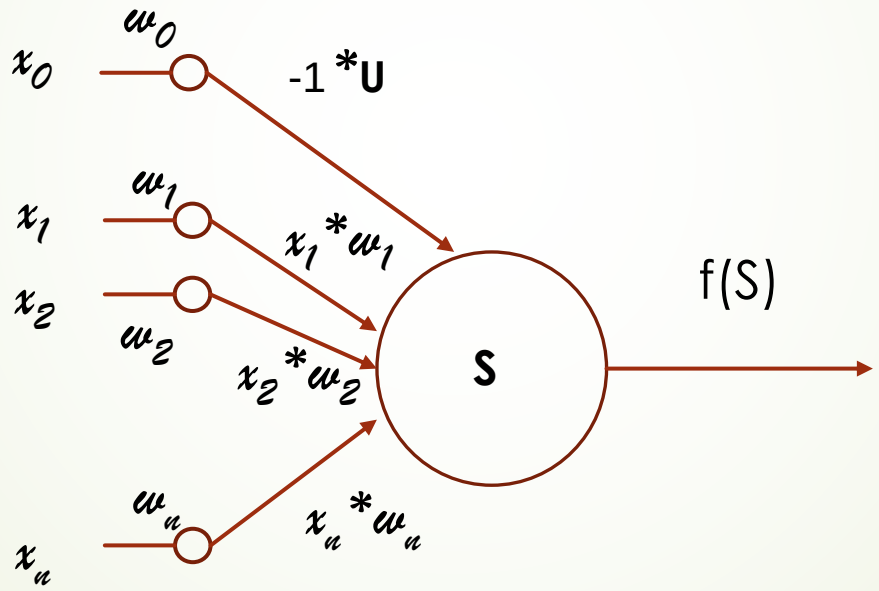
- **Entradas ($x1,x2,\ldots,xn$)**: Son las características (features) del dato.
- **Pesos ($w1,w2,\ldots,wn$)**: Son los parámetros que el perceptrón aprende. Cada peso está asociado a una entrada.
- **Sesgo (bias) o (b)**: Es un término adicional que permite ajustar el modelo.
- **Función de activación**: Una función que decide la salida del perceptrón. En el caso clásico, es una función escalón (step function).

Redes neuronales

El umbral se puede ver como una entrada mas con un valor x_0=-1 y U= w_0.

x *Son las entradas*

w *Son las pesos de cada entrada*



$$S = \sum x_i * w_i$$

$$f(s) = \begin{cases} 1 & \text{si } S > 0 \\ -1 & \text{si } S \leq 0 \end{cases}$$

F puede ser una función que mapea a la función sigmoide [0,1] o a la función tangente hiperbólica [-1,1]

Perceptron simple

Supongamos que tenemos una función f de R^n en $\{-1,1\}$, que aplica un patrón de entrada $\mathbf{x} = (x_1,x_2,\ldots,x_n)^T \in R^n$ en la salida deseada $z \in \{-1,1\}$, es decir, $f(\mathbf{x}) = z$. La información de que disponemos sobre dicha función viene dada por p pares de patrones de entrenamiento

$$\{\mathbf{x}^1,z^1\}, \{\mathbf{x}^2,z^2\},\ldots,\{\mathbf{x}^p,z^p\}$$

donde $\mathbf{x}^i \in R^n$ y $f(\mathbf{x}^i) = z^i \in \{-1,1\}$, $i=1,2,\ldots,p$. Dicha función realiza una partición en el espacio R^n de patrones de entrada; por una parte estarían los patrones con salida +1 y por otra parte los patrones con salida −1. Por lo tanto, diremos que la función f clasifica a los patrones de entrada en dos clases. Ejemplos de funciones f de este tipo son la función lógica OR o la función par.

Para el entrenamiento Debemos:

- Determinar los pesos sinápticos (wi)
- Determinar el umbral U (w0)

Esto lo realizaremos en un proceso iterativo calculando en cada iteración wi.

Entrenamiento

Para la determinación de los pesos sinápticos y del umbral vamos a seguir un proceso adaptativo que consiste en comenzar con unos valores iniciales aleatorios e ir modificándolos iterativamente cuando la salida de la unidad no coincide con la salida deseada. La regla que vamos a seguir para modificar los pesos sinápticos se conoce con el nombre de **regla de aprendizaje del Perceptrón simple** y viene dada por la expresión:

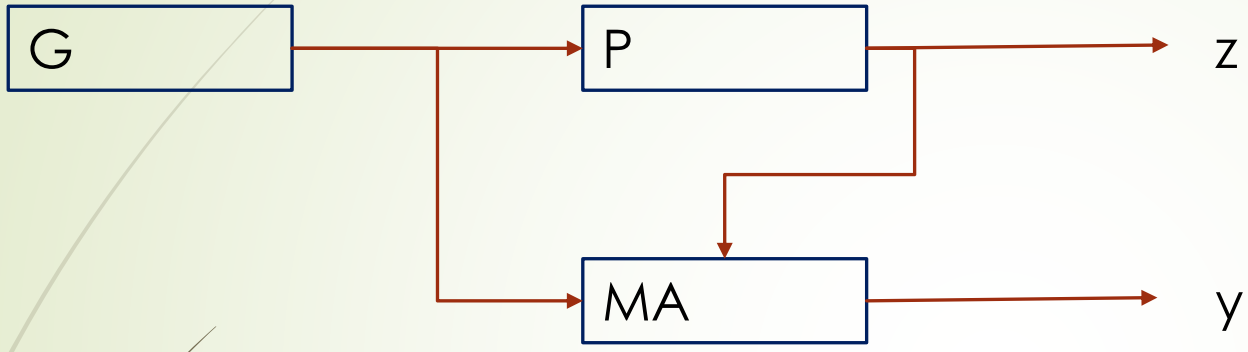
$$w_j(k+1) = w_j(k) + \Delta w_j(k), \quad k = 1,2,\ldots$$

siendo

$$\Delta w_j(k) = \eta(k)[z(k) - y(k)]x_j(k)$$

Donde
k es la iteración actual
y
K+1 es la siguiente iteración

Recordemos que en una máquina de aprendizaje



z es la salida esperada, **y** es la salida que nos da la máquina de aprendizaje.
El error de la MA en la k-esima iteración será:

z(k)-y(k)

Por lo tanto para calcular los nuevos pesos en la siguiente iteración:

$$w_j(k+1) = w_j(k) + \Delta w_j(k), \quad k = 1,2,\ldots$$

siendo

$$\Delta w_j(k) = \eta(k)[z(k) - y(k)]x_j(k) \tag{3}$$

esto nos indica que la variación del peso w_j es proporcional al producto del error $z(k) - y(k)$ por la componente *j*-ésima del patrón de entrada que hemos introducido en la iteración *k*, es decir, $x_j(k)$. La constante de proporcionalidad $\eta(k)$ es un parámetro positivo que se llama **tasa de aprendizaje** puesto que cuanto mayor es más se modifica el peso sináptico y viceversa. Es decir, es el parámetro que controla el proceso de aprendizaje. Cuando es muy pequeño la red aprende poco a poco. Cuando se toma constante en todas las iteraciones, $\eta(k) = \eta > 0$ tendremos la **regla de adaptación con incremento fijo**.

El ajuste de pesos (aprendizaje)en cada iteración será:

$$w_j(k+1) = \begin{cases} w_j(k) + 2\eta x_j(k) & \text{si } y(k) = -1 \text{ y } z(k) = 1, \\ w_j(k) & \text{si } y(k) = z(k) \\ w_j(k) - 2\eta x_j(k) & \text{si } y(k) = 1 \text{ y } z(k) = -1 \end{cases} \quad (4)$$

Por lo tanto, esta regla de aprendizaje es un método de detección del error y corrección. Solo aprende, es decir, modifica los pesos, cuando se equivoca. Cuando tenemos un patrón que pertenece a la primera clase ($z(k)=1$) y no es asignado a la misma, entonces corrige el valor del peso sináptico añadiéndole una cantidad proporcional al valor de entrada, es decir lo *refuerza*, mientras que si el patrón de entrada no pertenece a esta clase y el Perceptrón lo asigna a ella, lo que hace es *debilitar* el peso restándole una cantidad proporcional al patrón de entrada . No modificaremos los pesos cuando el valor deseado coincida con la salida de la red.

Umbral (sesgo)

¿Cómo se modifica el sesgo? De la misma manera, teniendo en cuenta que el sesgo se puede considerar como el peso sináptico correspondiente a un nuevo sensor de entrada que tiene siempre una entrada igual a $x_{n+1}=-1$, y como peso sináptico el valor del umbral, pues

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq \theta \quad \Leftrightarrow \quad w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_{n+1}x_{n+1} \geq 0$$

cuando $w_{n+1}=\theta$ y $x_{n+1}= -1$. Así, la red equivalente tendría $n+1$ sensores, su umbral sería siempre cero, los patrones de entrada $(x_1, x_2, \ldots, x_n)$ serán ahora $(x_1, x_2, \ldots, x_n, -1)$, los pesos asociados $(w_1, w_2, \ldots, w_n)$ serán $(w_1, w_2, \ldots, w_n, w_{n+1})$ con $w_{n+1} = \theta$ y la regla de aprendizaje:

$$\Delta\theta(k) = -\eta(k)[z(k) - y(k)], \quad k=1,2,\ldots$$

Algoritmo de aprendizaje del Perceptrón con entrenamiento individualizado

Paso 0: Inicialización

Inicializar los pesos sinápticos con números aleatorios del intervalo [-1,1]. Ir al paso 1 con k=1

Paso 1: (k-ésima iteración)

Calcular

$$y(k) = \operatorname{sgn}\left(\sum_{j=1}^{n+1} w_j x_j(k)\right)$$

Paso 2: Corrección de los pesos sinápticos

Si $z(k) \neq y(k)$ modificar los pesos sinápticos según la expresión:

$$w_j(k+1) = w_j(k) + \eta[z(k) - y(k)]x_j(k), \quad j = 1,2,\ldots,n+1$$

Paso 3: Parada

Si no se han modificado los pesos en las últimas p iteraciones, es decir,

$$w_j(r) = w_j(k), \quad j = 1,2,\ldots,n+1, \quad r = k+1,\ldots,k+p,$$

parar. La red se ha estabilizado.

En otro caso, ir al **Paso 1** con k=k+1.

Aplicaciones

Problemas linealmente separables