Informática y comunicaciones

ies azarquiel

CURSO ESPECIALIZACIÓN INTELIGENCIA ARTIFICIAL Y BIG DATA

Tema 6: ADALINES

Profesor: Sebastián Rubio Valero

Diciembre 2025

ARTIFICIAL INTELLIGENCE

1 Inconvenientes del perceptrón

En el perceptrón sólo podemos tener salidas cuyo valor son $(0-1,+1)$. Esto es así por que la función de activación escalón o signo. Lo cual implica que el perceptrón no puede producir una probabilidad ni un valor continuo. Simplemente clasifica la entrada en una de las categorías binarias posibles (en este caso representadas como -1, $+1$ o 0 para un caso límite o de no-clasificación), lo que significa que solo es útil para tareas de clasificación binaria simple.

Por otro lado el perceptrón es lento de convergencia ya que tiene que ajustar los pesos para encontrar el límite de decisión correcto.Esto en conjunto de datos muy gran requiere muchas iteraciones.

Otro inconveniente que podemos encontrar en los perceptrones, es que no acepta soluciones aproximadas, como sucede en el algoritmo de regresión logística. El perceptrón está diseñado para encontrar un solución exacta que separe perfectamente las dos clases si y sólo si los datos son linealmente separables (es decir, si existe una línea recta que puede dividir las dos clases sin error). Si el conjunto de datos no es perfectamente separable, el algoritmo del perceptrón nunca converge y no puede encontrar una solución aproximada con un margen de error aceptable. Simplemente seguirá ajustando sus pesos indefinidamente.

2 ADALINES: Neuronas Adaptativas Lineales

Fueron creadas en 1960 por Bernad Widrow y Marcian Hoff y se distinguen del perceptrón en lo siguiente:

1. Las entradas y salida pueden ser continuas
2. El entrenamiento se realiza por paquetes, no de forma iterativa
3. Los pesos se actualizan minimizando una función de coste global de tipo cuadrático: el valor cuadrático del error para el número total de pares de entrenamiento.

Estas características hacen que el entrenamiento sea mucho más rápido, y que pueda ofrece soluciones aproximadas en el caso de no encontrar la óptima, además utiliza el método del descenso del gradiente para redes multicapa.

El comportamiento de las ADALINES es el mismo que vimos en la regresión lineal

$$y(t) = \mathbf{w^T}.\mathbf{x} = \mathbf{x^T}.\mathbf{w}$$

Para medir el error cometido haremos exactamente igual que la regresión lineal, usaremos el error cuadrático.

$$\epsilon_k^2 = (d_k - \mathbf{w_k^T}.\mathbf{x_k})^\mathbf{2}$$

Este error se minimiza por el descenso del gradientes. Es decir

$$\mathbf{w_{k+1}} = \mathbf{w_k} + \mu(-\nabla_\mathbf{n}\epsilon_\mathbf{k}^\mathbf{2})$$

donde k es el indice de iteración y μ es la tasa de aprendizaje. El nuevo vector de pesos, $\mathbf{w_{k+1}}$, se obtiene sumando a cada una de las coordenadas del vector anterior, $\mathbf{w_k}$ el correspondiente vector gradientes $\frac{\partial \epsilon_k^2}{\partial \mathbf{w_k}}$, con lo que podemos reescribir la fórmula anterior de la siguiente forma

$$\mathbf{w_{k+1}} = \mathbf{w_k} - \mu(\frac{\partial \epsilon_\mathbf{k}^\mathbf{2}}{\partial \mathbf{w_k}})$$
$$\mathbf{w_{k+1}} = \mathbf{w_k} + \mu.\epsilon_\mathbf{k}.\mathbf{x_k}$$

Si observamos la fórmula final, las entradas, $\mathbf{x^n}$, y la salida, d^n, no dependen de los pesos, lo que constituyen pares de entrenamiento.

newline Hay dos formas de aplicar el método de los mínimos cuadrados propuesto por Widrow y Hoff:

1. Un entrenamiento iterativo, usando el gradiente instantáneo en cada par ($\mathbf{x^n}$, d^n) para obtener $\mathbf{w_{k+1}} = \mathbf{w_k} + \mathbf{2}.\mu.\epsilon_\mathbf{k}.\mathbf{x_k}$

2. En un entrenamiento no iterativo, por paquetes, acumulando primero los cambios en w que deberían introducirse para cada uno de los pares ($\mathbf{x^n}$, d^n), obteniendo su valor medio para el conjunto de los P pares
   $\mathbf{w_{final}} = \mathbf{w_{inicial}} + \mathbf{valormedio}[\mathbf{2}.\mu.\epsilon_\mathbf{k}.\mathbf{x_k}]$

Los pasos a seguir en el entrenamiento son análogos a los descritos para el perceptrón, sólo que ahora el error es analógico y, además, en el entrenamiento por paquetes tenemos que acumular los errores individuales y calcular el error cuadrático medio.
Este método puede aplicarse a cualquier red de ADALINES, la única condición es que todas las salidas este disponibles para poder compararlas con la salidas deseadas.

$$\mathbf{Y} = \mathbf{W^t}.\mathbf{x}$$
$$y_j = \sum_{i=1}^{M} w_{i,j}.x_i \text{ donde } j = 1..N$$
$$y_B = \sum_{i=1}^{2} w_{B,i}.x_i = w_{B.1}x_1 + w_{B,2}x_2$$

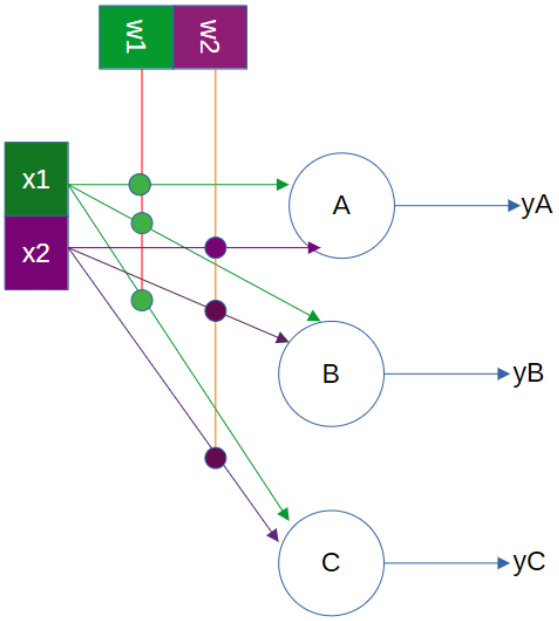

Figure 1: Enter Caption

Como se puede observar en la imagen, los pesos se distribuyen de igual forma para cada neurona. Esto dá lugar a una matriz de pesos de dimensión MxN, donde M es el número de características,y N es el número de neuronas. El el entrenamiento de una ADALINE queda de la siguiente forma

3 Entrenamiento ADALINES Sigmodeas

Hemos visto el entrenamiento de las adelines lineales por el método del descenso del gradiente del error cuadrático medio para un conjunto de entrenamiento $\mathbf{x^n}, \mathbf{d^n}$. Si sustituimos la función de activación lineal por una sigmoide, obtenemos una adaline no lineal a la que se puede entrenar usando también el método del descenso del gradiente sólo que ahora, al introducir una función de decisión de tipo sigmoide la expresión final del valor de los incrementos en los pesos se modifica en un factor $u' = u(1-u)$

Recordemos que el incremento en los pesos se hace proporcional al gradiente del cuadrado del error

$$\Delta w = -\mu\nabla(\tfrac{1}{2}\epsilon^2) = -\mu\epsilon\frac{\partial(d-y)}{\partial w} = \mu\epsilon\frac{\partial y}{\partial w}$$

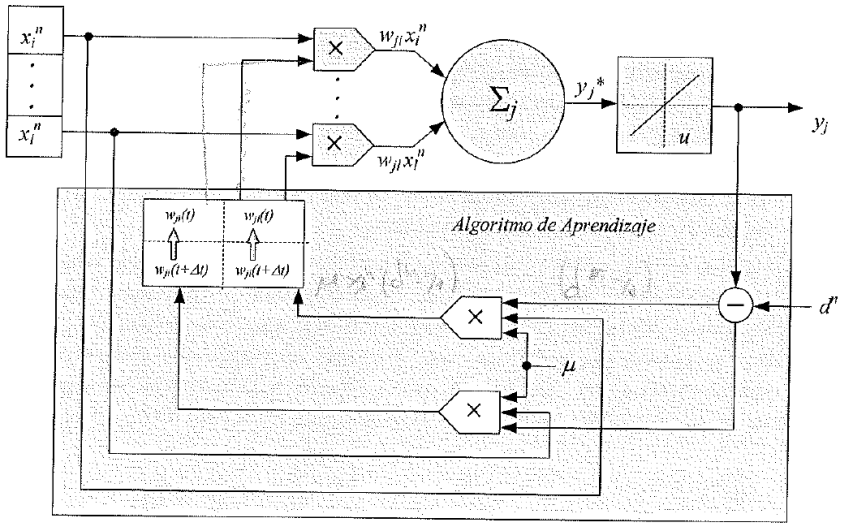

Figure 2: Enter Caption

$$\text{Como } y = u(y^*)$$
$$\frac{\partial y}{\partial w} = \frac{\partial u}{\partial y^*}\frac{\partial y^*}{\partial w}$$
$$\frac{\partial y^*}{\partial w} = \frac{\partial w^t x}{\partial w} = x$$
$$\text{Como } u(y^*) = \frac{1}{1+e^{-y^*}}$$
$$\text{derivando } u'(y^*) = \frac{\partial u}{\partial y^*} = u(y^*)[1-u(y^*)]$$
$$\Delta w = \mu.\epsilon.u(1-u).x = \delta.\epsilon.x$$
$$\text{donde } \delta = \mu.u.(1-u)$$

Ejemplo del calculo del incremento

Entrada: $x_1 = 2, x_2 = 3$
Pesos: $w_1 = 0.5, w_2 = -0.3$ sesgo $w_0 = 0.1$
$t = 0.9$ valor deseado de salida
$\mu = 0.1$
$y^* = (0.5)(2) + (-0.3)(3) + (0.1).1 = 0.2$
$y = u(y^*) = \frac{1}{1+e^{-y^*}} = \frac{1}{1+e^{-0.2}} \approx 0.55 = u$
$\Delta w_i = \mu.\epsilon.u(1-u)x_i$
$\epsilon = t - y; \epsilon = 0.9 - 0.55 = 0.35$
$u(1-u) = 0.55(1-0.55) \approx 0.2475$
$\delta = (0.1)(0.35)(0.2475 \approx 0.00866$

$\Delta w_1 = \delta.w_1 = (0.00866).2 \approx 0.01732$
$\Delta w_2 = \delta.w_2 = (0.00866).3 \approx 0.02598$
$\Delta w_0 = \delta.w_0 = (0.00866).1 = 0.00866$
Los nuevos pesos serán
$w_1(t+1) = (0.5) + 0.01732 = 0.51732$
$w_2(t+1) = (-0.3) + 0,02598 = -0.27402$
$w_0(t+1) = 0.1 + 0.00866 = 0.10866$

El término $u(1-u)$ que es la derivada de la función sigmoide ($\frac{\partial u}{\partial y^*}$), actúa como un factor de amortiguación o modular en la actualización de los pesos. Su valor siempre está entre 0 y $0,25$, que son las pendientes mínima y máxima respectivamente. La pendiente es máxima cuando $y = 0.5$. Esto ocurre cuando las entradas netas es aproximadamente cero. La neurona está en su región de máxima sensibilidad y tiene mayor incertidumbre sobre la clase a la que pertenece la entrada. Si $y \approx 0$ o $y \approx 1$, ocurre cuando el input neto es un valor muy positivo o muy negativo. La neurona está saturada y tiene muy alta confianza en su predicción. En este punto la derivada es casi cero, lo que provoca que el ajuste de pesoso Δw sea mínimo, incluso si el error es muy grande. La neurona "aprende lento" por que el cambio en su entrada no produce un cambio significativo en su salida.

Ejercicio calculo pesos iterativo

Dadas las siguientes entradas $\{(x_1, x_2, y)\} = \{(3, 3, 0.9), (1, 2, 0.2)\}$ donde y es la característica objetivo, la tasa de aprendizaje $\mu = 0.2$. Los pesos inicialmente valen $w_0 = -1$,$w_1 = 2$ y $w_2 = 1$ Obtener como quedan los pesos después de entrenar la neurona ADELINE sigmoide con las dos entradas.

3.1 Caso de la ADELINE clásica

La ADELINE utiliza la función de activación lineal (función identidad), donde $u(y^*) = y^*$:

$$\text{Su derivada es: } \frac{\partial u}{\partial y^*} = 1$$
$$\text{Por lo tanto: } \Delta w = \mu.\epsilon.1.x$$

El ajuste de los pesos depende directamente y linealmente del error (ϵ), sin ninguna amortiguación. En contraste la neurona logística añade la dependencia lineal $u(1-u)$, lo que permite modelar la no linealidad, y es la base para construir redes neuronales multicapa complejas.

El problema se complica cuando tenemos que entrenar unidades cuya salida no es accesible desde el exterior (capas "ocultas"). En este caso, necesitamos un algoritmo para hacer llegar a estas unidades ocultas la parte correspondiente del error que se detecta en la capa de salida. Este problema lo resuelve el método de la retropropagación del gradiente que veremos en el siguiente tema.

3.2 Redes Neuronales Multicapa

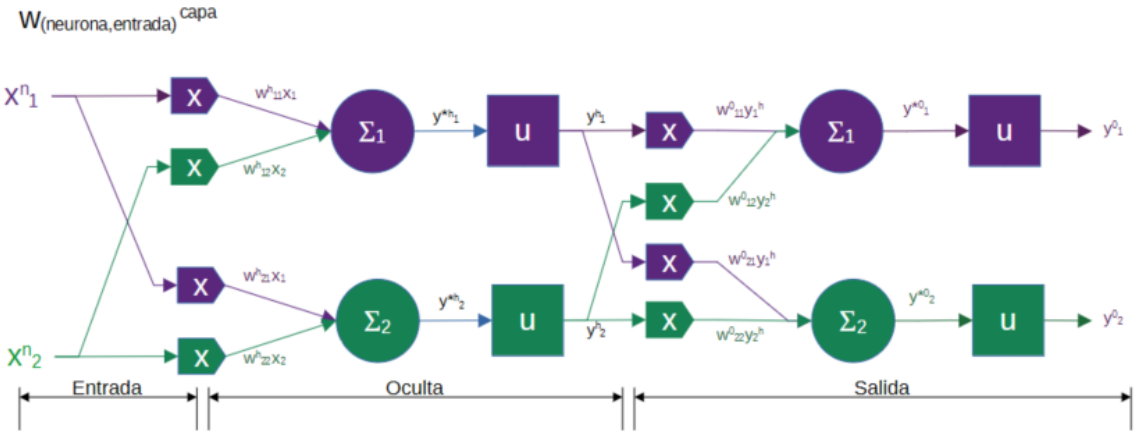

Figure 3: Enter Caption