

Tema 7: Redes neuronales- Conceptos básicos

- + Cada neurona u unidad (neurona artificial), se conecta a otras mediante arcos dirigidos
- + Cada arco  $i \rightarrow j$  conecta la salida de  $i$  (llamada  $a_i$ ) con las entradas de  $j$ .
- + El arco  $i \rightarrow j$  tiene un peso asociado  $w_{ij}$ , que determina la fuerza y signo de la conexión.
- + Cada unidad calcula su salida en función de sus entradas  
# El cálculo es muy simple
- + La red recibe una serie de entradas externas (unidades de entrada) y devuelve al exterior las salidas de algunas de sus neuronas (unidades de salida)

+ Cálculo por unidades

$$a_j = g(\sum_{i=0}^n w_{ij} \cdot a_i)$$

$g$  = función de activación

$\sum_{i=0}^n w_{ij} \cdot a_i$  ( $a_i$ ) = sumatoria de todas las unidades  $i$  que emiten su salida a la unidad  $j$  por el peso del enlace  $i \rightarrow j$

# Para  $i=0$  se considera que  $a_0=1$  y el peso lo determina el umbral ( $w_{0j}$ )

+ Umbral y funciones de activación

\* El umbral  $w_{0j}$  de cada unidad se interpreta como el apunte de una cantidad cuya entrada debe superar (hacer el límite para el cambio de resultado)

\* Función de activación: Función no lineal para aumentar la expresividad del modelo (que no sea lineal y se puedan dar distintos resultados)

\* Funciones comunes

$\text{Umbral } (x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$	
$\text{Sigmoid: } \sigma(x) = \frac{1}{1+e^{-x}}$	
$\text{ReLU } (x) = \max\{0, x\}$ (su derivada es la función umbral)	
$\text{Tangente hiperbólica: } \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	

## - Redes neuronales hacia adelante

\* Compuesta por capas: Entradas, intermedias y salidas

\* Las unidades de una misma capa tienen la misma función de activación

\* La función de activación de la capa de salida variará según el funcionamiento que se le quiera dar a la red

Regresión (predicción valores): sin función de activación

Clasificación binaria: una sola unidad en la capa de salida con función umbral o sigmoidal

Clasificación multiclase: sin función de activación (se aplica softmax)

$$\# \text{softmax } (a_1, \dots, a_m) = \frac{1}{\sum_k e^{a_k}} (e^{a_1}, \dots, e^{a_m})$$

; se interpreta que las unidades con mayor salida es la que indica el valor de clasificación

## + Aprendizaje

El aprendizaje o entrenamiento consiste en encontrar los pesos de las conexiones de manera que la red se comporte como dicta el conjunto de entrenamiento

\* Aprendizaje supervisado hacia adelante

Dada un conjunto de entrenamiento  $D = \{(\vec{x}_d, \vec{y}_d); x_d \in \mathbb{R}^n, y_d \in \mathbb{R}^m, d=1, \dots, k\}$ ; y una red que conocemos solo su estructura (capas, unidades por capa, conexiones) y su función de activación, encontrar los pesos  $w_{ij}$  para que se ajuste a los ejemplos del conjunto de entrenamiento.

## + Percepciones

Un percepción es el caso más simple de una red neuronal, sola una capa de entrada y una de salida. Un percepción con función de activación umbral puede representar las funciones lógicas (And, OR y NOT).

\* Limitaciones expresivas

Un percepción con  $n$  entradas, pesos  $w_i$  ( $i=0, \dots, n$ ) y función de activación umbral clasifica como positivas a aquellas  $(x_1, \dots, x_n)$ , tal que  $\sum_{i=0}^n w_i x_i > 0$  (siendo  $x_0 = 1$ )

# La activación sigmoidal suaviza los resultados del umbral

## + Algoritmo de entrenamiento del Percepción (umbral)

Entrada: conjunto de int.  $D$  con ejemplos  $(\vec{x}, y)$ ,  $\vec{x} \in \mathbb{R}^n$  e  $y \in \{0, 1\}$  y factor de aprendizaje  $\eta$

1º Generar los pesos iniciales aleatoriamente:

$$\vec{w} \leftarrow (w_0, w_1, w_2, \dots, w_n)$$

2º Bucle hasta condición de parada

Para cada ejemplo del conjunto  $D(\vec{x}, y)$ :

1) Calcular  $\sigma = \text{umbral} (\sum_{i=0}^n w_i x_i)$  (con  $x_0 = 1$ )

2) Para cada peso  $w_i \leftarrow w_i + \eta (y - \sigma) x_i$

3º Devolver  $\vec{w}$

Tema 7: redes neuronales+ Puntualidades sobre el entrenamiento del Perceptrón (umbral)

\*  $\eta$  (factor de aprendizaje) = constante que medien las actualizaciones.

\* En cada actualización si  $y=1$  y  $o=0$ ,  $y-o=1>0$ , por lo que los pesos  $w_i$  con  $x_i$  positivos aumentarán y las negativas disminuirán, aproximándose a la salida real, para lo mismo cuando  $y=0$  y  $o=1$ .

\* Mientras los ejemplos sean linearmente separables y  $\eta$  sea suficientemente pequeño el algoritmo converge, para eso, el criterio de parada debe ser que la clasificación final sea correcta para todos los ejemplos.

+ Algoritmo de entrenamiento (Descenso por gradiente)

Si el conjunto de entrenamiento no es linearmente separable, no sirve el algoritmo con umbral. Por lo que se utilizan otros tipos de algoritmos, para encontrar la salida esperada.

Este algoritmo busca los pesos para conseguir la salida esperada, en base a minimizar el error cuadrático.

$$E(\vec{w}) = \frac{1}{2} \sum_d [y_d - g(w_0x_0 + w_1x_1 + \dots + w_nx_n)]^2 \doteq \frac{1}{2} \sum_d (y_d - o_d)^2$$

# Siendo  $g$  la función de activación,  $y_d$  la salida real y  $o_d$  la salida esperada (obtenida). En este método buscamos aproximar el peso mediante un descenso gradual del gradiente.

$$w_i \leftarrow w_i + \Delta \vec{w}; \text{ siendo } \Delta \vec{w} = -\eta \nabla E(\vec{w}); \text{ siendo } \nabla E(\vec{w}) \text{ la derivada de } E \text{ para cada vector } w_i$$

La función se resume en:

$$w_i \leftarrow w_i + \eta \sum_d (y_d - o_d) g'(in^{(d)}) x_{i,d}$$

El algoritmo de búsqueda de pesos es similar al del umbral

\* Tipos de descenso del gradiente

batch: recorre todos los ejemplos para actualizar las pesos

estocástico: recorre los ejemplos tratandolos uno a uno

## + Descenso estocástico por el gradiente (Regla Delta)

En vez de tratar el error cuadrático cometido sobre todos los ejemplos del conjunto, se trata sobre el ejemplo  $(\bar{x}, \bar{y}) \in D$  que se está tratando en cada momento (uno a uno).

# La actualización de los pesos se llama Regla Delta

$$w_i \leftarrow w_i + \eta (\bar{y} - \bar{o}) g'(in) x_i \quad \# \text{ De forma batch analizábamos } \sum_d (\bar{y}_d - \bar{o}_d) g'(\bar{in}^d) x_{i,d} \\ \text{ es decir, computábamos el valor \bar{o} en el conjunto total}$$

# En cada iteración los ejemplos se toman en un orden aleatorio

- \* Casos particulares (Regla Delta)
- activación lineal ( $g$ ) constante: como es lineal su derivada será constante, por tanto:  $w_i \leftarrow w_i + \eta (\bar{y} - \bar{o}) x_i$
  - activación sigmoidal ( $g$ ):  $w_i \leftarrow w_i + \eta (\bar{y} - \bar{o}) o(1 - o)x_i$

## + Puntualidades sobre el algoritmo de descenso por el gradiente

- \* Búsqueda de mínimos locales, no globales
- \* Con un  $\eta$  suficientemente pequeño el método estocástico se acerca arbitrariamente a la resolución del batch
- \* En la actualización con Regla Delta se consiguen resultados de manera más simple, necesitando valores más pequeños para  $\eta$ . Escapa más fácil de mínimos locales.
- \* Unbral busca valor exacto y necesita de separación lineal entre hiperplano de las salidas; mientras Regla Delta buscar una aproximación asintótica y solo necesita un modelo de regresión.

## - Redes multicapa (hacia adelante)

Partiendo de los conceptos que sabemos del perceptrón, la idea de añadir capas nos permitirá aumentar la expresividad de los modelos (entradas, salidas y ocultas)

## + Entrenamiento de redes multicapa

Al igual que con el perceptrón partimos de un conjunto de entrenamiento  $D$ , tal que  $(\bar{x}, \bar{y}) \in D$ , contiene una salida  $\bar{y} \in \mathbb{R}^m$  para  $\bar{x} \in \mathbb{R}^n$  como entrada.

El algoritmo para calcular los pesos se basa en la idea del perceptrón y se llama retropropagación, basado en ~~basado en~~ el algoritmo de descensos por el gradiente

## + Notación

Suponemos una red neuronal con  $n$  entradas,  $m$  salidas y  $L$  capas en total

\* Capa de entrada es 1 y capa de salida  $L$

\* Cada unidad de la capa  $l$  está conectada con todas las unidades de la capa  $l+1$

\* La función de activación es  $g_j$ , puede ser distinta para cada capa

\*  $w_{ij}$ , es el peso de la conexión  $i$  y  $j$

\*  $in_i$ , es la entrada que recibe  $i$ ;  $o_i$ , es la salida que prepara  $i$

\* Ejemplo — Si  $i$  es una unidad de entrada,  $o_i = x_i$  (su salida es el propio valor)

— Si  $i$  es una unidad distinta a la de entrada,  $in_i = \sum_j w_{ji} o_j$  y  $o_i = g_i(in_i)$

## + Algoritmo de retropropagación

### \* Idea inicial

Dada un ejemplo  $(\bar{x}, \bar{y}) \in D$ , y una unidad  $i$  de salida; la actualización es igual que como se hace con la Regla Delta.

$$\Delta_i = g_L'(in_i)(y_i - o_i) \Rightarrow W_{ji} \leftarrow W_{ji} + \eta a_j \Delta_i$$

Como en las capas ocultas no sabemos el valor real ( $y_i$ ), la idea es calcular el error hacia atrás, desde la capa  $L$  hasta la 1. Este error se denota  $\Delta_j$  y resulta de calcular el error de  $L-1$  a partir del de  $L$

$$\Delta_j = g'_j(in_j) \sum_i w_{ji} \Delta_i \Rightarrow W_{kj} \leftarrow W_{kj} + \eta a_k \Delta_j$$

Las salidas se calculan propagando valores hacia adelante desde las unidades de entrada, mientras que el error es al revés

### \* Desarrollo del algoritmo

Dada un conjunto de entrenamiento  $D$ , con ejemplos de la forma  $(\bar{x}, \bar{y})$  con  $\bar{x} \in \mathbb{R}^n$  y  $\bar{y} \in \mathbb{R}^m$ , un factor de aprendizaje  $\eta$  y una estructura de red

1º Damos valores aleatorios a las pesos de la red (cerca de 0, positivos y negativos)

2º Inicializamos el bucle hasta criterio de parada:

2.1) Realizamos propagación hacia adelante para obtener todos los valores de salida de cada unidad ( $a_i$ )

\* Para calcular  $a_i$ , si  $i$  es una unidad de la capa de entrada  $a_i = x_i$ , si es de otra capa:

Para cada  $i$  hasta  $i=L$ :

Calculamos  $in_i = \sum_j w_{ji} a_j$  y calculamos la salida  $a_i = g_i(in_i)$   
#  $j$  son todas las unidades de la capa anterior

2.2) Calcular los errores de cada unidad  $\Delta_i$  y actualizar los pesos propios

\* Primero calculamos el error de cada unidad  $i$ :

$$\Delta_i = g'_i(in_i)(y_i - o_i)$$

Para cada unidad desde  $L-1$  hasta 1 decrementando, calculamos el error de  $j$ :

$$\Delta_j = g'_j(in_j) \sum_i w_{ij} \Delta_i \quad # \text{Sumatoria iterativa para los valores de entrada } i$$

Para cada unidad de la capa  $L+1$ , siendo  $j=L+1$  e  $i=L$ :

$$W_{ij} = W_{ij} + \eta a_i \Delta_j$$

3º Devuelves la red

### + Momentum de retropropagación

Una manera de evitar el problema de los mínimos locales es añadir un sumando a la actualización de los pesos, que tenga en cuenta la actualización realizada en ~~la~~ la iteración anterior.

### + Criterio de parada en retropropagación

Como el entorno se hace tan sólo ejemplos de forma aleatoria, este se puede reanudar en cualquier momento sin una parada.

Se puede usar parámetros prefijados para cada uno de estos, num. iteraciones, ...