INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES:
CUARTO CURSO DEL GRADO
DE ING. INFORMÁTICA EN COMPUTACION

Introducción a las Redes Neuronales Artificiales

César Hervás-Martínez
Pedro A. Gutiérrez Peña

Grupo de Investigación AYRNA

Departamento de Informática y Análisis Numérico
Universidad de Córdoba
Campus de Rabanales. Edificio Einstein.
Email: chervas@uco.es
pagutierrez@uco.es

2021-2022

Inspiración Biológica

Los animales son capaces de reaccionar de forma adaptativa a los cambios en su entorno externo e interno, y utilizan su sistema nervioso para realizar estas conductas.

Un modelo adecuado de simulación del sistema nervioso debe ser capaz de producir respuestas y comportamientos similares en los sistemas artificiales.

El sistema nervioso está construido por unidades relativamente simples, las neuronas, por lo que copiar su comportamiento y funcionalidad debe ser la solución.

NEUROINFORMATICA

- Moderna teoría acerca de nuevos principios y modelos matemáticos de procesamiento de la información, la cual está basada en prototipos biológicos y mecanismos basados en las actividades del cerebro humano.

APLICACIONES :

Imagen, Sonido y Reconocimiento de Patrones

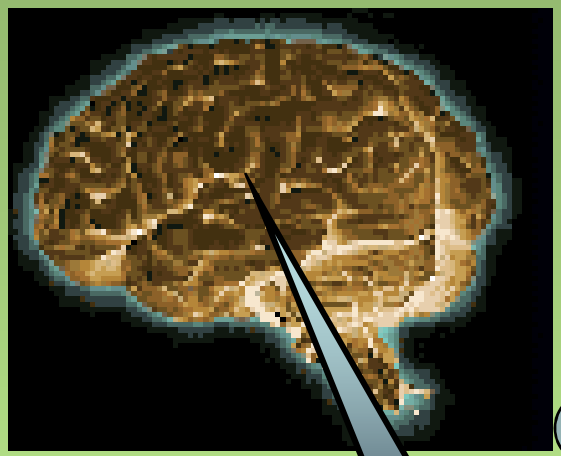
Toma de decisiones

Descubrir conocimiento

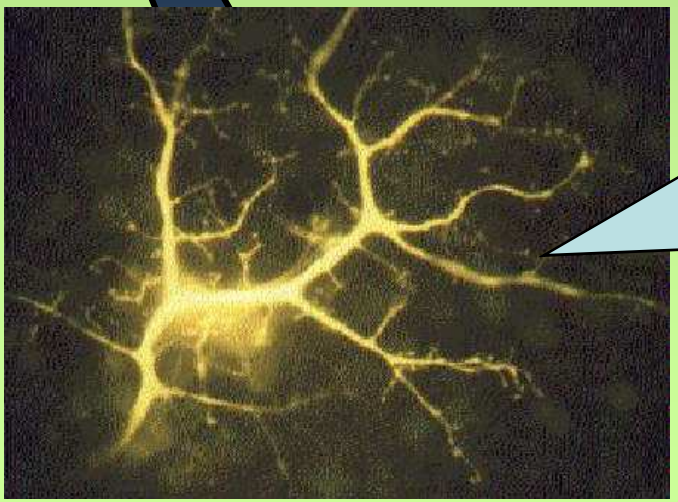
Análisis dependiente del contexto

……..

¿Que es la Computación Cerebral?



El cerebro humano contiene una red masivamente interconectada de 10^{10}-10^{11} (10 billones) neuronas (células corticales)



Neurona Biológica - Elemento más sencillo de "Computación Aritmética"

Reconocimiento de Imágenes: Regla de decisión y clasificador

- En la mayoría de los problemas de clasificación /reconocimiento, la formalización de la regla de decisión es muy complicada o imposible de formular.

- Una red neuronal es un modelo no lineal, el cual puede acumular conocimiento acerca de sus coeficientes y estructura a través de un proceso de aprendizaje.

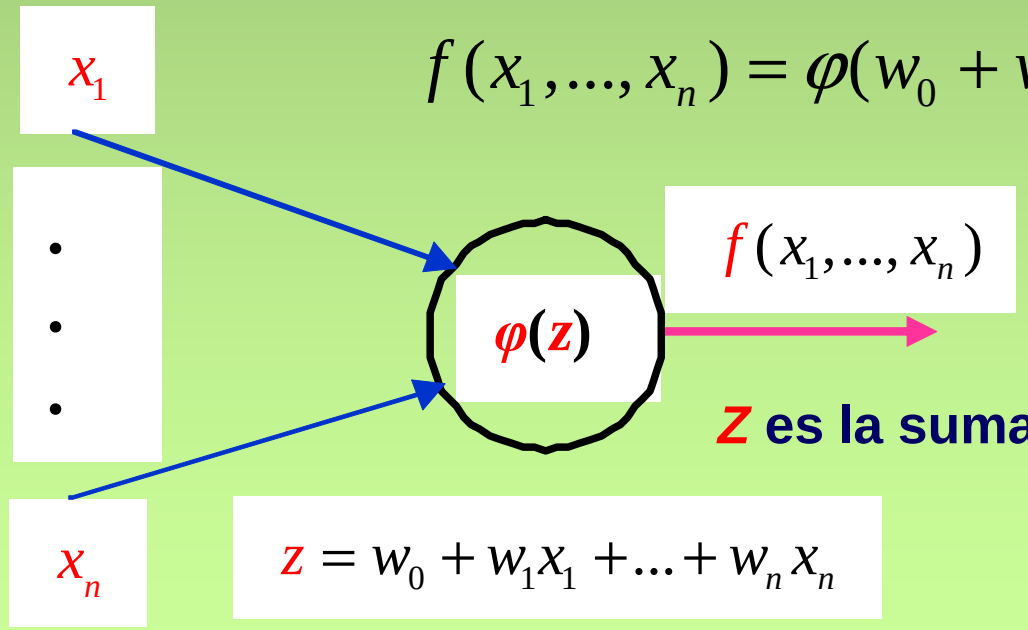
- Despues del proceso de aprendizaje, una red (modelo no lineal) es capaz de aproximar una función contínua, la cual se supone que será nuestra regla de decisión.

Neurona Artificial

f **es la función que tenemos que aprender**

$x_1, \ldots, x_n$ **son los valores de las variables de entrada**

φ **es la función de activación**



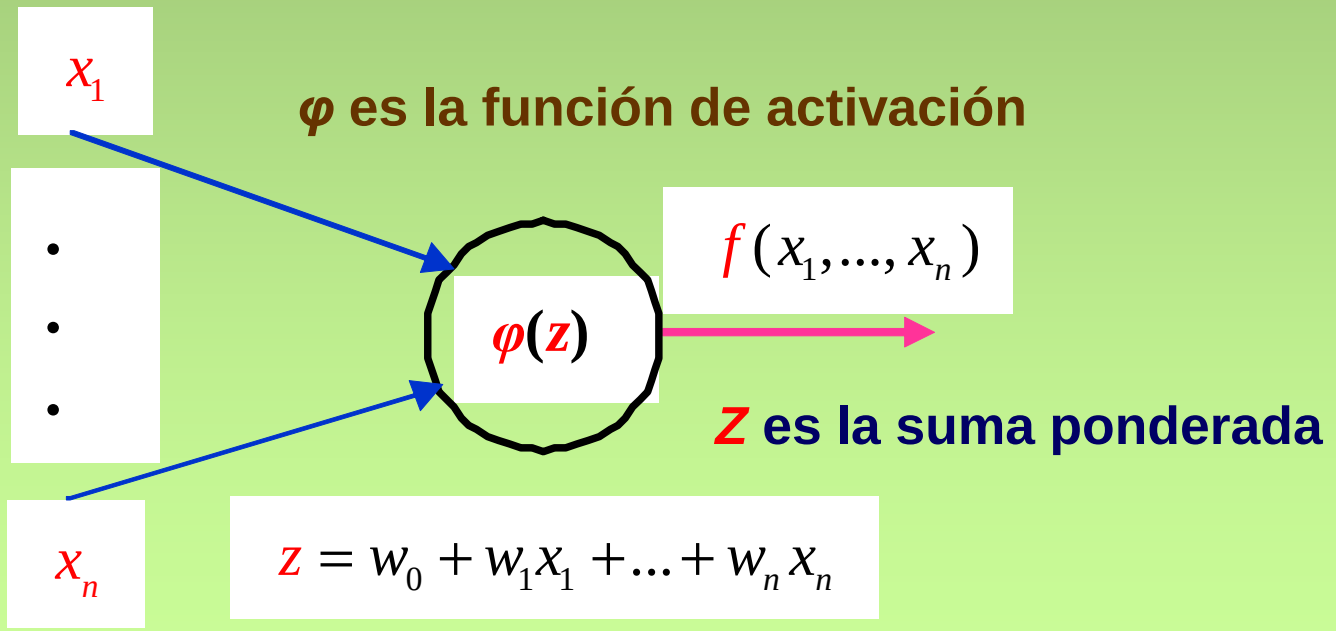
$$f(x_1, \ldots, x_n) = \varphi(w_0 + w_1 x_1 + \ldots + w_n x_n)$$

Z **es la suma ponderada**

$$z = w_0 + w_1 x_1 + \ldots + w_n x_n$$

Neurona Artificial

La funcionalidad de las neuronas se determina mediante la naturaleza de su función de activación, sus principales propiedades, su plasticidad y flexibilidad, su capacidad para aproximar una función que debe de aprenderse.



El modelo de McCullogh-Pitts :

- Los impulsos se interpretan como tasas de activación.

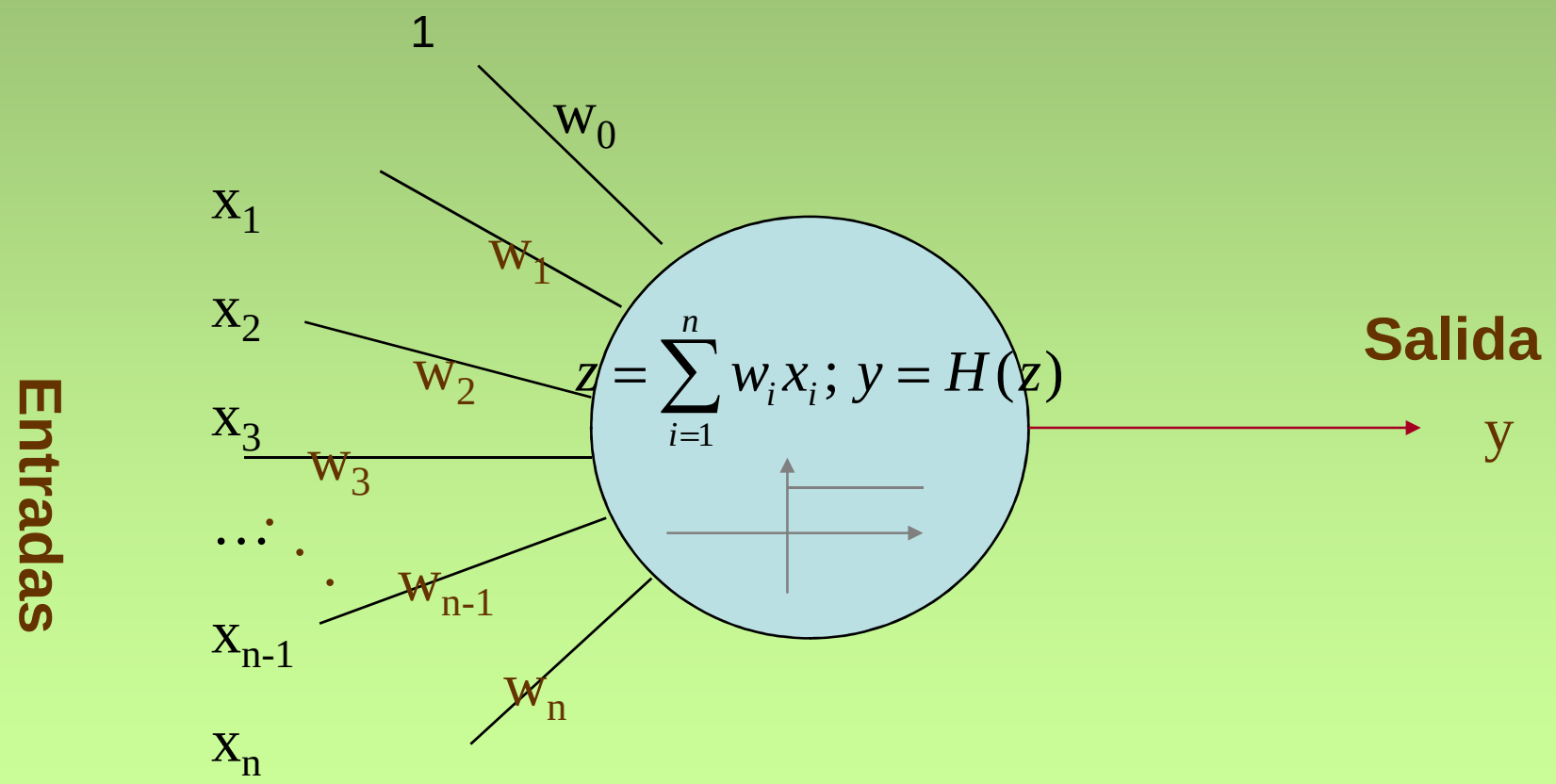
- La fuerza de la sinapsis se traslada a los pesos sinápticos o pesos de las conexiones de la red.

- **Excitación:** el producto positivo entre la tasa del impulso de entrada y el peso sináptico correspondiente.

- **Inhibición:** el producto negativo entre la tasa del impulso de entrada y el peso sináptico correspondiente.

El modelo de McCullogh-Pitts

Las neuronas realizan el trabajo de procesamiento de la información. Ellas reciben y proporcionan información en forma de impulsos.



Generalización no lineal de la neurona de McCullogh-Pitts:

$$y = f(\mathbf{x}, \mathbf{w})$$

y **es la salida de la neurona,** **x es el vector de entradas, y** **w es el vector de pesos sinápticos.**

Ejemplos:

$$y = \frac{1}{1 + e^{-\mathbf{w}^t \mathbf{x} - w_0}}$$

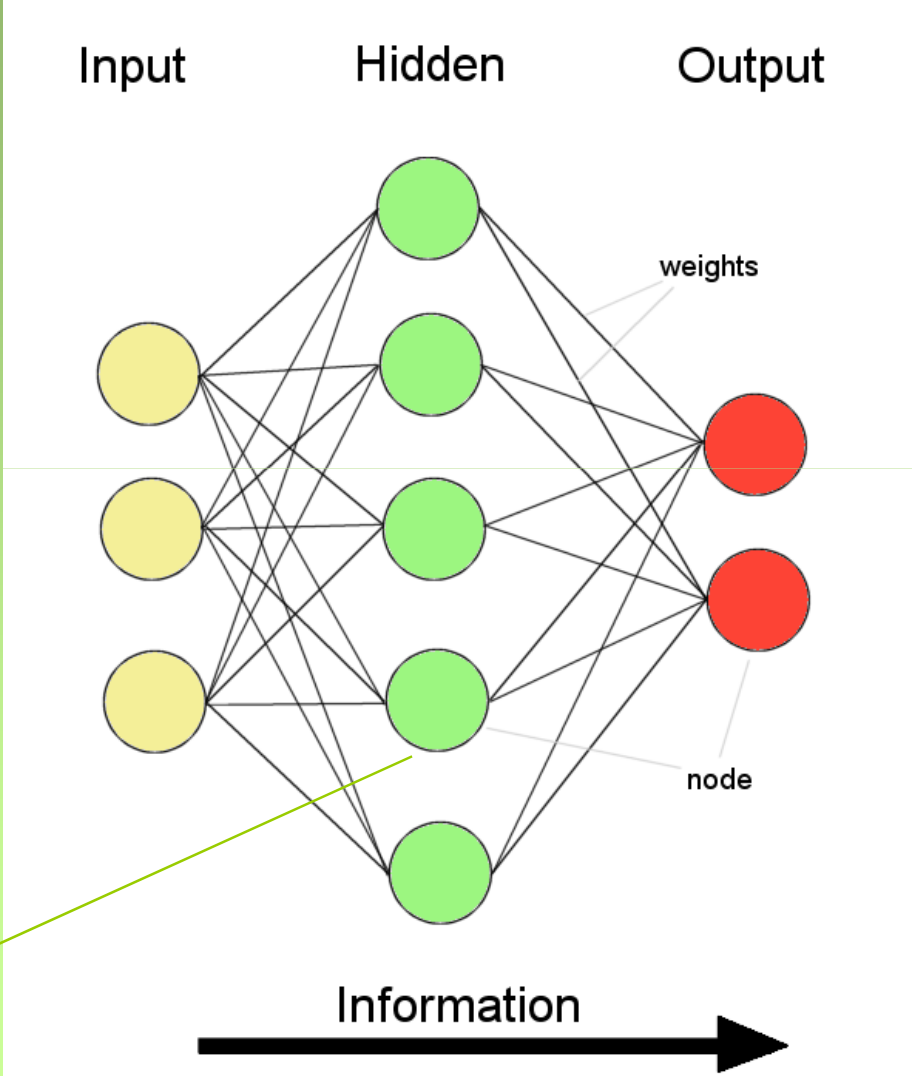
Neurona sigmoidal o sigmoide

$$y = e^{-\frac{\|\mathbf{x}-\mathbf{w}\|^2}{2a^2}}$$

Neurona Gaussiana

Redes Feed-forward (con flujo de información hacia adelante)

- El flujo de información es unidireccional
- Los datos se presentan a la Capa de Entrada
  - Pasan a la *Capa Oculta*
  - De esta pasan a la *Capa de Salida*
- Se distribuye la información
- La información se procesa de forma paralela

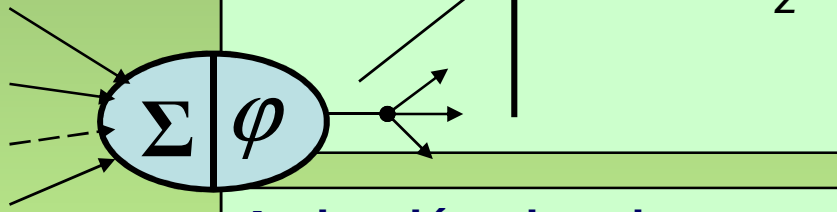


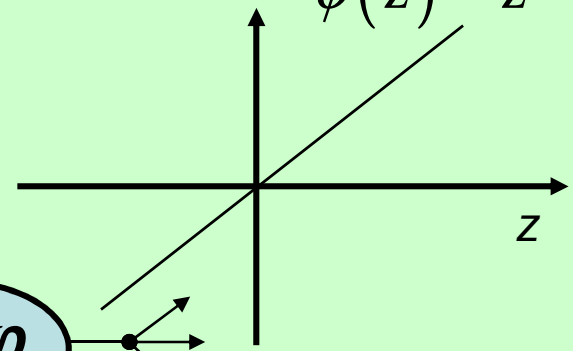
Representación interna de los datos (Interpretación)

Neurona Artificial : Funciones Clásicas de Activación

Activación Lineal

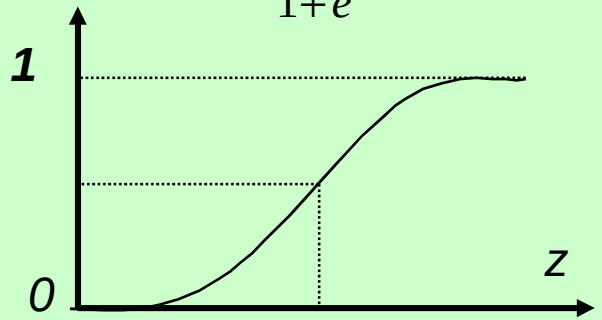
$$\phi(z) = z$$





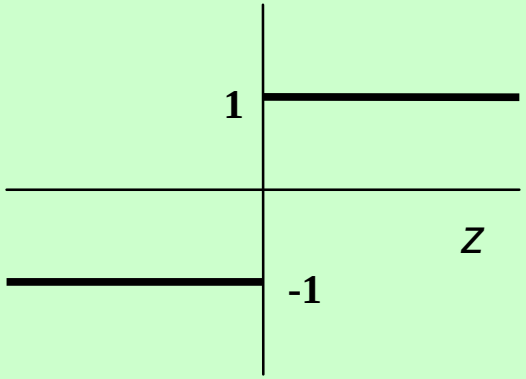
Activación Logistica o Sigmoidal

$$\phi(z) = \frac{1}{1+e^{-\alpha z}}$$



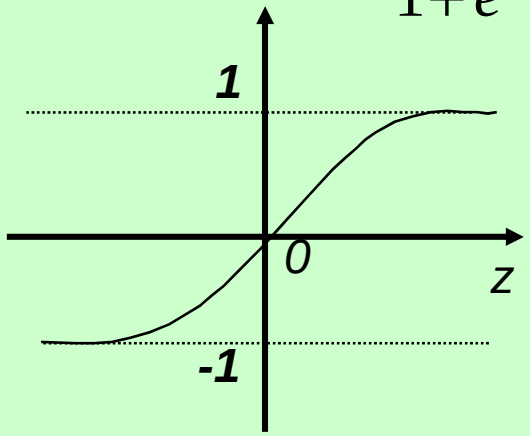
Activación de salto

$$\phi(z) = \text{sign}(z) = \begin{cases} 1, & si \quad z \geq 0, \\ -1, & si \quad z < 0. \end{cases}$$

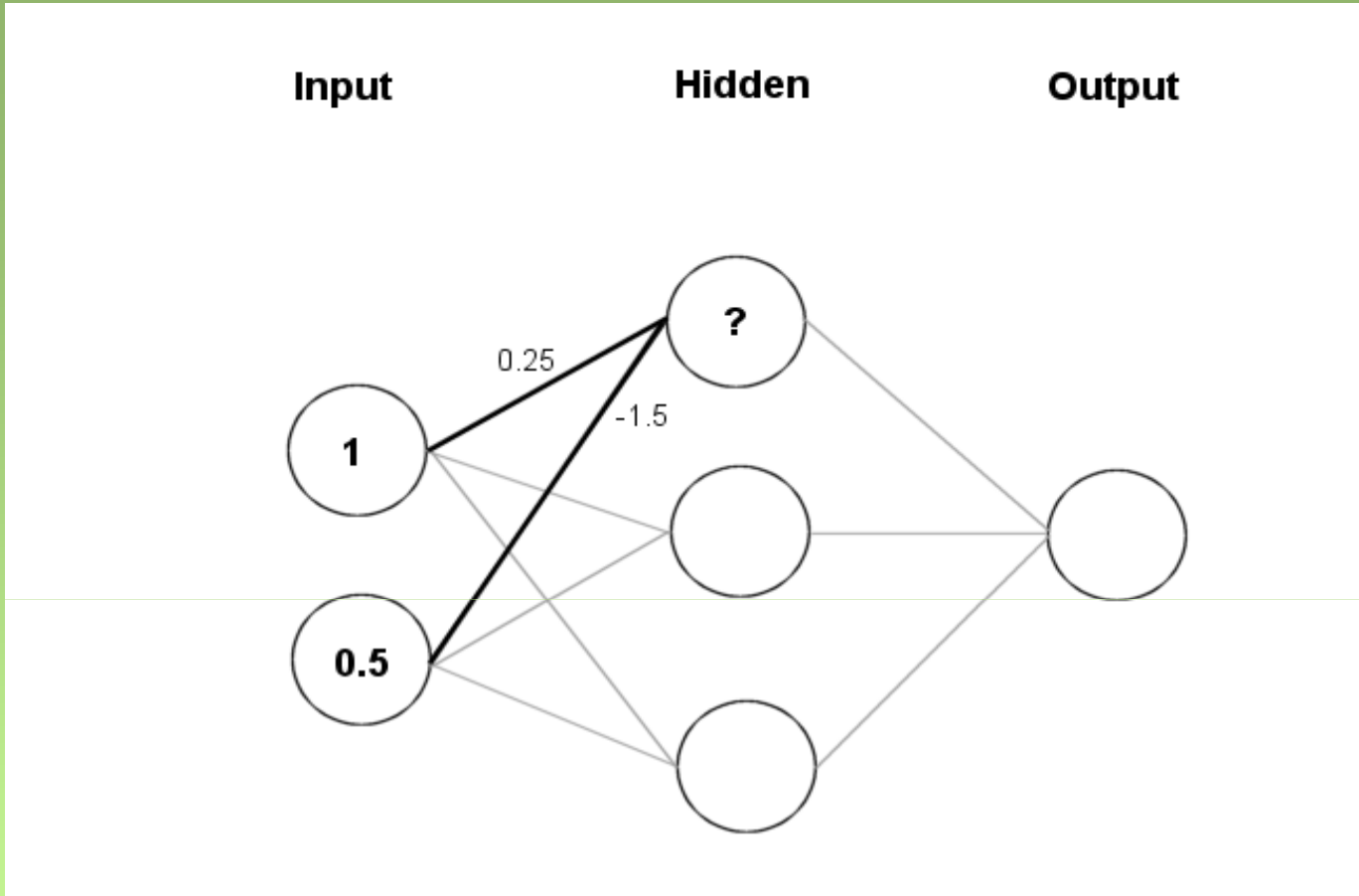


Activación Tangente Hiperbólica

$$\varphi(u) = tanh(\gamma u) = \frac{1-e^{-2\gamma u}}{1+e^{-2\gamma u}}$$



Flujo de datos a través de la red



$(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) = -0.5$

Alisado mediante una función sigmoide:

$$\frac{1}{1+e^{0.5}} = 0.3775$$

Neurona de Umbral (Perceptron)

- La salida de una neurona de umbral es binaria, mientras que las entradas pueden ser binarias o continuas.
- Si las entradas son binarias, una función de umbral implementa una función Booleana.
- El alfabeto Booleano {1, -1} es el que habitualmente se utiliza en teoría en vez del {0,1}. La correspondencia con el alfabeto clásico Booleano {0, 1} se establece en la forma:

$$0 \rightarrow 1; 1 \rightarrow -1;\ \ y \in \{0,1\}, x \in \{1,-1\} \Rightarrow$$

$$\Rightarrow x = 1 - 2y \quad o \quad x=(-1)^y$$

FUNCIONES BOOLEANAS DE UMBRAL

- **La función Booleana $f(x_1,\ldots,x_n)$ se llama una función de umbral *(linealmente separable)*, si es posible encontrar un vector de pesos, números reales, $W=(w_0,w_1,\ldots,w_n)$ tal que se verifique la ecuación**

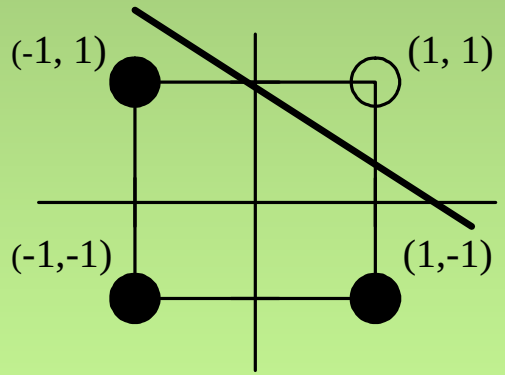
$$f(x_1,\ldots x_n)=sign(w_0+w_1x_1+\ldots+w_nx_n)$$

para todos los valores de las variables *x* del dominio de la función *f*.

- **Cualquier función de umbral se puede aprender mediante una sola neurona con la función umbral como función de activación.**

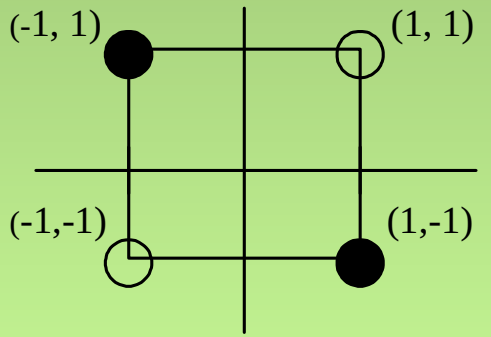
FUNCIONES BOOLEANAS DE UMBRAL: INTERPRETACIÓN GEOMÉTRICA

La función "OR" (Disyunción) es un ejemplo de función Booleana de umbral (linealmente separable): Los "-1" son separados de los "1" por una recta



- (1,1)→ 1
- (1,-1)→ -1
- (-1,1)→ -1
- (-1,-1)→ -1

La función XOR es un ejemplo de función Booleana no de umbral (no es linealmente separable) : Es imposible separar los "1" de los "-1" por cualquier recta



- (1,1) → 1
- (1,-1)→ -1
- (-1,1)→ -1
- (-1,-1)→ 1

NEURONA DE UMBRAL: APRENDIZAJE

- La principal propiedad de una neurona y de una red neuronal es su habilidad para aprender de su entorno, (datos), y para mejorar su rendimiento a través del aprendizaje.

- Una neurona (una red neuronal) aprende acerca de su entorno mediante un proceso iterativo de ajustes aplicados a sus pesos sinápticos.

- Idealmente, una red (una simple neurona) recibe mas conocimiento acerca del entorno, (datos), después de cada iteración del proceso de aprendizaje.

Neurona de Umbral: Aprendizaje

- **Sea un conjunto finito de vectores n-dimensionales que describen algunos objetos pertenecientes a algunas clases (suponemos por simplicidad, pero sin pérdida de generalidad, que hay sólo dos clases y que los vectores son binarios). Al conjunto le llamaremos conjunto de aprendizaje:**

$$\mathbf{X}^j = \left(x_1^j, ..., x_n^j \right); \mathbf{X}^j \in C_k, k = 1, 2;\ j = 1, ..., m;$$

$$x_i^j \in \{1, -1\}$$

NEURONA DE UMBRAL: APRENDIZAJE

- El aprendizaje de una neurona (o de una red) es el proceso de su adaptación a la identificación automática de la clase de pertenencia de todos los vectores del conjunto de entrenamiento, la cual se basa en el análisis de estos vectores: sus componentes forman el conjunto de neuronas de entrada de la red.

- Este proceso deberá realizarse a través de un algoritmo de aprendizaje.

Neurona de Umbral: Aprendizaje

- Sea T la salida deseada de una neurona de una red para un cierto vector de entrada y sea Y la salida actual de la neurona.

- Si T=Y, no hay nada que aprender

- Si T≠Y, entonces la neurona tiene que aprender, para asegurarse de que una vez que se han ajustado los pesos, su estado actual coincidirá con la salida deseada.

Aprendizaje de correción del error

- Si T≠Y, entonces $\delta = T - Y$ es el error.

- Una meta del aprendizaje es ajustar los pesos de las conexiones de la red de tal forma que para la nueva salida con los pesos cambiados tengamos lo siguiente:

$$\tilde{Y} = Y + \partial = T$$

- Esto es, la salida de la red una vez cambiados los pesos debe de coincidir con la salida deseada.

Aprendizaje de correción del error

- **La regla de aprendizaje de corrección del error se determina para que la salida actual, una vez cambiados los pesos, coincida con la salida deseada :**

$$W = (w_0, w_1, \ldots, w_n); \quad X = (x_1, \ldots, x_n)$$

$$\tilde{w}_0 = w_0 + \alpha\delta$$

$$\tilde{w}_i = w_i + \alpha\delta x_i; \quad \text{para } i = 1, \ldots, n$$

- **α es el coeficiente de aprendizaje (deberá ser igual a 1 para la neurona umbral, cuando la función a aprender sea Booleana)**

ALGORITMO DE APRENDIZAJE

- Un algoritmo de aprendizaje consta de un chequeo secuencial para todos los patrones de un conjunto de entrenamiento, para ver si su clase de pertenencia está reconocida correctamente.

  Si es así, no es necesaria ninguna acción.

  En otro caso, es necesario aplicar una regla de aprendizaje para ajustar los pesos de las conexiones de la red.

El proceso continúa iterativamente hasta que para todos los vectores del conjunto de entrenamiento su clase de pertenencia sea reconocida correctamente, o no pueda reconocerse para un aceptablemente pequeño número de vectores (patrones del conjunto de entrenamiento).

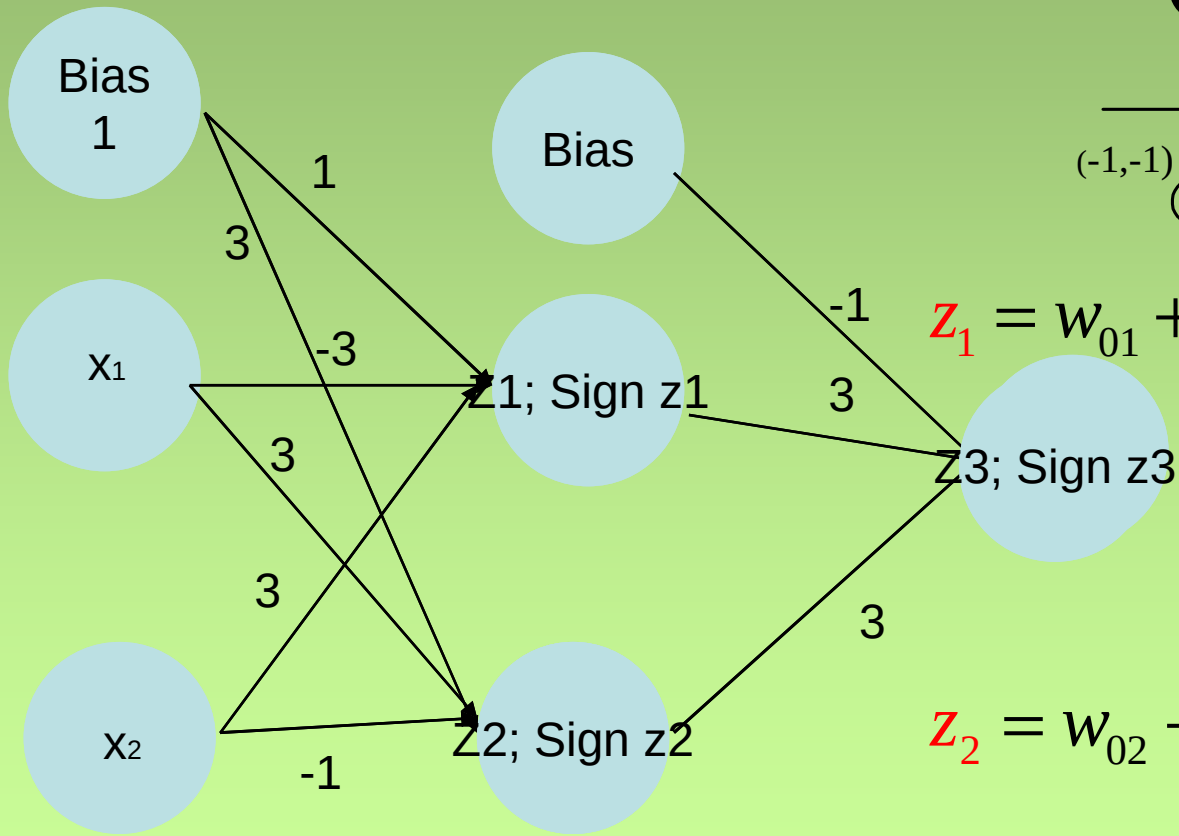
¿CUANDO NECESITAMOS UNA RED NEURONAL?

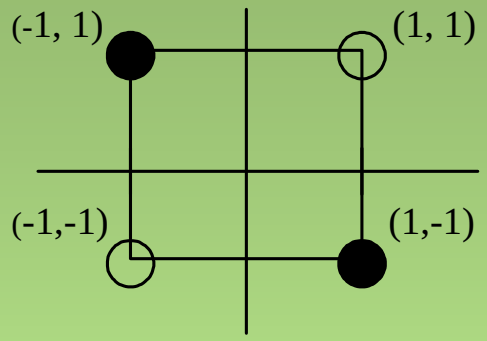
- **La funcionalidad de una sóla neurona es limitada. Por ejemplo, la neurona de umbral (el perceptron) no puede aprender funciones no linealmente separables, por ejemplo la función XOR.**

- **Para aprender estas funciones (transformaciones entre entradas y salidas) que no pueden ser aprendidas por una sóla neurona, debe de utilizarse una red neuronal.**

Resolviendo el problema XOR utilizando una red sencilla

$$x_1 \oplus x_2 = x_1\bar{x}_2 \vee \bar{x}_1 x_2 = f_1(x_1, x_2) \vee f_2(x_1, x_2)$$





$$z_1 = w_{01} + w_{11}x_1 + w_{21}x_2$$

$$z_2 = w_{02} + w_{12}x_1 + w_{22}x_2$$

Resolviendo el problema XOR utilizando una red más sencilla

#	Inputs		Neurona 1		Neurona 2		Neurona 3		XOR= $= x_1 \oplus x_2$
			$\tilde{W} = (1,-3,3)$		$\tilde{W} = (3,3,-1)$		$\tilde{W} = (-1,3,3)$		
	x_1	x_2	*Z1*	sign(z1) output	*Z2*	sign(z2) output	*Z3*	sign(z3) output	
1)	1	1	1	1	5	1	5	1	1
2)	1	-1	-5	-1	7	1	-1	-1	-1
3)	-1	1	7	1	-1	-1	-1	-1	-1
4)	-1	-1	1	1	1	1	5	1	1

$z_1 = w_{01} + w_{11}x_1 + w_{21}x_2 = 1 + (-3)\times 1 + 3\times 1,$ neurona 1

$z_2 = w_{02} + w_{12}x_1 + w_{22}x_2 = 3 + 3\times 1 + (-1)\times 1,$ neurona 2

$z_3 = \beta_0 + \beta_1 \times signz_1 + \beta_2 \times signz_2 = -1 + 3\times 1 + 3\times 1,$ neurona 3

Propiedades: Mapeo Universal

- **Pregunta:**
  - **¿Qué tipo de funciones puedo representar con una ANN?**
- **La idea se remonta al problema #13 de Hilbert (1900).**
  - **Representar una función de N variables como combinación lineal de funciones en una variable (bajar la dimensionalidad del problema)**
- **Respuesta:**
  - **Puedo representar el conjunto de funciones "suaves".**
  - **Hay varias demostraciones para diferentes arquitecturas**
  - **Kolgomorov (1957), Cybenko (1960)**
  - **Hornik (1989), Chen (1991)**

Propiedades: Reconocedor Universal

- **Teorema: (Chen '91)**
  - **F = { f: R→R, $f(x)=\Sigma_{i=1}^{N} c_i\ \sigma(x.w_i + d_i)$, σ sigmoide } es un conjunto denso en $C(R^{ext})$**
  - **Sigmoide generalizada: acotada, no necesariamente continua o monótona**
  - **Prueba constructiva, extensible a R^n.**
- **Demostración:**

$$\text{Dado } \begin{matrix}\varepsilon>0\\ S=\sup(|\sigma(x)|)\end{matrix}, \exists M,N>0 \begin{cases} x>M \rightarrow |f(x)-A|<\frac{\varepsilon}{4} \\ x<-M \rightarrow |f(x)-B|<\frac{\varepsilon}{4} \\ |x'-x''|\le\frac{1}{N} \rightarrow |f(x')-f(x'')|<\min(\frac{\varepsilon}{4},\frac{\varepsilon}{4S}) \end{cases}$$

$$-M=x_0<x_1<x_2<\cdots<x_{2MN-1}<x_{2MN}=M,\ t_i=\frac{1}{2}(x_i+x_{i+1})$$

$$\exists W>0, u>W \rightarrow |\sigma(u)-1|<\frac{1}{MN}, u<-W \rightarrow |\sigma(u)|<\frac{1}{MN}$$

$$\text{Sea } K>0, K>2NW$$

$$g(x)=f(-M)+\sum_{i=1}^{2MN}\big(f(x_i)-f(x_{i-1})\big).\sigma(K(x-t_{i-1}))$$

Propiedades: Reconocedor Universal

Luego se desarrollaron pruebas solamente con una capa oculta que son las más usadas en la actualidad

- **Otras usan el teorema de Stone-Weiertrass: (ej. Hornik'89)**
  - **Sea D un conjunto compacto en R^n, y sea F el conjunto de funciones reales sobre D que satisfacen:**
    - **Identidad: $\{f: f(x)=1\ \forall x \in D\} \in F$**
    - **Separabilidad: $\forall\ x_1 \neq x_2$ en D → $\exists\ f \in F \mid f(x_1) \neq f(x_2)$**
    - **Clausura: $f,g \in F \rightarrow f.g \in F$; $af+bg \in F\ \forall\ a,b \in R$**
  - **Teorema) F es denso en C(D) el conjunto de funciones reales continuas sobre D.**

$$Dado\ \varepsilon > 0 \quad y \quad g \in C(D),\ \exists\ f \in F\ /\ \left| f(x) - g(x) \right| < \varepsilon,\ \forall x \in D$$

Autor	Activacion	Dominio	Prueba
Cybenko	Cont, sigm	C(K)	Existencial
Hornik	Monot, sigm	C(K)	Constructiva
Ito	Monot, sigm	$C_o(R^n)$	Existencial
Hornik	Cont, no-cte	$C(R^n)$	Existencial
Stinchcombe	$L^p_{loc} \cap L_1(R)$	$L^p(K)$	Constructiva
Cybenko	Acotada sigm	$L^p(K)$	Existencial
Chen	Acotada sigm	$C(R^n)$	Constructiva
Chen	$L^p_{loc} \cap$ sigm	$L^p(K)$	Constructiva
Chen	No-cte en $C_0(R^n)$	$C(R^n)$	Existencial

SIMPLIFICACIONES PRÁCTICAS

- REDES FEEDFORWARD (flujo de información hacia adelante)
  - Eliminan la dinámica interna
    - puede existir dinámica a partir de interconexiones con otros sistemas
  - Utilizan una transformación estática de entrada-salida
  - Su uso se extiende a aplicaciones y modelos neuro-fuzzy
    - reconocimiento de patrones
    - data *mining* (minería de datos)
    - neuro-control
  - Tienen un mayor desarrollo teórico y experiencia práctica
    - mapeo universal
    - aprendizaje
      - algoritmos, convergencia, complejidad
    - regularización

APRENDIZAJE

- Dada la estructura de red, es el procedimiento de modificación de sus parámetros para lograr el comportamiento deseado de la misma
  - La estructura de la red también puede ser “aprendida”
- Métodos de aprendizaje:
  - Supervisado
    - existe un “tutor” que me dice lo que la red debe hacer
    - ej: reproducir transformación entrada-salida dado un (patrón)
  - No supervisado
    - generar clasificación propia del conjunto de patrones
    - ej: clustering (agrupamiento)
  - Aprendizaje por reforzamiento
    - aprendizaje basado en una señal de evaluación: bien/mal (recompensa)
    - ej: ajedrez, backgammon (señal de refuerzo: gané/perdí)

APRENDIZAJE: ON-LINE VS OFF-LINE

- Diferentes formas de adaptar la red de acuerdo con la forma en que se use la información de entrenamiento (patrones).
- Aprendizaje Off-line (batch)
  - adaptación de parámetros de la red tomando el conjunto total de patrones de entrenamiento
    - usualmente llamado época (*epoch)*
    - problemático con muchos datos
- Aprendizaje On-line (recursivo)
  - adaptación de parámetros hecha a medida que los patrones son presentados o usados por la red
    - problema de “olvido” de patrones “viejos”
- Métodos intermedios
  - adaptación cada *k* patrones (batch size)

APRENDIZAJE: OPTIMIZACIÓN

- La capacidad de aprender proviene en general de la elección de los parámetros de la red.
  - Comportamiento deseado de la red → error asociado
  - Adaptación de parámetros → optimización de una función de error
- Métodos de optimización:
  - Error lineal en parámetros
    - mínimos cuadrados y derivados
    - batch o recursivo
  - Error no-lineal en los parámetros
    - con/sin uso de derivadas
    - batch o recursivo
    - Más complejos que los lineales
  - Combinación de ambos según arquitectura de ANN
- Aprendizaje de la estructura de ANN
  - En general se realiza en un entorno superior del algoritmo (batch).

Entrenando la Red - Aprendizaje

Algoritmo de Retropropagación (Backpropagation)

- **Necesita un conjunto de entrenamiento (pares entrada / salida)**
- **Inicializar con pequeños pesos aleatorios los valores de las conexiones de la red**
- **El error se utiliza para ajustar los pesos (aprendizaje supervisado)**

→ **Método de gradiente descendente sobre la superficie de error**

PERCEPTRON MULTICAPA: BACKPROPAGATION

- Procedimiento para encontrar el vector gradiente de una función de error asociada a la salida de la red con respecto a los parámetros de la misma.
- El nombre *backpropagation* (retropropagación) viene de que el cálculo se hace en el sentido inverso de la red, propagándose desde los nodos de salida hacia los nodos de entrada.
- Werbos (1972) desarrolló la idea general de derivadas ordenadas siendo Rumelhart quien lo aplicó a las ANN en los años 80.
- Esto permite poder aplicar a posteriori alguno de los muchos métodos de optimización con gradiente para obtener el comportamiento deseado de la red.

PERCEPTRON MULTICAPA: BACKPROPAGATION

- Ventajas
  - Funciona aceptablemente bien
  - Relativamente rápido si acertamos en la inicialización de los pesos de la red
- Inconvenientes
  - Necesita que las funciones de error sean derivables
  - Puede ser lento
- Alternativas a la Retropropagación del error
- Aprendizaje Hebbiano
    - No tuvo éxito en las redes feedforward
  - Aprendizaje con reforzamiento
    - Sólo con éxito limitado
  - Algoritmos Evolutivos y Enjambres de Partículas
    - Mas general, pero es mucho más lento que la retropropagación

ALGORITMO DE RETROPROPAGACIÓN

- **Error Cuadrático**
  - **Como ya se menciono antes, el proceso de aprendizaje se realiza en orden a reducir el error, pero ¿como podemos describir ese error? Por lo general en regresión se utiliza el error cuadrático, E.**
  - **Observemos que esta función suma todos los errores sobre todas las unidades de salida después de que se hayan computado todos los patrones del conjunto de entrenamiento.**

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

ALGORITMO DE RETROPROPAGACIÓN

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

Para cada valor k se puede adaptar el peso mediante:

$$w_k = w_k + \Delta w_k, \quad \Delta w_k = -\eta \frac{\partial E}{\partial w_k}$$

Donde η es el coeficiente de aprendizaje, con valores entre 0 y 1. Si η vale 0 la red no aprende, y si vale 1 los cambios de pesos son muy abruptos y puede sobreaprender el algoritmo.

Pero en la práctica dado que la función E suma todos los errores sobre todos los patrones del conjunto de entrenamiento, el algoritmo necesita un mayor tiempo de cómputo para evaluar esta función, además de poder quedar fácilmente atrapado en un mínimo local.

ALGORITMO DE RETROPROPAGACIÓN

- **Así, construiremos una nueva función a la que denominaremos error cuadrático estocástico, Ed:**

$$E_d(w_k) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- **Como puede verse la función calcula sólo el error de un ejemplo o patrón. El gradiente de Ed se calcula mediante:**

$$\nabla E_d(w_k) = \frac{\partial E_d}{\partial w_k}(t_k - o_k) o_k (1 - o_k)$$

EJEMPLO: PERCEPTRON MULTICAPA

- Veamos como aplicamos el algoritmo de retropropagación del error con más detalle
  - Derivadas de las funciones de activación
  - Calculo del gradiente por capas
  - Algoritmo del gradiente descendente
- A tener en cuenta
  - Complejidad polinomial en #W (número de pesos de la red)
  - Arquitectura de la red
    - Tamaño (número de nodos y capas) y funciones de activación y de transferencia a usar
  - Mínimos locales
    - Inicialización de pesos
    - Aleatorizar el orden de presentación de los patrones
    - Introducir ruido en los patrones de entrenamiento
  - Problemas de generalización
    - Sobre-entrenamiento. Tamaño de red

Variantes de la Retro-propagation

- Variantes en aprendizaje
  - Momento
  - Adaptación de η
  - Otras funciones de costo
    - e.g.: entropía relativa
  - Inicialización de pesos $\sim(\text{fan-in})^{-1/2}$
  - Escalado de datos
    - preprocesamiento entrada-salida.
    - media nula y similar varianza
  - Generación de datos
    - con ruido
    - propiedades de estructura de datos

Validación, sobre-ajuste

Parada-temprana

Métodos mas avanzados de optimización de error

Arquitectura

Otras funciones de activación

continuas y derivables

mejor saturacion

Tamaño

importante para propiedades

Experimentación

minimización de riesgo, uso de validación

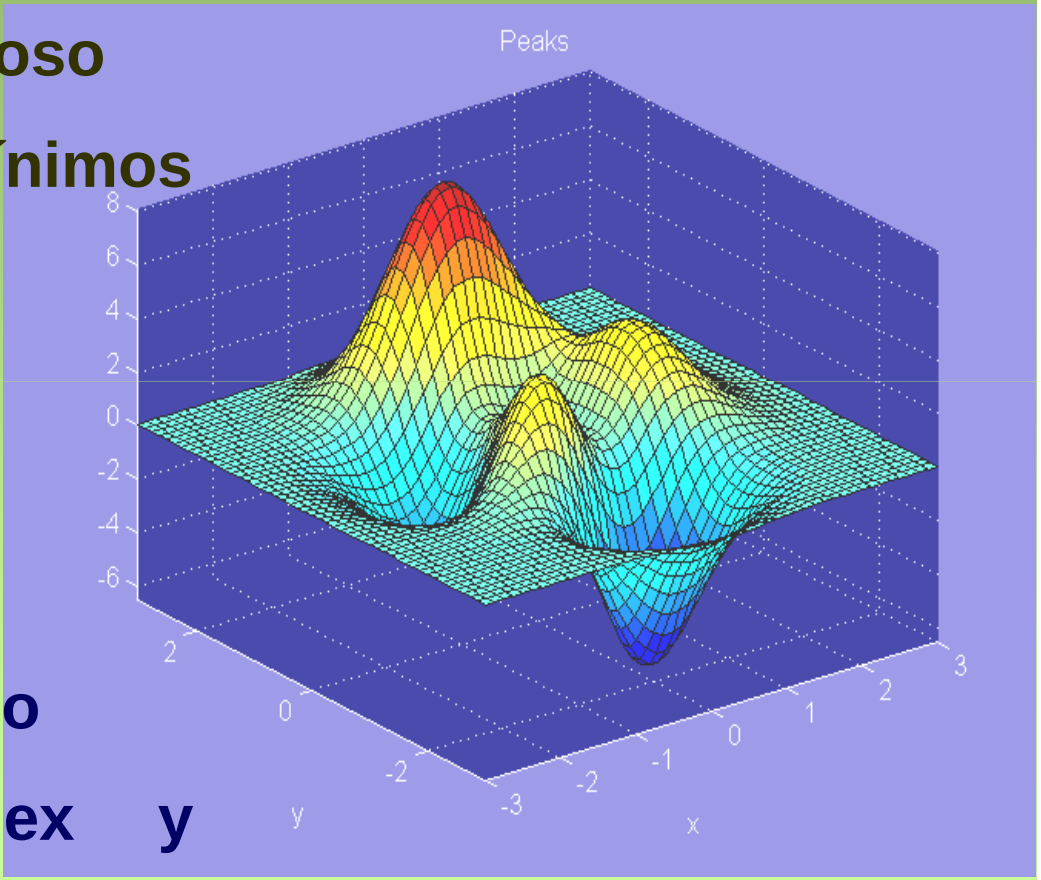
Optimización no lineal: Usando gradiente

- **Método de Levenberg-Marquardt**
  - Estimación del Hessiano usando una aproximación lineal. ($H=J^tJ$)
  - Específico para una función de error cuadrática
  - La dirección de optimización oscila entre los métodos de Gauss-Newton y el de máxima pendiente.
  - λ también controla el paso de actualización

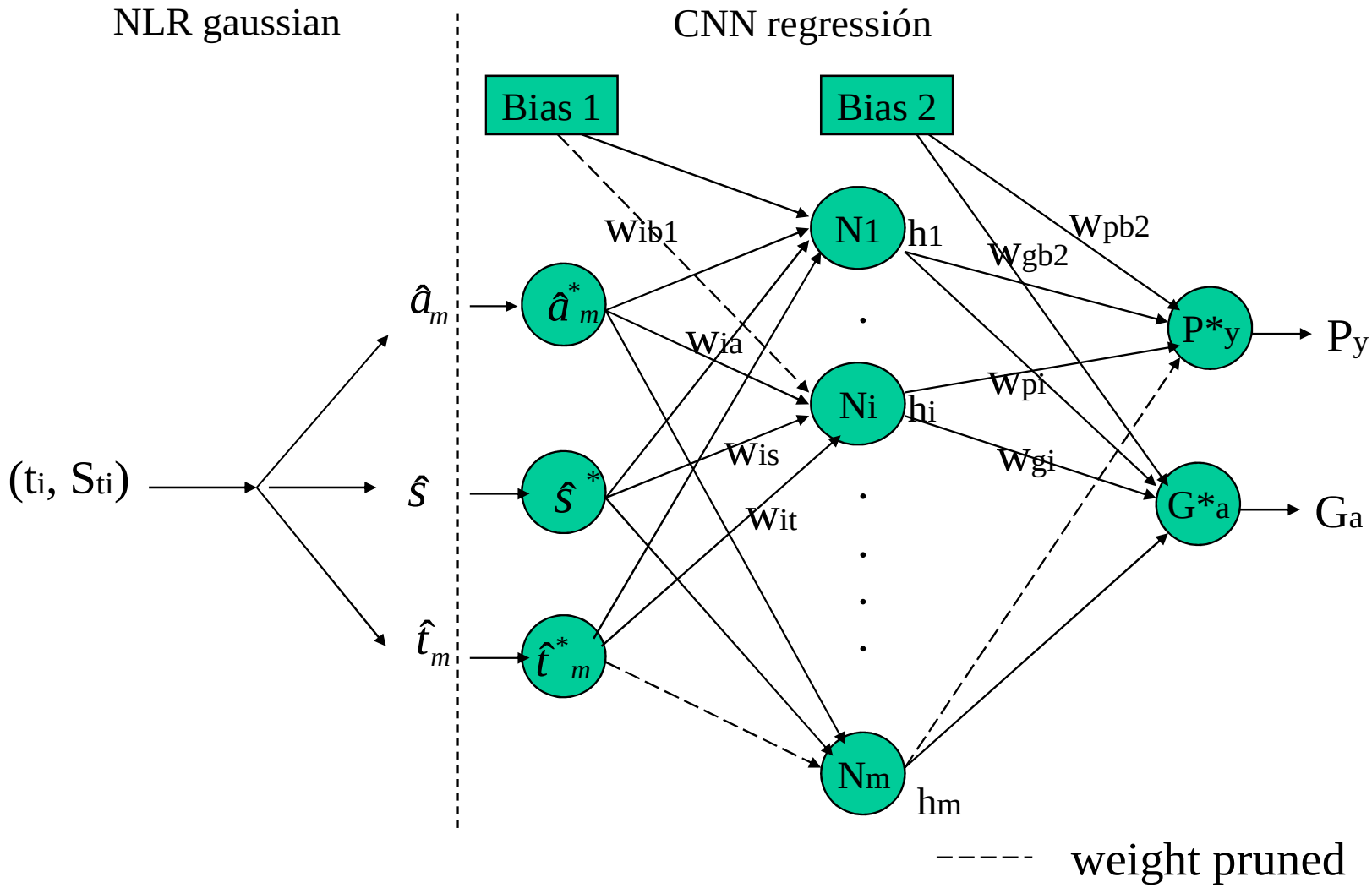
    si es muy grande dejaría de valer la aproximación lineal
  - Aproximación recursiva del Hessiano H y H^{-1}.

Optimización: Sin usar gradiente

- A veces no se puede utilizar el gradiente o es muy costoso
- Para tratar de evitar mínimos locales
- Más usados:
  - Algoritmos genéticos
  - Búsqueda aleatoria
  - Enfriamiento Simulado
  - Método del Simplex y variaciones



Ejemplo de red neuronal para Pirogallol y Gálico



Procedimiento de estimación de los parámetros del modelo

Ejemplo ecuaciones de red neuronal para P y GA

Tabla . Ecuaciones obtenidas a partir de modelos de Redes Neuronales podadas para la resolución de mezclas químicas de Pyrogallol y Acido Gálico

Modelo 1 (NT: 3:4:2; C: 19; DS: 44/22)

$$[P]^* = 1.25\,h_1 - 0.35\,h_2 + 0.09\,h_4$$

$$[GA]^* = -0.19 - 0.25h_1 + 1.74h_2 - 0.12h_3 - 0.34h_4$$

$$h_1 = \frac{1}{1+exp(2.45 - 2.18\,\hat{a}_m^* - 1.82\,\hat{s}^* - 0.93\,\hat{t}_m^*)}$$

$$h_2 = \frac{1}{1+exp(0.59 - 3.6\,\hat{a}_m^* + 2.07\,\hat{s}^*)}$$

$$h_3 = \frac{1}{1+exp(0.79)}$$

$$h_4 = \frac{1}{1+exp(3.56\,\hat{a}_m^* - 9.35\,\hat{s}^* - 4.41\,\hat{t}_m^*)}$$

Modelo 2 (NT: 3:3:2; C: 18; DS: 54:27)

$$[P]^* = 1.54h_1 + 0.37h_2 - 0.78h_3$$

$$[GA]^* = -0.69 - 1.74h_1 + 1.89h_2 + 1.22h_3$$

$$h_1 = \frac{1}{1+exp(3.69 - 0.07\,\hat{a}_m^* - 3.54\,\hat{s}^*)}$$

$$h_2 = \frac{1}{1+exp(0.74 - 3.71\,\hat{a}_m^* + 1.5\,\hat{s}^* - 0.38\,\hat{t}_m^*)}$$

$$h_3 = \frac{1}{1+exp(0.8 - 1.93\,\hat{a}_m^* + 7.05\,\hat{s}^* + 2.3\,\hat{t}_m^*)}$$

NT: network topology; C: connections; DS: data set (train/test); hi: output hidden node

Referencias

- C. Bishop, “Neural Networks for pattern recognition”, Oxford Press, 1995.
- Simon Haykin, “Neural Networks”, 2nd edition, Prentice Hall, 1999.
- Hertz, Krogh and Palmer, “Introduction to the theory of Neural Computation”, Addison-Wesley, 1991.
- Jang et al. “Neuro-fuzy and Soft Computing”, Cap. 8-11, Prentice Hall, 1997.
- C-T Lin y G. Lee, “Neural Fuzzy Systems”, Prentice Hall, 1995.
- Fernando Berzal. Redes Neuronales & Deep Learning. Noviembre 2018. Universidad de Granada

Introducción a las Redes Neuronales Artificiales

César Hervás-Martínez
Pedro A. Gutiérrez Peña

Grupo de Investigación AYRNA

Departamento de Informática y Análisis Numérico
Universidad de Córdoba
Campus de Rabanales. Edificio Einstein.
Email: chervas@uco.es
pagutierrez@uco.es

2021-2022