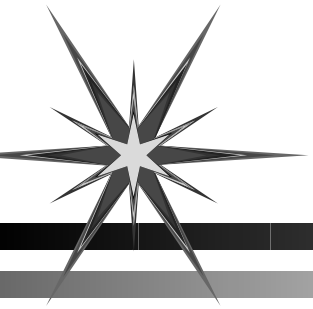


Introducción a los Modelos de Computación Conexionista

Patricio García Báez

Versión 1.1

Tabla de Contenidos

Tema 1. Introducción 3

- Tema 2. Características de las RNA . . 15
- Tema 3. Modelos 27

  Perceptrones simples 27

  - Backpropagation 40
  - Sistemas Autoorganizados 48
  - ART1 61
  - Red de Hopfield 67
  - Redes para el tratamiento temporal 74
  - Redes de Funciones de Base Radial 79
  - Redes Neuronales Modulares 87
- Tema 4. Neuro-Software 105
- Tema 5. Neuro-Hardware 114

Tema 1. Introducción

Introducción

“Entender el cerebro y emular su potencia”

- Definiciones:
  - Red Neuronal

    Una red neuronal (RN) es un procesador distribuido masivamente paralelo que tiene una propensión natural para almacenar el conocimiento a partir de su experiencia y hacerlo disponible para su uso. Se parece al cerebro en dos sentidos:

    El conocimiento es adquirido por la red mediante un proceso de aprendizaje.

    La fuerza de las conexiones entre neuronas conocidas como pesos sinápticos se utilizan para almacenar el conocimiento.

    (Alejsabder y Morton, 1990)

Introducción

Una red neuronal artificial (RNA) es un sistema de procesamiento de información que tiene ciertas aptitudes en común con las redes neuronales biológicas:

- El procesamiento de información ocurre en muchos elementos simples llamados neuronas.
- Las señales son transferidas entre neuronas a través de enlaces de conexión.
- Cada conexión tiene un peso asociado, el cual, típicamente, multiplica a la señal transmitida.
- Cada neurona aplica una función de activación (usualmente no lineal) a su entrada de red (suma de entradas pesadas) para determinar su salida.
- (Laurene Fausett)

Introducción

Una RN es una estructura de proceso de información distribuida y paralela, que consta de elementos de proceso (los cuales pueden poseer memoria local y pueden llevar a acabo operaciones localizadas de procesamiento de información) interconectados mediante canales de señales unidireccionales denominados conexiones. Cada elemento de proceso tiene una única conexión de salida que se ramifica (*fan out*) en tantas conexiones como se deseen, cada una llevando la misma señal, la señal de salida del elemento de proceso. Dicha salida puede ser de cualquier tipo matemático deseado. El procesamiento de información que acompaña a cada elemento de proceso puede ser definido arbitrariamente con las restricciones de que debe ser completamente local, esto es, solo puede depender de los valores actuales de las señales de entrada y de los valores almacenados en la memoria local de dicho elemento de proceso.

Introducción

- Algoritmo de aprendizaje

  Función que modifica los pesos sinápticos de un modo ordenado de manera que alcanza el objetivo deseado.

- Sinónimos de RN

  neurocomputador, neurocomputación, red conexionista, procesos distribuidos en paralelo (pdp's), redes neuronales (artificiales).

- Caracterización de RN
  - Arquitectura o topología:

    Patrón de conexionado.

  - Algoritmo de aprendizaje:

    Método de definición de pesos de conexión.

  - Función de activación

Inspiración Biológica

- Cerebro
  - Complejo, no lineal, paralelo.
  - Neurona:
    - Número: 10^{11}
    - Conexiones (sinapsis): 10^{15}
    - Tiempo eventos: milisegundos (10^{-3}s), en silicio es de nanosegundos (10^{-9}s)
    - Tamaño: 5 ó 6 órdenes de magnitud menor que puertas lógicas en silicio
  - Plasticidad Neuronal
    - Creación de nuevas neuronas
    - Creación de nuevas sinapsis
    - Modificación de sinapsis
  - Comparación con ordenadores:
    - Muerte neuronal
    - Optimizados para distintos problemas
      - Reconocimiento de patrones
      - Percepción
      - Control
    - Secuencialidad vs. Paralelismo

Inspiración Biológica

- Organización estructural en niveles
  - Sistema Nervioso Central
  - Circuitos entre regiones
  - Circuitos locales
  - Neuronas
  - Árboles dendríticos
  - Microcircuitos neuronales
  - Sinapsis
  - Canales iónicos
  - Moléculas

Historia

- Comienzos
  - 1943. Warren McCulloch y Walter Pitts. Definición de la neurona formal.
  - Norbert Wiener
  - John von Neumann
  - 1949. Donald Hebb. “The Organization of Behavior”. Regla de Hebb,
  - 1951. Marvin Minsky. “The Snark”
  - 1956. von Neumann. Concepto de redundancia.
  - 1959. Gabor. Filtros adaptativos no lineales.
- Primer éxito
  - 1957-58. Frank Rosenblatt. “El Perceptrón”.
  - 1960. Bernard Widrow. “Adaline”.
  - mediados 60’s: Abandono del campo
    - Falta de rigor
    - Falta de ideas
  - 1970. Marvin Minsky y Seymour Papert. “Perceptrons”
    - Separablidad lineal. Problema x-or.

Historia

- Periodo Obscuro (67-82)

  Fukushima, Grossberg, Kohonen, Anderson siguen trabajando a pesar de todo.

  1970. Malsburg. Mapas autoorganizados usando aprendizaje competitivo.

  1980. Grosberg. Base del ART (Adaptive Resonance Teory).

- Resurgimiento

  1982. Hopfield.

  1983. Fondos DARPA.

  1983. Kirkpatick, Galatt, Vecchi. “Simulated Annealing”, base de las máquinas de Boltzmann.

  1986. David Rumelhart y James McClelland. “PDP books”.

  Backpropagation:

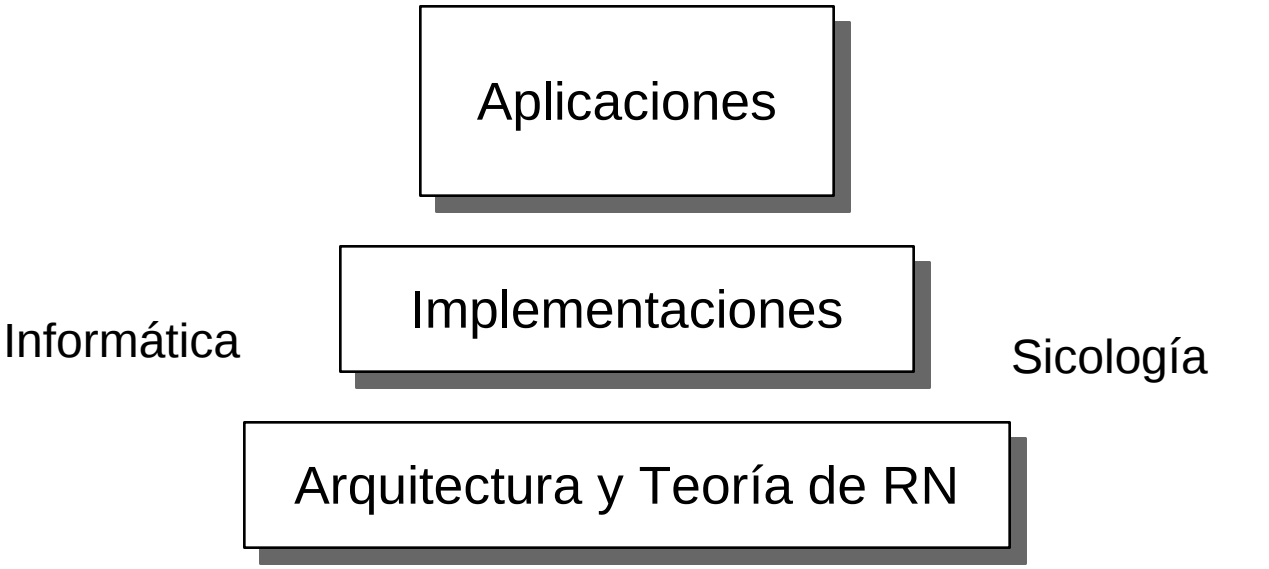
  Werbos, 1974; Parker, 1985; Lecun, 1985; Rumelhart, Hinton, Williams, 1986.

  1987. IEEE International Conference on Neural Networks (1700 participantes). Fundación de la INNS.

  1988-90. Revistas. Journal of NN. Neural Computation. IEEE Transaction on NN.

Áreas y Aplicaciones

Procesamiento de Señales
Control
Análisis de Datos
Reconocimiento de Patrones
Inteligencia Artificial



Propiedades y Capacidades

- Generalización
- Estructura altamente paralela
- No linealidad
- Mapeo de Entrada-Salida
- Adaptabilidad
- Respuesta graduada
- Información Contextual
- Tolerancia a fallos
- Implementación VLSI
- Uniformidad en el Análisis y Diseño
- Analogía Neurobiológica
- Infrmación Contextual

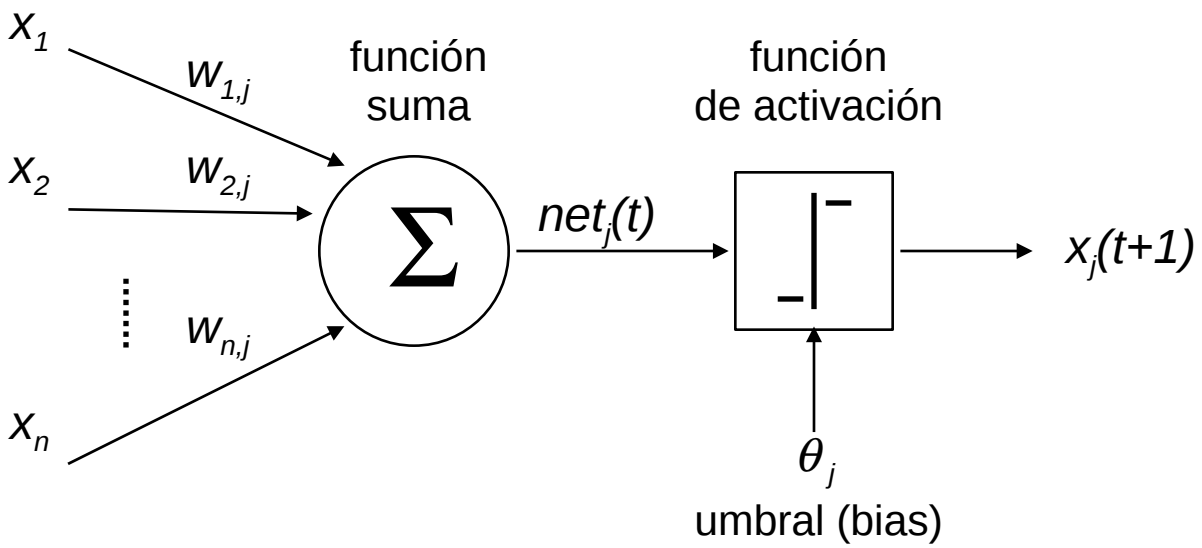
Bibliografía

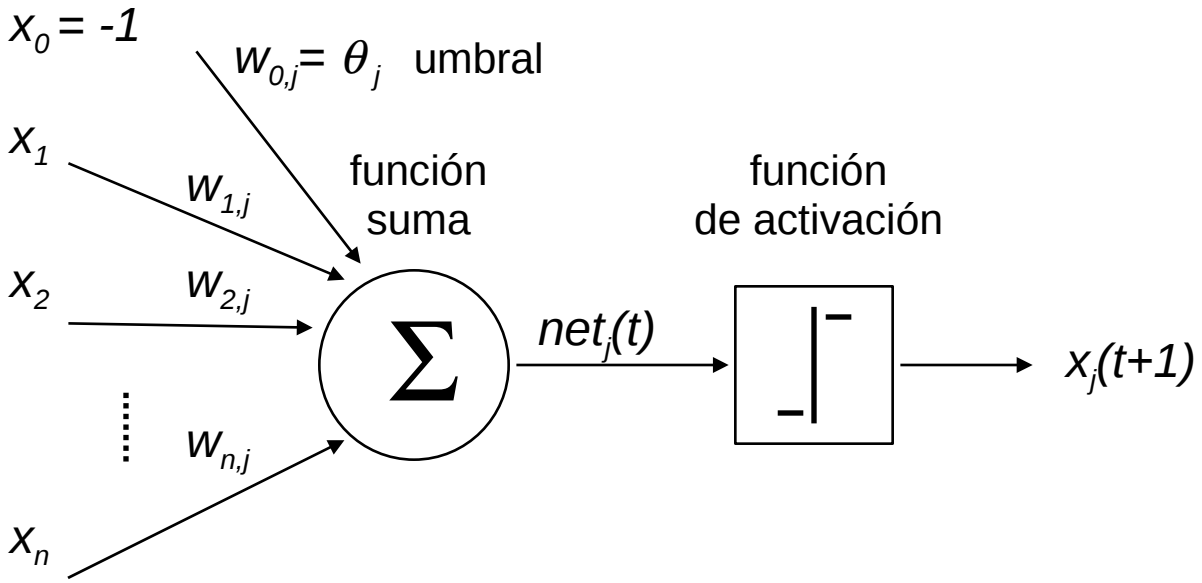
- *Introduction to the theory of neural computation*. Hertz, Krogh, Palmer. Addison-Wesley. 1991.
- *Fundamentals of Neural Networks. Architectures, algorithms, and applications*. Fausett. Prentice-Hall.1994.
- *Neural Networks. A comprehensive foundation*. Haykin. Macmillan.1994.
- *Neurocomputing*. Hecht-Nielsen. Addison-Wesley. 1989.
- *Redes Neuronales. Algoritmos, aplicaciones y técnicas de programación*. Freeman / Skapura. 1993.
- *Neural Computing. Theory and Practice. Wasserman*. Van Nostrand. 1989

- Tema 2. Características de las RNA

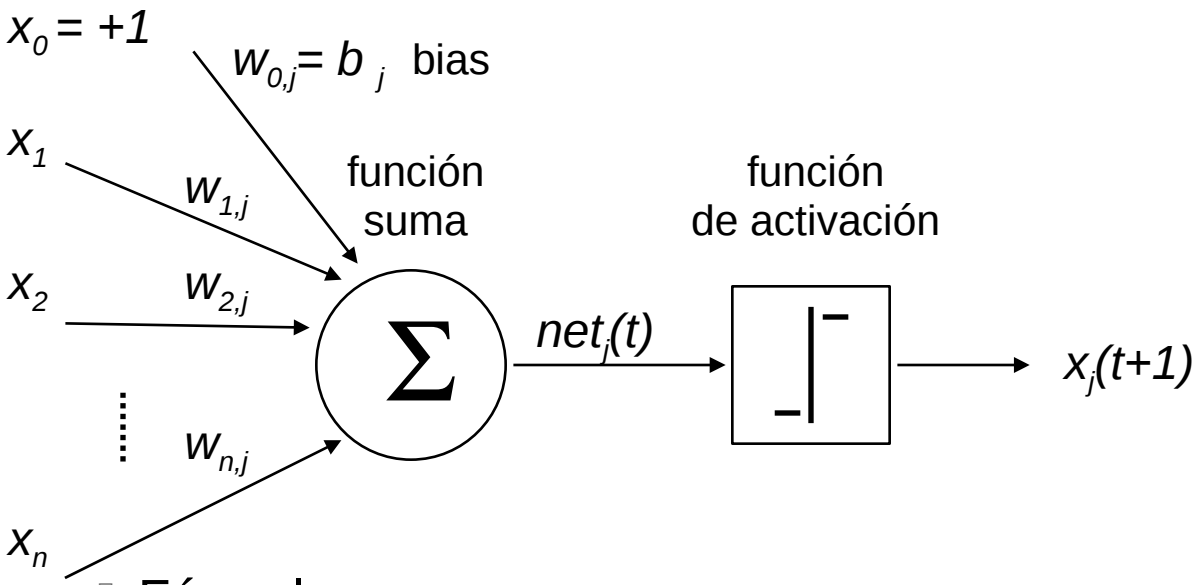
Características

- Modelo de Neurona





Características



- Fórmulas

$$net_j(t) = \sum_i w_{ij} x_i$$ función suma

$$x_j(t+1) = f_{act}\left(net_j(t), x_j(t), \theta_j\right)$$ función de activación

$$f_{act}\left(net_j(t), x_j(t), \theta_j\right) = \begin{cases} 1 \Rightarrow net_j(t) - \theta_j \geq 0 \\ 0 \Rightarrow net_j(t) - \theta_j < 0 \end{cases}$$ función escalón

$$f_{act}\left(net_j(t), x_j(t), \theta_j\right) = \begin{cases} 1 \Rightarrow net_j(t) - \theta_j \geq \frac{1}{2} \\ net_j(t) - \theta_j + \frac{1}{2} \\ 0 \Rightarrow net_j(t) - \theta_j \leq -\frac{1}{2} \end{cases}$$ función lineal a trozos

$$f_{act}\left(net_j(t), x_j(t), \theta_j\right) = \frac{1}{1 + \exp\left(-a \cdot \left(net_j(t) - \theta_j\right)\right)}$$ función sigmoide

$$f_{act}\left(net_j(t), x_j(t), \theta_j\right) = tanh\left(\frac{net_j(t) - \theta_j}{2}\right)$$ f. tangente hiperbólica

Características

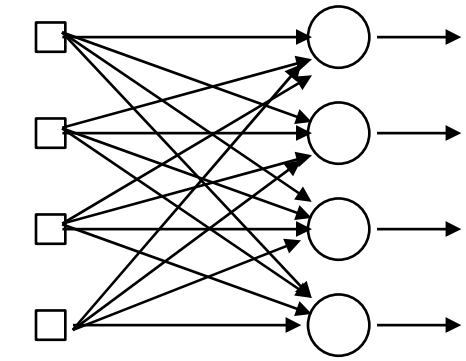
Arquitecturas de red

- Según e/o/s
  - Número y tipo de entradas
  - Elementos ocultos
  - Elementos de salida
- Según conectividad entre capas
  - Feedforward (hacia adelante)
    - Monocapa Feedforward
    - Multicapa Feedforward
  - Redes Recurrentes
    - Sin neuronas ocultas
    - Con neuronas ocultas
  - Estructuras Enrejadas (Lattice)
- Según conexión entre capas
  - Totalmente conectados (full-conexión)
  - Parcialmente conectados
  - Conexión uno a uno
- Sincronía (actualización de valores)
  - Simultánea
  - Aleatoria
  - Según orden topológico

Características

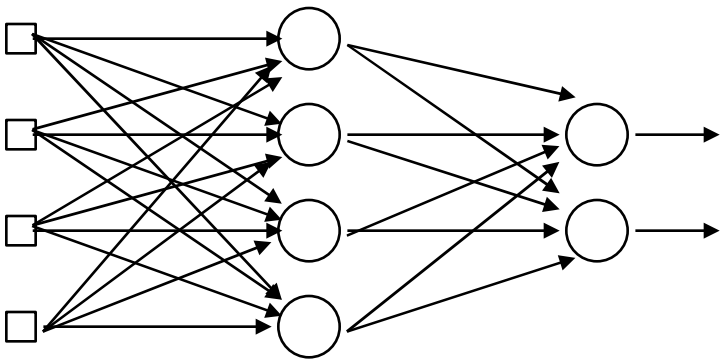
Arquitecturas

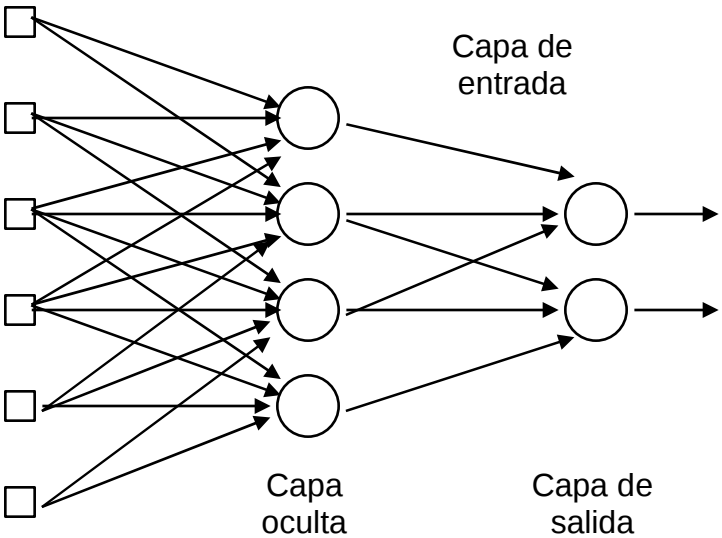
- Ejemplos feedforward



Monocapa feedforward

Multicapa feedforward



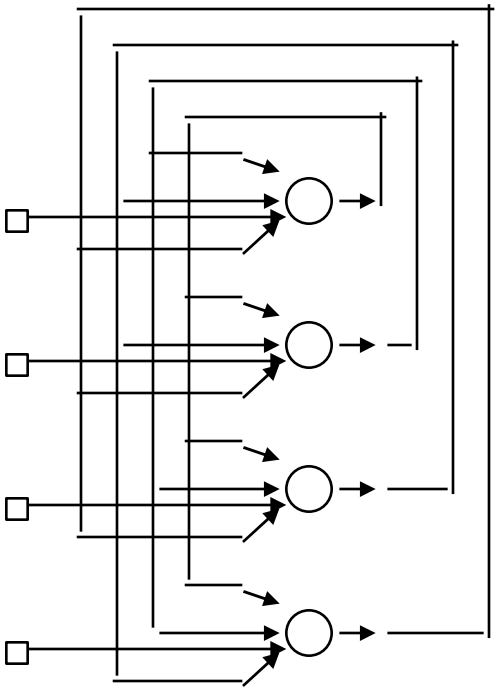


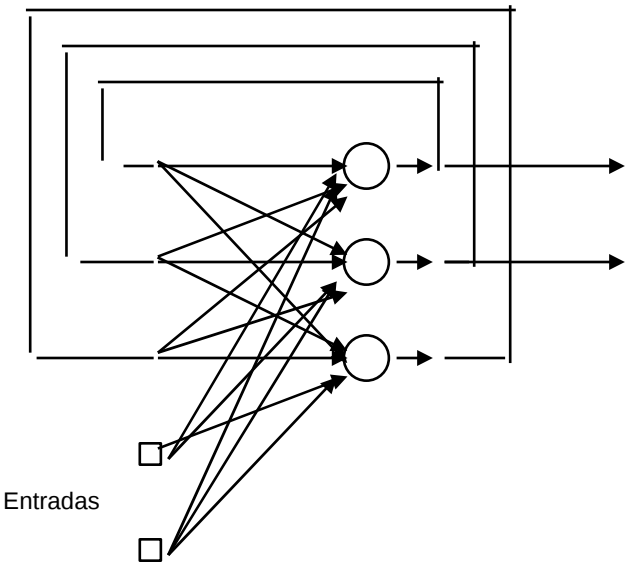
Multicapa feedforward parcialmente conectada

Características

- Ejemplos recurrentes

Recurrente con interconexiones laterales

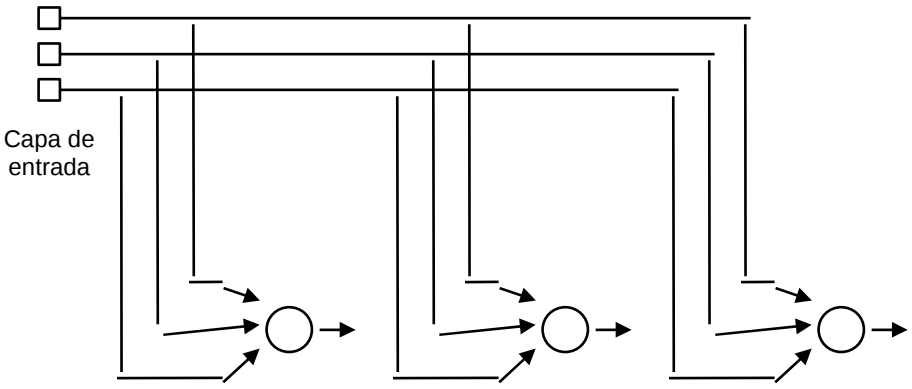




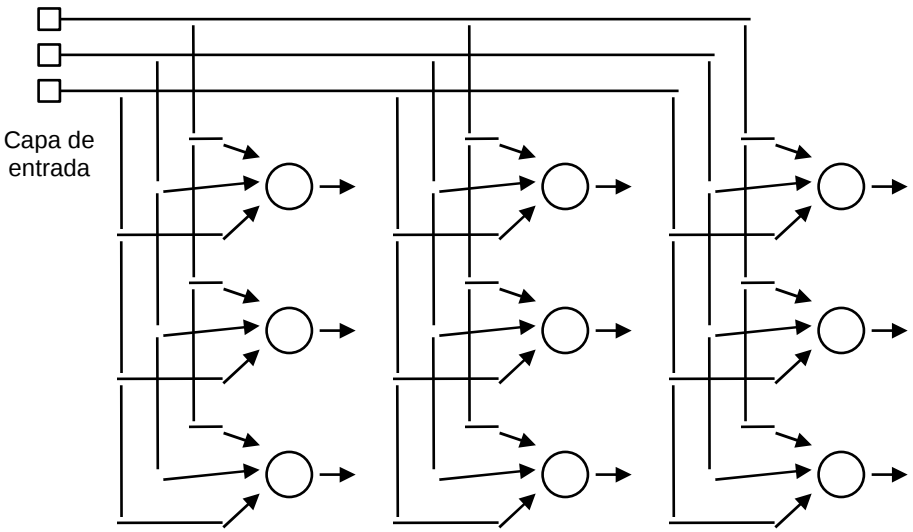
Recurrente con neuronas ocultas

Características

- Ejemplos enrejadas



Enrejada de una dimensión



Enrejada de dos dimensiones

Características

- Tipos de aprendizaje
  - Eventos:
    - Estimulación de la RN por el entorno
    - Cambios en la RN debido a estimulación
    - Nueva forma de responder debido a cambios de la estructura interna de la RN
  - Taxonomía:
    - Paradigmas de aprendizaje:
      - Aprendizaje Supervizado
      - Aprendizaje por Reforzamiento
      - Aprendizaje Auto-organizado (No Supervizado)
      - Precalculado o prefijado
    - Algoritmos de aprendizajes (reglas):
      - Corrección del error
      - Optimización de la energía (Apr. de Boltzmann)
      - Hebbiano
      - Competitivo
    - Otros:
      - local vs. global, por dato vs. por épocas, valores iniciales de pesos,

Características

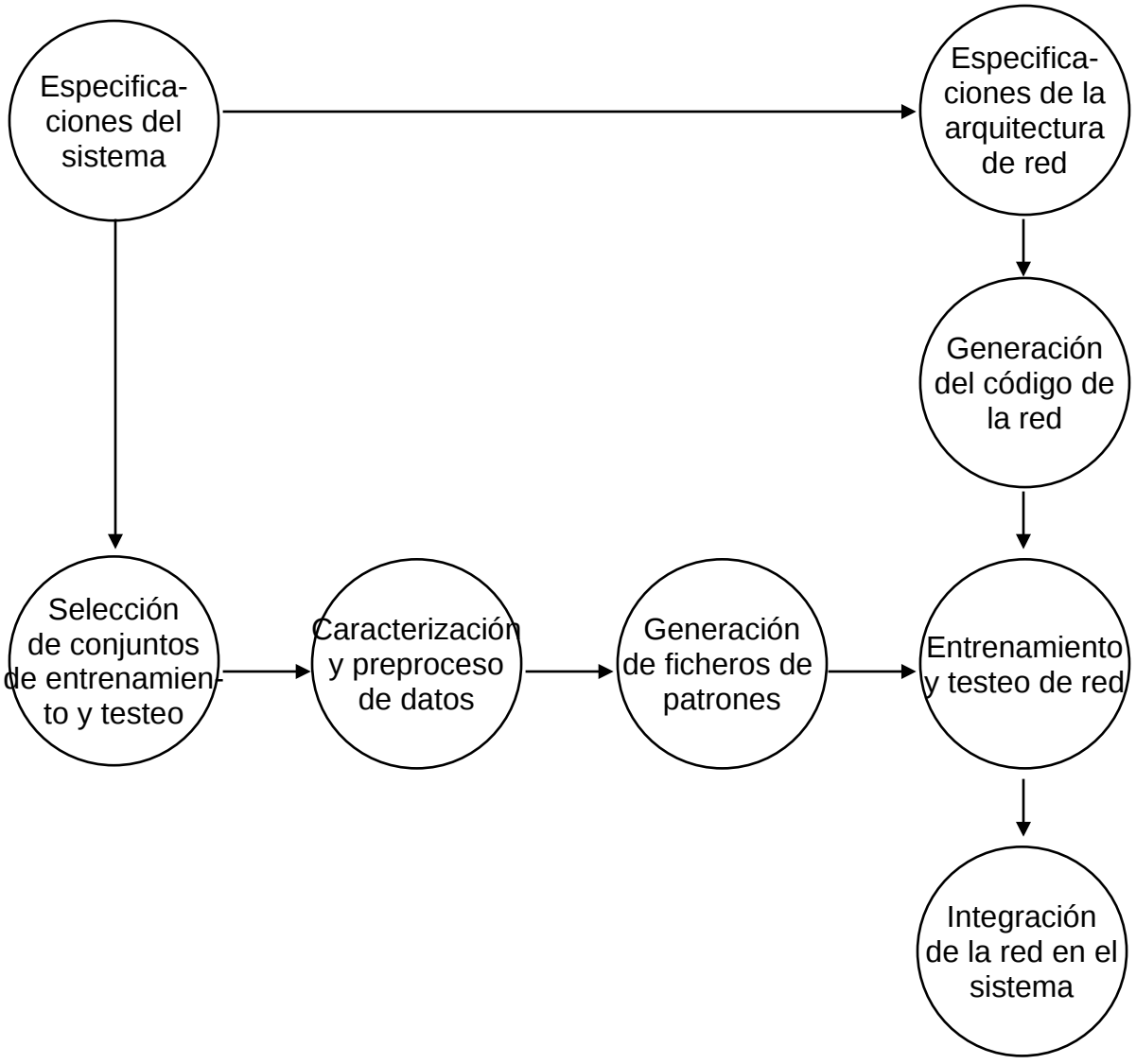
- Tipos de problemas
  - Aproximación
  - Asociación
  - Clasificación de Patrones
  - Predicción
  - Control
  - *Beamforming*

Características

- Codificación e/s
  - Tipo de señales
    - Global vs. Local
    - Discretas vs. Continuas
  - Relación e/s
    - Leer salida para cada entrada
    - Evolución de la salida ante la entrada
  - Preparación de datos
    - Diseño de conjuntos de entrenamiento y testeo
    - Longitud de la distribución (Grandes conjuntos)
    - Eliminación de información redundante e irrelevante
      - Manualmente
      - Preproceso de reducción de datos (extracción de características)
        - PCA, Momentos de Zernike, ...
    - Centrado y normalización
    - Eliminación de no-linealidades conocidas

Características

- Fases de desarrollo



Bibliografía

- *Neural Networks. A comprehensive foundation*. Haykin. Macmillan.1994.
- *Descripción Formal de Modelos de Redes Neuronales.* J.R. Álvarez. IX Cursos de Verano de la UNED.1998.
- *Neural Network PC Tools. A Practical Guide.* Eberhart and Dobbins. Academic Press. 1990.
- *Neurocomputing*. Hecht-Nielsen. Addison-Wesley. 1989.
- *Neurosmithing: Improving Neural Network Learning*. Reed and Marks. Arbid.Mit Press. 1995.

- Tema 3. Modelos
  - Perceptrones simples

Perceptrón Simple

- Perceptrón simple
  - Autor más representativo: Rosenblat(1962)
  - RNs de una capa de cómputo
  - E reales / S bipolares o binarias
  - Modelo de Neurona (3º):

    Función de suma:

$$net_j = b_j + \sum_i x_i w_{ij}$$

    - Función de activación: bipolar o binaria

$$f(net_j) = \begin{cases} 1 \Rightarrow net_j > \phi \\ 0 \Rightarrow -\phi \le net_j \le \phi \\ -1 \Rightarrow net_j < -\phi \end{cases}$$

  - Regla Aprendizaje
    - Supervizado mediante Corrección de Error
    - Ajuste de pesos:

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

      - Donde e_j es el error ajustado a [-1,0,1] y αel ratio de aprendizaje

Perceptrón Simple

Algoritmo Aprendizaje:

Inicailizar pesos y bias $\in$ (-1,0,1) , $0 < \alpha \leq 1.0$

Repetir

Para cada patrón

Calcular salida de cada neurona *j*

$$net_j = b_j + \sum_i x_i w_{ij}$$

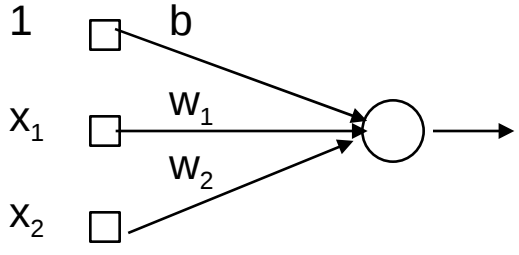
$$x_j = f(net_j) = \begin{cases} 1 \Rightarrow net_j > \phi \\ 0 \Rightarrow -\phi \leq net_j \leq \phi \\ -1 \Rightarrow net_j < -\phi \end{cases}$$

Modificar pesos de cada conexión *ij*

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

Hasta no se modifiquen pesos en bucle anterior

Perceptrón para función AND



Perceptrón Simple

Ejemplo de evolución al evaluar una función AND.

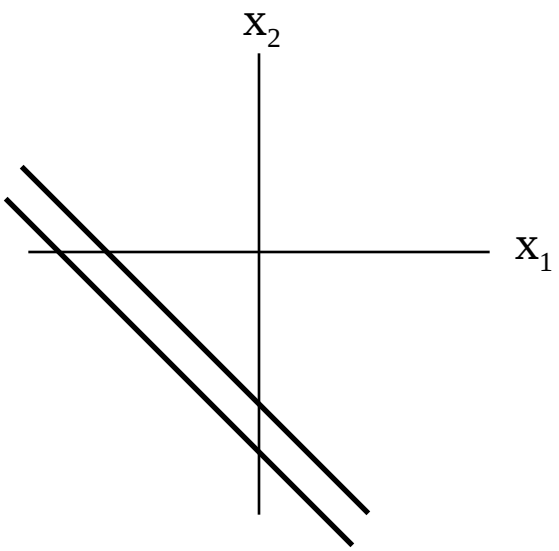
Entrada	Net	Salida	Objetivo	Cambio W	W Actuales
$\alpha = 1,\ \phi = 0.2$					
(x1, x2, 1)				*(ΔW)*	*(w1, w2, b)*
					(0 0 0)
(1 1 1)	0	0	1	(1 1 1)	(1 1 1)
(1 0 1)	2	1	-1	(-1 0 -1)	(0 1 0)
(0 1 1)	1	1	-1	(0 -1 -1)	(0 0 -1)
(0 0 1)	-1	-1	-1	(0 0 0)	(0 0 -1)
Resultados de la segunda pasada:					
(1 1 1)	-1	-1	1	(1 1 1)	(1 1 0)
(1 0 1)	1	1	-1	(-1 0 -1)	(0 1 -1)
(0 1 1)	0	0	-1	(0 -1 -1)	(0 0 -2)
(0 0 1)	-2	-1	-1	(0 0 0)	(0 0 -2)
Resultados de la tercera pasada:					
(1 1 1)	-2	-1	1	(1 1 1)	(1 1 -1)
(1 0 1)	0	0	-1	(-1 0 -1)	(0 1 -2)
(0 1 1)	-1	-1	-1	(0 0 0)	(0 1 -2)
(0 0 1)	-2	-1	-1	(0 0 0)	(0 1 -2)
Resultados de la cuarta pasada:					
(1 1 1)	-1	-1	1	(1 1 1)	(1 2 -1)
(1 0 1)	0	0	-1	(-1 0 -1)	(0 2 -2)
(0 1 1)	0	0	-1	(0 -1 -1)	(0 1 -3)
(0 0 1)	-3	-1	-1	(0 0 0)	(0 1 -3)
Continuamos iterando.....					
Resultados de la pasada décima en la que desaparece el error:					
(1 1 1)	1	1	1	(0 0 0)	(2 3 -4)
(1 0 1)	-2	-1	-1	(0 0 0)	(2 3 -4)
(0 1 1)	-1	-1	-1	(0 0 0)	(2 3 -4)
(0 0 1)	-4	-1	-1	(0 0 0)	(2 3 -4)

Perceptrón

Separabilidad lineal

$$w_{1j}x_1 + w_{2j}x_2 + b > \phi$$

$$w_{1j}x_1 + w_{2j}x_2 + b < -\phi$$



Perceptrón

- Teorema de Convergencia

  Si existe un vector de pesos *w** tal que para todo patrón *p* pertenciente al conjunto de patrones de entrenamiento *P*, $f_{act}(x(p) \cdot w^*) = d(p)$, entonces para cualquier *w* vector de pesos inicial el aprendizaje del perceptrón acabará convergiendo en un número finito de pasos a un vector de pesos (no necesariamente único ni necesariamente *w**) que da la respuesta correcta a todos los patrones de entrenamiento.

Perceptrón

- Discusión
  - Según Minsky y Papert(1969,1988)

    Limitaciones
    - Separabilidad lineal
    - Concavidad, convexidad
    - Problemas clásicos: xor, paridad
    - Incapacidad de generalización global en base a ejemplos localmente aprendidos
  - Conjetura de que estas limitaciones se extienden a los multicapas
  - Conjetura no justificada, véase algoritmos multicapas back-propagation y funciones de base radial

Adaline

Adaline (Adaptive Linear Neuron)

Widrow & Hoff (1960)

- RNs de una cápa de cómputo
- E/S bipolares (-1,1) ó E/S en el rango real
- Modelo de Neurona (3º):
  - Función de suma:

$$net_j = b_j + \sum_i x_i w_{ij}$$

  - Función de activación: identidad

$$f_{act}(net) = net$$

- Regla Aprendizaje
  - Supervizado mediante Corrección de Error
  - Regla: Delta, gradiente descendente , LMS (least mean squares) o Widrow-Hoff
  - Ajunste de pesos:

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

    - Donde e_j es el error (d_j - x_j) y α el ratio de aprendizaje

Adaline

- Algoritmo Aprendizaje:
  - Inicailizar pesos (valores aleatorios pequeños) , bias, α ($0.1 \leq n \cdot \alpha \leq 1.0$, / n=número entradas)
  - Repetir
    - $max\Delta w = 0$
    - Para cada patrón
      - Calcular salida de cada neurona *j*

$$x_j = net_j = b_j + \sum_i x_i \cdot w_{ij}$$

      - Modificar pesos de cada conexión *ij*

$$e_j = d_j - net_j$$

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

    - $max\Delta w = max(max\Delta w , |\Delta w_{ij}|)$
  - Hasta $max\Delta w < Tolerancia$
  - Opcionalmente, utilizar ahora función de activación:

$$f_{act}(net) = \begin{cases} 1, net \geq 0 \\ -1, net < 0 \end{cases}$$

Adaline

Derivación de la regla delta

Para una neurona:

Error cuadrático: $E = \frac{1}{2}(d_j - x_j)^2$

Para reducirlo, ajustar en dirección: $-\frac{\partial E}{\partial w_{ij}}$

Ya que:

$$x_j = \sum_k x_k \cdot w_{kj}$$

entonces:

$$\frac{\partial E}{\partial w_{ij}} = -(d_j - x_j)^2 \cdot \frac{\partial x_j}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = -(d_j - x_j) \cdot x_i$$

luego:

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

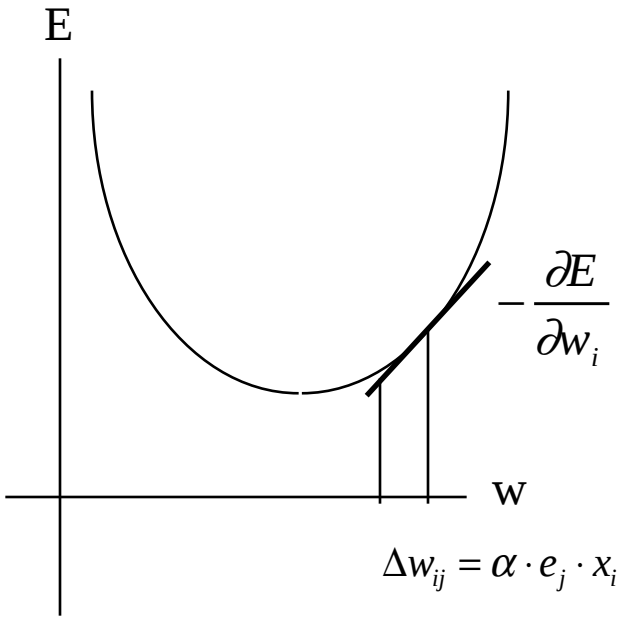
- Para varias neuronas

Error cuadrático: $E = \frac{1}{2}\sum_k (d_j - x_j)^2$

Cuya derivada resulta igual a la anterior

Adaline

Minimización bajo la derivada



- Variación de α según entorno de operación:
  - Estacionario
  - No estacionario:

    Seguimiento del mínimo con bájo α

Adaline

Ejemplo de evolución al evaluar una función AND.

Función a minimizar:

$$E = \sum_{k=1}^{4} \left(d_j^k - x_j^k \right)^2 =$$

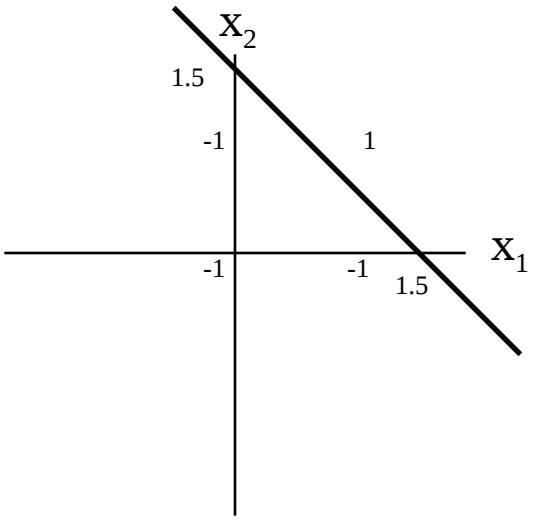
$$\sum_{k=1}^{4} \left(d_j^k - \left(x_0^k \cdot w_{0j} + x_1^k \cdot w_{1j} + x_2^k \cdot w_{2j} \right) \right)^2$$

- Pesos que minimizan dicha función:

$$w_{0j} = -\frac{3}{2},$$
$$w_{1j} = 1,$$
$$w_{2j} = 1$$

- Línea de separación:

$$x_1 + x_2 - \frac{3}{2} = 0$$



Bibliografía

Neural Networks. A comprehensive foundation. Haykin. Macmillan.1994.

- *Descripción Formal de Modelos de Redes Neuronales.* J.R. Álvarez. IX Cursos de Verano de la UNED.1998.
- *Fundamentals of Neural Networks.* Laurene Fausett. Prentice-Hall. 1994.

Tema 3. Modelos

- Backpropagation

Backpropagation

- Backpropagation

  Invención independiente de: Bryson, 1969; Werbos, 1974; Parker, 1985; Lecun, 1985; Rumelhart, Hinton, Williams, 1986.

  - Soporta una o más capas ocultas
  - E binarias o reales / S en principio binarias, aunque las salidas se aproximan por una función continua no lineal (sigmoide)
  - Modelo de Neurona (3º):

    Función de suma:

$$net_j = b_j + \sum_i x_i w_{ij}$$

    - Función de activación: sigmoide

$$f_{act}(net) = \frac{1}{1 + \exp(-net)}$$

  - Regla de aprendizaje:
    - Supervizado mediante Corrección de Error
    - Ajuste de pesos por Retropropagación del Error

Backpropagation

Algoritmo Aprendizaje:

Inicailizar pesos y bias

Repetir

Para cada patrón

Para cada capa (desde la 1ª a la última)

Calcular salida de cada neurona *j*

$$net_j = b_j + \sum_i x_i w_{ij}$$

$$f_{act}(net) = \frac{1}{1+\exp(-net)}$$

Para cada conexión *ij* de la capa de salida

$$\delta_j = f'_{act}\left(net_j\right) \cdot e_j$$

$$\Delta w_{ij} = \alpha \cdot \delta_j \cdot x_i$$

$$f'_{act}(net) = f_{act}(net) \cdot \left(1 - f_{act}(net)\right)$$

Para cada conexión *ij* de capas anteriores

$$\delta_j = f'_{act}\left(net_j\right) \cdot \sum_{k=1}^{m} \delta_k \cdot w_{jk}$$

$$\Delta w_{ij} = \alpha \cdot \delta_j \cdot x_i$$

Hasta error pequeño ó alcancemos nmi

Backpropagation

Elecciones

Pesos y bias

Influyen en alcanzar el error global o local y la velocidad con que se alcanza

- Si se dan valores muy grandes $\Rightarrow$caida en valores de derivada pequeños (saturación)
- Si se dan valores muy pequeños $\Rightarrow$salidas próximas a 0 $\Rightarrow$lentitud
- Experimentalmente en intervalo:

$$[-0.5,0.5] \text{ o } \left[-\frac{2.4}{fanin},\frac{2.4}{fanin}\right] \text{o } \left[-\frac{1}{\sqrt{fanin}},\frac{1}{\sqrt{fanin}}\right]$$

Tiempo de aprendizaje

- Hasta que el error sea menor que cota definida a priori
- Hasta que pendiente de error sea menor que cota definida a priori
- Hasta que pendiente de pesos sea menor que cota definida a priori
- Jugar con 2 conjuntos, mientras no empeore el error de testeo (*Early stopping*)

- Número de pares de entrenamiento
  - Según Baum-Haussler (1989) con *P*=número de patrones, *W*=número de pesos y ε fración de error => $P>W/\varepsilon$
  - *Shuffled* de patrones

Backpropagation

Elecciones

- Número de capas ocultas
  - Una capa oculta suele bastar
  - Más capas pueden facilitar la tarea
- Número de neuronas ocultas
  - Cuanto menos mejor:
    - Aprendizaje más rápido
    - Mayor capacidad de generalización
    - Menor posibilidad de parálisis
  - Técnicas de eliminación y/o creación de neuronas
- Ratios de aprendizaje
  - Grande $\Rightarrow$ parálisis o inestabilidad
  - Pequeños $\Rightarrow$ lentitud en aprendizaje
  - Ratios adaptivos

Backpropagation

Obtención de la regla:

- Error a minimizar:

$$E = \frac{1}{2} \cdot \sum_k \left(d_k - x_k \right)^2$$

- Para última capa:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \frac{1}{2} \cdot \sum_k \left(d_k - x_k \right)^2 =$$

$$= \frac{\partial}{\partial w_{ij}} \frac{1}{2} \cdot \sum_{k=j} \left(d_k - f_{act}\left(net_k \right) \right)^2 =$$

$$= -\left(d_j - x_j \right) \cdot \frac{\partial}{\partial w_{ij}} f_{act}\left(net_j \right) =$$

$$= -\left(d_j - x_j \right) \cdot f'_{act}\left(net_j \right) \cdot \frac{\partial}{\partial w_{ij}} net_j =$$

$$= -e_j \cdot f'_{act}\left(net_j \right) \cdot x_i = -\delta_j \cdot x_i$$

donde:

$$\delta_j = e_j \cdot f'_{act}\left(net_j \right)$$

Backpropagation

Para capas ocultas:

$$\frac{\partial E}{\partial w_{ij}} = \sum_k \frac{\partial E}{\partial x_k} \frac{\partial}{\partial w_{ij}} x_k = -\frac{1}{2} \cdot 2 \cdot \sum_k \left(d_k - x_k\right) \cdot \frac{\partial}{\partial w_{ij}} x_k =$$

$$= -\sum_k \left(d_k - x_k\right) \cdot f'_{act}\left(net_k\right) \cdot \frac{\partial}{\partial w_{ij}} net_k =$$

$$= -\sum_k e_k \cdot f'_{act}\left(net_k\right) \cdot \frac{\partial}{\partial w_{ij}} net_k = -\sum_k \delta_k \cdot \frac{\partial}{\partial w_{ij}} net_k$$

$$\text{donde}: \delta_k = e_k \cdot f'_{act}\left(net_k\right)$$

entonces:

$$-\sum_k \delta_k \cdot \frac{\partial}{\partial w_{ij}} net_k = -\sum_k \delta_k \cdot \frac{\partial net_k}{\partial x_j} \frac{\partial}{\partial w_{ij}} x_j =$$

$$= -\sum_k \delta_k \cdot w_{jk} \cdot \frac{\partial}{\partial w_{ij}} x_j =$$

$$= -\sum_k \delta_k \cdot w_{jk} \cdot f'_{act}\left(net_j\right) \cdot \frac{\partial}{\partial w_{ij}} net_j =$$

$$= -\sum_k \delta_k \cdot w_{jk} \cdot f'_{act}\left(net_j\right) \cdot x_i = -\delta_j \cdot x_i$$

donde:

$$\delta_j = f'_{act}\left(net_j\right) \cdot \sum_k \delta_k \cdot w_{jk}$$

Backpropagation

- Variantes
  - Momentum
    - Rumelhart, Hinton, Williams, 1986
    - Añade un término proporcional al último Δw

$$\Delta(t+1)\,w_{ij} = \alpha \cdot \delta_j \cdot x_i + \eta \cdot \Delta(t)\,w_{ij}$$

  - Quickprop
    - Fahlman, 1988
    - Utiliza segunda derivada para estimar mejor la posición del mínimo
  - Batch
    - Acumula los Δw durante un ciclo y los aplica al final
  - Desventajas
    - Convergencia no demostrada
    - Lento proceso de aprendizaje
    - Incertidumbre en la convergencia debido a:
      - Parálisis de red
      - Mínimos locales

Tema 3. Modelos

Sistemas Autoorganizados

Sistemas Autoorganizados

- No supervizados
- Descubren:
  - Patrones
  - Características
  - Regularidades
  - Correlaciones
  - Categorias
- Basadas en la redundancia en las entradas
- Patrones detectables:
  - Familiaridad con patrones típicos o promedios del pasado
  - Análisis de las Componentes Principales
  - Clustering
  - Prototipos, correspondientes a las categorias existentes
  - Codificación
  - Mapa de Características

Sistemas Autoorganizados

- Diversos patrones pueden ser detectados por la misma red
  - Codificación <=> Análisis de las Componentes Principales (Reducción de Dimensionalidad)
  - Codificación <=> Clustering (Vector Quantization)
- Es posible combinar estos sistemas con sistemas supervizados, para:
  - Descubrimiento de características
    - Extraer características significativas de los datos de entrada.
    - Suponen cierta compresión óptima de datos, la alta dimensión de las entrada de los sensores es reducida a un vector de características.
    - Sólo la extracción de las componentes principales cumplen estos requerimientos.
  - Cuantización
    - Cuantizar las entradas utilizando reglas de aprendizaje competitivo.
    - Dicho preproceso permite simplificar los mapeos que necesitamos realizar, pudiendo transformarlos de no lineales a lineales.
    - Los algoritmos de aprendizaje para mapeos lineales presentan grandes ventajas:
    - Convergencia garantizada.
    - Alta velocidad de convergencia.
  - Seguimiento del mínimo
    - Una vez finalizado el aprendizaje supervizado.

PCA

- Red de Sanger o Generalized Hebbian Algorithm (GHA)
  - Red de una única capa
  - E/S continuas
  - Modelo de Neurona (1º):
    - Función de suma:

$$net_j = \sum_i x_i \cdot w_{ij}$$

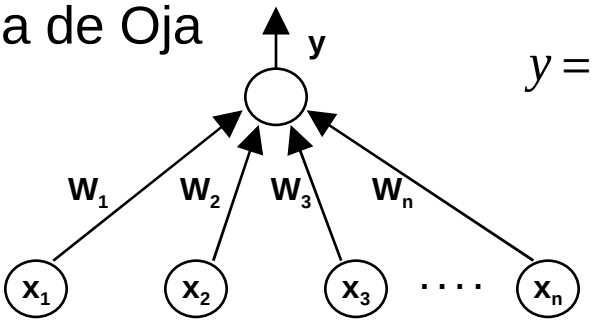
      - no existen umbrales
    - Función de activación: identidad

$$f_{act}(net) = net$$

  - Regla de aprendizaje:
    - No supervizado
    - Basada en la Regla de Sanger, una extensión de la Regla de Oja
    - Ajuste de pesos por aprendizaje Hebbiano

PCA

Regla de Oja



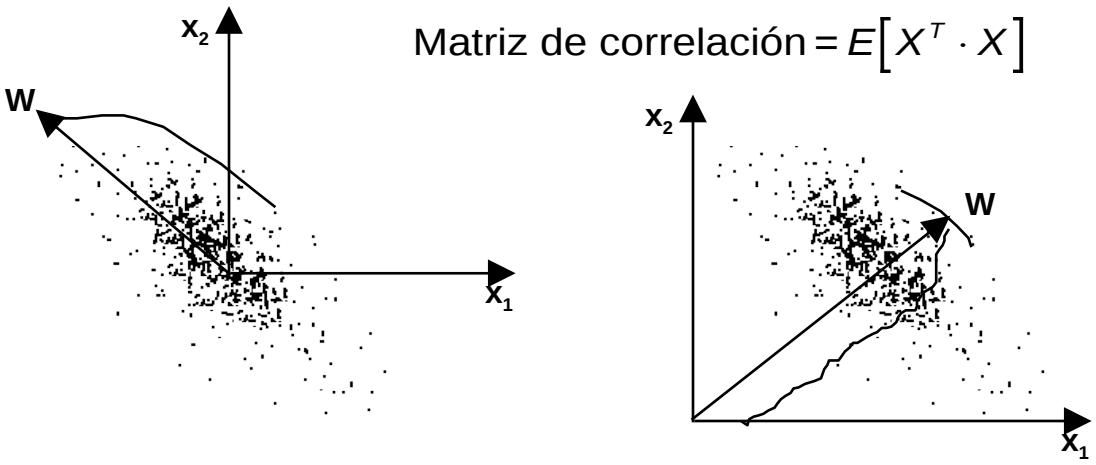
$$y = \mathbf{w} \cdot \mathbf{x} = \sum_i x_i \cdot w_i$$

Ajuste de pesos:

$$\Delta w_i = \alpha \cdot y \cdot (x_i - y \cdot w_i)$$

- Tiene la característica que:
  - La longitud de el vector de pesos W converge hacia 1, (|w| = 1) sin necesidad de hacer normalizaciones a mano.
  - El vector W converge a la Componente Principal de la entrada, esto es el autovector dominante de la matriz de correlación:

$$\text{Matriz de correlación} = E\left[X^T \cdot X\right]$$



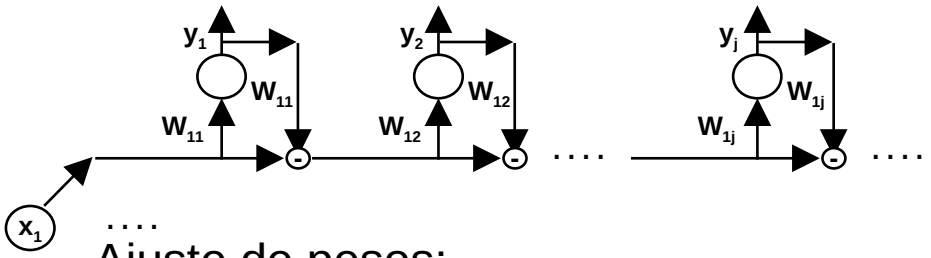
Ejemplo de ejecución de la regla de Oja

PCA

Regla de Sanger (Inhibición hacia atrás)

No basta el 1er autovector para representar fielmente las entradas, existen otros autovectores no tan dominantes.

- Para halla estos otros autovectores podemos basarnos en eliminar de los datos la influencia de los autovectores más dominantes.

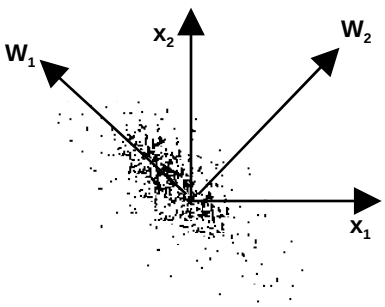


Ajuste de pesos:

$$\Delta w_{ij} = \alpha \cdot y_j \cdot \left(x_i - \sum_{k=1}^{j} y_k \cdot w_{ik} \right)$$

- En W obtenemos las Componentes Principales ordenadas por su magnitud
- Salida corresponde a transformar la entrada al espacio definido por las *j* primeras Componentes Principales.

$$Y = W \cdot X$$



Ejemplo de ejecución de la regla de Sanger

PCA

Algoritmo Aprendizaje:

Inicailizar pesos, α decreciente en *t*

Repetir

Para cada patrón

Calcular salida de cada neurona *j*

$$y_j = \sum_i x_i \cdot w_{ji}$$

Para cada conexión *ij*

$$\Delta w_{ij} = \alpha \cdot y_j \cdot \left(x_i - \sum_{k=1}^{j} y_k \cdot w_{ik} \right)$$

Hasta $|\Delta W| <$ cota fijada a priori

SOM

Self-Organizing Maps (mapas autoorganizados)

- Basado en los mapas que preservan la topología dentro de la corteza cerebral
- Unidades físicamente próximas responden a clases de vectores cercanos
- Modelo competitivo, con existencia de una neurona ganadora, que consigue modificar sus pesos y los de sus vecinas
- Topología monocapa basada en un array (1 ó 2D) de unidades totalmente conectadas con las entradas
- Modelo de Neurona (1º):

  Función de suma, según entradas normalizadas o no:

$$net_j = \sum_i x_i \cdot w_{ij}$$

$$net_j = \sum_i \left(x_i - w_{ij} \right)^2$$

  - Función de activación:
    - lineal
    - no existen umbrales

SOM

Regla de aprendizaje:

- No supervizado
- Basada en el algoritmo de Kohonen
- Ajuste de pesos por aprendizaje Competitivo

- Ajuste de pesos:

$$\Delta w_{ji} = \alpha(t) \cdot \Lambda(i,c,t) \cdot (x_i - w_{ji})$$

- Donde:
  - *c* es la neurona ganadora tal que:

$$\|X - W_c\| = \min_i \{\|X - W_i\|\}$$

  - $\alpha(t)$ es el ratio de aprendizaje, decreciente con el tiempo
  - $\Lambda(i,c,t)$ es la vecindad, que depende de la distancia con la unidad ganadora, y decrece con el tiempo, tiene diversas formas:

Sombrero mejicano

Gausiana

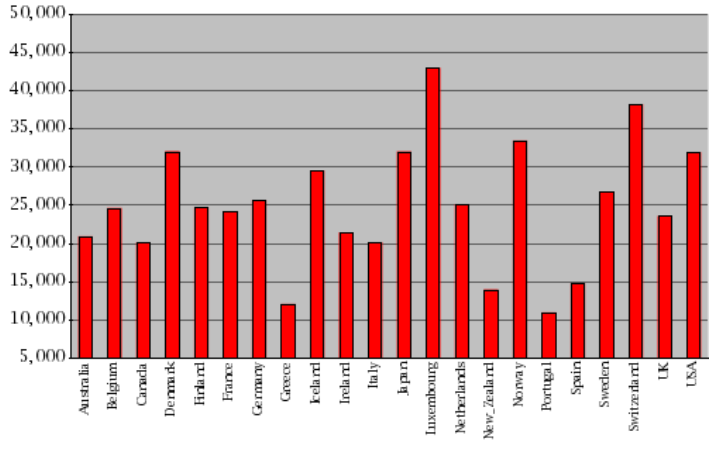
Rectangular

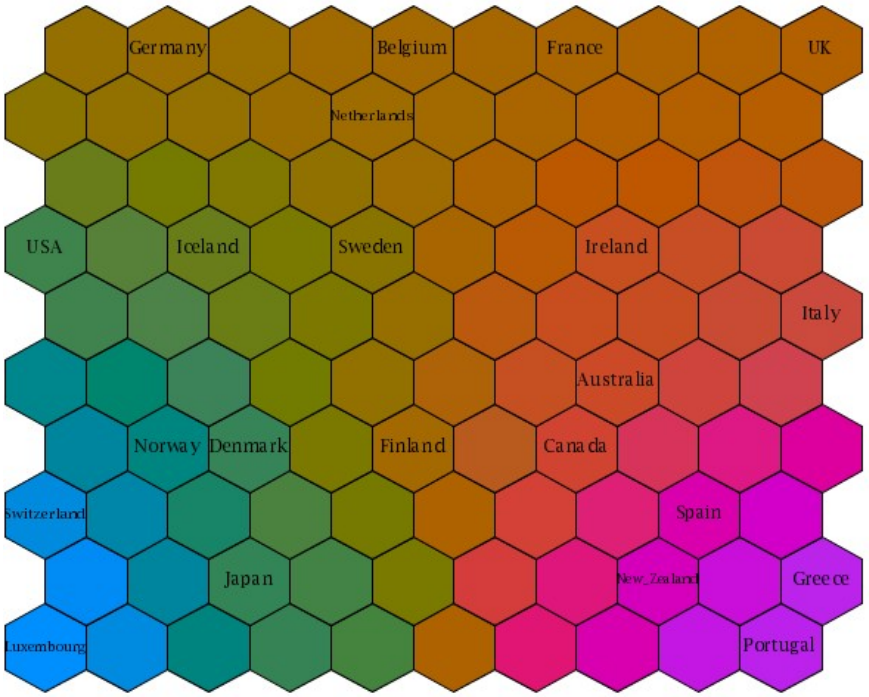
SOM

Visualización de datos multidimensionales (I)

http://products.davisor.com/chart/som/

Datos de Banco Mundial, 18 variables económicas y demográficas de paises de la OCDE en 1999



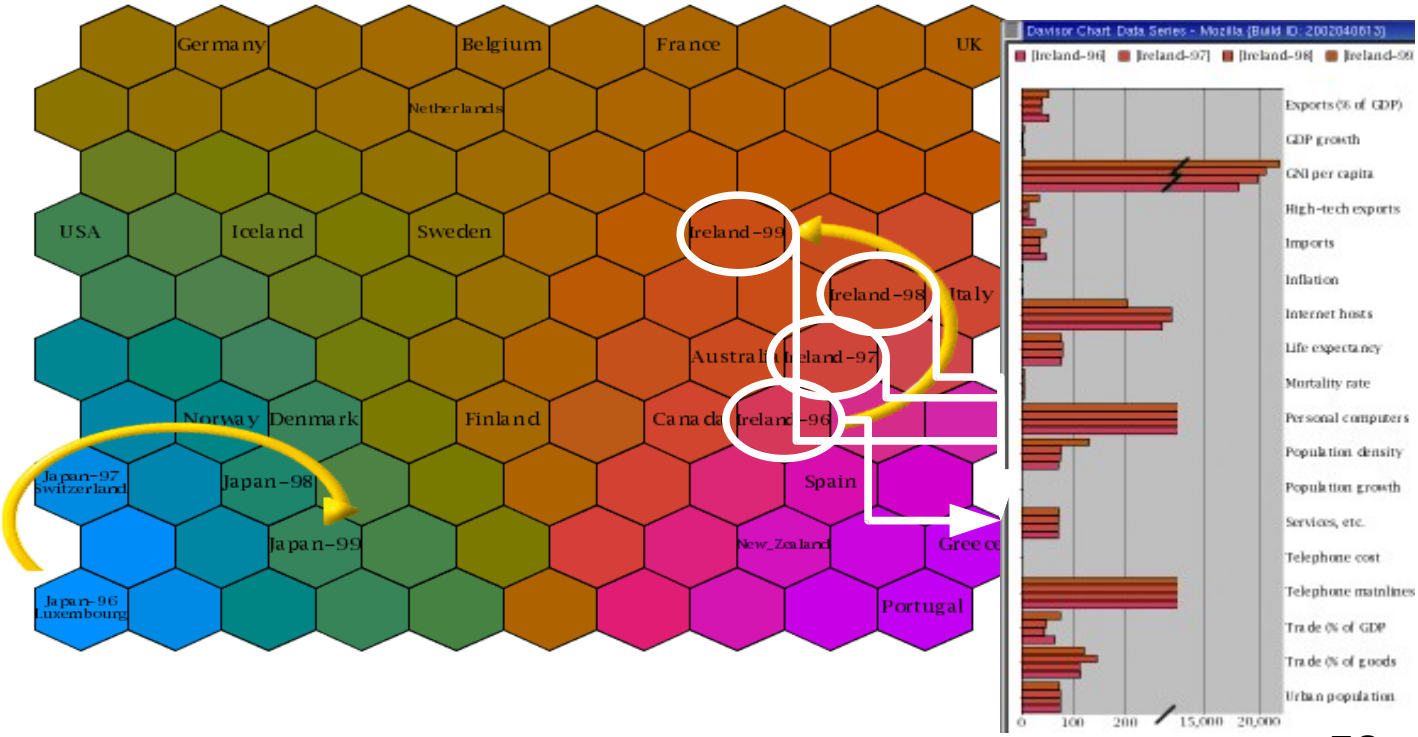


SOM

- Visualización de datos multidimensionales (II)

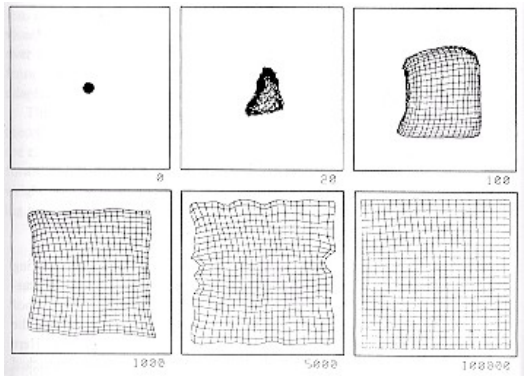
Ingresos brutos per cápita

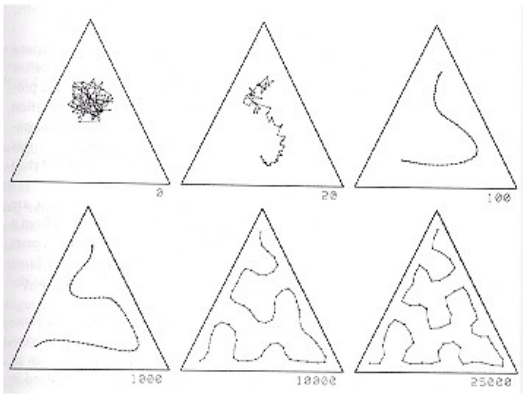
Exportación de materias y servicios

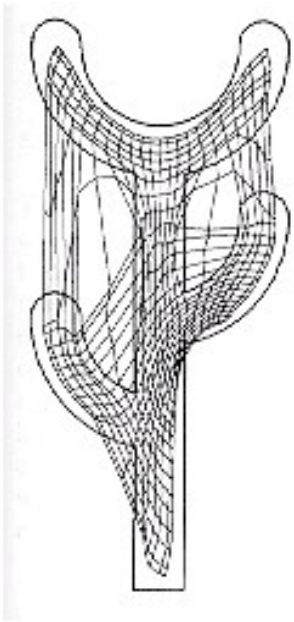


SOM

- Visualización de resultados observando los vectores de pesos en el espacio de entradas







- Variante: LVQ (Learning Vector Quantization)
  - Algoritmo de aprendizaje supervizado que utiliza la información de las clases para mejorar sobre SOM los vectores de pesos
- CounterPropagation
  - Kohonen + Grosberg

Bibliografía

Neural Networks. A comprehensive foundation. Haykin. Macmillan.1994.

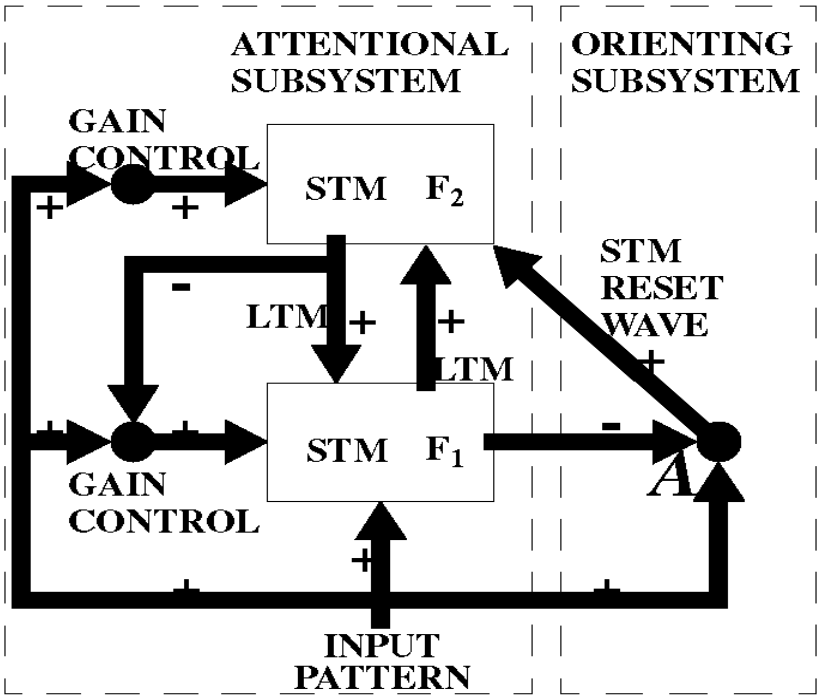
- *Descripción Formal de Modelos de Redes Neuronales.* J.R. Álvarez. IX Cursos de Verano de la UNED.1998.
- *Introduction to de Theory of Neural Computation.* J. Hertz, A Krogh, R. Plamer. Addison-Wesley. 1991.
- *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación*. J. A. Freeman, D.M. Sakapura. Addison-Wesley. 1993.
- *Neurocomputing*. Hecht-Nielsen. Addison-Wesley. 1989.
- *Fundamentals of Neural Networks. Architectures, algorithms, and applications*. Fausett. Prentice-Hall.1994.

Tema 3. Modelos

- ART1

ART1

- Adaptive Resonance Theory 1
  - Stephen Grossberg y Gail Carpenter (1987)
  - Satisface el dilema de estabilidad-plasticidad
  - Red recurrente de múltiples capas: F_1, F_2, ...
  - E/S binarias
  - Arquitectura y mecánica



  - Regla de aprendizaje:
    - No supervizado
    - Ajuste de pesos por aprendizaje Competitivo

ART1

- Algoritmo Aprendizaje:

  Inicailizar N al núm. max. clust., $0 < \rho \leq 1$, prototipos P_j todos a 1

  Repetir

    Para cada patrón I

      n_inhibidas = ∅

      Repetir

        j = mas_cercano(not(n_inhibidas), I)

        n_inhibidas = n_inhibidas + *j*

      Hasta suf_cercanos(P_j, I, ρ) o |n_inhibidas| = N

      Si not(suf_cercanos(P_j, I, ρ)) entonces

        error('Todas las neuronas inhibidas')

      FinSi

      $P_j = P_j \cap I$

    FinPara

  Hasta (sin cambios de pesos) o (NMI)

ART1

mas_cercano(n_activas, I)

Devuelve $j \in$ *n_activas* tal que maximice:

$$\frac{\|P_j \cap I\|}{\beta + \|P_j\|}$$

- suf_cercanos(P_j, I, ρ)

$$\frac{\|P_j \cap I\|}{\|I\|} \geq \rho$$

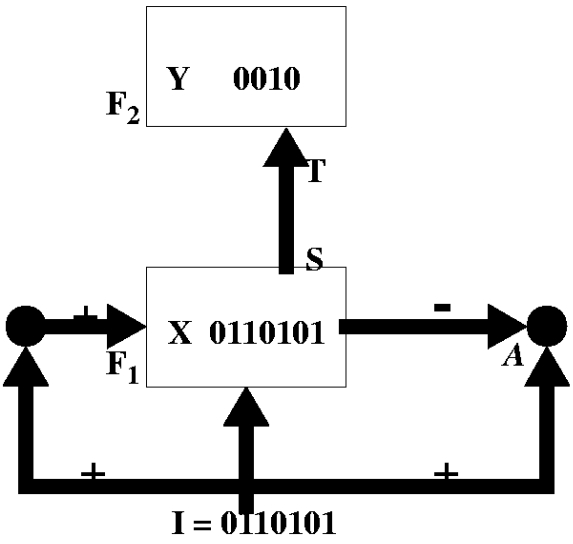
donde ρ es el parámetro de vigilancia

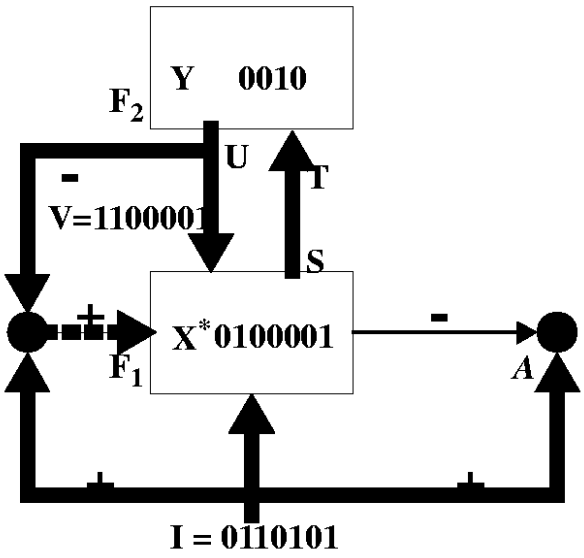
y ||...|| cuenta los unos

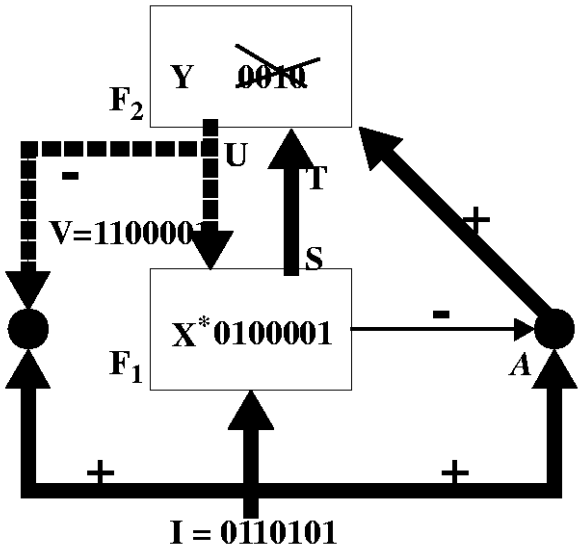
- Variantes
  - ART2: Versión con entradas reales
  - ART2A: ART2 rápida
  - ARTMAP: Versión supervizada de ART1
  - Fuzzy ART y Fuzzy ARTMAP: Basadas en lógica difusa
  - MART: ART para canales de entrada múltiple

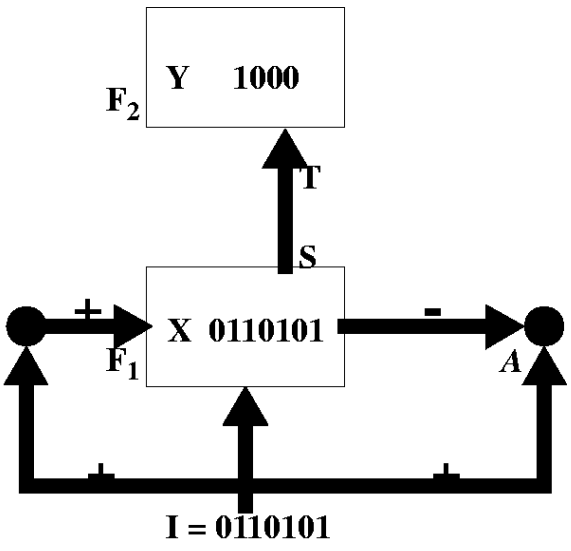
ART1

- Mecanismo de funcionamiento









Bibliografía

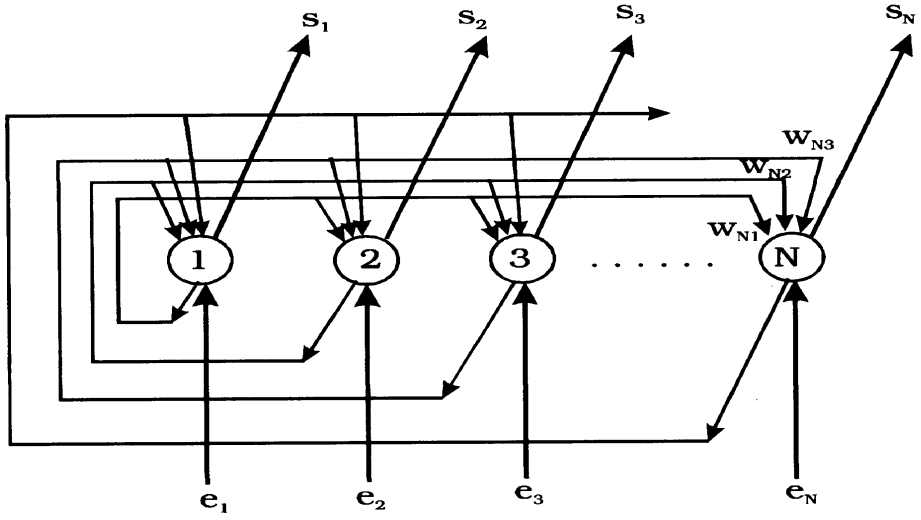
- *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación*. J. A. Freeman, D.M. Sakapura. Addison-Wesley. 1993.
- *Fundamentals of Neural Networks. Architectures, algorithms, and applications*. Fausett. Prentice-Hall.1994.
- http://web.umr.edu/~tauritzd/art/

Tema 3. Modelos

- Red de Hopfield

Red de Hopfield

- **Modelo de Hopfield Discreto**
  - **Introducción: Los modelos continuos y discretos de Hopfield establecen un paralelismo entre disciplinas tan dispares a priori como la neurocomputación y la física estadística. Los problemas en los que se centran estos modelos son problemas autoasociativos clásicos**
  - **Arquitectura: Es una red monocapa con N neuronas cuyos valores de salida son binarios 0/1 ó bipolares +1/-1. Cada neurona de la red se encuentra conectada a todas las demás (conexiones laterales), pero no consigo misma $w_{ii}=0$, y además los pesos simétricos son iguales $w_{ij}=w_{ji}$**



  - **La función de transferencia es de tipo escalón**
    - » **La versión discreta en principio se ideo para trabajar con valores bipolares (aunque se puede hacer un pequeño cambio de variable y pasar a el caso binario). La función de transferencia era:**

$$g(x)=\begin{cases}+1 & \text{si } x>\theta_i \\ \text{igual al valor en el paso anterior.} & \\ -1 & \text{si } x<\theta_i\end{cases}$$

$$\text{con} \quad \theta_i = k*\sum_{j=1}^{N} w_{ji} \quad \text{con } k=1/2 \text{ para binario}$$

    - » **aunque θ_i se suele tomar a 0, ya que no es útil para patrones aleatorios**

Red de Hopfield

- **Funcionamiento:**
  - **El modelo de Hopfield es una red autoasociativa, que puede ser visto como una derivación del modelo de Kosko, BAM. El cometido de la red es aprender una serie de patrones (sin una fase de entrenamiento, los pesos se calculan de una pasada) y después en la fase de evaluación cuando se presente alguno de los patrones aprendidos obtengamos como salida una copia de la entrada. No obstante si el patrón de entrada es alguno que se aprendió pero con ruido la red evolucionará generando como salida aquel patrón que se aprendió que más se asemeje.**
    - **1.- Los patrones de entrada se aplican una vez a la red y toman el valor de salida en el instante t=0.**

$$y_i(t=0) = x_i \quad \text{con } 1 \le i \le N$$

    - **2.- La red realiza iteraciones hasta alcanzar la convergencia (y(t+1)=y(t)).**

$$y_i(t+1) = g(\sum_{j=1}^{N} w_{ij} y(t) - \theta_i) \quad 1 \le i \le N$$

- **Aprendizaje:**
  - **La red de Hopfield tiene un aprendizaje precalculado. Los valores de los pesos se fijan en función de los patrones que se pretenden aprender. Una vez fijados los pesos la red entra en funcionamiento tal y como se acaba de ver.**
  - **El tipo de aprendizaje se puede ver como no supervisado, de regla basada en la Optimización de la energía (ver más adelante) y Hebbiana, dado que el peso que une dos neuronas i, j se calcula en función del valor de salida de ambos. Si el número de patrones a aprender es M el producto vectorial se repite a lo largo de todos los patrones.**
  - **En el caso de la red de Hopfield bipolar (-1/+1), este aprendizaje se puede expresar como:**

$$w_{ij} = \begin{cases} \sum_{k=1}^{M} x_i^{(k)} * x_j^{(k)} & 1 \le i, j \le N; i \ne j \\ 0 & 1 \le i, j \le N; i = j \end{cases}$$

w_{ij}: Peso que une las neuronas j e i.

$x_i^{(k)}$: Valor de la componemte i - esima del k - esimo patron.

N: Numero de neuronas de la red.

M: Numero de patrones que se debe aprender.

Red de Hopfield

» **Si la red trabajase con valores discretos binarios (0/1), los pesos de deben calcular según:**

$$w_{ij} = \begin{cases} \sum_{k=1}^{M} \left(2x_i^{(k)} - 1\right) * \left(2x_j^{(k)} - 1\right) & 1 \le i, j \le N; i \ne j \\ 0 & 1 \le i, j \le N; i = j \end{cases}$$

» **La información con la que se puede establecer la dinámica de la red es:**

- **Matriz de pesos: simétrica respecto a la diagonal principal, esta última con todos sus valores a cero.**

$$W = \begin{bmatrix} w_{11} & w_{21} & \ldots\ldots\ldots & w_{N1} \\ w_{12} & w_{22} & \ldots\ldots\ldots & w_{N2} \\ w_{13} & w_{23} & \ldots\ldots\ldots & w_{N3} \\ \ldots\ldots\ldots & & & \\ w_{1N} & w_{2N} & \ldots\ldots\ldots & w_{NN} \end{bmatrix}$$

- **Conjunto de M vectores representando la información que se pretende aprender.**

$$X_1 = \left[x_1^{(1)}, x_2^{(1)}, \ldots\ldots x_N^{(1)}\right]$$
$$X_2 = \left[x_1^{(2)}, x_2^{(2)}, \ldots\ldots x_N^{(2)}\right]$$
$$\ldots\ldots\ldots\ldots$$
$$X_M = \left[x_1^{(M)}, x_2^{(M)}, \ldots\ldots x_N^{(M)}\right]$$

» **Si empleamos la notación matricial podemos expresar la obtención de los pesos como:**

$$W = \sum_{k=1}^{M} \left[X_k^T X_k - I\right]$$

– **Ejemplo: Supongase que se quiere aprender la siguiente información, ver figura:**

Figura a:

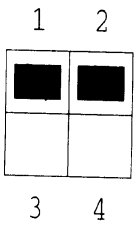
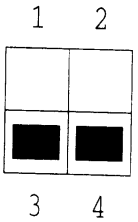


Figura b:



Red de Hopfield

» **Podemos representar los valores de entrada como:**

$$X_1 = [1,1,-1,-1]$$
$$X_2 = [-1,-1,1,1]$$

» **El aprendizaje OFF_LINE consiste en la obtención de la matriz de pesos:**

$$X_1{}^T X_1 - I = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} * [1,1,-1,-1] - \begin{bmatrix} 1000 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}$$

» **Y para el segundo patrón:**

$$X_2{}^T X_2 - I = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} * [-1,-1,1,1] - \begin{bmatrix} 1000 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}$$

» **Y la matriz de pesos definitivos se obtiene como:**

$$W = \sum_{k=1}^{M} \left[X_k{}^T X_k - I \right] = \begin{bmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{bmatrix}$$

» **Si suponemos que la función de transferencia es de tipo escalón centrada en el origen y queremos evaluar hasta que punto la red actúa como autoasociador, testeamos la red con el patrón:**

$$X_t = [1,-1,-1,-1]$$

» **Inicialmente la salida de la red corresponde con la entrada de la red, obteniendose las siguiente activaciones:**

$$X_t(t)\,W = [1,-1,-1,-1] \begin{bmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{bmatrix} = [2,5,-2,-2]$$

» **La salida de la red es:**

$$Y(t) = X(t+1) = [1,1,-1-1]$$

Red de Hopfield

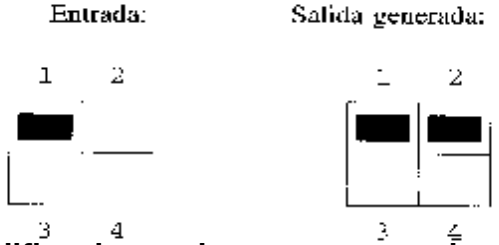
Si se repite el proceso, los valores de activación serán:

$$X_t(t+1)\,W = [1,1,-1,-1]\begin{bmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{bmatrix} = [6,6,-6,-6]$$

» **Luego después de la segunda iteración, la red ha llegado a un estado estable:**

$$Y(t+1) = X(t+2) = [1,1,-1-1]$$

•Para la entrada proporcionada a la red, esta responde con el patrón aprendido más parecido. Ver figura:



– **Algunas modificaciones tienen en cuenta las conexiones autorecurrentes, con lo que no siempre $w_{ij} = 0$.**

– **Las funciones de energías:**

» **Dado que de nuevo se establecen a priori los pesos que almacenan los patrones deseados, y la dinámica del sistema esta determinada por las variables de estado, entradas y salidas de la red.**

» **La existencia de estados estables pasa por existencia de una Función de Lyapunov o función de energía.**

» **La función de energía de una red de Hopfield discreta tiene la siguiente forma:**

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ i\neq j}}^{N} w_{ij}y_i y_j + \sum_{i=1}^{N}\theta_i y_i$$

Donde:

w_{ij}: Peso de la conexion entre i y j.

y_i : Valor de salida de la neurona i.

θ_i : Umbral de la funcion transferencia de la neurona i.

» **La función de energía puede verse como una superficie de error que almacena M puntos estables, o mínimos, tantos como patrones haya aprendido la red.**

» **Cuando la red de Hopfield se utiliza como memoria asociativa, el objetivo es conseguir que los patrones a memorizar se sitúen en dichos mínimos.**

» **Se puede demostrar que esta situación se presenta cuando los pesos $w_{ii} = 0$, y en el resto se cumple la regla de Hebb:**

$$w_{ij} = \sum y_i y_j$$

Red de Hopfield

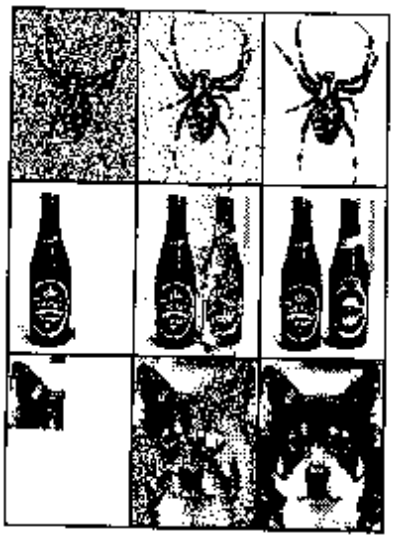
- **Limitaciones del modelo Hopfield.**
  - » **Los problemas relacionados son por un lado la cantidad limitada de datos que se pueden almacenar y por otro el que los datos deben ser ortonormales para lograr una buena separación lineal.**
  - » **La capacidad de la red se puede expresar para patrones binarios y bipolares respectivamente como:**

$$\text{Capacidad} = \begin{cases} 0.138N & \text{Para una recuperacion con errores} \\ \dfrac{N}{4\ln(N)} & \text{Para una recuperacion perfecta} \\ N & \text{Para patrones ortogonales} \end{cases}$$

  - » **La segunda limitación del modelo es que no siempre se puede garantizar que la red realice una buena separación, debido a que los patrones no suficientemente diferentes, esto es, ortonormales . Este problema puede ser minimizado mediante determinados procedimientos de ortogonalización, que garanticen una diferencia suficiente entre las informaciones que se desea aprender. Esta última condición ocurre entre dos patrones si al menos N/2 elementos de los patrones difieren. Esta condición se puede expresar como:**

$$\sum_{i=1}^{N} x_i^{(k)} x_i^{(m)} \leq 0 \quad \forall k \neq m$$

- **Aplicaciones del modelo discreto:**
  - » **Reconocimiento de imágenes.**
    - **Con este tipo de aplicación la red puede reconstruir, partiendo de imágenes parcialmente dañadas, las imágenes que en el proceso de aprendizaje se utilizaron para calcular los pesos.**
    - **Ver ejemplo:**



Tema 3. Modelos

Redes para el tratamiento temporal

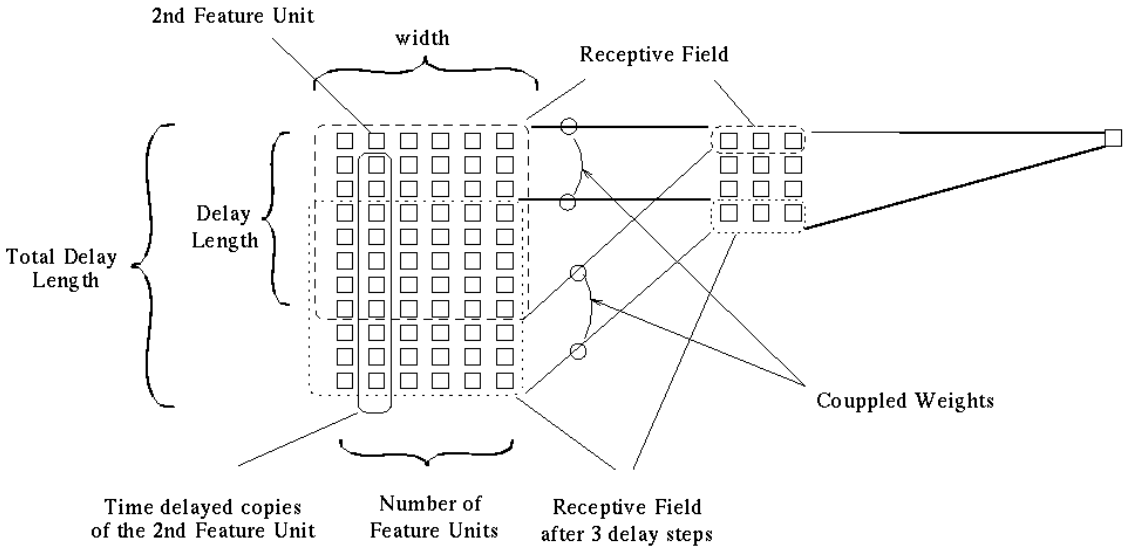
Redes para tratamiento temporal

- Time Delay Neural Network (TDNN)

  Invención independiente de: Long, Hinton, 1988; Waibel, 1989.

  - Similar al Backpropagation con enlaces y neuronas replicados en el tiempo
  - Desarrolla representaciones internas invariantes al desplazamiento
  - Utilizada inicialmente para reconocimiento de palabras a partir de espectrogramas
  - Arquitectura:

    Basada en líneas de delay o buffers



Redes para tratamiento temporal

- Modelo de Neurona (3º):
  - Función de suma:

$$net_j = b_j + \sum_i x_i w_{ij}$$

  - Función de activación: sigmoide

$$f_{act}(net) = \frac{1}{1 + \exp(-net)}$$

- Sincronía
  - Según la red de retardo temporal mas orden topológico
- Regla de aprendizaje:
  - Supervizado mediante Corrección de Error
  - Ajuste de pesos por Retropropagación del Error (similar al backpropagation)
  - Especial atención a conexiones acopladas:
    - Las correccciónes de pesos son promediadas para cada grupo éstas

Redes para tratamiento temporal

- Otros modelos
  - Temporal Back-propagation
    - Wan, 1990
  - Redes recurrente
    - Back-propagation Trough the Time
      Werbos, 1974
    - Real Time Recurrent Learning
      Zipser, 1989; Mc Bride, Neardendra, 1965

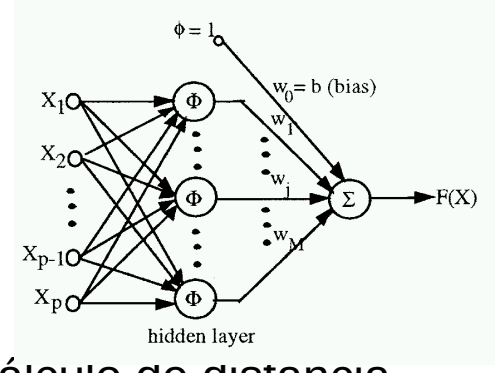
Bibliografía

- *Neural Networks. A comprehensive foundation*. Haykin. Macmillan.1994.
- *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación*. J. A. Freeman, D.M. Sakapura. Addison-Wesley. 1993.
- *Temporal Pattern Processing.* DLiang Wang. The Handbook of B.T and N.N. 1995.

- Tema 3. Modelos
  - Redes de Funciones de Base Radial

Redes de Funciones de Base Radial (RBFN)

- Redes de Funciones de Base Radial
  - Moody, Darken: 1990, Poggio, Girosi: 1990
  - Buscan la simplicidad, generalidad y rapidez
  - Tipo feedforward con aprendizaje híbrido: no supervizado + supervizado
  - Arquitectura:
    - Dos capas: oculta y salida
    - Feedforward todas a todas
    - Capa oculta:
      - Función de red como cálculo de distancia
      - Función de activación de base radial
      - Conexiones de cada neurona *j* = vector prototipo μ_j
      - Respuesta localizada
      - Aprendizaje no supervizado
    - Capa de salida:
      - Función de activación lineal o sigmoide
      - Aprendizaje supervizado de corrección de error por descenso del gradiente (adaline o LSM)

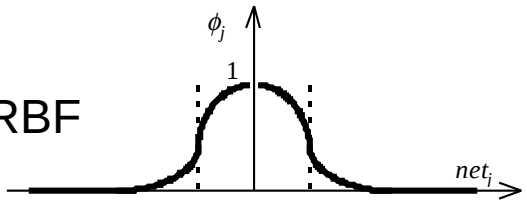


RBFN

- Neurodinámica capa oculta:
  - Función de distancia:

$$net_j = \left\| x - \mu_j \right\|$$

  - Función de activación: RBF



$$\phi_j = e^{-\frac{net_j^2}{2\sigma_j^2}} \qquad \phi_j = \sigma_j^2 net_j^2 \ln\left(net_j\right)$$

- Neurodinámica capa de salida:
  - Función de suma:

$$net_j = b_j + \sum_i \phi_i w_{ij}$$

  - Función de activación: identidad (o sigmoide)

$$f_{act}(net) = net$$

- Elementos a fijar
  - Número de neuronas ocultas
  - Inicialización/aprendizaje pesos μ_j

    Inicialización/aprendizaje parámetros σ_j

RBFN

Fijar número de neuronas ocultas

Crece exponencialmente con la dimensión de la capa de entrada

- Ha de contrastarse con el error alcanzable y la capacidad de generalización obtenida
- Existen métodos que comienzan con un número reducido y aumentan según necesidad

Inicialización/aprendizaje pesos μ

- Representativos de la densidad de probabilidad de entrada => no supervizado
- Métodos:
  - Subconjunto de datos de entrada: aleatorio o eliminando menos relevantes
  - Cuadrados mínimos ortogonales (orthogonal least squares)
  - Clustering: k-medias ó Kohonen
  - Modelos de mezclas gausianas

RBFN

- Inicialización/aprendizaje parámetros σ_j
  - Heurístico
  - Métodos:
    - Iguales y múltiplos del promedio de distancias entre los prototipos
    - En función de la distancia con sus α prototipos vecinos cercanos
    - En función del promedio de distancias a sus patrones representativos
- Aprendizaje supervizado
  - Principalmente para la capa de salida
  - Regla delta o backprop

$$e_j = d_j - f(net_j)$$

$$\Delta w_{ij} = \alpha \cdot e_j \cdot \phi_i$$

$$\Delta w_{ij} = \alpha \cdot f'_{act}(net_j) \cdot e_j \cdot \phi_i$$

RBFN

- Tipos de problemas tratables
  - Interpolación
  - Teoría de la regularización
  - Clasificación de patrones
- Características del las RBFN
  - Pueden dividir en 2 etapas el aprendizaje, utilizando mas datos en la etapa no supervisada
  - Solventan problemas no lineales
  - Aprendizaje mas rápido que backprop
  - Pueden haber problemas debido a dimensiones de entradas irrelevantes para las salidas
  - El número de neuronas ocultas necesarias puede llegar a ser muy alto
  - Fijar los parámetros de las neuronas ocultas puede ser dificultoso
  - Mayor lentitud en fase de ejecución

RBFN

- RBFN en SNNS
  - Tres diferentes funciones de activación para neuronas ocultas
  - Varias funciones de inicialización de pesos:
    - RBF_Weights
    - RBF_Weights_Redo
    - RBF_Weights_Kohonen
  - Incluye aprendizaje supervisado por gradiente descendiente también para los nodos de la capa oculta
- RBFN en ZISC
  - Dos simples funciones de activación para neuronas ocultas
  - Número de neuronas ocultas aumentan según patrones no cubiertos
  - Parámetros σ_j se reducen según conflictos en la clasificación
  - Pseudo-capa de salida con pesos 1 ó 0, neuronas OR y chequeo de conflictos
  - Limitaciones en número y tipo de entradas, salidas, número neuronas ocultas y representación de pesos

Bibliografía

- *Redes Neuronales y Sistemas Borrosos.* Martín, Sanz. Rama. 2001.
- *Neural Networks for Pattern Recognition.* Bishop. Oxford U. P. 1995.
- *Neural Networks. A comprehensive foundation*. Haykin. Macmillan.1994.
- *SNNS user manual ver. 4.1*. Zell. 1995.
- *ZISC ISA and PCI Cards*. IBM. 1998.

Tema 3. Modelos

- Redes Neuronales Modulares

RN Modulares

- Ventajas:
  - Construir módulos más simples y pequeños que una red monolítica equivalente
  - Debido a dichas modificaciones el aprendizaje de las tareas a realizar por cada módulo normalmente es mas sencillo que la tarea global
  - Probablemente los distintos entrenamientos se puedan realizar de forma independiente y/o en paralelo
- Usualmente se ven las RN como cajas negras, donde todo va bien mientras no ocurran problemas
- Cuando crecen o trabajamos con aplicaciones más difíciles, pueden surgir diferentes problemas

RN Modulares

- Problemas que conducen a la necesidad de usar sistemas modulares:
  - Reducir la complejidad del modelo:

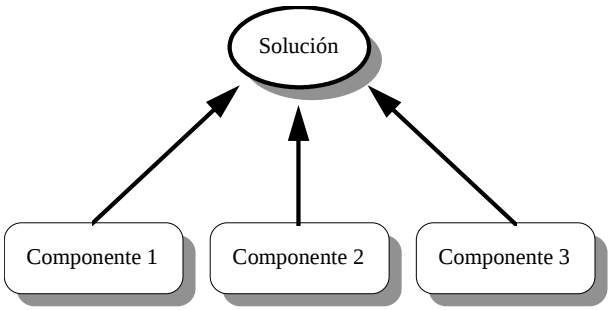
    La complejidad de una red neuronal monolítica usualmente se incrementa drásticamente con el tamaño o la dificultad de la tarea a resolver. Los sistemas modulares nos permiten mantener dicha complejidad proporcional a la tarea
  - Incorporar conocimiento:
    - Es posible incorporar a la arquitectura del sistema conocimiento a priori cuando se comprende ituitiva o matemáticamente la descomposición del problema a tratar. Dicho conocimiento actúa como una restricción en el aprendizaje del problema y permite mejor eficacia en la generalización
  - Fusión de datos y predicciones promediadas:
    - Los sistemas modulares permite tener en cuenta datos provenientes de diferentes fuentes y con distintas naturalezas, combinando o promediando la salida de varios módulos puede conducirnos a mejorar la robustez y mejorar la generalización
  - Sistemas Híbridos:
    - Los sistemas heterogéneos permite combinar diferentes técnicas, así es común utilizar módulos neuronales junto a módulos basados en procesamiento simbólico
  - Aprendiendo diferentes tareas simultáneamente:
    - Los módulos especializados pueden ser compartidos entre varios sistemas entrenados para llevar a cabo diferentes tareas
  - Robustez e incrementabilidad:
    - Los sistemas cooperativos funcionan mejor y mas robustamente que los basados en una única etapa, y además son fáciles de entender, modificar y extender

RN Modulares

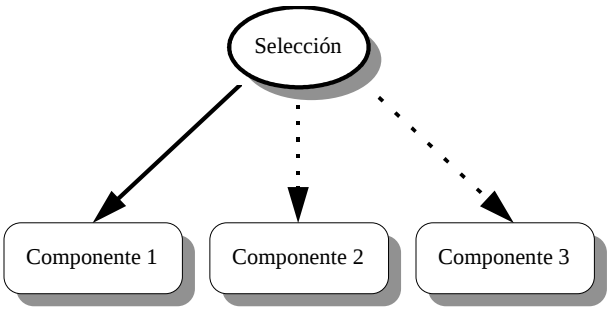
- Metas perseguidas
  - Examinar e investigar las posibilidades de descomponer el aprendizaje en RN
  - Mostrar que la descomposición del aprendizaje ayuda a incrementar la eficiencia de la RN para algunas clases de tareas.
  - Dar medios para soportar la visión ingenieril de la modularización
- Métodos de Creación de Módulos
  - Descomposición Explícita
  - Descomposición de Clases
  - Descomposición Automática
  - Fusión multisensorial
  - Módulos definidos naturalmente
  - Modularizar el aprendizaje
  - Modularizar la estructura

RN Modulares

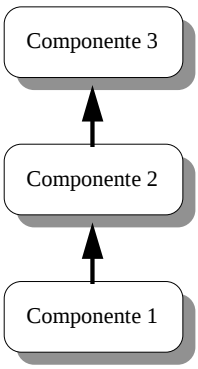
- Métodos de Combinación de Módulos
  - Combinación Cooperativa (a)
  - Combinación Competitiva (b)
  - Combinación Secuencial (c)
  - Combinación Supervisada (d)



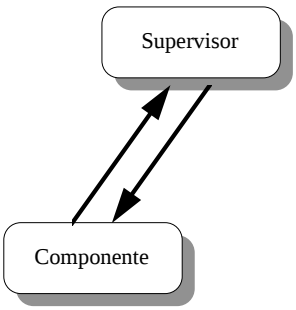
(a)



(b)



(c)

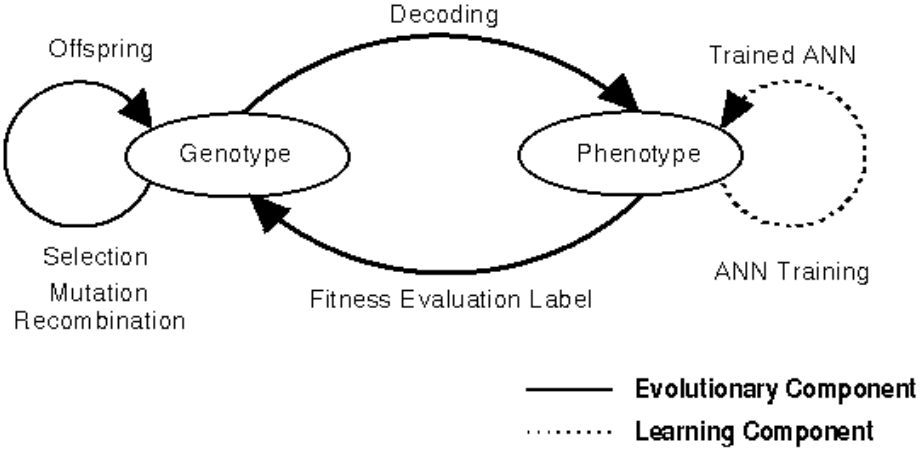


(d)

RN Modulares

- Descomposición de la tarea de la aplicación en subtareas de aprendizaje independiente:
  - Algoritmos genéticos (AG) y RNs

    Mediante el uso de AG se pude diseñar (optimizar) tanto la arquitectura como los pesos y parámetros más adecuados
    - Los AG son uno de los métodos más potentes para resolver problemas de optimización
    - Los AG son métodos multipunto: parten de una población (conjunto de RNs) en su búsqueda
    - El criterio a optimizar es la eficacia (fitness) individual de la red
    - A partir de la población inicial, mediante una serie de operaciones se genera la descendencia (offsprings)
    - Entre las operaciones más usuales tenemos:
      - Mutación:
        - Descendiente igual al padre, pero con pequeñas variaciones.
      - Recombinación (crossover):
        - Propiedades proveniente de 2 (ó más) padres
        - Selección de padres aleatoria, prefiriendose los mejores

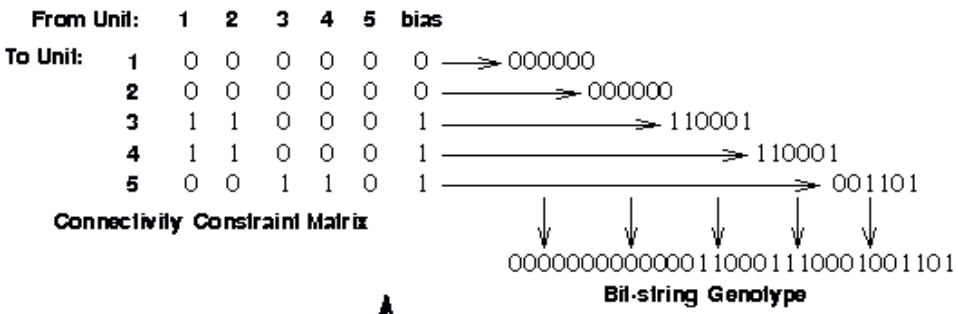


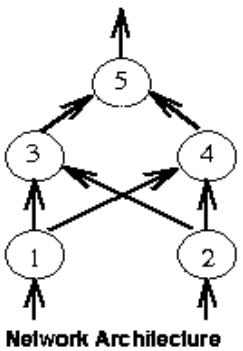
RN Modulares

Para poder aplicar AG a un problema es necesario:

- Poder representar las soluciones como strings
- Poder calcular la eficacia (fitness) de las soluciones

- En las RN, se plantean por tanto dos tipos de problemas de optimización:
  - Topología o Arquitectura de red (problema discreto):
    - Nº de unidades ocultas
    - Estructura de interconexionado
  - Ajuste de parámetros (problema continuo):
    - Parámetros de aprendizaje
    - Pesos





RN Modulares

- ENZO
  - Es un sistema evolutivo de optimización de RN que se utiliza en conjunción con SNNS (Stutgart Neural Network Simulator)
  - ENZO se encarga de optimizar únicamente la topología de la red, dejando el resto del trabajo al SNNS o al diseñador
  - Actualmente sólo trabaja con redes que utilicen el algoritmo de descenso de gradiente rápido (RPROP)
  - Tener en cuenta que:
    - No optimiza los parámetros del aprendizaje
    - No realiza aprendizaje de pesos
    - Sólo utiliza redes tipo RPROP

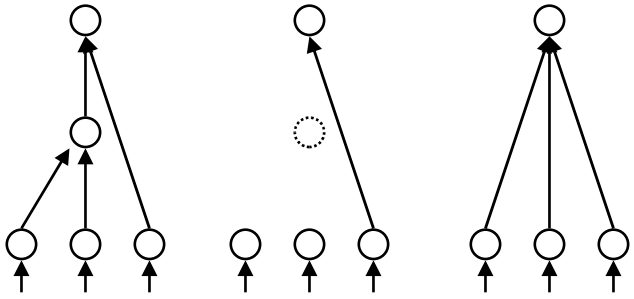
RN Modulares

- El ciclo evolutivo se compondrá de los siguientes pasos:
  - Preevolución:
    - Usando las especificaciones del usuario se genera una población de redes con una determinada densidad de conexionado
  - Ciclo:
    - Selección de un padre, prefiriendo los mejores
    - Generar un descendiente tras mutar un duplicado del padre
    - Si se elige recombinación, añadir aleatoriamente uno o varios nodos ocultos del segundo padre
    - Entrenar el nuevo miembro:
      - Usar decaimiento de pesos
      - Podar pesos inútiles y reentrenar si procede
    - Evaluar la eficacia del nuevo miembro:
      - Considerar criterios de diseño más importantes
    - Insertar en la población, eliminando al peor
  - Postevolución

RN Modulares

- Mutaciones:
  - Links:
    - Borrado de links (soft pruning): Opción: Más probable el de menor peso ($p_{del} \cdot N_{\theta}(0,w)$)
    - Añadido de links: Igua probabilidad (p_{add})
    - No se evalúan directamente los efectos de pruning, lo hará la propia evolución
  - Borrado de Unidades:
    - Mayor impacto en la red
    - Dos heurísticas: PWU y Algoritmo Bypass
    - PWU: Preferir unidades débiles
      - Ordena unidades según conectividad relativa (conexiones activas / conexiones máximas)
      - Borra preferiblemente las débilmente conectadas
      - Complementa al soft-pruning
    - Algoritmo Bypass:
      - Aproxima la función no lineal implementada por una lineal
      - Aumenta la proporción de éxitos

RN Modulares

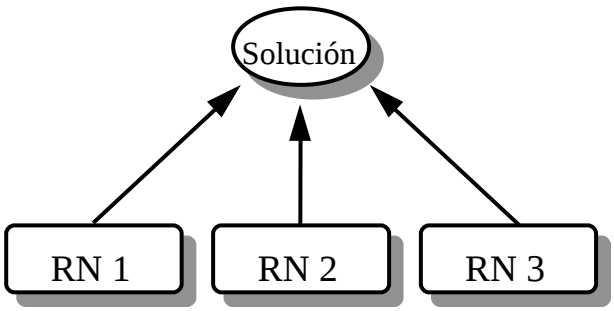


- Conclusiones:
  - Al utilizar 2 técnicas de búsqueda:
    - Descendiente rápido del gradiente, para optimización local de pesos
    - Evolución, para optimización global de la topología
  - evita quedar atrapado en mínimos locales.
  - Acelera el aprendizaje y la calidad de los descendientes mejora el promedio de su topología
  - Trata de eliminar las unidades de entradas redundantes

RN Modulares

Ensembles Neuronales

- Combinan un conjunto de RN redundantes
- Cada RN da una solución a la misma tarea
- Basados en el proceso de consulta a varios “expertos” para solucionar un problema
- Intentan mejorar la generalización, salvaguardándose de los fallos de una RN individual
- La eficacia que obtienen es más producto de reducir la varianza que el bias
- Los ensembles no garantizan una eficacia superior a la mejor de sus RN individuales, pero reducen el riesgo de hacer selecciones particularmente pobres
- Las RNs miembros del ensemble debe ser eficaces individualmente y diversas entre ellas
- Un factor clave es seleccionar/producir RN diversas



RN Modulares

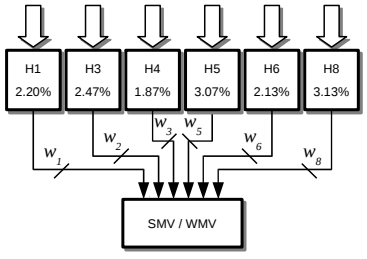
- Diversidad en Ensembles:
  - Un grupo de RNs se consideran más diversas si cuanto más diferentes sean los errores que cometen
  - Sharkey & Sahrkey (1995) proponen 4 niveles:
    - Nivel 1: No hay errores coincidentes
    - Nivel 2: Hay errores coincidentes, pero la mayoría siempre es correcta
    - Nivel 3: La mayoría no siempre es correcta, pero la salida correcta es producida siempre al menos por una RN
    - Nivel 4: La mayoría no siempre es correcta y hay algunas entradas con errores en todas las RN
  - Medidas de diversidad:
    - Para valorar cuan diverso es un grupo de RNs se han propuesto distintas métricas
      - Correlación entre las salidas de dos RNs
      - Estadístico Q
      - Desacuerdo y doble fallo
      - Varianza de Kohavi-Wolpert
      - Medida de dificultad

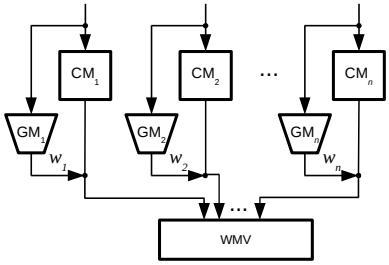
RN Modulares

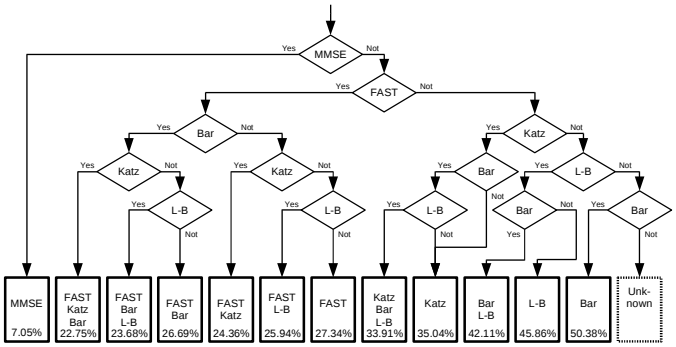
- Métodos de Creación de componentes en Ensembles:
  - Variar el conjunto inicial de pesos aleatorios
  - Variar la topología de la RN
  - Variar el algoritmo de entrenamiento
  - Variar los datos (más común):
    - Muestreo de datos distinto por RN
      - Ej.: Boostrap, *k*-fold
    - Conjuntos de entrenamietos disjuntos
    - Boosting y Remuestreo Adaptativo
    - Fuentes de datos diferentes
    - Variar el preproceso de cada RN

RN Modulares

- Métodos de Combinación de componentes en Ensembles:
  - Promediado y promediado pesado
  - Métodos de combinación no-lineales:
    - Ej.: Voto, basados en rankings, Dempster-Shafer
  - Supra Bayesiano:
    - Salidas de RN son también datos, estudiar su distribución
  - Stacked Generalization:
    - RN no lineal aprende a combinar las RN componentes
    - Ej.: Gating Network







RN Modulares

- Modelos de Ensembles (I):
  - Bagging:
    - Usa bootstrap para obtener diferentes conjuntos de entrenamiento
    - Combinación mediante voto mayoritario
    - Intuitivo y simple con buena eficacia
  - Boosting:
    - Usa bootstrap para obtener diferentes conjuntos de entrenamiento
    - Crea tres Clasificadores:
      - C1: entrenado con datos muestreados aleatoriamente
      - C2: entrenado con datos cuya mitad fue bien clasificada por C1 y otra mitad mal clasificada por C1
      - C3: entrenado con datos en los que C1 y C2 están en desacuerdo
    - Combinación mediante voto mayoritario de tres vías
    - Uno de los más importantes desarrollos de la historia reciente del machine learning

RN Modulares

- Modelos de Ensembles (II):
  - AdaBoost:
    - Versión más general del Boosting
  - Stacked Generalization:
    - Clasificador de segundo nivel que aprende a combinar las salidas de las RNs componentes
  - Mezcla de Expertos:
    - Las salidas de cada RN son ponderadas por una Gating Network, entrenada a partir de las entradas
    - Las RN componentes también intentan minimizar el error global, diversificándose entre sí
  - Negative Correlation Learning:
    - Las RN componentes incorporan un término de penalización de la correlación en su función del error a minimizar
    - Esto fomenta que las distintas RNs se especialicen en diferentes partes o aspectos del conjunto de entrenamiento
    - Utiliza promediado para combinar las salidas de las RNs

RN Modulares

Bibliografía:

- [Hrycej, 1992] Hrycej, T. (1992) Modular Learning in Neural Networks. John Wiley and Sons, New York, NY.
- [Sharkey, 1999] Sharkey, A.J.C. (1999) Multi-Net Systems. Combining Artificial Neural Nets, Springer-Verlag, pp: 1-30.
- [Gallinari, 1995] Gallinari, P. (1995) Training of Modular Neural Net Systems. In The Handbook of Brain Theory and NeuralNetwork.

- Tema 4. Neuro-Software

Neuro-Software

- Marcos formales
  - Engloban las características de los modelos que se pueden describir
  - Modelos específicos
    - Íntimamente ligados a un algoritmo concreto
    - Gran escalabilidad
  - Modelos tradicional
    - Inducidos por los modelos tradicionales de neuronas
    - Puede tener uno o varios niveles de descripción
    - Niveles típicos:
      - Red, capa, neurona y sinápsis
    - Estilo de aprendizaje prefijado y sin niveles

Neuro-Software

- Marcos Jerárquicos
  - Describen las redes mediante varios niveles no prefijados que forman una jerarquía de niveles
  - La propiedad característica principal es la *recursividad*

  Formalismo MIND

  Extensión de la teoría de Procesos Secuenciales Comunicados (CSP)
  - Totalmente orientados a una implementación específica
- Notación Leslie S. Smith
  - Técnica general de especificación de "arriba a abajo"
  - Cada nivel define un espacio de nombres, generadores, conexiones y espacio de estados
  - Notación demasiado compleja y abstracta para ser ejecutable

Neuro-Software

Programación directa

Aspectos a tener en cuenta:

- Elección del compilador
- Operaciones de Matrices y Vectores
- Almacenamiento de la información
  - Ficheros: ascii / binarios
  - Memoria: dinámica vs. estática, flotantes vs. enteros

Librerías

- Añaden a muchos lenguajes tradicionales facilidades para la implementación de RN
- Pueden ser de propósito específico o de propósito general

- Facilidades:
  - Operaciones matriciales
  - Funciones de Activación
  - Algoritmos de aprendizaje
  - Funciones de Visualización
  - Funciones de Análisis de Redes
- Ej.: Simulink-Mathlab/Neural Network/SOM Toolbox

Neuro-Software

- Entornos de desarrollo
  - Posibilidades:
    - Lenguaje y/o herramientas para modelar sistemas
    - Capacidad de especificar dinámicas
    - Ejecución de experimentos
    - Adquisición de datos
    - Transmisión de señales de salida
    - Análisis de resultados
    - Visualización de resultados
    - Incorporación en aplicaciones
  - Características deseables:
    - Facilidad de uso
    - Potencia
    - Eficiente
    - Extensibilidad
  - Generalmente se basan en un lenguaje de modelado de redes, para definir las redes y/o los algoritmos de aprendizaje

Neuro-Software

- Ejemplos de Neurosimuladores
  - Para novatos
    - Cortex, Cortex III: Asociado al libro "An Introduction to Neural Computing"
    - PDP++ Simulator: Asociado al libro "Explorations in Parallel Distributed Processing", basado en OOP (C++)
  - Para educación
    - SNNS
    - Statistica NN (STNN)
    - MIRRORS/II integrado con NSL(Neural Simulator Language)
    - Aspirin/MIGRAINANES: fichero de especificaciones de red => código C, datos =>GNUPLOT
    - Neuralogic Windows: simulado de alta velocidad para distintas plataformas
    - JOONE: Java, Núcleo y GUI, entorno distribuido de aprendizaje
  - Modelado de Neuronas Individuales
    - NEURON
    - GENESIS (GEneral NEural SImulation System)
    - BIOSIM

Neuro-Software

- Negocios e Industria
  - NeuralWorks Professional II/Plus
    - Gran rango de paradigmas
    - Intercambio de datos con otros programas
    - Extensible mediante un lenguaje especial o rutinas C
    - Posiblidad de generar código C
  - HNC NeuroSoft
    - Basado en lenguaje AXON
    - Utilizable con las neurotarjetas de la casa HNC
  - Explorer 3000
  - NeuroSolutions
    - Múltiples arquitectura
    - Wizards
    - Análisis de redes
    - Optimización genética de parámetros
    - Implantable en aplicaciones: Código C++, DLL, OLE

Neuro-Software

- Futuros Neurosimuladores
  - GALATEA
    - Proyecto ESPRIT
    - Basado en lenguaje N (ext. C++).
    - Mapeo a distintos sistemas paralelos
    - Incluira un compilador a Silicio
  - MetaNet
    - Mediante POINTs (Problem Oriented Interaction Tools)
    - El usuario podrá definir incluso la forma de sus visualizaciones
  - SESAME
    - Gernman National Research Center of Computer Science
    - Bloques segun diferentes clases: matemáticos, utilidades, graficos...
    - Autoconfiguración de parámetros via Satisfacción de Restricciones.

Bibliografía

- *Descripción Formal de Modelos de Redes Neuronales.* J.R. Álvarez. IX Cursos de Verano de la UNED.1998.
- *Introducción a los Entornos de Simulación y Desarrollo de Redes Neuronales*. A. Manjarrés. IX Cursos de Verano de la UNED.1998.
- *Neurosimulators*. J.M.J. Murre. Arbid Ed. Mit-Press.1995.
- *NSL: Neural Simulation Language*. A. Weitzenfeld. Arbid Ed. Mit-Press.1995.
- *Neural Network PC Tools. A Practical Guide.* Eberhart and Dobbins. Academic Press. 1990.

- Tema 4. Neuro-Hardware

Neuro-Hardware

- Hardware utilizable
  - VLSI analógico
  - Opto-Electrónicos
  - Neuro-Chips (VLSI Digital)
  - Neuro-Tarjetas
  - Neuro-Computadores
  - Máquinas paralelas de propósito general
- Objetivo:Acelerar fases de aprendizaje y ejecución
- Características deseables
  - Escalable
  - Programable
  - Modular
  - Posiblidad de implementar redes virtuales
- Niveles de paralelismo de redes
  - De red
  - De entrenamiento de patrones
  - De nivel o capa
  - De neurona
  - De sinapsis

Neuro-Hardware

- Analog VLSI
  - Mejor velocidad
  - Precisión baja (0.1 a 5%)
  - Hacen uso de la tolerancia a fallos de las RN
  - Neuronas de dimensiones similares o inferiores a las naturales
  - Bajo consumo
  - Limitado número de ramificaciones a sinapsis
  - Ej.:
    - Aplicaciones en el modelado de fenómenos neurofisiológicos (Sílicon Neuron, Silicon retina, ...)
    - Boahen, 46 neuronas heteroasociativas
    - Linares, memorias adaptables autoasociativas
    - ETANN de Intel

Neuro-Hardware

- Opto-Electónicos
  - Aprovechan la gran capacidad de esta tecnología para las interconexiones
- Digital VLSI
  - Mas costosos que los analógicos
  - Mas fácil de diseñar, ya que no hay que cablear el algoritmo de entrenamiento
  - Mayor flexibilidad
  - Mayor precisión
  - Basado en matrices de DSPs ó microprocesadores
    - RAP (Ring Array Processor) (4x10 TMS32C30 DSPs) (10^7 conx./sg.)
    - Adaptive Solutions CNAPS (Connected Node Architecture for Parallel Systems) (64 Processor Nodes (PN) en SIMD) (300x10^6 conx./sg.)
    - Intel/Nestor Ni1000 (RBF) (512 Proc. x 2 Neur.) (256 x 1024 x 64) (10 bill. conx./sg.)
    - ZISC36 de IBM, 36 neuronas RBF, >250K salidas/sg.
    - TOTEM (32 Neur., 1000x10^6 oper/sg). Tarjetas de 2 procesadores.
    - SAND/1 (4 mult. paralelos) 200x10^6 conex/sg. Tarjetas de hasta 4 procesadores.

Neuro-Hardware

- Neuro-Tarjetas
  - Utilizando los Neuro-Chips
    - ZISC PCI, hasta 19 ZISC36
  - Utilizando coprocesador(es) o DSPs
    - HNC Balboa, Neuromatrix
  - Utilizando FPGAs
- Neuro-Computadores
  - Configuración:
    - Ordenador Host, Unidad Interface, Red de Interconexión, Array de elementos de Proceso
  - Ej.:
    - SYNAPSE-1 (Siemens) (Analisis de cálculos mas frecuentes)
    - MindShape (Univ. de Leden (Hol.)) (PE de proposito especial)
    - BSP-400 (Univ. de Leden (Hol.)) 400 = 25 x 16
    - GPPN (Paralelismo a varios niveles) (DSPs TMS-320C25 , conv. A/D, D/A)
- Máquinas paralelas de propósito general
  - Ej.: CM-5 (SIMD/MIMD)

Bibliografía

- *Arquitecturas avanzadas para IA y Redes Neuronales.* M. Rincón. IX Cursos de Verano de la UNED.1998.
- *Analog VLSI for Neural Networks*. R.W. Newcomb, J.D. Lohn. Arbid Ed. Mit-Press.1995.
- *Digital VLSI for Neural Networks.* D. Hammerstrom. Arbid Ed. Mit-Press.1995.
- *Neural Networks. A comprehensive foundation*. Haykin. Macmillan.1994.
- *Neural Network PC Tools. A Practical Guide.* Eberhart and Dobbins. Academic Press. 1990.
- ZISC036 Neurons User's Manual. IBM France. 1998.