Sistemas de Aprendizaje Automático

Conforme a contenidos del «Curso de Especialización en Inteligencia Artificial y Big Data»

Universidad de Castilla-La Mancha

Escuela Superior de Informática
Ciudad Real

Capítulo 8

8. Introducción a las redes neuronales artificiales.

José Angel Olivas Varela

En este capítulo se introducen los conceptos básicos de las redes neuronales artificiales, se describen sus arquitecturas y tipos de funciones de activación, desarrollando con más profundidad el Perceptrón multicapa y los mapas autoorganizados, en particular los Mapas de Kohonen como muestras más representativas del aprendizaje supervisado y no supervisado respectivamente.

8.1. ¿Qué es una red neuronal?

Las neuronas son los constituyentes estructurales del cerebro. Al cerebro siempre se le ha asociado tremenda capacidad de cálculo, y se puede definir mediante el símil de que es un ordenador muy complejo, no lineal y paralelo. Sus cualidades las alcanza mediante una sofisticada red de neuronas con interacciones sinápticas.

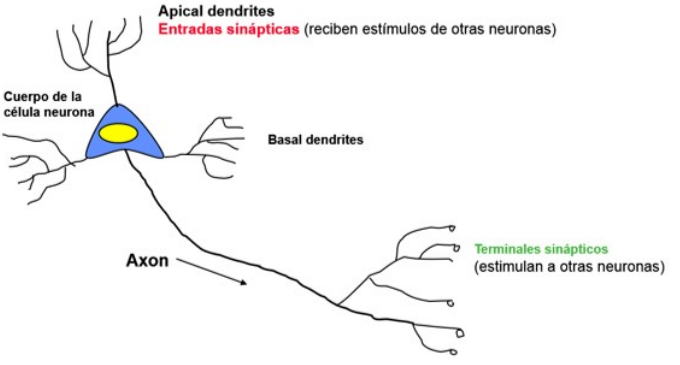

Figura 8.1: Representación de una neurona humana.

Una red neuronal artificial es un algoritmo matemático con capacidad para recordar experiencias y hacerlas disponibles para su uso. Recuerda al cerebro humano en dos aspectos:

- El conocimiento es adquirido por la red a través de un proceso de aprendizaje.
- La fuerza de la conexión entre neuronas (pesos sinápticos) es usada para almacenar el conocimiento.

Una red neuronal aprende mediante la modificación de sus pesos sinápticos. Se puede modificar también la topología. Otras formas de llamar a una red neuronal: sistemas conexionistas, procesadores distribuidos paralelos o neurocomputadores.

Un poco de historia de las RNA.

- **1943** primer trabajo de McCulloch y Pitts. MacCulloch Psiquiatra y neuroanatomista y Pitts un matemático prodigio describieron un cálculo lógico de redes neuronales y Von Neumann usó elementos neurales ideales a base de elementos discretos switch-delayed para el desarrollo del EDVAC que resultó en el ENIAC.
- **1948** publicación del libro Cybernetics de Wiener.
- **1949** publicación del libro "The Organization of Behaviour" de Hebb (La conectividad del cerebro está continuamente cambiando). El libro de Hebb fue fuente de inspiración para el desarrollo de modelos de aprendizaje y de sistemas adaptativos
- **1952** publicación del libro de Ashby: "Design for a Brain: The Origin of Adaptive Behaviour".
- **1954** tesis doctoral de Minsky "Theory of Neural-Analog Reinforcement Systems and Its Applications to theBrain-Model Problem".
- **1958** publicación de Perceptron trabajo de Rosenblatt y del teorema de convergencia del perceptrón.
- **1960** Widrow y Hoff introducen el algoritmo de mínimos cuadrados para formular el Adaline.
- **Años 70**, son años de silencio. Motivos tecnológicos y el- artículo muy crítico de Minsky y Papert (1969).
- **Años 80**, resurgimiento. Hay trabajos en varios frentes: Grossberg, Hopfield (1982), Kohonen, en 1986 el algoritmo de back-propagation de Rumelhart, Hinton y Williams y en 1988 las Radial Basis Functions de Broomhead y Lowe.

Ventajas de las RNA.

- Modelan relaciones no lineales.
- Modelan relaciones entrada-salida.

- Capacidad de adaptación.
- Tiene en cuenta el contexto de trabajo.
- Posibilidad de desarrollo de dispositivos VLSI.
- Uniformidad de análisis y diseño.
- Analogía neurobiológica.

Modelos de neuronas artificiales

Una neurona artificial es la unidad de procesado básica de una red neuronal artificial. Sus elementos básicos son:

- Sinapsis o conexiones cada una de ellas con un peso.
- Un sumador capaz de sumar entradas pesadas.
- Una función de activación que limita la amplitud de la salida.

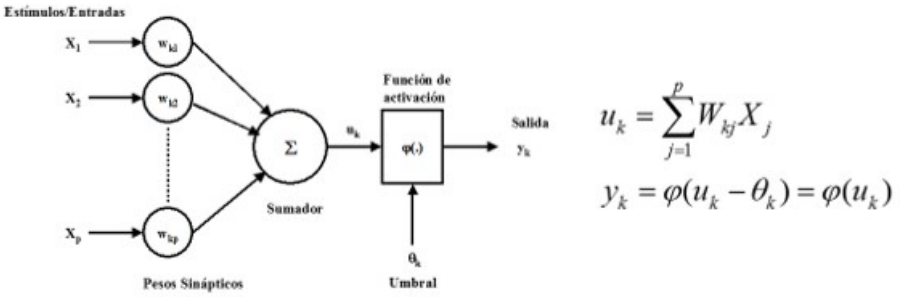

Figura 8.2: Modelo básico de neurona artificial.

8.2. Función de activación.

La función de activación es uno de los principales elementos de una neurona artificial porque limita la amplitud de la salida de cada neurona a [0,1]. Los tipos de función de activación más usados son los siguientes:

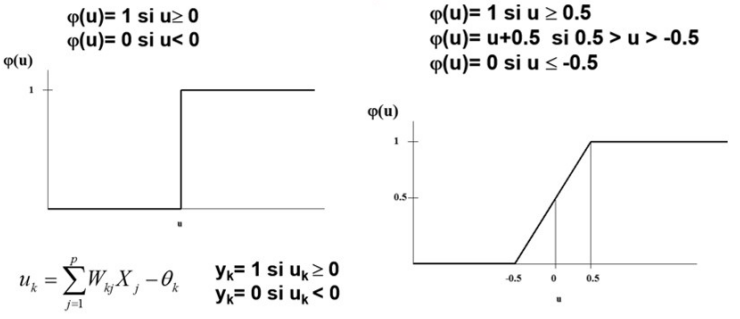

Figura 8.3: Funciones de activación de tipo UMBRAL y RAMPA.

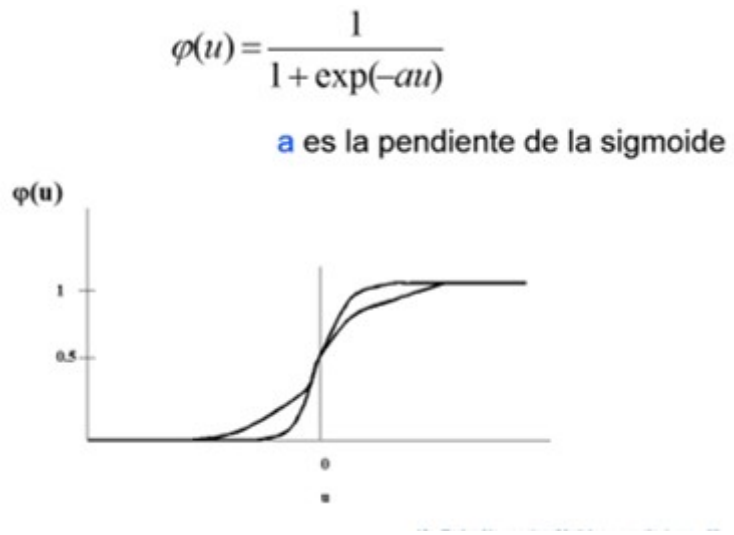

Figura 8.4: Función de activación de tipo SIGMOIDAL.

También son muy usadas las funciones de activación de tipo TANGENTE HIPERBÓLICA:

$$\varphi(u) = \tanh\left(\frac{u}{2}\right) = \frac{1-\exp(-u)}{1+\exp(-u)}$$

Y las de tipo SIGNO:

$$\varphi(u) = 1 si u > 0$$

$$\varphi(u) = 0 si u > 0$$

$$\varphi(u) = -1 si u > 0$$

8.3. Arquitecturas de RNA.

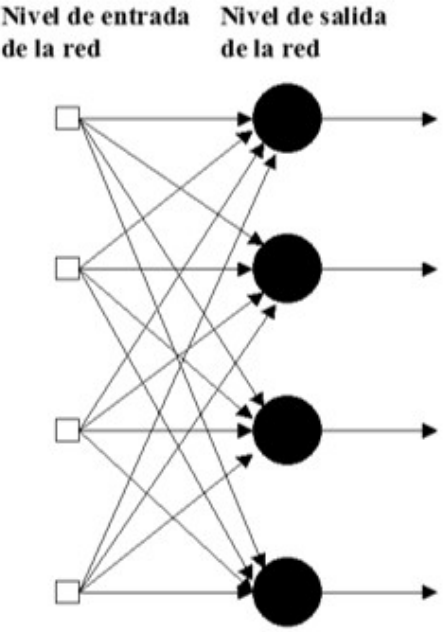

Figura 8.5: Red Feedforward de una sola capa.

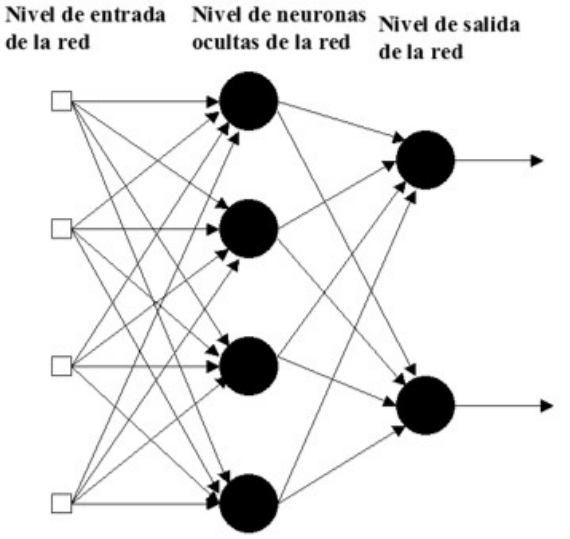

Figura 8.6: Red Feedforward multicapa completamente conectada.

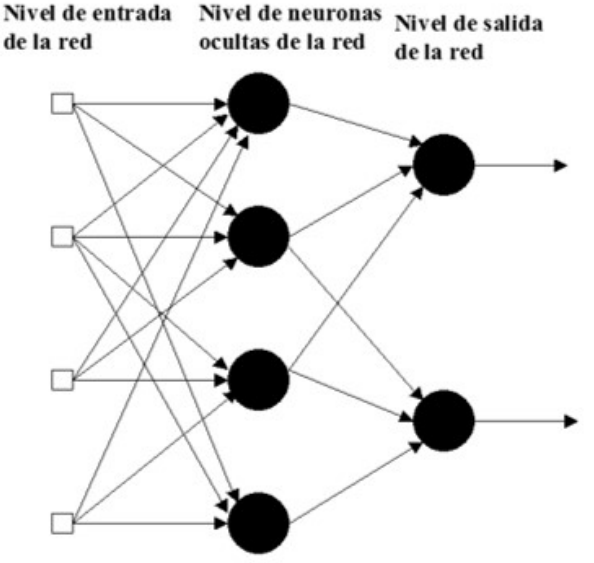

Figura 8.7: Red Feedforward multicapa parcialmente conectada.

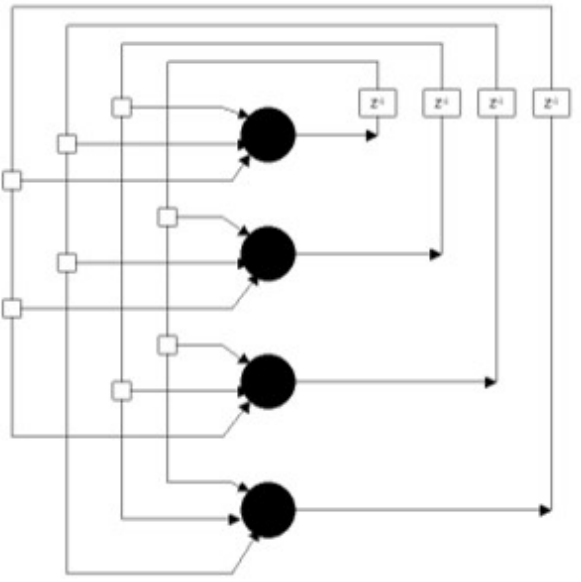

Figura 8.8: Redes recurrentes (completamente recurrentes).

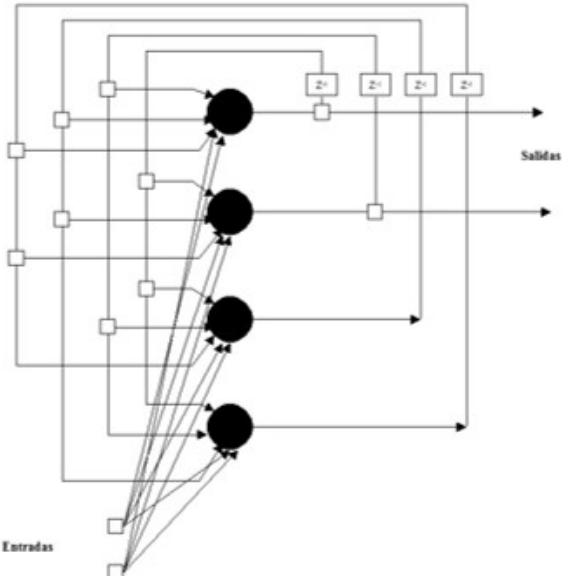

Figura 8.9: Redes recurrentes (parcialmente recurrentes).

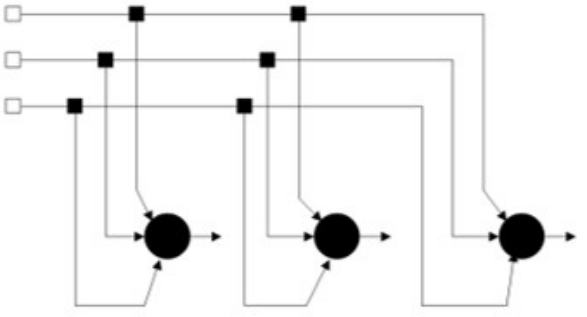

Figura 8.10: Estructura Lattice 3x1.

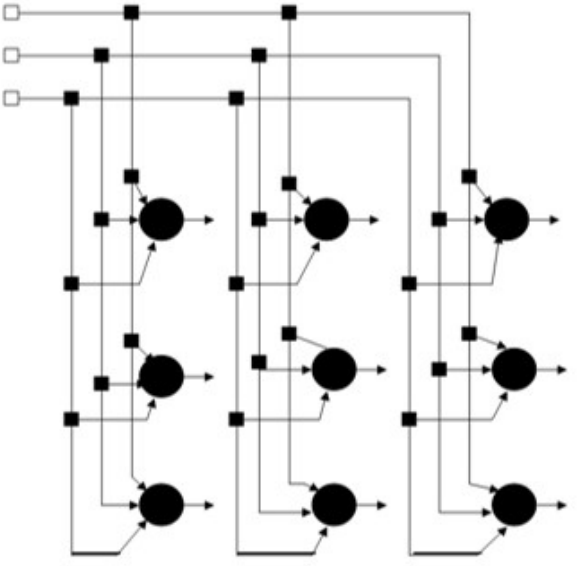

Figura 8.11: Estructura Lattice 3x3.

8.4. Aprendizaje de las RNA.

Esencialmente, el proceso de aprendizaje de una red neuronal artificial consta de dos etapas claramente diferenciadas:

- **Factores de certeza**, los describimos con detalle a continuación.
- **1ª etapa: entrenamiento.** Proceso por el que los pesos de las conexiones sinápticas de las neuronas son adaptados a través de una continua estimulación del entorno. El tipo de aprendizaje viene determinado por la manera enque los cambios en los parámetros tienen lugar. Secuencia de sucesos en aprendizaje:

- Estimulación de la red neuronal por el entorno.
- Modificación de la fuerza de las conexiones.
- Respuesta de la red en esta nueva situación.

- **2ª etapa: validación.**

  Una taxonomía del proceso de aprendizaje podría ser la siguiente:

- **Algoritmos de aprendizaje**
  - Aprendizaje por **corrección de error.**
  - Aprendizaje de Bolzmann.
  - Aprendizaje Hebbiano.
  - Aprendizaje **competitivo.**
- **Paradigmas de aprendizaje**
  - Aprendizaje **supervisado.**
  - Aprendizaje **no supervisado.**
  - Aprendizaje por refuerzo.

En este capítulo nos centraremos en los algoritmos más usados que, en cuanto a los algoritmos de aprendizaje SUPERVISADO son los de corrección del error. Como ya hemos visto, las principales características de este tipo de aprendizaje es que debe existir un maestro, es decir, el conjunto de ejemplos debe ser de entrada-salida (deben estar etiquetados). El conocimiento del maestro es transferido a la red y los parámetros de la red se ajustan por una influencia combinada del vector de entrenamiento y la señal de error. Los principales algoritmos de este tipo son:

- Least-mean square (LMS).
- **Backpropagation.**

En lo que concierne al aprendizaje NO SUPERVISADO es el que se suele denominar auto-organizado (self-organized). No hay maestro externo y una medida independiente de la tarea, define la calidad de la representación o ajuste a las regularidades estadísticas de los datos de entrada que se le pide a la red y de acuerdo a esa medida los parámetros de la red se van ajustando. El algoritmo más representativo de los de este tipo son los Mapas autoorganizados, Mapas de Kohonen (Self Organized Maps, SOM -toolbox de Matlab-).

Actualmente es frecuente hablar de aprendizaje SEMISUPERVISADO debido a que la mayoría de las bases de datos de las que se puede disponer como conjunto de entrenamiento no están etiquetadas inicialmente, por ejemplo, una base de datos de imágenes, lo que impide usar algoritmos de aprendizaje supervisado, que suelen ser más indicados para este tipo de problemas de clasificación. Lo que se hace es etiquetar un subconjunto de los datos disponibles y a partir de ellos ejecutar los algoritmos.

Aprendizaje por corrección de error.

El objetivo de este método de aprendizaje es minimizar una función de coste basada en la función de la señal de error ek(n) de forma que la respuesta de cada neurona de salida en la red se aproxime a la respuesta que debería dar esta neurona en algún sentido estadístico.

$$e_k(n) = d_k(n) - y_k(n)$$

$d_k(n)$ respuesta deseada de la neurona de salida k en el instante n.

$y_k(n)$ respuesta medida de cada neurona de salida ante el vector de estímulos x(n).

$e_k(n)$ error de cada neurona de la capa de salida.

$$J = E[\frac{1}{2}\Sigma_k e_k^2(n)]$$

Se suele usar el criterio del error cuadrático medio, donde: E es la esperanza y se extiende a todas las neuronas de la capa de salida. La minimización de la función de coste J con respecto a los parámetros de la red da lugar al método del descenso del gradiente. El problema de este método es que necesita conocer las características estadísticas de E. Una solución aproximada al problema de optimización es usar el valor instantáneo de la suma en vez del valor esperado. La red es optimizada minimizando (n) con respecto a los pesos sinápticos. Esto da lugar a la regla de aprendizaje por corrección de error o regla delta (Widrow y Hoff, 1960), donde es la tasa o factor de aprendizaje:

$$\Delta W_{kj}(n) = \eta e_k(n) X_j(n)$$

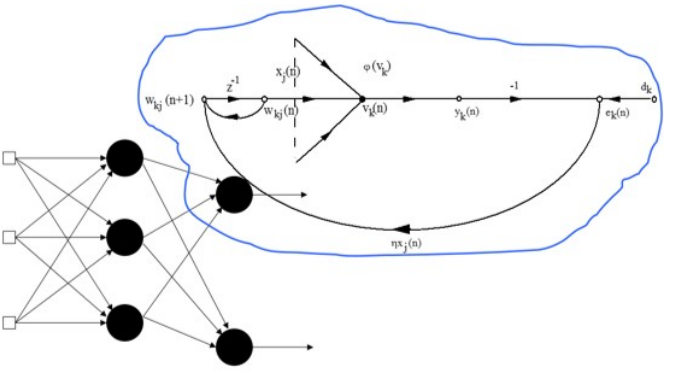

Figura 8.12: Criterio del error cuadrático medio.

La importancia de la tasa o factor de aprendizaje radica en que su valor asegura la estabilidad del proceso. Valor pequeño, proceso de aprendizaje lento y valor grande, se acelera el proceso con riesgo de converger. Es típico disponer de la superficie de error (J en función de pesos) y el algoritmo de corrección de error recorre la superficie de J en función de los pesos hasta encontrar un mínimo global.

Aprendizaje competitivo.

Las neuronas de salida compiten entre ellas por conseguir su activación. Sólo una neurona se activa cada vez (buena característica para ser usada como clasificador). Los elementos básicos de una red de aprendizaje competitivo son:

- Un conjunto de neuronas iguales en todo excepto en la distribución de sus pesos que hacen que respondan de formas diferentes a un conjunto de modelos de entrada.
- Un límite impuesto en la fuerza de cada neurona.
- Un mecanismo que permite competir a las neuronas por el derecho de responder a unas entradas.

Según esto las neuronas se especializan y resultan ser hábiles detectores de rasgos característicos. Para que una neurona sea ganadora su nivel de actividad interno para un modelo de entrada específico (x) ha de ser el mayor de todas las neuronas. Si es así la salida será 1 y en las demás 0.

Si w_{ji} es el peso sináptico entre i y la neurona j, cada neurona tiene una cantidad fija de peso sináptico. Fijamos $\Sigma_i w_{ji}$ = 1 para todo j (neurona):

$$\Delta w_{ji} = \begin{cases} \eta(x_i - w_{ji}) & \text{si la neurona j gana} \\ 0 & \text{si la neurona j pierde} \end{cases}$$

Mueve los pesos wj de la neurona j hacia el modelo de entrada X.

8.5. El Perceptrón multicapa con backpropagation.

La red neuronal artificial denominada 'Perceptrón Multicapa' [1] tiene habitualmente una arquitectura feeedforward multicapa completamente conectada tal y como se muestra en la figura 8.12:

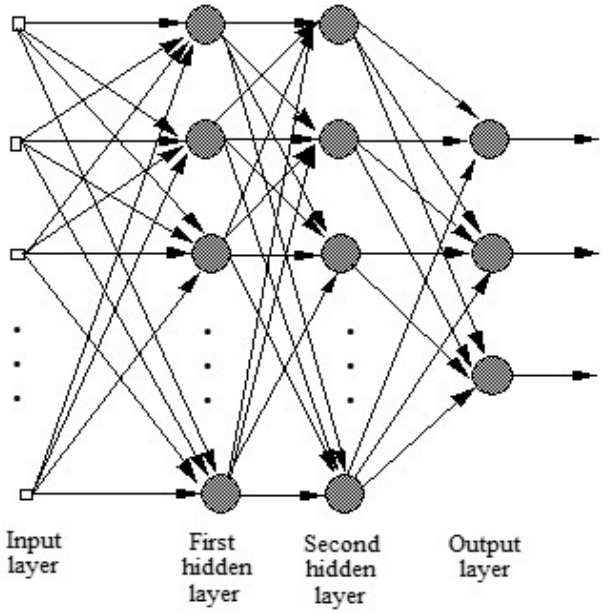

Figura 8.13: Arquitectura de un Perceptrón multicapa.

Dos tipos de señales:

1. Señales Función. Señales de entrada que se propagan a lo largo de la red desde la entrada hasta la salida
2. Señales Error. Señales que se originan en la capa de salida y se propagan hacia la de entrada.

La señal de error a la salida será la diferencia entre lo deseado y lo obtenido por la red. Siendo j la neurona de salida:

$$e_j(n) = d_j(n) - y_j(n)$$

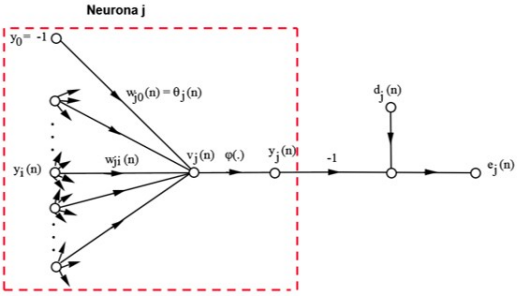

Figura 8.14: Flujo de las señales función.

Valor instantáneo del error cuadrático para todas las neuronas de la capa de salida:

$$\varepsilon(n) = \frac{1}{2}\Sigma_{j\epsilon C} e_j^2(n)$$

C conjunto de neuronas de la capa de salida:

$$Vj(n) = \Sigma_{i=0}^{p} wji(n)yi(n)$$

p es el número de estímulos a la neurona j:

$$yj(n) = \varphi j(vj(n))$$

Salida de la neurona j.

La corrección del peso es proporcional al gradiente instantáneo:

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} * \frac{\partial e_j(n)}{\partial y_j(n)} * \frac{\partial y_j(n)}{\partial v_j(n)} * \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$* \quad \frac{\partial \varepsilon(n)}{\partial e_j(n} = e_j(n) \qquad \left(\varepsilon(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n)\right)$$

$$* \quad \frac{\partial e_j(n)}{\partial y_j(n)} = -1 \qquad (\mathrm{ej(n) = dj(n) - yj(n)})$$

$$* \quad \frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \qquad (\mathrm{yj(n) = \varphi j(vj(n))})$$

$$* \quad \frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \qquad \left(\mathrm{vj(n) = \sum_{i=0}^{P} wji(n)\, yi(n)}\right)$$

Por lo que, teniendo en cuenta la regla delta:

$$\Delta wji(n) = \eta \delta j(n) yi(n)$$

En la neurona de **salida** el gradiente local será:

$$\delta_j(n) = e_j(n)\,\varphi_j'(v_j(n)) = [d_j(n) - y_j(n)]\,y_j(n)\,[1-y_j(n)]$$

En la neurona **oculta** el gradiente local será:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n)\,w_{kj}(n) = y_j(n)\,[1-y_j(n)] \sum_k \delta_k(n)\,w_{kj}(n)$$

El factor de aprendizaje:

η pequeño $\Rightarrow$ Cambios pequeños en los pesos

η demasiado $\Rightarrow$ grande peligro de inestabilidad

Proceso de aprendizaje del PMC.

- **Paso 0.** Normalizar (Altamente recomendable).
- **Paso 1.** Inicialización aleatoria y valores alrededor de cero de los pesos y umbrales de la red.
- **Paso 2.** Tomar un ejemplo n del conjunto de entrenamiento [x(n), d(n)] y propagarlo hacia delante para obtener la salida y(n).
- **Paso 3.** Evaluar el error e(n) para el caso n.
- **Paso 4.** Modificar pesos y umbrales según:
  - **4.1** Calcular el gradiente local para todas las neuronas de la capa de salida.
  - **4.2** Modificar pesos en capa de salida.
  - **4.3** Calcular el gradiente local para las demás neuronas de capa oculta desde la capa más cercana a la salida hacia la capa de entrada (retro-propagación)
  - **4.4** Modificar pesos en las capas por donde se va calculando.
- **Paso 5.** Se repiten los pasos 2, 3 y 4 para todos los ejemplos del conjunto de entrenamiento completando así una iteración o ciclo de aprendizaje.
- **Paso 6.** Se evalúa el error total cometido por la red o error de entrenamiento
- **Paso 7.** Se repiten los pasos 2, 3, 4, 5 y 6 hasta alcanzar un mínimo de error de entrenamiento, para lo cual se realizan m ciclos de aprendizaje.

Criterio de parada:

El algoritmo de aprendizaje ha **convergido:**

- Cuando el cambio en el error cuadrático medio por ciclo es **suficientemente pequeño** (0.1

- Cuando se ha conseguido sea **menor** que un umbral.
- Cuando los parámetros de la red (pesos) **no se mueven.**
- Cuando la generalización da **buenos resultados.**

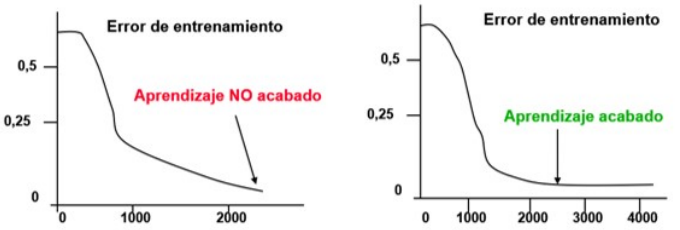

Figura 8.15: Criterio de parada del aprendizaje de un Perceptrón multicapa.

Proceso de entrenamiento, 2 formas:

- Proceso continuo o aprendizaje basado en ejemplo o modelo:
  - Los pesos se actualizan tras la presentación de cada ejemplo del conjunto de entrenamiento.
  - Se aprende ciclo a ciclo (o época a época) hasta la estabilización de pesos y convergencia de error de aprendizaje a un mínimo.
  - Es bueno presentar los ejemplos de entrenamiento en orden aleatorio de una época a otra.
- Aprendizaje por lotes o batch:
  - Los pesos se actualizan tras presentación de todos los ejemplos de un ciclo.

Capacidad de generalización:

Para evaluar el comportamiento de cualquier red neuronal, y en particular de un PMC, es imprescindible conocer el comportamiento de la red ante ejemplos NO usados en el entrenamiento. Esto se conoce como capacidad de generalización de la red.Habrá generalizado cuando se ha alcanzado un buen nivel de generalización o cuando nuevos ejemplos se aproximan bien a la función a que pertenecen (figura 8.15):

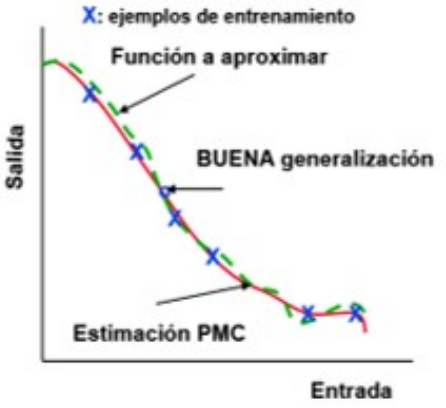

Figura 8.16: Buena generalización del aprendizaje de un Perceptrón multicapa.

La red ha aprendido bien los patrones pero generaliza mal ante nuevos ejemplos (figura 8.16):

Figura 8.17: Mala generalización del aprendizaje de un Perceptrón multicapa.

Para entrenar una red hacen falta DOS conjuntos de ejemplos, uno para entrenar la red y modificar sus pesos y umbrales: Conjunto de entrenamiento y otro diferente para medir la capacidad de generalización de la red: Conjunto de validación. Ambos han de tener ejemplos diferentes pero representativos de los modelos de conocimiento a aprender. La proporción de ejemplos alrededor de 60 % entrenamiento y 40 % validación, o 70/30... También se debe analizar la evolución del error de validación. El error de validación se va analizando al mismo tiempo que el de entrenamiento. Cada cierto número de ciclos de validación, se debe presentar a la red un conjunto de ejemplos de validación y calcular el error cometido sobre dicho conjunto. A veces un entrenamiento riguroso puede anular la capacidad de

generalización y por tanto es mejor hacer en paralelo el cálculo del error de entrenamiento y el de validación. Si los errores de entrenamiento y validación permanecen estables tras un cierto número de ciclos entonces el aprendizaje ha acabado con éxito (figura 8.17):

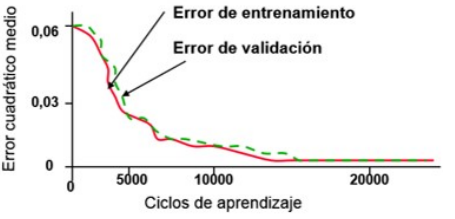

Figura 8.18: Aprendizaje exitoso de un Perceptrón multicapa.

Si a partir de cierto número de ciclos el error de validación comienza a aumentar frente al de entrenamiento es que se ha encontrado un mínimo de error pero se ha perdido capacidad de generalización y el aprendizaje no es correcto. En el ejmplo de la figura 8.18 se debería haber parado alrededor del ciclo 10.000, es lo que se conoce como sobreaprendizaje u overfitting:

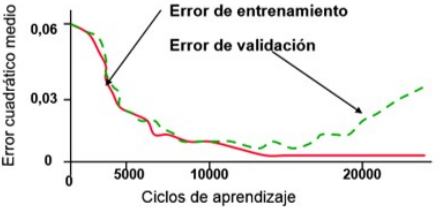

Figura 8.19: Sobreaprendizaje en el aprendizaje de un Perceptrón multicapa.

El **sobreaprendizaje** en un PMC se da cuando la red ha aprendido bien los ejemplos del conjunto de entrenamiento (demasiado bien) pero NO es capaz de generalizar o responder adecuadamente a los ejemplos del conjunto de validación. Hay varias causas del sobreaprendizaje. Entre otras el excesivo número de ciclos de aprendizaje, la poca diversidad en la información de los ejemplos del conjunto de entrenamiento o el excesivo número de neuronas ocultas.

El problema de los **mínimos locales** se da Cuando la superficie de error tiene muchos valles y crestas y el proceso de minimización del error puede acabar en un mínimo local. Esto es indeseable sobre todo si se queda lejos del mínimo global. Posibles soluciones pueden ser el aumentar algo el número de neuronas ocultas, hacer decrecer la tasa de aprendizaje a lo largo del proceso de aprendizaje o añadir ruido al método usado del descenso del gradiente.

El problema de la **saturación** o parálisis se da cuando la entrada total a una neurona de la red toma valores muy altos (positivos o negativos) y ésta se satura alcanzando un valor de activación máximo o mínimo. Se parece a un problema de mínimos locales, pero puede que luego crezca el error. La solución es evitar que la neurona trabaje en saturación.

8.6. Mapas de Kohonen.

Son redes **autoorganizadas** [2] capaces de codificar y luego reconocer modelos de rasgos característicos de un entorno, reconocen patrones (Clustering). Son uni o bi-dimensionales lattice de neuronas que calculan simples funciones discriminantes sobre las entradas recibidas mediante la selección de una unidad neuronal ganadora en función del mayor valor discriminante.

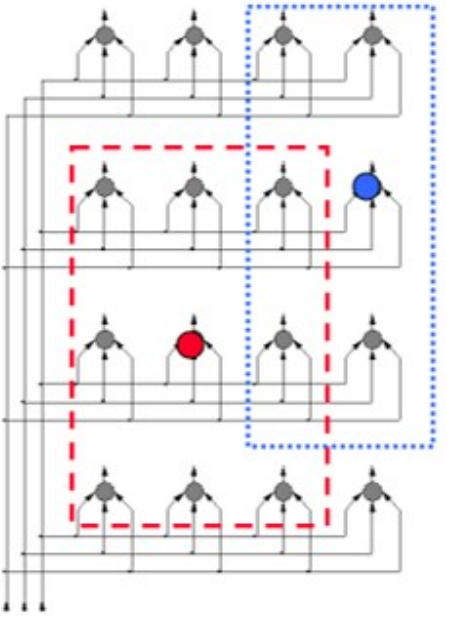

Figura 8.20: Arquitectura de un Mapa de Kohonen.

Función de vecindad topológica:

Se define $\Lambda_{i(x)}(n)$ como la función de **vecindad** topológica de la neurona ganadora i(x).

$\Lambda_{i(x)}$ es dependiente de n, número de ejemplos pasados en el aprendizaje en un momento del mismo.

La función de vecindad se suele elegir de valor grande al comienzo del aprendizaje y se va estrechando con el aumento de n

Algoritmo de aprendizaje:

1. **Inicialización:** Elegir valores aleatorios para el vector inicial de pesos wj(0), debe de ser diferente en cada neurona j=1,2, ... N
2. **Muestreo:** Escoger un ejemplo x de estímulo a la red representante con una cierta probabilidad del modelo que se desea aprenda la red.
3. **Similitud:** Encontrar el mejor índice ganador al pasar el ejemplo n. Esto es saber cuál es la neurona ganadora o más parecida al estímulo x (Norma Euclídea).
4. **Actualización:**

$$w_j(n+1) = \begin{cases} w_j(n) + \eta(n)[x(n) - w_j(n)] & \Rightarrow j \subset \Lambda_{i(x)}(n) \\ w_j(n) & \Rightarrow j \not\subset \Lambda_{i(x)}(n) \end{cases}$$

5. **Continuación:** Ir al paso 2 hasta que no se experimenten cambios importantes en los pesos y por tanto, en el mapa.

8.7. Bibliografía

1. Chauvin, Y. y Rumelhart, D. E. (1995). Backpropagation: Theory, Architectures, and Applications. Psychology Press.
2. Kohonen, T. (1990). The self-organizing map. Proc. of the IEEE, vol. 78, no. 9, (pp. 1464-1480).