TRATAMIENTO DIGITAL DE SEÑALES
Ingeniería de Telecomunicación (4º, 2º c)

Unidad 13ª: Redes Neuronales

Aníbal R. Figueiras Vidal
Jesús Cid Sueiro
Ángel Navia Vázquez

Área de Teoría de la Señal y Comunicaciones
Universidad Carlos III de Madrid

Generalidades

Una Red Neuronal (NN) es una máquina:

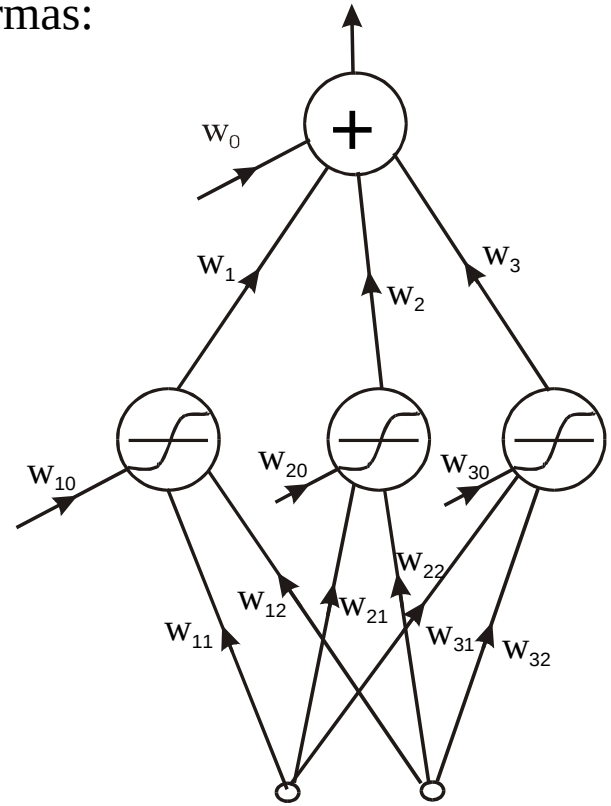
- de arquitectura paralela
- Con unidades no lineales (“neuronas”) sencillas
- Estando dichas unidades dispuestas normalmente en capas

Forma típica: $o = \sum_j w_j \phi_{\mathbf{w}'_j}(\underline{x}) + w_0$

$\phi_j = th\left(\sum_i w_{ji} x_i + w_{j0}\right)$: **Perceptrones** (iterables (MLP))

$\phi_j = G\left(\mathbf{x} - \mathbf{m}_j, V_j\right)$ (o similares con centros $\mathbf{m}_j$): Red de **Funciones Base Radiales** (RBFN) (puede prescindirse del factor de la exponencial)

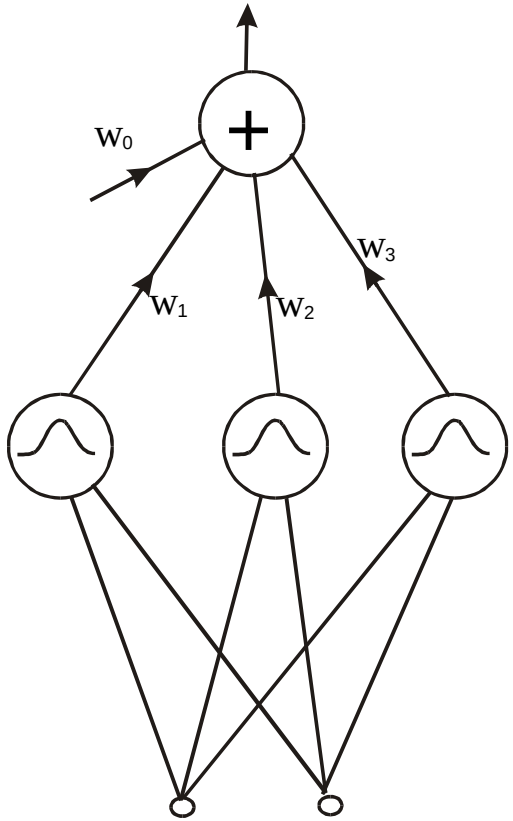
Formas:



MLP

(para decisión: sustituir sumador por sigmoide)

(Aproximadores globales)



RBFN

(Aproximadores locales)

Características

Favorables:
- rapidez (implementadas en paralelo)
- degradación suave ante fallos (por distribución)
- gran potencia expresiva (sin explosión dimensional)

Desfavorables:
- difícil elección
- difícil dimensionado
- difícil y lento entrenamiento (dan lugar a muchos mínimos locales) (consecuencia: baja capacidad de seguimiento)
- difícil interpretación

Los MLP

Ya se ha visto que derivan de los Perceptrones con activación blanda.

Ya se ha visto que tienen alta capacidad expresiva; en puridad,

* Un MLP con una capa oculta (dos capas) del adecuado número de unidades puede establecer cualquier correspondencia entre entrada y salida

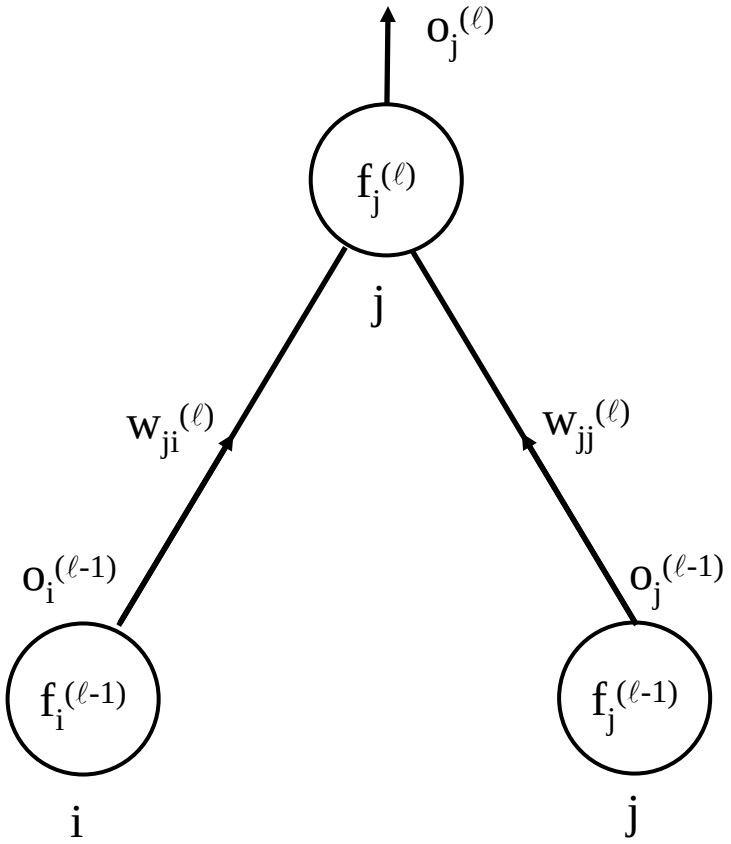
(Nótese: - no es constructivo

- no implica que no pueda ser más eficaz utilizar dos capas ocultas)

Entrenamiento del MLP: **El algoritmo de Retropropagación (BP)**

(Werbos; Parker; Rumelhart)

Notación:



Algoritmo:

Se basa en la utilización de la regla de la cadena para calcular gradientes:

$$C(\mathbf{w}) = \frac{1}{2}\sum_{j'=1}^{N_L}\left(d_{j'} - o_{j'}^{(L)}\right)^2$$

$$\frac{\partial C(\mathbf{w})}{\partial w_{ji}^{(\ell)}} = \frac{\partial C(\mathbf{w})}{\partial o_j^{(\ell)}}\frac{\partial o_j^{(\ell)}}{\partial w_{ji}^{(\ell)}} = \frac{\partial C(\mathbf{w})}{\partial o_j^{(\ell)}}\, o_i^{(\ell-1)} f_j'^{(\ell)} \qquad (w_{j0}^{(\ell)} : \sin o_i)$$

$$\left(o_j^{(\ell)} = f_j^{(\ell)}\left(w_{j0}^{(\ell)} + \sum_{i'} w_{ji'}^{(\ell)}\, o_{i'}^{(\ell-1)}\right)\right)$$

$$\frac{\partial C(\mathbf{w})}{\partial o_j^{(\ell)}} = \begin{cases} -(d_j - o_j^{(L)}) & , \ell = L \\ \sum_{n_{\ell+1}=1}^{N_{\ell+1}} \frac{\partial C(\mathbf{w})}{\partial o_{n_{\ell+1}}^{(\ell+1)}}\frac{\partial o_{n_{\ell+1}}^{(\ell+1)}}{\partial o_j^{(\ell)}} = \sum_{n_{\ell+1}=1}^{N_{\ell+1}} \frac{\partial C(\mathbf{w})}{\partial o_{n_{\ell+1}}^{(\ell+1)}}\, w_{n_{l+1}j}^{(\ell+1)}\, f_{n_{\ell+1}}'^{(\ell+1)} & , \ell < L \end{cases}$$

$$\left(o_{n_{\ell+1}}^{(\ell+1)} = f_{n_{\ell+1}}^{(\ell+1)}\left(w_{n_{\ell+1}0}^{(\ell+1)} + \sum_{j'=1}^{N_L} w_{n_{\ell+1}j'}^{(\ell+1)}\, o_{j'}^{(\ell)}\right)\right)$$

con la última expresión:

- se puede calcular $\partial C/\partial o$ para la capa de salida directamente
- y, consecutivamente, para ℓ = L-1, L-2, ..., 1 (¡retropropagación!)

basta insertar los valores resultantes en la expresión de $\partial C/\partial w$ para completar el algoritmo.

Obsérvese que, si se emplea $\Delta_j^{(\ell)} = -\frac{\partial C(\mathbf{w})}{\partial o_j^{(\ell)}} f_j'^{(\ell)}$, se tiene

$$w_{ji}^{(\ell)}(k+1) = w_{ji}^{(\ell)}(k) + \eta^{(\ell)} \Delta_j^{(\ell)}(k) o_i^{(\ell-1)}(k)$$

formalmente análogo al LMS, con $\Delta_j^{(\ell)}$ en el papel del error: por eso se llama también Regla Delta Generalizada.

(Nótese que no conviene o entre 0 y 1: 0 para el entrenamiento)

(Recuérdese que, para la th, $f' = 1-o^2$)

Observaciones

* El algoritmo BP se puede aplicar para cualquier función de coste sin más que cambiar $\partial C/\partial o$ para $\ell = L$.

* De modo análogo al visto se pueden extender los otros métodos de búsqueda local al MLP.

T: Establecer el algoritmo que ha de aplicarse para el cálculo de las derivadas segundas.

Sobre la aplicación del MLP

A. Dimensionado

- En la mayoría de los casos basta con una capa oculta; la necesidad de 2 es infrecuente en estimación, y muy infrecuente para decisión.
- Para decisión (como habitualmente) conviene un nodo por clase (uno solo en casos binarios).
- Reglas prácticas para el número de unidades:
  - una capa oculta: $2N+1 \lesssim N_1 \lesssim 3N$ (Kudrycki)
  - dos capas ocultas: $N_1 \lesssim 3N$, $N_2 \lesssim 2N_3$ (Lippmann)

tanteando de menor a mayor.

- En todo caso, el número de muestras disponibles limita el número de pesos: conviene no bajar de 10 muestras por peso (y un límite absoluto es 1 muestra por peso)

- Existen
  - algoritmos constructivos o de crecimiento: establecen la arquitectura a partir de una versión subdimensionada, añadiendo elementos;
  - algoritmos de reducción o de poda: justo al revés;

que proporcionan ventajas en prestaciones y, obviamente, en carga computacional para el entrenamiento (sobre todo, los primeros).

T: (Familias de) Algoritmos de Poda.

B. Algoritmo BP

- Los pasos $\eta(\ell)$ han de tantearse (como en el de gradiente) entre valores pequeños (típicamente, décimas a milésimas); y debe notarse que convienen valores mayores en las capas más lejanas de la salida: $\Delta(\ell)$ va incorporando factores f'.

- Debe notarse que pueden producirse efectos de parálisis por saturación temprana de neuronas internas (por lo que hay que cuidar mucho los pasos).

  Se puede combatir tal saturación temprana:

  - incrementando artificialmente las derivadas;
  - reduciendo el nivel de las entradas;
  - modificando la función de coste (p.ej.: no corrigiendo momentáneamente para muestras bien clasificadas).

- También es obvio que no pueden inicializarse los pesos a cero (se hacen nulas las o y no hay entrenamiento); típicamente se inicializan en valores aleatorios pequeños (p.ej., tomados de una distribución U[-0.5, 0.5]);

- Por último, ha de señalarse que, dado lo complicado de la topografía del espacio de búsqueda (C vs. **w**), es muy habitual que los algoritmos de gradiente conjugado o Newton no den ventaja respecto al BP (suele ser preferible gestionar el escalón).

C. Muestras

Además de lo habitual

- muestras representativas del problema,
- posibilidad de selección, ...

- Conviene normalizar las componentes de las muestras (típicamente a potencia unitaria o valor de pico unitario) para evitar el dominio de alguna(s) de ellas en el inicio de la búsqueda.
- Conviene aleatorizar el orden de presentación de las muestras, pero ciclar la presentación de la secuencia resultante hasta la detención (se evitan así problemas de inestabilidad en la búsqueda).

- En problemas sencillos (MLP resultantes de tamaño reducido), suele facilitar el entrenamiento representar las muestras mediante códigos sencillos (tipo 1 bit: B bits, termómetro, y análogos).

- Hay que ser cuidadoso con posibles preprocesados de las muestras: sólo son recomendables cuando se tiene la certeza de eliminar información que se sabe de cierto que es irrelevante para la resolución del problema abordado.

Limitaciones intrínsecas de los MLP

- La escalabilidad es mala: cuando se cambia una arquitectura (aunque sea simplemente para dividir una clase en dos, p. ej.), no se puede reaprovechar nada de lo hecho.
- No son adecuados para problemas espaciotemporales (datos de la forma $x[\theta(t)]$), al tratarse de estructuras estrictamente progresivas.

Soluciones:

- preprocesado temporal
- versiones realimentadas (son delicadas)

Las RBFN

Nótese que hay tres tipos de parámetros:

- los centroides $\mathbf{m}_j$
- las covarianzas V_j (suele haber circularidad : v_j)
- los pesos de salida w_j

y que los efectos de las unidades son locales.

Hay tres modos (básicos) de entrenamiento:

- mediante selección aleatoria de los centroides;
- mediante preestablecimiento de los centroides;
- mediante algoritmos de búsqueda convencionales (gradiente).

Entrenamiento por selección aleatoria

- los centroides se seleccionan aleatoriamente de entre las muestras;
- las varianzas de las Gauss se estiman agrupando muestras por proximidad, calculando

$$v'_j = \frac{1}{\#\, j} \sum_{k_j} \left\| \mathbf{x}^{(k_j)} - \mathbf{m}_j \right\|_2^2$$

y sobredimensionando este valor: $v_j \cong (1.5 - 2)\, v'_j$
(nótose que, al agrupar, no se consideran “muestras lejanas”)

- los pesos de salida se entrenan convencionalmente (LMS, p. ej.)

Entrenamiento por preselección

- los centroides se fijan mediante un proceso previo : p. ej., un modelado EM o un agrupamiento tipo K-medias;
- varianzas y pesos, como en el caso anterior.

(Al contrario de lo que puede parecer, no deben esperarse grandes ventajas: EM o K-medias tienden a modelar densidades de probabilidad, y los costes típicos a forzar a la salida estimaciones de las probabilidades “a posteriori”; y entre unas y otras no hay una correspondencia directa: recuérdese la forma del clasificador basado en EM presentada en la Unidad 8).

Entrenamiento por búsqueda directa (gradiente)

Para una salida y supuesto coste cuadrático (las generalizaciones son inmediatas):

$$C = \frac{1}{2}(d-o)^2 = \frac{1}{2}(d - \sum_{j'} w_{j'} \exp\left[-\frac{\|\mathbf{x}-\mathbf{m}_{j'}\|_2^2}{2v_{j'}}\right])^2$$

$$\frac{\partial C}{\partial w_j} = -(d-o)\exp\left[-\frac{\|\mathbf{x}-\mathbf{m}_j\|_2^2}{2v_j}\right]$$

$$\frac{\partial C}{\partial \mathbf{m}_j} = -(d-o)\exp\left[-\frac{\|\mathbf{x}-\mathbf{m}_j\|_2^2}{2v_j}\right]\frac{\mathbf{x}-\mathbf{m}_j}{v_j}$$

$$\frac{\partial C}{\partial v_j} = -(d-o)\exp\left[-\frac{\|\mathbf{x}-\mathbf{m}_j\|_2^2}{2v_j}\right]\frac{\|\mathbf{x}-\mathbf{m}_j\|_2^2}{2v_j^2}$$

que, particularizadas para la iteración k y con sus correspondientes escalones, se insertan en la expresión general del algoritmo de gradiente.

Observaciones

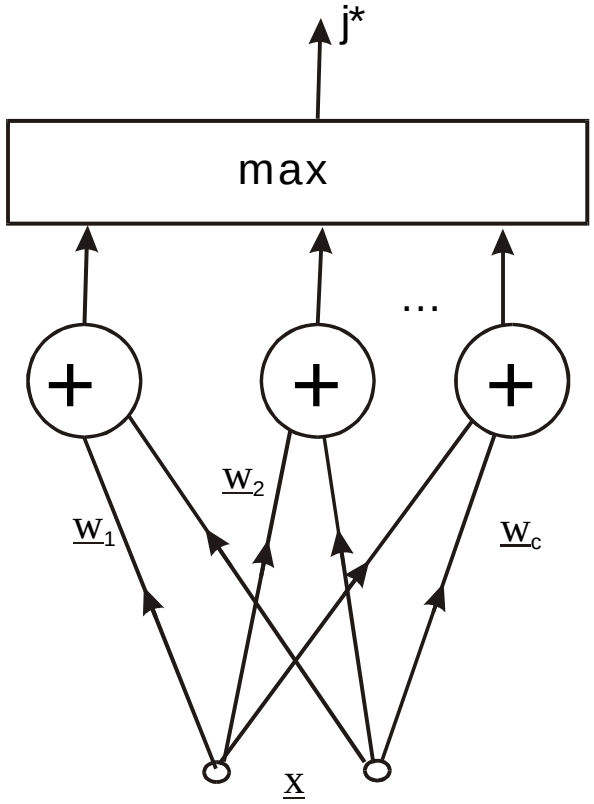
- La búsqueda directa no suele dar buenos resultados: tanto por la existencia de mínimos locales como por la dificultad de asignar η_w, η_m, η_v.

- En general, la dependencia de los resultados con $\{v_j\}$ es grande: puede resultar conveniente expresarlas de la forma $\{\alpha_j v\}$ y realizar una exploración de v (incluso puede procederse así por regiones).

- Como cabe esperar, las RBFN dan muy buenos resultados en problemas con importantes variaciones locales.

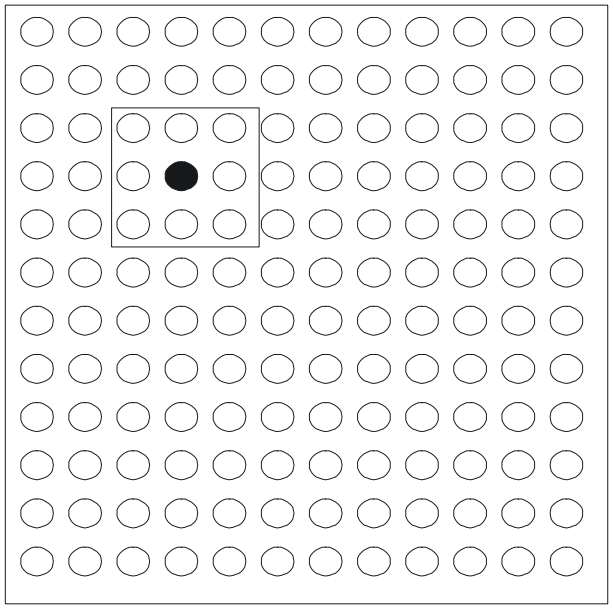
Versiones neuronales del Agrupamiento

Es inmediata la estructuración en paralelo de los métodos competitivos; así, el Algoritmo (Básico) de Kohonen es asociable con la **Red (Básica) de Kohonen**:

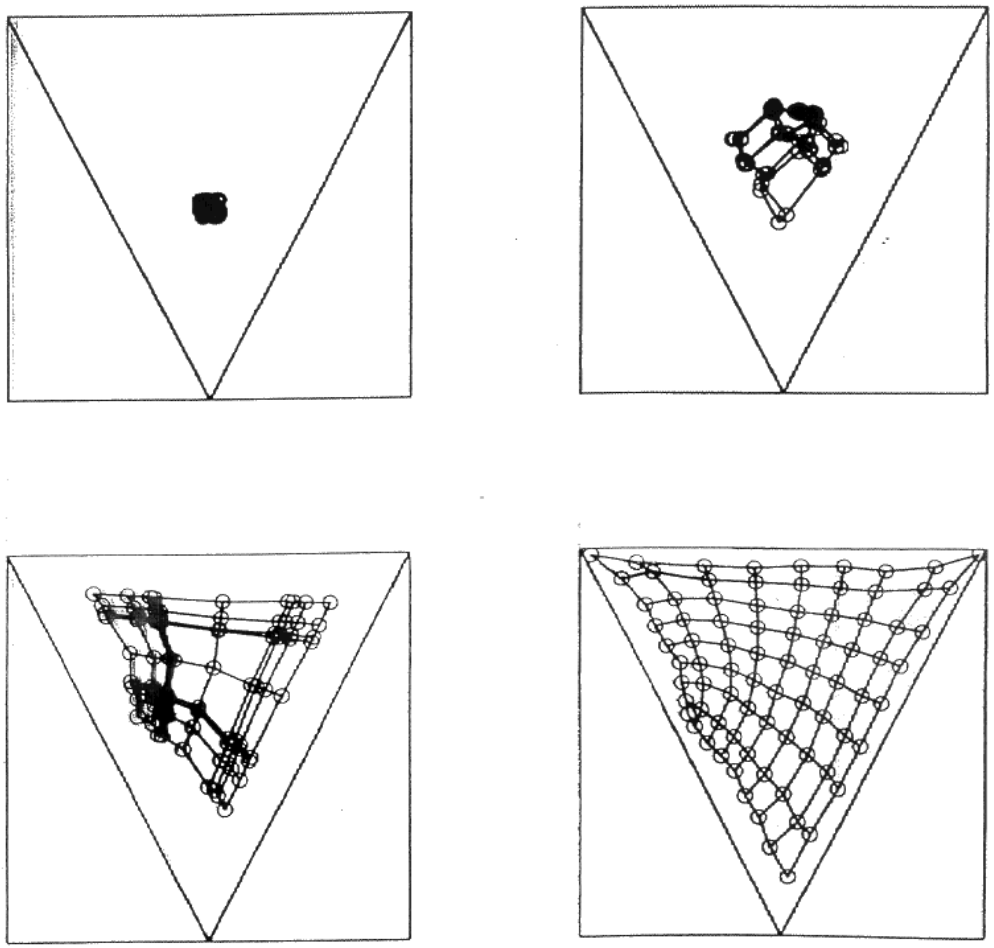


$$\mathbf{w}_j(k+1) = \begin{cases} \mathbf{w}_j(k) + \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_j(k)] & , j = j^* \\ \mathbf{w}_j(k) & , j \neq j^* \end{cases}$$

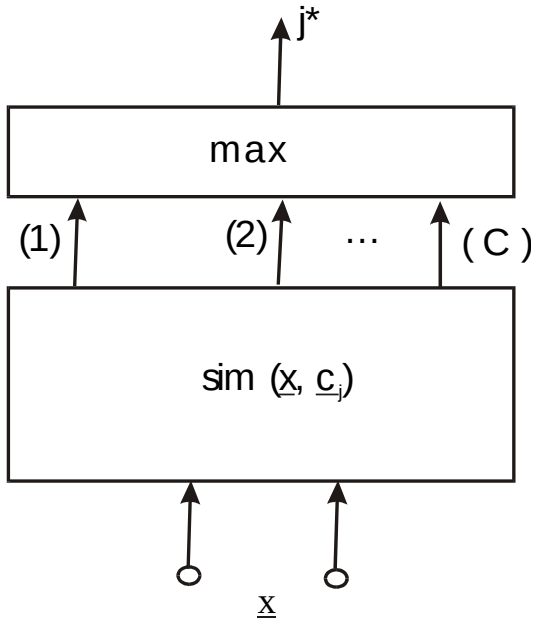
De esta forma (NN) propuso Kohonen la aplicación de algoritmos competitivos con inicialización por arrastre, que denominó Mapas Autoorganizables (de Características)) (SO(F)M: Self-Organizing (Feature) Maps), porque recubren el espacio de las muestras con una “red” de representantes conectados según los entornos que se definan.



Despliegue del SOFM



- Las generalizaciones son obvias para otras medidas de similitud: basta sustituir la capa de productos escalares por <u>otra capa de entrada</u> en que se calculen las similitudes con los representantes $\{c_j\}$



actualizando los $\{c_j\}$ del modo (acretivo) que corresponda.

- Y también lo son para modos no acretivos, e incluso para redes no competitivas (minimización de un objetivo).

Los Cuantificadores Vectoriales con Aprendizaje ("Learning Vector Quantizers", LVQ)

Se propusieron por Kohonen como paso siguiente a su Red Básica para clasificación: es un caso claro de aprovechar el agrupamiento para reducir el esfuerzo de diseño de un clasificador.

Hay tres versiones fundamentales (y algunas variantes):

* **LVQ1**:

$$\mathbf{w}_j(k+1) = \begin{cases} \mathbf{w}_j(k) + \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_j(k)], & \text{si } \begin{cases} j = \arg[\max_l \{\mathbf{w}_j^T(k)\mathbf{x}^{(k)}\}], \\ \text{y } \mathbf{x}^{(k)} \text{ está en su clase} \end{cases} \\ \mathbf{w}_j(k) - \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_j(k)], & \text{si } \begin{cases} j = \arg[\max_l \{\mathbf{w}_j^T(k)\mathbf{x}^{(k)}\}], \\ \text{y } \mathbf{x}^{(k)} \text{ no está en su clase} \end{cases} \\ \mathbf{w}_j(k), \text{ en otro caso} \end{cases}$$

* **LVQ2**

- sólo hay adaptación si hay error y además $\mathbf{x}^{(k)}$ está en una cierta ventana entre el peso ganador $\mathbf{w}_j$ y el más próximo de los que pertenecen a su clase, $\mathbf{w}_m$; entonces:

$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) - \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_j(k)]$$

$$\mathbf{w}_m(k+1) = \mathbf{w}_m(k) + \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_m(k)]$$

(hay que detenerlo)

* **LVQ3**

- sólo hay adaptación si hay clasificación correcta y además $\mathbf{x}^{(k)}$ está en una cierta ventana entre el peso ganador $\mathbf{w}_j$ y el más próximo, $\mathbf{w}_m$; entonces

$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) + \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_j(k)],$$

$$\mathbf{w}_m(k+1) = \begin{cases} \mathbf{w}_m(k) - \eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_m(k)], & \text{si } \mathbf{w}_m \text{ representa a otra clase} \\ \mathbf{w}_m(k) - \varepsilon\eta(k)[\mathbf{x}^{(k)} - \mathbf{w}_m(k)], & \text{si } \mathbf{w}_m \text{ representa a la misma clase} \end{cases}$$

$$(0.1 \le \varepsilon \le 0.5)$$