Capítulo 6

Red de Contrapropagación y ART

6.1. Red de Contrapropagación

La red de contrapropagación[1] es una combinación de redes, al estilo de lo que ocurría con el LVQ. Sin embargo, es esencialmente distinta. Por ejemplo, la CPN codifica un vector de entrada en otro vector, de salida (mientras que en el LVQ se toma un vector de entrada y se coge el valor de salida).
Combina **aprendizaje sin supervisar** con aprendizaje supervisado[2], donde el aprendizaje sin supervisar corresponde a la primera capa y el supervisado a la segunda. El conjunto funciona como aprendizaje supervisado (téngase en cuenta que a la capa de salida le son necesarias las salidas deseadas para cada patrón).
La CPN asocia entradas X con salidas Y. Es decir, si existe una función F / $X = F(Y)$, entonces la CPN encontrará esa función F. Además, y simétricamente, si existe la función F^{-1} / $Y = F^{-1}(X)$, entonces la CPN también la aprenderá.

6.1.1. Características

Las siguientes son características que hacen a la CPN una red interesante:

- La CPN combina distintos tipos de redes y, por tanto, distintos tipos de aprendizaje en cada capa.
  Esto, para muchos problemas, es una ventaja, pues permite combinar

[1] también conocida como CPN
[2] según alguna bibliografía, una *red de Kohonen* y una *capa de perceptrones*

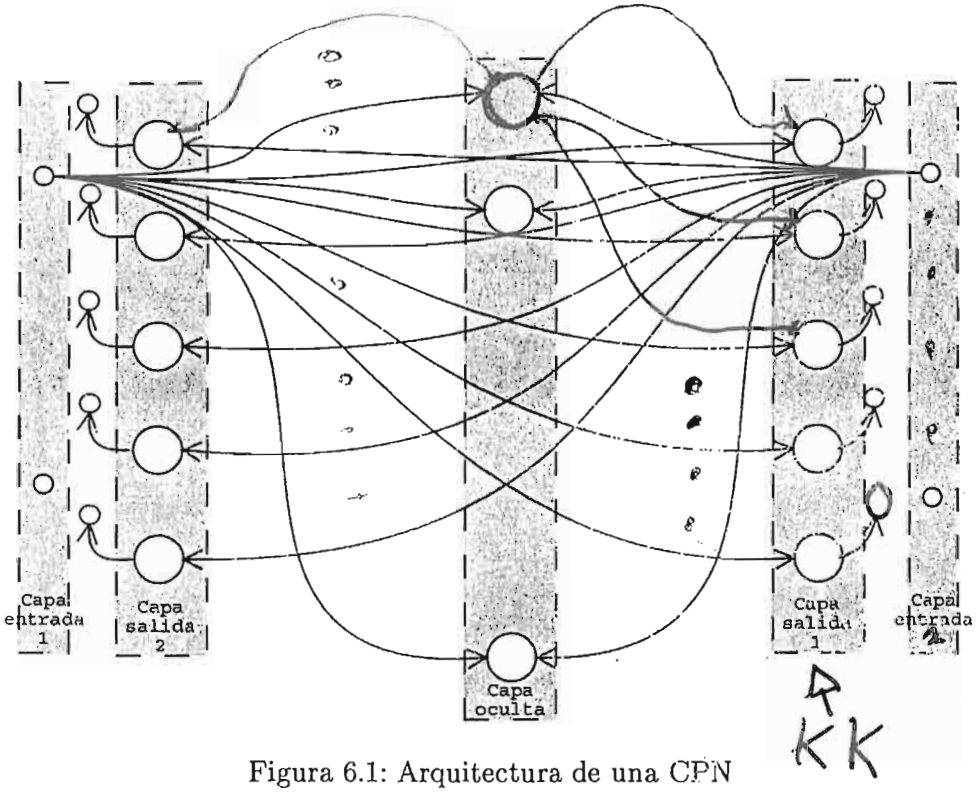

Figura 6.1: Arquitectura de una CPN

las ventajas de cada técnica[3].

- Se consigue con esta combinación una velocidad de aprendizaje alta.

Otras características de las redes CPN son:

- Son redes totalmente conectadas.
- Globalmente funcionan como aprendizaje supervisado.
- **Limitación**: Es necesario que vectores de entrada similares lleven asociados vectores de salida también parecidos. En concreto, las entradas que sean clasificadas por un único PE deberán (como veremos) tener vectores de salida cercanos para que la red funcione aceptablemente. Si esto no es así, y queremos utilizar una CPN, deberemos aumentar el número de PEs de la capa competitiva. Con ello lograremos que cada PE represente a un número menor de entradas; entonces, el número de vectores de entrada que deben tener salidas similares se reduce, tanto en número como en distancia entre ellos. Así, será más fácil cumplir la restricción.

[3]hecho que, en determinados problemas facilita la resolución de los mismos

6.1.2. Entrenamiento de la CPN

Podemos dividir el entrenamiento de la CPN en dos grandes fases:

1. Entrenamiento de la estructura competitiva, usando el *algoritmo de aprendizaje de la* ***Instar*** (ver página 78.

   a) Inicializar los pesos de la estructura.

   b) Aplicar un vector de entrada a la estructura. La red lo procesará con lo que tendremos un PE *ganador* en la capa competitiva. El PE ganador se decidirá, de nuevo, eligiendo al que *esté más próximo* (distancia Euclídea) de la entrada.

   c) Modificar los pesos del ganador utilizando la *regla de Kohonen* (ver página 48).

   $$\omega_{ki}(t+1) = \omega_{ki}(t) + \mu(t)(x_i(t) - \omega_{ki}(t)) \qquad (6.1)$$

   d) Repetir los dos pasos anteriores para todos los patrones del CE.

   e) Repetir el paso anterior hasta que se estabilicen los pesos.

   Cuando se termine esta fase del entrenamiento, los pesos de cada PE de esta capa competitiva serán, aproximadamente, la *media de todas las entradas que activan ese PE.*

2. Entrenamiento de la estructura supervisada, usando el *algoritmo de aprendizaje de la* ***Outstar*** (ver página 79). Pueden darse, en este entrenamiento, varios casos:

   A) Cada clase (cada PE) sólo tiene **una salida deseada**. En ese caso, el vector de pesos del PE será *exactamente* la salida deseada para esa clase[4].

   B) Cada vector de entrada de una clase tiene una salida deseada diferente. En esta situación, un PE debería proporcionar diferentes salidas deseadas según cuál sea la entrada que lo ha hecho ganar. La solución es utilizar un *algoritmo de aprendizaje por corrección de error*, haciendo que los pesos se aproximen lo máximo a las salidas deseadas.
   Como se puede suponer, el resultado de la aplicación de este algoritmo es que los pesos de salida correspondientes a una clase(a un PE) tienden hacia la media de las salidas deseadas para esa clase.

[4]nótese que la salida ante una entrada es el vector de pesos del PE ganador

1) Aplicar un vector de entrada (con su correspondiente salida deseada) a la red.
2) Obtener el PE de la capa oculta (competitiva) que gana.
3) Modificar los pesos entre el PE ganador de la capa oculta y los de la capa de salida según:

$$\omega_i(t+1) = \omega_i(t) + \beta(y_i - \omega_i(t)) \tag{6.2}$$

4) Repetir los pasos 2B1, 2B2 y 2B3 hasta obtener las salidas deseadas para todos los patrones (o una buena aproximación) Observar que, como sólo un PE de la capa oculta gana, sólo los pesos de ese PE se entrenarán.

Explicaremos ahora con un poco de detenimiento los dos tipos de PEs en los que se basan los entrenamientos anteriores.

Instar

[5] Fue un PE ideado por Grossberg, y está totalmente relacionado con la Outstar. El motivo de su creación era, en principio, intentar explicar el fenómeno de la visión. Además de idear estos PEs, también preparó *reglas de aprendizaje propias.*
La **Instar** es el primero de estos PEs. Trabaja con *vectores de entrada* y *pesos* **normalizados.** Los pesos de este PE tienden, durante el entrenamiento, a las entradas. Veamos el porqué.

El valor neto de una Instar se calcula como el *producto escalar* de la entrada por los pesos, es decir:

$$neta = I\omega \tag{6.3}$$

Y la salida de la Instar está gobernada por una ecuación diferencial:

$$\dot{y} = -ay + bneta \qquad \text{con } a, b > 0 \tag{6.4}$$

Esto es, la Instar va tendiendo poco a poco hacia su salida definitiva, que viene dada por el valor de equilibrio de y[6] en la ecuación 6.4, que es:

$$y^{eq} = \frac{b}{a} neta \tag{6.5}$$

[5] Utilizaremos en este punto la notación $\frac{dy}{dt} = \dot{y}$.
[6] calculada en la ecuación 6.4 cuando $\dot{y} = 0$

Todas las Instar de la capa competirán por ganar. Y el ganador será el que tenga un mayor valor de y^{eq}; observando la ecuación 6.5, vemos que esto ocurrirá para el PE cuya *neta* sea máxima (pues b y a son parámetros iguales para todas las Instar). Por lo tanto, ganará la Instar cuyos pesos ω se parezcan más a la entrada I. Siguiendo la idea de siempre del aprendizaje competitivo y de la regla de Hebb (el que gana entrena para tener más posibilidades de ganar con esa entrada la próxima vez), los pesos del ganador tenderán a acercarse a la entrada. La modificación de pesos del ganador se rige, en teoría, por la siguiente expresión:

$$\dot{\omega} = -c\omega + dIy \tag{6.6}$$

Fijémonos en esta ecuación que, cuando quitemos la entrada (y por tanto $I = 0$), la modificación de pesos nos quedaría $\dot{\omega} = -c\omega$, con lo cual los pesos tenderían a 0. Por ello surgió otra aproximación a la ecuación 6.6, para evitar este hecho:

$$\dot{\omega} = (-c\omega + dI)U(neta) \tag{6.7}$$

con lo cual, cuando no haya entrada para aprender, los pesos no se modifican, y cuando haya entrada, el valor de equilibrio sería $\omega^{eq} = \frac{d}{c}I$ que, haciendo $c = d$ queda $\omega^{eq} = I$.
Sobre la ecuación 6.7 hacemos la siguiente aproximación: $\frac{d\omega}{dt} \approx \frac{\Delta\omega}{\Delta t}$. Entonces

$$\Delta\omega = (-c\omega + dI)\Delta t \tag{6.8}$$

y, aproximando de nuevo $c = d$,

$$\Delta\omega = (-\omega + I)c\Delta t \tag{6.9}$$

que es, al fin y al cabo, la ecuación de cambio de pesos que utilizan la mayor parte de las estructuras competitivas.

Outstar

La Outstar es un PE que toma un valor (proporcionado, generalmente, por una Instar) y lo transforma (lo asocia) con un vector.

Razonando de un modo similar, llegaríamos a la expresión

$$\omega_i(t+1) = \omega_i(t) + \beta(y_i - \omega_i(t)) \tag{6.10}$$