Introducción a las redes neuronales aplicadas

Conceptos básicos

Las Redes Neuronales (*NN*: Neural Networks) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros.

El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de *actividad nerviosa*. Este modelo era un modelo binario, donde cada neurona tenía un escalón o umbral prefijado, y sirvió de base para los modelos posteriores.

Una primera clasificación de los modelos de NN es:

1. **Modelos inspirados en la Biología**: Estos comprenden las redes que tratan de simular los sistemas neuronales biológicos, así como ciertas funciones como las auditivas o de visión.

2. **Modelos artificiales aplicados**: Estos modelos no tienen por qué guardar similitud estricta con los sistemas biológicos. Sus arquitecturas están bastante ligadas a las necesidades de las aplicaciones para las que son diseñados.

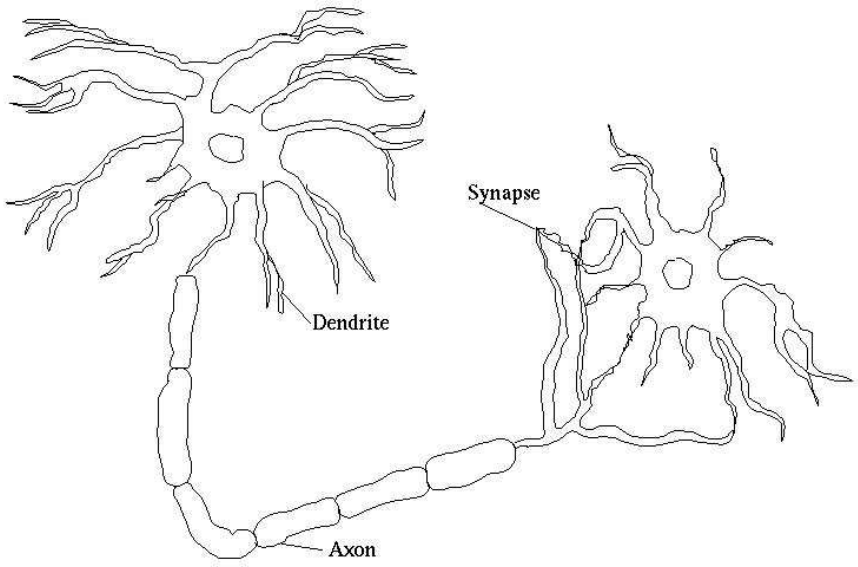
Redes Neuronales de tipo biológico

Se estima que el cerebro humano contiene más de cien mil millones (10^{11}) de neuronas y 10^{14} sinapsis en el sistema nervioso. Los estudios realizados sobre la anatomía del cerebro

humano concluyen que hay, en general, más de 1000 sinapsis por término medio a la entrada y a la salida de cada neurona.

Aunque el tiempo de conmutación de las neuronas biológicas (unos pocos milisegundos) es casi un millón de veces mayor que en las actuales componentes de las computadoras, las neuronas naturales tienen una *conectividad* miles de veces superior a la de las artificiales.

El objetivo principal de las redes neuronales de tipo biológico es desarrollar operaciones de síntesis y procesamiento de información, relacionadas con los sistemas biológicos.



Las neuronas y las conexiones entre ellas (sinapsis) constituyen la clave para el procesado de la información.

La mayor parte de las neuronas poseen una estructura de árbol, llamada *dendritas*, que reciben las señales de entrada procedentes de otras neuronas a través de las *sinapsis*.

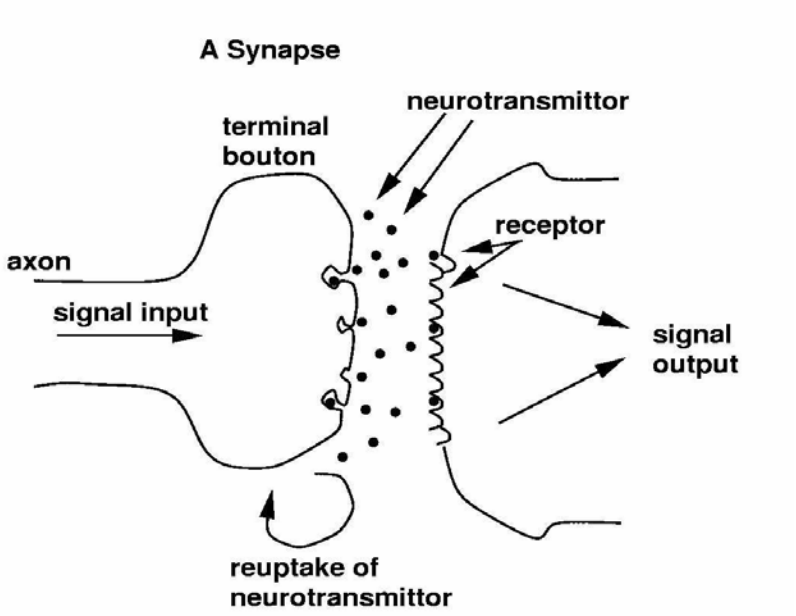
Una neurona consta de tres partes:

1. El cuerpo de la neurona,

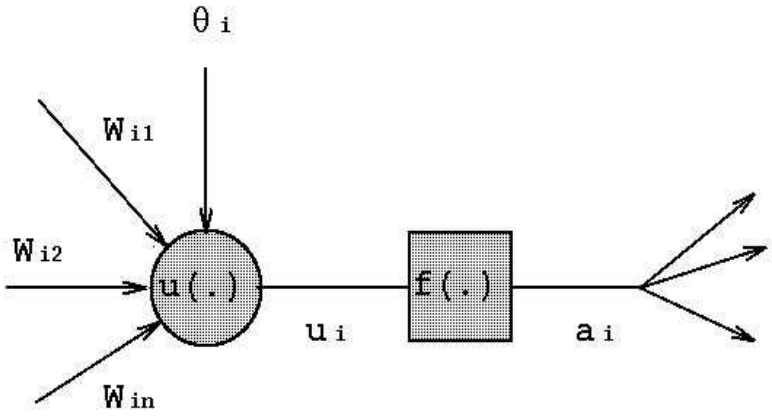
2. Las dendritas, que reciben las entradas,

3. El axón, que lleva la salida de la neurona a las dendritas de otras neuronas.

La forma completa en la que dos neuronas se relacionan no es totalmente conocida, y depende además del tipo particular de cada neurona. En general, una neurona envía su salida a otras por su axón, y éste lleva la información por medio de diferencias de potencial eléctrico.



Este proceso es a menudo modelizado como una regla de propagación representada por una función $u(\cdot)$. La neurona recoge las señales por su sinapsis sumando todas las influencias excitadoras e inhibidoras. Si las influencias excitadoras positivas dominan, entonces la neurona produce una señal positiva y manda este mensaje a otras neuronas por sus sinapsis de salida. En este sentido, la neurona actúa como una simple función escalón $f(\cdot)$.



Redes Neuronales Artificiales (NN)

Las NN aplicadas están, en general, inspiradas en las redes neuronales biológicas, aunque poseen otras funcionalidades y estructuras de conexión distintas a las vistas desde la perspectiva biológica. Las características principales de las NN son las siguientes:

1. *Auto-Organización y Adaptabilidad*: utilizan algoritmos de aprendizaje adaptativo y auto-organización, por lo que ofrecen mejores posibilidades de procesado robusto y adaptativo.

2. *Procesado no Lineal*: aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumenta su inmunidad frente al *ruido*.

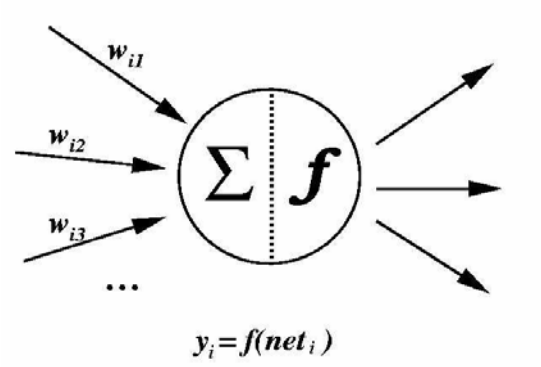
3. *Procesado Paralelo*: normalmente se usa un gran número de nodos de procesado, con alto nivel de *interconectividad*.

El elemento básico de computación (modelo de neurona) se le llama habitualmente *nodo* o *unidad*. Recibe un *input* desde otras unidades o de una fuente externa de datos. Cada input tiene un **peso** asociado w, que se va modificando en el llamado proceso de aprendizaje . Cada unidad aplica una función dada f de la suma de los inputs ponderadas

mediante los pesos

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

El resultado puede servir como output de otras unidades.



Las características de las NN juegan un importante papel, por ejemplo, en el procesado de señales e imágenes. Se usan *arquitecturas* que comprenden elementos de procesado adaptativo paralelo, combinados con estructuras de interconexiones jerárquicas.

Hay dos fases en la modelización con redes neuronales:

- **Fase de entrenamiento**: se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos (parámetros) que definen el modelo de red neuronal. Se calculan de manera iterativa, de acuerdo con los valores de los valores de entrenamiento, con el objeto de minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada.

- **Fase de Prueba**: en la fase anterior, el modelo puede que se ajuste demasiado a las particularidades presentes en los patrones de entrenamiento, perdiendo su habilidad de generalizar su aprendizaje a casos nuevos (sobreajuste).

  Para evitar el problema del sobreajuste, es aconsejable utilizar un segundo grupo de datos diferentes a los de entrenamiento, el grupo de *validación*, que permita controlar el proceso de aprendizaje.

Normalmente, los pesos óptimos se obtienen optimizando (minimizando) alguna función de *energía*. Por ejemplo, un criterio muy utilizado en el llamado entrenamiento supervisado, es minimizar el *error cuadrático medio* entre el valor de salida y el valor real esperado.

Algoritmos de aprendizaje más conocidos

Paradigma	Regla de aprendizaje	Arquitectura	Algoritmo de aprendizaje	Tareas
Supervisado	Corrección del error	Perceptrón o perceptrón multicapa	Algoritmos de aprendizaje perceptrón, retropropagación del error, ADALINE, MADALINE	Clasificación de patrones, aproximación de funciones, predicción, control, ...
		Elman y Jordan recurrentes	Retropropagación del error	Síntesis de series temporales
	Boltzmann	Recurrente	Algoritmo de aprendizaje Boltzmann	Clasificación de patrones
	Competitivo	Competitivo	LVQ	Categorización intra-clase, compresión de datos
		Red ART	ARTMap	Clasificación de patrones, categorización intra-clase
No supervisado	Corrección del error	Red de Hopfield	Aprendizaje de memoria asociativa	Memoria asociativa
		Multicapa sin realimentación	Proyección de Sannon	Análisis de datos
	Competitiva	Competitiva	VQ	Categorización, compresión de datos
		SOM	Kohonen SOM	Categorización, análisis de datos
		Redes ART	ART1, ART2	Categorización
Por refuerzo	Hebbian	Multicapa sin realimentación	Análisis lineal de discriminante	Análisis de datos, clasificación de patrones
		Sin realimentación o competitiva	Análisis de componentes principales	Análisis de datos, compresión de datos

Redes Neuronales Supervisadas y No Supervisadas

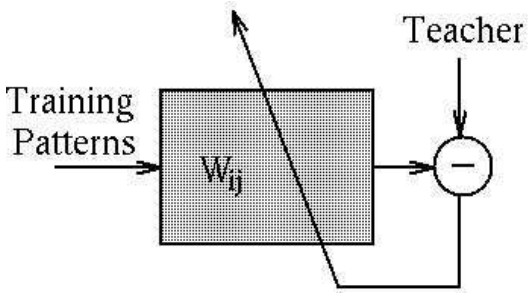
Las redes neuronales se clasifican comúnmente en términos de sus correspondientes algoritmos o métodos de entrenamiento: redes de pesos fijos, redes no supervisadas, y redes de entrenamiento supervisado. Para las redes de pesos fijos no existe ningún tipo de entrenamiento.

Reglas de entrenamiento Supervisado

Las redes neuronales de entrenamiento supervisado son las más populares. Los datos para el entrenamiento están constituidos por varios pares de patrones de entrenamiento de entrada y de salida. El hecho de conocer la salida implica que el entrenamiento se beneficia de la supervisión de un *maestro*. Dado un nuevo patrón de entrenamiento, en la etapa $(m+1)$-ésima, los pesos se adaptan de la siguiente forma:

$$w_{ij}^{m+1} = w_{ij}^{m} + \Delta w_{ij}^{m}$$

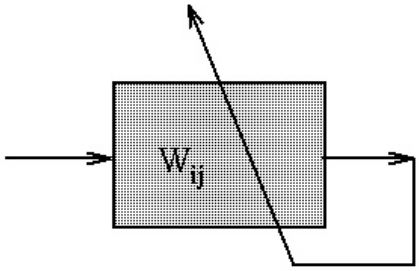
Se puede ver un diagrama esquemático de un sistema de entrenamiento supervisado en la siguiente figura:



Reglas de Entrenamiento No Supervisado

Para los modelos de entrenamiento no supervisado, el conjunto de datos de entrenamiento consiste sólo en los patrones de entrada. Por lo tanto, la red es entrenada sin el beneficio de un maestro. La red aprende a adaptarse basada en las experiencias recogidas

de los patrones de entrenamiento anteriores. Un esquema típico de un sistema *No Supervisado*:



Ejemplos típicos son la Regla de Aprendizaje de *Hebb*, y la Regla de Aprendizaje Competitivo. Un ejemplo del primero consiste en reforzar el peso que conecta dos nodos que se excitan simultáneamente.

En el aprendizaje competitivo, si un patrón nuevo pertenece a una clase reconocida previamente, entonces la inclusión de este nuevo patrón a esta clase matizará la representación de la misma. Si el nuevo patrón no pertenece a ninguna de las clases reconocidas anteriormente, entonces la estructura y los pesos de la red neuronal serán ajustados para reconocer a la nueva clase.

Funciones de Base y Activación

Una red neuronal típica se puede caracterizar por la función de base y la función de activación.

Cada nodo (unidad de proceso), suministra un valor y_j a su salida. Este valor se propaga a través de la red mediante conexiones unidireccionales hacia otros nodos de la red. Asociada a cada conexión hay un peso sináptico denominado $\{w_{ij}\}$, que determina el efecto del nodo j-ésimo sobre el nodo i-ésimo.

Las entradas al nodo i-ésimo que provienen de las otros nodos son acumulados junto con el valor umbral θ_i, y se aplica la función base f, obteniendo u_i. La salida final y_i se obtiene aplicando la función de activación sobre u_i.

Función Base (Función de Red)

La función de base tiene dos formas típicas:

- Función lineal de tipo *hiperplano*: El valor de red es una combinación lineal de las entradas,

$$u_i(w, x) = \sum_{j=1}^{n} w_{ij} x_j$$

- Función radial de tipo *hiperesférico*: es una función de base de segundo orden no lineal. El valor de red representa la distancia a un determinado patrón de referencia,

$$u_i(w, x) = \sqrt{\sum_{j=1}^{n} (x_j - w_{ij})^2}$$

Función de Activación (Función de neurona)

El valor de red, expresado por la función de base, $u(w, x)$, se transforma mediante una función de activación no lineal. Las funciones de activación más comunes son la función sigmoidal y gausiana:

- Función sigmoidal

$$f(u_i) = \frac{1}{1 + \exp\{-\frac{u_i}{\sigma^2}\}}$$

- Función gausiana

$$f(u_i) = c \exp\{-\frac{u_i^2}{\sigma^2}\}$$

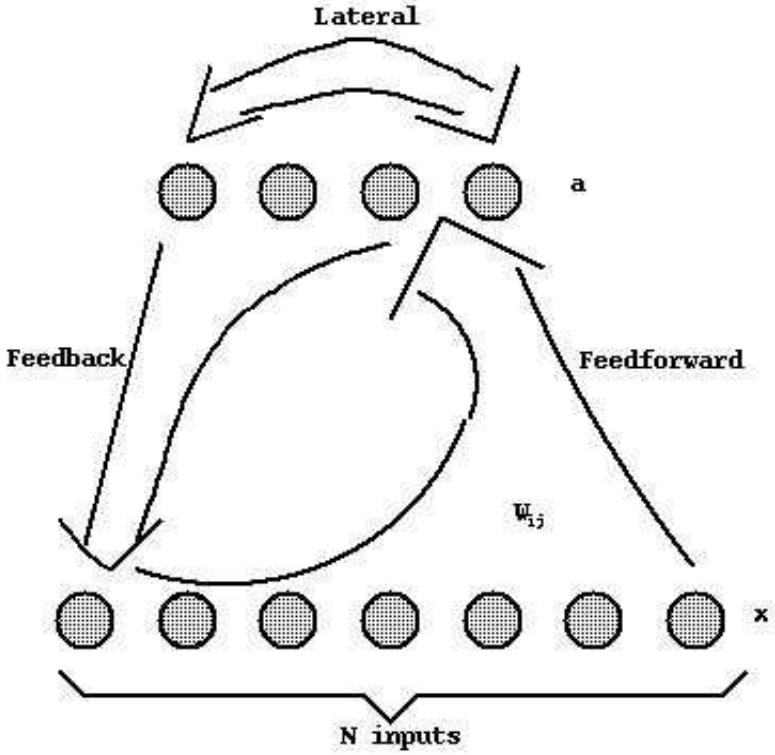
Estructuras de conexión de atrás hacia delante

Una red neuronal se determina por las neuronas y la matriz de pesos.

Hay tres tipos de capas de neuronas:

- la capa de entrada,
- la capa oculta y
- la capa de salida.

Entre dos capas de neuronas existe una red de pesos de conexión, que puede ser de los siguientes tipos:
hacia delante, hacia atrás, lateral y de retardo:



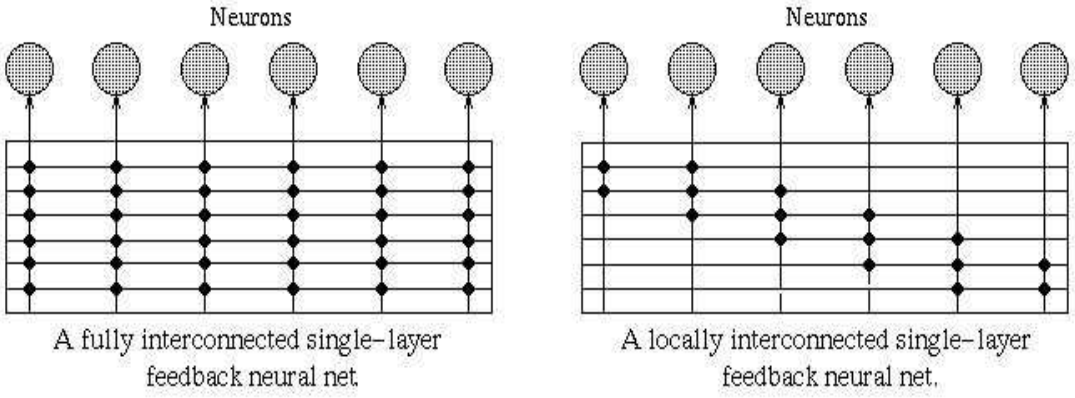
1. *Conexiones hacia delante*: los valores de las neuronas de una capa inferior son propagados hacia las neuronas de la capa superior por medio de las redes de conexiones hacia adelante.

2. *Conexiones hacia atrás*: estas conexiones llevan los valores de las neuronas de una capa superior a otras de la capa inferior.

3. *Conexiones laterales*: Un ejemplo típico de este tipo es el circuito "el ganador toma todo" (*winner-takes-all*), que cumple un papel importante en la elección del ganador: a la neurona de salida que da el valor más alto se le asigna el valor total (por ejemplo, 1), mientras que a todas las demás se le da un valor de 0

4. *Conexiones con retardo*: los elementos de retardo se incorporan en las conexiones para implementar modelos dinámicos y temporales, es decir, modelos que precisan de *memoria*.

Las conexiones sinápticas pueden ser total o parcialmente interconectadas, como se muestra la siguiente figura:



A fully interconnected single-layer feedback neural net.

A locally interconnected single-layer feedback neural net.

También es posible que las redes sean de una capa con el modelo de pesos hacia atrás o bien el modelo multicapa hacia adelante. Es posible así mismo, el conectar varias redes de una sola capa para dar lugar a redes más grandes.

Tamaño de las Redes Neuronales

En una red multicapa de propagación hacia delante, puede haber una o más capas ocultas entre las capas de entrada y salida. El tamaño de las redes depende del número de capas y del número de neuronas ocultas por capa.

El número de unidades ocultas está directamente relacionado con las capacidades de la red. Para que el comportamiento de la red sea correcto, se tiene que determinar apropiadamente el número de neuronas de la capa oculta.

Métodos de Regresión y Redes Neuronales

Los modelos de regresión estudian la relación entre una serie de variables, denominadas *independientes* x_i, y otras variables *dependientes* o *respuesta* que se denotan como y. El objetivo es predecir los valores de y en función de los valores de x_i.

El modelo básico se puede expresar como

$$\eta = \sum_{i=0}^{N} \beta_i x_i$$

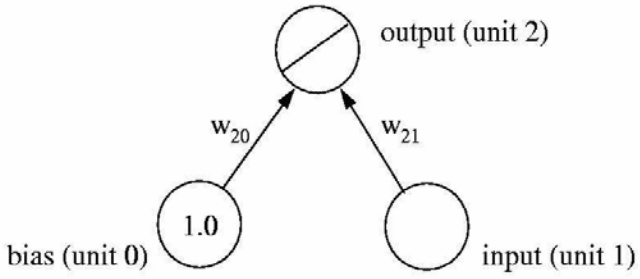
donde

$$\begin{aligned} \eta &= h(\mu) \\ E(y) &= \eta \end{aligned}$$

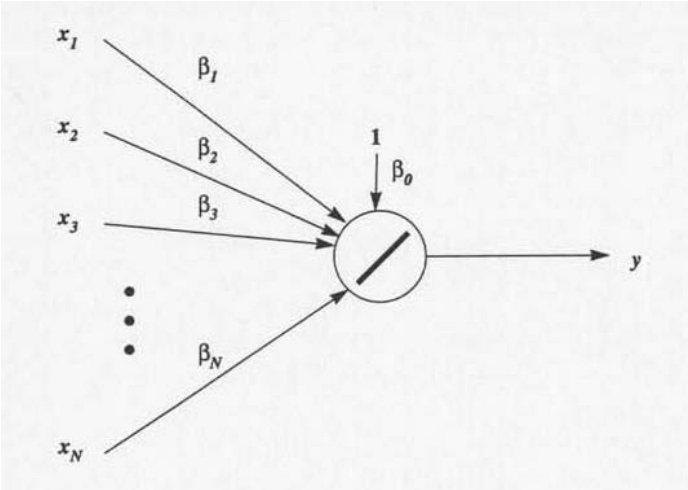
En estas expresiones $h(\cdot)$ es la función que liga los componentes, β_i son los coeficientes, N es el número de variables independientes y β_0 es la pendiente.

Un modelo lineal se puede implementar como una red neuronal simple: tiene una unidad de sesgo, una unidad de input y una unidad de salida. El input se refiere a una variable x, mientras que el sesgo siempre es una constante igual a 1. El output sería

$$y_2 = y_1 w_{21} + 1{,}0 w_{20}$$



Gráficamente se puede representar como



El modelo tiene tres componentes:

1. Un componente aleatorio de la variable respuesta y con media μ y varianza σ^2.

2. Un componente que relaciona las variables independientes x_i con un predictor lineal $\eta = \sum_{i=0}^{N} \beta_i x_i$.

3. Una función que relaciona la media con el predictor lineal $\eta = h(\mu)$.

El modelo lineal generalizado se reduce al modelo de regresión lineal múltiple si se considera que el componente aleatorio se distribuye como una normal y la función $h(\cdot)$ se asume que es la identidad. El modelo queda, entonces, como

$$y_p = \beta_0 + \sum_{i=1}^{N} \beta_i x_{pi} + \varepsilon_p$$

donde $\varepsilon_p \sim N(0, \sigma^2)$.

El objetivo es encontrar los coeficientes β_i que minimizan la suma de cuadrados de los errores

$$E = \sum_{p=1}^{n} \left(y_p - \sum_{i=0}^{N} \beta_i x_{pi} \right)^2 .$$

Este problema es equivalente al recogido por una red neuronal con una sola capa, donde los parámetros β_i equivalen a los pesos de la misma y la función de activación es la identidad.

El Perceptrón Multicapa

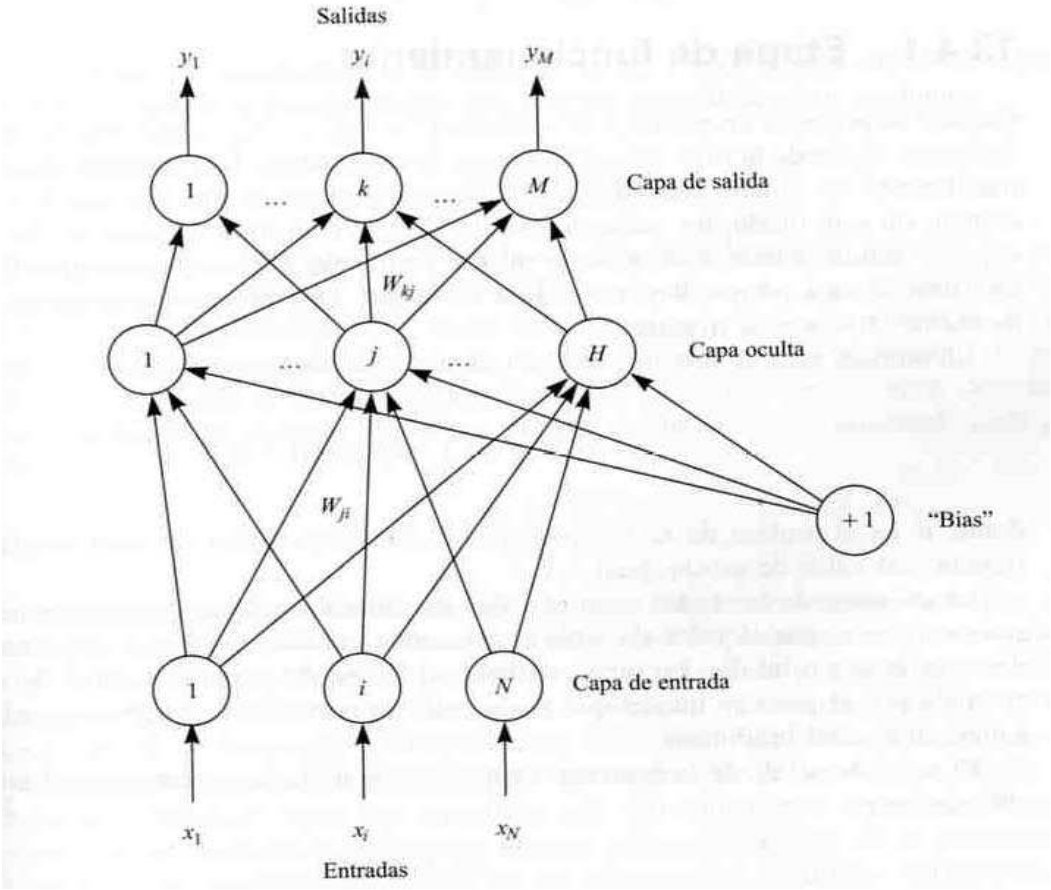
Es capaz de actuar como un aproximador universal de funciones: una red *backpropagation* conteniendo al menos una capa oculta con suficientes unidades no lineales puede aproximar cualquier tipo de función o relación continua entre un grupo de variables de entrada y salida. Esta propiedad convierte a las redes perceptrón multicapa en herramientas de propósito general, flexibles y no lineales.

Rumelhart et al. (1986) formalizaron un método para que una red del tipo perceptrón multicapa aprendiera la asociación que existe entre un conjunto de patrones de entrada y sus salidas correspondientes: método *backpropagation error* (propagación del error hacia atrás).

Esta red tiene la capacidad de *generalización*: facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento.

Arquitectura

Un perceptrón multicapa está compuesto por una capa de entrada, una capa de salida y una o más capas ocultas; aunque se ha demostrado que para la mayoría de problemas bastará con una sola capa oculta. En la figura siguiente se puede observar un perceptrón típico formado por una capa de entrada, una capa oculta y una de salida.



Las conexiones entre neuronas son siempre hacia delante: las conexiones van desde las neuronas de una determinada capa hacia las neuronas de la siguiente capa; no hay conexiones laterales ni conexiones hacia atrás. Por tanto, la información siempre se transmite desde la capa de entrada hacia la capa de salida.

Se considera w_{ji} como el peso de conexión entre la neurona de entrada i y la neurona oculta j, y v_{kj} como el peso de conexión entre la neurona oculta j y la neurona de salida k.

Algoritmo *backpropagation*

Se considera una etapa de funcionamiento donde se presenta, ante la red entrenada, un patrón de entrada y éste se transmite a través de las sucesivas capas de neuronas hasta obtener una salida y, después, una etapa de entrenamiento o aprendizaje donde se

modifican los pesos de la red de manera que coincida la salida deseada por el usuario con la salida obtenida por la red.

Etapa de funcionamiento

Cuando se presenta un patrón p de entrada X^p: $x_1^p, \ldots, x_i^p, \ldots, x_N^p$, éste se transmite a través de los pesos w_{ji} desde la capa de entrada hacia la capa oculta. Las neuronas de esta capa intermedia transforman las señales recibidas mediante la aplicación de una función de activación proporcionando, de este modo, un valor de salida. Este se transmite a través de los pesos v_{kj} hacia la capa de salida, donde aplicando la misma operación que en el caso anterior, las neuronas de esta última capa proporcionan la salida de la red.

Este proceso se resume en lo siguiente:

La entrada total o neta que recibe una neurona oculta j, net_j^p, es:

$$net_j^p = \sum_{i=1}^{N} w_{ji} x_i^p + \theta_j$$

donde θ_j es el umbral de la neurona que se considera como un peso asociado a una neurona ficticia con valor de salida igual a 1.

El valor de salida de la neurona oculta j, y_j^p, se obtiene aplicando una función $f(\cdot)$ sobre su entrada neta:

$$y_j^p = f\left(net_j^p\right)$$

De igual forma, la entrada neta que recibe una neurona de salida k, net_k^p, es:

$$net_k^p = \sum_{j=1}^{H} v_{kj} y_j^p + \theta_k$$

Por último, el valor de salida de la neurona de salida k, y_k^p, es:

$$y_k^p = f\left(net_k^p\right)$$

Etapa de aprendizaje

En la etapa de aprendizaje, el objetivo es hacer mínimo el error entre la salida obtenida por la red y la salida deseada por el usuario ante la presentación de un conjunto de patrones denominado grupo de entrenamiento.

Así el aprendizaje en las redes *backpropagation* es de tipo supervisado.

La función de error que se pretende minimizar para cada patrón p viene dada por:

$$E^p = \frac{1}{2}\sum_{k=1}^{M}(d_k^p - y_k^p)^2$$

donde d_k^p es la salida deseada para la neurona de salida k ante la presentación del patrón p.

A partir de esta expresión se puede obtener una medida general de error mediante:

$$E = \sum_{p=1}^{P} E^p$$

La base del algoritmo *backpropagation* para la modificación de los pesos es la técnica conocida como gradiente decreciente.

Como E^p es función de todos los pesos de la red, el gradiente de E^p es un vector igual a la derivada parcial de E^p respecto a cada uno de los pesos. El gradiente toma la dirección que determina el incremento más rápido en el error, mientras que la dirección opuesta, es decir, la dirección negativa, determina el decremento más rápido en el error. Por tanto, el error puede reducirse ajustando cada peso en la dirección:

$$-\sum_{p=1}^{P}\frac{\partial E^p}{\partial w_{ji}}$$

Un peligro que puede surgir al utilizar el método de gradiente decreciente es que el aprendizaje converja a un mínimo local. Sin embargo, el problema potencial de los mínimos locales se da en raras ocasiones en datos reales.

A nivel práctico, la forma de modificar los pesos de forma iterativa consiste en aplicar la regla de la cadena a la expresión del gradiente y añadir una tasa de aprendizaje η. Así, en una neurona de salida:

$$\Delta v_{kj}(n+1) = -\eta\frac{\partial E^p}{\partial v_{kj}} = \eta\sum_{p=1}^{P}\delta_k^p y_j^p$$

donde

$$\delta_k^p = (d_k^p - y_k^p)\, f'(net_k^p)$$

y n indica la iteración.

En una neurona oculta:

$$\Delta w_{ji}\left(n+1\right)=\eta\sum_{p=1}^{P}\delta_{j}^{p}x_{i}^{p}$$

donde

$$\delta_{j}^{p}=f\left(net_{j}^{p}\right)\sum_{k=1}^{M}\delta_{k}^{p}v_{kj}$$

Se puede observar que el error o valor delta asociado a una neurona oculta j, viene determinado por la suma de los errores que se cometen en las k neuronas de salida que reciben como entrada la salida de esa neurona oculta j. De ahí que el algoritmo también se denomine *propagación del error hacia atrás.*

Para la modificación de los pesos, la actualización se realiza después de haber presentado todos los patrones de entrenamiento. Este es el modo habitual de proceder y se denomina aprendizaje por lotes o modo *batch.*

Otra modalidad denominada aprendizaje en serie o modo on line consistente en actualizar los pesos tras la presentación de cada patrón de entrenamiento. Ha de hacerse en orden aleatorio.

Para acelerar el proceso de convergencia de los pesos, Rumelhart et al. (1986) sugirieron añadir un factor momento, α, que tiene en cuenta la dirección del incremento tomada en la iteración anterior:

$$\Delta v_{kj}\left(n+1\right)=\eta\left(\sum_{p=1}^{P}\delta_{k}^{p}y_{j}^{p}\right)+\alpha\Delta v_{kj}\left(n\right)$$

Fases en la aplicación de un perceptrón multicapa

Una red del tipo perceptrón multicapa intenta resolver dos tipos de problemas:

– Problemas de predicción, que consisten en la estimación de una variable continua de salida, a partir de la presentación de un conjunto de variables predictoras de entrada (discretas y/o continuas).

– Problemas de clasificación, que consisten en la asignación de la categoría de pertenencia de un determinado patrón a partir de un conjunto de variables predictoras de entrada

(discretas y/o continuas).

Selección de las variables relevantes y preprocesamiento de los datos

Para obtener una aproximación funcional óptima, se deben elegir cuidadosamente las variables a emplear: se trata es de incluir en el modelo las variables predictoras que realmente predigan la variable dependiente o de salida, pero que a su vez no tengan relaciones entre sí ya que esto puede provocar un sobreajuste innecesario en el modelo.

Las variables deben seguir una distribución normal o uniforme, y el rango de posibles valores debe ser aproximadamente el mismo y acotado dentro del intervalo de trabajo de la función de activación empleada en las capas ocultas y de salida de la red neuronal.

Así, las variables de entrada y salida suelen acotarse en valores comprendidos entre 0 y 1 ó entre -1 y 1.

Si la variable es discreta, se utiliza la codificación *dummy*. Por ejemplo, la variable sexo podría codificarse como: 0 = hombre, 1 = mujer; estando representada por una única neurona. La variable nivel social podría codificarse como: 100 = bajo, 010 = medio, 001 = alto; estando representada por tres neuronas. Por su parte, si la variable es de naturaleza continua, ésta se representa mediante una sola neurona, como, por ejemplo, el CI de un sujeto.

Entrenamiento de la red neuronal

Elección de los pesos iniciales

Se hace una asignación de pesos pequeños generados de forma aleatoria, en un rango de valores entre $-0{,}5$ y 0,5 o algo similar.

Arquitectura de la red

Respecto a la arquitectura de la red, se sabe que para la mayoría de problemas prácticos bastará con utilizar una sola capa oculta .

El número de neuronas de la capa de entrada está determinado por el número de variables predictoras.

Así, en los ejemplos anteriores, la variable sexo estaría representada por una neurona

que recibiría los valores 0 ó 1. La variable status social estaría representada por tres neuronas. La variable puntuación en CI estaría representada por una neurona que recibiría la puntuación previamente acotada, por ejemplo, a valores entre 0 y 1.

El número de neuronas de la capa de salida está determinado bajo el mismo esquema que en el caso anterior. Cuando intentamos discriminar entre dos categorías, bastará con utilizar una única neurona (por ejemplo, salida 1 para la categoría A, salida 0 para la categoría B). Si estamos ante un problema de estimación, tendremos una única neurona que dará como salida el valor de la variable a estimar.

El número de neuronas ocultas determina la capacidad de aprendizaje de la red neuronal. Recordando el problema del sobreajuste, se debe usar el mínimo número de neuronas ocultas con las cuales la red rinda de forma adecuada. Esto se consigue evaluando el rendimiento de diferentes arquitecturas en función de los resultados obtenidos con el grupo de validación.

Tasa de aprendizaje y factor momento

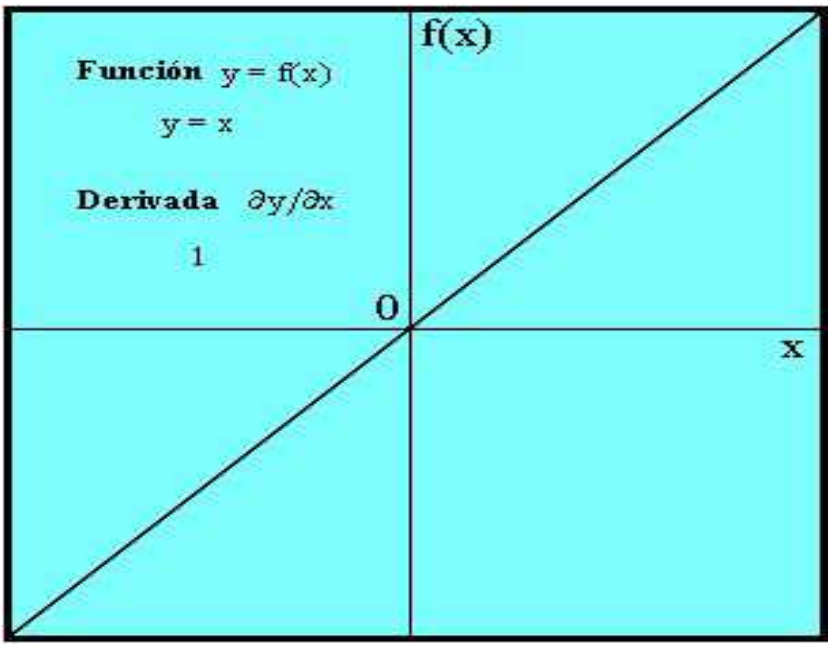
El valor de la tasa de aprendizaje (η) controla el tamaño del cambio de los pesos en cada iteración. Se deben evitar dos extremos: un ritmo de aprendizaje demasiado pequeño puede ocasionar una disminución importante en la velocidad de convergencia y la posibilidad de acabar atrapado en un mínimo local; en cambio, un ritmo de aprendizaje demasiado grande puede conducir a inestabilidades en la función de error, lo cual evitará que se produzca la convergencia debido a que se darán saltos en torno al mínimo sin alcanzarlo.

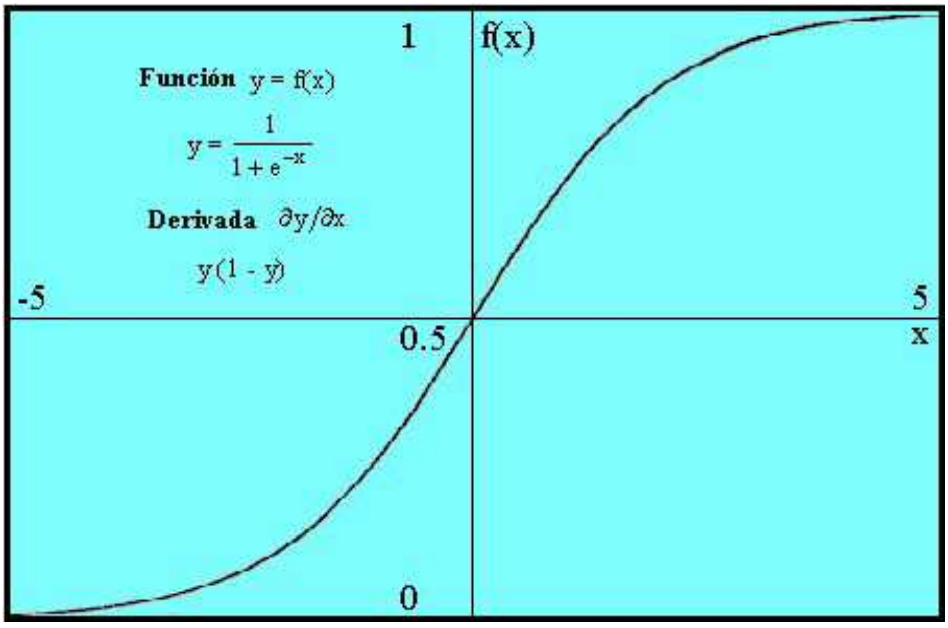
Por tanto, se recomienda elegir un ritmo de aprendizaje lo más grande posible sin que provoque grandes oscilaciones. En general, el valor de la tasa de aprendizaje suele estar comprendida entre 0.05 y 0.5.

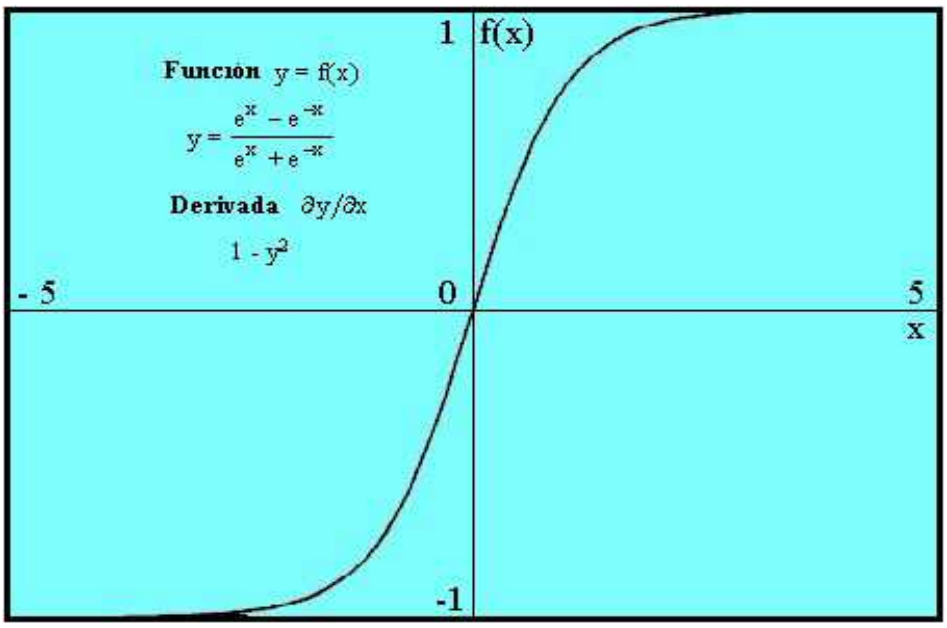
El factor momento (α) acelera la convergencia de los pesos. Suele tomar un valor próximo a 1 (por ejemplo, 0.9).

Función de activación de las neuronas ocultas y de salida

Hay dos formas básicas que cumplen esta condición: la función lineal (o identidad) y la función sigmoidal (logística o tangente hiperbólica).







Para aprovechar la capacidad de las NN de aprender relaciones complejas o no lineales entre variables, se recomienda la utilización de funciones no lineales al menos en las neuronas de la capa oculta. Por tanto, en general se utilizará una función sigmoidal (logística o tangente hiperbólica) como función de activación en las neuronas de la capa oculta.

La elección de la función de activación en las neuronas de la capa de salida dependerá del tipo de tarea impuesto.

En tareas de clasificación, las neuronas normalmente toman la función de activación sigmoidal. En cambio, en tareas de predicción o aproximación de una función, generalmente las neuronas toman la función de activación lineal.

Evaluación del rendimiento del modelo

Una vez seleccionado el modelo de red cuya configuración de parámetros ha obtenido la mejor ejecución ante el conjunto de validación, debemos evaluar la capacidad de generalización de la red de una forma completamente objetiva a partir de un tercer grupo de datos independiente, el conjunto de *test*.

Cuando se trata de la estimar una función, normalmente se utiliza la media cuadrática del error para evaluar la ejecución del modelo:

$$MC_{error} = \frac{\sum_{p=1}^{P} \sum_{k=1}^{M} (d_k^p - y_k^p)^2}{P \cdot M}$$

En problemas de clasificación de patrones es mejor la frecuencia de clasificaciones correctas e incorrectas. Se puede construir una tabla de confusión y calcular diferentes índices de asociación y acuerdo entre el criterio y la decisión tomada por la red neuronal.

Interpretación de los pesos obtenidos

Se trata de interpretar los pesos de la red neuronal. El método más popular es el *análisis de sensibilidad*.

El análisis de sensibilidad está basado en la medición del efecto que se observa en una salida y_k debido al cambio que se produce en una entrada x_i. Cuanto mayor efecto se observe sobre la salida, mayor sensibilidad se puede deducir que presenta respecto a la entrada.

Un método común consiste en fijar el valor de todas las variables de entrada a su valor medio e ir variando el valor de una de ellas a lo largo de todo su rango, registrando el valor de salida de la red.

Redes Neuronales como generalización de las técnicas de Análisis Discriminante

En este apartado se trata la visión de las redes neuronales en términos de una generalización de las funciones lineales empleadas en Análisis Discriminante.

En éste se clasifica un objeto como resultado de una aplicación lineal, cuyos parámetros se estiman a partir de un proceso de optimización.

Se puede generalizar el concepto anterior, considerando diferentes formas de la función discriminante ϕ. De modo específico, se considera una función de la forma

$$g_j(\mathbf{x}) = \sum_{i=1}^{m} w_{ji} \phi_i \left(x; \mu_i\right) + w_{j0},$$

$j = 1, \ldots, C$;

donde hay m funciones base, ϕ_i, cada una de las cuales tiene una serie de parámetros, y se utiliza la siguiente regla discriminante, dado un nuevo elemento $\mathbf{x}$:

$$\text{Asignar } \mathbf{x} \text{ a la clase } \omega_i \text{ si } g_i(\mathbf{x}) = \max_j g_j(\mathbf{x}),$$

esto es, $\mathbf{x}$ se asigna a la clase cuya función discriminante alcanza mayor valor.

La ecuación anterior generaliza el Análisis Discriminante lineal, usando funciones no lineales ϕ_i.

Si,

- $\phi_i(x; \mu_i) \equiv (x)_i$, esto es, una función lineal discriminante $m = p$, donde p es la dimensión de $\mathbf{x}$.
- $\phi_i(x; \mu_i) \equiv \phi_i(x)_i$ esto es, una función generalizada discriminante que puede ser no lineal.

Se puede interpretar como una transformación de los datos $x \in \mathbb{R}^p$ a $\mathbb{R}^C$ mediante un espacio *intermediario* $\mathbb{R}^m$ determinado por las funciones ϕ_i.

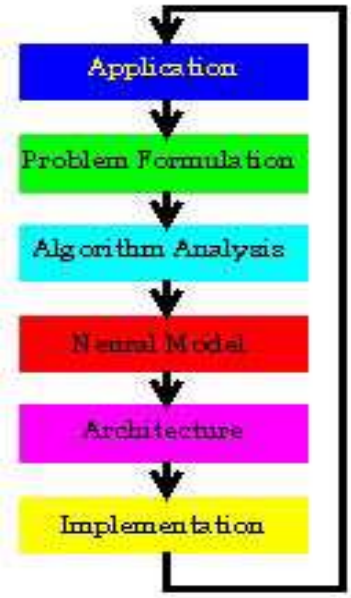
Ejemplo: La llamada función de base radial (RBF) se puede describir como una combinación lineal de funciones de base no lineales radialmente simétricas

$$g_j(\mathbf{x}) = \sum_{i=1}^{m} w_{ji} \phi_i(|x - \mu_i|) + w_{j0}$$

donde $j = 1, \ldots n$. Los parámetros w_{ji} son los *pesos*; w_{j0} se denomina el *sesgo* y a los vectores μ_i se les denomina los *centros*.

Aplicaciones

Con el fin de llegar al entendimiento global de NN, se suele adoptar la siguiente perspectiva, llamada *top-down*, que empieza por la aplicación, después se pasa al algoritmo y de aquí a la arquitectura:



Las principales aplicaciones son para el procesado de señales y el reconocimiento de patrones. Así, la ventaja de las NN reside en el procesado paralelo, adaptativo y no lineal. Las NN han encontrado muchas aplicaciones con éxito en la visión artificial, en el procesado de señales e imágenes, reconocimiento del habla y de caracteres, sistemas expertos, análisis de imágenes médicas, control remoto, control de robots e inspección industrial.

Las aplicaciones de las NN se puede clasificar de la siguiente forma: asociación y clasificación, regeneración de patrones, regresión y generalización, y optimización.

Asociación y Clasificación

En esta aplicación, los patrones de entrada estáticos o señales temporales deben ser clasificadas o reconocidas. Idealmente, un clasificador debería ser entrenado para que cuando se le presente una versión distorsionada ligeramente del patrón, pueda ser reconocida correctamente sin problemas. De la misma forma, la red debería presentar cierta inmunidad contra el ruido, esto es, debería ser capaz de recuperar una señal limpia de ambientes o canales ruidosos.

- *Asociación.* De especial interés son las dos clases de asociación autoasociación y heteroasociación. El problema de la autoasociación es recuperar un patrón enteramente, dada una información parcial del patrón deseado. La heteroasociación es recuperar un conjunto de patrones B, dado un patrón de ese conjunto. Los pesos en las redes asociativas son a menudo predeterminados basados en la llamada regla de *Hebb*: cuando una neurona participa constantemente en activar una neurona de salida, la influencia de la neurona de entrada es aumentada. Normalmente, la autocorrelación del conjunto de patrones almacenado determina los pesos en las redes autoasociativas. Por otro lado, la correlación cruzada de muchas parejas de patrones se usa para determinar los pesos de la red de heteroasociación.

- *Clasificación.* En muchas aplicaciones de clasificación, por ejemplo en reconocimiento de voz, los datos de entrenamiento consisten en pares de patrones de entrada y salida. En este caso, es conveniente adoptar las redes supervisadas, como las redes de *retropropagación.* Este tipo de redes son apropiadas para las aplicaciones que tienen una gran cantidad de clases con límites de separación complejos.

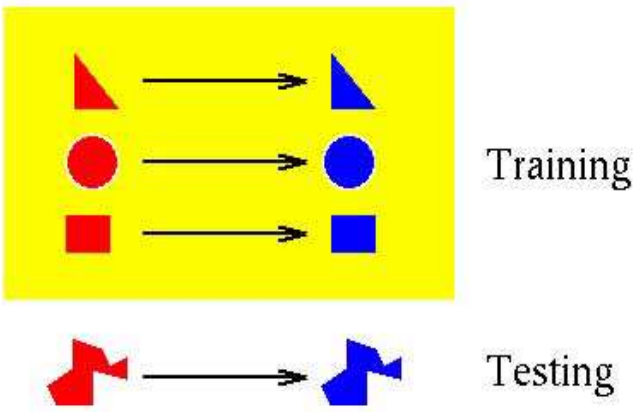
  En muchos problemas de clasificación una cuestión a solucionar es la recuperación la información, esto es, recuperar el patrón original cuando se dispone de sólo una información parcial.

Generalización

Se puede extender a un problema de interpolación. El sistema es entrenado por un gran conjunto de muestras de entrenamiento basados en un procedimiento de aprendizaje supervisado.

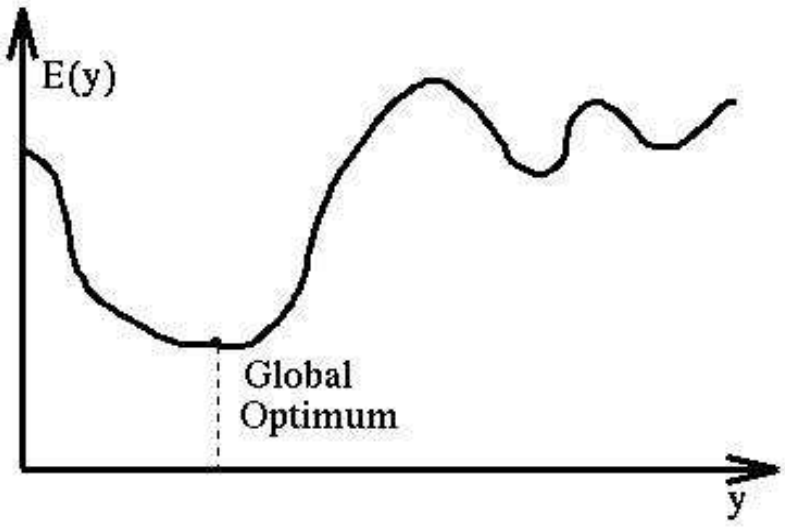
Una red se considera que esta entrenada con éxito si puede aproximar los valores de los patrones de entrenamiento y puede dar interpolaciones suaves para el espacio de datos no entrenado. El objetivo de la generalización es dar una respuesta correcta a la salida para un estímulo de entrada que no ha sido entrenado con anterioridad. El sistema debe inducir la característica saliente del estímulo a la entrada y detectar la regularidad. Tal habilidad

para el descubrimiento de esa regularidad es crítica en muchas aplicaciones. Esto hace que el sistema funcione eficazmente en todo el espacio, incluso ha sido entrenado por un conjunto limitado de ejemplos. Por ejemplo, en la siguiente figura:



Optimización

Las NN son una herramienta interesante para la optimización de aplicaciones, que normalmente implican la búsqueda del mínimo absoluto de una función de energía.



Una vez que se define la función de energía, se determinan los pesos sinápticos. Para algunas aplicaciones, la función de energía es fácilmente deducible. En otras, sin embargo, esta función de energía se obtiene de ciertos criterios de coste y limitaciones especiales. El mayor problema asociado al problema de optimización es la alta posibilidad de converger hacia un mínimo local, en vez de hacia el mínimo absoluto. Para evitar este problema se utilizan algunas técnicas estadísticas como, por ejemplo, procedimientos estocásticos.

Páginas sobre Redes Neuronales

Tutorial sobre Redes Neuronales en Castellano

`http://www.bibliopsiquis.com/psicologiacom/vol5num2/2833/`

An Introduction to Neural Networks

`http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html`

CS-449: Neural Networks

`http://www.willamette.edu/%7Egorr/classes/cs449/intro.html`

Neural Computing Publications Worldwide

`http://www.ewh.ieee.org/tc/nnc/research/nnpubs.html`

Tutor in Neural Nets

`http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html`

Neural Nets de MatLab

`http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/`

Stuttgart Neural Network Simulator

`http://www-ra.informatik.uni-tuebingen.de/SNNS/`

Redes Neuronales con R

En R para estudiar redes neuronales con una sola capa se tiene la librería ***nnet***.
El formato de la función principal es

```
nnet(formula, data, weights, size, Wts, linout=F, entropy=F, softmax=F, skip=F,
rang=0.7, decay=0, maxit=100, trace=T)
```

Para estudiar sus opciones escribe:

```
help(nnet)
```

Ejemplo (procede del libro *Modern Applied Statistics with S* (4ª Edición), de W.N. Venables y B.D. Ripley. Springer. ISBN 0-387-95457-0 (2002):

```
library(nnet)
attach(rock)
area1 <- area/10000
peri1 <- peri/10000
rock1 <- data.frame(perm, area=area1, peri=peri1, shape)
rock.nn <- nnet(log(perm)~area+peri+shape, rock1, size=3, decay=1e-3, linout=T,
skip=T, maxit=1000, Hess=T)
summary(rock.nn)
```

Se obtiene una red neuronal 3-3-1 con 19 pesos.
En la salida de *summary* la letra *b* se refiere al sesgo (input 0) y, por otro lado *i*, *h*, *o* son los nodos de *input*, *hidden* y *output*.

```
sum((log(perm)-predict(rock.nn))^2)
detach()
eigen(rock.nn$Hessian,T)$values
```

Se pueden obtener distintas soluciones cuando se toma un punto inicial diferente.
Para obtener la superficie predicha para los datos:

```
library(lattice)
Xp <-expand.grid(area=seq(0.1,1.2,0.05), peri=seq(0,0.5,0.02), shape=0.2)
trellis.device()
rock.grid <- cbind(Xp, fit=predict(rock.nn,Xp))
wireframe(fit~area+peri, rock.grid, screen=list(z=160,x=-60), aspect=c(1,0.5),
drape=T)
```

Ejemplo con la librería ***neural*** que ofrece una interfaz gráfica.

```
# Ejemplo de interfaz grafica de redes neuronales
library(neural)

# Entreno una red de base radial
neurons <- 16
data <- rbftrain(x, neurons, y, sigma=NaN)
rbf(x, data$weigth, data$dist, data$neurons, data$sigma)

# Genero un conjunto de datos artificial
x <- matrix(c(1,1,0,0,1,0,1,0), 4, 2)
y <- matrix(c(0,1,1,0), 4, 1)

# Entreno un perceptron multicapa
neurons <- 4
data <- mlptrain(x, neurons, y, it=1000);
mlp(x, data$weigth, data$dist, data$neurons, data$actfns)
```

Ejemplo con los datos clásicos de Fisher con las flores del género Iris

```
library(nnet)
data(iris)

# Flores del genero Iris
table(iris$Species)
iS <- iris$Species=="setosa"
iV <- iris$Species=="versicolor"

matplot(c(1, 8), c(0, 4.5), type="n", xlab="Longitud", ylab="Anchura",
main="Dimensiones de pétalos y sépalos en Iris")
matpoints(iris[iS,c(1,3)], iris[iS,c(2,4)], pch="sS", col=c(2,4))
matpoints(iris[iV,c(1,3)], iris[iV,c(2,4)], pch="vV", col=c(2,4))
legend(1,4,c("      Pétalos Setosa", "     Sépalos Setosa",
" Pétalos Versicolor", "Sépalos Versicolor"), pch="sSvV", col=rep(c(2,4),2))

# Uso la mitad del fichero de datos de Iris
ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])
targets <- class.ind(c(rep("s",50), rep("c",50), rep("v",50)))
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
ir.nn1 <- nnet(ir[samp,], targets[samp,], size=2, rang=0.1,
         decay=5e-4, maxit=200)
names(ir.nn1)
ir.nn1$wts
ir.nn1$fitted.values
test.cl <- function(true, pred) {
    true <- max.col(true)
    cres <- max.col(pred)
    table(true, cres)
}
test.cl(targets[-samp,], predict(ir.nn1, ir[-samp,]))

# O bien
ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
     species=c(rep("s",50), rep("c",50), rep("v",50)))
ir.nn2 <- nnet(species~., data=ird, subset=samp, size=2,
         rang=0.1, decay=5e-4, maxit=200)

names(ir.nn2)
ir.nn2$wts
ir.nn2$fitted.values

table(ird$species[-samp], predict(ir.nn2, ird[-samp,], type = "class"))
```

Ejemplo con la librería ***RWeka***

```
library(RWeka)

NB <- make_Weka_classifier("weka/classifiers/functions/MultilayerPerceptron")
NB

WOW(NB)


# Genero un conjunto de datos artificial
x <- t(matrix(-5:10*24,1,160))
y <- t(matrix(sin(pi/180*(-5:10*24)),1,160))
plot(x,y,type="l",col="blue")

datos <- as.data.frame(cbind(x,y))
train <- sample(100)
```

```
nnodos <- "4"
modelo <- NB(y~x, data=datos, subset=train,
control=Weka_control(H=nnodos,N=1000,G=TRUE), options=NULL)

print(modelo)

predigo <- predict(modelo,datos)
cbind(datos[-train,1:2], predigo[-train])

par(mfrow=c(2,1))
plot(datos[-train,1], datos[-train,2],type="l",col="red")
plot(datos[-train,1], predigo[-train],type="l",col="blue")
```