Apéndice 4

REDES NEURONALES ARTIFICIALES. EJEMPLOS

José Antonio López Orozco

A4.1 INTRODUCCIÓN

A continuación se describen distintos ejemplos de aplicación de las redes neuronales mostradas en el Capítulo 4. Todos estos ejemplos se han realizado en código Matlab, proporcionándose las rutinas correspondientes. Se ha procurado que no sea necesario disponer de ninguna *toolbox* específica de Matlab para la ejecución de los ejemplos mostrados, por ello todas las rutinas han sido implementadas utilizando exclusivamente funciones que existen el núcleo de Matlab básico. Además pueden utilizarse en cualquier versión de Matlab superior a la versión 5.3.

A lo largo de este apéndice se muestra con detalle cómo reproducir los ejemplos incluidos en el Capítulo 4 y algunos otros que permitan destacar distintos aspectos de las redes tratadas. Por supuesto el lector es libre de modificar cualquier parámetro para observar los efectos que provocan en el aprendizaje.

Además, se ha separado el archivo demostración de las funciones específicas de las redes, de modo que puedan ser utilizadas por el lector en otras aplicaciones propias que él mismo pueda desarrollar.

Recuerde que para ejecutar cualquier función o ejemplo deberá estar situado en el directorio correspondiente donde se ubica el fichero que contiene la función.

A4.2 ESTRUCTURA DE LOS ARCHIVOS DE MATLAB

Para facilitar la comprensión de los ejemplos, cada tipo de red, perceptrón simple, red feedforward o mapas autoorganizativos, se encuentra en un directorio diferente y todos utilizan la misma metodología tanto en la nomenclatura de los nombres de los archivos como en el código interno:

- Un archivo *demo_red.m*, donde la palabra "*red*" viene sustituida por el tipo de red que en cada caso se trata, por ejemplo *demo_perceptron*, *demo_adaline*, etc. Se ocupa de cargar los archivos de datos necesarios y ejecutar el ejemplo que el usuario haya seleccionado dependiendo de la opción marcada. Estos archivos si se editan muestran cómo utilizar las funciones creadas para manejar las redes neuronales. En ellos se encuentran las variables necesarias que definen el problema a ejecutar y los parámetros para cada uno de los casos señalados.

- Los archivos con las funciones para manejar las redes neuronales. Se encargan de entrenar la red, calcular la función de activación, reproducir la red ya entrenada y contienen todas aquellas funciones necesarias para el manejo de la red correspondiente. Se ha procurado que estas funciones tengan una estructura lo más general posible para que puedan ser utilizadas en otras aplicaciones que el lector desee desarrollar. Las principales son (recuerde que "*Red"* debe sustituirse por el nombre de la red que se esté utilizando):

    - *Red_train.m*, encargada de entrenar la red y calcular los pesos necesarios para encontrar la solución a los datos cargados.

    - *Red.m*, es la que ejecuta la red para una entrada y pesos dados.

A4.2.1 Archivos de demostración

Estos archivos se denominan *demo_red.m*, siendo cada uno de ellos un *script* de Matlab que tiene por finalidad facilitar la comprensión al usuario de cómo trabajar con las redes desarrolladas. Al ser un script, todas las variables de entrada/salida y los parámetros más importantes de las redes quedan en memoria del computador después de su ejecución para que puedan ser examinadas

detenidamente. Debe tenerse en cuenta esto si se cambia de ejemplo, puesto que en algunos casos si no se realiza un borrado de la memoria mediante la función *clear* antes de ejecutar el nuevo ejemplo podrían aparecer errores al utilizar tamaños de matrices diferentes.

Los mencionados *scripts* se ocupan de ejecutar los distintos ejemplos. Para cada ejemplo escogido, cargan los datos necesarios, inicializan las variables requeridas en cada caso y definen los parámetros necesarios para ejecutar el entrenamiento de la red.

A continuación llaman a la rutina de entrenamiento de la red y finalmente muestran los resultados obtenidos. De este modo no sólo sirve de ejemplo de uso de las funciones de entrenamiento de cada una de las redes creadas, sino también como ejemplo de cómo mostrar o tratar los resultados obtenidos. No se incluyen todos los casos posibles, pero sí los suficientes para comprender el funcionamiento de todas las funciones incluidas en el directorio.

Para utilizar estas funciones primero deberían editarse (utilice desde la línea de comandos de Matlab *edit demo_red*) para escoger el ejemplo que se desee ejecutar (también se puede cambiar el valor de cualquiera de los parámetros disponibles) y después ejecutarla (escribiendo *demo_red*) desde la línea de comandos de Matlab.

ESTRUCTURA DEL CÓDIGO

Todos los archivos *demo_red.m* tienen los mismos apartados:

- *Selección del ejemplo*: aquí se encuentra la variable definiendo el ejemplo que se quiere seleccionar. Para ejecutar un ejemplo concreto se debe poner un valor que está indicado en los comentarios en el propio código.

- *Preparación de los datos de entrada y parámetros de la red*: en esta sección, para cada ejemplo, se inicializan los datos que se utilizarán, o que se cargarán de un archivo, como entrada para el entrenamiento de la red. Además se definen todas las variables necesarias en cada ejemplo, así como los parámetros para el aprendizaje.

- *Entrenamiento de la red*: aquí se llama a la función que se ocupa de entrenar la red.

- *Representación de resultados*: en esta sección se muestra para cada uno de los ejemplos disponibles, cómo tratar los datos obtenidos en el aprendizaje, el uso de la red resultante después del entrenamiento y cómo representar gráficamente los resultados.

A4.2.2 Funciones de la red neuronal

En cada directorio se encuentran todas las funciones necesarias para ejecutar cada uno de los tipos de redes incluidas en el Capítulo 4. Estos directorios son independientes uno de otro, de modo que si es necesaria alguna función común se incluye en todos los directorios donde sea necesario.

Estas funciones se encargan de entrenar la red, calcular la función de activación, reproducir la red ya entrenada y todas aquellas funciones necesarias para el aprendizaje o uso de la red correspondiente. Todas estas funciones son lo más generales posible para que puedan ser utilizadas en otras aplicaciones desarrolladas por el lector. Las principales son:

- *Red_train.m*, encargada de entrenar la red y calcular los pesos necesarios para encontrar la solución a los datos cargados. Para una explicación detallada de cada uno de los parámetros de entrada utilice *help red_train*.

- *Red.m*, es la que ejecuta la red para una entrada y pesos dados. Permite mostrar el funcionamiento de la red después del aprendizaje. Utilice *help red.m* para obtener información sobre los parámetros que utiliza.

A4.3 PERCEPTRÓN

En el directorio *perceptron* se encuentran las rutinas necesarias para reproducir los ejemplos sobre el Perceptrón simple y la red Adaline. Además de los ejemplos mostrados en el Capítulo 4 se incluyen algunos otros que se describen a continuación.

A.4.3.1 Perceptrón simple

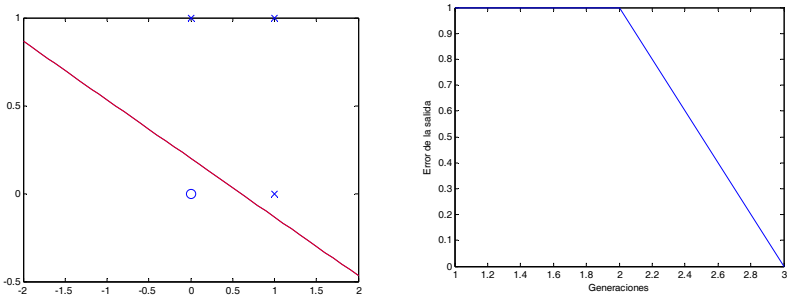
El ejemplo más sencillo de red neuronal es el Perceptrón simple. Éste se ha descrito en la sección 4.3.1. En dicha sección se han mostrado dos ejemplos sencillos: la función OR y la función AND, véase la tabla A4.1.

x1	x2	OR
0	0	0
0	1	1
1	0	1
1	1	1

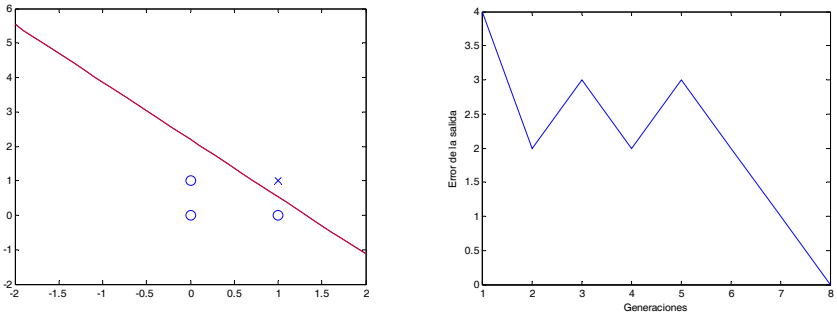
x1	x2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Tabla A4.1. Funciones OR y AND

En estos ejemplos se muestran tres gráficas: la recta final que separa los dos conjuntos junto con los datos de entrada (unos con "o" y otros con "x"); la evolución del error; y todas las rectas obtenidas en el proceso de aprendizaje. Esta última gráfica no se muestra en la figura A4.1 puesto que puede verse en la propia sección 4.3.1. Observése cómo el error va disminuyendo hasta llegar a cero. La oscilación es muy brusca porque el factor de aprendizaje es 1.



a) Ejemplo OR



a) Ejemplo AND

Figura A4.1. Salida de la red y evolución del error para los ejemplos OR y AND

El siguiente ejemplo es algo más complejo. Consiste en un conjunto de datos que se quieren clasificar en dos grupos. Se dispone de un archivo de datos de entrenamiento (*entrena1.txt*) que se carga seleccionando el Ejemplo1 en el archivo *demo_perceptron.m*. Lo primero que debe observarse es que los datos de entrenamiento están clasificados como 1 y 2. Esto no puede utilizarse directamente como salida deseada puesto que sería imposible que la red diese esos valores y por lo tanto no alcanzaría nunca la convergencia. En el archivo *demo_perceptron.m* se carga el archivo *entrena1.txt* y se procesa la salida para que los datos se clasifiquen como 0 y 1 respectivamente. El motivo de clasificar los datos de esta forma es para recordar al lector que es muy importante el análisis y pre-procesamiento de los datos que se van a utilizar en el aprendizaje de la red neuronal.

El Perceptrón realmente no tiene posibilidad de variar la velocidad de aprendizaje, si bien lo hemos incluido para mostrar algunos efectos interesantes. Si se prueba con los valores de la velocidad de aprendizaje, α igual a 1 y 0.3 se observará que no se encuentra solución en 300 iteraciones. Sin embargo si utiliza α igual a 0.01, se obtiene el resultado mostrado en la figura A4.2. Esto es debido a que valores altos de la velocidad de aprendizaje provocan una gran oscilación alrededor de la solución correcta, por lo que es necesario un gran número de iteraciones para que consiga converger llegando a esa solución correcta. Sin embargo, al contrario de lo que podría parecer, con una velocidad menor se obtiene la solución con menos iteraciones ya que se evitan oscilaciones y se converge a la solución más directamente.

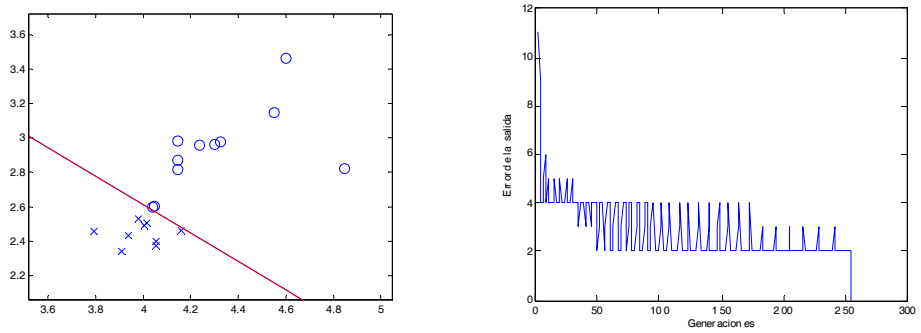


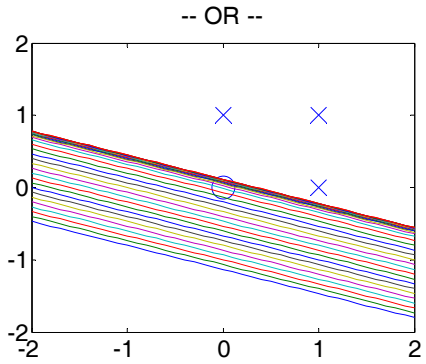
Figura A4.2 Solución del perceptron para clasificar un conjunto de puntos y evolución del error en el aprendizaje.

A.4.3.2 Adaline

La red *Adaline*, como se ha mostrado en la sección 4.3.2, minimiza el error existente aplicando el gradiente descendente, ecuación 4.21. Como la función de salida es lineal la superficie de error es suave en comparación con la del perceptrón, lo que permite que la técnica del gradiente sea aplicable.

Un efecto inmediato que se puede observar debido a que la salida es lineal, es que la red ya no clasifica al modo del perceptrón, dando el valor 0 si es de un tipo y 1 si es de otro, sino que intenta reproducir la salida que corresponda. Por lo tanto procurará aproximarse a la salida deseada hasta reducir el error al indicado. Observése, en la figura A4.3, que la salida para el problema de la función OR no es exactamente 1 y 0 y que las rectas de aprendizaje obtenidas intentan ajustarse para obtener un error marcado a pesar de que la solución se había encontrado mucho antes. Si se quiere clasificar con precisión lo más recomendable es utilizar después de la red Adaline una función escalón con un cierto umbral que sea capaz de distinguir el tipo de la salida.

Por último si resolvemos el problema de clasificación mostrado en la figura A4.2 para el caso de la red Adaline, observaremos que la solución es similar, pero, al igual que en el caso anterior, los datos no son clasificados como 0 o 1, sino que la salida se aproxima a estos valores. Este ajuste se produce más lentamente sobre una superficie de error sin los picos pronunciados, lo que da lugar, como se puede observar en la figura A4.4, a un descenso más suave. Téngase en cuenta que en el ejemplo se ha utilizado un función logística con al parámetro *a* igual a 18, lo que se asemeja mucho a un escalón, a la vez que mantiene la propiedad de derivabilidad. Si se toma un valor menor, la superficie de error es mucho más suave y por tanto la variación del error también.



x1	x2	OR	OR_red
0	0	0	0.0422
0	1	1	1.0000
1	0	1	0.9973
1	1	1	1.0000

Figura A4.3. Resultado de la red adaline y su salida final.

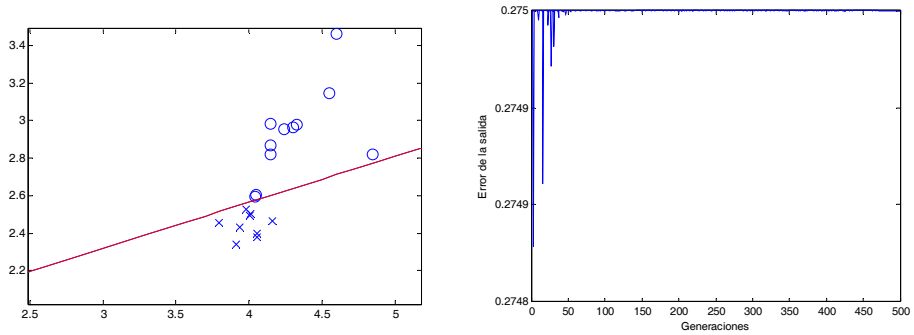


Figura A4.4. Aprendizaje de la red Adaline en la clasificación de un grupo de datos. Salida final y evolución del error.

A4.4 REDES HACIA DELANTE

En el directorio *BackProp* se encuentran las rutinas para reproducir los ejemplos relativos a una red de conexiones hacia delante o red *feedforward*. Se incluyen funciones para manejar redes feedforward de varias capas ocultas y se utiliza el método de retropropagación del error basado en el gradiente descendente como algoritmo de aprendizaje. Además se incluyen algunos otros ejemplos que se comentan a continuación.

En el ejemplo de aproximación de funciones de la sección 4.3.3.2.2 se señala que el conjunto de validación sirve para indicar cuándo se debe dejar de ajustar la red a los puntos obtenidos y así evitar que se aprenda el ruido contenido en los datos, es decir, se incurra en un sobreajuste. En la figura A4.5 se muestra el error cometido en el conjunto de entrenamiento y en el conjunto de validación durante el entrenamiento de la red. Como puede observarse, alrededor de la iteración 90 es cuando el error del conjunto de validación comienza a aumentar mientras que el error del conjunto de entrenamiento sigue disminuyendo. Esta divergencia marca el momento óptimo para detener el aprendizaje de la red.

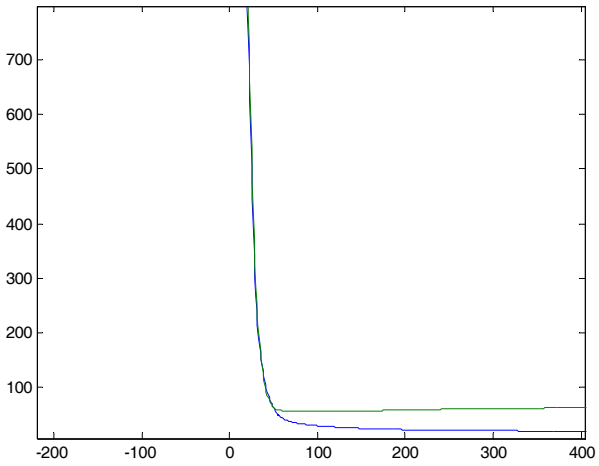


Figura A4.5. Error cometido en el conjunto de validación y entrenamiento.

En la rutina *bp_train.m* no se ha implementado un mecanismo automático para detener el aprendizaje cuando ocurre un sobreajuste, sino que la condición de parada es que se obtenga el error que ha marcado el usuario o se alcance el número de iteraciones señaladas. Esto permite que sea más sencillo para el lector observar el sobreajuste si se realizan excesivas iteraciones. Para evitar el sobreajuste, puede observar la evolución del error y ajustar convenientemente el número de iteraciones en una segunda ejecución o añadir en el código una parada automática cuando ocurra la divergencia entre el error en los datos de entrada y el error en los datos de validación.

Otro ejemplo de interés es el que muestra el uso de funciones distintas a las sigmoides, que son las utilizadas tradicionalmente. En el archivo *demo_bp.m* se incluye un ejemplo que utiliza funciones *seno* (y su derivada el *coseno*) como funciones de activación. Se han utilizado estas funciones de activación en el problema de ajuste de funciones porque estamos ajustando los puntos con suma de funciones base y como la función que deseamos ajustar en el ejemplo es periódica, el uso de funciones base periódicas resulta ser muy apropiado. El resultado utilizando funciones sigmoidales es bueno en la zona de ajuste pero en los dos extremos de la función se obtiene un resultado plano. El uso de funciones periódicas como base de funciones para realizar el ajuste dará como resultado una función periódica y esperamos que permita un ajuste mejor en las zonas no contempladas en el aprendizaje. En la figura A4.6 se muestra el ajuste realizado con funciones de activación seno para el mismo caso que el mostrado en la sección 4.3.3.2.2 utilizando los mismos pesos iniciales y el mismo número de iteraciones,

esto es 10000. Sin embargo, se han utilizado una velocidad de aprendizaje y un momento (α=0.0001, μ=0.009) muy pequeños. Esto es debido a que las funciones de activación deben ser siempre monótonas crecientes o decrecientes, pero en el caso del seno no es así y además de aumentar y disminuir su valor a medida que se aumenta la entrada, son periódicas. Esto provoca que el método del gradiente no se pueda aplicar si no trabajamos en una zona donde se cumpla esta restricción. Para ello podríamos saturar los extremos del seno entre 0 y 90º o utilizar un valor de α muy pequeño que impida salirnos de la zona de crecimiento o decrecimiento monótono. Si se utiliza un valor de α pequeño conseguimos esto y además evitamos que se pase a una segunda, tercera o mayor vuelta. Este efecto se observa por el crecimiento constante de los pesos en el caso de usar un valor de α mayor, y por supuesto sin obtener ningún ajuste correcto.

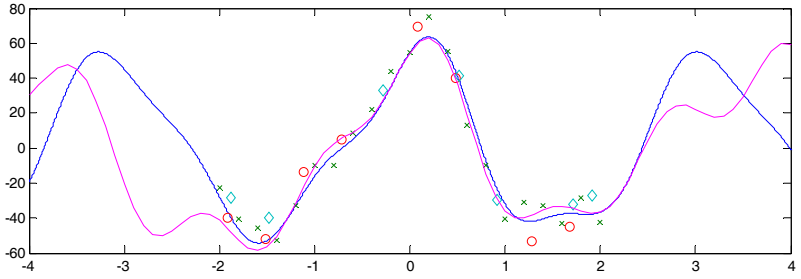


Figura A4.6 Ajuste de una función mediante funciones de activación seno.

Indicar que en la figura A4.6 no se ajustan correctamente las zonas no contempladas en el aprendizaje porque no se ha utilizado en el proceso de aprendizaje un periodo completo, en cuyo caso no se puede ajustar nada más que lo que resulta conocido. Incluso así, puede observarse cómo el error cometido es mucho menor que en el caso de utilizar funciones sigmoidales.

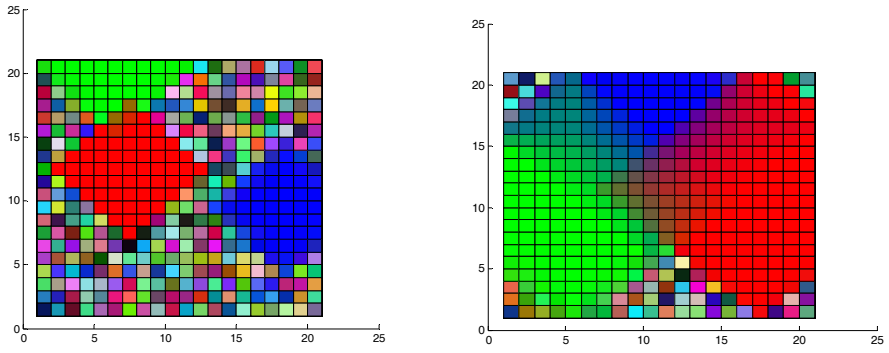
A4.5 MAPAS AUTO-ORGANIZADOS (SOM)

En el directorio denominado *SOM* se encuentran las funciones necesarias para reproducir los ejemplos mostrados en el Capítulo 4 sobre redes competitivas de este tipo. Además se incluye otro ejemplo que se comenta a continuación.

Los mapas autoorganizados son redes competitivas en las que sólo una neurona es la ganadora por cada patrón de entrada. La red es capaz de identificar la estructura de los datos y representarlos en la capa de salida. Un ejemplo típico que muestra esta capacidad es marcar la posición de la neurona como el patrón que representan, de esta forma se puede ver gráficamente cómo evolucionan las

neuronas y cómo aprenden la estructura de los datos de entrada (que para el ejemplo representan una determinada figura). Esto se ha mostrado en el ejemplo de la sección 4.4.1.5. En el archivo *demo_som.m* se incluye otro ejemplo con un patrón de tipo rectángulo.

Otro sencillo ejemplo consiste en mostrar un mapa de colores, de modo que los patrones de entrada sean un determinado color y entrenar la red SOM para que reconozca y clasifique los diferentes colores. Para mostrar gráficamente el aprendizaje de la red, se pinta un rectángulo por cada neurona (en la misma posición que tiene esa neurona en la capa de salida) y se rellena del color que representan los pesos almacenados en la neurona correspondiente. En la figura A4.7 se muestran dos salidas para tres patrones de entrada: {Rojo, Verde, Azul}. Se parte de una distribución en un plano de las neuronas de dimensión 20x20 y con valores aleatorios de sus pesos iniciales, es decir, de colores aleatorios. En la figura A4.7a se representa el resultado de la red si se utiliza un factor de vecindad inicial de 5 y en la figura A4.7b si se utiliza un factor de vecindad inicial de 10. Como puede observarse el número de neuronas que tienden al mismo color que la ganadora es mayor cuanto mayor es la vecindad. También se puede observar que algunas neuronas no cambian al no verse influidas por las ganadoras dada su considerable distancia.



a) Factor de vecindad 5 b) Factor de vecindad 10

Figura A4.7 Mapa de colores aprendido por la red SOM