Neural Network Toolbox

Sistemas Conexionistas - Curso 07/08

La Neural Network Toolbox es un paquete de Matlab que contiene una serie de funciones para crear y trabajar con redes de neuronas artificiales. Con `help nnet` se obtiene la lista de todas las funciones de este paquete.

1. Adaline

1.1. Funciones generales

mse Calcula el error cuadrático medio y se utiliza para evaluar la precisión de una red de neuronas.

```
error = mse(salida_deseada - salida_obtenida)
```

maxlinlr Calcula la velocidad máxima de aprendizaje para una red lineal. Toma como parámetro obligatorio los patrones de entrada a la red, los cuales se disponen en una matriz en la que cada columna representa un patrón de entrada.

```
P = [e11 e12; e21 e22]
lr = maxlinlr(P)
lr = maxlinlr(P,'bias')
```

learnwh Algoritmo de aprendizaje LMS (Widrow-Hoff). En un ciclo de aprendizaje, calcula la variación de los pesos como $dw = lr * e * p'$ y la variación del bias como $db = lr * e$. Su sintaxis es:

```
dW = learnwh(W,P,[].[],[],T,E,[],[],[],LP,[]);
```

donde

- W es la matriz de pesos de tamaño SxR, cada fila representa los pesos de un elemento de procesado
- P es la matriz de entrada de tamaño RxQ, cada columna representa un patrón de entrada
- T es la matriz de salidas deseadas de tamaño SxQ, cada columna representa una salida deseada
- E es la matriz de errores de tamaño SxQ, con un error por columna
- LP es una estructura que contiene los parámetros de aprendizaje. En concreto:
  - LP.lr es la velocidad de aprendizaje (learning rate)
  - LP.mc es el momento (momentum constant)
  - LP.dr es la tasa de decaimiento (decay rate)

Esta función devuelve una matriz de tamaño SxR que contiene el incremento de los pesos.

1.2. Creación

El comando `newlin` se utiliza para crear una red neuronal de tipo adaline. Su sintaxis es:

```
NET = newlin(PR,S,ID,LR)
```

donde

- PR es una matriz con los valores máximos y mínimos que puede tomar cada una de las entradas

  ```
  PR = [min_1 max_1; min_2 max_2; ... ; min_n max_n]
  ```

- S es el número de elementos del vector de salida
- ID es retardo de la entrada, por defecto su valor es [0]
- LR es la velocidad de aprendizaje, por defecto 0.01

`newlin` devuelve una red de tipo adaline. La red presenta una serie de propiedades configurables que definen las características básicas de la red[1]. Por ejemplo:

- `NET.biasConnect` Define que capas tienen bias.
- `NET.trainParam.epochs` Máximo número de ciclos de entrenamiento
- `NET.trainParam.goal` Error objetivo
- `NET.IW` Matrices de pesos de las capas de la red. Es un cell array de tamaño $N_l \times N_i$, donde N_l representa el número de capas mientras que N_i es el número de entradas de la red. Si la red contiene una única capa, podremos visualizar sus pesos con `Net.IW{1,1}`.
- `NET.b` Define los vectores de bias para cada capa con bias. Es un cell array de tamaño $N_l \times 1$.
- `NET.layers` Define las propiedades de cada una de las capas de la red. Con `NET.layers{i}` se accede a las propiedades de la capa `i`.

1.3. Entrenamiento

Una vez creada la red, el siguiente paso es realizar el entrenamiento con los patrones de entrada y las salidas deseadas. Existen dos tipos de entrenamiento:

Estático En cada ciclo de entrenamiento se recalculan los pesos de la red tras presentar todos los patrones de entrenamiento. Se realiza con la función `train`:

```
[net,TR,Y,E] = train(NET,P,T)
```

Los parámetros de entrada son:

- `NET` Una red inicializada
- `P` Los patrones de entrada

[1]Para más información consultar los capítulos 12 y 13 del manual de referencia del Neural Network Toolbox

- `T` Las salidas deseadas

Y los parámetros de salida son:

- `net` Red entrenada
- `TR` Error en función de la iteración
- `Y` Salida de la red
- `E` Errores de la red

Adaptativo En cada ciclo de entrenamiento se recalculan los pesos tras presentar cada uno de los patrones de entrenamiento. Se realiza con la función `adapt`, cuya sintaxis es la siguiente:

```
[net,Y,E] = adapt(NET,P,T)
```

Los parámetros de entrada son:

- `NET` Una red inicializada
- `P` Los patrones de entrada
- `T` Las salidas deseadas

Y los parámetros de salida son:

- `net` Red entrenada
- `Y` Salida de la red
- `E` Errores de la red

Antes de utilizar este tipo de entrenamiento es necesario especificar el número de pasadas de entrenamiento adaptativo con `NET.adaptParam.passes`.

1.4. Utilización

Tras la fase de entrenamiento, la red está lista para ser usada, es decir, la red es capaz de producir una salida adecuada a un conjunto de patrones de entrada. La función `sim` es la encargada de pasar un conjunto de patrones de entrada a la red y de obtener su salida:

```
Y =  sim(NET,P)
```

Donde

- `NET` representa una red entrenada
- `P` es el conjunto de patrones de entrada
- `Y` es la salida de la red

2. Perceptrón multicapa

2.1. Creación

El comando `newff` crea una red de neuronas de tipo feedforward. Su sintaxis es:

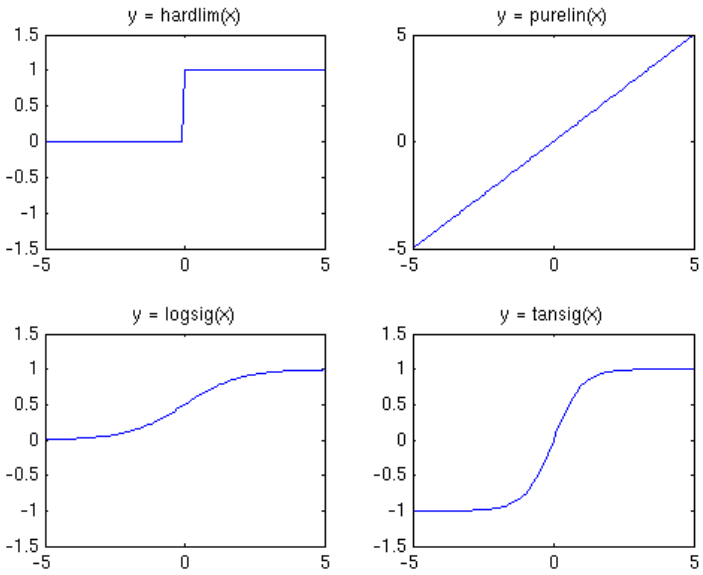
```
net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

- `PR` Matriz con los valores mínimo y máximo de los elementos de entrada
- `Si` Tamaño de la capa i
- `TFi` Función de transferencia de la capa i, por defecto es 'tansig'.
- `BTF` Función de entrenamiento, por defecto 'trainlm'.
- `BLF` Función de aprendizaje de los pesos/bias, por defecto 'learngdm'.
- `PF` Función de evaluación, por defecto 'mse'

Esta función devuelve una red feedforward con N capas.

Funciones de transferencia

Un elemento de procesado tiene N entradas. La suma de estas entradas ponderadas por los pesos y el bias constituye la entrada a la función de transferencia, la cual determina cómo serán las salidas del elemento de procesado. Los elementos de procesado pueden utilizar cualquier tipo de función de transferencia diferenciable para generar su salida, por ejemplo:



La función de transferencia de una capa se establece al crear la red o bien alterando el valor del parámetro `NET.layers{i}.transferFcn` en una red existente.
Es muy importante que la función de transferencia se adecúe al problema a resolver.

2.2. Entrenamiento

Se usan las mismas funciones descritas en el apartado Adaline, `adapt` para entrenamiento adaptativo y `train` para entrenamiento estático.

Funciones de entrenamiento

Existen diversos métodos para realizar el entrenamiento estático de una red de neuronas. Estos métodos se basan en algoritmos que intentan minimizar el error en base a diversas técnicas. Cada uno de estos métodos presenta sus ventajas e inconvenientes en cuanto a convergencia y coste computacional[2]. Algunas funciones de entrenamiento son `trainlm`, `traingd`, `traingdx`, `trainbr`, etc
Al igual que las funciones de transferencia, la función de entrenamiento se establece al crear la red o bien alterando el valor del parámetro `NET.transferFcn`

[2]El capítulo 5 del manual de referencia hace una comparativa entre los distintos algoritmos de entrenamiento.

2.3. Utilización

Se usa la función `sim` descrita en la sección anterior.

3. Mapas autoorganizativos

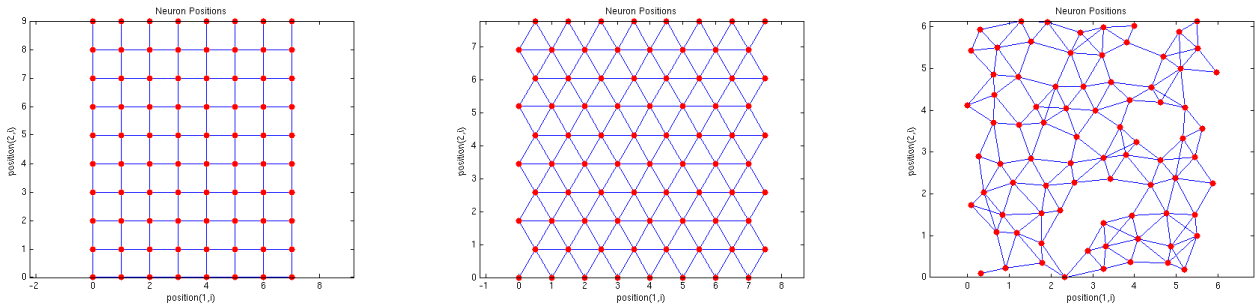
3.1. Creación

La función `newsom` se utiliza para crear un mapa autoorganizativo. Su sintaxis es:

```
NET = newsom(PR, [D1, D2, ...], TFCN)
```

Los parámetros de esta función son:

- `PR` es una matriz con los valores máximos y mínimos de las entradas
- `Di` es la dimensión de la i-ésima capa
- `TFCN` representa la topología (gridtop,hextop o randtop)



La función `newsom` devuelve un mapa autoorganizativo.
Algunos parámetros de interés de la nueva red son:

- `NET.trainParam.epochs` Máximo número de ciclos de entrenamiento
- `NET.trainParam.goal` Error objetivo

La función `plotsom` dibuja mapas autoorganizativos:

- `plotsom(net.layers{i}.positions)` representa la posición de los elementos de procesado de la capa i-ésima del mapa autoorganizativo. Cada elemento de procesado está situado a una distancia euclídea de 1 con respecto a sus vecinos.

- `plotsom(net.IW{i,j}, net.layers{i}.distances)` representa la posición *real* de cada elemento de procesado con respecto a sus vecinos en la capa i-ésima.

3.2. Entrenamiento

Se utiliza la función `train` descrita anteriormente.

3.3. Utilización

La función `sim` descrita en el apartado Adaline permite aplicar el mapa autoorganizativo entrenado a un nuevo conjunto de patrones. En este caso, la función `sim` devuelve no un valor sino un vector con tantas posiciones como elementos de procesado. Todas las posiciones de este vector están ocupadas por ceros excepto la posición correspondiente al elemento de procesado activado en la capa competitiva, el cual toma valor uno. En estas casos Matlab muestra únicamente el índice con valor uno:

```
>> Y = sim(net,X)

Y =

  (18,1)        1
```

A veces es necesario extraer el índice correspondiente al elemento de procesado activado. Para ello se utiliza la función `vec2ind`:

```
>> vec2ind(Y)

ans =

    18
```