Práctica 1. Adaline.

Sistemas Conexionistas - Curso 08/09

1. Implementación de un Adaline

1.1. Entrenamiento *batch*

Implementar la función *adaline1* en MATLAB que simule el entrenamiento de un Adaline genérico utilizando la regla LMS.
La función recibirá como **entradas**:

- Los patrones de entrenamiento. El programa debe aceptar n patrones formados por m componentes.
- Las salidas deseadas para cada patrón. Habrá n salidas (una salida por patrón de entrada) de 1 componente.
- La velocidad de aprendizaje
- El error máximo permitido

La función entrenará adaline utilizando la regla LMS (sin bias) y recalculando los pesos tras introducir todos los patrones de entrada. Se considerarán varias alternativas a la hora de inicializar los pesos. El entrenamiento finalizará cuando se alcance la tasa de error especificada o se supere un número máximo de iteraciones.
La función devolverá como **salida** los pesos finales tras el proceso de entrenamiento.
La función mostrarán **gráficamente**:

- La evolución de cada peso durante el entrenamiento
- La evolución del error durante el entrenamiento

1.2. Entrenamiento continuo

Implementar la función *adaline2* en MATLAB que simule el entrenamiento de un Adaline genérico utilizando la regla LMS.
La función recibirá como **entradas**:

- Los patrones de entrenamiento. El programa debe aceptar n patrones formados por m componentes.
- Las salidas deseadas para cada patrón. Existirán n salidas (una salida por patrón de entrada) de 1 componente.
- La velocidad de aprendizaje
- El número de iteraciones de entrenamiento

La función entrenará el adaline utilizando la regla LMS (sin bias) y recalculando los pesos tras introducir cada patrón. Se considerarán varias alternativas a la hora de inicializar los pesos. El entrenamiento finalizará cuando se realicen las iteraciones especificadas.
La función devolverá como **salida**:

- Los pesos finales tras el proceso de entrenamiento
- Las salidas obtenidas por el adaline durante el proceso de entrenamiento en la última iteración

La función mostrará **gráficamente**;

- La evolución del error durante el entrenamiento
- La salida obtenida con cada patrón de entrenamiento en la última iteración

2. Simulación de un Adaline

Realizar la función *simular* en Matlab que permita aplicar un Adaline entrenado a una serie de patrones de entrada para obtener la salida de la red.
La función recibirá como entradas:

- Los pesos del Adaline entrenado
- Las nuevos patrones de entrada

La función devolverá las salidas correspondientes a aplicar los patrones de entrada al Adaline.

3. Aplicaciones

Utilizar las **funciones implementadas** para abordar las siguientes aplicaciones:

3.1. Asociación de patrones

Entrenar un Adaline para que asocie los siguientes patrones de entrada y salidas deseadas:

Patrones de entrada	**Salidas deseadas**
3, 4, 2	-7
2, -2, 1	5
-1, 3, -2	-5
2, -1, 3	1

Comprobar como afecta el valor de la velocidad de aprendizaje al entrenamiento.
Tras realizar el entrenamiento, introducir los patrones de entrada en el adaline para comprobar que la salida obtenida se aproxima a las salidas deseadas.

3.2. Identificación de transformaciones lineales

La señal almacenada en el fichero `s2.mat` es el resultado de aplicar una transformación lineal a 5 componentes consecutivas de otra señal almacenada en el fichero `s1.mat`.
Seleccionar un conjunto de patrones de entrada y salidas deseadas a partir de las señales `s1` y `s2` para entrenar el adaline.
Tras realizar el entrenamiento, simular el comportamiento del adaline con todos los patrones de entrada de la señal `s1`. Mostrar gráficamente la salida del adaline frente a la señal original `s2`.

3.3. Predicción de señales

Aplicar el adaline a la señal almacenada en el fichero `s3.mat` para realizar una tarea de predicción.
Entrenar el adaline de forma que a partir de los tres valores anteriores de la señal de entrada, sea capaz de predecir el valor actual.
Representar gráficamente la señal predecida frente a la señal original.