Práctica 1
Introducción a las Redes Neuronales Artificiales

Laboratorio de Neurocomputación
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Fecha de inicio: Lunes 07/02/2021 (2462) / Martes 08/02/2021 (2461)
Fecha de entrega: Domingo 06/03/2021 (2462) / Lunes 07/03/2021 (2461)

1. Implementación de una librería para el manejo de redes neuronales

El objetivo de esta práctica es diseñar e implementar distintos modelos neuronales artificiales tradicionales. Para facilitar el desarrollo de esta práctica y las posteriores, lo más interesante es plantear el diseño de una librería que permita gestionar los valores internos de las neuronas, agruparlas en capas y realizar sinapsis. Utilizad para ello la especificación de la Fig. 1.

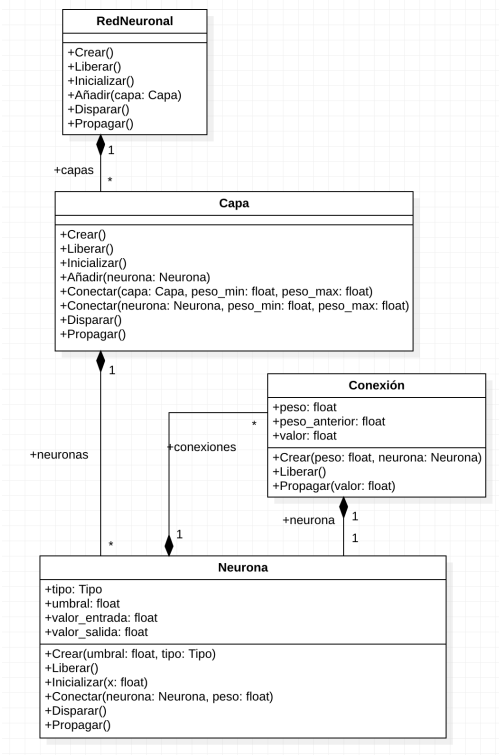

Figura 1: Especificación UML para la librería de manejo de redes neuronales.

2. Neuronas de McCulloch-Pitts

En una neurona típica de McCulloch–Pitts la activación es binaria (su salida es 0 o 1), las conexiones excitadoras que llegan a una misma neurona tienen el mismo peso, el umbral se escoge de forma que una sola conexión inhibidora fuerce a que la salida de la neurona sea 0 en ese paso y las salidas tardan un intervalo de tiempo en llegar a las neuronas receptoras. Todas estas propiedades hacen que las neuronas de McCulloch-Pitts sean útiles para modelar fenómenos que requieran funciones lógicas y retrasos temporales.

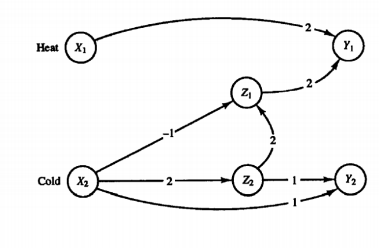

Figura 2: Frío y calor

Diseñad e implementad una red de McCulloch-Pitts que implemente la función de la Fig. 2 vista en clase. El programa recibirá mediante una opción el nombre de un fichero texto con los impulsos de con el siguiente formato:

```
0 0
1 1
1 1
0 1
1 0
0 0
0 0
0 0
```

El valor de todas las neuronas de salida de la red se guardarán en otro archivo de texto que será indicado también mediante una opción del programa:

```
x1  x2  z1  z2  y1  y2
0   0   0   0   0   0
1   1   0   0   0   0
1   1   0   1   0   1
0   1   1   1   0   1
1   0   1   1   1   1
0   0   1   0   1   1
0   0   0   0   1   0
0   0   0   0   0   0
...
```

Una vez finalizada la lectura de la entrada, la red debe seguir funcionando hasta que se haya propagado toda la información.

2.1. Memoria

- Incluid el diseño de la red con los sesgos, conexiones y pesos utilizados.
- Discutid la validez de vuestro diseño e incluid explicaciones de ejemplos que demuestren el correcto funcionamiento interno de cada elemento que conforma el circuito.
- ¿Qué calcula la red con una entrada?

2.2. Código

El código debe ir acompañado con un `Makefile` que permita la compilación y ejecución. Tendrá, al menos, los siguientes objetivos:

- `ayuda_mp`: Explicación de los argumentos necesario para ejecutar el programa.
- `compila_mp`: Compila lo necesario para ejecutar el programa.
- `ejecuta_mp`: Un ejemplo de ejecución.

3. Lectura de datos

Los ficheros de datos están formados por líneas. En la primera línea se indica en número de atributos (M) y clases (N). El resto de líneas contienen los patrones. Cuando haya N clases, se codificarán con N valores, de manera que solo la posición correspondiente a la clase estará a 1 y el resto valdrá -1 (codificación bipolar). En el siguiente ejemplo, se pueden ver dos patrones, con cuatro atributos y tres clases:

```
4 3
0.2 0.3 0.4 0.1  1 -1 -1
0.0 0.1 0.5 0.1 -1 -1  1
```

Para esta prácticas y las siguientes, deben existir tres modos de funcionamiento cuando se leen ficheros de datos:

- Modo 1: Requiere de un solo fichero de datos, se debe indicar por tanto que porción es utilizada como fichero de entrenamiento. Los datos de entrenamiento se eligen aleatoriamente en cada lectura. Con el resto de datos se realizará el test.
- Modo 2: Requiere un solo fichero donde los datos servirán como entrenamiento y test.
- Modo 3: Se indican dos ficheros: el primero como entrenamiento y el segundo como test.

Para el Modo 3, las predicciones del fichero de test estarán sincronizados al nivel de linea. Para estos tres modos, las funciones, sus argumentos de entrada y las salidas serán las siguientes:

```
leer1(fichero_de_datos, por)
 -> (entradas_entrenamiento, salidas_entrenamiento, entradas_test, salidas_test)

leer2(fichero_de_datos) -> (entradas_datos, salidas_datos)

leer3(fichero_de_entrenamiento, fichero_de_test)
 -> (entradas_entrenamiento, salidas_entrenamiento, entradas_test, salidas_test)
```

4. Perceptrón y Adaline

Una de las características más destacables de las redes neuronales es su capacidad de aprendizaje y generalización. El objetivo de esta práctica es el estudio y la implementación de las reglas de aprendizaje que utilizan el Perceptrón y el Adaline, y su aplicación a la clasificación de patrones.

4.1. Problemas lógicos

Entrenad ambos tipos de redes con los problemas lógicos `and.txt`, `or.txt`, `nand.txt` y `xor.txt`, leyendo los ficheros en Modo 2. Comprobar con estos problemas que las implementaciones son correcta.

4.1.1. Memoria

- Dad las fronteras de decisión de cada algoritmo para los cuatro problemas lógicos en la forma $w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$.
- ¿Es posible solucionar todos los problemas? En caso de no ser posible: ¿cuál es el problema que no es posible? ¿cómo podría solucionarse?

4.2. Problema real 1

Diseñad un Perceptrón y un Adaline que solucione `problema_real1.txt`. Los parámetros que afectan a las redes y el aprendizaje (épocas, umbrales, tasa de aprendizaje…) deben ser incluidos como argumentos de los programas.

4.2.1. Memoria

- Describid la implementación de ambas redes neuronales.
- Compara ambas redes, ¿qué sucede con el Error Cuadrático Medio (ECM) en cada una de ellas?.
- ¿Que influencia presentan en el resultado cada uno de los parámetros que definen el comportamiento de las dos redes? Presentad gráficas que muestren como varía el resultado modificando un solo parámetro de control (p. ej.: umbral). Utilizad `problema_real1` y el Modo 1.

4.2.2. Código

El código debe ir acompañado con un `Makefile` que permita la compilación y ejecución. Tendrá, al menos, los siguientes objetivos:

- `compilar`: Compilación de todo, incluir aunque no haga nada.
- `ayuda_perceptron`: Explicación de los argumentos necesario para ejecutar el programa.
- `ejecuta_perceptron`: Ejemplo de ejecución (Problema real 1 y Modo 1 con buen resultado).
- `ayuda_adaline`: Explicación de los argumentos necesario para ejecutar el programa.
- `ejecuta_adaline`: Ejemplo de ejecución (Problema real 1 y Modo 1 con buen resultado).

4.3. Problema real 2

Diseñad un Perceptrón y un Adaline y que solucione `problema_real2.txt`. Mediante el Modo 3 realiza predicciones para `problema_real2`. No te preocupes por el porcentaje de test, el segundo fichero esta sin etiquetar.

4.3.1. Memoria

Describid la implementación de ambas redes neuronales y de los parámetros usados y resultados.

4.3.2. Código

Incluye los ficheros `prediccion_perceptron.txt` y `prediccion_adaline.txt` en una carpeta `predicciones` en la carpeta raíz de tu entrega.

5. Entrega y memoria

- Contestar a las cuestiones planteadas en los diferentes apartados en una memoria. Incluir los resultados en gráficas que muestren datos.
- La entrega en Moodle se hará con el siguiente formato:

Practica1-ParejaXX-Apellido1-Apellido2.zip