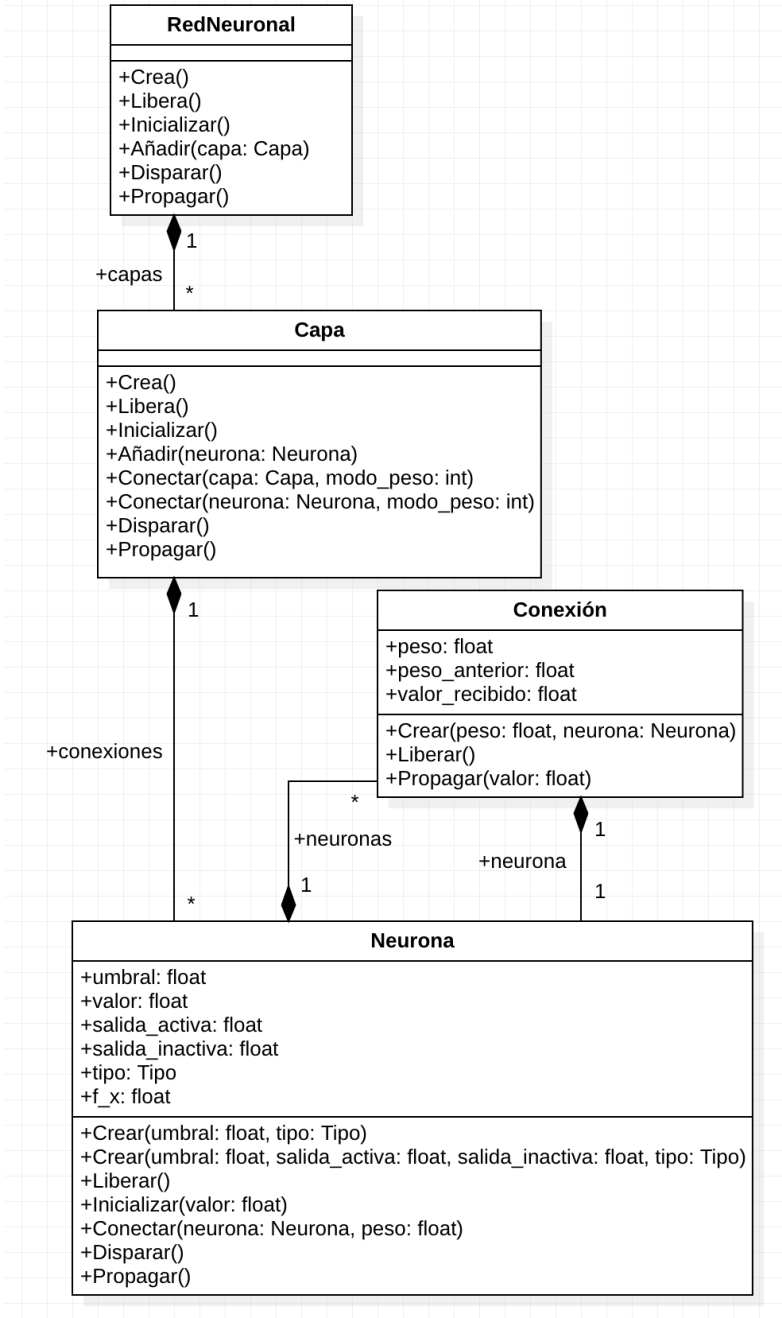
Práctica 1: Introducción a las Redes Neuronales Artificiales

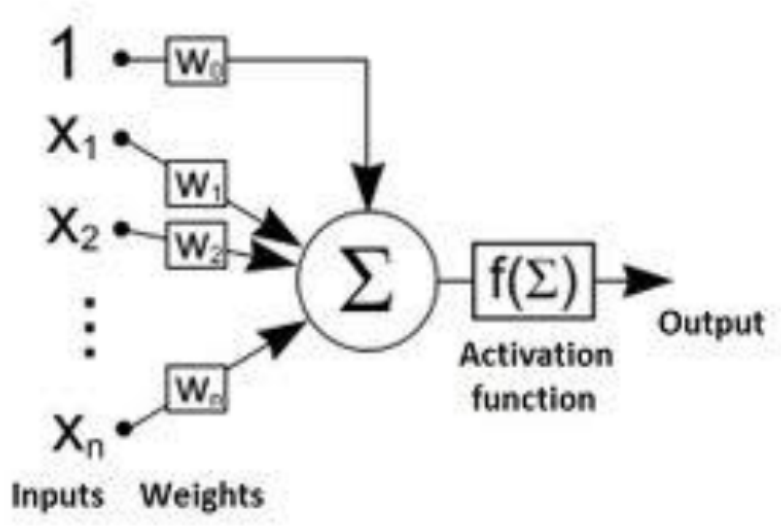
Laboratorio de Neurocomputación

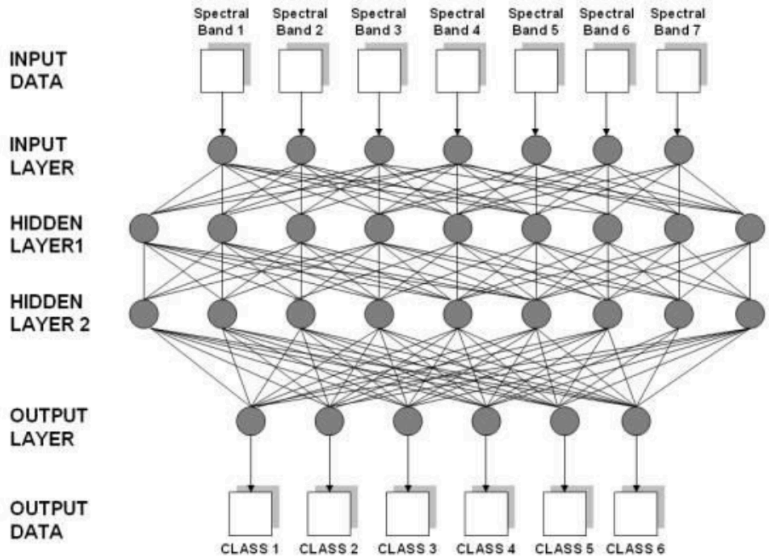
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Curso 2020-2021

Librería para el manejo de redes neuronales







Notación / Nomenclatura

- x_i, y_i : Activaciones de las neuronas X_i, Y_j

  Para neuronas de entrada X_i : x_i = señal de entrada

  Para otras neuronas: Y_j : $y_j = f(y\_in_j)$, f función de transferencia

- w_{ij} : peso de la conexión de la neurona X_i a la neurona Y_j

- b_j : sesgo *o* bias de la neurona Y_j (actúa como un peso de una conexión desde una neurona que tiene una activación constante 1)

- y_in_j : input o entrada a la neurona Y_j : $y\_in_j = b_j + \sum_i x_i w_{ij}$

- W : matriz de pesos $W=\{w_{ij}\}$

- $\boldsymbol{w}_{\cdot j}$: vector de pesos $\boldsymbol{w}_{\cdot j} = (w_{1j}, w_{2j}, \ldots, w_{nj})^T$ (columna j-ésima de la matriz de pesos)

- θ_j : Umbral para la activación de la neurona Y_j (lo utilizan las funciones de transferencia)

- $\mathbf{s}$: vector de entrenamiento de entrada: $\mathbf{s} = (s_1, \ldots., s_i, \ldots, s_n)$

- t: vector de objetivo de salida (*target*) $\mathbf{t} = (t_1, \ldots., t_i, \ldots, t_n)$

- $\mathbf{x}$: vector de entrada (estímulo a la red) $\mathbf{x} = (x_1, \ldots., x_i, \ldots, x_n)$

- Δw_{ij} cambio de pesos por el aprendizaje: $\Delta w_{ij} = [w_{ij}(\text{nuevo}) - w_{ij}\ (\text{anterior})]$

  recordando $y\_in_j = b_j + \sum_i x_i w_{ij}$

- α : tasa de aprendizaje (se utiliza para controlar la cantidad de ajuste de peso en cada paso del entrenamiento).

Clase Neurona (C++)

```
1   class Neurona {
2   public:
3     enum {
4           Directa,
5           McCulloch,
6           Sesgo,
7           SigmoideBipolar,
8           SigmoidePersonalizada
9     };
10    Neurona(float umbral, int tipo);
11    Neurona(float umbral, float salida_activa, float salida_inactiva, int tipo);
12    virtual ~Neurona();
13    void inicializar(float valor);
14    void conectar(Neurona* neurona, float peso);
15    void disparar();
16    void propagar();
17
18  public:
19    float umbral;
20    float valor;
21    float salida_activa;
22    float salida_inactiva;
23    int tipo;
24    float f_x;
25    vector<Conexion> conexiones;
26  };
```

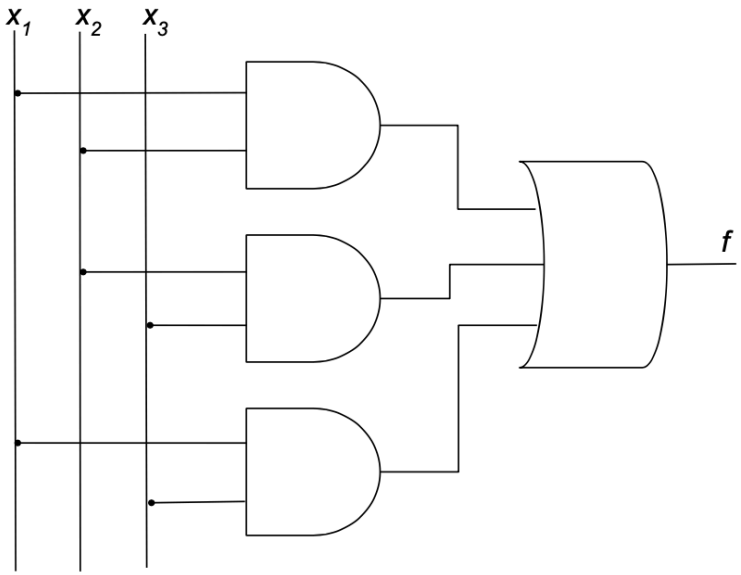
Métodos disparar y propagar de Neurona (C++)

```
1   void Neurona::disparar()
2   {
3     if (tipo == Directa)
4       f_x = valor;
5     else if (tipo == Sesgo)
6       f_x = 1.0;
7     else if (tipo == McCulloch)
8       f_x = valor >= umbral ? salida_activa : salida_inactiva;
9     else ...
10
11    for (auto& conexion : conexiones)
12      conexion.valor_recibido = f_x;
13  }
14
15  void Neurona::propagar()
16  {
17    for (auto& conexion : conexiones)
18      conexion.neurona->valor += conexion.peso * conexion.valor_recibido;
19  }
```

Redes de McCullogh-Pitts: Problema y entrada

$$y\_in = \sum_i x_i w_i$$

$$f(y\_in) = \begin{cases} 1, & \text{si } y\_in \geq \theta \\ 0, & \text{si } y\_in < \theta \end{cases}$$



0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	0	0

Redes de McCullogh-Pitts: Salida

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  x3  a12 a13 a23 y
0   0   0   0   0   0   0
0   0   1   0   0   0   0
0   1   0   0   0   0   0
0   1   1   0   0   0   0
1   0   0   0   0   1   0
1   0   1   0   0   0   1
...
```

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

Redes de McCullogh-Pitts: 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 función $f(t+2)$ para unos $x_1(t)$, $x_2(t)$ y $x_3(t)$ dados?

Redes de McCullogh-Pitts: 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.

Perceptrón y Adaline: Ficheros de entrada

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
```

Perceptrón y Adaline: Ficheros de entrada

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.

Perceptrón y Adaline: Ficheros de entrada

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)
```

Algoritmo de aprendizaje del Perceptrón

Paso 0: Inicializar todos los pesos y sesgos (por simplicidad a cero)
Establecer la tasa de aprendizaje $\alpha\ (0 < \alpha \leq 1)$

Paso 1: Mientras que la condición de parada sea falsa, ejecutar pasos 2-6

Paso 2: Para cada par de entrenamiento ($\mathbf{s}$:t), ejecutar los pasos 3-5:

Paso 3: Establecer las activaciones a las neuronas de entrada

$$x_i = s_i \quad (i=1 \ldots n)$$

Paso 4: Calcular la respuesta de la neurona de salida:

$$y\_in = b + \sum_i x_i w_i$$

$$f(y\_in) = \begin{cases} 1 & \text{si } y\_in > \theta \\ 0 & \text{si } -\theta \leq y\_in \leq \theta \\ -1 & \text{si } y\_in < -\theta \end{cases}$$

Paso 5: Ajustar los pesos y el sesgo si ha ocurrido un error para este patrón:

Si $y \neq t$

$$w_i(\text{nuevo}) = w_i(\text{anterior}) + \alpha t x_i$$
$$b(\text{nuevo}) = b(\text{anterior}) + \alpha t$$

Si no

$$w_i(\text{nuevo}) = w_i(\text{anterior})$$
$$b(\text{nuevo}) = b(\text{anterior})$$

Paso 6: Comprobar la condición de parada: si no han cambiado los pesos en el paso 2: parar; en caso contrario, continuar.

Algoritmo de aprendizaje del Adaline

Paso 0: Inicializar todos los pesos y sesgos (valores aleatorios pequeños)
Establecer la tasa de aprendizaje α.

Paso 1: Mientras que la condición de parada sea falsa, ejecutar pasos 2-6

Paso 2: Para cada par de entrenamiento (***s***:*t*) bipolar, ejecutar los pasos 3-5:

Paso 3: Establecer las activaciones a las neuronas de entrada

$$x_i = s_i \quad (i=1 \ldots n)$$

Paso 4: Calcular la respuesta de la neurona de salida:

$$y\_in = b + \sum_i x_i w_i$$

Paso 5: Ajustar los pesos y el sesgo:

$$w_i(\text{nuevo}) = w_i(\text{anterior}) + \alpha(t\text{-}y\_in)x_i$$

$$b(\text{nuevo}) = b(\text{anterior}) + \alpha(t\text{-}y\_in)$$

Paso 6: Comprobar la condición de parada: si el cambio de peso más grande en el paso 2 es menor que una tolerancia especificada: parar; en caso contrario, continuar.

Perceptró y Adaline: 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 correctas.

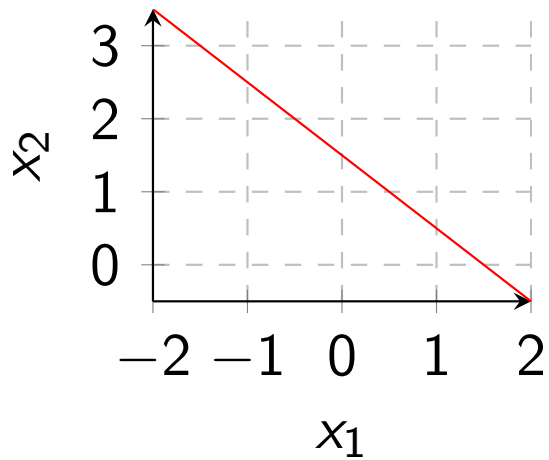
Perceptrón y Adaline: Memoria

- 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?

Perceptrón y Adaline: Ejemplo

Por ejemplo, la frontera de decisión que calcula Adaline para AND es:

AND: $x_2 = -x_1 + 1.5$



Perceptrón y Adaline: Problema real 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?.

$$\text{Error cuadrático medio (ECM)} = \sum_{j=1}^{m} (t_j - y\_in_j)^2$$

- ¿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.

Perceptrón y Adaline: Problema real 1 - 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_per`: Explicación de los argumentos necesario para ejecutar el programa.
- `ejecuta_per`: Ejemplo de ejecución (Problema real 1 y Modo 1 con buen resultado).
- `ayuda_ada`: Explicación de los argumentos necesario para ejecutar el programa.
- `ejecuta_ada`: Ejemplo de ejecución (Problema real 1 y Modo 1 con buen resultado).

Perceptrón y Adaline: 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 (`problema_real2`$_{no}$*etiquetado.txt*.
- Describid la implementación de ambas redes neuronales y de los parámetros usados y resultados.
- Incluye los ficheros `prediccion_perceptron.txt` y `prediccion_adaline.txt` en una carpeta `predicciones` en la carpeta raíz de tu entrega.

Perceptrón y Adaline: Problema real 2 - Memoria

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

Perceptrón y Adaline: Problema real 2 - Código

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

Práctica 1: Memoria y entrega

- 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
- Fecha de inicio:
  - Lunes 15/02/2021 (2462)
  - Viernes 19/02/2021 (2461)
- Fecha de entrega:
  - Domingo 14/03/2021 (2462)
  - Lunes 15/03/2021 (2461)