Análisis y Diseño de Algoritmos.

Sesión 13. 18 de Noviembre de 2015.

Maestría en Sistemas Computacionales.

Por: Hugo Iván Piza Dávila.

Reconocimiento de Patrones

- El propósito de un sistema de reconocimiento de patrones es identificar a qué clase pertenece un objeto dado.
- El sistema tiene identificado un conjunto finito de clases.
  - "*Objeto no reconocido*" puede considerarse como una clase.
- Esquema general de un sistema de RP:

Sensor

- Sistemas de software/hardware encargados de:
  - Adquisición de datos del ambiente.
  - Transformación del formato con el que viene el dato en uno que pueda ser manipulado por el sistema de RP
    - De: temperatura, presión, inclinación, humedad, intensidad,…
    - A: números reales, arreglos, matrices, …

Selección de características

- La representación de un dato/objeto procesado por el sensor consiste en una secuencia de valores asignados para un conjunto de atributos o características.
- Los valores de cada atributo pertenecen a un dominio específico.
- Pueden existir atributos más relevantes que otros.
- La selección de atributos relevantes permitirá:
  1. Aumentar precisión en la clasificación porque se eliminan atributos que introducen ruido y no aportan nada.
  2. Acelerar el proceso de clasificación porque cada objeto es más pequeño: implica menos iteraciones.

Clasificación

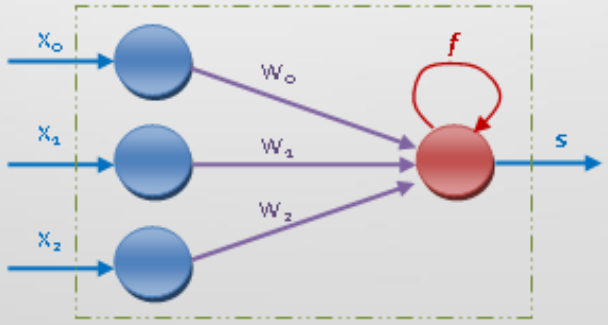
- Consiste en determinar a cuál clase se le asemeja más un objeto dado, utilizando los atributos seleccionados.
- Dependiendo de la información *a priori* que se cuente para poder llevar a cabo la clasificación, ésta puede ser:
  1. Supervisada
     - También llamada *clasificación con aprendizaje*.
     - Se cuenta con un conjunto de objetos y se conoce a qué clase pertenecen.
     - Métodos populares: redes neuronales y vecino más cercano.
  2. No supervisada
     - También llamada *clasificación sin aprendizaje*.
     - Se desconocen las clases.
     - Los individuos más próximos se van agrupando formando clases: *clustering*.

Redes Neuronales

- Agrupa un conjunto de modelos computacionales inspirados en la forma en que funciona el sistema nervioso de los animales para realizar clasificación supervisada.
- Necesitan ser previamente entrenadas:
  - Dotar a la RN de un conjunto de objetos y de la clase esperada de cada uno. Realizar modificaciones a ciertos valores de la RN tantas veces hasta que la clase calculada por la RN para cada objeto sea igual a su clase esperada.
  - El entrenamiento puede ser muy tardado.
  - La clasificación es inmediata.

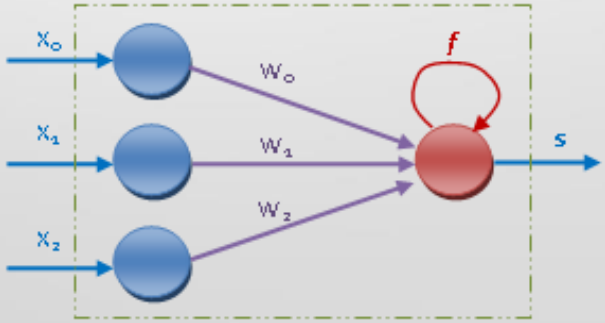
Perceptrón

- Es uno de los tipos de RN más sencillos.
- Consta de dos capas: entrada (objeto) y salida (clase).
- Una capa consta de neuronas. Cada neurona de una capa se comunica con todas las neuronas de la siguiente capa.
- Los arcos que comunican neuronas tienen un peso: [0..1.0].



Perceptrón

- La neurona de la capa de salida tiene una entrada que se calcula como el producto punto entre los vectores x, w: $x_0w_0 + x_1w_1 + x_2w_2$.
- La salida es el resultado de una *función de activación* que se ejecuta sobre la entrada. Su propósito es mapear entradas con *clases*.
- El entrenamiento de una RN consiste en asignar pesos adecuados a todos los arcos de forma que todos los objetos del conjunto de entrenamiento sean clasificados correctamente.



Caso didáctico

- Implementar una red neuronal que aprenda a realizar una operación NAND entre 3 valores *booleanos* (0, 1).
  - Como en la figura, 3 neuronas en la entrada, 1 en la salida.
- Conjunto de entrenamiento (entrada, salida):
  1. (1, 0, 0), 1
  2. (1, 0, 1), 1
  3. (1, 1, 0), 1
  4. (1, 1, 1), 0

Caso didáctico

- Función de activación:
  - Como la entrada de la capa de salida es un número en [0..1.0] y las salidas posibles son cero o uno, esta función utilizará un valor de *umbral* para decidir qué salida tomar:
  - Si $x \bullet w > 0.5$, $s = 1$, si no, $s = 0$.
- Todos los pesos de los arcos comienzan en 0.0.
  - Para el primer objeto [(1, 0, 0), 1] tendríamos lo siguiente:
    - $x \bullet w = 1 \times 0 + 0 \times 0 + 0 \times 0 = 0$. Por tanto, s = 0.
    - Pero la salida esperada es 1.0. Tenemos un error de 1 – 0 = 1.

Caso didáctico

- Ajustar los pesos de los arcos mediante la propagación del error obtenido hacia atrás: *back-propagation*
- Sea e = d – s el error de la red: salida deseada menos salida obtenida. Calcular un factor de corrección: $\delta = LR \cdot e$.
  - LR = 0.1 es la tasa de aprendizaje y define qué tan fina es la corrección de los pesos.
- A cada peso w_k se le sumará $x_k \cdot \delta$.
- Repetir con el siguiente objeto del conjunto de entrenamiento. Si ya estamos en el último, regresamos al primero. A cada tanda (4 objetos) se le denomina *epoch*.
- Cuando e = 0 para cuatro objetos consecutivos, terminamos.

Caso didáctico

- Ejecución:

	x0	x1	x2	d	w0	w1	w2	$x \bullet w$	s	e	δ
1	1	0	0	1	0	0	0	0	0	1	0.1
2	1	0	1	1	0.1	0	0	0.1	0	1	0.1
3	1	1	0	1	0.2	0	0.1	0.2	0	1	0.1
4	1	1	1	0	0.3	0.1	0.1	0.5	0	0	0
5	1	0	0	1	0.3	0.1	0.1	0.3	0	1	0.1
6	1	0	1	1	0.4	0.1	0.1	0.5	0	1	0.1
7	1	1	0	1	0.5	0.1	0.2	0.6	1	0	0
8	1	1	1	0	0.5	0.1	0.2	0.8	1	-1	-0.1
9	1	0	0	1	0.4	0	0.1	0.4	0	1	0.1
10	1	0	1	1	0.5	0	0.1	0.6	1	0	0
11	1	1	0	1	0.5	0	0.1	0.5	0	1	0.1
12	1	1	1	0	0.6	0.1	0.1	0.8	1	-1	-0.1
13	1	0	0	1	0.5	0	0	0.5	0	1	0.1
14	1	0	1	1	0.6	0	0	0.6	1	0	0
15	1	1	0	1	0.6	0	0	0.6	1	0	0
16	1	1	1	0	0.6	0	0	0.6	1	-1	-0.1
17	1	0	0	1	0.5	-0.1	-0.1	0.5	0	1	0.1

Caso didáctico

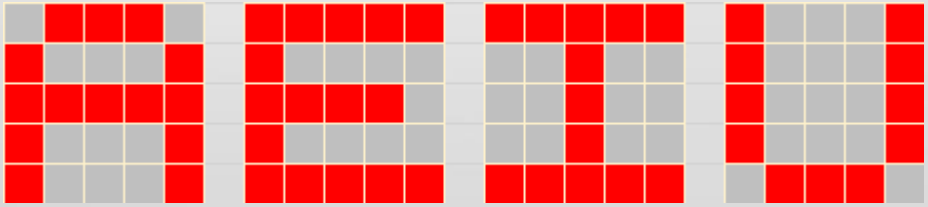
- Ejecución:

	x0	x1	x2	d	w0	w1	w2	f	s	e	δ
18	1	0	1	1	0.6	-0.1	-0.1	0.5	0	1	0.1
19	1	1	0	1	0.7	-0.1	0	0.6	1	0	0
20	1	1	1	0	0.7	-0.1	0	0.6	1	-1	-0.1
21	1	0	0	1	0.6	-0.2	-0.1	0.6	1	0	0
22	1	0	1	1	0.6	-0.2	-0.1	0.5	0	1	0.1
23	1	1	0	1	0.7	-0.2	0	0.5	0	1	0.1
24	1	1	1	0	0.8	-0.1	0	0.7	1	-1	-0.1
25	1	0	0	1	0.7	-0.2	-0.1	0.7	1	0	0
26	1	0	1	1	0.7	-0.2	-0.1	0.6	1	0	0
27	1	1	0	1	0.7	-0.2	-0.1	0.5	0	1	0.1
28	1	1	1	0	0.8	-0.1	-0.1	0.6	1	-1	-0.1
29	1	0	0	1	0.7	-0.2	-0.2	0.7	1	0	0
30	1	0	1	1	0.7	-0.2	-0.2	0.5	0	1	0.1
31	1	1	0	1	0.8	-0.2	-0.1	0.6	1	0	0
32	1	1	1	0	0.8	-0.2	-0.1	0.5	0	0	0
33	1	0	0	1	0.8	-0.2	-0.1	0.8	1	0	0
34	1	0	1	1	0.8	-0.2	-0.1	0.7	1	0	0

Objetivo

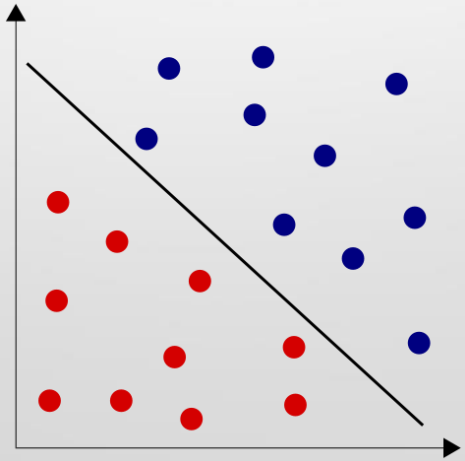
- Implementar un Perceptrón Multicapa que reconozca el símbolo especificado mediante una matriz binaria de 5 x 5.
  - Caso 1. Símbolos: O, X

  - Caso 2. Símbolos: A, E, I, U



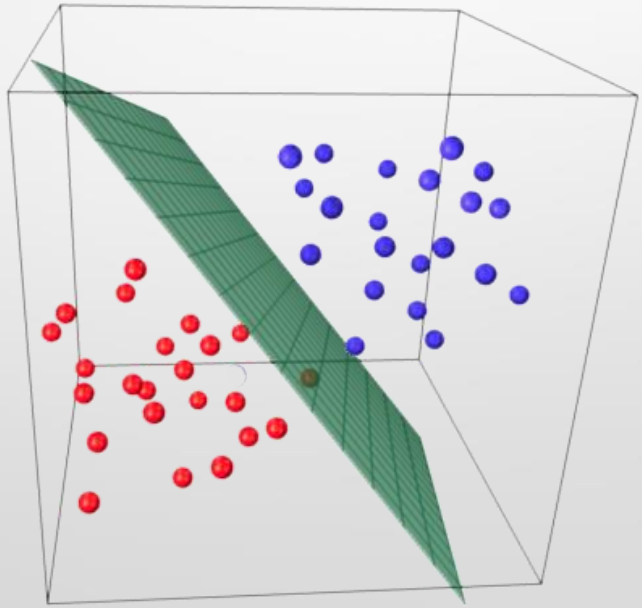
Separabilidad lineal

- Imaginemos que todos los objetos a reconocer puedan mapearse en un plano y que se reconocen dos clases: rojo y azul.
- Si podemos trazar una línea recta que pueda separar a todos los objetos azules de los rojos: los datos son linealmente separables.



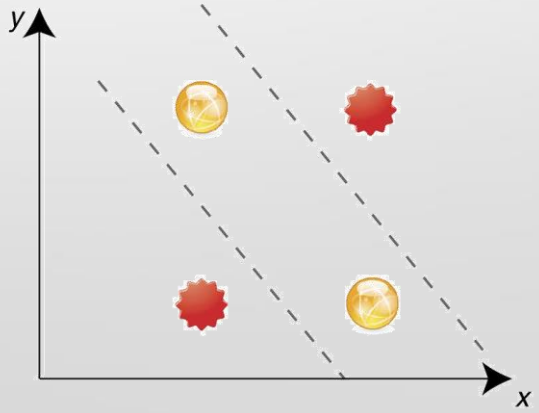
Separabilidad lineal

- Este concepto lo podemos llevar a más dimensiones.
- En lugar de una línea recta: un híper-plano.



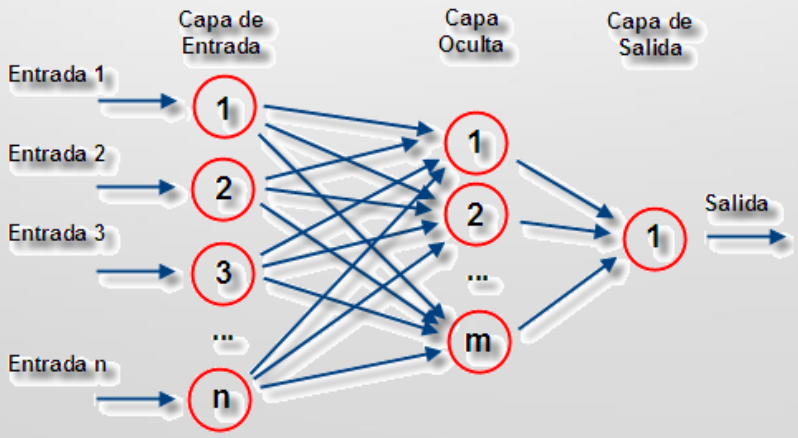
Separabilidad lineal

- Cuando los objetos a clasificar son separables linealmente, es posible que no se necesite una red neuronal para poder clasificar, si no encontrar la función que define al híper-plano.
- Una red neuronal tiene mayor aplicación cuando los objetos presentan traslapes en el dominio del problema.



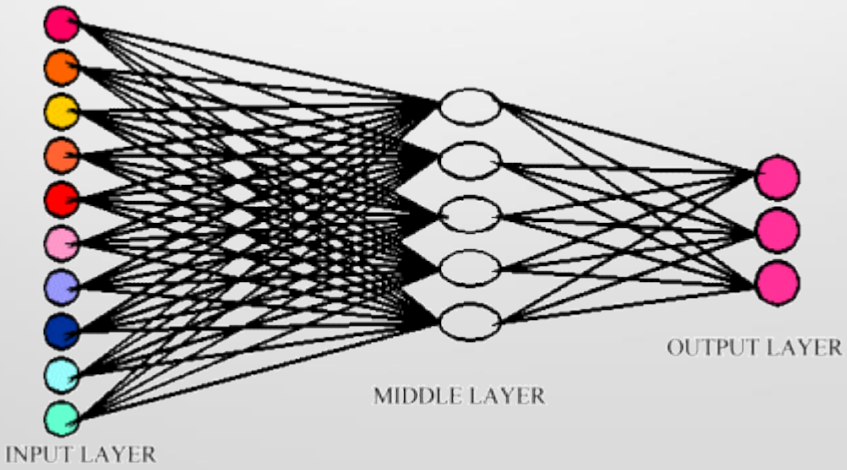
Perceptrón Multicapa

- Un perceptrón puede tener dos o más capas.
- La precisión en la clasificación que se obtiene con tres capas es casi siempre mayor que cuando sólo se usan dos.
- En muchos problemas, usar más de tres capas no mejora la precisión, pero si aumenta el costo computacional.



Perceptrón Multicapa

- A la capa intermedia también se le denomina capa oculta.
- La capa de salida puede estar formada por más de una neurona.
  - Necesario cuando existen muchas clases y/o se desea un nivel mayor de especificidad por cada clase.

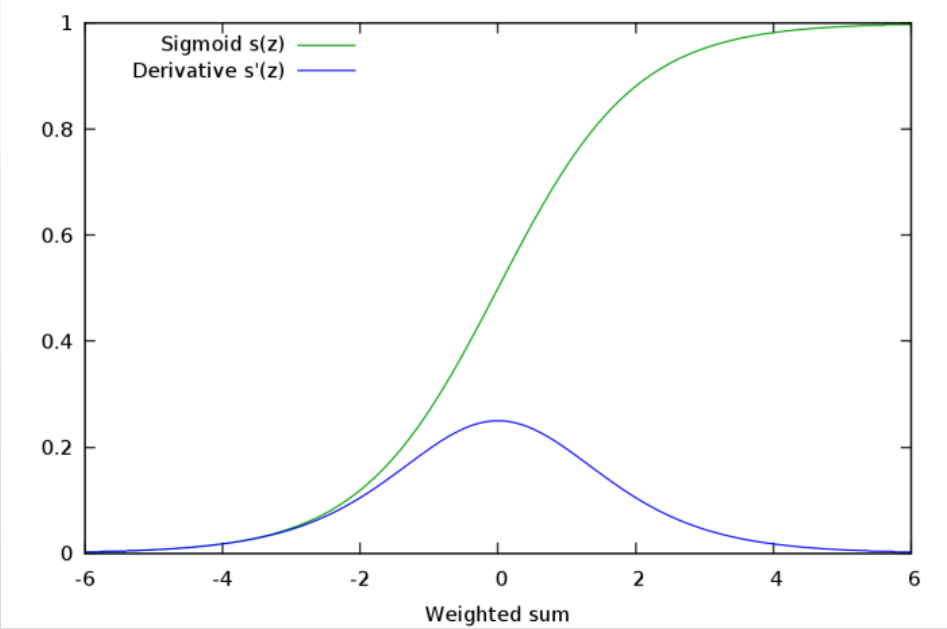


Función de Activación

- Cada neurona tiene asignado un valor.
- Las neuronas de la capa de entrada tomarán los valores indicados por el objeto a procesar.
- El valor de las neuronas de las capas restantes será el producto punto entre los valores de las neuronas de la capa anterior y los pesos de los arcos que las comunican.
- Es recomendable mantener todos los valores en el rango [0..1]
- Para ello, se selecciona una función de activación que sea *diferenciable* para ejecutar el proceso inverso (*back*).

Función de Activación

- Una función de activación muy utilizada es la *Sigmoidea*.
  - $s(z) = \frac{1}{1+ e^{-z}}$
- Y su derivada es:
  - $s'(z) = z(1 - z)$
  - Siempre positiva



Cálculo de la salida

- El algoritmo que calcula la salida supone 3 capas:
  1. Capa de entrada. El tamaño es igual a la longitud del patrón (25).
  2. Capa oculta. Se sugiere un tamaño igual al promedio entre los tamaños de las capas de entrada y salida.
  3. Capa de salida. El tamaño es igual al número de elementos de la salida esperada (1, 2).
- Datos globales:
  a. Conjunto de objetos de entrenamiento.
  b. Salidas esperadas del conjunto anterior.
  c. Pesos de los arcos que conectan a las capas de entrada y oculta.
  d. Pesos de los arcos que conectan a las capas oculta y de salida.
  e. Los valores de la capa oculta.

Cálculo de la salida

- El método recibirá el índice del patrón a procesar.
  - Servirá para obtener el valor $ivalue_i$ de cada neurona *i* de la capa de entrada.
- Calcular el valor de cada neurona *k* de la capa oculta:
  - $hvalue_j = sigmoid\left(\sum ivalue_i \cdot ihweight_{ij}\right)$
- Calcular el valor de cada neurona *k* de la capa de salida:
  - $ovalue_k = sigmoid\left(\sum hvalue_j \cdot howeight_{jk}\right)$

Ajuste de los pesos

- El método recibirá el índice del patrón que fue procesado y las salidas obtenidas por el método anterior.
- Los pesos de las aristas se ajustarán siguiendo el algoritmo *back-propagation*.
- Al peso de cada arco (*j*, *k*) se le sumará el valor de la neurona *j* multiplicado por la tasa de aprendizaje y un valor *delta* que corresponde a la neurona *k.*
- Lo nuevo con respecto a la sesión anterior es la presencia de un *delta* por cada neurona (no de la capa inicial).

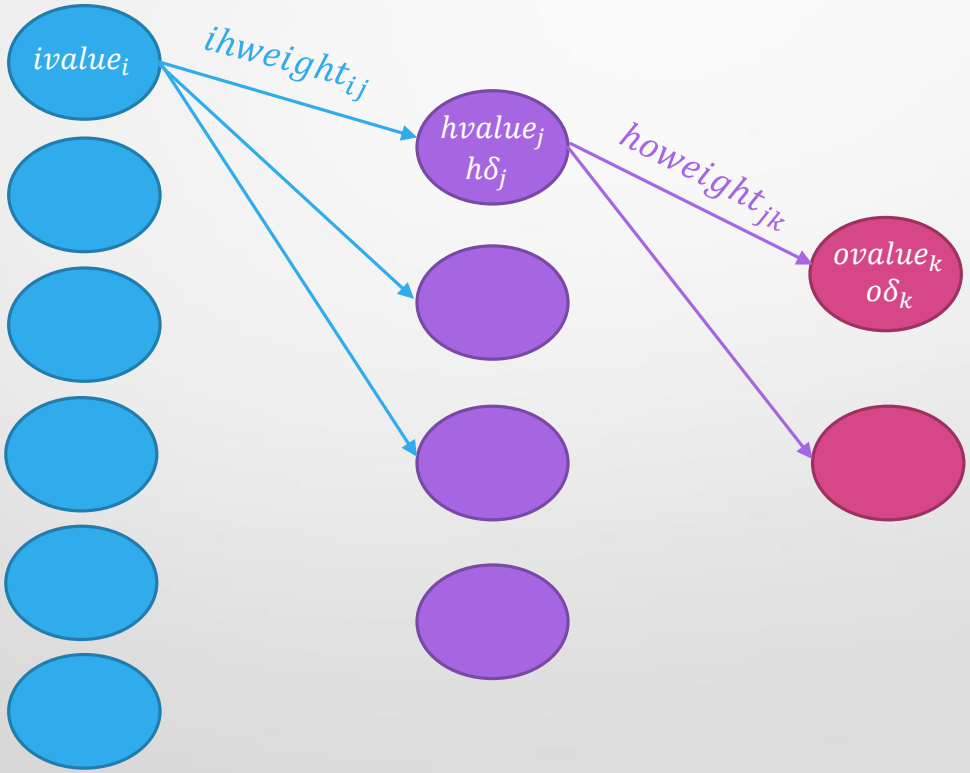
Ajuste de los pesos

- El valor *delta* representa un factor de error que se propagó hacia atrás desde cada neurona de salida, y que fue ocasionado por diferencias entre las salidas esperada y obtenida.
- Calculando *delta* de cada neurona *k* de la capa de salida:
  - $o\delta_k = sigmoid'(ovalue_k) \cdot (error_k)$
  - $ovalue_k$ es el valor obtenido en el método anterior
  - $error_k = eovalue_k - ovalue_k$ es la diferencia entre la salida esperada y la salida obtenida.

Ajuste de los pesos

- Calculando el peso de cada arco (*j*,*k*) entre las capas oculta y de salida:
  - $howeight_{jk} = LR \cdot o\delta_k \cdot hvalue_j$
- Calculando *delta* de cada neurona *j* de la capa oculta:
  - $h\delta_j = sigmoid'(hvalue_j) \cdot \sum o\delta_k \cdot howeight_{jk}$
- Calculando el peso de cada arco (*i*,*j*) entre las capas de entrada y oculta:
  - $ihweight_{ij} = LR \cdot h\delta_j \cdot ivalue_i$

Vistazo al perceptrón



Entrenamiento

- Establecer pesos aleatorios a todos los arcos en el rango [-0.1 .. 0.1].
- Para evitar ciclos infinitos, delimitar el número máximo de *epochs* que se ejecutarán (≤ 10,000). Lo siguiente es un *epoch*:
- Por cada patrón del conjunto de entrenamiento:
  - Calcular la salida.
  - Si la salida es correcta,
    - Incrementar el número de correctos.
    - Terminar si el número de correctos es igual al número de patrones.
  - Si no,
    - Correctos = 0
    - Ajustar los pesos.

¿La salida es correcta?

- Calcular la diferencia absoluta entre cada valor de la capa de salida obtenida y la salida esperada.
- Si todas las diferencias fueron menores a un valor de umbral, la salida es correcta.
- En otro caso, es incorrecta.

Valores de umbral

- Se manejarán dos valores de umbral:
  1. Para probar la red neuronal. El valor más pequeño posible que haga que una salida pueda pertenecer a dos clases. También, el punto medio entre dos clases consecutivas:
     - Clases: 0.2, 0.4, 0.6, 0.8, 1.0. Umbral: 0.1.
       - ¿La salida 0.5 a qué clase pertenece? ¿0.4, 0.6?
     - Clases: 0.0, 1.0. Umbral: 0.5.
  2. Para entrenar al perceptrón. Menor al otro umbral para lograr mayor precisión (un medio, un cuarto, un octavo, …)
     - Mientras menor sea, obligamos a que las salidas obtenidas se aproximen más a las esperadas.