INGENIERIA INFORMATICA
Escuela Politécnica Superior
Universidad Autónoma De Madrid

Práctica Neurocomputación

P1

David Teófilo Garitagoitia Romero
Daniel Cerrato Sánchez

01/03/2023

Índice de Contenidos

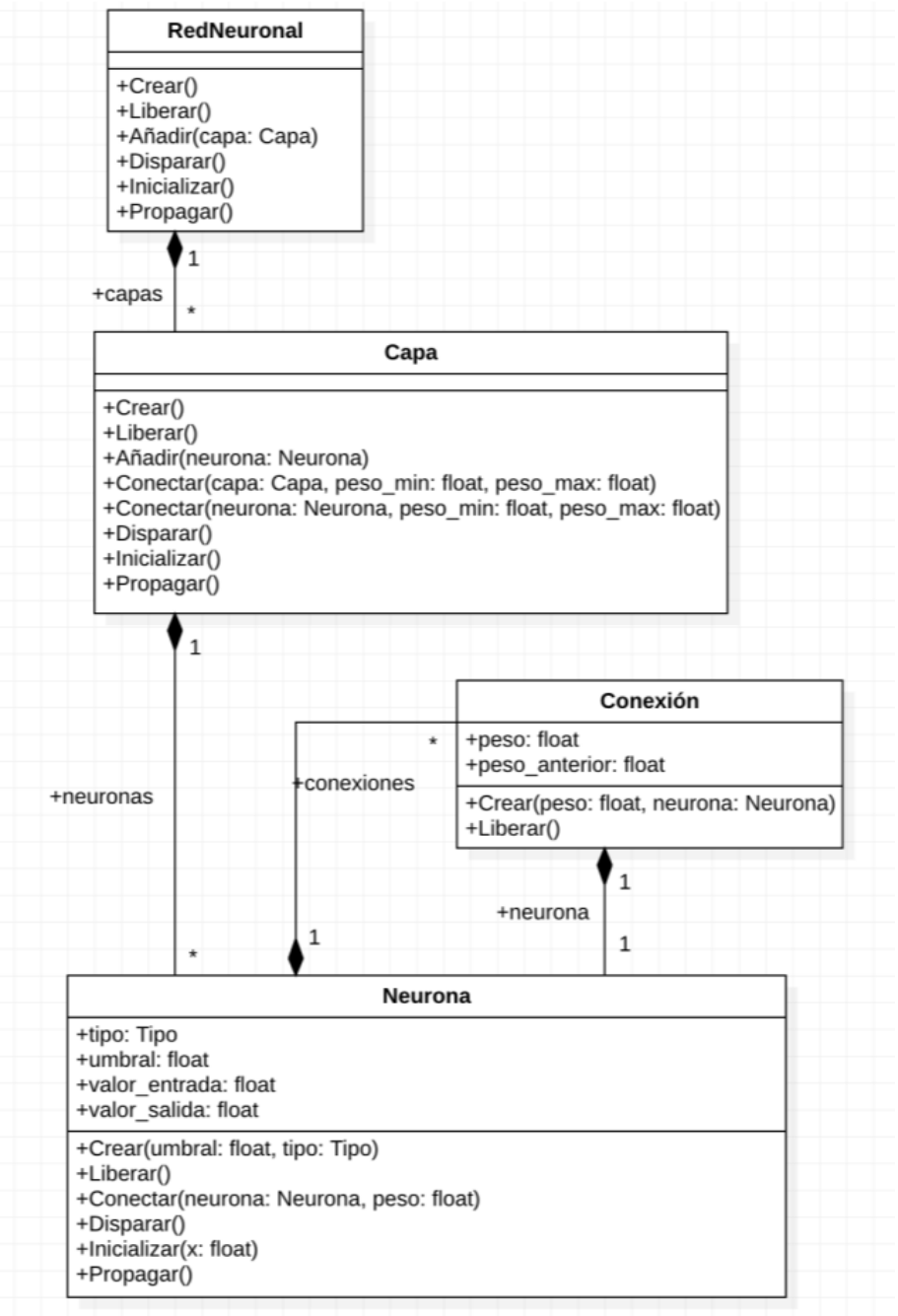
Lista de Tablas y Figuras

1. Introducción

Práctica 1 de la asignatura de redes neuronales.

Objetivos: implementación de una librería para el manejo de redes neuronales y resolver las cuestiones planteadas a continuación.

Ilustración 1. Librería redes neuronales



2. Neuronas de McCulloch-Pitts.

Diseñad e implementad una red de McCulloch-Pitts que detecte el movimiento horizontal de un objeto.

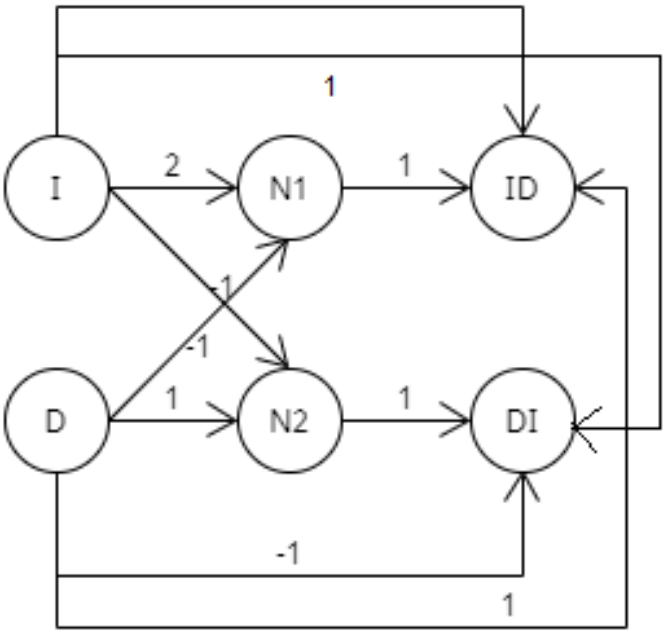
La red tendrá dos neuronas de entrada (Izquierda I y Derecha D) y dos neuronas de salida (una indica movimiento de derecha a izquierda DI y la otra movimiento de izquierda a derecha ID).

Para que se detecte un movimiento de izquierda a derecha, en el instante de tiempo t la neurona de entrada de la izquierda debe detectar actividad pero no la de la derecha, y en el instante t + 1 es solo la neurona de entrada de la derecha la que detecta movimiento. La detección del movimiento de derecha a izquierda será parecido. En cualquier otro caso, las neuronas de salida estarán inactivas.

Ilustración 2. Tabla Ej 1

t	I	D	ID	DI
1	1	0	0	0
2	0	1	0	0
3	1	1	1	0
4	0	1	0	0
5	1	0	0	0
6	0	1	0	1
7	0	0	1	0
8	1	0	0	0
9	1	0	0	0

Ilustración 3. Red de McCulloch



Nuestro diseño se basa en una red de dos capas (oculta + salida)

En la capa de entrada existen dos neuronas que actúan como "sensores de movimiento"

En la capa de salida existen otras dos neuronas que nos indican si el movimiento ha sido de izq a der o al revés. En caso de recibir estímulos incorrectos, devuelve (0,0)

En la capa oculta existen otras dos neuronas donde se detecta si, en el primer tiempo, se ha activado tan solo una de las neuronas de entrada

Para ello, estas funcionan como una puerta lógica NAND. Se invierten dependiendo de qué "sensor" queremos aislar. Además, esta capa nos sirve para temporizar el reconocimiento (según piden en la práctica)

Las neuronas de salida reciben 3 entradas:

entrada de la capa oculta: si está activa es porque se ha activado solo una neurona de entrada en el tiempo t

entradas de la capa entrada: funcionan como un tipo de NAND para comprobar que el tiempo t+1 sólo se ha activado la neurona contraria a la del tiempo t

Ejemplos de ejecuta_mp:

En la versión de prueba del programa hemos añadido varias pruebas que demuestran que el correcto funcionamiento de la red:

Prueba 1 (Tiempos 1-3): Se demuestra la detección del movimiento de izq a der (1,0)

Prueba 2 (Tiempos 4-6): Se demuestra la detección del movimiento de der a izq (0,1)

Prueba 3 (Tiempos 7-11): Se demuestra la detección de varios cambios de sentido seguidos

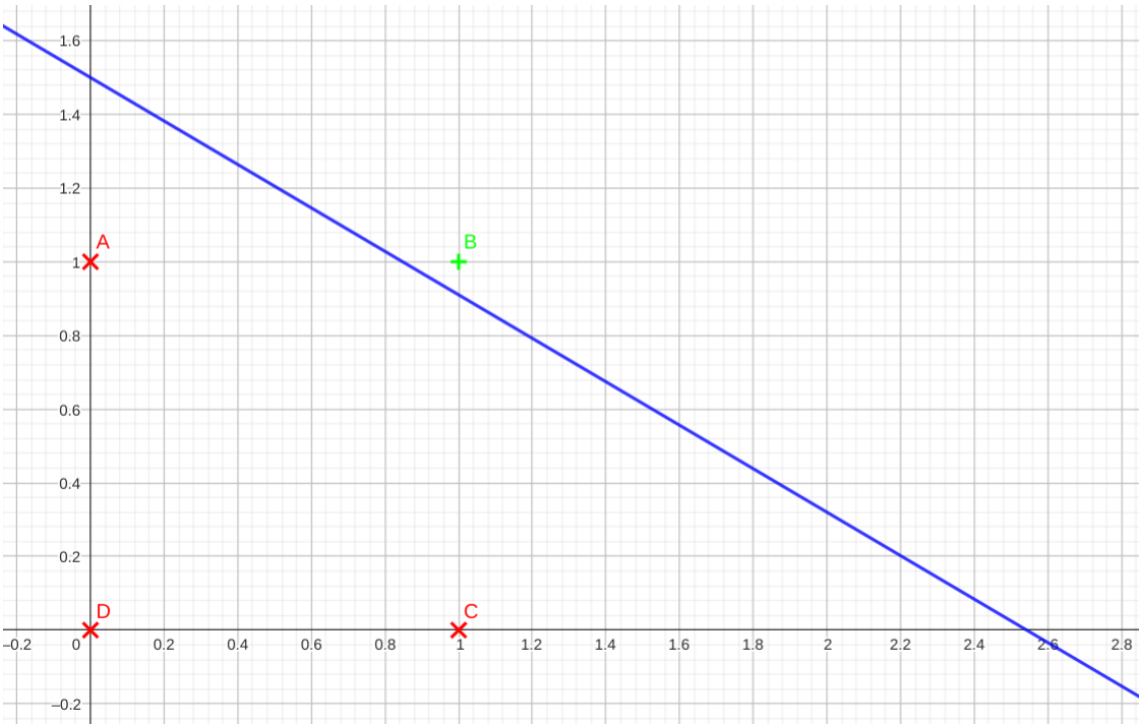
Resto de pruebas (Cada 3 tiempos): Se demuestra que si se activan las dos neuronas de entrada en algún momento de los 2 tiempos de reconocimiento, se detecta el error y devuelve (0,0)

3. Perceptrón y Adaline

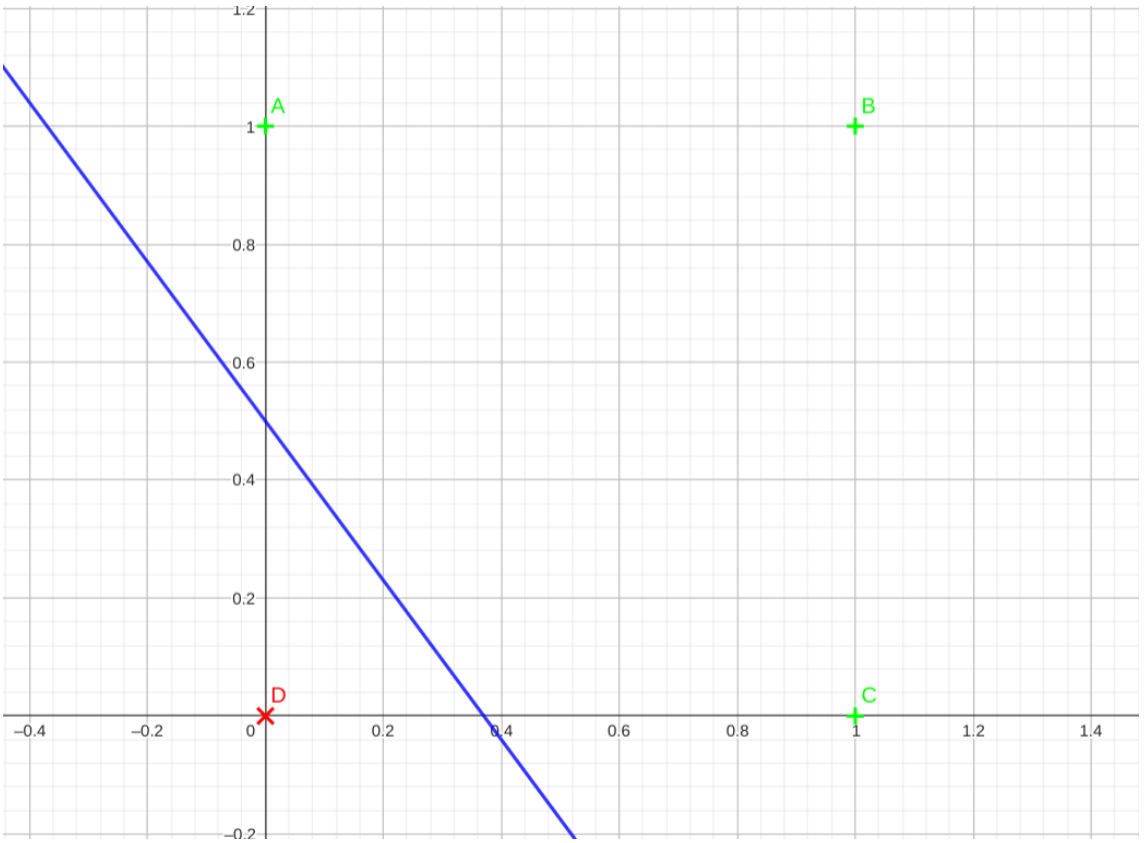
Entrenad ambos tipos de redes con los problemas lógicos and.txt, or.txt, nand.txt y xor.txt, leyendo los ficheros en Modo 2 y comprobar con estos problemas que las implementaciones son correctas.

Adaline:

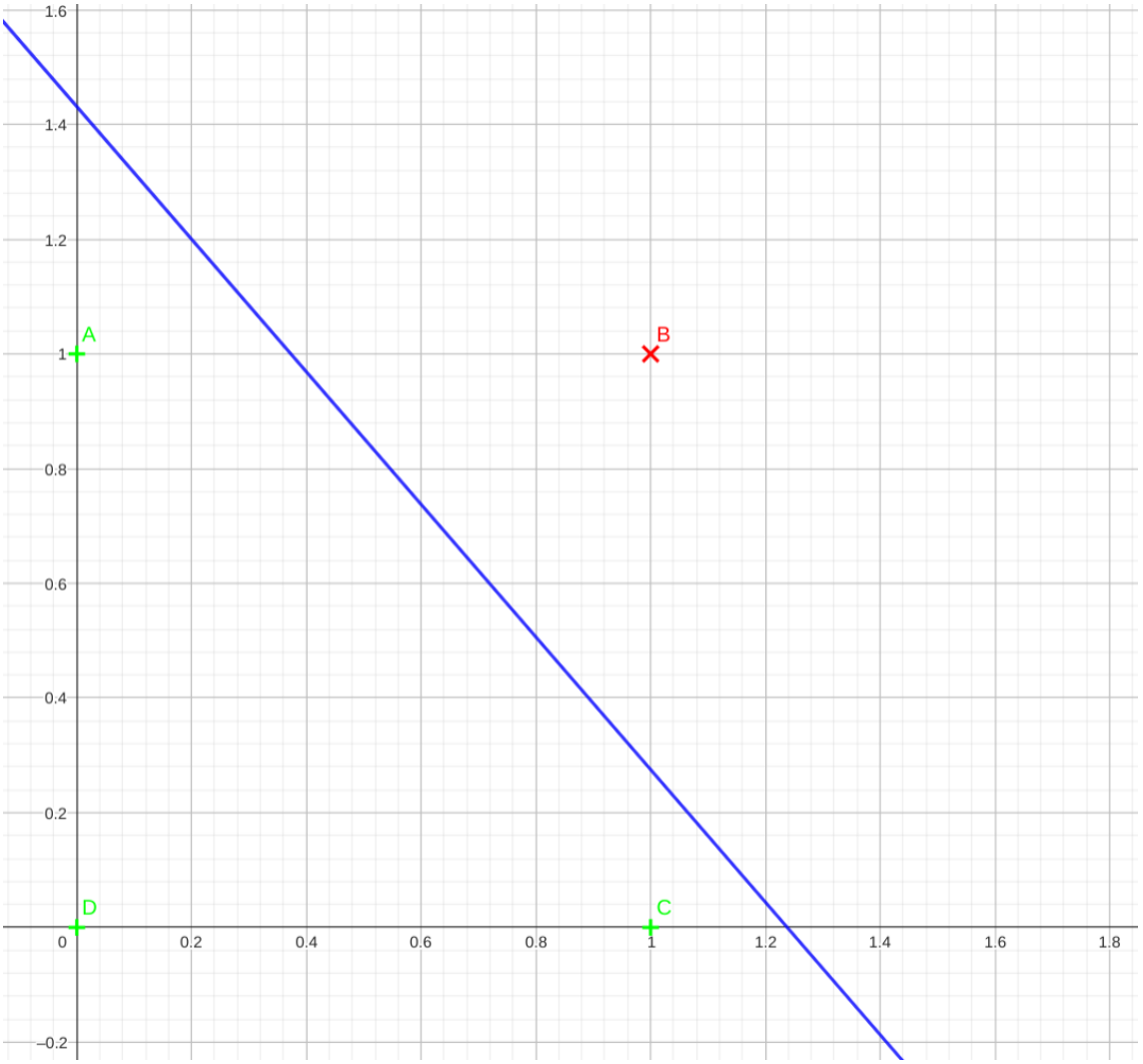
AND -> 0.59x1 + x2 - 1.5 = 0



OR -> 1.35x1 + x2 - 0.5 = 0



NAND -> -1.26x1 - 1.09x2 + 1.56 = 0

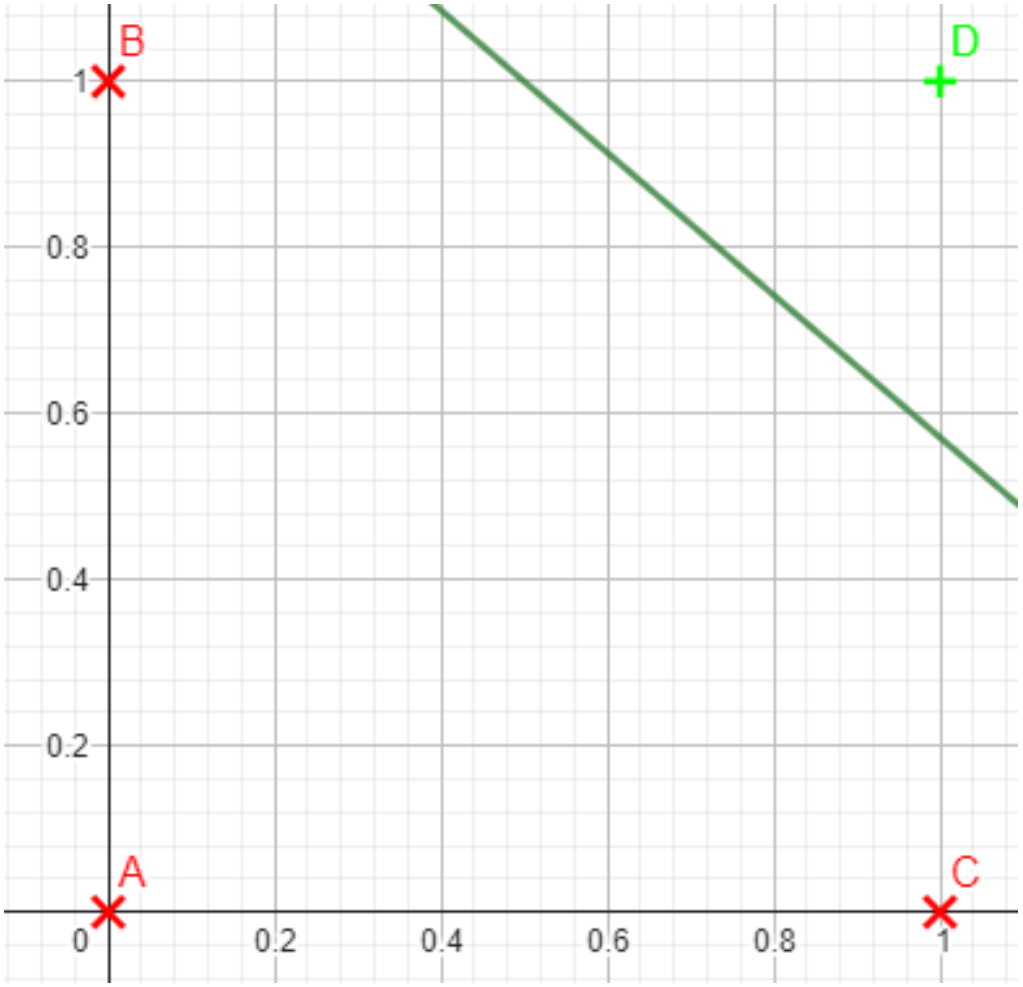


XOR -> 0.52x1 = 0

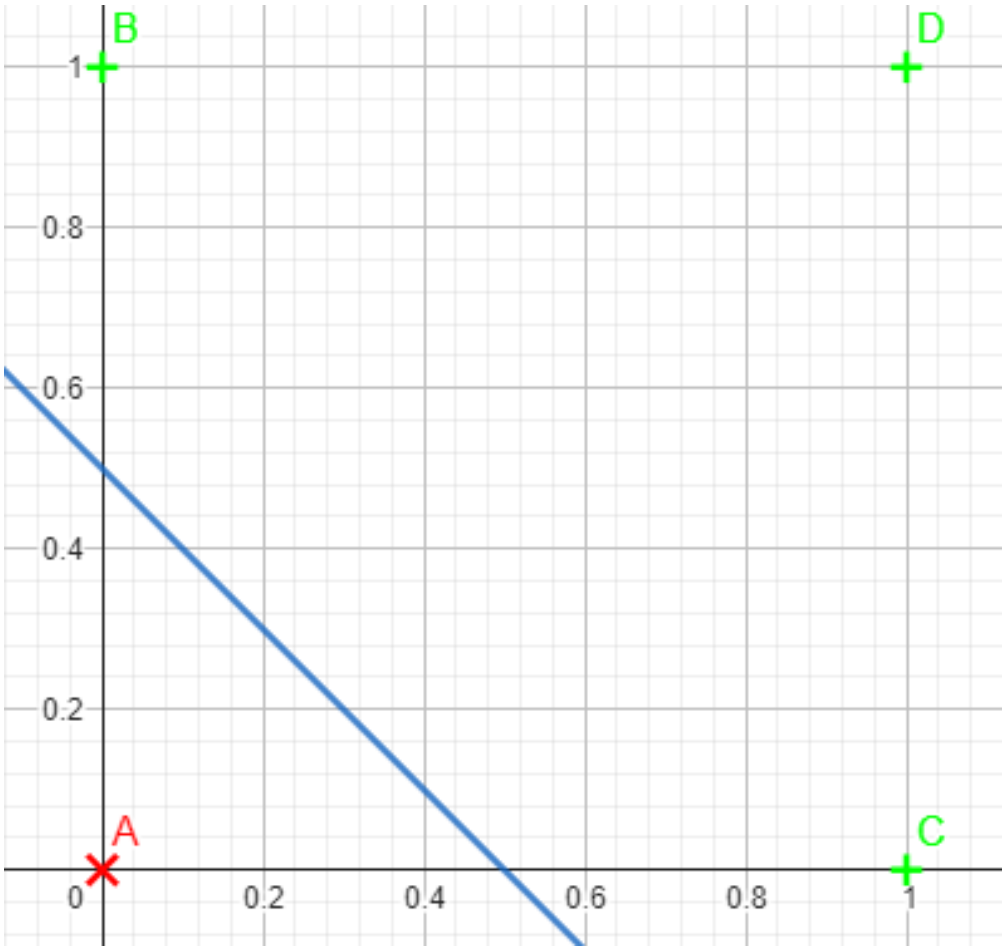
No es posible darle solución al problema del XOR al no ser linealmente separable, para solucionarlo deberíamos emplear una capa oculta

Perceptrón:

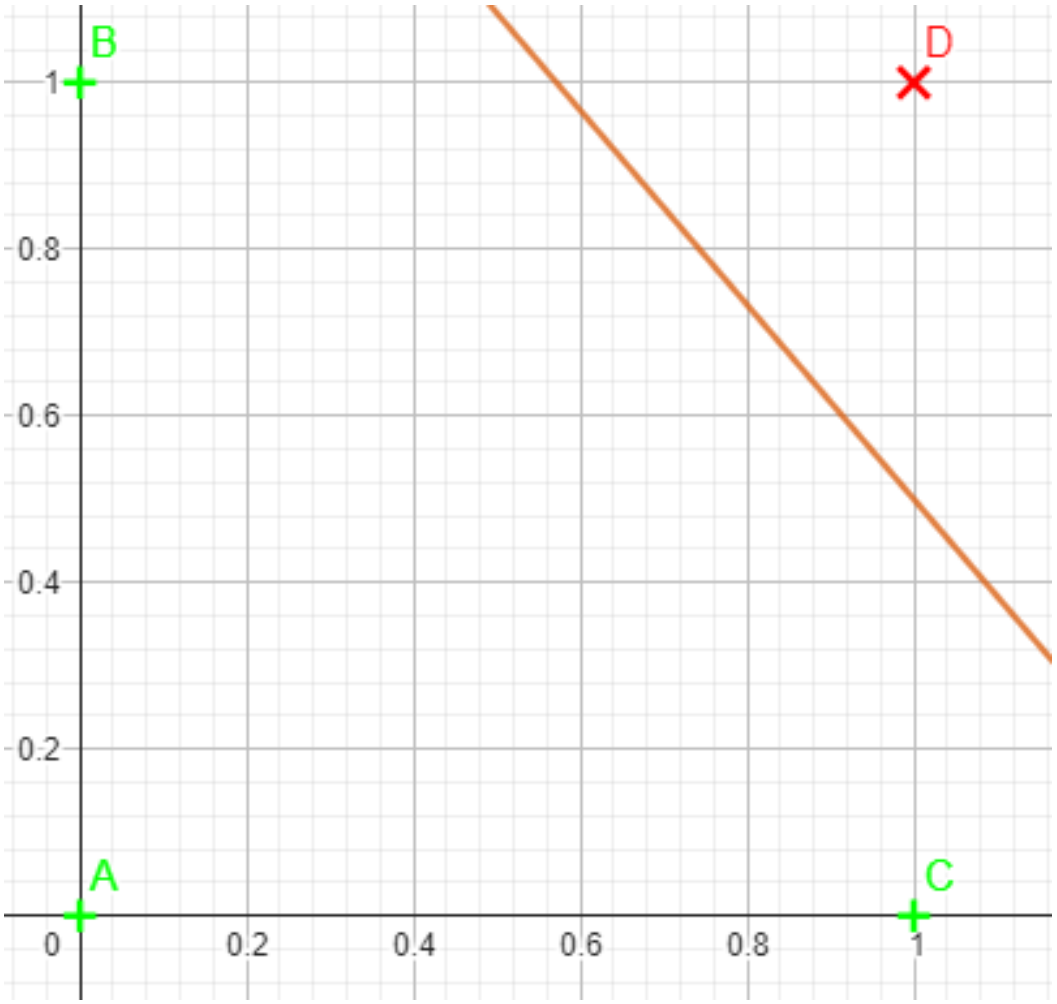
AND -> 1.2x1 + 1.4x2 - 2 = 0



OR -> 1.2x1 + 1.2x2 - 0.6 = 0



NAND -> -1.4x1 - 1.2x2 + 2 = 0



XOR -> 0 = 0

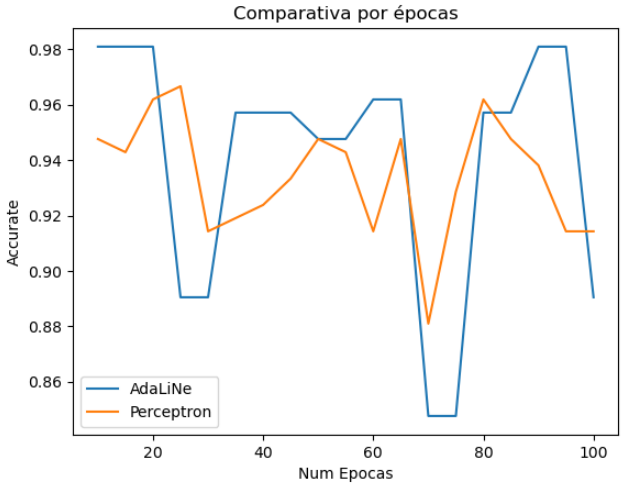
No es posible darle solución al problema del XOR al no ser linealmente separable, para solucionarlo deberíamos emplear una capa oculta

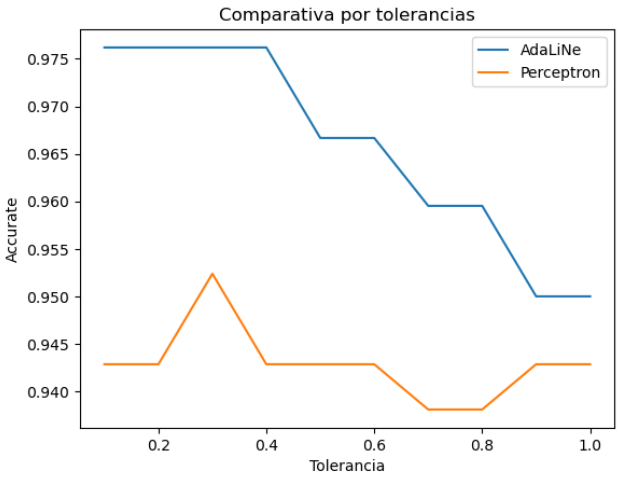
4. Problema real 1

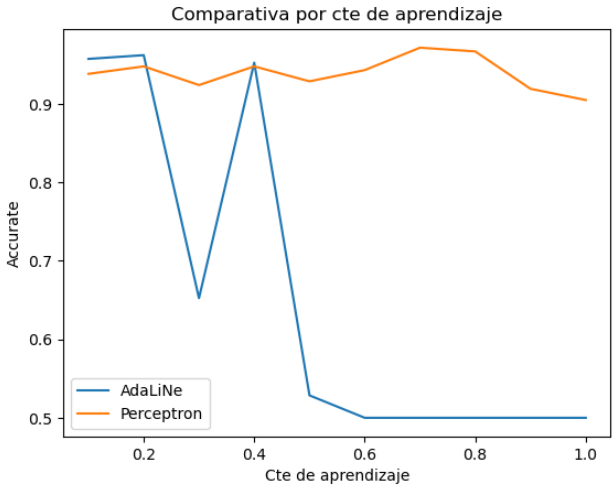
Diseñad un Perceptrón y un Adaline que solucione problema_real1.txt. Los parámetros que afectan a las redes y el aprendizaje (épocas, umbrales, tasa de aprendizaje…) deben ser incluidos como argumentos de los programas

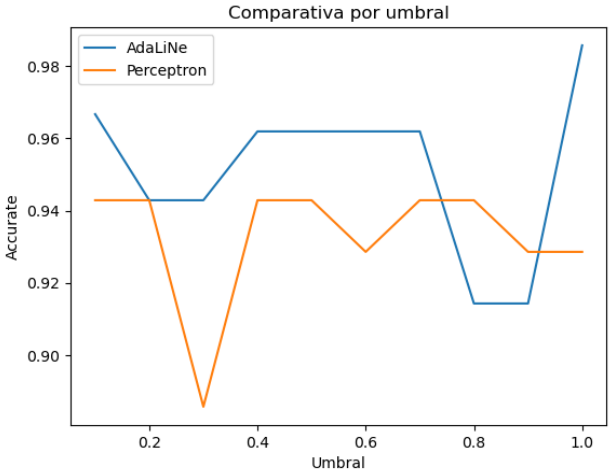
Para este apartado, hemos realizado un hiperparameter tunning sencillo, modificando en cada bucle un solo parámetro como se indica en la práctica. La configuración final será la combinación de los parámetros que mejor resultado han dado por separado.

Para esto, se han establecido valores constantes para cada parámetro, a saber: porcentaje de entrenamiento = 0.7, número de épocas = 100, tolerancia = 0.1, constante de aprendizaje = 0.2 y umbral = 0.5. Estos han sido los resultados









Como se puede observar, los mejores resultados se obtienen con la siguiente combinación:
Adaline => Épocas = 95, Tolerancia = 0.1, Cte Aprendizaje = 0.2. El umbral es siempre 0.
Perceptrón => Épocas = 80, Cte Aprendizaje = 0.7, Umbral = Varios. La tolerancia no se usa.

El error cuadrático medio se ve reflejado en el score final. Por lo general, ambas redes tienen un alto score, pero se puede apreciar que AdaLiNe suele estar por encima. Esto puede ser debido a que este tipo de redes aprende usando la diferencia entre la salida esperada y la obtenida; no como pasa en Perceptrón, el cuál usa solo el valor esperado, sin saber que valor ha obtenido. Tiene algo más de pecado sabiendo que puede obtener 3 resultados distintos por neurona.

Como podemos ver en las gráficas, con el número de épocas el score varía bastante. Aún así, es preferible escoger una cantidad algo mayor de lo necesario, puesto que ambas redes pueden parar antes de tiempo porque no consigan aprender “más”. En AdaLiNe, el parámetro “tolerancia” puede ser más o menos influyente en este tema: una tolerancia baja, tenderá a usar todas las épocas, mientras que una alta corta el entrenamiento antes, por lo general.

AdaLiNe usa un umbral con valor 0 siempre, por lo que las fluctuaciones que se aprecian en su gráfica no vienen dadas por este parámetro; además de que en el entrenamiento se usa el valor de entrada sin activar de la neurona de salida, en vez de pasarla por un filtro usando el umbral, como pasa en la fase de explotación. En Perceptrón, pueden marcar la diferencia en algunos casos concretos donde la clasificación no termina de quedar clara. Un umbral más pequeño fuerza la decisión por una clase u otra, mientras que uno más alto, permite dejar casos en la banda de indecisión.

Por último, la constante de aprendizaje. Este parámetro es decisivo a la hora de entrenar, puesto que modifica los pesos en proporción a su valor. Un valor muy grande puede hacer que el entrenamiento no sea efectivo, mientras que uno más pequeño puede requerir de más épocas para alcanzar el valor óptimo. En ambas redes se ha preparado una constante opcional aleatoria en un rango óptimo de valores para el caso en el que no se introduzca una constante de aprendizaje definida por el usuario.

[FINAL DE DOCUMENTO]