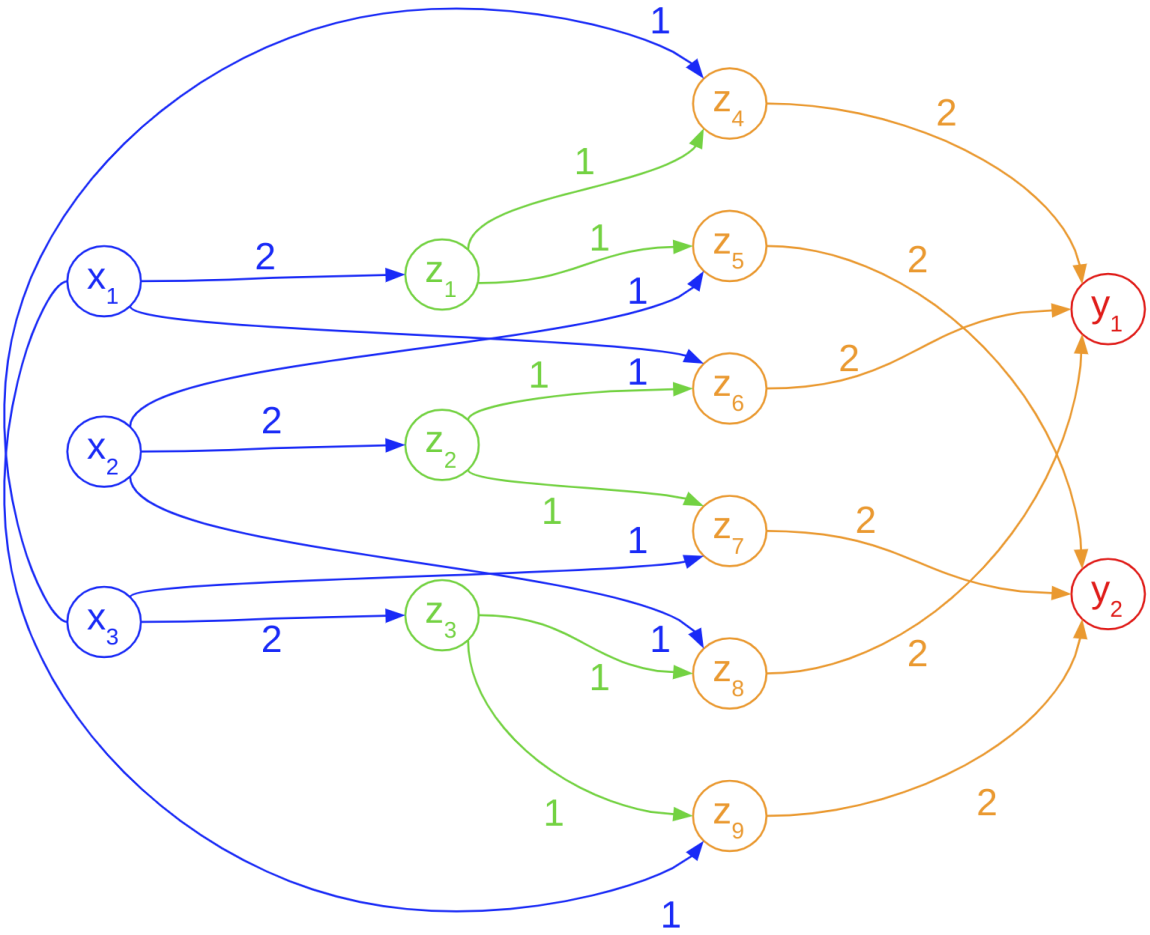
NEUROCOMPUTACIÓN – PRACTICA 1

Daniel Cabañas y Judith Manchón (Pareja 35)

Neuronas de McCulloch-Pitts

Incluir el diseño de la red con los sesgos, conexiones y pesos utilizados:



θ = 2 para todas las neuronas de la red.

Discutir la validez de vuestro diseño e incluir explicaciones de ejemplos que demuestren el correcto funcionamiento interno de cada elemento que conforma el circuito:

Nuestro diseño se ha centrado en dos partes distinguidas:

La primera de ellas está formada por las neuronas de las capas 0 y 1. Las combinaciones de activación entre estas dos capas dan lugar a la activación de neuronas en las posteriores capas. En la capa 0 comenzarán los primeros estímulos, y una unidad de tiempo después tendrán lugar en la capa 1. Al combinarse los estímulos de la capa 1 junto con los nuevos que aparecen en la capa 0 podemos saber si ha tenido lugar una secuencia decreciente, creciente o estática, y en función de esto las neuronas dispararán a unas neuronas u otras de las siguientes capas.

La segunda parte está formada por la capa 2 y la capa 3. La capa 2 consiste en una serie de neuronas cuya activación indica si la secuencia de elementos de entrada a la red neuronal es una secuencia creciente o decreciente. Las neuronas que indican que una secuencia es creciente son las z_4, z_6 y z_8. Estas neuronas estimularán la neurona de salida y_1 en caso de ser activadas. Las neuronas que indican que una secuencia es decreciente son las z_5, z_7 y z_9. Estas neuronas estimularán la neurona de salida y_2 en caso de ser activadas. En el caso de que tenga lugar una secuencia estática, es decir, que la secuencia no varíe en una unidad de tiempo, no tendrá lugar la activación de ninguna neurona de la capa 2 y por lo tanto tampoco habrá ninguna activación en la capa de salida (capa 3).

Analicemos un ejemplo con las siguientes entradas:

t1	t2	t3	t4
1	0	1	1
0	1	0	0
0	0	0	0

(El orden de las entradas corresponde al orden de las neuronas, que es un orden decreciente, siendo 1 la entrada para x_1, 0 para x_2 y 0 para x_3 en el instante t1).

1. En el instante t1 se activará la neurona x_1 únicamente por el valor de entrada.
2. En el instante t2 se activará la neurona x_2 (por el valor de entrada) y la neurona z_1 (gracias a la neurona x_1 que fue activada en el instante anterior).
3. En el instante t3 se activará la neurona x_1 (por el valor de entrada), la neurona z_2 (gracias a la neurona x_2 que fue activada en el instante anterior) y la neurona z_5 (gracias a la activación de las neuronas x_2 y z_1 en el instante anterior).
4. En el instante t4 se activará la neurona x_1 (por el valor de entrada), la neurona z_1 (gracias a la neurona x_1 que fue activada en el instante anterior), la neurona z_6 (gracias a la activación de las neuronas z_2 y x_1 en el instante anterior) y la neurona y_2 (gracias a la activación de la neurona z_5 en el instante anterior).
5. En el instante t5 se activará la neurona z_1 (gracias a la neurona x_1 que fue activada en el instante anterior) y la neurona y_1 (gracias a la activación de la neurona z_6 en el instante anterior).
6. En el instante t6 no llegará a activarse ninguna neurona más.

La salida resultante será la siguiente:

t1	t2	t3	t4	t5	t6
0	0	0	0	1	0
0	0	0	1	0	0

(El orden de las salidas corresponde al orden de las neuronas, que es un orden decreciente, siendo la primera fila la salida de y_1, y la segunda fila la salida de y_2).

A continuación, explicamos el significado de la salida resultante:

1. Los tres primeros instantes de tiempo son 0 para ambas neuronas de salida ya que se necesitan dos unidades de tiempo para que la red entregue una salida para cada entrada, y un instante más de tiempo en el que las entradas de la red no han realizado ningún movimiento ascendente o descendente (el instante en el que solo ha tenido lugar la primera entrada en t1).
2. En el instante t4 podemos observar que, gracias a la secuencia descendente que tuvo lugar entre t1 y t2, la salida y_2 se activa indicando un movimiento descendente.
3. En el instante t5 podemos observar que, gracias a la secuencia ascendente que tuvo lugar entre t2 y t3, la salida y_1 se activa indicando un movimiento ascendente.
4. En el instante t5 podemos observar que no se activa ninguna salida, ya que la secuencia entre t3 y t4 es estática, no hay variación.

Perceptrón y Adaline

Descripción breve de la implementación de las redes neuronales:

Red Perceptrón:

Para implementar la red neuronal basada en el Perceptrón hemos seguido los algoritmos explicados en la teoría. Nuestro programa genera una serie de neuronas (basadas en la clase NeuronaPX dentro del archivo Neurona.py) de entrada en función al número de atributos que contenga el archivo, y genera una única neurona de salida (NeuronaPY) que será la encargada de clasificar según la función de activación. Esta neurona establece su umbral a través de un parámetro de entrada (Umbral).

Acto seguido, creamos un enlace (Enlace.py) por cada neurona de entrada y conectamos cada neurona de entrada a la neurona de salida, creando así la red neuronal.

A continuación, establecemos los pesos y el sesgo con sus respectivos valores iniciales para poder dar paso al entrenamiento de la red.

El entrenamiento está basado en el algoritmo que hemos visto en la clase de teoría:

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 (***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.

Así que entrenamos la red neuronal cambiando y variando el valor de los pesos y del sesgo.

Una vez termina el entrenamiento de la red, ya sea porque se ha cumplido la condición de parada especificada en el paso 6 o porque se han cumplido un número máximo de épocas establecido por parámetro (Épocas), mostramos una serie de datos (tasas de error, ECM, matriz de confusión) para el conjunto de datos de entrenamiento (Train) y también mostramos el número de épocas realizada y el motivo de la parada del algoritmo.

Procedemos, a continuación, a probar la red neuronal ya entrenada con un conjunto de datos de prueba (Test):

Para ello ejecutamos únicamente los pasos 3 y 4 del algoritmo de entrenamiento para cada una de las entradas en el conjunto de datos. De este modo clasificamos los datos de entrada, que más adelante compararemos con el objetivo de clasificación (Target) para ver si nuestra red ha clasificado correctamente.

Por último, mostraremos los datos (tasas de error, ECM, matriz de confusión) para el conjunto de Datos de Test.

Red Adaline:

La estructura de una red neuronal Adaline es la misma que la de una red neuronal Perceptrón. Por ello nuestra implementación es muy parecida a la descrita en la red neuronal Perceptrón. La configuración de neuronas de entrada (NeuronaAX) y salida (NeuronaAY) respecto de la red es igual (tantas neuronas de entrada como atributos haya en el conjunto de datos y una única neurona de salida, conectadas mediante enlaces). La estructura de las neuronas de entrada también es la misma, pero la estructura de la neurona de salida cambia ligeramente, puesto que el umbral siempre es 0, y hemos añadido una función de activación específica a la hora de entrenar la red (para nuestra mayor comodidad), que calcula el valor de entrada de la neurona de salida:

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

El ajuste de pesos y del sesgo tiene lugar del mismo modo que en la red Perceptrón.

El entrenamiento de la red se ha basado en la explicación realizada en las clases teóricas sobre el algoritmo de aprendizaje de la red 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.

Como podemos observar, el algoritmo de aprendizaje varía en cuanto al paso 4 (encargado de calcular la respuesta de la neurona de salida), paso 5 (encargado de variar los pesos y el sesgo) y el paso 6 (el cual dicta la condición de parada del algoritmo).

Del mismo modo que con el Perceptrón, vamos entrenando la red neuronal con un conjunto de datos, variando con cada entrada los pesos y el sesgo, y repetimos una serie de épocas.

Cuando se cumple alguna de las dos condiciones de parada (se alcanza el número máximo de épocas o no se supera la tolerancia especificada por parámetro), la red deja de entrenarse y entonces mostramos los datos correspondientes al entrenamiento, que son los mismo que fueron explicados en el Perceptrón.

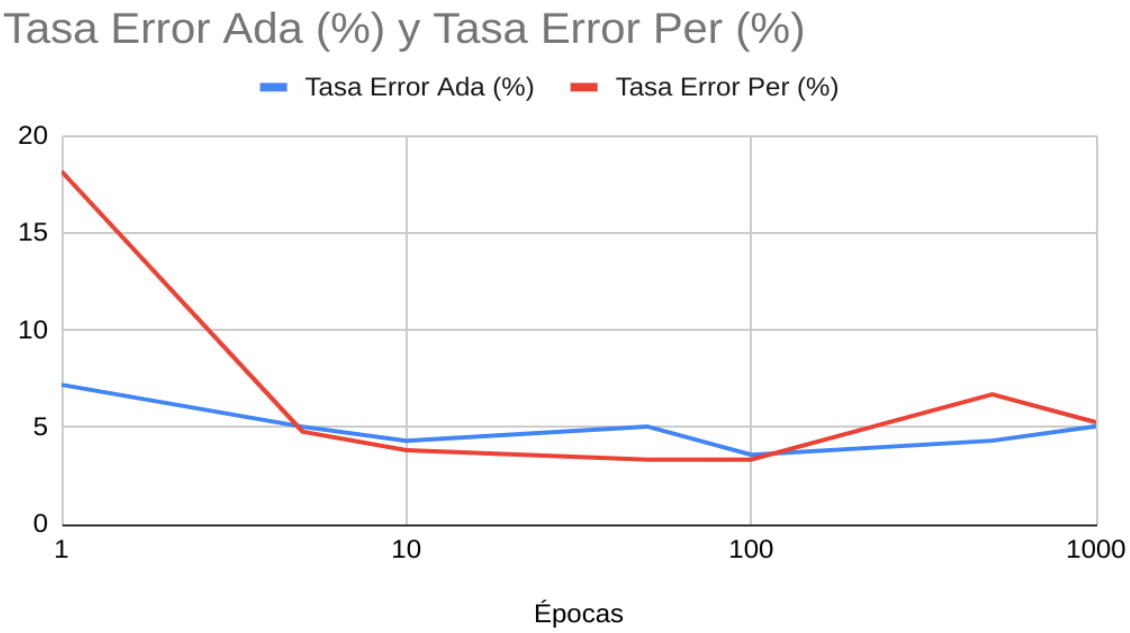
Acto seguido, utilizamos un conjunto de datos de pruebas para ver la eficacia de nuestra red neuronal ya entrenada. A la hora de clasificar las entradas del conjunto de datos de pruebas, la neurona de salida utiliza la función de activación correspondiente a la fase de explotación:

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

Comparamos las salidas generadas de cada entrada con el objetivo de clasificación y obtenemos una tasa de error. También calculamos el ECM y generamos una matriz de confusión.

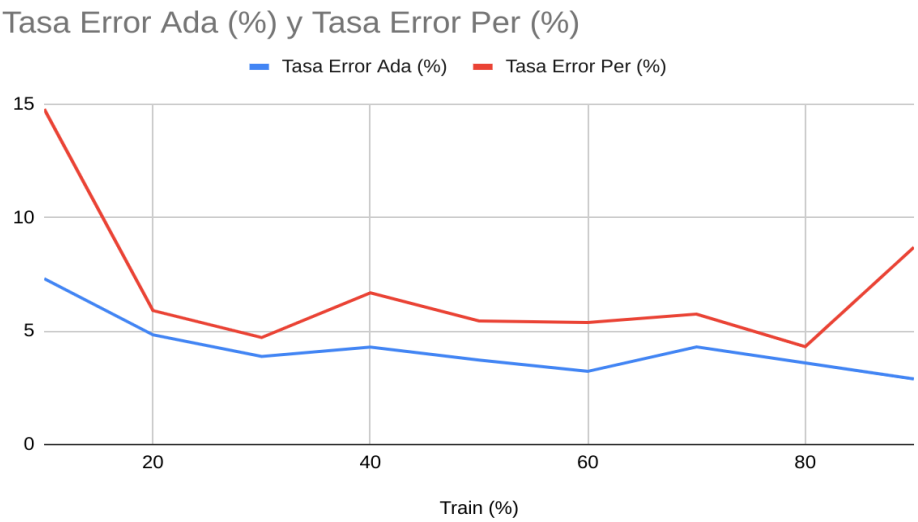
¿Qué influencia presentan en el resultado cada uno de los parámetros que definen el comportamiento de las dos redes? Presentad gráficas que muestren cómo varía el resultado modificando un solo parámetro de control (ej: umbral). Utilizad el problema_real1 y el Modo 1:

Número de Épocas máximo:



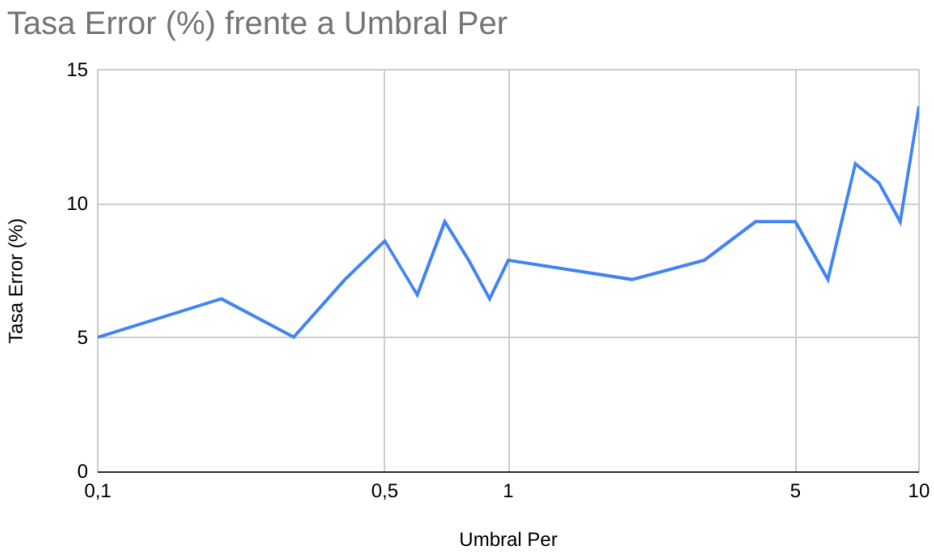
Podemos ver en la gráfica que, a partir de un cierto número de épocas, la red no consigue mejorar más. Por lo tanto, hemos decidido dejar el número de épocas en 50 para Perceptrón y en 10 para Adaline (el cual suele detener su algoritmo de entrenamiento a las 5 épocas).

Porcentaje del conjunto de datos de entrenamiento:



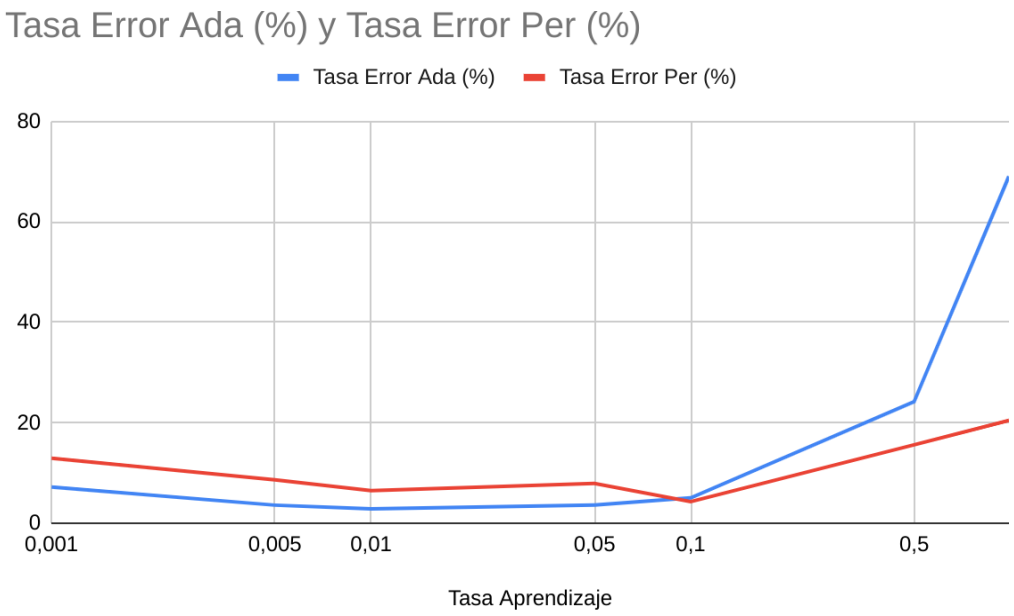
Con esta gráfica podemos comprender que, si el conjunto de datos de entrenamiento no es muy grande, la red neuronal no entrena correctamente para el conjunto de pruebas. Si el conjunto de datos es muy grande, los resultados tienen una tendencia irregular leve. Por ello hemos decidido que una correcta elección es elegir un 80% de datos para Train.

Umbral (solo Perceptrón):



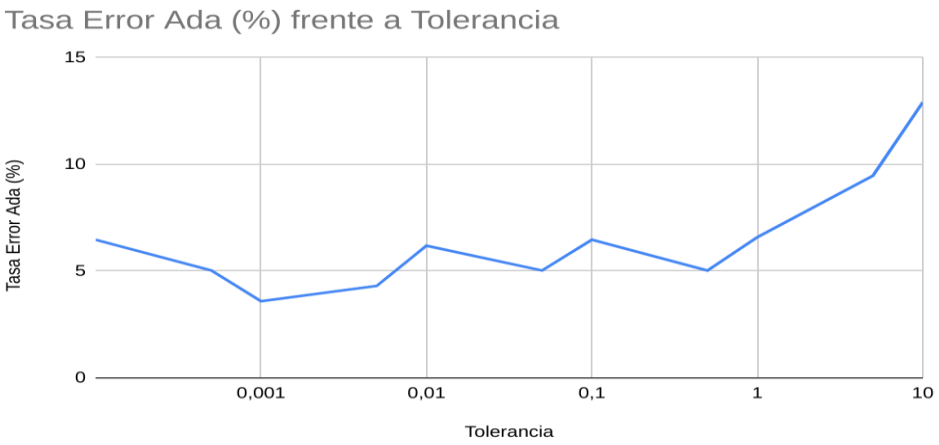
Al realizar diversas pruebas, concluimos que cuanto más alto es el umbral, más irregular es el resultado. La variación de umbral no ha tenido mucho peso a la hora de las pruebas, y creemos que esto se debe a que los pesos y el sesgo se adaptan al umbral a la hora de entrenarse. Creemos que un buen umbral sería 0,2.

Tasa de aprendizaje:



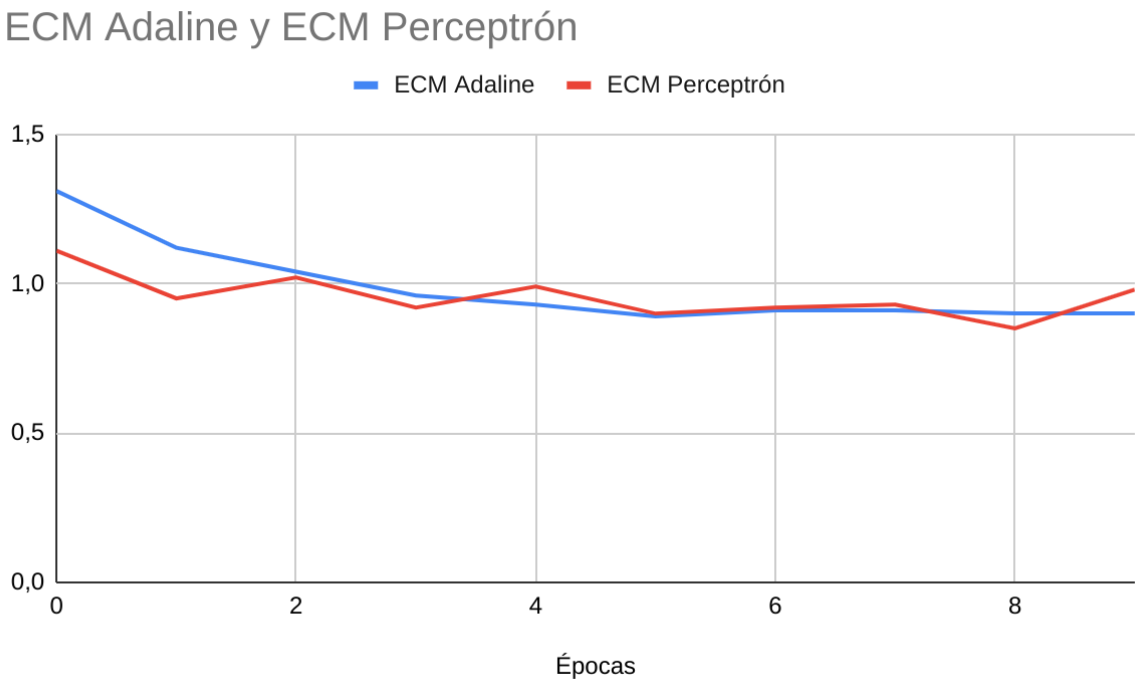
Con estos datos observamos que la tasa de aprendizaje mejora la eficacia del algoritmo cuanto más pequeña es hasta un punto donde se empieza a volver un poco ineficiente (sobre todo porque cuanto más pequeña es, mayor es el número de épocas). Sin embargo, si la tasa es muy grande, aumenta el error en la predicción de la red. Por ello concluimos que unos buenos valores para esta tasa podrían ser 0,01 - 0,03. En la gráfica además se observa que la tasa tiene un impacto mucho mayor en Adaline que en Perceptrón.

Tolerancia (solo Adaline):



Al apreciar esta última gráfica podemos ver que si estas es muy pequeña tiende a ser ineficiente. En cambio, si los valores de tolerancia son muy altos la red no consigue clasificar correctamente, alcanzando valores de la tasa de error de hasta un 13%. Por ello creemos que el mejor valor para fijar la tolerancia es de 0,01.

Compara ambas redes, ¿Qué sucede con el Error Cuadrático Medio (ECM) en cada una de ellas?



Esta gráfica es el resultado de calcular el Error Cuadrático Medio para el conjunto de entrenamiento durante 10 épocas. Podemos observar que el error varía levemente al comienzo y enseguida se regula alcanzando un valor constante para Adaline, mientras que Perceptrón sigue regulando de forma indefinida pero manteniendo un valor constante.

Los valores del ECM en Test para Adaline y Perceptrón son los siguientes:

- Adaline: 1.12
- Perceptrón: 1.35

Intenta predecir ahora los problemas lógicos “nand.txt”, “nor.txt” y “xor.txt” (Modo 2). ¿Es posible al 100%? En caso de no ser posible, ¿Cuál es el problema y como puede ser solucionado?

Realizamos en modo 2 la ejecución de los tres problemas lógicos tanto con el Perceptrón como con el Adaline y los resultados son iguales para ambas redes.

En el caso de los problemas “nand.txt” y “nor.txt” se predice correctamente al 100.

En cambio, cuando intentamos predecir “xor.txt” el resultado es totalmente diferente. En el caso de Perceptrón el resultado es del 100% de errores y en el Adaline suele ser del 75% de tasa de error.

Esto se debe a que el problema “nand.txt” y “nor.txt” son separables linealmente pero el problema “xor.txt” no lo es. Por tanto, no podemos resolverlo con este tipo de red, aunque debería poder resolverse con más capas, por ejemplo, utilizando una red Madaline.