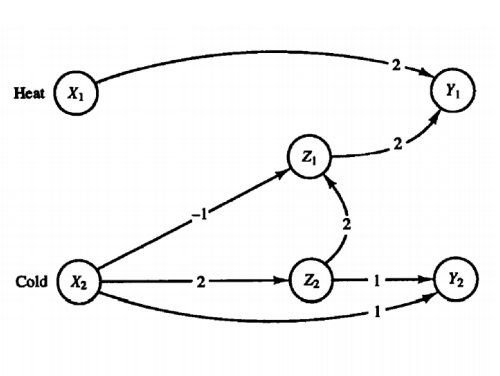
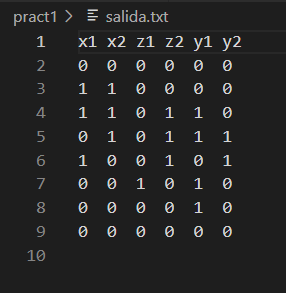
Ej 2.1

El diseño de la red es el siguiente donde las neuronas se dividen en una capa de las entradas(x1, x2), una capa oculta(z1, z2) y una capa de salida(y1, y2) con el peso de las conexiones que se aprecian. Las neuronas de la capa de entrada se tratan de neuronas directas donde su entrada se pasa directamente a través de la conexión, mientras que el resto de las neuronas del resto de capas se tratan de neuronas McCulloch-Pitts que cuentan con un umbral de valor 2.



El diseño es correcto ya que por un lado está la capa de entrada cuya función será transmitir las entradas a las capas posteriores, algo que se consigue al estar está constituida de neuronas directas y el resto de las neuronas de las siguientes capas al ser de McCulloch-Pitts su disparo cambia y saca 1 cuando se supera el umbral de 2 o 0 en caso contrario. Todas estas capas se añaden a la red neuronal, formándose así el diseño correcto de la red. Como prueba de su correcto funcionamiento ante la entrada de frio-calor que se nos proporciona desde el enunciado de la práctica se van obteniendo las sucesivas líneas de la siguiente imagen tras inicializar, propagar y disparar en la red neuronal, las cuales se corresponden con la solución esperada.



Con una entrada saca el valor detectado de ella, si es calor, tras el siguiente disparo, si es frio tras el segundo disparo siguiente. Se basa en estudios de fisiología sobre la percepción que tenemos tras aplicar ciertos estímulos de frio o calor.

La secuencia que seguimos es inicializar toda la red, inicializar la entrada con su valor correspondiente, dispara la capa, inicializa la capa disparada y se propaga hacia siguiente capa. A diferencia respecto otras redes neuronales, todas las capas de McCulloch-Pitts se disparan y propagan a la vez, por lo tanto, se puede llamar el método *dispara*, *inicializar* y *propagar* de la clase RedNeuronal que itera todas las capas, pero para redes como puede ser Perceptrón o Adaline, habría que seguir la secuencia de que, dispara una capa, inicializa esa capa, propaga hacia siguiente capa, dispara esa siguiente capa, inicializa, propaga etc.

Ej 4.1.1

Para implementación del perceptrón y Adaline, se han utilizado los mismos criterios de parada como han explicado en la clase de teoría, pero como hay casos como XOR que genera un bucle infinito puesto que la red no pueda resolver, se ha implementado otro criterio de parada llamado *época*, que cuando llega la época indicada, se para el entrenamiento.

En perceptrón se utiliza una tasa de aprendizaje igual a 1 y umbral a 0.2 como lo habíamos visto en clase.

En Adaline se utiliza una tasa de aprendizaje igual a 0,3 puesto que 0,1 <= 3 neuronas \* 0,3 <= 1, el umbral es igual a 0.2 y la tolerancia igual a 0.22(mejor tolerancia para el caso del AND utilizando tasa de aprendizaje 0.3). Esta tolerancia es tan alta porque el criterio de parada para Adaline es que cuando el cambio de peso más grande es menor que la tolerancia, pero el cambio de peso es , donde suele ser un número mayor que 0.1 puesto equivale -1 o 1 y más de lo mismo.

Existe un cambio de nombre respecto los métodos indicados por UML del enunciado, que son los métodos *conectar* de la clase Capa, ya que en Python no se puede saber qué tipo de clase es el atributo *capa* o *neurona*. Por lo tanto, para resolver este problema, el método *conectar* para las neuronas se va a llamarse como *conectar\_neurona*.

AND frontera decisión:

Perceptron -> 2\*x1 + 3\*x2 - 4 = 0

Adaline -> 0,8\*x1 + 1\*x2 -1,5 = 0

OR frontera decisión:

Perceptron -> 2\*x1 + 2\*x2 -1 = 0

Adaline -> 1.2\*x1 +1\*x2 -0.5 = 0

NAND frontera decisión:

Perceptron -> -3\*x1 -2\*x2 + 4 = 0

Adaline -> -1.4\*x1 -1.1\*x2 + 1.7 = 0

XOR frontera decisión:

Perceptron -> -x1 + 0\*x2 -1 = 0

Adaline -> -0,23\*x1 -0,1\*x2 -0,5= 0

Los problemas lógicos son solucionables en todos los casos salvo en el caso de XOR, ya que no es linealmente separable. Se podría solucionar con ciertas redes neuronales de más de una capa, como el perceptrón multicapa, ya que con redes de una capa no se pueden resolver problemas que no sean linealmente separables.

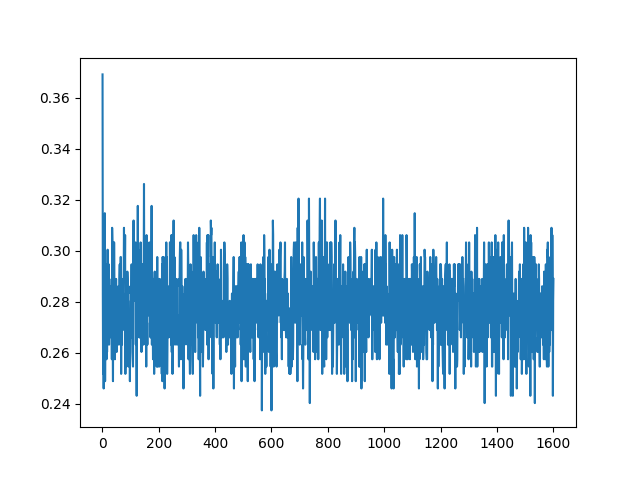
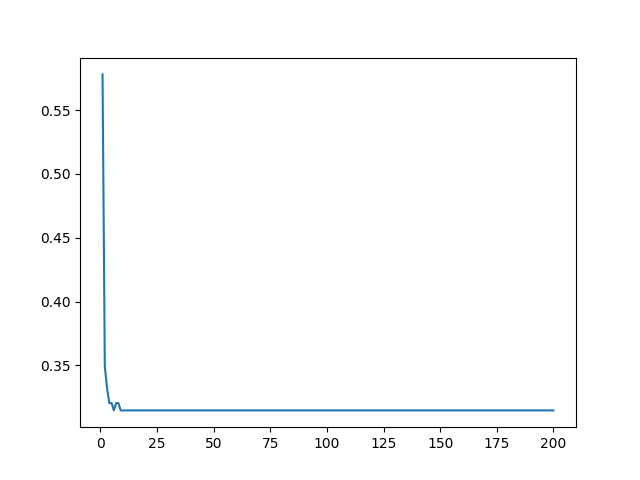
Ej 4.2.1

La implementación de ambas redes ya está explicada en la sección anterior donde se respeta todos los pasos vistos de la clase de teoría, pero añadiendo un nuevo criterio de parada llamado *época*, por otra parte, hubo un cambio de nombre en el método conectar puesto que Python no puede saber qué tipo de atributo le está pasando.

Se realizo un hiperparameter tuning simulando con diferentes valores para los hiperparametros obteniendo la configuración más óptima para la resolución del problema en Adaline con 40 épocas, umbral 0.2, coeficiente Alpha 0.01 y tolerancia 0.01 con un porcentaje de acierto del 94.82%.

Para el Perceptrón se obtuvo como configuración más óptima umbral 0.2, con 20 épocas y coeficiente Alpha 0.73 con un porcentaje de acierto total del 95.89%.

Para el error cuadrático medio obtenido durante el entrenamiento se observa que para el perceptrón varía bastante a lo largo de las épocas en el entorno de entre el 0.32 y 0.24, mientras que en el caso de adaline se mantiene bajo y constante sin apenas cambios. Estos valores se aprecian en las siguientes gráficas:



ECM de umbral Adaline ECM de umbral Perceptron

Respecto a la variación de cada uno de los hiperparámetros en el porcentaje de acierto del algoritmo se observa por un lado que para valores pequeños de umbral cercanos 0.5 se tiene un mayor acierto para Perceptron y Adaline mientras que a valores mayores se mantiene constante en un porcentaje de acierto bajo

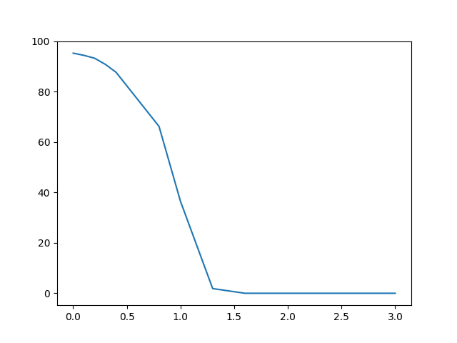


Ilustración Acierto en función de umbral Adaline

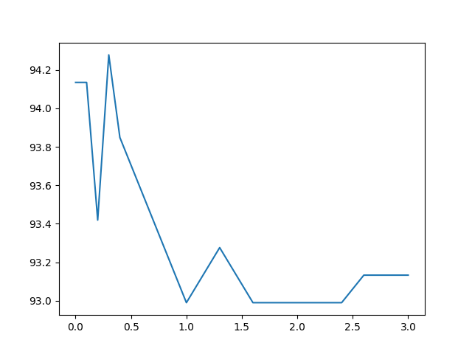


Ilustración Acierto en funcion de umbral, Perceptron

En cuanto al porcentaje de acierto en función del parámetro Alpha, con valores del entorno de entre el 0 y 0.4 se tiene un mayor acierto en ambos casos, mientras que para valores mayores el perceptrón decrece gradualmente y Adeline se mantiene constante en valores mínimo.

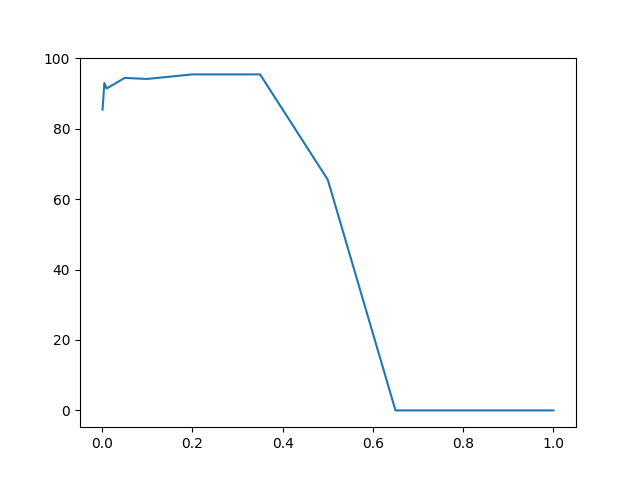


Ilustración Acierto de funcion de alpha, Adeline

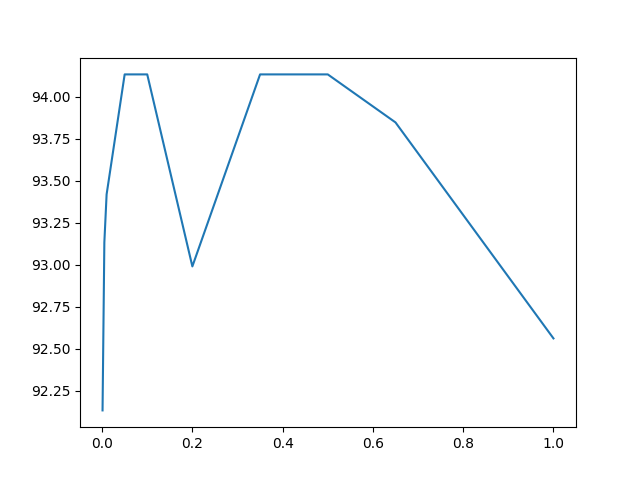


Ilustración Acierto de funcion de alpha, Perceptrón

En cuanto al porcentaje de acierto en función del número de épocas Adeline es ligeramente dispar a medida que se van sucediendo las épocas mientras que el perceptrón alcanza valores máximo en sobre las 100 épocas y más adelante va decreciendo su acierto gradualmente.

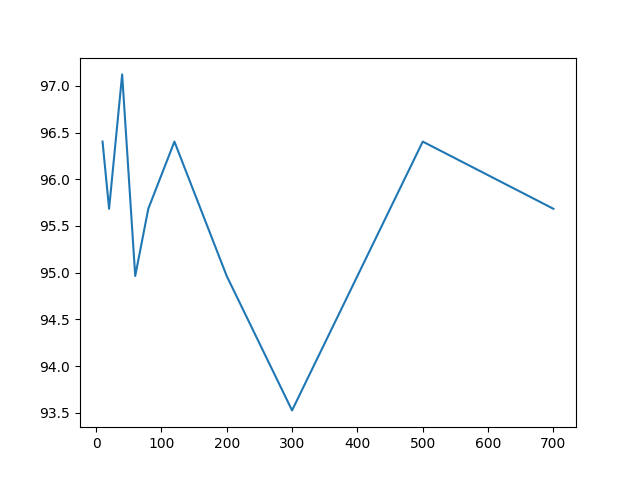


Ilustración Porcentaje acierto en funcion epocas, adeline

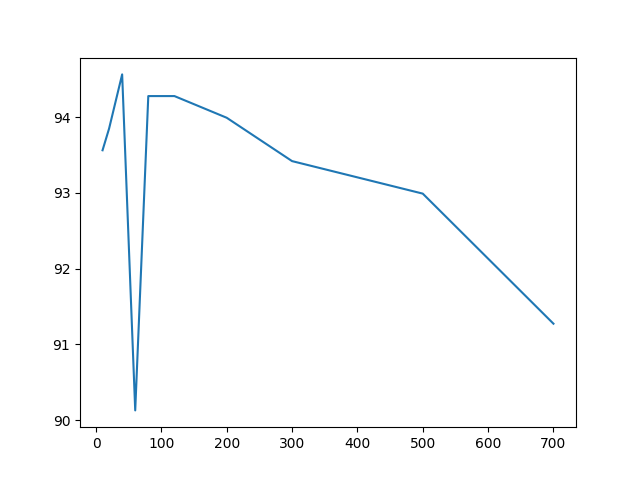


Ilustración Porcentaje acierto en funcion epocas, perceptrón

Como dato extra, los porcentajes proporcionados en variación de épocas, por ejemplo, normalmente debería reducir el error y aumentar el acierto, pero como estamos ejecutando en el modo1, el porcentaje de acierto valía en función de los datos del train y de test.

Ej 4.3.1

La implementación del perceptrón y del adeline está explicada en la sección 2.1, por otra parte para realizar el hiperparameter tuning, se realizaron una serie de simulaciones para comprobar el grado de acierto en función de estos parámetros. Para el perceptrón la mejor composición hallada fue de 160 epocas, umbral de valor 0.2 y parámetro Alpha con valor 0.792, con un porcentaje de acierto del 72, 7%. Para el adaline la mejor composición hallada fue de 20 epocas, umbral de valor 0.2 y parámetro Alpha de valor 0.08 con un porcentaje de acierto del 67,7%.