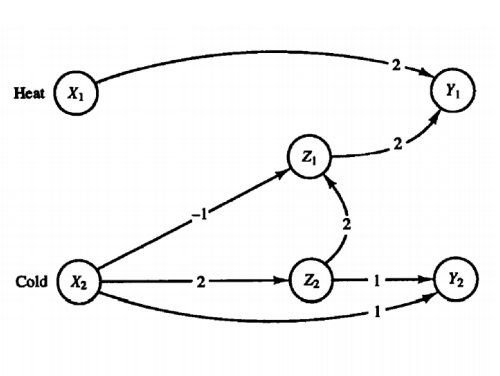
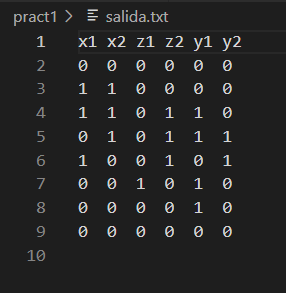
Ej 2.1

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.

Cabe destacar que, en la lógica de las capas, una vez de se disparan las neuronas, directamente llama a inicializar ya los pesos de las conexiones han sido actualizadas para propagar, y se ahorra algunas líneas de códigos a la hora de construir otras redes. 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 disparar y propagar de la clase RedNeuronal que itera todas las capas, para redes como puede ser Perceptron 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 0,30 puesto que 0,1 <= 3 neuronas \* 0,30 <= 1, el umbral es igual a 0.2 y la tolerancia igual a 0.01.

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*.

SANTOS：para calcular la ecuación mira la teoría, que básicamente es poner la ecuación de frontera < umbral o > umbral, luego despejas y ya esta. El umbral del momento ponemos a 0.2 y ya esta.

AND frontera decisión:

**Perceptron** -> 2\*w1 + 3\*w2 - 4 = 0

**Adaline** -> 0.9\*w1 + 1.2\*w2 - -1.8 = 0

AND2 frontera decisión:

Adaline -> w1=-1.0, w2=-1.0, b=1

Perceptron -> w1=-1.0, w2=-1.0, b=1

OR frontera decisión:

Adaline ->

Perceptron -> w1=0, w2=0.0, b=1

NAND frontera decisión:

Adaline -> w1=1.0, w2=1.0, b=1

Perceptron -> w1=1.0, w2=1.0, b=1

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.