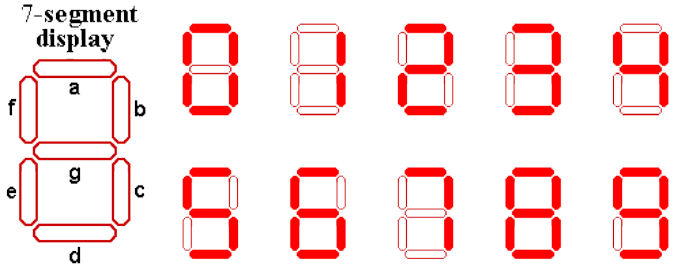
Práctica 1 - Introducción a los Modelos de Computación Conexionistas

INTRODUCCIÓN

Profesor responsable: Patricio García Báez
Fecha tope de corrección: 9 de diciembre
Objetivo: Dominio del SNNS y capacidad representación del conocimiento en las RN monocapas



Utilizando el neurosimulador SNNS se pretende diseñar y entrenar varias redes neuronales monocapa capaces de aprender a **transformar valores** numéricos entre **diferentes tipos de codificaciones** y la codificación utilizada para los **visualizadores de siete segmentos**, tan comunes en dispositivos electrónicos. Para ello habrán de diseñarse los ficheros correspondientes de patrones de entrenamiento así como los que almacenen las diferentes redes en sí.

Utilizaremos los **números del 0 al 9**, los cuales los codificaremos de cuatro diferentes maneras:

1. Utilizando **una entrada o neurona de salida** asociada **a cada uno de estos números** (un total de **10 entradas o salidas**), en dicha codificación ha de estar activo un único elemento, que indica el número a representar.
2. En **modo binario**, esto es, mediante **cuatro elementos**, representando respectivamente las diferentes potencias de dos de la codificación binaria del número a representar. El elemento que este activo indica un uno en la codificación binaria y el inactivo un cero.
3. En **modo decimal**, con **un solo elemento**, de forma que la magnitud de éste indique el valor que se quiere representar, una neurona por tanto.
4. En el formato utilizado por los **visualizadores de siete segmentos** (http://www.physics.udel.edu/wwwusers/watson/scen103/7seg.html), representado por **siete elementos** correspondientes a los valores (on/off = 1/0) de los segmentos a, b, c, d, e, f, g (ver figura).

Se pretende por tanto generar y entrenar **seis tipos de codificadores diferentes**:

1. Conversor de código 1 al 4.
2. Conversor de código 4 al 1.
3. Conversor de código 2 al 4.
4. Conversor de código 4 al 2.
5. Conversor de código 3 al 4.
6. Conversor de código 4 al 3.

Se hará uso de los modelos de redes monocapas **Perceptrón simple**, **Adaline** y **Backpropagation**. El SNNS no implementa de modo exclusivo los modelos de Perceptrón y Adaline, pero **haciendo uso** de las funciones de activación **Act_StepFunc** y **Act_Identity** junto a la función de aprendizaje **Std_Backpropagation** dichos modelos pueden ser simulados casi con entera exactitud[1]. Además, si es necesario, una vez finalizado el aprendizaje del modelo Adaline es posible acoplarle la función de salida **Out_Threshold05**.

StepFunc	$a_j(t) = \begin{cases} 1 & \text{if } net_j(t) > 0 \\ 0 & \text{if } net_j(t) \leq 0 \end{cases}$
Identity	$a_j(t) = net_j(t)$
Threshold_0.5	$o_j(t) = \begin{cases} 0 & \text{if } a_j(t) \leq 0.5 \\ 1 & \text{if } a_j(t) > 0.5 \end{cases}$

Ha de estudiarse los resultados de aplicar ambos modelos ante los patrones anteriores, tomando nota de si **convergen o no**, la **influencia de los ratios** de aprendizaje en el **tiempo de convergencia** y los **errores obtenidos**.

Con los resultados de los entrenamientos anteriores se confeccionará una **tabla de resultados** en la que se indique **tipo de patrón**, variante de **RN empleada**, valor del **ratio de aprendizaje óptimo**, **número de ciclos** empleado para el aprendizaje y **mejor error** obtenido, tanto su valor **SSE** como en **porcentaje de aciertos** así como las **observaciones** que se quieran hacer constar. Dicha tabla será la que finalmente se entregue como justificación de la práctica.

1 Para ello es necesario pasar del modelo de neurona tipo 1 (con umbral), que implementa el SNNS, al modelo 3 (con *bias*), para ello habría que añadir a cada patrón una entrada adicional con valor de activación a 1. El peso de la conexión de esa entrada con las neuronas hará las veces de *bias*, que el algoritmo Std_Backpropagation se encargará de actualizar.