

### Ejercicio 1:

Diseñe una neurona perceptrón con dos entradas binarias y una función de activación por umbral que calcule la función lógica de la coimplicación, es decir, que calcule  $x_1 \leftrightarrow x_2$ . Determine un conjunto de pesos adecuados para que la neurona calcule esta función (No es necesario aplicar ningún algoritmo concreto.)

### **SOLUCIÓN:**

La coimplicación en la lógica corresponde a la función representada en la tabla.

$x_1$	$x_2$	$x_1 \leftrightarrow x_2$
0	0	1
0	1	0
1	0	0
1	1	1

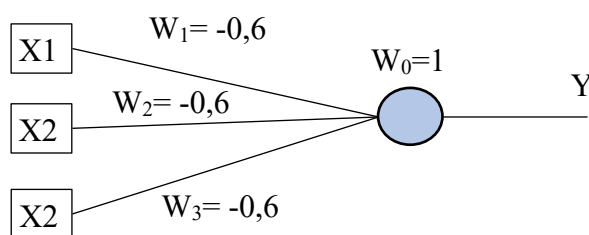
Este ejercicio no tiene solución. No es posible encontrar un conjunto de pesos de una única neurona de tal forma que se calcule esta función. El ejemplo es similar a la función XOR. Una neurona perceptrón representa un separador o clasificador lineal y no es posible separar los casos de activación y no-activación de esta función de forma lineal.

### Ejercicio 2:

Diseñe una neurona perceptrón con tres entradas binarias y una función de activación por umbral cuya salida sea activa, si y sólo si dos o más de las entradas tienen un valor 0. Determine un conjunto de pesos adecuados para que la neurona calcule esta función (No es necesario aplicar ningún algoritmo concreto.)

### **SOLUCIÓN:**

En este caso sí es posible diseñar una neurona para calcular la función deseada. Una posible solución es la que se presenta a continuación:



### Ejercicio 3:

Un banco quiere clasificar los clientes potenciales en fiables o no fiables. El banco tiene un dataset de clientes antiguos, con los siguientes atributos:

- Estado civil: {casado/a, soltero/a, divorciado/a}
- Género: {varón, mujer}
- Edad: {[18-30], [31-50], [51-65], [65+]}
- Ingresos: {[10K-25K], [26K-50K], [51K-65K], [66K-100K], [100K+]}

- Diseñe una red neuronal con neuronas perceptrón simples de activación umbral y de una única capa que se podría entrenar para predecir si un cliente es fiable o no.
- Inicializa los pesos de la red, inventa un caso de entrenamiento y reproduce cómo el algoritmo de aprendizaje realizaría modificaciones en la red con este ejemplo.

### SOLUCIÓN:

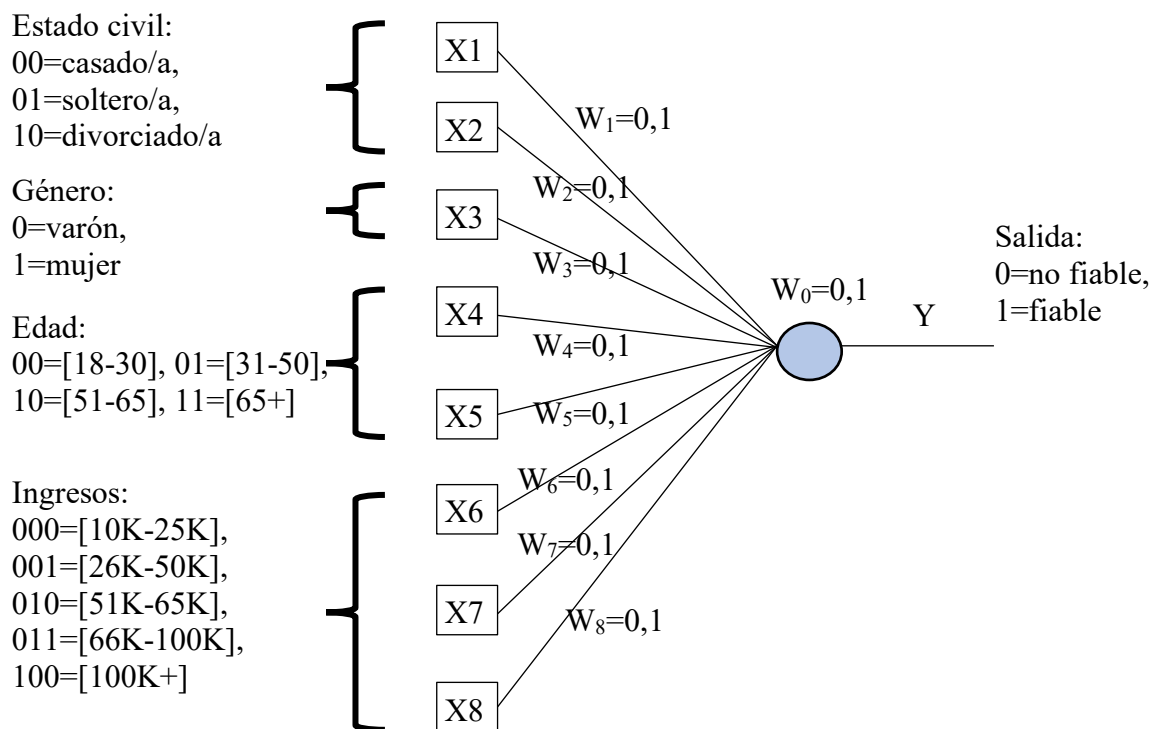
a) Para diseñar la red debemos determinar en primer lugar las posibles entradas y las salidas necesarias. Una red neuronal trabaja con entradas y salidas sub-simbólicas, es decir cada neurona tendrá como entrada o salida un bit, 0 o 1. Por tanto, es necesario determinar un mapeo (codificación /decodificación) de las entradas y salidas a vectores de bits.

Como tenemos que clasificar clientes en fiables o no, nuestra red tendrá una única neurona de salida, cuyo resultado será identificado de la siguiente manera: 0=no fiable, 1=fiable.

Por lo que se refiere a las entradas, tenemos

- Estado civil: 3 valores, que se pueden expresar con dos entradas (de un bit cada una): 00=casado/a, 01=soltero/a, 10=divorciado/a
- Género: 2 valores, que se pueden expresar con una entrada: 0=varón, 1=mujer
- Edad: 4 valores, que se pueden expresar con dos entradas 00=[18-30], 01=[31-50], 10=[51-65], 11=[65+]
- Ingresos: 5 valores, que se pueden expresar con tres entradas, 000=[10K-25K], 001=[26K-50K], 010=[51K-65K], 011=[66K-100K], 100=[100K+]

La red resultante sería la siguiente:



b) Todos los pesos han sido inicializados a 0,1 (véase la imagen). Ahora consideremos el siguiente caso de entrenamiento: <(casado, varón, 31 años, 27k de ingresos), no fiable>

Primero hay que decodificar el ejemplo. Se obtiene: <00001001, 0>

Ahora se pasa la entrada de este ejemplo a la red y se calcula la salida. En este caso, la salida será:  
 $0*0,1+0*0,1+0*0,1+0*0,1+1*0,1+0*0,1+0*0,1+1*0,1+0,1=0,3 >0 \rightarrow \text{salida } 1$

Se calcula el error:  $\text{error}=0-1=-1$

Se ajustan los pesos. Para ello asumimos una constante  $\alpha=0,1$ , por ejemplo.

Se ajustan los siguientes pesos:

$W_0=0$ ;  $W_8=0$ ;  $W_5=0$ ; el resto de los pesos no cambiarían porque sus entradas son 0.

#### **Ejercicio 4:**

Una empresa de videojuegos está desarrollando un FPSG (first person shooting game). Para implementar los personajes artificiales del juego, el jefe de proyecto, ex estudiante del curso de IA en la URJC, ha pensado que podría ser interesante e innovador utilizar una red neuronal. Dicha red tendría que implementar el algoritmo de control de los personajes artificiales, usando los siguientes inputs:

- Salud: de 0 (débil) hasta 2 (fuerte)
- Tiene cuchillo: si o no
- Tiene arma: si o no
- Enemigos: número de enemigos en el campo visual

Las acciones que el personaje puede ejecutar son:

- Escapar
- Andar
- Atacar
- Esconderse

a) Diseña una red neuronal (de una única capa) que se podría entrenar para implementar el algoritmo de control de los personajes artificiales.

b) Considera el siguiente vector de input:

$x=[\text{Salud}=2, \text{Tiene cuchillo}=\text{no}, \text{Tiene arma}=\text{si}, \text{Enemigos}=2]$

y la salida deseada:

$y=\text{Atacar}$ .

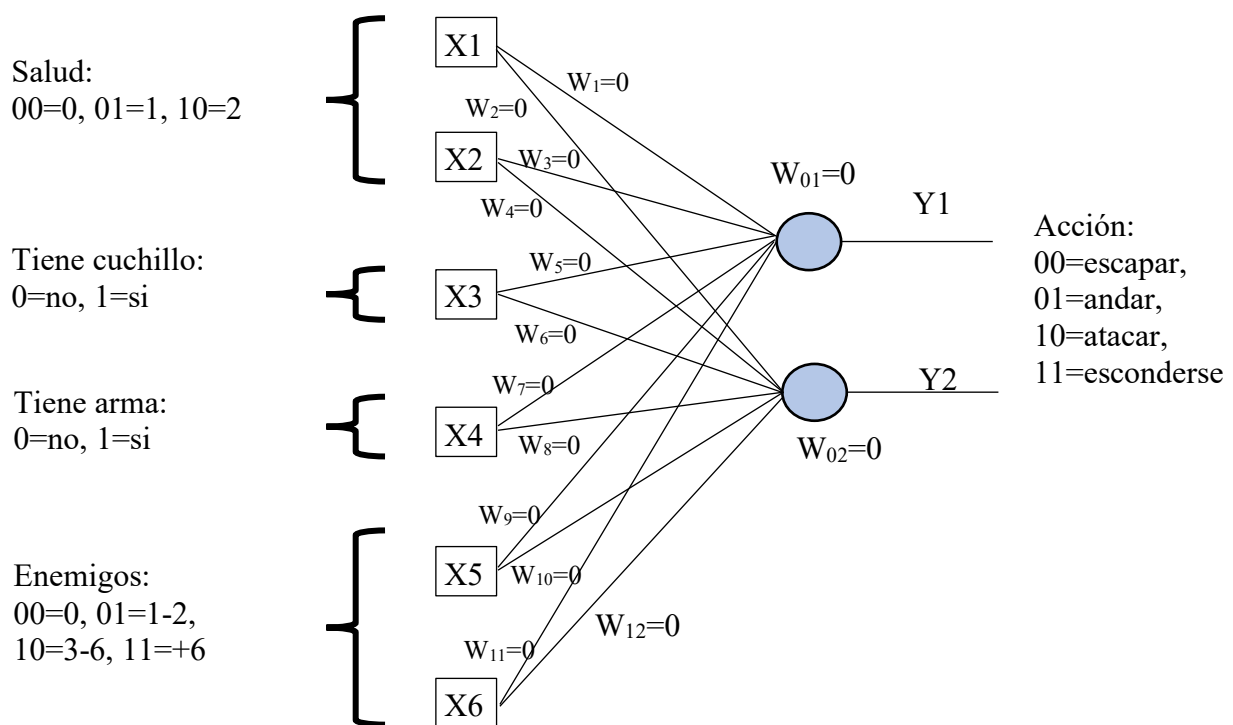
Inicializa los pesos y los sesgos de las neuronas de tu red de manera aleatoria en el rango  $[-0.5, 0.5]$ . Usando el elemento de entrenamiento definido arriba indica como cambiarían los pesos. Por simplicidad, asume que las neuronas tienen una función de activación por umbral y que se emplea  $g'(x)=1$  como sustituto de la derivada de la función de activación.

#### **SOLUCIÓN:**

a) Las entradas son: 2 bits para salud (00=débil(0), 01=medio(1), 10=fuerte(2)), 1 bit para Tiene cuchillo (0=no, 1=si), 1 bit para Tiene arma (0=no, 1=si). Respecto a los enemigos, hay que analizar el problema en más detalle (cuantos enemigos puede haber) y establecer intervalos razonables. En este caso vamos a elegir que debe haber un valor para enemigos=0, otro valor para pocos enemigos (1-2), un valor para un número mediano (3-6) y un valor para muchos enemigos (más de 6). Por tanto, usamos 2 bits para Enemigos: 00=0, 01=1-2, 10=3-6 y 11=más que 6.

Respecto a las clases, tenemos que representar 4 acciones. Por lo tanto, necesitamos 2 neuronas de salida, y codificaremos las donde 00=escapar, 01=andar, 10=atacar, y 11=esconderse.

Una posible red podría ser la que se presenta en la figura siguiente figura:



b) Inicializamos los pesos de la red y de los sesgos de las neuronas todos al valor 0 (otros valores serían posibles).

Analizamos el caso:

La entrada corresponde al vector  $x=100101$ , y la salida esperada al vector  $y_{esperada}=10$ . La salida actual con esta entrada y la red actual sería  $y_{actual}=00$ .

Por tanto, el error para la primera neurona es  $e = y_{esperada} - y_{actual} = 1 - 0 = 1$  y en el caso de la segunda neurona:  $e = y_{esperada} - y_{actual} = 0 - 0 = 0$ .

Con estos valores, los pesos de las entradas de la segunda neurona, así como su sesgo ( $W_{02}$ ) no cambian.

Respecto a la primera neurona los pesos de entrada se calculan según la fórmula:

$$W_i = W_i + \alpha * X_i * \text{error}$$

Con ello, los pesos que cambian son:

$$W_1 = 0 + 0,2 * 1 * 1 = 0,2 \quad W_7 = 0 + 0,2 * 1 * 1 = 0,2 \quad W_{11} = 0 + 0,2 * 1 * 1 = 0,2$$

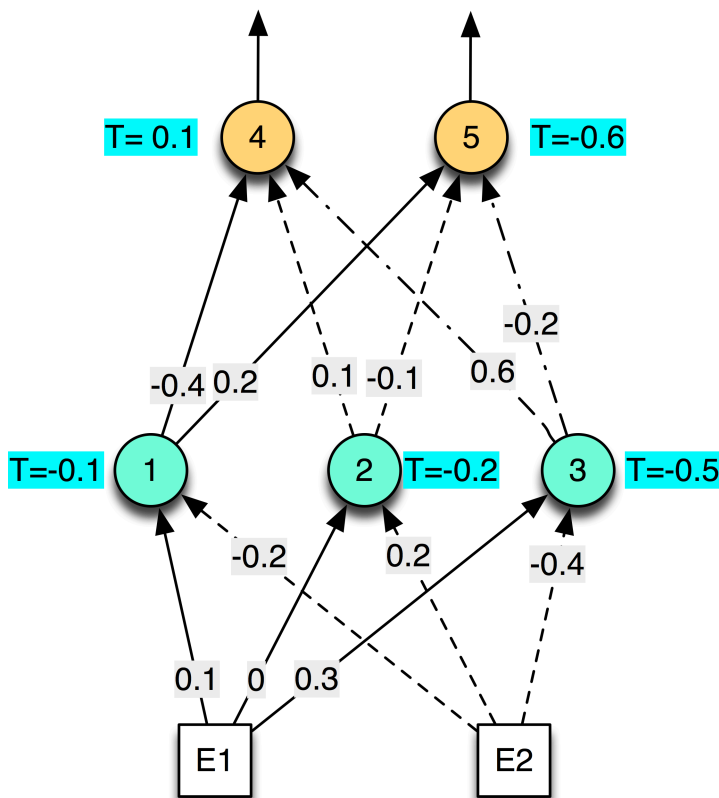
El resto de los pesos de conexión no cambian porque sus entradas son 0.

Respecto al sesgo de la neurona 1, este sesgo cambia de la siguiente forma:

$$W_{01} = W_{02} + \alpha * \text{error} = 0 + 0,2 * 1 = 0,2$$

### **Ejercicio 5:**

Sea la red neuronal presentada en la figura. Supón que la función de activación de las neuronas 1,2,3,4 y 5 sea la función umbral que devuelve 1 si la suma pesada de las entradas es mayor que 0. Además, en la retropropagación de los errores emplea  $g'(x)=1$  como sustituto de la derivada de la función de activación. T denota el sesgo de cada neurona.



Las entradas de una red neuronal no necesariamente tienen que tener valores binarios, también podrían ser de valores continuos en el intervalo  $[0..1]$ . Para neuronas con una función de activación por umbral, la salida, sin embargo, si será binaria.

Dado el siguiente elemento del conjunto de entrenamiento:

$$(x,y) = (x=[E1=0.6, E2=0.1], y=[4=0, 5=1])$$

donde el valor la entrada 1 es 0.6, el valor de la entrada 2 es 0.1, la salida de la neurona 4 es 0 y la salida de la neurona 5 es 1.

Sea 0.1 la constante de aprendizaje, aplica el algoritmo de retropropagación para actualizar los pesos de la red con este elemento del conjunto de entrenamiento e indicando como cambian los pesos.

### SOLUCIÓN:

Calculamos la salida de las neuronas de la capa oculta

- $1 \rightarrow 0.1 * 0.6 - 0.2 * 0.1 - 0.1 = -0.06 < 0 \rightarrow 0$
- $2 \rightarrow 0 * 0.6 + 0.2 * 0.1 - 0.2 = -0.18 < 0 \rightarrow 0$
- $3 \rightarrow 0.3 * 0.6 - 0.4 * 0.1 - 0.5 = -0.36 < 0 \rightarrow 0$

Calculamos la salida de las neuronas de salida

- $4 \rightarrow -0.4 * 0 + 0.1 * 0 + 0.6 * 0 + 0.1 = 0.1 > 0 \rightarrow 1$
- $5 \rightarrow 0.2 * 0 - 0.1 * 0 - 0.2 * 0 - 0.6 = -0.6 < 0 \rightarrow 0$

Calculamos el error de la capa de salida

- $e_4 \rightarrow 0 - 1 = -1$
- $e_5 \rightarrow 1 - 0 = 1$

Calculamos los errores de la capa oculta

- $e_1 \rightarrow -0.4 * e_4 + 0.2 * e_5 = 0.6$
- $e_2 \rightarrow 0.1 * e_4 - 0.1 * e_5 = -0.2$
- $e_3 \rightarrow 0.6 * e_4 - 0.2 * e_5 = -0.8$

Calculamos los nuevos pesos de la capa de salida:

Respecto a la capa de salida, los pesos  $w_{i,4/5}$  se actualizan con la regla  $w_{i,4/5} = w_{i,4/5} + x_i * 0.1 * e_{4/5}$  pero como todos los  $x_i$  son 0, los pesos de la capa de salida no se modifican.

Los sesgos de las neuronas si cambian por  $T_{4/5} = T_{4/5} + 0.1 * e_{4/5}$ . Con ello, se obtiene:  $T_4 = 0$  y  $T_5 = -0.5$

Respecto a la capa oculta podemos observar que el error  $e_1$  es positivo, mientras que  $e_2$  y  $e_3$  son negativos. Entonces los pesos  $w_{E1/E2,1}$  deberían aumentar, y los pesos  $w_{E1/E2,2}$  y  $w_{E1/E2,3}$  deberían disminuir.

Calculamos los nuevos pesos de la capa oculta:

$$\begin{aligned} W_{E1,1} &= W_{E1,1} + x_1 * \alpha * e_1 = 0.1 + 0.6 * 0.1 * 0.6 = 0.136 \\ W_{E2,1} &= W_{E2,1} + x_2 * \alpha * e_1 = -0.2 + 0.1 * 0.1 * 0.6 = -0.194 \\ W_{E1,2} &= W_{E1,2} + x_1 * \alpha * e_2 = 0 + 0.6 * 0.1 * -0.2 = -0.012 \\ W_{E2,2} &= W_{E2,2} + x_2 * \alpha * e_2 = 0.2 + 0.1 * 0.1 * -0.2 = 0.198 \\ W_{E1,3} &= W_{E1,3} + x_1 * \alpha * e_3 = 0.3 + 0.6 * 0.1 * -0.8 = 0.252 \\ W_{E2,3} &= W_{E2,3} + x_2 * \alpha * e_3 = -0.4 + 0.1 * 0.1 * -0.8 = -0.408 \end{aligned}$$

Los pesos correspondientes a los sesgos de las neuronas de esta capa se actualizan como sigue:

$$T_1 = T_1 + 0,1 * e_1 = -0,1 + 0,1 * 0,6 = -0,04$$

$$T_2 = T_2 + 0,1 * e_2 = -0,2 + 0,1 * -0,2 = -0,22$$

$$T_3 = T_3 + 0,1 * e_3 = -0,5 + 0,1 * -0,8 = -0,58$$