Sesión 5
Procesamiento paralelo
Docente: Mg. Huarote Zegarra Raúl

Objetivos:

- Conoce el proceso de aprendizaje de la red neuronal back propagation, para el proceso de aprendizaje de la compuerta XOR.

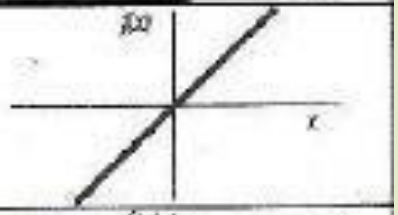
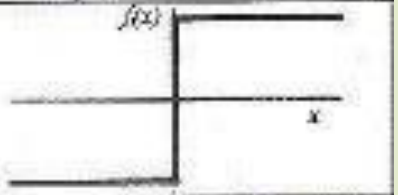
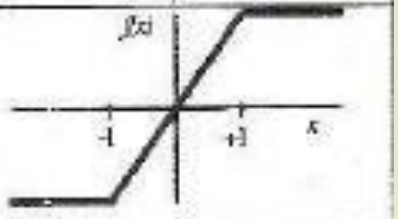
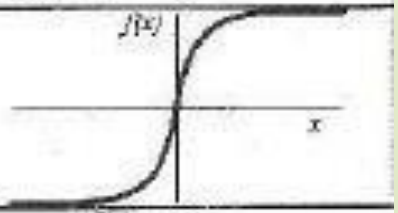
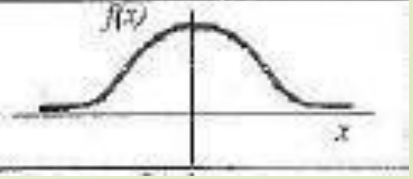
TEMAS A TRATAR:

- Proceso de aprendizaje de la red neuronal back propagation.
- Aprendizaje de la compuerta lógica XOR en Python.

Motivación

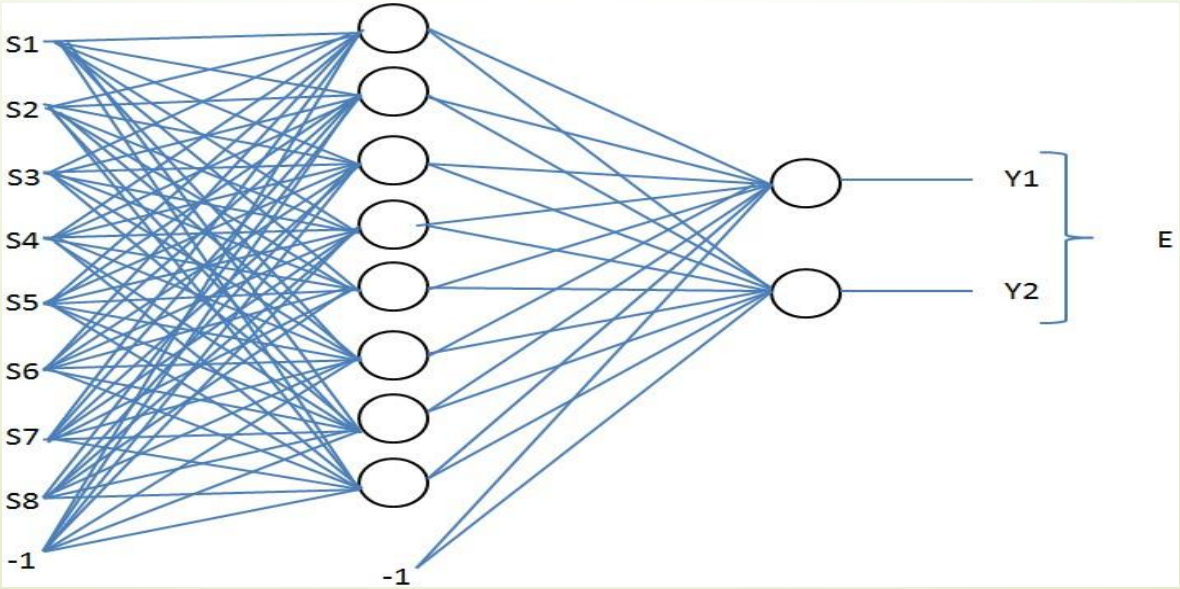
https://www.youtube.com/watch?v=f-GYR-GfJ7o

Dentro del proceso de aprendizaje nos damos cuenta que requiere una función de adaptación tales como:

Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$ 
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$ 
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	$[-1, +1]$ 
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$ 
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$ 

Estructura de una redeneuronales

- Debemos de tener en cuanta sus capas:
  - Para los modelos de hoy por la complejidad de problemas que se presenta, requieren necesariamente ser multicapa.
  - Para una capa es el Perceptron simple.



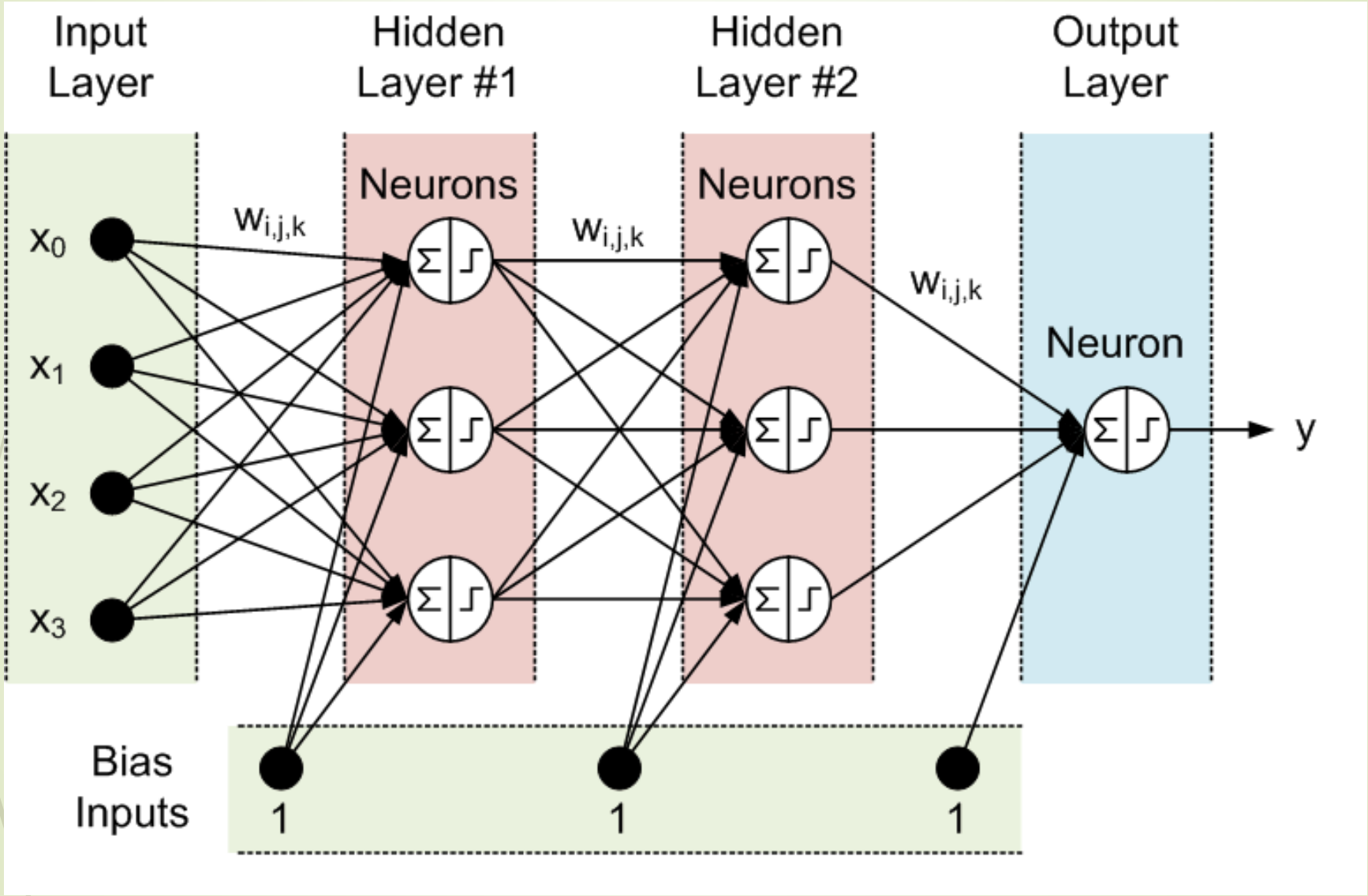
Tipos de redes neuronales artificiales

1.- Adaline y Madaline
2.- ART
3.- Back-Propagation
4.- BAM
5.- The Boltzman Machine
6.- Brain-State-in a Box
7.- Cascade-Correlation-Networks
8.- Counter-Propagation
9.- DBD
10.- DNNA
11.- DRS
12.- FLN
13.- Hamming
14.- Hopfield
15.- LVQ
16.- Perceptron
17.- PNN
18.- Recirculation
19.- SOM
20.- SPR

BackPropagation (Propagación hacia atrás).

El algoritmo de backpropagation es un algoritmo para entrenar de izquierda a derecha y supervisada, con un número indeterminado de capas. El objeto de la red será hacer que los pesos configuren la salida de la red para que ésta sea lo más cercana posible a la salida deseada.

BackPropagation (Propagación hacia atrás).



Para el proceso de aprendizaje de la red neuronal back propagation, se debe tener en cuenta lo siguiente:

- Patrón de entrada:

$$X_p = (x_{p0}, x_{p1}, x_{p2}, x_{p3}, x_{p4}, \ldots, x_{pn})$$

p es la p-ésimo registro a aprender.

N = n +1 es tamaño de patrones de entrada de la red.

Ejemplo:
XOR

n = 1

N = n + 1 = 2

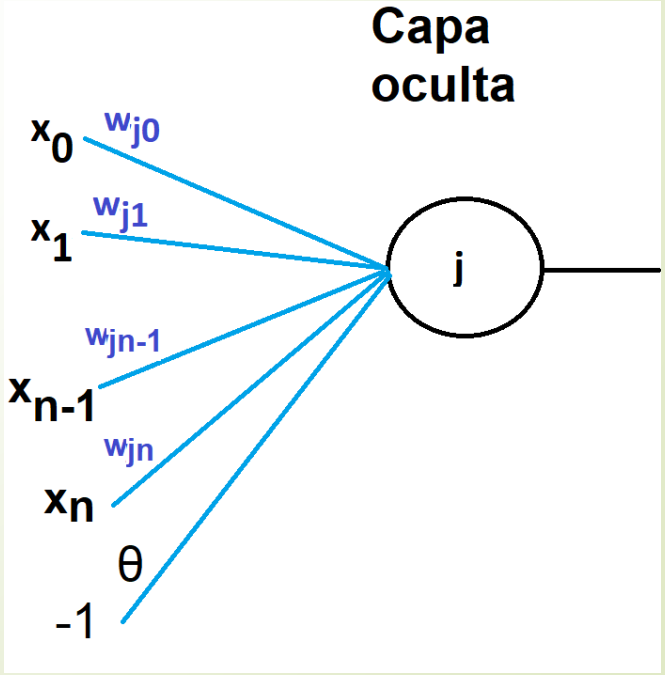
	X0	X1	T
p=0	1	1	-1
p=1	1	-1	1
p=2	-1	1	1
p=3	-1	-1	-1

p-ésimo registro

Entrada neta de la j-ésima neurona de la capa oculta (hidden).

$$neta_{pj}^{h} = \sum_{j=0}^{n} w_{ji}^{h}\, x_{pi} + \theta_{j}^{h}$$

representa la sumatoria de los patrones de entrada multiplicado por sus respectivos pesos mas el umbral (bia).



- Salida de la j-esima neurona de la capa oculta:

$$I_{pj} = f_j^h(neta_{pj}^h)$$

Donde I representa la salida de la capa oculta donde a su vez va a ser entrada para la capa de salida con su respectivo peso.

La función que muestra es la sigmoidea.

- La entrada neta de la k-esima neurona de la capa de salida:

$$neta_{pk}^{o} = \sum_{j=0}^{m} w_{kj}^{o} I_{pj} + \theta_{k}^{o}$$

Donde M representa la cantidad de neuronas de la capa oculta:

$$M = m + 1$$

La sumatoria de la salida de cada neurona de la capa oculta multiplicado por su respectivo peso, mas el umbral.

- Salida de la k-esima neurona de la capa de salida:

$$o_{pk} = f_k^o(neta_{pk}^o)$$

Donde O_{pk} la salida de cada una de las capas de salida. La función f es la sigmoidea.

- Error en una de las neuronas de la capa de salida:

$$\delta_{pk} = (T_{pk} - o_{pk})$$

Donde T_{pk} es la salida deseada, O_{pk} es la salida real (u obtenida) , p se refiere al p-ésimo registro, k se refiere a la k-ésima neurona de salida.

- El objetivo para el proceso de aprendizaje es minimizar el Error medio cuadrático (Emc) en cada registro *p*:

$$Emc_p = \frac{1}{2}\sum_{k=0}^{s} \delta_{pk}^2$$

$$Emc_p = \frac{1}{2}\sum_{k=0}^{s} (T_{pk} - o_{pk})^2$$

Donde S *es la cantidad de neuronas de la capa de salida:*

S = *s* + *1*

- Por lo tanto en cada neurona de la capa de salida tiene error:

$$\delta_{pk}^{o} = (T_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^{o})$$

- La actualización de los pesos de la capa oculta a la capa de salida va de acuerdo a los resultados obtenidos en el proceso.

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta \delta_{pk}^{o} I_{pj}$$

Donde:

w(t+1): Es el nuevo peso.

w(t): Es el peso actual.

η: Es el factor de aprendizaje(valor entre 0 y 1)

- La actualización de los pesos de la capa de entrada a la capa oculta.

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta \delta_{pj}^{h} x_i$$

Donde:

w(t+1): Es el nuevo peso.

w(t): Es el peso actual.

η : Es el factor de aprendizaje(valor entre 0 y 1)

x_i: El patrón de entrada a quien corresponde.

- Funciones de activación.

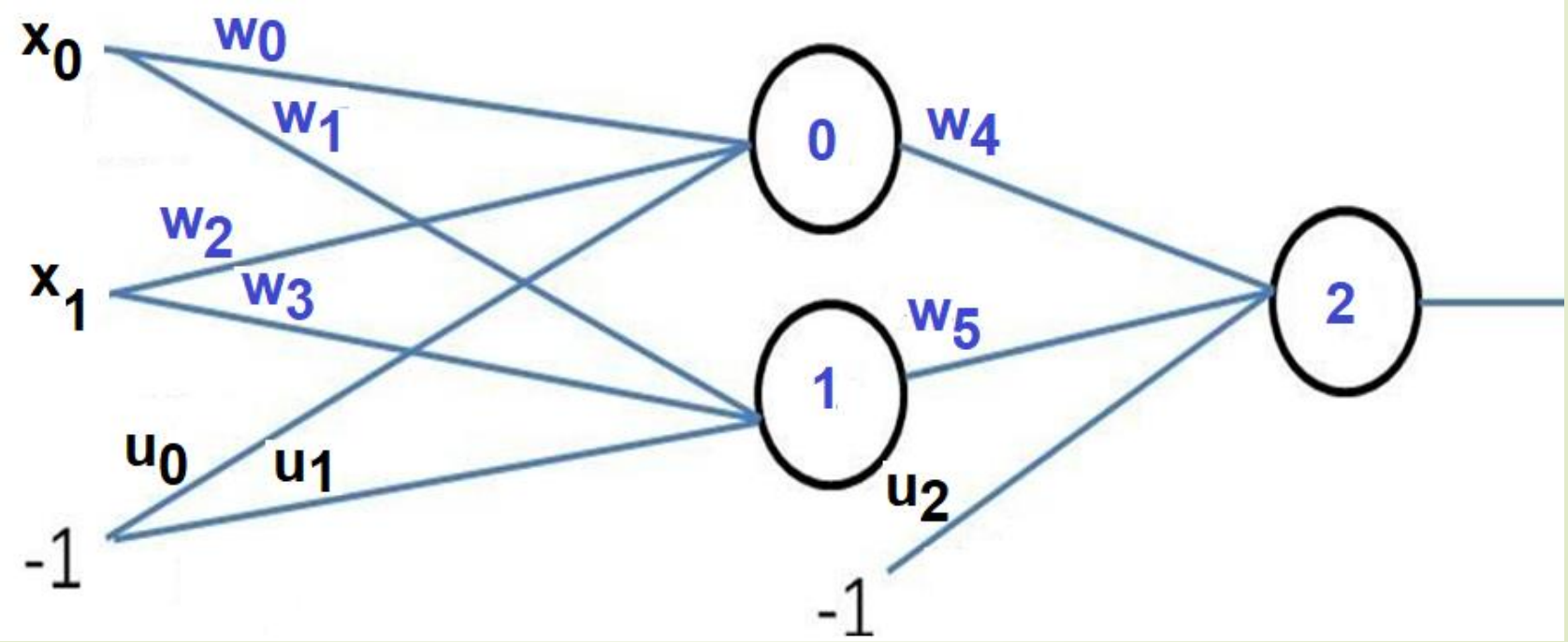
$$f_k^o(neta_{jk}^o) = (1 + e^{-neta_{jk}^o})^{-1}$$

Función f prima.

$$f_k^{o\,\prime} = f_k^o(1 - f_k^o)$$

Ejemplo del Backpropagation para la compuertalógica XOR

- Consideremos la representación del preceptron multicapa para nuestro caso.



Ejemplo del Backpropagation para la compuertalógica XOR

- Los valores para el entrenamiento son:

X0	X1	T
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

- Los valores de los pesos y los umbrales son valores aleatorios entre 0 y 1 y son:

w0 = 0.5 w1 = 0.3 w2 = 0.7 w3 = 0.1

w4 = 0.4 w5 = 0.9 u0 = 0.8 u1 = 0.2

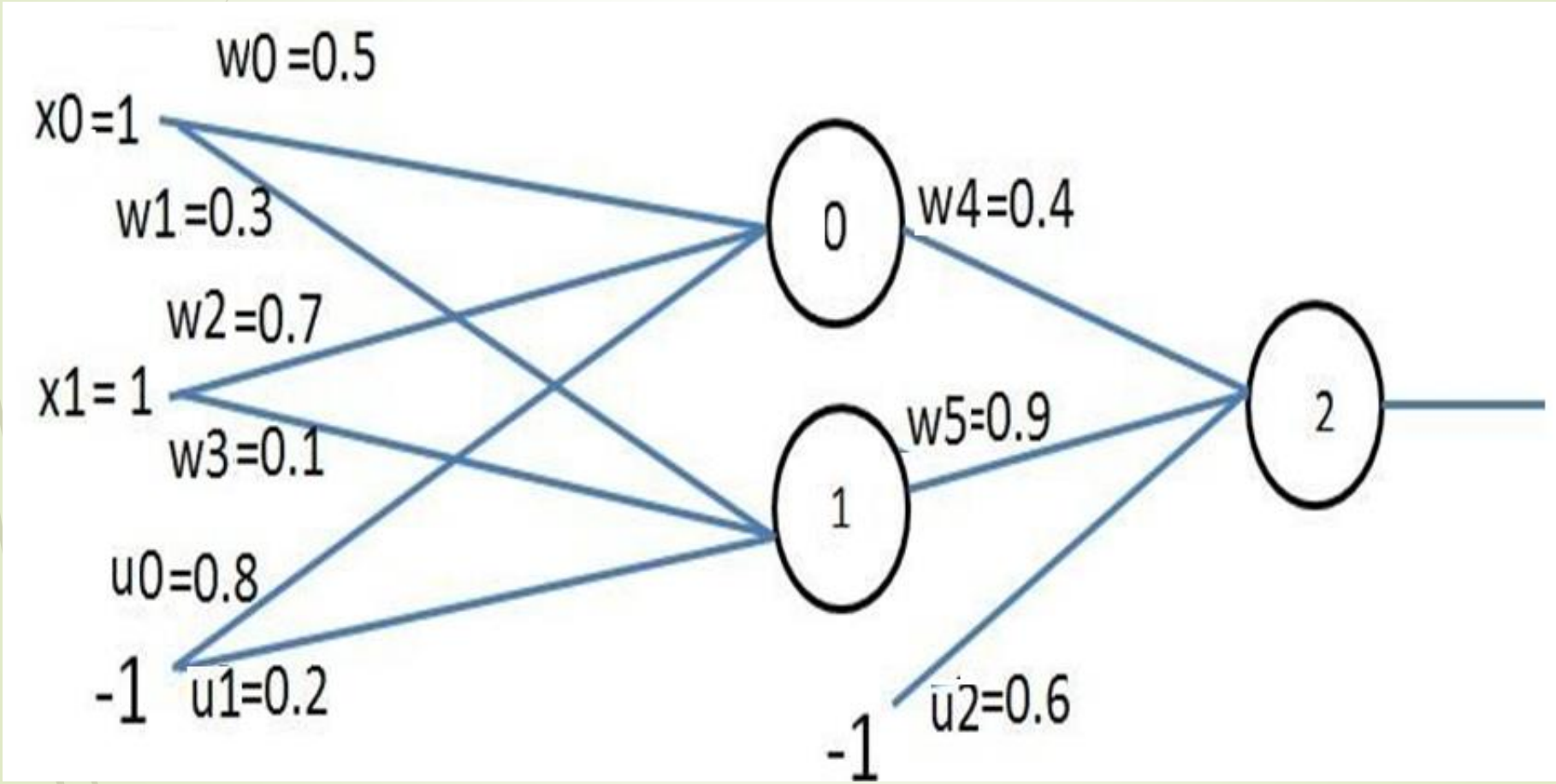
u2 = 0.6

Con factor de aprendizaje = 0,65

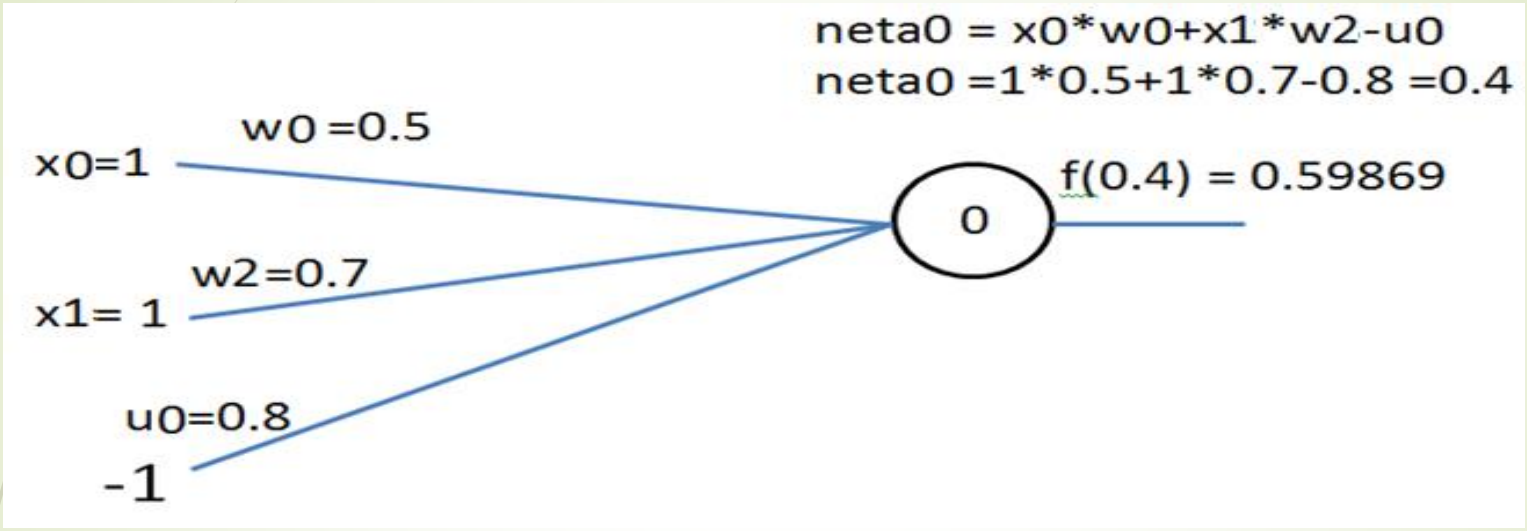
Máximo iteraciones = 1000

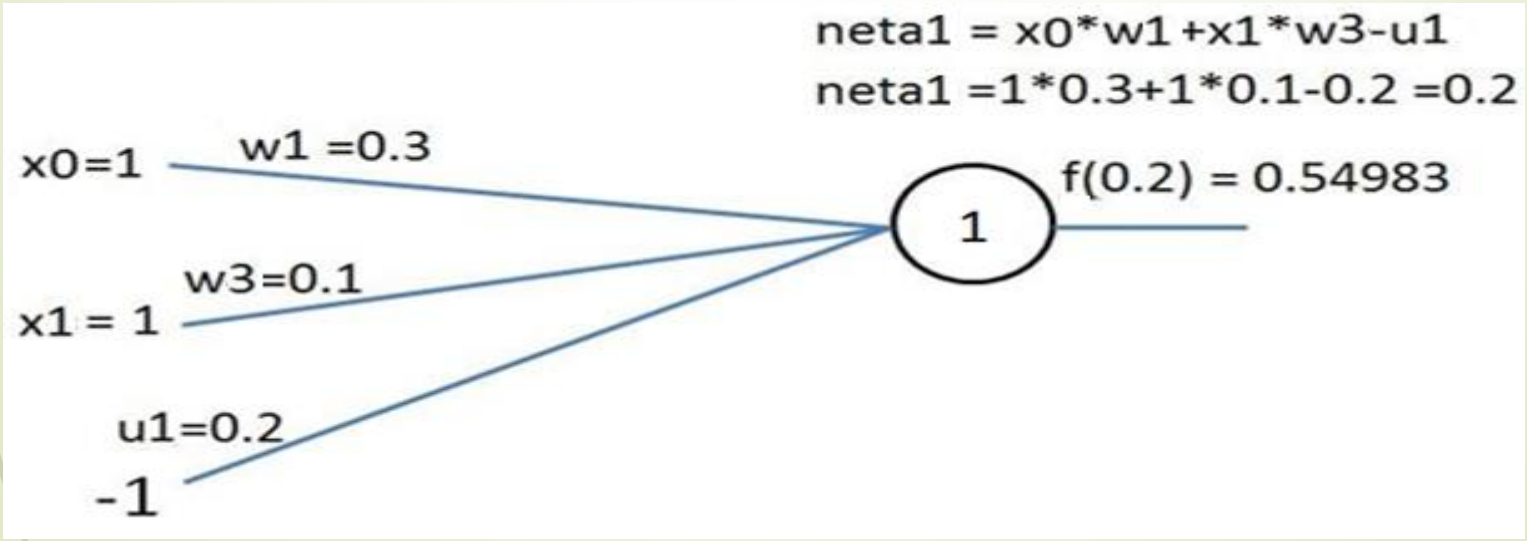
Mínimo error permitido = 0.1

- Ósea tenemos:

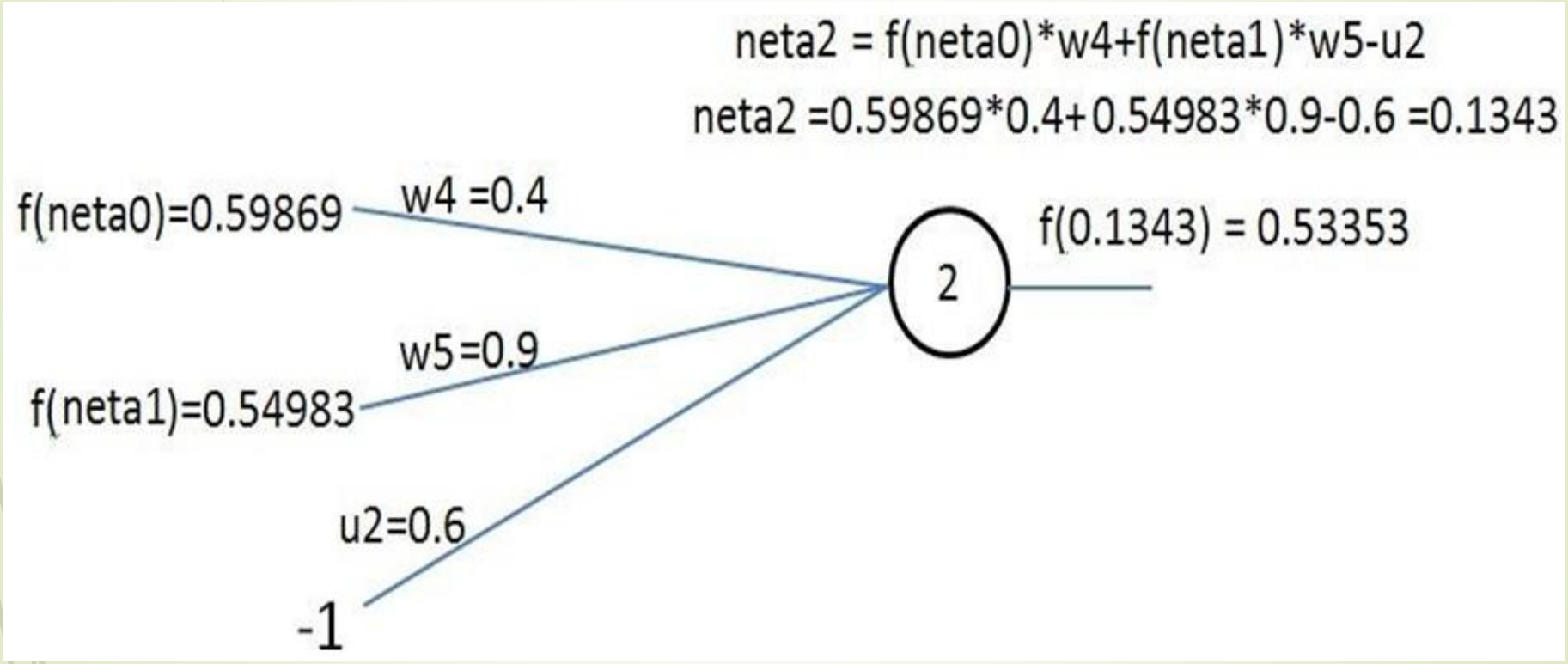


- Para las neuronas de la capa oculta tenemos:





- Para las neuronas de la capa de salida tenemos:



- Mostrar los errores para cada una de las neuronas:

$$\delta_0 = f'(neta0) * \delta_2 * w4$$

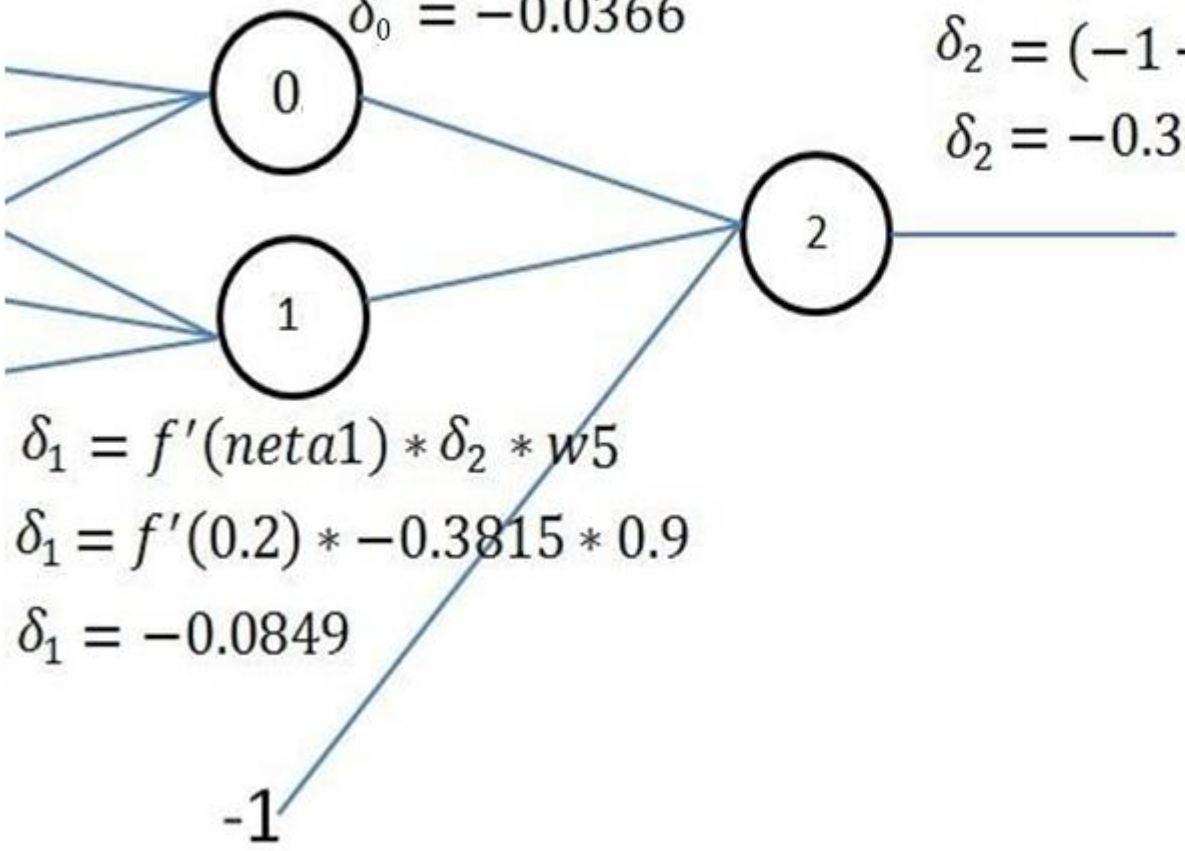
$$\delta_0 = f'(0.4) * -0.3815 * 0.4$$

$$\delta_0 = -0.0366$$

$$\delta_2 = \left(T - f(neta2)\right) * f'(neta2)$$

$$\delta_2 = (-1 - 0.53353) * 0.2488$$

$$\delta_2 = -0.3815$$



$$\delta_1 = f'(neta1) * \delta_2 * w5$$

$$\delta_1 = f'(0.2) * -0.3815 * 0.9$$

$$\delta_1 = -0.0849$$

- Actualización de pesos de la capa oculta a la capa de salida:

$$w5 = w5 + n * \delta_2 * f(neta1)$$
$$w5 = 0.9 + 0.65 * -0.3815 * 0.54983$$
$$w5 = 0.76387$$

$$w4 = w4 + n * \delta_2 * f(neta0)$$
$$w4 = 0.4 + 0.65 * -0.3815 * 0.59869$$
$$w4 = 0.2515$$

- **Actualización de pesos de la capa entrada a la capa oculta**:

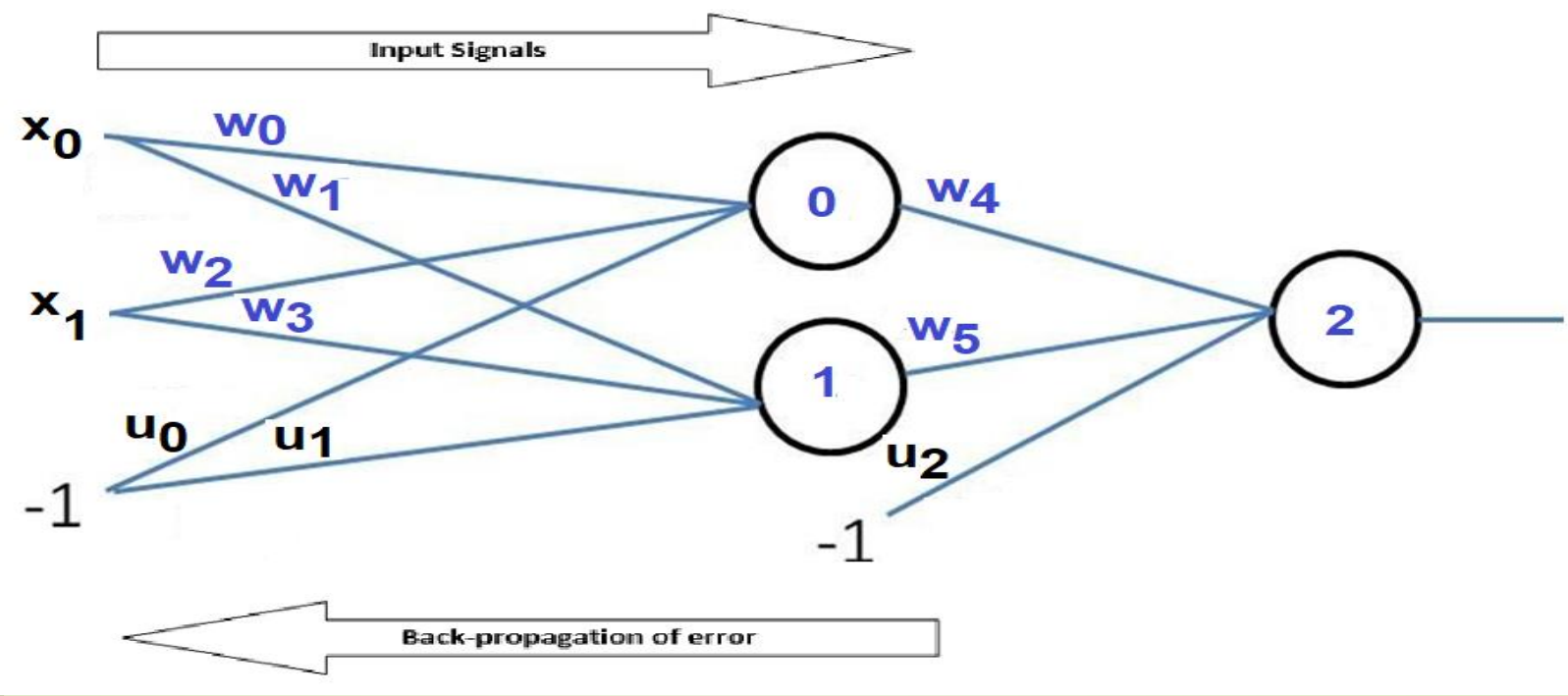
$$w0 = w0 + n * \delta_0 * x0$$
$$w0 = 0.5 + 0.65 * -0.0366 * 1$$
$$w0 = 0.4762$$

$$w1 = w1 + n * \delta_1 * x0$$
$$w1 = 0.3 + 0.65 * -0.0849 * 1$$
$$w1 = 0.2448$$

$$w2 = w2 + n * \delta_0 * x1$$
$$w2 = 0.7 + 0.65 * -0.0366 * 1$$
$$w2 = 0.6762$$

$$w3 = w3 + n * \delta_1 * x1$$
$$w3 = 0.1 + 0.65 * -0.0849 * 1$$
$$w3 = 0.0448$$

La situación se repite para cada uno de los patrones de entrada de cada registro, ahora con los nuevos pesos. Esto se repite hasta un limite de valor de error o de cantidad de iteraciones; en algunos casos conjuga los 2 parámetros de restricciones para detener el proceso de aprendizaje.



Comprobación

- Comúnmente llamado Mapeo, es la parte en el cual llega un caso (donde no pertenece dentro del grupo de patrones de entrenamiento).
- Para esto se debe obtener los pesos adecuados (ya entrenados) para poder discernir en un posible resultado idóneo.
- La comprobación va a ser de acuerdo a solo el primer paso del proceso de entrenamiento (Ósea tan solo el Forware). Pues las salidas van a ser los resultados.

```
1  import random
2  import numpy as np
3  import threading
4  import math
5  class Neurona:
6      n = 0.65
7      W = None#np.array([random.random(),random.random()])
8      U = None#random.random()
9      itera = -1
10     Emc = 10000000
11     def __init__(self):
12         print("Creado")
13
14     def f(self,x):
15         return 1.0/(1.0+math.exp(-x))
16     def fprima(self,x):
17         return self.f(x)*(1.0-self.f(x))
18
19     def operar(self,X,T,maxitera,minError):
20         while self.itera<maxitera and self.Emc >minError:
21             self.itera = self.itera + 1
22             Error = [0.0,0.0,0.0,0.0]
23             for p in range(4):
```

```
            for p in range(4):
                neta0 = X[p][0]*self.W[0] + X[p][1]*self.W[2] - self.U[0]
                neta1 = X[p][0]*self.W[1] + X[p][1]*self.W[3] - self.U[1]
                fneta0 = self.f(neta0)
                fneta1 = self.f(neta1)
                neta2 = fneta0*self.W[4] + fneta1*self.W[5] - self.U[2]
                fneta2 = self.f(neta2)# salida obtenida
                landa2 = (T[p] - fneta2)*self.fprima(neta2)
                landa1 = self.fprima(neta1)*landa2*self.W[5]
                landa0 = self.fprima(neta0)*landa2*self.W[4]
                # actualizar los pesos sinapticos
                # los pesos de la capa salida
                self.W[5] = self.W[5] +self.n*landa2*fneta1
                self.W[4] = self.W[4] +self.n*landa2*fneta0
                # actualiza los pesos de la capa oculta
                self.W[0] = self.W[0] + self.n*landa0*X[p][0]
                self.W[1] = self.W[1] + self.n*landa1*X[p][0]
                self.W[2] = self.W[2] + self.n*landa0*X[p][1]
                self.W[3] = self.W[3] + self.n*landa1*X[p][1]
                # actualizar los umbrales
                self.U[2] = self.U[2] + self.n*landa2*-1
                self.U[1] = self.U[1] + self.n*landa1*-1
                self.U[0] = self.U[0] + self.n*landa0*-1
```

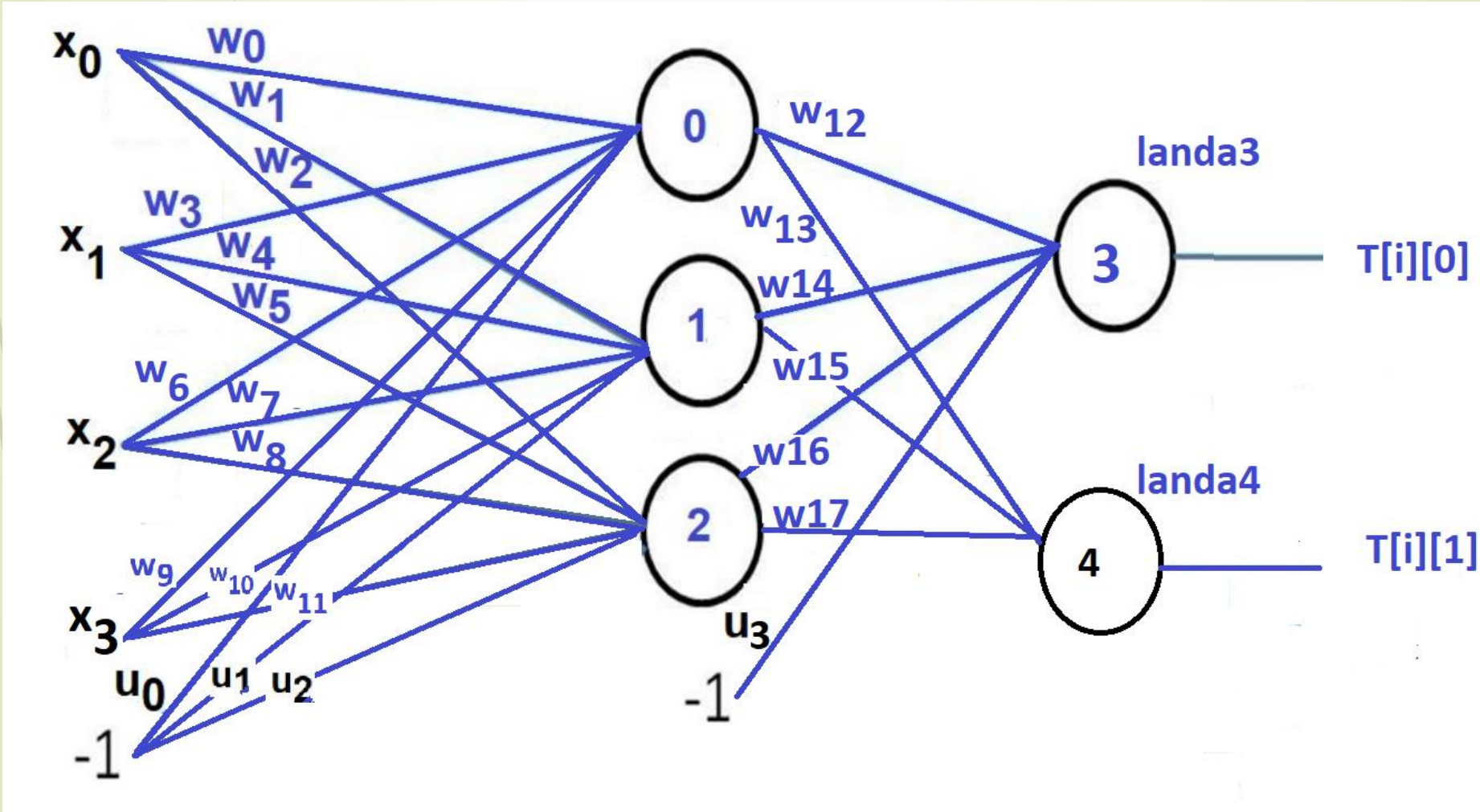
```
45                  self.U[0] = self.U[0] + self.n*landa0*-1
46                  Error[p] = 0.5*(T[p] - fneta2)*(T[p] - fneta2)
47
48              self.Emc = (Error[0] + Error[1] + Error[2] + Error[3])/len(Error)
49              print(self.itera," Error medio cuadratico: ",self.Emc)
50
51          print("Pesos sinapticos aprendidos")
52
53      def entrenar(self,X,T,maxitera,minError):
54          self.W = [random.random() for _ in range(6)]
55          self.U = [random.random() for _ in range(3)]
56          hilo = threading.Thread(target=self.operar, args=(X,T,maxitera,minError,))
57          hilo.start()
58          hilo.join()
59
60      def valor(x):
61          if x>0.5:
62              return 1
63          else:
64              return -1
65
66      def mapear(self,Xi):
67          neta0 = Xi[0]*self.W[0] + Xi[1]*self.W[2] - self.U[0]
```

```
67            neta0 = Xi[0]*self.W[0] + Xi[1]*self.W[2] - self.U[0]
68            neta1 = Xi[0]*self.W[1] + Xi[1]*self.W[3] - self.U[1]
69            fneta0 = self.f(neta0)
70            fneta1 = self.f(neta1)
71            neta2 = fneta0*self.W[4] + fneta1*self.W[5] - self.U[2]
72            fneta2 = self.f(neta2)# salida obtenida
73            if fneta2>0.5:
74                return 1
75            else:
76                return -1
77
78    X = np.array([[1.0 , 1.0],
79          [1.0, -1.0 ],
80          [-1.0 , 1.0],
81          [-1.0 ,-1.0]])
82    T = np.array([-1.0,1.0 ,1.0,-1.0])
83    n = Neurona()
84    maxitera = 1000
85    minError = 0.1
86    n.entrenar(X,T,maxitera,minError)
87    print("----- mapeo ----------")
88    for i in range(4):
89        print(X[i,:],n.mapear(X[i,:]))
```

- Preguntas ?

Tarea individual de laboratorio 1

Implementar en Python la siguiente red neuronal artificial



$w_0 = 0.5$ $w_1 = 0.3$ $w_2 = 0.7$ $w_3 = 0.1$ $w_4 = 0.4$
$w_5 = 0.9$ $w_6 = 0.8$ $w_7 = 0.9$ $w_8 = 0.5$ $w_9 = 0.4$
$w_{10} = 0.7$ $w_{11} = 0.9$ $w_{12} = 0.4$ $w_{13} = 0.7$ $w_{14} = 0.9$
$w_{15} = 0.8$ $w_{16} = 0.7$ $w_{17} = 0.6$

$u_0 = 0.8$ $u_1 = 0.2$ $u_2 = 0.6$ $u_3 = 0.9$ $u_4 = 0.8$

$n = 0.65$

maxiteraciones = 2500

Error mínimo cuadrático permitido = 0.1

X_0	X_1	X_2	X_3	T_0	T_1
1	1	1	1	1	-1
1	-1	-1	1	-1	1
-1	1	1	1	1	-1
-1	-1	-1	1	-1	1

Tarea idividual de laboratorio

Crear la compuerta lógica XOR con el backpropagtion, usando procesos o hilos.

Nota: Cada neurona debe ser un proceso o hilo.