Red multicapa

jafh

Inicio

Objetivo:

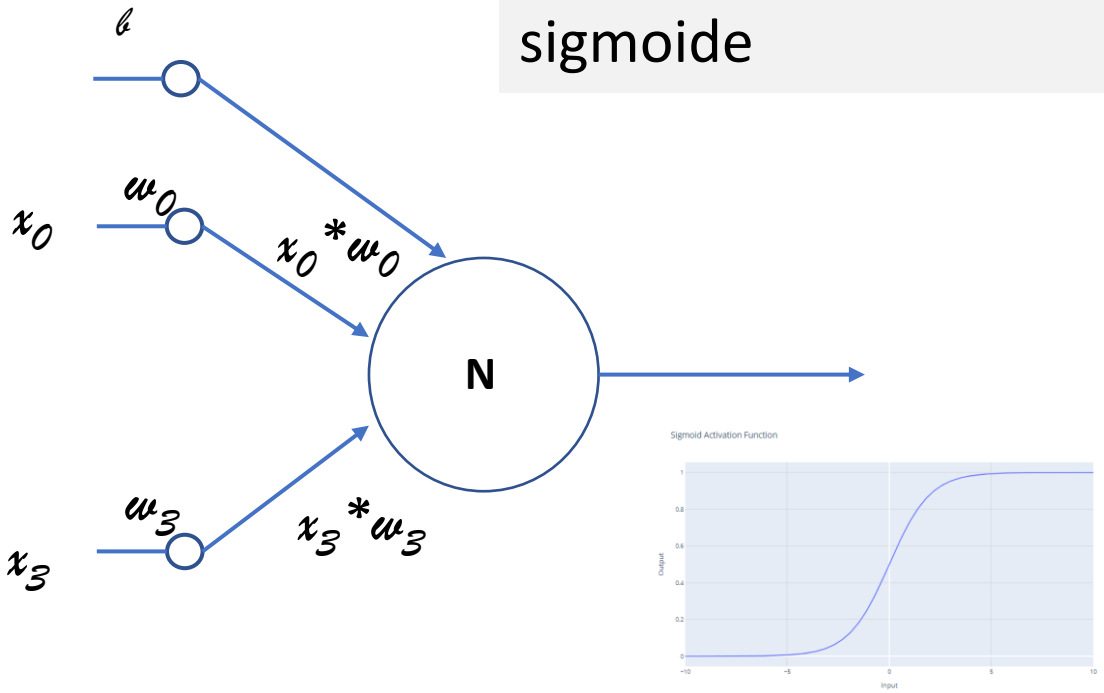
Entender el funcionamiento de una red de 4 entradas 10 neuronas ocultas y 3 neuronas de salida.

Repaso de la neurona

Usaremos neuronas del tipo:

x0w0+x1w1+x2w2+x3w3+b

Usaremos la función de activación sigmoide



Repaso de funcionamiento de l neurona

1. Iniciar los pesos wi con valores aleatorios: w[w0,w1,w2,w3]
2. Calcular salida Z(k)= $sign(\sum_{j=0}^{3} wjxj$+b) para cada iteración k
3. Si Z(k)!=Y(k) modificar los pesos wj como sigue:

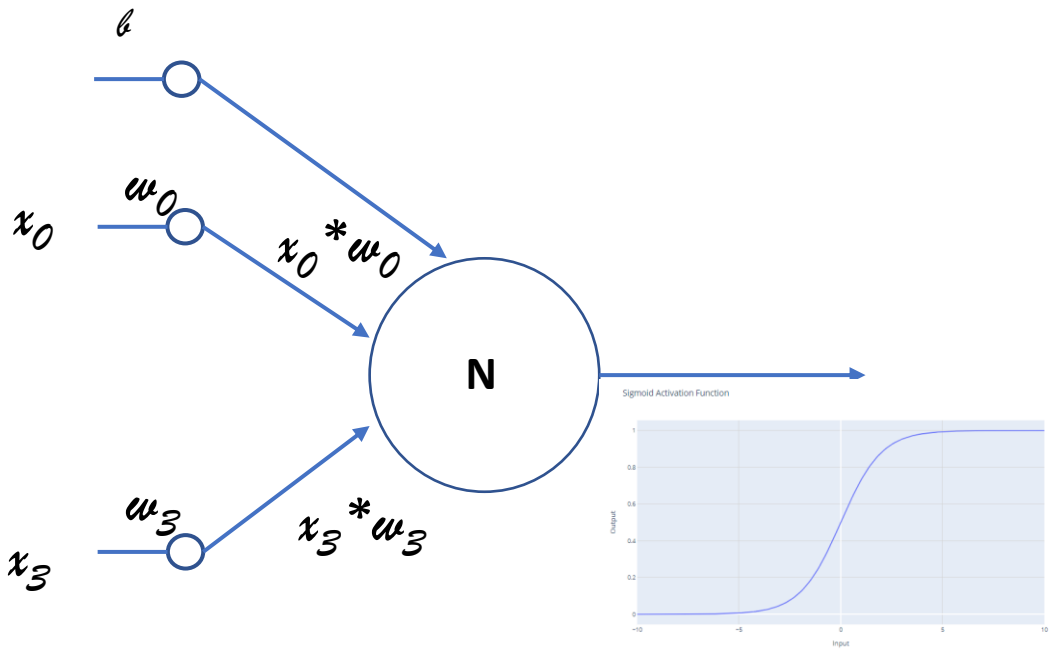
   w_j(k+1)=$w_j(k) + \eta[z(k) - y(k)] * x_j$(k)

   b(k+1)=$b(k) + \eta[z(k) - y(k)]$

4. Si se cumple criterio de paro? fin si no regresar a 2

El algoritmo anterior muestra que se usa la función signo, nosotros usaremos la función sigmoide.

Gradiente



x0w0+x1w1+x2w2+x3w3+b

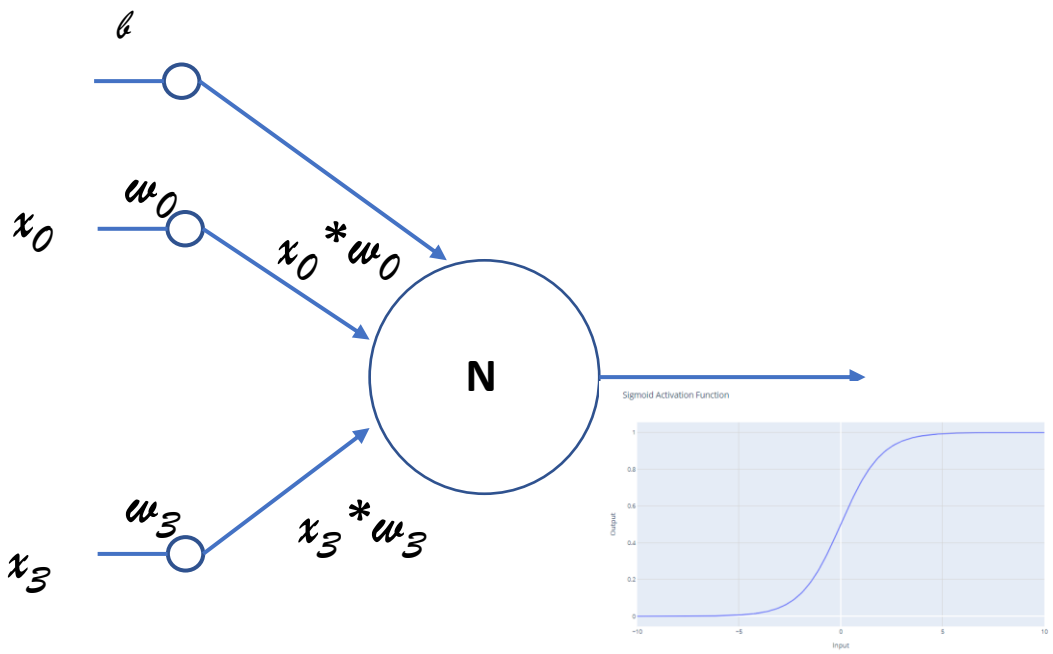
Si usamos activación sigmoide, el gradiente de:

f=b+x0w0+x1w1+x2w2+x3w3

Es:

$\nabla f=(1,x0,x1,x2,x3)$

Gradiente



Para reducir la perdida usaremos la técnica de descenso de gradiente en la que en cada iteración se agrega a wi alguna fracción de la magnitud del gradiente:

Y= salida correcta

Z = $sigmoid(\sum_{j=0}^{3} wjxj$+b)

η = tasa de aprendizaje

error=Y-Z

wi=wi+η(error* sigmoid_derivada(Z));

Redes multicapa

La retro propagación es un algoritmo clave utilizado en redes neuronales para entrenar modelos de aprendizaje profundo.

1. Propagación hacia adelante: El modelo toma la entrada y la pasa a través de la red, capa por capa, hasta llegar a la salida. En cada capa, se aplican funciones de activación y pesos para transformar la entrada en una salida.

2. Cálculo del error: Una vez obtenida la salida, se compara con el valor esperado (etiqueta) utilizando una función de pérdida (como el error cuadrático medio o la entropía cruzada). Esto nos da la diferencia entre la predicción de la red y el valor real.

Retro propagación

3. Retro propagación del error: Aquí es donde entra la magia. El error se propaga hacia atrás a través de la red, desde la salida hasta las capas intermedias y la entrada. Se utilizan las derivadas parciales (gradientes) de la función de pérdida con respecto a los pesos de la red para ajustar estos pesos y minimizar el error.

4. Actualización de pesos: Los pesos de la red se actualizan en función del gradiente calculado y una tasa de aprendizaje predeterminada. Este proceso se repite para cada ejemplo en el conjunto de entrenamiento hasta que la red alcance un rendimiento aceptable.

5. Iteración: El ciclo de propagación hacia adelante, cálculo del error, retro propagación y actualización de pesos se repite muchas veces a lo largo de varias épocas hasta que la red aprende a realizar predicciones precisas.

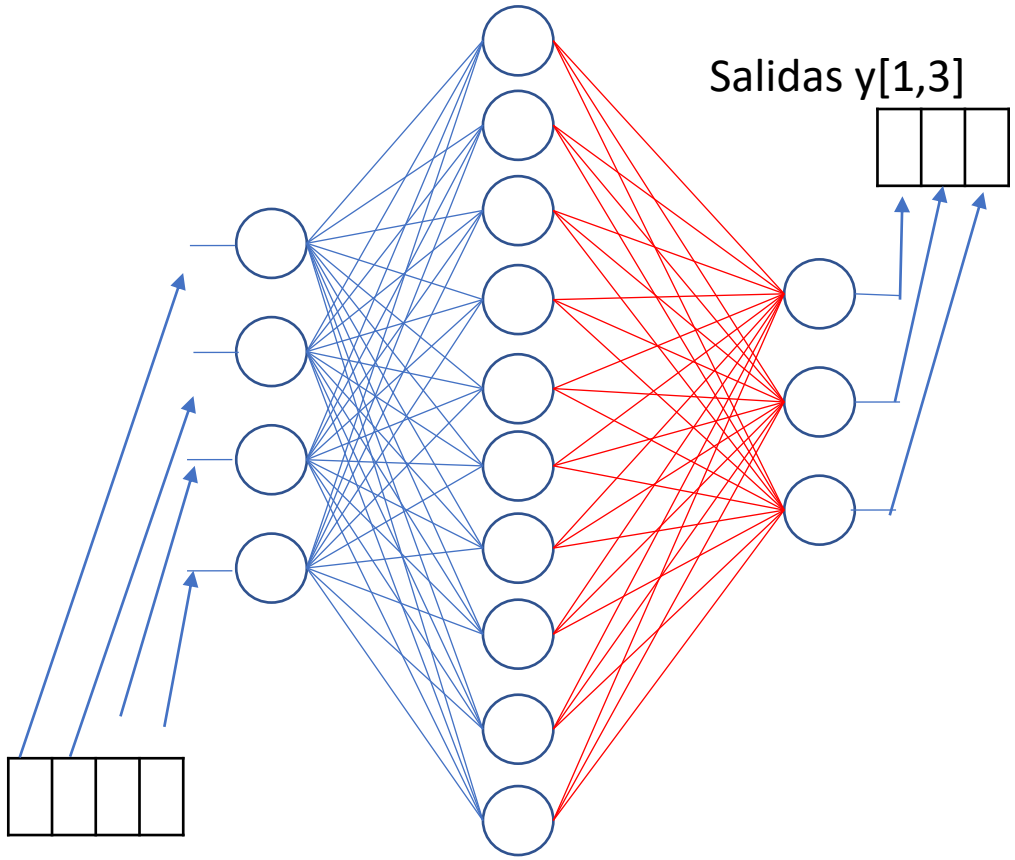
Este proceso es fundamental para el aprendizaje profundo y permite que las redes neuronales ajusten sus parámetros de manera eficiente para resolver problemas complejos como reconocimiento de imágenes, procesamiento de lenguaje natural.

Código en Scilab: Usaremos estas funciones

```
//% Función de activación sigmoide
function y = sigmoid(x)
    y = 1 ./ (1 + exp(-x));
endfunction

//% Derivada de la sigmoide
function y = sigmoid_derivada(x)
    y = sigmoid(x) .* (1 - sigmoid(x));
endfunction
```

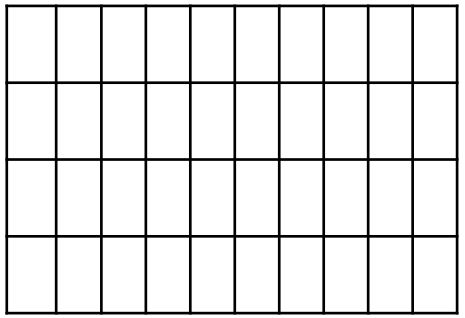
Red de 4 * 10 * 3

b1=[1,10]

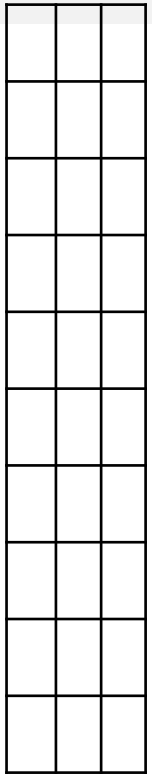



```
// Parámetros de la red
n_entradas = 4;  // Número de entradas
n_ocultas = 10;  // Neuronas en la capa oculta
n_salidas = 3;   // Neuronas en la capa de salida
n_num_dat_ent = 30 //# datos de entrenamiento

// Inicializar pesos y sesgos aleatoriamente
W1 = rand(n_entradas, n_ocultas);  // Pesos capa oculta
b1 = rand(1, n_ocultas);           // Sesgos capa oculta
W2 = rand(n_ocultas, n_salidas);   // Pesos capa salida
b2 = rand(1, n_salidas);           // Sesgos capa salida
```

W1=[4,10]



W2=[10,3]

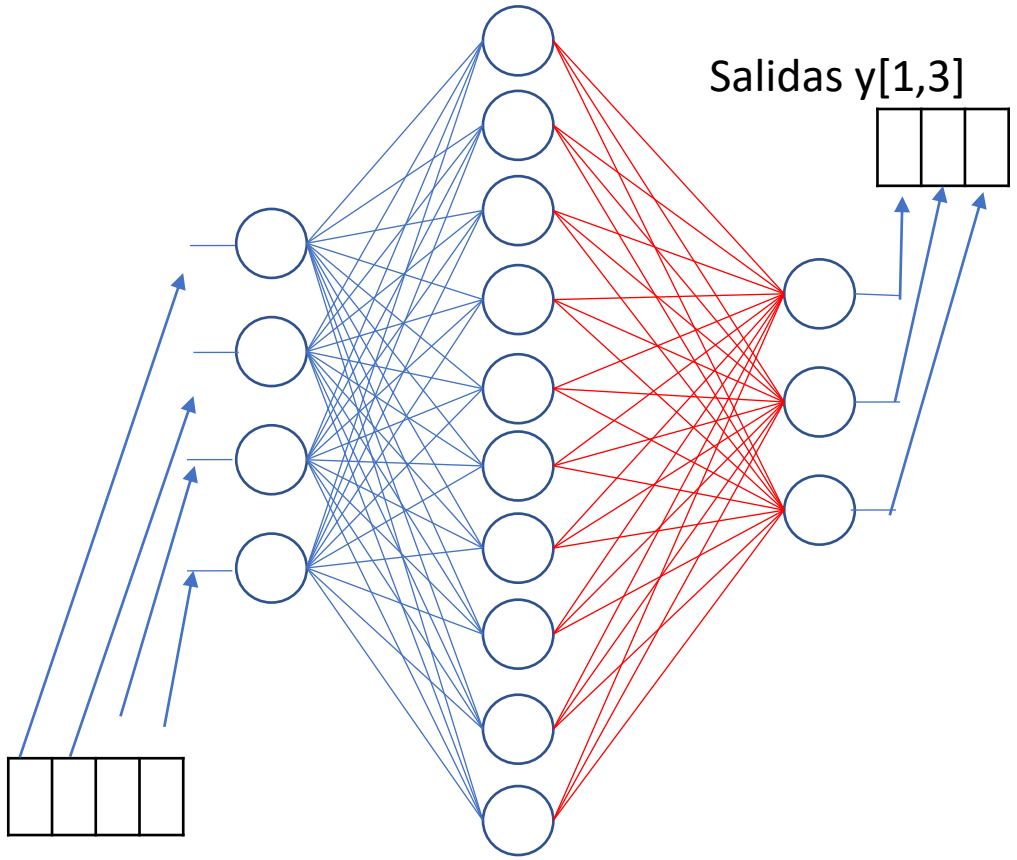


b2=[1,3]

Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

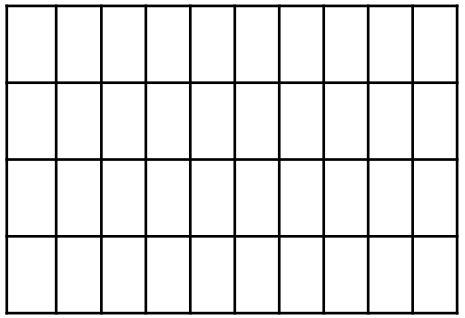
Entrenamiento

b1=[1,10]

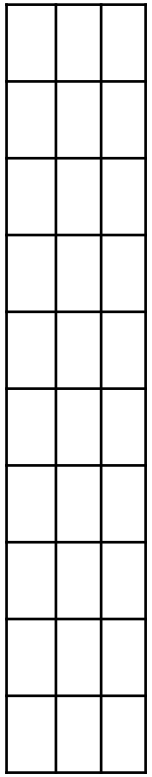



```
TMP = X * W1;
//TMP=X[30,4]*W1[4,10]=[30,10]
//cada renglón de X es un ejemplo
//cada columna i de W1 son los 4 pesos de cada neurona oculta
//TMP en cada renglón tiene las entradas ponderadas
//ahora a cada renglón le agregamos el sesgo de b1
//para agregarlo a todas las muestra expandamos b1
b1_expanded = repmat(b1, n_num_dat_ent,1);
Z1 = X * W1 + b1_expanded;
//Z1[30,4]
```

W1=[4,10]



W2=[10,3]

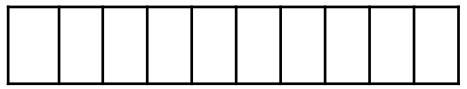


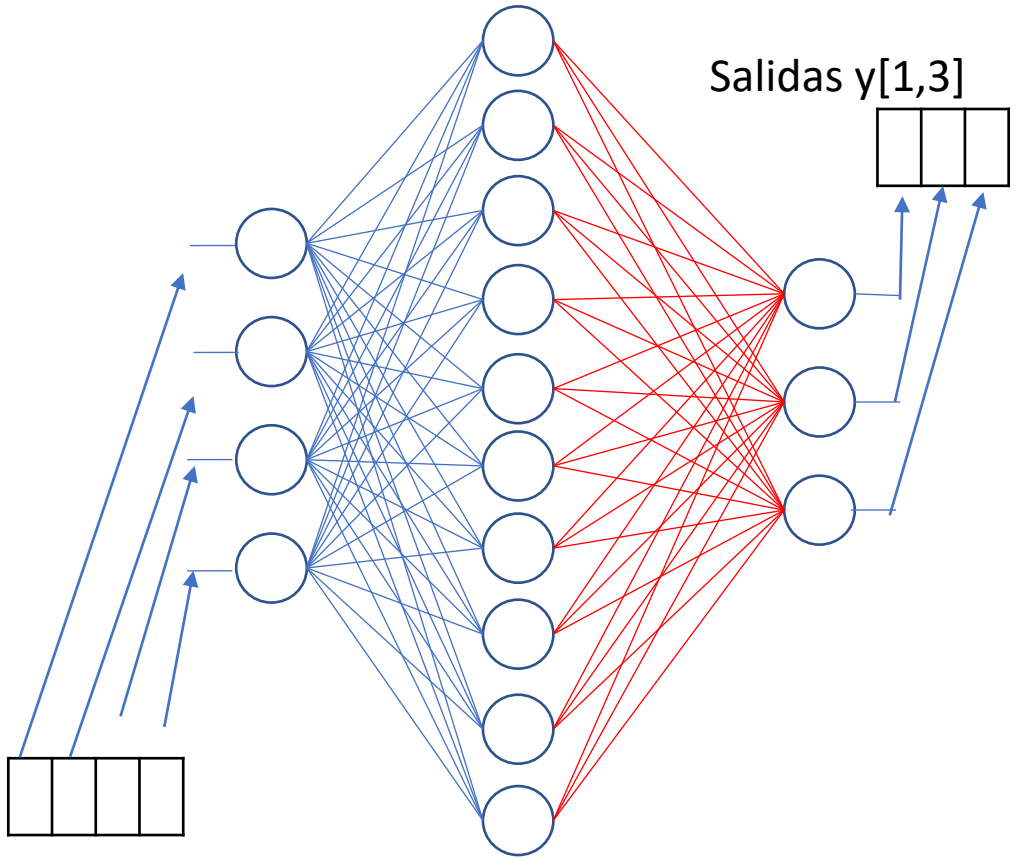
b2=[1,3]

Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

Entrenamiento

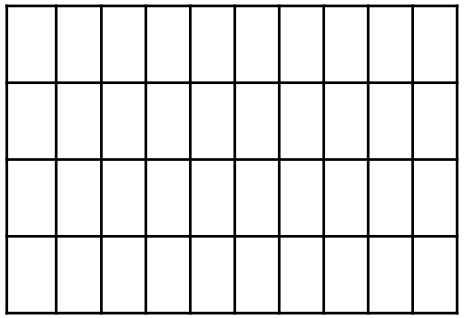
b1=[1,10]



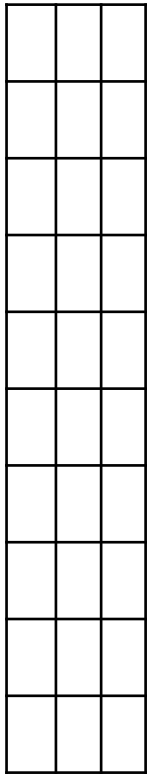



```
//recordemos que Z1 = X * W1 + b1_expanded;
//ahora calculamos la salida de cada neurona oculta
A1 = sigmoid(Z1); // en rojo
//A1[30,4] son las salidas ocultas para todas las muestras
//caculamos las sumas de la capa de salida
TMP2= A1 * W2 ; //TMP2=[30,4]*[10,3]=[30,3]
b2_expanded = repmat(b2, n_num_dat_ent,1);
Z2 = TMP2 + b2_expanded;
//Z2[30,3], cada renglón de Z2 tiene
//las sumas ponderadas de cada neurona de salida
```

W1=[4,10]



W2=[10,3]



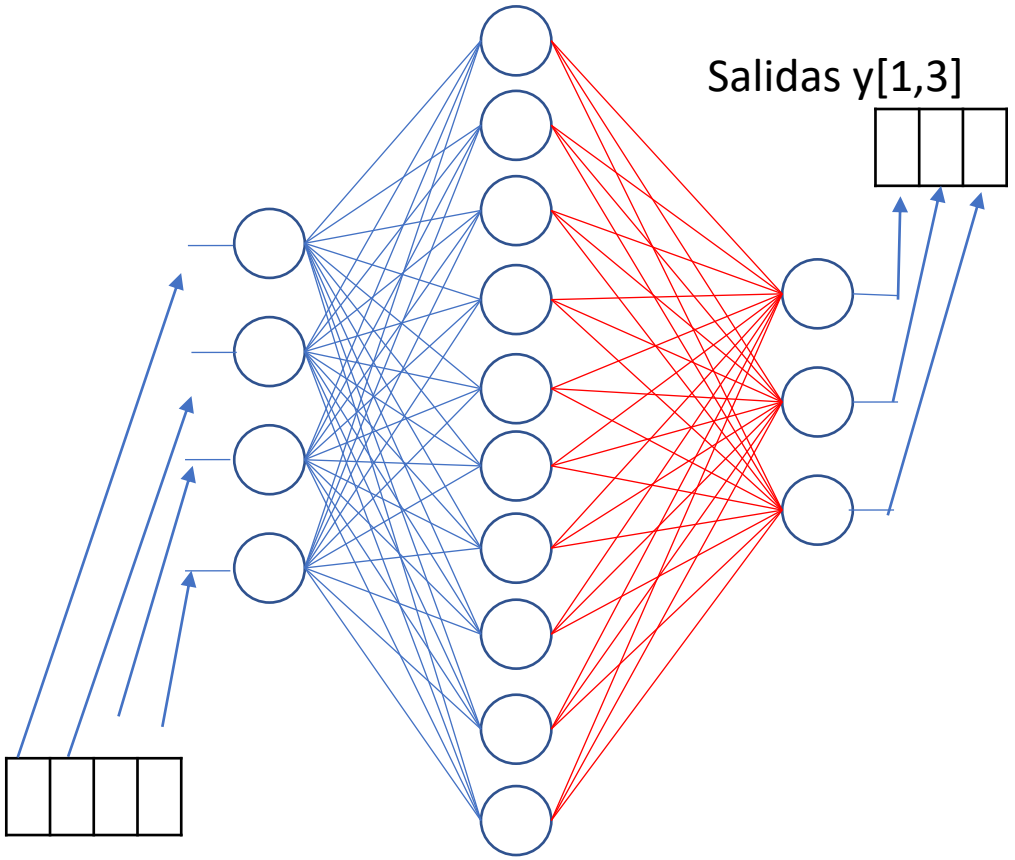
b2=[1,3]

Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

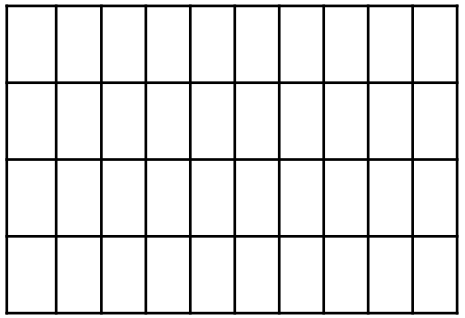
Entrenamiento

```
//Calculamos las salidas de la capa de salida
//para todas las muestras
A2 = sigmoid(Z2);
//A2[30,3]
```

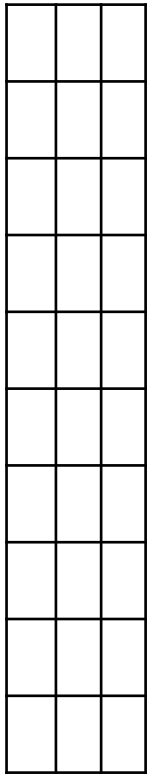
b1=[1,10]



W1=[4,10]



W2=[10,3]



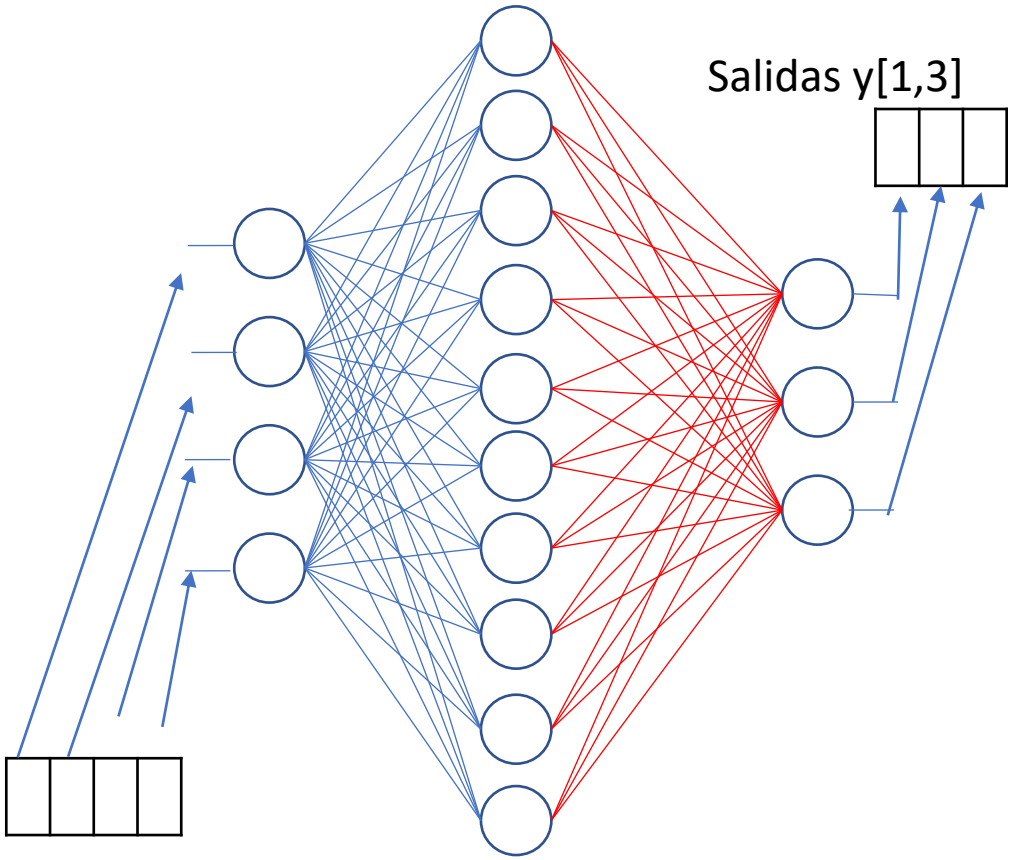
b2=[1,3]

Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

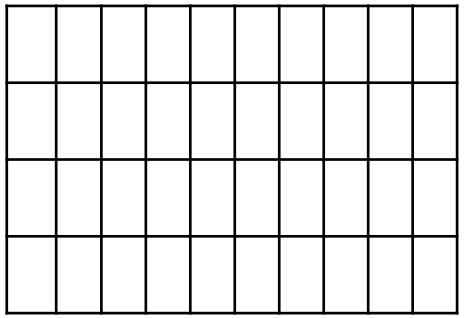
Calcular error

```
error = Y - A2;
//error=[30,3] – [30,3] = [30,3]
//cada renglón de error tiene el error  del forward
//de cada muestra
```

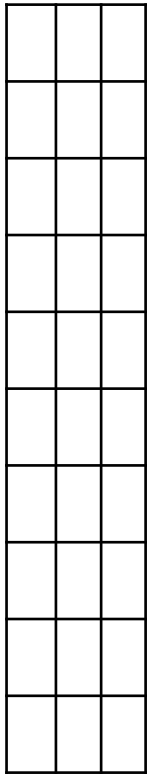
b1=[1,10]



W1=[4,10]



W2=[10,3]



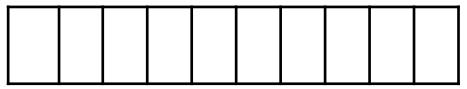
b2=[1,3]

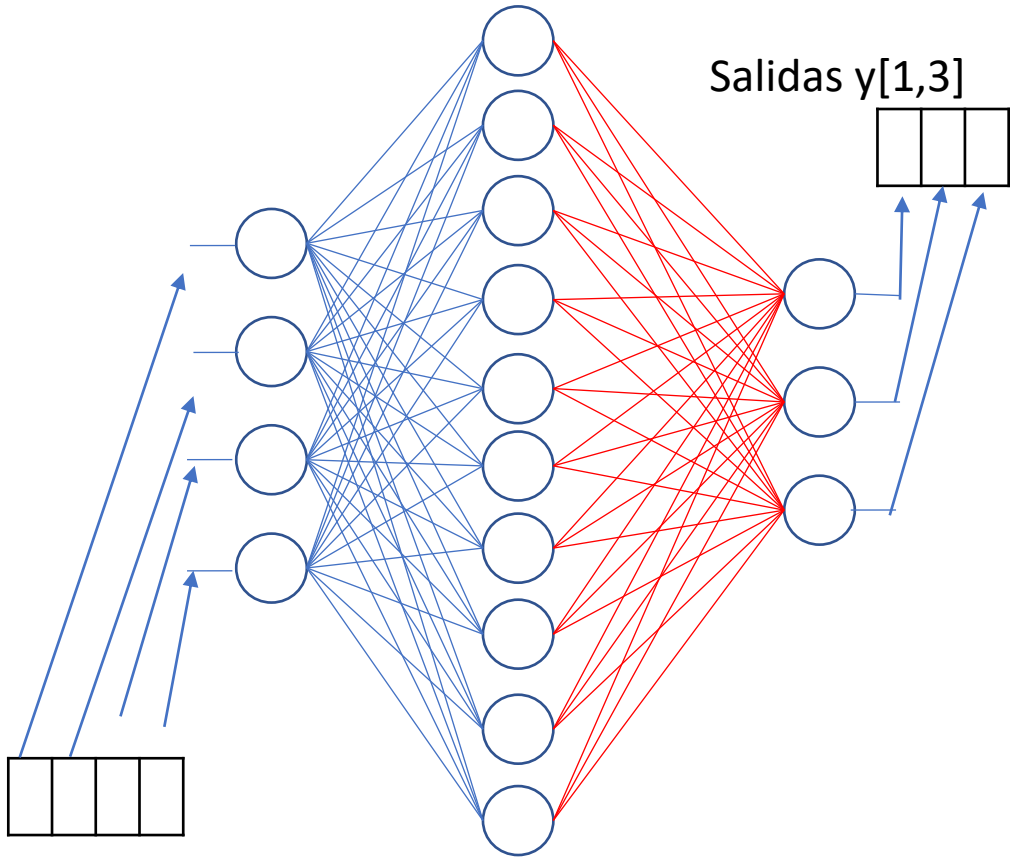
Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

Retropropagación

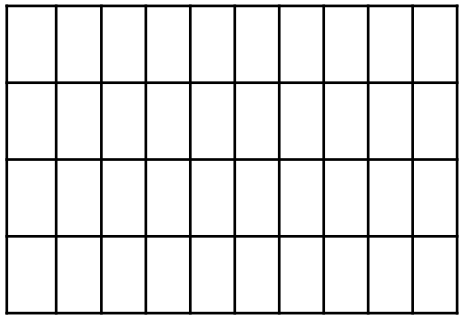
```
//Retropropagación
dZ2 = error .* sigmoid_derivada(Z2);
dW2 = A1' * dZ2;
db2 = sum(dZ2, 1);

dZ1 = (dZ2 * W2') .* sigmoid_derivada(Z1);
dW1 = X' * dZ1;
db1 = sum(dZ1, 1);
```

b1=[1,10]

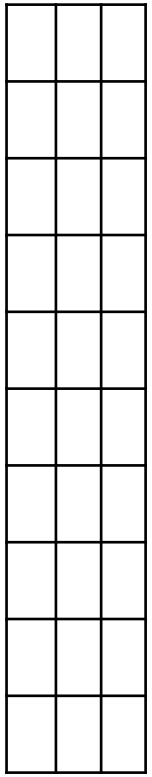




W1=[4,10]



W2=[10,3]



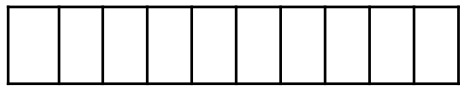
b2=[1,3]

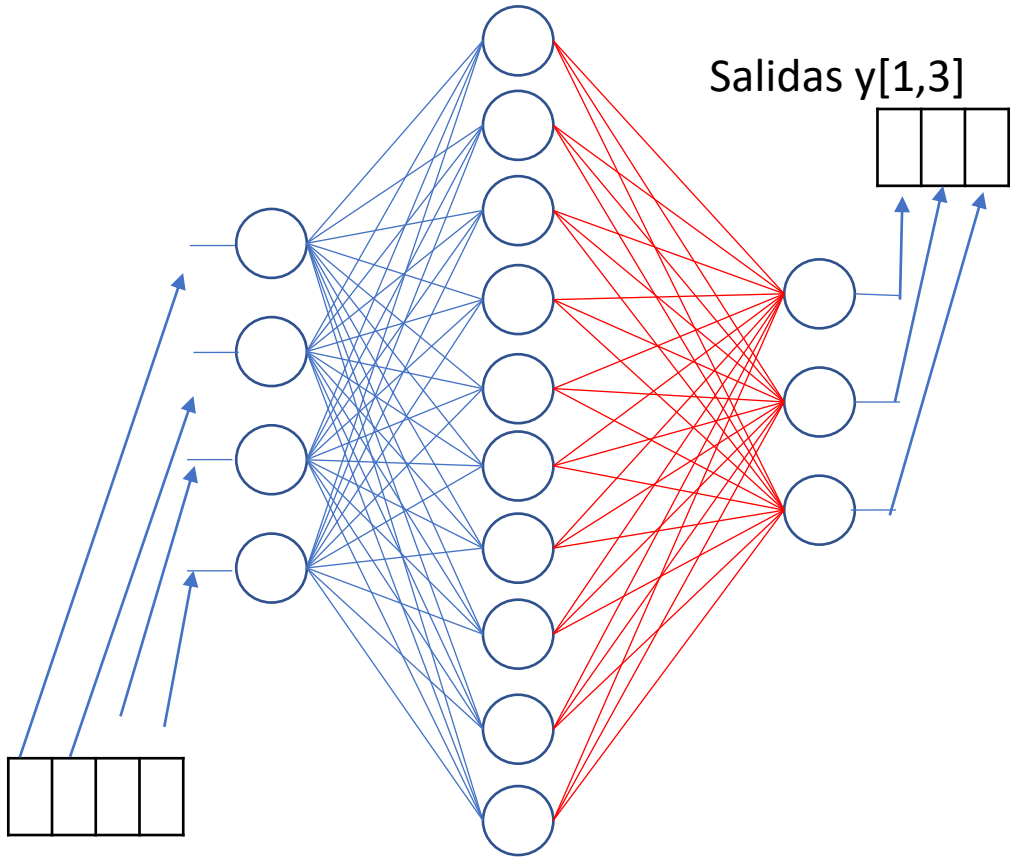
Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

Actualizar pesos

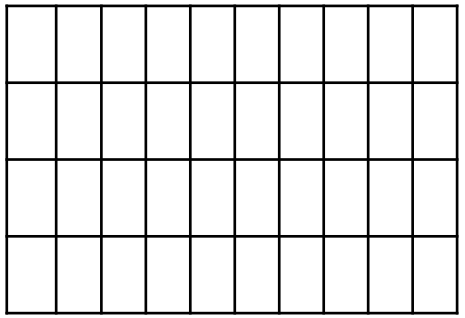
```
//% Actualizar pesos y sesgos
W2 = W2 + tasa_aprendizaje * dW2;
b2 = b2 + tasa_aprendizaje * db2;
W1 = W1 + tasa_aprendizaje * dW1;
b1 = b1 + tasa_aprendizaje * db1;
```

b1=[1,10]

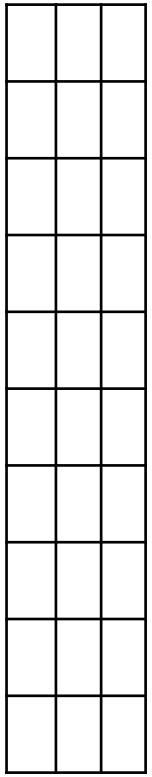




W1=[4,10]



W2=[10,3]



b2=[1,3]

Entradas de la neurona x=[1,4]
Para el entrenamiento usaremos 30 datos =>
Necesitamos X[30,4], Y[30,3]

Función repmat de SCILAB

La sintaxis básica de la función repmat de SCILAB es la siguiente:

B = repmat(A, m, n) retorna un array formado por un mosaico m-por-n con copias de A.

Si b1 es de [1,10]

repmat(b1,30,1) es de [30,10] repitiendo los renglones 30 veces.

Fin

Jafh