Universidad Autónoma del Carmen

FACULTAD DE CIENCIAS DE LA INFORMACION

Asignatura: Reconocimiento de patrones

Docente: Mtr. Jesús Alejandro Flores Hernández

Alumno: Valeria De Los Angeles Cruz May

Carrera: Ing. En tec. De cómputo y comunicaciones

Matricula: 220103

Ciudad del Carmen, Campeche,
jueves, 20 de marzo de 2025.

```
//Autor: Jflores
//marzo 2025
//red neuronal de 4 entradas 10 neuronas ocultas y 3 salidas
//para predecir salidas de las observaciones de la flor IRIS
//probado con 30 observaiones de IRIS
//Las salidas que pueden ser 3 valores (1,2,3) se toman como
//[1,0,0] paraa 1, [0,1,0] para 2 y [0,0,1] para 3 y se usa
//la función sigmoide en la salida de las neuronas.
```

```
10 // Parámetros de la red
11 n_entradas = 4;   // Número de entradas
12 n_ocultas = 10;   // Neuronas en la capa oculta
13 n_salidas = 3;    // Neuronas en la capa de salida
14 n_num_dat_ent = 30 //numero de datos de entrenamiento
15
```

Número de entradas:

Este parámetro define el número de características que se utilizan como entrada para la red neurona que son:

•Largo del sépalo

•Ancho del sépalo

•Largo del pétalo

•Ancho del pétalo

Neuronas en la capa oculta:

Este parámetro define el número de neuronas en la capa oculta de la red neuronal. toma las entradas, las multiplica por los pesos, suma los sesgos y aplica una función de activación (sigmoide) para producir una salida.

Neuronas en la capa de salida:

Este parámetro define el número de neuronas en la capa de salida de la red neuronal. En este caso, hay tres neuronas en la capa de salida porque estamos clasificando las flores en tres especies diferentes:

1. Iris setosa
2. Iris versicolor
3. Iris virginica

Número de datos de entrenamiento:

Este parámetro define el número de observaciones del conjunto de datos que se utilizan para entrenar la red neuronal. En este caso, se están utilizando 30 observaciones de las flores Iris.

```
16  //% Función de activación sigmoide
1   function y = sigmoid(x)
2       y = 1 ./ (1 + exp(-x));
3   endfunction
20
```

La **Función de activación sigmoide** transforma el valor de entrada x en un valor entre 0 y 1. Esto es útil para modelar probabilidades.

La función devuelve un valor y que está en el rango de 0 a 1. Este valor representa la activación de la neurona y se utiliza para modelar probabilidades.

```
21  //% Derivada de la sigmoide
1   function y = sigmoid_derivada(x)
2       y = sigmoid(x) .* (1 - sigmoid(x));
3   endfunction
25
```

La derivada de la función sigmoide se utiliza durante el proceso de retropropagación en el entrenamiento de la red neuronal.

Entrada: Un valor x (puede ser un número, vector o matriz).

Proceso: Aplica la función sigmoide a x y calcula el producto de $\sigma(x)\sigma(x)$ y $(1-\sigma(x))(1-\sigma(x))$.

Salida: Devuelve la derivada de la función sigmoide aplicada a x.

Ayuda a calcular los gradientes necesarios para ajustar los pesos y sesgos de la red.

```
// Inicializar pesos y sesgos aleatoriamente
W1 = rand(n_entradas, n_ocultas);  // Pesos capa oculta
```

- **W1**: Matriz de pesos para la capa oculta.
- **rand(n_entradas, n_ocultas):** Genera una matriz de tamaño n_entradas x n_ocultas con valores aleatorios entre 0 y 1.
- **n_entradas:** Número de características de entrada (4 en este caso).
- **n_ocultas**: Número de neuronas en la capa oculta (10 en este caso).

```
b1 = rand(1, n_ocultas);                    // Sesgos capa oculta
```

- **b1:** Vector de sesgos para la capa oculta.
- **rand(1, n_ocultas):** Genera un vector de tamaño 1 x n_ocultas con valores aleatorios entre 0 y 1

```
W2 = rand(n_ocultas, n_salidas);  // Pesos capa salida
```

- **W2: Matriz de pesos para la capa de salida.**
- **rand(n_ocultas, n_salidas): Genera una matriz de tamaño n_ocultas x n_salidas con valores aleatorios entre 0 y 1.**
- **n_ocultas: Número de neuronas en la capa oculta (10 en este caso).**
- **n_salidas: Número de neuronas en la capa de salida (3 en este caso).**

```
b2 = rand(1, n_salidas);                    // Sesgos capa salida
```

- **b2: Vector de sesgos para la capa de salida.**
- **rand(1, n_salidas): Genera un vector de tamaño 1 x n_salidas con valores aleatorios entre 0 y 1.**

```
//30 datos de las observaciones IRIS y sus etiquetas
IRIS_DATA=[
[5.1,3.5,1.4,0.2];[4.9,3,1.4,0.2];[4.7,3.2,1.3,0.2];[4.6,3.1,1.5,0.2];[5,3.6,1.4,0.2];
[5.4,3.9,1.7,0.4];[4.6,3.4,1.4,0.3];[5,3.4,1.5,0.2];[4.4,2.9,1.4,0.2];[4.9,3.1,1.5,0.1];
[6.4,3.2,4.5,1.5];[6.9,3.1,4.9,1.5];[5.5,2.3,4,1.3];[6.5,2.8,4.6,1.5];[5.7,2.8,4.5,1.3];
[6.3,3.3,4.7,1.6];[4.9,2.4,3.3,1];[6.6,2.9,4.6,1.3];[5.2,2.7,3.9,1.4];[5,2,3.5,1];
[6,3,4.8,1.8];[6.9,3.1,5.4,2.1];[6.7,3.1,5.6,2.4];[6.9,3.1,5.1,2.3];[5.8,2.7,5.1,1.9];
[6.8,3.2,5.9,2.3];[6.7,3.3,5.7,2.5];[6.7,3,5.2,2.3];[6.3,2.5,5,1.9];[6.5,3,5.2,2]
];
Y_DATA=[
[1,0,0];[1,0,0];[1,0,0];[1,0,0];[1,0,0];[1,0,0];[1,0,0];[1,0,0];[1,0,0];[1,0,0];
[0,1,0];[0,1,0];[0,1,0];[0,1,0];[0,1,0];[0,1,0];[0,1,0];[0,1,0];[0,1,0];[0,1,0];
[0,0,1];[0,0,1];[0,0,1];[0,0,1];[0,0,1];[0,0,1];[0,0,1];[0,0,1];[0,0,1];[0,0,1]
];
```

En esta sección, **IRIS_DATA** contiene las características de las flores (longitud y ancho del sépalo, longitud y ancho del pétalo) y

Y_DATA contiene las etiquetas correspondientes a las clases de las flores (Setosa, Versicolor, Virginica) en formato de codificación one-hot.

```
52 X=IRIS_DATA;
53 Y=Y_DATA;
54 //mostrar entradas
55 disp("Datos de entrada")
56 disp(cat(2,X,Y))
57
```

X se asigna a IRIS_DATA, que contiene las características de las flores IRIS, y
Y se asigna a **Y_DATA**, que contiene las etiquetas correspondientes

Muestra los datos de entrada en la consola.

La función **cat(2, X, Y)** concatena las matrices X y Y horizontalmente (a lo largo de las columnas), de modo que cada fila de la salida muestra las características de una flor junto con su etiqueta correspondiente.

```
58 // entrenamiento
59 disp("Entrenamiento:")
60 //% Hiperparámetros
61 tasa_aprendizaje = 0.1;
62 max_iter = 1000;
63
```

tasa_aprendizaje: Este hiperparámetro controla la velocidad con la que la red neuronal ajusta sus pesos durante el entrenamiento. Una tasa de aprendizaje de 0.1 significa que los pesos se ajustarán en cada iteración en un 10% de la magnitud del gradiente calculado.

max_iter: Este hiperparámetro define el número máximo de iteraciones que el algoritmo de entrenamiento realizará. En este caso, el entrenamiento se ejecutará durante 1000 iteraciones.

Este bucle se ejecuta max_iter veces (en este caso, 1000 veces) para ajustar los pesos de la red neuronal.

```
64  //% Entrenamiento
65  for iter = 1:max_iter
66      //% Propagación hacia adelante
67      b1_expanded = repmat(b1, n_num_dat_ent,1);
68      //Expande b1[1,10] para que sea [30,10]
69      //para sumar a esto:  X * W1 = [30,4]*[4,10]=[30,10]
70      //suma de las entradas ponederadas + los sesgos
71      Z1 = X * W1 + b1_expanded
72      //A1 salidas de la capa oculta
73      A1 = sigmoid(Z1);
74      //A2 salidas de la capa de salida
75      b2_expanded = repmat(b2, n_num_dat_ent,1);
76      Z2 = A1 * W2 + b2_expanded;
77      A2 = sigmoid(Z2);
```

Aquí, se multiplican las entradas X por los pesos W1 y se suman los sesgos expandidos b1_expanded. Esto da como resultado Z1, que es la entrada ponderada para la capa oculta.

La función sigmoide se aplica a Z1 para obtener A1, que son las salidas de la capa oculta.

Esta parte del código realiza la propagación hacia adelante, calculando las salidas de cada capa de la red neuronal utilizando las entradas, pesos y sesgos, y aplicando la función de activación sigmoide.

```
79      //Cálculo del error
80      error = Y - A2;
```

Esta línea calcula el error entre las etiquetas reales **Y** y las predicciones de la red **A2**. El error se utiliza para ajustar los pesos durante la retropropagación.

```
82      //Retropropagación
83      dZ2 = error .* sigmoid_derivada(Z2);
84      dW2 = A1' * dZ2;
85      db2 = sum(dZ2, 1);
86
87      dZ1 = (dZ2 * W2') .* sigmoid_derivada(Z1);
88      dW1 = X' * dZ1;
89      db1 = sum(dZ1, 1);
```

Cálculo del gradiente de la capa de salida:

Cálculo del gradiente de la capa oculta

```
91      %Actualizar pesos y sesgos
92      W2 = W2 + tasa_aprendizaje * dW2;
93      b2 = b2 + tasa_aprendizaje * db2;
94      W1 = W1 + tasa_aprendizaje * dW1;
95      b1 = b1 + tasa_aprendizaje * db1;
96  end
```

Los pesos W2 y W1, y los sesgos b2 y b1 se actualizan utilizando los gradientes calculados y la tasa de aprendizaje.
Esto ajusta la red para reducir el error en las siguientes iteraciones.

```
98  //Probar la red
99  b1_exp=repmat(b1,n_num_dat_ent,1)
100 b2_exp=repmat(b2,n_num_dat_ent,1)
101 Y_pred = sigmoid(sigmoid(X * W1 + b1_exp) * W2 + b2_exp);
102 disp("Predicciones:");
103 disp(cat(2,X,fix(Y_pred+0.5)));
104 //disp(Y_pred);
```

Se expande **b1** y **b2** nuevamente para que coincidan con el número de datos de entrada.

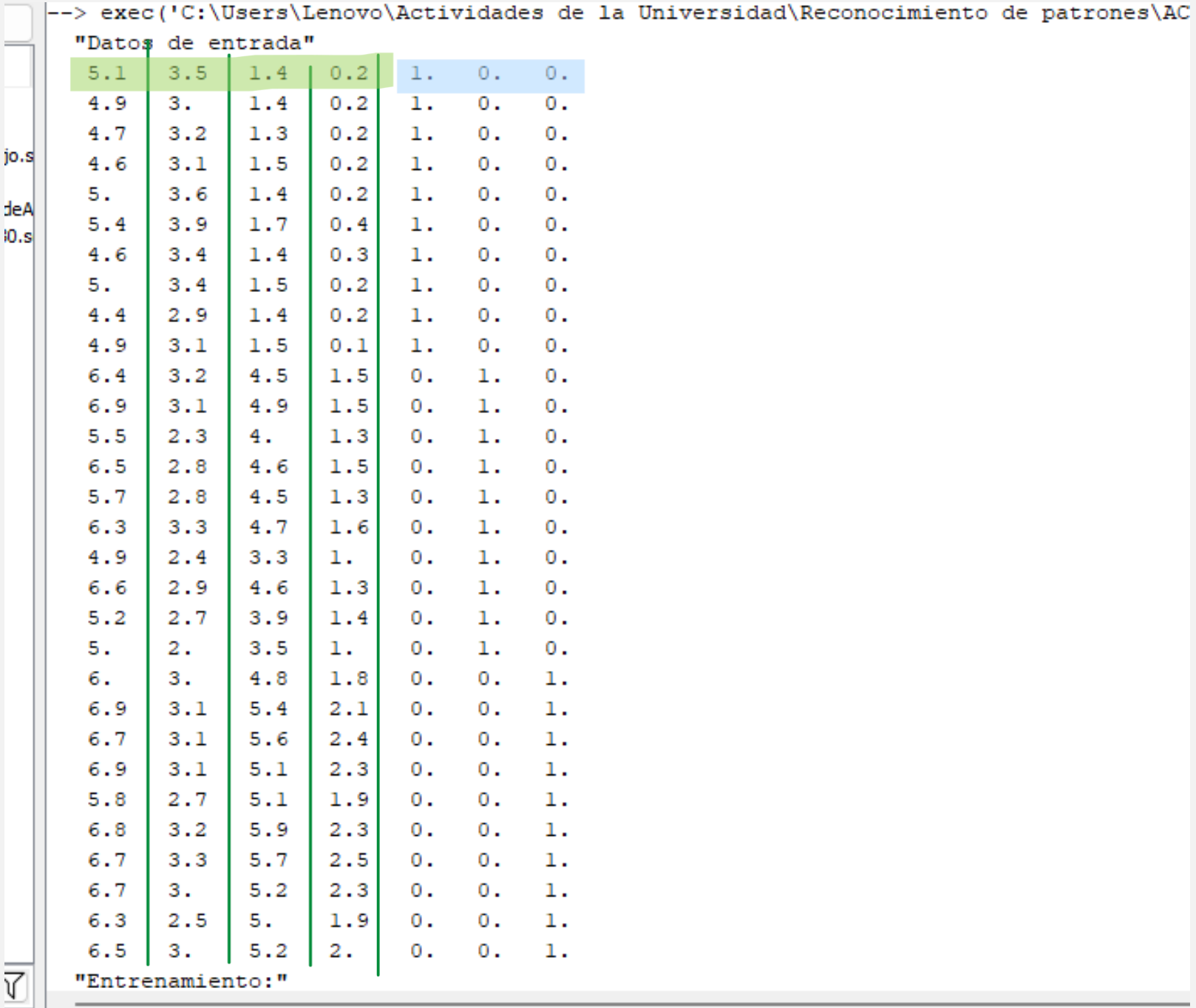
Luego, se realiza la propagación hacia adelante para obtener las predicciones **Y_pred**. Las predicciones se muestran en la consola junto con los datos de entrada, redondeadas a los valores más cercanos (0 o 1) para facilitar la interpretación.

Resultados en la consola

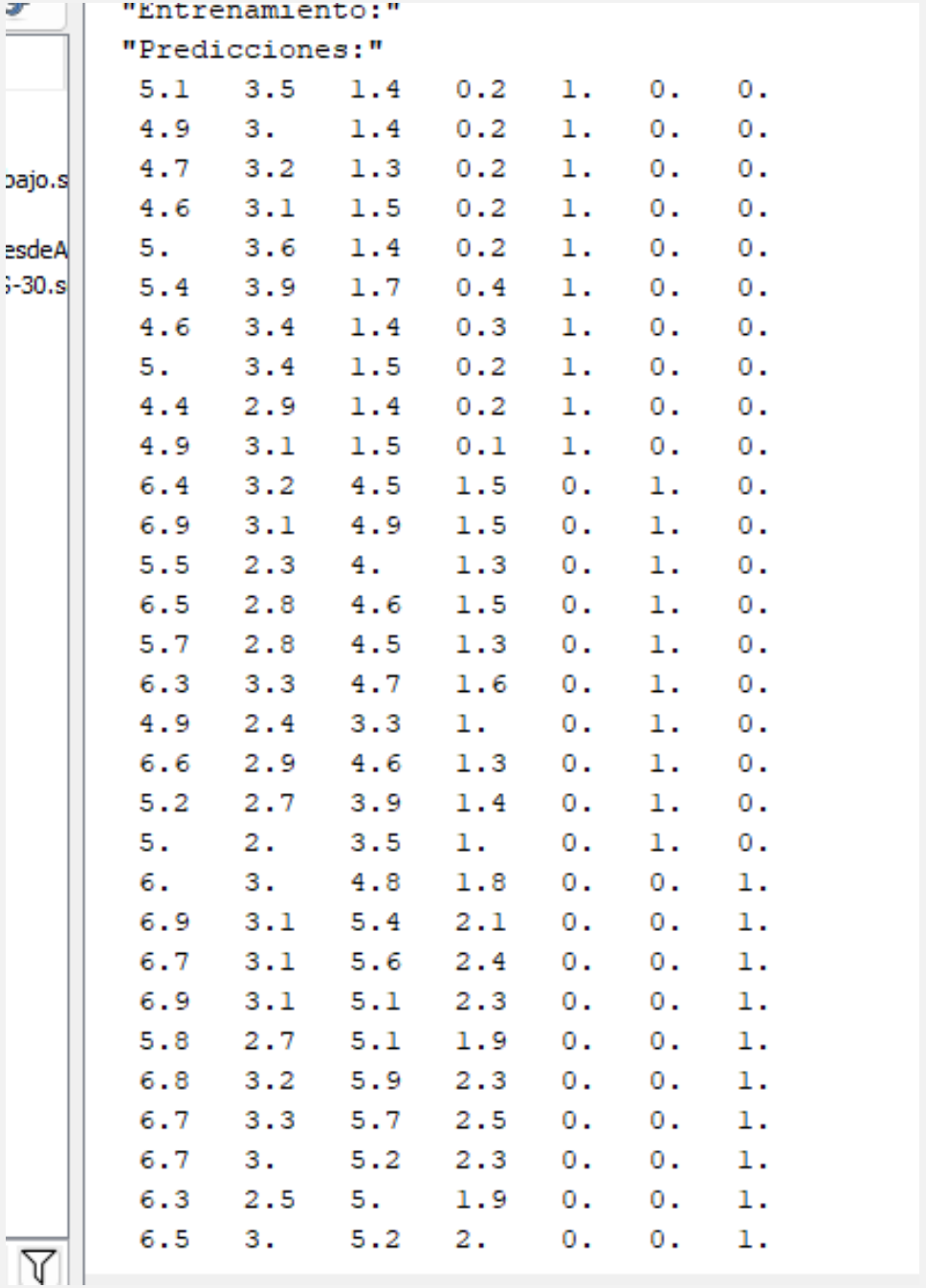
Muestra los Datos de entrada

Columnas 1 a 4: Estas columnas contienen las características de la flor (longitud y ancho del sépalo, longitud y ancho del pétalo).

Columnas 5 a 7: Estas columnas contienen las predicciones de la red neuronal en formato one-hot. Cada columna representa una clase (Setosa, Versicolor, Virginica).



Predicciones: todo coincide correctamente. Las características de las flores en los datos de entrada y las predicciones.



"Entrenamiento:"
"Predicciones:"

5.1	3.5	1.4	0.2	1.	0.	0.
4.9	3.	1.4	0.2	1.	0.	0.
4.7	3.2	1.3	0.2	1.	0.	0.
4.6	3.1	1.5	0.2	1.	0.	0.
5.	3.6	1.4	0.2	1.	0.	0.
5.4	3.9	1.7	0.4	1.	0.	0.
4.6	3.4	1.4	0.3	1.	0.	0.
5.	3.4	1.5	0.2	1.	0.	0.
4.4	2.9	1.4	0.2	1.	0.	0.
4.9	3.1	1.5	0.1	1.	0.	0.
6.4	3.2	4.5	1.5	0.	1.	0.
6.9	3.1	4.9	1.5	0.	1.	0.
5.5	2.3	4.	1.3	0.	1.	0.
6.5	2.8	4.6	1.5	0.	1.	0.
5.7	2.8	4.5	1.3	0.	1.	0.
6.3	3.3	4.7	1.6	0.	1.	0.
4.9	2.4	3.3	1.	0.	1.	0.
6.6	2.9	4.6	1.3	0.	1.	0.
5.2	2.7	3.9	1.4	0.	1.	0.
5.	2.	3.5	1.	0.	1.	0.
6.	3.	4.8	1.8	0.	0.	1.
6.9	3.1	5.4	2.1	0.	0.	1.
6.7	3.1	5.6	2.4	0.	0.	1.
6.9	3.1	5.1	2.3	0.	0.	1.
5.8	2.7	5.1	1.9	0.	0.	1.
6.8	3.2	5.9	2.3	0.	0.	1.
6.7	3.3	5.7	2.5	0.	0.	1.
6.7	3.	5.2	2.3	0.	0.	1.
6.3	2.5	5.	1.9	0.	0.	1.
6.5	3.	5.2	2.	0.	0.	1.