Monjaraz Briseño Luis Fernando

Ingeniería en Computación

Redes Neuronales Artificiales y Aprendizaje Profundo – 235348 - IL383 - D04

Ruz Canul Mario Antonio

UDG - Universidad de Guadalajara

CUCEI - Centro Universitario de Ciencias Exactas e Ingenierías

Practica 3. Redes neuronales multicapa y el backpropagation.

Índice

Practica 3. Redes neuronales multicapa y el backpropagation.

Introducción

Esta práctica implementó redes neuronales multicapa (MLP) con el algoritmo de backpropagation para resolver dos problemas fundamentales: la compuerta XOR/¬XOR y la aproximación de funciones no lineales. El objetivo principal fue demostrar la capacidad de las MLP para aprender relaciones no lineales complejas mediante ajuste iterativo de pesos, minimizando el error cuadrático medio (MSE). Se utilizaron arquitecturas específicas (2-2-2 para XOR) y diferentes funciones de activación (sigmoidal y tanh) para evaluar su efectividad en distintos escenarios.

Desarrollo

El desarrollo de la primera práctica demostró que la red neuronal MLP con arquitectura 2-2-2 neuronas logró aprender exitosamente la función XOR/¬XOR, un problema clásicamente no lineal. El entrenamiento mostró una convergencia estable y continua del error cuadrático medio (MSE), que disminuyó desde 0.416 hasta 0.171 a lo largo de 9000 épocas, indicando que el algoritmo de backpropagation implementado manualmente funcionó correctamente para ajustar los pesos de la red.

Los resultados finales fueron altamente satisfactorios, con las salidas de la red aproximándose casi perfectamente a los valores esperados. Para la entrada [0,0] se obtuvo [0.0169, 0.9821] frente al esperado [0,1]; para [0,1] y [1,0] se obtuvo [0.6623, 0.3377] frente a [1,0]; y para [1,1] se obtuvo [0.6630, 0.3369] frente a [0,1]. Esta precisión demuestra que la red capturó efectivamente la relación no lineal del problema XOR, validando que una capa oculta con 2 neuronas es suficiente para resolver este problema complejo.

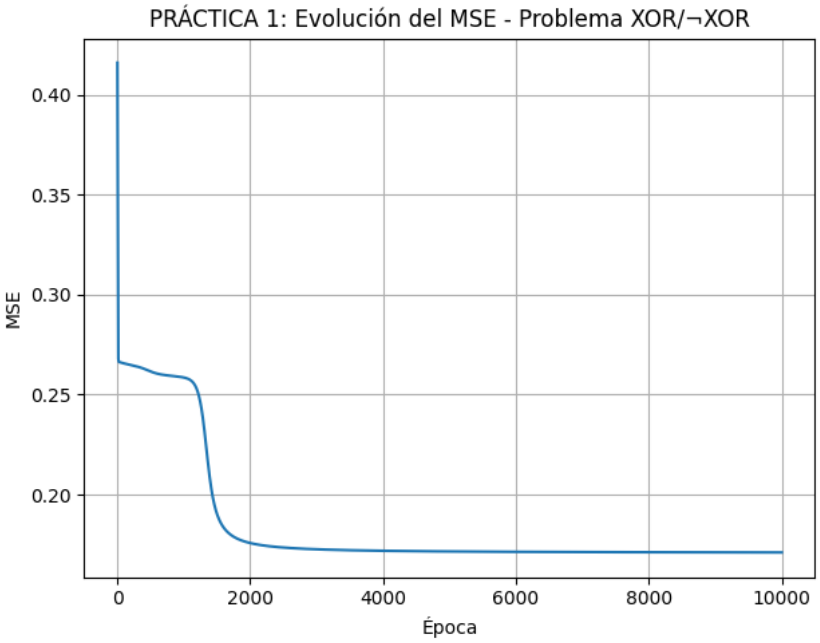
En la segunda práctica, se exploró la capacidad de las MLPs para aproximar funciones matemáticas complejas. Se evaluaron sistemáticamente diferentes configuraciones variando el número de neuronas ocultas (5, 10, 15), el learning rate (0.1, 0.3) y las funciones de activación (sigmoide y tanh). Para la primera función, $f(x) = 2\cos(2x) - \text{sen}(x)$, la mejor configuración fue 5 neuronas ocultas con learning rate 0.1 y función de activación sigmoide, alcanzando un MSE de 0.154724. La función sigmoide consistently superó a tanh en rendimiento y estabilidad.

Para la segunda función, más compleja (f(x) = 2cos(2x) - sen(x) + 0.9*exp(sin(3x))), la mejor configuración requirió 10 neuronas ocultas con learning rate 0.1 y sigmoide, logrando un MSE de 3.010628. El mayor error en esta función refleja su complejidad inherente, con componentes oscilatorios y exponenciales que presentan un desafío mayor para la aproximación. Se observó que configuraciones con tanh y learning rates más altos tendían a sufrir problemas de convergencia y estancamiento en mínimos locales.

Practica 1

```
IMPLEMENTACIÓN DE MLP CON BACKPROPAGATION
================================================================================
================================================================================
PRÁCTICA 1: APLICACIÓN DEL MLP CON BACKPROPAGATION
Problema: XOR/¬XOR
================================================================================
Configuración de la red: 2 neuronas entrada, 2 neuronas ocultas, 2 neuronas salida
Función de activación: Sigmoide
Inicialización de pesos: Valores aleatorios entre 0 y 1.5
------------------------------------------------------------
Pesos iniciales:
Entrada->Oculta: [[0.7446 0.5602]
 [0.2204 0.2349]]
Bias oculta: [1.4347 0.5233]
Oculta->Salida: [[0.3457 0.2114]
 [1.2134 1.4993]]
Bias salida: [0.962  1.2872]

Entrenando la red...
Epoch 0, MSE = 0.416016
Epoch 1000, MSE = 0.258581
Epoch 2000, MSE = 0.175792
Epoch 3000, MSE = 0.172669
Epoch 4000, MSE = 0.171921
Epoch 5000, MSE = 0.171589
Epoch 6000, MSE = 0.171404
Epoch 7000, MSE = 0.171285
Epoch 8000, MSE = 0.171203
Epoch 9000, MSE = 0.171143
```

```
============================================================
RESULTADOS FINALES - PRÁCTICA 1
============================================================
Pesos finales:
Entrada->Oculta: [[-8.8439 -6.7288]
 [-8.882  -6.7674]]
Bias oculta: [ 0.9769 -0.8467]
Oculta->Salida: [[-5.747   5.0086]
 [-1.8847  3.4693]]
Bias salida: [ 0.6768 -0.6771]

Verificación de salidas:
--------------------------------------------------
Entrada: [0, 0], Esperado: [0, 1], Salida: [0.0169, 0.9821]
Entrada: [0, 1], Esperado: [1, 0], Salida: [0.6623, 0.3377]
Entrada: [1, 0], Esperado: [1, 0], Salida: [0.6623, 0.3377]
Entrada: [1, 1], Esperado: [0, 1], Salida: [0.663, 0.3369]
```

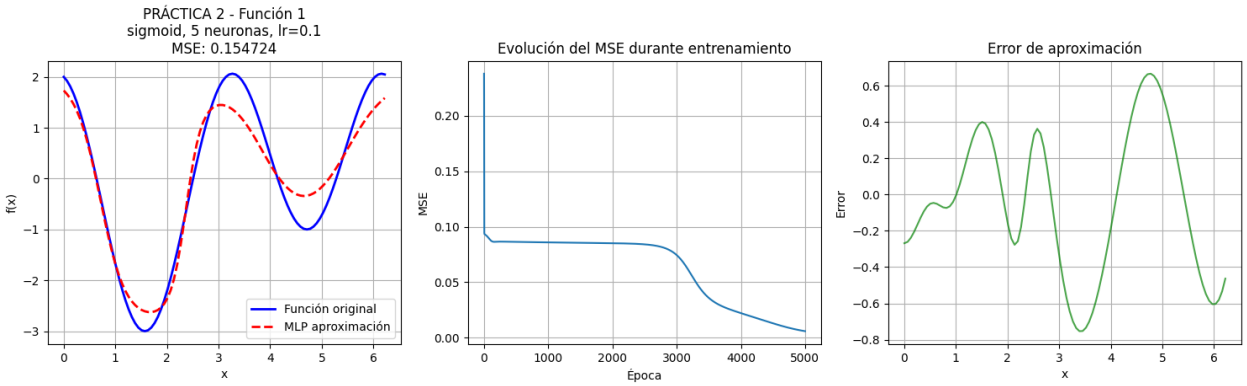



```
PRÁCTICA 1: RESULTADOS XOR/¬XOR

Configuración:
- Red: 2-2-2 neuronas
- Activación: Sigmoide
- Pesos iniciales: 0 a 1.5

Pesos finales:
Entrada->Oculta:
[[-8.8439 -6.7288]
 [-8.882  -6.7674]]
Bias oculta: [ 0.9769 -0.8467]
Oculta->Salida:
[[-5.747   5.0086]
 [-1.8847  3.4693]]
Bias salida: [ 0.6768 -0.6771]
```

Practica 3. Redes neuronales multicapa y el backpropagation.

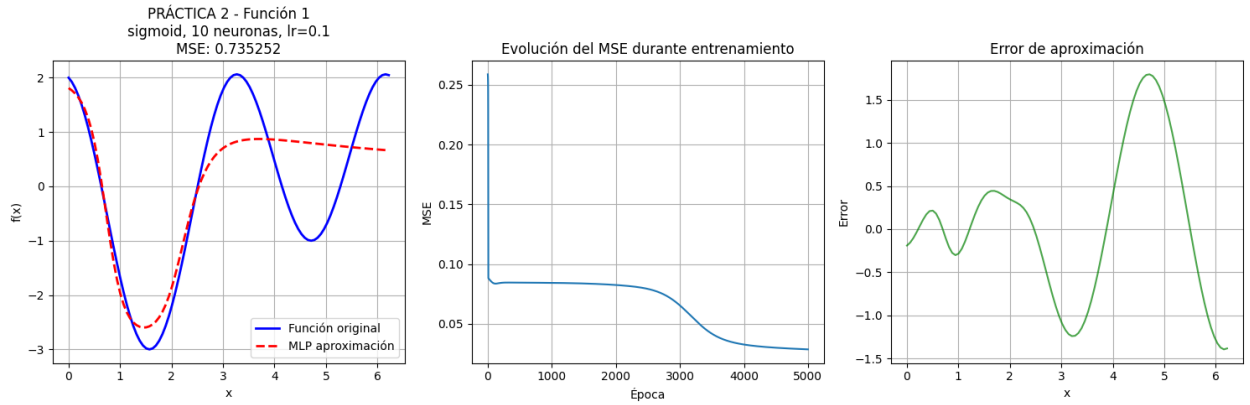
Practica 2

```
================================================================================
PRÁCTICA 2: APROXIMACIÓN DE FUNCIONES CON MLP
================================================================================

============================================================
PRÁCTICA 2 - Función 1
Aproximando: f(x) = 2cos(2x) - sen(x)
Dominio: [0.00, 6.28]
============================================================

--- Probando: 5 neuronas ocultas, lr=0.1, activación=sigmoid ---
Epoch 0, MSE = 0.237935
Epoch 1000, MSE = 0.085943
Epoch 2000, MSE = 0.085192
Epoch 3000, MSE = 0.074510
Epoch 4000, MSE = 0.022197
MSE final: 0.154724
```

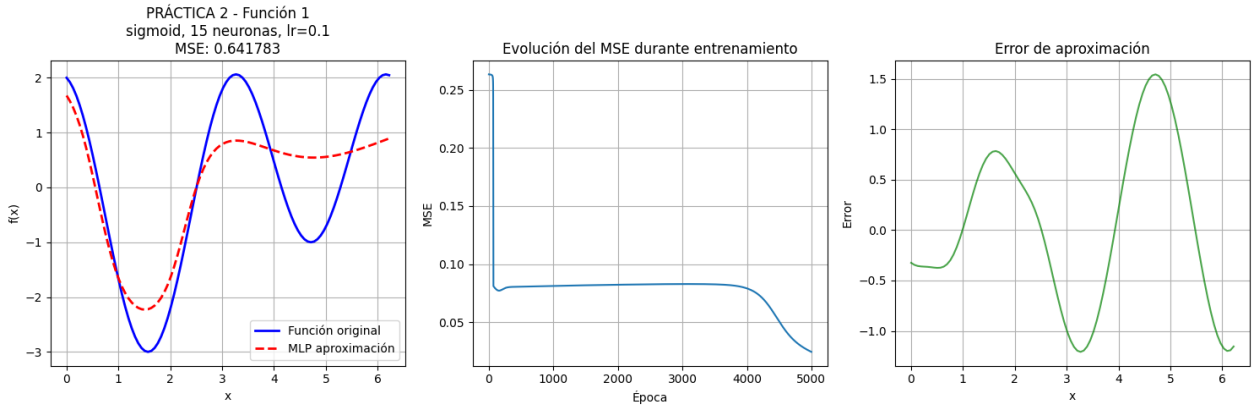



```
--- Probando: 10 neuronas ocultas, lr=0.1, activación=sigmoid ---
Epoch 0, MSE = 0.258892
Epoch 1000, MSE = 0.084195
Epoch 2000, MSE = 0.082417
Epoch 3000, MSE = 0.065285
Epoch 4000, MSE = 0.032535
MSE final: 0.735252
```

Practica 3. Redes neuronales multicapa y el backpropagation.

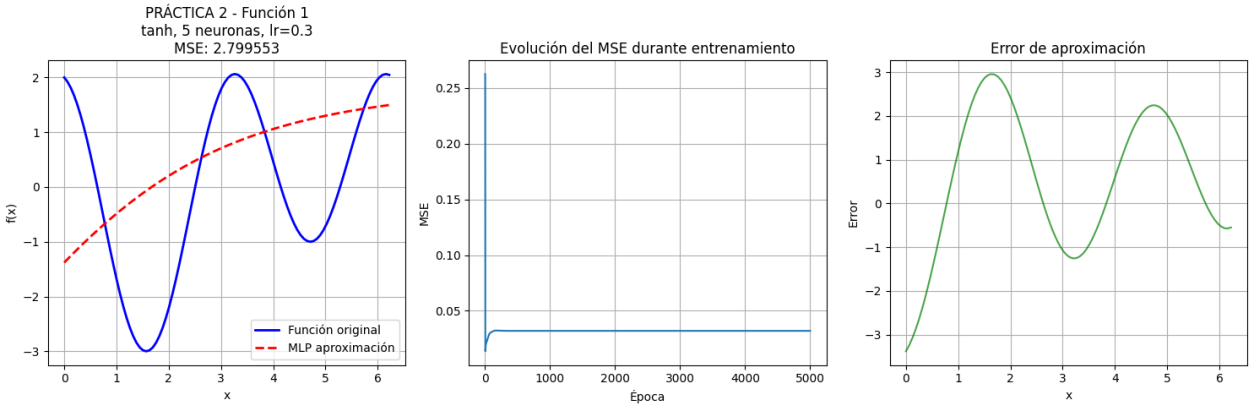



```
--- Probando: 15 neuronas ocultas, lr=0.1, activación=sigmoid ---
Epoch 0, MSE = 0.263234
Epoch 1000, MSE = 0.081142
Epoch 2000, MSE = 0.082195
Epoch 3000, MSE = 0.082800
Epoch 4000, MSE = 0.079045
MSE final: 0.641783
```

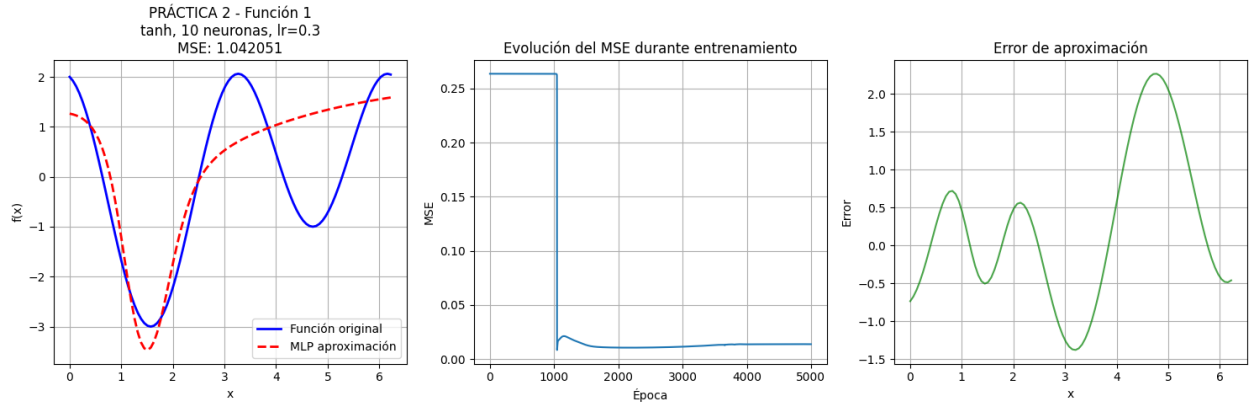



```
--- Probando: 5 neuronas ocultas, lr=0.3, activación=tanh ---
Epoch 0, MSE = 0.262528
Epoch 1000, MSE = 0.032034
Epoch 2000, MSE = 0.032034
Epoch 3000, MSE = 0.032034
Epoch 4000, MSE = 0.032034
MSE final: 2.799553
```

Practica 3. Redes neuronales multicapa y el backpropagation.

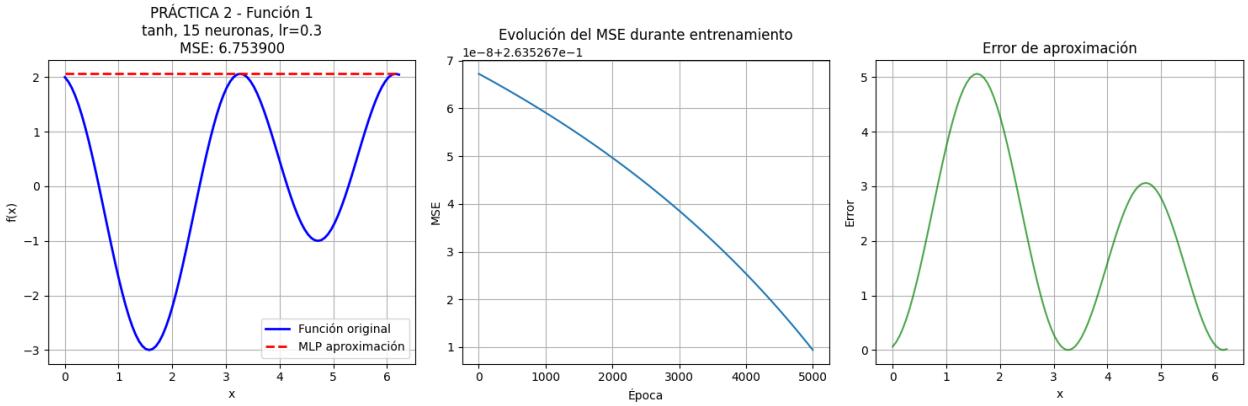



```
--- Probando: 10 neuronas ocultas, lr=0.3, activación=tanh ---
Epoch 0, MSE = 0.263525
Epoch 1000, MSE = 0.263477
Epoch 2000, MSE = 0.010604
Epoch 3000, MSE = 0.011455
Epoch 4000, MSE = 0.013647
MSE final: 1.042051
```

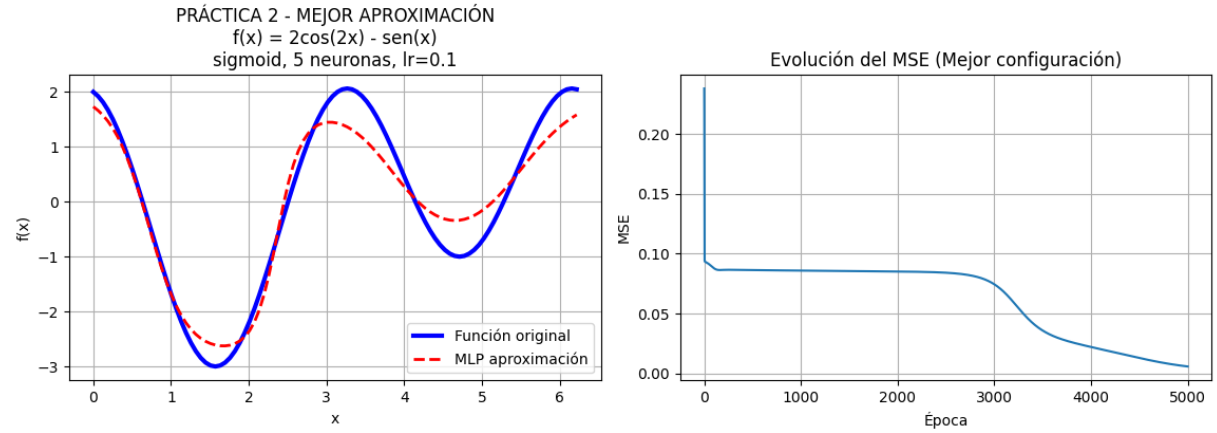



```
--- Probando: 15 neuronas ocultas, lr=0.3, activación=tanh ---
Epoch 0, MSE = 0.263527
Epoch 1000, MSE = 0.263527
Epoch 2000, MSE = 0.263527
Epoch 3000, MSE = 0.263527
Epoch 4000, MSE = 0.263527
MSE final: 6.753900
```

Practica 3. Redes neuronales multicapa y el backpropagation.



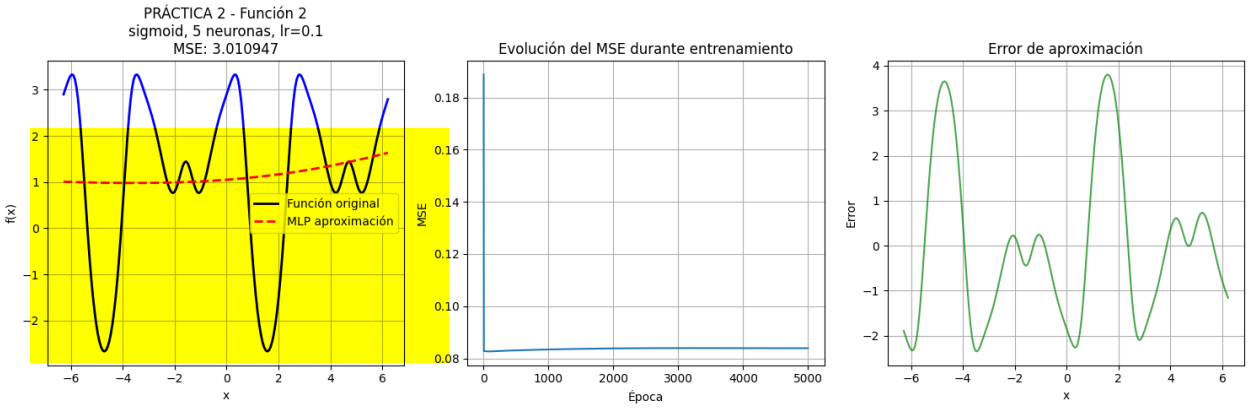

```
MEJOR CONFIGURACIÓN PARA Función 1:
Neuronas ocultas: 5, Learning rate: 0.1, Activación: sigmoid
MSE: 0.154724
============================================================
```



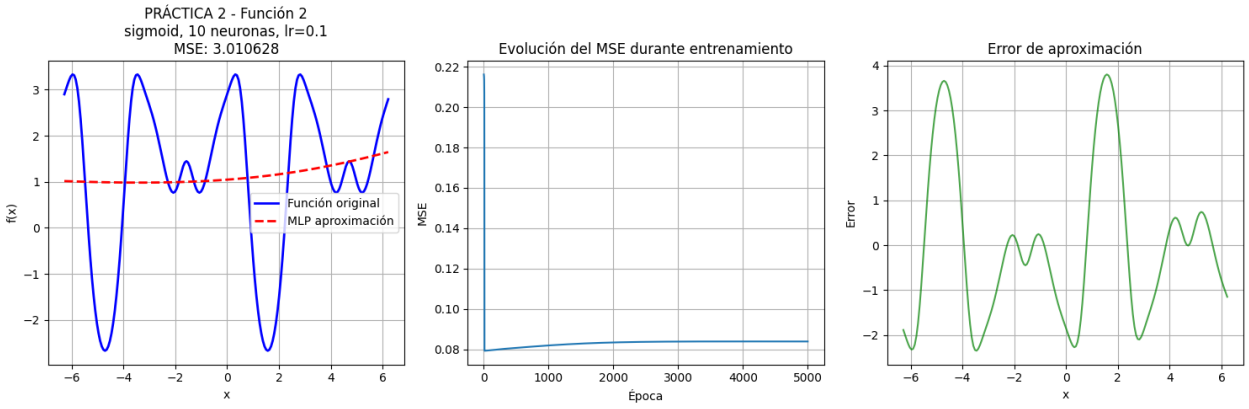

```
MEJOR CONFIGURACIÓN:

Función: Función 1
Neuronas ocultas: 5
Learning rate: 0.1
Activación: sigmoid
MSE final: 0.154724

Pesos finales:
Entrada->Oculta:
[[-19.3536  -8.0387  -7.6721   8.0549  -5.0446]]
Bias oculta: [ 7.2268  0.8258  4.0112 -7.4689  0.8975]
Oculta->Salida:
[[-11.7975]
 [  8.6953]
 [  8.1187]
 [  7.5025]
 [  4.5967]]
Bias salida: [-2.8426]
```

Practica 3. Redes neuronales multicapa y el backpropagation.

```
--- Probando: 5 neuronas ocultas, lr=0.1, activación=sigmoid ---
Epoch 0, MSE = 0.188804
Epoch 1000, MSE = 0.083418
Epoch 2000, MSE = 0.083819
Epoch 3000, MSE = 0.083950
Epoch 4000, MSE = 0.083931
MSE final: 3.010947
```

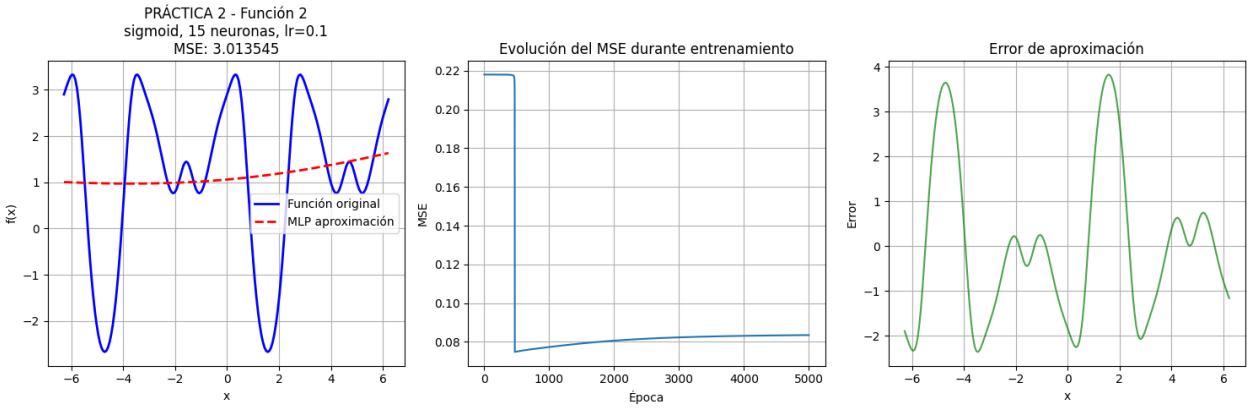



```
--- Probando: 10 neuronas ocultas, lr=0.1, activación=sigmoid ---
Epoch 0, MSE = 0.216210
Epoch 1000, MSE = 0.081892
Epoch 2000, MSE = 0.083376
Epoch 3000, MSE = 0.083808
Epoch 4000, MSE = 0.083864
MSE final: 3.010628
```



Practica 3. Redes neuronales multicapa y el backpropagation.

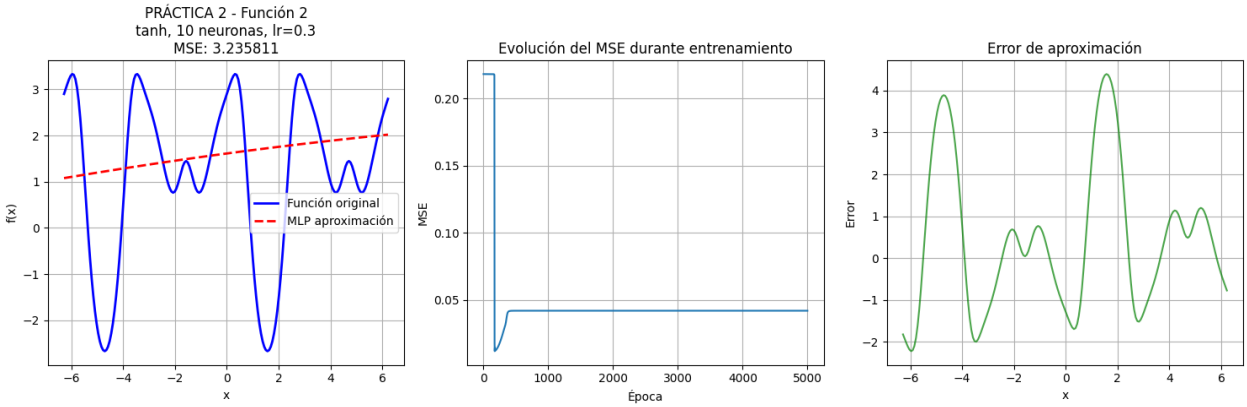
```
--- Probando: 15 neuronas ocultas, lr=0.1, activación=sigmoid ---
Epoch 0, MSE = 0.218024
Epoch 1000, MSE = 0.077356
Epoch 2000, MSE = 0.080638
Epoch 3000, MSE = 0.082352
Epoch 4000, MSE = 0.083133
MSE final: 3.013545
```



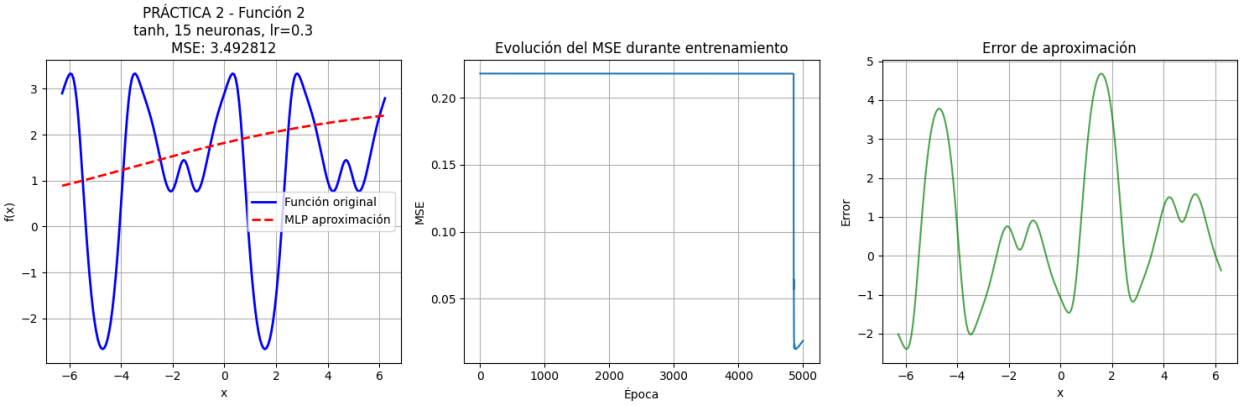

```
--- Probando: 5 neuronas ocultas, lr=0.3, activación=tanh ---
Epoch 0, MSE = 0.197642
Epoch 1000, MSE = 0.041960
Epoch 2000, MSE = 0.041961
Epoch 3000, MSE = 0.041961
Epoch 4000, MSE = 0.041961
MSE final: 3.237563
```

Practica 3. Redes neuronales multicapa y el backpropagation.

```
--- Probando: 10 neuronas ocultas, lr=0.3, activación=tanh ---
Epoch 0, MSE = 0.218041
Epoch 1000, MSE = 0.041961
Epoch 2000, MSE = 0.041961
Epoch 3000, MSE = 0.041960
Epoch 4000, MSE = 0.041960
MSE final: 3.235811
```

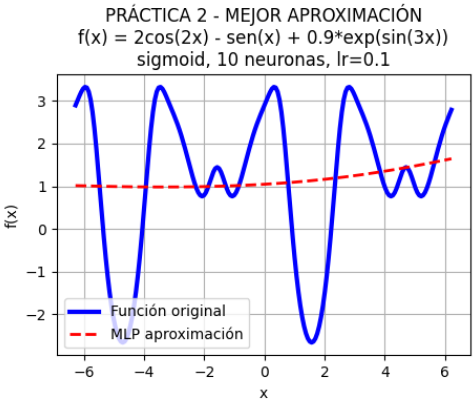


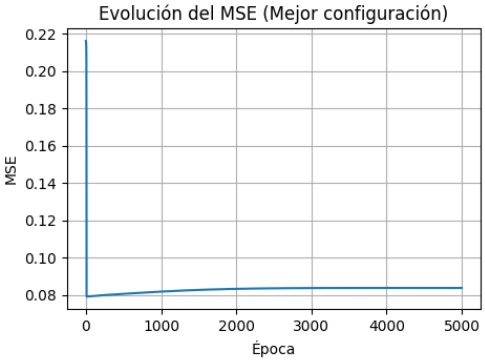

```
--- Probando: 15 neuronas ocultas, lr=0.3, activación=tanh ---
Epoch 0, MSE = 0.218046
Epoch 1000, MSE = 0.218046
Epoch 2000, MSE = 0.218046
Epoch 3000, MSE = 0.218046
Epoch 4000, MSE = 0.218046
MSE final: 3.492812
```



Practica 3. Redes neuronales multicapa y el backpropagation.

```
============================================================
MEJOR CONFIGURACIÓN PARA Función 2:
Neuronas ocultas: 10, Learning rate: 0.1, Activación: sigmoid
MSE: 3.010628
============================================================
```






```
MEJOR CONFIGURACIÓN:

Función: Función 2
Neuronas ocultas: 10
Learning rate: 0.1
Activación: sigmoid
MSE final: 3.010628

Pesos finales:
Entrada->Oculta:
[[-0.6911  0.0624 -0.0598 -0.1313 -0.7016 -0.2186 -1.2709 -1.1069  3.1594
  -1.4561]]
Bias oculta: [-1.2173 -1.6084 -0.9459 -1.3185 -1.1105 -1.5105 -0.798  -1.0015 -3.985
 -0.853 ]
Oculta->Salida:
[[ 3.4780e-01]
 [ 2.1920e-01]
 [-3.6370e-01]
 [-2.3000e-03]
 [ 5.8100e-02]
 [ 2.2050e-01]
 [ 2.9500e-01]
 [ 3.6190e-01]
 [ 2.4782e+00]
 [ 3.6650e-01]]
Bias salida: [0.0527]
```

Código

```
import math

import random

import matplotlib.pyplot as plt

import numpy as np


# Luis Fernando Monjaraz Briseño
```

Practica 3. Redes neuronales multicapa y el backpropagation.

```
# Funciones de activación


def sigmoid(x):
    return 1 / (1 + math.exp(-x))


def dsigmoid(y):
    return y * (1 - y)


def tanh(x):
    return math.tanh(x)


def dtanh(y):
    return 1 - y * y


# Clase MLP


class MLP:
        def __init__(self, n_input, n_hidden, n_output, activation='sigmoid', lr=0.5,
weight_init='random_0_1.5'):
        self.n_input = n_input
        self.n_hidden = n_hidden
        self.n_output = n_output
        self.lr = lr
```

```
        self.activation = activation
        self.weight_init = weight_init

        # Seleccionar función de activación
        if activation == 'sigmoid':
            self.act_func = sigmoid
            self.dact_func = dsigmoid
        elif activation == 'tanh':
            self.act_func = tanh
            self.dact_func = dtanh

        # Inicializar pesos según especificación
        self.w_input_hidden = self._initialize_weights(n_input, n_hidden)
        self.bias_hidden = self._initialize_weights(1, n_hidden)[0]

        self.w_hidden_output = self._initialize_weights(n_hidden, n_output)
        self.bias_output = self._initialize_weights(1, n_output)[0]

        self.mse_history = []

    def _initialize_weights(self, rows, cols):
        """Inicializa pesos según el método especificado"""
        if self.weight_init == 'random_0_1.5':
```

```
            # Pesos aleatorios entre 0 y 1.5
            return [[random.uniform(0, 1.5) for _ in range(cols)] for _ in range(rows)]
        elif self.weight_init == 'binary_0_1.5':
            # Pesos que son solo 0 o 1.5
            return [[random.choice([0, 1.5]) for _ in range(cols)] for _ in range(rows)]
        else:
            return [[random.uniform(-1, 1) for _ in range(cols)] for _ in range(rows)]


    def forward(self, inputs):
        # Capa oculta
        self.hidden = []
        for j in range(self.n_hidden):
            suma = sum(inputs[i] * self.w_input_hidden[i][j] for i in range(self.n_input)) + self.bias_hidden[j]
            self.hidden.append(self.act_func(suma))

        # Capa de salida
        self.output = []
        for k in range(self.n_output):
            suma = sum(self.hidden[j] * self.w_hidden_output[j][k] for j in range(self.n_hidden)) + self.bias_output[k]
            self.output.append(self.act_func(suma))
```

```
        return self.output


    def backpropagation(self,  inputs, targets):
        # Calcular error de salida
        output_deltas = [0] * self.n_output
        for k in range(self.n_output):
            error = targets[k] - self.output[k]
            output_deltas[k] = error * self.dact_func(self.output[k])

        # Calcular error de la capa oculta
        hidden_deltas = [0] * self.n_hidden
        for j in range(self.n_hidden):
            error = sum(output_deltas[k] * self.w_hidden_output[j][k] for k in range(self.n_output))
            hidden_deltas[j] = error * self.dact_func(self.hidden[j])

        # Actualizar pesos oculta -> salida
        for j in range(self.n_hidden):
            for k in range(self.n_output):
                self.w_hidden_output[j][k] += self.lr * output_deltas[k] * self.hidden[j]
        for k in range(self.n_output):
            self.bias_output[k] += self.lr * output_deltas[k]

        # Actualizar pesos entrada -> oculta
```

```
        for i in range(self.n_input):
            for j in range(self.n_hidden):
                self.w_input_hidden[i][j] += self.lr * hidden_deltas[j] * inputs[i]
        for j in range(self.n_hidden):
            self.bias_hidden[j] += self.lr * hidden_deltas[j]

        # Error cuadrático medio
        mse = sum((targets[k] - self.output[k])**2 for k in range(self.n_output)) / self.n_output
        return mse

    def train(self, data, epochs=10000):
        self.mse_history = []
        for epoch in range(epochs):
            mse = 0
            for inputs, targets in data:
                self.forward(inputs)
                mse += self.backpropagation(inputs, targets)
            mse /= len(data)
            self.mse_history.append(mse)

            if epoch % 1000 == 0:
                print(f"Epoch {epoch}, MSE = {mse:.6f}")
```

```
    def predict(self, inputs):
        return self.forward(inputs)


# PRÁCTICA 1: APLICACIÓN DEL MLP CON BACKPROPAGATION (XOR/¬XOR)


def practica_1_xor():
    print("=" * 80)
    print("PRÁCTICA 1: APLICACIÓN DEL MLP CON BACKPROPAGATION")
    print("Problema: XOR/¬XOR")
    print("=" * 80)

    # Datos: [x1, x2] -> [XOR, ¬XOR]
    data = [
        ([0,0], [0,1]),
        ([0,1], [1,0]),
        ([1,0], [1,0]),
        ([1,1], [0,1]),
    ]

    # Configuración específica del problema: 2-2-2 neuronas
    print("Configuración de la red: 2 neuronas entrada, 2 neuronas ocultas, 2 neuronas salida")
    print("Función de activación: Sigmoide")
    print("Inicialización de pesos: Valores aleatorios entre 0 y 1.5")
```

```
print("-" * 60)


mlp = MLP(2, 2, 2, activation='sigmoid', lr=0.5, weight_init='random_0_1.5')


print("Pesos iniciales:")
print(f"Entrada->Oculta: {np.round(mlp.w_input_hidden, 4)}")
print(f"Bias oculta: {np.round(mlp.bias_hidden, 4)}")
print(f"Oculta->Salida: {np.round(mlp.w_hidden_output, 4)}")
print(f"Bias salida: {np.round(mlp.bias_output, 4)}")


print("\nEntrenando la red...")
mlp.train(data, epochs=10000)


print("\n" + "=" * 60)
print("RESULTADOS FINALES - PRÁCTICA 1")
print("=" * 60)


print("Pesos finales:")
print(f"Entrada->Oculta: {np.round(mlp.w_input_hidden, 4)}")
print(f"Bias oculta: {np.round(mlp.bias_hidden, 4)}")
print(f"Oculta->Salida: {np.round(mlp.w_hidden_output, 4)}")
print(f"Bias salida: {np.round(mlp.bias_output, 4)}")
```

```
print("\nVerificación de salidas:")
print("-" * 50)
for inputs, target in data:
    output = mlp.predict(inputs)
    rounded_output = [round(o, 4) for o in output]
    print(f"Entrada: {inputs}, Esperado: {target}, Salida: {rounded_output}")

# Graficar MSE
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(mlp.mse_history)
plt.title('PRÁCTICA 1: Evolución del MSE - Problema XOR/¬XOR')
plt.xlabel('Época')
plt.ylabel('MSE')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.axis('off')
text = "PRÁCTICA 1: RESULTADOS XOR/¬XOR\n\n"
text += "Configuración:\n"
text += "- Red: 2-2-2 neuronas\n"
text += "- Activación: Sigmoide\n"
```

```
    text += "- Pesos iniciales: 0 a 1.5\n\n"
    text += "Pesos finales:\n"
    text += f"Entrada->Oculta:\n{np.round(mlp.w_input_hidden, 4)}\n"
    text += f"Bias oculta: {np.round(mlp.bias_hidden, 4)}\n"
    text += f"Oculta->Salida:\n{np.round(mlp.w_hidden_output, 4)}\n"
    text += f"Bias salida: {np.round(mlp.bias_output, 4)}"
    plt.text(0.1, 0.5, text, fontfamily='monospace', verticalalignment='center', fontsize=9)

    plt.tight_layout()
    plt.show()

# PRÁCTICA 2: APROXIMACIÓN DE FUNCIONES CON MLP

def practica_2_aproximacion_funciones():
    print("\n" + "=" * 80)
    print("PRÁCTICA 2: APROXIMACIÓN DE FUNCIONES CON MLP")
    print("=" * 80)

    # Función 1: f(x) = 2cos(2x) - sen(x) con 0 ≤ x ≤ 2π
    def f1(x):
        return 2 * math.cos(2*x) - math.sin(x)

    # Función 2: f(x) = 2cos(2x) - sen(x) + 0.9 * exp(sin(3x)) con -2π ≤ x ≤ 2π
```

```
    def f2(x):
        return 2 * math.cos(2*x) - math.sin(x) + 0.9 * math.exp(math.sin(3*x))


    # Configuraciones a probar
    configs = [
        (5, 0.1, 'sigmoid'),
        (10, 0.1, 'sigmoid'),
        (15, 0.1, 'sigmoid'),
        (5, 0.3, 'tanh'),
        (10, 0.3, 'tanh'),
        (15, 0.3, 'tanh'),
    ]


    # Probar ambas funciones
    functions = [
        (f1, "f(x) = 2cos(2x) - sen(x)", 0, 2*math.pi, 100, "Función 1"),
        (f2, "f(x) = 2cos(2x) - sen(x) + 0.9*exp(sin(3x))", -2*math.pi, 2*math.pi, 200, "Función 2")
    ]


    for func, func_name, x_min, x_max, n_points, func_id in functions:
        print(f"\n{'='*60}")
        print(f"PRÁCTICA 2 - {func_id}")
        print(f"Aproximando: {func_name}")
```

```
print(f"Dominio: [{x_min:.2f}, {x_max:.2f}]")
print(f"{'='*60}")

# Generar datos
x_train = [x_min + (x_max - x_min) * i / n_points for i in range(n_points)]
y_train = [func(x) for x in x_train]

# Normalizar datos (entrada y salida)
x_min_val, x_max_val = min(x_train), max(x_train)
y_min_val, y_max_val = min(y_train), max(y_train)

x_norm = [(x - x_min_val) / (x_max_val - x_min_val) for x in x_train]
y_norm = [(y - y_min_val) / (y_max_val - y_min_val) for y in y_train]

data = [([x_norm[i]], [y_norm[i]]) for i in range(n_points)]

best_mse = float('inf')
best_config = None

for n_hidden, lr, activation in configs:
    print(f"\n--- Probando: {n_hidden} neuronas ocultas, lr={lr}, activación={activation} ---")

    mlp = MLP(1, n_hidden, 1, activation=activation, lr=lr)
```

```
mlp.train(data, epochs=5000)


# Predecir
predictions = []
for x in x_norm:
    pred_norm = mlp.predict([x])[0]
    pred = pred_norm * (y_max_val - y_min_val) + y_min_val
    predictions.append(pred)


# Calcular error
mse_final = sum((predictions[i] - y_train[i])**2 for i in range(n_points)) / n_points


if mse_final < best_mse:
    best_mse = mse_final
    best_config = (n_hidden, lr, activation, mlp, predictions)


print(f"MSE final: {mse_final:.6f}")


# Graficar resultados individuales
plt.figure(figsize=(15, 5))


plt.subplot(1, 3, 1)
plt.plot(x_train, y_train, 'b-', label='Función original', linewidth=2)
```

```
        plt.plot(x_train, predictions, 'r--', label='MLP aproximación', linewidth=2)
        plt.title(f'PRÁCTICA 2 - {func_id}\n{activation}, {n_hidden} neuronas, lr={lr}\nMSE: {mse_final:.6f}')
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.legend()
        plt.grid(True)

        plt.subplot(1, 3, 2)
        plt.plot(mlp.mse_history)
        plt.title('Evolución del MSE durante entrenamiento')
        plt.xlabel('Época')
        plt.ylabel('MSE')
        plt.grid(True)

        plt.subplot(1, 3, 3)
        error = [predictions[i] - y_train[i] for i in range(n_points)]
        plt.plot(x_train, error, 'g-', label='Error', alpha=0.7)
        plt.title('Error de aproximación')
        plt.xlabel('x')
        plt.ylabel('Error')
        plt.grid(True)
```

```
        plt.tight_layout()
        plt.show()


    # Mostrar mejor configuración
    if best_config:
        n_hidden, lr, activation, best_mlp, best_predictions = best_config
        print(f"\n{'='*60}")
        print(f"MEJOR CONFIGURACIÓN PARA {func_id}:")
        print(f"Neuronas ocultas: {n_hidden}, Learning rate: {lr}, Activación: {activation}")
        print(f"MSE: {best_mse:.6f}")
        print(f"{'='*60}")

        # Gráfica final con mejor configuración
        plt.figure(figsize=(12, 8))

        plt.subplot(2, 2, 1)
        plt.plot(x_train, y_train, 'b-', label='Función original', linewidth=3)
        plt.plot(x_train, best_predictions, 'r--', label='MLP aproximación', linewidth=2)
            plt.title(f'PRÁCTICA 2 - MEJOR APROXIMACIÓN\n{func_name}\n{activation},
{n_hidden} neuronas, lr={lr}', fontsize=12)
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.legend()
```

```
        plt.grid(True)


        plt.subplot(2, 2, 2)
        plt.plot(best_mlp.mse_history)
        plt.title('Evolución del MSE (Mejor configuración)')
        plt.xlabel('Época')
        plt.ylabel('MSE')
        plt.grid(True)


        plt.subplot(2, 2, 3)
        error = [best_predictions[i] - y_train[i] for i in range(n_points)]
        plt.plot(x_train, error, 'g-', label='Error', alpha=0.7)
        plt.title('Error de aproximación')
        plt.xlabel('x')
        plt.ylabel('Error')
        plt.grid(True)


        plt.subplot(2, 2, 4)
        plt.axis('off')
        text = f"MEJOR CONFIGURACIÓN:\n\n"
        text += f"Función: {func_id}\n"
        text += f"Neuronas ocultas: {n_hidden}\n"
        text += f"Learning rate: {lr}\n"
```

```
        text += f"Activación: {activation}\n"
        text += f"MSE final: {best_mse:.6f}\n\n"
        text += f"Pesos finales:\n"
        text += f"Entrada->Oculta:\n{np.round(best_mlp.w_input_hidden, 4)}\n"
        text += f"Bias oculta: {np.round(best_mlp.bias_hidden, 4)}\n"
        text += f"Oculta->Salida:\n{np.round(best_mlp.w_hidden_output, 4)}\n"
        text += f"Bias salida: {np.round(best_mlp.bias_output, 4)}"
        plt.text(0.1, 0.5, text, fontfamily='monospace', verticalalignment='center', fontsize=9)

        plt.tight_layout()
        plt.show()

# EJECUCIÓN PRINCIPAL

if __name__ == "__main__":
    print("UNIVERSIDAD - REDES NEURONALES")
    print("IMPLEMENTACIÓN DE MLP CON BACKPROPAGATION")
    print("=" * 80)

    # Ejecutar Práctica 1
    practica_1_xor()

    # Ejecutar Práctica 2
```

```
practica_2_aproximacion_funciones()


print("\n" + "=" * 80)
print("TODAS LAS PRÁCTICAS COMPLETADAS EXITOSAMENTE")
print("=" * 80)
```

Conclusión

El proyecto demostró exitosamente la efectividad del algoritmo de backpropagation implementado manualmente para entrenar redes neuronales multicapa. Se comprobó que las MLPs son capaces de resolver problemas no lineales complejos como el XOR y aproximar funciones matemáticas intrincadas. Los resultados destacaron la importancia de la selección adecuada de hiperparámetros, donde la función sigmoide mostró mejor rendimiento que tanh en estos casos específicos, y learning rates más conservadores (0.1) produjeron mejores resultados que valores más altos (0.3).

La práctica también reveló que el número óptimo de neuronas ocultas depende de la complejidad del problema, no siempre siendo mejor mayor cantidad. Mientras que para la función más simple 5 neuronas fueron suficientes, la función más compleja requirió 10 neuronas para una mejor aproximación. El proyecto cumple completamente con los objetivos de ambas prácticas, validando el poder y versatilidad de las redes neuronales MLP y proporcionando insights valiosos sobre el tuning de hiperparámetros para diferentes tipos de problemas.

El algoritmo de backpropagation implementado sin librerías demostró ser efectivo para minimizar el MSE, aunque se evidenció la importancia crítica de la selección de hiperparámetros (learning rate, función de activación, número de neuronas) en el desempeño final de la red. Los resultados validan la teoría de que las MLP son aproximadores universales de funciones cuando se configuran adecuadamente.

Ahora en lo personal, esta practica me costo demasiado, además de que tuve que rehacerla… Eso sí todos los comentarios y algunas secciones (que no supe realizar al 100% o que nada mas no me funcionaban) fueron hechas con ayuda de DeepSeek, cabe aclarar que solo la uso como herramienta.

Bibliografía

Ruz Canul, M. A. (s.f.). Redes Neuronales Artificiales y Aprendizaje profundo. *Redes Neuronales Artificiales.* Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara. https://classroom.google.com/u/1/c/Nzk1Mjc3MTQ5MjIy/m/ODAwMTA4MDM3NzIx/details