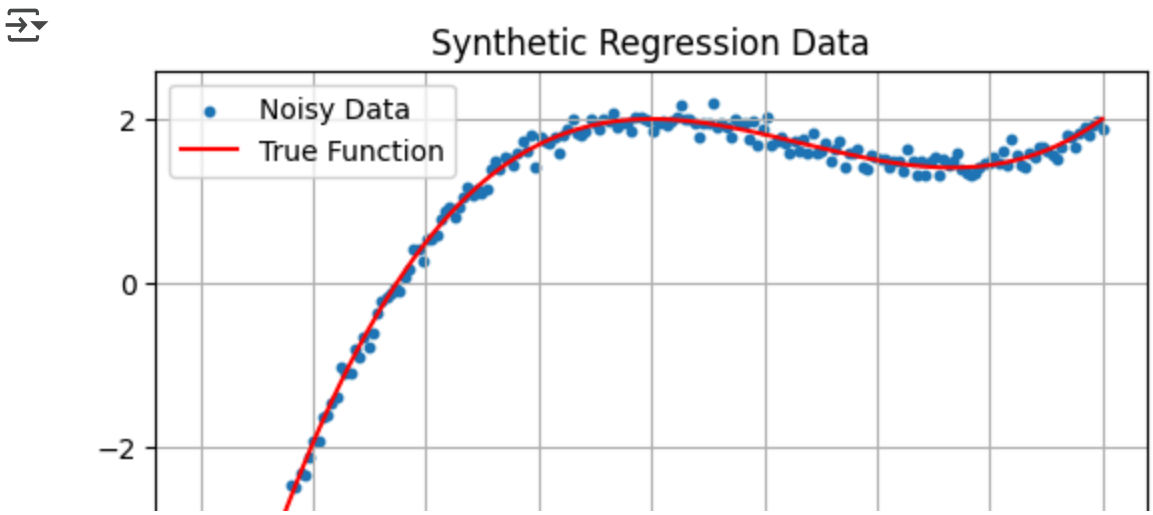
1. From Scratch Neural-Net Regression with NumPy Only

(i)Create Dataset

```
# dataset_generator.py (Optional file if needed)
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
X = np.linspace(-2, 2, 200).reshape(-1, 1)
y_true = 0.5 * X**3 - X**2 + 2
noise = np.random.normal(0, 0.1, size=y_true.shape)
y = y_true + noise

# Save dataset
np.savez("synthetic_dataset.npz", X=X, y=y)

# Optional: Plot
plt.scatter(X, y, s=10, label="Noisy Data")
plt.plot(X, y_true, color='red', label="True Function")
plt.title("Synthetic Regression Data")
plt.legend()
plt.grid()
plt.show()
```



(ii)model.py – Neural Network Components

```
# model.py
# The code from this cell has been moved to the cell below to fix the ModuleNotFoundError
```

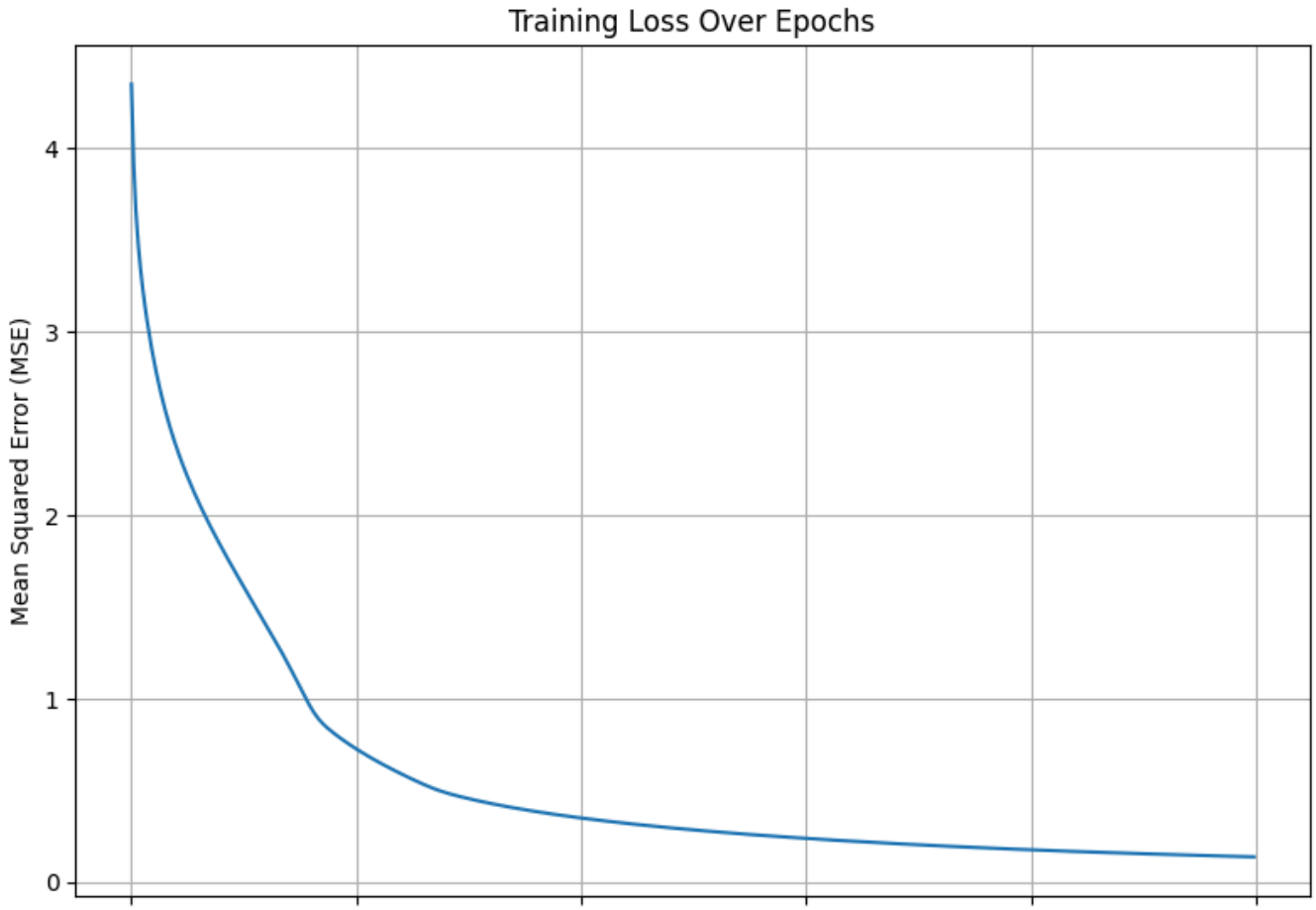
(iii)train.ipynb – Training and Visualization

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Activation Functions
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

# Loss Function
def mse(y_pred, y_true):
    return np.mean((y_pred - y_true) ** 2)

def mse_derivative(y_pred, y_true):
    return 2 * (y_pred - y_true) / y_true.size

# Neural Network Class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01):
        self.lr = learning_rate
        self.W1 = np.random.randn(input_size, hidden_size)
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size)
        self.b2 = np.zeros((1, output_size))

    def forward(self, X):
        self.Z1 = X @ self.W1 + self.b1
        self.A1 = relu(self.Z1)
        self.Z2 = self.A1 @ self.W2 + self.b2
        return self.Z2

    def backward(self, X, y, y_pred):
        dZ2 = mse_derivative(y_pred, y)
        dW2 = self.A1.T @ dZ2
        db2 = np.sum(dZ2, axis=0, keepdims=True)

        dA1 = dZ2 @ self.W2.T
        dZ1 = dA1 * relu_derivative(self.Z1)
        dW1 = X.T @ dZ1
        db1 = np.sum(dZ1, axis=0, keepdims=True)

        self.W2 -= self.lr * dW2
        self.b2 -= self.lr * db2
        self.W1 -= self.lr * dW1
        self.b1 -= self.lr * db1

# Load dataset with error handling
try:
    data = np.load("synthetic_dataset.npz")
    X, y = data["X"], data["y"]
except FileNotFoundError:
    print("Error: synthetic_dataset.npz not found.")
    exit(1)

# Ensure X and y have correct shapes
if X.ndim == 1:
    X = X.reshape(-1, 1)  # Ensure X is (n_samples, 1)
if y.ndim == 1:
    y = y.reshape(-1, 1)  # Ensure y is (n_samples, 1)

# Verify input size matches
if X.shape[1] != 1:
```

```
    print(f"Error: Expected input size 1, got {X.shape[1]}.")
    exit(1)

# Initialize network
model = NeuralNetwork(input_size=1, hidden_size=10, output_size=1, learning_rate=0.01)

# Training loop
epochs = 500
losses = []

for epoch in range(epochs):
    y_pred = model.forward(X)
    loss = mse(y_pred, y)
    losses.append(loss)
    model.backward(X, y, y_pred)

    if epoch % 50 == 0:
        print(f"Epoch {epoch}: Loss = {loss:.4f}")

# Plot loss
plt.figure(figsize=(8, 6))
plt.plot(losses)
plt.title("Training Loss Over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Mean Squared Error (MSE)")
plt.grid(True)
plt.tight_layout()
plt.show()

# Final prediction plot
y_pred = model.forward(X)

# Sort X and y_pred for smooth plotting
sort_idx = np.argsort(X[:, 0])
X_sorted = X[sort_idx]
y_pred_sorted = y_pred[sort_idx]

plt.figure(figsize=(8, 6))
plt.scatter(X, y, s=10, label="Actual Data", alpha=0.5)
plt.plot(X_sorted, y_pred_sorted, color='green', label="Model Prediction")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Model Prediction vs Actual Data")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Epoch 0: Loss = 4.3504
Epoch 50: Loss = 1.6098
Epoch 100: Loss = 0.7268
Epoch 150: Loss = 0.4557
Epoch 200: Loss = 0.3505
Epoch 250: Loss = 0.2855
Epoch 300: Loss = 0.2394
Epoch 350: Loss = 0.2044
Epoch 400: Loss = 0.1771
Epoch 450: Loss = 0.1558
```



(iv)README.md

```
import numpy as np
import matplotlib.pyplot as plt

# Activation Functions
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

# Loss Function
def mse(y_pred, y_true):
    return np.mean((y_pred - y_true) ** 2)

def mse_derivative(y_pred, y_true):
    return 2 * (y_pred - y_true) / y_true.size
```

```
# Neural Network Class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01):
        self.lr = learning_rate
        self.W1 = np.random.randn(input_size, hidden_size)
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size)
        self.b2 = np.zeros((1, output_size))

    def forward(self, X):
        self.Z1 = X @ self.W1 + self.b1
        self.A1 = relu(self.Z1)
        self.Z2 = self.A1 @ self.W2 + self.b2
        return self.Z2

    def backward(self, X, y, y_pred):
        dZ2 = mse_derivative(y_pred, y)
        dW2 = self.A1.T @ dZ2
        db2 = np.sum(dZ2, axis=0, keepdims=True)

        dA1 = dZ2 @ self.W2.T
        dZ1 = dA1 * relu_derivative(self.Z1)
        dW1 = X.T @ dZ1
        db1 = np.sum(dZ1, axis=0, keepdims=True)

        self.W2 -= self.lr * dW2
        self.b2 -= self.lr * db2
        self.W1 -= self.lr * dW1
        self.b1 -= self.lr * db1

# Load dataset with error handling
try:
    data = np.load("synthetic_dataset.npz")
    X, y = data["X"], data["y"]
except FileNotFoundError:
    print("Error: synthetic_dataset.npz not found.")
    exit(1)

# Ensure X and y have correct shapes
if X.ndim == 1:
    X = X.reshape(-1, 1)  # Ensure X is (n_samples, 1)
if y.ndim == 1:
    y = y.reshape(-1, 1)  # Ensure y is (n_samples, 1)

# Verify input size matches
if X.shape[1] != 1:
    print(f"Error: Expected input size 1, got {X.shape[1]}.")
    exit(1)

# Initialize network
model = NeuralNetwork(input_size=1, hidden_size=10, output_size=1, learning_rate=0.01)

# Training loop
epochs = 500
losses = []

for epoch in range(epochs):
    y_pred = model.forward(X)
    loss = mse(y_pred, y)
```

```
    losses.append(loss)
    model.backward(X, y, y_pred)

    if epoch % 50 == 0:
        print(f"Epoch {epoch}: Loss = {loss:.4f}")

# Plot loss
plt.figure(figsize=(8, 6))
plt.plot(losses)
plt.title("Training Loss Over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Mean Squared Error (MSE)")
plt.grid(True)
plt.tight_layout()
plt.show()

# Final prediction plot
y_pred = model.forward(X)

# Sort X and y_pred for smooth plotting
sort_idx = np.argsort(X[:, 0])
X_sorted = X[sort_idx]
y_pred_sorted = y_pred[sort_idx]

plt.figure(figsize=(8, 6))
plt.scatter(X, y, s=10, label="Actual Data", alpha=0.5)
plt.plot(X_sorted, y_pred_sorted, color='green', label="Model Prediction")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Model Prediction vs Actual Data")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Epoch 0: Loss = 5.4315
Epoch 50: Loss = 0.7522
Epoch 100: Loss = 0.4513
Epoch 150: Loss = 0.3448
Epoch 200: Loss = 0.2842
Epoch 250: Loss = 0.2411
Epoch 300: Loss = 0.2076
Epoch 350: Loss = 0.1813
Epoch 400: Loss = 0.1597
Epoch 450: Loss = 0.1418
```