scexp4

October 2, 2024

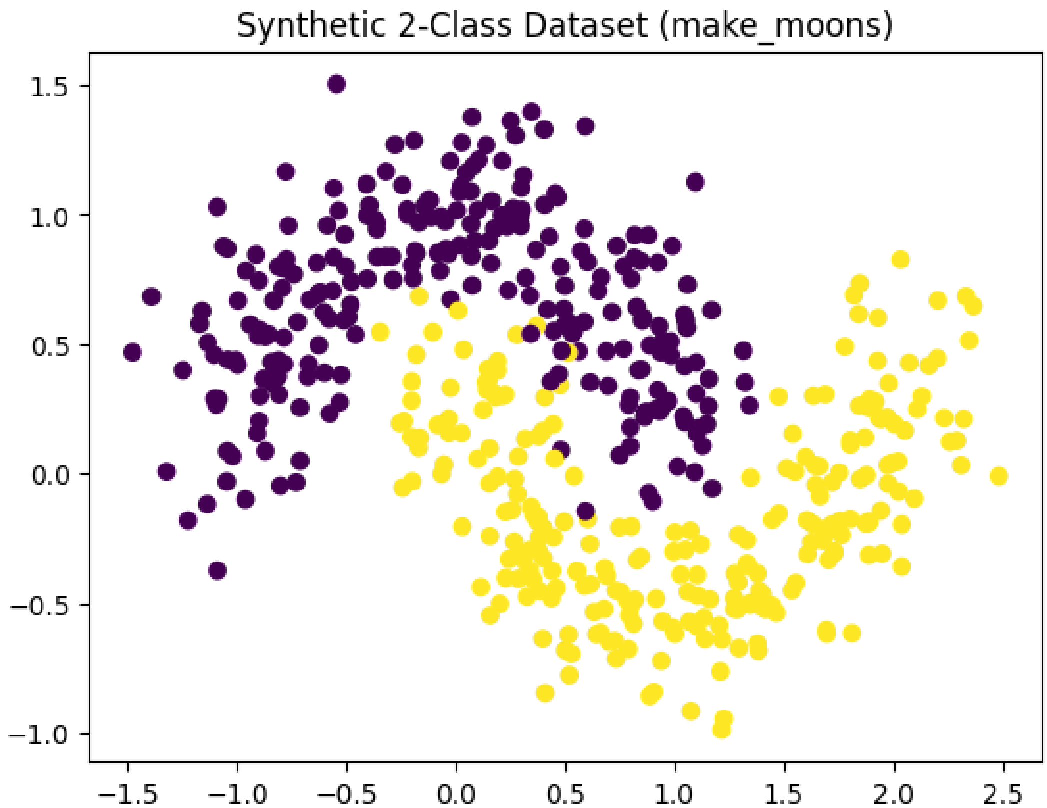
#Vedant Shah, UID: 2022700052, Batch: D, Class: CSE-DS

1 Importing necessary libraries

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

1.1 Synthetic 2-class dataset creation

[3]:
```python
# Step 2: Creating the synthetic 2-class dataset using make_moons (non-linear␣
↪boundary)
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)

# Reshaping the target variable to a column vector
y = y.reshape(-1, 1)

# Splitting the dataset into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Plotting the dataset to visualize the non-linear boundary
plt.scatter(X[:, 0], X[:, 1], c=y[:, 0], cmap='viridis')
plt.title("Synthetic 2-Class Dataset (make_moons)")
plt.show()
```



1.2 Sigmoid and derivative function creation

[6]:
```
# Step 3: Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
```

2 Neural network class definition for EBPTA implementation

[4]:
```
# Step 4: Defining the Neural Network class implementing EBPTA
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate):
        # Randomly initializing weights
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.learning_rate = learning_rate
```

```
    def forward(self, X):
        # Forward propagation
        self.hidden_input = np.dot(X, self.weights_input_hidden)
        self.hidden_output = sigmoid(self.hidden_input)
        self.final_input = np.dot(self.hidden_output, self.
↪weights_hidden_output)
        self.final_output = sigmoid(self.final_input)
        return self.final_output

    def backward(self, X, y, output):
        # Error calculation
        output_error = y - output
        output_delta = output_error * sigmoid_derivative(output)

        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error * sigmoid_derivative(self.hidden_output)

        # Update weights
        self.weights_hidden_output += self.hidden_output.T.dot(output_delta) *␣
↪self.learning_rate
        self.weights_input_hidden += X.T.dot(hidden_delta) * self.learning_rate

    def train(self, X, y, epochs):
        errors = []
        for epoch in range(epochs):
            output = self.forward(X)
            self.backward(X, y, output)
            error = np.mean(np.abs(y - output))
            errors.append(error)
            if epoch % 100 == 0:
                print(f'Epoch {epoch}, Error: {error}')
        return errors
```

3 Model Training

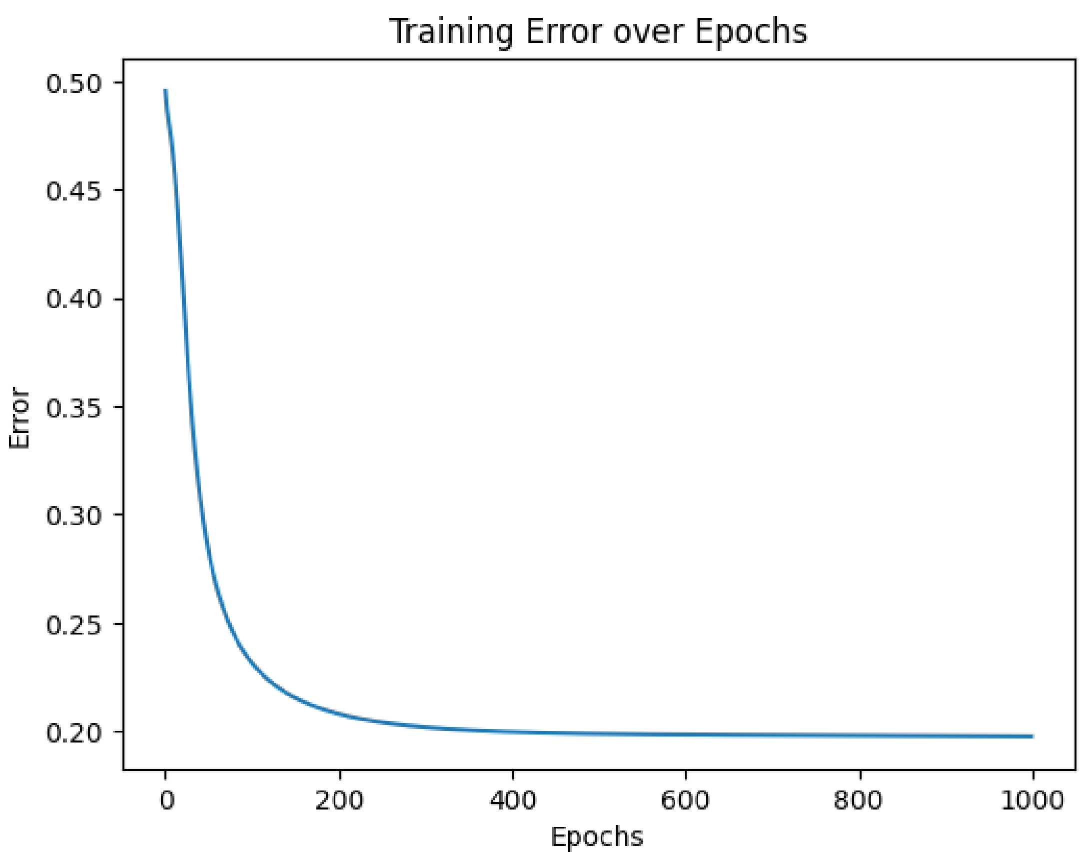
[7]:
```
# Step 5: Training the Neural Network model with one hidden layer
input_size = 2  # Two features from the dataset
hidden_size = 5  # Five neurons in the hidden layer
output_size = 1  # Binary classification (output 0 or 1)
learning_rate = 0.01  # Learning rate for weight updates
epochs = 1000  # Number of training iterations

# Initialize the model
nn_model = NeuralNetwork(input_size=input_size, hidden_size=hidden_size,␣
↪output_size=output_size, learning_rate=learning_rate)
```

```python
# Train the model and collect error history
train_errors = nn_model.train(X_train, y_train, epochs=epochs)
```

```
Epoch 0, Error: 0.49569977483087757
Epoch 100, Error: 0.2313891649661509
Epoch 200, Error: 0.20796847786739597
Epoch 300, Error: 0.20184245561913539
Epoch 400, Error: 0.19967855320995243
Epoch 500, Error: 0.19879841325258676
Epoch 600, Error: 0.19839029963938828
Epoch 700, Error: 0.1981591079290929
Epoch 800, Error: 0.1979832677996559
Epoch 900, Error: 0.1978030289640052
```

4 Error Plots

[8]:
```python
# Step 6: Plotting the error over epochs
plt.plot(train_errors)
plt.title("Training Error over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.show()
```



[9]:
```python
# Step 7: Testing the model on the test dataset
y_pred_train = nn_model.forward(X_train)
y_pred_test = nn_model.forward(X_test)

# Convert output probabilities to binary predictions (0 or 1)
y_pred_train = (y_pred_train > 0.5).astype(int)
y_pred_test = (y_pred_test > 0.5).astype(int)

# Calculate accuracy for training and testing sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f'Training Accuracy: {train_accuracy * 100:.2f}%')
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

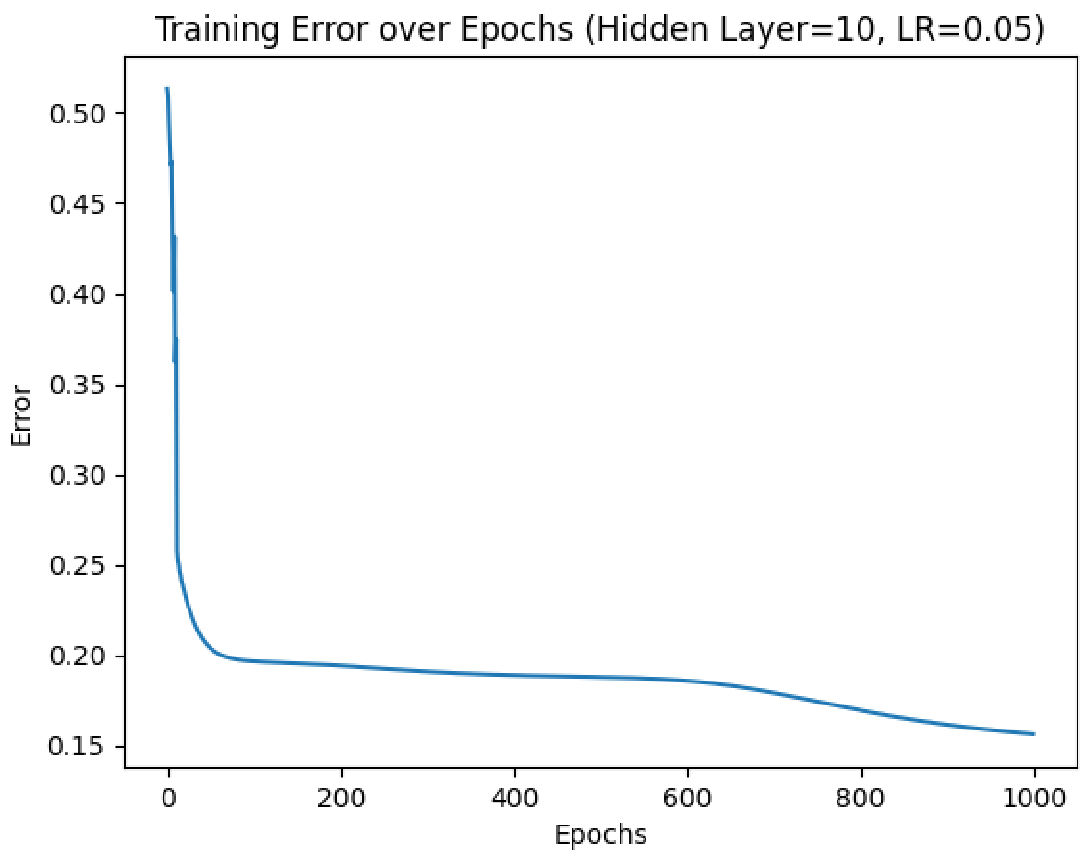
```
Training Accuracy: 85.75%
Test Accuracy: 86.00%
```

#Experimenting with varied parameters

[10]:
```python
# New configuration: More neurons in hidden layer and higher learning rate
nn_model2 = NeuralNetwork(input_size=2, hidden_size=10, output_size=1,␣
    ↪learning_rate=0.05)
train_errors2 = nn_model2.train(X_train, y_train, epochs=1000)

# Plot the error graph for this model
plt.plot(train_errors2)
plt.title("Training Error over Epochs (Hidden Layer=10, LR=0.05)")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.show()

# Evaluate the new model
y_pred_test2 = nn_model2.forward(X_test)
y_pred_test2 = (y_pred_test2 > 0.5).astype(int)
test_accuracy2 = accuracy_score(y_test, y_pred_test2)

print(f'Test Accuracy for Model 2: {test_accuracy2 * 100:.2f}%')
```

```
Epoch 0, Error: 0.5133481544709437
Epoch 100, Error: 0.19633986782068966
Epoch 200, Error: 0.19398322130636544
Epoch 300, Error: 0.19070832712089486
Epoch 400, Error: 0.18867764261423947
Epoch 500, Error: 0.1876512940691068
Epoch 600, Error: 0.18555335151413185
```

```
Epoch 700, Error: 0.17885367107773745
Epoch 800, Error: 0.1691403846376854
Epoch 900, Error: 0.16123480932016457
```




```
Test Accuracy for Model 2: 89.00%
```

5 Comparing Results

[11]:
```python
# Model 1 (5 hidden neurons, 0.01 learning rate)
print(f"Model 1: Hidden Neurons = 5, LR = 0.01, Test Accuracy = {test_accuracy␣
↪* 100:.2f}%")

# Model 2 (10 hidden neurons, 0.05 learning rate)
print(f"Model 2: Hidden Neurons = 10, LR = 0.05, Test Accuracy =␣
↪{test_accuracy2 * 100:.2f}%")

# Additional configurations can be added and compared here
```

```
Model 1: Hidden Neurons = 5, LR = 0.01, Test Accuracy = 86.00%
Model 2: Hidden Neurons = 10, LR = 0.05, Test Accuracy = 89.00%
```

6 Conclusion

In this experiment, we applied the Error Back-Propagation Training Algorithm (EBPTA) to a synthetic 2-class dataset with non-linear boundaries. By varying hyperparameters such as the number of hidden neurons, hidden layers, and learning rate, we observed changes in model performance. Increasing the number of hidden neurons and the learning rate led to faster convergence and improved accuracy. However, larger models also showed a higher risk of overfitting. Ultimately, balancing model complexity with adequate training time and parameter tuning was crucial in achieving optimal performance.