Introduction to Neural Networks

Practical Exercises

Introduction to Machine Learning 2024 - 2025

Introduction

This document contains practical exercises designed to reinforce your understanding of neural networks. Each exercise builds upon the theoretical concepts covered in the lectures, from basic perceptrons to multilayer networks.

1 Exercise 1: Single Neuron Implementation

1.1 Objective

Implement a single artificial neuron and understand its components: weights, bias, and activation function.

1.2 Dataset

Use the following synthetic binary classification data:

```
import numpy as np
import pandas as pd

# Generate synthetic binary classification data
np.random.seed(42)
n_samples = 100

# Generate two classes of points
class1 = np.random.normal(loc=[2, 2], scale=0.5, size=(n_samples
    //2, 2))
class2 = np.random.normal(loc=[-2, -2], scale=0.5, size=(n_samples
    //2, 2))

# Combine data
X = np.vstack([class1, class2])
y = np.hstack([np.ones(n_samples//2), np.zeros(n_samples//2)])
```

1.3 Tasks

1. **Neuron Implementation**

- Implement weight initialization
- Add bias term
- Implement sigmoid activation function
- Create forward propagation method

2. **Training Implementation**
   - Calculate prediction error
   - Implement gradient descent
   - Update weights and bias
   - Track learning progress
3. **Analysis**
   - Visualize decision boundary
   - Plot learning curve
   - Test with different learning rates

2 Exercise 2: Multilayer Perceptron

2.1 Objective

Build a complete multilayer perceptron for solving the XOR problem.

2.2 Dataset

```
# XOR problem dataset
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

# Create network architecture
input_size = 2
hidden_size = 4
output_size = 1
```

2.3 Tasks

1. **Network Architecture**
   - Implement multiple layer structure
   - Initialize weights for all layers
   - Add bias to each layer
   - Implement ReLU activation for hidden layer
2. **Forward Propagation**

- Calculate layer outputs
- Implement activation functions
- Store intermediate values
- Calculate final output

3. **Backpropagation**
   - Calculate output error
   - Implement chain rule
   - Update all weights
   - Implement learning rate scheduling

3 Exercise 3: Deep Neural Network

3.1 Objective

Implement a deep neural network for MNIST digit classification.

3.2 Dataset

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

# Load MNIST data
digits = load_digits()
X, y = digits.data, digits.target

# Normalize data
X = X / 16.0

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

3.3 Tasks

1. **Implementation**
   - Create multiple hidden layers
   - Implement dropout regularization
   - Add batch normalization
   - Implement mini-batch training

2. **Training**
   - Implement cross-entropy loss

- Add momentum to optimization
- Implement early stopping
- Add learning rate decay

3. **Evaluation**
    - Calculate accuracy metrics
    - Create confusion matrix
    - Visualize misclassified digits
    - Analyze network predictions