Assignment 3: FashionMNIST Classification

Overview

This assignment explores different neural network configurations for image classification using the FashionMNIST dataset. The goal is to experiment with various hyperparameters and architectures to achieve optimal performance.

Score: 90/100 ⭐

Objectives

- Implement neural networks for FashionMNIST classification
- Experiment with different network architectures
- Compare performance across various configurations
- Optimize model accuracy through hyperparameter tuning

Dataset

FashionMNIST: A dataset of 70,000 grayscale images (28x28 pixels) of 10 fashion categories.

Task Requirements

Step 1: Download Sample Code

Base your implementation on the FashionMNIST tutorial from PyTorch: PyTorch Optimization Tutorial

Step 2: Use SGD Optimizer

All experiments must use Stochastic Gradient Descent (SGD) as the optimizer.

Step 3: Experiment with Network Configurations

Test different configurations by varying:

1. Number of Hidden Layers

- Shallow networks (1-2 layers)
- Medium networks (3-4 layers)
- Deep networks (5+ layers)

2. Number of Neurons per Hidden Layer

- Small: 64, 128 neurons
- Medium: 256, 512 neurons

- Large: 1024+ neurons

3. Activation Functions

- ReLU
- LeakyReLU
- Tanh
- Sigmoid
- Other activation functions

4. Learning Rate Scheduler

- With scheduler (e.g., StepLR, ExponentialLR)
- Without scheduler (constant learning rate)

Step 4: Training Configuration

- **Epochs**: 20 epochs for each configuration
- **Comparison**: Compare performance across all configurations

Step 5: Visualization

Draw loss and accuracy curves to compare performance:

- Plot multiple configurations in one figure for easier comparison
- Include both training and test curves
- Label each configuration clearly

Step 6: Report Results

Show final accuracy for both training and test sets for each configuration:

- Create a summary table or list
- **Highlight your best performing configuration**

CUDA Support (Optional)

If using GPU acceleration:

```
python
```

```python
# Get device
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

# Move model to device
model = NeuralNetwork().to(device)

# Move data to device in training loop
for batch, (X, y) in enumerate(dataloader):
    X, y = X.to(device), y.to(device)
    pred = model(X)
    # ... rest of training code
```

Important: All tensors must be on the same device (CPU or CUDA).

Example Network Configurations to Try

1. **Baseline**: 512 → 512 → 10 (ReLU, no scheduler)
2. **Deep**: 512 → 256 → 128 → 10 (ReLU, StepLR)
3. **Wide**: 1024 → 512 → 10 (LeakyReLU, no scheduler)
4. **Alternative**: 256 → 256 → 256 → 10 (Tanh, ExponentialLR)

Deliverables

- Jupyter notebook containing:
  - Multiple network configurations
  - Training code for 20 epochs each
  - Loss and accuracy visualization curves
  - Final accuracy table/summary
  - Highlighted best result

Important:

- ✅ Include all result plots and tables in the notebook
- ❌ Do NOT compress the notebook file

Requirements

```
python
```

```
torch
torchvision
matplotlib
numpy
```

Installation

```bash
pip install torch torchvision matplotlib numpy
```

Evaluation Criteria

- Variety of configurations tested
- Clear visualization and comparison
- Proper documentation of results
- Best model performance

Results

Successfully experimented with multiple neural network architectures and hyperparameters. Identified optimal configuration through systematic comparison of training and test accuracies across 20 epochs.

Assignment completed as part of Deep Learning coursework