Assignment 4: Neural Networks

Due Date: November 17, 11:59 pm

A brief discussion of the assignment will take place in class on November 6. An online training session for PyTorch will be offered by TA Hazem Taha.

Overview

In this assignment, you are required to train a neural network to solve a multi-class classification problem using both your own neural network implementation and using **PyTorch**. You will use the **Fashion MNIST** dataset, which contains 28×28 grayscale images of articles of clothing and fashion accessories. Each image is assigned a label from ten possible classes.

Requirements

1. Dataset

Fashion MNIST Dataset:

There are two ways to download the dataset:

1. **Through Kaggle:** The dataset can be downloaded from `https://www.kaggle.com/datasets/zalando-research/fashionmnist?resource=download`. The dataset will be in CSV files with the following content:

   - `fashion-mnist_train.csv`: 60,000 training examples.
   - `fashion-mnist_test.csv`: 10,000 test examples.

   Each example is a 28×28 grayscale image, resulting in 784 pixels per image. Each pixel-value is an integer between 0 and 255, indicating the lightness or darkness of that pixel. The CSV files have 785 columns: the first column is the class label, and the remaining 784 columns are the pixel-values.

2. **Using `datasets.FashionMNIST`:** The data will be downloaded as tensors using the following code:

```python
from torchvision import datasets

train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True)
```

Data Split:

- The dataset is already split into a training set of 60,000 examples and a test set of 10,000 examples.
- You will further split the training data into a **training set** and a **validation set** (e.g., an 80/20 split).
- The validation set is needed for **early stopping** and for choosing your own hyper-parameters.

2. Neural Network Implementations

You are required to implement two neural networks:

1. **Custom Implementation:**
   - **Architecture:**
     - Input Layer: 784 units (raw pixel values).
     - Hidden Layers: At least **two hidden layers**.
       - Activation Function: Your choice (ReLU is recommended).
     - Output Layer: 10 units.
       - Activation Function: **Softmax**.
   - **Training:**
     - Loss Function: **Cross-entropy loss**.
     - Techniques:
       - **Mini-batch gradient descent**.
       - **Weight decay** (L2 regularization).
       - **Early stopping**.
2. **PyTorch Implementation:**
   - Replicate the same neural network architecture using **PyTorch**.
   - Utilize PyTorch functionalities for defining the model, loss function, optimizer, and training loop.

3. Model Training and Evaluation

- Train **two models** with different initializations using each of the two implementations discussed above (total of **four models**).
- Plot the **learning curves** (cross-entropy loss vs. number of epochs) for both training and validation sets.
- Record the **training and test misclassification errors** for each model.
- Organize the results in a clear table.
- Specify all hyperparameter values used and the activation function used at the hidden units.
- Include any other observations you find valuable.

Guidelines

1. Code Development

- **Modularity:** Write functions for each of the main tasks and for any repeated operations.
- **Vectorization:** Use vectorized operations with NumPy wherever possible for efficiency.
- **Comments and Documentation:** Include instructive comments explaining your code.
- **Libraries:**
  - **Allowed:**
    - **NumPy** for linear algebra and array manipulation.
    - **scikit-learn** or **PyTorch (for data manipulation tasks only, not for neural network implementation in your custom code)** (e.g., shuffling, splitting, standardization).
    - **Matplotlib** or similar libraries for plotting.
  - **Not Allowed in Custom Implementation:**
    - Using **PyTorch** or any high-level neural network libraries (e.g., Keras) for implementing the neural network in your custom implementation.

2. Randomization

- Whenever you use randomization in your code, use a number formed with the **last 4 digits of your student ID**, in any order, as the **seed** for the pseudo number generator.
- Ensure that the seed is set for all random number generators used (e.g., NumPy, PyTorch).

3. Report Writing

Your report should include:

- **Introduction:** Brief overview of the assignment objectives.
- **Methodology:** Description of neural network architectures, training processes, and hyperparameters.
- **Results:**
  - Learning curves for each model (training and validation loss over epochs).
  - Training and test misclassification errors.
  - A table summarizing the results for all models.

- **Discussion:** Analysis of results, impact of different initializations and hyperparameters, comparison between custom and PyTorch implementations.
- **Conclusion:** Summary of findings and suggestions for potential improvements if more time was available.

4. Demo Video

- **Duration:** Maximum of **1 minute**.
- **Content:** Scroll through your code and briefly explain what each part does.
- **Format:** Ensure the video is clear and audible. Acceptable formats include MP4, AVI, or MOV.

Deliverables

Submit the following by **November 17, 11:59 pm**:

1. **Report:** A PDF file containing your detailed report.
2. **Code Files:** Python files (`.py`) of your implementations.
   - Your code should include the implementations of the four neural networks as discussed (two custom implementations with different initializations and two PyTorch implementations).
   - Include any necessary instructions to run your code.
3. **Demo Video:** A video file showcasing your code with explanations.

Additional Notes

1. Hyperparameters

You may use the following suggested hyperparameters or search for better ones:

Hidden Layers	Hidden Units	Learning Rate	Batch Size	Weight Decay (λ)
2	156	0.01	128	0.0018738
3	92	0.01	128	0.00231
4	80	0.01	128	0.001232

[1]

[1] These values are based on Hazem Taha's implementation. The test misclassification rate obtained with the models trained with these values was below 14%.

2. Data Preprocessing

Below is a sample code reference for loading and preprocessing the data using `datasets.FashionMNIST`:

```
import numpy as np
import torch
from torchvision import datasets

# Seed the pseudo number generators (e.g., if your ID ends with 0393)
np.random.seed(3093)
torch.manual_seed(3093)

# Define the number of classes
num_classes = 10  # For Fashion MNIST

# Load Fashion MNIST dataset
train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True)

print(train_dataset.data.shape, test_dataset.data.shape)

# Prepare the data as numpy arrays
X_train = train_dataset.data.numpy().reshape(-1, 28 * 28).astype('float32') / 255.0
Y_train = train_dataset.targets.numpy()

X_test = test_dataset.data.numpy().reshape(-1, 28 * 28).astype('float32') / 255.0
Y_test = test_dataset.targets.numpy()

# Split the training set into train and validation sets (80% / 20%)
validation_size = int(0.2 * X_train.shape[0])
X_validation, Y_validation = X_train[:validation_size], Y_train[:validation_size]
X_train, Y_train = X_train[validation_size:], Y_train[validation_size:]

# Save original labels before one-hot encoding
Y_train_orig = Y_train
Y_validation_orig = Y_validation
Y_test_orig = Y_test

# Convert labels to one-hot encoding for multi-class classification
def one_hot_encode(labels, num_classes):
    return np.eye(num_classes)[labels]

```

```
Y_train = one_hot_encode(Y_train, num_classes)
Y_validation = one_hot_encode(Y_validation, num_classes)
Y_test = one_hot_encode(Y_test, num_classes)

# Standardizing the data

# Calculate the mean and standard deviation of the training features
X_train_mean = X_train.mean(axis=0)
X_train_std = X_train.std(axis=0)
X_train_std[X_train_std == 0] = 1  # To avoid division by zero

# Standardize all three subsets of data
X_train = (X_train - X_train_mean) / X_train_std
X_validation = (X_validation - X_train_mean) / X_train_std
X_test = (X_test - X_train_mean) / X_train_std
```

3. Performance Benchmark

- Models trained with the suggested hyperparameters achieved a test misclassification rate below **14%**.

4. Improvement Suggestions

- You may experiment with different hyperparameters, activation functions, or architectures to improve performance.
- Document any changes and their impact on the results.

5. Academic Integrity

- Ensure all submitted work is your own.
- Cite any external resources or code snippets used.
- Adhere to the university's policies on academic integrity.