CS 273P: Machine Learning and Data Mining

Homework 4

Due date: **See Canvas**

Instructor: Xiaohui Xie

This homework is graded automatically on **Gradescope** using an autograder. All questions are designed to be unit-testable: you will implement specific functions/classes that return specified outputs that satisfy the autograder. Unless explicitly required, do not print anything.

Allowed libraries: You may use `numpy`, `pandas`, and `PyTorch` (`torch`, `torchvision`). You may use `torch.nn` modules and `torch.optim`. You may use `torchvision.datasets` to load data. Do **not** use pre-trained models, `torchvision.models`, or external training frameworks.

Compute note: The autograder will run on CPU with limited time. Your code must be efficient and deterministic. We will train for only a small number of epochs and/or a limited number of minibatches for grading.

Submission rules (READ CAREFULLY):

1. Download the starter files from the GitHub link.
2. Complete your work by editing only the provided Python files: `problem1.py`, `problem2.py`, and `problem3.py`.
3. Create a plain text file called `statement.txt` containing your collaboration statement.
4. Submit to Gradescope by uploading a single **ZIP file** containing exactly:
   - `problem1.py`
   - `problem2.py`
   - `problem3.py`
   - `statement.txt`

   No folders/subdirectories; all files must be at the top level of the zip.
5. You may resubmit before the deadline; your **highest score** counts.

If you run into issues with the autograder or submission format, post on Ed Discussion so others can benefit.

Points: This homework adds up to a total of **110 points**, as follows:

Problem 1: Feedforward Neural Net (MLP) on CIFAR-10	35 points
Problem 2: Convolutional Neural Net (CNN) on CIFAR-10	45 points
Problem 3: Recurrent Neural Net (RNN) treating images as sequences	25 points
Statement of Collaboration	5 points

Dataset: CIFAR-10 (Shared)

We will use CIFAR-10 (50k train, 10k test, 32×32 RGB images, 10 classes). You will load CIFAR-10 using `torchvision.datasets.CIFAR10`. Unless otherwise specified:

- Normalize images to floating point in $[0, 1]$.
- Use the provided random seed argument to make your results deterministic.
- Use `CrossEntropyLoss` for all classification training in this homework.

To keep grading fast, the autograder will evaluate correctness on a small subset of data and a small training budget. Your code must handle arbitrary batch sizes.

Problem 1: Feedforward Neural Net (MLP) on CIFAR-10 (35 points)

In this problem, you will implement a multilayer perceptron (MLP) classifier in PyTorch and train it on CIFAR-10.

All required functions/classes for Problem 1 are in `problem1.py`.

1.1 Data utilities (10 points)

Implement:

1. `set_seed(seed) -> None` (3 pts)
   Set seeds for `random`, `numpy`, and `torch` so results are deterministic.
2. `get_cifar10_loaders(batch_size, seed, limit_train=None, limit_test=None) -> (train_loader, test_loader)` (7 pts)
   Return PyTorch `DataLoader`s. If `limit_train` is not `None`, use only the first `limit_train` training examples (same for `limit_test`). Do not shuffle test data.

1.2 MLP model (15 points)

Implement a model class:

1. `class MLP(nn.Module)` (15 pts)
   Your MLP must:
   - Flatten input images to vectors of length $32 \cdot 32 \cdot 3 = 3072$.
   - Have at least two hidden layers.
   - Use ReLU activations.
   - Use dropout in at least one hidden layer (dropout probability as a constructor argument).
   - Output logits of shape $(B, 10)$.

   The autograder will check layer shapes and forward outputs.

1.3 Training loop + metrics (10 points)

Implement:

1. `train_one_epoch(model, loader, optimizer, device) -> dict` (6 pts)
   Run one epoch of training and return a dict with `loss` (float) and `acc` (float).
2. `evaluate(model, loader, device) -> dict` (4 pts)
   Run evaluation and return a dict with `loss` and `acc`.

Problem 2: Convolutional Neural Net (CNN) on CIFAR-10 (45 points)

In this problem, you will implement a CNN that significantly improves over the MLP. Your model must be small enough to train quickly under the grading budget.

All required functions/classes for Problem 2 are in `problem2.py`.

2.1 CNN architecture (25 points)

Implement:

1. `class SmallCNN(nn.Module)` (25 pts)
   Your CNN must include:
   - At least 3 convolutional layers.
   - Batch normalization in at least 2 places.

- At least one pooling operation (max-pool or avg-pool).
- A global pooling or adaptive pooling before the classifier head.
- A classifier head that outputs logits of shape $(B, 10)$.

You may choose kernel sizes/widths. The autograder will check: (1) output shape, (2) parameter count within a given range, and (3) correctness of forward pass.

2.2 Challenging: robustness check via simple corruption (10 points)

Implement:

1. `apply_corruption(x, kind, severity, seed) -> x_corr` (10 pts)
   Given a batch of images `x` in $[0, 1]$, apply a deterministic corruption: `kind` is one of {`"gaussian_noise"`, `"channel_drop"`, `"cutout"`}. Severity controls strength/area. Return corrupted images in $[0, 1]$. The autograder will test correctness on fixed seeds.

2.3 Training budget comparison (10 points)

Implement:

1. `compare_mlp_cnn(mlp, cnn, train_loader, test_loader, device, epochs, seed) -> dict` (10 pts)
   Train both models for the same number of epochs and return a dict with: `{"mlp_test_acc": ..., "cnn_test_acc": ..., "delta": ...}` where `delta = cnn_test_acc - mlp_test_acc`. The autograder will check that your CNN learns better than the MLP under the given budget (on a small subset, with fixed seed).

Problem 3: RNN treating images as sequences (25 points)

In this problem, you will treat each CIFAR-10 image as a sequence and classify it using an RNN. We will treat each image as a sequence of **columns**: each timestep is one column of pixels.

For an image of shape $(3, 32, 32)$, the sequence length is 32. At each timestep t, the input vector is the concatenation of RGB values in column t: a vector of dimension $3 \cdot 32 = 96$.

All required functions/classes for Problem 3 are in `problem3.py`.

3.1 Sequence conversion (10 points)

Implement:

1. `image_to_column_sequence(x) -> seq` (10 pts)
   Input: `x` of shape $(B, 3, 32, 32)$ in $[0, 1]$.
   Output: `seq` of shape $(B, 32, 96)$ representing columns as timesteps.

3.2 RNN model (15 points)

Implement:

1. `class ColumnRNN(nn.Module)` (15 pts)
   Your model must:

   - Use a `torch.nn.RNN` or `torch.nn.GRU` or `torch.nn.LSTM` module.
   - Take input sequences of shape $(B, 32, 96)$ and output logits of shape $(B, 10)$.
   - Use the final hidden state (or pooled hidden states) to predict classes.

   The autograder will check output shapes and that the model can train (loss decreases) on a small subset.

Statement of Collaboration (5 points)

Add a plain text file `statement.txt` to your submission. List the names of collaborators and the nature of collaboration, or write "No collaboration." if you worked alone. Do not share code. Academic honesty policies apply.