Assignment 10

Please refer to MyCourses/Gradescope for assignment deadlines.

The goal of this assignment is to give you hands-on practice with the following concepts:

- Handling image datasets in PyTorch
- Training and evaluation of a simple image classification model

Part 1 of 2: Datasets and DataLoaders

In this part of the assignment, you'll briefly poke around the MNIST and CIFAR-10 datasets using the `torchvision` library. You'll write two functions whose job it is to simply print out some basic information about a given dataset.

In PyTorch, a `Dataset` class is a simple way to represent a collection of data samples. Custom `Datasets` are super easy to implement; essentially, any object that subclasses `torch.utils.data.Dataset` and provides an implementation of the `__len__` and `__getitem__` methods can be used as a dataset. For instance, you could write a custom `Dataset` class to load data from a CSV file, or from a directory of images, or from a database. As easy as that is, we're going to do something even easier and use some of the built-in datasets that PyTorch (specifically the `torchvision` library) provides.

Let's say you call `ds = MyDataset()`. Then, you could access the `i`-th sample in the dataset by calling `ds[i]`, which would call the `__getitem__` method of the `MyDataset` class. However, this isn't how we use datasets in practice, for a few reasons:

1. we'll want to get an entire *batch* of samples at a time,
2. it's useful to shuffle the dataset so that we don't always see the samples in the same order, and
3. if the `__getitem__` method is slow (e.g., if it's reading from disk), we'd ideally run the slow i/o operations for the next batch in the background while the model is training on the current batch.

All of this is handled by the `DataLoader` class in PyTorch. The `DataLoader` class wraps a `Dataset` object and provides an iterator that returns batches of samples. It also handles shuffling, batching, and (if you set `num_workers` to a value greater than 0) running the slow `__getitem__` method in parallel with the model training. It looks like this:

```
dl = DataLoader(ds, batch_size=32, shuffle=True, num_workers=4,
↪  pin_memory=True)
```

The `pin_memory` argument is useful if you're using a GPU and can speed up data transfer to the GPU. You can read about how it works **here** if you're curious.

Instructions

You must run this code on the CS department servers. The datasets have already been downloaded for you and placed in `/local/sandbox/csci631/datasets`. This is configured in `config.py`, which you should not modify except perhaps temporarily for debugging purposes.

Fill in the missing code in `datasets.py` to print out useful diagnostic information about a dataset or dataloader. The string formatting is done for you, you just need to inspect the properties of the `Dataset` and `DataLoader` object. *Do not modify the string in what you submit so that we can unit-test it*. Run the `datasets.py` script with `--dataset mnist` or `--dataset cifar10` to see some information about each one.

Example output

Output from running `python datasets.py --dataset mnist`:

Dataset with 60000 samples, 10 classes, image shape torch.Size([1, 28, 28]) and dtype torch.float32. DataLoader with 60000 total samples split across 1875 batches of size 32. Batch shape is torch.Size([32, 1, 28, 28]).

Part 2 of 2

In this part of the assignment, you'll train a simple **logistic regression** model to classify small hand-written digits (MNIST dataset) and small color images (CIFAR-10 dataset). This is one of the simplest model types we can use for image classification. We're using it here so that

1. You can get experience with PyTorch's image handling and training process without the added complexity of big fancy models, and
2. You get a good sense of the baseline performance of a simple model on these datasets.

Instructions

1. Fill in the missing code in `models.py`. You must implement the `__init__` and `forward` methods of the `LogisticRegression` class. The `__init__` method should initialize the weights and biases of the model, and the `forward` method should compute the output of the model given an input tensor. Logistic regression from data X to labels y is given by the equation $p(y) = \sigma(z)$, where $z = WX + b$ and σ is the softmax function. This maps from an input vector X to a probability distribution over class labels. Importantly, a single `nn.Linear` layer

accomplishes the $WX + b$ part of the equation, and we're actually going to return z from the `forward` method, so the model itself is a bit simpler than the equation might suggest.

2. Fill in the missing code in `train.py`, and study the code that is already provided for you there. Your primary task is to implement the `train_single_epoch` function, which is called by the `train` function. The `train_single_epoch` function should train the model for one epoch (i.e., one loop over the dataloader). It should log the accuracy and loss of the model on the training set in the Tensorboard `SummaryWriter`. Use the `evaluate()` function as a reference. The primary differences between `train_single_epoch` and `evaluate` are that

   - `train_single_epoch` should call `model.train()` at the start rather than `model.eval()`
   - it should call `optimizer.zero_grad()` at the start of each batch
   - it should call `loss.backward()` and `optimizer.step()` at the end of each batch
   - it should log the loss and accuracy to the `SummaryWriter` every batch
   - it should increment `step` each batch
   - it should not have a `with torch.no_grad()` block (we do want grads while training!)

3. Run the training script **at least** four times: twice for MNIST and twice for CIFAR-10, using at least two different learning rates. You can use the `--dataset` and `--lr` flags to specify the dataset and learning rate, respectively. For instance, to train on CIFAR-10 with a learning rate of 0.01, you would run `python train.py --dataset cifar10 --lr 0.01`. You can also specify the number of epochs to train for with the `--epochs` flag. Remember, the goal at this stage is not to make the best or fanciest model. It's to get comfortable with the training process and to get a sense of how well a simple model can do on these datasets.

   Earlier, you used a Tensorboard `SummaryWriter` object to log the loss and accuracy of the model over the course of training. Tensorboard is a great tool for visualizing how your model training is progressing. It works by writing logs to a specified directory, which you can then view as interactive graphs in a browser window once you start the tensorboard server. To start the server, SSH into the CS department server, activate your virtual environment where tensorboard is installed, and run

   ```
   tensorboard --bind_all --logdir=logs --port=6006
   ```

   Then, open a browser window and navigate to `http://<servername>.cs.rit.edu:6006` to see the Tensorboard UI. **Note: only one person can run a tensorboard server on a given port at a time. If you get an error message about the port being in use, try a value other than 6006. When you're done, kill your tensorboard server to free up ports and processing for other students!**

   Once you've successfully trained at least 2 models on each dataset with different hyperparame-

ters, take a screenshot of the Tensorboard UI and include it in your submission in the `images` directory.

Expected output

Here's what a functioning tensorboard UI might look like after training two MNIST models with different learning rates:

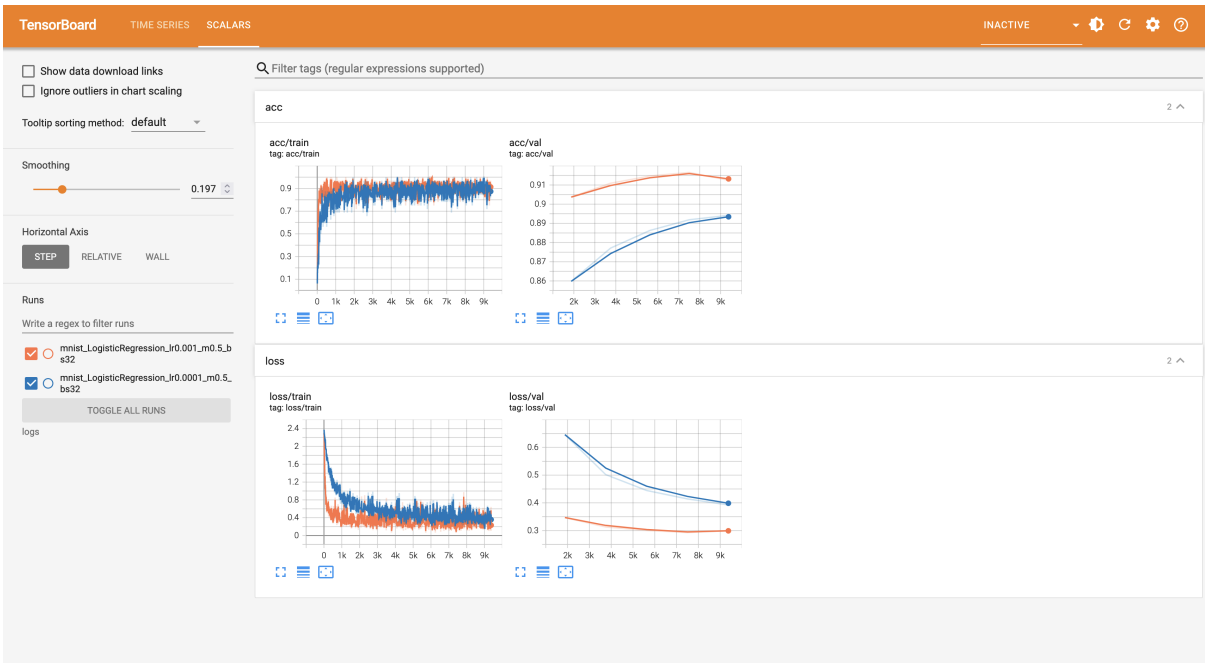

Figure 1: Example tensorboard screenshot

Your screenshot(s) must also include loss and accuracy plots for logistic regression trained on the CIFAR-10 dataset.

Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did in the `collaboration-disclosure.txt` file.

Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?
- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Give your reflections in the `reflection.txt` file if you choose to give us feedback.

Submitting

Use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline. (This is to prevent us from having to grade your work twice. Let us know so that we don't start grading files that will be replaced later).