ECE-GY 9143 - High Performance Machine Learning

Homework Assignment 2

Prof. Zehra Sura & Prof. Robert Kingan

Due Date: March 4, 2026 Spring 2026 Max Points: 110

Instructions

This lab is intended to be performed **individually**, great care will be taken in verifying that students are authors of their own submission. Theoretical questions are identified by Q⟨number⟩ while coding exercises are identified by C⟨number⟩.

1 Introduction

This assignment will give you experience on how to profile machine learning training and/or inference workloads, which is critical to understanding and improving system performance. You will also learn how to optimize a trained model for deployment using TorchScript. To achieve this, we will work with a CNN in PyTorch to classify images. We will use the CIFAR10 dataset, which contains 50K 32×32 color images. The reference code is at pytorch-cifar We will work with the ResNet-18 model, as described in Deep Residual Learning for Image Recognition.

2 Setup

2.1 Model

Create a ResNet-18 model as defined in Deep Residual Learning for Image Recognition. You can rely on existing open-source implementations. However, your code should define the layers and not just import the model using torch.

Specifically, the first convolutional layer should have 3 input channels, 64 output channels, 3×3 kernel, with *stride=1* and *padding=1*.

Followed by 8 basic blocks in 4 sub groups (i.e. 2 basic blocks in each subgroup):

- The first sub-group contains convolutional layer with 64 output channels, 3×3 kernel, *stride=1, padding=1*.
- The second sub-group contains convolutional layer with 128 output channels, 3×3 kernel, *stride=2, padding=1*.
- The third sub-group contains convolutional layer with 256 output channels, 3×3 kernel, *stride=2, padding=1*.
- The fourth sub-group contains convolutional layer with 512 output channels, 3×3 kernel, *stride=2, padding=1*.

- The final linear layer is of 10 output classes.

For all convolutional layers, use ReLU activation functions, and use batch normalization layers to avoid covariant shift. Since batch-norm layers regularize the training, set bias=0 for all convolutional layers. Use SGD optimizer with learning rate 0.1, momentum 0.9, weight decay $5e^{-4}$. The loss function is cross entropy.

2.2 DataLoader

Create a PyTorch program with a DataLoader that loads the images and the related labels from the torchvision CIFAR10 dataset. Import CIFAR10 dataset for the torchvision package, with the following sequence of transformations:

1. Random cropping, with size 32×32 and padding 4
2. Random horizontal flipping with a probability 0.5
3. Normalize each image's RGB channel with mean (0.4914, 0.4822, 0.4465) and variance (0.2023, 0.1994, 0.2010)

You will only need one data loader to complete this assignment. For your convenience, here are the default settings for the train loader: minibatch size of 128 and 3 IO processes (i.e., `num_workers=2`).

3 Part A: Training and Profiling (60 points)

C1: Training in PyTorch — 15 points

Create a main function that creates the *DataLoaders* for the training set and the neural network, then run 5 epochs with a complete training phase on all the minibatches of the training set. Write the code as device-agnostic, use the *ArgumentParser* to be able to read parameters from input, such as the use of cuda, the *data-path*, the number of dataloader workers and the optimizer (as string, eg: 'sgd').

Calculate the per-batch training loss and the top-1 training accuracy of the predictions, measured on training data.

Note: (i) Typically, we would like to examine test accuracy as well, however, it is sufficient to just measure training loss and training accuracy for this assignment. (ii) You don't need to submit any outputs for C1. You'll only need to submit the relevant code for this question. C2–C6 will be based on the code you wrote for C1.

C2: Time Measurement — 10 points

Report the running time (by using `time.perf_counter()` or other timers you are comfortable with) for the following sections of the code:

(C2.1) Data-loading time for each epoch

(C2.2) Training (i.e., mini-batch calculation) time for each epoch

(C2.3) Total running time for each epoch

Note: Data-loading time here is the time it takes to load batches from the generator (exclusive of the time it takes to move those batches to the device).

C3: I/O Optimization — 10 points

(C3.1) Report the total time spent for the Dataloader varying the number of workers starting from zero and increment the number of workers by 4 (0, 4, 8, 12, 16...) until the I/O time does not decrease anymore. Draw the results in a graph to illustrate the performance you are getting as you increase the number of workers. (C3.2) Report how many workers are needed for the best runtime performance.

C4: Training on GPUs vs CPUs — 10 points

Report the average running time over 5 epochs using the GPU vs using the CPU (using the number of I/O workers found in C3.2).

C5: Experimenting with Different Optimizers — 10 points

Run 5 epochs with the GPU-enabled code and the optimal number of I/O workers. For each epoch, report the average training time, training loss and top-1 training accuracy using these Optimizers: SGD, SGD with Nesterov, and Adam. Note please use the same default hyperparameters: learning rate 0.1, weight decay $5e^{-4}$, and momentum 0.9 (when it applies) for all these optimizers.

C6: Experimenting without Batch Norm — 5 points

With the GPU-enabled code and the optimal number of workers, report the average training loss, top-1 training accuracy for 5 epochs with the default SGD optimizer and its hyperparameters but without batch norm layers.

Q1 — 4 points

How many convolutional layers are in the ResNet-18 model?

Q2 — 4 points

What is the input dimension of the last linear layer?

Q3 — 8 points

How many trainable parameters and how many gradients in the ResNet-18 model that you build (please show both the answer and the code that you use to count them), when using SGD optimizer?

Q4 — 4 points

Same question as Q3, except now using Adam (only the answer is required, not the code).

4 Part B: Model Optimization with TorchScript (30 points)

In this part of the lab, you will experiment with the process of transitioning the ResNet-18 model that you trained in Part A to TorchScript using the TorchScript API. TorchScript has two core modalities for converting an eager-mode model to a TorchScript graph representation: **tracing** and **scripting**. You can use the PyTorch TorchScript tutorial to help you perform this section of the lab.

Q5 — 3 points

Explain the differences between tracing and scripting and how they are used in TorchScript.

Q6 — 2 points

Explain any changes needed in the ResNet-18 model to allow for scripting.

C7: Convert to TorchScript — 5 points

Convert the trained ResNet-18 model from Part A to TorchScript using `torch.jit.script` to convert the model. Make any necessary modifications to the model code so that it is compatible with TorchScript's static analysis (refer to your answer in Q6).
Save the converted model to disk using `torch.jit.save`. Verify that the saved model can be reloaded with `torch.jit.load`.

C8: Print Model Graph — 5 points

Print the graph of the converted TorchScript model.

C9: Evaluate the TorchScript Model — 5 points

Evaluate the TorchScript model on the CIFAR10 test set and report the top-1 accuracy.

C10: Latency Comparison — 10 points

Compare the evaluation latency of the TorchScript model and the original PyTorch model on both CPU and GPU. Create a latency comparison table. How much speedup are you getting, if any? Explain your findings.

	Latency on CPU (ms)	Latency on GPU (ms)
PyTorch		
TorchScript		

Extra Credit — 10 points

Save and serialize the TorchScript model for use in a non-Python deployment environment. Show how to load and run the model in a C++ program using LibTorch. Submit your C++ code and a README file explaining how to compile and run it.

Appendix – Submission Instructions

Please submit a `.tar.gz` archive containing the following files:

- A file named `lab2.py` (or equivalent set of files) containing all the code used for Part A. It should be clear what code is needed for which exercises C1–C6 and Q3.
- A file named `lab2_torchscript.py` containing all the code for Part B (C7–C10).
- A file named `lab2.pdf` with a report of the outputs requested in C2–C10 and Q1–Q6. The file should include the terminal screenshots with outputs of executions.
- A file named `README.md` which explains how to run your code. The course staff should be able to run all experiments with one command.
- Failing to follow the right directory/file name specification is −1 point. Failing to have programs executed in sequence is −1 point.

Appendix – How to Run Experiments

Running jobs

- All jobs that do not require a GPU need to be executed on the CPU-only nodes.
- Please run all jobs in the same computing environment for consistency of results.

Measuring GPU time

Using `time.time()` (or similar) alone is not accurate to measure GPU times. You will need to use the following to synchronize your code with the CUDA kernel:

```
...
torch.cuda.synchronize()  # wait for any running kernels to finish
start = time.time()
<code you want to profile>
torch.cuda.synchronize()  # wait for any running kernels to finish
end_epoch = time.time()
...
```