High Performance Machine Learning Lab2

This lab is intended to be performed **individually**, great care will be taken in verifying that students are authors of their own submission and that the exam is different from previous courses.

Exercises need to be executed on the **Prince NYU cluster** in the standard compute nodes.

Theoretical questions are identified by **Q<number>** while coding exercises are identified by **C<number>**.

In this part of the lab we create a Neural Network in PyTorch to classify a dataset of images. The dataset we are going to use is CIFAR10, which contains 50K 32x32 color images. The model we are going to build is ResNet-18, as described in https://arxiv.org/abs/1512.03385 . The refence code is at https://github.com/kuangliu/pytorch-cifar

C1 Training in PyTorch [10 points]

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

1. Random cropping, with size 32x32 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)

The DataLoader for the training set uses a minibatch size of 128 and 3 IO processes (i.e., num_workers=2)
The DataLoader for the testing set uses minibatch size of 100 and 3 IO processes (i.e., num_workers =2)

Create a ResNet-18 model as defined in https://arxiv.org/abs/1512.03385
Specifically, The first convolutional layer should have 3 input channels, 64 output channels, 3x3 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, 3x3 kernel, stride=1, padding=1.
The second sub-group contains convolutional layer with 128 output channels, 3x3 kernel, stride=2, padding=1.
The third sub-group contains convolutional layer with 256 output channels, 3x3 kernel, stride=2, padding=1.
The forth sub-group contains convolutional layer with 512 output channels, 3x3 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 normal layers to avoid covariant shift. Since batch-norm layers regularize the training, set bias to 0 for all the convolutional layers. Use SGD optimizers with 0.1 as the learning rate, momentum 0.9, weight decay 5e-4. The loss function is cross entropy.

Create a main function that creates the DataLoaders for the training set and the neural network, then do a cycle of 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').
For each minibatch calculate the loss value, the precision@1 of the predictions.
Precision@k means that you get the first K higher predictions for each sample (with output.topk()) and you count the true labels among these prediction.

C2: Time measurement of code in C1 [5]
Report the real time using *time.perf_counter()* for the following sections of the code:

- Aggregated time spent waiting to load the batch from the DataLoader during the training
- Aggregated time for a mini-batch computation (dataloading and NN forward/backward)
- Aggregated time for each epoch

C3: I/O optimization starting from code in C2 [5]

- Report the total time spent waiting 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 time doesn't decrease anymore.
- Report how many workers are needed for best performance

C4: Profiling starting from code in C3 [5]
Save the profiling file for the exercise C3 using 1 worker and the number of workers needed for best performance.
Visualize the data-loading time, computing time for these two runs and explain (in few words) the differences.

C5: Training in GPUs and optimizer starting from the code in C3 [5]
Report the average epoch time over 5 epochs using the CPU vs using the GPU (using the number of IO workers found in C3)

C6: Experimenting with different optimizers [5]
With the GPU-enabled code and the optimal number of workers, report the average epoch time and loss, precision@1 for 5 epochs using these Optimizer algorithms:
SGD, SGD with nesterov, Adagrad, Adadelta, and Adam

C7: Experimenting without Batch Norm layer [5]
With the GPU-enabled code and the optimal number of workers, report the average loss, precision@1 for 5 epochs using the default SGD optimizer, without batch norm layers.

Q1: How many convolutional layers are in the ResNet-18 model ?[2]
Q2: What is the input dimension of the last linear layer ? [2]
Q3: 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. [4]
Q4: Same question as Q2, except now using Adam (only the answer is required not the code). [2]

Grading rules

- Total is 50 points
- Late submission is -3 points for every day, maximum 3 days.

Appendix – Submission instructions

Submission through NYUClasses.

Please submit a targz archive with file name *<your-netID>.tgz* with a folder named as your netID (example: am9031/) containing the following files:

- A file named lab2.py containing all the exercises (please also include the model file if you use separate python files). Insert in this file the code used for all the exercise, commenting all the lines needed for the exercise C2-C7 and Q3.
- A file named lab2.pdf with a report of the outputs requested in C2-C7 and Q1-Q4.
- 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

All jobs that do not require a GPU need to be executed on the **CPU-only nodes**. There are only a few **GPU nodes**