HPC

William Jiang(wj419)

**Report**

C2.1. When run using optimizer SGD and number of workers = 2, the dataloading portion takes about 0.233 seconds per epoch after the first epoch.

C2.2. The minibatch calculation is recorded in the forward pass with a rough average runtime of 1.55 seconds per epoch and the backward pass at around 4.39 seconds (combined at around 5.94 seconds) after the first epoch.

C2.3. The total runtime per epoch is 38.11 seconds after the first epoch.

C3. The dataloader runtime for:

* 0 workers = 0.144 seconds
* 4 workers = 0.245 seconds
* 8 workers = 0.249 seconds
* 12 workers = 0.246 seconds
* 16 workers = 0.246 seconds

Interestingly, for the purposes of this example, it seems that setting num\_of\_workers = 0 gives the best IO performance.

C4. With 0 workers (num\_of\_workers=0), we have the main process on the CPU fetching the data which gives us the best performance. When num\_workers = 1, the dataload time is 0.254 and we begin to see a plateaued performance decrease as we increase the number of workers. This is most likely due to the memory-bound overhead of cudamemcpy onto the same GPU.

C5. The average runtime with the num\_of\_workers=0 using the GPU is 39.26 seconds. I was not able to get it to run on CPU since the job is killed immediately in HPC Prince cluster during training.

C6. Here are the average loss, accuracy and training time for each epoch of the optimizers:

* SGD: avg loss = 0.00685, accuracy = 79.55%, training time = 2.91s
* Nesterov: avg loss = 0.00703, accuracy = 81.24%, training time = 2.66s
* Adagrad: avg loss = 0.008, accuracy = 63.27%, training time = 3.82s
* Adadelta: avg loss = 0.0042, accuracy = 81.82%, training time = 5.04s
* Adam: avg loss = 0.014, accuracy = 32.22%, training time = 4.78s

C7. Without using Batch norm with SGD, the avg loss = 0.00687, accuracy = 69.52%, training time = 1.92s

Q1. There are 17 convnet layers. 1 in the initial convnet layer 4 per \_make\_layer function call of which there are 4.

Q2. The dimension of the inputs to the last fully connected layer is [128,512]

Q3. There is a total of 11173962 training parameters counted using the code using optimizer SGD:

total\_params = *sum*(p.numel() *for* p *in* net.parameters() *if* p.requires\_grad)

Likewise, the total gradients to be calculated is also 11173962 since the parameters that are to be calculated and changed are the gradients. To double check the validity, I added another code snippet that counts only the gradients:

params = *list*(net.parameters())  
total\_grads = 0  
*for* param *in* params:  
 total\_grads += param.grad.view(1, -1).squeeze().shape[0]  
*print*("total grads: %d" % total\_grads)

Q4. If the Adam optimizer is used instead, we get the same 11173962 training parameters and active gradients.