HPML ASSIGNMENT 2

Name: Sree Gowri Addepalli

NetID: sga297

GPU specifications used for the entire experiments:

```
(py3.6.3) [sga297@gpu-11 finalCopy]$ uname -a
Linux gpu-11 3.10.0-514.10.2.el7.x86_64 #1 SMP Fri Mar 3 00:04:05 UTC 2017 x86_6
4 x86_64 x86_64 GNU/Linux
```

This report is made by logs run previously. There is a fresh source of logs made through the batch script (hpml2Run.sh)

C1:

See the code attached to re run the experiments.

Attached log file c1.txt and logTestc1.txt for results.

PS: Used max instead of top as here precision@1 is needed where max =top (1).

The default hyperparameters are used to run this code.

C2:

See the code attached to re run the experiments.

(All logs have the following terms printed to reduce difference in subjectivity):

Minibatch Level Logs:

Train Epoch: – The Epoch Number

Batch Number – The mini batch number

Batch Loss - The mini batch loss

Batch Data Load Time – The time to load the batch data

Batch Computation Time- The Time to perform all operations in a minibatch

Batch precision value – The minibatch precision value.

Epoch Level Logs:

Total Epoch Time: – The total Time for each epoch to run

Total Epoch Loss: – The total epoch loss (sum of all minibatches loss)

Average Epoch Loss: – Average loss in an epoch for a minibatch

Total Precision Value: – The total precision in an epoch (sum of all precisions)

Average Precision Value: - Average precision in an epoch for a minibatch

Total Time loading all batches data in this epoch: – The total data loading time for all batches in this epoch.

Average Time loading all batches data in this epoch: – The average data loading time for all batches in this epoch.

Total time for all batches computation in this epoch: – The total data computation time for all batches in this epoch.

Average time for all batches computation in this epoch: - The average data computation time for all batches in this epoch.

Cumulative Log levels:

Epochs.arg – The total number of epochs mentioned

Total Epochs time: – The cumulative sum of time taken to run all epochs mentioned in epochs.arg

Average all Epochs Time: – The average per epoch time taken to run each epoch.

All Epoch Data Loading Time: – The cumulative sum of time taken to load all data in all epochs mentioned in epochs.arg

Average all Epochs Data Loading Time: – The average of time taken to load all data in each epoch mentioned in epochs.arg

All Epoch Computation Time: – The cumulative sum of computation time taken in all epochs mentioned in epochs.arg

Average all Epochs Total Computation Time: – The average of computation time taken in each epoch mentioned in epochs.arg

All Epoch Loss: – The cumulative sum of loss in all epochs mentioned in epochs.arg

Average all Epochs Loss: – The average of loss per epoch.

All Epoch Precision Value: - The cumulative precision of loss in all epochs mentioned in epochs.arg

Average all Epochs Precision Value: - The average of all precision per epoch.

Test Level logs:

Batch Test Loss: The mini batch loss.

Batch Test Precision: The mini batch precision.

Average Epoch Test Precision Value: - The average of all precision per epoch in epoch. args.

Average Epochs Precision Value: - The average of all precision per epoch in epoch. args.

Here is a sample set of values for 5 Epochs, with worker =2

Epoch Number	Data Loading Time	Computation Time	Epoch Time
0	1.04	18.608	39.87
1	0.906	16.367	38.709
2	0.906	16.262	38.775
3	0.89368	16.32	38.811
4	0.8944	16.33775	38.796

See attached log file c1.txt and logTestc1.txt for results

C3:

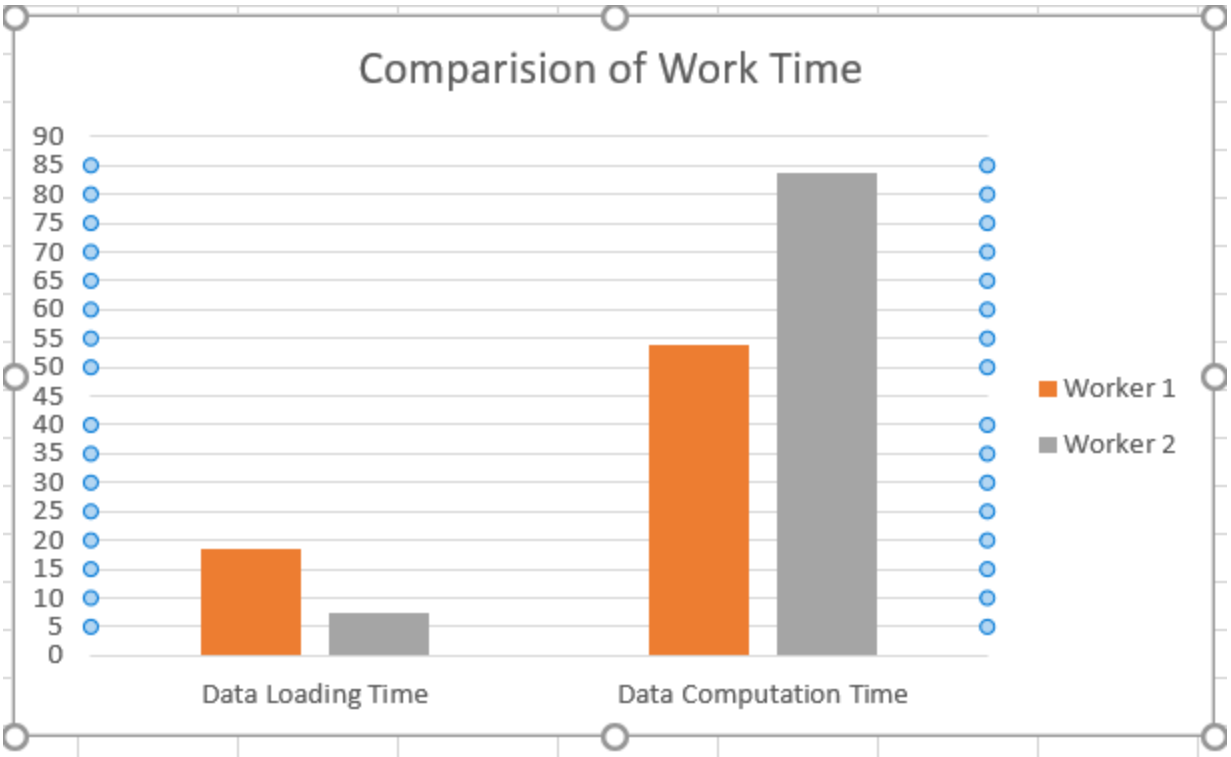
5 epochs have been used:

Number of Workers	Data Loading Time
0	74.8366
4	6.9844
8	13.2572
12	13.56512

4 workers are needed for optimal performance, as after that data loading time never decrease.

C4:

No.of Workers	Data Loading Time	Data Computation Time
1	18.587	53.798
4	7.439	83.74



(Worker 2 here means worker with 4 worker nodes)

As the optimal number of workers, we can see that the Data loading time for optimal number **(Workers = 4)**of workers is less, as with one worker, the time is spent in waiting for the file to read, while when the workers are increased, data is parallelly processed which is why it takes less data loading time when there are more number of workers which improves CPU performance. While data computation time is linearly correlated to the number of workers, as each worker spends time in computation on the data(parallelly).

C5:

Average Epoch Time using 4 workers (5 epochs) (GPU)– 37.476 secs

Average Epoch Time using 4 workers (5 epochs) (CPU)– 9996.878 secs

See logc5CPU.txt and logc5GPU.txt for log results.

GPU:

Epoch Number	Epoch Time
0	41.659
1	35.838
2	36.307
3	36.446
4	37.133

CPU:

Epoch Number	Epoch Time
0	9984.45
1	10002.12
2	9999.45
3	9997.09
4	10001.28

C6:

Config - GPU, no_of_workers =4, Epochs = 5

Optimizer	Avg Epoch time	Average Loss	Precision@1
SGD	39.0602	1.3860	0.4978
SGD with Nesterov	37.2809	1.408	0.4866
AdaGrad	37.2875	1.068	0.61525
Adadelta	35.909	1.006	0.63723

Adam	**41.3621**	**0.88044**	**0.68437**

The best performing Optimizer is Adam. Test Precisions are in the report. See the following logs for the answers:

- Logc6SGD.txt
- Logc6SGDNes.txt
- Logc6AdaGrad.txt
- Logc6Adadelta.txt
- Logc6Adam.txt

C7: Here are the following logs for without batch Norm.

See the logs for the results and test precision.

Epoch 0:

Avg Epoch Time- 31.375 | Avg Loss- 1.53 | Precision@1 – 0.4317

Epoch 1:

Avg Epoch Time- 30.213 | Avg Loss- 1.11 | Precision@1 – 0.6

Epoch 2:

Avg Epoch Time- 30.214 | Avg Loss- 0.93644 | Precision@1 – 0.67021

Epoch 3:

Avg Epoch Time- 30.2838 | Avg Loss- 0.8200 | Precision@1 – 0.7113

Epoch 4:

Avg Epoch Time- 30.166 secs | Avg Loss - 0.7523 | Precision@1 – 0.7390

All Epochs Avg:

Avg Epoch Time- 30.4507 secs | Avg Loss- 1.031 | Precision@1 – 0.6304

See Log Files for test precision or any other values.

Q1:

There are 17 convolutional layers in Resnet 18 model as per standard architecture.

There are 20 convolutional layers in Resnet 18 model as per Pytorch Resnet 18 (Pytorch Summary code.)

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,728
       BatchNorm2d-2          [-1, 64, 32, 32]             128
            Conv2d-3          [-1, 64, 32, 32]          36,864
       BatchNorm2d-4          [-1, 64, 32, 32]             128
            Conv2d-5          [-1, 64, 32, 32]          36,864
       BatchNorm2d-6          [-1, 64, 32, 32]             128
        BasicBlock-7          [-1, 64, 32, 32]               0
            Conv2d-8          [-1, 64, 32, 32]          36,864
       BatchNorm2d-9          [-1, 64, 32, 32]             128
           Conv2d-10          [-1, 64, 32, 32]          36,864
      BatchNorm2d-11          [-1, 64, 32, 32]             128
       BasicBlock-12          [-1, 64, 32, 32]               0
           Conv2d-13         [-1, 128, 16, 16]          73,728
      BatchNorm2d-14         [-1, 128, 16, 16]             256
           Conv2d-15         [-1, 128, 16, 16]         147,456
      BatchNorm2d-16         [-1, 128, 16, 16]             256
           Conv2d-17         [-1, 128, 16, 16]           8,192
      BatchNorm2d-18         [-1, 128, 16, 16]             256
       BasicBlock-19         [-1, 128, 16, 16]               0
           Conv2d-20         [-1, 128, 16, 16]         147,456
      BatchNorm2d-21         [-1, 128, 16, 16]             256
           Conv2d-22         [-1, 128, 16, 16]         147,456
      BatchNorm2d-23         [-1, 128, 16, 16]             256
       BasicBlock-24         [-1, 128, 16, 16]               0
           Conv2d-25           [-1, 256, 8, 8]         294,912
      BatchNorm2d-26           [-1, 256, 8, 8]             512
           Conv2d-27           [-1, 256, 8, 8]         589,824
      BatchNorm2d-28           [-1, 256, 8, 8]             512
           Conv2d-29           [-1, 256, 8, 8]          32,768
      BatchNorm2d-30           [-1, 256, 8, 8]             512
       BasicBlock-31           [-1, 256, 8, 8]               0
           Conv2d-32           [-1, 256, 8, 8]         589,824
      BatchNorm2d-33           [-1, 256, 8, 8]             512
           Conv2d-34           [-1, 256, 8, 8]         589,824
      BatchNorm2d-35           [-1, 256, 8, 8]             512
       BasicBlock-36           [-1, 256, 8, 8]               0
           Conv2d-37           [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-38           [-1, 512, 4, 4]           1,024
           Conv2d-39           [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-40           [-1, 512, 4, 4]           1,024
           Conv2d-41           [-1, 512, 4, 4]         131,072
      BatchNorm2d-42           [-1, 512, 4, 4]           1,024
       BasicBlock-43           [-1, 512, 4, 4]               0
           Conv2d-44           [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-45           [-1, 512, 4, 4]           1,024
           Conv2d-46           [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-47           [-1, 512, 4, 4]           1,024
       BasicBlock-48           [-1, 512, 4, 4]               0
           Linear-49                 [-1, 10]           5,130
           ResNet-50                 [-1, 10]               0
```

Q2: 512 is the input dimension of the last linear layer. As you can see from the Pytorch Summary architecture:

The input architecture is 512 x 4 X 4.

See the above architecture.

Q3: As per the diagram below, the number of trainable parameters and gradients using SGD optimizer:

11,173,962. (Code attached in file.) (Used Pytorch.summary from pytorch)

from torchsummary import summary

summary(net,(3,32,32))

```
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.25
Params size (MB): 42.63
Estimated Total Size (MB): 53.89
```

Q4: As per the diagram below, the number of trainable parameters and gradients using Adam optimizer:

11,173,962. (Code attached in file.)

```
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.25
Params size (MB): 42.63
Estimated Total Size (MB): 53.89
```