CS 6886: Systems Engineering for Deep Learning

Mini Assignment 3

Total Marks: 5

- This assignment is based on using GCP through the vouchers shared earlier.
- Enable access to GPUs on GCP with the instructions shared on the course stream.
- The questions require that you create accounts on wandb.com for managing your experiments and reporting results.

This question is on hyperparameter tuning, which forms a major part of the working life of a DL engineer.

1. In this document you are provided a detailed description of a neural network. (You may not find this network online, so it is not much use trying to look for optimized hyperparameters.)
2. You are required to implement this network in PyTorch. You should share your model through a secret gist. We recommend you use version control tools such as Github to keep track of your work.
3. This network is then to be trained on the CIFAR100 dataset using the standard train and test splits.
4. Your goal is to try out different hyperparameters to maximize the accuracy of the model. You are not allowed to change the dataset itself - for example by augmenting the images or adding preprocessing. You are also not to change the neural network itself - for example by increasing kernel size or adding dropout. Instead, we want you to make good choices for the learning algorithm (eg. SGD) and the algorithm's parameters (eg. learning rate, batch size, number of epochs, etc.).
5. You are required to log these optimization runs on wandb.com and share the report.
6. You are also poor - you have only 50 USD which does not count for much in deep learning. So, this assignment also requires that you judiciously use your compute resources. You are only allowed to report results on 15 hours of compute on Tesla P4

(which costs less than 10 USD). For this you are required to track the amount of time taken for each experiment, and also report it. You may take a couple of hours of extra compute to get going in learning about GCP, moving data etc.

7. This will be graded in three parts
    a. Correctly implementing the neural network (2 marks)
    b. Reporting a wide range of hyperparameter tuning experiments (2 marks)
    c. Relative performance of your best model against other submissions (1 mark)

Gist of PyTorch model - https://gist.github.com/omshri22121999/476eb3de0bf2eaa9ca2bd229e5b797c1

Model Gist :

```
x = self.conv1(x)
x = self.hswish(x)
x = self.batchn1(x)

x = self.fuse1(x)
x = self.fuse2(x)
x = self.fuse3(x)
x = self.fuse4(x)
x = self.fuse5(x)
x = self.fuse6(x)
x = self.fuse7(x)
x = self.fuse8(x)
x = self.fuse9(x)
x = self.fuse10(x)
x = self.fuse11(x)

x = self.conv2(x)
x = self.hswish(x)
x = self.batchn2(x)
x = self.adap(x)
x = self.conv3(x)
x = self.hswish(x)
x = x.flatten(start_dim=1)
x = self.drop(x)
x = self.lin(x)
```

Report of wandb.com for hyperparameter tuning - https://wandb.ai/omshri/fusenet-runs/reports/Mini-Assignment-3-Report--VmlldzoyODMxMzg?accessToken=fon1862j4vuu2ikn2f58mygr3xu5pf53zomswi2tyrlcgc96xm3non1yngv1lzc3

Also add a table here with rows as experiments, columns as hyperparameters, tracked metrics, and execution time of each experiment. Highlight the best 3 configurations.

Note : Tuned hyperparameters for **Adam** optimizer., also mentioning **best accuracy**

	batch _size	**beta0**	**beta1**	**epochs**	**eps**	**weight _decay**	**lr**	**Accuracy**	**Time**
run1	64	0.99	0.999	40	1e-8	0	1e-4	28.12	28m 47s
run2	128	0.99	0.999	40	1e-8	0	1e-4	27.49	18m 39s
run3	256	0.99	0.999	40	1e-8	0	1e-4	25.77	14m 34s
run4	128	0.99	0.999	40	1e-8	0	3e-4	30.16	18m 47s
run5	128	0.99	0.999	40	1e-8	0	1e-3	28.38	19m 7s
run6	128	0.9	0.999	40	1e-8	0	3e-4	31.26	18m 53s
run7	128	0.8	0.999	40	1e-8	0	3e-4	30.93	19m 1s
run8	128	0.7	0.999	40	1e-8	0	3e-4	30.45	19m 8s
run9	128	0.9	0.9	40	1e-8	0	3e-4	31.83	18m 42s
run 10	128	0.9	0.8	40	1e-8	0	3e-4	31.16	18m 36s
run 11	128	0.9	0.7	40	1e-8	0	3e-4	30.85	19m 6s
run 12	128	0.9	0.9	60	1e-8	0	3e-4	31.07	27m 59s
run 13	128	0.9	0.9	60	1e-8	0	7e-4	31.16	28m 21s

run 14	128	0.9	0.999	60	1e-8	0	1e-3	27.44	27m 48s
run 15	128	0.9	0.999	60	1e-8	1e-4	3e-4	34.29	28 37s
run 16	128	0.9	0.999	60	1e-8	1e-3	3e-4	33.08	29m 4s
run 17	128	0.9	0.999	60	1e-8	1e-5	3e-4	31.7	29m 1s
run 18	128	0.9	0.999	80	1e-8	1e-4	3e-4	33.97	38m 31s
run 19	128	0.9	0.999	80	1e-8	1e-4	7e-4	35.64	38m 7s
run 20	128	0.9	0.999	80	1e-8	1e-4	1e-3	35.33	38m 44s

Any observations or curious findings based on your experiments

- Increasing batch size reduced time taken for training and also increased
- 128 batches offered a good tradeoff between time taken for training and accuracy
- Changing the betas parameters (beta0 & beta1) affected the model very minimally
- Introducing regularization helped in improving the accuracy
- Regularization helped use higher learning rates better
- Training the model for more epoch in **run 19** and **run 20** might have improved accuracy, which shows the usefulness of higher learning rate with regularization.