CSCI-GA 3033-091
INTRO DEEP LEARNING SYS
Instructor: Parijat Dube
Due: April 4, 2022

# Homework 3

## Problem 1 - *Adaptive Learning Rate Methods, CIFAR-10* 20 points

We will consider five methods, AdaGrad, RMSProp, RMSProp+Nesterov, AdaDelta, Adam, and study their convergence using CIFAR-10 dataset. We will use multi-layer neural network model with two fully connected hidden layers with 1000 hidden units each and ReLU activation with minibatch size of 128.

1. Write the weight update equations for the five adaptive learning rate methods. Explain each term clearly. What are the hyperparameters in each policy? Explain how AdaDelta and Adam are different from RMSProp. (5+1)
2. Train the neural network using all the five methods with $L_2$-regularization for 200 epochs each and plot the training loss vs number of epochs. Which method performs best (lowest training loss) ? (5)
3. Add dropout (probability 0.2 for input layer and 0.5 for hidden layers) and train the neural network again using all the five methods for 200 epochs. Compare the training loss with that in part 2. Which method performs the best? For the five methods, compare their training time (to finish 200 epochs with dropout) to the training time in part 2 (to finish 200 epochs without dropout). (5)
4. Compare test accuracy of trained model for all the five methods from part 2 and part 3. Note that to calculate test accuracy of model trained using dropout you need to appropriately scale the weights (by the dropout probability). (4)

*References:*

- The CIFAR-10 Dataset.

## Problem 2 - *Data Parallelism in Pytorch* 25 points

We are going to experiment with PyTorch's DataParallel Module, which is PyTorch's Synchronous SGD implementation across a number of GPUs on the same server. In particular, we will train ResNet-18 implementation from https://github.com/kuangliu/pytorch-cifar with num_workers=2, running up to 4 GPUs with DataParallel (DP) Module. Use SGD optimizers with 0.1 as the learning rate, momentum 0.9, weight decay 5e-4. For this question, you need to do experiment with multiple GPUs on the same server. You may need to execute this on NYU Greene Cluster.

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

- Random cropping, with size 32x32 and padding 4
- Random horizontal flipping with a probability 0.5
- 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 main function that creates the DataLoaders for the training set and the neural network.

1. Measure how long does it take to compete 1 epoch of training using different batch size on a single GPU. Start from batch size 32, increase by 4-fold for each measurement (i.e., 32, 128, 512 ...) until

| | Batch-size 32 per GPU | | Batch-size 128 per GPU | | Batch-size 512 per GPU | |
|---|---|---|---|---|---|---|
| | Time(sec) | Speedup | Time (sec) | Speedup | Time (sec) | Speedup |
| 1-GPU | | 1 | | 1 | | 1 |
| 2-GPU | | | | | | |
| 4-GPU | | | | | | |

Table 1: Speedup Measurement for different batch size.

| | Batch-size 32 per GPU | | Batch-size 128 per GPU | | Batch-size 512 per GPU | |
|---|---|---|---|---|---|---|
| | Compute(sec) | Comm(sec) | Compute(sec) | Comm(sec) | Compute(sec) | Comm(sec) |
| 2-GPU | | | | | | |
| 4-GPU | | | | | | |

Table 2: Compute and Communication time for different batch size.

single GPU memory cannot hold the batch size. For each run, run 2 epochs, the first epoch is used to warmup CPU/GPU cache; and you should report the training time (*excluding* data I/O; but *including* data movement from CPU to GPU, gradients calculation and weights update) based on the 2nd epoch training. (5)

2. Measure running time with batch size per GPU you used in part 1 (i.e., 32, 128, ...) on 2 GPUs and 4 GPUs and calculate speedup for each setup. Again, for each setup, run 2 epochs, and only measure the 2nd epoch. When measuring speedup, one should include all the training components (e.g., data loading, cpu-gpu time, compute time). (5).
   *Expected Answer*: Table 1 records the training time and speedup for different batch size up to 4 GPUs. Comment on which type of scaling we are measuring: weak-scaling or strong-scaling? Comment on if the other type scaling was used speedup number will be better or worse than what you we are measuring.

3. Report for each batch size per gpu (i.e., 32, 128, 512 ...), how much time spent in computation (including CPU-GPU transferring and calculation) and how much time spent in communication in 2-GPU and 4-GPU case for one epoch. (hint You could use the training time reported in Question 1 to facilitate your calculation). (5)
   *Expected Answer*: First, describe how do you get the compute and communication time in each setup. Second, list compute and communication time in Table 2.

4. Assume PyTorch DP implements the all-reduce algorithm as discussed in the class (reference below), calculate communication bandwidth utilization for each multi-gpu/batch-size-per-gpu setup. (5)
   *Expected Answer*: First, list the formula to calculate how long does it take to finish an allreduce. Second, list the formula to calculate the bandwidth utilization. Third, list the calculated results in Table 3.

   *References:*

   - PyTorch Data Parallel, Available at https://pytorch.org/docs/stable/_modules/ torch/nn/parallel/data_parallel.html.
   - Bringing HPC Techniques to Deep Learning

## Problem 3 - *Convolutional Neural Networks Architectures* 35 points

In this problem we will study and compare different convolutional neural network architectures. We will calculate number of parameters (weights, to be learned) and memory requirement of each network. We will also analyze inception modules and understand their design.

# Homework 3

| | Batch-size-per-GPU 32 | Batch-size-per-GPU 128 | Batch-size-per-GPU 512 |
|---|---|---|---|
| | Bandwidth Utilization(GB/s) | Bandwidth Utilization(GB/s) | Bandwidth Utilization(GB/s) |
| 2-GPU | | | |
| 4-GPU | | | |

Table 3: Communication Bandwidth Utilization

1. Calculate the number of parameters in Alexnet. You will have to show calculations for each layer and then sum it to obtain the total number of parameters in Alexnet. When calculating you will need to account for all the filters (size, strides, padding) at each layer. Look at Sec. 3.5 and Figure 2 in Alexnet paper (see reference). Points will only be given when explicit calculations are shown for each layer. (5)

2. VGG (Simonyan et al.) has an extremely homogeneous architecture that only performs 3x3 convolutions with stride 1 and pad 1 and 2x2 max pooling with stride 2 (and no padding) from the beginning to the end. However VGGNet is very expensive to evaluate and uses a lot more memory and parameters. Refer to VGG19 architecture on page 3 in Table 1 of the paper by Simonyan et al. You need to complete Table 1 below for calculating activation units and parameters at each layer in VGG19 (without counting biases). Its been partially filled for you. (6)

3. VGG architectures have smaller filters but deeper networks compared to Alexnet (3x3 compared to 11x11 or 5x5). Show that a stack of $N$ convolution layers each of filter size $F \times F$ has the same receptive field as one convolution layer with filter of size $(NF - N + 1) \times (NF - N + 1)$. Use this to calculate the receptive field of 3 filters of size 5x5. (4)

4. The original Googlenet paper (Szegedy et al.) proposes two architectures for Inception module, shown in Figure 2 on page 5 of the paper, referred to as naive and dimensionality reduction respectively.

   (a) What is the general idea behind designing an inception module (parallel convolutional filters of different sizes with a pooling followed by concatenation) in a convolutional neural network ? (3)

   (b) Assuming the input to inception module (referred to as ”previous layer” in Figure 2 of the paper) has size 32x32x256, calculate the output size after filter concatenation for the naive and dimensionality reduction inception architectures with number of filters given in Figure 1. (4)

   (c) Next calculate the total number of convolutional operations for each of the two inception architecture again assuming the input to the module has dimensions 32x32x256 and number of filters given in Figure 1. (4)

   (d) Based on the calculations in part (c) explain the problem with naive architecture and how dimensionality reduction architecture helps (*Hint: compare computational complexity*). How much is the computational saving ? (2+2)

5. Faster-RCNN is a CNN based architecture for object detection which is much faster that Fast-RCNN. Read about Faster-RCNN in Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks and answer the following questions:

   (a) What is the main difference between Fast-RCNN and Faster-RCNN that resulted in faster detection using Faster-RCNN? (2)

   (b) What is Region Proposal Network (RPN)? Clearly explain its architecture. (2)

   (c) Explain how are region proposals generated from RPN using an example image.(3)

   (d) There is a lot of overlap in the region proposals generated by RPN. What technique is used in Faster-RCNN to reduce the number of proposals to roughly 2000? Explain how does this technique work using an example. (3)

# Homework 3

| Layer | Number of Activations (Memory) | Parameters (Compute) |
|---|---|---|
| Input | 224*224*3=150K | 0 |
| CONV3-64 | 224*224*64=3.2M | (3*3*3)*64 = 1,728 |
| CONV3-64 | 224*224*64=3.2M | (3*3*64)*64 = 36,864 |
| POOL2 | 112*112*64=800K | 0 |
| CONV3-128 | | |
| CONV3-128 | | |
| POOL2 | 56*56*128=400K | 0 |
| CONV3-256 | | |
| CONV3-256 | 56*56*256=800K | (3*3*256)*256 = 589,824 |
| CONV3-256 | | |
| CONV3-256 | | |
| POOL2 | | 0 |
| CONV3-512 | 28*28*512=400K | (3*3*256)*512 = 1,179,648 |
| CONV3-512 | | |
| CONV3-512 | 28*28*512=400K | |
| CONV3-512 | | |
| POOL2 | | 0 |
| CONV3-512 | | |
| CONV3-512 | | |
| CONV3-512 | | |
| CONV3-512 | | |
| POOL2 | | 0 |
| FC | 4096 | |
| FC | 4096 | 4096*4096 = 16,777,216 |
| FC | 1000 | |
| TOTAL | | |

Table 1: VGG19 memory and weights

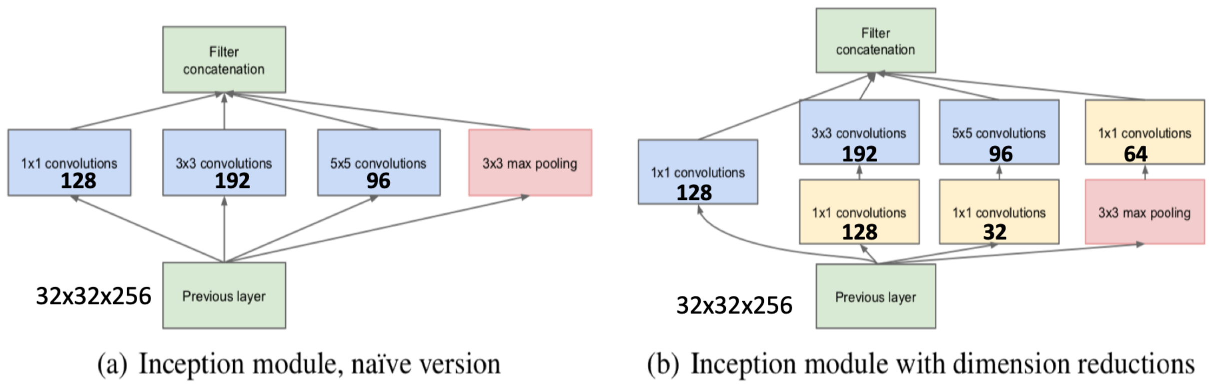

Figure 1: Two types of inception module with number of filters and input size for calculation in Question 3.4(b) and 3.4(c).

*References:*

- (Alexnet) Alex Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. Paper available at https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- (VGG) Karen Simonyan et al. Very Deep Convolutional Networks for Large-scale Image Recognition. Paper available at https://arxiv.org/pdf/1409.1556.pdf
- (Googlenet) Christian Szegedy et al. Going deeper with convolutions. Paper available at https://arxiv.org/pdf/1409.4842.pdf

## Problem 4 - *Batch Augmentation, Cutout Regularization* 20 points

**The training in this question takes substantial time. You need to start early to meet the deadline.**
In this problem we will be achieving large-batch SGD using batch augmentation techniques. In batch augmentation instances of samples within the same batch are generated with different data augmentations. Batch augmentation acts as a regularizer and an accelerator, increasing both generalization and performance scaling. One such augmentation scheme is using Cutout regularization, where additional samples are generated by occluding random portions of an image.

1. Explain cutout regularization and its advantages compared to simple dropout (as argued in the paper by DeVries et al) in your own words. Select any 2 images from CIFAR10 and show how does these images look after applying cutout. Use a square-shaped fixed size zero-mask to a random location of each image and generate its cutout version. Refer to the paper by DeVries et al (Section 3) and associated github repository. (2+4)
2. Using CIFAR10 datasest and Resnet-44 we will first apply simple data augmentation as in He et al. (look at Section 4.2 of He et al.) and train the model with batch size 64. Note that testing is always done with original images. Plot validation error vs number of training epochs. (4)

3. Next use cutout for data augmentation in Resnet-44 as in Hoffer et al. and train the model and use the same set-up in your experiments. Plot validation error vs number of epochs for different values of $M$ (2,4,8,16,32) where $M$ is the number of instances generated from an input sample after applying cutout $M$ times effectively increasing the batch size to $M \cdot B$, where $B$ is the original batch size (before applying cutout augmentation). You will obtain a figure similar to Figure 3(a) in the paper by Hoffer et al. Also compare the number of epochs and wallclock time to reach 94% accuracy for different values of M. Do not run any experiment for more than 100 epochs. If even after 100 epochs of training you did not achieve 94% then just report the accuracy you obtain and the corresponding wallclock time to train for 100 epochs. *Before attempting this question it is advisable to read paper by Hoffer et al. and especially Section 4.1.* (5+5)

You may reuse code from github repository associated with Hoffer et al. work for answering part 2 and 3 of this question.
*References:*

- DeVries et al. Improved Regularization of Convolutional Neural Networks with Cutout.
  Paper available at https://arxiv.org/pdf/1708.04552.pdf
  Code available at https://github.com/uoguelph-mlrg/Cutout
- Hoffer et al. Augment your batch: better training with larger batches. 2019
  Paper available at https://arxiv.org/pdf/1901.09335.pdf
  Code available at https://github.com/eladhoffer/convNet.pytorch/tree/master/models
- He et al. Deep residual learning for image recognition.
  Paper available at https://arxiv.org/abs/1512.03385