COMSE 6998 section 012
PRACT DEEP LEARNING SYS PERF
Instructor: Parijat Dube
Due: Nov 11, 2019

Homework 3

Problem 1 - *Horovod scaling using Amazon DLAMI* 15 points

In this problem we will compare distributed training scaling using Horovod on Amazon DLAMI. For this question you need to read the blog "Tensorflow with Horovod" (link provided below) and will need access to Amazon EC2. Follow instructions at the blog on launching training jobs using Horovod. You will use Resnet50 with synthetic data and run each training job for 25 epochs. Your Amazon DLAMI ships with an example script to train a model with synthetic data.

- *Single-node, multi-gpu training*: Train using a single AMI with 1, 2, 4, and 8 GPUs. You will be running 4 training jobs.
  1. Plot average throughput (speed) vs number of nodes and total runtime vs number of jobs. (4)
  2. Take the average throughput with 1 GPU as the base, plot the scaling of average throughput with GPUs. How does Horovod scaling compare with ideal scaling (i.e., with N GPUs throughput is N times the throughput with a single GPU) ? (4)
  3. What is the effective batch size for each training job ? Does Horovod employ strong or weak scaling ? (2)
- *Multi-node, multi-gpu training* : Training using two AMIs each with 8 GPUs. So effectively you have 16 GPUs. You will be running 1 training job with 16 GPUs. Measure the average througphput and training time with 16 GPUs. How does this compare with 8 GPUs case from part 1 ? (5)

References

- Tensorflow with Horovod.
  Available at https://docs.aws.amazon.com/dlami/latest/devguide/tutorial-horovod-tensorflow.html
- Horovod github repo.
  Available at https://github.com/horovod/horovodwhy-not-traditional-distributed-tensorflow

Problem 2 - *TF 2.0, tensorflow.distribute.strategy, Strong and Weak Scaling* 30 points

In this problem we will compare strong scaling and weak scaling in distributed training using tensorflow.distribute.strategy in Tensorflow 2.0. **tf.distribute.Strategy** is a TensorFlow API to distribute training across multiple GPUs, multiple machines or TPUs. In strong scaling, each worker computes with (batch size/# workers) training examples whereas in weak scaling, the effective batch size of SGD grows as the number of workers increases. For example, in strong scaling, if the batch size with 1 worker is 256, with 2 workers it will be 128 per worker, with 4 workers it will be 64 per worker, thus keeping the effective batch size at 256. In weak scaling, if the batch size with 1 worker is 64, with 2 workers it will be still be 64 per worker (with an effective batch size of 128 with 2 workers), thus the effective batch size increases linearly with number of workers. So the amount of compute per worker decreases in strong scaling whereas with weak scaling it remains constant.

Using FashionMNIST dataset and Resnet50 you will run distributed training using tensorflow.distribute.strategy and compare strong and weak scaling scenarios. Using an effective batch size of 256 you will run training jobs with 1,2,4,8,16 learners (each learner is a K80 GPU). For 8 or less number of learners, all the GPUs can be allocated on the same node (GCP povides 8 K80s on one node). You will run each training job for 10 epochs and measure average throughout, training time, and training cost. In total, you will be running 10 training jobs, 5 (with 1,2,4,8,16 GPUs) for weak scaling and 5 for strong scaling. For single node (worker) training using multiple GPUs you will use **tf.distribute.MirroredStrategy** with default all-reduce. For training with two or more workers you will use **tf.distribute.experimental.MultiWorkerMirroredStrategy** with CollectiveCommunication.AUTO.

1. Plot throughput vs number of learners for weak and strong scaling. (5)
2. Plot training time vs number of learners for weak and strong scaling. (5)
3. Plot training cost vs number of learners for weak and strong scaling. The training cost can be estimated using GPU per unit hour cost and the training time. (2)
4. For weak scaling, calculate scaling efficiency defined as the increase in time to finish one iteration at a learner as the number of learners increases. Show the plot of scaling efficiency vs number of learners for weak scaling. (5)
5. MirroredStrategy uses NVIDIA NCCL (tf.distribute.NcclAllReduce) as the default all-reduce. Change this to tf.distribute.HierarchicalCopyAllReduce and tf.distribute.ReductionToOneDevice and compare throughput of the three all-reduce implementations. You will be doing this for 1,2,4,8 GPUs single-node training. So you will be running 8 new training jobs (4 with HierarchicalCopyAllReduce and 4 with ReductionToOneDevice). For NcclAllReduce you can reuse results from part 1 of the question. (8)
6. Change MultiWorkerMirroredStrategy to use CollectiveCommunication.NCCL and CollectiveCommunication.RING and repeat the experiment with 2 nodes. Yow will be running two new training jobs (one with RING and one with NCCL). For AUTO you can reuse throughput from part 1 of the question. Compare the throughput of the three all-reduce methods (AUTO, NCCL, RING) ? Does AUTO gives the best throughput ? (5)

References:

- Tensorflow Blog. Distributed Training with Tensorflow.

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.

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. (10)
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 wit filter of size $(NF - N + 1) \times (NF - N + 1)$. Use this to calculate the receptive field of 3 filters of size 5x5. (5)

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

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. (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. (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)

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
- CS231n Convolutional Neural Networks for Visual Recognition.

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

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