HOROVOD: FAST AND EASY DISTRIBUTED DEEP LEARNING IN TENSORFLOW

AUTHORS: ALEXANDER SERGEEV AND MIKE DEL BALSO

UBER TECHNOLOGIES, INC
OUTLINE

• MOTIVATION

• BEFORE HOROVOD – DISTRIBUTED APPROACHES

• HOROVOD

1. WHAT IS CO-DESIGN?

2. HOW MANY HAVE WORKED WITH HOROVOD?

3. TRIED DEEP-LEARNING ON DISTRIBUTED WITHOUT CO-DESIGN FRAMEWORK?
MOTIVATION

• AT UBER:
  • MODELS ARE SMALL – FIT ON ONE SERVER
  • BUT THE TRAINING DATA GREW (~1+ WEEK TRAINING TIME)
  • THEY HAVE SEVERAL DIFFERENT TEAMS BUILDING VARIOUS DNN MODELS

• WHAT POTENTIAL SOLUTIONS:
  • MODELS ARE SMALL + HUGE TRAINING DATA: ?
  • EASE TO CONVERT ONE NODE TRAINING MODEL TO DISTRIBUTED MODEL: ?
MOTIVATION

• AT UBER:
  • MODELS ARE SMALL – FIT ON ONE SERVER
  • BUT THE TRAINING DATA GREW (~1+ WEEK TRAINING TIME)
  • THEY HAVE SEVERAL DIFFERENT TEAMS BUILDING VARIOUS DNN MODELS

• WHAT POTENTIAL SOLUTIONS:
  • MODELS ARE SMALL + HUGE TRAINING DATA: DATA PARALLELISM
  • EASE TO CONVERT ONE NODE TRAINING MODEL TO DISTRIBUTED MODEL: CO-DESIGN
MOTIVATION

NEED FOR DISTRIBUTED ML

TENSORFLOW STANDARD DISTRIBUTED ML LIBRARY

(workers, parameter servers, tf.server(), tf.clusterspec(), tf.train.syncreplicasoptimizer(), and tf.train.replicas_device_setter())

• SUBTLE, HARD TO-DIAGNOSE BUGS
• FIXING THEM NEEDS A STEEP LEARNING CURVE
• DELAYING DEVELOPMENT AND NOT ADDING TO MODEL ACCURACIES
• YET NOT COMPUTATIONALLY EFFICIENT – WHY?
lost about half of our resources due to communication overhead when training on 128 GPUs.
MOTIVATION

NEED FOR CO-DESIGN

• PROBLEMS INVOLVED IN DISTRIBUTED ML (NEED FOR CO-DESIGN)
  • THE TRAINING LIBRARY MUST SUPPORT INTER-GPU/CPU COMMUNICATION (NEGLIGIBLE TO SIGNIFICANT COMMUNICATION OVERHEAD)
  • THE USER MUST MODIFY TRAINING CODE TO CATER TO INTER-GPU/CPU COMMUNICATION (SIGNIFICANT OR MINIMAL MODIFICATION)

• HOROVOD (OPEN-SOURCE LIBRARY FOR FASTER & EASIER DISTRIBUTED TRAINING)
  • EFFICIENT INTER-GPU COMMUNICATION VIA RING REDUCTION
  • REQUIRES ONLY A FEW LINES OF MODIFICATION TO USER CODE (THEY STATE 4 MAJOR CHANGES TO SINGLE GPU/CPU CODE)
HOROVOD

• **AN OPEN-SOURCE COMPONENT** OF UBER’S MICHELANGELO FRAMEWORK: “an internal ml-as-a-service platform that democratizes machine learning and makes it easy to build and deploy these systems at scale”.

• “THE PRIMARY MOTIVATION FOR THIS PROJECT IS TO MAKE IT EASY TO TAKE A SINGLE-GPU TRAINING SCRIPT AND SUCCESSFULLY SCALE IT TO TRAIN ACROSS MANY GPUS IN PARALLEL”. this has two aspects:
  • Ease of implementation and execution: how much modification does one have to make to a program to make it distributed, and how easy is it to run it?
  • Performance: how much faster would it run in distributed mode?

• KEY IMPLEMENTATIONS FROM
  • FACEBOOK’S DATA PARALLELISM APPROACH - Accurate, Large Minibatch SGD: Training Imagenet In 1 Hour (Caffe2)
  • BAIDU’S RING-ALLREDUCE APPROACH - https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/
HOW TO UTILIZE MORE GPU?

• FACEBOOK’S DATA PARALLEL APPROACH (TRAINED RESNET-50 NETWORK IN **ONE HOUR** ON 256 GPUS) - “large-scale distributed training can have an enormous impact on model developer productivity”

• THE DATA-PARALLEL DISTRIBUTED TRAINING PARADIGM:
  1. RUN MULTIPLE COPIES OF THE TRAINING SCRIPT (EACH NODE):
     (A) READS A CHUNK OF THE DATA
     (B) RUNS IT THROUGH THE MODEL
     (C) COMPUTES MODEL UPDATES (GRADIENTS)
  2. AVERAGE GRADIENTS AMONG THOSE MULTIPLE COPIES
  3. UPDATE THE MODEL
  4. REPEAT (FROM STEP 1A)
FACEBOOK’S DATA PARALLEL APPROACH – AVERAGE GRADIENTS

- STANDARD DISTRIBUTED TENSORFLOW - **PARAMETER SERVER** APPROACH - AVERAGING GRADIENTS
- PROCESS ROLE: A WORKER (PROCESS THE TRAINING DATA, COMPUTE GRADIENTS) OR A PARAMETER SERVER (AVERAGES GRADIENTS AND REDISTRIBUTES).

Performance is improved **BUT…?**

a) **WORKER: PARAMETER SERVER**
b) **TF DISTRIBUTED EXECUTION COMPLEXITY**
• IDENTIFYING THE RIGHT RATIO OF WORKER TO PARAMETER SERVERS
  • HIGH RATIO: NETWORKING OR COMPUTATIONAL BOTTLENECK
  • LOW RATIO: SATURATE NETWORK INTERCONNECTS

• HANDLING INCREASED TENSORFLOW PROGRAM COMPLEXITY (STEEP LEARNING CURVE)
  • MANUALLY START EACH WORKER AND PARAMETER SERVER
  • PASS AROUND SERVICE DISCOVERY INFORMATION (HOSTS AND PORTS) OF ALL THE WORKERS AND PARAMETER SERVERS,
  • MODIFY THE TRAINING PROGRAM TO CONSTRUCT TF.SERVER() WITH AN APPROPRIATE TF.CLUSTERSPEC()
  • CHECK ALL THE OPERATIONS WERE PLACED APPROPRIATELY USING TF.TRAIN.DEVICE_REPLICA_SETTER()

• HTTPS://COLAB.RESEARCH.GOOGLE.COM/GITHUB/TENSORFLOW//DOCS/BLOB/MASTER/SITE/EN/TUTORIALS/DISTRIBUTE/MULTI_WORKER_WITH_KERAS.IPYNB
AllReduce to Ring to baidu’s ring-allreduce
Two steps:
a) A scatter-reduce - the GPUs will exchange data such that every GPU ends up with a chunk of the final result
b) A allgather - the GPUs will exchange those chunks such that all GPUs end up with the complete final result.

Figure 4: The ring-allreduce algorithm allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server.
EACH COMMUNICATION - A NODE SENDS AND RECEIVES CHUNKS OF THE DATA BUFFER

RECEIVED VALUES ARE ADDED TO THE VALUES IN THE NODE’S BUFFER
CHUCK OF FINAL RESULTS
allgather: RECEIVED VALUES REPLACE THE VALUES HELD IN THE NODE'S BUFFER
Total no. of communications per node?
What new potential problems?
Total no. of communications per node? N nodes communicates with two of its peers $2 \times (N - 1)$ times Independent of no. of GPUs

What new potential problems? IS ALGORITHM IS BANDWIDTH-OPTIMAL? if the buffer is large enough, it will optimally utilize the available network
HOROVOD IMPLEMENTATION

• STARTING BUILDING ON TOP OF BAIDU RESEARCH – TENSORFLOW-ALLREDUCE
• STAND ALONE LIBRARY COMPATIBLE WITH DIFFERENT VERSIONS OF TENSORFLOW
• REPLACE BAIDU’S RING-ALLREDUCE WITH NCCL LIBRARIES FOR ALL-REDUCE
  • NCCL- NVIDIA COLLECTIVE COMMUNICATION LIBRARY
• MODELS FIT ON SINGLE SERVER (MULTIPLE GPUS) – ORIGINALLY ONLY ONE GPU
• API CHANGES – BASED IN INTERNAL FEEDBACK:
  • A BROADCAST OPERATION - CONSISTENT INITIALIZATION OF THE MODEL ON ALL WORKERS.
  • NUMBER OF OPERATIONS TO MAKE SINGLE GPU PROGRAM INTRO DISTRIBUTED - 4
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()  # 1. initializes Horovod

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()  # 2. assigns a GPU to each of the TensorFlow processes.
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)  # Single GPU code

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)  # 3. wraps any regular TensorFlow optimizer with Horovod optimizer which takes care of averaging gradients using ring-allreduce.

# Add hook to broadcast variables from rank 0 to all other processes during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]  # 4

# Make training operation
train_op = opt.minimize(loss)  # Single GPU code
# The MonitoredTrainingSession takes care of session initialization, 
# restoring from a checkpoint, saving to a checkpoint, and closing 
# when done or an error occurs.

```python
with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
    config=config,
    hooks=hooks) as mon_sess:

    while not mon_sess.should_stop():
        # Perform synchronous training.
        mon_sess.run(train_op)
```

4. `hvd.BroadcastGlobalVariablesHook(0)` broadcasts variables from the first process to all other processes to ensure consistent initialization. If the program does not use `MonitoredTrainingSession`, users can run the `hvd.broadcast_global_variables(0)` operations instead.

```bash
```

WHAT DOES THIS DO?
The `MonitoredTrainingSession` takes care of session initialization, restoring from a checkpoint, saving to a checkpoint, and closing when done or an error occurs.

```python
with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
config=config,
hooks=hooks) as mon_sess:
    while not mon_sess.should_stop():
        # Perform synchronous training.
        mon_sess.run(train_op)
```

4. `hvd.BroadcastGlobalVariablesHook(0)` broadcasts variables from the first process to all other processes to ensure consistent initialization. If the program does not use `MonitoredTrainingSession`, users can run the `hvd.broadcast_global_variables(0)` operations instead.

```
```

WHAT DOES THIS DO?

DISTRIBUTES train.py TO FOUR NODES AND RUNS IT ON FOUR GPUs PER NODE
WHAT IF TENSORS ARE NOT LARGE ENOUGH? - RING-ALLREDUCE

• RING-ALLREDUCE UTILIZES THE NETWORK IN AN OPTIMAL WAY IF THE TENSORS ARE LARGE ENOUGH – what if they are small? “CAN MULTIPLE TINY TENSORS COULD BE FUSED TOGETHER BEFORE PERFORMING RING-ALLREDUCE ON THEM?”

• POTENTIAL SOLUTION - TENSOR FUSION, AN ALGORITHM THAT FUSES TENSORS TOGETHER BEFORE WE CALL HOROVOD’S RING-ALLREDUCE.

1. Determine which tensors are ready to be reduced. Select the first few tensors that fit in the buffer.

2. Allocate a fusion buffer if it was not previously allocated (default fusion buffer size is 64mb)

3. Copy data of selected tensors into the fusion buffer.

4. Execute the allreduce operation on the fusion buffer.

5. Copy data from the fusion buffer into the output tensors.

6. Repeat until there are no more tensors to reduce in the cycle
• 65 PERCENT IMPROVEMENT WITH UNOPTIMIZED TCP/IP

Figure 6: A comparison of images processed per second with standard distributed TensorFlow and Horovod when running a distributed training job over different numbers of NVIDIA Pascal GPUs for Inception V3 and ResNet-101 TensorFlow models over 25GbE TCP.
Figure 7: A comparison of the images processed per second of the Horovod over plain 25GbE TCP and the Horovod with 25GbE RDMA-capable networking when running a distributed training job over different numbers of NVIDIA Pascal GPUs for Inception V3, ResNet-101 and VGG-16.
RESOURCES

- **GIT:** [HTTPS://GITHUB.COM/HOROVOD/HOROVOD](HTTPS://GITHUB.COM/HOROVOD/HOROVOD)
- **OSC:** [HTTPS://WWW.OSC.EDU/RESOURCES/AVAILABLE_SOFTWARE/SOFTWARE_LIST/HOROVOD](HTTPS://WWW.OSC.EDU/RESOURCES/AVAILABLE_SOFTWARE/SOFTWARE_LIST/HOROVOD)
- **HTTPS://DEVELOPER.NVIDIA.COM/NCCL**