DISTRIBUTED DEEP NEURAL NETWORKS
INTRODUCTION
INTRODUCTION

DEEP LEARNING APPLICATIONS

COMPUTER VISION
- Image Classification
- Object Detection

SPEECH & AUDIO
- Voice Recognition
- Language Translation

NATURAL LANGUAGE PROCESSING
- Recommendation Engines
- Sentiment Analysis

source: Nvidia
INTRODUCTION

GPUS ARE GETTING FASTER

Training ImageNet to accuracy (90 epochs) with ResNet-50

Training OpenNMT to accuracy (13 epochs)

source: Nvidia
INTRODUCTION

COMPUTATIONAL COMPLEXITY OF NEURAL NETWORKS IS EXPLODING

7 ExaFLOPS
60 Million Parameters

2015 - Microsoft ResNet
Superhuman Image Recognition

20 ExaFLOPS
300 Million Parameters

2016 - Baidu Deep Speech 2
Superhuman Voice Recognition

100 ExaFLOPS
8700 Million Parameters

2017 - Google Neural Machine Translation
Near Human Language Translation

source: Nvidia
PROBLEM DECOMPOSITION
The data is partitioned across tasks and each task only works on its portion of the data.

- SPMD (single program, multiple data)
The focus is on partitioning the computations rather than the data. The problem itself is decomposed and each task performs a portion of the overall work.

Typically used when pieces of data require different processing times.
DISTRIBUTED DEEP NEURAL NETWORKS
DISTRIBUTED DEEP NEURAL NETWORKS

A NEURAL NETWORK
Each strategy is straightforward; partition the work of the batch across different machines.

Different machines have a complete copy of the model.

Each machine simply gets a different portion of the batch, and results from each are combined.

(source: skymind)
Scaling the performance of data parallelism requires increasing the “effective batch size”.

The increased batch size can result in a decrease of the model’s final accuracy.

Source: Skymind
This strategy divides the work according to the neurons in each layer.

Different machines in the distributed system are responsible for the computations in different parts of a single network.

For example, each layer in the neural network may be assigned to a different machine.
While model parallelism can work well in practice, data parallelism is arguably the preferred approach for distributed systems and has been the focus of more research.

For one thing, implementation, fault tolerance and good cluster utilization is easier for data parallelism than for model parallelism.

Furthermore, model parallelism requires massive amount of communication.
HYBRID PARALLELISM

- The combination of multiple parallelism schemes can overcome the drawbacks of each scheme.

source: skymind
DISTRIBUTED DEEP NEURAL NETWORKS

DISTRIBUTED TRAINING

Parameter Server

\[ w' = w - \eta \Delta w \]

Model Replicas

Data Shards
SYNCHRONOUS TRAINING
SYNCHRONOUS TRAINING

SYNCHRONOUS ALGORITHM

1. Each worker computes gradients on its part of the data.

2. Average gradients from all workers.

3. Update the model.
SYNCHRONOUS TRAINING

SYNCHRONOUS SCHEME

source: “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”
SYNCHRONOUS TRAINING

RING-ALLREDUCE

Arrays Being Summed

<table>
<thead>
<tr>
<th>GPU 0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_0</td>
<td>b_0</td>
<td>c_0</td>
<td>d_0</td>
<td>e_0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_2</td>
<td>b_2</td>
<td>c_2</td>
<td>d_2</td>
<td>e_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU 3</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_3</td>
<td>b_3</td>
<td>c_3</td>
<td>d_3</td>
<td>e_3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU 4</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_4</td>
<td>b_4</td>
<td>c_4</td>
<td>d_4</td>
<td>e_4</td>
</tr>
</tbody>
</table>

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

RING-ALLREDUCE

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

RING-ALLREDUCE

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

RING-ALLREDUCE

source: http://andrew.gibiansky.com/
**SYNCHRONOUS TRAINING**

**RING-ALLREDUCE**

- **GPU 0**
  - $a_0$
  - $b_2 + b_1 + b_3 + b_4 + b_0$
  - $c_3 + c_2 + c_4 + c_0$
  - $d_4 + d_3 + d_0$
  - $e_0 + e_4$

- **GPU 1**
  - $a_1 + a_0$
  - $b_1$
  - $c_3 + c_2 + c_4 + c_0 + c_1$
  - $d_4 + d_3 + d_0 + d_1$
  - $e_0 + e_4 + e_1$

- **GPU 2**
  - $a_1 + a_0 + a_2$
  - $b_2 + b_1$
  - $c_2$
  - $d_4 + d_3 + d_0 + d_1 + d_2$
  - $e_0 + e_4 + e_1 + e_2$

- **GPU 3**
  - $a_1 + a_0 + a_2 + a_3$
  - $b_2 + b_1 + b_3$
  - $c_3 + c_2$
  - $d_3$
  - $e_0 + e_4 + e_1 + e_2 + e_3$

- **GPU 4**
  - $a_1 + a_0 + a_2 + a_3 + a_4$
  - $b_2 + b_1 + b_3 + b_4$
  - $c_3 + c_2 + c_4$
  - $d_4 + d_3$
  - $e_4$

*source: http://andrew.gibiansky.com/*
SYNCHRONOUS TRAINING

RING-ALLREDUCE

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

RING-ALLREDUCE

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

RING-ALLREDUCE

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

RING-ALLREDUCE EXAMPLE

source: https://eng.uber.com/horovod/
SYNCHRONOUS TRAINING

SPEED-UP

Data Parallelism Speed-up with Ring Allreduce

source: http://andrew.gibiansky.com/
SYNCHRONOUS TRAINING

SPEED-UP

Training with synthetic data on NVIDIA® Pascal™ GPUs

source: https://eng.uber.com/horovod/
ASYNCHRONOUS TRAINING
ASYNCHRONOUS TRAINING

ASYNCHRONOUS SCHEME

source: “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”
ASYNCHRONOUS TRAINING

GRADIENT STALENESS

- ASGD suffers from **gradient staleness**: gradients sent by workers are often based on parameters that are older than the master’s (parameter server) current parameters.
This gradient staleness is a major obstacle to scaling ASGD since it grows as we increase the number of workers, which decreases gradient accuracy, and ultimately reduces the accuracy of the trained model.
Adjust the learning rate to the staleness magnitude.

**Softsync** - Instead of updating the parameters immediately, the master waits to collect a number of updates from any of workers, and only then updates the parameters.
PARAMETER SERVER
If one parameter server is used, it will likely become a networking or computational bottleneck.
DECENTRALIZED TRAINING
One bottleneck of centralized algorithms lies on high communication cost to the parameter server.

No centralized parameter server is present in the system. Instead a peer to peer communication is used to transmit model updates between workers.