Yet Another Accelerated SGD: ResNet-50 Training on ImageNet in 74.7 seconds

Yamazaki et al. (Fujitsu Laboratories Ltd)

Presented By: Piyush Chawla
Overview

- Trained ResNet-50 ImageNet in **74.7** seconds

- Using **81,920** mini-batch size and 2,048 GPUs

- AI Bridging Cloud Infrastructure (ABCI) and MXNet deep learning framework (self optimized)
Data Parallelism & Challenges

Problem: Training on large datasets takes time (several hours to days). Need to increase minibatch size.

Solution -> Data Parallelism: Make multiple copies of the same model on several GPUs, use inter server communication (allreduce) to share gradient information.

Challenges: (1.) Communication overhead (allreduce operation) and (2.) Large minibatch are difficult to train.
Related Work

**Improve Allreduce**
- Image Classification at Supercomputer Scale. Ying et al.

**Improve other communication overhead**

**Tricks to make SGD work on large mini-batch**
- Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. Goyal et al.
Motivation

**Goal:** Train deep learning models faster by increasing minibatch size using data parallelism.

**Solution:**
1. Communication overhead in data parallelism.
2. Large mini batch size is harder to train. Use existing methods.
1. Warm-up: Need high learning rate because of smaller updates. Training with high learning rate is unstable. Use warm-up to raise learning rate gradually.

2. Layerwise Adaptive Rate Scaling (LARS): Adjust learning rate of each layer differently.

3. Label smoothing: Use soft labels (using noise distribution) instead of hard labels (one-hot encodings)
MXNet Optimizations

MXNet: Deep learning Framework written in C++ and cuda (and many language bindings.)

They found and optimized the bottlenecks in MXNet to improve the training throughput.

1. DNN model initialization: Use same seed across all the processes. Default method is to use broadcasting.
2. Batch Norm on GPU: Created special GPU kernel to do batch norm in parallel across model layers.
Communication (Allreduce) Optimizations

1. Communication Data Size: Reduce the number of allreduce operations. Gather gradients from several layers and do allreduce on memory chunks that are several MBs big.

2. Communication Scheduling: Allreduce is started before the complete forward pass if there are enough gradients. Layers are statistically grouped together to ensure same layers are communicated across processes.
Experiment Environment

A single ABCI node

- 2 CPUs (Xeon Gold 6148)
- 4 GPUs (NVIDIA Tesla V100 SXM2) with NVLink
## Experiment Results

<table>
<thead>
<tr>
<th></th>
<th>Batch Size</th>
<th>Processor</th>
<th>DL Library</th>
<th>Time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>He et al. [1]</td>
<td>256</td>
<td>Tesla P100 (\times 8)</td>
<td>Caffe</td>
<td>29 hours</td>
<td>75.3 %</td>
</tr>
<tr>
<td>Goyal et al. [2]</td>
<td>8,192</td>
<td>Tesla P100 (\times 256)</td>
<td>Caffe2</td>
<td>1 hour</td>
<td>76.3 %</td>
</tr>
<tr>
<td>Smith et al. [3]</td>
<td>8,192 (\rightarrow 16,384)</td>
<td>full TPU Pod</td>
<td>TensorFlow</td>
<td>30 mins</td>
<td>76.1 %</td>
</tr>
<tr>
<td>Akiba et al. [4]</td>
<td>32,768</td>
<td>Tesla P100 (\times 1,024)</td>
<td>Chainer</td>
<td>15 mins</td>
<td>74.9 %</td>
</tr>
<tr>
<td>Jia et al. [5]</td>
<td>65,536</td>
<td>Tesla P40 (\times 2,048)</td>
<td>TensorFlow</td>
<td>6.6 mins</td>
<td>75.8 %</td>
</tr>
<tr>
<td>Ying et al. [6]</td>
<td>65,536</td>
<td>TPU v3 (\times 1,024)</td>
<td>TensorFlow</td>
<td>1.8 mins</td>
<td>75.2 %</td>
</tr>
<tr>
<td>Mikami et al. [7]</td>
<td>55,296</td>
<td>Tesla V100 (\times 3,456)</td>
<td>NNL</td>
<td>2.0 mins</td>
<td>75.29 %</td>
</tr>
<tr>
<td><strong>This work</strong></td>
<td><strong>81,920</strong></td>
<td><strong>Tesla V100 (\times 2,048)</strong></td>
<td><strong>MXNet</strong></td>
<td><strong>1.2 mins</strong></td>
<td><strong>75.08%</strong></td>
</tr>
</tbody>
</table>
Experiment Results

Training Throughput: 1.7 Million img/sec

Scalability: 77%
Experiment Results

Validation accuracy drops below 75.0 for minibatch sizes over 81920.
Conclusion and Critique

- Large minibatch size training (with ~25% improvement over SOTA).
- Optimizations for MXNet.

- Exact implementation details are missing.
- Which allreduce algorithm are they using?
- Missing results for vanilla MXNet (without their optimizations)
Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Goyal et al. (Facebook)

Presented By: Piyush Chawla
Overview

**Goal:** Make Distributed Synchronous SGD work for large mini-batch size (8192). Do not lose validation and training accuracy.

**Main Contributions:**
1. Linear scaling rule with new warm-up strategy for large minibatch.
2. Comprehensive experiments to support the claims.
SGD (Stochastic Gradient Descent)

Loss

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w).$$

Weight update rule

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in B} \nabla l(x, w_t).$$
Optimization Improvements

1. Linear Scaling Rule: When the minibatch size is multiplied by $k$, multiply the learning rate by $k$. This can be mathematically proven (with some strong assumptions).

However, these assumptions do not always hold.

- During the initial training process.
- For very large batch sizes.
2. Warmup: Use less aggressive learning rate at the beginning.

I. Constant warmup: Use low constant learning rate at the beginning (first 5 epochs) and move to higher value from then on.

II. Gradual warmup: Start with a base value and linearly move to the higher value. This gives better performance.
Optimization Improvements

3. Batch Normalization: Standardize the inputs to each layer. The activations from previous layers are made to have zero mean and standard deviation of one.

They prove that as long work load of each process (gpu) is same, batch normalization on large minibatch is equivalent to that on small minibatch. They fix 32 images workload for each gpu.
Common pitfalls in distributed SGD

1. Weight decay: Scaling the cross-entropy loss is not equivalent to scaling the learning rate.

\[ w_{t+1} = w_t - \eta \lambda w_t - \eta \frac{1}{n} \sum_{x \in B} \nabla \varepsilon(x, w_t). \]
2. Momentum correction: Apply momentum correction after changing learning rate if using the formulation on the right.

\[
\begin{align*}
  u_{t+1} &= mu_t + \frac{1}{n} \sum_{x \in B} \nabla l(x, w_t) \\
  w_{t+1} &= w_t - \eta u_{t+1}. \\

  v_{t+1} &= mv_t + \eta \frac{1}{n} \sum_{x \in B} \nabla l(x, w_t) \\
  w_{t+1} &= w_t - v_{t+1}.
\end{align*}
\]
Common pitfalls in distributed SGD

3. Gradient aggregation: Normalize the per-worker loss by total minibatch size $kn$ (8192), not per-worker size $n$ (32).
Common pitfalls in distributed SGD

4. Data shuffling: Use a single random shuffling of the training data (per epoch) that is divided amongst all the k workers.
Gradient Aggregation (allreduce)

Optimized allreduce implementation.

Three phases

1. Buffers from the 8 GPUs within a server summed into a single buffer.
2. Buffer results are shared and summed across all servers.
3. Final result broadcasted onto each GPU.

They used NVIDIA Collective Communication Library (NCCL)
Implemented two best known algorithms:

1. Recursive halving and doubling algorithm: $2\log_2(p)$ communication steps.
2. Bucket (ring) algorithm: $2(p-1)$ communication steps.

For buffer size of a million elements, they found halving/doubling algorithm outperform bucket algorithm by a factor of 3.
Hardware

- Facebook’s Big Basin GPU server
- 8 NVIDIA Tesla P100 GPUs with NVLink
- 3.2 TB NVMe SSDs
Experimental Setting

- 1000-way ImageNet
  - ~1.28 million training images
  - 50,000 validation images
  - Top-1 error
- Learning rate of $0.1 \times \frac{kn}{256}$, $n = 32$
- Baseline $k = 8$
- All results reported as average of 5 random runs
Validation Error

(a) no warmup  
(b) constant warmup  
(c) gradual warmup

<table>
<thead>
<tr>
<th></th>
<th>$k$</th>
<th>$n$</th>
<th>$kn$</th>
<th>$\eta$</th>
<th>top-1 error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline (single server)</td>
<td>8</td>
<td>32</td>
<td>256</td>
<td>0.1</td>
<td>23.60 ± 0.12</td>
</tr>
<tr>
<td>no warmup, Figure 2a</td>
<td>256</td>
<td>32</td>
<td>8k</td>
<td>3.2</td>
<td>24.84 ± 0.37</td>
</tr>
<tr>
<td>constant warmup, Figure 2b</td>
<td>256</td>
<td>32</td>
<td>8k</td>
<td>3.2</td>
<td>25.88 ± 0.56</td>
</tr>
<tr>
<td>gradual warmup, Figure 2c</td>
<td>256</td>
<td>32</td>
<td>8k</td>
<td>3.2</td>
<td>23.74 ± 0.09</td>
</tr>
</tbody>
</table>
Training Error v/s Minibatch Size

The training trajectory closely matches that of baseline for minibatch size <=8k (8192)
Minibatch Size v/s Error

- Error is close to baseline for minibatch size $\leq 8k$
- Same behavior observed for ImageNet-5k
Comparison with the first paper

Yamazaki et al.
Throughput

- ~90% scaling achieved
Conclusion

● Comprehensive empirical study for training with large minibatch size
● Discussion on common pitfalls
● Comprehensive experiment analysis
● Verified results on downstream tasks.