Taming Unbalanced Training Workloads in Deep Learning with Partial Collective Operations

Shigang Li, Tal Ben-Nun, Salvatore Di Girolamo, Torsten Hoefler
ETH Zurich

Dan Alistarh
IST Austria

PPoPP’20
Feb. 22-26, 2020
San Diego, CA, US
Deep learning training

The overall objective function:

\[ f(w) = \mathbb{E}_{\xi \sim D} F(w; \xi) \]

**Training**: optimize \( w \) to minimize \( f \) (using SGD).

\( w \) denotes the model parameters.
\( F \) is the loss function.
\( \xi \) is a data point sampled from a distribution \( D \).

Dataset

Model parallelism

P0
P1
P2
Deep learning training

The overall objective function:

$$f(w) = \mathbb{E}_{\xi \sim D} F(w; \xi)$$

- $w$ denotes the model parameters.
- $F$ is the loss function.
- $\xi$ is a data point sampled from a distribution $D$.

**Training**: optimize $w$ to minimize $f$ (using SGD).
Deep learning training

The overall objective function:

$$f(w) = \mathbb{E}_{\xi \sim D} F(w; \xi)$$

- \( w \) denotes the model parameters.
- \( F \) is the loss function.
- \( \xi \) is a data point sampled from a distribution \( D \).

**Training**: optimize \( w \) to minimize \( f \) (using SGD).

Dataset
Unbalanced training workloads

- **Load imbalance on application level**
  - Recurrent Neural Networks (RNN/LSTM/GRU)
  - Transformers

  *Challenge: stragglers dominate the performance.*

- **Load imbalance on system level**
  - Performance variability on multitenant cloud systems
  - System or network noise

Different types of RNNs

- (One input multiple outputs)
- (Multiple inputs one output)
- (Multiple inputs multiple outputs)
Many-to-one RNN for video classification

RNN: \( h_t = f_W(h_{t-1}, x_t) \)

Backward pass

\( \nabla L(w) \)

\( \nabla L(W_1) \)

\( \nabla L(W_2) \)

\( \nabla L(W_3) \)

\( \nabla L(W_T) \)

Workload is proportional to \( T \)

FC1

FC2

0.13

Playing Basketball

0.14

0.41

0.09

0.13

0.10

Playing Basketball

...
Workload statistics for video classification

Distributio**n**: $201 \sim 3,410 \text{ ms}$
**Mean**: $1,235 \text{ ms}$
**Standard deviation**: $706 \text{ ms}$

(a) Video length distribution for UCF101 dataset

**Distribution**: $29 \sim 1,776 \text{ frames}$
**Mean**: $187 \text{ frames}$
**Standard deviation**: $97 \text{ frames}$

(b) Runtime distribution for the mini-batches to train a LSTM model on P100
Transformer


Knowledge is power.

The workload is proportional to $input\_size \times output\_size$.

### Distribution
- Mean: 475 ms
- Standard deviation: 144 ms

Runtime distribution for the mini-batches to train a Transformer model (using WMT16) on P100

Knowledge is power.
Compared with imbalanced applications (e.g., LSTM, Transformer), the load imbalance on cloud servers is relatively light.
Deep learning training is robust

- **Allreduce**
- **Gossiping**
- **Gradients sparsification**
- **1-bit gradients quantization**
- **Hidden units dropout**
Eager-SGD to solve the load imbalance problem

(a) synch-SGD

(b) eager-SGD

Eager-SGD exploits the robustness of the training by allowing \textit{allreduce} on stale gradients.

Gossip-based SGDs

<table>
<thead>
<tr>
<th></th>
<th>Communication participants</th>
<th>Number of steps for update propagation</th>
<th>Consistency mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-PSGD [1]</td>
<td>2</td>
<td>$O(P)$</td>
<td>synchronous</td>
</tr>
<tr>
<td>AD-PSGD [2]</td>
<td>1</td>
<td>$O(\log P)$</td>
<td>asynchronous</td>
</tr>
<tr>
<td>eager-SGD</td>
<td>$P$</td>
<td>1</td>
<td>asynchronous</td>
</tr>
</tbody>
</table>
Partial Allreduce operations

- **Two phases:** the activation and the collective operation

- **Asynchronous execution:** an auxiliary thread would progress the execution (activation and collective) in the background.

- **Multiple initiators:** the same operation is only executed once even if we may have multiple initiators, i.e. multiple processes arrive at the same time.
Solo allreduce and majority allreduce

- Two variants: solo allreduce \[^3\] and majority allreduce.
- For solo, at least one process “actively” participates.
- For majority, a majority of processes must “actively” participate.

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Solo allreduce</th>
<th>Majority allreduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiator</td>
<td>The fastest process</td>
<td>A randomly specified process</td>
</tr>
<tr>
<td>Attributes</td>
<td>Wait-free</td>
<td>Wait for the randomly specified initiator</td>
</tr>
<tr>
<td>The expectation of the participants</td>
<td>(\Omega(1))</td>
<td>(\Omega(P/2))</td>
</tr>
</tbody>
</table>

Implementation eager-SGD based on Tensorflow

Customized distributed optimizer based on Tensorflow

Eager-SGD utilizes the execution engine of TF to exploit the parallelism in the computation DAG.
Execution of eager-SGD

1. Two processes and P1 is faster.
2. P1 finishes the calculation for the gradients of step $t$, and triggers partial-allreduce. P0 contributes NULL.
3. P0 finishes step $t$, and discovers partial-allreduce is already done. P0 copies the stale gradients to its send buffer.
4. P0 catches up P1 in step $t+1$. The stale gradients are combined with the latest gradients, and then commit to partial-allreduce.
Convergence of eager-SGD

- For a learning rate value

\[ \alpha \leq \min \left( \frac{\sqrt{\epsilon P}}{\sqrt{12L^2 \tau M^2 (P - Q)}}, \frac{\epsilon}{12M^2 L}, \frac{\sqrt{\epsilon P}}{\sqrt{4L\tau M^2 (P - Q)}} \right) , \]

eager-SGD converges after

\[ T = \Theta \left( \frac{f(w_0) - m}{\epsilon \alpha} \right) \]

iterations.

\[ T \geq \Theta \left( \frac{(f(w_0) - m) \sqrt{\tau (P - Q)}}{P \epsilon^{3/2}} \right) \]

- Note the dependence in \( \tau \) (staleness bound) and \( P-Q \) (the number of stale gradients) for iterations \( T \).
- Eager-SGD would converge slower if too many stale gradients are used.
Evaluation

- CSCS Piz Daint supercomputer.
- Cray Aries interconnected network.
- Cray MPICH 7.7.2 communication library.
- Each node contains a 12-core Intel Xeon E5-2690 CPU, and one NVIDIA Tesla P100 GPU.
- We compare eager-SGD with the allreduce-based synch-SGD (Horovod and Deep500), the asynchronous centralized SGD (TF parameter server), and the gossip SGDs (D-PSGD, SGP).

Table 1. Neural networks used for evaluation

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Models</th>
<th>Parameters</th>
<th>Train data size</th>
<th>Batch size</th>
<th>Epochs</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperplane regression</td>
<td>One-layer MLP</td>
<td>8,193</td>
<td>32,768 points</td>
<td>2,048</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>Cifar-10</td>
<td>ResNet-32 [21]</td>
<td>467,194</td>
<td>50,000 images</td>
<td>512</td>
<td>190</td>
<td>8</td>
</tr>
<tr>
<td>UCF101 [53]</td>
<td>Inception+LSTM [61]</td>
<td>34,663,525</td>
<td>9,537 videos</td>
<td>128</td>
<td>50</td>
<td>8</td>
</tr>
</tbody>
</table>
Hyperplane regression (light load imbalance)

- Eager-SGD (solo) achieves **1.50x**, **1.75x**, and **2.01x** speedup over synch-SGD (Deep500), respectively.

- The loss value is equivalent with synch-SGD (Deep500).

Synch-SGD vs eager-SGD for hyperplane regression using 8 GPUs. "synch/eager-SGD-200/300/400" represent 200/300/400 ms load imbalance injection for 1 out of 8 processes.
ResNet-50 on ImageNet (light load imbalance)

Synch-SGD vs eager-SGD for ResNet-50 on ImageNet using 64 GPUs. "synch/eager-SGD-300/460" represent 300/460 ms load imbalance injection for 4 out of 64 processes.

- Eager-SGD (solo) achieves **1.25x** and **1.29x** speedup over Deep500, respectively; **1.14x** and **1.27x** speedup over Horovod, respectively. Top-1 accuracy is almost equivalent (75.2% vs 75.8%).

- Eager-SGD (solo) achieves **2.64x**, **1.26x**, **1.17x** over aysnch-PS and gossip-based SGDs (D-PSGD, SGP) respectively.
LSTM on UCF101 (severe load imbalance)

Top-1 test accuracy and runtime for LSTM on UCF101 using 8 GPUs.

<table>
<thead>
<tr>
<th></th>
<th>eager-SGD (solo)</th>
<th>eager-SGD (majority)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup over Horovod</td>
<td>1.64x</td>
<td>1.27x</td>
</tr>
<tr>
<td>Top-1 test accuracy</td>
<td>60.6% on average, up to 70.4%</td>
<td>69.7% on average, up to 72.8%</td>
</tr>
</tbody>
</table>
Conclusion

1. Eager-SGD deals with the imbalanced workloads using partial allreduce operations.

2. Eager-SGD has two variants, solo and majority.

3. Solo allreduce is suitable for light load imbalance, while majority allreduce works for severe load imbalance.

4. For the future work, we will verify the idea of eager-SGD on model-averaging SGD algorithms.

Questions: shigang.li@inf.ethz.ch