Delayed Gradient Averaging: Tolerate the Communication Latency for Federated Learning

Ligeng Zhu, Hongzhou Lin, Yao Lu*, Yujun Lin, Song Han

Massachusetts Institute of Technology, Google*
Federated Learning Allows Training without Sharing

- **Security**: Data never leaves devices thus promises security and regularization.

- **Customization**: Models continually adapt to new data from the sensors.
There is huge gap between the network connection of conventional distributed training and federated learning.
Network Bottleneck in Federated Learning

- Bandwidth can be always improved by
  - Hardware upgrade
  - Gradient compression[1] and quantization[2]

- Latency is hard to improve because
  - Physical limits: Shanghai to Boston, even considering the speed of light, still takes 162ms.
  - Signal congestion: Urban office and home creates a lot of signal contention.

[1] Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training
[2] 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs
High Latency Slows Federated Learning

Within a Rack

Normalized Throughput

Higher training speed

Higher Latency

Network latency

Normalized Throughput

1us 1ms 10ms 100ms 500ms 1s
High Latency Slows Federated Learning

Within a Rack
Within a Data Center

Normalized Throughput

Network latency

0.00
0.25
0.50
0.75
1.00

Higher training speed

Higher latency

In cluster network latency does not affect training
High Latency Slows Federated Learning

Within a Rack | Same Data Center | Wireless
---|---|---
1us | 1ms | 10ms
100ms | 500ms | 1s

Normalized Throughput

Network latency

Higher training speed

In home wireless connection slows the training by certain margin.
High Latency Slows Federated Learning

Long-distance connection slows the training by a large margin.
Can existing distributed optimizations handle high latency?

NO
Conventional Algorithms Suffer from High Latency

Distributed Synchronous SGD

1. Sample and calculate $\nabla w_{(i,j)}$
2. Send $\nabla w_{(i,j)}$ to other nodes
3. Recv $\nabla w_{(i,j)}$ from other nodes
4. $\overline{\nabla w_{(i)}} = \frac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)}$
5. $w_{(i,j)} = w_{(i,j)} - \eta \overline{\nabla w_{(i)}}$

Latency increases

Local updates and communication are performed sequentially. Worker **has to wait the transmission finish** before next step.

$i$: iteration, $j$: work index, $x$: training data, $w$: model weights
Conventional Algorithms Suffer from High Latency

**Federated Averaging** [McMahan 16]

1. Sample and calculate $\nabla w_{(i,j)}$
2. If $i \mod K$:
   1. Send $\nabla w_{(i,j)}$ to other nodes
   2.Recv $\nabla w_{(i,j)}$ from other nodes
3. $G_i = \frac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)}$
3. Else
1. $G_i = \nabla w_{(i,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta G_i$

Increase $K$ (K=2 in the example) can **amortize the effect**, but the training still **slows when latency is high**.

- **Computation**
- **Communication**

$i$: iteration, $j$: work index, $x$: training data, $w$: model weights
**Conventional Algorithms Suffer from High Latency**

**Federated Averaging** [McMahan 16]

1. \( \nabla w_{(i,j)} = \frac{\partial F(x_{(i,j)}, y_{(i,j)}; w)}{\partial w} \)
2. If \( i \mod K \):
   1. Send \( \nabla w_{(i,j)} \) to others
3. \( G_i = \frac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)} \)
3. Else
   1. \( G_i = \nabla w_{(i,j)} \)
4. \( w_{(i,j)} = w_{(i,j)} - \eta G_i \)

**How to improve training throughput under high latency?**

**Pipeline computation and communication!**

Increase \( K \) can amortize the effect, but still, the training suffers from high latency.
Delayed Gradient Averaging

Delay Gradient Averaging [Ours]

1. Sample and calculate $\nabla w(i, j)$
2. If $i \bmod K == 0$
   1. Send fresh $\nabla w(i, j)$ to other nodes
3. If $i \bmod K == D$
   1. Delay the averaging to a later iteration.
   2. Send fresh $\nabla w(i, j)$ to other nodes
4. $w(i, j) = w(i, j) - \eta(\nabla w(i, j) - \nabla w(i-D, j) + \overline{\nabla w(i-D)})$

$i$: iteration, $j$: work index, $x$: training data, $w$: model weights
**Delayed Gradient Averaging**

**Delay Gradient Averaging [Ours]**

1. Sample and calculate $\nabla w_{(i,j)}$
2. If $i \mod K == 0$
   1. Send fresh $\nabla w_{(i,j)}$ to other nodes
3. If $i \mod K == D$
   1.Recv stale $\nabla w_{(i-D,j)}$ from other nodes
   2. $\nabla w_{(i-D)} = \frac{1}{J} \sum_{j=1}^{J} \nabla w_{(i-D,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)})$

**W/o delay:** all the local machines are blocked to wait for the synchronization to finish

**With delay:** Worker keep performing local updates while the parameters are in transmission.

$i$: iteration, $j$: work index, $x$: training data, $w$: model weights
Delayed Gradient Averaging

Delay Gradient Averaging [Ours]

1. Sample and calculate $\nabla w_{(i,j)}$
2. If $i \mod K == 0$
   1. Send fresh $\nabla w_{(i,j)}$ to other nodes
3. If $i \mod K == D$
   1.Recv stale $\nabla w_{(i-D,j)}$ from other nodes
   2. $\overline{\nabla w_{(i-D)}} = \frac{1}{j} \sum_{j=1}^{j} \nabla w_{(i-D,j)}$
4. $w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \overline{\nabla w_{(i-D)}})$

Communication is covered by computation.

As long as the transmission finishes within $D \times T_{\text{computation}}$ the training will not be blocked.

$i$: iteration, $j$: work index, $x$: training data, $w$: model weights
The Design of Correction Term

\[ w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Consider the 3rd iteration with \( D = 2 \)

\[ w_{(3,j)} = w_{(1,j)} - \eta (\nabla w_{(1,j)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

Local gradients
The Design of Correction Term

Current local gradients  Stale local gradients  Stale global gradients

\[ w_{(i,j)} = w_{(i,j)} - \eta \left( \nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)} \right) \]

Consider the 3rd iteration with \( D = 2 \)

\[ w_{(3,j)} = w_{(1,j)} - \eta \left( \nabla w_{(1,j)} + \nabla w_{(2,j)} + \nabla w_{(3,j)} \right) \]
The Design of Correction Term

Current local gradients

\[ w_{(i,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Stale global gradients

Stale local gradients

Consider the 3rd iteration with \( D = 2 \)

\[ w_{(3,j)} = w_{(1,j)} - \eta(\nabla w_{(1,j)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

\[ \nabla w_{(1)} \]
The Design of Correction Term

Current local gradients  Stale global gradients

\[ w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Stale local gradients

Consider the **3rd iteration** with \( D = 2 \)

\[ w_{(3,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

Replacing oldest local gradients with global averaged ones!
The Design of Correction Term

Current local gradients

\[ w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Stale local gradients

Stale global gradients

Consider the 4th iteration with \(D = 2\)

\[ w_{(3,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

\[ w_{(4,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2,j)} + \nabla w_{(3,j)} + \nabla w_{(4,j)}) \]
The Design of Correction Term

\[ w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Consider the 4th iteration with \( D = 2 \)

\[ w_{(3,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

\[ w_{(4,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2,j)} + \nabla w_{(3,j)} + \nabla w_{(4,j)}) \]
The Design of Correction Term

Current local gradients  
Stale global gradients

\[ w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Stale local gradients

Consider the \textbf{4th iteration} with \( D = 2 \)

\[ w_{(3,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

\[ w_{(4,j)} = w_{(1,j)} - \eta (\nabla w_{(1)} + \nabla w_{(2)} + \nabla w_{(3,j)} + \nabla w_{(4,j)}) \]
The Design of Correction Term

Current local gradients

\[ w_{(i,j)} = w_{(i,j)} - \eta (\nabla w_{(i,j)} - \nabla w_{(i-D,j)} + \nabla w_{(i-D)}) \]

Stale global gradients

Stale local gradients

\[ w_{(3,j)} = w_{(1,j)} - \eta (\overline{\nabla w_{(1)}} + \nabla w_{(2,j)} + \nabla w_{(3,j)}) \]

\[ w_{(4,j)} = w_{(1,j)} - \eta (\overline{\nabla w_{(1)}} + \overline{\nabla w_{(2)}} + \nabla w_{(3,j)} + \nabla w_{(4,j)}) \]

\[ w_{(i,j)} = w_{(1,j)} - \eta (\overline{\nabla w_{(1)}} + \ldots + \overline{\nabla w_{(i-D,j)}} + \overline{\nabla w_{(i-D+1,j)}} + \ldots + \overline{\nabla w_{(i,j)}}) \]

Only most recent \( D \) updates are local gradients.
The Design of Correction Term

Our DGA:

\[
W_{(i,j)} = W_{(1,j)} - \eta(\nabla W_{(1)} + \ldots + \nabla W_{(i-D,j)} + \nabla W_{(i-D+1,j)} + \ldots + \nabla W_{(i,j)})
\]

Vanilla Distributed SGD:

\[
W_{(i,j)} = W_{(1,j)} - \eta(\nabla W_{(1)} + \ldots + \nabla W_{(i-D,j)} + \nabla W_{(i-D+1,j)} + \ldots + \nabla W_{(i,j)})
\]

Usual training consists of >10k iterations, such divergence is small.

The divergence is bounded.
DGA Guarantees the Convergence

• Assumption 1: the loss function $F(w; x, y)$ is **Lipchitz smooth**

$$\nabla f_j(x) - \nabla f_j(y) \leq L \| x - y \| . \quad \forall x, y \in \mathbb{R}^d$$

• Assumption 2: **Bounded gradients and variances**

$$\mathbb{E}_{\zeta_j} \| \nabla F_j(w; \zeta_i) \|^2 \leq G^2, \forall w, \forall j, \quad \mathbb{E}_{\zeta_j} \| \nabla F_j(w; \zeta_j) - \nabla f_j(w) \|^2 \leq \sigma^2, \forall w, \forall j.$$  

The convergence rate of DGA is $O\left(\frac{\Delta + \sigma^2}{\sqrt{JN}} + \frac{Jd^2}{N}\right)$ (details in paper)

When $D < O(N^{\frac{1}{4}}J^{-\frac{3}{4}})$, **DGA converges as fast as original SGD** which is $O\left(\frac{\Delta + \sigma^2}{\sqrt{JN}}\right)$.

$(N$: iterations, $J$: number of machines)
# DGA Improves the Accuracy

<table>
<thead>
<tr>
<th>Paritions</th>
<th>FedAvg(k=5)</th>
<th>FedAvg(k=10)</th>
<th>FedAvg(k=20)</th>
<th>DGA(K=5,D=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.I.D</td>
<td>88.7</td>
<td>88.5</td>
<td>88.1</td>
<td><strong>88.6</strong></td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>48.2</td>
<td>47.2</td>
<td>43.9</td>
<td>48.0</td>
</tr>
</tbody>
</table>

|               | 1.0x        | 1.51x        | 2.05x        | 3.16x         |

DGA shows **negligible accuracy drop**.
## DGA Improves the Accuracy

<table>
<thead>
<tr>
<th>Paritions</th>
<th>FedAvg(k=5)</th>
<th>FedAvg(k=10)</th>
<th>FedAvg(k=20)</th>
<th>DGA(K=5,D=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR</td>
<td>88.7</td>
<td>88.5</td>
<td>88.1</td>
<td>88.6</td>
</tr>
<tr>
<td>I.I.D</td>
<td>1.0x</td>
<td>1.51x</td>
<td>2.05x</td>
<td>3.16x</td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>48.2</td>
<td>47.2</td>
<td>43.9</td>
<td>48.0</td>
</tr>
</tbody>
</table>

DGA shows **much better accuracy** on non I.I.D partitions.
## DGA Improves the Accuracy

<table>
<thead>
<tr>
<th>Paritions</th>
<th>FedAvg(k=5)</th>
<th>FedAvg(k=10)</th>
<th>FedAvg(k=20)</th>
<th>DGA(K=5,D=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR I.I.D</td>
<td>88.7</td>
<td>88.5</td>
<td>88.1</td>
<td>88.6</td>
</tr>
<tr>
<td></td>
<td>1.0x</td>
<td>1.51x</td>
<td>2.05x</td>
<td>3.16x</td>
</tr>
<tr>
<td>CIFAR Non-I.I.D</td>
<td>48.2</td>
<td>47.2</td>
<td>43.9</td>
<td>48.0</td>
</tr>
</tbody>
</table>

While producing higher accuracy, DGA also demonstrates **faster training speed** as it fully covers communication with computation.
DGA Improves the Accuracy

<table>
<thead>
<tr>
<th>Paritions</th>
<th>FedAvg(k=5)</th>
<th>FedAvg(k=10)</th>
<th>FedAvg(k=20)</th>
<th>DGA(K=5,D=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIFAR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.I.D</td>
<td>88.7</td>
<td>88.5</td>
<td>88.1</td>
<td>88.6</td>
</tr>
<tr>
<td></td>
<td>1.0x</td>
<td>1.51x</td>
<td>2.05x</td>
<td>3.16x</td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>48.2</td>
<td>47.2</td>
<td>43.9</td>
<td>48.0</td>
</tr>
<tr>
<td><strong>ImageNet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.I.D</td>
<td>76.6</td>
<td>76.5</td>
<td>76.2</td>
<td>76.4</td>
</tr>
<tr>
<td></td>
<td>1.0x</td>
<td>1.43x</td>
<td>1.81x</td>
<td>2.55x</td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>55.4</td>
<td>52.5</td>
<td>48.6</td>
<td>54.9</td>
</tr>
</tbody>
</table>
## DGA Improves the Accuracy

<table>
<thead>
<tr>
<th>Paritions</th>
<th>FedAvg(k=5)</th>
<th>FedAvg(k=10)</th>
<th>FedAvg(k=20)</th>
<th>DGA(K=5,D=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIFAR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.I.D</td>
<td>88.7</td>
<td>88.5</td>
<td>88.1</td>
<td>88.6</td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>48.2</td>
<td>47.2</td>
<td>43.9</td>
<td>48.0</td>
</tr>
<tr>
<td><strong>ImageNet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.I.D</td>
<td>76.6</td>
<td>76.5</td>
<td>76.2</td>
<td>76.4</td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>55.4</td>
<td>52.5</td>
<td>48.6</td>
<td>54.9</td>
</tr>
<tr>
<td><strong>Shakespeare</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.I.D</td>
<td>47.6</td>
<td>47.3</td>
<td>47.3</td>
<td>47.1</td>
</tr>
<tr>
<td>Non-I.I.D</td>
<td>36.9</td>
<td>34.3</td>
<td>30.1</td>
<td>36.3</td>
</tr>
</tbody>
</table>
Real-world Benchmark

We build a raspberry pi cluster to simulate real-world federated learning scenarios.

• Device: 8 x Raspberry Pi 4B+ Models
• Device OS: Debian 10
• Router: Netgear R6300v2
• Router OS: OpenWRT
When scaling the training to two devices, the normalized throughput is only 0.6, which is even slower than single device.
Even we set a larger value of $K$, the scalability is still less than 0.5 and not comparable with training throughput based on in-cluster networks.
Benchmark on Raspberry Pi Farms

Our proposed DGA demonstrates ideal scalability under high-latency network. The speedup on eight-device is about 7.1, which close to what conventional algorithms achieved inside a data center.
We design Delayed Gradient Averaging (DGA) that

• Delays averaging to a later iteration to tolerate high network latency

• New update formula to compensate the accuracy

We evaluate the algorithm’s

• Convergence and accuracy both theoretically and empirically.

• Training throughput under a real-world pi-cluster.