Running Sparse & Low-Precision Networks

Hai Li, Ph.D.
Center for Computational Evolutionary Intelligence
Duke University

Duke Chapel against a blue sky,
Duke campus, Durham, NC, USA
Rise and Decline of Neural Network

Convolutional Network
(1980s)

Dark period
(1990s)

- Serious problem: Vanishing gradient
- No benefits observed by adding more layers
- No high performance computing devices

Renaissance
(2006 ~ Present)


Machine Learning in Academia

Journal articles mentioning “deep learning” or “deep neural networks”

NIPS registrations growth

Source: Office of Science and Technology Policy/The White House

- 2015: 3755
- 2016: 5600
- 2017: 7850
Machine Learning in Industry

Machine learning is a core transformative way by which we are *rethinking everything* we are doing.

~ Sundar Pichai (CEO Google)

Number of software projects within Google that use key machine learning technologies:

Source: Bloomberg

10x increase in 3 years!

Machine Learning in the Market

Technology cycle - from PC, to smartphone, to artificial intelligence?

“Pure Play” Share Price Performance

Source: Bloomberg, Jefferies
Why is Deep Learning Hot Now?

<table>
<thead>
<tr>
<th>Application</th>
<th>Algorithm</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Data</td>
<td>ML Techniques</td>
<td>Computation Power</td>
</tr>
</tbody>
</table>

Numbers:
- 5 KB/record
- 500 KB/record
- 1000 KB/picture
- 5000 KB/song
- 5,000,000 KB/movie

Advances in algorithm innovation, including neural networks, leading to better accuracy in training models

- Transistor density doubles every 18 months
- Computation/kwh doubles every 18 months
- Cost/Gigabyte in 1995: $1000.00
- Cost/Gigabyte in 2015: $0.03
## Our Work on Deep Learning

<table>
<thead>
<tr>
<th>Application</th>
<th>Algorithm</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Data</td>
<td>ML Techniques</td>
<td>Computation Power</td>
</tr>
</tbody>
</table>

- Image Segmentation
- Invisible Shield on Gesture Security

- Efficient Algorithms for DNN deployment on the Cloud and the Edge

- FPGA-based Deep Learning Acceleration
- Learning on IBM TrueNorth Chip
- ReRAM-based Neuromorphic Computing
Outline

• Lightweight computing engine restricts the deployment of cumbersome DNNs on the edge
  – *Structurally sparse DNN*: simplify the “rocket” to lighten the load on the “engine”

• Communication bottleneck limits the scalability of distributed deep learning in the cloud
  – *TernGrad*: quantize gradients to reduce communication volume

• Novel architecture for future computing
  – *ReRAM-based neuromorphic chip*
Structurally Sparse Deep Neural Networks
Sparse Convolutional Neural Networks (SCNNs)

Significantly reduces the storage size of DNNs [1]
Good speedup with customized hardware [2]

Inefficiency of SCNNs

- Format: Sparse matrixes are formatted as Compressed Sparse Row (CSR)
- Acceleration library: cuSPARSE

W. Wen et. al. NIPS 2016

Speedup by SCNNs in GPUs
The same issue in CPUs

Non-structured sparsity

Structured sparsity

Inefficiency of SCNNs

W. Wen et. al. ICLR 2018
Structurally Sparse Deep Neural Networks

What is Structurally Sparse Deep Neural Networks (SSDNNs)?

Weights/connections are removed group by group.
A group can be a rectangle block, a row, a column or even the whole matrix.
In a nutshell, a group can be any form of a weight block, depending on what sparse structure you want to learn.

In CNNs, a group of weights can be a channel, a 3D filter, a 2D filter, a filter shape fiber (i.e., a weight column), and even a layer (in ResNets).
Group Lasso Regularization is **ALL** You Need

Step 1: Weights are split to $G$ groups $w^{(1..G)}$

  e.g. $(w_1, w_2, w_3, w_4, w_5) \rightarrow$ group 1: $(w_1, w_2, w_3)$, group 2: $(w_4, w_5)$

Step 2: Add group Lasso on each group $w^{(g)}$

  \[ \|w^{(g)}\|_g = \sqrt{\sum_{i=1}^{G} |w^{(g)}_i|^2} \]

  i.e. vector length

  $\sqrt{w_1^2 + w_2^2 + w_3^2}$, $\sqrt{w_4^2 + w_5^2}$

Step 3: Sum group Lasso over all groups as a regularization:

  \[ R_g(w) = \sum_{g=1}^{G} \|w^{(g)}\|_g, \]

  $\sqrt{w_1^2 + w_2^2 + w_3^2} + \sqrt{w_4^2 + w_5^2}$

Step 4: SGD optimizing

  \[ \arg\min_w \left\{ E(w) \right\} = \arg\min_w \left\{ E_d(w) + \lambda_g \cdot R_g(w) \right\} \]

  We refer to our method as **Structured Sparsity Learning (SSL)**
Structured Sparsity Learning (SSL)

Why can SSL learn to remove weights group by group?

The gradient-based explanation:

\[
\begin{align*}
\mathbf{w}_k^{(n)} & \leftarrow \mathbf{w}_k^{(n)} - \eta \cdot \left( \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_k^{(n)}} + \lambda \cdot \frac{\mathbf{w}_k^{(n)}}{\|\mathbf{w}_k^{(n)}\|_2} \right)
\end{align*}
\]

- A group of weights
- Regular gradients to minimize error
- Additional gradients to learn sparsity

Many groups are pushed to zeros, but not all
# Structurally Sparse AlexNet

Table 4: Sparsity and speedup of AlexNet on ILSVRC 2012

<table>
<thead>
<tr>
<th>#</th>
<th>Method</th>
<th>Top1 err.</th>
<th>Statistics</th>
<th>conv1</th>
<th>conv2</th>
<th>conv3</th>
<th>conv4</th>
<th>conv5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\ell_1$</td>
<td>44.67%</td>
<td>sparsity</td>
<td>67.6%</td>
<td>92.4%</td>
<td>97.2%</td>
<td>96.6%</td>
<td>94.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU ×</td>
<td>0.80</td>
<td>2.91</td>
<td>4.84</td>
<td>3.83</td>
<td>2.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPU ×</td>
<td>0.25</td>
<td>0.52</td>
<td>1.38</td>
<td>1.04</td>
<td>1.36</td>
</tr>
<tr>
<td>2</td>
<td>SSL</td>
<td>44.66%</td>
<td>column sparsity</td>
<td>0.0%</td>
<td>63.2%</td>
<td>76.9%</td>
<td>84.7%</td>
<td>80.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>row sparsity</td>
<td>9.4%</td>
<td>12.9%</td>
<td>40.6%</td>
<td>46.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU ×</td>
<td>1.05</td>
<td>3.37</td>
<td>6.27</td>
<td>9.73</td>
<td>4.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPU ×</td>
<td>1.00</td>
<td>2.37</td>
<td>4.94</td>
<td>4.03</td>
<td>3.05</td>
</tr>
<tr>
<td>3</td>
<td>pruning [6]</td>
<td>42.80%</td>
<td>sparsity</td>
<td>16.0%</td>
<td>62.0%</td>
<td>65.0%</td>
<td>63.0%</td>
<td>63.0%</td>
</tr>
<tr>
<td>4</td>
<td>$\ell_1$</td>
<td>42.51%</td>
<td>sparsity</td>
<td>14.7%</td>
<td>76.2%</td>
<td>85.3%</td>
<td>81.5%</td>
<td>76.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU ×</td>
<td>0.34</td>
<td>0.99</td>
<td>1.30</td>
<td>1.10</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPU ×</td>
<td>0.08</td>
<td>0.17</td>
<td>0.42</td>
<td>0.30</td>
<td>0.32</td>
</tr>
<tr>
<td>5</td>
<td>SSL</td>
<td>42.53%</td>
<td>column sparsity</td>
<td>0.00%</td>
<td>20.9%</td>
<td>39.7%</td>
<td>39.7%</td>
<td>24.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU ×</td>
<td>1.00</td>
<td>1.27</td>
<td>1.64</td>
<td>1.68</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPU ×</td>
<td>1.00</td>
<td>1.25</td>
<td>1.63</td>
<td>1.72</td>
<td>1.36</td>
</tr>
</tbody>
</table>

Structurally Sparse ResNets


<table>
<thead>
<tr>
<th></th>
<th># layers</th>
<th>error</th>
<th># layers</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>20</td>
<td>8.82%</td>
<td>32</td>
<td>7.51%</td>
</tr>
<tr>
<td>SSL-ResNet-14</td>
<td>14</td>
<td>8.54%</td>
<td>18</td>
<td>7.40%</td>
</tr>
</tbody>
</table>
Structures in LSTMs

“Dimension Consistency”:
1. **Basic Structures (in blue):** hidden states, input updates, cells, gates, outputs
2. All basic structures must have the same dimension
3. Basic structures cannot be independently removed
Structurally Sparse LSTMs

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

A group in SSL

Removing one group == reducing hidden size by one
Structurally Sparse LSTMs

Learned structurally sparse LSTMs

Table 1: Learning ISS sparsity from scratch in stacked LSTMs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Dropout keep ratio</th>
<th>Perplexity (validate, test)</th>
<th>ISS # in (1st, 2nd) LSTM</th>
<th>Weight #</th>
<th>Total time*</th>
<th>Speedup</th>
<th>Mult-add reduction†</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>0.35</td>
<td>(82.57, 78.57)</td>
<td>(1500, 1500)</td>
<td>66.0M</td>
<td>157.0ms</td>
<td>1.00×</td>
<td>1.00×</td>
</tr>
<tr>
<td>ISS</td>
<td>0.60</td>
<td>(82.59, 78.65)</td>
<td>(373, 315)</td>
<td>21.8M</td>
<td>14.82ms</td>
<td>10.59×</td>
<td>7.48×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(80.24, 76.03)</td>
<td>(381, 535)</td>
<td>25.2M</td>
<td>22.11ms</td>
<td>7.10×</td>
<td>5.01×</td>
</tr>
<tr>
<td>direct design</td>
<td>0.55</td>
<td>(90.31, 85.66)</td>
<td>(373, 315)</td>
<td>21.8M</td>
<td>14.82ms</td>
<td>10.59×</td>
<td>7.48×</td>
</tr>
</tbody>
</table>

* Measured with 10 batch size and 30 unrolled steps.
† The reduction of multiplication-add operations in matrix multiplication. Defined as (original Mult-add)/(left Mult-add)
Question Answering with SSDNNs

Question Answering in search engine:

Table 4: Remaining ISS components in BiDAF by training from scratch.

<table>
<thead>
<tr>
<th>EM</th>
<th>F1</th>
<th>ModFwd1</th>
<th>ModBwd1</th>
<th>ModFwd2</th>
<th>ModBwd2</th>
<th>OutFwd</th>
<th>OutBwd</th>
<th>weight #</th>
<th>Total time*</th>
</tr>
</thead>
<tbody>
<tr>
<td>67.98</td>
<td>77.85</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>2.69M</td>
<td>6.20ms</td>
</tr>
<tr>
<td>67.36</td>
<td>77.16</td>
<td>87</td>
<td>81</td>
<td>87</td>
<td>92</td>
<td>74</td>
<td>96</td>
<td>2.29M</td>
<td>5.83ms</td>
</tr>
<tr>
<td>66.32</td>
<td>76.22</td>
<td>51</td>
<td>33</td>
<td>42</td>
<td>58</td>
<td>37</td>
<td>26</td>
<td>1.17M</td>
<td>4.46ms</td>
</tr>
<tr>
<td>65.36</td>
<td>75.78</td>
<td>20</td>
<td>33</td>
<td>40</td>
<td>38</td>
<td>31</td>
<td>16</td>
<td>0.95M</td>
<td>3.59ms</td>
</tr>
<tr>
<td>64.60</td>
<td>74.99</td>
<td>23</td>
<td>22</td>
<td>35</td>
<td>35</td>
<td>25</td>
<td>14</td>
<td>0.88M</td>
<td>2.74ms</td>
</tr>
</tbody>
</table>

* Measured with batch size 1.
Efficiency of Structurally Sparse DNNs (after us)

S. Gray, A. Radford and D. P. Kingma, GPU Kernels for Block-Sparse Weights, OpenAI 2017

H. Mao, S. Han, et. al., CVPR Workshop 2017
Hardware/Software Co-design Framework

- Processing engines perform concurrent layer operations such as sparse convolutional, max-pooling, batch normalization, and ReLU.
Holistic Sparsity on CNNs

Storage-oriented or Performance-oriented
FPGA Accelerator Comparison

- Our design reduces the classification time by 2.6X w.r.t. the state-of-the-art FPGA implementation on ImageNet.
- The practical performance of our design is 71.2 GOPS, which corresponds to 271.6 GOPS of a dense model.

<table>
<thead>
<tr>
<th></th>
<th>FPGA-15</th>
<th>FPGA-16</th>
<th>FPGA-16</th>
<th>FPGA-16</th>
<th>This Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA Chip</td>
<td>Virtex7</td>
<td>Stratix V</td>
<td>Zynq7000</td>
<td>Stratix V</td>
<td>Zynq7000</td>
</tr>
<tr>
<td>DSP Blocks</td>
<td>2240</td>
<td>1963</td>
<td>780</td>
<td>1963</td>
<td>824</td>
</tr>
<tr>
<td>Model</td>
<td>AlexNet</td>
<td>AlexNet</td>
<td>VGG16</td>
<td>VGG16</td>
<td>AlexNet</td>
</tr>
<tr>
<td>Power</td>
<td>18.61W</td>
<td>19.1W</td>
<td>9.63W</td>
<td>19.1W</td>
<td>13.5W</td>
</tr>
<tr>
<td>Precision</td>
<td>32 float</td>
<td>16 fix</td>
<td>16 fix</td>
<td>16 fix</td>
<td>16 fix</td>
</tr>
<tr>
<td>Accuracy</td>
<td>-</td>
<td>55.41%</td>
<td>64.64%</td>
<td>66.58%</td>
<td>55.34%</td>
</tr>
<tr>
<td>CNN Size</td>
<td>1.33</td>
<td>1.45</td>
<td>30.76</td>
<td>30.9</td>
<td>0.38</td>
</tr>
<tr>
<td>Practical Gflops</td>
<td>61.62</td>
<td>72.4</td>
<td>136.97</td>
<td>117.8</td>
<td>71.2</td>
</tr>
<tr>
<td>Time/Image</td>
<td>21.73ms</td>
<td>20.1ms</td>
<td>224.6ms</td>
<td>262.9ms</td>
<td>7.57ms</td>
</tr>
</tbody>
</table>
TernGrad: Ternary Gradients in Distributed Deep Learning
**Background - Distributed Deep Learning**

Parameter server(s)

\[
\begin{align*}
\overline{g_t} &= \frac{1}{NB} \sum_{i=1}^{NB} g_t^{(i)} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta_t \overline{g_t}
\end{align*}
\]

Batch size=NB

Synchronized Data Parallelism for Stochastic Gradient Descent (SGD):

1. Training data is split to N subsets
2. Each worker has a model replica (copy)
3. Each replica is trained on a data subset
4. Synchronization in parameter server(s)

Scalability:

1. Computing time decreases with N
2. Communication can be the bottleneck
3. This work: quantizing gradient to three (i.e., ternary) levels \{-1, 0, 1\} (<2bits)

\[
\overline{g_t} = \frac{1}{NB} \sum_{i=1}^{NB} g_t^{(i)}
\]

Worker 1

Data 1

Worker 2

Data 2

Worker N

Data N

Batch size=NB

\[
\overline{g_t} = \frac{1}{NB} \sum_{i=1}^{NB} g_t^{(i)}
\]

\[
\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \overline{g_t}
\]
An Alternative Setting

1. Only exchange gradients
2. Gradient quantization can reduce communication in both directions

Parameter server:
\[ \overline{g}_t = \sum g^{(i)}_t \]

Worker 1:
\[ w_{t+1} \leftarrow w_t - \eta_t \overline{g}_t \]

Worker 2:
\[ w_{t+1} \leftarrow w_t - \eta_t \overline{g}_t \]

Worker N:
\[ w_{t+1} \leftarrow w_t - \eta_t \overline{g}_t \]
Stochastic Gradients without Bias

Batch Gradient Descent

\[ C(w) \triangleq \frac{1}{n} \sum_{i=1}^{n} Q(z_i, w) \]

\[ w_{t+1} = w_t - \eta_t \sum_{i=1}^{n} g_t^{(i)} \]

SGD

\[ w_{t+1} = w_t - \eta_t \cdot g_t^{(I)} \]

I is randomly drawn from [1, n]

\[ E\{g_t^{(I)}\} = \nabla C(w) \]

No bias

TernGrad

\[ w_{t+1} = w_t - \eta_t \cdot \text{ternarize} \left( g_t^{(I)} \right) \]

\[ E\left\{ \text{ternarize} \left( g_t^{(I)} \right) \right\} = \nabla C(w) \]

No bias
TernGrad is Simple

\[ \tilde{g}_t = \text{ternarize}(g_t) = s_t \cdot \text{sign}(g_t) \circ b_t \]

\[ s_t \triangleq \| g_t \|_{\infty} \triangleq \max(\text{abs}(g_t)) \]

\[ \begin{align*}
P(b_{tk} = 1 \mid g_t) &= \frac{|g_{tk}|}{s_t} \\
P(b_{tk} = 0 \mid g_t) &= 1 - \frac{|g_{tk}|}{s_t}
\end{align*} \]

\[ E_{z, b} \{ \tilde{g}_t \} = E_{z, b} \{ s_t \cdot \text{sign}(g_t) \circ b_t \} \]

\[ = E_z \{ s_t \cdot \text{sign}(g_t) \circ E_b \{ b_t \mid z_t \} \} = E_z \{ g_t \} = \nabla_w C(w_t) \]

No bias

Example:

\[ g_t^{(i)} : [0.30, -1.20, ..., 0.9] \]

\[ s_t : 1.20 \]

Signs: [1, -1, ..., 1]

\[ P(b_{tk} = 1 \mid g_t) : \left[ \frac{0.3}{1.2}, \frac{1.2}{1.2}, ..., \frac{0.9}{1.2} \right] \]

\[ b_t : [0, 1, ..., 1] \]

\[ g_t : [0, -1, ..., 1] \cdot 1.20 \]
Convergence


**Assumption 1:**

\[ C(w) \] has a single minimum \( w^* \) and \( \forall \epsilon > 0, \inf_{||w-w^*||^2 > \epsilon} (w - w^*)^T \nabla_w C(w) > 0 \]

**Assumption 2:**

Learning rate \( \gamma_t \) decreases neither very fast nor very slow

\[
\begin{align*}
\sum_{t=0}^{+\infty} \gamma_t^2 &< +\infty \\
\sum_{t=0}^{+\infty} \gamma_t &= +\infty
\end{align*}
\]

**Assumption 3 (gradient bound):**

\[ \mathbb{E} \{ ||g||^2 \} &\leq A + B ||w - w^*||^2 \]

Standard SGD *almost-truly* converges

\[ \mathbb{E} \{ ||g||_\infty \cdot ||g||_1 \} &\leq A + B ||w - w^*||^2 \]

*TernGrad almost-truly* converges

\[ \mathbb{E} \{ ||g||^2 \} \leq \mathbb{E} \{ ||g||_\infty \cdot ||g||_1 \} \leq A + B ||w - w^*||^2 \]

*Stronger gradient bound in TernGrad*
Closing Bound Gap

Two methods to push the gradient bound of TernGrad closer to the bound of standard SGD

Layer-wise ternarizing

Gradient clipping
Integration with Manifold Optimizers

(All experiments: All hyper-parameters are tuned for standard SGD and fixed in TernGrad)

LeNet (total mini-batch size 64): close accuracy & randomness in TernGrad results in small variance

![Graphs showing accuracy vs. number of workers for momentum SGD and vanilla SGD]

<table>
<thead>
<tr>
<th>SGD</th>
<th>base LR</th>
<th>total mini-batch size</th>
<th>iterations</th>
<th>gradients</th>
<th>workers</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>0.0002</td>
<td>128</td>
<td>300K</td>
<td>floating</td>
<td>2</td>
<td>86.56%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TernGrad</td>
<td>2</td>
<td>85.64% (-0.92%)</td>
</tr>
<tr>
<td>Adam</td>
<td>0.0002</td>
<td>2048</td>
<td>18.75K</td>
<td>floating</td>
<td>16</td>
<td>83.19%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TernGrad</td>
<td>16</td>
<td>82.80% (-0.39%)</td>
</tr>
</tbody>
</table>

Adam: D. P. Kingma, 2014

CIFAR-10, mini-batch size 64 per worker
Scaling to Large-scale Deep Learning

TernGrad: Randomness & regularization

\{(1) decrease randomness in dropout or (2) use smaller weight decay\}
No new hyper-parameters added

AlexNet

<table>
<thead>
<tr>
<th>base LR</th>
<th>mini-batch size</th>
<th>workers</th>
<th>iterations</th>
<th>gradients</th>
<th>weight decay</th>
<th>DR †</th>
<th>top-1</th>
<th>top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>256</td>
<td>2</td>
<td>370K</td>
<td>floating</td>
<td>0.0005</td>
<td>0.5</td>
<td>57.33%</td>
<td>80.56%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TernGrad</td>
<td>0.0005</td>
<td>0.2</td>
<td>57.61%</td>
<td>80.47%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TernGrad-noclip †</td>
<td>0.0005</td>
<td>0.2</td>
<td>54.63%</td>
<td>78.16%</td>
</tr>
<tr>
<td>0.02</td>
<td>512</td>
<td>4</td>
<td>185K</td>
<td>floating</td>
<td>0.0005</td>
<td>0.5</td>
<td>57.32%</td>
<td>80.73%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TernGrad</td>
<td>0.0005</td>
<td>0.2</td>
<td>57.28%</td>
<td>80.23%</td>
</tr>
<tr>
<td>0.04</td>
<td>1024</td>
<td>8</td>
<td>92.5K</td>
<td>floating</td>
<td>0.0005</td>
<td>0.5</td>
<td>56.62%</td>
<td>80.28%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TernGrad</td>
<td>0.0005</td>
<td>0.2</td>
<td>57.54%</td>
<td>80.25%</td>
</tr>
</tbody>
</table>

† DR: dropout ratio, the ratio of dropped neurons. ‡ TernGrad without gradient clipping.

N. S. Keskar, et al., ICLR 2017
Scaling to Large-scale Deep Learning

AlexNet trained on 4 workers with mini-batch size 512

GoogLeNet <2% on avg.

Tune hyper-parameters for TernGrad may reduce accuracy gap
Performance Model

*TernGrad* gives higher speedup when
1. using more workers
2. using smaller communication bandwidth (Ethernet vs InfiniBand)
3. training DNNs with more fully-connected layers (VggNet vs GoogLeNet)
ReRAM-based Neuromorphic Chip
Nonvolatile Memory Based Approaches

PCRAM Synapses
H.S.P. Wong (Stanford). IEDM. 2013

Spin-Neuron
K. Roy (Purdue). TED. 2017

Spin Oscillator Based spoken digit recognition
J. Grollier (Paris-Sud). Nature. 2017

Spiking Neural Network

Image Reconstruction
W. Lu (Umich). IEDM. 2016

Processing In Memory
Y. Xie (UCSB). ISCA. 2016

G. S. Snider (HPE). Nanotechnology. 2007

Y. Xie (UCSB). ISCA. 2016
ReRAM for Neuromorphic Computing

- Negative Differential Resistance
- Bifurcation and Chaotic Dynamics

R. S. Williams (HPE). Nature. 2017

- Scalable
- Linear resistance
- Binary
- High HRS/LRS ratio


D. Strukov (UCSB). Nanotechnology. 2012

Multi-level Cells

W. D. Lu (UMich). Nano Letters. 2015

Spike-Time-Dependent-Plasticity
Weight Mapping

- Minimize Square Mapping Error (SME)
- There is a linear coefficient between algorithm and circuits.
Weight Quantization

Floating Point

Ours

Yandan Wang, et al., DAC’17
Rank Clipping

Reconstruction error:

```
\varepsilon_K = \frac{\| W - \tilde{W} \|_2^2}{\|W\|_2^2}
```

Tolerable clipping error:

*** Original: no LRA
*** Direct LRA: using LRA after training
*** Rank clipping: integrate LRA into training

Table 1: Accuracy and ranks

<table>
<thead>
<tr>
<th>Database</th>
<th>Net</th>
<th>Method</th>
<th>Accuracy</th>
<th>conv1†</th>
<th>conv2</th>
<th>conv3</th>
<th>fc1†</th>
<th>fc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>LeNet [15]</td>
<td>Original</td>
<td>99.15%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Direct LRA</td>
<td>96.44%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rank clipping</td>
<td>99.14%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>ConvNet [1]</td>
<td>Original</td>
<td>82.01%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Direct LRA</td>
<td>43.29%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rank clipping</td>
<td>82.09%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Conv1 denotes the first convolutional layer, fc1 is the first fully-connected layer, and so forth.
† The corresponding rank indicates the number of filters in convolutional layers or the number of output neurons in fully-connected layers.
**Setup:**
BL: 0.3V, SL: 0.1V  
(stabilized by built-in CA)
High/Low Res Ratio: 9.0
Input Pattern dimension: 8*4
Our Perspectives

1. AI is going mainstream, showing potential on both the cloud and edge, however, is limited by infrastructure.

2. Future AI will be more user friendly, more automatic, and more cost efficient.

3. Securing the next generation of AI is of utmost importance, as more widespread use results in greater reward for hackers.
Thank you and Q&A