Adaptive Deep Reuse for Deep Learning

Xipeng Shen

North Carolina State University

Based on joint work with Ning Ling & Hui Guan
ML Speed is Important

Lightening-fast inferences are needed in many scenarios.

Image credit: NVIDIA & Google Image
ML Speed is Important

Shortening training time is key for fast AI product delivery & upgrade.
Deep Reuse for Fast Deep Learning
Outline

➢ Adaptive Deep Reuse
➢ Deep Reuse for Inference
➢ Adaptive Deep Reuse for Training
Prior Work on Expediting DNN

➢ Aspects focused by prior work:
  • Reduce weights
  • Reduce precision
  • Leverage sparsity

Our focus: Computation Reuse based on properties in convolutional layers’ inputs
Intuition
Basic Idea of Computation Reuse
Basic Idea of Computation Reuse

Unfold the image to a large input matrix
Basic Idea of Computation Reuse

Unfold the image to a large input matrix

A neuron vector with 2 elements
Basic Idea of Computation Reuse

Unfold the image to a large input matrix

Group similar neuron vectors into groups
Basic Idea of Computation Reuse

Unfold the image to a large input matrix

Group similar neuron vectors into groups

Reuse the computation results for neuron vectors in the same group
Basic Idea of Computation Reuse

Unfold the image to a large input matrix

Group similar neuron vectors into groups

Reuse the computation results for neuron vectors in the same group
Basic Idea of Computation Reuse

• Originally, 16 vector dot products.
• After computation reuse, 8 vector dot products

Unfold the image to a large input matrix

Group similar neuron vectors into groups

Reuse the computation results for neuron vectors in the same group
Deep Reuse – Forward Propagation
Deep Reuse – Forward Propagation

Stage 1: Clustering
Deep Reuse – Forward Propagation

Stage 1: Clustering

Stage 2: MM between the centroid matrix and weight matrix

Remaining ratio: \( r_c = \frac{|C|}{N} \)
Deep Reuse – Forward Propagation

Stage 1: Clustering
Stage 2: MM between the centroid matrix and weight matrix
Stage 3: Reconstructing the output matrix
Deep Reuse – Forward Propagation

Stage 1: Clustering
Stage 2: MM between the centroid matrix and weight matrix
Stage 3: Reconstructing the output matrix
Deep Reuse – Forward Propagation

Stage 1: Clustering
Stage 2: MM between the centroid matrix and weight matrix
Stage 3: Reconstructing the output matrix

Overhead must be smaller than the time savings.
Deep Reuse – Similarity Identification

➢ Clustering Method:
   ➢ Locality-Sensitive Hashing (LSH).  [**H**: #hashing functions]

➢ Cluster Scope [Single-input level, single-batch level, across-batch level]

➢ Clustering granularity [**L**: the length of the sub-vector]

➢ Similarity Metric  [Angular cosine distance]
Experiments

<table>
<thead>
<tr>
<th>Network</th>
<th>Dataset</th>
<th># ConvLayers</th>
<th>K</th>
<th>M</th>
<th>Image Order</th>
<th>Image Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFARNet</td>
<td>CIFAR10</td>
<td>2</td>
<td>75 ~ 1600</td>
<td>64</td>
<td>RANDOM</td>
<td>32 × 32</td>
</tr>
<tr>
<td>ALEXNet</td>
<td>IMAGE_NET</td>
<td>5</td>
<td>363 ~ 3456</td>
<td>64 ~ 384</td>
<td>RANDOM</td>
<td>224 × 224</td>
</tr>
<tr>
<td>VGG-19</td>
<td>IMAGE_NET</td>
<td>16</td>
<td>27 ~ 4068</td>
<td>64 ~ 512</td>
<td>RANDOM</td>
<td>224 × 224</td>
</tr>
</tbody>
</table>

**Server:** an Intel(R) Xeon(R) CPU E5-1607 v2 and a GTX1080 GPU.

**Mobile:** Huawei SE mate mobile phone with Huawei HiSilicon KIRIN 659 processor and a 4 GB memory.
Existence of similarity

Alexnet Cov3
Inference time per batch

- CifarNet
  - Baseline: 1.1 ms
  - Deep Reuse: 1.3 ms

- AlexNet
  - Baseline: 1.3 ms
  - Deep Reuse: 2.2 ms

- VGG-19
  - Baseline: 90 ms
  - Deep Reuse: 100 ms
### On Server Machine

<table>
<thead>
<tr>
<th>Network</th>
<th>Acc</th>
<th>Performance</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ΔAcc</td>
<td>S</td>
</tr>
<tr>
<td>CifarNet</td>
<td>0.7892</td>
<td>-0.0011</td>
<td>1.75X</td>
</tr>
<tr>
<td>AlexNet</td>
<td>0.5360</td>
<td>-0.0002</td>
<td>2.02X</td>
</tr>
<tr>
<td>VGG-19</td>
<td>0.7118</td>
<td>+0.0005</td>
<td>1.89X</td>
</tr>
</tbody>
</table>

### On Smartphone

<table>
<thead>
<tr>
<th>Network</th>
<th>CT_b(ms)</th>
<th>S</th>
<th>σ(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CifarNet</td>
<td>29.42</td>
<td>2.12X</td>
<td>0.163</td>
</tr>
<tr>
<td>AlexNet</td>
<td>326.15</td>
<td>2.55X</td>
<td>0.336</td>
</tr>
</tbody>
</table>
rc-accuracy relation on conv2 of CifaNet
Speedups on Compressed Network

On compressed AlexNet

<table>
<thead>
<tr>
<th>Layer</th>
<th>conv1</th>
<th>conv2</th>
<th>conv3</th>
<th>conv4</th>
<th>conv5</th>
</tr>
</thead>
<tbody>
<tr>
<td>speedup</td>
<td>1.81X</td>
<td>3.29X</td>
<td>3.64X</td>
<td>3.45X</td>
<td>2.71X</td>
</tr>
</tbody>
</table>
Outline

➢ Adaptive Deep Reuse
➢ Deep Reuse for Inference
➢ Adaptive Deep Reuse for Training
Special Aspects

Forward
Backward
Special Aspects

Tolerate approximation errors, esp. at the early stage of the training.
Avoid Reclustering in Back Propagation

Forward Prop.

\[ y = x \cdot W \]

\[ y_c = x_c \cdot W \]
Deep Reuse in Back Propagation

Forward Prop.
\[ y = x \cdot W \]
\[ y_c = x_c \cdot W \]

Backward Prop. (Weights)
\[ \nabla W = x^T \cdot \delta y \]
\[ \delta y_1 = \delta y_{1,s} = \delta y_1 + \delta y_3 \]

No reclustering is needed!
Deep Reuse in Back Propagation

Forward Prop.

\[ y = x \cdot W \]

\[ y_c = x_c \cdot W \]

Backward Prop. (Inputs)

\[ \delta y = \delta y^T \]

\[ \delta x = \delta y \cdot W^T \]

No reclustering is needed!
Adaptive Deep Reuse for Training

Expected computation time: \( \varepsilon_t \sim \frac{H}{M} + \tau_c + \frac{1}{L} \)

Reuse-caused accuracy loss: \( \delta a \)

**Main Observations:**

- Smaller \( L \) leads to smaller \( \delta a \)
- Larger \( H \) leads to smaller \( \delta a \)
- ......

Start with \( L_{max} \) and \( H_{min} \)

Search range: \( \{L_{max}, H_{min}\} \) to \( \{L_{min}, H_{max}\} \)

Iteration 0

- More aggressive, Larger \( L \), smaller \( H \)
- Smaller \( \varepsilon_t \), larger \( \delta a \)

Iteration \( N_t \)

- Less aggressive, Smaller \( L \), larger \( H \)
- Larger \( \varepsilon_t \), smaller \( \delta a \)
Time Savings

Three strategies:

➢ Strategy 1: Fixed \(\{L,H\}\)
➢ Strategy 2: Adaptive \(\{L,H\}\)
➢ Strategy 3: Adaptive cluster scope

See ICDE’2019 paper for details.
Conclusion

➢ Tremendous neuron vector similarities exist
➢ Online LSH clustering is efficient enough
➢ Reuse benefits both training and inference
➢ > 60% training time savings via adaptivity
➢ ~2X speedups of inference

No extra hardware needed!
Directly deployable!
Compatible w/ model compression
Potential for various DNNs.