Learning Structured Sparsity in Deep Neural Networks

Wei Wen\textsuperscript{1} Chunpeng Wu\textsuperscript{1} Yandan Wang\textsuperscript{1} Yiran Chen\textsuperscript{1} Hai Li\textsuperscript{1}

\textsuperscript{1}University of Pittsburgh

NIPS, 2016

Presenter: Bargav Jayaraman
Outline

1 Introduction

2 Related Works

3 Proposed Structure Sparsity Learning Approach
   - SSL for Generic Structures
   - SSL for Filters and Channels
   - SSL for Filter Shapes
   - SSL for Layer Depth
   - SSL for Computationally Efficient Structures

4 Experimental Results

5 Summary
Introduction

- **Problem:** Deployment of large-scale deep learning model is computationally expensive
- **Solution:** Occam’s Razor - Simple is better!
  Remove or zero-out the non-essential weights / layers of the model
Introduction

- **Problem:** Deployment of large-scale deep learning model is computationally expensive
- **Solution:** Occam’s Razor - Simple is better!
  Remove or zero-out the non-essential weights / layers of the model
  **Catch:** Trade-off between model complexity and accuracy
Related Works

Connection pruning and weight sparsifying. Connection pruning removes unwanted weight connections from the fully connected layers of a CNN. Not much beneficial for convolutional layers! Hard-coding sparse weights for convolutional layers introduces non-structured sparsity with slight accuracy loss.
- This work achieves structured sparsity in adjacent memory space.
**Related Works**

- **Connection pruning and weight sparsifying.** Connection pruning removes unwanted weight connections from the fully connected layers of a CNN. **Not much beneficial for convolutional layers!** Hard-coding sparse weights for convolutional layers introduces non-structured sparsity with slight accuracy loss.
  - This work achieves structured sparsity in adjacent memory space

- **Low rank approximation.** LRA compresses the deep network by decomposing the weight matrix $W \in \mathbb{R}^{u \times v}$ at every layer into product of two matrices $U \in \mathbb{R}^{u \times \alpha}$ and $V \in \mathbb{R}^{\alpha \times v}$, where $\alpha < u, v$.
  - This work dynamically optimizes the model and obtains lower rank approximation
**Related Works**

- **Connection pruning and weight sparsifying.** Connection pruning removes unwanted weight connections from the fully connected layers of a CNN. *Not much beneficial for convolutional layers!*
  
  Hard-coding sparse weights for convolutional layers introduces non-structured sparsity with slight accuracy loss.
  
  - This work achieves structured sparsity in adjacent memory space

- **Low rank approximation.** LRA compresses the deep network by decomposing the weight matrix \( W \in \mathbb{R}^{u \times v} \) at every layer into product of two matrices \( U \in \mathbb{R}^{u \times \alpha} \) and \( V \in \mathbb{R}^{\alpha \times v} \), where \( \alpha < u, v \).
  
  - This work dynamically optimizes the model and obtains lower rank approximation

- **Model structure learning.** Group Lasso has been used for structure sparsity in deep models to learn the appropriate number of filters or filter shapes.
  
  - This work applies group Lasso at various levels of the deep model
Outline

1 Introduction
2 Related Works
3 Proposed Structure Sparsity Learning Approach
   • SSL for Generic Structures
   • SSL for Filters and Channels
   • SSL for Filter Shapes
   • SSL for Layer Depth
   • SSL for Computationally Efficient Structures
4 Experimental Results
5 Summary
Consider the weights of a deep network as a 4-D tensor:

\( W^{(l)} \in \mathbb{R}^{N_l \times C_l \times M_l \times K_l} \), where \( N_l, C_l, M_l \) and \( K_l \) are the dimensions of the \( l \)-th layer \((1 \leq l \leq L)\) weight tensor along the axes of filter, channel, spatial height and spatial width. \( L \) denotes the number of convolutional layers. Then the proposed generic optimization is:

\[
E(W) = E_D(W) + \lambda R(W) + \lambda_g \sum_{l=1}^{L} R_g(W^{(l)})
\]

\( E_D(W) \) is the loss on data, \( R(.) \) is the non-structured regularizer, like \( l_2 \)-norm, and \( R_g(.) \) is the structured regularizer. This work uses group Lasso for \( R_g(.) \).
Group Lasso

- The regularization of group Lasso on a set of weights $w$ is given as:
  $$R_g(w) = \sum_{g=1}^{G} \|w^{(g)}\|_g,$$
  where $g$ is a group of partial weights in $w$ and $G$ is the total number of groups.

- $\| \cdot \|_g$ is the group Lasso, or $\|w^{(g)}\|_g = \sqrt{\sum_{i=1}^{|w^{(g)}|} (w_i^{(g)})^2}$, where $|w^{(g)}|$ is the number of weights in $w^{(g)}$. 

**Question:** Why is this called group “Lasso” if it uses $l_2$-regularization?

**Answer:** $l_2$-regularization has all-or-none zero effect!
Group Lasso

- The regularization of group Lasso on a set of weights $w$ is given as:
  $$R_g(w) = \sum_{g=1}^{G} \| w^{(g)} \|_g$$
  where $g$ is a group of partial weights in $w$ and $G$ is the total number of groups.

- $\| \cdot \|_g$ is the group Lasso, or $\| w^{(g)} \|_g = \sqrt{\sum_{i=1}^{\|w^{(g)}\|} (w^{(g)}_i)^2}$, where $\|w^{(g)}\|$ is the number of weights in $w^{(g)}$.

Question: Why is this called group “Lasso” if it uses $l_2$-regularization?
The regularization of group Lasso on a set of weights $w$ is given as: 

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

where $g$ is a group of partial weights in $w$ and $G$ is the total number of groups.

$\|\cdot\|_g$ is the group Lasso, or $\|w^{(g)}\|_g = \sqrt{\sum_{i=1}^{|w^{(g)}|} (w^{(g)}_i)^2}$, where $|w^{(g)}|$ is the number of weights in $w^{(g)}$.

Question: Why is this called group “Lasso” if it uses $l_2$-regularization?

Answer: $l_2$-regularization has all-or-none zero effect!
Outline

1. Introduction

2. Related Works

3. Proposed Structure Sparsity Learning Approach
   - SSL for Generic Structures
   - SSL for Filters and Channels
   - SSL for Filter Shapes
   - SSL for Layer Depth
   - SSL for Computationally Efficient Structures

4. Experimental Results

5. Summary
SSL for Filters and Channels

Suppose $W_{n_l,:,:,:}$ is the $n_l$-th filter and $W_{:,c_l,:,:}$ is the $c_l$-th channel of all filters in the $l$-th layer. Then the optimization target is defined as:

$$E(W) = E_D(W) + \lambda_n \sum_{l=1}^{L} \left( \sum_{n_l=1}^{N_l} \| W_{n_l,:,:,:}^{(l)} g \| \right) + \lambda_c \sum_{l=1}^{L} \left( \sum_{c_l=1}^{C_l} \| W_{:,c_l,:,:}^{(l)} g \| \right)$$
Outline

1. Introduction
2. Related Works
3. Proposed Structure Sparsity Learning Approach
   - SSL for Generic Structures
   - SSL for Filters and Channels
   - SSL for Filter Shapes
   - SSL for Layer Depth
   - SSL for Computationally Efficient Structures
4. Experimental Results
5. Summary
SSL for Filter Shapes

Suppose $W_{:,c_l,m_l,k_l}^{(l)}$ denotes the vector of all corresponding weights of spatial position $(m_l, k_l)$ in the filters across $c_l$-th channel, then:

$$E(W) = E_D(W) + \lambda_s \sum_{l=1}^{L} (\sum_{c_l=1}^{C_l} \sum_{m_l=1}^{M_l} \sum_{k_l=1}^{K_l} \| W_{:,c_l,m_l,k_l}^{(l)} \|_g )$$
Outline

1. Introduction

2. Related Works

3. Proposed Structure Sparsity Learning Approach
   - SSL for Generic Structures
   - SSL for Filters and Channels
   - SSL for Filter Shapes
   - SSL for Layer Depth
   - SSL for Computationally Efficient Structures

4. Experimental Results

5. Summary
SSL for Layer Depth

Depth sparsity reduces the computation cost and improves accuracy. The optimization is given as:

\[ E(W) = E_D(W) + \lambda_d \cdot \sum_{l=1}^{L} \| W^{(l)} \|_g \]

Zeroing out all filters in a layer can hinder the message passing across layers, and hence shortcut is used to transfer the feature map.
Outline

1 Introduction

2 Related Works

3 Proposed Structure Sparsity Learning Approach
   - SSL for Generic Structures
   - SSL for Filters and Channels
   - SSL for Filter Shapes
   - SSL for Layer Depth
   - SSL for Computationally Efficient Structures

4 Experimental Results

5 Summary
SSL for Computationally Efficient Structures

- **2D-filter-wise sparsity for convolution.** Fine-grain variant of filter-wise sparsity is zeroing out 2D filters instead of 3D filters for efficient computation reduction. Since, 2D filters are smaller groups and hence easy to zero-out.

- **Combination of filter-wise and shape-wise sparsity for GEMM.** Convolutional operation is represented as a matrix in GEneral Matrix Multiplication (GEMM) such that each row is represented as a feature and each column is a collection of weight corresponding to shape sparsity. Combining filter-wise and shape-wise sparsity zeroes out the rows and columns of the weight matrix and hence reduces the dimensionality.
Experimental Results

- Filter-wise, Channel-wise and Shape-wise SSL on LeNet
- SSL on fully-connected MLP
- Filter-wise and Shape-wise SSL on ConvNet
- Depth-wise SSL on ResNet
- SSL on AlexNet
Table 1: Results after penalizing unimportant filters and channels in *LeNet*

<table>
<thead>
<tr>
<th><em>LeNet</em> #</th>
<th>Error</th>
<th>Filter # $^\S$</th>
<th>Channel # $^\S$</th>
<th>FLOP $^\S$</th>
<th>Speedup $^\S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (baseline)</td>
<td>0.9%</td>
<td>20–50</td>
<td>1–20</td>
<td>100%–100%</td>
<td>1.00×–1.00×</td>
</tr>
<tr>
<td>2</td>
<td>0.8%</td>
<td>5–19</td>
<td>1–4</td>
<td>25%–7.6%</td>
<td>1.64×–5.23×</td>
</tr>
<tr>
<td>3</td>
<td>1.0%</td>
<td>3–12</td>
<td>1–3</td>
<td>15%–3.6%</td>
<td>1.99×–7.44×</td>
</tr>
</tbody>
</table>

$^\S$ In the order of *conv1*–*conv2*

Table 2: Results after learning filter shapes in *LeNet*

<table>
<thead>
<tr>
<th><em>LeNet</em> #</th>
<th>Error</th>
<th>Filter size $^\S$</th>
<th>Channel #</th>
<th>FLOP</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (baseline)</td>
<td>0.9%</td>
<td>25–500</td>
<td>1–20</td>
<td>100%–100%</td>
<td>1.00×–1.00×</td>
</tr>
<tr>
<td>4</td>
<td>0.8%</td>
<td>21–41</td>
<td>1–2</td>
<td>8.4%–8.2%</td>
<td>2.33×–6.93×</td>
</tr>
<tr>
<td>5</td>
<td>1.0%</td>
<td>7–14</td>
<td>1–1</td>
<td>1.4%–2.8%</td>
<td>5.19×–10.82×</td>
</tr>
</tbody>
</table>

$^\S$ The sizes of filters after removing zero shape fibers, in the order of *conv1*–*conv2*

Figure 3: Learned *conv1* filters in *LeNet* 1 (top), *LeNet* 2 (middle) and *LeNet* 3 (bottom)
### MLP

<table>
<thead>
<tr>
<th>MLP #</th>
<th>Error</th>
<th>Neuron # per layer $\text{§}$</th>
<th>FLOP per layer $\text{§}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (baseline)</td>
<td>1.43%</td>
<td>784--500--300--10</td>
<td>100%--100%--100%</td>
</tr>
<tr>
<td>2</td>
<td>1.34%</td>
<td>469--294--166--10</td>
<td>35.18%--32.54%--55.33%</td>
</tr>
<tr>
<td>3</td>
<td>1.53%</td>
<td>434--174--78--10</td>
<td>19.26%--9.05%--26.00%</td>
</tr>
</tbody>
</table>

$\text{§}$ In the order of input layer--hidden layer 1--hidden layer 2--output layer

![Image](image.png)

**Figure 4:** (a) Results of learning the number of neurons in *MLP*. (b) the connection numbers of input neurons (i.e. pixels) in *MLP 2* after SSL.
Table 3: Learning row-wise and column-wise sparsity of ConvNet on CIFAR-10

<table>
<thead>
<tr>
<th>ConvNet #</th>
<th>Error</th>
<th>Row sparsity $^$</th>
<th>Column sparsity $^$</th>
<th>Speedup $^$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (baseline)</td>
<td>17.9%</td>
<td>12.5%–0%–0%</td>
<td>0%–0%–0%</td>
<td>1.00×–1.00×–1.00×</td>
</tr>
<tr>
<td>2</td>
<td>17.9%</td>
<td>50.0%–28.1%–1.6%</td>
<td>0%–59.3%–35.1%</td>
<td>1.43×–3.05×–1.57×</td>
</tr>
<tr>
<td>3</td>
<td>16.9%</td>
<td>31.3%–0%–1.6%</td>
<td>0%–42.8%–9.8%</td>
<td>1.25×–2.01×–1.18×</td>
</tr>
</tbody>
</table>

$^\$ in the order of conv1–conv2–conv3

Figure 5: Learned conv1 filters in ConvNet 1 (top), ConvNet 2 (middle) and ConvNet 3 (bottom)
Figure 6: Error vs. layer number after depth regularization by SSL. ResNet-# is the original ResNet in [5] with # layers. SSL-ResNet-# is the depth-regularized ResNet by SSL with # layers, including the last fully-connected layer. 32×32 indicates the convolutional layers with an output map size of 32×32, and so forth.
### 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[7]</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>
Summary

- Filter-wise, channel-wise, shape-wise and depth-wise SSL
- Dynamic compact structure learning without loss of accuracy
- Significant speed-ups with both CPUs and GPUs