Learning Structured Sparsity in Deep Neural Networks

Wei Wen[1] Chunpeng Wu[1] Yandan Wang[1] Yiran Chen[1] Hai Li[1]

[1]University of Pittsburgh

NIPS, 2016
Presenter: Bargav Jayaraman

Outline

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

Structure Sparsity Learning for Generic Structures

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.
- $\|.\|_g$ 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 $w^{(g)}$.

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.
- $\|.\|_g$ 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?

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.
- $\|.\|_g$ 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 $w^{(g)}$.

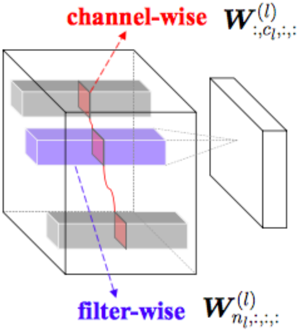
Question: Why is this called group "Lasso" if it uses l_2-regularization?
Answer: l_2-regularization has all-or-none zero effect!

Outline

SSL for Filters and Channels

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

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

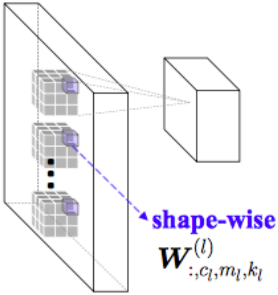


Outline

SSL for Filter Shapes

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

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



Outline

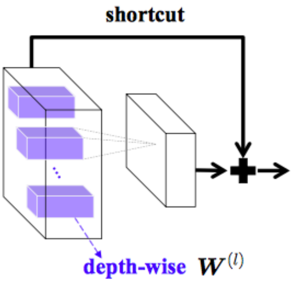
1. Introduction
2. Related Works
3. Proposed Structure Sparsity Learning Approach
   - SSL for Generic Structures
   - SSL for Filters and Channels
   - SSL for Filter Shapes
   - **SSL for Layer Depth**
   - SSL for Computationally Efficient Structures
4. Experimental Results
5. Summary

SSL for Layer Depth

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

$$E(W) = E_D(W) + \lambda_d . \sum_{l=1}^{L} \|W^{(l)}\|_g$$

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

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

LeNet

Table 1: Results after penalizing unimportant filters and channels in *LeNet*

LeNet #	Error	Filter # §	Channel # §	FLOP §	Speedup §
1 (*baseline*)	0.9%	20—50	1—20	100%—100%	1.00×—1.00×
2	0.8%	5—19	1—4	25%—7.6%	1.64×—5.23×
3	1.0%	3—12	1—3	15%—3.6%	1.99×—7.44×

§ In the order of *conv1*—*conv2*

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

LeNet #	Error	Filter size §	Channel #	FLOP	Speedup
1 (*baseline*)	0.9%	25—500	1—20	100%—100%	1.00×—1.00×
4	0.8%	21—41	1—2	8.4%—8.2%	2.33×—6.93×
5	1.0%	7—14	1—1	1.4%—2.8%	5.19×—10.82×

§ The sizes of filters after removing zero shape fibers, in the order of *conv1*—*conv2*

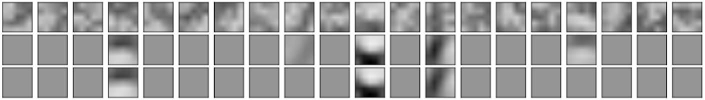


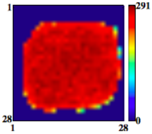
Figure 3: Learned *conv1* filters in *LeNet 1* (top), *LeNet 2* (middle) and *LeNet 3* (bottom)

MLP

MLP #	Error	Neuron # per layer §	FLOP per layer §
1 (*baseline*)	1.43%	784–500–300–10	100%–100%–100%
2	1.34%	469–294–166–10	35.18%–32.54%–55.33%
3	1.53%	434–174–78–10	19.26%–9.05%–26.00%

§In the order of *input layer–hidden layer 1–hidden layer 2–output layer*

(a)



(b)

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.

ConvNet

Table 3: Learning row-wise and column-wise sparsity of *ConvNet* on CIFAR-10

ConvNet #	Error	Row sparsity §	Column sparsity §	Speedup §
1 (*baseline*)	17.9%	12.5%–0%–0%	0%–0%–0%	1.00×–1.00×–1.00×
2	17.9%	50.0%–28.1%–1.6%	0%–59.3%–35.1%	1.43×–3.05×–1.57×
3	16.9%	31.3%–0%–1.6%	0%–42.8%–9.8%	1.25×–2.01×–1.18×

§ in the order of *conv1*–*conv2*–*conv3*

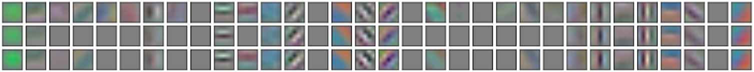


Figure 5: Learned *conv1* filters in *ConvNet 1* (top), *ConvNet 2* (middle) and *ConvNet 3* (bottom)

ResNet

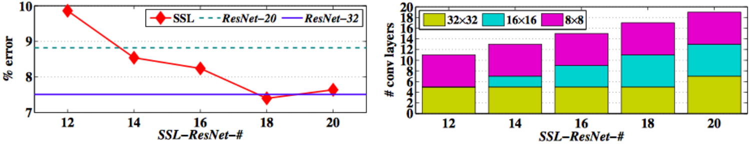


Figure 6: Error vs. layer number after depth regularization by SSL. *ResNet-#* is the original *ResNet* in [5] with # layers. *SSL-ResNet-#* is the depth-regularized *ResNet* by SSL with # layers, including the last fully-connected layer. 32×32 indicates the convolutional layers with an output map size of 32×32, and so forth.

AlexNet

Table 4: Sparsity and speedup of *AlexNet* on ILSVRC 2012

#	Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5
1	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%
			CPU ×	0.80	2.91	4.84	3.83	2.76
			GPU ×	0.25	0.52	1.38	1.04	1.36
2	SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%
			row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%
			CPU ×	1.05	3.37	6.27	9.73	4.93
			GPU ×	1.00	2.37	4.94	4.03	3.05
3	pruning[7]	42.80%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%
4	ℓ_1	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%
			CPU ×	0.34	0.99	1.30	1.10	0.93
			GPU ×	0.08	0.17	0.42	0.30	0.32
5	SSL	42.53%	column sparsity	0.00%	20.9%	39.7%	39.7%	24.6%
			CPU ×	1.00	1.27	1.64	1.68	1.32
			GPU ×	1.00	1.25	1.63	1.72	1.36

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