



CMSC 5743

Efficient Computing of Deep Neural Networks

Model 01: Pruning

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

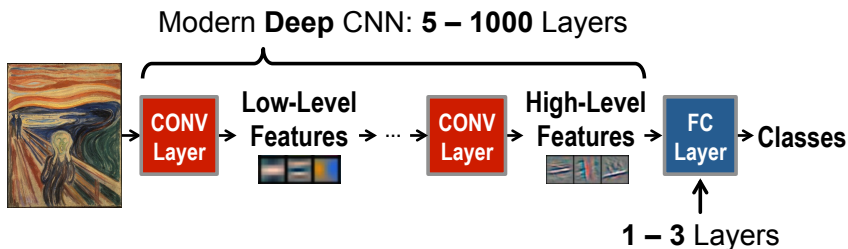
(Latest update: February 6, 2026)

2026 Spring

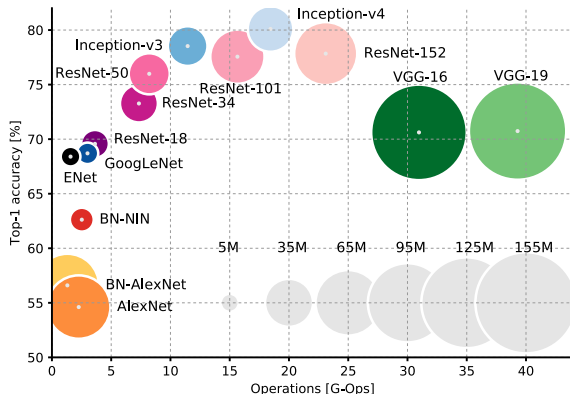


These slides contain/adapt materials developed by

- Wei Wen et al. (2016). “Learning structured sparsity in deep neural networks”. In: *Proc. NIPS*, pp. 2074–2082
- Yihui He, Xiangyu Zhang, and Jian Sun (2017). “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proc. ICCV*
- Ruichi Yu et al. (2018). “NISP: Pruning networks using neuron importance score propagation”. In: *Proc. CVPR*, pp. 9194–9203
- Shijin Zhang et al. (2016). “Cambricon-x: An accelerator for sparse neural networks”. In: *Proc. MICRO*. IEEE, pp. 1–12
- Jorge Albericio et al. (2016). “Cnvlutin: Ineffectual-neuron-free deep neural network computing”. In: *ACM SIGARCH Computer Architecture News* 44.3, pp. 1–13



- Researchers design deeper and larger networks to ensure model performance.
- 😊 VGG-16, 16 parameter layers
- 😊 VGG-19, 19 parameter layers
- 😊 GoogLeNet, 22 parameter layers
- 😊 ResNet : -18, -34, -50, -101, -152 layers



- The size of the blob is proportional to the number of network parameters.
- More than millions of parameters and billions of operations.
- Challenges in **memory** and **energy**, finally affect the performance.



② Sparse Regression

③ Pruning



Sparse Regression



Input

- $\mathbf{y} = (y_1, \dots, y_N)^\top$: N samples to measure performance
- $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)})^\top$: N parameters, where $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)})^\top$ is parameter vector for sample y_i

Output

- $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)^\top$: linear regression model coefficients, s.t. $\mathbf{y} \approx \mathbf{X}\boldsymbol{\beta}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix} \approx \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_p^{(N)} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_p \end{bmatrix}$$



Input

- $\mathbf{y} = (y_1, \dots, y_N)^\top$: N samples to measure performance
- $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)})^\top$: N parameters, where $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)})^\top$ is parameter vector for sample y_i

Output

- $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)^\top$: linear regression model coefficients, s.t. $\mathbf{y} \approx \mathbf{X}\boldsymbol{\beta}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix} \approx \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_p^{(N)} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_p \end{bmatrix}$$

Objective

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

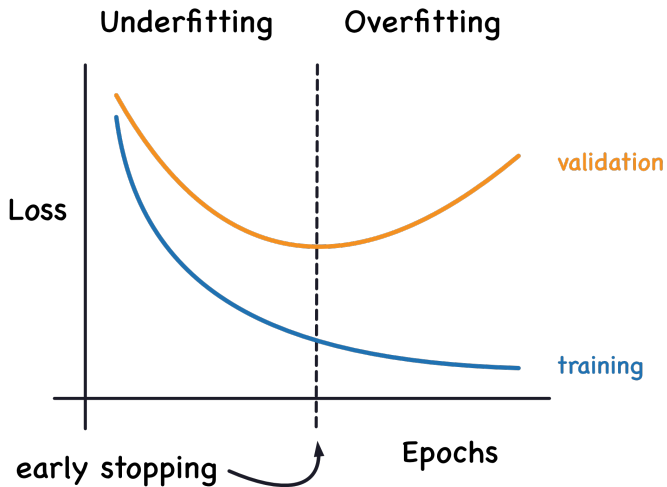


$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_p^{(N)} \end{bmatrix}$$

N : sample #

p : parameter #

- Time consuming to run simulation or measure $\rightarrow$ sample# N is **limited**
- If $N < \text{parameter\# } p$, $\rightarrow$ **no unique** solutions
- **Overfitting** problem
- **Should reduce parameter#**





$$S_i = \frac{f(x_1, \dots, x_i + \Delta x_i, \dots, x_K) - f(x_1, \dots, x_K)}{\Delta x_i}$$

- 😊 Computationally efficient
- 😞 Only take into account local variation around nominal value



$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \rightarrow \beta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

- 😊 Global view
- ☹️ Too complicated model after analysis
- ☹️ Need large simulation size ($N > p$)
- ☹️ Otherwise $\mathbf{X}^\top \mathbf{X}$ may be singular (difficult to invert)



Please prove:

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \quad \rightarrow \quad \beta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$



Definition

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$



Definition

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

- L-0 norm: $\|\mathbf{x}\|_0$ = number of non-zero values



Definition

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

- L-0 norm: $\|\mathbf{x}\|_0$ = number of non-zero values
- L-1 norm: $\|\mathbf{x}\|_1 = \sum_i |x_i|$



Definition

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

- L-0 norm: $\|\mathbf{x}\|_0$ = number of non-zero values
- L-1 norm: $\|\mathbf{x}\|_1 = \sum_i |x_i|$
- L-2 norm: $\|\mathbf{x}\|_2 = \sqrt{\sum_i |x_i|^2} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$



Definition

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

- L-0 norm: $\|\mathbf{x}\|_0$ = number of non-zero values
- L-1 norm: $\|\mathbf{x}\|_1 = \sum_i |x_i|$
- L-2 norm: $\|\mathbf{x}\|_2 = \sqrt{\sum_i |x_i|^2} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$
- L-infinity norm: $\|\mathbf{x}\|_\infty = \max_i |x_i|$



Which of the following is L0?

- A. $x_1 + x_2 + x_3$
- B. $\max\{x_1, x_2, x_3\}$
- C. $|x_1| + |x_2| + |x_3|$
- D. none of them



$$\begin{aligned} &\text{minimize} && \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|, \\ &\text{subject to} && \|\boldsymbol{\beta}\|_0 \leq \lambda. \end{aligned}$$

- 😊 Global view
- 😞 $\mathcal{NP}$ -hard
- Orthogonal matching pursuit ([OMP](#)): iterative heuristics
- 😞 Computational expensive



Why we dont use L0 as regularization term?

- A. complicated inference
- B. no direct gradient obtained
- C. none of above
- D. both of above



$$\arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_{j=1}^p |\beta_j|^2$$



$$\arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_{j=1}^p |\beta_j|^2$$
$$\rightarrow \beta = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

$$\arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_{j=1}^p |\beta_j|$$

- “ ℓ_1 penalty” (Lasso)
- β optimally solved by Coordinate Descent [Friedman+, AOAS'07]
- λ : nonnegative regularization parameter



Which of the following regularization term can introduce sparsity?

- A. L1
- B. L2
- C. both of them
- D. None of them



Which regularization is used to reduce the over fit problem?

- A. L1
- B. L2
- C. both of them
- D. None of them

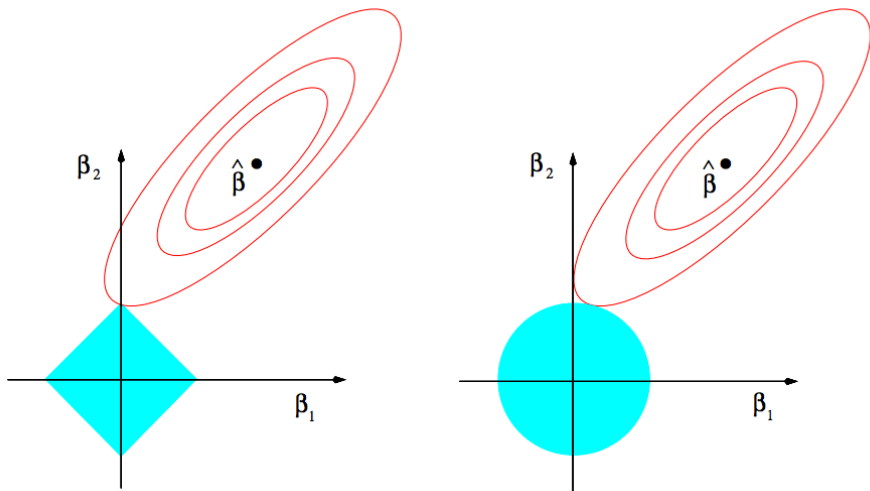
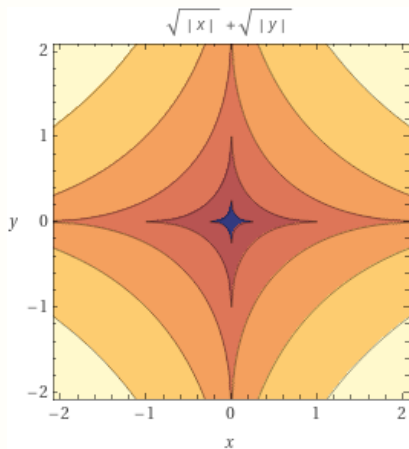


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

Can we introduce a more sparse regularization term? Why

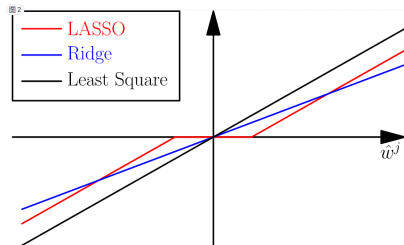
For example, $|x|^{1/2} + |y|^{1/2}$.





Provide closed-form solution to single variable Lasso problem:

$$\min_x L(x) = \frac{1}{2}(x - w)^2 + \lambda|x|$$



Note: Lasso solution

- If $x > 0$, then $\omega - \lambda > 0 (\omega > \lambda) \rightarrow x = \omega - \lambda$
- If $x < 0$, then $\omega + \lambda < 0 (\omega < -\lambda) \rightarrow x = \omega + \lambda$
- $x = 0$, others



- The idea behind coordinate descent is, simply, to optimize a target function with respect to a **single parameter at a time**, iteratively cycling through all parameters until convergence is reached
- Coordinate descent is particularly suitable for problems, like the lasso, that have a simple closed form solution in a single dimension but lack one in higher dimensions



- Let us consider minimizing Q with respect to β_j , while temporarily treating the other regression coefficients β_{-j} as fixed:

$$Q(\beta_j | \beta_{-j}) = \frac{1}{2n} \sum_{i=1}^n (y_i - \sum_{k \neq j} x_{ik} \beta_k - x_{ij} \beta_j)^2 + \lambda |\beta_j| + \text{Constant}$$

- Let

$$\begin{aligned} \tilde{r}_{ij} &= y_i - \sum_{k \neq j} x_{ik} \tilde{\beta}_k \\ \tilde{z}_j &= n^{-1} \sum_{i=1}^n x_{ij} \tilde{r}_{ij}, \end{aligned}$$

where $\{\tilde{r}_{ij}\}_{i=1}^n$ are the partial residuals with respect to the j^{th} predictor, and $\tilde{z}_j$ is the OLS estimator based on $\{\tilde{r}_{ij}, x_{ij}\}_{i=1}^n$



- We have already solved the problem of finding a one-dimensional lasso solution; letting $\tilde{\beta}_j$ denote the minimizer of $Q(\beta_j | \tilde{\beta}_{-j})$,

$$\tilde{\beta}_j = S(\tilde{z}_j | \lambda)$$

- This suggests the following algorithm:

repeat

for $j = 1, 2, \dots, p$

$$\tilde{z}_j = n^{-1} \sum_{i=1}^n x_{ij} r_i + \tilde{\beta}_j^{(s)}$$

$$\tilde{\beta}_j^{(s+1)} \leftarrow S(\tilde{z}_j | \lambda)$$

$$r_i \leftarrow r_i - (\tilde{\beta}_j^{(s+1)} - \tilde{\beta}_j^{(s)}) x_{ij} \text{ for all } i.$$

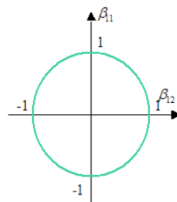
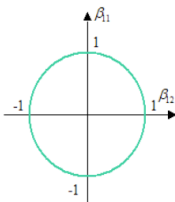
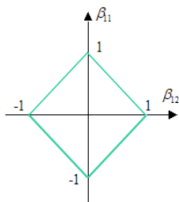
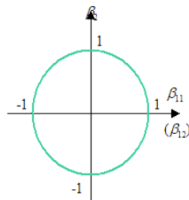
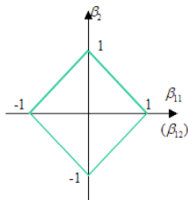
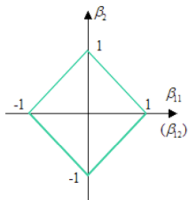
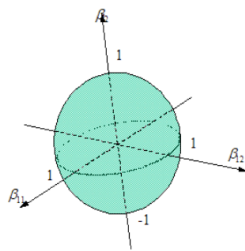
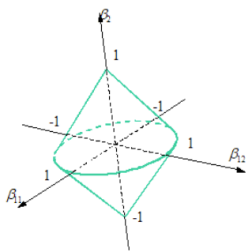
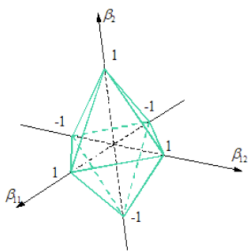
until convergence



- We denote X as being composed of J groups $X_1, X_2, \dots, X_J$
- $X\beta = \sum_j X_j\beta_j$, where β_j represents the coefficients belonging to the j th group

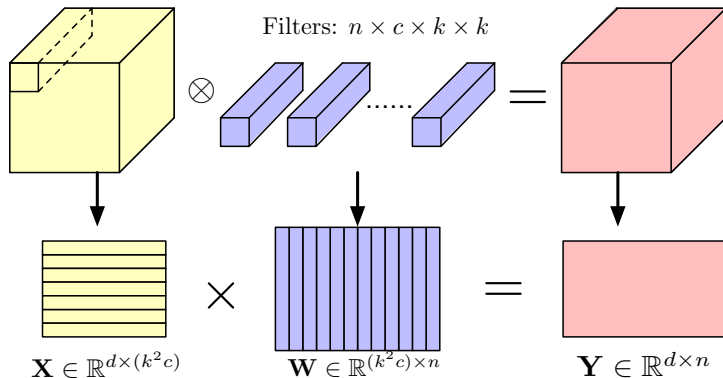
$$\begin{aligned} & \arg \min_{\beta} \|y - X\beta\|_2^2 + \sum_j \lambda_j \|\beta_j\| \\ &= \arg \min_{\beta} \|y - \sum_j X_j\beta_j\|_2^2 + \sum_j \lambda_j \|\beta_j\| \end{aligned}$$

Example:

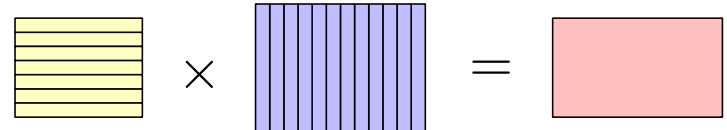




Pruning

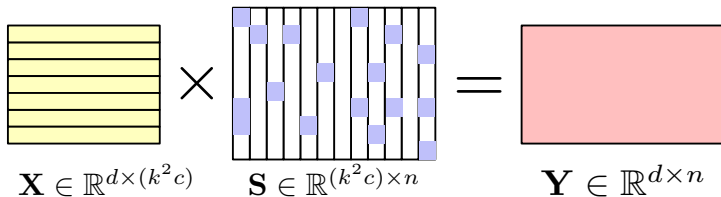


- Transform convolution to **matrix multiplication**
- **Unified** calculation for both convolution and fully-connected layers


$$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)} \quad \mathbf{W} \in \mathbb{R}^{(k^2 c) \times n} \quad \mathbf{Y} \in \mathbb{R}^{d \times n}$$

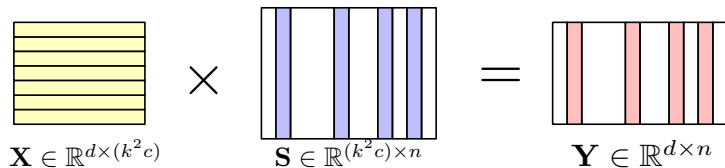
Which is better?

- Matrix approximation: $\mathbf{W} \approx \mathbf{W}'$
- Matrix regression: $\mathbf{Y} = \mathbf{W} \cdot \mathbf{X} \approx \mathbf{W}' \cdot \mathbf{X}$


$$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)} \times \mathbf{S} \in \mathbb{R}^{(k^2 c) \times n} = \mathbf{Y} \in \mathbb{R}^{d \times n}$$

Sparse DNN

- *Sparsification*: weight pruning;
- *Compression*: compressed sparse format for storage;
- *Potential acceleration*: sparse matrix multiplication algorithm.


$$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)} \times \mathbf{S} \in \mathbb{R}^{(k^2 c) \times n} = \mathbf{Y} \in \mathbb{R}^{d \times n}$$

Structured Sparse DNN

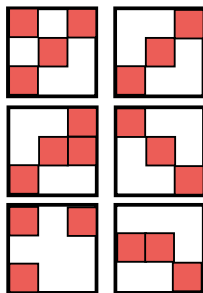
- *Potential acceleration:* GEMM or directed convolution.

Exploring the Granularity of Sparsity that is Hardware-friendly

4 types of pruning granularity



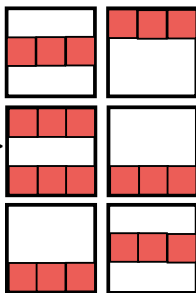
irregular sparsity



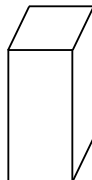
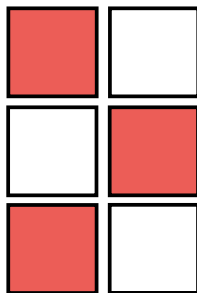
[Han et al, NIPS'15]



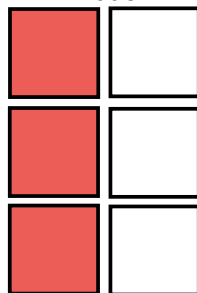
regular sparsity



more regular sparsity

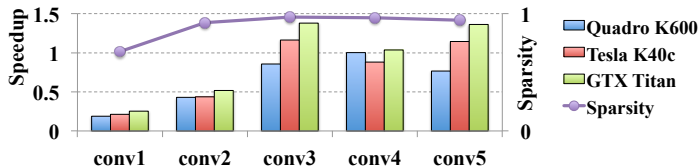


fully-dense model

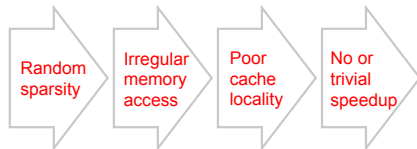


[Molchanov et al, ICLR'17]

Random sparsity, theoretical Speedup $\neq$ practical Speedup



Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.



Hardcoding nonzero weights in source code in B. Liu, etc., CVPR 2015

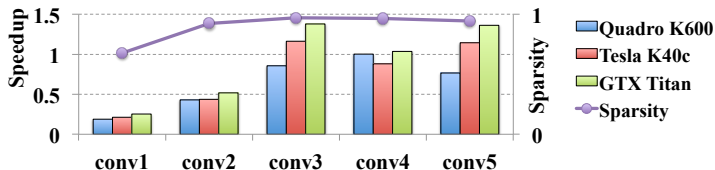
Software customization

Hardware customization

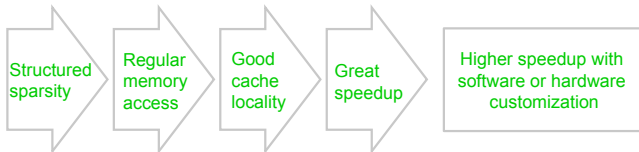
Customizing an EIE chip accelerator for compressed DNN in S. Han ISCA 2017

¹Wei Wen et al. (2016). “Learning structured sparsity in deep neural networks”. In: *Proc. NIPS*, pp. 2074–2082.

Structural Sparsity



Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.



Dense matrix to block sparse matrix

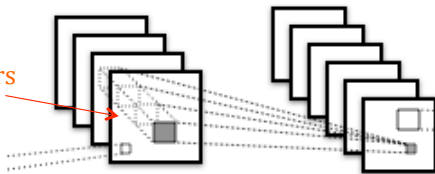
0.2	0.1	0.2	-0.6	0.1	0.4	-0.1	0.6
0.4	-0.3	0.4	0.1	0.2	-0.4	0.1	0.5
0.7	-0.1	-0.3	0.1	0.5	-0.1	0.5	0.1
-0.1	0.6	-0.5	0.3	-0.4	-0.2	0.3	0.6



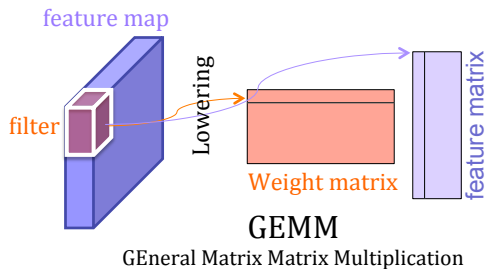
		0.2	-0.6			-0.1	0.6
		0.4	0.1			0.1	0.5
0.7	-0.1					0.5	0.1
-0.1	0.6					0.3	0.6

Removing 2D filters in convolution (2D-filter-wise sparsity)

3D filter =
stacked 2D filters



Removing rows/columns in GEMM (row/column-wise sparsity)



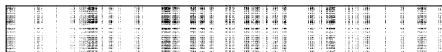
Non-structured sparsity

conv2_1: weight sparsity (col:8.7% row:19.5% elem:94.6%)



Structured sparsity

conv2_1: weight sparsity (col:75.2% row:21.9% elem:91.5%)



5.17X speedup

Group Lasso Regularization

- $E_D(W)$ is the loss on data.
- $R(\cdot)$ is non-structured regularization applying on every weight, *e.g.*, ℓ_2 -norm.
- $R_g(\cdot)$ is the structured sparsity regularization for G groups on each layer:

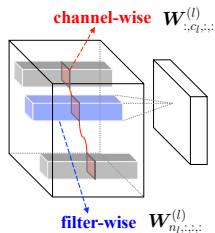
$$R_g(w) = \sum_{g=1}^G \|w^{(g)}\|_g.$$

- Here $\|\cdot\|_g$ is **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 $w^{(g)}$.

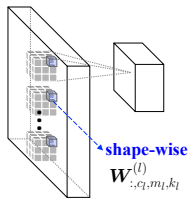
Group Lasso Regularization

Learned structured sparsity is determined by the way of splitting groups.

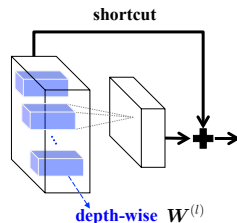
Penalize unimportant
filters and channels



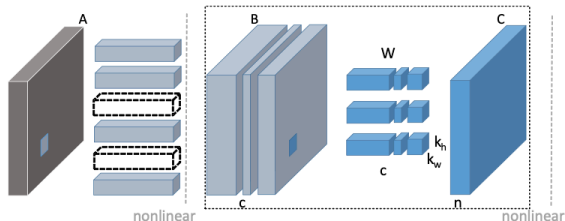
Learn filter shapes



Learn the depth of layers



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



We aim to reduce the width of feature map B, while minimizing the reconstruction error on feature map C. Our optimization algorithm performs within the dotted box, which does not involve nonlinearity. This figure illustrates the situation that two channels are pruned for feature map B. Thus corresponding channels of filters W can be removed. Furthermore, even though not directly optimized by our algorithm, the corresponding filters in the previous layer can also be removed (marked by dotted filters). c, n : number of channels for feature maps B and C, $k_h \times k_w$: kernel size.

²Yihui He, Xiangyu Zhang, and Jian Sun (2017). “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proc. ICCV*.



Formally, to prune a feature map with c channels, we consider applying $n \times c \times k_h \times k_w$ convolutional filters W on $N \times c \times k_h \times k_w$ input volumes X sampled from this feature map, which produces $N \times n$ output matrix Y . Here, N is the number of samples, n is the number of output channels, and k_h, k_w are the kernel size. For simple representation, bias term is not included in our formulation. To prune the input channels from c to desired c' ($0 \leq c' \leq c$), while minimizing reconstruction error, we formulate our problem as follow:

$$\begin{aligned} \arg \min_{\beta, W} \frac{1}{2N} \left\| Y - \sum_{i=1}^c \beta_i X_i W_i^T \right\|_F^2 \\ \text{subject to } \|\beta\|_0 \leq c' \end{aligned} \quad (1)$$

$\|\cdot\|_F$ is Frobenius norm. X_i is $N \times k_h k_w$ matrix sliced from i th channel of input volumes X , $i = 1, \dots, c$. W_i is $n \times k_h k_w$ filter weights sliced from i th channel of W . β is coefficient vector of length c for channel selection, and β_i is i th entry of β . Notice that, if $\beta_i = 0$, X_i will be no longer useful, which could be safely pruned from feature map. W_i could also be removed.

²Yihui He, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: *Proc. ICCV*.



Solving this ℓ_0 minimization problem in Eqn. 1 is NP-hard. we relax the ℓ_0 to ℓ_1 regularization:

$$\begin{aligned} \arg \min_{\beta, W} \frac{1}{2N} \left\| Y - \sum_{i=1}^c \beta_i X_i W_i^\top \right\|_F^2 + \lambda \|\beta\|_1 \\ \text{subject to } \|\beta\|_0 \leq c', \forall i \|W_i\|_F = 1 \end{aligned} \quad (2)$$

λ is a penalty coefficient. By increasing λ , there will be more zero terms in β and one can get higher speed-up ratio. We also add a constrain $\forall i \|W_i\|_F = 1$ to this formulation, which avoids trivial solution. Now we solve this problem in two folds. First, we fix W , solve β for channel selection. Second, we fix β , solve W to reconstruct error.

²Yihui He, Xiangyu Zhang, and Jian Sun (2017). “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proc. ICCV*.

(i) The subproblem of β : In this case, W is fixed. We solve β for channel selection.

$$\hat{\beta}^{LASSO}(\lambda) = \arg \min_{\beta} \frac{1}{2N} \left\| Y - \sum_{i=1}^c \beta_i Z_i \right\|_F^2 + \lambda \|\beta\|_1 \quad (3)$$

subject to $\|\beta\|_0 \leq c'$

Here $Z_i = X_i W_i^\top$ (size $N \times n$). We will ignore i th channels if $\beta_i = 0$.

(ii) The subproblem of W : In this case, β is fixed. We utilize the selected channels to minimize reconstruction error. We can find optimized solution by least squares:

$$\arg \min_{W'} \left\| Y - X'(W')^\top \right\|_F^2 \quad (4)$$

Here $X' = [\beta_1 X_1 \ \beta_2 X_2 \ \dots \ \beta_i X_i \ \dots \ \beta_c X_c]$ (size $N \times ck_h k_w$). W' is $n \times ck_h k_w$ reshaped W , $W' = [W_1 \ W_2 \ \dots \ W_i \ \dots \ W_c]$. After obtained result W' , it is reshaped back to W . Then we assign $\beta_i \leftarrow \beta_i \|W_i\|_F$, $W_i \leftarrow W_i / \|W_i\|_F$.

Constrain $\forall i \|W_i\|_F = 1$ satisfies.

²Yihui He, Xiangyu Zhang, and Jian Sun (2017). “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proc. ICCV*.