CENG 5030 Energy Efficient Computing

Lecture 03: Pruning

Bei Yu

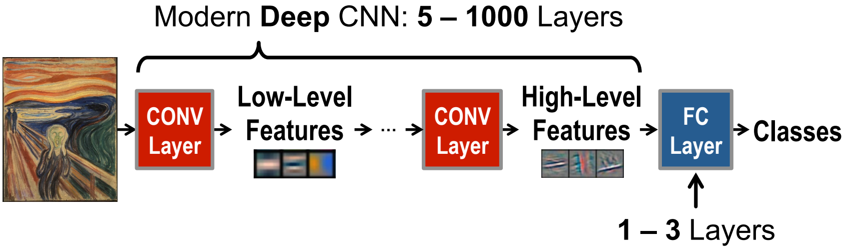
(Latest update: December 19, 2020)

Spring 2021

These slides contain/adapt materials developed by

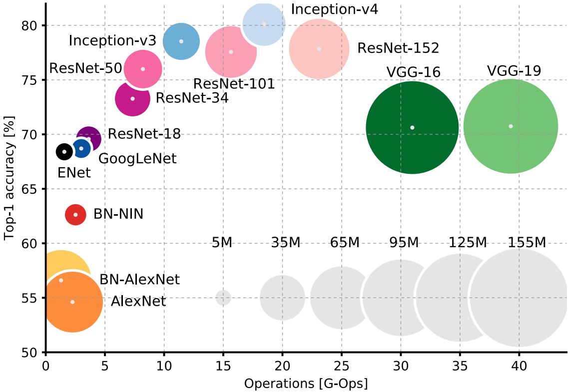
- Wei Wen et al. (2016). "Learning structured sparsity in deep neural networks". In: *Proc. NIPS*, pp. 2074–2082
- Yihui He, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: *Proc. ICCV*
- Ruichi Yu et al. (2018). "NISP: Pruning networks using neuron 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

Deeper and Larger Networks



- 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

Memory and Computations



- 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.

Overview

Sparse Regression

Pruning

Sparse Hardware Architecture

Overview

Sparse Regression

Pruning

Sparse Hardware Architecture

Linear Regression

Input

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

Output

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

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

Linear Regression

Input

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

Output

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

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

Objective

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

Challenges in Linear Regression

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

N: sample #
p: parameter #

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

Local Analysis

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

- ☺ Computationally efficient
- ☹ Only take into account local variation around nominal value

Local Analysis

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

- ☺ Computationally efficient
- ☹ Only take into account local variation around nominal value

Least Squares

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

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

ℓ_0-Norm Regularization

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

- ☺ Global view
- ☹ $\mathcal{NP}$-hard
- Orthogonal matching pursuit (OMP): iterative heuristics
- ☹ Computational expensive
- Good in temperature analysis, but NOT good in energy analysis

Ridge Regression

$$\underset{\boldsymbol{\beta}}{\text{arg min}}\, \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{i=j}^{p} \|\beta_j\|_2^2$$

Ridge Regression

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

$$\rightarrow \quad \boldsymbol{\beta} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

Lasso

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

- "ℓ_1 penalty" (Lasso)
- $\boldsymbol{\beta}$ optimally solved by Coordinate Descent [Friedman+,AOAS'07]
- λ: nonnegative regularization parameter

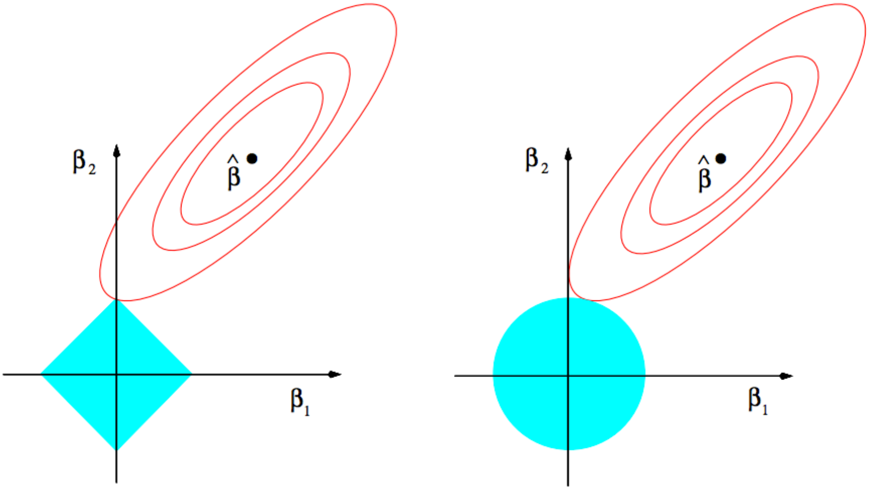
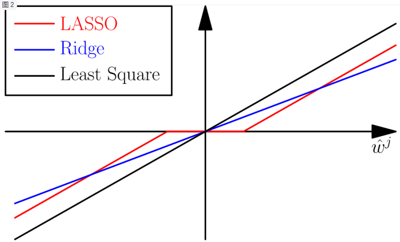


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

Closed-Form For Single Variable



Coordinate Descent

- 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

Coordinate Descent (cont.)

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

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

- Let

$$\tilde{r}_{ij} = y_i - \sum_{k\neq j} x_{ik}\widetilde{\beta}_k$$

$$\tilde{z}_j = n^{-1}\sum_{i=1}^{n} x_{ij}\tilde{r}_{ij},$$

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 $\{\tilde{r}_{ij}, x_{ij}\}_{i=1}^{n}$

Coordinate Descent (cont.)

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

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

- This suggests the following algorithm:

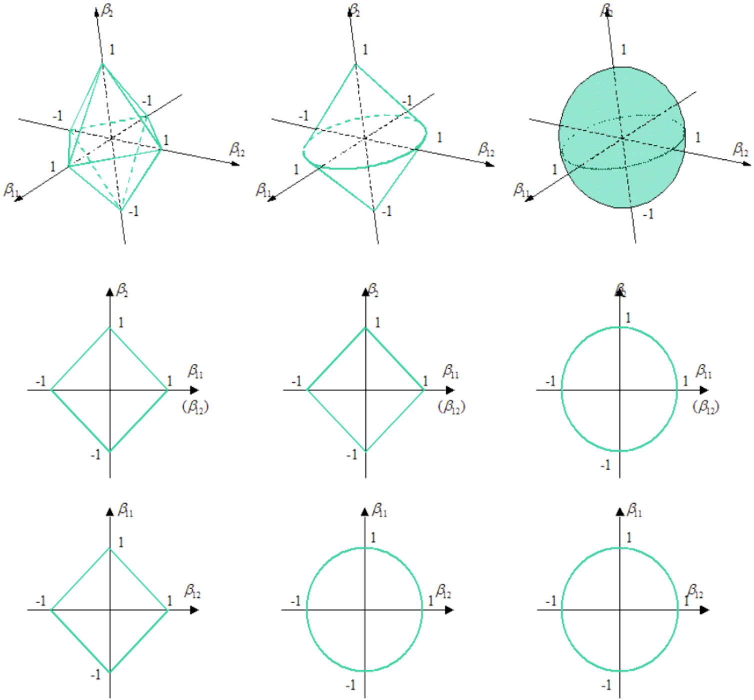
repeat
    for $j = 1, 2, \ldots, p$
        $\tilde{z}_j = n^{-1}\sum_{i=1}^{n} x_{ij}r_i + \widetilde{\beta}_j^{(s)}$
        $\widetilde{\beta}_j^{(s+1)} \leftarrow S(\tilde{z}_j|\lambda)$
        $r_i \leftarrow r_i - (\widetilde{\beta}_j^{(s+1)} - \widetilde{\beta}_j^{(s)})x_{ij}$ for all i.
until convergence

Group Lasso

- We denote $\boldsymbol{X}$ as being composed of J groups $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_J$
- $\boldsymbol{X}\boldsymbol{\beta} = \sum_j \boldsymbol{X}_j\boldsymbol{\beta}_j$, where $\boldsymbol{\beta}_j$ represents the coefficients belonging to the jth group

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

Example:


β_2
1
-1
-1
1
1
-1
β_{12}
β_{11}
β_2
1
-1
1
-1
1
-1
β_{12}
β_{11}
β_2
1
-1
1
1
-1
β_{12}
β_{11}
β_2
1
-1
1
-1
β_{11}
(β_{12})
β_2
1
-1
1
-1
β_{11}
(β_{12})
β_2
1
-1
1
-1
β_{11}
(β_{12})
β_{11}
1
-1
1
-1
β_{12}
β_{11}
1
-1
1
-1
β_{12}
β_{11}
1
-1
1
-1
β_{12}

Overview

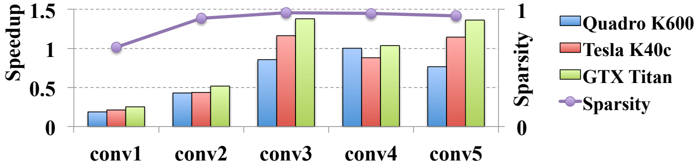
Sparse Regression

Pruning

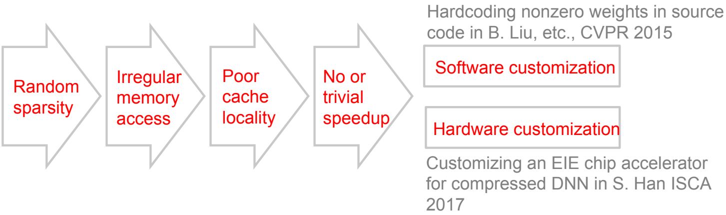
Sparse Hardware Architecture

Structured Sparsity Learning[1]

Random sparsity, theoretical Speedup $\neq$ practical Speedup



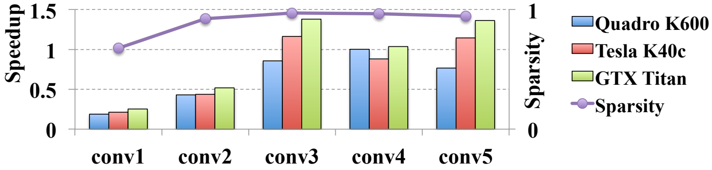
Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.



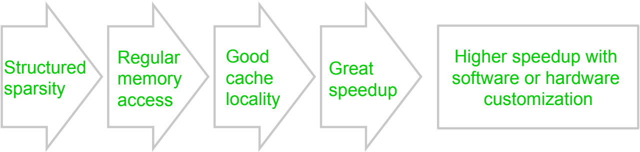
[1] Wei Wen et al. (2016). “Learning structured sparsity in deep neural networks”. In: *Proc. NIPS*, pp. 2074–2082.

Structured Sparsity Learning

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.



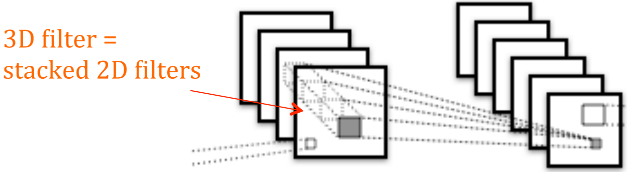
Structural Sparsity Learning – Some Examples

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

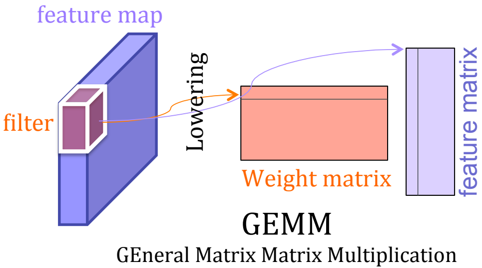
		0.2	-0.6			-0.1	0.6
		0.4	0.1			0.1	0.5
0.7	-0.1					0.5	0.1
-0.1	0.6					0.3	0.6

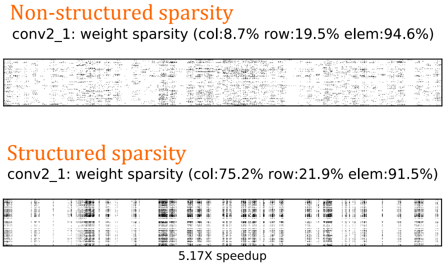
Removing 2D filters in convolution (2D-filter-wise sparsity)



Structural Sparsity Learning – Some Examples

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





Structured Sparsity Learning

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:

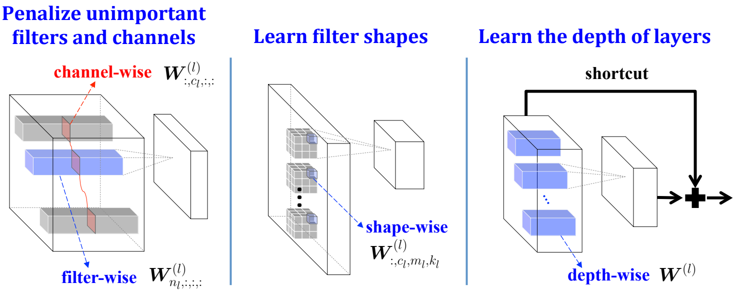
$$R_g(w) = \sum_{g=1}^{G} \|w^{(g)}\|_g.$$

- 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 $w^{(g)}$.

Structural Sparsity Learning

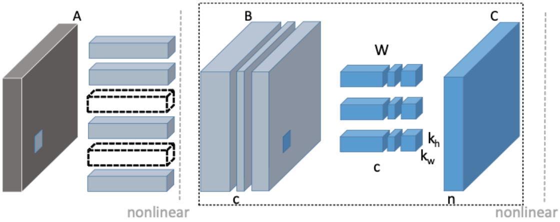
Group Lasso Regularization

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



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

Channel Pruning[2]



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.

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

Channel Pruning[2]

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:

$$\begin{aligned} &\underset{\boldsymbol{\beta}, \mathrm{W}}{\arg\min} \frac{1}{2N} \left\| \mathrm{Y} - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^\top \right\|_F^2 \\ &\text{subject to } \|\boldsymbol{\beta}\|_0 \leq c' \end{aligned} \tag{1}$$

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

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

Channel Pruning[2]

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

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

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

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

Channel Pruning[2]

(i) The subproblem of $\boldsymbol{\beta}$: In this case, W is fixed. We solve $\boldsymbol{\beta}$ for channel selection.

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

$$\text{subject to } \|\boldsymbol{\beta}\|_0 \le c'$$

Here $\mathrm{Z_i} = \mathrm{X_i}\mathrm{W_i}^\top$ (size $N \times n$). We will ignore ith channels if $\beta_i = 0$.

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

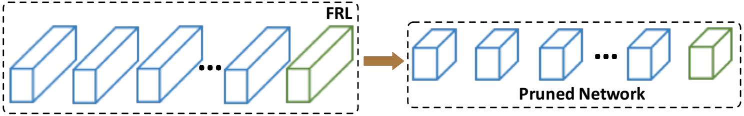
$$\arg\min_{\mathrm{W'}} \left\| \mathrm{Y} - \mathrm{X'}(\mathrm{W'})^\top \right\|_F^2 \tag{4}$$

Here $\mathrm{X'} = [\beta_1\mathrm{X_1}\ \beta_2\mathrm{X_2}\ \dots\ \beta_i\mathrm{X_i}\ \dots\ \beta_c\mathrm{X_c}]$ (size $N \times ck_hk_w$). $\mathrm{W'}$ is $n \times ck_hk_w$ reshaped W, $\mathrm{W'} = [\mathrm{W_1}\ \mathrm{W_2}\ \dots\ \mathrm{W_i}\ \dots\ \mathrm{W_c}]$. After obtained result $\mathrm{W'}$, it is reshaped back to W. Then we assign $\beta_i \leftarrow \beta_i \|\mathrm{W_i}\|_F$, $\mathrm{W_i} \leftarrow \mathrm{W_i}/\|\mathrm{W_i}\|_F$. Constrain $\forall i \; \|\mathrm{W_i}\|_F = 1$ satisfies.

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

Feature Pruning[3]

Pruning Networks using Neuron Importance Score Propagation (NISP)



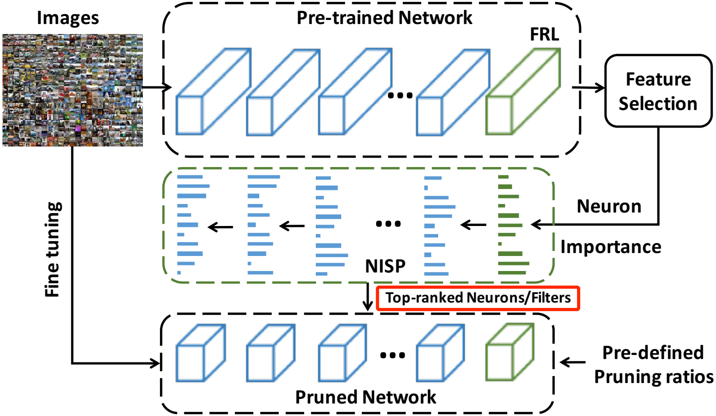
- FRL: final response layer
- Measure the importance of the neurons across the entire model;
- Rank features on the final response layer;
- Minimize the reconstruction errors of (important) final responses;
- Back-propagate the importance values and prune the neurons.

[3] Ruichi Yu et al. (2018). "NISP: Pruning networks using neuron importance score propagation". In: *Proc. CVPR*, pp. 9194–9203.

Feature Pruning

Pruning Networks using Neuron Importance Score Propagation (NISP)

- Prune network using NISP.
- Fine-tune the pruned network.



Feature Pruning

Pruning Networks using Neuron Importance Score Propagation (NISP)

Some notations:

- The l-th layer $f^{(l)}(x)$ is represented as:

$$f^{(l)}(x) = \sigma^{(l)}(w^{(l)}x + b^{(l)}).$$

Feature Pruning

Pruning Networks using Neuron Importance Score Propagation (NISP)

Some notations:

- The l-th layer $f^{(l)}(x)$ is represented as:

$$f^{(l)}(x) = \sigma^{(l)}(w^{(l)}x + b^{(l)}).$$

- A network with depth n as a function $F^{(n)}$:

$$F^{(n)} = f^{(n)} \circ f^{(n-1)} \circ \cdots \circ f^{(1)}.$$

Feature Pruning

Pruning Networks using Neuron Importance Score Propagation (NISP)

Some notations:

- The l-th layer $f^{(l)}(x)$ is represented as:

$$f^{(l)}(x) = \sigma^{(l)}(w^{(l)}x + b^{(l)}).$$

- A network with depth n as a function $F^{(n)}$:

$$F^{(n)} = f^{(n)} \circ f^{(n-1)} \circ \cdots \circ f^{(1)}.$$

- The sub-network from i-th to j-th layer:

$$G^{(i,j)} = f^{(j)} \circ f^{(j-1)} \circ \cdots \circ f^{(i)}.$$

Feature Pruning

Pruning Networks using Neuron Importance Score Propagation (NISP)

- Define a binary vector s_l^*: neuron prune indicator for the l-th layer.
- The objective function for a single sample is defined as:

$$\mathcal{F}(s_l^*|x, s_n; F) = \langle\, s_n, |F(x) - F(s_l^* \odot x)|\, \rangle,$$

where $\langle\cdot,\cdot\rangle$ is dot product, $\odot$ is element-wise product, and $|\cdot|$ is element-wise absolute value.

Feature Pruning

Pruning Networks using Neuron Importance Score Propagation (NISP)

- Define a binary vector s_l^*: neuron prune indicator for the l-th layer.
- The objective function for a single sample is defined as:

$$\mathcal{F}(s_l^* | x, s_n; F) = \langle\, s_n, |F(x) - F(s_l^* \odot x)|\, \rangle,$$

where $\langle \cdot, \cdot \rangle$ is dot product, $\odot$ is element-wise product, and $|\cdot|$ is element-wise absolute value.

- For all samples in the dataset:

$$\arg\min_{s_l^*} \sum_{m=1}^{M} \mathcal{F}(s_l^* | x_l^{(m)}, s_n; G^{(l+1,n)})$$

- Derive an upper-bound on this objective and minimize the upper-bound.

Overview

Sparse Regression

Pruning

Sparse Hardware Architecture

EIE: Efficient Inference Engine on Compressed Deep Neural Network

Han et al.
ISCA 2016

Deep Learning Accelerators

- First Wave: Compute (Neu Flow)
- Second Wave: Memory (Diannao family)
- Third Wave: Algorithm / Hardware Co-Design (EIE)

Google TPU: "This unit is designed for dense matrices. Sparse architectural support was omitted for time-to-deploy reasons. Sparsity will have high priority in future designs"

EIE: the First DNN Accelerator for Sparse, Compressed Model

[Han et al. ISCA'16]

0 * A = 0	W * 0 = 0	~~2.09, 1.92~~=> 2
Sparse Weight 90% *static* sparsity	**Sparse Activation** 70% *dynamic* sparsity	**Weight Sharing** 4-bit weights

10x less computation

3x less computation

5x less memory footprint

8x less memory footprint

[Han et al. ISCA'16]

EIE: Parallelization on Sparsity

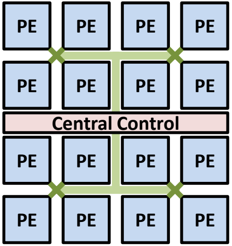
$$\vec{\boldsymbol{a}} \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

$$\left(\begin{array}{c:c:c:c} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{array}\right) = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \vec{\boldsymbol{b}} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

[Han et al. ISCA'16]

EIE: Parallelization on Sparsity



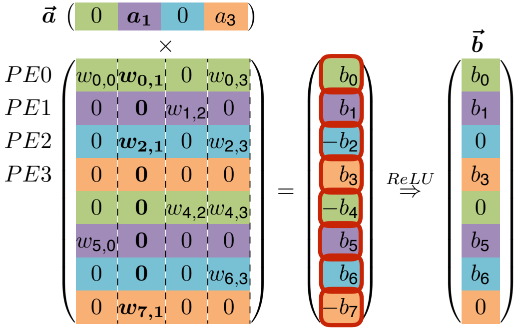
$$\vec{a} \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

$$\begin{matrix} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \vec{b} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

[Han et al. ISCA'16]

Dataflow

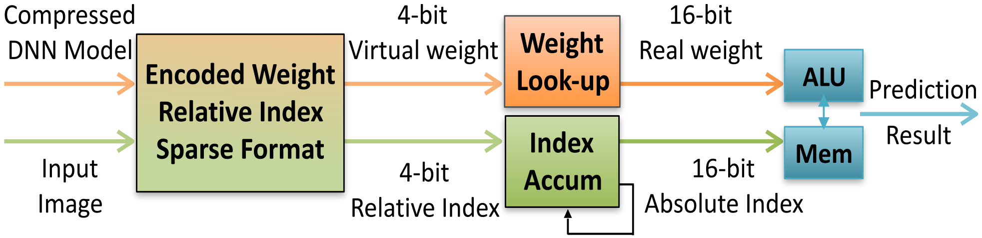


rule of thumb:

0 * A = 0   W * 0 = 0

EIE Architecture

Weight decode

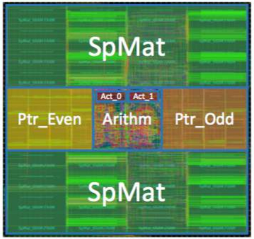


Address Accumulate

rule of thumb: 0 * A = 0 W * 0 = 0 ~~2.09, 1.92~~=> 2

[Han et al. ISCA'16]

Post Layout Result of EIE

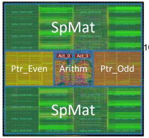


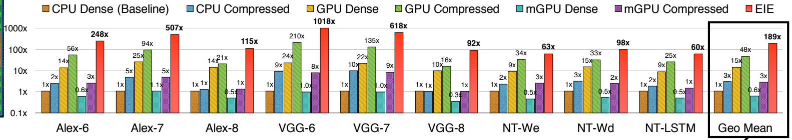
Technology	40 nm
# PEs	64
on-chip SRAM	8 MB
Max Model Size	84 Million
Static Sparsity	10x
Dynamic Sparsity	3x
Quantization	4-bit
ALU Width	16-bit
Area	40.8 mm^2
MxV Throughput	81,967 layers/s
Power	586 mW

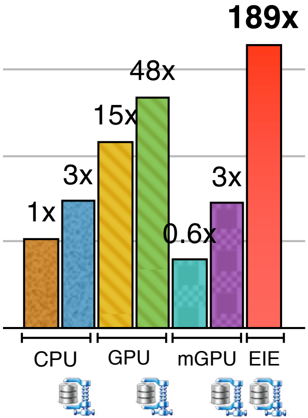
1. **Post layout result**
2. **Throughput measured on AlexNet FC-7**

[Han et al. ISCA'16]

Speedup on EIE

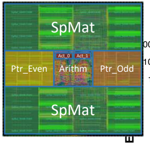


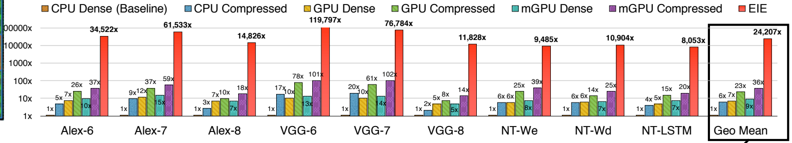


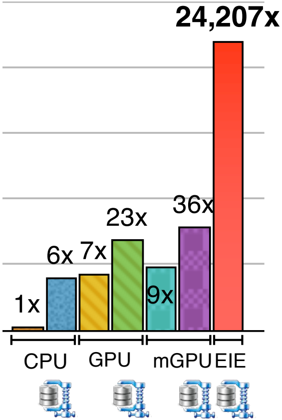


Geo Mean

Energy Efficiency on EIE



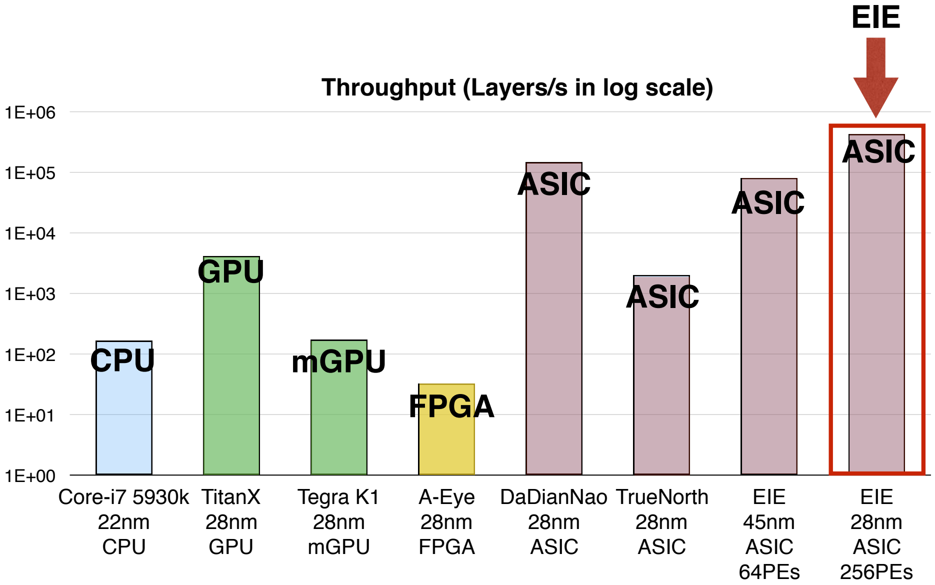




Geo Mean

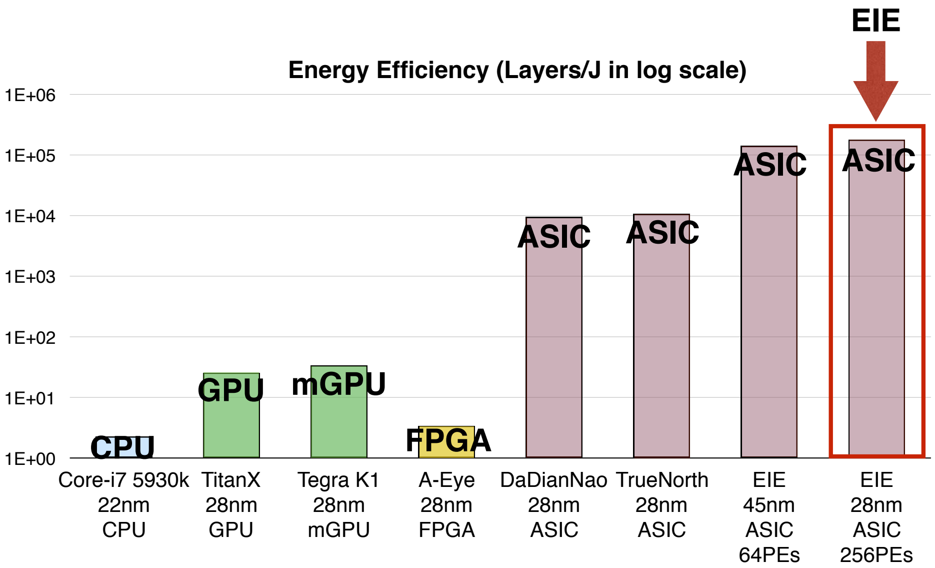
[Han et al. ISCA'16]

Comparison: Throughput



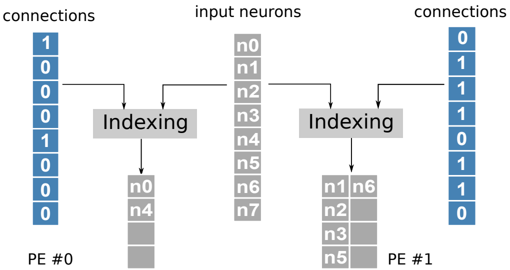
[Han et al. ISCA'16]

Comparison: Energy Efficiency



Weight Sparsity[4]

Indexing Module (IM) for sparse data



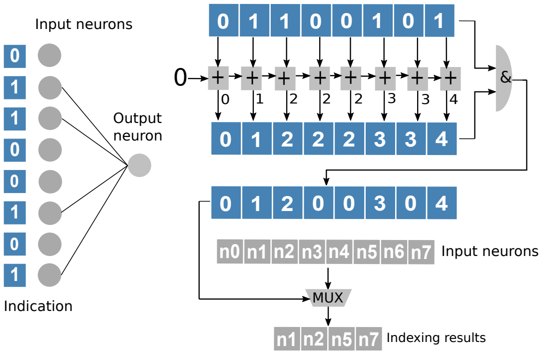
- IM is used for indexing needed neurons of sparse networks with different levels of sparsities.
- A centralized IM is designed in the buffer controller and only transfer the indexed neurons to processing engines.

---

[4] Shijin Zhang et al. (2016). "Cambricon-x: An accelerator for sparse neural networks". In: *Proc. MICRO*. IEEE, pp. 1–12.

Weight Sparsity

Direct indexing and hardware implementation

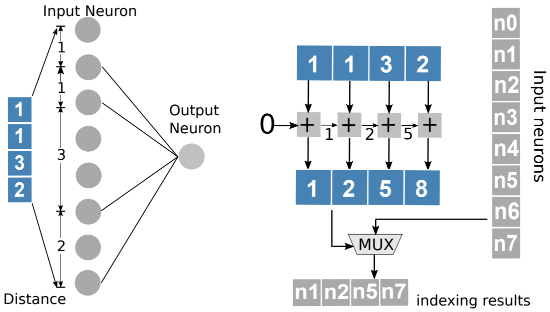


(a) (b)

- Neurons are selected from all input neurons directly based on existed connections in the binary string.

Weight Sparsity

Step indexing and hardware implementation



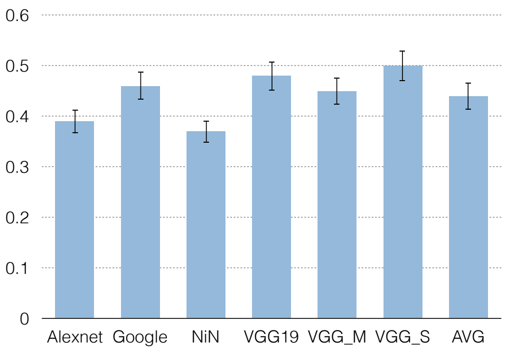
(a) (b)

- Neurons are selected based on the distances between input neurons with existed synapses.

Feature Sparsity[5]

Lots of Runtime Zeroes

Ineffectual zero computations.

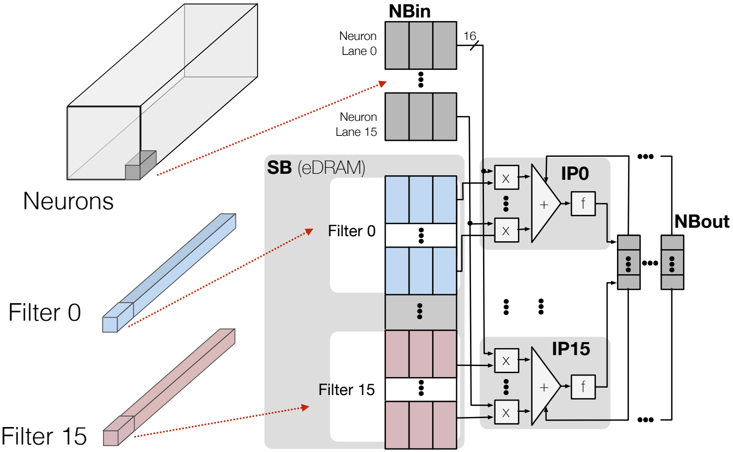


Fraction of zero neurons in multiplications

[5] Jorge Albericio et al. (2016). "Cnvlutin: Ineffectual-neuron-free deep neural network computing". In: *ACM SIGARCH Computer Architecture News* 44.3, pp. 1–13.

Feature Sparsity

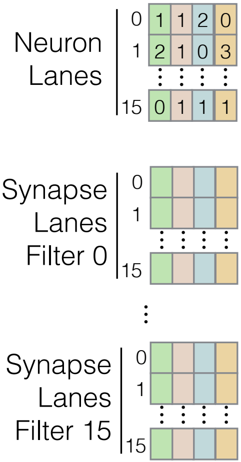
DaDianNao[6]



[6] Yunji Chen et al. (2014). “Dadiannao: A machine-learning supercomputer”. In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, pp. 609–622.

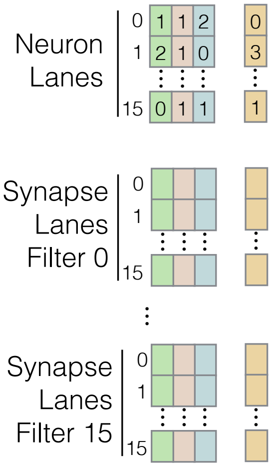
Feature Sparsity

Processing in DaDianNao



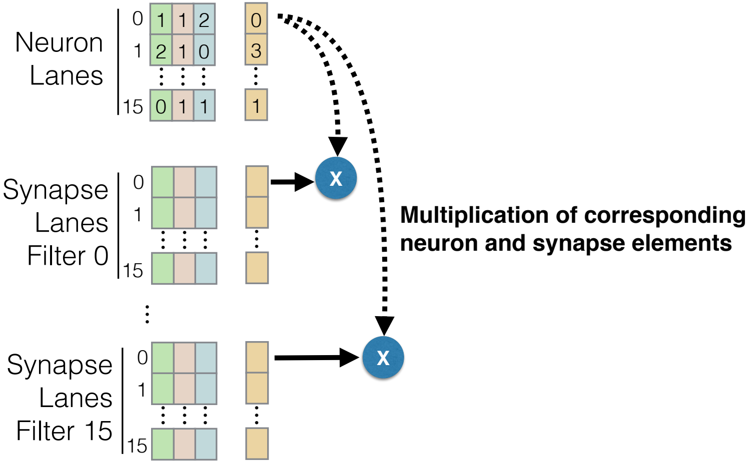
Feature Sparsity

Processing in DaDianNao



Feature Sparsity

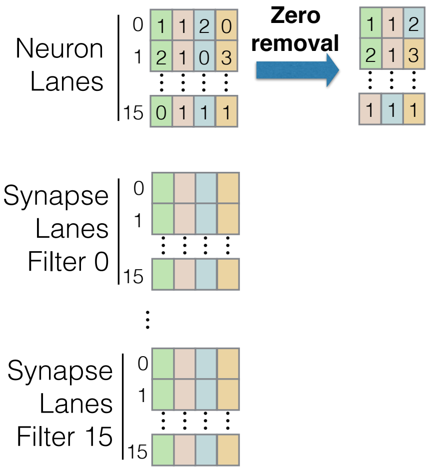
Processing in DaDianNao



Feature Sparsity

Processing in DaDianNao

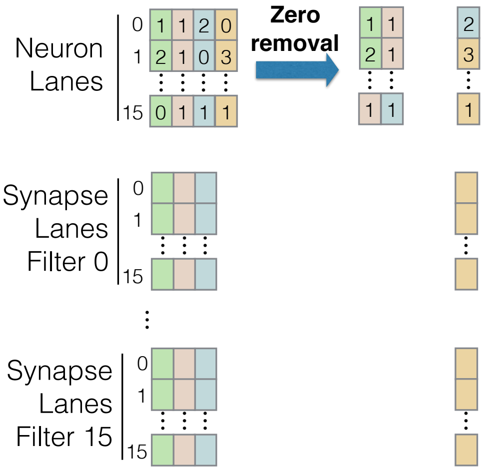
Zero removal.



Feature Sparsity

Processing in DaDianNao

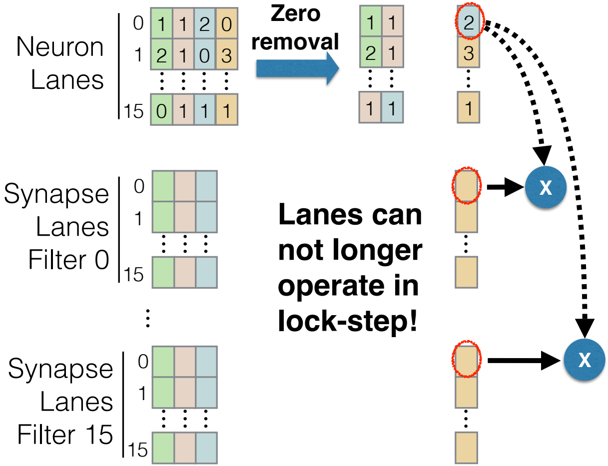
Zero removal.



Feature Sparsity

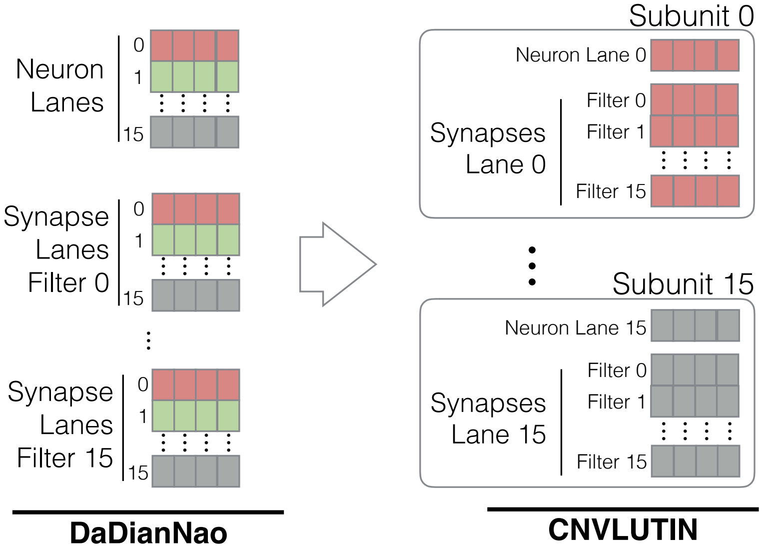
Processing in DaDianNao

Lanes can not longer operate in lock-step.



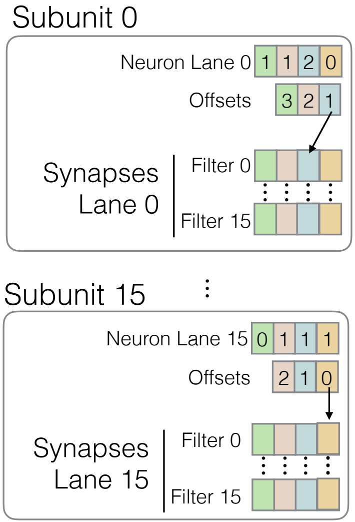
Feature Sparsity

CNVLUTIN: Decoupling Lanes



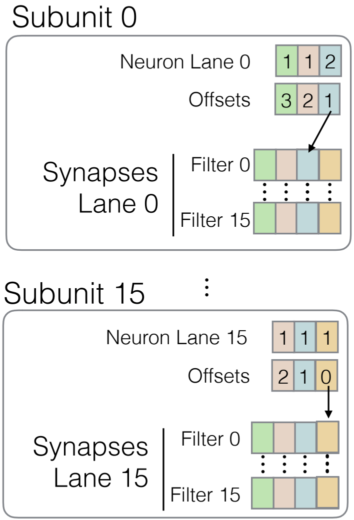
Feature Sparsity

CNVLUTIN: Decoupling Lanes



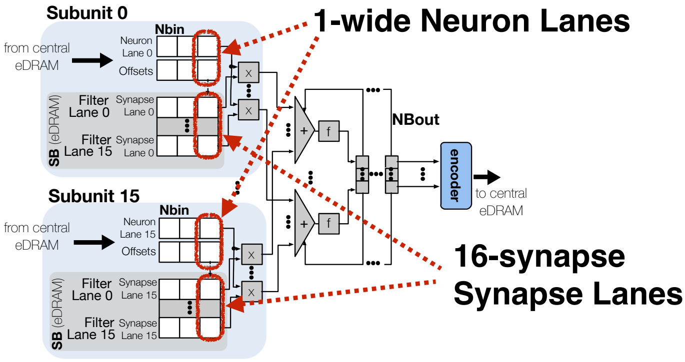
Feature Sparsity

CNVLUTIN: Decoupling Lanes



Feature Sparsity

CNVLUTIN: Decoupling Lanes



Decoupled Neuron Lanes:

Neuron + coordinate

Proceed independently

Partitioned SB:

16-wide accesses

1 synapse per filter

Further Discussion: Reading List

- Wenlin Chen et al. (2015). "Compressing neural networks with the hashing trick". In: *Proc. ICML*, pp. 2285–2294
- Huizi Mao et al. (2017). "Exploring the granularity of sparsity in convolutional neural networks". In: *CVPR Workshop*, pp. 13–20
- Zhuang Liu et al. (2017). "Learning efficient convolutional networks through network slimming". In: *Proc. ICCV*, pp. 2736–2744
- Chenglong Zhao et al. (June 2019). "Variational convolutional neural network pruning". In: *Proc. CVPR*
- Junru Wu et al. (2018). "Deep k-Means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions". In: *Proc. ICML*