Network Compression

AI602: Recent Advances in Deep Learning

Lecture 14

Slide made by

Jongheon Jeong and Insu Han
KAIST EE

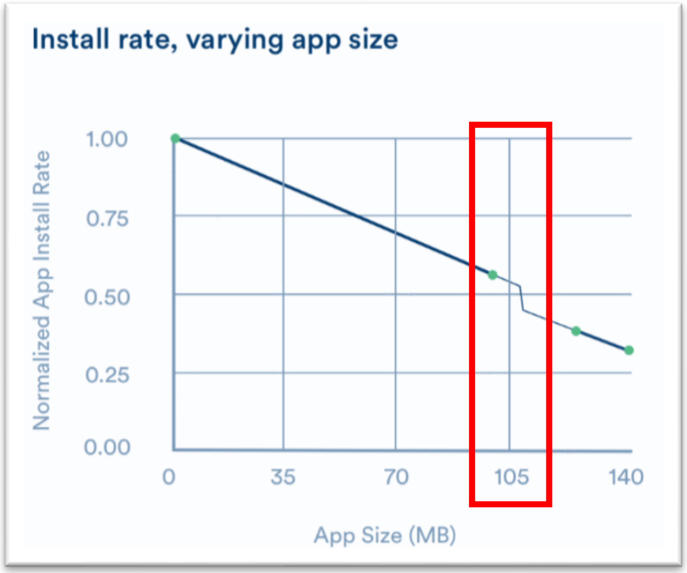
Presentation Schedule

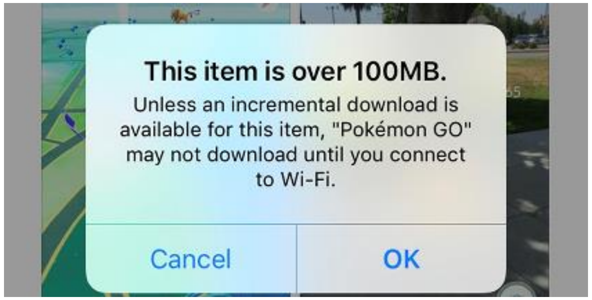
	11/21	11/26	12/10	12/12
1	정성희	David Albert	김도연	이동헌
2	김두연	Divyam Madaan	함동훈	이홍희
3	탁지훈	김경만	김재형	신재웅
4	박주혜	조인영	김종협	최중원
5	홍석찬	김규석	박관용	한문수
6	김성환	박석준	조명식	김명준
7	차현탁	이선경	서석인	도승헌
8	Thomas Thanh Minh DEFARD	Xuan Thanh Nguyen	이세웅	이수열
9	Tooba Imtiaz	신유주		
10		XUAN TRUNG PHAM		

Presentation Schedule

	11/21	11/26	12/10	12/12
1	정성희	David Albert	김도연	이동헌
2	김두연	Divyam Madaan	함동훈	이홍희
3	탁지훈	김경만	김재형	신재웅
4	박주혜	조인영	김종협	최중원
5	홍석찬	김규석	박관용	한문수
6	김성환	박석준	조명식	김명준
7	차현탁	이선경	서석인	도승헌
8	Thomas Thanh Minh DEFARD	Xuan Thanh Nguyen	이세웅	이수열
9	Tooba Imtiaz	신유주		XUAN TRUNG PHAM

- **Deploying deep neural networks** (DNNs) has been increasingly difficult
  - Constraints on power consumption, memory usage, inference overhead, …
- Inference with a large-scale network consumes huge costs
- In mobile apps, such issues become more serious
  - **“The dreaded 100MB effect”**
- Can we make DNNs to perform inferences more efficiently?





*source: https://www.recode.net/2016/10/4/13151432/app-size-calculator-bloat-experiment-developers-segment

Table of Contents

Table of Contents

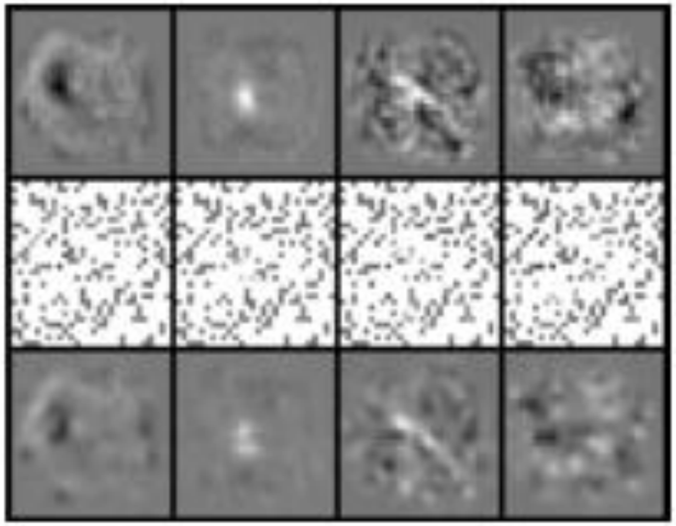
Redundancies in Deep Neural Networks [Denil et al., 2013]

- DNNs include a significant number of **redundant parameters**
- Denil et al. (2013): Predicting > 95% of weights from < 5%
  - A simple kernel ridge regression is sufficient
  - … without any drop in accuracy!
  - Many of the weights **need not be learned at all**

(a) Original weights

(b) Randomly selected

(c) Predicted from (b)

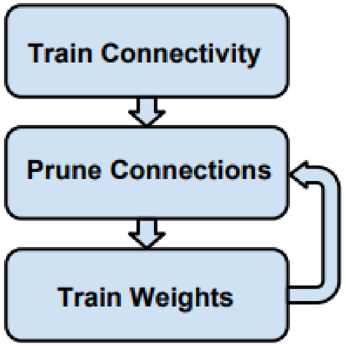


- Such redundancy can be exploited via **network pruning**

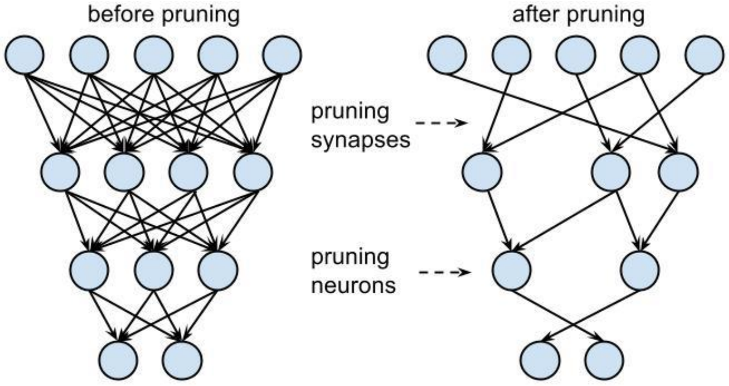
*source: Denil et al., "Predicting Parameters in Deep Learning", NIPS 2013

- Determining **low-saliency parameters**, given a pre-trained network
- Follows the framework proposed by LeCun et al. (1990):

1. **Train** a deep model until convergence
2. **Delete** "unimportant" connections w.r.t. a certain criteria
3. **Re-train** the network
4. **Iterate** to step 2, or stop



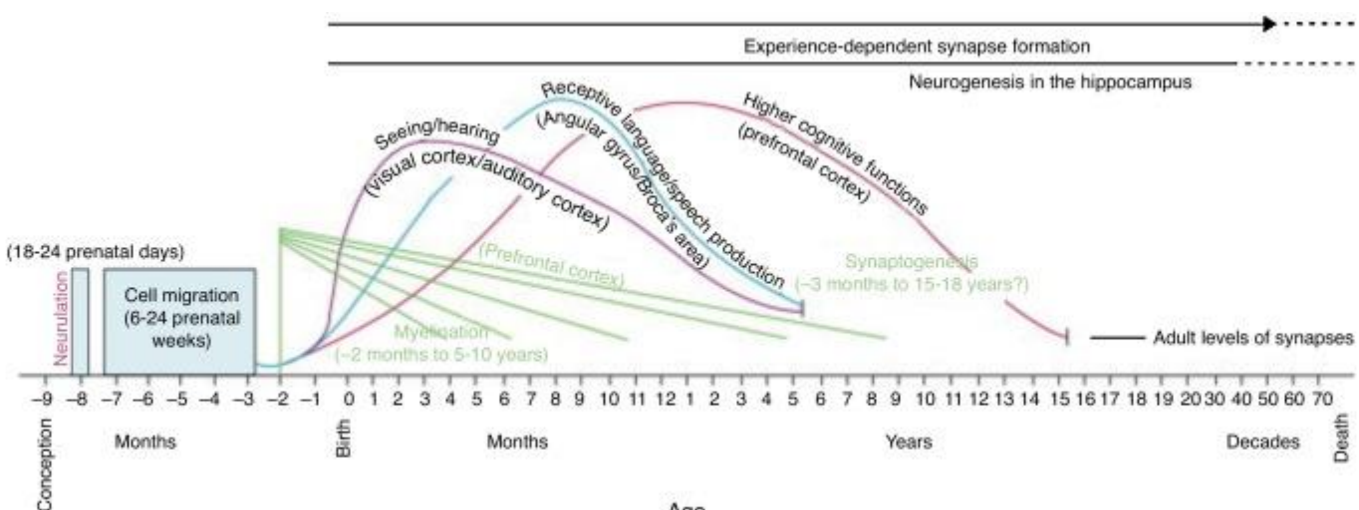
- Defining which connection is unimportant can vary
    - Weight magnitudes (L^2, L^1, …)
    - Mean activation [Molchanov et al., 2016]
    - Avg. % of Zeros (APoZ) [Hu et al., 2016]
    - Low entropy activation [Luo et al., 2017]
    - …



*source: LeCun et al., "Optimal Brain Damage", NIPS 1990

Synaptic Pruning in Human Brain

- Human brains are also using pruning schemes as well
- **Synaptic pruning** removes redundant synapses in the brain during lifetime



Next: OBD

*source: Leisman et al., "The neurological development of the child with the educational enrichment in mind.", Psicología Educativa 2015

Optimal Brain Damage (OBD) [LeCun et al., 1990]

- Network pruning **perturbs weights $\mathbf{W}$** by **zeroing** some of them
- How the loss L would be changed when $\mathbf{W}$ is perturbed?

- **OBD** approximates L by the 2^{nd} order Taylor series:

$$\delta L \simeq \underbrace{\sum_i \frac{\partial L}{\partial w_i}\delta w_i}_{\text{1st order}} + \underbrace{\frac{1}{2}\sum_i \frac{\partial^2 L}{\partial {w_i}^2}\delta w_i^2 + \frac{1}{2}\sum_{i,j} \frac{\partial^2 L}{\partial w_i \partial w_j}\delta w_i \delta w_j}_{\text{2nd order}} + O(||\delta \mathbf{W}||^3)$$

- **Problem:** Computing $H = \left(\frac{\partial L}{\partial w_i \partial w_j}\right)_{i,j}$ is usually intractable
  - Requires $O(n^2)$ on # weights
  - Neural networks usually have enormous number of weights
    - e.g. AlexNet: **60M** parameters ⇒ H consists ≈ $\mathbf{3.6 \times 10^{15}}$ elements

*source: LeCun et al., "Optimal Brain Damage", NIPS 1990

Optimal Brain Damage (OBD) [LeCun et al., 1990]

- **Problem:** Computing $H = \left(\frac{\partial L}{\partial w_i \partial w_j} \right)_{i,j}$ is usually intractable

- Two additional assumptions for tractability

  1. **Diagonal** approximation: $H = \frac{\partial^2 L}{\partial w_i \partial w_j} = 0 \quad \text{if} \quad i \neq j$
  2. **Extremal** assumption: $\frac{\partial L}{\partial w_i} = 0 \quad \forall i$
     - **W** would be in a local minima if it's pre-trained

- Now we get: $\delta L \simeq \frac{1}{2} \sum_i \frac{\partial^2 L}{\partial {w_i}^2} \delta w_i^2 + O(||\delta \mathbf{W}||^3)$
  - It only needs $\mathrm{diag}(H) := \left(\frac{\partial^2 L}{\partial w_i^2} \right)_i$

- $\mathbf{diag}(\boldsymbol{H})$ can be computed in $O(n)$, allowing a backprop-like algorithm
  - For details, see [LeCun et al., 1987]

*source: LeCun et al., "Optimal Brain Damage", NIPS 1990

Optimal Brain Damage (OBD) [LeCun et al., 1990]

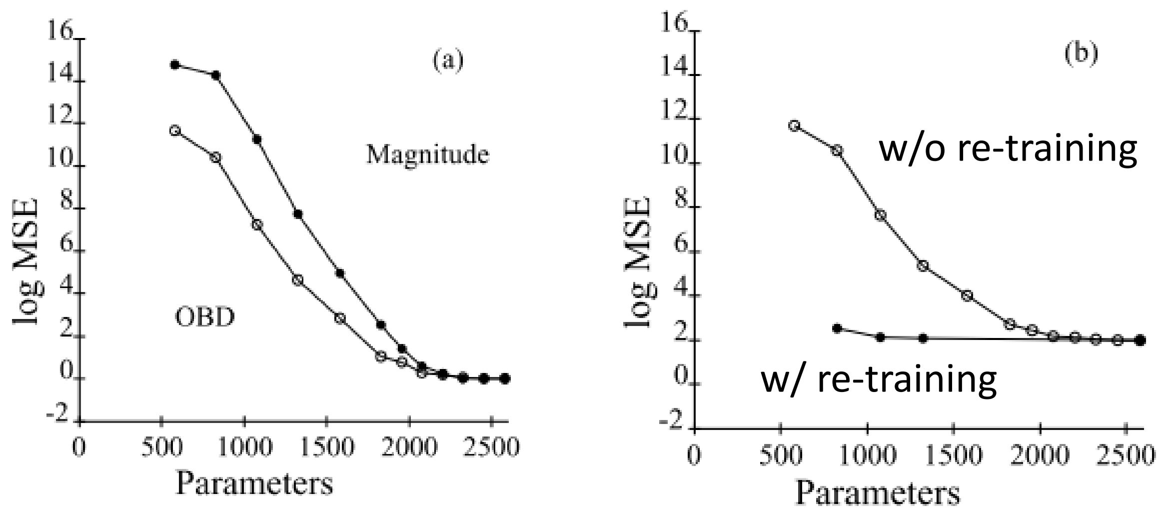
- How the loss L would be changed when $\mathbf{W}$ is perturbed?

$$L(\delta\mathbf{W}) \simeq \frac{1}{2}\sum_i \frac{\partial^2 L}{\partial {w_i}^2}\delta w_i^2 =: \sum_i \frac{1}{2} h_{ii}\delta w_i^2$$

- The **saliency** for each weight ⇒ $s_i := \frac{1}{2}h_{ii}|w_i|^2$

$\boxed{s_i := |w_i|}$

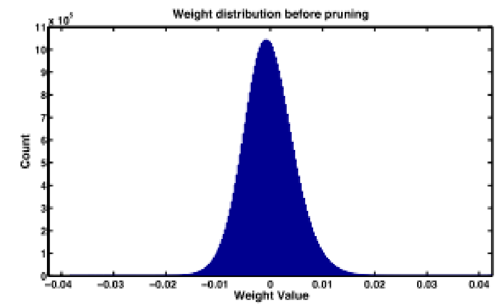
- OBD shows robustness on pruning compared to magnitude-based deletion
- After re-training, the original test accuracy is recovered

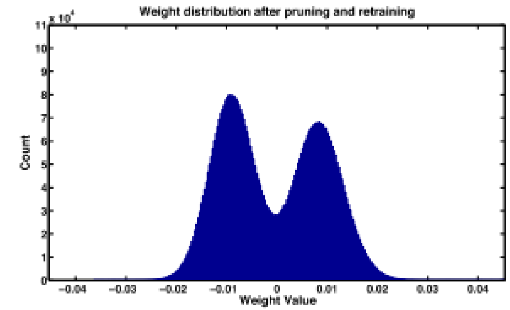


Next: Pruning modern DNNs

*source: LeCun et al., "Optimal Brain Damage", NIPS 1990

- Han et al. (2015): Pruning larger DNNs
  - LeNet, AlexNet, VGG-16, … on ImageNet
  - Highlights the **practical efficiency of pruning**





- OBD introduces extra computation on larger models
  - It requires an additional, separated backward pass
- The simple **magnitude-based pruning** works very well as long as the network is re-trained

Comparison with other model reduction methods on AlexNet

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
Baseline Caffemodel [26]	42.78%	19.73%	61.0M	1×
Data-free pruning [28]	44.40%	-	39.6M	1.5×
Fastfood-32-AD [29]	41.93%	-	32.8M	2×
Fastfood-16-AD [29]	42.90%	-	16.4M	3.7×
Collins & Kohli [30]	44.40%	-	15.2M	4×
Naive Cut	47.18%	23.23%	13.8M	4.4×
SVD [12]	44.02%	20.56%	11.9M	5×
Network Pruning	**42.77%**	**19.67%**	**6.7M**	**9×**

*source: Han et al., “Learning both Weights and Connections for Efficient Neural Networks”, NIPS 2015

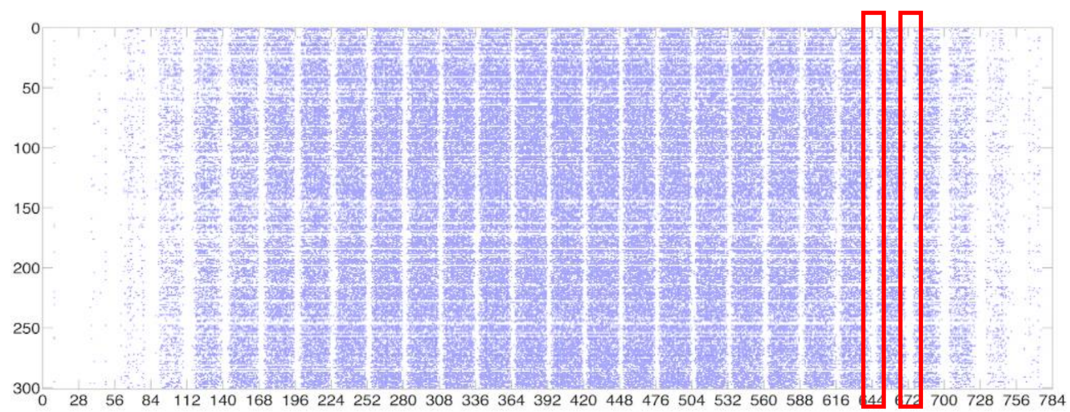
Pruning Modern DNNs [Han et al., 2015]

- Han et al. (2015): Pruning larger DNNs
  - Highlights the **practical efficiency of pruning**
- The **magnitude-based pruning** works well as long as the network is re-trained

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13**×

- Network pruning detects visual attention regions

Edge parts of MNIST images



*source: Han et al., “Learning both Weights and Connections for Efficient Neural Networks”, NIPS 2015

- The **magnitude-based pruning** works well as long as the network is re-trained

- Mittal et al. (2018): In fact, **pruning criteria are not that important**
  - … as long as the re-training phase exists
- Many strategies cannot even beat **random pruning** after fine-tuning

Heuristic	25 %	50%	75%
Random	0.650	0.569	0.415
Mean Activation	0.652	0.570	0.409
Entropy	0.641	0.549	0.405
Scaled Entropy	0.637	0.550	0.401
l_1-norm	**0.667**	**0.593**	**0.436**
APoZ	0.647	0.564	0.422
Sensitivity	0.636	0.543	0.379

Table 1: Comparison of different filter pruning strategies on VGG-16.

Heuristics	#Layers Pruned	25 %	50%	75%
Random	16	0.722	0.683	0.617
l_1-norm	16	0.714	0.677	0.610
Random	32	0.696	0.637	0.518
l_1-norm	32	0.691	0.633	0.514

Table 3: Comparison of different filter pruning strategies on ResNet (Top-1 accuracy of unpruned network is 0.745)

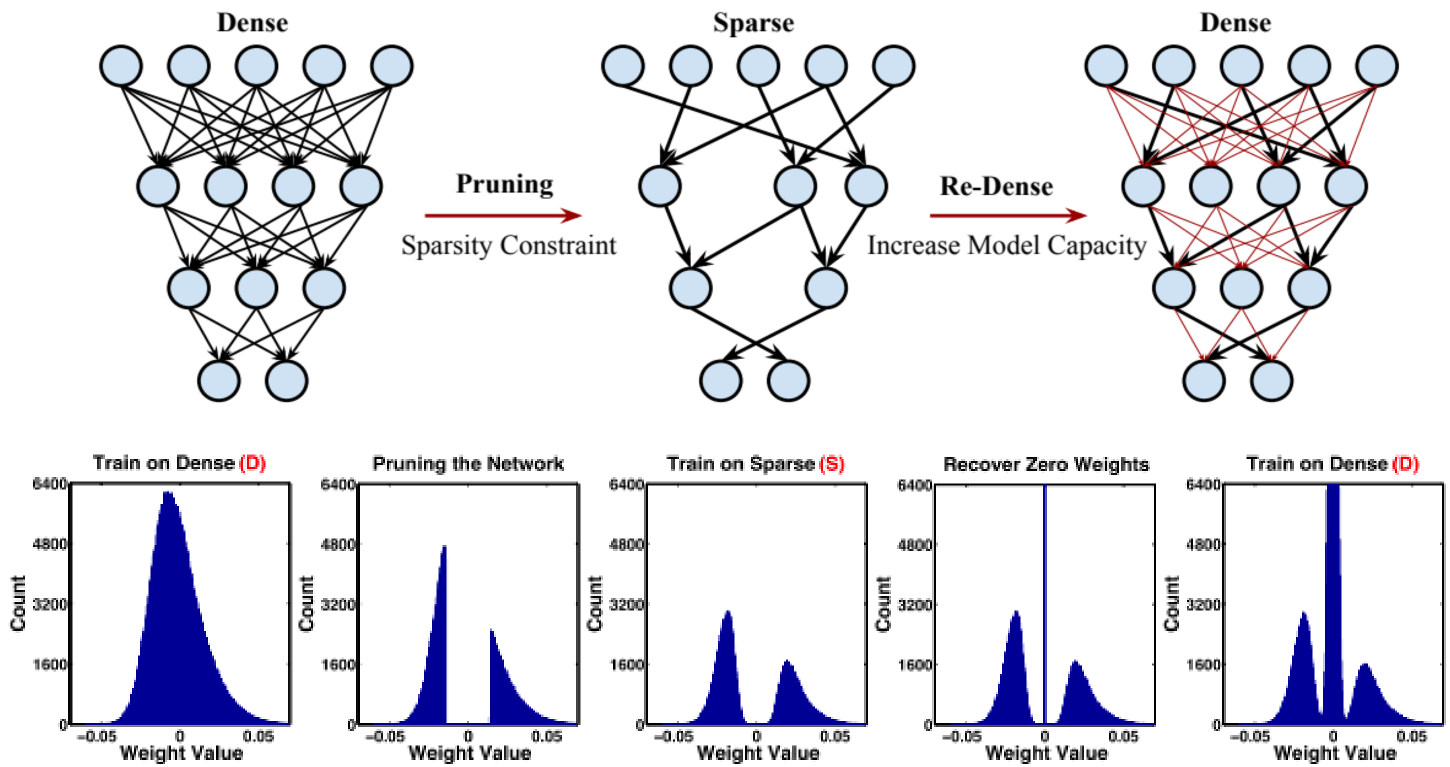
- The compressibility of DNNs are NOT due to the specific criterion
  - … but due to the inherent plasticity of DNNs

Next: Dense-Sparse-Dense

*source: Mittal et al., "Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks", WACV 2018

Network Re-wiring: Dense-Sparse-Dense Training Flow

- Network pruning preserves accuracy of the original network
- Han et al. (2017): Re-wiring the pruned connections improves DNNs further
  - **"Dense-Sparse-Dense"** training flow



*source: Han et al., "DSD: Dense-Sparse-Dense Training for Deep Neural Networks", ICLR 2017

Network Re-wiring: Dense-Sparse-Dense Training Flow

- Network pruning preserves accuracy of the original network
- Han et al. (2017): Re-wiring the pruned connections improves DNNs further
  - **"Dense-Sparse-Dense"** training flow

- Pruning discovers better optimum that the current training cannot find

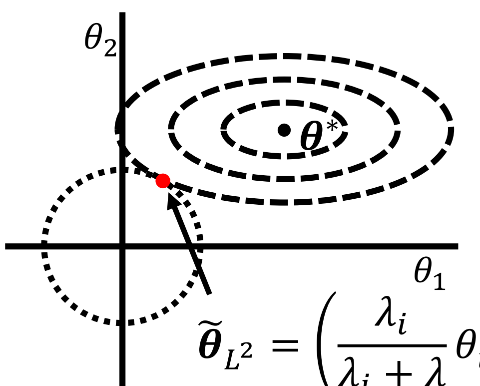
Neural Network	Domain	Dataset	Type	Baseline	DSD	Abs. Imp.	Rel. Imp.
GoogLeNet	Vision	ImageNet	CNN	31.1%[1]	**30.0%**	1.1%	3.6%
VGG-16	Vision	ImageNet	CNN	31.5%[1]	**27.2%**	4.3%	13.7%
ResNet-18	Vision	ImageNet	CNN	30.4%[1]	**29.2%**	1.2%	4.1%
ResNet-50	Vision	ImageNet	CNN	24.0%[1]	**22.9%**	1.1%	4.6%
NeuralTalk	Caption	Flickr-8K	LSTM	16.8[2]	**18.5**	1.7	10.1%
DeepSpeech	Speech	WSJ'93	RNN	33.6%[3]	**31.6%**	2.0%	5.8%
DeepSpeech-2	Speech	WSJ'93	RNN	14.5% [3]	**13.4%**	1.1%	7.4%

*source: Han et al., "DSD: Dense-Sparse-Dense Training for Deep Neural Networks", ICLR 2017

Table of Contents

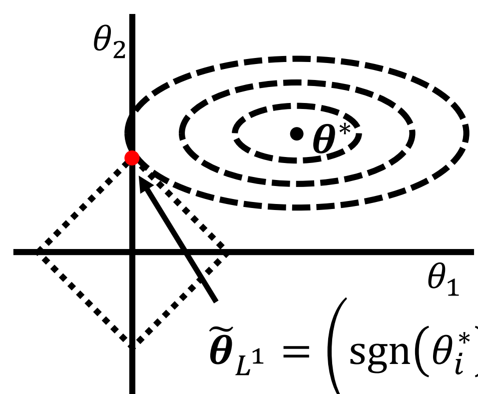
- The performance of pruning depends on the **initial training scheme**
  - e.g. Which regularization to use: L^2 or L^1?
- Which training scheme will maximize the pruning performance?
  - We still don't know about the optimal points of a DNN

- One prominent way: **Sparse network learning**
  - Inducing to a sparse solution from training a network
  - Weights with value 0 can safely be removed ⇒ it **does not** require re-training
- **Example**: L^1-regularization



$$\tilde{\boldsymbol{\theta}}_{L^2} = \left(\frac{\lambda_i}{\lambda_i + \lambda}\theta_i^*\right)_i$$

"rescaling" eigenvalues



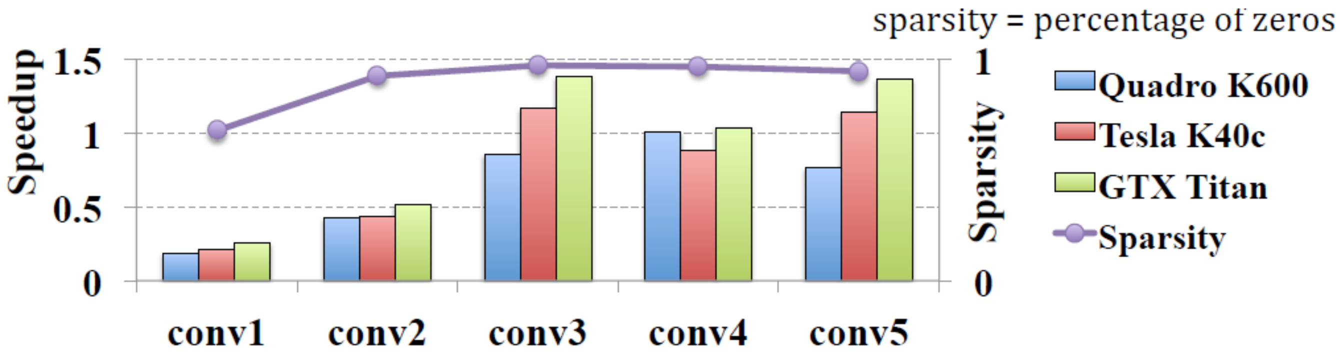
$$\tilde{\boldsymbol{\theta}}_{L^1} = \left(\mathrm{sgn}(\theta_i^*)\max\left\{|\theta_i^*| - \frac{\lambda}{\lambda_i}, 0\right\}\right)_i$$

"**sparse**" solution

Next: Structured sparsity learning

Structured Sparsity Learning [Wen et al., 2016]

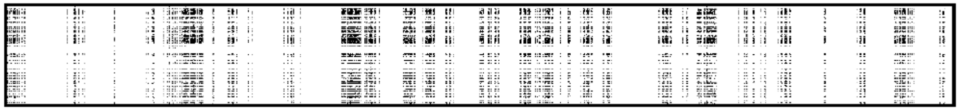
- "Un-structured" weight-level pruning may not engage a **practical speed-up**
  - Despite of extremely high sparsity, actual speed-ups in GPU is limited



Speed-up ratio of weight-level pruning

Non-structured sparsity (poor data pattern)

Structured sparsity (regular data pattern)



5× speedup after concatenation of nonzero rows and columns

*source: Wen et al., "Learning structured sparsity in deep neural networks." NIPS 2016

- **Structured sparsity** can be induced by adding group-lasso regularization

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} R_g(\mathbf{W}^{(l)}), \;\; R_g(\mathbf{w}) = \sum_{g=1}^{G} \|\mathbf{w}^{(g)}\|_2$$

groups

  - **Filter-wise** and **channel-wise**: # filters # channels

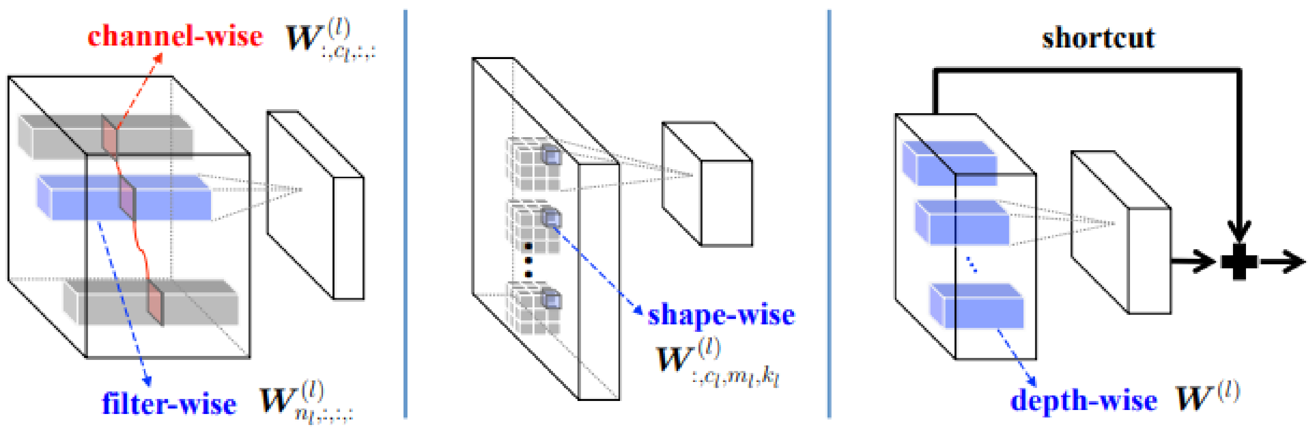
$$R_g(\mathbf{W}^{(l)}) = \sum_{n_l=1}^{N_l} \|\mathbf{W}^{(l)}_{n_l,:,:,:}\|_2 + \sum_{c_l=1}^{C_l} \|\mathbf{W}^{(l)}_{:,c_l,:,:}\|_2$$

  - **Shape-wise** sparsity: width height

$$R_g(\mathbf{W}^{(l)}) = \sum_{c_l=1}^{C_l} \sum_{m_l=1}^{M_l} \sum_{k_l=1}^{K_l} \|\mathbf{W}^{(l)}_{:,c_l,m_l,k_l}\|_2$$

  - **Depth-wise** sparsity (applicable only for ResNet):

$$R_g(\mathbf{W}^{(l)}) = \|\mathbf{W}^{(l)}\|_2$$



*source: Wen et al., "Learning structured sparsity in deep neural networks." NIPS 2016

Structured Sparsity Learning [Wen et al., 2016]

- **Structured sparsity** can be induced by adding group-lasso regularization

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} R_g(\mathbf{W}^{(l)}), \;\; R_g(\mathbf{w}) = \sum_{g=1}^{G} \|\mathbf{w}^{(g)}\|_2$$

  - **Filter-wise** and **channel-wise**:

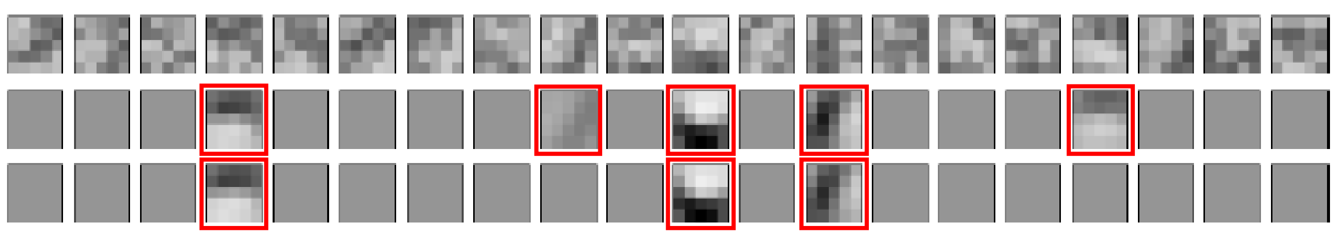
$$R_g(\mathbf{W}^{(l)}) = \underbrace{\textstyle\sum_{n_l=1}^{N_l}}_{\text{\# filters}} \|\mathbf{W}^{(l)}_{n_l,:,:,:}\|_2 + \underbrace{\textstyle\sum_{c_l=1}^{C_l}}_{\text{\# channels}} \|\mathbf{W}^{(l)}_{:,c_l,:,:}\|_2$$

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*

LeNet 1
LeNet 2
LeNet 3



conv1 filters

Fewer but smoother feature extractors

*source: Wen et al., "Learning structured sparsity in deep neural networks." NIPS 2016

- **Structured sparsity** can be induced by adding group-lasso regularization

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} R_g(\mathbf{W}^{(l)}), \;\; R_g(\mathbf{w}) = \sum_{g=1}^{G} \|\mathbf{w}^{(g)}\|_2$$

  - **Shape-wise** sparsity:

width height

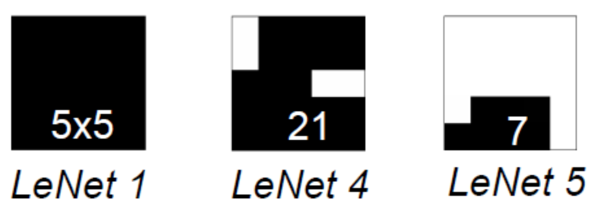
$$R_g(\mathbf{W}^{(l)}) = \sum_{c_l=1}^{C_l} \sum_{m_l=1}^{M_l} \sum_{k_l=1}^{K_l} \|\mathbf{W}^{(l)}_{:,c_l,m_l,k_l}\|_2$$

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*

Learned shapes of conv1 filters:



*source: Wen et al., "Learning structured sparsity in deep neural networks." NIPS 2016

- **Structured sparsity** can be induced by adding group-lasso regularization

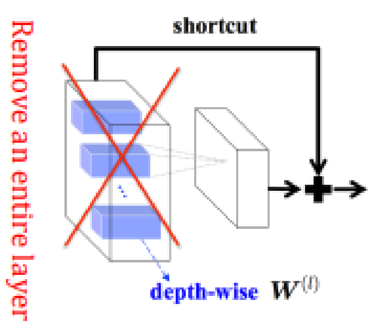
$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} R_g(\mathbf{W}^{(l)}), \;\; R_g(\mathbf{w}) = \sum_{g=1}^{G} \|\mathbf{w}^{(g)}\|_2$$

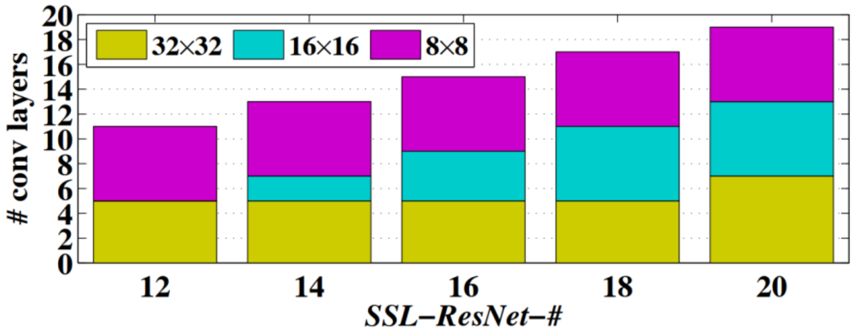
  - **Depth-wise** sparsity: $R_g(\mathbf{W}^{(l)}) = \|\mathbf{W}^{(l)}\|_2$

ResNet-20/32: baseline with 20/32 layers

SSL-ResNet-#: Ours with # layers after learning depth of ResNet-20

	# layers	error	# layers	error
ResNet	20	8.82%	32	7.51%
SSL-ResNet	**14**	**8.54%**	**18**	**7.40%**





Next: Sparsification via variational dropout

*source: Wen et al., "Learning structured sparsity in deep neural networks." NIPS 2016

Recall: Variational Dropout [Kingma et al., 2015]

- **Variational dropout** (VD) allows to learn the dropout rates separately

- Unlike dropout, VD imposes noises on **weights $\boldsymbol{\theta}$**:

$$w_i := \theta_i \cdot \xi_i, \qquad \text{where} \quad p_{\alpha_i}(\xi_i) = \mathcal{N}(1, \alpha_i)$$

  - A Bayesian generalization of Gaussian dropout [Srivastava et al., 2014]
  - $\mathbf{w} = (w_i)_i$ is adapted to data in Bayesian sense by optimizing $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$

- **Re-parametrization trick** allows **w** to be learned via minibatch-based gradient estimation methods [Kingma & Welling, 2013]
  - $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ can be optimized separated from noises

$$w_i = \theta_i + (\theta_i \sqrt{\alpha_i}) \cdot \varepsilon_i, \qquad \text{where} \quad \varepsilon_i \sim \mathcal{N}(0, 1)$$

*source : Kingma et al., "Variational dropout and the local reparametrization trick", NIPS 2015

- VD imposes noises on weights θ:

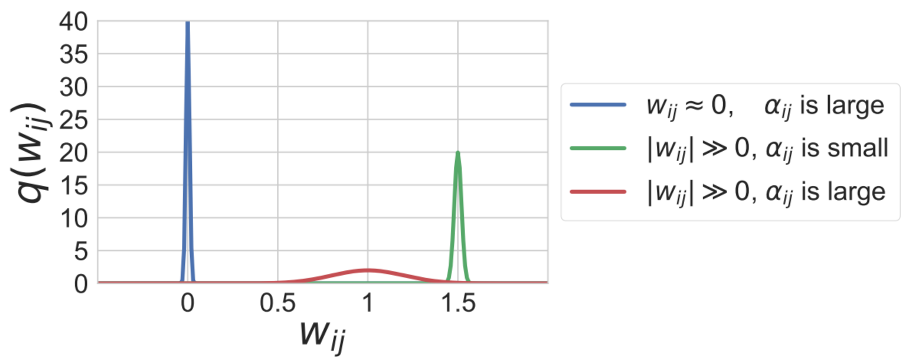
$$w_i := \theta_i \cdot \xi_i, \qquad \text{where} \quad p_{\alpha_i}(\xi_i) = \mathcal{N}(1, \boxed{\alpha_i})$$

- The original VD set a constraint $\alpha_i \leq 1$ for technical reasons
  - It corresponds to $p \leq 0.5$ in binary dropout

Q. What if $\alpha_i > 1$? What happens when $\alpha_i \to \infty$?

- $p(w_i) = \theta_i \cdot p(\xi_i) = \mathcal{N}(\theta_i, \alpha_i \theta_i^2)$
- w_i will be completely random as $\alpha_i \to \infty$
- Such w_i will **corrupt** the expected log likelihood
- … except that $\theta_i \to 0$ as well!

$$\boxed{\begin{array}{c} \theta_{ij} \to 0, \quad \alpha_{ij}\theta_{ij}^2 \to 0 \\ \Downarrow \\ q(w_{ij} \,|\, \theta_{ij}, \alpha_{ij}) \to \mathcal{N}(w_{ij} \,|\, 0, 0) = \delta(w_{ij}) \end{array}}$$



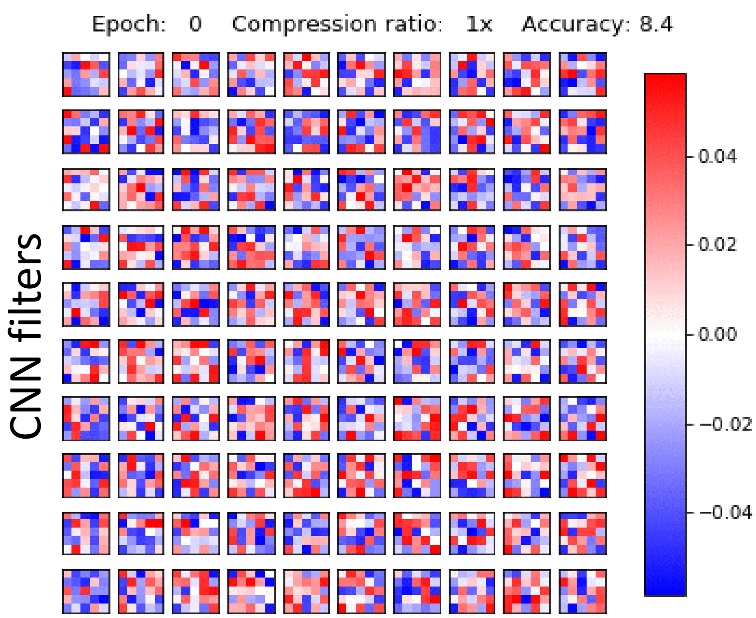
*source : Molchanov et al., “Variational Dropout Sparsifies Deep Neural Networks”, ICML 2017

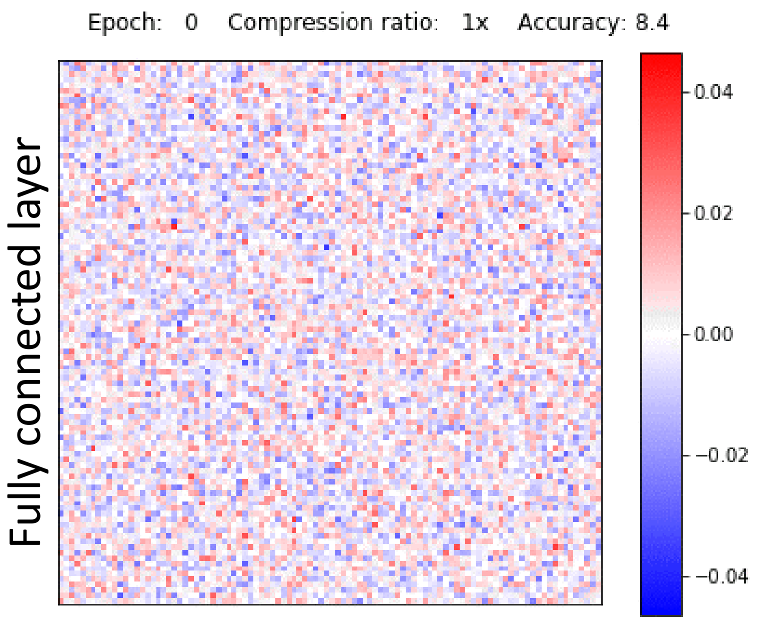
Variational Dropout Sparsifies DNNs [Molchanov et al., 2017]

Q. What if $\alpha_i > 1$? What happens when $\alpha_i \to \infty$?

- It will **corrupt** the expected log likelihood except that $\theta_i \to 0$ as well

- Molchanov et al. (2017): **Extending VD for** $\alpha_i > 1 \Rightarrow$ Super sparse solutions
  - Weights with $\log \alpha > 3$ are pruned away during training





*source : Molchanov et al., "Variational Dropout Sparsifies Deep Neural Networks", ICML 2017

Variational Dropout Sparsifies DNNs [Molchanov et al., 2017]

Q. What if $\alpha_i > 1$? What happens when $\alpha_i \to \infty$?

- It will **corrupt** the expected log likelihood except that $\theta_i \to 0$ as well

- Molchanov et al. (2017): **Extending VD for** $\alpha_i > 1 \Rightarrow$ Super sparse solutions
  - Weights with $\log \alpha > 3$ are pruned away during training

Network	Method	Error %	Sparsity per Layer %	$\frac{\lvert W \rvert}{\lvert W_{\neq 0} \rvert}$	
	Original	1.64		1	
	Pruning	1.59	$92.0 - 91.0 - 74.0$	12	[Han et al., 2015]
LeNet-300-100	DNS	1.99	$98.2 - 98.2 - 94.5$	56	
	SWS	1.94		23	
(ours)	Sparse VD	1.92	$98.9 - 97.2 - 62.0$	**68**	
	Original	0.80		1	
	Pruning	0.77	$34 - 88 - 92.0 - 81$	12	[Han et al., 2015]
LeNet-5-Caffe	DNS	0.91	$86 - 97 - 99.3 - 96$	111	
	SWS	0.97		200	
(ours)	Sparse VD	0.75	$67 - 98 - 99.8 - 95$	**280**	

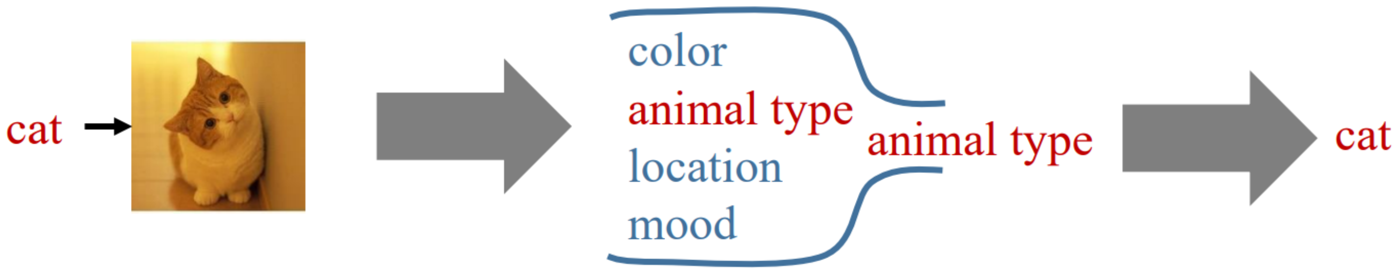
Next: Variational information bottleneck

*source : Molchanov et al., "Variational Dropout Sparsifies Deep Neural Networks", ICML 2017

Variational Information Bottleneck [Dai et al., 2018]

- **Motivation**: Markov chain interpretation of DNN [Tishby & Zaslavsky, 2015]

$$\boldsymbol{y} \to \boldsymbol{x} = \boldsymbol{h}_0 \to \boldsymbol{h}_1 \to \cdots \to \underbrace{\boldsymbol{h}_{i-1} \to \boldsymbol{h}_i}_{p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1})} \to \cdots \to \underbrace{\boldsymbol{h}_L \to \hat{\boldsymbol{y}}}_{\text{Approximate } p(\boldsymbol{y}|\boldsymbol{h}_L) \text{ via tractable } p(\hat{\boldsymbol{y}}|\boldsymbol{h}_L)}$$



1. **Maximize** $I(\boldsymbol{h}_i; \boldsymbol{y})$ for high-accuracy prediction
2. **Minimize** $I(\boldsymbol{h}_i; \boldsymbol{h}_{i-1})$ for compression ⇒ **"information bottleneck"**

- **Layer-wise losses** become:

Mutual information

$$\mathcal{L}_i = \gamma_i I(\boldsymbol{h}_i; \boldsymbol{h}_{i-1}) - I(\boldsymbol{h}_i; \boldsymbol{y})$$

The relative strength of bottleneck

*source : Dai et al., "Compressing Neural Networks using the Variational Information Bottleneck", ICML 2018

Variational Information Bottleneck [Dai et al., 2018]

- **Layer-wise losses** become $\mathcal{L}_i = \gamma_i I(\boldsymbol{h}_i; \boldsymbol{h}_{i-1}) - I(\boldsymbol{h}_i; \boldsymbol{y})$
- **Problem**: Computing $I(\cdot;\cdot)$ is usually intractable
- Instead, we minimize variational upper bound of it

$$\mathcal{L}_i \leq \tilde{\mathcal{L}}_i = \gamma_i \mathbb{E}[\mathrm{KL}(\boxed{p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1})}||\boxed{q(\boldsymbol{h}_i)})] - \mathbb{E}[\log \boxed{q(\boldsymbol{y}|\boldsymbol{h}_L)}]$$

variational approx. of $p(\boldsymbol{h}_i)$

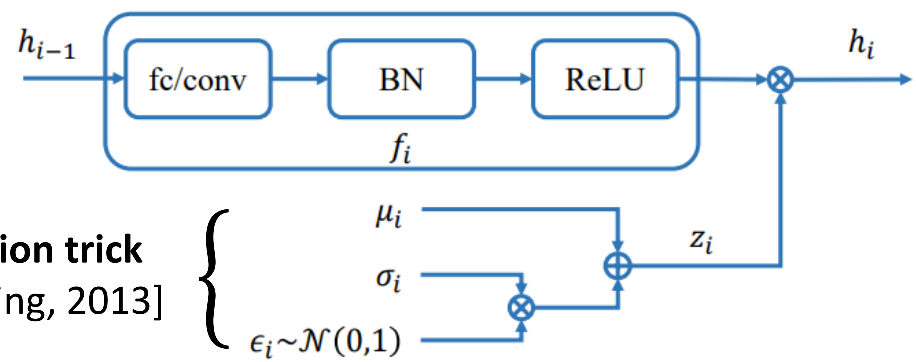
variational approx. of $p(\boldsymbol{y}|\boldsymbol{h}_L)$

$\begin{cases} \textbf{multinomal} \text{ for classification} \\ \textbf{Gaussian} \text{ for regression} \end{cases}$

- **Variational Information Bottleneck (VIB)** model

$$p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1}) := f_i(\boldsymbol{h}_{i-1}) \odot \mathcal{N}(\boldsymbol{h}_i|\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2))$$
$$q(\boldsymbol{h}_i) := \mathcal{N}(\boldsymbol{h}_i|\boldsymbol{0}, \mathrm{diag}(\boldsymbol{\xi}_i))$$

Reparametrization trick
[Kingma & Welling, 2013]



*source : Dai et al., "Compressing Neural Networks using the Variational Information Bottleneck", ICML 2018

Variational Information Bottleneck [Dai et al., 2018]

- We minimize variational upper bound of $\mathcal{L}_i$

$$\mathcal{L}_i \leq \tilde{\mathcal{L}}_i = \gamma_i \mathbb{E}[\mathrm{KL}(p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1})||q(\boldsymbol{h}_i))] - \mathbb{E}[\log q(\boldsymbol{y}|\boldsymbol{h}_L)]$$

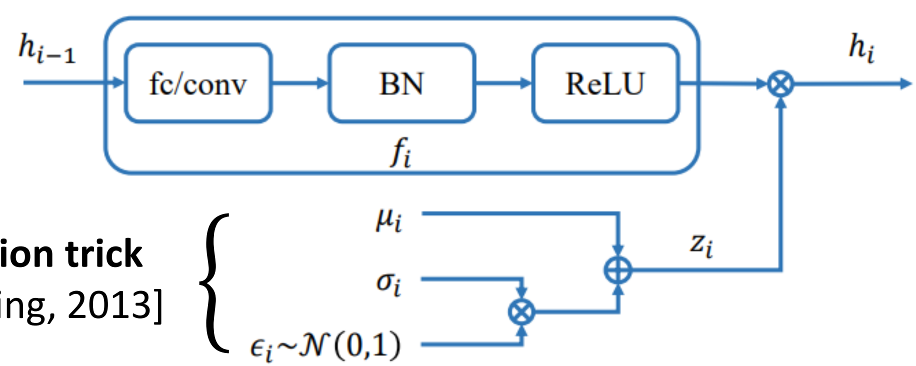
- Final variational objective function (**VIBNet**):

layers

$$\tilde{\mathcal{L}} = \underbrace{\sum_{i=1}^{L} \gamma_i \sum_j \log\left(1 + \boxed{\frac{\mu_{ij}^2}{\sigma_{ij}^2}}\right)}_{\text{regularization}} - \underbrace{L \cdot \mathbb{E}[\log q(\boldsymbol{y}|\boldsymbol{h}_L)]}_{\text{data-fit}}$$

- **Pruning criteria**: $\alpha_{ij} := \frac{\mu_{ij}^2}{\sigma_{ij}^2} \to 0$
    - Neurons with low value of α_{ij}'s are pruned after training

Reparametrization trick
[Kingma & Welling, 2013]



*source : Dai et al., "Compressing Neural Networks using the Variational Information Bottleneck", ICML 2018

- **VIBNet** outperforms various methods by large margins
  - $r_W(\%)$: ratio of # parameters
  - $r_N(\%)$: ratio of memory footprint

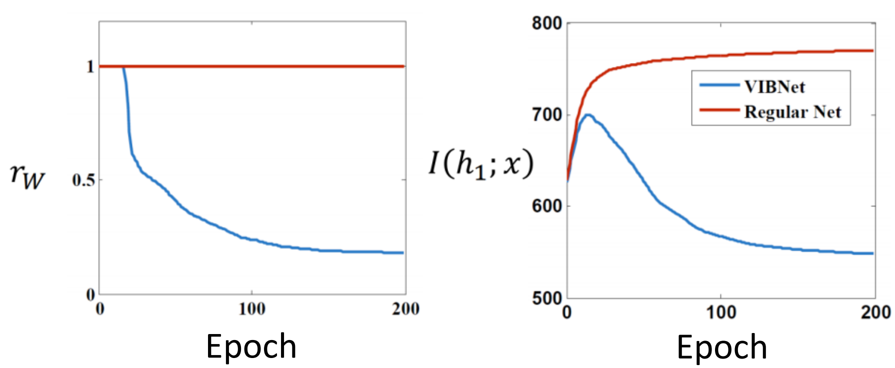
Method	$r_W(\%)$	$r_N(\%)$	error(%)	Pruned Model
VD	25.28	58.95	1.8	512-114-72
BC-GNJ	10.76	32.85	1.8	278-98-13
BC-GHS	10.55	34.71	1.8	311-86-14
L0	26.02	45.02	**1.4**	219-214-100
L0-sep	10.01	32.69	1.8	266-88-33
DN	23.05	57.94	1.8	542-83-61
VIBNet	**3.59**	**16.98**	1.6	**97-71-33**

Table 1. Compression results on MNIST using LeNet-300-100.

After fine-tuning

Method	$r_W(\%)$	FLOP(Mil)	$r_N(\%)$	error(%)
BC-GNJ	6.57	141.5	81.68	8.6
BC-GHS	5.40	121.9	74.82	9.0
VIBNet	**5.30**	**70.63**	**49.57**	8.8 (**8.5**)
PF	35.99	206.3	83.97	6.6
SBP	7.01	136.0	80.72	7.5
SBPa	5.78	99.20	66.46	9.0
VIBNet	**5.45**	**86.82**	**57.86**	6.5 (**6.1**)
NS-Single	11.50	195.5	-	6.2
NS-Best	8.60	147.0	-	5.9
VIBNet	**5.79**	**116.0**	**59.60**	6.2 (**5.8**)

Table 3. Compression results on CIFAR10 using VGG-16.



Method	$r_W(\%)$	FLOP(Mil)	$r_N(\%)$	error(%)
RNP	-	160	-	38.0
VIBNet	**22.75**	**133.6**	**59.80**	37.6 (**37.4**)
NS-Single	24.90	250.5	-	26.5
NS-Best	20.80	214.8	-	26.0
VIBNet	**15.08**	**203.1**	**73.80**	25.9 (**25.7**)

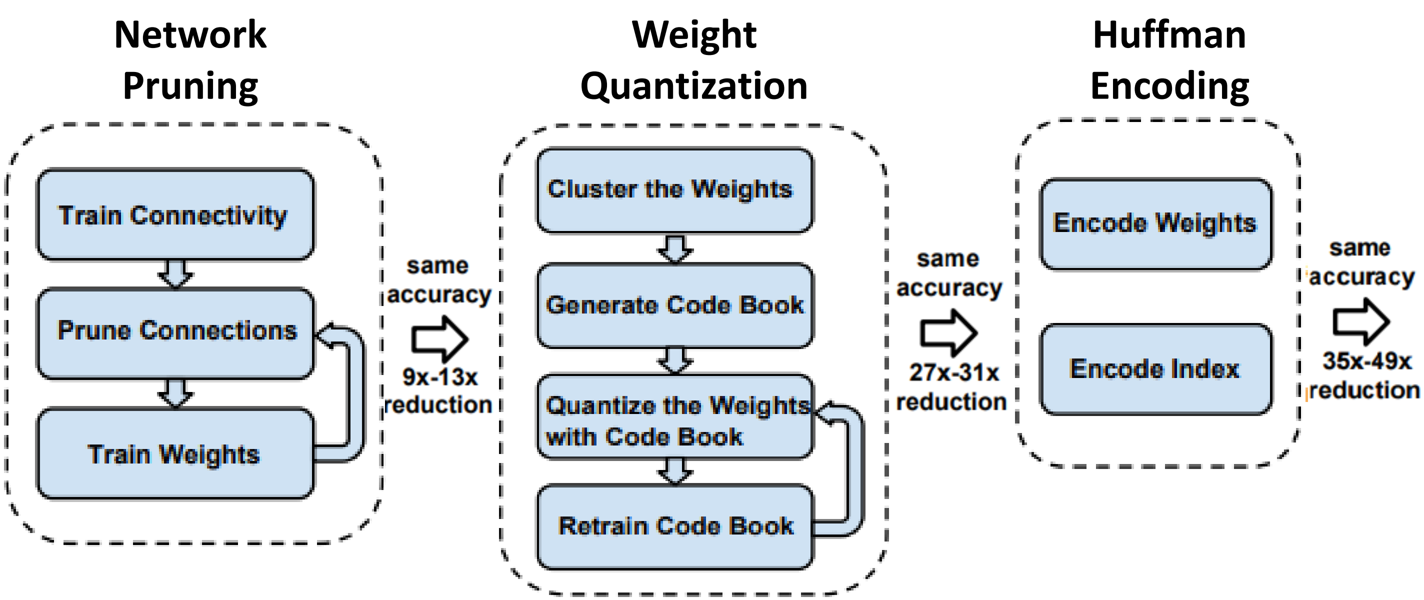
Table 4. Compression results on CIFAR100 using VGG-16.

*source : Dai et al., “Compressing Neural Networks using the Variational Information Bottleneck”, ICML 2018

Table of Contents

- **Quantizing weights** can further compress the pruned networks
  - Weights are clustered into discrete values
  - The network is represented only with several centroid values

- Han et al. (2015): Pruning DNNs ⇒ 9x-13x reduction
- Han et al. (**2016**): Pruning + Quantization + Huffman ⇒ **35x-49x** reduction



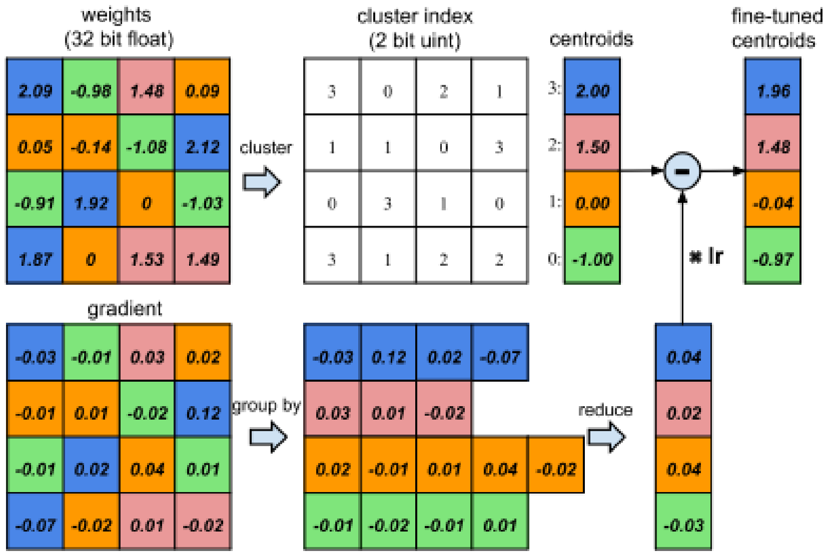
*source : Han et al., "Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding", ICLR 2016

- **Quantizing weights** can further compress the pruned networks
  - Weights are clustered into discrete values
  - The network is represented only with several centroid values

1. **Train** a deep model until convergence
2. **Find** k clusters that minimizes **within-cluster sum of squares** (WCSS):
$$\mathrm{argmin}_C \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2$$
3. **Quantize** with the cluster C via weight sharing
4. **Fine-tune** the network with the shared weights

- In the **fine-tuning** phase, gradients in each cluster are aggregated:

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k}$$
$$= \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbf{1}(W_{ij} \in C_k)$$

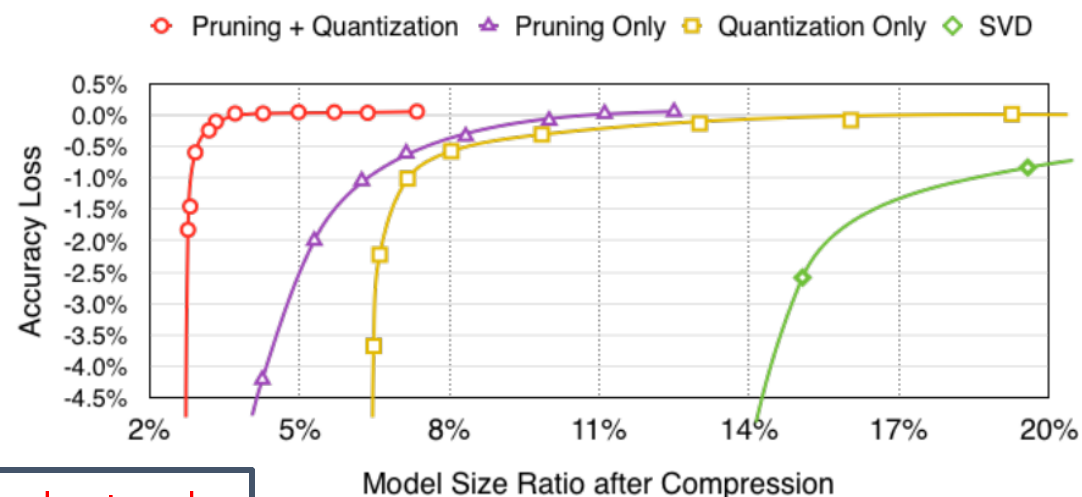


*source : Han et al., “Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”, ICLR 2016

Deep Compression [Han et al., 2016]

- **Deep compression** reduces the model size significantly

Network	Original Size	Compressed Size	Compression Ratio	Original Accuracy (%)	Compressed Accuracy (%)
LeNet-300	1070KB ⟶	**27KB**	**40x**	98.36 ⟶	98.42
LeNet-5	1720KB ⟶	**44KB**	**39x**	99.20 ⟶	99.26
AlexNet	240MB ⟶	**6.9MB**	**35x**	80.27 ⟶	80.30
VGGNet	550MB ⟶	**11.3MB**	**49x**	88.68 ⟶	89.09
GoogLeNet	28MB ⟶	**2.8MB**	**10x**	88.90 ⟶	88.92
SqueezeNet	4.8MB ⟶	**0.47MB**	**10x**	80.32 ⟶	80.35



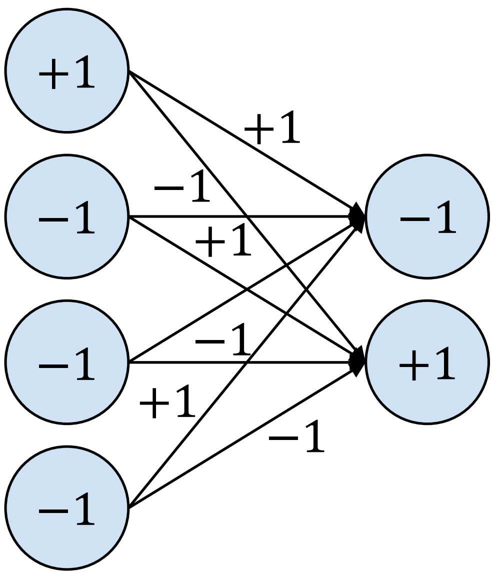
Next: Binarized neural networks

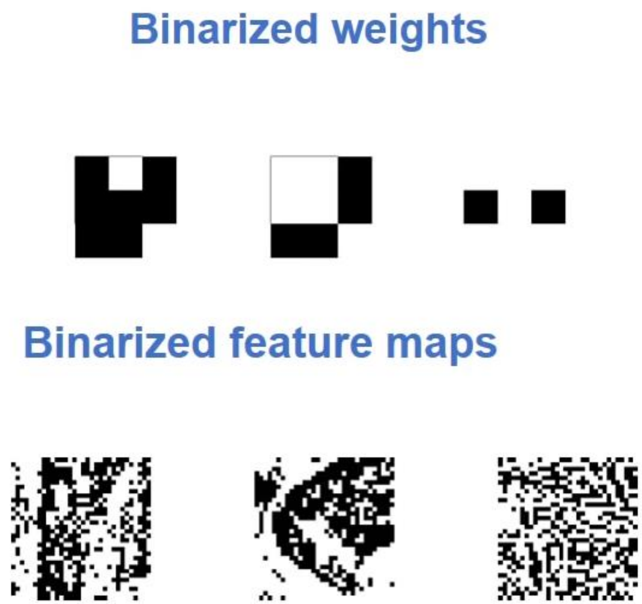
*source : Han et al., "Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding", ICLR 2016

- Neural networks can be even **binarized** (**+1** or **-1**)
  - DNNs trained to use binary weights and binary activations

- Expensive **32-bit MAC** (**M**ultiply-**AC**cumulate) ⇒ Cheap **1-bit XNOR-Count**
  - "MAC == XNOR-Count": when the weights and activations are ± 1

1s in bits





*source : Hubara et al., "Binarized Neural Networks", NIPS 2016

Binarized Neural Networks [Hubara et al., 2016]

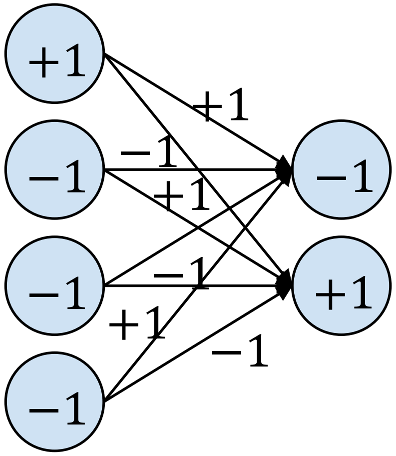
- **Idea**: Training real-valued nets (W_r) treating binarization (W_b) as noise
  - Training W_r is done by **stochastic gradient descent**

- **Binarization** ($W_r \rightarrow W_b$) occurs for each forward propagation
  - On each of **weights**: $W_b = \mathrm{sign}(W_r)$
  - … also on each **activation**: $a_b = \mathrm{sign}(a_r)$

- Gradients for W_r is estimated from $\frac{\partial L}{\partial W_b}$ [Bengio et al., 2013]
  - "Straight-through estimator": Ignore the binarization during backward!

$$\frac{\partial L}{\partial W_r} = \frac{\partial L}{\partial W_b} \underline{\mathbf{1}_{|W_r| \leq 1}}$$

$$\frac{\partial L}{\partial a_r} = \frac{\partial L}{\partial a_b} \underline{\mathbf{1}_{|a_r| \leq 1}}$$



  - Cancelling gradients for better performance
    - When the value is too large

*source : Hubara et al., "Binarized Neural Networks ", NIPS 2016

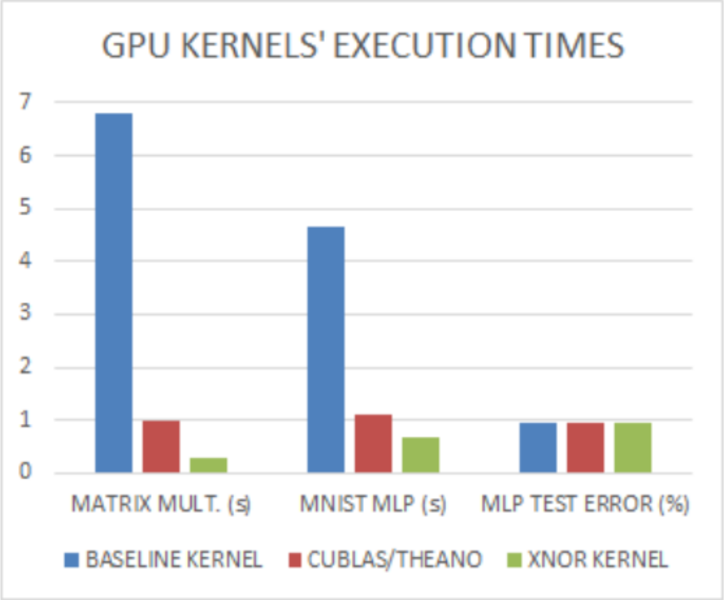
Binarized Neural Networks [Hubara et al., 2016]

- Neural networks can be even **binarized** (**+1** or **-1**)
  - DNNs trained to use binary weights and binary activations

- BNN yields **32x less memory** compared to the baseline 32-bit DNNs
  - … also expected to reduce energy consumption drastically

- **23x faster** on kernel execution times
  - BNN allows us to use XNOR kernels
  - **3.4x** faster than cuBLAS

Operation	MUL	ADD
8bit Integer	0.2pJ	0.03pJ
32bit Integer	3.1pJ	0.1pJ
16bit Floating Point	1.1pJ	0.4pJ
32tbit Floating Point	3.7pJ	0.9pJ



*source : Hubara et al., “Binarized Neural Networks ”, NIPS 2016

Binarized Neural Networks [Hubara et al., 2016]

- Neural networks can be even **binarized** (**+1** or **-1**)
  - DNNs trained to use binary weights and binary activations

- **BNN** achieves comparable error rates over existing DNNs

Data set	MNIST	SVHN	CIFAR-10
Binarized activations+weights, during training and test			
BNN (Torch7)	1.40%	2.53%	10.15%
BNN (Theano)	0.96%	2.80%	11.40%
Committee Machines' Array (Baldassi et al., 2015)	1.35%	-	-
Binarized weights, during training and test			
BinaryConnect (Courbariaux et al., 2015)	1.29± 0.08%	2.30%	9.90%
Binarized activations+weights, during test			
EBP (Cheng et al., 2015)	2.2± 0.1%	-	-
Bitwise DNNs (Kim & Smaragdis, 2016)	1.33%	-	-
Ternary weights, binary activations, during test			
(Hwang & Sung, 2014)	1.45%	-	-
No binarization (standard results)			
Maxout Networks (Goodfellow et al.)	0.94%	2.47%	11.68%
Network in Network (Lin et al.)	-	2.35%	10.41%
Gated pooling (Lee et al., 2015)	-	1.69%	7.62%

*source : Hubara et al., "Binarized Neural Networks ", NIPS 2016

Table of Contents

Summary

- **Broad economic viability requires energy efficient AI** [Welling, 2018]
  - "Energy efficiency of a brain is 100x better than current hardware"
  - "AI algorithms will be measured by the amount of intelligence per kWh"
- **Network pruning and re-wiring**
  - A simple but effective way to compress DNNs
  - Allow us to find better optimum that the current training cannot
- **Sparse network learning**
  - Which training scheme will maximize the pruning performance?
  - It has gained significant attention recently
- **Various other techniques have been also proposed**
  - Weight quantization
  - Anytime/adaptive networks [Huang et al., 2018]
  - …

References

- [LeCun, 1987] Lecun, Y. (1987). PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models).
  Link: https://nyuscholars.nyu.edu/en/publications/phd-thesis-modeles-connexionnistes-de-lapprentissage-connectionis
- [LeCun et al., 1990] LeCun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In Advances in neural information processing systems (pp. 598-605).
  Link: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf
- [Bengio et al., 2013] Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432.
  Link: https://arxiv.org/abs/1308.3432
- [Denil et al., 2013] Denil, M., Shakibi, B., Dinh, L., & De Freitas, N. (2013). Predicting parameters in deep learning. In Advances in neural information processing systems (pp. 2148-2156).
  Link: http://papers.nips.cc/paper/5025-predicting-parameters-in-deep-learning
- [Kingma & Welling, 2013] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
  Link: https://arxiv.org/abs/1312.6114
- [Han et al., 2015] Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. In Advances in neural information processing systems (pp. 1135-1143).
  Link: http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network
- [Kingma et al., 2015] Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational dropout and the local reparameterization trick. In Advances in Neural Information Processing Systems (pp. 2575-2583).
  Link: http://papers.nips.cc/paper/5666-variational-dropout-and-the-local-reparameterization-trick

References

- [Leisman et al., 2015] Leisman, G., Mualem, R., & Mughrabi, S. K. (2015). The neurological development of the child with the educational enrichment in mind. Psicología Educativa, 21(2), 79-96.
  Link: https://www.sciencedirect.com/science/article/pii/S1135755X15000226
- [Tishby & Zaslavsky, 2015] Tishby, N., & Zaslavsky, N. (2015, April). Deep learning and the information bottleneck principle. In Information Theory Workshop (ITW), 2015 IEEE (pp. 1-5). IEEE.
  Link: https://ieeexplore.ieee.org/abstract/document/7133169
- [Han et al., 2016] Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In International Conference on Learning Representations.
  Link: https://arxiv.org/abs/1510.00149
- [Hu et al., 2016] Hu, H., Peng, R., Tai, Y. W., & Tang, C. K. (2016). Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250.
  Link: https://arxiv.org/abs/1607.03250
- [Hubara et al., 2016] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks. In Advances in neural information processing systems (pp. 4107-4115).
  Link: http://papers.nips.cc/paper/6573-binarized-neural-networks
- [Molchanov et al., 2016] Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440.
  Link: https://arxiv.org/abs/1611.06440
- [Wen et al., 2016] Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. In Advances in Neural Information Processing Systems (pp. 2074-2082).
  Link: http://papers.nips.cc/paper/6503-learning-structured-sparsity-in-deep-neural-networks
- [Han et al., 2017] Han, S., Pool, J., Narang, S., Mao, H., Gong, E., Tang, S., ... & Catanzaro, B. (2017). Dsd: Dense-sparse-dense training for deep neural networks. In International Conference on Learning Representations.
  Link: https://openreview.net/forum?id=HyoST_9xl

References

- [Luo et al., 2017] Luo, J. H., Wu, J., & Lin, W. (2017). ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In Proceedings of the IEEE International Conference on Computer Vision (pp. 5058-5066).
  Link: http://openaccess.thecvf.com/content_iccv_2017/html/Luo_ThiNet_A_Filter_ICCV_2017_paper.html
- [Molchanov et al., 2017] Molchanov, D., Ashukha, A. & Vetrov, D.. (2017). Variational Dropout Sparsifies Deep Neural Networks. Proceedings of the 34th International Conference on Machine Learning, in PMLR 70:2498-2507
  Link: http://proceedings.mlr.press/v70/molchanov17a.html
- [Dai et al., 2018] Dai, B., Zhu, C., Guo, B. & Wipf, D.. (2018). Compressing Neural Networks using the Variational Information Bottleneck. Proceedings of the 35th International Conference on Machine Learning, in PMLR 80:1135-1144
  Link: http://proceedings.mlr.press/v80/dai18d.html
- [Huang et al., 2018] Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., & Weinberger, K. Q. (2018). Multi-scale dense networks for resource efficient image classification. In International Conference on Learning Representations
  Link: https://openreview.net/forum?id=Hk2aImxAb
- [Mittal et al., 2018] Mittal, D., Bhardwaj, S., Khapra, M. M., & Ravindran, B. (2018, March). Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks. In 2018 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 848-857). IEEE.
  Link: https://www.computer.org/csdl/proceedings/wacv/2018/4886/00/488601a848-abs.html
- [Welling, 2018] Welling, M. (2018). Intelligence per Kilowatthour.
  Link: https://icml.cc/Conferences/2018/Schedule?showEvent=1866