Learning Sparse DNN Architecture

Presenter: Gyuseung Baek

January 7, 2020

Introduction

- Neural Networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems
- Simple structure may achieve better performance

Learning both Weights and Connections for Efficient Neural Networks

- Simple way to achieve sparse network: prune redundant weights. $(|w| < T \rightarrow w = 0)$
- Simple pruning hinders model performance poorly. Add retraining.
- Pruning algorithm
  1. Train the network as usual.
  2. Prune the unimportant connections.
  3. Retrain the network to fine tune the weights of the remaining connections.

Learning both Weights and Connections for Efficient Neural Networks

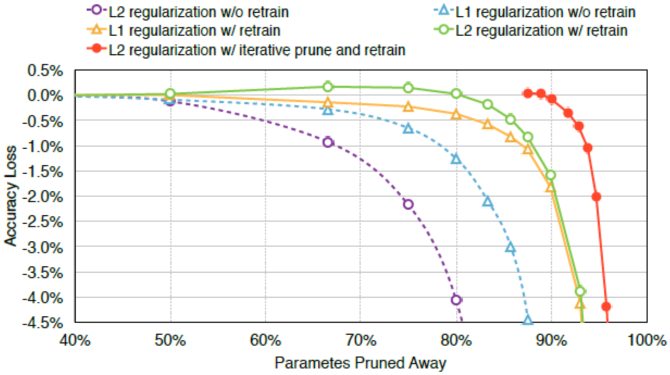
- Regularization: use L2 rather than L1. (performance issue)
- Retraining is iteratively implemented. (As many as possible)
- Pruning threshold: $C\times$ (std. of layer's weights).
- After pruning, learning rate is decayed

Learning both Weights and Connections for Efficient Neural Networks

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×

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1_1	2K	0.2B	53%	58%	58%
conv1_2	37K	3.7B	89%	22%	12%
conv2_1	74K	1.8B	80%	34%	30%
conv2_2	148K	3.7B	81%	36%	29%
conv3_1	295K	1.8B	68%	53%	43%
conv3_2	590K	3.7B	70%	24%	16%
conv3_3	590K	3.7B	64%	42%	29%
conv4_1	1M	1.8B	51%	32%	21%
conv4_2	2M	3.7B	45%	27%	14%
conv4_3	2M	3.7B	34%	34%	15%
conv5_1	2M	925M	32%	35%	12%
conv5_2	2M	925M	29%	29%	9%
conv5_3	2M	925M	19%	36%	11%
fc6	103M	206M	38%	4%	1%
fc7	17M	34M	42%	4%	2%
fc8	4M	8M	100%	23%	9%
total	138M	30.9B	64%	**7.5%**	**21%**

Learning both Weights and Connections for Efficient Neural Networks



Learning Sparse Neural Networks Through L_0 Regularization

- L_0 regularizer is non-differentiable - latent variable and smooth binary dist'n
- L_0 penalty training: find θ^* as

$$\theta^* = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; \theta), y_i) + \lambda \|\theta\|_0$$

- supposse $\theta = \tilde{\theta} \cdot z, z \in \{0, 1\}^{|\theta|}$. by letting $q(z_j|\pi_j) = Ber(\pi_j)$, L_0 penalty training is same as finding $\tilde{\theta}^*, \pi^*$ as

$$\tilde{\theta}^*, \pi^* = \operatorname*{argmin}_{\tilde{\theta}, \pi} \frac{1}{n} \mathbb{E}_{q(z|\pi)} \left[\sum_{i=1}^{n} \ell(f(x_i; \tilde{\theta} \cdot z), y_i) \right] + \lambda \sum_{j=1}^{|\theta|} \pi_j$$

Learning Sparse Neural Networks Through L_0 Regularization

- lather than $Ber(\pi_j)$(discrete), use continuous function

$$s \sim q(\cdot|\phi), z = min(1, max(0, s)) = g(s)$$

- version of smooth the binary Bernoulli gates z:

$$\tilde{\theta}^*, \phi^* = \underset{\tilde{\theta},\pi}{\text{argmin}} \frac{1}{n}\mathbb{E}_{q(s|\phi)}\left[\sum_{i=1}^{n} \ell(f(x_i; \tilde{\theta} \cdot g(s)), y_i)\right] + \lambda \sum_{j=1}^{|\theta|}(1 - Q(s_j \leq 0|\phi_j))$$

  wher Q: cdf of q.

- choice of q: $s = \bar{s}\xi + (1 - \bar{s})\gamma, \bar{s} = Sigmoid((\log \frac{\alpha u}{1-u})/\beta), u \sim \mathcal{U}(0, 1)$
- final solution:

$$\hat{z} = g(Sigmoid(\log \alpha)(\xi - \gamma) + \gamma), \theta^* = \tilde{\theta}^* \cdot \hat{z}$$

Learning Sparse Neural Networks Through L_0 Regularization

Experiments

Network & size	Method	Pruned architecture	Error (%)
MLP 784-300-100	Sparse VD (Molchanov et al., 2017)	512-114-72	1.8
	BC-GNJ (Louizos et al., 2017)	278-98-13	1.8
	BC-GHS (Louizos et al., 2017)	311-86-14	1.8
	$L_{0_{hc}}, \lambda = 0.1/N$	219-214-100	1.4
	$L_{0_{hc}}, \lambda$ sep.	266-88-33	1.8
LeNet-5-Caffe 20-50-800-500	Sparse VD (Molchanov et al., 2017)	14-19-242-131	1.0
	GL (Wen et al., 2016)	3-12-192-500	1.0
	GD (Srinivas & Babu, 2016)	7-13-208-16	1.1
	SBP (Neklyudov et al., 2017)	3-18-284-283	0.9
	BC-GNJ (Louizos et al., 2017)	8-13-88-13	1.0
	BC-GHS (Louizos et al., 2017)	5-10-76-16	1.0
	$L_{0_{hc}}, \lambda = 0.1/N$	20-25-45-462	0.9
	$L_{0_{hc}}, \lambda$ sep.	9-18-65-25	1.0

Learning Sparse Neural Networks via Sensitivity-Driven Regularization

- Sensitivity of weights: the relation between wight variation and output variation.
- Low sensitivity = Not important
- Make low sensitivity weights toward zero by regularized learning. (Hard thresholding)

Learning Sparse Neural Networks via Sensitivity-Driven Regularization

- Model output $\mathbf{y} = (y_1, \cdots, y_C)$
- Sensitivity of weight w

$$S(\mathbf{y}, w) = \sum_{k=1}^{C} \alpha_k \left| \frac{\partial y_k}{\partial w} \right|$$

- bounded insensitivity

$$\bar{S}_b(\mathbf{y}, w) = max[0, 1 - S(\mathbf{y}, w)] \in [0, 1]$$

Learning Sparse Neural Networks via Sensitivity-Driven Regularization

- Update rule

  $$w^t := w^{t-1} - \eta \frac{\partial L}{\partial w} - \lambda w^{t-1} \bar{S}_b(\mathbf{y}, w^{t-1})$$

  by Holder inequaltiy,

  $$\left| \frac{\partial L}{\partial w} \right| \leq \left\| \frac{\partial \mathbf{y}}{\partial w} \right\|_1$$

  If gradient term is large, insensitivity term became small. If gradient term is small, insensitivity term dominates whole update.
- Using insensitivity term is equal to using such regularization term

  $$R(\theta) = \sum_w R(w) = \sum_w \frac{w^2}{2}(1 - S(\mathbf{y}, w))$$

Learning Sparse Neural Networks via Sensitivity-Driven Regularization

Table 1: LeNet300 network trained over the MNIST dataset

	Remaining parameters FC1	FC2	FC3	Total	Memory footprint	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}$	Top-1 error
Han *et al.* [9]	8%	**9%**	**26%**	21.76k	87.04kB	12.2x	1.6%
Proposed (S^{unspec})	**2.25%**	11.93%	69.3%	**9.55k**	**34.2kB**	**27.87x**	1.65%
Proposed (S^{spec})	4.78%	24.75%	73.8%	19.39k	77.56kB	13.73x	**1.56%**
Louizos *et al.* [13]	9.95%	9.68%	33%	26.64k	106.57kB	12.2x	1.8%
SWS[10]	N/A	N/A	N/A	11.19k	44.76kB	23x	1.94%
Sparse VD[16]	1.1%	2.7%	38%	3.71k	14.84kB	68x	1.92%
DNS[24]	1.8%	1.8%	**5.5%**	4.72k	18.88kB	56x	1.99%
Proposed (S^{unspec})	**0.93%**	**1.12%**	5.9%	**2.53k**	**10.12kB**	**103x**	1.95%
Proposed (S^{spec})	1.12%	1.88%	13.4%	3.26k	13.06kB	80x	1.96%

Table 2: LeNet5 network trained over the MNIST dataset

	Remaining parameters Conv1	Conv2	FC1	FC2	Total	Memory footprint	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}$	Top-1 error
Han *et al.* [9]	**66%**	12%	8%	**19%**	36.28k	145.12kB	11.9x	0.77%
Prop. (S^{unspec})	67.6%	**11.8%**	**0.9%**	31.0%	**8.43k**	**33.72kB**	**51.1x**	0.78%
Prop. (S^{spec})	72.6%	12.0%	1.7%	37.4%	10.28k	41.12kB	41.9x	0.8%
Louizos *et al.* [13]	45%	36%	0.4%	5%	6.15k	24.6kB	70x	1.0%
SWS [10]	N/A	N/A	N/A	N/A	2.15k	8.6kB	200x	0.97%
Sparse VD [16]	33%	**2%**	**0.2%**	5%	**1.54k**	**6.16kB**	**280x**	**0.75%**
DNS [24]	**14%**	3%	0.7%	**4%**	3.88k	15.52kB	111x	0.91%

Table 3: VGG16 network trained on the ImageNet dataset

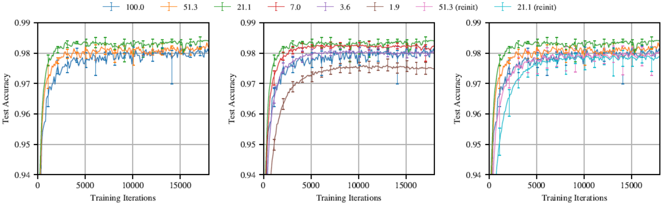
	Remaining parameters Conv	FC	Total	Memory footprint	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}$	Top-1 error	Top-5 error
Han *et al.* [9]	**32.77%**	4.61%	10.35M	41.4 MB	13.33x	31.34%	10.88%
Prop. (S^{unspec})	64.73%	2.9%	11.34M	45.36 MB	12.17x	**29.29%**	**9.80%**
Prop. (S^{spec})	56.49%	**2.56%**	**9.77M**	**39.08 MB**	**14.12x**	30.92%	10.06%

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

- Lottery Ticket Hypothesis: Every network has a subnetwork which can match the test accuracy of the original one after training for at most the same number of iterations (j).
- j is fixed by optimization methods.

- Identifying winning tickets
  1. Randomly initialize a nerual network $f(x; \theta_0)$.
  2. Train the network for j iterations, arriving at parameters θ_j.
  3. Prune p% of the parameters in θ_j, creating a mask m.
  4. Reset the remaining parameters to their values in θ_0, creating the winning ticket.

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks



Picking winning tickets before training by preserving gradient flow

- Various network pruning
  - After training (Han et al., 2015) - long training time
  - During training (Louizos et al., 2018, Tartaglione et al., 2018)
  - Before training (Frankle and Carbin, 2019, SNIP algorithm, GraSP algorithm)

Picking winning tickets before training by preserving gradient flow

- Neural Tangent Kernel (NTK)
  - $\mathcal{X}$: training set, ℓ: loss ftn, $n = |\mathcal{X}|$. then $\mathcal{Z} = f(\mathcal{X}; \theta) \in \mathbb{R}^{nC \times 1}$ and

    $$f(\mathcal{X}; \theta_{t+1}) = f(\mathcal{X}; \theta_t) - \eta \Theta_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \ell$$

    where the matrix $\Theta_t(\mathcal{X}, \mathcal{X})$ is the Neural Tangent Kernel(NTK) at time step t:

    $$\Theta_t(\mathcal{X}, \mathcal{X}) = \nabla_\theta f(\mathcal{X}; \theta_t) \nabla_\theta f(\mathcal{X}; \theta_t)^\top \in \mathbb{R}^{nC \times nC}$$
  - By NTK, training dynamics can be analyzed in closed form (arora et al., 2019):

    $$\|\mathcal{Y} - f(\mathcal{X}; \theta_t\|_2 = \sqrt{\sum_{i=1}^{n} (1 - \eta \lambda_i)^{2t} (u_i^\top \mathcal{Y})^2} \pm \epsilon$$

    where $\mathcal{Y} \in \mathbb{R}^{nC \times 1}$, $\Theta_t = \Theta(\text{const.}) = \sum_{i=1}^{n} \lambda_i u_i u_i^\top$

Picking winning tickets before training by preserving gradient flow

- Algorithms for pruning before training (pruning ratio p, training data $\mathcal{D}$, initial: θ_0
  1. select mini batch $\mathcal{D}_m \sim \mathcal{D}$
  2. choose selection function $S(\theta_0) \in \mathbb{R}^{|\theta_0|}$
  3. compute p_{th} percentile of $S(\theta_0)$ as τ
  4. $m = S(\theta_0) < \tau \in \{0, 1\}^{\|\theta_0|}$
  5. train the network $f_{m \cdot \theta}$ on $\mathcal{D}$ until converges

Picking winning tickets before training by preserving gradient flow

- SNIP and GraSP
  - SNIP(Single-shot network pruning, Lee et al., 2018)

$$S(\theta_0) = \left\{ \lim_{\epsilon \to 0} \left| \frac{\ell(\theta_0) - \ell(\theta_0 + \epsilon \delta_q)}{\epsilon} \right| \right\}_{q=1}^{|\theta_0|} == \left\{ \theta_q \frac{\partial \ell}{\partial \theta_q} \right\}_{q=1}^{|\theta_0|}$$

  - GraSP(Gradient Signal preservation): gradient variation
    Let $\Delta\ell(\theta) = \lim_{\epsilon\to 0} \dfrac{\ell(\theta_0 + \epsilon \nabla \ell(\theta)) - \ell(\theta)}{\epsilon} = \nabla \ell(\theta)^\top \nabla \ell(\theta)$ (directional derivative)

$$\tilde{S}(\delta) \simeq \Delta\ell(\theta_0 + \delta) - \Delta\ell(\theta_0) = 2\delta^\top \nabla^2 \ell(\theta_0) \nabla \ell(\theta_0) = 2\delta^\top Hg$$

$$S(\theta_0) = \tilde{S}(-\theta_0)$$

Picking winning tickets before training by preserving gradient flow

Experiments

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG19 (Baseline)	94.23	-	-	74.16	-	-
SNIP (Lee et al., 2018)	**93.63±0.06**	**93.43±0.20**	92.05±0.28	**72.84±0.22**	**71.83±0.23**	58.46±1.10
GraSP	93.30±0.14	93.04±0.18	**92.19±0.12**	71.95±0.18	71.23±0.12	**68.90±0.47**
ResNet32 (Baseline)	94.80	-	-	74.64	-	-
SNIP (Lee et al., 2018)	**92.59±0.10**	91.01±0.21	87.51±0.31	68.89±0.45	65.22±0.69	54.81±1.43
GraSP	92.38±0.21	**91.39±0.25**	**88.81±0.14**	**69.24±0.24**	**66.50±0.11**	**58.43±0.43**