Channel Pruning for Accelerating Very Deep Neural Networks

CONTANTS

Background | Related Works | Approach | Experiment | Conclusion

1. Most of the existing deep models are structurally very complex, making them difficult to be deployed on the embedded system.

2. More parameters means more storage requirement and more floating-point operations(FLOPs).

3. The battery capacity can be another bottlenneck. For example,at 20Hz would requile (20Hz) (1G) (640pJ) = 12.8W just for DRAM access - well beyond the power envelope of a typical mobile device.

Methods — Pointwise

Purposeful

Structured Pruning

Model Compression

Unstructured Pruning

Randomness

Structured Pruning

1. Based on the order of magnitude, the sum of all the absolute values of weight in filter is taken as the evaluation criterion of the filter, and the lower filter of one layer is cut off.

2. Based on entropy, it is worth pruning, calculate the entropy of each feature map, then judge the importance of filter, and cut off the unimportant filter.

Unstructured Pruning

1. Learning both Weights and Connections for Effient Neural Networks

2. Deep Compression: Compression Deep Neural Networks With Pruning,Trained Quantization And Huffman Coding

3. Dynamic Network Surgery for Effient DNNs

4. Channel Pruning for Accelerating Very Deep Neural Networks

Channel Pruning for Accelerating Very Deep Neural Networks

Paper thinking：基于CNN网络模型的稀疏性(能够达到某一个upper bound), 找到neuron之间的关系，留下代表性地neuron。

问题解决：

step 1：找出每一层具有代表性地neuron，我们利用lasso regression来进行类似model selection的过程。将剩余的neuron去掉（pruning）。

step 2: 利用剩下的代表性neuron来重构（reconstruction）这一层原本的输出。

Channel Pruning for Accelerating Very Deep Neural Networks

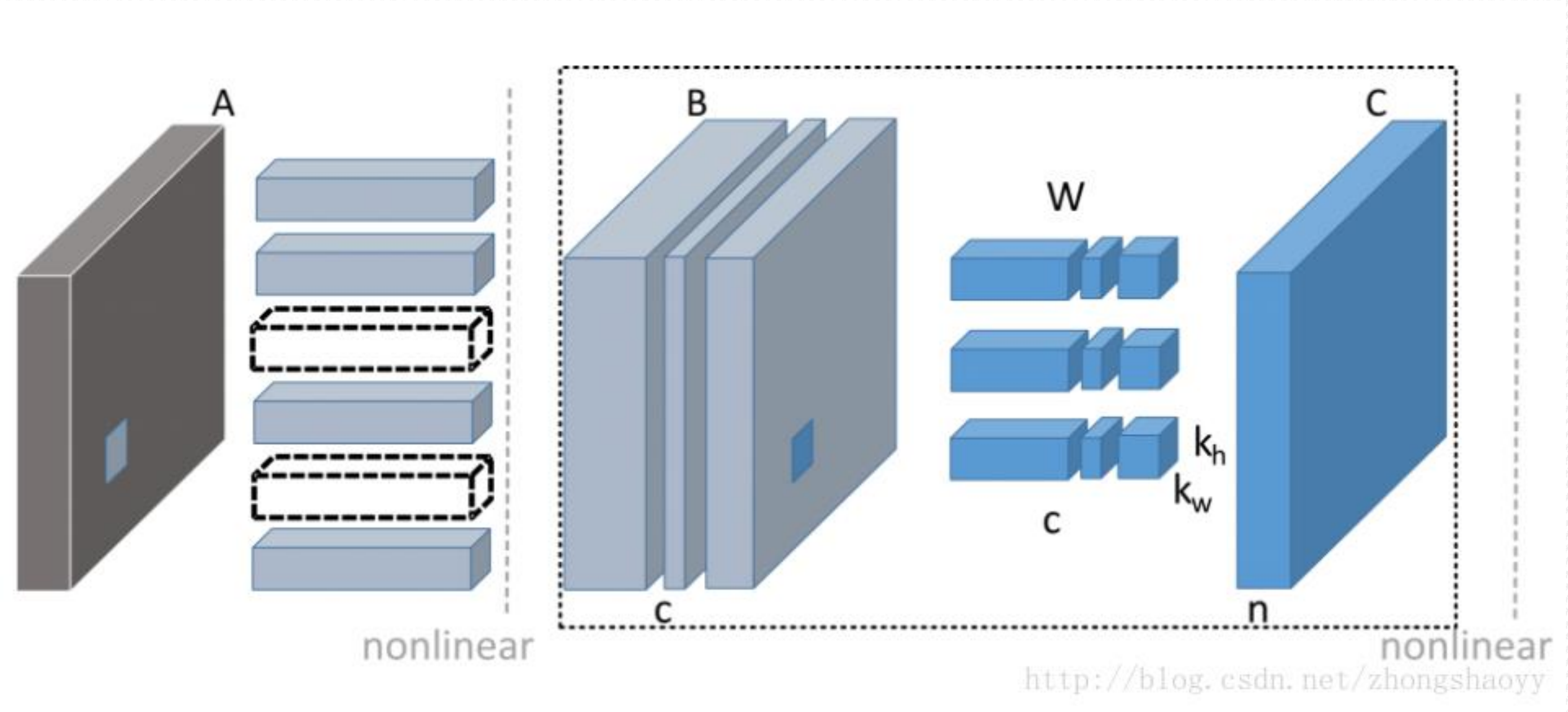
Paper thinking：基于CNN网络模型的稀疏性(能够达到某一个upper bound), 找到neuron之间的关系，留下代表性地neuron。

问题解决：

step 1：找出每一层具有代表性地neuron，我们利用lasso regression来进行类似model selection的过程。将剩余的neuron去掉（pruning）。

step 2: 利用剩下的代表性neuron来重构（reconstruction）这一层原本的输出。

Channel Pruning for Accelerating Very Deep Neural Networks



Channel Pruning for Accelerating Very Deep Neural Networks

$$\arg\min_{\beta, W} \frac{1}{2N} \left\| Y - \sum_{i=1}^{c} \beta_i X_i W_i^\top \right\|_F^2 \tag{1}$$

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

$$\arg\min_{\beta, W} \frac{1}{2N} \left\| Y - \sum_{i=1}^{c} \beta_i X_i W_i^\top \right\|_F^2 + \lambda \|\beta\|_1 \tag{2}$$

subject to $\|\beta\|_0 \le c', \forall i \|W_i\|_F = 1$

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

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

(3)

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

Channel Pruning for Accelerating Very Deep Neural Networks

主要贡献在于：

相较于原来那种暴力的pruning，利用数学方法优化目标函数，使得pruning前后的输出差异最小，取得了一定效果。

缺点在于：

觉得人工加入的限定太多，而且这种方法引入了很多调节参数，调整和优化都麻烦，实用性不强。而且文章开头说不需要retrain，其实还是pruning之后再来finetune一下效果比较好。

Channel Pruning for Accelerating Very Deep Neural Networks

启发思路:

1. 采channel pruning的方法对算法以及模型进行压缩和加速，能够应用到我们的嵌入式端(TX2)

2. 该方法本身需要设置较多的超参数，可以考虑找到相应的方法设置较少的参数，简化算法的优化过程。

Learning both Weights and Connections for Effient Neural Networks

Learning important connection ⟶ Pruning unimportant ⟶ Fine tune

Methods:

1. Regularization   L2 regularization gives the best pruning results
2. Dropout Ratio Adjustment
3. Local Pruning and Parameter Co-adaptation
4. Iterative Pruning
5. Pruning Neurals

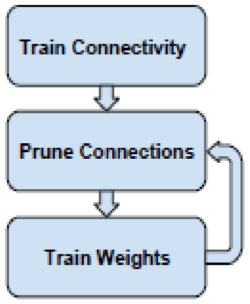


Figure 2: Three-Step Training Pipeline.

Table 1: Network pruning can save 9× to 13× parameters with no drop in predictive performance.

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×**

Table 2: For Lenet-300-100, pruning reduces the number of weights by 12× and computation by 12×.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
fc1	235K	470K	38%	8%	8%
fc2	30K	60K	65%	9%	4%
fc3	1K	2K	100%	26%	17%
Total	266K	532K	46%	**8%**	**8%**

Table 3: For Lenet-5, pruning reduces the number of weights by 12× and computation by 6×.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	0.5K	576K	82%	66%	66%
conv2	25K	3200K	72%	12%	10%
fc1	400K	800K	55%	8%	6%
fc2	5K	10K	100%	19%	10%
Total	431K	4586K	77%	**8%**	**16%**

where T denotes the test matrix and Pi denotes the index values of the top-N predicted items in the original matrix for the i-th user.
The overall Precision@N value is computed as the average of Precision@N(i) over all users.
MAP@N can be computed as the mean of the average precision of the top-N predictions for all users

Table 4: For AlexNet, pruning reduces the number of weights by $9\times$ and computation by $3\times$.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
Total	61M	1.5B	54%	11%	30%

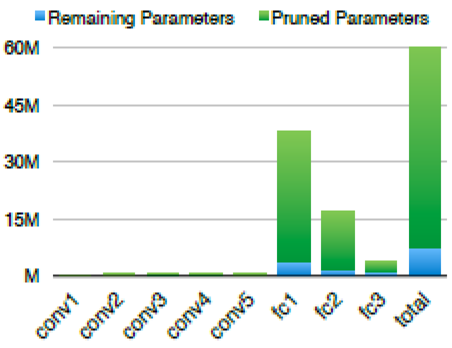


Table 5: For VGG-16, pruning reduces the number of weights by $12\times$ and computation by $5\times$.

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%

where T denotes the test matrix and Pi denotes the index values of the top-N predicted items in the original matrix for the i-th user.
The overall Precision@N value is computed as the average of Precision@N(i) over all users.
MAP@N can be computed as the mean of the average precision of the top-N predictions for all users

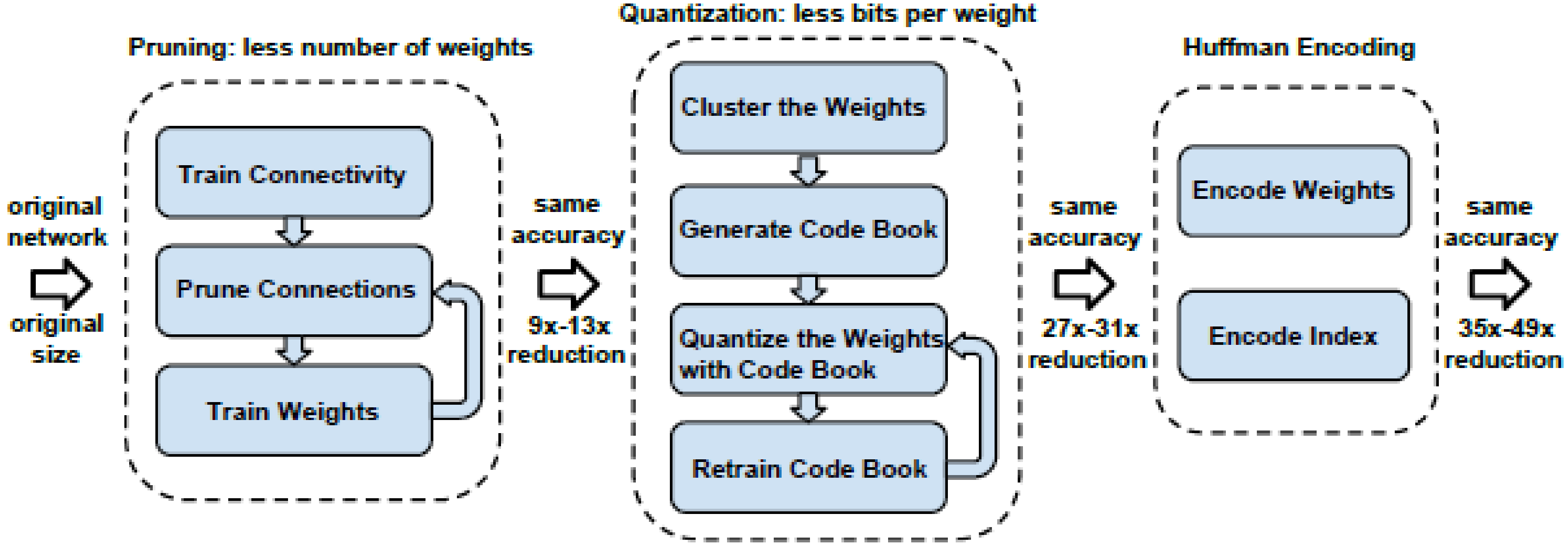
Deep Compression: Compression Deep Neural Networks With Pruning,Trained Quantization And Huffman Coding

Pruning ⟶ Trained Quantization ⟶ Huffman Coding

Methods:

1. Pruning(Store in a more compact way, like CSRorCSC)

2. Weight shared and Quantization
   2.1 采用kmeans算法对权值进行聚类，所有权值共享该类的聚类中心。
   2.2 聚类中心初始化
   2.3 前向反馈和后项传播(前向时将每个权值用对应的聚类中心代替，后向计算每个类的权值梯度)
3. Huffman Coding
   (解决编码长度不一带来的冗余问题，作者对卷积层统一使用8bit编码，全连接层采用5bit)



where T denotes the test matrix and Pi denotes the index values of the top-N predicted items in the original matrix for the i-th user.
The overall Precision@N value is computed as the average of Precision@N(i) over all users.
MAP@N can be computed as the mean of the average precision of the top-N predictions for all users

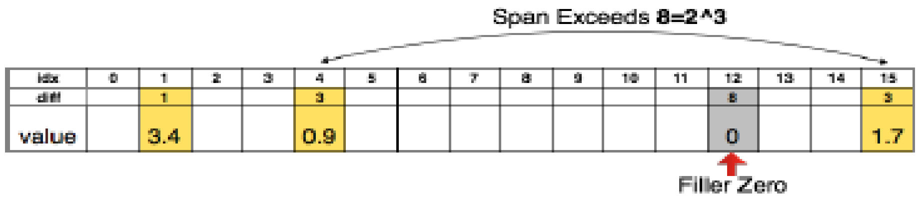


Figure 2: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow.

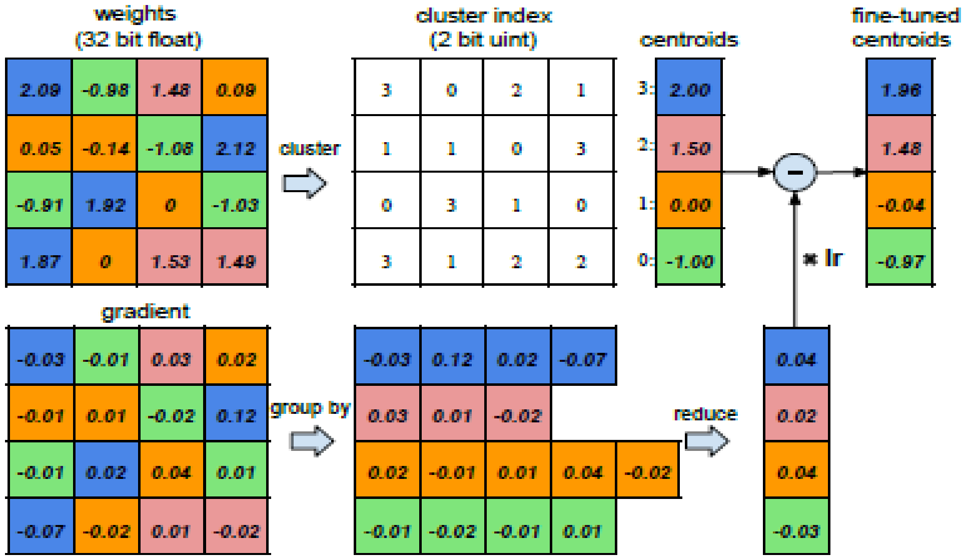


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

where T denotes the test matrix and Pi denotes the index values of the top-N predicted items in the original matrix for the i-th user.
The overall Precision@N value is computed as the average of Precision@N(i) over all users.
MAP@N can be computed as the mean of the average precision of the top-N predictions for all users

Table 1: The compression pipeline can save 35× to 49× parameter storage with no loss of accuracy.

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40**×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39**×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35**×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	**49**×

where T denotes the test matrix and Pi denotes the index values of the top-N predicted items in the original matrix for the i-th user.
The overall Precision@N value is computed as the average of Precision@N(i) over all users.
MAP@N can be computed as the mean of the average precision of the top-N predictions for all users

Dynamic Network Surgery for Effient DNNs

Methods:

1. Pruning 2. Splicing

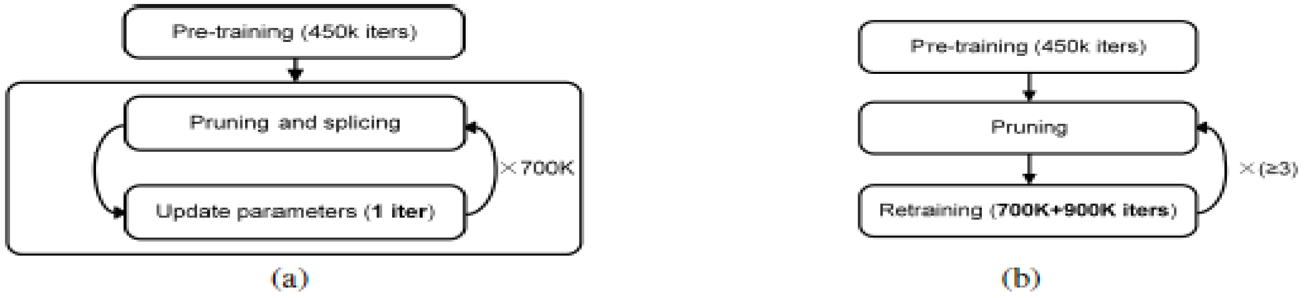


Figure 1: The pipeline of (a) our dynamic network surgery and (b) Han et al.'s method [9], using AlexNet as an example. [9] needs more than 4800K iterations to get a fair compression rate (9×), while our method runs only 700K iterations to yield a significantly better result (17.7×) with comparable prediction accuracy.

Dynamic Network Surgery for Efficient DNNs

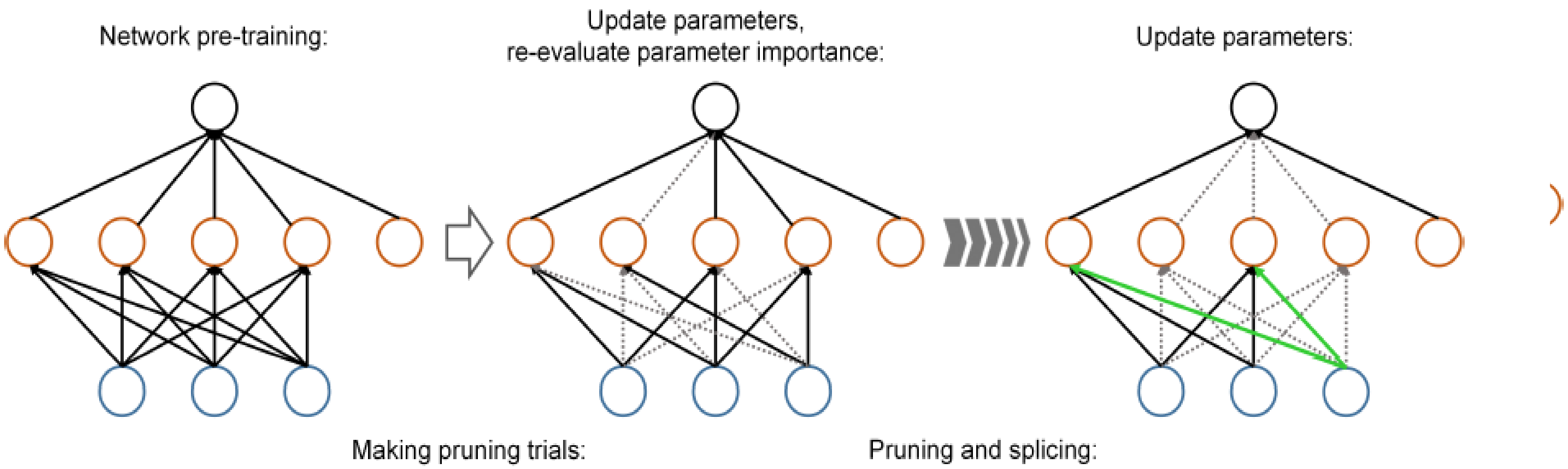


Figure 2: Overview of the dynamic network surgery for a model with parameter redundancy.

Dynamic Network Surgery for Efficient DNNs

$$\min_{\mathbf{W}_k, \mathbf{T}_k} L\left(\mathbf{W}_k \odot \mathbf{T}_k\right) \quad \text{s.t. } \mathbf{T}_k^{(i,j)} = \mathbf{h}_k(\mathbf{W}_k^{(i,j)}), \ \forall (i,j) \in \mathcal{I},$$

$$\mathbf{W}_k^{(i,j)} \leftarrow \mathbf{W}_k^{(i,j)} - \beta \frac{\partial}{\partial(\mathbf{W}_k^{(i,j)} \mathbf{T}_k^{(i,j)})} L\left(\mathbf{W}_k \odot \mathbf{T}_k\right), \ \forall (i,j) \in \mathcal{I},$$

$$\mathbf{h}_k(\mathbf{W}_k^{(i,j)}) = \begin{cases} 0 & \text{if } a_k > |\mathbf{W}_k^{(i,j)}| \\ \mathbf{T}_k^{(i,j)} & \text{if } a_k \le |\mathbf{W}_k^{(i,j)}| < b_k \\ 1 & \text{if } b_k \le |\mathbf{W}_k^{(i,j)}| \end{cases}$$

Table 1: Dynamic network surgery can remarkably reduce the model complexity of some popular networks, while the prediction error rate does not increase.

model	Top-1 error	Parameters	Iterations	Compression
LeNet-5 reference	0.91%	431K	10K	
LeNet-5 pruned	0.91%	4.0K	16K	**108**×
LeNet-300-100 reference	2.28%	267K	10K	
LeNet-300-100 pruned	1.99%	4.8K	25K	**56**×
AlexNet reference	43.42%	61M	450K	
AlexNet pruned	43.09%	3.45M	700K	**17.7**×

where T denotes the test matrix and Pi denotes the index values of the top-N predicted items in the original matrix for the i-th user.
The overall Precision@N value is computed as the average of Precision@N(i) over all users.
MAP@N can be computed as the mean of the average precision of the top-N predictions for all users

In this paper, we have investigated the way of compressing DNNs and proposed a novel method called dynamic network surgery. Unlike the previous methods which conduct pruning and retraining alternately, our method incorporates connection splicing into the surgery and implements the whole process in a dynamic way. By utilizing our method, most parameters in the DNN models can be deleted, while the prediction accuracy does not decrease. The experimental results show that our method compresses the number of parameters in LeNet-5 and AlexNet by a factor of 108x and 17.7x, respectively, which is superior to the recent pruning method by considerable margins. Besides, the learning efficiency of our method is also better thus less epochs are needed.

Thanks

https://github.com/Jluxcs/papers

[Cremonesi et al., 2010] P. Cremonesi, Y. Koren, and R. Turrin.
Performance of recommender algorithms on top-N recommendation tasks. In Proc. of the ACM Conference on Recommender Systems (RecSys), 2010
https://github.com/Jluxcs/papers

[Ning and Karypis, 2011] X. Ning and G. Karypis.
SLIM: sparse linear methods for top-N recommender systems. In Proc. of the IEEE International Conference on Data Mining (ICDM), 2011.
https://github.com/Jluxcs/papers

[Park et al., 2015] D. Park, J. Neeman, J. Xhang, and S. Sanghavi.
Preference completion: Large-scale collaborative ranking from pairwise comparisons. In Proc. of the International Conf. on Machine Learning (ICML), 2015.
https://github.com/Jluxcs/papers