Dynamic network surgery for efficient DNNs

Yiwen Guo*
Intel Labs China
yiwen.guo@intel.com

Anbang Yao
Intel Labs China
anbang.yao@intel.com

Yurong Chen
Intel Labs China
yurong.chen@intel.com

Freya

2020|11|29

- 论文地址：https://arxiv.org/pdf/1608.04493.pdf
- 实现代码：https://github.com/yiwenguo/Dynamic-Network-Surgery
- 第三方：https://github.com/HolmesShuan/Dynamic-Network-Surgery-Caffe-Reimplementation

Dynamic Network Surgery: Pipeline

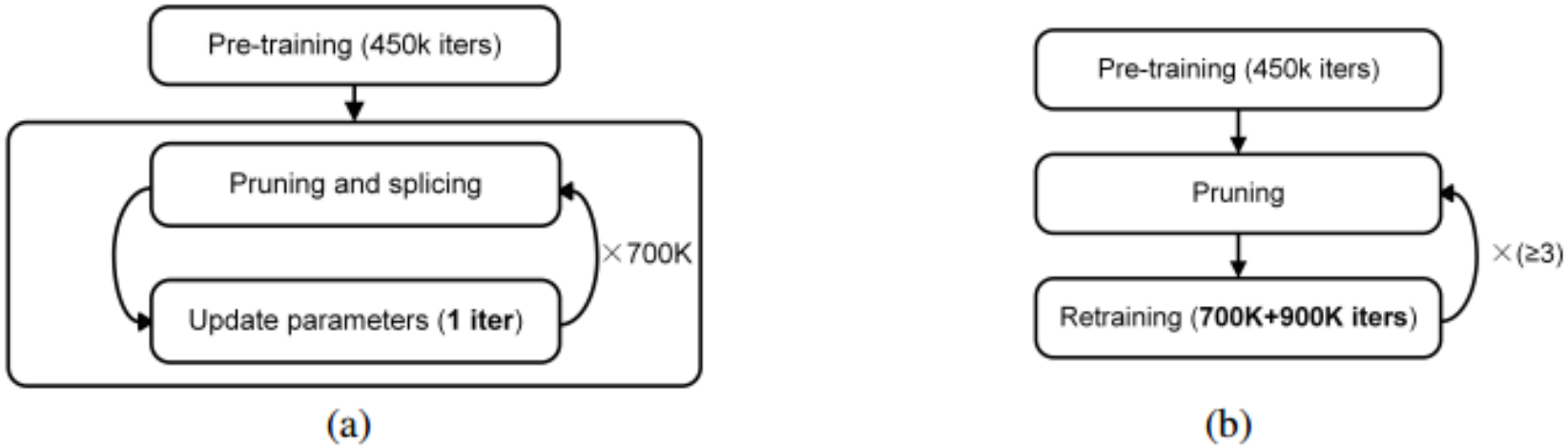

Figure 1: The pipeline of (a) our dynamic network surgery and (b) Han et al.'s method [9], using AlexNet as an example. [9] needs more than 4800K iterations to get a fair compression rate (9×), while our method runs only 700K iterations to yield a significantly better result (17.7×) with comparable prediction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

Dynamic Network Surgery: Model

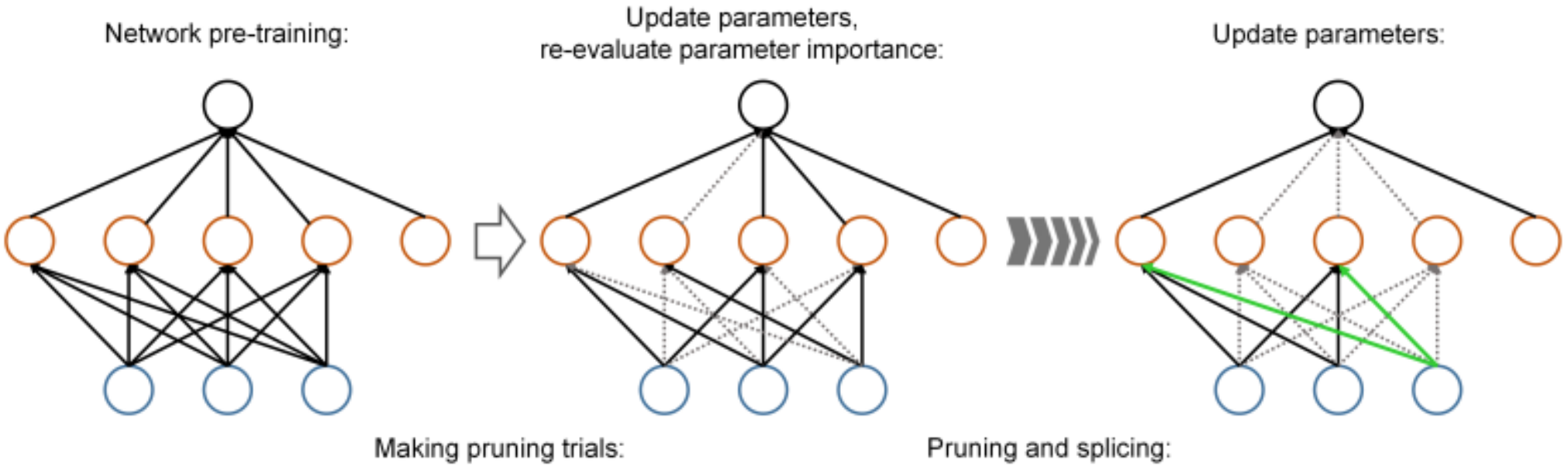

Figure 2: Overview of the dynamic network surgery for a model with parameter redundancy.

Pruning & Scaling

- Mask

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

- Update

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

Algorithm

Algorithm 1 Dynamic network surgery: the SGD method for solving optimization problem (1):

Input: $\mathbf{X}$: training datum (with or without label), $\{\widehat{\mathbf{W}}_k : 0 \leq k \leq C\}$: the reference model, α: base learning rate, f: learning policy.
Output: $\{\mathbf{W}_k, \mathbf{T}_k : 0 \leq k \leq C\}$: the updated parameter matrices and their binary masks.
Initialize $\mathbf{W}_k \leftarrow \widehat{\mathbf{W}}_k$, $\mathbf{T}_k \leftarrow \mathbf{1}, \forall 0 \leq k \leq C$, $\beta \leftarrow 1$ and iter $\leftarrow 0$
repeat
  Choose a minibatch of network input from $\mathbf{X}$
  Forward propagation and loss calculation with $(\mathbf{W}_0 \odot \mathbf{T}_0)$,...,$(\mathbf{W}_C \odot \mathbf{T}_C)$
  Backward propagation of the model output and generate ∇L
  for $k = 0, ..., C$ **do**
    Update $\mathbf{T}_k$ by function $\mathbf{h}_k(\cdot)$ and the current $\mathbf{W}_k$, with a probability of $\sigma(\text{iter})$
    Update $\mathbf{W}_k$ by formula (2) and the current loss function gradient ∇L
  end for
  Update: iter $\leftarrow$ iter $+ 1$ and $\beta \leftarrow f(\alpha, \text{iter})$
until iter reaches its desired maximum

Parameter Importance

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

$$threshold1 = 0.9 * (mean + cRate * std)$$
$$threshold2 = 1.1 * (mean + cRate * std)$$

```
        // Calculate the weight mask and bias mask with probability
        Dtype r = static_cast<Dtype>(rand())/static_cast<Dtype>(RAND_MAX);
        if (pow(1+(this->gamma)*(this->iter_),-(this->power))>r && (this->iter_)<(this->iter_stop_))
            for (unsigned int k = 0;k < this->blobs_[0]->count(); ++k) {
                if (weightMask[k]==1 && fabs(weight[k])<=0.9*std::max(mu+crate*std,Dtype(0)))
                    weightMask[k] = 0;
                else if (weightMask[k]==0 && fabs(weight[k])>1.1*std::max(mu+crate*std,Dtype(0)))
                    weightMask[k] = 1;
            }
            if (this->bias_term_) {
                for (unsigned int k = 0;k < this->blobs_[1]->count(); ++k) {
                    if (biasMask[k]==1 && fabs(bias[k])<=0.9*std::max(mu+crate*std,Dtype(0)))
                        biasMask[k] = 0;
                    else if (biasMask[k]==0 && fabs(bias[k])>1.1*std::max(mu+crate*std,Dtype(0)))
                        biasMask[k] = 1;
                }
            }
        }
```

Convergence Acceleration

- Slowing down the pruning and splicing frequencies
- Pruning the convolutional layers and fully connected layers separately

Experiments

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**×

Experiments

Table 4: Compare our method with [9] on AlexNet.

Layer	Params.	Params.% [9]	Params.% (Ours)
conv1	35K	~ 84%	53.8%
conv2	307K	~ 38%	40.6%
conv3	885K	~ 35%	29.0%
conv4	664K	~ 37%	32.3%
conv5	443K	~ 37%	32.5%
fc1	38M	~ 9%	3.7%
fc2	17M	~ 9%	6.6%
fc3	4M	~ 25%	4.6%
Total	61M	~ 11%	**5.7%**

Conclusion

- ~~conduct pruning and retraining alternately~~
- incorporate connection splicing into the surgery and implement the whole process in a dynamic way

- most parameters in the DNN models can be deleted, while the prediction accuracy does not decrease