Towards Fast and Efficient Representation Learning

Hao Li

PhD Dissertation Defense

June 14, 2018
“Learning representations of the data that make it easier to extract useful information when building classifiers or other predictors.” Bengio et al, PAMI’13
Evolution of Convolutional Neural Networks

<table>
<thead>
<tr>
<th>Year</th>
<th>Network</th>
<th>Top-5 Acc.</th>
<th>Layers</th>
<th>Params</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>AlexNet</td>
<td>18.2%</td>
<td>8</td>
<td>60 M</td>
<td>0.86B</td>
</tr>
<tr>
<td>2014</td>
<td>VGG-19</td>
<td>7.3%</td>
<td>19</td>
<td>140 M</td>
<td>19 B</td>
</tr>
<tr>
<td>2014</td>
<td>GoogLeNet</td>
<td>6.7%</td>
<td>22</td>
<td>5 M</td>
<td>1.5B</td>
</tr>
<tr>
<td>2016</td>
<td>ResNet</td>
<td>5.7%</td>
<td>152</td>
<td>60 M</td>
<td>11 B</td>
</tr>
<tr>
<td>2016</td>
<td>DenseNet</td>
<td>5.3%</td>
<td>264</td>
<td>32 M</td>
<td>11 B</td>
</tr>
<tr>
<td>2017</td>
<td>NASNet</td>
<td>3.8%</td>
<td>28</td>
<td>88 M</td>
<td>23 B</td>
</tr>
</tbody>
</table>

2012 | 2014 | 2016 | 2017 | 2018
<table>
<thead>
<tr>
<th>Enabling Factors and Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Large-Scale Dataset</strong></td>
</tr>
<tr>
<td>applications with limited data</td>
</tr>
<tr>
<td>Data Efficiency</td>
</tr>
</tbody>
</table>
Towards Fast and Efficient Representation Learning

Reducing the Inference Cost

• How to reduce the model size?
• How to reduce the number of operations for fast inference?

Reducing the Training Cost

• How to accelerate training with HPC?
• How to train low-precision models on resource-constrained devices?

Understanding the “black box”

• Understand the generalization ability.
• Understand the choice of architectures.
• Understand the optimization process.
Reducing the Inference Cost
Pruning Filters for Efficient ConvNets, ICLR’17

Reducing the Training Cost
Training Quantized Network: A Deeper Understanding, NIPS’17

Understanding the “black box”
Visualizing the Loss Landscape of Neural Nets, arXiv’17
Pruning Filters for Efficient ConvNets

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, Hans Peter Graf

International Conference on Learning Representations (ICLR), 2017
Model Compression by Pruning Weights

Pruning weights with small magnitude

Impressive results for model compression

Han, et al, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”, ICLR 2016
Compression ≠ Acceleration

1. **90%** of the total parameters of VGG-Net comes from the FC layers, which takes less than **1%** total FLOPS.

2. **ResNets** have replaced the FC layers with average pooling layers.

3. Pruning weights generates **sparse convolutional** kernels, which requires **special library** or **hardware**.
Convolutional Operations
Acceleration by Pruning Filters

Prune a filter, its corresponding feature map and the connected kernels.
Acceleration by Pruning Filters

Copy the remaining weights to a new model without using mask.

No mask/sparse convolution, no special library/hardware
Determining a Filter’s Importance

Filters ranked by L1-norm

CIFAR-10, VGG-16

CIFAR10, VGG-16, prune filters with smallest $l_1$-norm

accuracy vs. filters pruned away (%)
Determining a Filter’s Importance

Filters ranked by L1-norm
Determining a Filter’s Importance

pruning the smallest filters works better than pruning random or the largest filters.
Determining a Layer’s Sensitivity to Pruning

Prune the smallest filters

Retrain to recover accuracy
Pruning Filters across Multiple Layers

Independent pruning

$$\| \mathcal{F}_{i,j} \|_1 = \sum \left| \begin{array}{c} \text{filter entries} \end{array} \right|$$

Greedy pruning

$$\| \mathcal{F}_{i,j} \|_1 = \sum \left| \begin{array}{c} \text{important filter entries} \end{array} \right|$$
Pruning Filters across Multiple Layers
layers that are close to pooling layers are sensitive to pruning
## Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Error</th>
<th>Pruned FLOPs</th>
<th>Pruned Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>6.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG-16-pruned-A</td>
<td>6.60</td>
<td>34.2%</td>
<td>64%</td>
</tr>
<tr>
<td>VGG-16-pruned-A scratch train</td>
<td>6.88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-56</td>
<td>6.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-56-pruned-A</td>
<td>6.90</td>
<td>10.4%</td>
<td>9.4%</td>
</tr>
<tr>
<td>ResNet-56-pruned-B</td>
<td>6.94</td>
<td>27.6%</td>
<td>13.7%</td>
</tr>
<tr>
<td>ResNet-56-pruned-B scratch train</td>
<td>8.69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-110</td>
<td>6.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-110-pruned-A</td>
<td>6.45</td>
<td>15.9%</td>
<td>2.3%</td>
</tr>
<tr>
<td>ResNet-110-pruned-B</td>
<td>6.70</td>
<td>38.6%</td>
<td>32.4%</td>
</tr>
<tr>
<td>ResNet-110-pruned-B scratch train</td>
<td>7.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-34</td>
<td>26.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-34-pruned-A</td>
<td>27.44</td>
<td>15.5%</td>
<td>7.6%</td>
</tr>
<tr>
<td>ResNet-34-pruned-B</td>
<td>27.83</td>
<td>24.2%</td>
<td>10.8%</td>
</tr>
</tbody>
</table>

- **24%–38%** reduction in FLOPs for VGG-16 and ResNets without losing accuracy on CIFAR-10.

- **24%** reduction in FLOPs for ResNet-34 with ~1% loss in accuracy on ImageNet.

- Pruning a **wider** model with retraining performs better than training the pruned **thin** network from **scratch**.
Reducing the Inference Cost
Pruning Filters for Efficient ConvNets, ICLR’17

Reducing the Training Cost
Training Quantized Network: A Deeper Understanding, NIPS’17

Understanding the “black box”
Visualizing the Loss Landscape of Neural Nets, arXiv’17
Training Quantized Networks: a Deeper Understanding

Hao Li*, Soham De*, Zheng Xu, Christoph Studer, Hanan Samet, Tom Goldstein

Neural Information Processing Systems (NIPS), 2017
Low-Precision Neural Networks

32-bit W, 32-bit A

1-bit W, 32-bit A

1-bit W, 1-bit A

Full-precision model

32× smaller, ~2× faster with addition and subtraction ops

~58× faster with XNOR and bit counting ops

Training DNNs: from HPC to Embedded Devices

Training on HPC

Quantization

Inference on Devices

How to train quantized models on resource-constrained devices?

TPU 1.0

TPU 2.0
How to Train Quantized Nets?

- **Non-quantized**: SGD
  \[ w^{t+1} = w^t - \alpha_t \nabla \tilde{f}(w^t) \]

- **Semi-quantized**: BinaryConnect [Courbariaux et al NIPS’15]
  - fast forward propagation
    \[ w_b^t = Q(w_r^t) \]
  - accumulating gradients
    \[ w_r^{t+1} = w_r^t - \alpha_t \nabla \tilde{f}(w_b^t) \]
  requires full-precision weights

- **Fully-quantized**: Deterministic / Stochastic Rounding [Gupta et al, ICML’15]
  \[ w_b^{t+1} = w_b^t - Q(\alpha_t \nabla \tilde{f}(w_b^t)) \]
  no full-precision weights

\[ Q_d(0.3) \]

- 100%
- 30%
- 100%

\[ Q_s(0.3) \]

- 70%
- 30%
- 100%
Empirical Result

Train CNNs with binary weights on CIFAR-10

Keeping the real-valued weights seems to really help empirically.

Why does keeping floating point weights help?
Toy Example for Non-Convex Problems

learning rate = 1

Quantized scalar weight $\Delta = 0.5$
Weight Distribution after 1M Steps

\[ \text{lr} = 1 \quad \text{lr} = 0.1 \quad \text{lr} = 0.01 \quad \text{lr} = 0.001 \]
Exploration-Exploitation Tradeoff

SGD/BinaryConnect

Exploration

Exploitation

Exploration shrink learning rate

Exploration shrink learning rate

Stochastic Rounding

Exploitation

Exploitation
Markov Chain Interpretation

The probability of moving to the next state depends only on the present state and not on the previous states.

Markov process

SR starts at some state $x$, and moves to a new state $y$ with some transition probability $T(x, y)$ that depends only on $x$ and the learning rate $\alpha$. For fixed $\alpha$, this is a Markov process with transition matrix $T_\alpha(x, y)$.  

$$w_{b+1}^t = w_b^t - Q(\alpha_t \nabla \tilde{f}(w_b^t))$$
Markov Chain Interpretation

\[ w_{b}^{t+1} = w_{b}^{t} - Q(\alpha_{t} \nabla \tilde{f}(w_{b}^{t})) \]

\[ p_{\alpha}(\tilde{z}) = \alpha^{-1} p(z/\alpha) \]

Shrink learning rate

The conditional probability does not depend on the learning rate!
Conditional Probability \( T_\alpha(0, 1| x^1 \neq x^0) \)

**Plot Descriptions**

- **lr = 1**
- **lr = 0.1**
- **lr = 0.01**
- **lr = 0.001**

**Diagram Details**

- The plots are organized into two categories: **BC** and **SR**.
- Each category contains four subplots, each corresponding to a different learning rate.
- The x-axis represents different values, and the y-axis shows the probability distribution.
- The learning rates are compared across the two categories.
Transition Probability $T_\alpha(x, y)$

<table>
<thead>
<tr>
<th>lr</th>
<th>BC</th>
<th>lr</th>
<th>BC</th>
<th>lr</th>
<th>BC</th>
<th>lr</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1.png" alt="Image" /></td>
<td>0.1</td>
<td><img src="image2.png" alt="Image" /></td>
<td>0.01</td>
<td><img src="image3.png" alt="Image" /></td>
<td>0.001</td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image5.png" alt="Image" /></td>
<td></td>
<td><img src="image6.png" alt="Image" /></td>
<td></td>
<td><img src="image7.png" alt="Image" /></td>
<td></td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
</tbody>
</table>

SR
Long Term Behavior

\[ \text{lr} = 1 \quad \text{lr} = 0.1 \quad \text{lr} = 0.01 \quad \text{lr} = 0.001 \]

The stationary distribution of BinaryConnect concentrates on stationary points.

These algorithms have an exploitation phase!

The stationary distribution of Stochastic Rounding does not concentrate on stationary points.

Exploration slows down, but exploitation never happens!
Experiment Results

- The BC method changes less than 20% of the weights, which indicates that BC is able to change from explorative to exploitative – it drives more towards local minimizers and explores less aggressively.
- The SR method is not able to exit the exploration phase; it keeps changing weights until 50% of the weights differ from their starting values.
Experiment Results

Table 1: Top-1 test error after training with full-precision (ADAM), binarized weights (R-ADAM, SR-ADAM, BC-ADAM), and binarized weights with big batch size (Big SR-ADAM).

<table>
<thead>
<tr>
<th>Model</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAM</td>
<td>7.97</td>
<td>7.12</td>
<td>8.10</td>
</tr>
<tr>
<td>BC-ADAM</td>
<td>10.36</td>
<td>8.21</td>
<td>8.83</td>
</tr>
<tr>
<td>Big SR-ADAM</td>
<td>16.95</td>
<td>16.77</td>
<td>19.84</td>
</tr>
<tr>
<td>SR-ADAM</td>
<td>23.33</td>
<td>20.56</td>
<td>26.49</td>
</tr>
<tr>
<td>R-ADAM</td>
<td>23.99</td>
<td>21.88</td>
<td>33.56</td>
</tr>
</tbody>
</table>

• The binary model trained by BC-ADAM has comparable performance to the full-precision model.

• There is a performance gap between SR-ADAM and BC-ADAM across all models and datasets.

• Wide-Residual Networks are easier to train and generalize better.
Reducing the Inference Cost
Pruning Filters for Efficient ConvNets, ICLR’17

Reducing the Training Cost
Training Quantized Network: A Deeper Understanding, NIPS’17

Understanding the “black box”
Visualizing the Loss Landscape of Neural Nets, arXiv’17
Visualizing the Loss Landscape of Neural Nets

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein
Why Visualize the Loss Surface

Training neural networks requires minimizing a **high-dimensional non-convex** loss function

\[
L(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(x_i, y_i; \theta)
\]

which is a process of searching for a weight vector which gives the smallest loss value.

Geometric intuition of the loss surface is a key way of improving optimization methods and architecture design.
The Flatness/Sharpness of a Minimizer

“flat region” of the loss landscape are robust to
• data perturbations
• noise in the activations
• perturbations of the parameters (can be specified with lower-precision weights)

Interpreting High-Dimension Loss Surface

The high dimensionality of the weight space makes visualization of the loss surface very difficult.

The computation cost of high resolution surface visualization is very expensive.
Interpreting High-Dimension Loss Surface

1-D Linear Interpolation [Goodfellow’15]

Given two solutions $\theta_1$, $\theta_2$, and the targeted direction $\theta_2 - \theta_1$:

$$f(\alpha) = L(\theta_1 + \alpha(\theta_2 - \theta_1))$$

Interpreting High-Dimension Loss Surface

1-D Linear Interpolation [Goodfellow’15]

Given two solutions $\theta_1$, $\theta_2$, and the targeted direction $\theta_2 - \theta_1$:

$$f(\alpha) = L(\theta_1 + \alpha(\theta_2 - \theta_1))$$

Small-batch vs Large-batch

<table>
<thead>
<tr>
<th>Weight Decay</th>
<th>Loss</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>7.37%</td>
<td>11.07%</td>
</tr>
<tr>
<td>5e-4</td>
<td>6.00%</td>
<td>10.19%</td>
</tr>
</tbody>
</table>

![Graphs showing loss and accuracy comparison between small-batch and large-batch with different weight decays.](image-url)
Scale Invariance of NN’s Weights

The $\alpha$ scale transformation does not affect the generalization as the behavior of the function is identical.

The loss function will change very quickly in the direction of a small filter, and very slowly in the direction of large filter.
Normalized 2D Surface Plotting

1. Create two random directions $\delta$ and $\eta$

2. Normalize each filter in $\delta$ and $\eta$ to have the same norm of the corresponding filter

   $$\delta_i \leftarrow \frac{\delta_i}{\|\delta_i\|} \|\theta_i\| \quad \eta_i \leftarrow \frac{\eta_i}{\|\eta_i\|} \|\theta_i\|$$

3. Plot the function $f(\alpha, \beta) = L(\theta + \alpha \delta + \beta \eta)$
Normalized 1D Visualization: VGG-9

SGD, 128
WD = 0
7.37%
WD = 5e-4
6.00%

SGD, 8192
11.07%

Adam, 128
7.44%

Adam, 8192
10.91%

6.00%
Normalized 2D Visualization: VGG-9

WD = 0

SGD, 128

7.37%

SGD, 8192

11.07%

Adam, 128

7.44%

Adam, 8192

10.91%

WD = 5e-4

6.00%

10.19%

7.80%

9.52%
We now know how optimization hyperparameters affect the loss landscape optimizer, batch size, weight decay....

How does the network architecture affect the loss landscape? depth, width, shortcut connection....
Effect of Identity Mapping

ResNet-56

ResNet-56-noshort
Effect of Identity Mapping

ResNet-56

ResNet-56-noshort
Effect of Identity Mapping

ResNet-56

ResNet-56-noshort
Effect of Network Depth

ResNet

- 20 layers: 7.37%
- 56 layers: 5.89%
- 110 layers: 5.79%

VGG-like

- 20 layers: 8.18%
- 56 layers: 10.83% (lr=0.01)
- 110 layers: 16.44% (lr=0.01)
Effect of Network Width: Wide-ResNet-56-k

$k = 1$ | $k = 2$ | $k = 4$ | $k = 8$
---|---|---|---
ResNet | 5.89% | 5.07% | 4.34% | 3.93%
VGG-like | 13.31% | 10.26% | 9.69% | 8.70%

increased width prevents chaotic behavior, and skip connections dramatically widen minimizers!
Are we really seeing convexity?
Summary

- The local geometry of landscape has a correlation with the generalization.

- The sharpness of different minima cannot be compared with 1D linear interpolation due to the weight scale invariance.

- We provide a more accurate visual correlation between flatness and generalization, allowing side-by-side comparison.

- Some neural architectures are easier to minimize than others, such as wide networks, NNs with residual connections.
Reducing the Inference Cost
Pruning Filters for Efficient ConvNets, ICLR’17

Reducing the Training Cost
Training Quantized Network: A Deeper Understanding, NIPS’17

Understanding the “black box”
Visualizing the Loss Landscape of Neural Nets, arXiv’17

Future Work
Understanding the loss surface of RNN, GAN and RL.
Understanding the generalization ability of neural networks.
Architecture search with good loss landscape.
Optimization methods for training low-precision networks.
Selected Publications

- Hao Li, Zheng Xu, Gavin Taylor, Tom Goldstein. *Visualizing the Loss Landscape of Neural Nets*. International Conference on Learning Representations (ICLR) Workshop Track, 2018
  *in submission to* Neural Information Processing Systems (NIPS), 2018


Many Thanks!
Thanks for listening!
Repeatability of the Loss Surface Visualization

Figure 14: Repeatability of the surface plots for VGG-9 with filter normalization. The shape of minima obtained using 10 different random filter-normalized directions.
Figure 15: Repeatness of the 2D surface plots for ResNet-56-noshort. The model is trained with batch size 128, initial learning rate 0.1 and weight decay 5e-4. The final training loss is 0.192, the training error is 6.49 and the test error is 13.31.
Visualizing the Optimization Path

(a) SGD, WD=5e-4
(b) Adam, WD=5e-4
(c) SGD, WD=0
(d) Adam, WD=0
Normalized 1D Visualization: ResNet-56

WD = 0

WD = 5e-4
Normalized 2D Visualization: ResNet-56

WD = 0

SGD, 128

8.26%

SGD, 8192

13.93%

Adam, 128

9.55%

Adam, 8192

14.30%

WD = 5e-4

5.89%

10.59%

7.67%

12.36%
Effect of Network Width

ResNets for CIFAR-10

- ResNets-20 7.37%
- ResNets-56 5.89%
- ResNets-110 5.79%

ResNets for ImageNet

- ResNets-18 5.42%
- ResNets-34 4.73%
- ResNets-50 4.55%