Learning Accurate Low-bit Deep Neural Networks with Stochastic Quantization

Yinpeng Dong¹, Renkun Ni², Jianguo Li³, Yurong Chen³, Jun Zhu¹, Hang Su¹

¹ Department of CST, Tsinghua University
² University of Virginia
³ Intel Labs China
Deep Learning is Everywhere

- **Self-Driving**
- **Alpha Go**
- **Machine Translation**
- **Dota**
Limitations

- More data + deeper models $\rightarrow$ more FLOPs + larger memory

**Revolution of Depth**

- Computation Intensive
- Memory Intensive
- Hard to deploy on mobile devices
Low-bit DNNs for Efficient Inference

- High Redundancy in DNNs;
- Quantize full-precision (32-bits) weights to binary (1 bit) or ternary (2 bits) weights;
- Replace multiplication (convolution) by addition and subtraction;
Typical Low-bit DNNs

- **BinaryConnect:**
  \[ B_i = \begin{cases} 
  +1 & \text{with probability } p = \sigma(W_i) \\
  -1 & \text{with probability } 1 - p 
  \end{cases} \]

- **BWN:** minimize \( \|W - \alpha B\| \)
  \[ B_i = \text{sign}(W_i), \quad \alpha = \frac{\sum_{i=1}^d |W_i|}{d} \]

- **TWN:** minimize \( \|W - \alpha T\| \)
  \[ T_i = \begin{cases} 
  +1 & \text{if } W_i > \Delta \\
  0 & \text{if } |W_i| < \Delta, \\
  -1 & \text{if } W_i < -\Delta 
  \end{cases} \]
  \[ \alpha = \frac{\sum_{i \in I_\Delta} |W_i|}{|I_\Delta|} \]

\[ I_\Delta = \{i||W_i| > \Delta\}, \quad \Delta = \frac{0.7}{d} \sum_{i=1}^d |W_i| \]
Training & Inference of Low-bit DNN

- Let $W$ be the full-precision weights, $Q$ be the low-bit weights ($B$, $T$, $\alpha B$, $\alpha T$).
- Forward propagation: quantize $W$ to $Q$ and perform convolution or multiplication.
- Backward propagation: use $Q$ to calculate gradients.
- Parameter update: $W^{t+1} = W^t - \eta^t \frac{\partial L}{\partial Q^t}$.
- Inference: only need to keep low-bit weights $Q$. 
Motivations

- Quantize all weights simultaneously;
- Quantization error $\|W - Q\|$ may be large for some elements/filters;
- Induce inappropriate gradient directions.

- Quantize a portion of weights
- Stochastic selection
- Could be applied to any low-bit settings
Roulette Selection Algorithm

<table>
<thead>
<tr>
<th>Weight Matrix</th>
<th>Quantization Error</th>
<th>Stochastic Partition with r = 50%</th>
<th>Hybrid Weight Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 1.3 -1.1 0.75 0.85</td>
<td>0.2</td>
<td>Rotation</td>
<td>1.3 -1.1 0.75 0.85</td>
</tr>
<tr>
<td>C2 0.95 -0.9 1.05 -1.0</td>
<td>0.05</td>
<td>Selection Point</td>
<td>1 -1 1 -1</td>
</tr>
<tr>
<td>C3 1.4 -0.9 -0.8 0.9</td>
<td>0.2</td>
<td></td>
<td>1 -1 -1 1</td>
</tr>
<tr>
<td>C4 -1.2 0.8 1.0 -1.0</td>
<td>0.1</td>
<td>Rotation</td>
<td>-1.2 0.8 1.0 -1.0</td>
</tr>
</tbody>
</table>

Quantization Error: \[ e_i = \frac{\|W_i - Q_i\|_1}{\|W_i\|_1} \]

Quantization Probability: Larger quantization error means smaller quantization probability, e.g. \[ p_i \propto \frac{1}{e_i} \]

Quantization Ratio r: Gradually increase to 100%
Training & Inference

- Hybrid weight matrix $\tilde{Q}$

$$
\tilde{Q}_i = \begin{cases} 
Q_i & \text{if channel } i \text{ being selected} \\
W_i & \text{else} 
\end{cases}
$$

- Parameter update

$$
W^{t+1} = W^t - \eta^t \frac{\partial L}{\partial \tilde{Q}^t}
$$

- Inference: all weights are quantized; use $Q$ to perform inference
Ablation Studies

- Selection Granularity:
  - **Filter-level** > Element-level

- Selection/partition algorithms
  - **Stochastic** (roulette) > deterministic (sorting) ~ fixed (selection only at first iteration)

- Quantization functions
  - **Linear** > Sigmoid > Constant ~ Softmax
    - \( p_i = \frac{\exp(f_i)}{\sum \exp(f_i)} \), where \( f = \frac{1}{e} \)

- Quantization Ratio Update Scheme
  - **Exponential** > Fine-tune > Uniformly
    - 50% \(\rightarrow\) 75% \(\rightarrow\) 87.5% \(\rightarrow\) 100%
Results -- CIFAR

<table>
<thead>
<tr>
<th>Bits</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VGG-9</td>
<td>ResNet-56</td>
</tr>
<tr>
<td>FWN</td>
<td>32</td>
<td>9.00</td>
</tr>
<tr>
<td>BWN</td>
<td>1</td>
<td>10.67</td>
</tr>
<tr>
<td>SQ-BWN</td>
<td>1</td>
<td>9.40</td>
</tr>
<tr>
<td>TWN</td>
<td>2</td>
<td>9.87</td>
</tr>
<tr>
<td>SQ-TWN</td>
<td>2</td>
<td>8.37</td>
</tr>
</tbody>
</table>

Table 6: Test error (%) of AlexNet-BN and ResNet-18 trained with 5 different methods on the ImageNet dataset.
Results -- ImageNet

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Bits</th>
<th>AlexNet-BN</th>
<th>ResNet-18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>top-1</td>
<td>top-5</td>
</tr>
<tr>
<td>FWN</td>
<td>32</td>
<td>44.18</td>
<td>20.83</td>
</tr>
<tr>
<td>BWN</td>
<td>1</td>
<td>51.22</td>
<td>27.18</td>
</tr>
<tr>
<td>SQ-BWN</td>
<td>1</td>
<td>48.78</td>
<td>24.86</td>
</tr>
<tr>
<td>TWN</td>
<td>2</td>
<td>47.54</td>
<td>23.81</td>
</tr>
<tr>
<td>SQ-TWN</td>
<td>2</td>
<td>44.70</td>
<td>21.40</td>
</tr>
</tbody>
</table>

Table 6: Test error (%) of AlexNet-BN and ResNet-18 trained with 5 different methods on the ImageNet dataset.
Conclusions

- We propose a stochastic quantization algorithm for Low-bit DNN training
- Our algorithm can be flexibly applied to all low-bit settings;
- Our algorithm help to consistently improve the performance;
- We release our codes to public for future development
  
  □ [https://github.com/dongyp13/Stochastic-Quantization](https://github.com/dongyp13/Stochastic-Quantization)
Q & A