Systematic Neural Network Quantization

Amir Gholami, Z. Yao, Z. Dong, S. Kim, Z. Zheng, E. Tan, Q. Huang, S. Yu, L. Wang, Y. Wang, Michael Mahoney, Kurt Keutzer

University of California Berkeley, in collaboration with AWS

GTC Conference, April, 2021
Quantization:
Workhorse for Efficient Inference

- Uniform quantization is a mapping from floating point values to quantized integer values

Quantization

Let’s take a closer look at a layer

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
\end{bmatrix} =
\begin{bmatrix}
  w_{00} & w_{01} & w_{02} \\
  w_{10} & w_{11} & w_{12} \\
  w_{20} & w_{21} & w_{22} \\
  w_{30} & w_{31} & w_{32} \\
  w_{40} & w_{41} & w_{42} \\
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
\end{bmatrix}
\]

\[
y = W \cdot x
\]

Illustration from Sahni Manas
For simplicity, let’s consider symmetric quantization => Z=0

We first need to accumulate results in INT32 and then rescale back to INT4

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix} = 
\begin{bmatrix}
w_{00} & w_{01} & w_{02} \\
w_{10} & w_{11} & w_{12} \\
w_{20} & w_{21} & w_{22} \\
w_{30} & w_{31} & w_{32} \\
w_{40} & w_{41} & w_{42} \\
\end{bmatrix} \cdot 
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
\end{bmatrix}
\]

\[
y^{i32} = W^{i4}x^{i4}
\]

\[
y^{i4} = \text{Round}\left(\frac{S_x S_w}{S_y} y^{i32}\right)
\]

This is Integer-only (aka fixed-point/Dyadic quantization)

But not all quantization algorithms use this method
Simulated Quantization

- Simulated quantization performs arithmetic in software using FP32 arithmetic, but this can lead to discrepancy when deployed on integer-only hardware.
- Also many components such as BN or residual connection are computed using FP32 which is not executable on integer-only hardware.

Simulated/Fake Quantization Error

- Simulated/Fake quantization can create $O(1)$ error for simple operations such as residual connection. This is because rounding operation is not linear:

$$\text{Int}(a + b) \neq \text{Int}(a) + \text{Int}(b)$$

Example:

$$\text{Int}(2.4 + 1.3) = 4 \neq \text{Int}(2.4) + \text{Int}(1.3) = 3$$

- This difference is $O(1)$ and will propagate throughout the network, especially for low precision quantization as shown in the figure.

The normalized difference between fake quantization in Pytorch and corresponding values when deployed in hardware for ResNet50 on ImageNet. As one can see the error becomes quite significant, especially for low precision quantization.

• With integer only quantization we perform all the arithmetic using INT multiplication and addition, without any FP calculation for the entire inference. This includes:
  – CONV, BN, Residual Connection, Pooling, and even non-linear activation
  ➢ **BN**: Need to first fine-tune BN statistics using high precision and after correctly computing the statistics it can be quantized and fused with conv layer

## Integer-only Quantization Works: CV

### (a) MobileNetV2

<table>
<thead>
<tr>
<th>Method</th>
<th>Int Uni</th>
<th>BL</th>
<th>Precision</th>
<th>Size</th>
<th>BOPS</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>✓</td>
<td>73.03</td>
<td>W32A32</td>
<td>13.2</td>
<td>322</td>
</tr>
<tr>
<td>RVQuant (Park et al., 2018)</td>
<td>✗</td>
<td>✗</td>
<td>70.10</td>
<td>W8A8</td>
<td>3.3</td>
<td>20</td>
</tr>
<tr>
<td>CalibTIB (Hubara et al., 2020)</td>
<td>✗</td>
<td>✓</td>
<td>71.90</td>
<td>W8A8</td>
<td>3.3</td>
<td>20</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>73.03</td>
<td>W8A8</td>
<td>3.3</td>
<td>20</td>
</tr>
</tbody>
</table>

### (b) ResNet50

<table>
<thead>
<tr>
<th>Method</th>
<th>Int Uni</th>
<th>BL</th>
<th>Precision</th>
<th>Size</th>
<th>BOPS</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✓</td>
<td>✓</td>
<td>77.72</td>
<td>W32A32</td>
<td>97.8</td>
<td>3951</td>
</tr>
<tr>
<td>Integer Only (Jacob et al., 2018)</td>
<td>✓</td>
<td>✓</td>
<td>76.40</td>
<td>W8A8</td>
<td>24.5</td>
<td>247</td>
</tr>
<tr>
<td>RVQuant (Park et al., 2018)</td>
<td>✗</td>
<td>✗</td>
<td>75.92</td>
<td>W8A8</td>
<td>24.5</td>
<td>247</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>77.72</td>
<td>W8A8</td>
<td>24.5</td>
<td>247</td>
</tr>
</tbody>
</table>

### (c) InceptionV3

<table>
<thead>
<tr>
<th>Method</th>
<th>Int Uni</th>
<th>BL</th>
<th>Precision</th>
<th>Size</th>
<th>BOPS</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>✓</td>
<td>78.88</td>
<td>W32A32</td>
<td>90.9</td>
<td>5850</td>
</tr>
<tr>
<td>Integer Only (Jacob et al., 2018)</td>
<td>✓</td>
<td>✓</td>
<td>78.30</td>
<td>W8A8</td>
<td>22.7</td>
<td>366</td>
</tr>
<tr>
<td>RVQuant (Park et al., 2018)</td>
<td>✗</td>
<td>✗</td>
<td>74.19</td>
<td>W8A8</td>
<td>22.7</td>
<td>366</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>78.88</td>
<td>W8A8</td>
<td>22.7</td>
<td>366</td>
</tr>
</tbody>
</table>
# Integer-only Quantization Works: Transformers

(a) RoBERTa-Base

<table>
<thead>
<tr>
<th></th>
<th>Int-only</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>87.8</td>
<td>87.4</td>
<td>90.4</td>
<td>92.8</td>
<td>94.6</td>
<td>61.2</td>
<td>91.1</td>
<td>90.9</td>
<td>78.0</td>
<td>86.0</td>
</tr>
<tr>
<td>I-BERT</td>
<td>✓</td>
<td>87.5</td>
<td>87.4</td>
<td>90.2</td>
<td>92.8</td>
<td>95.2</td>
<td>62.5</td>
<td>90.8</td>
<td>91.1</td>
<td>79.4</td>
<td>86.3</td>
</tr>
<tr>
<td>Diff</td>
<td></td>
<td>-0.3</td>
<td>0.0</td>
<td>-0.2</td>
<td>0.0</td>
<td>+0.6</td>
<td>+1.3</td>
<td>-0.3</td>
<td>+0.2</td>
<td>+1.4</td>
<td>+0.3</td>
</tr>
</tbody>
</table>

(b) RoBERTa-Large

<table>
<thead>
<tr>
<th></th>
<th>Int-only</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>90.0</td>
<td>89.9</td>
<td>92.8</td>
<td>94.1</td>
<td>96.3</td>
<td>68.0</td>
<td>92.2</td>
<td>91.8</td>
<td>86.3</td>
<td>89.0</td>
</tr>
<tr>
<td>I-BERT</td>
<td>✓</td>
<td>90.4</td>
<td>90.3</td>
<td>93.0</td>
<td>94.5</td>
<td>96.4</td>
<td>69.0</td>
<td>92.2</td>
<td>93.0</td>
<td>87.0</td>
<td>89.5</td>
</tr>
<tr>
<td>Diff</td>
<td></td>
<td>+0.4</td>
<td>+0.4</td>
<td>+0.2</td>
<td>+0.4</td>
<td>+0.1</td>
<td>+1.0</td>
<td>0.0</td>
<td>+1.2</td>
<td>+0.7</td>
<td>+0.5</td>
</tr>
</tbody>
</table>

# Integer-only Quantization Works: ASR

## (a) QuartzNet-15x5

<table>
<thead>
<tr>
<th>Method</th>
<th>W/A</th>
<th>dev</th>
<th>test</th>
<th>Size (MB)</th>
<th>BOPs (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>32/32</td>
<td>3.80</td>
<td>10.05</td>
<td>3.82</td>
<td>10.08</td>
</tr>
<tr>
<td>Q-ASR</td>
<td>8/8</td>
<td>3.92</td>
<td>10.28</td>
<td>4.04</td>
<td>10.37</td>
</tr>
</tbody>
</table>

## (b) JasperDr-10x5

<table>
<thead>
<tr>
<th>Method</th>
<th>W/A</th>
<th>dev</th>
<th>test</th>
<th>Size (MB)</th>
<th>BOPs (T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>32/32</td>
<td>3.47</td>
<td>10.40</td>
<td>3.68</td>
<td>10.49</td>
</tr>
<tr>
<td>Q-ASR</td>
<td>8/8</td>
<td>3.47</td>
<td>10.49</td>
<td>3.68</td>
<td>10.57</td>
</tr>
</tbody>
</table>

What about Sub-INT8 Quantization?

Can we go even further and perform lower precision quantization?

Uniform low precision does not work as it can significantly degrade accuracy => Use mixed-precision

- But how should we set the precision for each kernel?
Hessian Aware Quantization

This is somewhat similar to the **Jenga** game. We only remove blocks that are not sensitive.

- Only use low precision quantization for insensitive parameters (flat loss landscape)
- Use high precision quantization for sensitive parameters (sharp loss landscape)

This sensitivity can be calculated through Hessian which quantifies the relative sharpness/flatness of the loss landscape.

Theorem 1 After quantizing the model to same precision, fine tuning layers that have smaller average trace of Hessian can achieve a smaller loss, compared to layers with larger average trace of Hessian.

\[
\text{average } tr(H) = \frac{1}{n} \sum_i H_{ii} = \frac{1}{n} \sum_i \lambda_i(H)
\]

- Relatively large Hessian trace => sharp loss landscape => Use high precision
- Relatively small Hessian trace => flat loss landscape => Use low precision

### Results: ResNet50

#### (b) ResNet50

<table>
<thead>
<tr>
<th>Method</th>
<th>Int Uni</th>
<th>BL</th>
<th>Precision</th>
<th>Size BOPS</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✓</td>
<td>✓</td>
<td>77.72</td>
<td>W32A32</td>
<td>97.8</td>
</tr>
<tr>
<td>Integer Only (Jacob et al., 2018)</td>
<td>✓</td>
<td>✓</td>
<td>76.40</td>
<td>W8A8</td>
<td>24.5</td>
</tr>
<tr>
<td>RVQuant (Park et al., 2018)</td>
<td>x</td>
<td>x</td>
<td>75.92</td>
<td>W8A8</td>
<td>24.5</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>77.72</td>
<td>W8A8</td>
<td>24.5</td>
</tr>
<tr>
<td>PACT (Choi et al., 2018)</td>
<td>x</td>
<td>✓</td>
<td>76.90</td>
<td>W5A5</td>
<td>16.0</td>
</tr>
<tr>
<td>LQ-Nets (Zhang et al., 2018)</td>
<td>x</td>
<td>x</td>
<td>76.50</td>
<td>W4A32</td>
<td>13.1</td>
</tr>
<tr>
<td>RVQuant (Park et al., 2018)</td>
<td>x</td>
<td>x</td>
<td>75.92</td>
<td>W5A5</td>
<td>16.0</td>
</tr>
<tr>
<td>HAQ (Wang et al., 2019)</td>
<td>x</td>
<td>x</td>
<td>76.15</td>
<td>WMPA32</td>
<td>9.62</td>
</tr>
<tr>
<td>OneBitWidth (Chin et al., 2020)</td>
<td>x</td>
<td>✓</td>
<td>76.70</td>
<td>W1*A8</td>
<td>12.3</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>77.72</td>
<td>W4/8A4/8</td>
<td>18.7</td>
</tr>
<tr>
<td>HAWQV3+DIST</td>
<td>✓</td>
<td>✓</td>
<td>77.72</td>
<td>W4/8A4/8</td>
<td>18.7</td>
</tr>
</tbody>
</table>

---


https://github.com/zhen-dong/hawq
Hessian AWare Quantization Timing on a 4-Titan RTX system:

- All the calculations, including Hessian, for bit selection takes **less than 1-hour**
- Up to **120x faster than AutoML** based methods, despite using Hessian, with **higher accuracy**

### Comparison between HAWQ-V2 and HAQ on ResNet50, InceptionV3 and SqueezeNext models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Top-1 Accuracy</th>
<th>W-Comp</th>
<th>Size(MB)</th>
<th>Search Time(hours)</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>Baseline Model</td>
<td>77.39</td>
<td>1.00×</td>
<td>97.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet50</td>
<td>HAQ</td>
<td>75.30</td>
<td>10.57×</td>
<td>9.22</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ResNet50</td>
<td>HAWQ-V2</td>
<td>75.56</td>
<td>12.24×</td>
<td>7.99</td>
<td>0.5</td>
<td>&gt; 20×</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>Baseline Model</td>
<td>77.45</td>
<td>1.00×</td>
<td>91.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InceptionV3</td>
<td>HAQ</td>
<td>71.60</td>
<td>10.00×</td>
<td>9.12</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>InceptionV3</td>
<td>HAWQ-V2</td>
<td>75.68</td>
<td>12.04×</td>
<td>7.57</td>
<td>0.4</td>
<td>&gt; 125×</td>
</tr>
<tr>
<td>SqueezeNext</td>
<td>Baseline Model</td>
<td>69.38</td>
<td>1.00×</td>
<td>10.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SqueezeNext</td>
<td>HAQ</td>
<td>65.87</td>
<td>10.00×</td>
<td>1.01</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>SqueezeNext</td>
<td>HAWQ-V2</td>
<td>68.38</td>
<td>9.40×</td>
<td>1.07</td>
<td>0.9</td>
<td>&gt; 55×</td>
</tr>
</tbody>
</table>

Summary: One Size Does Not Fit All

- Quantization (and pruning) of a model depends on the loss landscape which is a function of:
  - Model Architecture
  - Dataset
- We can automatically quantify the sensitivity to quantization and pruning through Hessian.


Images from Li H et al.
Summary

- Integer-only quantization is possible without any accuracy degradation for INT8:
  - Up to **4.5% higher accuracy compared to previous SOTA** from Google and close accuracy degradation gap with integer-only quantization
  - Works on compact models such as MobileNet (even with their strong baseline)
  - Works across different tasks: CV, ASR, and even NLP with I-BERT

- Sub-INT8 quantization can be performed even without AutoML through HAWQv3, a fully automatic framework without any manual bit selection
  - Key idea is to only perform sub-INT8 quantization for **insensitive layers**
    - Sensitivity can be efficiently calculated through Hessian
    - Up to **1.5x speed up with INT4/INT8 compared to INT8 quantization**

- All results verified with direct, end-to-end hardware implementation, and open-sourced
Thanks

Thank You for Listening!
amirgh@berkeley.edu