Lecture 9 - Compression (High-perf hardware)

Models are Getting Larger

**IMAGE RECOGNITION**

- **16X Model**
  - 8 layers
  - 1.4 GFLOP
  - -16% Error
  - 2012 AlexNet

- **152 layers**
  - 22.6 GFLOP
  - -3.5% error
  - 2015 ResNet

**SPEECH RECOGNITION**

- **10X Training Ops**
  - 80 GFLOP
  - 7,000 hrs of Data
  - -8% Error
  - 2014 Deep Speech 1

- **465 GFLOP**
  - 12,000 hrs of Data
  - -5% Error
  - 2015 Deep Speech 2

Microsoft

Baidu

Larger data sets and models lead to better accuracy but also increase computation time. Therefore progress in deep neural networks is limited by how fast the networks can be computed.

Likewise the application of convnets to low latency inference problems, such as pedestrian detection in self driving car video imagery, is limited by how fast a small set of images, possibly a single image, can be classified.

Acceleration
• Run a network faster (Performance, inf/s)
• Run a network more efficiently
  – Energy (inf/J)
  – Cost (inf/s$)

• Inference
  – Just running the network forward

• Training
  – Running the network forward
  – Back-propagation of gradient
  – Update of parameters

Key operations are Matrix Vector multiplications (dense data), sparse during inference.

Why GPUs? SIMD What about other approaches? We will cover 1) Reduced Precision Arithmetic (the goal is to increase the amount of data we can perform operations over), 2) Compression (reduce operations we have to perform), and 3) Better algorithms (Low-Rank approximation).

1. Reducing precision
Reducing precision

Reduces storage

Reduces energy

Improves performance

Has little effect on accuracy – to a point

DNN, key operation is dense $M \times V$

$$b_i = f \left( \sum_j w_{ij} a_i \right)$$
How much accuracy do we need in the computations:

\[ b_i = f \left( \sum_j w_{ij} a_i \right) \]

\[ w_{ij} = w_{ij} + \alpha a_i g_j \]

**Number Representation**

<table>
<thead>
<tr>
<th>Number Type</th>
<th>Exponent</th>
<th>Significand</th>
<th>Range</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>1 8 23</td>
<td>M</td>
<td>(10^{-38} - 10^{38})</td>
<td>.000006%</td>
</tr>
<tr>
<td>FP16</td>
<td>1 5 10</td>
<td>M</td>
<td>(6 \times 10^5 - 6 \times 10^4)</td>
<td>.05%</td>
</tr>
<tr>
<td>Int32</td>
<td>1 31</td>
<td>M</td>
<td>(0 - 2 \times 10^8)</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>Int16</td>
<td>1 15</td>
<td>M</td>
<td>(0 - 6 \times 10^4)</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>Int8</td>
<td>1 7</td>
<td>M</td>
<td>(0 - 127)</td>
<td>(\frac{1}{2})</td>
</tr>
</tbody>
</table>

FP32 = single precision
FP16 = half-precision
Traditional floating point (IEEE 754 style)

- $-1^s \times 2^e \times (1 + f)$
- $-1^s \times 2^0 \times (1 + \frac{1}{2} + \frac{1}{8}) = -1.625$

Cost of Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy (pJ)</th>
<th>Area (μm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>0.03</td>
<td>36</td>
</tr>
<tr>
<td>16b Add</td>
<td>0.05</td>
<td>67</td>
</tr>
<tr>
<td>32b Add</td>
<td>0.1</td>
<td>137</td>
</tr>
<tr>
<td>16b FP Add</td>
<td>0.4</td>
<td>1360</td>
</tr>
<tr>
<td>32b FP Add</td>
<td>0.9</td>
<td>4184</td>
</tr>
<tr>
<td>8b Mul</td>
<td>0.2</td>
<td>282</td>
</tr>
<tr>
<td>32b Mul</td>
<td>3.1</td>
<td>3495</td>
</tr>
<tr>
<td>16b FP Mul</td>
<td>1.1</td>
<td>1640</td>
</tr>
<tr>
<td>32b FP Mul</td>
<td>3.7</td>
<td>7700</td>
</tr>
<tr>
<td>32b SRAM Read (8KB)</td>
<td>5</td>
<td>N/A</td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Mixed Precision

Store weights as 4b using Trained quantization, decode to 16b

Store activations as 16b

16b x 16b multiply round result to 16b

accumulate 24b or 32b to avoid saturation

Batch normalization important to ‘center’ dynamic range
Stochastic rounding -> let’s say we add 0.3 to 0 100 times if we round 0.3 we will get zero. If we round it 70% of the time to 0 and 30% to 1 then we get E[Sum]=30

\[
\text{Round}(x) = \begin{cases} 
\lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \\
\lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor
\end{cases}
\]
Examples
3.5 has a 50% chance to round to 3, and a 50% chance to round to 4
2.4 has a 60% chance to round to 2, and a 40% chance to round to 3
1.6 has a 40% chance to round to 1, and a 60% chance to round to 2
-2.1 has a 90% chance to round to -2, and a 10% chance to round to -3
-4.7 has a 30% chance to round to -4, and a 70% chance to round to -5

Summary of Reduced Precision
- Can save memory capacity, memory bandwidth, memory power, and arithmetic power by using smaller numbers
- FP16 works with little effort
  - 2x gain in memory, 4x in multiply power
- With care, one can use
  - 8b for convolutions
  - 4b for fully-connected layers
- Batch normalization – important to ‘center’ ranges
- Stochastic rounding – important to retain small increments

2. Pruning
Pruning

before pruning

after pruning

pruning synapses

pruning neurons

Retrain to Recover Accuracy

Train Connectivity

Prune Connections

Train Weights

Accuracy Loss

-0.5%

-1.0%

-1.5%

-2.0%

-2.5%

-3.0%

-3.5%

-4.0%

-4.5%

40% 50% 60% 70% 80% 90% 100%

Pruned
3. Reduce weight storage for remaining weights

Trained Quantization
(Weight Sharing)

Figure 9: Compared with the original network, pruned network layer achieved 3× speedup on CPU, 3.5× on GPU and 4.2× on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

Intel Core i7 5930K: MKL CBLAS GEMV, MKL SPBLAS CSR MV
NVIDIA GeForce GTX Titan X: cuBLAS GEMV, cuSPARSE CSR MV
NVIDIA Tegra K1: cuBLAS GEMV, cuSPARSE CSR MV
Weight Sharing via K-Means

Huffman Coding

- In-frequent weights: use more bits to represent
- Frequent weights: use less bits to represent
5. Low Rank Approximations

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with \( d \) filters with filter size \( k \times k \times c \) to
  - A layer with \( d' \) filters \( k \times k \times c \)
  - A layer with \( d \) filter \( 1 \times 1 \times d' \)
Low Rank Approximation for FC

Build a mapping from row / column indices of matrix $W = [W(x, y)]$ to vectors $i$ and $j$: $x \leftrightarrow i = (i_1, \ldots, i_d)$ and $y \leftrightarrow j = (j_1, \ldots, j_d)$.

TT-format for matrix $W$:

$W(i_1, \ldots, i_d; j_1, \ldots, j_d) = W(x(i), y(j)) = G_1[i_1, j_1] G_2[i_2, j_2] \cdots G_d[i_d, j_d]$

<table>
<thead>
<tr>
<th>Type</th>
<th>1 im. time (ms)</th>
<th>100 im. time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU fully-connected layer</td>
<td>16.1</td>
<td>97.2</td>
</tr>
<tr>
<td>CPU TT-layer</td>
<td>1.2</td>
<td>94.7</td>
</tr>
<tr>
<td>GPU fully-connected layer</td>
<td>2.7</td>
<td>33</td>
</tr>
<tr>
<td>GPU TT-layer</td>
<td>1.9</td>
<td>12.9</td>
</tr>
</tbody>
</table>