Agenda

PART 1
1. Motivation
2. Classical Algorithms / Build Blocks
3. Hardware

PART 2
4. Recent Works
Motivation

Why Seeking Efficiency?
Facebook Under Fire: How Privacy Crisis Could Change Big Data Forever
AlphaGo

1,920 CPUs
280 GPUs
$3,000 Electricity Bill per game

AlphaGo vs Lee Sedol, Game 1
Lee, playing black, is in total control of this region

Lee Sedol vs AlphaGo, Game 2
And ends up taking a significant amount of territory from Lee

Lee, playing white, invades a region controlled by AlphaGo
AlphaGo defies convention by the invasion altogether, instead setting up its defenses in another area.
AppStore Download Restriction

To download an app over 100MB onto your mobile device,
you must connect to WiFi.

Putting this in perspective, VGG-16 has 130 million parameters (520MB).
Model Size

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>Layers</th>
<th>GFLOPS</th>
<th>Top-5 Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>AlexNet</td>
<td>8</td>
<td>1.4</td>
<td>~16%</td>
</tr>
<tr>
<td>2015</td>
<td>ResNet</td>
<td>152</td>
<td>22.6</td>
<td>~3.5%</td>
</tr>
</tbody>
</table>

16x increase from 2012 to 2015.
Speed of Training

Benchmarked with fb.resnet.torch using four M40 GPUs

Error Rate (%) vs. Training Time (days)

- ResNet18: 10.76 days, 2.5%
- ResNet50: 7.02 days, 5%
- ResNet101: 6.21 days, 7%
- ResNet152: 6.16 days, 10.5%

Error % ↓ 0.05
Days ↑ 3.5
Cost of data movement is much more huge. When compared, arithmetic ops is more like a noise.

- ALU: Fetching & moving data for computation
- Local SRAM (KB): 5 pJ (32 bits)
- On-Chip SRAM (MB): 50 pJ (32 bits)
- Off-Chip LPDDR DRAM (GB): 640 pJ (32 bits)

3.7 pJ (32-bit FP Multiplication)

c.f.) 1.1 pJ for 16-bit FP Mult
0.2 pJ for 8-bit Mult

Mark Horowitz "Computing's Energy Problem (and what we can do about it)", ISSCC 2014
Jaeyoung Chun

**Pruning**
- Weight Sharing
- Quantization
- Huffman Coding
- Low Rank Approximation
- Binary/Ternary Network
- Deep Compression
- Winograd

**Sparse-Wingograd**

**Deep Compression**
- SqueezeNet
- Dense-Sparse-Dense

**Parallelization**
- Model Distillation

**Mixed Precision**

**Deep Gradient Compression**

**Local Reparameterization**

**Efficient Inference**

**Efficient Training**

(*) no clear cut

**Algos: Classic/Building Block**

**Algos: Relatively New**

**Hardware**

**CPU**

**TPU** (Tensor Processing Unit)

**EIE** (Efficient Inference Engine)

**GPU**
Something to Keep in Mind...

- Losing any accuracy?
- Multiple methods interfering each other?

1. Pruning
2. Weight Sharing Trained Quantization
3. Huffman Coding

Understanding the underlying concepts & getting insights, and intuitions → Implementation and mathematical details
Algorithms for Efficient Inference
Pruning

Less number of parameters with almost no loss of accuracy

-0.01x^2 + x + 1

not just to reduce the network complexity, but also to avoid overfitting

Lecun et al. NIPS'89, Han et al. NIPS'15
Pruning

1. **Train Connectivity**
   Learn the connectivity via normal network training, as you would normally do.

2. **Prune Connections**
   Prune the small-weight connections from the network. (below a certain threshold)

3. **Retrain Weights**
   Retrain the network to learn the final weights for the remaining sparse connections.
Pruning

Pruning w/o Retraining

Accuracy Loss

Parameters Pruned Away
Pruning

Train Connectivity

Prune Connections

Train Weights

Accuracy Loss

Parameters Pruned Away

Pruning
Pruning+Retraining
Iterative Pruning and Retraining

Han et al. NIPS'15
Pretty much you have to look at all the pixels of the image.
If our brain loses 90% of neurons, can we still describe this image with this high accuracy?

**ORIGINAL**

a basketball player in a white uniform is playing with a **ball**.

**PRUNED 90%**

a basketball player in a white uniform is playing with a **basketball**.
Pruning in Human?

Peter Huttenlocher (1931-2013)

- New Born: 50 trillion synapses
- 1 Year Old: 1,000 trillion synapses
- Adolescent: 500 trillion synapses
- Adult: 1,000 trillion synapses

Weight Sharing

Before

<table>
<thead>
<tr>
<th>Weight</th>
<th>Edge (Connection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.09</td>
<td></td>
</tr>
<tr>
<td>2.12</td>
<td></td>
</tr>
<tr>
<td>1.92</td>
<td></td>
</tr>
<tr>
<td>1.87</td>
<td></td>
</tr>
</tbody>
</table>

After

<table>
<thead>
<tr>
<th>Weight</th>
<th>Edge (Connection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td></td>
</tr>
<tr>
<td>2.00</td>
<td></td>
</tr>
<tr>
<td>2.00</td>
<td></td>
</tr>
<tr>
<td>2.00</td>
<td></td>
</tr>
</tbody>
</table>

Less number of bits per parameter
Too accurate leads to overfitting anyway

Effective Weight or Shared Weight

Han et al. ICLR’16
Weight Sharing

Weights are not shared across layers. The shared weights approximate the original network because the method determines weight sharing after a network is fully trained.

K-Means Clustering (with K=4)

<table>
<thead>
<tr>
<th>Color (Cluster)</th>
<th>Original Weights</th>
<th>Effective Weight (Centroid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple</td>
<td>[2.09, 2.12, 1.92, 1.87]</td>
<td>2.00</td>
</tr>
<tr>
<td>Green</td>
<td>[1.48, 1.53, 1.49]</td>
<td>1.50</td>
</tr>
<tr>
<td>Orange</td>
<td>[0.09, 0.05, -0.14, 0, 0]</td>
<td>0.00</td>
</tr>
<tr>
<td>Yellow</td>
<td>[-0.98, -1.08, -0.91, -1.03]</td>
<td>-1.00</td>
</tr>
</tbody>
</table>

Only 4 numbers
## Weight Sharing

<table>
<thead>
<tr>
<th>Color</th>
<th>Effective Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple</td>
<td>2.00</td>
</tr>
<tr>
<td>Green</td>
<td>1.50</td>
</tr>
<tr>
<td>Orange</td>
<td>0.00</td>
</tr>
<tr>
<td>Yellow</td>
<td>-1.00</td>
</tr>
</tbody>
</table>

Also called “Codebook”
Weight Sharing / Quantization

<table>
<thead>
<tr>
<th>Color</th>
<th>Effective Weight</th>
<th>Index (Int)</th>
<th>Index (Binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple</td>
<td>2.00</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>Green</td>
<td>1.50</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>Orange</td>
<td>0.00</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Yellow</td>
<td>-1.00</td>
<td>3</td>
<td>11</td>
</tr>
</tbody>
</table>

No more 32-bit FP and only 2 bits

Lookup
Trained Quantization: Weight Distribution

- Naive quantization with 4-bit integer
  Sampling the space uniformly which doesn't capture the distribution well.

- Weight Sharing with K-Means better represents the distribution (esp. bimodal part)

The middle part is “pruned”, making the distribution look like bimodal.

Han et al. ICLR'16
Pruning and/or Quantization: Accuracy

- **Pruning + Quantization**
- **Pruning Only**
- **Quantization Only**

Accuracy Gain

Accuracy Identical

Accuracy Loss

Model Size Ratio After Compression

Compressed model is 8% of the original

Han et al. ICLR'16
Pruning and/or Quantization: Accuracy

Accuracy vs. Number of bits per effective weight in all layers.

- Top 5, quantized only
- Top 1, quantized only
- Top 5, pruned + quantized
- Top 1, pruned + quantized

Accuracy values:
- 0% for 1 bit
- 17% for 2 bits
- 34% for 3 bits
- 51% for 4 bits
- 68% for 5 bits
- 85% for 8 bits
Huffman Coding

2 bits * (24 + 5 + 4 + 3) = 72 bits

<table>
<thead>
<tr>
<th>Color</th>
<th>Effective Weight</th>
<th>Index (Int)</th>
<th>Index (Binary)</th>
<th>Count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple</td>
<td>2.00</td>
<td>0</td>
<td>00</td>
<td>24</td>
<td>66.67</td>
</tr>
<tr>
<td>Orange</td>
<td>0.00</td>
<td>2</td>
<td>10</td>
<td>5</td>
<td>13.89</td>
</tr>
<tr>
<td>Yellow</td>
<td>-1.00</td>
<td>3</td>
<td>11</td>
<td>4</td>
<td>11.11</td>
</tr>
<tr>
<td>Green</td>
<td>1.50</td>
<td>1</td>
<td>01</td>
<td>3</td>
<td>8.33</td>
</tr>
</tbody>
</table>

Frequent weights → use less bits to represent
Infrequent weights → use more bits to represent

6x6=36 numbers
Huffman Coding

<table>
<thead>
<tr>
<th>Color</th>
<th>Effective Weight</th>
<th>Index (Int)</th>
<th>Huffman Code</th>
<th>Count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple</td>
<td>2.00</td>
<td>0</td>
<td>1</td>
<td>24</td>
<td>66.67</td>
</tr>
<tr>
<td>Orange</td>
<td>0.00</td>
<td>2</td>
<td>00</td>
<td>5</td>
<td>13.89</td>
</tr>
<tr>
<td>Yellow</td>
<td>-1.00</td>
<td>3</td>
<td>011</td>
<td>4</td>
<td>11.11</td>
</tr>
<tr>
<td>Green</td>
<td>1.50</td>
<td>1</td>
<td>010</td>
<td>3</td>
<td>8.33</td>
</tr>
</tbody>
</table>

(1 bit * 24) + (2 bits * 5) + (3 bits * 4) + (3 bits * 3) = 55 bits

Han et al. ICLR'16
Deep Compression

Han et al. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding” (2015)
Compression Ratio (w/o accuracy loss)

- **40X**
  - LeNet-300: Original Size 0.03 MB, Compressed Size 1.07 MB

- **39X**
  - LeNet-5: Original Size 0.04 MB, Compressed Size 1.72 MB

- **35X**
  - AlexNet: Original Size 6.90 MB, Compressed Size 240.00 MB

- **49X**
  - VGGNet: Original Size 11.30 MB, Compressed Size 550.00 MB

- **10X**
  - GoogleNet: Original Size 2.80 MB, Compressed Size 28.00 MB

- **11X**
  - ResNet-18: Original Size 4.00 MB, Compressed Size 44.60 MB
Deep Compression

Large DNNs such as AlexNet, VGGNet are **fully fit** into **on-chip SRAM**.
SqueezeNet + Deep Compression

Can we apply Deep Compression to **already compact model** such as SqueezeNet?

Algorithms for Efficient Training
Moore’s Law

Scaling a CPU in a number of dimensions

- Transistors (thousands) (Original Moore’s Law)
- Single-thread Performance (SPECint) (Popular version of Moore’s Law)
- Frequency (MHz)
- Typical Power (watts)
- Number of Cores

Still increasing

A benchmark spec for CPU integer processing power

Chuck Moore, "DATA PROCESSING IN EXASCALE-CLASS COMPUTER SYSTEMS", The Salishan Conference on High Speed Computing (2011)
Without Moore’s Law (popular version)

Q: How are we going to continue to scale the performance we need to build a better DNN?

A: Use multiple processors in parallel
Data Parallelism

Run different images on different GPUs in parallel

CIFAR-10

Weight updates must be coordinated between workers.

Dally, High Performance Hardware for Machine Learning, NIPS'2015
Data Parallelism

Each GPU works on some portion of the dataset and sends their Δp (changes in weights) to Parameter Server.

Aggregate everyone's Δp's, add to the weights, calculate the new weights, and send them back to everybody.

$$p' = p + \Delta p$$
Data Parallelism

Add my weight changes and send the new weight to the next worker. Go all the way around the ring.
Hyper-Parameter Parallelism

Try many alternative networks in parallel

- Different number of layers
- Different size of convolutional kernels
- Different number of neurons per layer
- ...

Search in the parameter space
Mixed Precision Training

More precision than required → reduce precision

Save by factor of 2

- Storage
- Memory bandwidth
## Mixed Precision Training: Comparison

### AlexNet

<table>
<thead>
<tr>
<th>Mode</th>
<th>Top1 accuracy, %</th>
<th>Top5 accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fp32</td>
<td>58.62</td>
<td>81.25</td>
</tr>
<tr>
<td>Mixed precision training</td>
<td>58.12</td>
<td>80.71</td>
</tr>
</tbody>
</table>

### Inception V3

<table>
<thead>
<tr>
<th>Mode</th>
<th>Top1 accuracy, %</th>
<th>Top5 accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fp32</td>
<td>71.75</td>
<td>90.52</td>
</tr>
<tr>
<td>Mixed precision training</td>
<td>71.17</td>
<td>90.10</td>
</tr>
</tbody>
</table>

### ResNet-50

<table>
<thead>
<tr>
<th>Mode</th>
<th>Top1 accuracy, %</th>
<th>Top5 accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fp32</td>
<td>73.85</td>
<td>91.44</td>
</tr>
<tr>
<td>Mixed precision training</td>
<td>73.6</td>
<td>91.11</td>
</tr>
</tbody>
</table>

Hardware
Quick Overview of Hardware Side, But Why?

01 We write algorithms and software that runs on hardware.

02 Some of the algorithms we reviewed are actually used in the hardware design.
Google TPU

- Tensor Processing Unit
- Compared to GPU & CPU
  - 15x to 30x faster
  - 30x to 80x better energy efficiency
- Only internal use
  - e.g. AlphaGo, Street View, Photos, ...

Can be inserted into a SATA hard disk slot for easy/fast deployment to existing server infrastructure
Google Cloud TPU (2nd gen)

- Mid Feb 2018
  Cloud TPU\textsuperscript{BETA} announced.
- Supports for inference as well as training
Google Cloud TPU (2nd gen)
The Secret of Google TPU

Quantization
Optimization technique that uses an 8-bit integer to approximate an arbitrary value between a preset minimum and a maximum value.

CISC Instruction Set
High-level instructions specifically designed for neural network inference.

Matrix Multiply Unit
Processes hundreds of thousands of operations (= matrix operation) in a single clock cycle.

Minimal Design
Optimized for neural network inference only. In the TPU, the control logic is minimal and takes under 2% of the die.
Google TPU: Quantization

We have already seen the power of Quantization when discussing Deep Compression.

-3.4E+38

Not smooth, but with careful design, we can still prevent accuracy loss.
Google TPU: CISC Instruction Set

Reduced vs. Complex Instruction Set Computer

**RISC**

LOAD A, 2:3  
LOAD B, 5:2  
PROD A, B  
STORE 2:3, A

**CISC**

MULT 2:3, 5:2

Low-level simple instructions that are commonly used
- e.g. load, store, add, multiply

High-level instructions that perform complex operations
- e.g. compute multiply-and-add many times

# Google TPU: CISC Instruction Set

<table>
<thead>
<tr>
<th>TPU Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read_Host_Memory</td>
<td>Read data from memory</td>
</tr>
<tr>
<td>Read_Weights</td>
<td>Read weights from memory</td>
</tr>
<tr>
<td><strong>MatrixMultiply / Convolve</strong></td>
<td>Multiply or convolve with the data and weights, accumulate the results</td>
</tr>
<tr>
<td>Activate</td>
<td>Apply activation functions</td>
</tr>
<tr>
<td>Write_Host_Memory</td>
<td>Write result to memory</td>
</tr>
</tbody>
</table>

High-level instructions specifically designed for neural network inference.
Google TPU: Matrix Multiplier Unit

- **Unified Buffer for Local Activations** (96K x 256 x 8b = 24 MiB)
- **Matrix Multiply Unit** (256 x 256 x 8b = 64K)
- **Activations** (65,536 × 700M operations per sec)
- **Accumulators** (4K x 256 x 32b = 4 Mib)
- **Control**
- **Activation Pipeline**
- **Host Interface**
- **DRAM Port DDR3**
- **DRAM Port DDR3**
- **PCle Interface**
- **Misc I/O**

65,536
(multiply-and-add operations per cycle)

700
(TPU clock in MHz)

46\times10^{12}
(65,536 \times 700M operations per sec)
Google TPU: Minimal Design

Unified Buffer for Local Activations
(96K x 256 x 8b = 24 MiB)

Matrix Multiply Unit
(256 x 256 x 8b = 64K)

Data

Others
21.0%

I/O
10.0%

Compute
30.0%

Control
2.0%

37.0%

Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit" (2017)
EIE (Efficient Inference Engine)

**Weight Sharing**
With K-means clustering, e.g. the blues (2.09, 2.12, 1.92, 1.87) are treated as 2.0 instead (i.e. effective weights)

**Sparse Weight**
\[ W \times A = 0 \]
If \( W = 0 \) → \( 0 \times A = 0 \)

**Sparse Activation**
\[ W \times A = 0 \]
If \( A = 0 \) → \( W \times 0 = 0 \)

**Benefits**
- 10x less computation
- 5x less memory footprint
- 8x less memory footprint
- 3x less computation

Han et al. ISCA'16
EIE: Speedup and Energy Efficiency

**Speedup**
- Alex-8: 14 (CPU Dense), 0.5 (GPU Dense), 0.3 (mGPU Dense), 60 (EIE)
- VGG-8: 10 (CPU Dense), 1 (GPU Dense), 1 (mGPU Dense), 9 (EIE)
- NT-LSTM: 1 (CPU Dense), 1 (GPU Dense), 1 (mGPU Dense), 0.5 (EIE)

**Energy Efficiency**
- Alex-8: 14826 (CPU Dense), 7 (GPU Dense), 7 (mGPU Dense), 8053 (EIE)
- VGG-8: 11828 (CPU Dense), 5 (GPU Dense), 5 (mGPU Dense), 8053 (EIE)
- NT-LSTM: 8053 (CPU Dense), 5 (GPU Dense), 7 (mGPU Dense), 8053 (EIE)
Break