Exploration of Numerical Precision in Deep Neural Networks

Zhaoqi Li, Yu Ma, Catalina Vajiac, Yunkai Zhang
Industry Mentors: Nicholas Malaya, Allen Rush
Academic Mentor: Hangjie Ji

Research in Industrial Projects for Students (RIPS)
2017

Overview

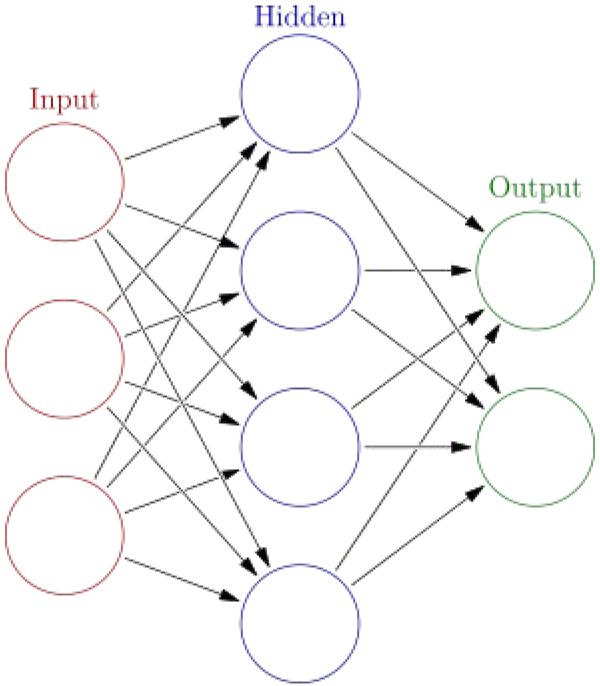
- Introduction
  - Background
  - Motivation
  - Goals
- Method
  - Arbitrary Precision
  - Mixed Precision
  - Quantization
- Future Work

AMD - The Company

- Semiconductor company
- Circuit design & computer architecture
- Interested in performance of algorithms

Convolutional Neural Network

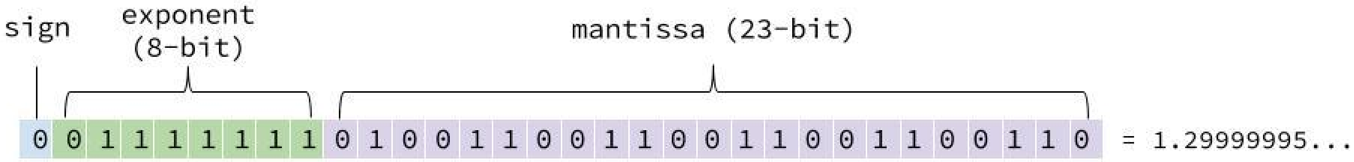
- What is a neural network?
  - Motivated by human brain
- Convolutional: sparsely-connected
  - Avoid waste of unnecessary interactions between units



Numerical Representation in Memory

- $x = \pm 1.\textit{mantissa} * 2^{\text{exponent}}$

- 1.3?



Motivation for Reduced Precision

- Memory-bound systems
  - Neural networks for simple devices (i.e. mobile)
- GPUs
  - Overhead for transferring data from CPU to GPU
  - Good for performing many small tasks in parallel
- Distributed ML Algorithms

Previous Work

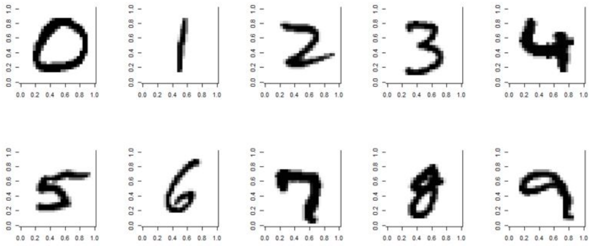
- Reduced precision works for inference (Gupta, et al. 2015)
- Low precision works for the training process (Courbariaux, et al. 2014)
- Intel supports half precision, Google supports 8-bit arithmetic

Goals

- Gain understanding of
  - Deep Neural Networks (DNNs)
  - Numerical representations in memory
  - Computer arithmetic
- Optimize deep learning performance with low precision
  - Precision/stability vs. memory/speed
- Estimate precision bounds for particular networks

Arbitrary Precision: Introduction

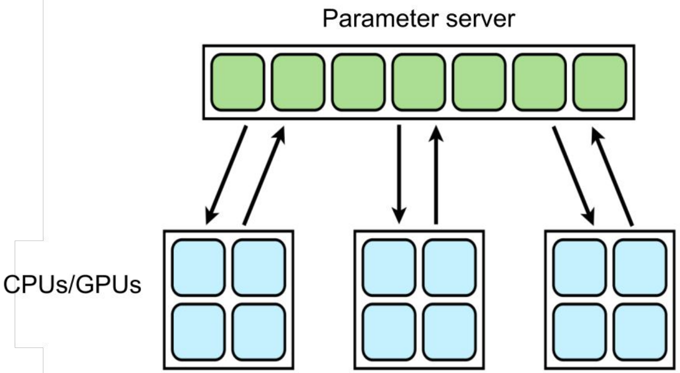
- Represent a number with any number of bits
- Software:
  - Tensorflow: black boxed functions
  - CAFFE: only supports float 32
  - Theano: more transparent than the rest
- Datasets:
  - MNIST (digit recognition)
  - CIFAR10 (object classification)



Arbitrary Precision: Truncation

Different levels of truncation:

- After each basic arithmetic operation
  - Emulates hardware-like setting where nothing can be done with higher precision
  - Complicated: need to redefine Theano functions
- After each batch
  - Represents parameter server for distributed ML algorithms
  - Not necessary to redefine Theano functions

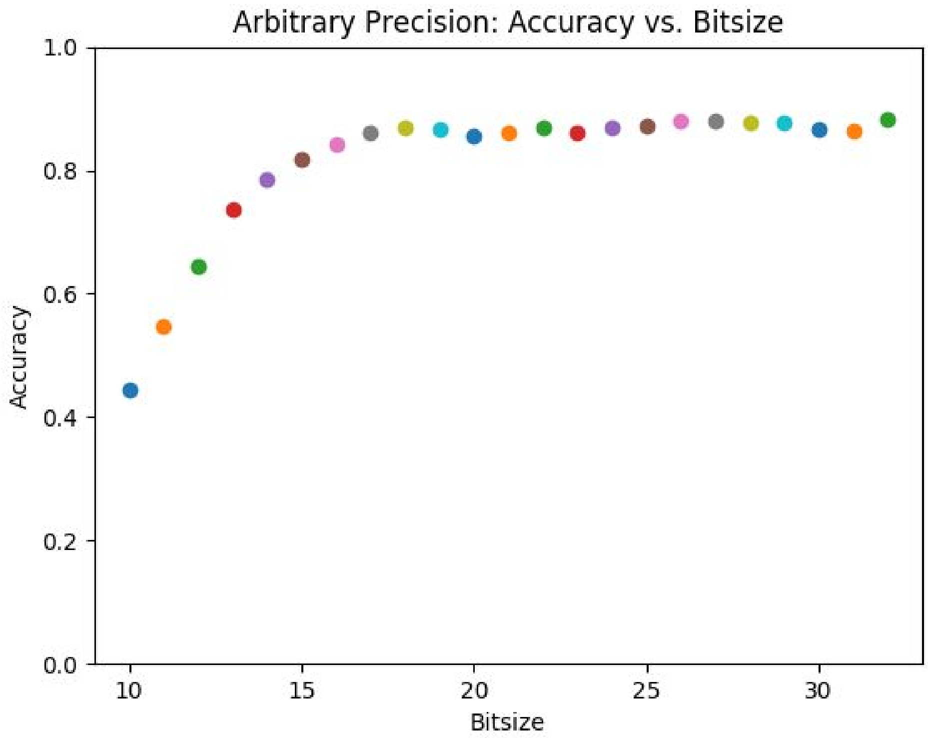


Arbitrary Precision: Implementation

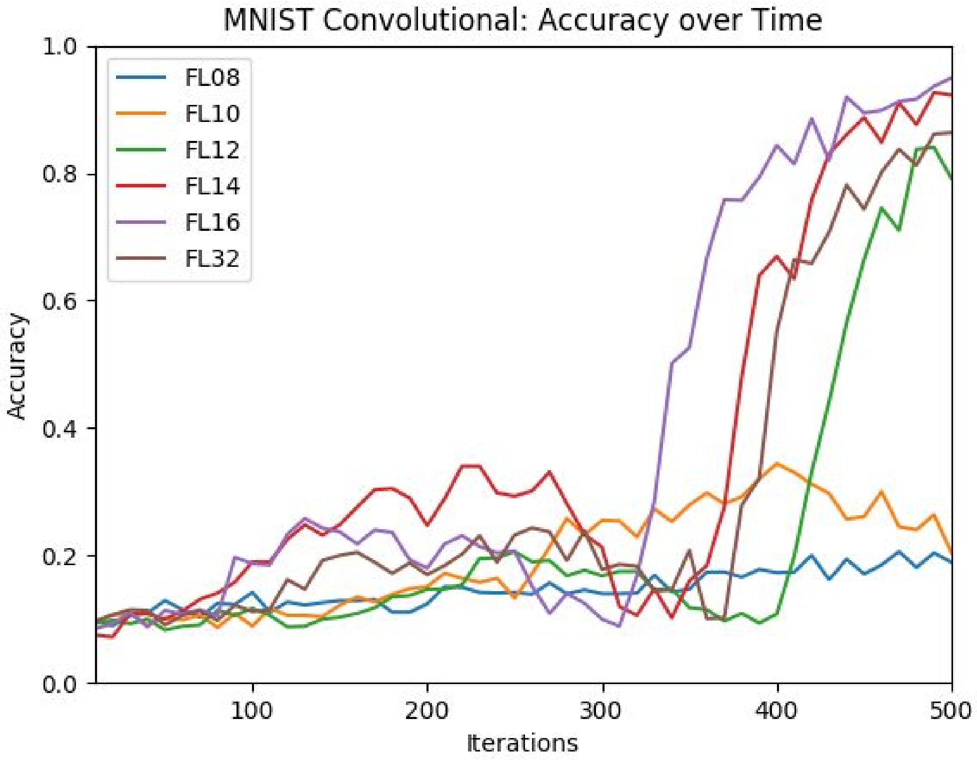
- How to truncate:
  - Create filter
  - Bitwise AND with original number

```
   0 0111 1111 1111 1100 1100 1100 1100 110   1.9875
&
   1 1111 1111 1111 1110 0000 0000 0000 000   16-bit filter
   ----------------------------------------
   0 0111 1111 1111 1100 0000 0000 0000 000   1.984375
```

Arbitrary Precision: Logistic Regression Results

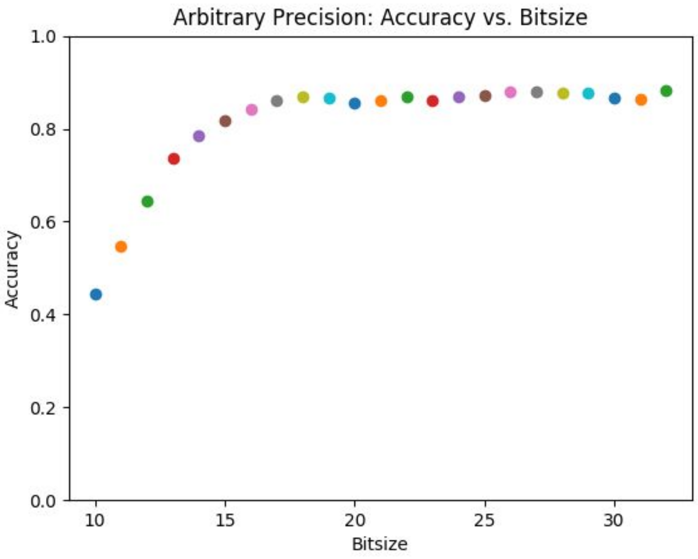


Arbitrary Precision: CNN Results



Mixed Precision

- Change precision after a certain number of iterations
  - Could happen multiple times per several epochs/batches
  - Current direction: low to high
- The Hypothesis
  - High precision determines the better local minima to approach
  - Low precision speeds up the convergence process



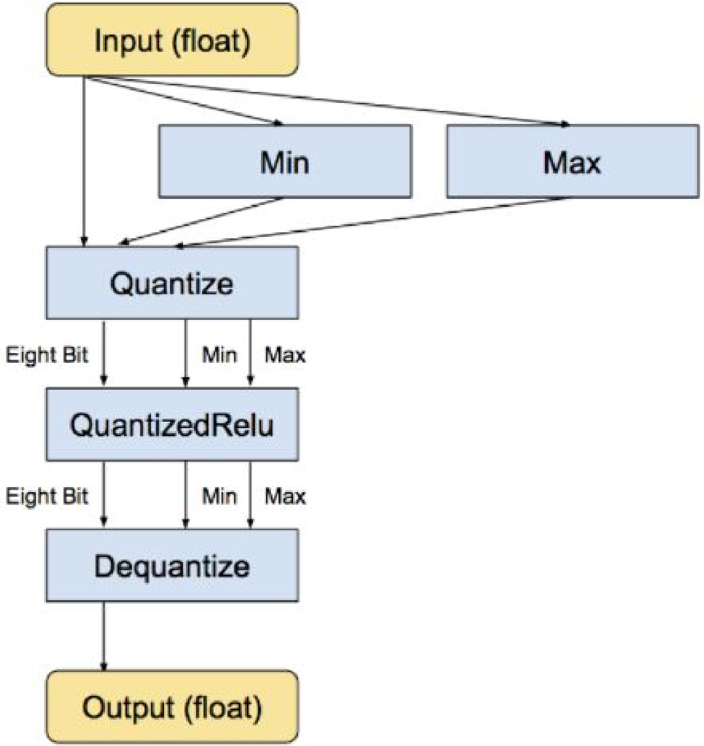
Quantization - Introduction

- Google TPU contains 256 x 256 8bit multiply-add computational units
- Google low precision GEMM library
- Reduce computational resources
- Theano based code by Matthieu Courbariaux

Quantized	Float
0	-10.0
255	30.0
128	10.0

Quantization - Implementation

- BinaryNet (quantize to -1 and 1)
  - Benchmark performance on MNIST, CIFAR-10 and SVHN
- 8-bit CNN
  - Quantize input, weight, and output gradient
  - Difficult to customize arbitrary uint datatype
- Nvidia GPU, Theano configuration for CUDA, Lasagne
  - Fast compilation and fast convergence



Next Step I

- One model for all

  - # of layers/neurons vs runtime

  - accumulation error?

- Truncation after each layer

  - need to feed data first

  - automatic differentiation

Next Step II

Python Code

```
1. y = T.nnet.softmax(T.dot(x, W) + b)
2. cross_entropy = -T.sum(y_*T.log(y))
3. g_W = T.grad(cost=cross_entropy, wrt=W)
   g_b = T.grad(cost=cross_entropy, wrt=b)
4. updates = [(W, W - learning_rate * g_W), (b, b-learning_rate * g_b)]
5. train_model = theano.function(inputs=[x, y_], outputs=cross_entropy,
updates=updates)
```

Human Language

```
1. y = xW + b
2. Cost function = ∑(y_*T.log(y))
3. Differentiate cost function with respect to w and b
4. Update W = W - learning_rate * g_W, b = b-learning_rate * g_b
5. Define train_model that takes x, y_ as inputs, find the cost function,
and update w and b correspondingly.
```

Next Step II

Processed Using Symbols

```
y = T.nnet.softmax(T.dot(x, W) + b)
cross_entropy = -T.sum(y_*T.log(y))
g_W = T.grad(cost=cross_entropy, wrt=W)
g_b = T.grad(cost=cross_entropy, wrt=b)
updates = [(W, W - learning_rate * g_W), (b, b - learning_rate * g_b)]
train_model = theano.function(inputs=[x, y_], outputs=cross_entropy,
updates=updates)
```

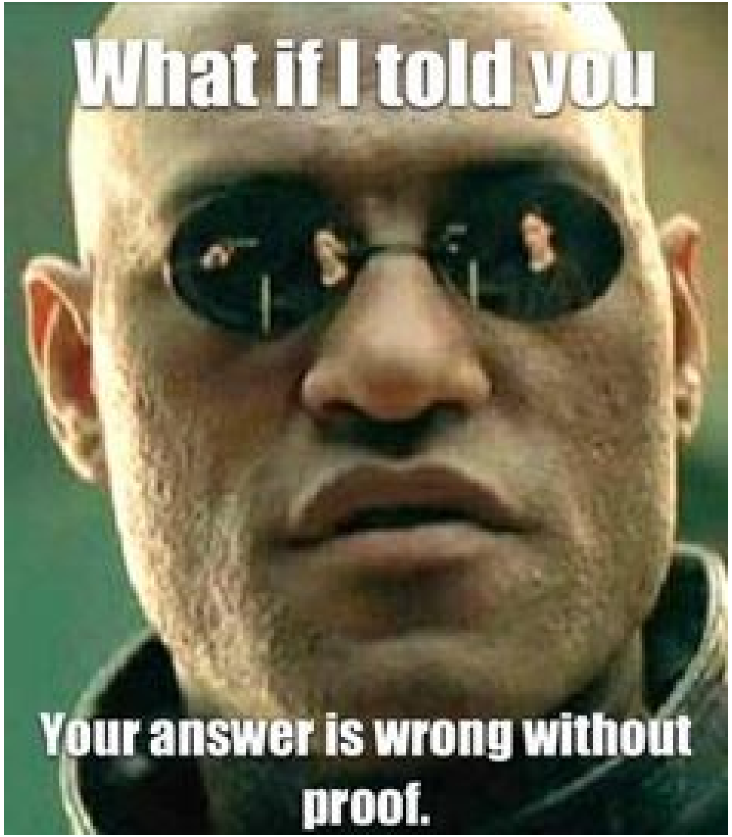
#Y = xW+b

#gradient

Want to change to numbers to truncate

Finally...

- “Prove” it!



Thank you for your attention!

Questions?

References

[1] Gupta, Suyog, et al. “Deep Learning with Limited Numerical Precision." ICML. 2015.

[2] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[3] Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Training deep neural networks with low precision multiplications." arXiv preprint arXiv:1412.7024 (2014).