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)

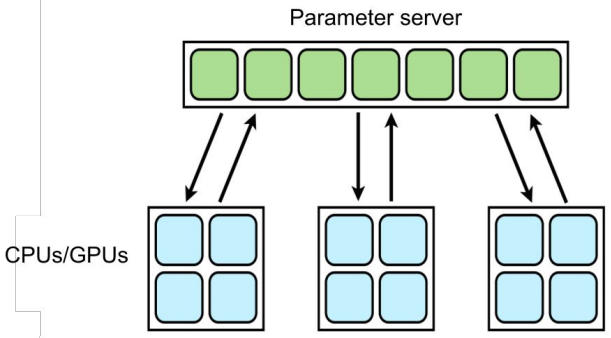
2017

About RIPS

- Research in Industrial Projects for Students (RIPS)
  - Conducted through Institute of Pure and Applied Mathematics (IPAM) at UCLA
  - Nine projects, each sponsored by a different company

Motivation for Reduced Precision

- Reduces time and space required for computation
  - Memory-bound or compute-bound systems
    - e.g. CPUs, GPUs, mobile devices
- Distributed ML Algorithms
  - Reduced precision => reduced message size
- Deep Neural Networks (DNNs):
  - Computationally expensive
  - Exhibit natural resilience to errors



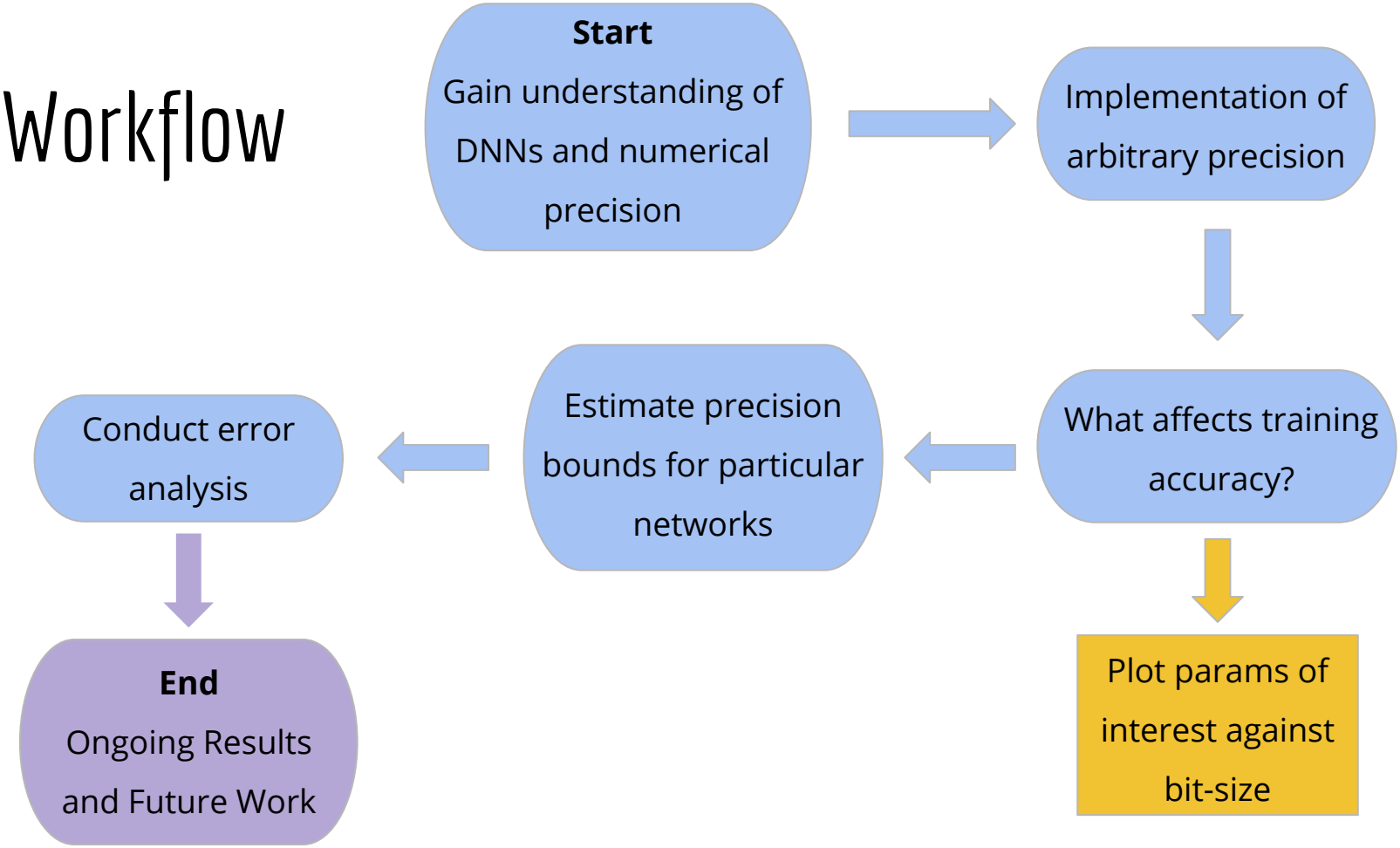
Previous Work

- Train with 16-bit and stochastic rounding (Gupta, et al. 2015)
- Train with low-precision multiplications (Courbariaux, et al. 2014)
- Train with binary weights (Courbariaux, et al. 2015)
- One-bit gradient for parallelization of SGD (Seide, et al. 2014)

However...

- No previous work capable of predicting sensitivity to precision
- No estimate of precision tolerance established
- No previous study on exactly how bit size affects neural network
    - only an approximate analysis on CNN's resiliency to approximation error in general

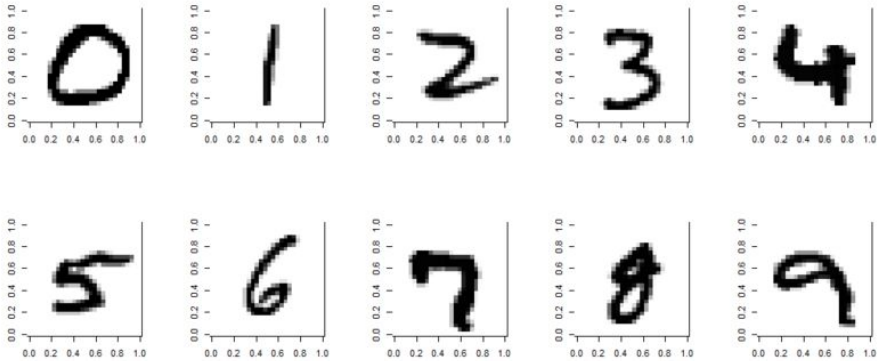
Workflow


Start
Gain understanding of DNNs and numerical precision
Implementation of arbitrary precision
What affects training accuracy?
Plot params of interest against bit-size
Estimate precision bounds for particular networks
Conduct error analysis
End
Ongoing Results and Future Work

MNIST Dataset

- Digit recognition (Classification Problem)
- 28x28 pixel, grayscale
- Input: 60,000 images for training

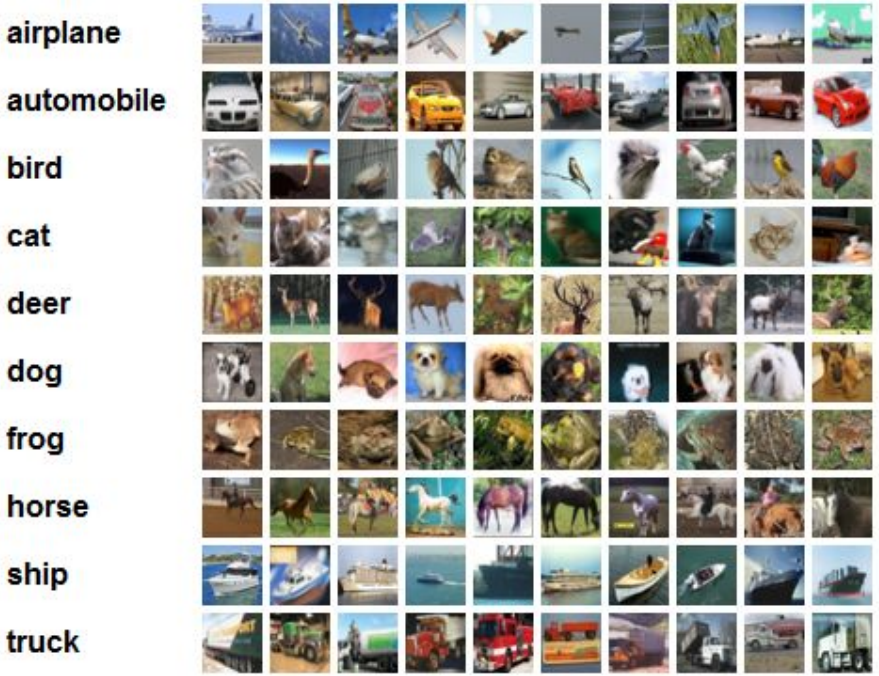
  10,000 images for testing
- Output: 10-element vector
- Simple ML dataset



  - 1 iteration = go through the entire dataset once with mini-batch
  - Takes less than 50 iterations to converge
  - Usually get to over 95% accuracy

CIFAR-10 Dataset

- Object classification
- 32x32 pixel, colored
- Input: 60,000 images for training
  10,000 images for testing
- Output: 10-element vector
- More complex than MNIST
  - Needs higher precision
  - ~80%: good

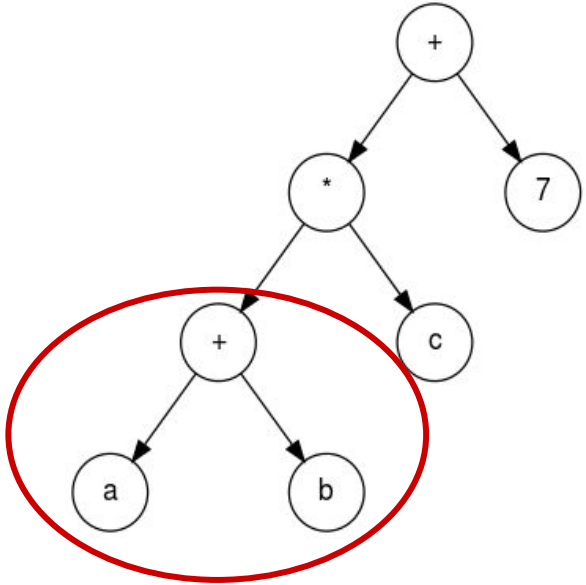


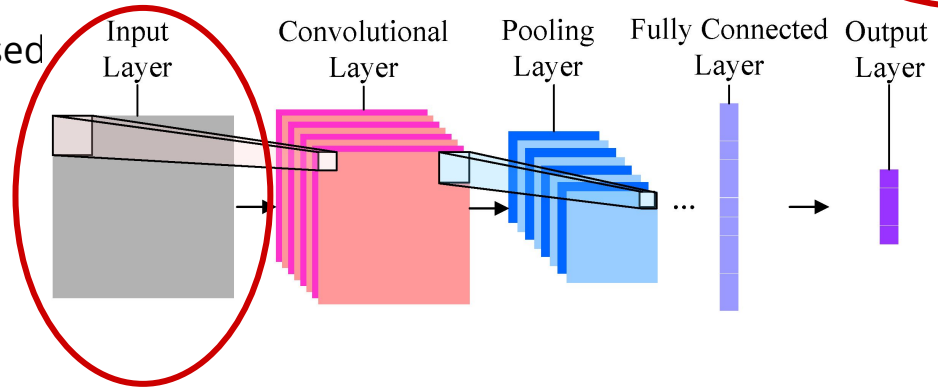
Software

- Need framework in order to experiment with precision
- Common frameworks:
  - Tensorflow: too many basic operations per function
  - CAFFE: only supports float-32 by default
  - Theano: less “black-boxed” than Tensorflow

Different Truncation Methods

- Truncation by elementary operation
  - Future work
- Truncation by layer
  - Currently used method
- Truncation by batch
  - Currently used



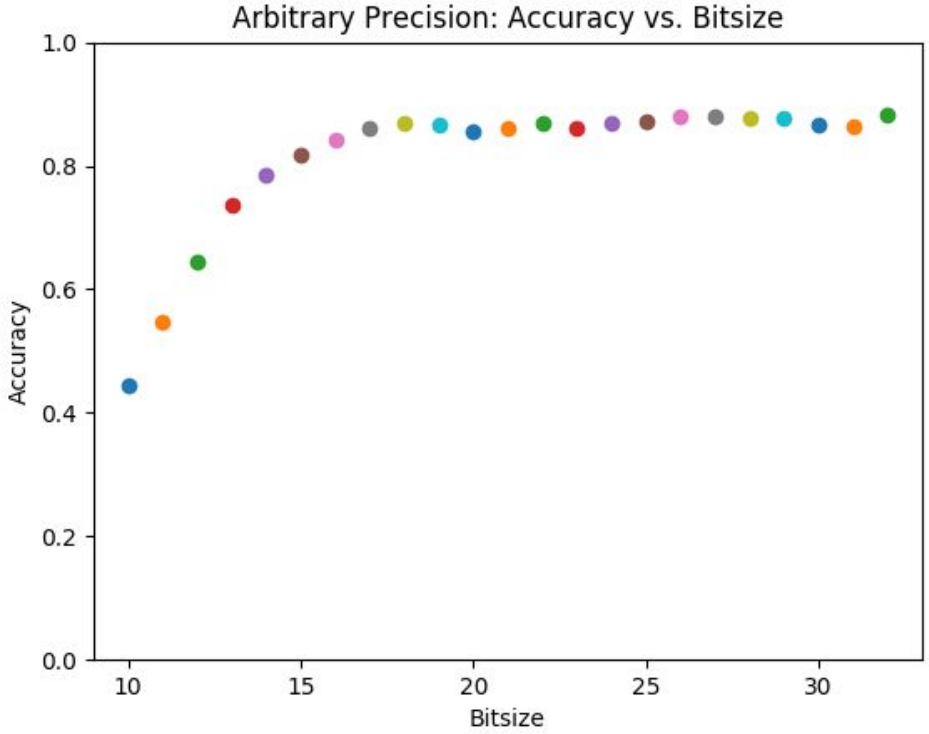


Arbitrary Precision: Introduction

- Goal: be able to explore low precision on a continuous scale
- To implement:
  - Create filter
  - Bitwise AND with original number

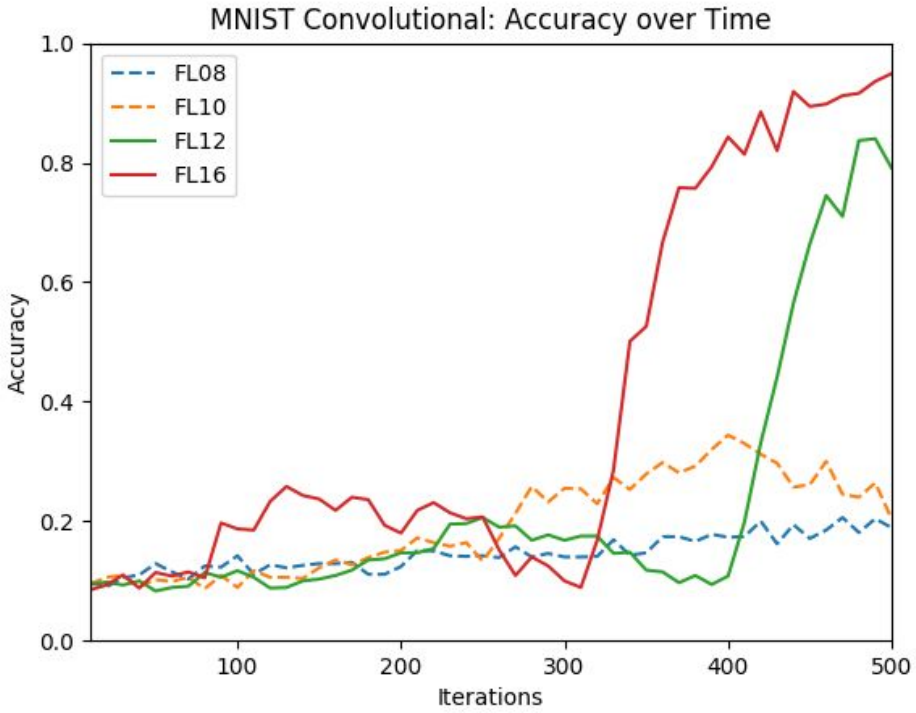
```
   0 0111 1111 1111 1100 1100 1100 1100 110    1.98749995
&
   1 1111 1111 1111 1110 0000 0000 0000 000    16-bit filter
   ---------------------------------------
   0 0111 1111 1111 1100 0000 0000 0000 000    1.984375
```

Arbitrary Precision: Logistic Regression Results



- y = Wx + b
- Applied to MNIST
- Accuracy after running logistic regression for 1000 iterations
- Avg. of 30 trials per bitsize

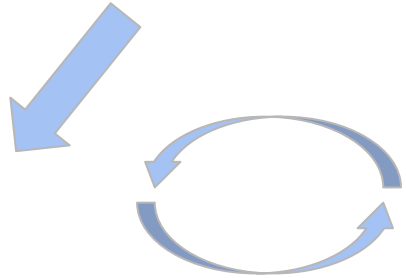
Arbitrary Precision: CNN Results



- More complicated than logistic regression
- Accuracy vs. iterations
- One trial per bitsize
- 12 bits: inflection point

How to generalize the results?

Start with one simple architecture



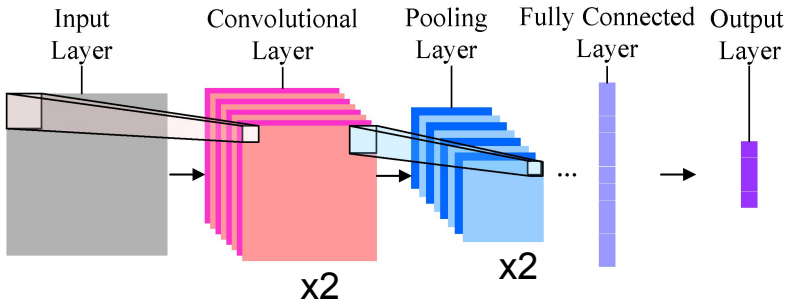
Change some parameters

Does reduced precision still work?

Big question: what parameters are more sensitive to reduced precision?

Current Architecture

- Input - 2x convolution/pooling - dense - softmax - output
  - Initial weights: small random numbers; mini-batch: 128; activation: ReLu
- Reason:
  - Fast runtime
  - Parameters can be easily changed
  - Convenient to feed in different datasets
- Data analysis:
  - Final accuracy
  - Convergence speed



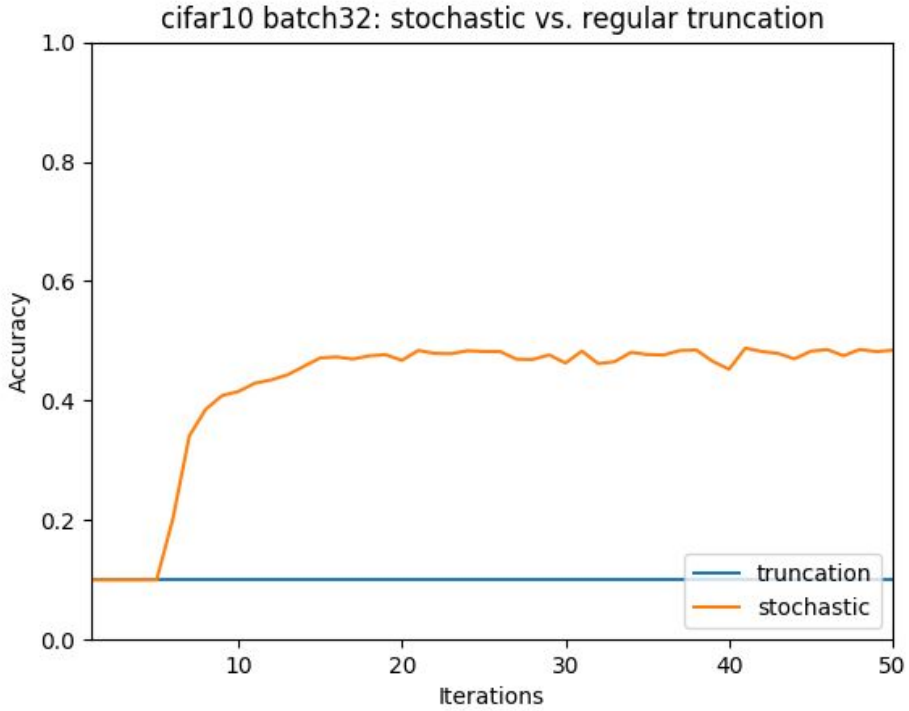
Exploration of Parameters of Interest

- Rounding scheme (stochastic/deterministic)
- Number of dense layers
- Number of units in dense layer
- Batch size
- Initial weight conditions

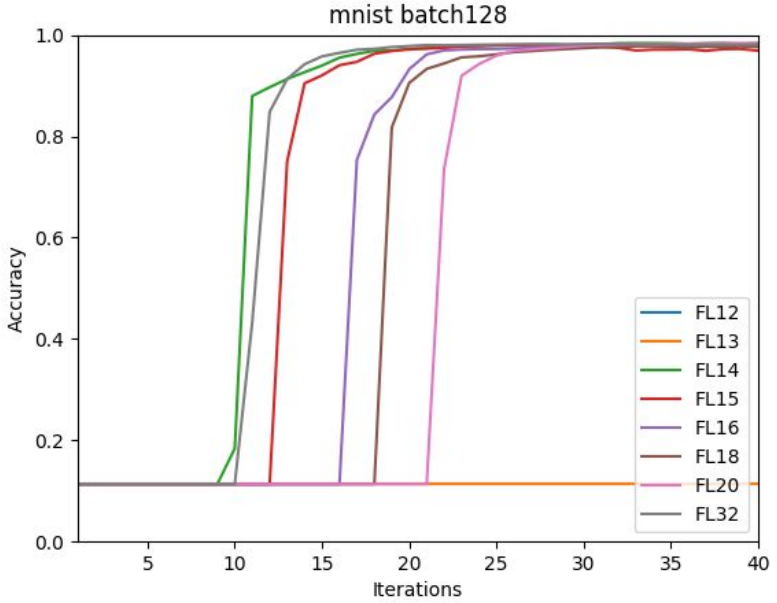
Truncation vs. Stochastic Rounding

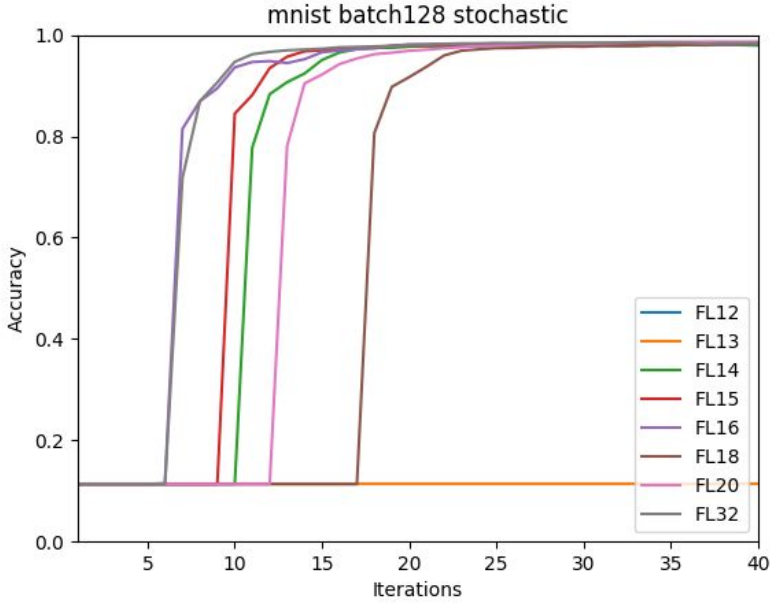
- Two methods to implement reduced precision
- Truncation: always rounding down
- Stochastic rounding:
  - Rounding x to $\lfloor x \rfloor + \varepsilon$ is proportional to how close x is to $\lfloor x \rfloor + \varepsilon$
  - Unbiased rounding scheme
- Trends using stochastic rounding:
  - Some cases converge vs. not converging with truncation
  - ↑ speed of convergence

Truncation vs. Stochastic Rounding: Results



Truncation vs. Stochastic Rounding: Results



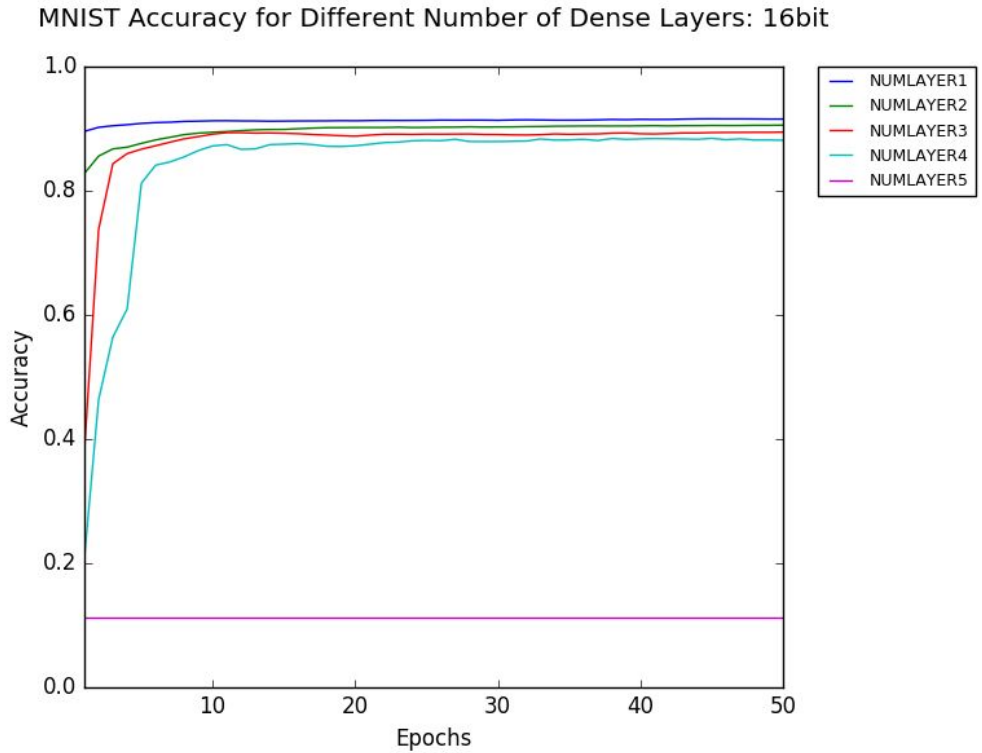


On average, stochastic rounding improves speed of convergence by 25%

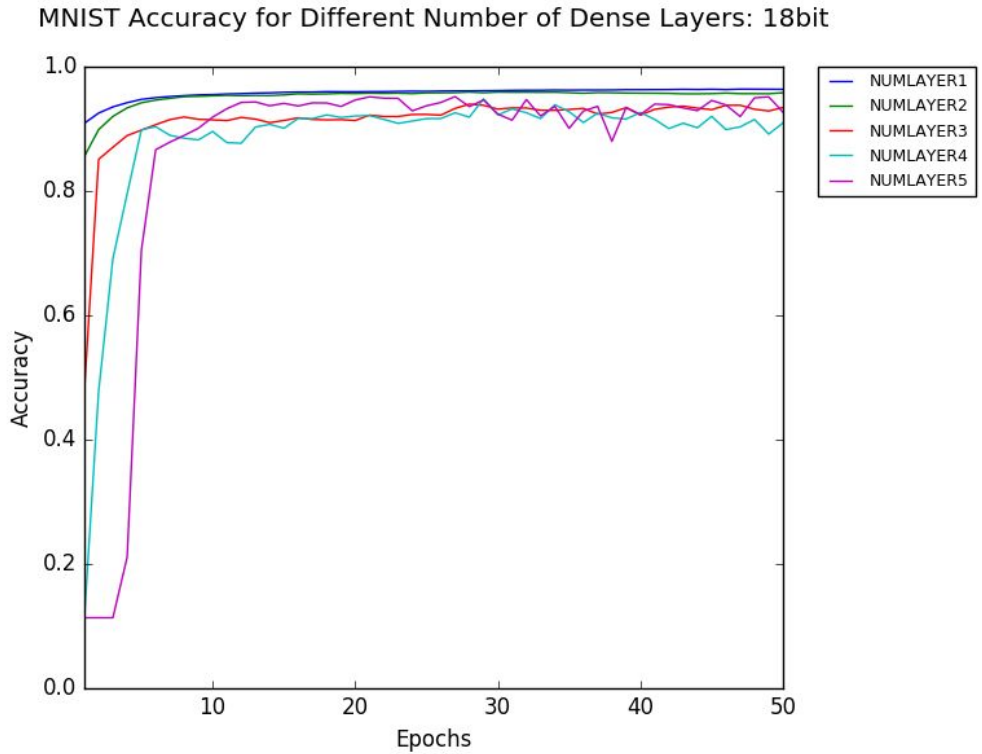
Number of Dense Layers

- Attempted one to five dense layers with 100 units per layer
- Difference between number of dense layers (at all bit size)
  - number of dense layers ↑, convergence speed ↓
  - number of dense layers ↑, final accuracy ↓
  - number of dense layers ↑, fluctuations during convergence ↑
    - May caused by accumulation of error

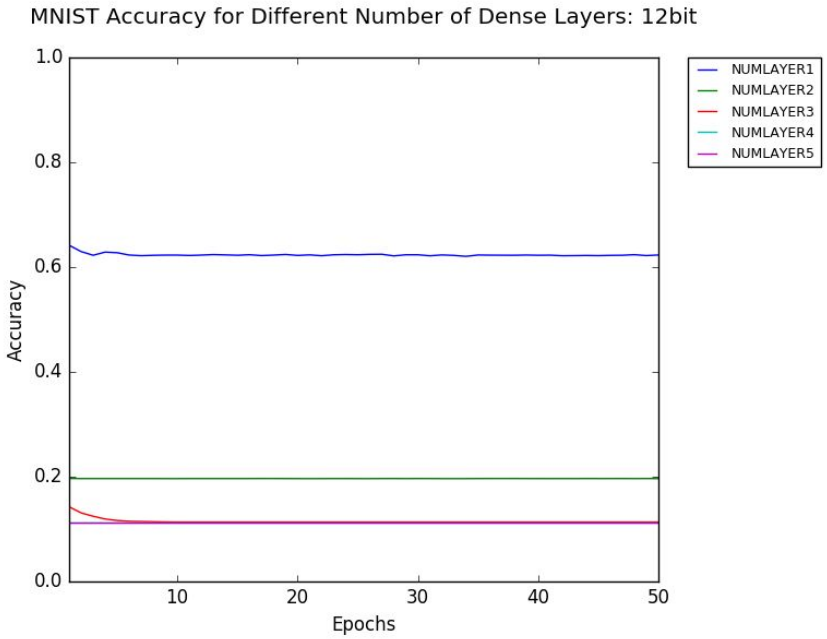
Number of Dense Layers: Results

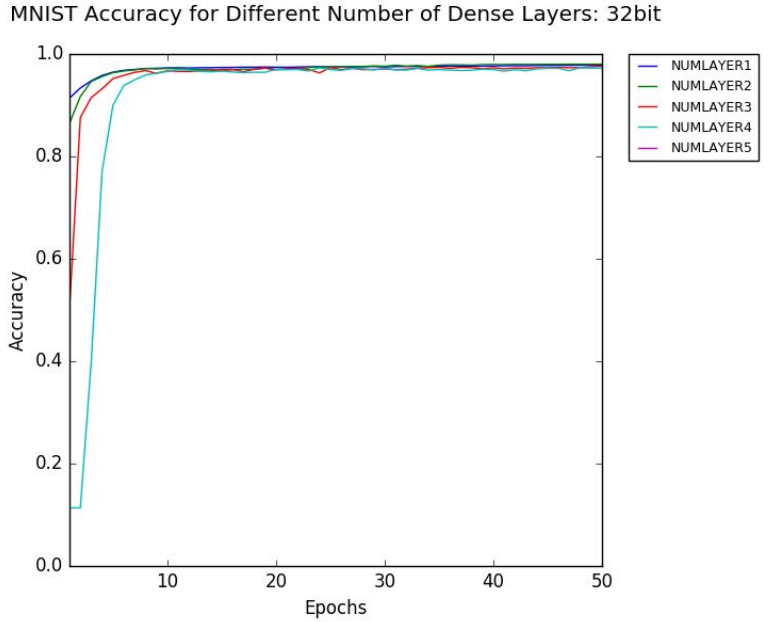


Number of Dense Layers: Results



Number of Dense Layers: Results



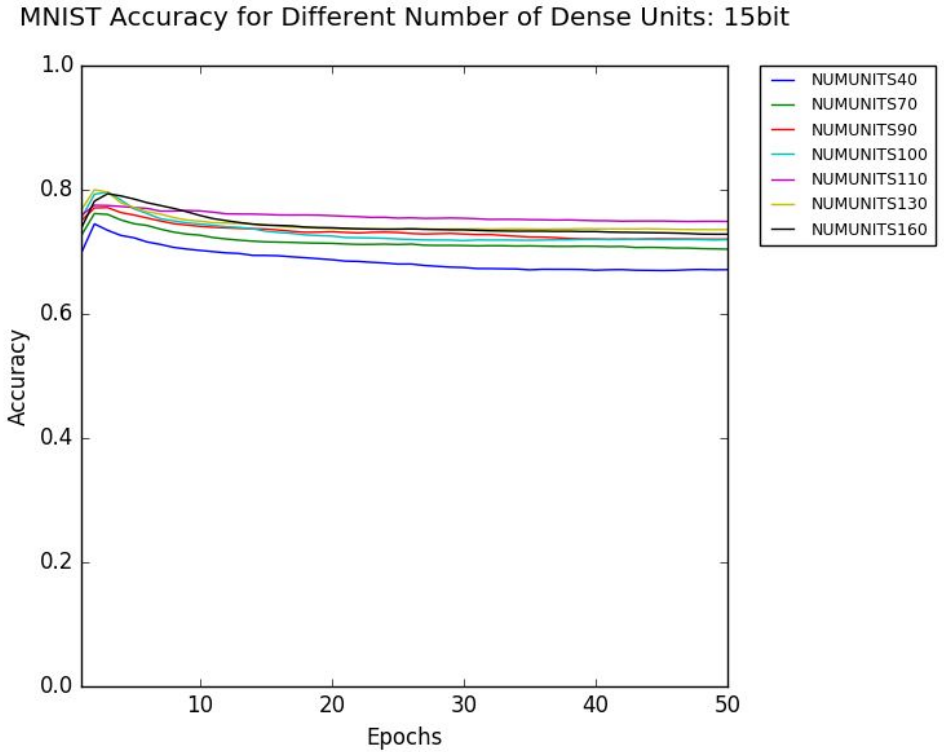


Low precision has significant influence when number of layers ↑

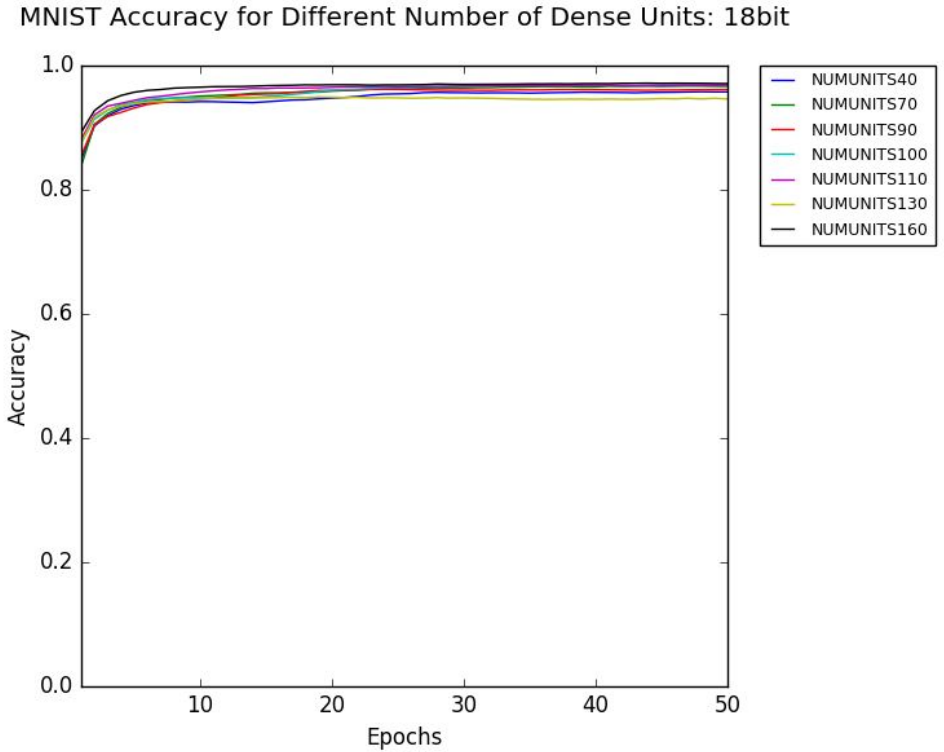
Number of Dense Units

- A two convolutional layer and one dense layer structure
- Tested 160, 130, 110, 100, 90, 70 and 40 units per dense layer
- Difference between number of dense units
  - In general, a random final accuracy distribution with respect to bit size
  - 16 bit: number of dense units ↑, final accuracy ↑

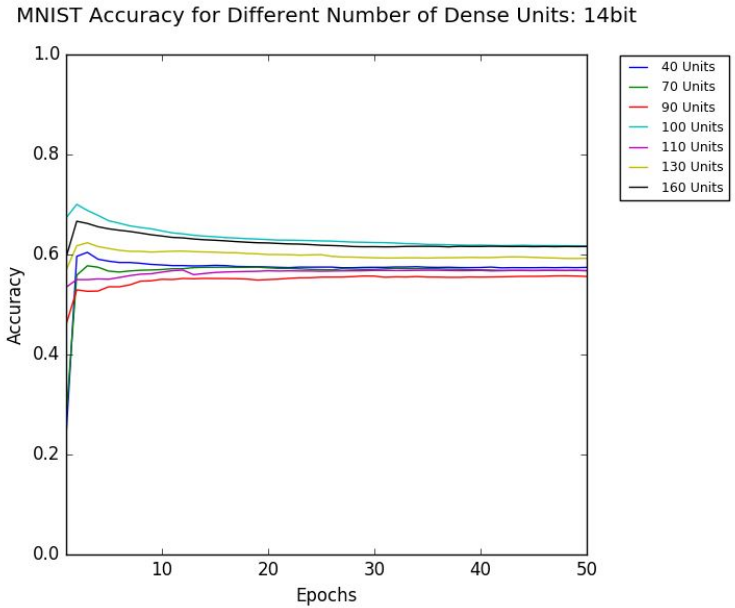
Number of Dense Units - Results

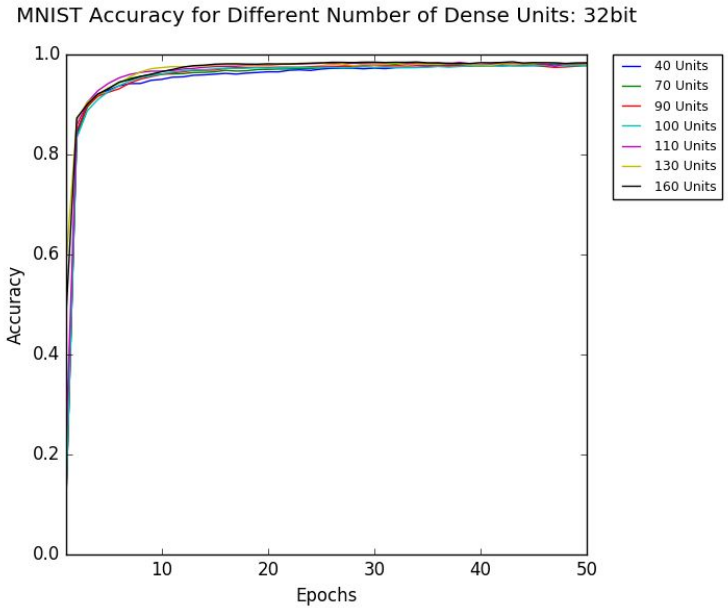


Number of Dense Units - Results



Number of Dense Units - Result Summary



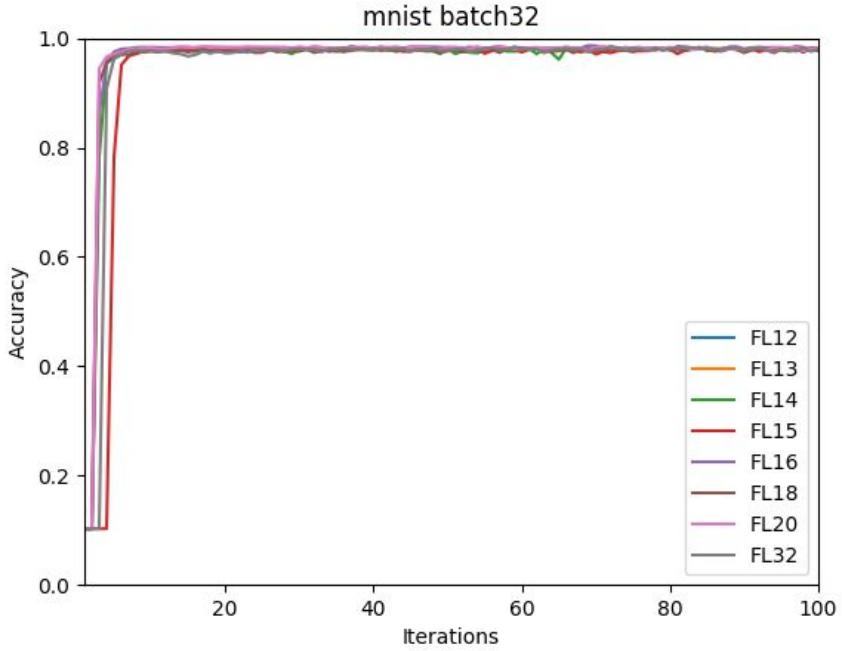


Optimum at low precision may not require high number of dense units

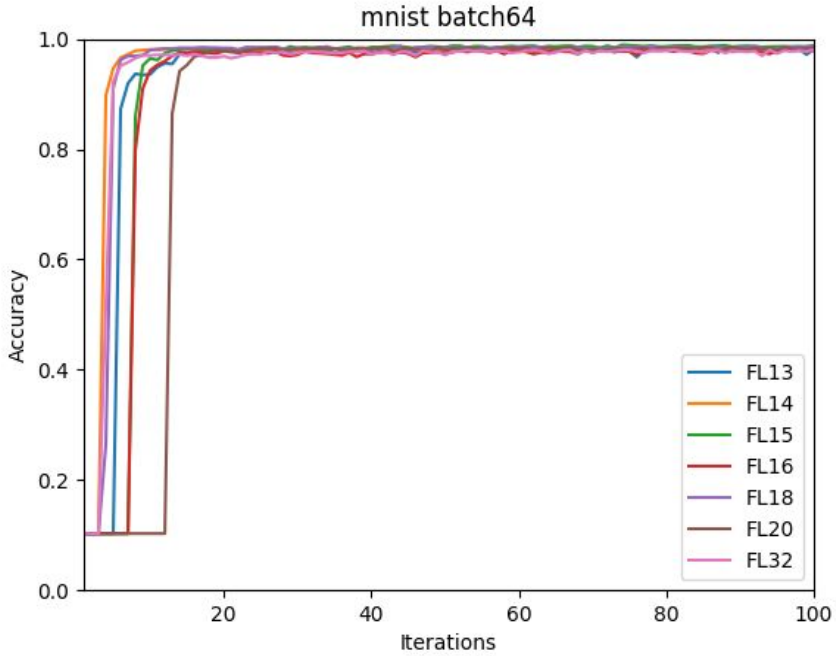
Batch Size: Introduction

- Number of samples to propagate through algorithm at a time (mini-batch)
  - 32, 64, 128, 256, 512
  - Requires less memory and typically reduces runtime vs. full batch
  - Eliminates random data variations vs. stochastic gradient descent
- Difference between batch sizes
  - Level of accuracy: unaffected
  - ↓ batch size, ↑ speed of convergence

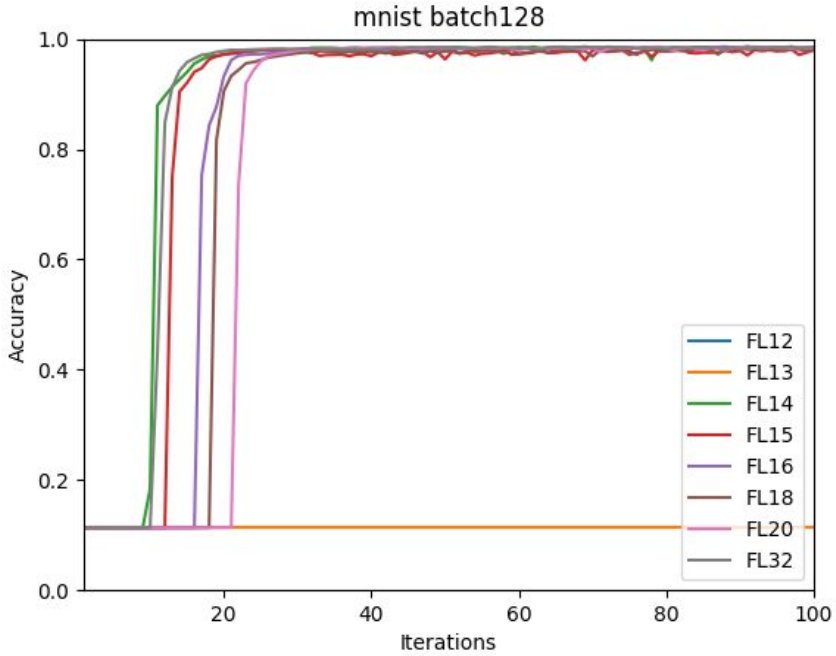
Batch Size: Results



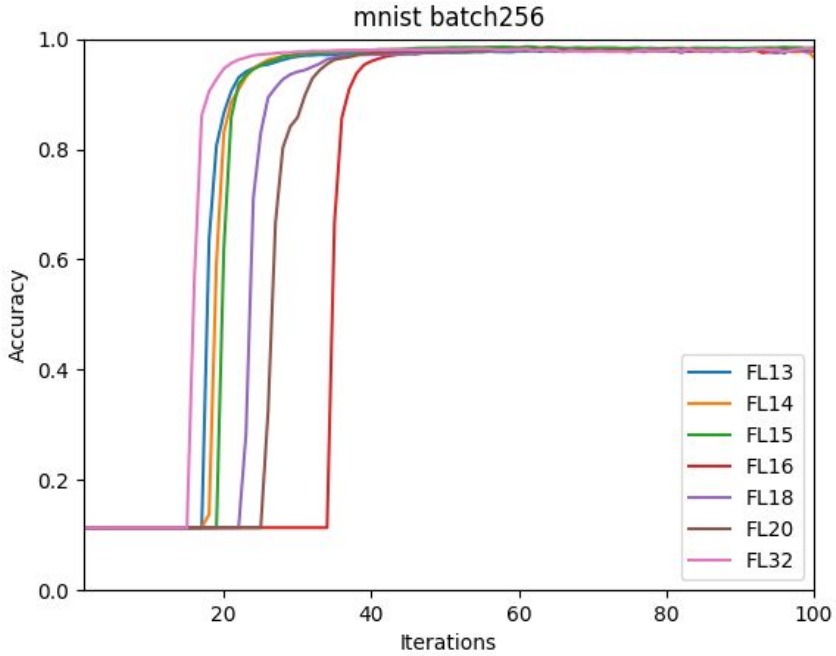
Batch Size: Results



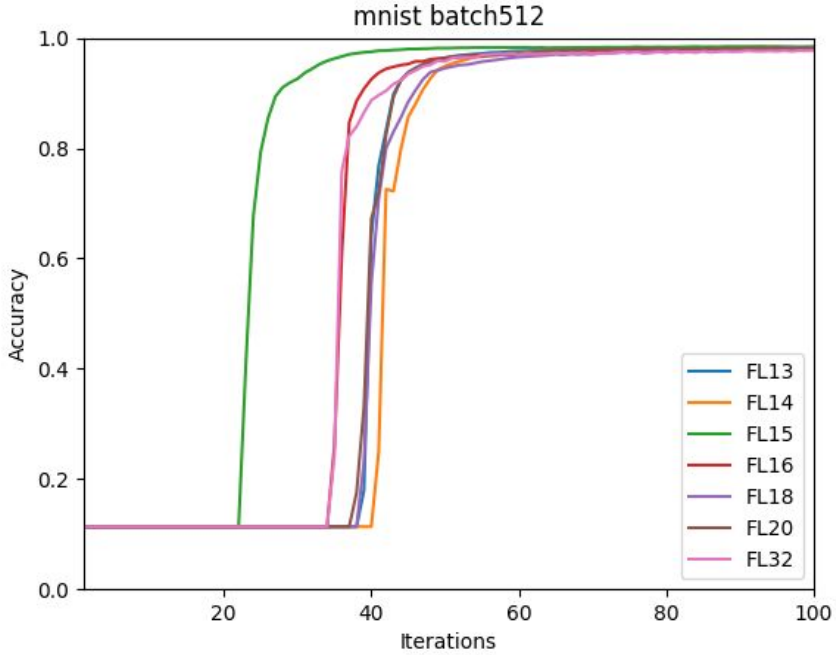
Batch Size: Results



Batch Size: Results



Batch Size: Results

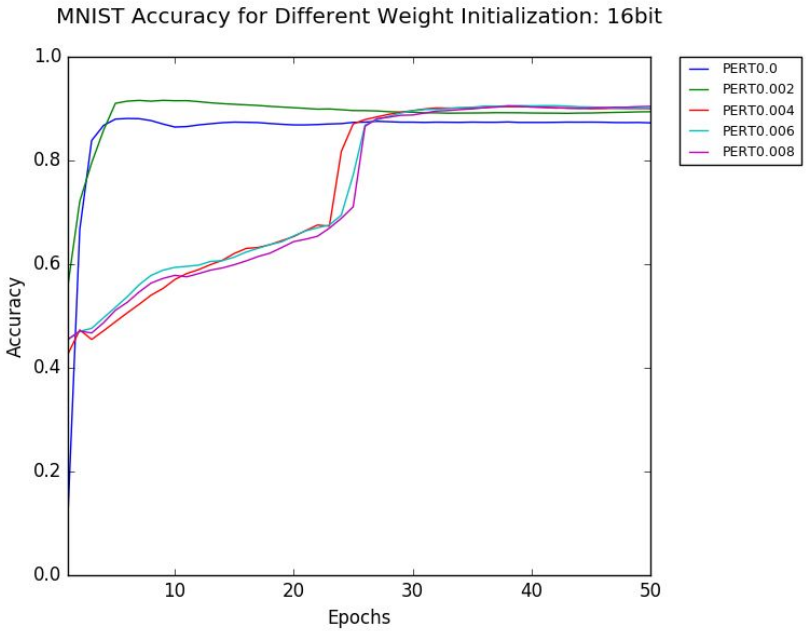


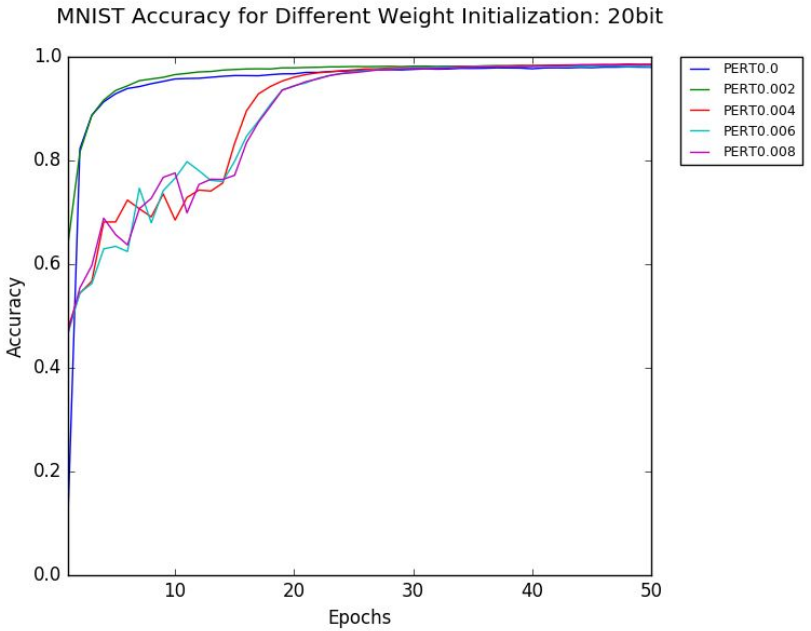
Convergence speed goes down as batch size increases

Perturbation to Initial Weights

- Weight initialization affects final accuracy (Stanford CS231n)
- Positive initial weights: 80%
- Random initial weights to break the symmetry
- Show that CNN is sensitive to input weights
- Reduced precision have a direct impact on CNN

Weight Initialization - Results





Sensitive to weight initialization in both accuracy and training time

Summary

- Reduced precision affects accuracy:
  - As number of layers goes up
  - As number of dense units changes
  - Affects both accuracy and training time as initial weight changes
- Reduced precision affects time, while not accuracy:
  - Stochastic rounding speeds up convergence by 25%
  - Larger batch sizes take longer to converge

Error Analysis

Recall the structure of convolutional neural network:

Input → Convolution → Pooling → ... → Output

Question: how does the error accumulate during forward and backward propagations?

- Traditional truncation method
- Rounding method

Error Analysis: Forward Propagation

Convolution: use a discretized version

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) W(m,n)$$

Assuming that $O(I) = O(W)$, equivalently,

$$\tilde{S}(i,j) = \sum_m \sum_n (I(i+m, j+n) - \varepsilon)(W(m,n) - \varepsilon)$$

Error Analysis: Forward Propagation

An error estimate for the forward propagation:

$$\tilde{S}_n(i,j) \approx S_n(i,j) - \left(\prod_{i=0}^{n} M_i N_i W_i\right) \cdot \left(\sum_{i=0}^{n} \frac{1}{W_i}\right) \varepsilon_w$$

⇒ Linear scaling of error with dimension of weight matrices

Error Analysis: Back Propagation

- Similar method to forward propagation
- Requires computing the gradients and using chain rule
- No regularization, squared error measure, Sigmoid activation function
- Last layer: scales with $O(\varepsilon^2)$ depending on output and weights
- Complicated to predict the dependency on previous layers

Error Analysis: Back Propagation

- Similar method to forward propagation
- Requires computing the gradients and using chain rule
- No regularization, squared error measure, Sigmoid activation function
- Last layer: scales with $O(\varepsilon^2)$ depending on output and weights
- Complicated to predict the dependency on previous layers
- Conclusion: Forward propagation is dominating the sensitivity

Conclusion

- Implement arbitrary precision and test on MNIST
- Explore sensitivity of convolutional neural networks
  - Accuracy is sensitive to: weight initialization, dense units, # of dense layers
  - Time is sensitive to: rounding scheme, batch size
- Demonstrate error analysis using truncation method

Ongoing Effort

- Truncation by layer
- Truncation by operation
- Verify conclusions on more advanced architectures
- More variations of error analysis

Acknowledgement

This research was carried out as part of the 2017 RIPS program at the University of California, Los Angeles, and was supported by NSF grant DMS-0931852.

The authors would like to thank Nicholas Malaya, Allen Rush, Hangjie Ji for their mentorship, support, and valuable advice. The authors would also like to thank Dimi Mavalski and Susana Serna for their help on organizing the RIPS program.

The authors thank AMD Company for their sponsorship and support.

Thank you for your attention!

Questions?

Zhaoqi Li: zli@macalester.edu

Yu Ma: midsummer@berkeley.edu

Catalina Vajiac: cvajiac01@saintmarys.edu

Yunkai Zhang: zhang.yunkai98@gmail.com

References

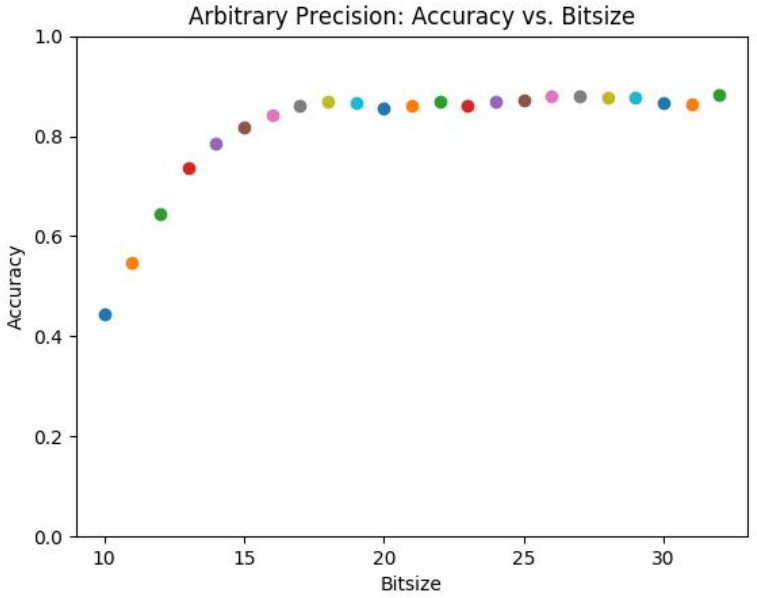
[1] Gupta, Suyog, et al. “Deep Learning with Limited Numerical Precision." ICML. 2015.

[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).

Mixed Precision

- Change precision after a certain number of iterations
  - Could happen multiple times per several epochs/batches
  - Current direction: low to high
- High precision to low precision

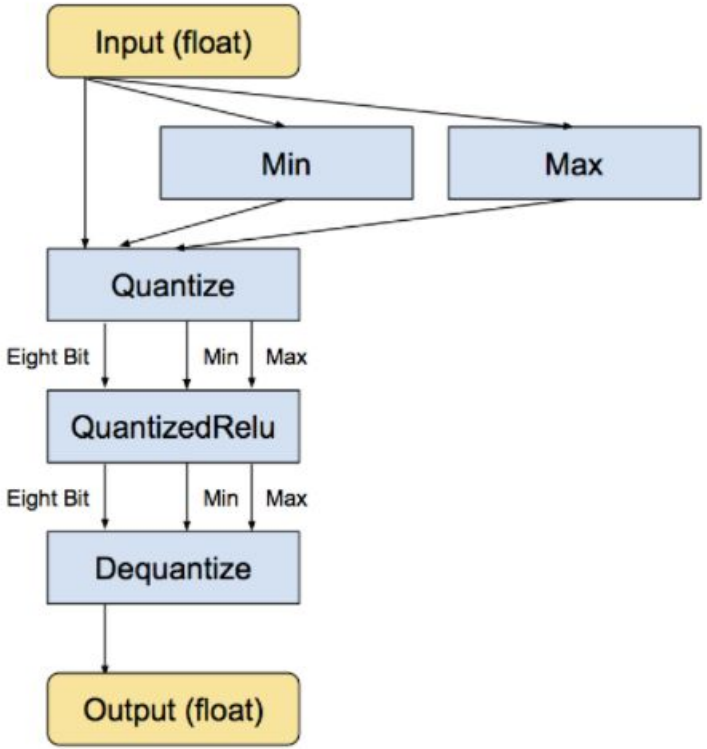


Number of Convolutional Layers: Conclusion

- One to five convolutional layers with one dense layer
- Controlled previous layers' dimensions when adding new layers
- Difference between number of conv. layers
  - One layer converges faster at all bit size
  - One layer has lower final accuracy at low bit size
  - Three and above do not converge at all for all bit size
- 18bit - the threshold for convergence

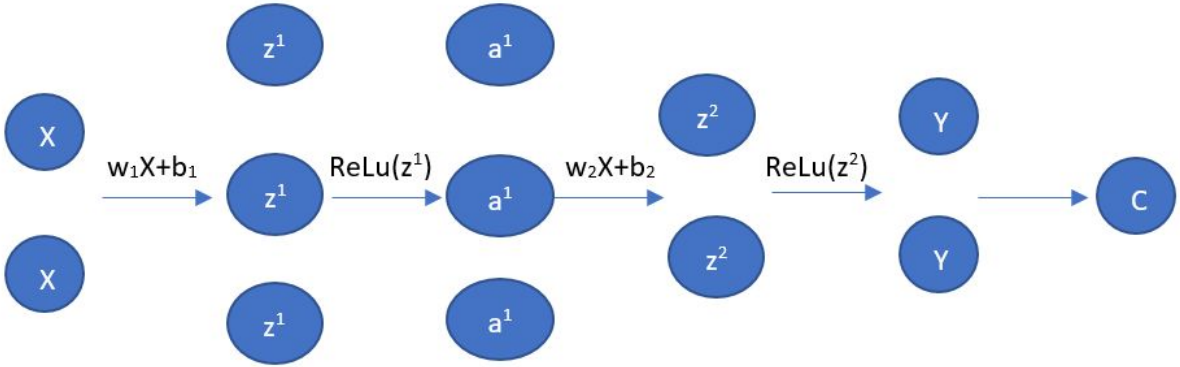
Quantization

- Theano and Lasagne based code by Matthieu Courbariaux
- Binary Net (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



4. Truncation by layer

- Cannot truncate directly -> one line does all numerical computations
- Alternative: Chain rule
  - Example: one hidden layer

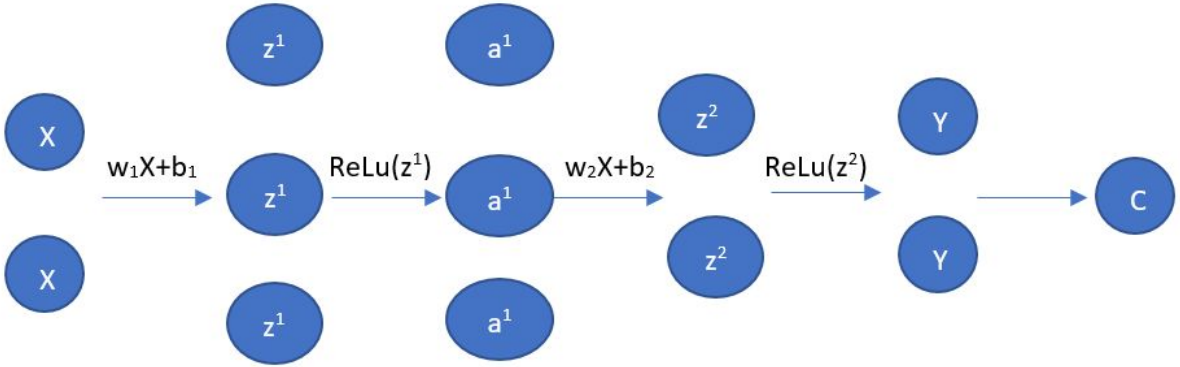


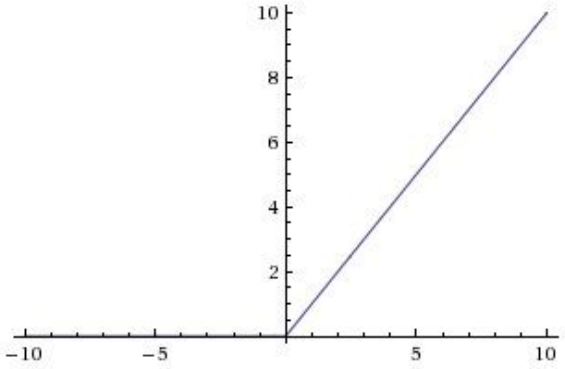
$$\frac{\delta C}{\delta z^2} = \frac{\delta C}{\delta a_2} \cdot \frac{\delta a_2}{\delta z^2}$$

$$\frac{\delta C}{\delta z^1} = (w_2^T \frac{\delta C}{\delta z^2}) \circ \frac{\delta a_1}{\delta z^1}$$

$$\frac{\delta C}{\delta w_1} = \frac{\delta C}{\delta z^1} a_1^T$$

$$\frac{\delta C}{\delta b_1} = \frac{\delta C}{\delta z^1}$$





ReLu

Cost = Squared Error

Values:

$$\frac{\partial C}{\partial z^2} = \frac{\partial (y-\hat{y})^2}{\partial y} \cdot \begin{cases} = 1 & \text{for } y > 0 \\ = 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial C}{\partial z^1} = (w_2^T \frac{\partial C}{\partial Z^2}) \circ \begin{cases} = 1 & \text{for } w_1 X + b_1 > 0 \\ = 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial z^1} a_1^T$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial z^1}$$

5. Truncation by operation

- Implementation:
  - Hard to be done with Theano: too many basic operations within theano.function
  - Use sympy to represent equations in tree structure
  - Recursively iterate through tree and truncate after each basic operation

Notation

- Representation error: ε
- $\tilde{x}$: the approximation of x. $x = \tilde{x} + \varepsilon_x$.
- I: input data
- W: weight matrix
- S: output

Error Analysis: Forward Propagation

$$\theta(I) = \theta(W) \implies \varepsilon_I \cong \varepsilon_W$$

Assuming that weight matrix W_i has dimension $M_i \times N_i$,

$$\begin{aligned}\tilde{S_1}(i,j) &\approx S_1(i,j) - M_1 N_1 (I\varepsilon_w + W\varepsilon_I) + M_1 N_1 \varepsilon_w \varepsilon_I \\ &\approx S_1(i,j) - M_1 N_1 (I + W_1)\varepsilon_w\end{aligned}$$

Error Analysis: Forward Propagation

Similarly, for the second layer,

$$\begin{aligned}\tilde{S}_2(i,j) &= \sum_m \sum_n \tilde{S}_1(i+m, j+n)\tilde{W}_2(m,n) \\ &= S_2(i,j) + \sum_m \sum_n \big(- \varepsilon_w S_1(i+m, j+n) \\ &\qquad - M_1 N_1 (I + W_1) W_2 \varepsilon_w + M_1 N_1 (I + W_1) \varepsilon_w^2 \big)\end{aligned}$$

Approximating S_1 as $S_1(i+m, j+n) \approx M_1 N_1 I W_1$,

$$\tilde{S}_2(i,j) \approx S_2(i,j) - M_1 N_1 M_2 N_2 (I W_1 + I W_2 + W_1 W_2) \varepsilon_w$$

More variations of error analysis

- Stochastic rounding instead of truncation
- Error term will be a distribution
- Truncation by layer/batch instead of by elementary operation
- To validate our current conclusions
- More than one error terms

Previous Work: Google TPU

- Quantization: use an 8-bit integer to approximate an arbitrary value between preset min & max values
- Pros:
  - Reduces hardware footprint and energy consumption
  - TPUs: 65,536 8-bit integer multipliers
  - GPUs: few thousands of 32-bit floating-point multipliers
- Cons:
  - Some networks are too sensitive for this, can diminish accuracy