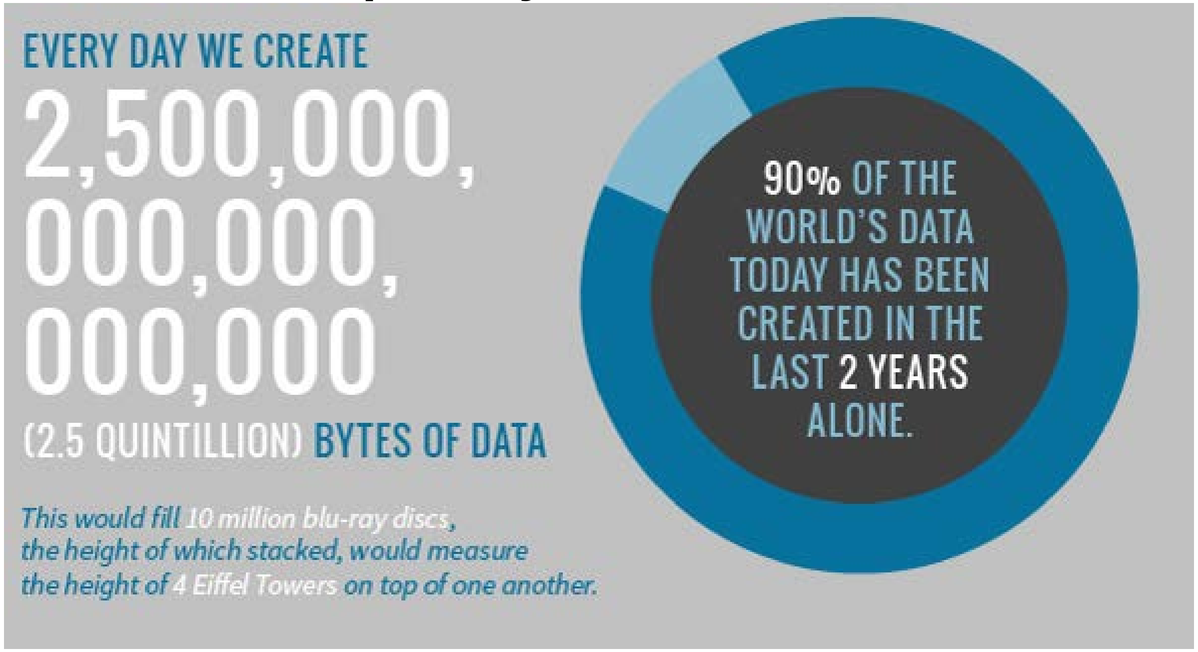
CANNA: Neural Network Acceleration using Configurable Approximation on GPGPU

Mohsen Imani, Max Masich, **Daniel Peroni**, Pushen Wang, Tajana S. Rosing

University of California San Diego

Motivation

- **Billions** of interconnected devices (large scale data problem)
- Amount of data generation in 2015 was **8 zettabytes** and is exponentially growing
- Impractical to send all data to cloud
- Processing data (at least partially) locally is **scalable**, allows **real-time** response, and ensures **privacy**

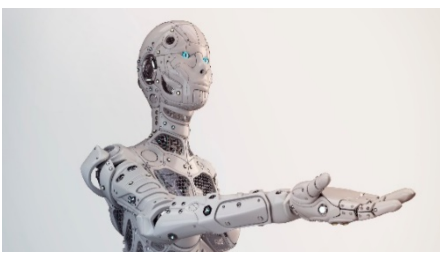


Machine Learning is Changing Our Life

Self Driving Cars

Healthcare

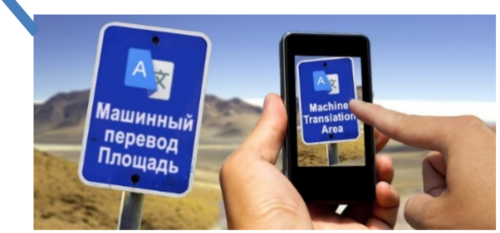
Smart Robots



Finance

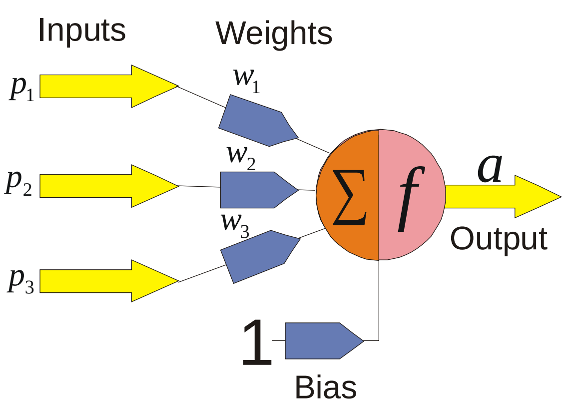
Gaming

Machine Translation

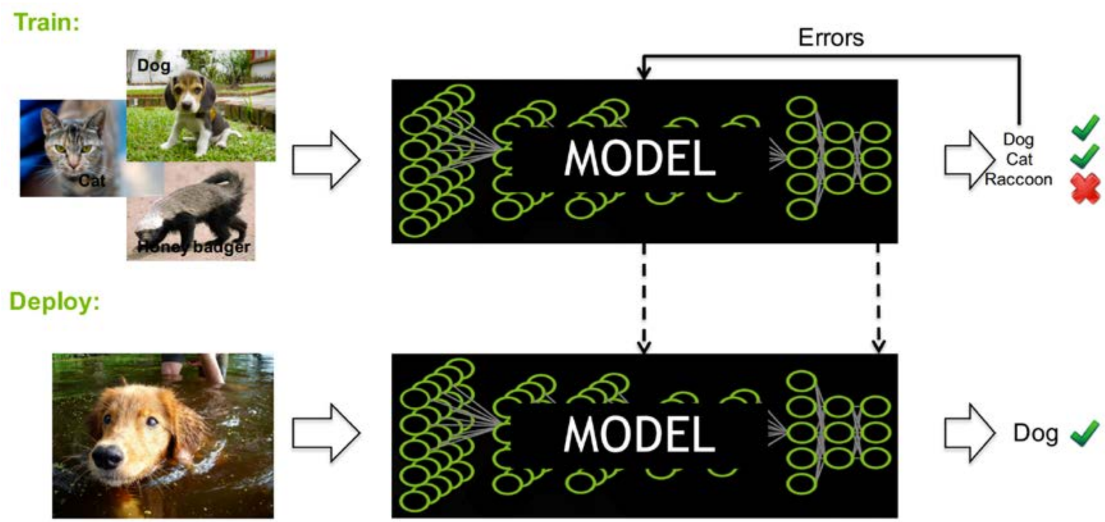


Neural Networks

- **Neuron:**
  - A processing unit which takes one or more inputs and produces an output
  - Each input has an associated **weight** which modifies its strength
  - Neuron simply adds together all the inputs and calculates an output to be passed on
- **Training:**
  - Based on the error in training phase, update the weights using gradient descent in back propagation



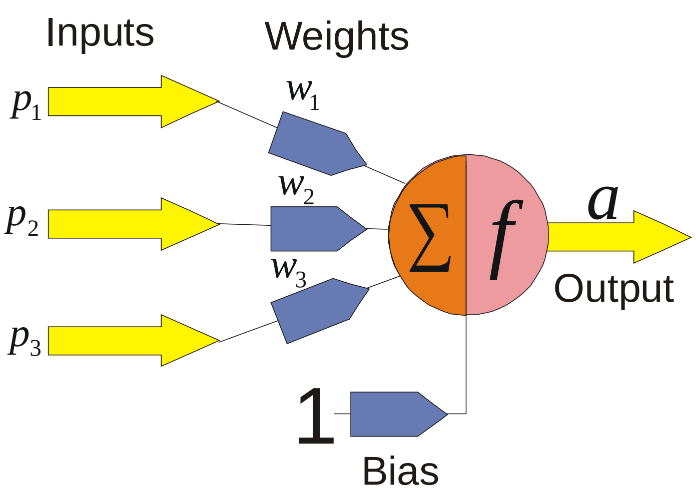
$$a = f(p_1 w_1 + p_2 w_2 + p_3 w_3 + b) = f\left(\sum p_i w_i + b\right)$$



http://on-demand.gputechconf.com/gtc/2015/webinar/deep-learning-course/intro-to-deep-learning.pdf

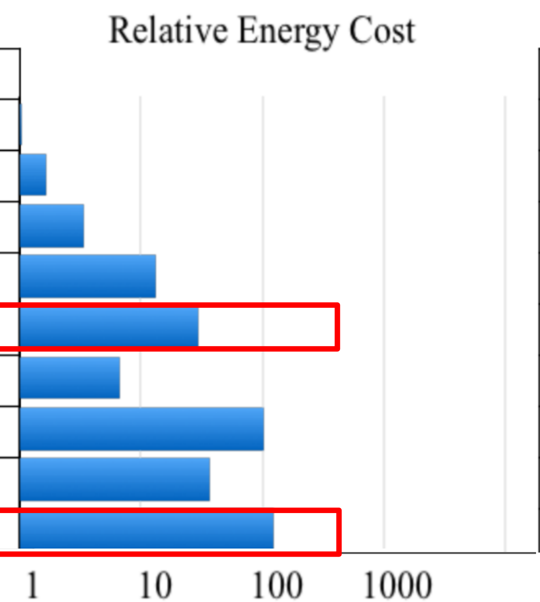
Cost of Floating Point Multiplication

- Floating point multiplication consumes majority of computational power in NN
- A 32-bit FPU multiply is 4x more costly than an FPU add [Horowitz 2014]



45nm CMOS Technology

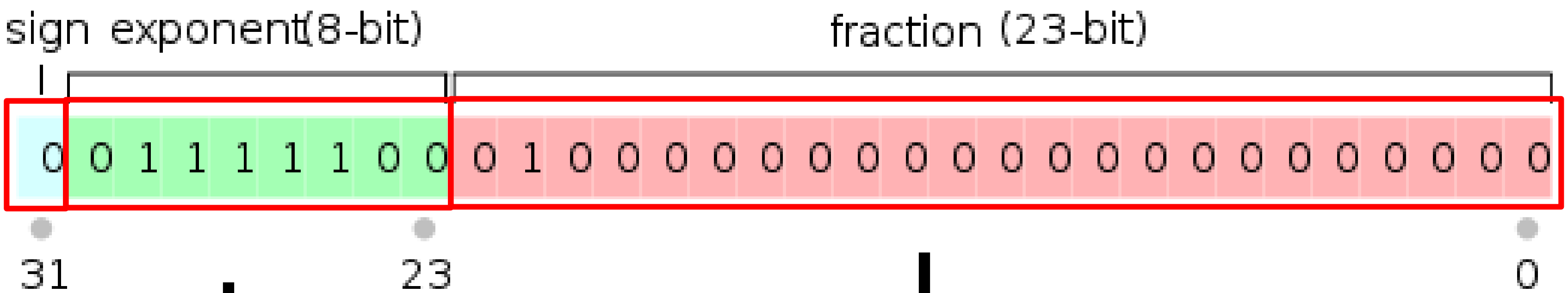
Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7



Related Work

- Approximate NN parameters:
  - Implementing fixed-point quantized numbers improves performance [Lin 2015]
  - Binarized weights can be used to avoid multiplication [Lin 2015]
  - More accurate results require higher precision and these approaches have difficulties with additive quantization noise
- Approximate Multipliers:
  - Reduced bit multiplication [Hashemi 2015]
  - Approximate multiplier designed from approximate adders [Liu 2014]
  - Configurable floating point multiplier [Imani 2017]

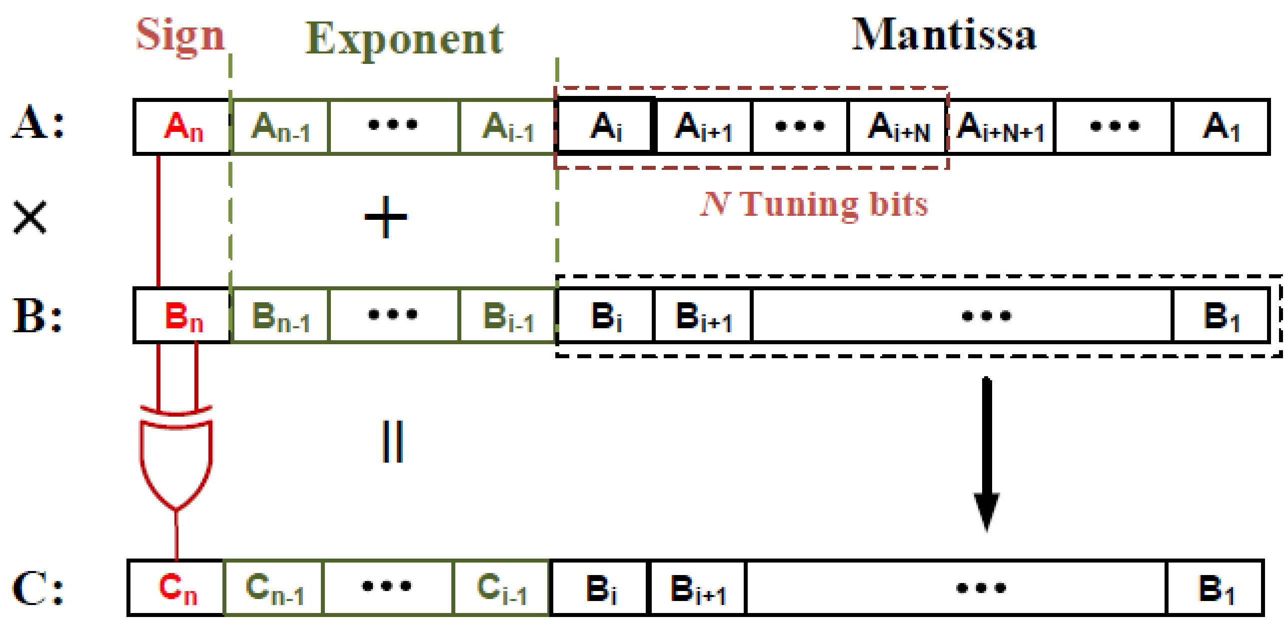
IEEE 754 Floating Point Values



$+ \quad 2^{-3} = 0.125 \quad x \quad 1.25 \quad = \quad 0.15625$

- Floating point values has 3 components:
  - Sign bit – Determines positive or negative value
  - Exponent – 2^x where $\boldsymbol{x}$ ranges from -127 to 128
  - Fraction (Mantissa) – Ranges from 1 to 2
- Multiply all three together to get decimal value

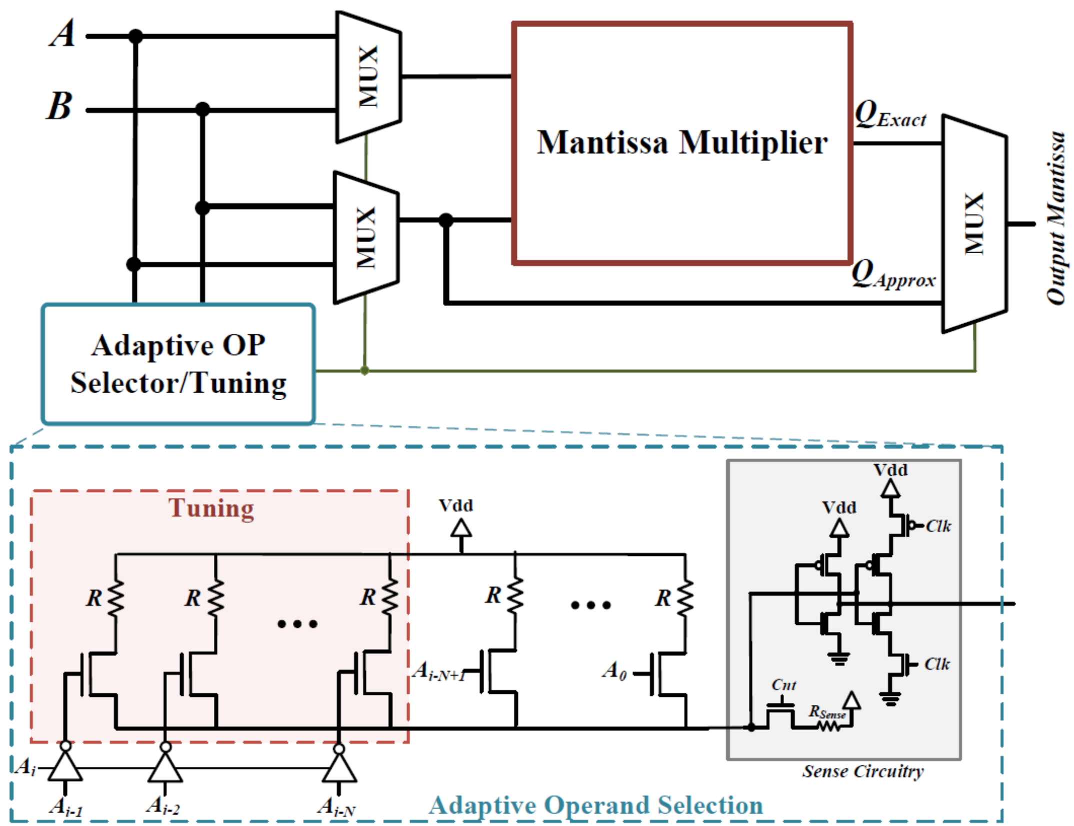
Configurable Floating Point Unit (CFPU)



- Modify FPU to approximate multiplication
- Copy mantissa from one input and discard the other
- Tuning allows FPU to run in either exact or approximate mode
  - Only 2.7% energy overhead compared to unmodified FPU

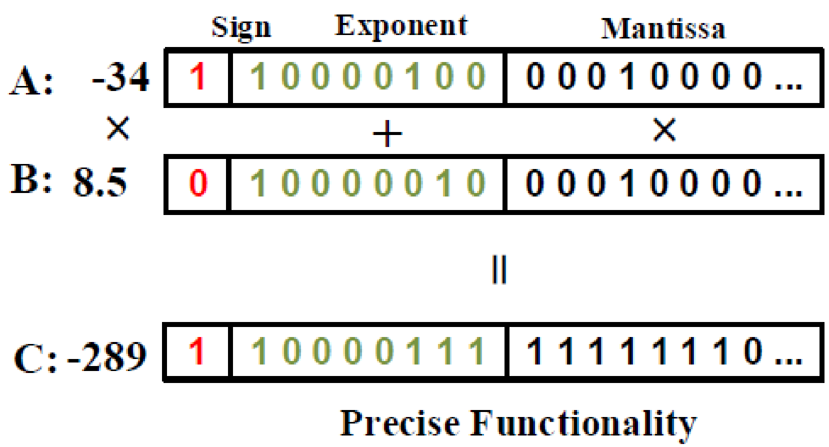
Configurable Floating Point Unit (CFPU)

- Adaptive operand selection to detect which mantissa to discard
- Mantissa closest to 0x000… or 0x111… will result in lowest error
- If predicted error is too large based on tuning bits, run exact mantissa multiply



$$Error = \left| \sum_{i=N}^{n-1} 2^{-((n-i)A_{n-i-1})} - 0.5 \right|$$

Example Approximate Multiply



- Copy mantissa from B
- Check n-tuning bits
  - 3-bits match, error must be less than 6.25%
- Result: -272 (5.9% error)

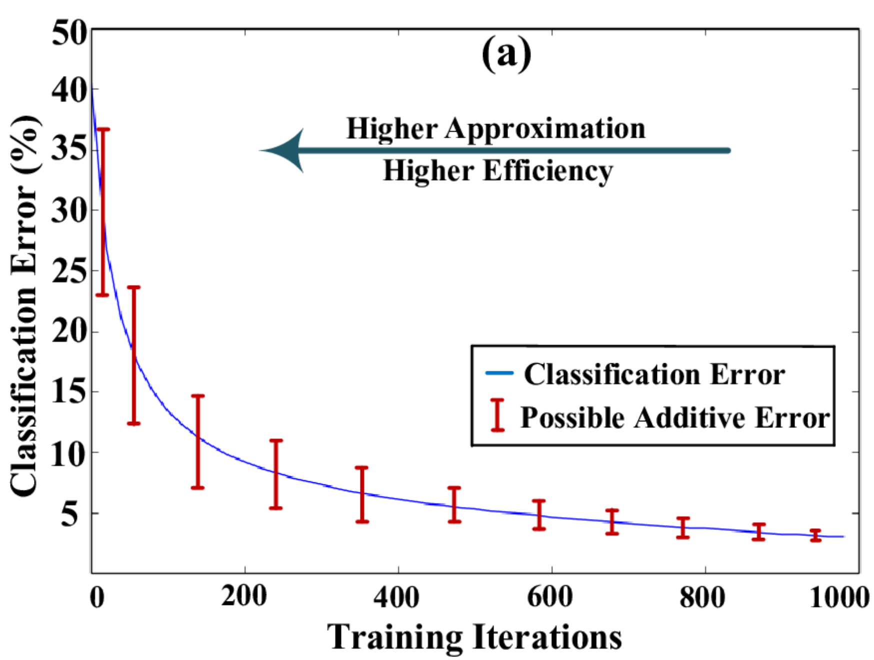
CANNA - Configurable Approximation for Neural Network Acceleration

- **Training phase**: Use less accurate computation and gradually increase the accuracy as the solution converges

- **Inference phase**: Use approximate hardware for calculations, but adjust accuracy based on sensitivity of layers

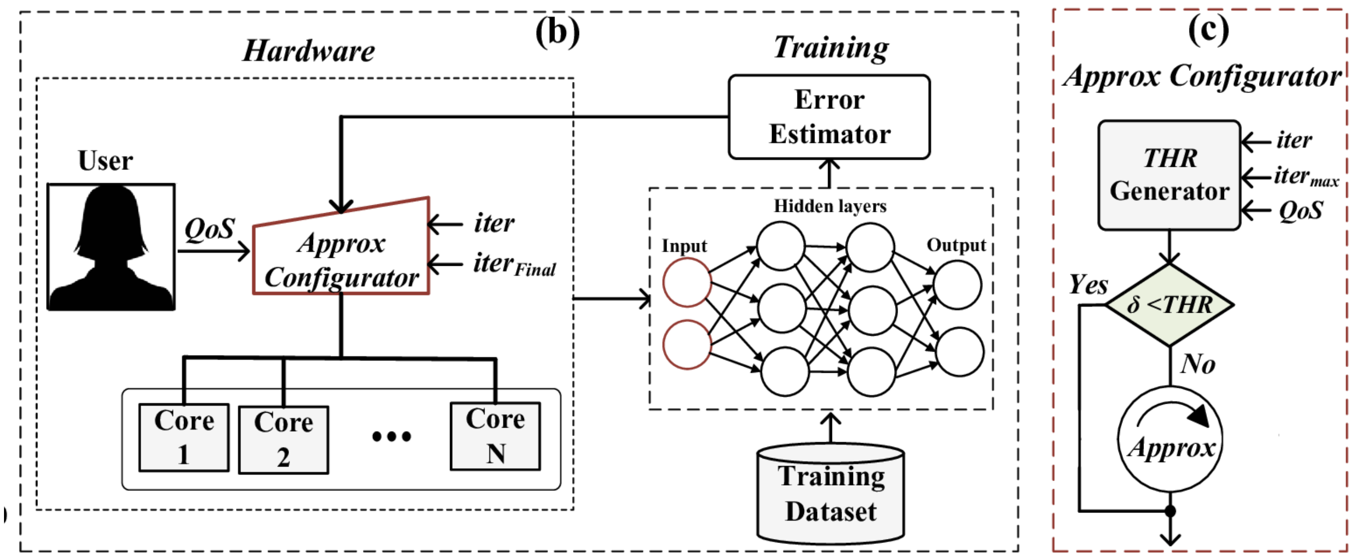
Gradual Training Approximation (GTA)

- Initialize weights to random values
- Start training with high levels of approximation
- Reduce the level of approximation as a function of NN error until the desired accuracy is reached



Gradual Training Approximation (GTA)

- Use combination of current iteration and desired accuracy to generate threshold (THR) for current iteration
- Apply THR to CFPU to adjust accuracy



Layer Based Inference Approximation

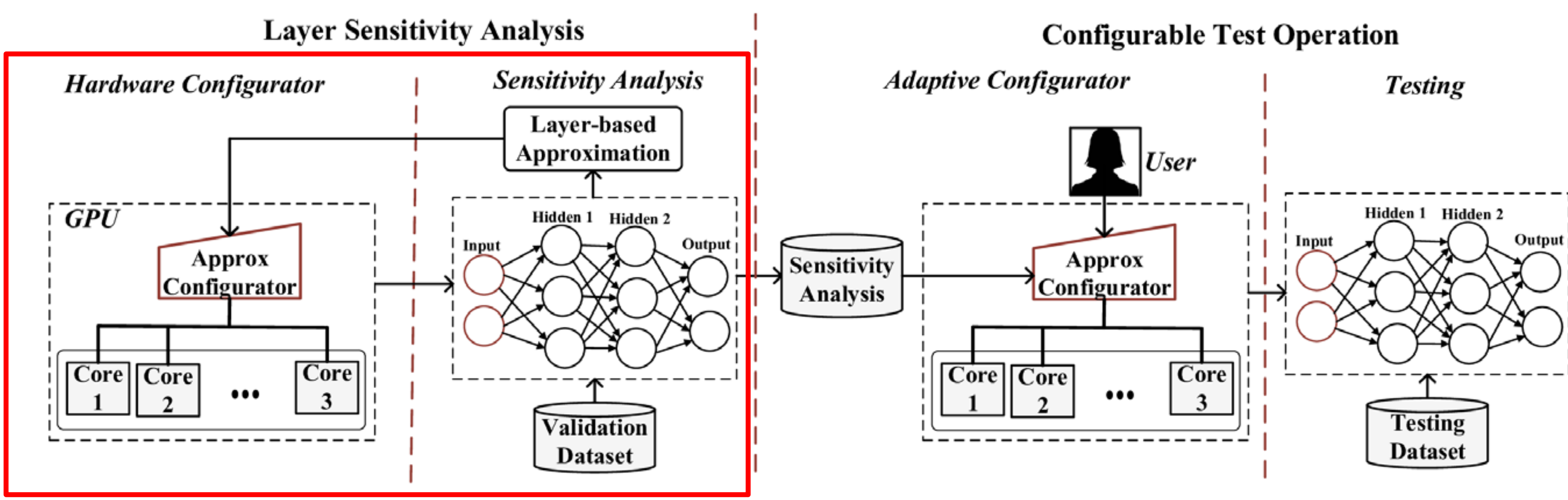
- Each layer has a different impact on output error
- Approximation is reduced for layers with higher sensitivity and vice versa
- Tested data shows first and last layers are the most sensitive

Relative Sensitivity of Each Layer

	Layer 1	Layer 2	Layer 3	Layer 4
MNIST	1	0.52	0.34	0.89
ISOLET	1	0.57	0.65	0.83
HYPER	0.92	0.72	0.59	1
HAR	1	0.48	0.41	0.92

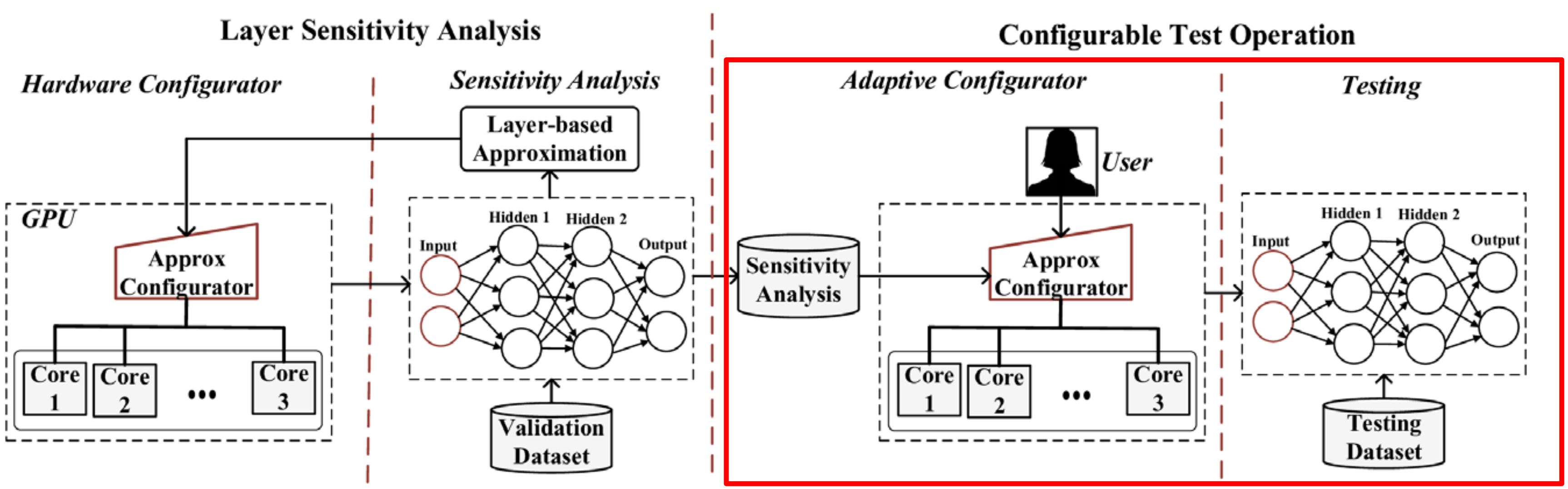
Layer Based Inference Approximation

- Run validation dataset on NN with no approximation to generate baseline accuracy
- Run validation dataset while adjusting approximation in each layer
- Determine the sensitivity of each layer



Layer Based Inference Approximation

- Based on sensitivity values select level of approximation for each layer
- Goal: minimize error while ensuring maximum approximation for each layer



Experimental Setup

- Multi2sim for architectural simulation
  - AMD Southern Island Architecture
  - Cycle accurate simulator

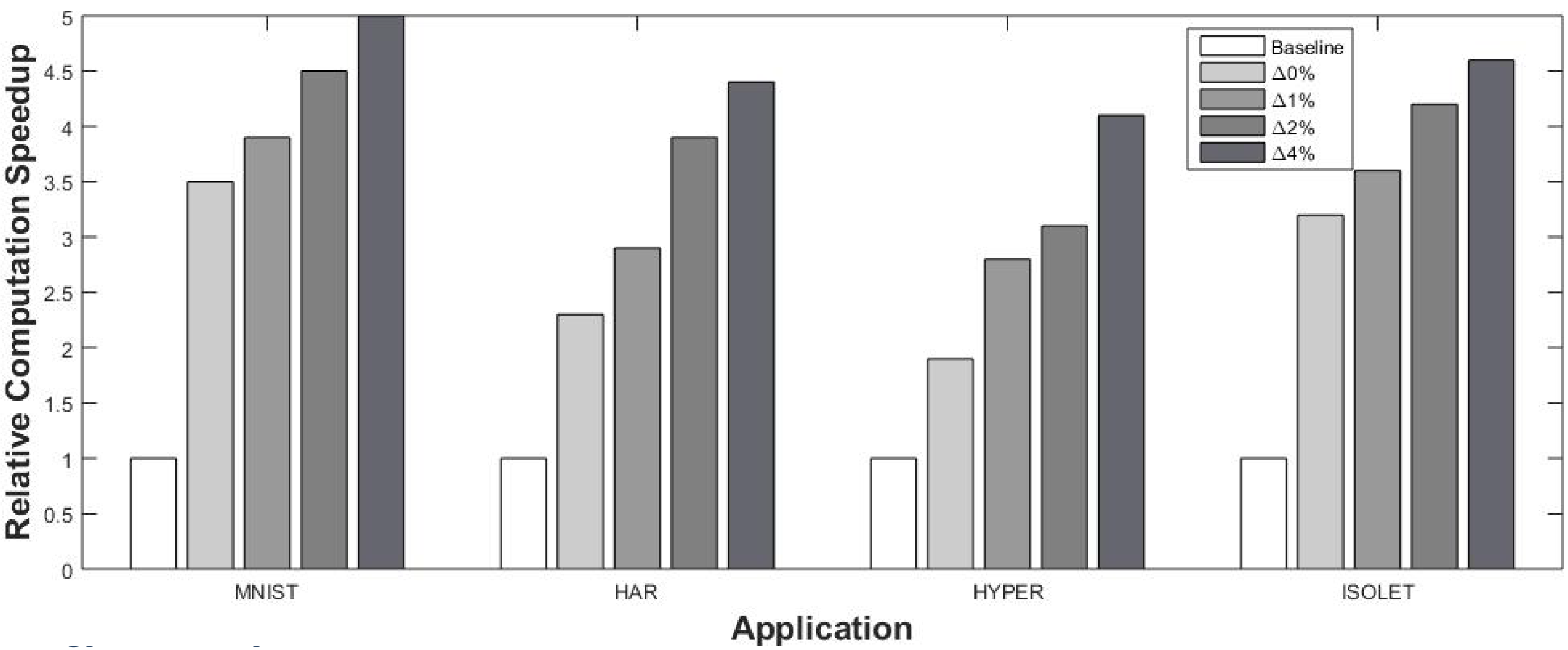
Model	Radeon 7970
Compute Units	32
Shading Units	2048
Memory	3072 MB

- Test Applications
  - MNIST – Handwritten image recognition
  - ISOLET – Voice recognition
  - HYPER – Hyperspectral imaging
  - HAR – Human activity recognition
- Power/Performance Measurement
  - McPAT for power estimation
- RCA Circuit Level
  - Transistor-level HSPICE simulation for power and delay in 45nm

Gradual Training Approximation - Speedup

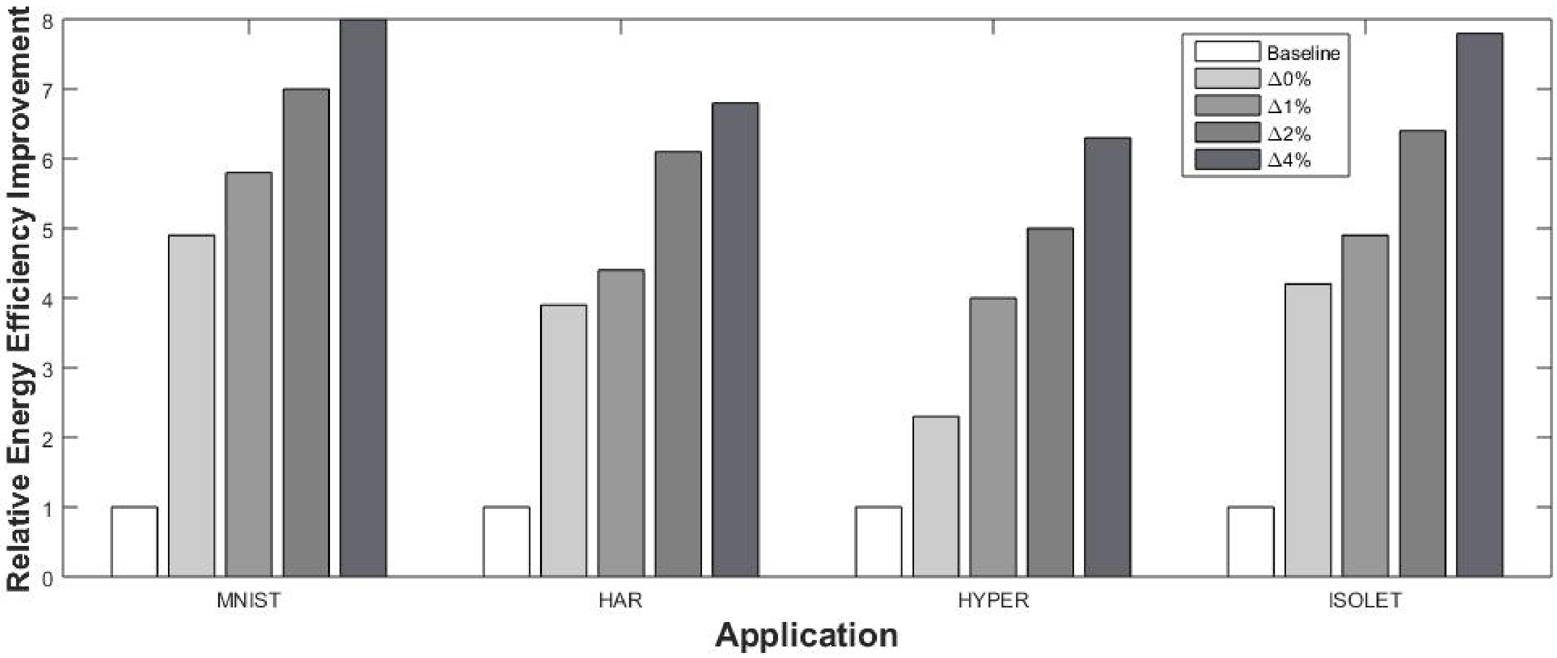
- Baseline is neural network running on exact hardware
- GTA achieves 3.8x average speedup with 1% additive error

Application	Network Topology (l^0, l^1, l^2, l^3)	$e_{test}(\%)$
MNIST	784, 500, 500, 10	2.4
ISOLET	617, 500, 500, 26	4.4
HYPER	200, 500, 500, 9	6.6
HAR	561, 500, 500, 12	3.4



Gradual Training Approximation - Energy

- Results are compared to an unmodified AMD GPU
  - Δe is the difference between the output error of the exact NN and the approximately trained NN
- GTA achieves 4.8x average energy improvement at 1% additive error



Layer based approximation - Sensitivity

- Outermost layers are the most sensitive to approximation
- Inner layers can allow 1.5-3x more approximation error
- Need to round sensitivity to due to approx granularity

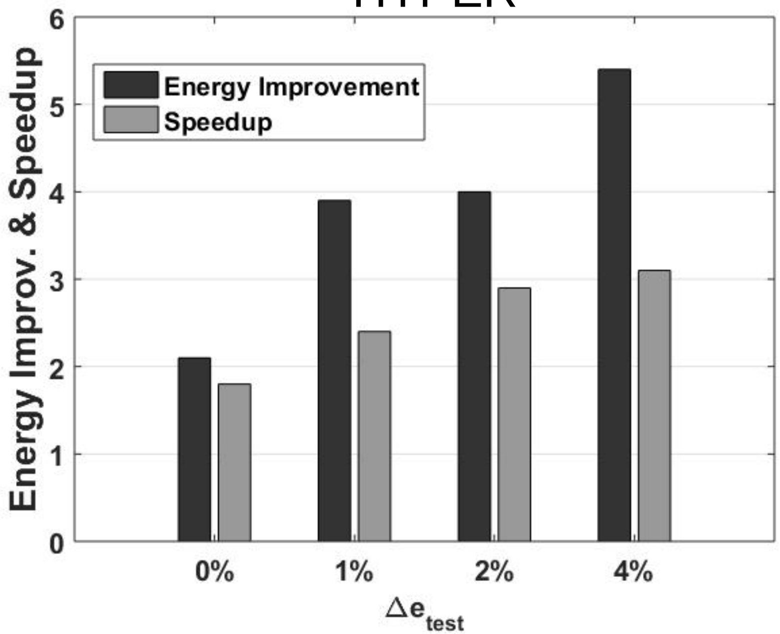
Relative Sensitivity of Each Layer

	Layer 1	Layer 2	Layer 3	Layer 4
MNIST	1	0.52	0.34	0.89
ISOLET	1	0.57	0.65	0.83
HYPER	0.92	0.72	0.59	1
HAR	1	0.48	0.41	0.92

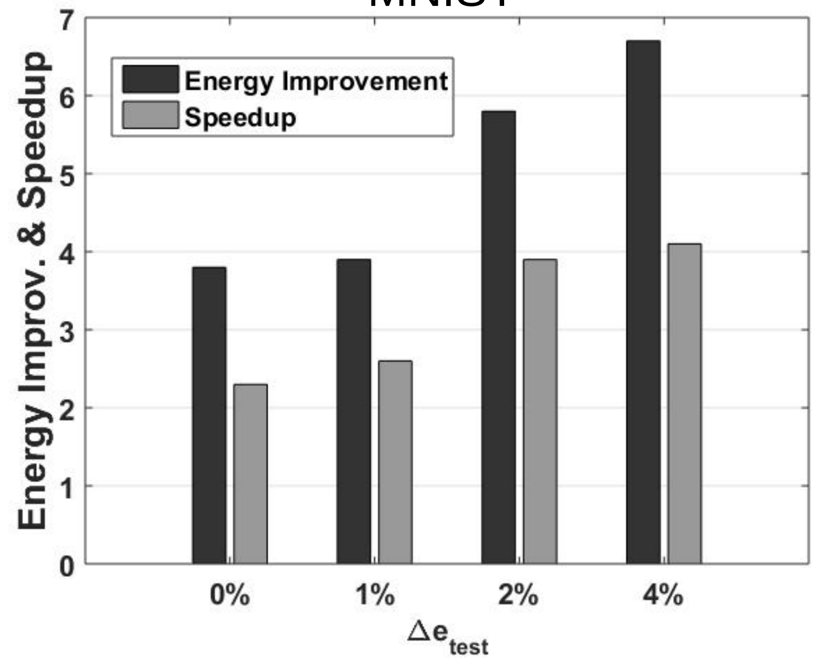
Layer based approximation - Performance

- Layer based inference achieves 3.6x energy savings and 2.3x speedup for 1% additive error

HYPER



MNIST



* Δe is the difference between the output error of the exact NN and the approximately trained NN

Conclusion

- Neural networks can benefit greatly from approximation
- A configurable approximate multiplier allows the level of approximation to be adjusted
- Gradual Training Approximation:
  - During training gradually increasing accuracy results in better energy and performance than using a uniform approach
- Layer-based Inference Approximation:
  - During inference approximation of individual layers can be adjusted to optimize performance

- Training – 4.8x energy savings and 3.8x speedup*
- Inference – 3.6x energy savings and 2.3x speedup*

*With 1% added error

Questions