Exploiting Hidden Layer Modular Redundancy for Fault-Tolerance in Neural Network Accelerators

Schuyler Eldridge Ajay Joshi

Department of Electrical and Computer Engineering, Boston University
schuye@bu.edu

January 30, 2015

This work was supported by a

NASA Office of the Chief Technologist's Space Technology Research Fellowship.

Motivation

Leveraging CMOS Scaling for Improved Performance is Becoming Increasingly *Hard*

- Contributing factors making it difficult include:
  - Fixed power budgets
  - An eventual slowdown of Moore's Law
- Computer engineers increasingly turn towards *alternative* designs

Alternative Designs

- As an alternative, others are investigating general and special purpose accelerators
- One actively researched accelerator architecture is that of *neural network accelerators*

Artificial Neural Networks

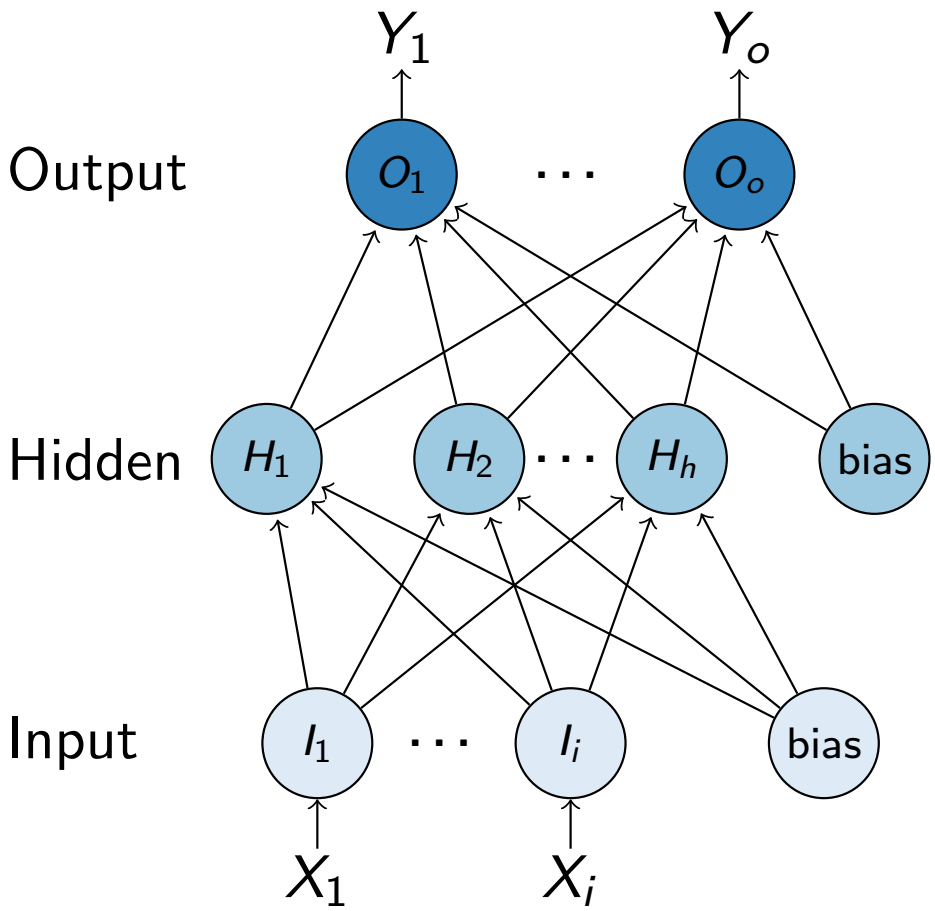

Figure: Two-layer neural network with $i \times h \times o$ nodes.

Artificial Neural Network

- Directed graph of neurons
- Edges between neurons are weighted

Use in Applications

- Machine Learning
- Big Data
- Approximate Computing
- State Prediction

Neural Networks and Fault-Tolerance

The Brain is Fault-Tolerant!

- Ergo neural networks are fault-tolerant
- **This isn't generally the case!**

Do Neural Networks have the *potential* for Fault-Tolerance?

- Neural networks have a redundant structure
  - There are multiple paths from input to output
- Regression tasks often approximate smooth functions
  - Small changes in inputs or internal computations may only cause small changes in the output
- However, **there is no implicit guarantee of fault-tolerance** unless you train a neural network to specifically demonstrate those properties

N-MR Technique

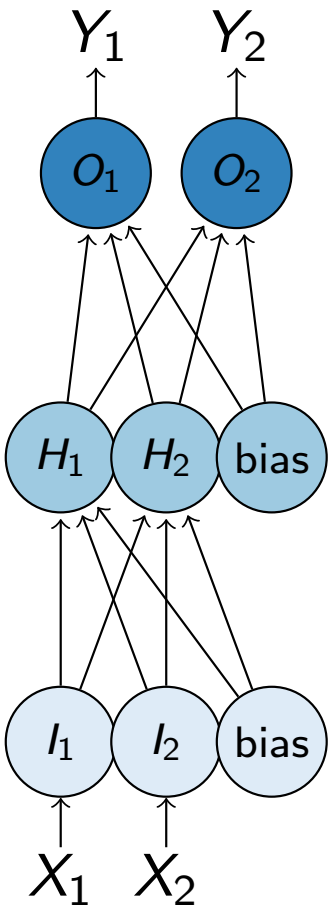

Figure: N-MR-1

Steps for Amount of Redundancy N

1. Replicate each hidden neuron N times
2. Replicate each hidden neuron connection for each new neuron
3. Multiply all connection weights by $1/N$

N-MR Technique

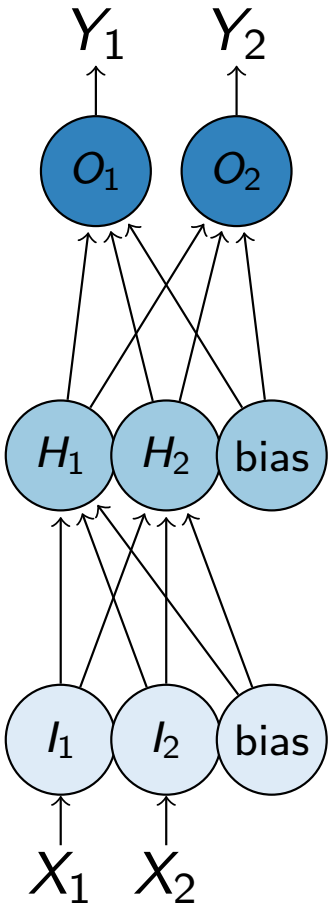

Figure: N-MR-1

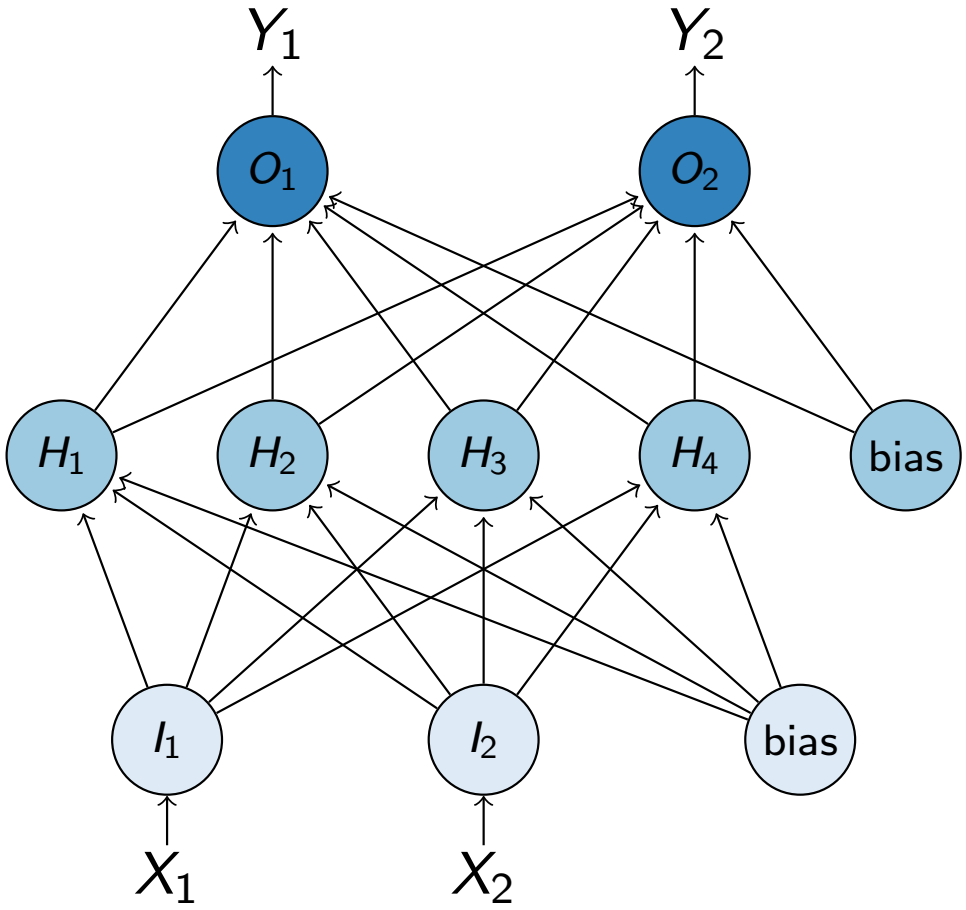

Figure: N-MR-2

N-MR Technique

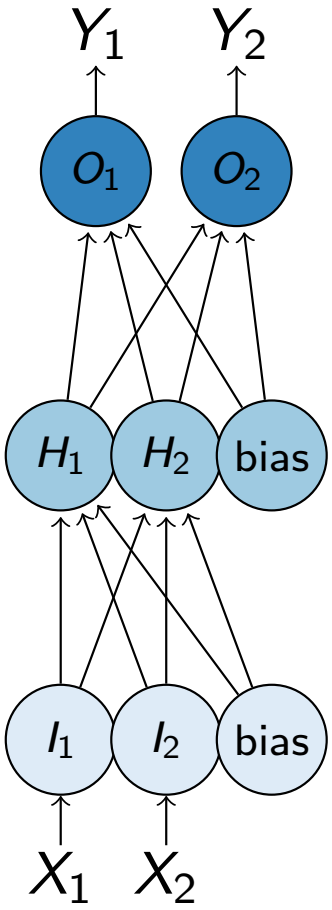

Figure: N-MR-1

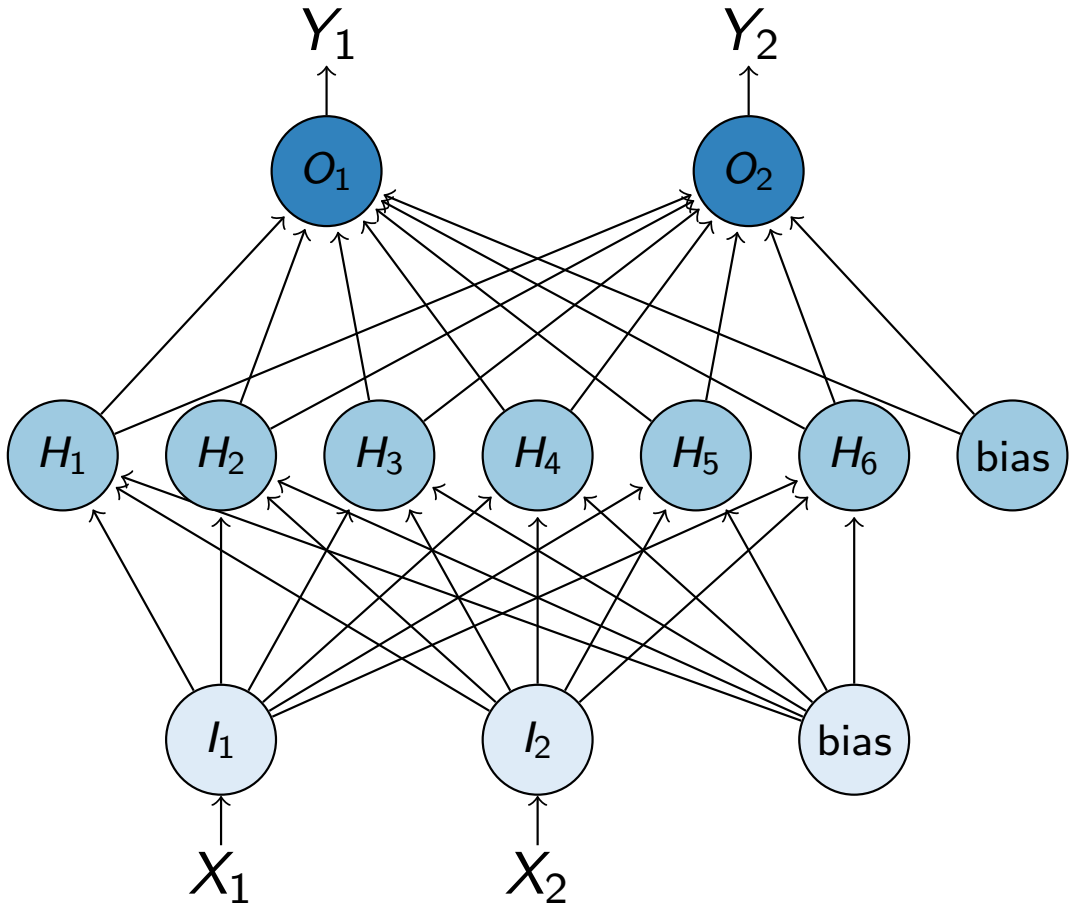

Figure: N-MR-3

N-MR Technique

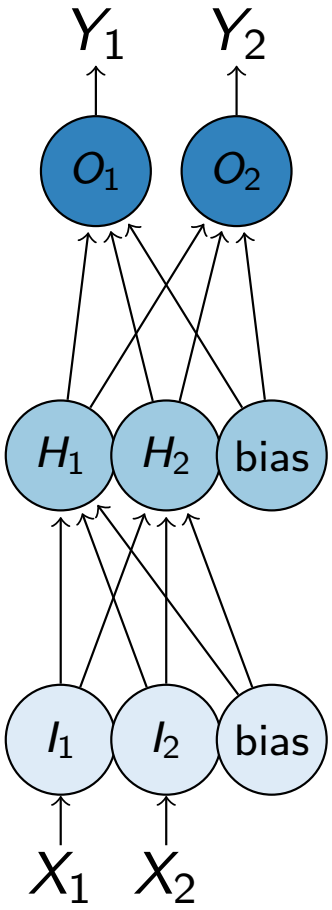

Figure: N-MR-1

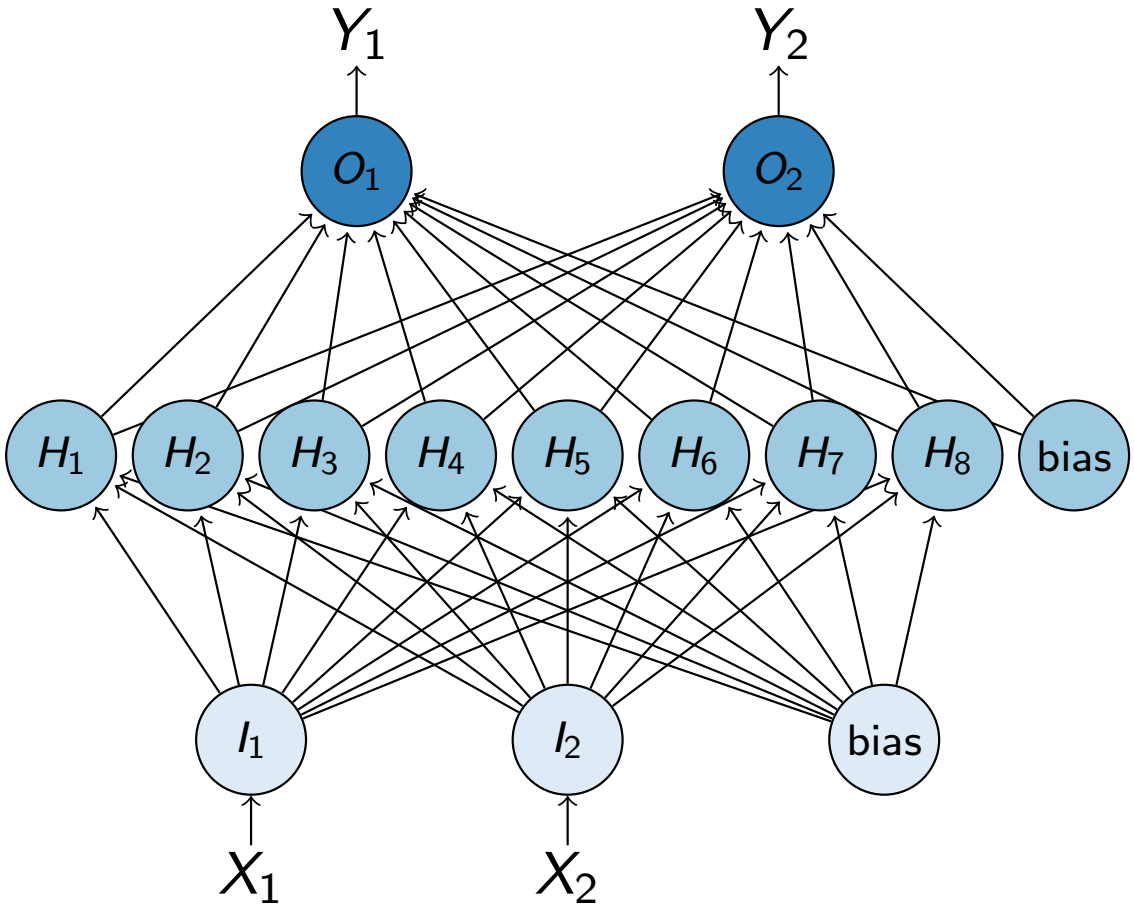

Figure: N-MR-4

Neural Network Accelerator Architecture

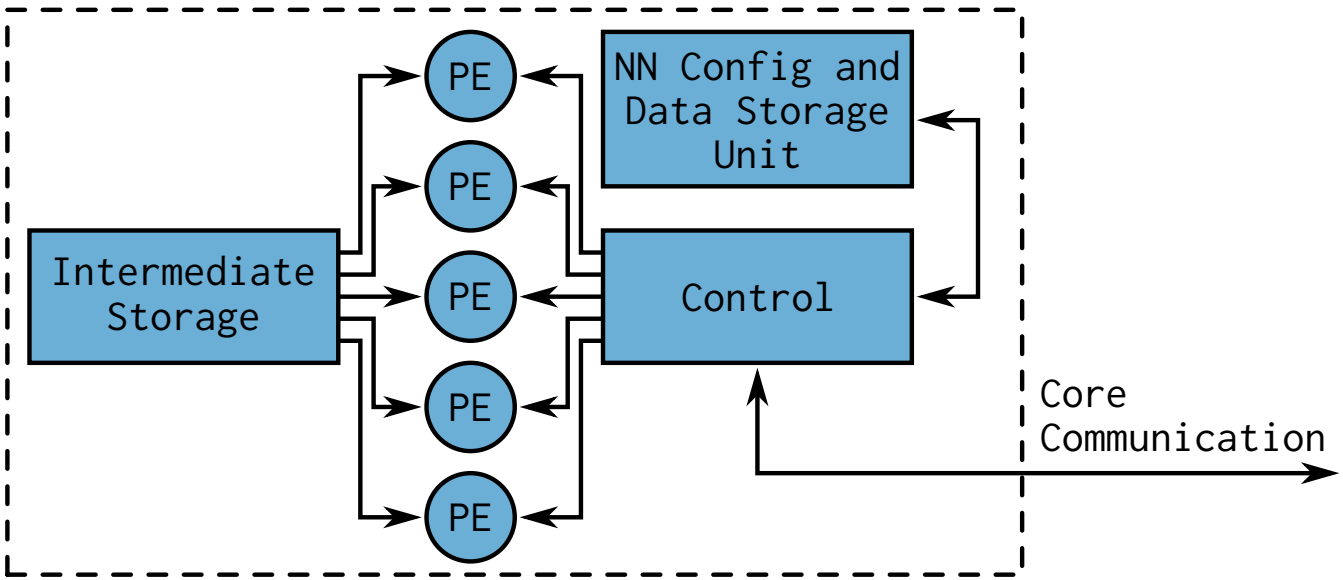

Figure: Block diagram of our neural network accelerator

Basic Operation in a Multicore Environment

- Threads communicate neural network computation requests to this accelerator
- The accelerator allocates processing elements (PEs) to compute the outputs of all pending requests

Neural Network Accelerator Architecture

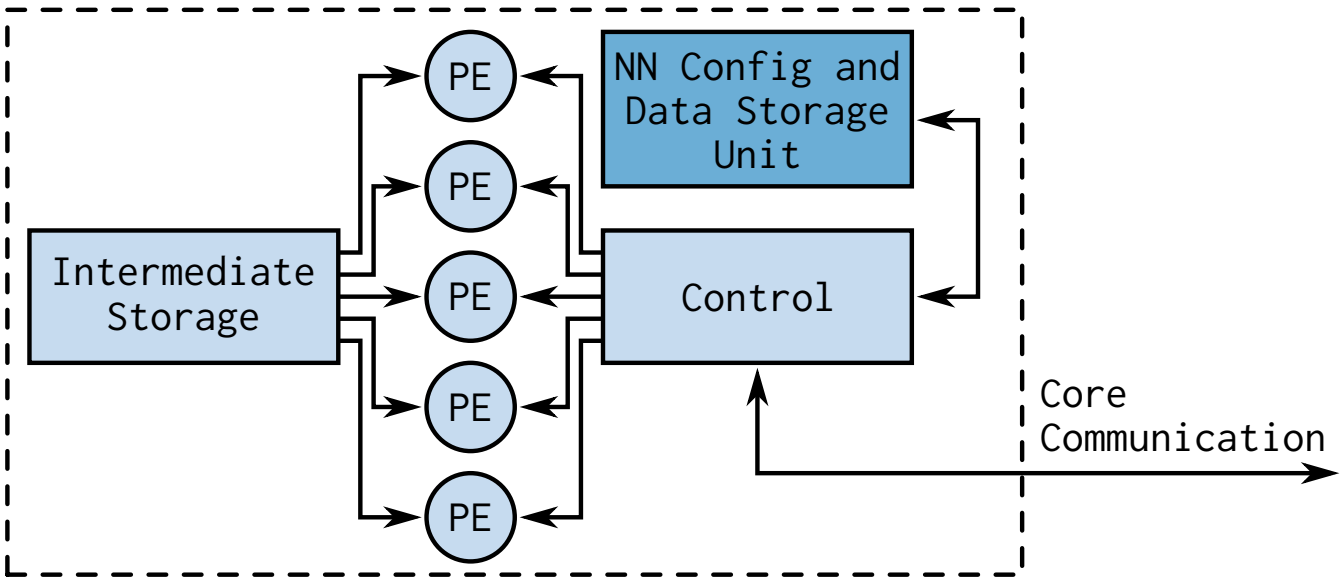

Figure: Block diagram of our neural network accelerator

Basic Operation in a Multicore Environment

- Threads communicate neural network computation requests to this accelerator
- The accelerator allocates processing elements (PEs) to compute the outputs of all pending requests

Neural Network Accelerator Architecture

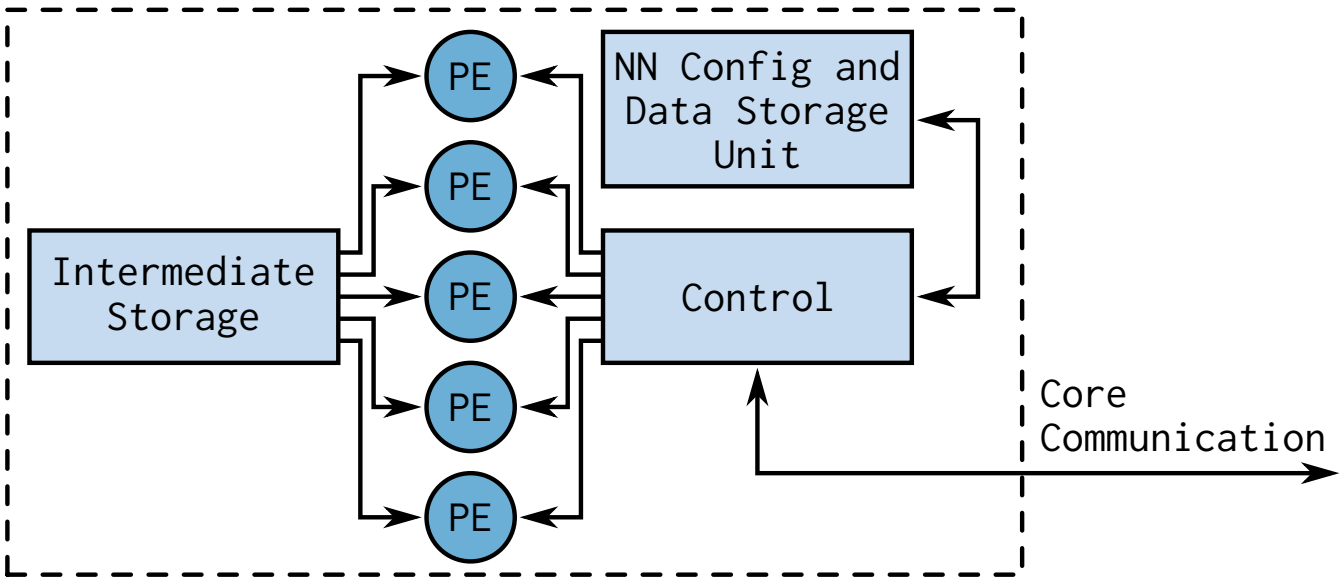

Figure: Block diagram of our neural network accelerator

Basic Operation in a Multicore Environment

- Threads communicate neural network computation requests to this accelerator
- The accelerator allocates processing elements (PEs) to compute the outputs of all pending requests

Neural Network Accelerator Architecture

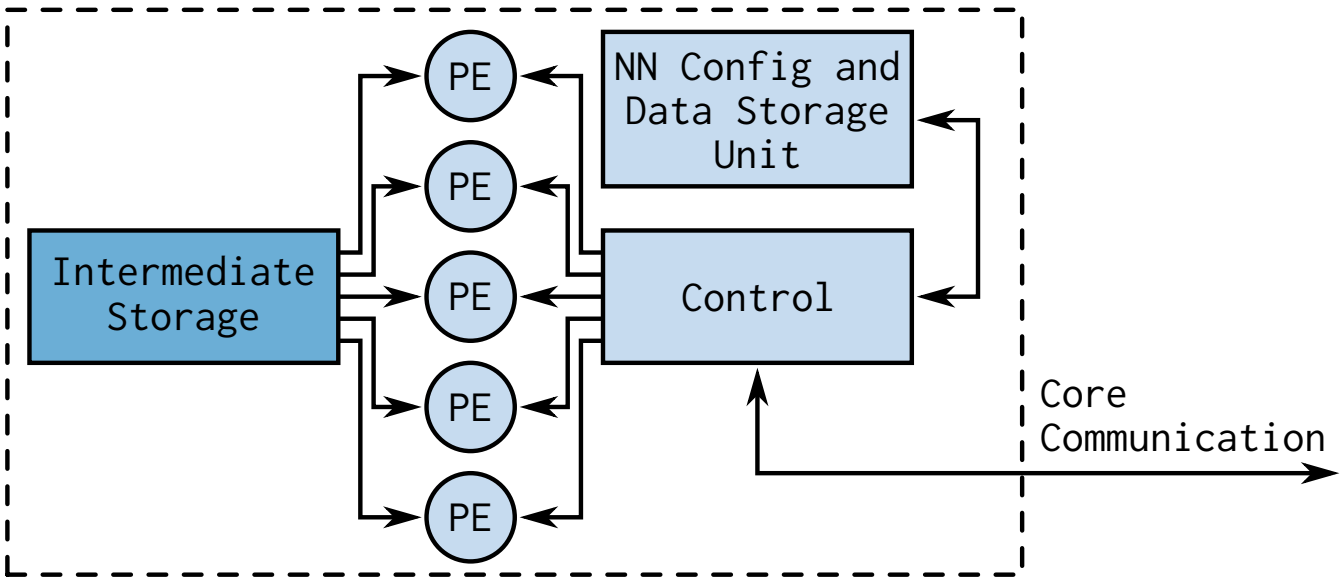

Figure: Block diagram of our neural network accelerator

Basic Operation in a Multicore Environment

- Threads communicate neural network computation requests to this accelerator
- The accelerator allocates processing elements (PEs) to compute the outputs of all pending requests

Neural Network Accelerator Architecture

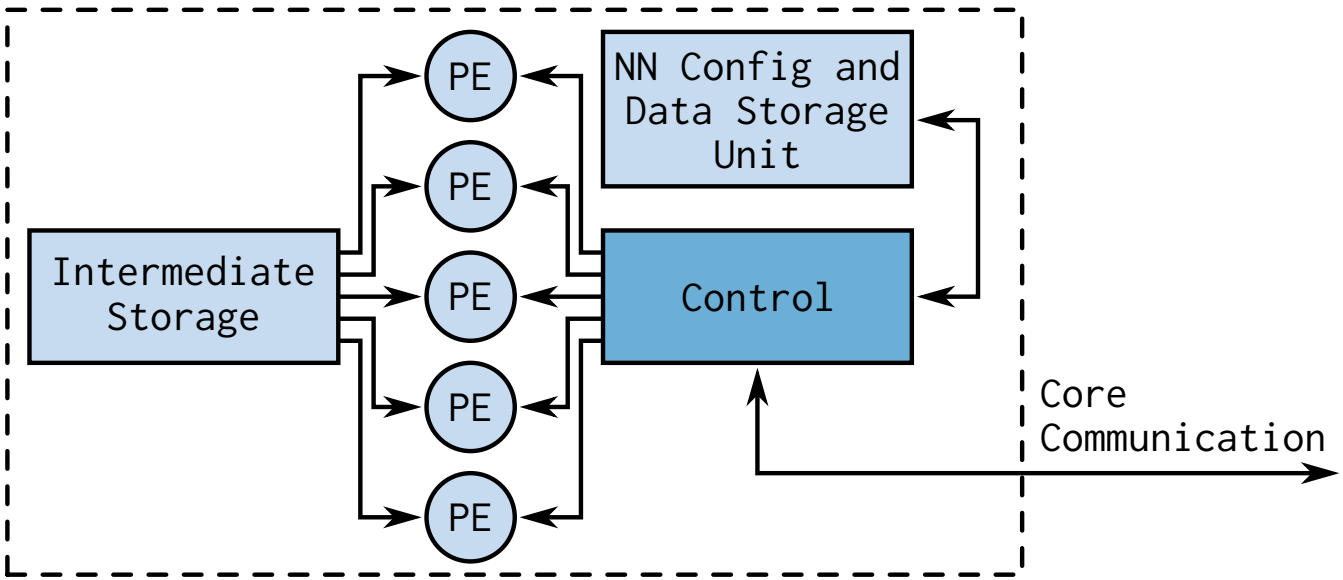

Figure: Block diagram of our neural network accelerator

Basic Operation in a Multicore Environment

- Threads communicate neural network computation requests to this accelerator
- The accelerator allocates processing elements (PEs) to compute the outputs of all pending requests

Evaluation Overview

Table: Evaluated neural networks and their topologies

Application	NN Topology	Description
`blackscholes (b)` [1]	$6 \times 8 \times 8 \times 1$	Financial option pricing
`rsa (r)` [2]	$30 \times 30 \times 30$	Brute-force prime factorization
`sobel (s)` [1]	$9 \times 8 \times 1$	3×3 Sobel filter

Methodology

- We vary the amount of *N*-MR for the applications in Table 1 running on our NN accelerator architecture
- We introduce a random fault into a neuron and measure the accuracy and latency

R. St. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014, pp. 505–516.

A. Waterland *et al.*, "Asc: Automatically scalable computation," in *ASPLOS*. ACM, 2014, pp. 575–590.

Evaluation – Normalized Latency

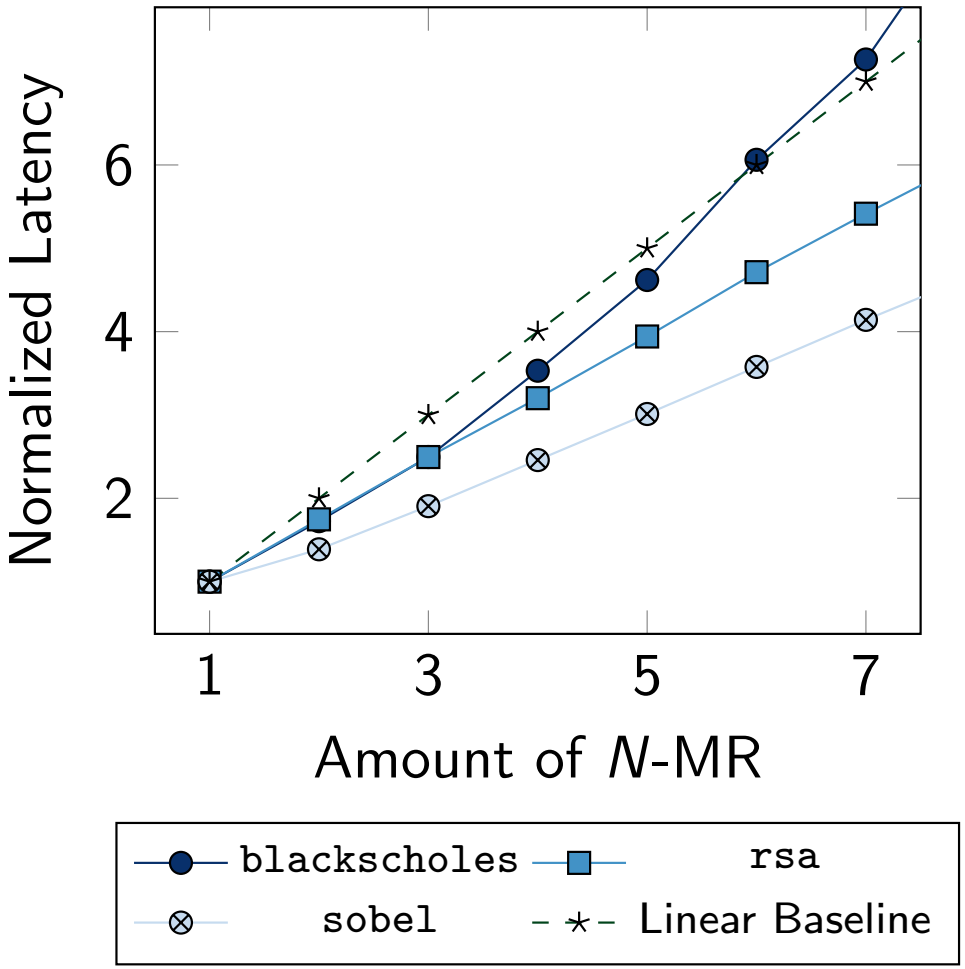

Figure: Latency normalized to N-MR-1

Latency Scaling with N-MR

- Work, where work is the number of edges to compute, scale with N-MR
- However, latency scales sublinearly for our accelerator
- Increasing N-MR means more work, but also more efficient use of the accelerator

Evaluation – Accuracy

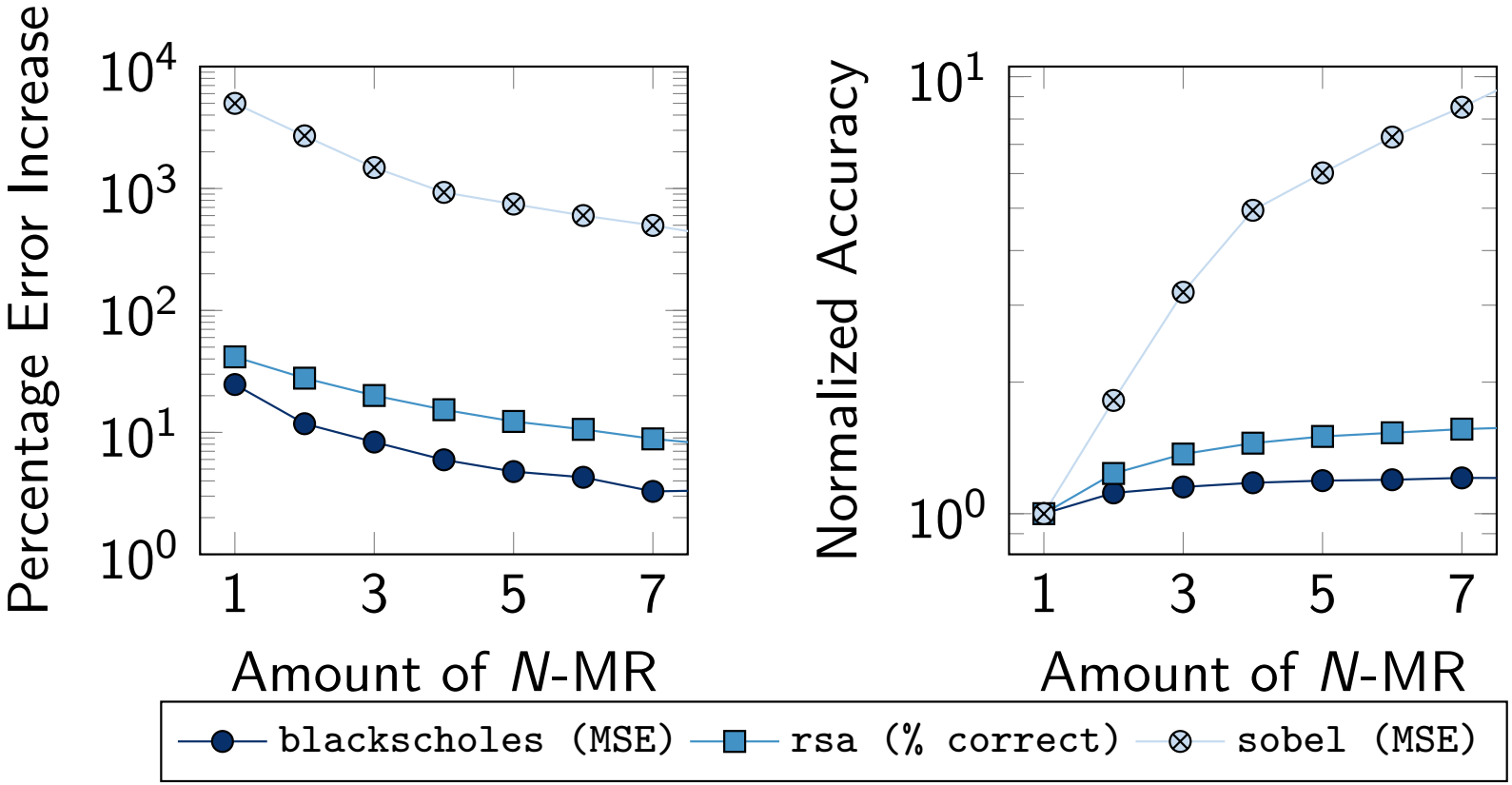

Figure: *Left*: percentage accuracy difference, *Right*: accuracy normalized to *N*-MR-1

Accuracy and *N*-MR

- Generally, accuracy improves with increasing *N*-MR

Evaluation – Combined Metrics

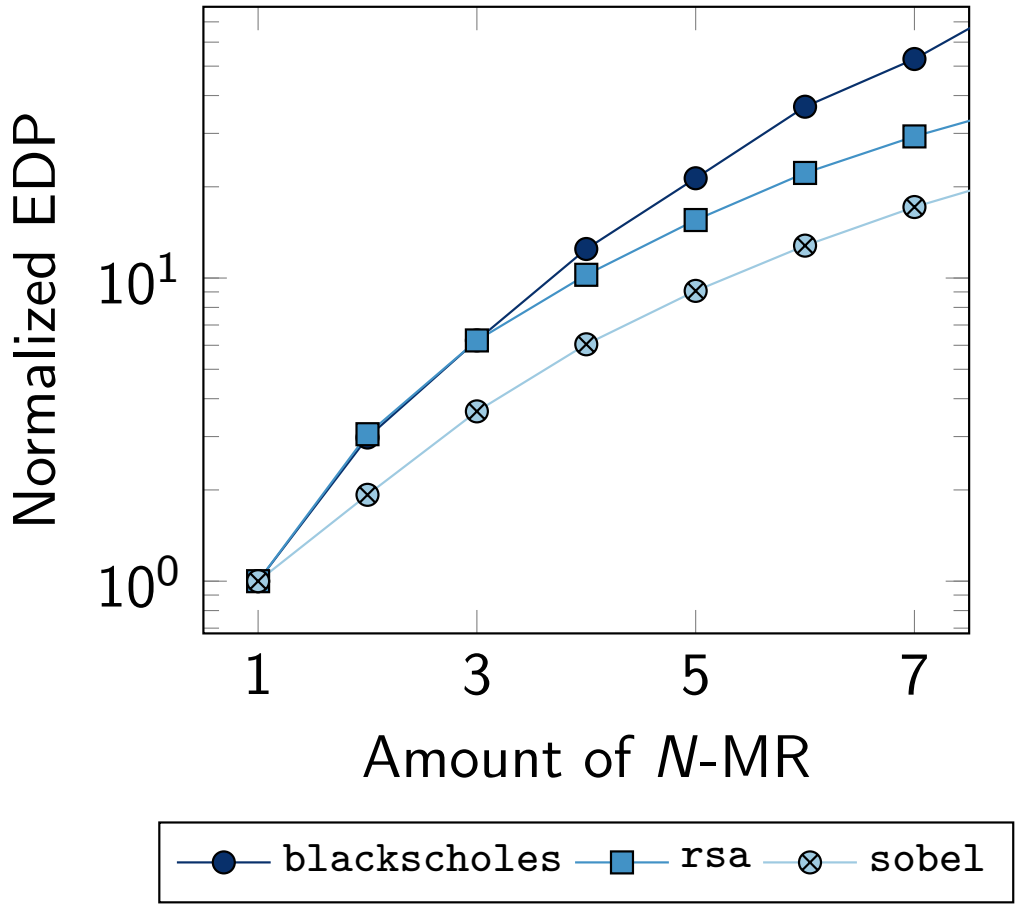

Figure: Energy-Delay Product (EDP) for varying *N*-MR

Cost of *N*-MR

- We evaluate the cost using Energy-Delay product (EDP)
- A high cost as *N*-MR increases *both* energy and delay

Discussion and Conclusion

An Initial Approach

- As neural network accelerators become mainstream, approaches to improve their fault-tolerance will have increased value
- *N*-MR is a preliminary step to leverage the *potential* for fault-tolerance in neural networks
- Other approaches do exist:
  - Training with faults
  - Splitting important neurons and pruning unimportant ones

Future Directions

- Varying *N*-MR at run-time
- Faults are currently assumed to be intermittent, but by varying internal PE structure and enforcing scheduling neurons on different PEs, a more robust approach can be developed
- Run-time splitting of important nodes or not computing unimportant nodes

Summary and Questions

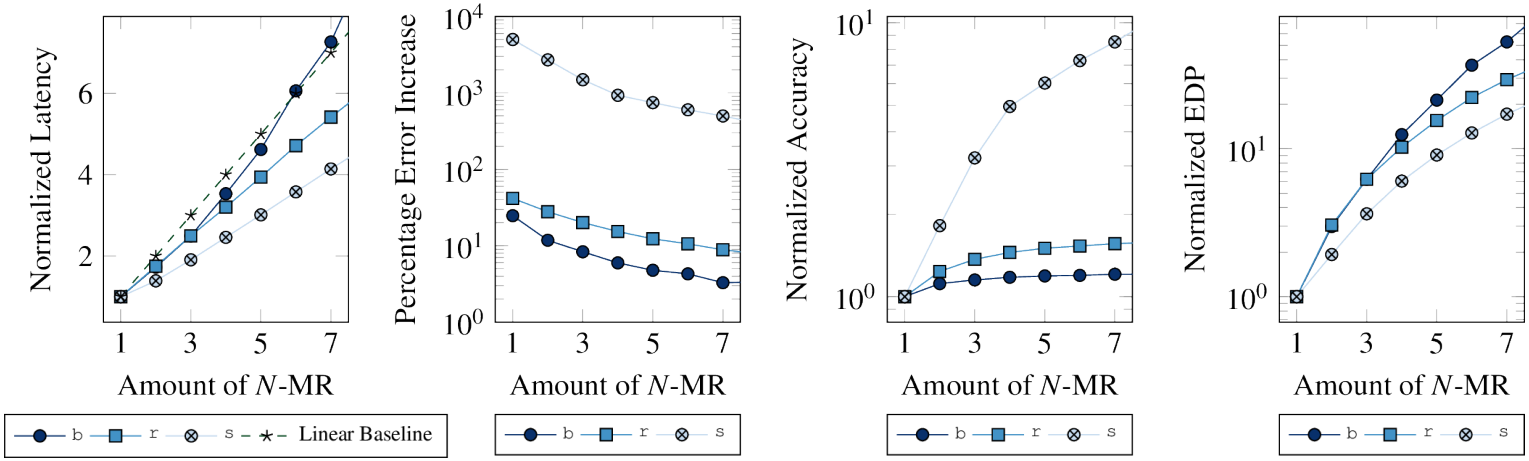

Figure: Latency, accuracy, and combined metrics

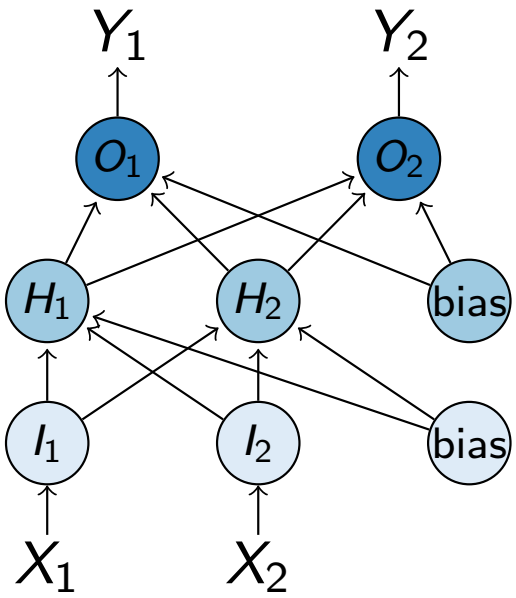

Figure: A two-layer NN

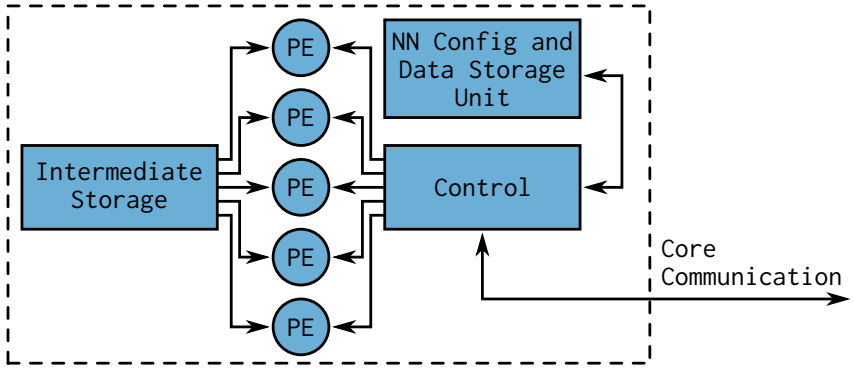

Figure: NN accelerator architecture