

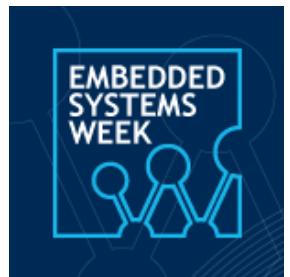
Heterogeneous Processing-in-Memory enabled Manycore Systems for Deep Learning



Bires Joardar @
University of Houston



Jana Doppa @
Washington State University



Tutorial at Embedded Systems Week (ESWEEK), 2022

Outline of Tutorial

- Exponential growth of Deep Learning and Hardware Challenges
- Introduction to Deep Learning
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- Methods to improve Reliability of ReRAMs
- Summary and Promising Directions

Outline of Tutorial

- Exponential growth of Deep Learning and Hardware Challenges
- Introduction to Deep Learning
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- Methods to improve Reliability of ReRAMs
- Summary and Promising Directions

Trend in deep learning models



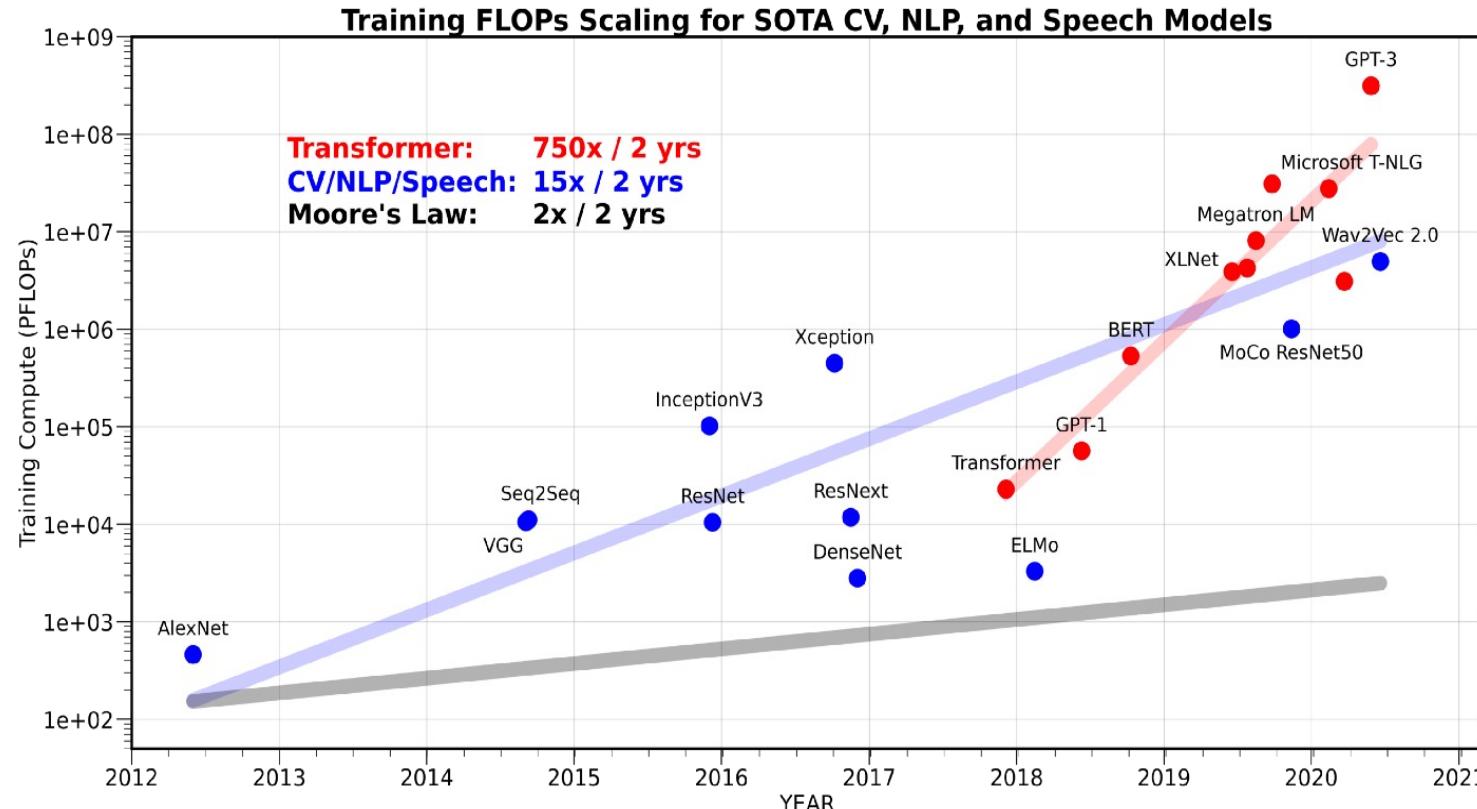
- Gigabytes of data, model parameters
 - Huge processing time
 - Monetary cost
- Optimized hardware necessary

Deep learning on GPUs/CPUs

- Common Hardware platforms for Deep learning applications:
 - CPU + *External* GPU
 - Cloud Services e.g., Microsoft Azure
- CPUs/GPUs are not optimized for Deep Learning tasks
 - Bandwidth bottleneck
 - Sub-optimal performance, energy



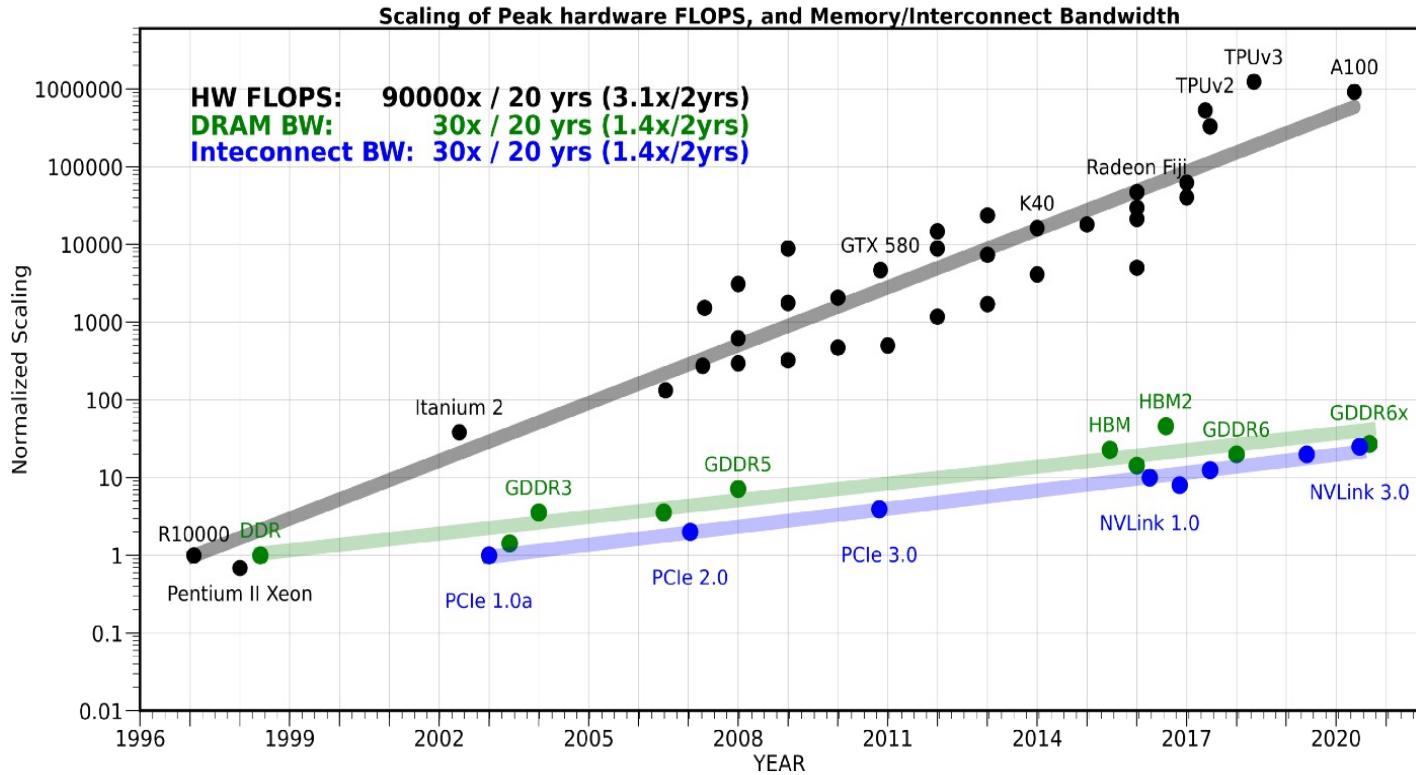
DL growth vs Hardware growth



- Transformer models have been growing at 750x every 2 years
- Number of transistors only doubles every 2 years
- Gap between what is needed vs what is available
- Slows innovation in ML

<https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8>

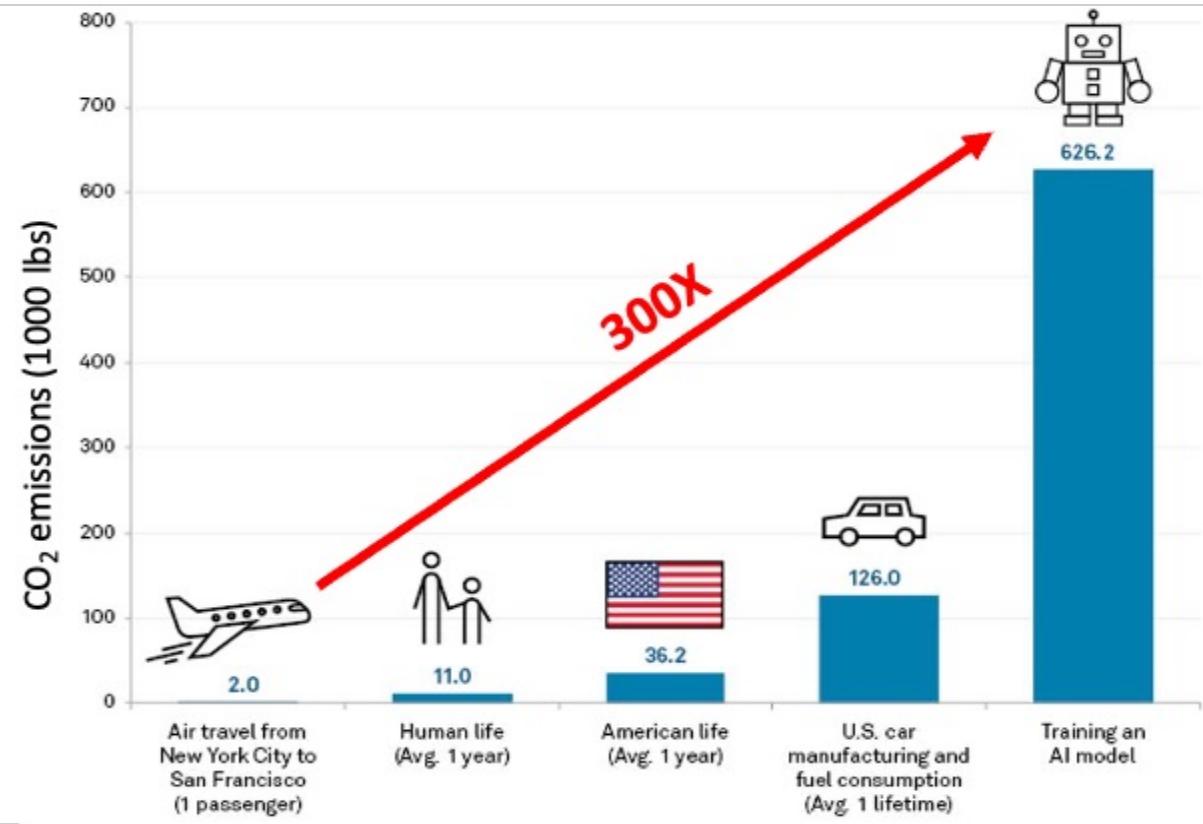
How to design better hardware



<https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8>

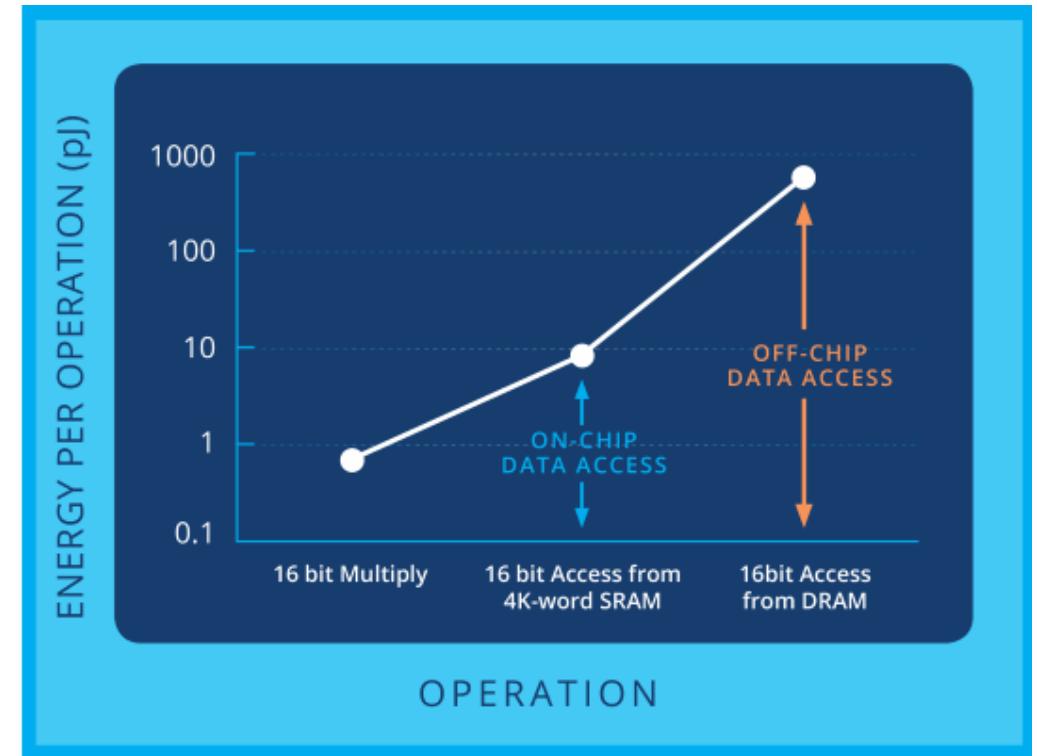
- Computation is super fast!
- End of Moore's law
- Data movement is a huge bottleneck
 - The “Memory wall” problem

Environmental impact of AI



- ML models are getting larger and larger
- Not sustainable and environment friendly
- Develop better hardware to continue innovation in ML

Communication bottleneck

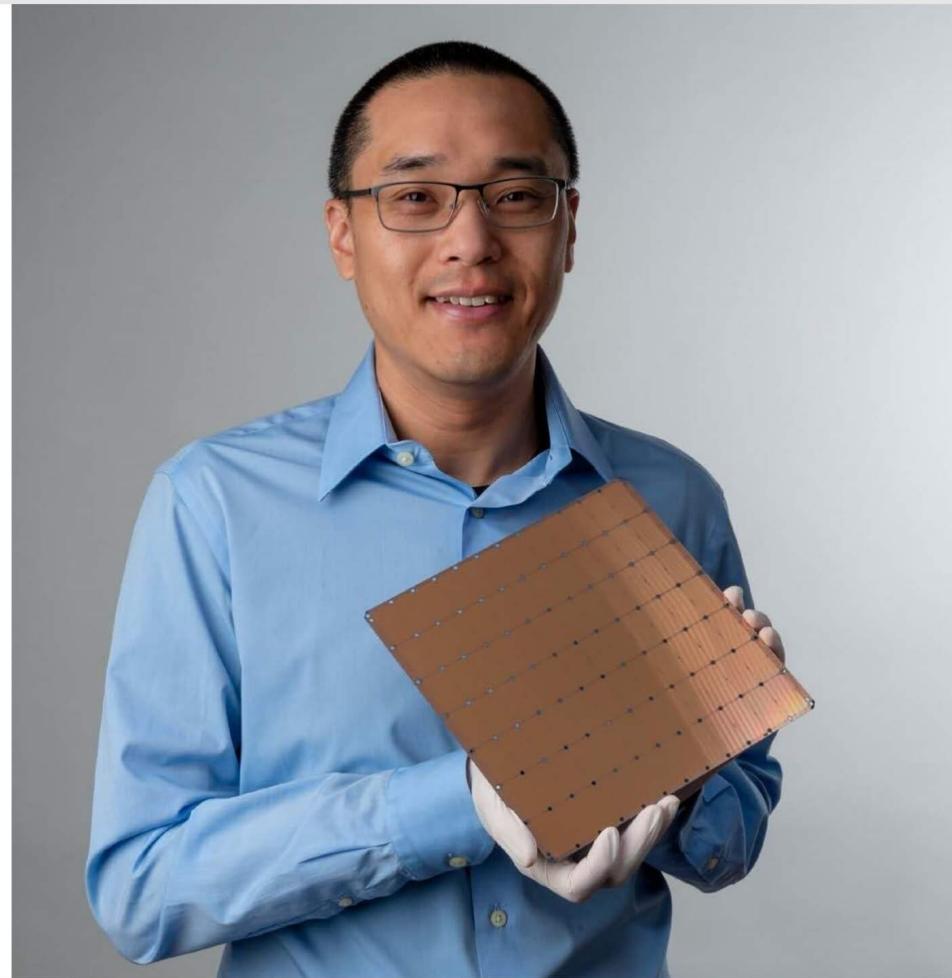


- Computation is super fast!
- Communication and memory access is the new bottleneck

Some latest AI accelerators

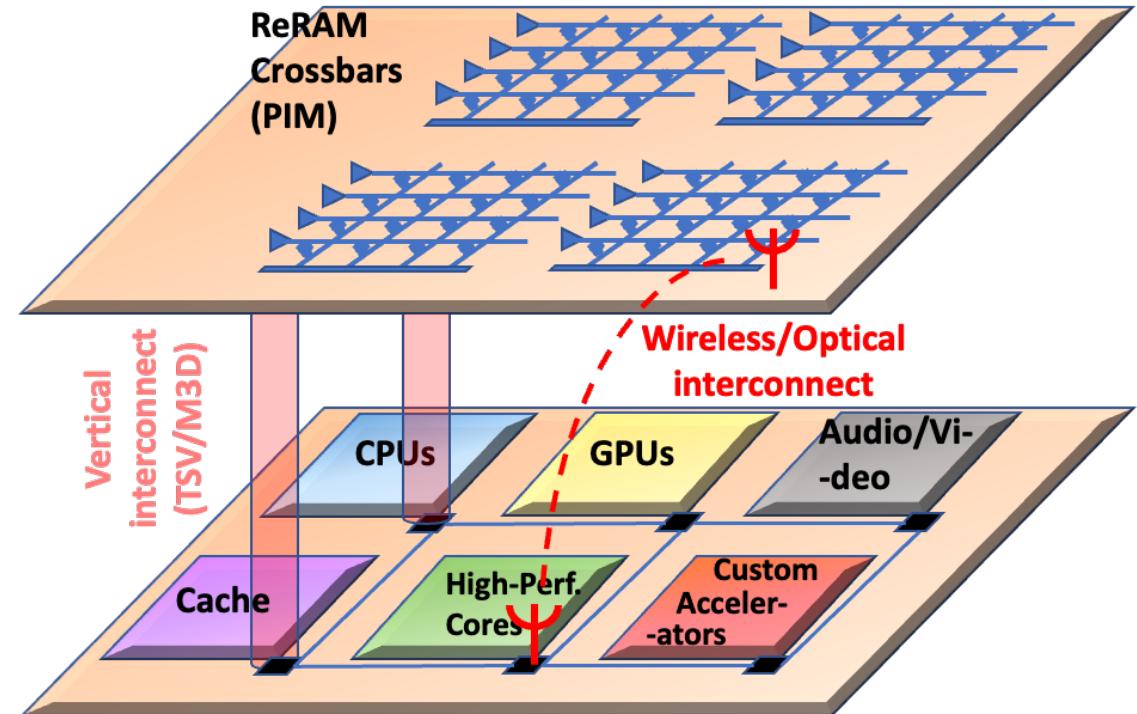
Largest Chip Ever Built

- 46,225 mm² silicon
- 1.2 trillion transistors
- 400,000 AI optimized cores
- 18 Gigabytes of On-chip Memory
- 9 PByte/s memory bandwidth
- 100 Pbit/s fabric bandwidth
- TSMC 16nm process

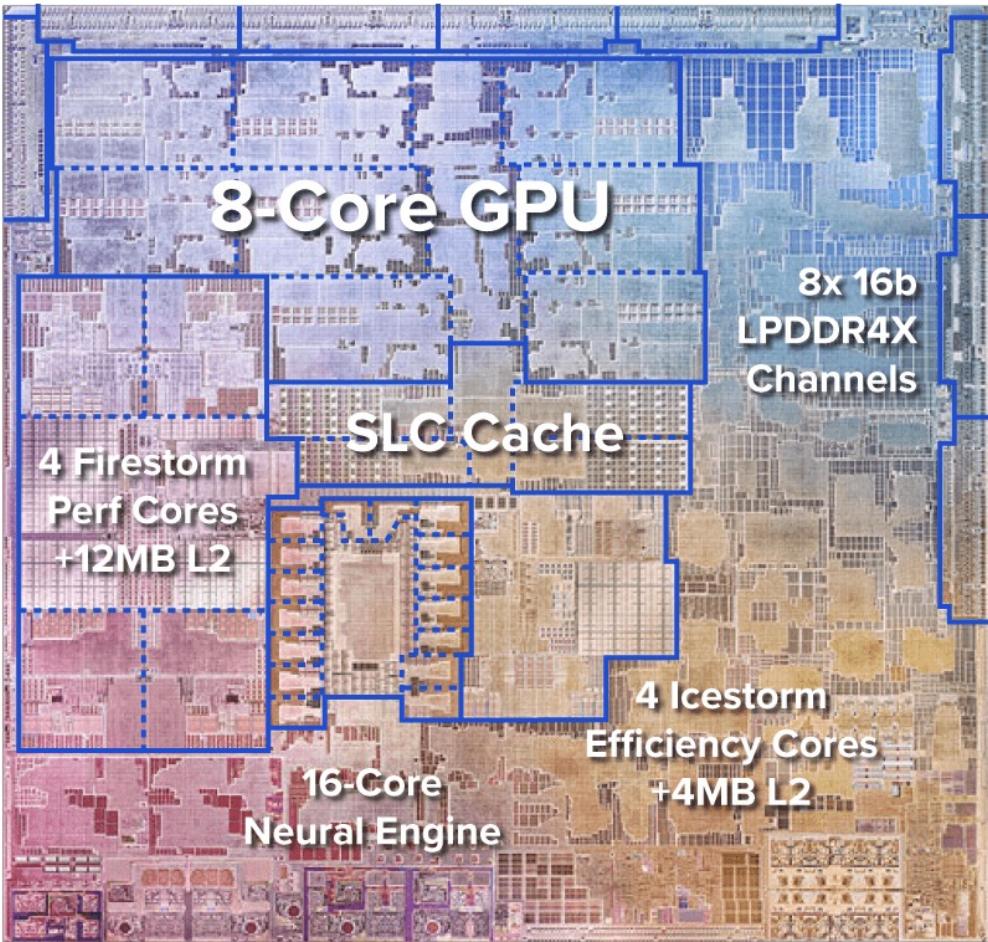


Envisioned future architecture

- 3D enables heterogeneous integration
- 3D architectures solve the communication bottleneck
 - High throughput
 - Low latency
 - Energy efficient



Apple M1 heterogeneous design

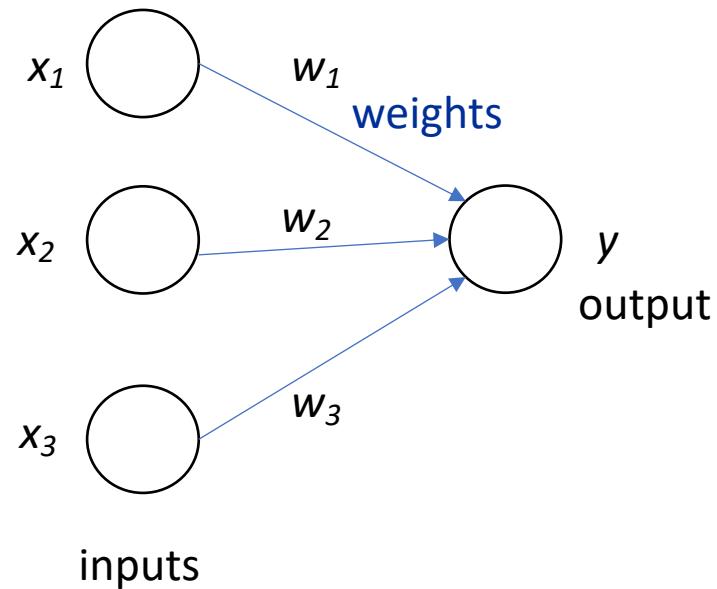


- Heterogeneous architectures are used commercially
- Apple M1 has two types of CPUs, GPUs, Neural engine
- High performance and energy efficiency

Outline of Tutorial

- Exponential growth of Deep Learning and Hardware Challenges
- **Introduction to Deep Learning**
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- Methods to improve Reliability of ReRAMs
- Summary and Promising Directions

Let's start with the simplest unit!

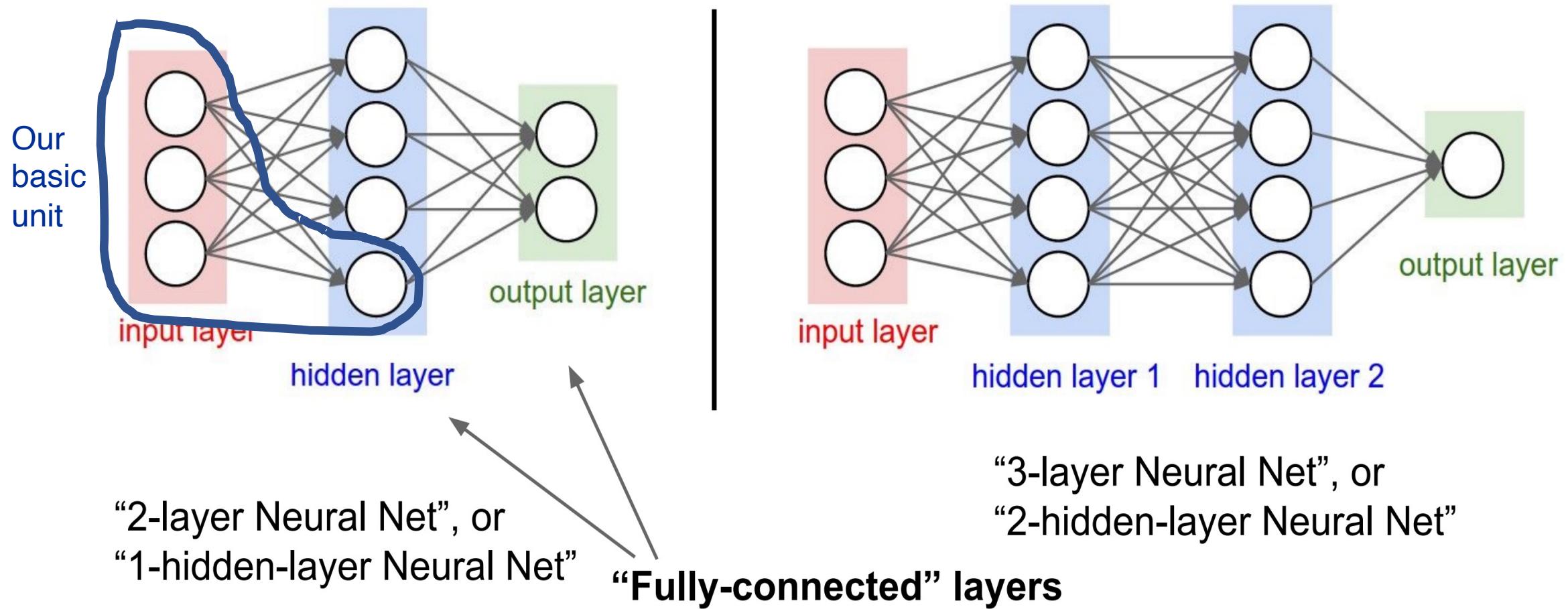


How to read this graphical representation

$y = \sigma(Wx)$, where φ is a non-linear function called as activation function

These units are much more powerful if we stack many of them together! This stacked network will be called a neural network!

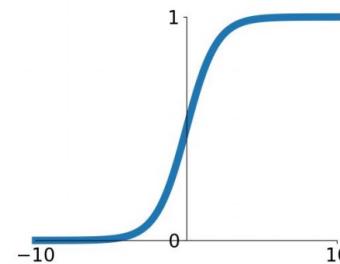
Neural Network



Common activation functions

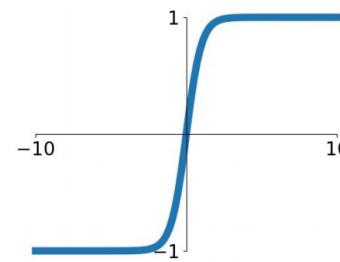
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



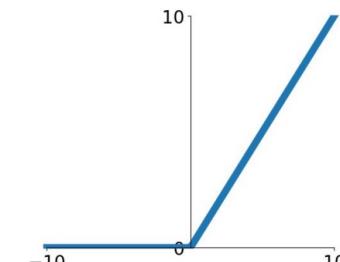
tanh

$$\tanh(x)$$



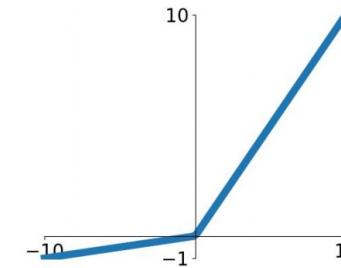
ReLU

$$\max(0, x)$$



ReLU is a good
default choice

Leaky ReLU

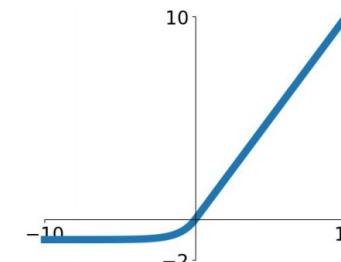
$$\max(0.1x, x)$$


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

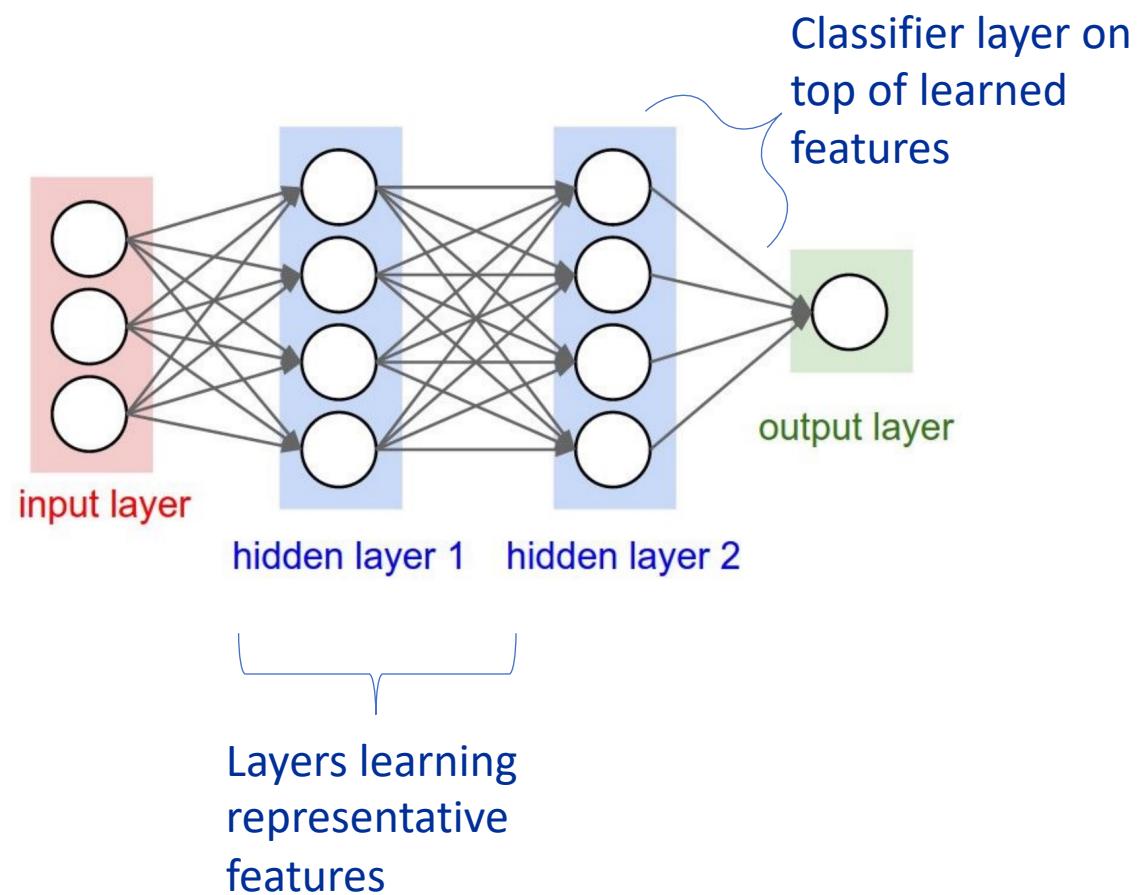
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



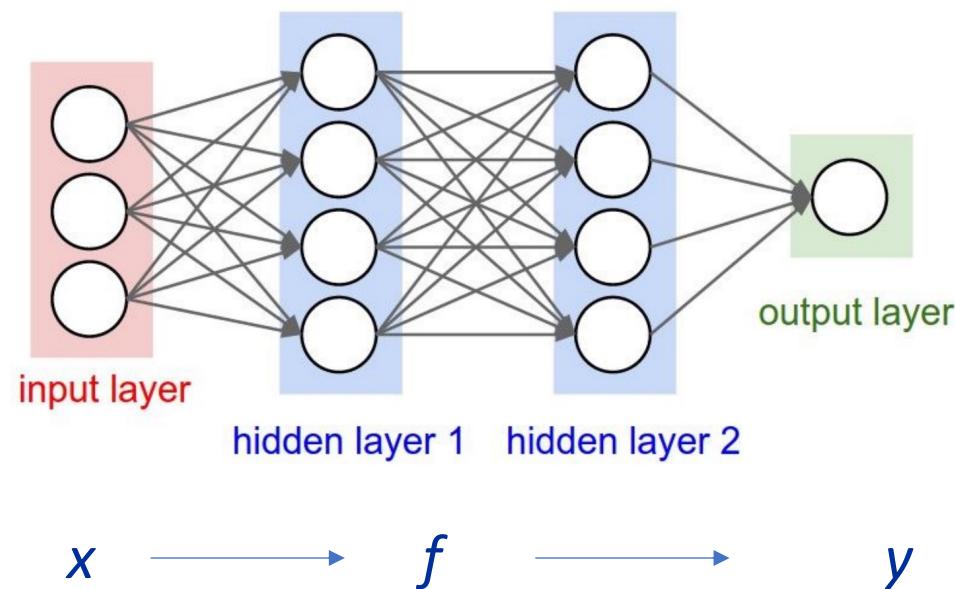
Differentiable Programming Paradigm

- The user writes a differentiable program
- Use training data to optimize the parameters of this program so that the program behaves as desired

Automatic Feature Learning

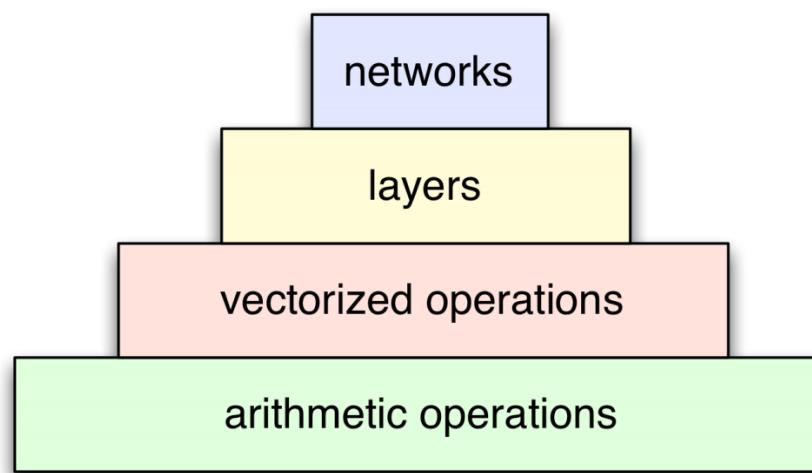


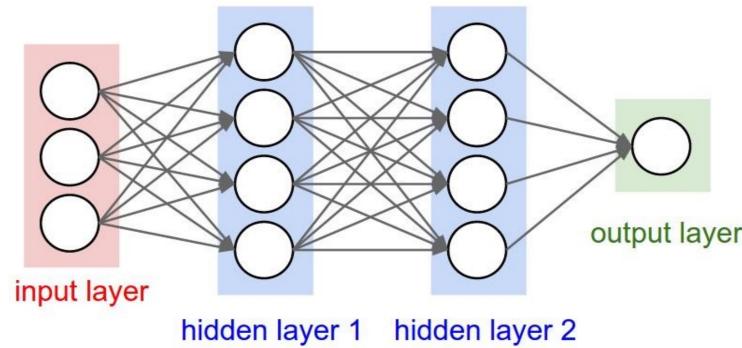
Function Approximator



$$f(x) = \sigma[W_2 [\sigma(W_1 x + b_1)] + b_2]$$

When you design neural networks and machine learning algorithms, you'll need to think at multiple levels of abstraction.



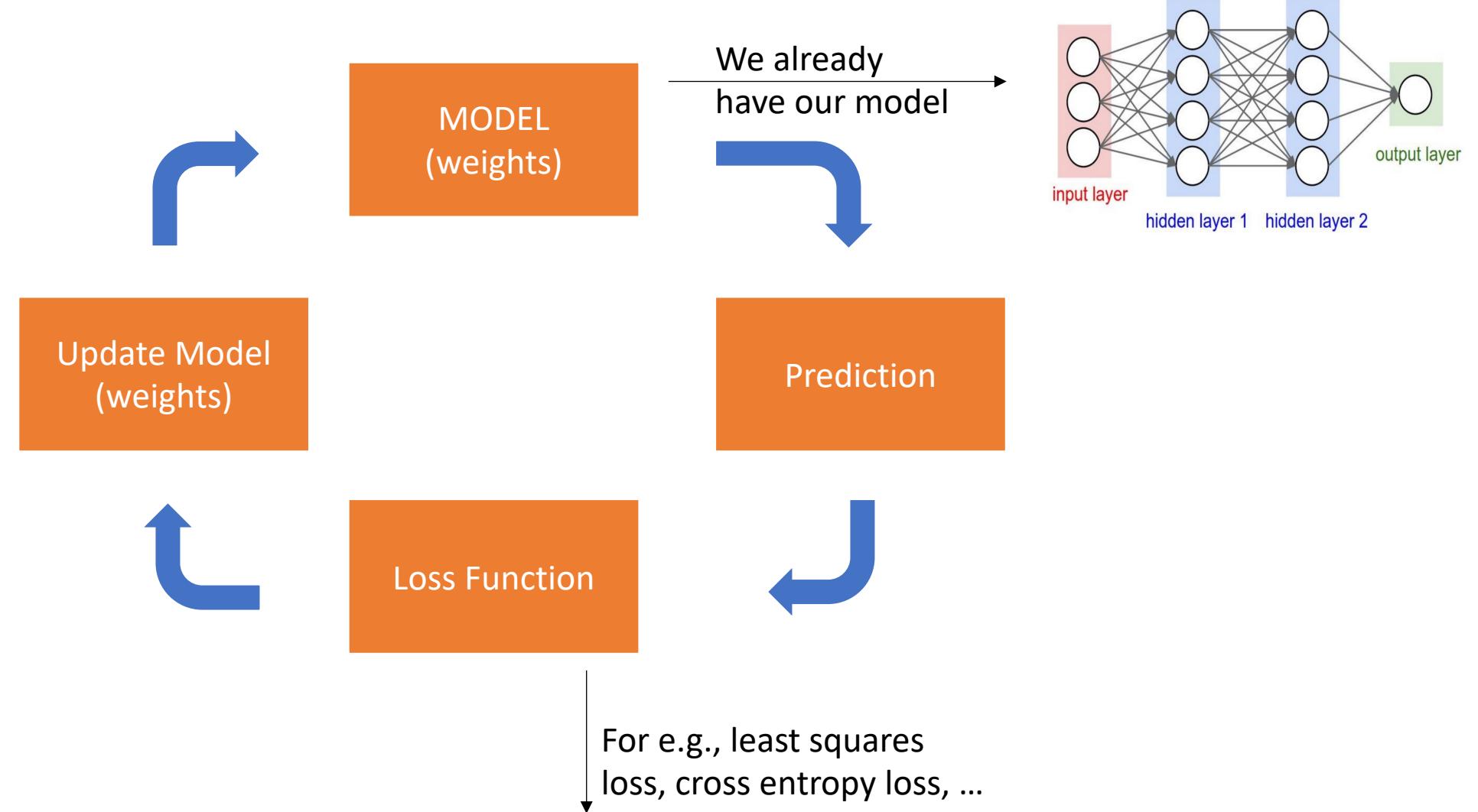


Now, we have our model!

How do we train it?

Backpropagation!

General (Supervised) ML Training Loop



Aside: Gradient Descent

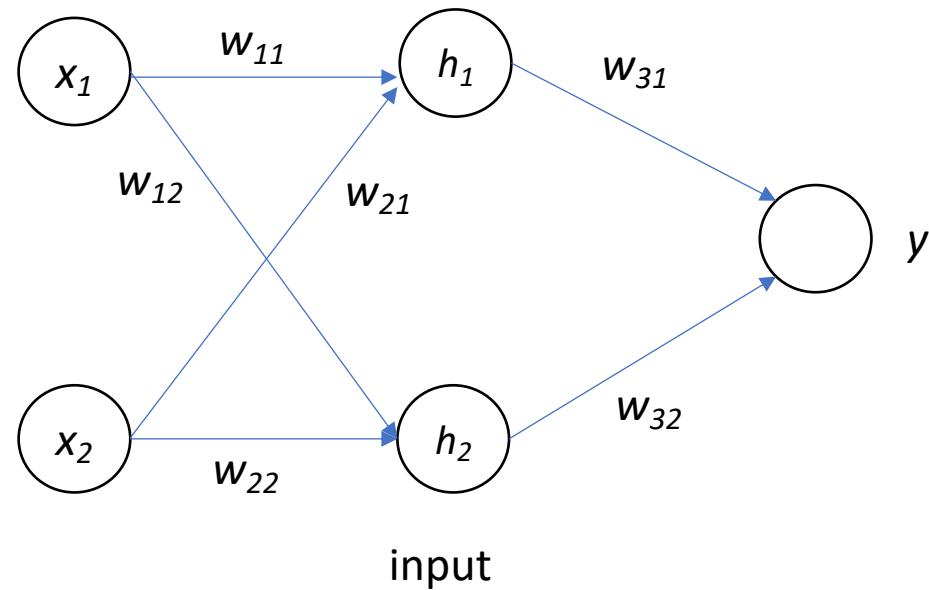
- Generic algorithm to minimize a continuous objective function.
- In our case, loss function is the objective.
- Key idea: take steps proportional to the negative of the gradient at the current point.
- Key equation:
 - $w_{t+1} = w_t - \eta \cdot \nabla L(w_t)$, where η is the learning rate
- Essentially, the main thing we require now
 - gradient of the loss function with respect to the weights ($\nabla L(w_t)$)

How do we compute the
gradients?

Backpropagation

- Efficient algorithm to compute gradients w.r.t weights
- Key Idea: Chain Rule of Calculus
- Central algorithm for training neural networks
- Let's consider a simple example to illustrate the idea

Backprop example on 1 hidden layer neural network

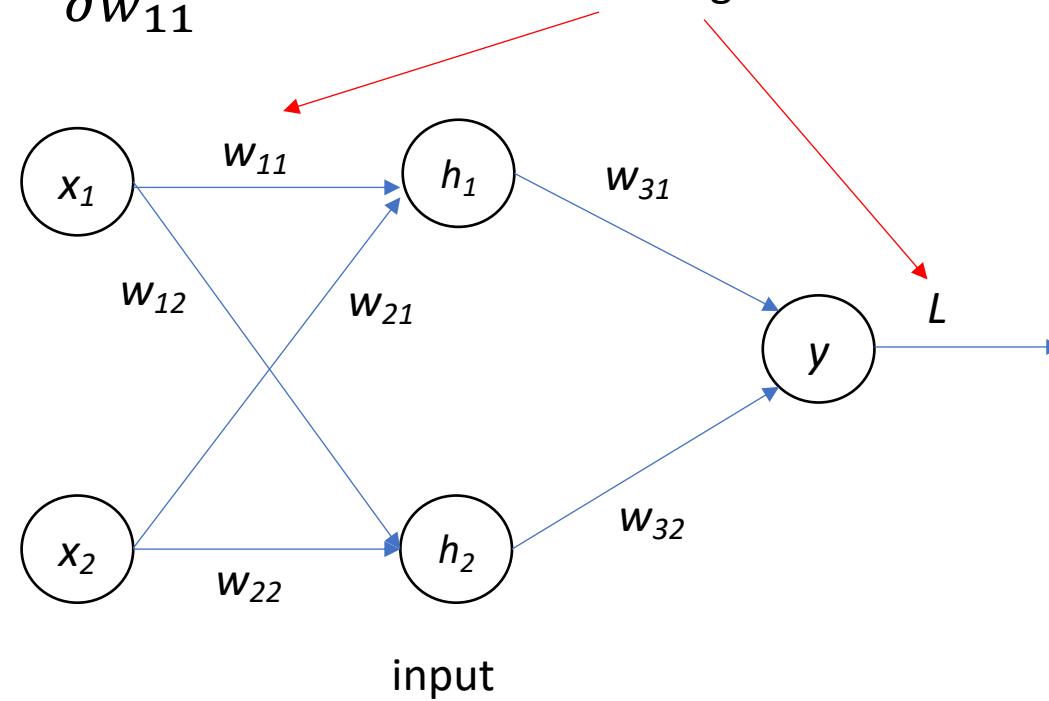


- $y = \sigma(w_{31}h_1 + w_{32}h_2)$ where $h_1 = \sigma(w_{11}x_1 + w_{21}x_2)$ and $h_2 = \sigma(w_{12}x_1 + w_{22}x_2)$

Backpropagation Example

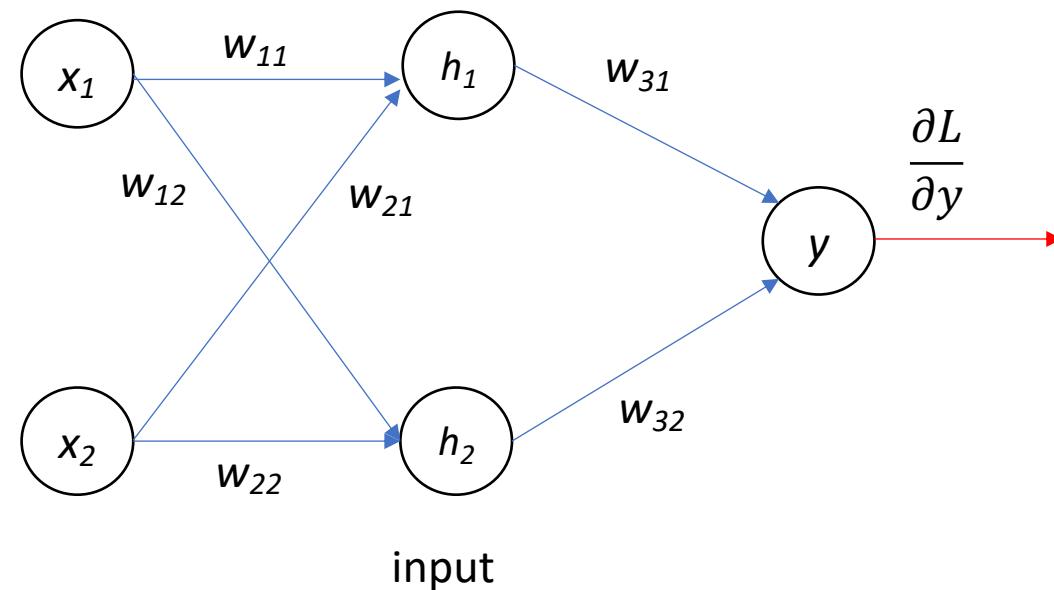
- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$

Compute the derivative of L
with this weight



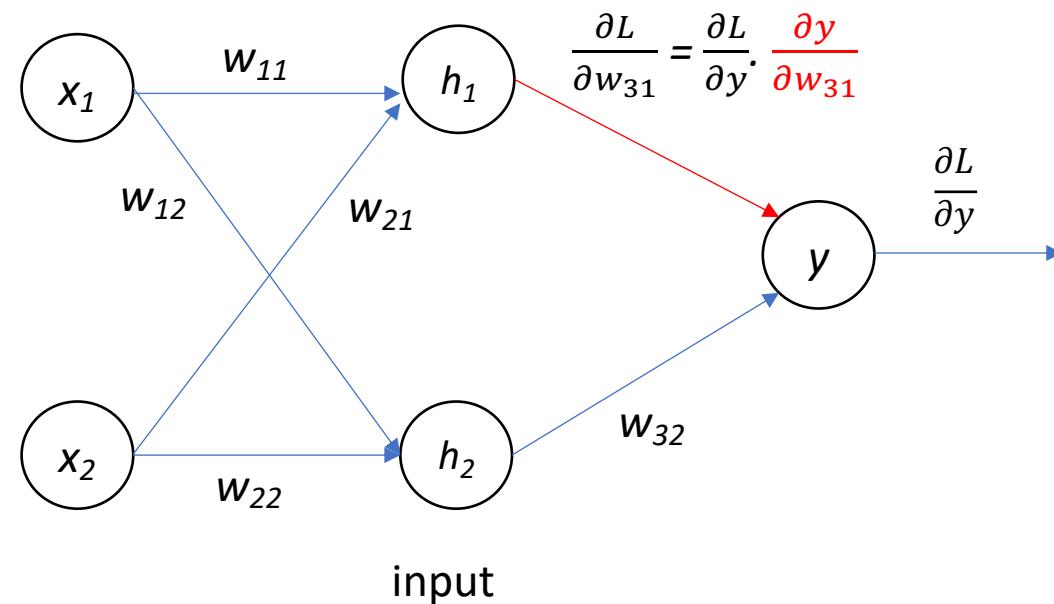
Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



Backpropagation Example

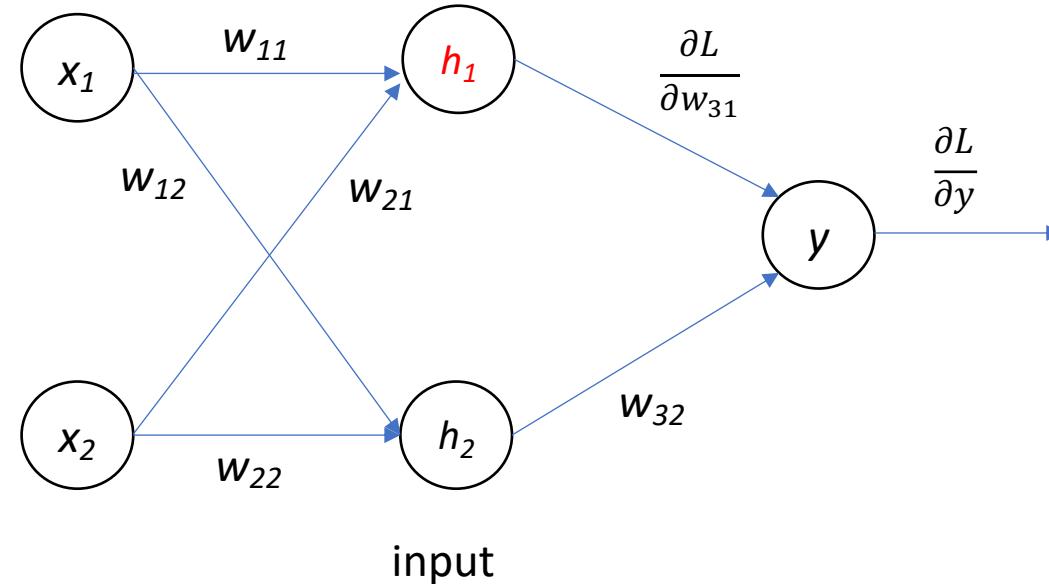
- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



Backpropagation Example

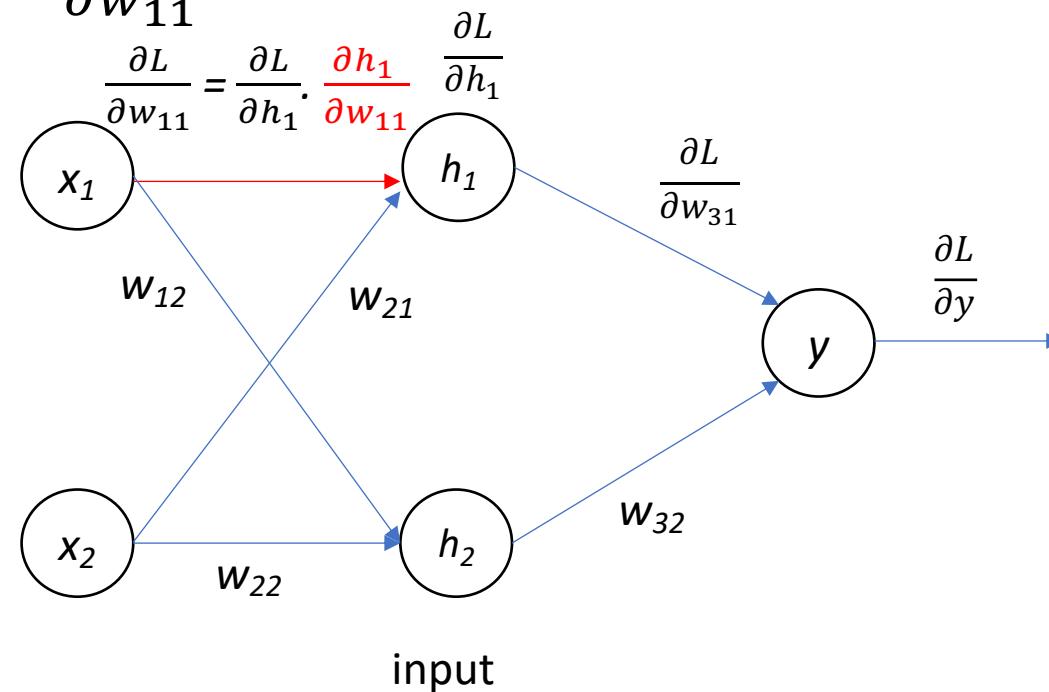
- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h_1}$$



Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



Overall Training

- Same procedure can be applied to other weights
- Overall Training of Neural Networks:
 - *Till convergence (for e.g. when validation loss stops decreasing), repeat:*
 - *Forward Pass - Compute the predictions*
 - *Compute training loss*
 - *Backward Pass – Compute the gradients*
 - *Update the weights with gradient descent equation*

Are you wondering that you need to calculate all these gradients by hand for each weight?

Answer is No!

Automatic Differentiation is the solution

Auto differentiation

- General way of taking a program which computes a value, and automatically constructing a procedure for computing gradients of that value.
 - For e.g. gradients of loss function.
- Implementing backprop by hand is like programming in assembly language.
 - You'll probably never do it, but it's important for having a mental model of how everything works
- Most of the deep learning libraries (PyTorch, TensorFlow) have this functionality as default

Auto differentiation

- An autodiff system will convert the program into a sequence of **primitive operations (ops)** which have specified routines for computing derivatives.
- In this representation, backprop can be done in a completely mechanical way.

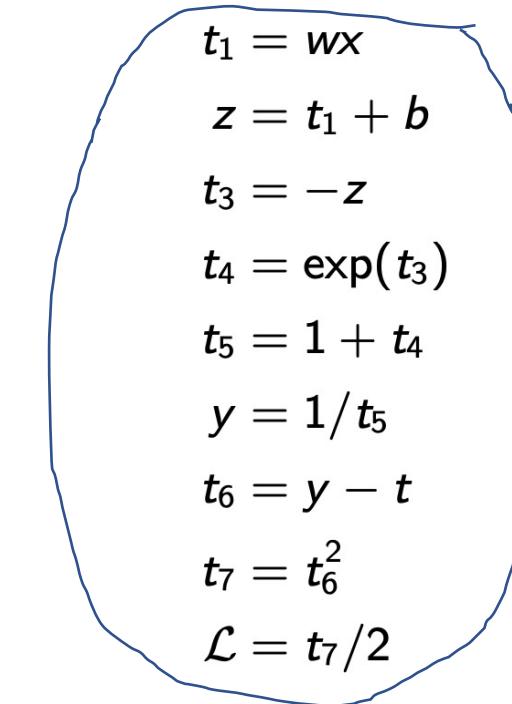
Sequence of primitive operations:

Original program:

$$z = wx + b$$

$$y = \frac{1}{1 + \exp(-z)}$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$



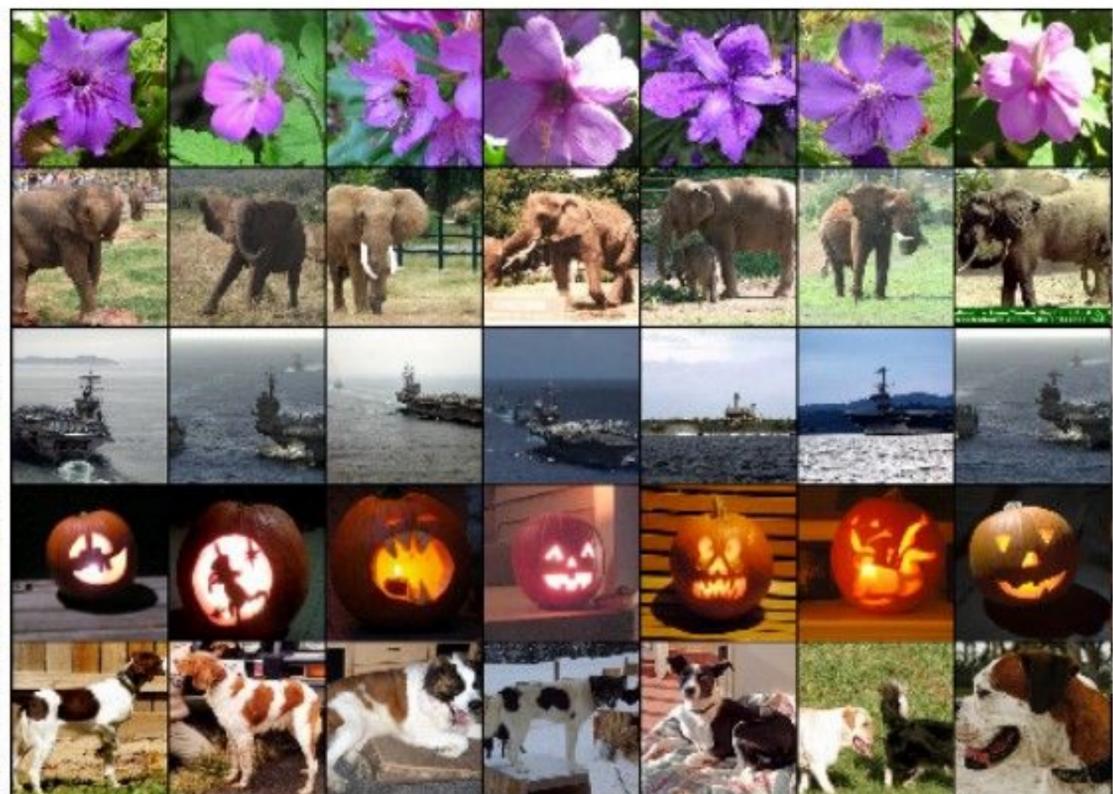
You can create all kinds of functions by combining these primitive operations

Convolutional Neural Networks

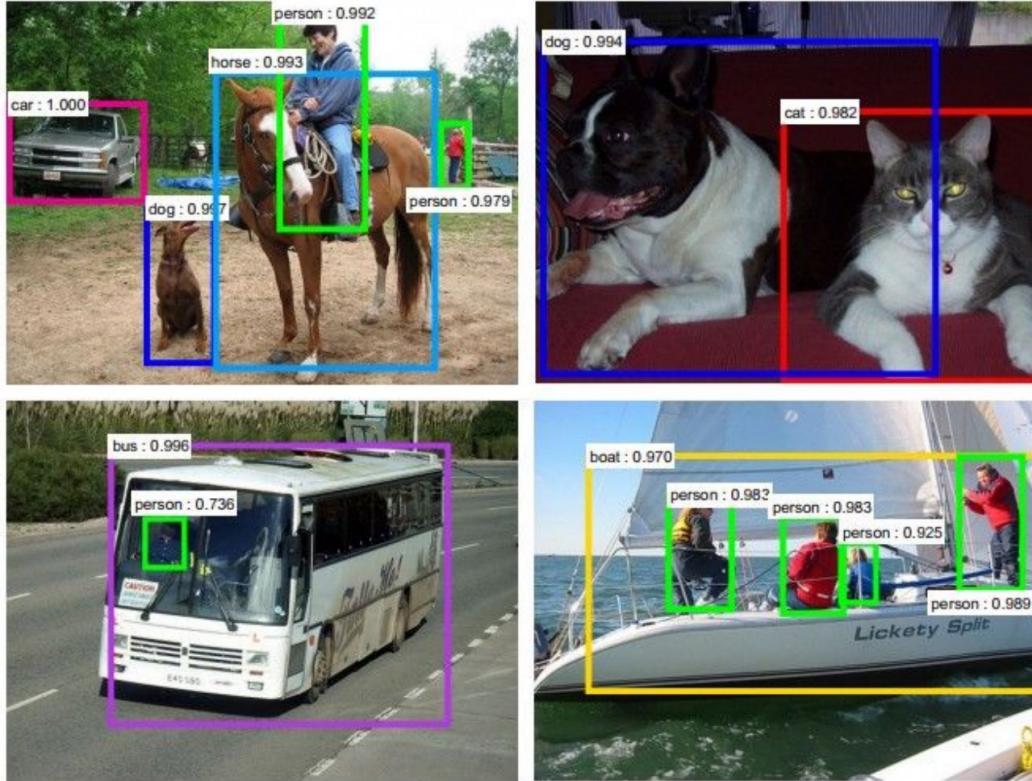
Classification



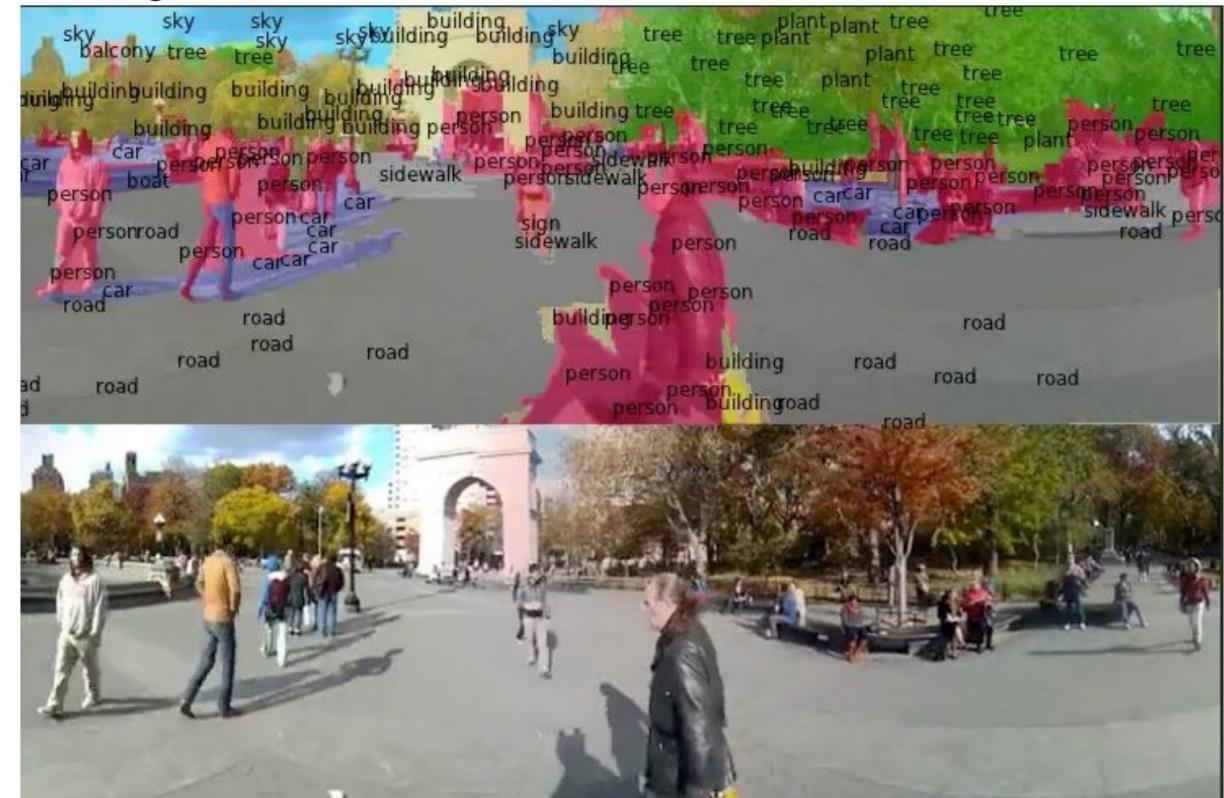
Retrieval

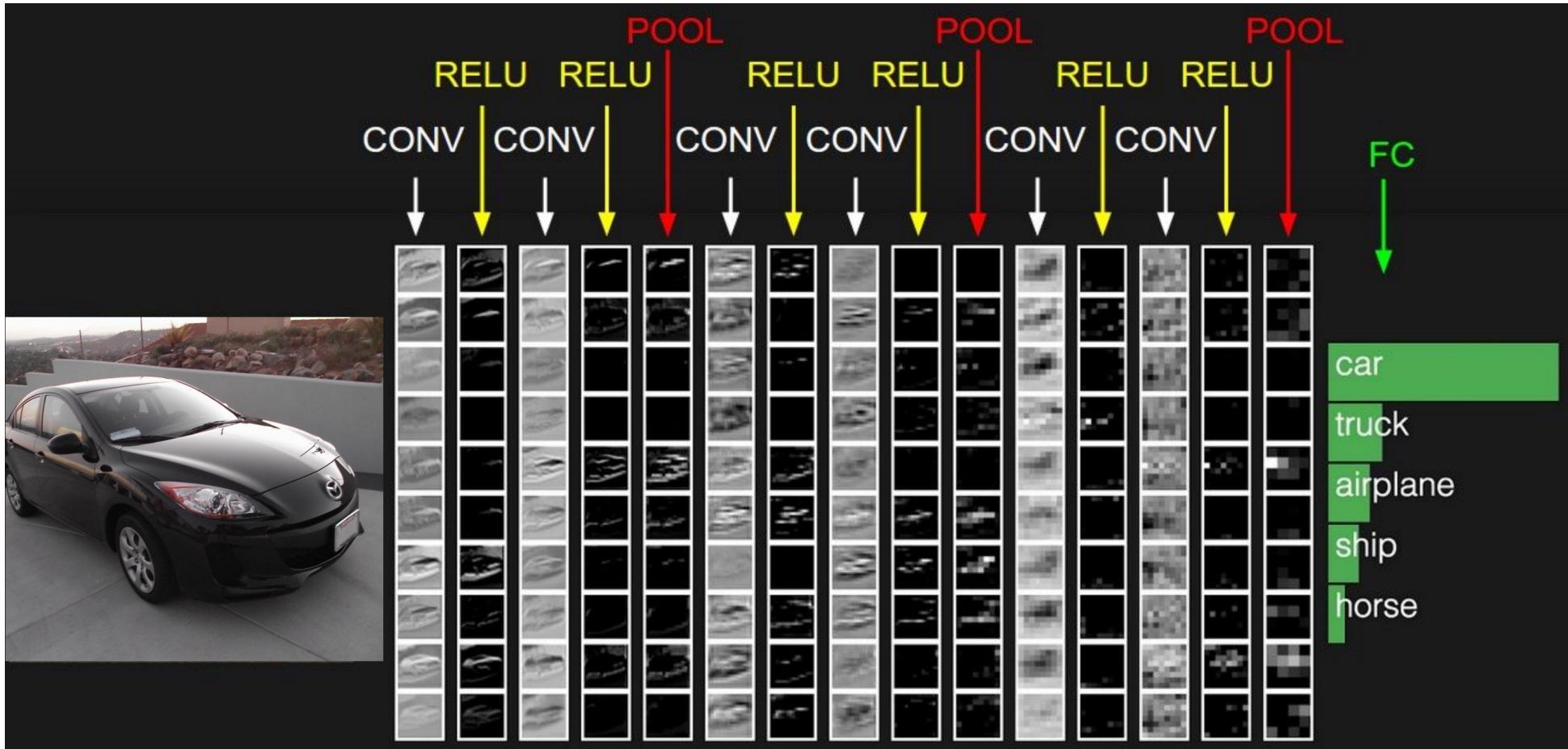


Detection



Segmentation



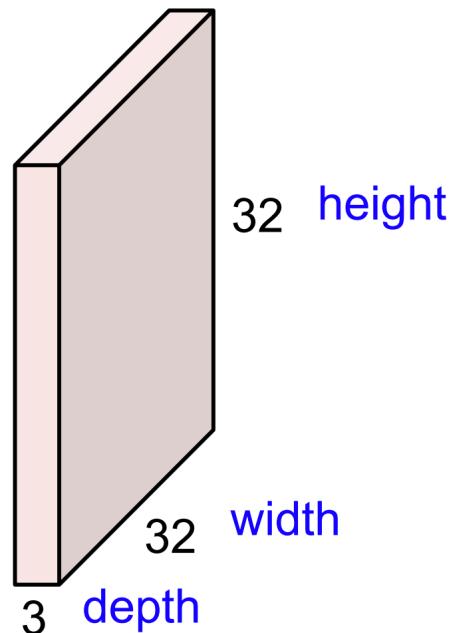


We will talk about each layer in detail shortly!

- What kind of assumptions do you think we need?
- The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.
- E.g., edges, corners, contours, object parts, ...
- Let's discuss a new layer: Convolution layer

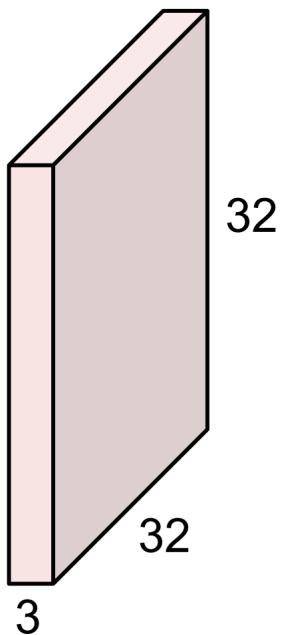
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image



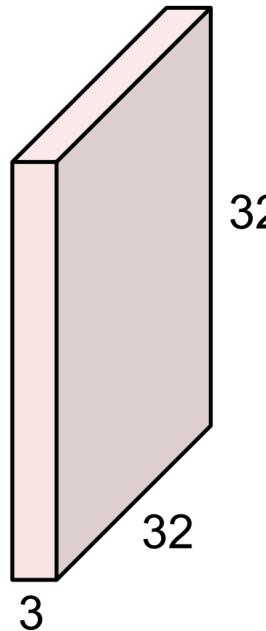
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

32x32x3 image



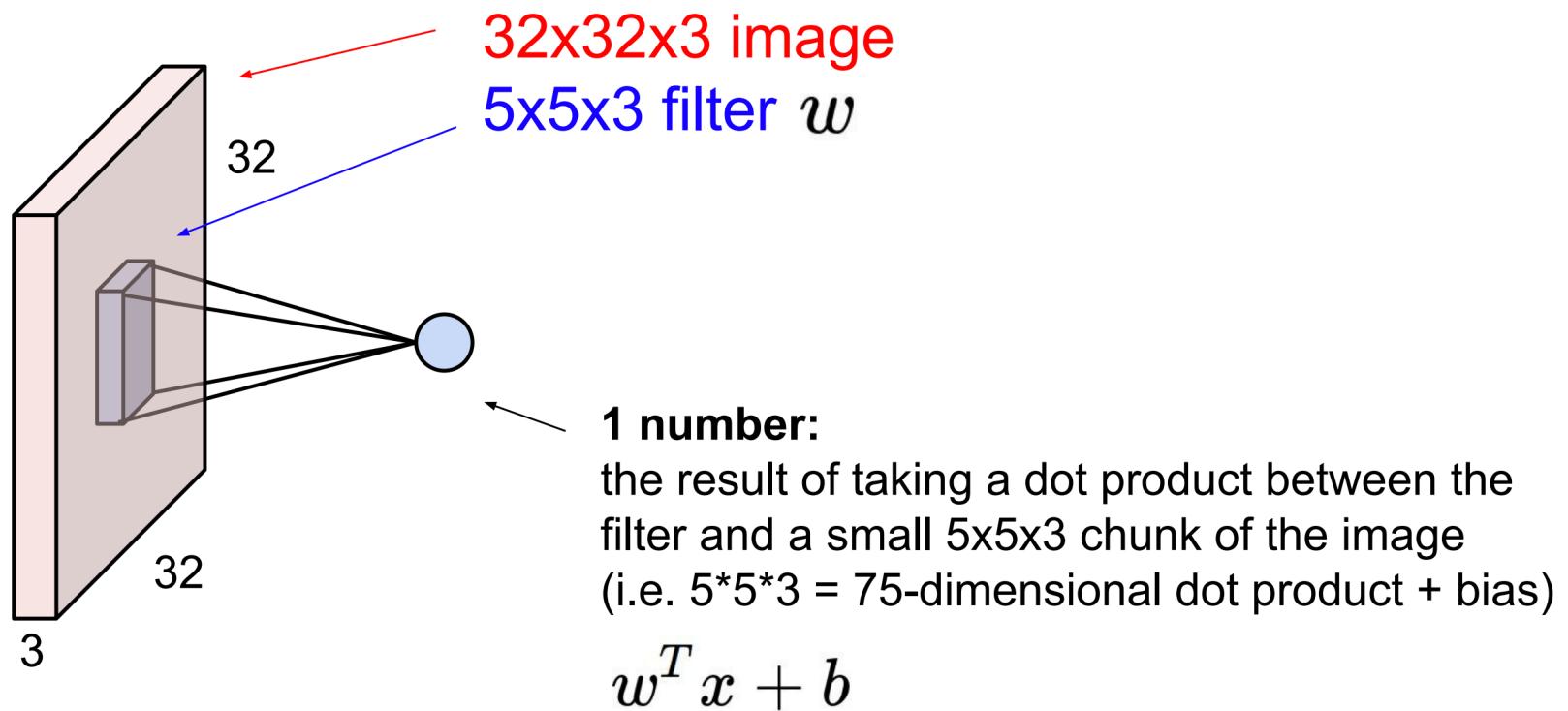
5x5x3 filter



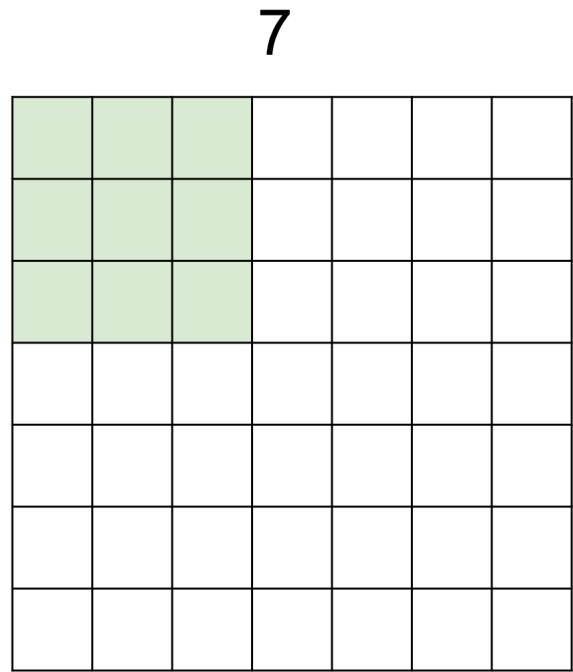
Filters always extend the full depth of the input volume

Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

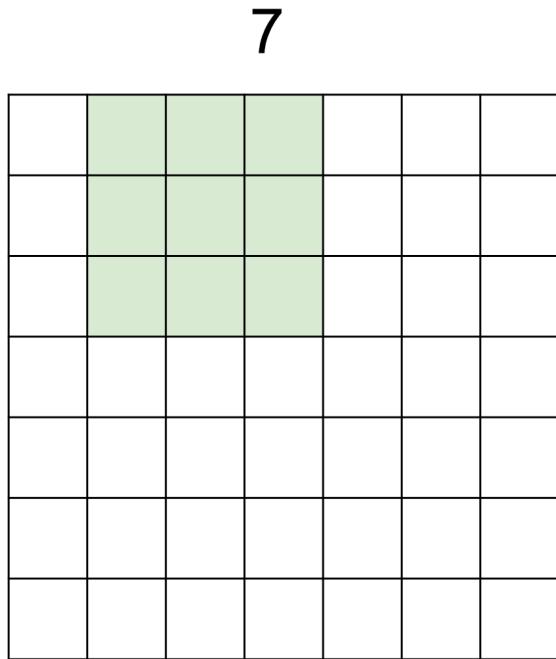


A closer look at spatial dimensions:



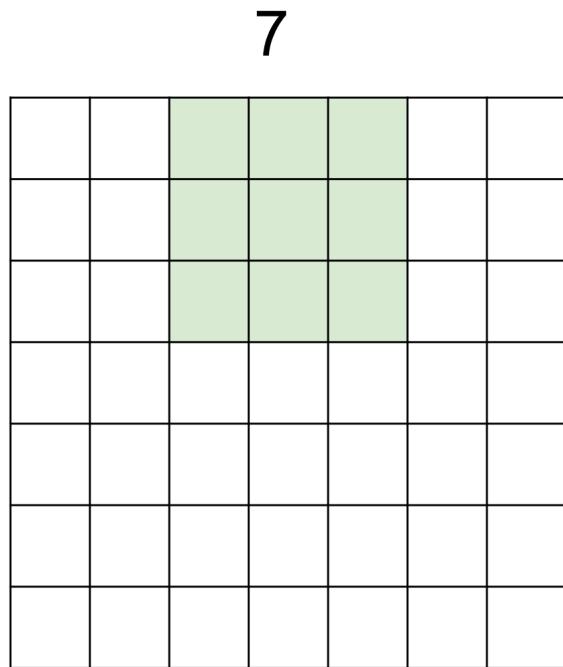
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



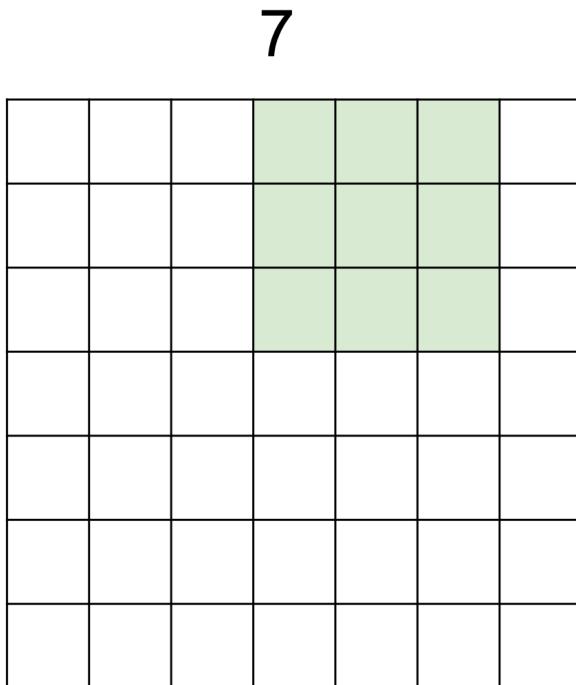
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



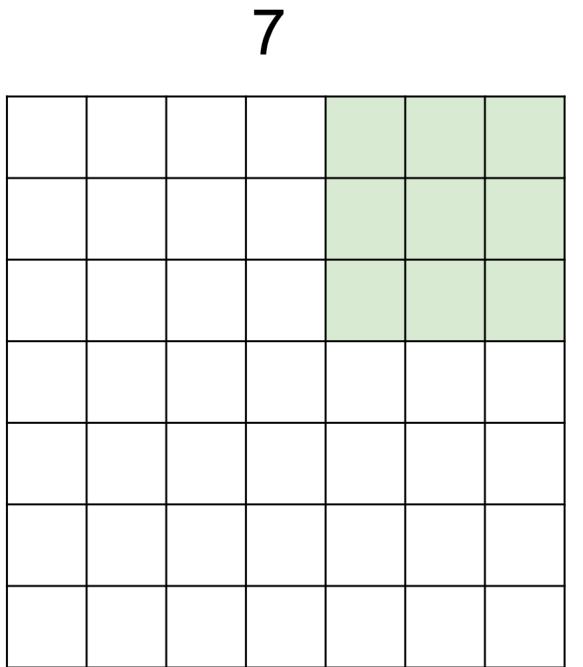
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

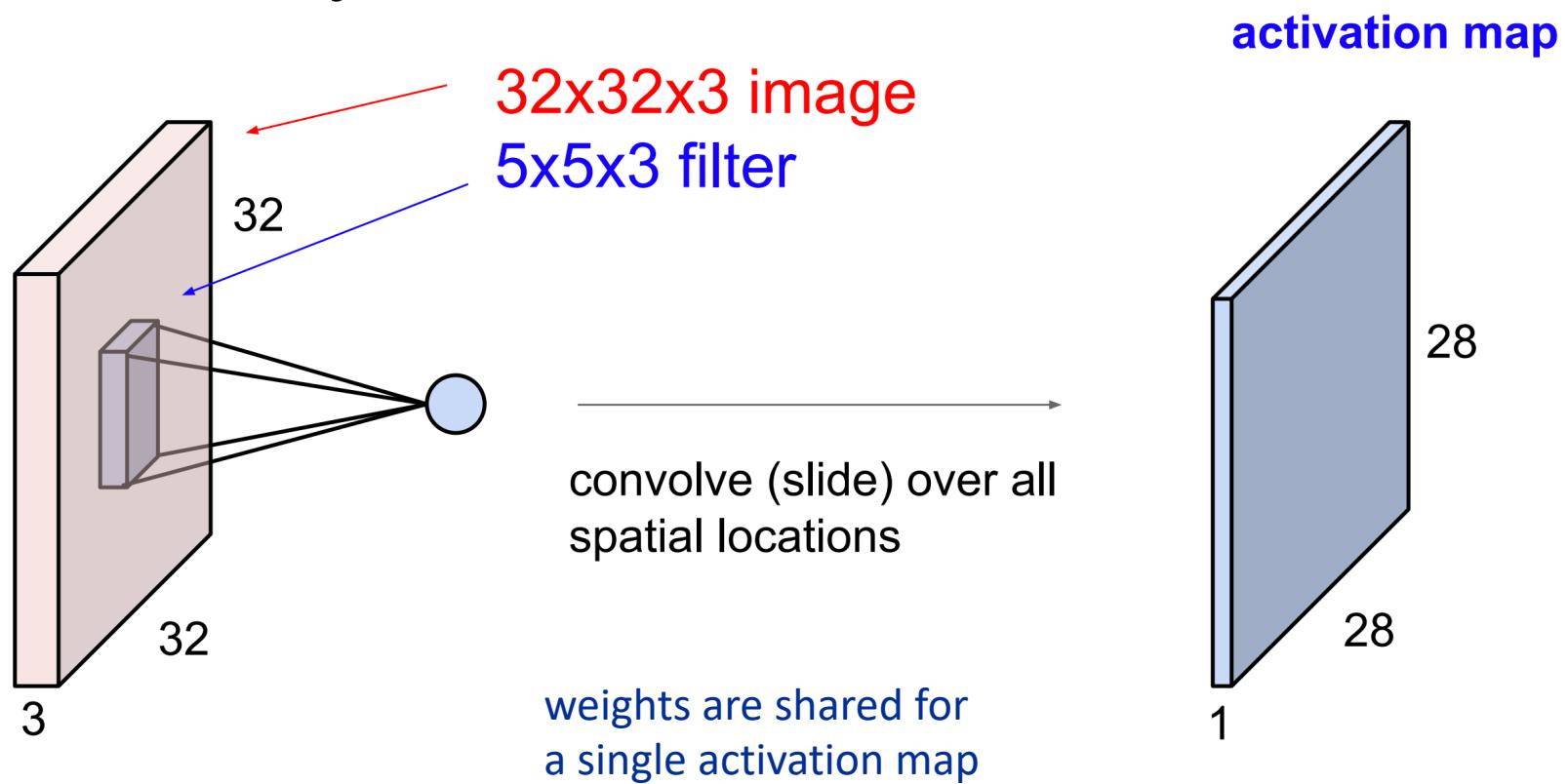
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

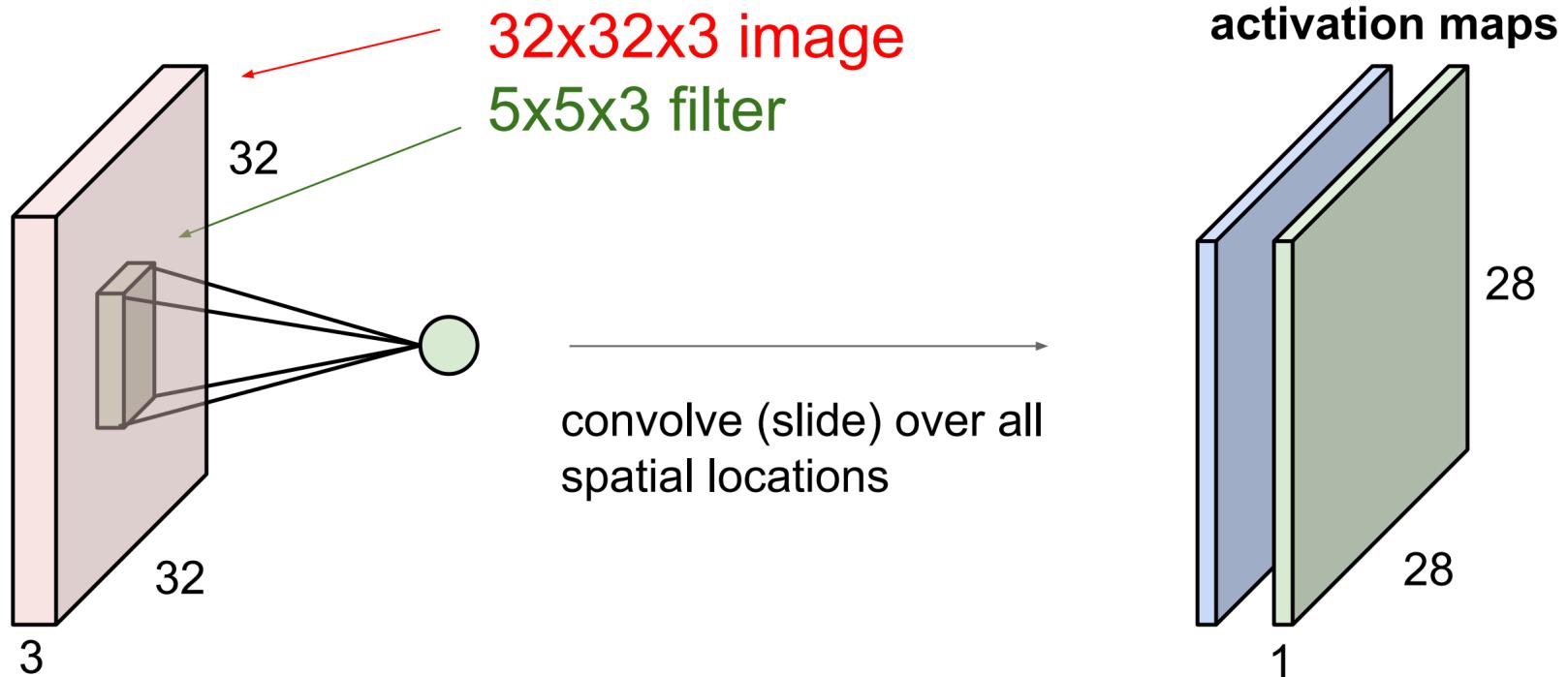
=> 5x5 output

Convolution Layer



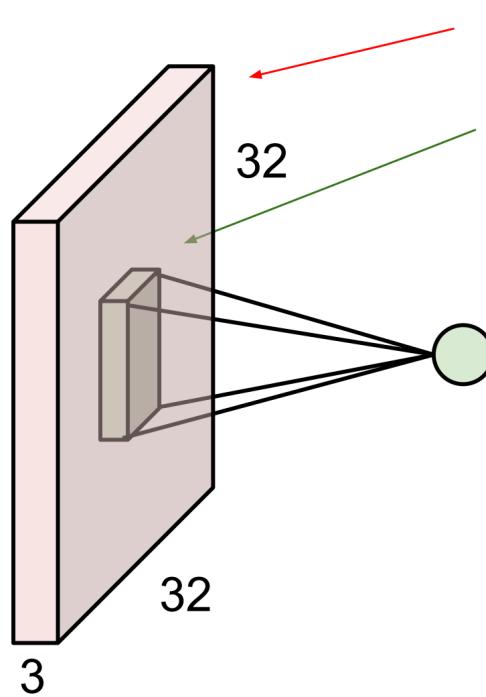
Convolution Layer

consider a second, green filter



Convolution Layer

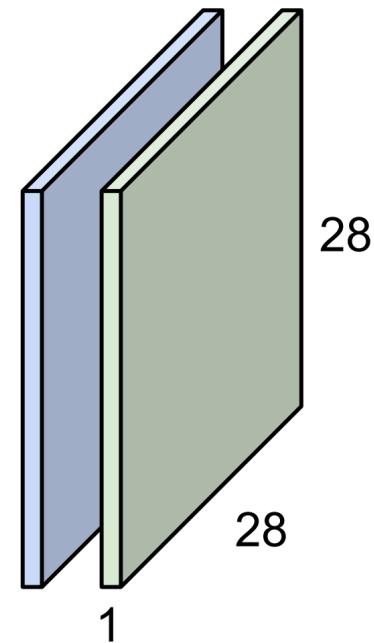
consider a second, green filter



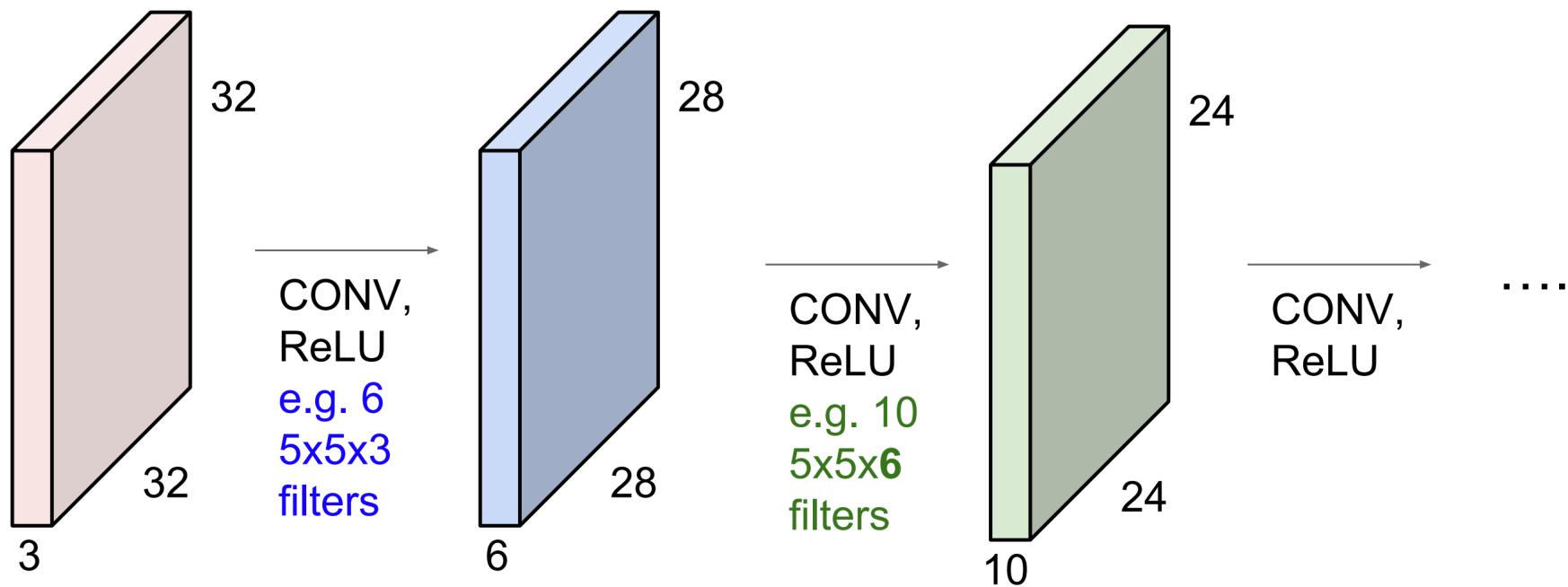
32x32x3 image
5x5x3 filter

convolve (slide) over all
spatial locations

activation maps (also called as
channels)

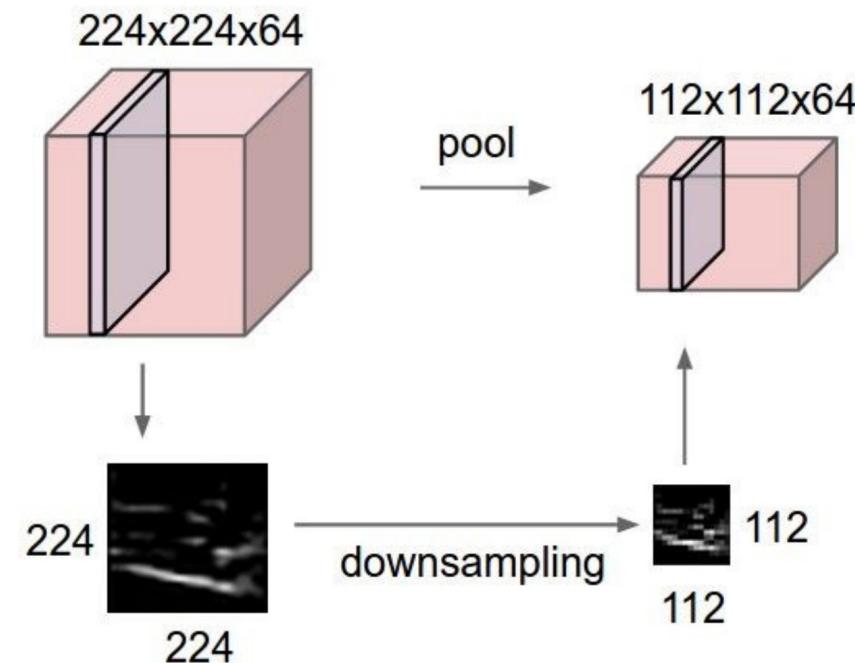


Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

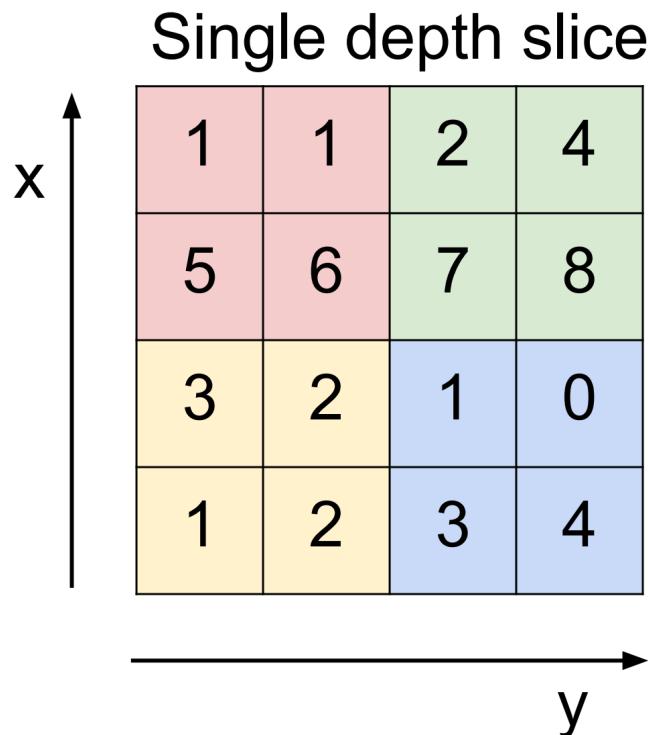


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

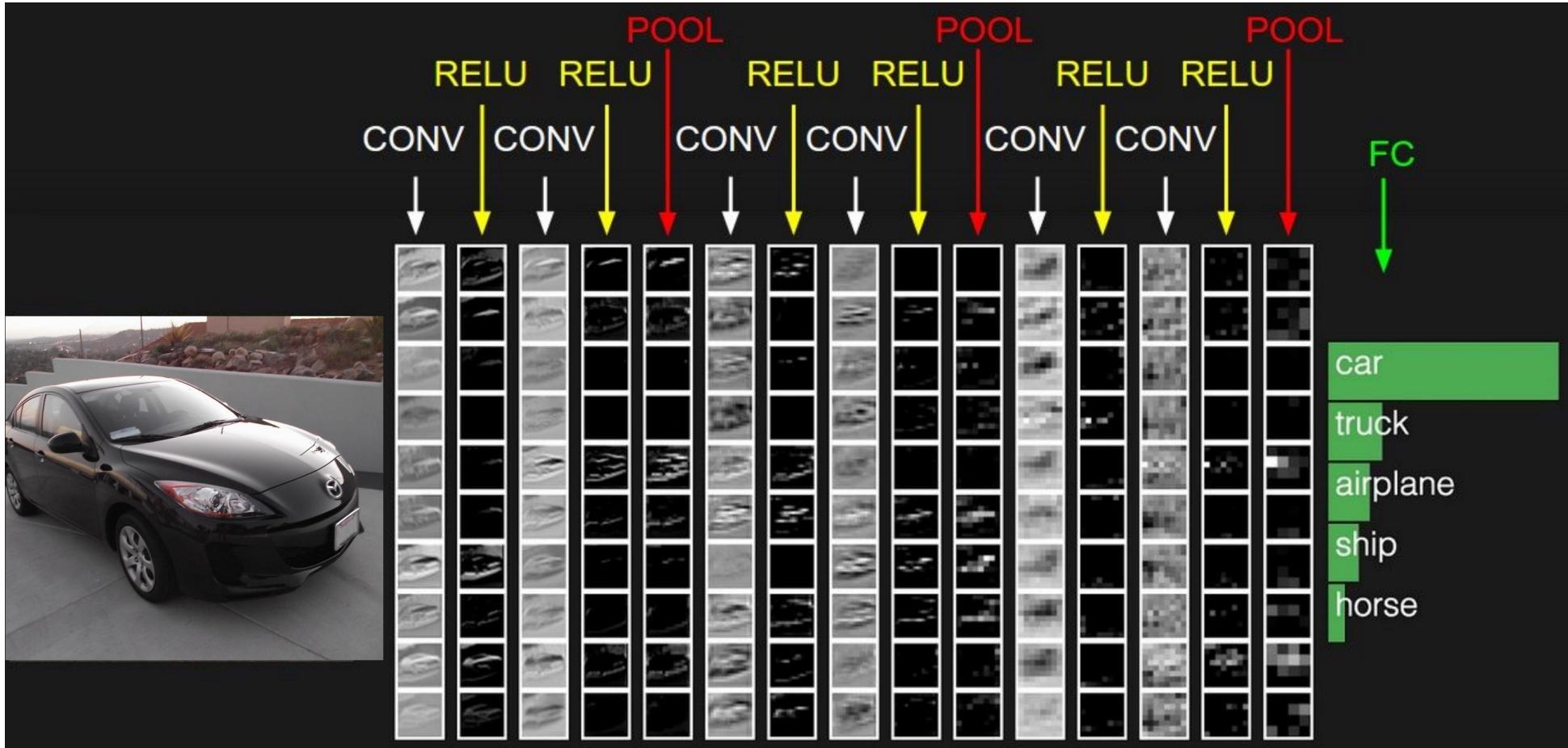


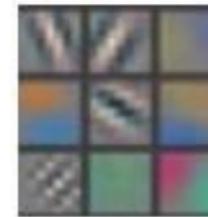
MAX POOLING



max pool with 2x2 filters
and stride 2

6	8
3	4

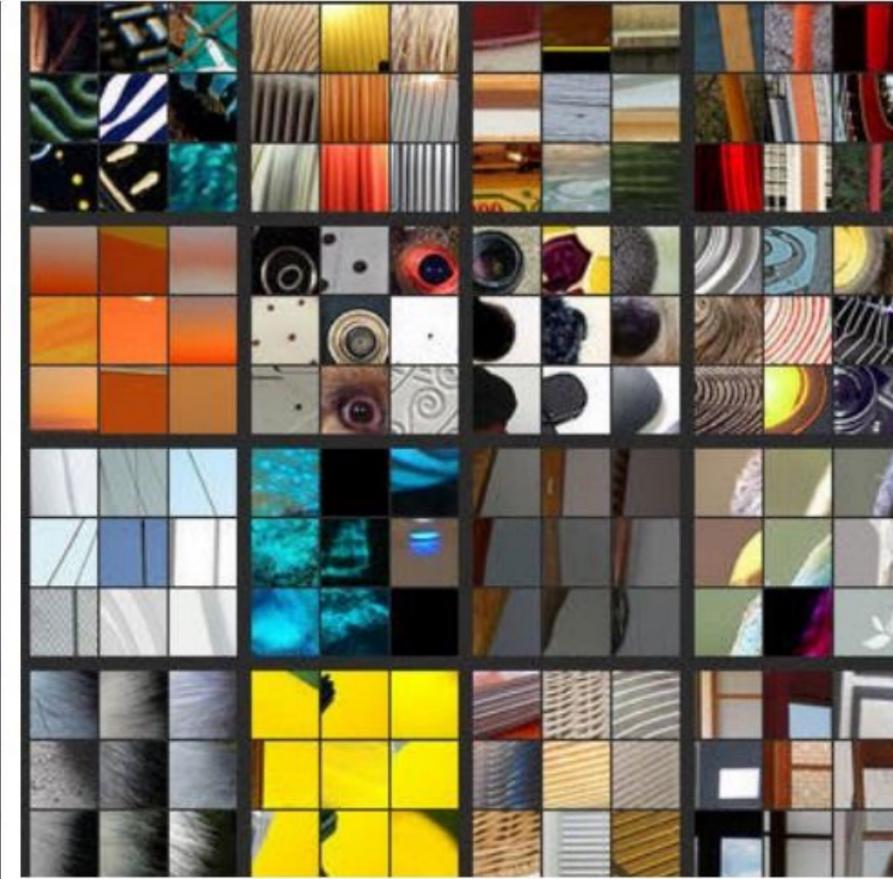
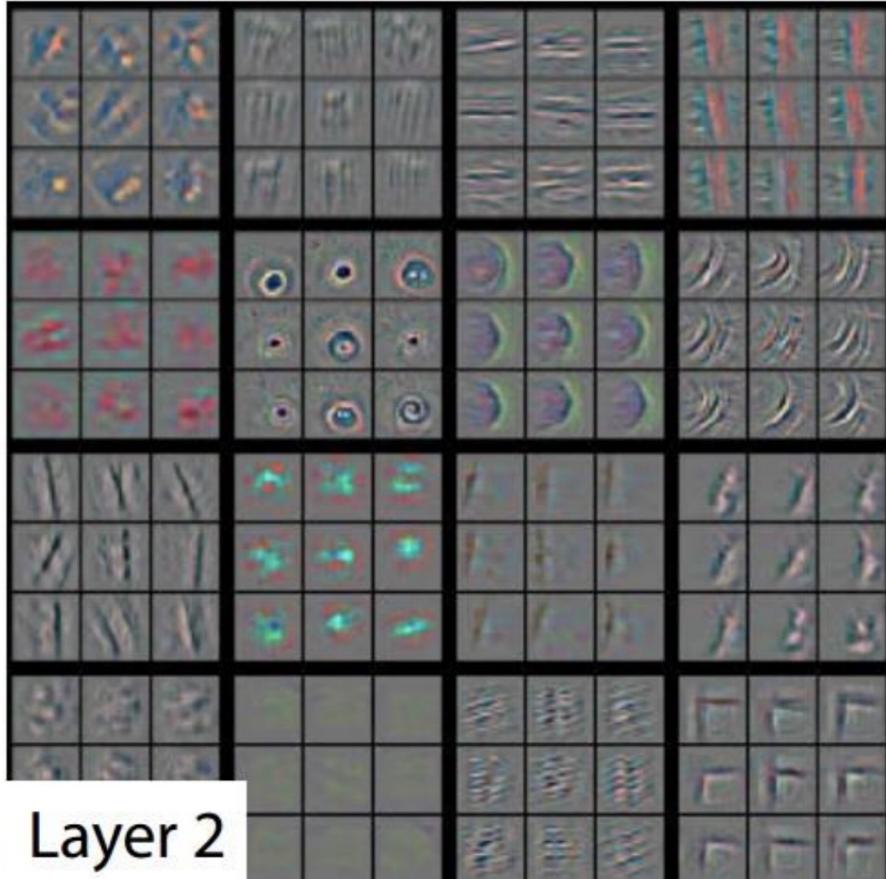


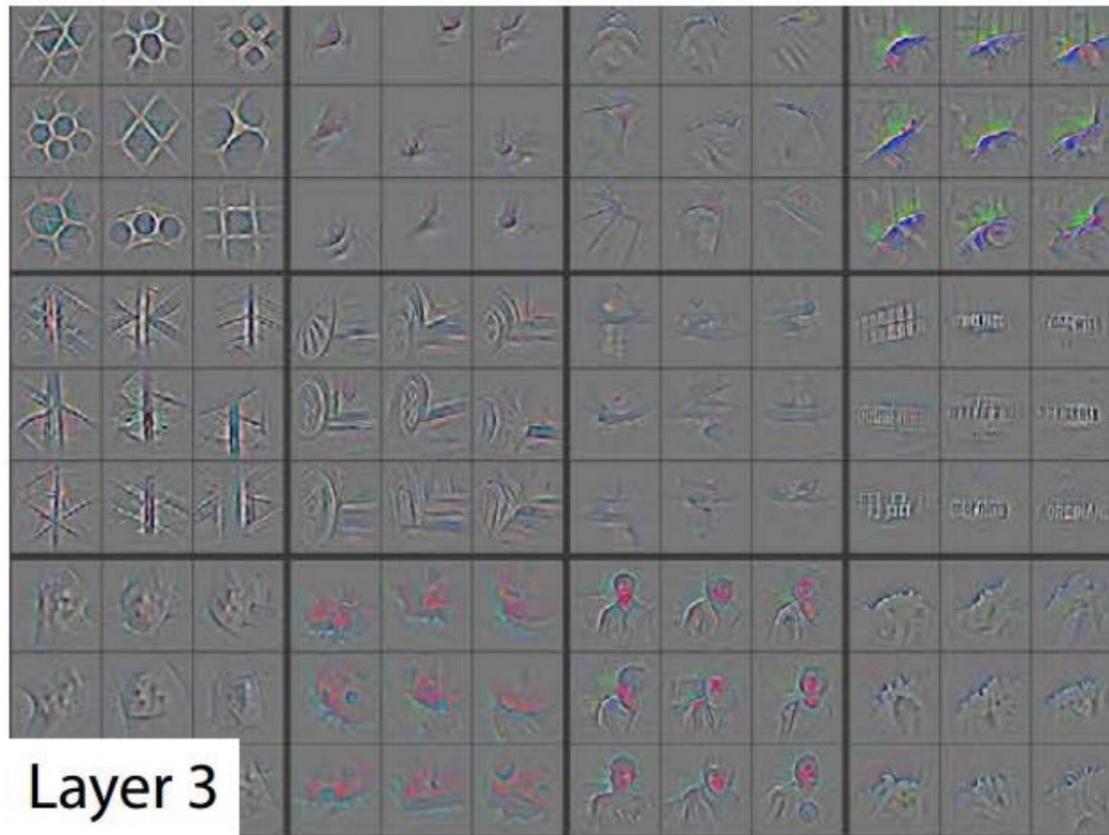


Layer 1



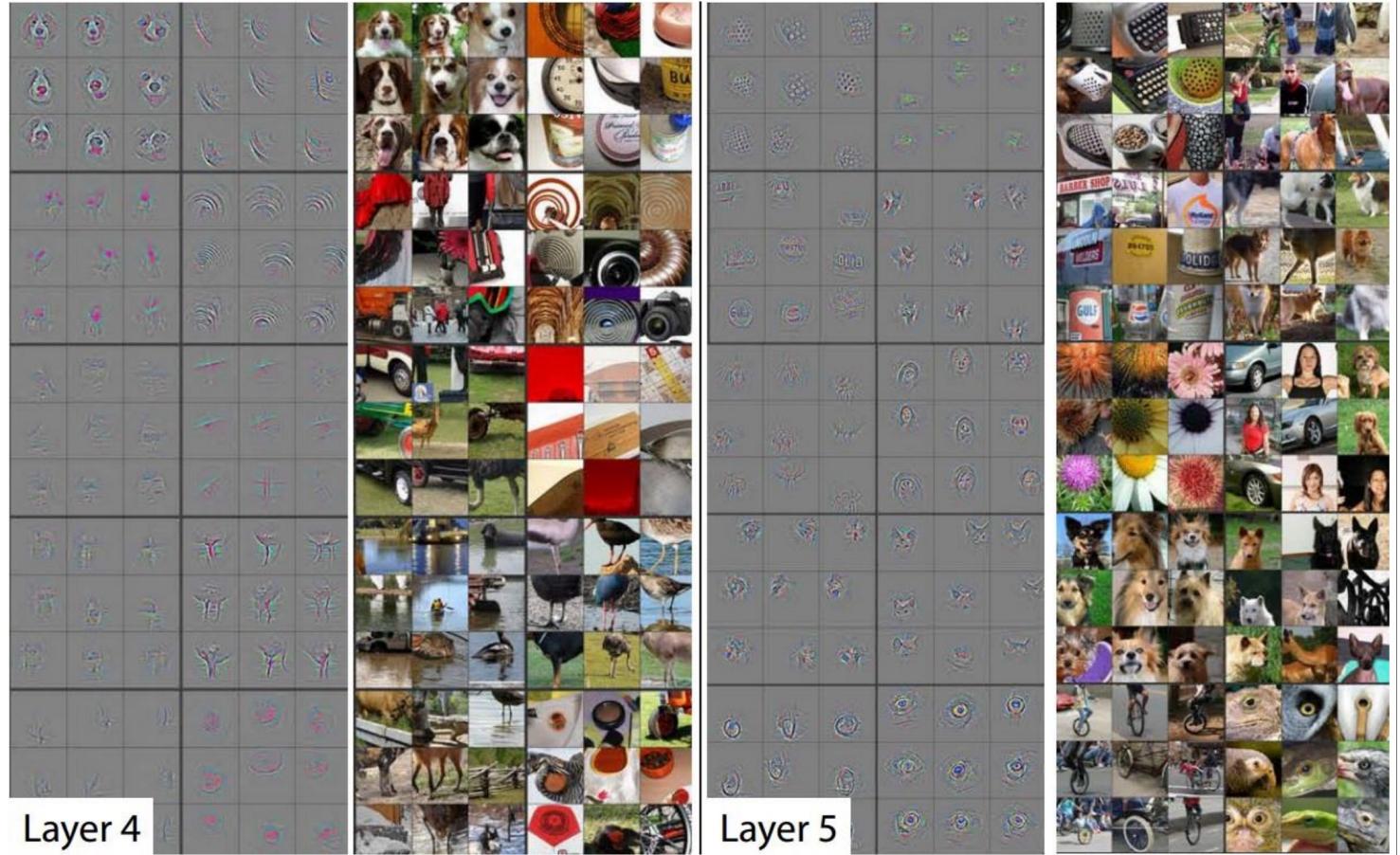
Increasingly complex features at each level
Starting from generic ones (lines, edges, colours etc.) at lower layers to specific ones at the higher layers!





Layer 3





Sequence based Neural Networks

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei,
2015]

No errors



A white teddy bear sitting in the grass



A man riding a wave on top of a surfboard

Minor errors



A man in a baseball uniform throwing a ball



A cat sitting on a suitcase on the floor

Somewhat related



A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
<https://pixabay.com/en/luggage-antique-cat-1643010/>
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>
<https://pixabay.com/en/handstand-lake-meditation-496008/>
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [Neuraltalk2](#)

Sentiment Classification

"I love this movie.
I've seen it many times
and it's still awesome."



"This movie is bad.
I don't like it at all.
It's terrible."



DETECT LANGUAGE

HINDI

ENGLISH

SPANISH



HINDI

ENGLISH

SPANISH



What is there for dinner today



आज खाने के लिए क्या है



aaj khaane ke lie kya hai



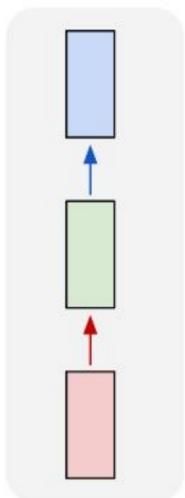
30/5000



Machine Translation

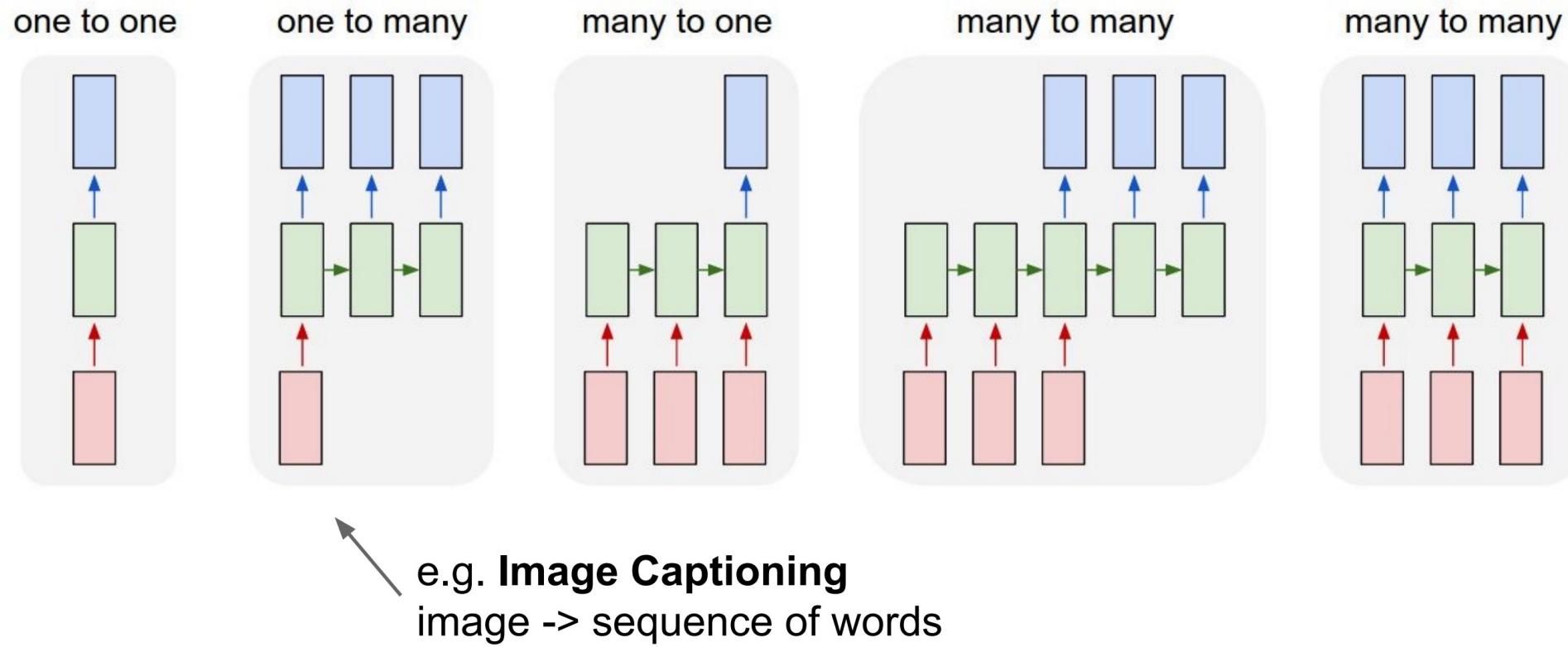
“Vanilla” Neural Network

one to one



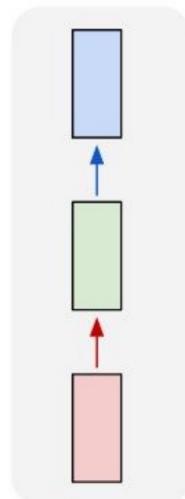
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences

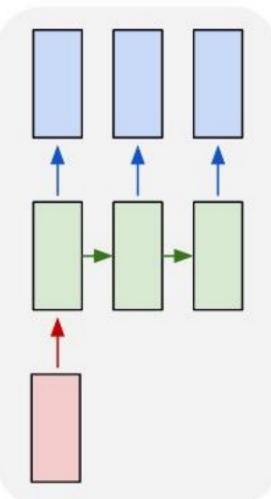


Recurrent Neural Networks: Process Sequences

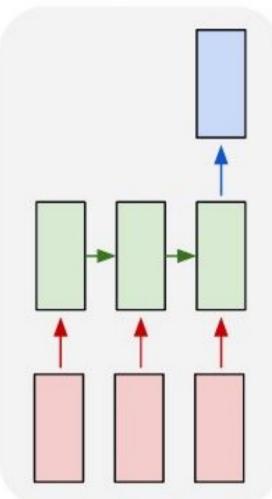
one to one



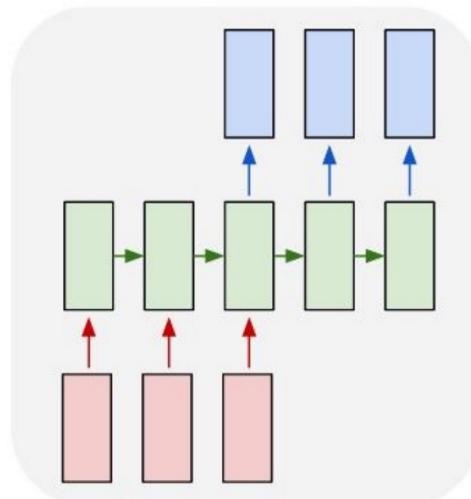
one to many



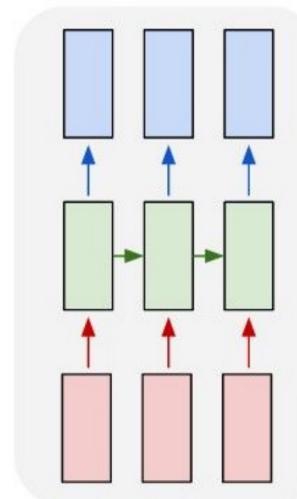
many to one



many to many



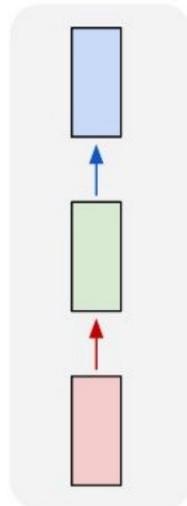
many to many



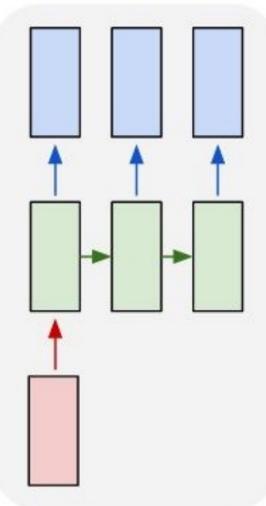
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Neural Networks: Process Sequences

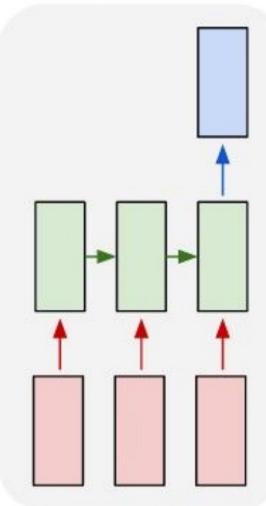
one to one



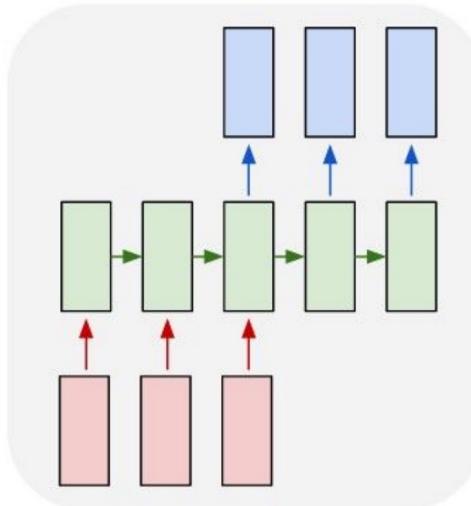
one to many



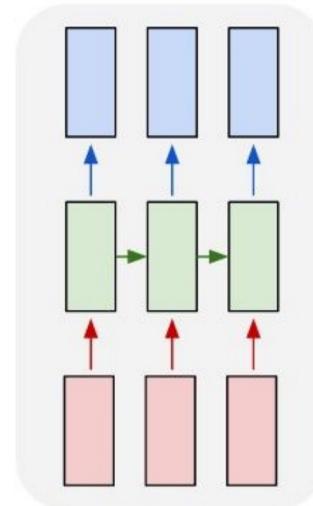
many to one



many to many

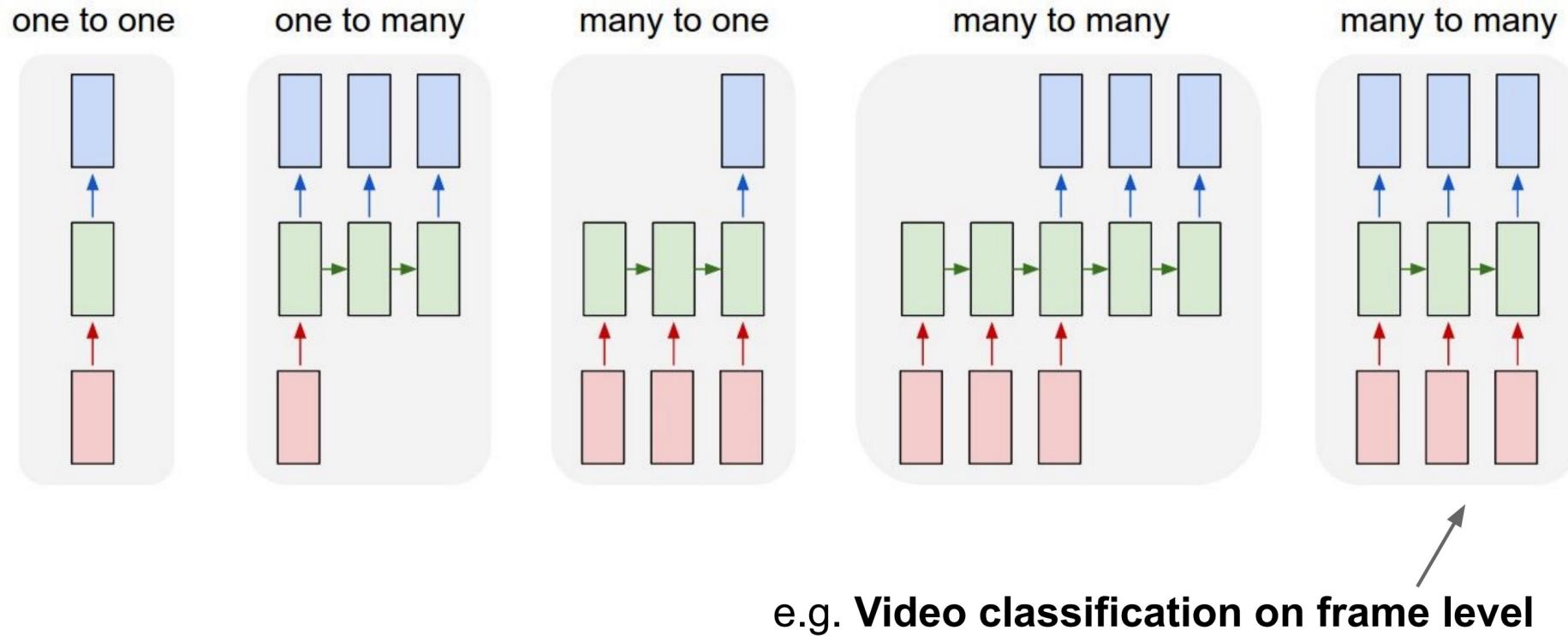


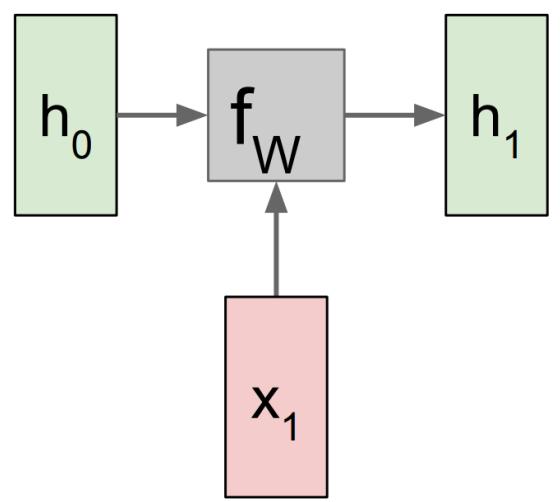
many to many

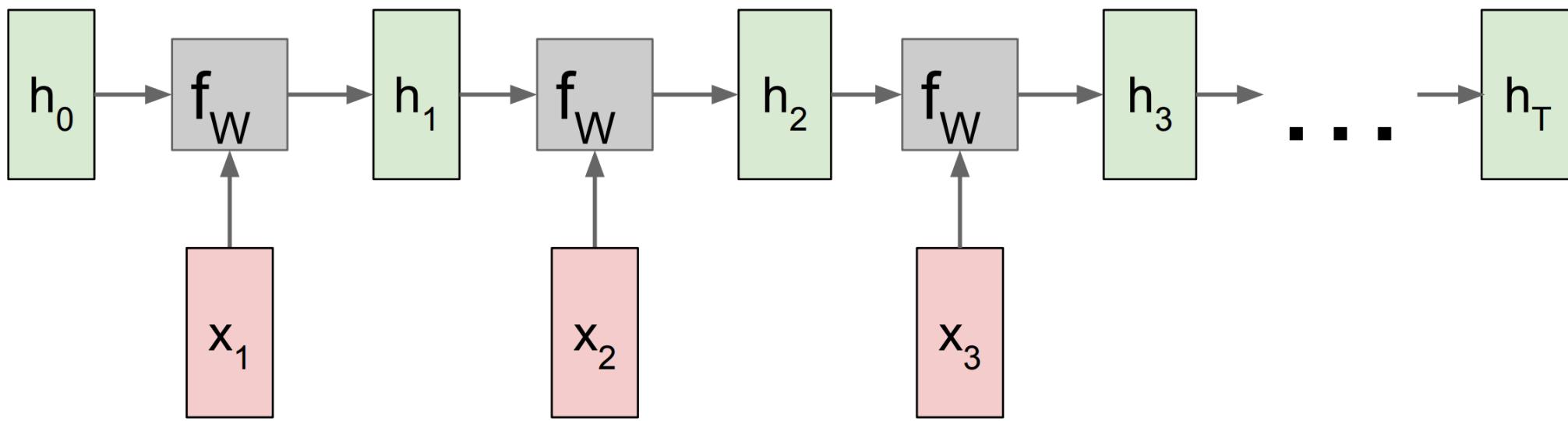


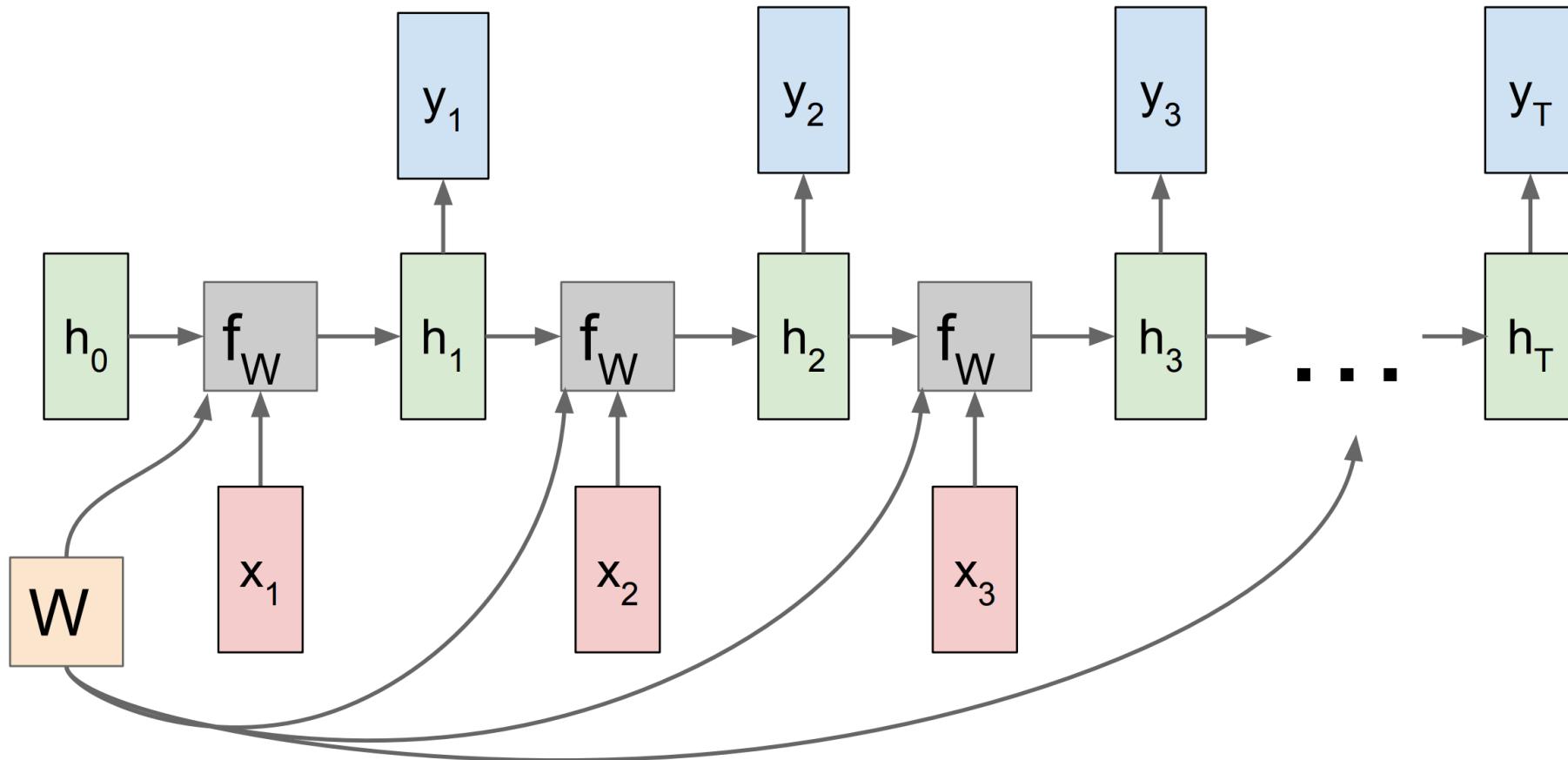
e.g. **Machine Translation**
seq of words -> seq of words

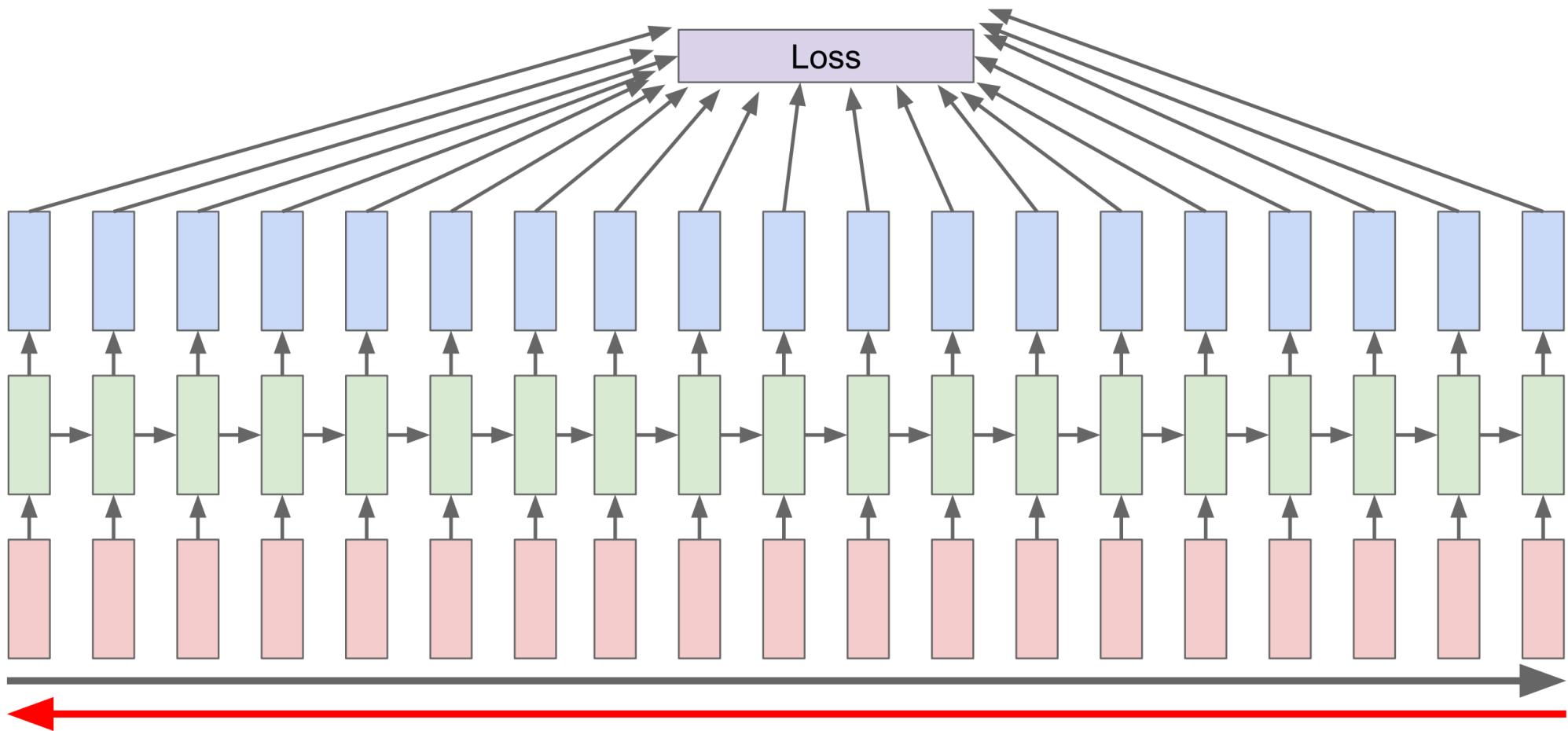
Recurrent Neural Networks: Process Sequences











Forward through entire sequence to compute loss, then
backward through entire sequence to compute gradient

Advanced Sequence Models

- Vanilla RNNs have almost fallen out of favor because of multiple issues with their training procedure
 - For e.g. [Vanishing Gradients](#)
- Multiple advanced techniques overcome these issues in different forms. Some of them are:
 - [LSTMs, GRUs](#)
 - Attention models

Graph Neural Networks

Success of deep learning on grid/sequence structured data...

Images

Language

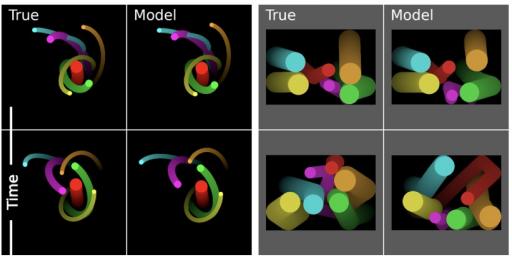
How about more complex data



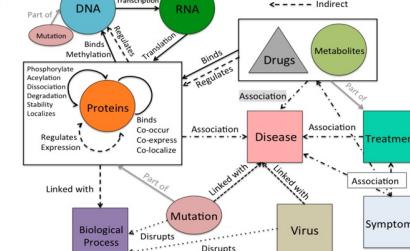
(Instagram:@kyokofukada_official)

之	の	無	な	キ	レ	矛	ほ	こ
盾	たて	不	ざ	ル	レ	也	な	ト
如	い	レ	ル	コ	ト	也	ナ	ト
何	ん	陷	と	キ	ト	也	ナ	ト

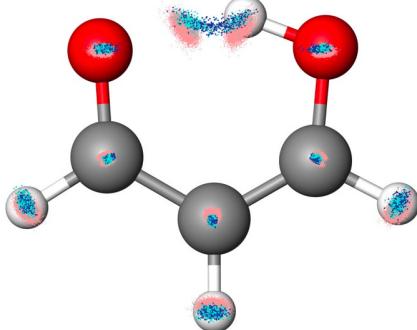
(Wikibooks)



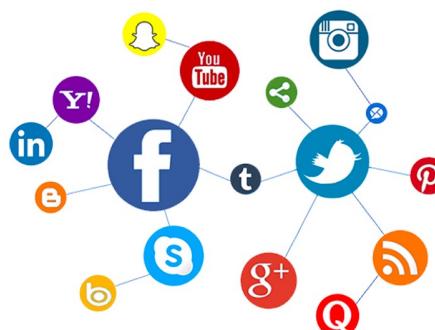
Intuitive Physics (Battaglia et al. 2016)



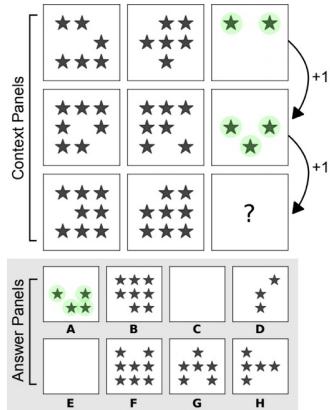
Biomedicine (Zitnik et al. 2018)



Drug design (Duvenaud et al. 2015)



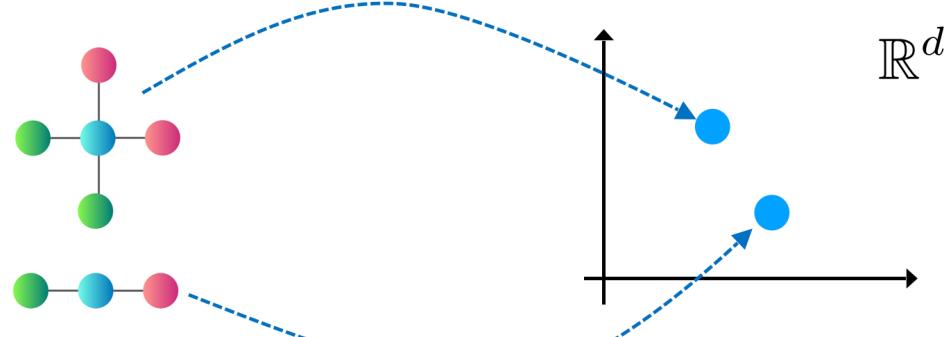
Social networks and social media graphs
(Hamilton et al. 2017)



Human IQ test (Barrett et al. 2018)

Complex graph data

Input: Graph with node features



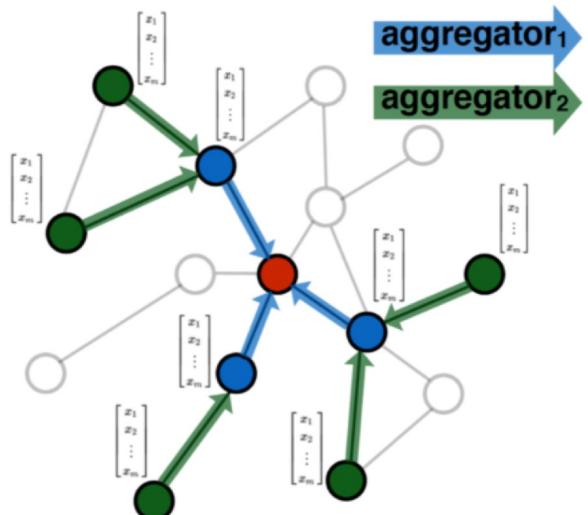
Embedding space

$$\mathbb{R}^d$$

Task: Node/Graph classification

Goal: Learn node/graph embeddings that capture graph structure

GNNs learn node representations with **Neighborhood Aggregation**



Aggregate from neighbors

$$a_v^{(k)} = \text{AGGREGATE}^{(k)} \left(\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \} \right)$$

Combine with current node

$$h_v^{(k)} = \text{COMBINE}^{(k)} \left(h_v^{(k-1)}, a_v^{(k)} \right)$$

...

For Graph classification use **readout to pool node embeddings**

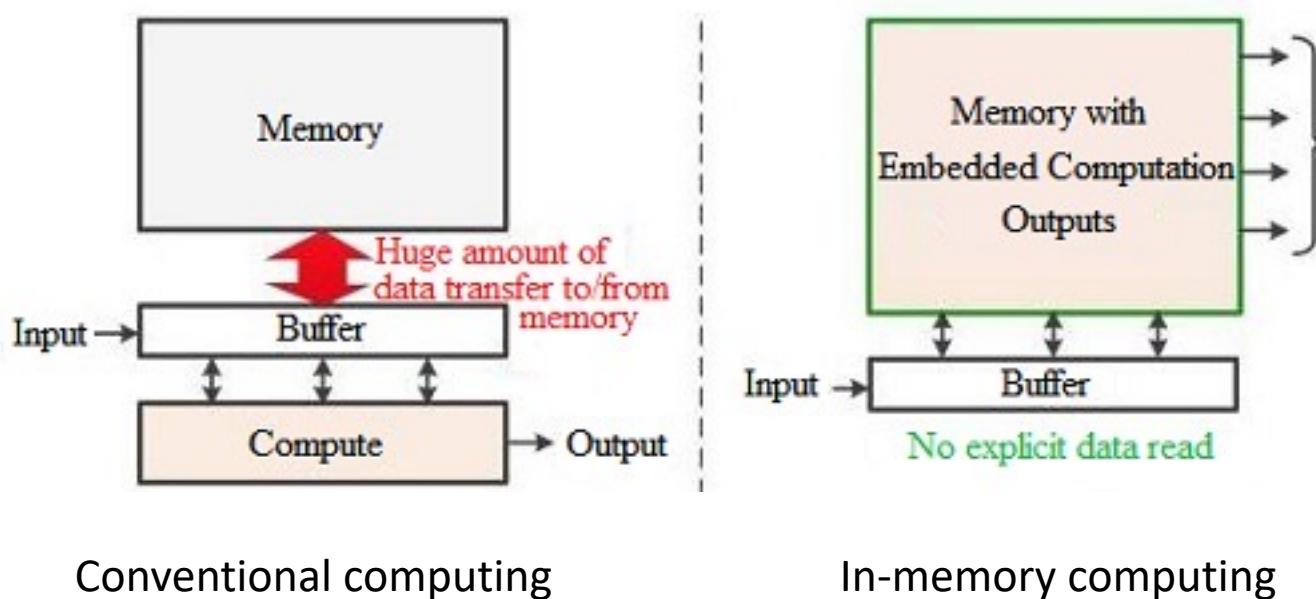
$$h_G = \text{READOUT}(\{ h_v^{(K)} \mid v \in G \})$$

K GNN iterations captures K hop network structure

Outline of Tutorial

- Exponential growth of Deep Learning and Hardware Challenges
- Introduction to Deep Learning
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- Methods to improve Reliability of ReRAMs
- Summary and Promising Directions

In-memory computing



- Memory units include simple compute units
- Computations are done inside the memory
- Cost of moving data is reduced

Comparing different memory technologies

Table 1: Emerging Non-volatile Memory Comparison

	SRAM	DRAM	STT-RAM	PCM	ReRAM
Cell Size (F ²)	>100	6-10	6-50	4-30	≤2
Multibit	1	1	1	>2	2-7
Endurance	>10 ¹⁶	>10 ¹⁶	>10 ¹⁵	10 ⁸ -10 ¹⁵	10 ⁸ -10 ¹²
Read Time (ns)	~1	~10	<10	<10	<10
Write Time (ns)	~1	~10	<10	50	<10
Write Energy (J/bit)	~ 10 ⁻¹⁵	~ 10 ⁻¹⁴	~ 10 ⁻¹³	~ 10 ⁻¹¹	~ 10 ⁻¹³

Source: [3–6]. Note: F represents the feature size.

- In-memory computing requires computing near or inside memory
- SRAM is area inefficient
- DRAM is volatile
- ReRAM is a good candidate

Non-volatile memory use

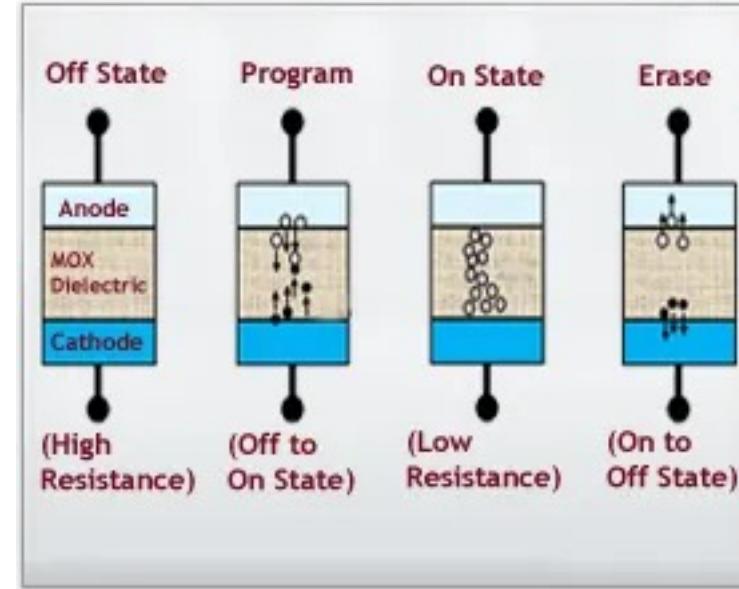
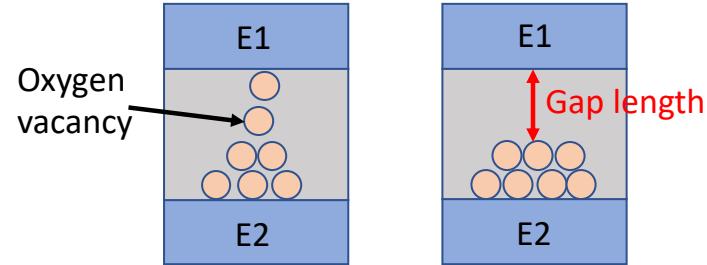
Table 2: An Overview of In-memory Processing Designs

Works	Types	Locations	Design Levels	Functions	Applications
Guo <i>et al.</i> [7]	STT-RAM	Cache	Circuit; System	Logic; Arithmetic	Generic
AC-DIMM [8]		Main Memory	Circuit; System	Associative	Generic
Kang <i>et al.</i> [9]		-	Circuit	Logic	Bitmap
STT-CiM [10]		Scratchpad	Circuit; System	Logic; Addition; Vector	Generic
HielM [11]		-	Circuit; System	Logic	Encryption, Database
Pan <i>et al.</i> [12]		Co-processor	Circuit; System	Logic	Binary CNN
Cassinerio <i>et al.</i> [13]	PCM	-	Device	Logic	-
Wright <i>et al.</i> [14, 15]		-	Device	Arithmetic	-
Hosseini <i>et al.</i> [16]		-	Device	Arithmetic	-
Pinatubo [17]		Main Memory	Circuit; System	Logic	Generic
Burr <i>et al.</i> [18, 19]		-	Circuit	MVM	DNN
Sebastian <i>et al.</i> [20]		-	Circuit	MVM	Unsupervised Learning
Le <i>et al.</i> [21, 22]		-	Circuit	MVM	Transfer Learning
MAGIC [23]	ReRAM	Co-processor	Circuit	Logic; Arithmetic	Adder
Bojnordi <i>et al.</i> [24]		Co-processor	System	MVM	Boltzmann machine
ISAAC [25]		Co-processor	System	MVM	CNN
PipeLayer [26]		Co-processor	System	MVM	CNN
AtomLayer [27]		Co-processor	System	MVM	CNN
GraphR [28]		Co-processor	System	MVM	Graph

Note: MVM – Matrix-Vector Multiplication; DNN – Deep Neural Network.

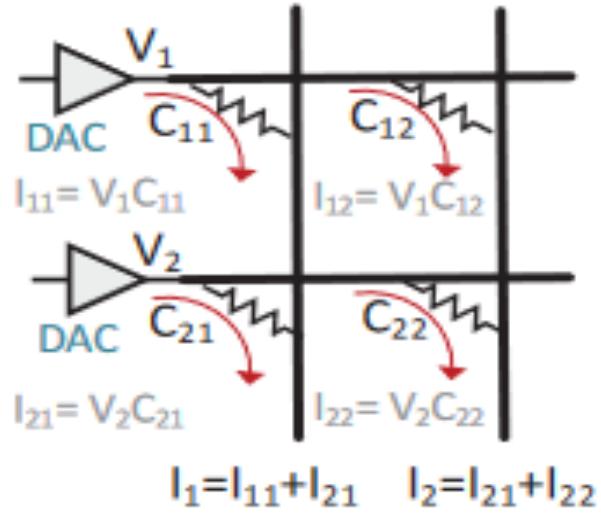
- PCM, STT-RAM, ReRAM have been used for in-memory computing
- We will focus ReRAMs for this presentation

ReRAM cell



- Resistance of ReRAM cell modulated by voltage
- Oxygen vacancies form a bridge between the two electrodes
 - ON state
- No bridge = OFF state

ReRAM for Matrix vector multiplication



$$\begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix} \times \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}$$

- Resistors are natural multipliers ($V = I * R$)
- Crossbar structure enables high-throughput matrix-vector multiplications
- Perfect for ML applications

Convolution layer CNN

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

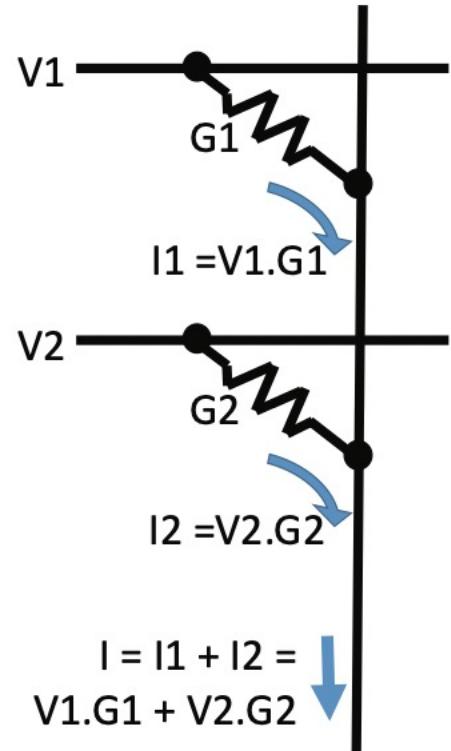
=

6		

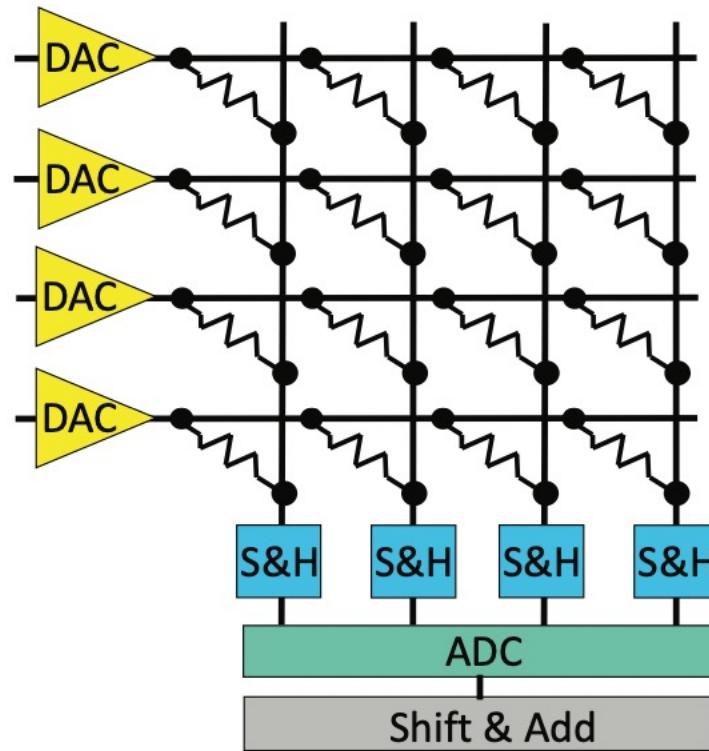
$$\begin{aligned} & 7 \times 1 + 4 \times 1 + 3 \times 1 + \\ & 2 \times 0 + 5 \times 0 + 3 \times 0 + \\ & 3 \times -1 + 3 \times -1 + 2 \times -1 \\ & = 6 \end{aligned}$$

- Convolution and Fully-connect layers are dominated by matrix-vector multiplications
- Can be implemented using ReRAM crossbars

ReRAM crossbar



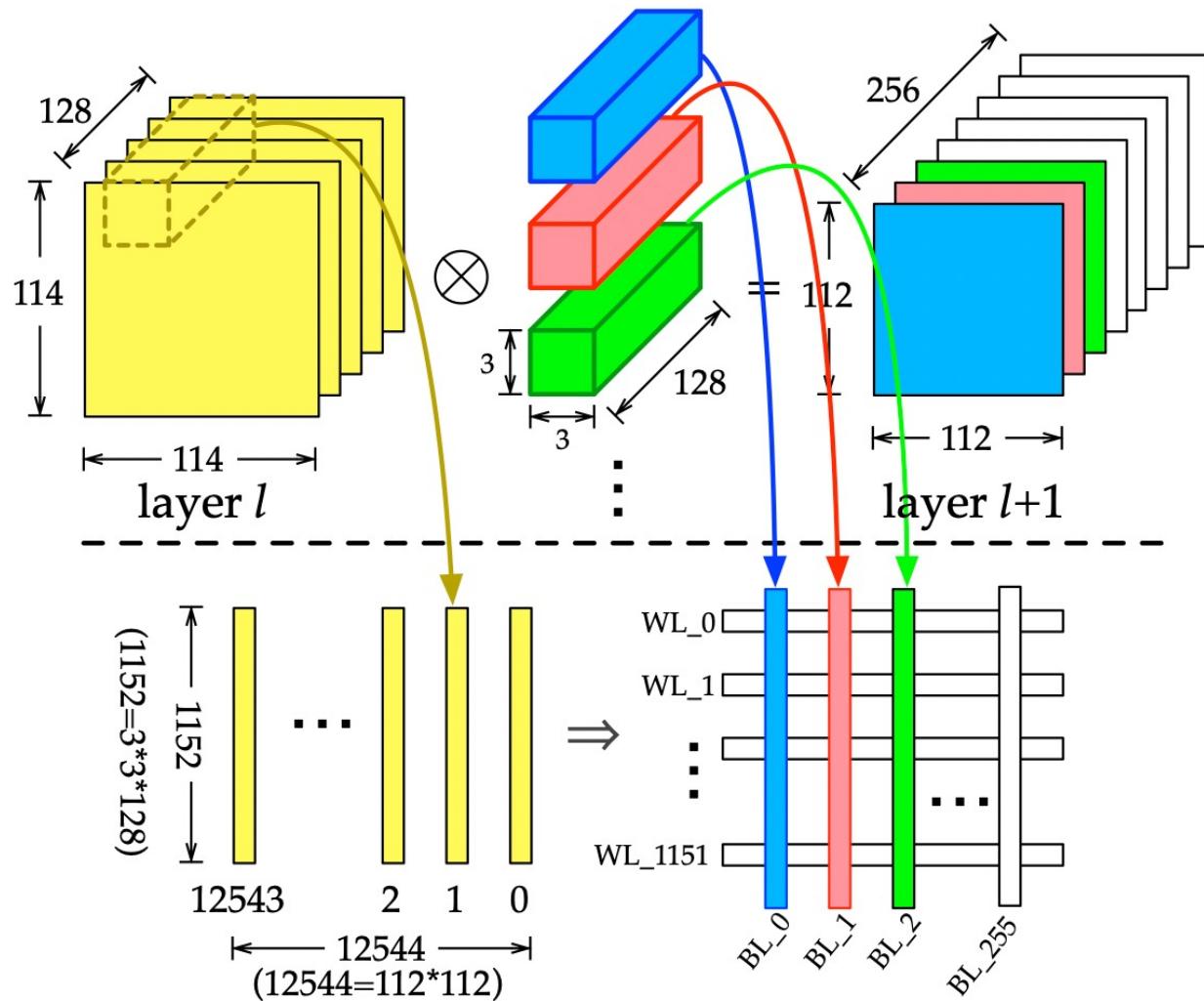
(a) Multiply-Accumulate operation



(b) Vector-Matrix Multiplier

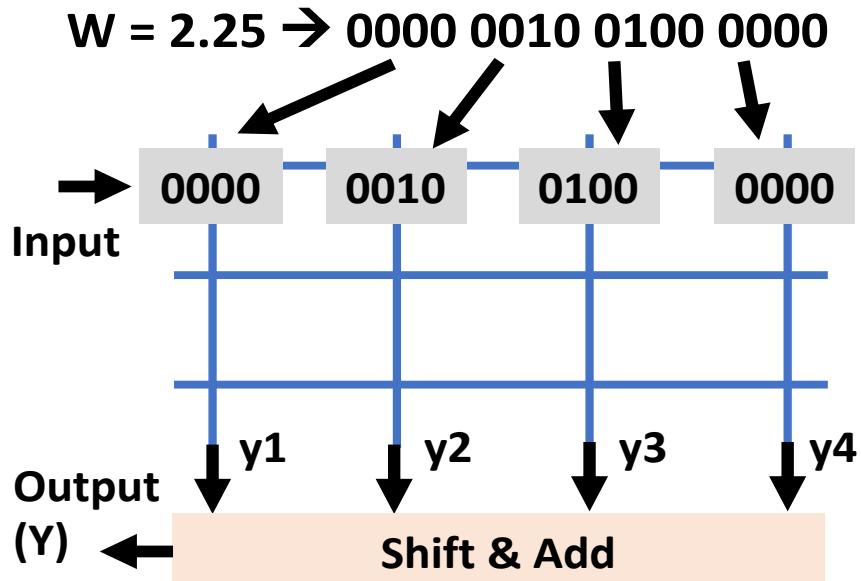
- ReRAM crossbar includes multiple peripherals, e.g., ADC, DAC, etc.

Mapping weights to ReRAMs



- Mapping weights of convolution layer to ReRAM crossbar
- Input images are applied as input voltage to wordlines
- Output is obtained at the bitlines

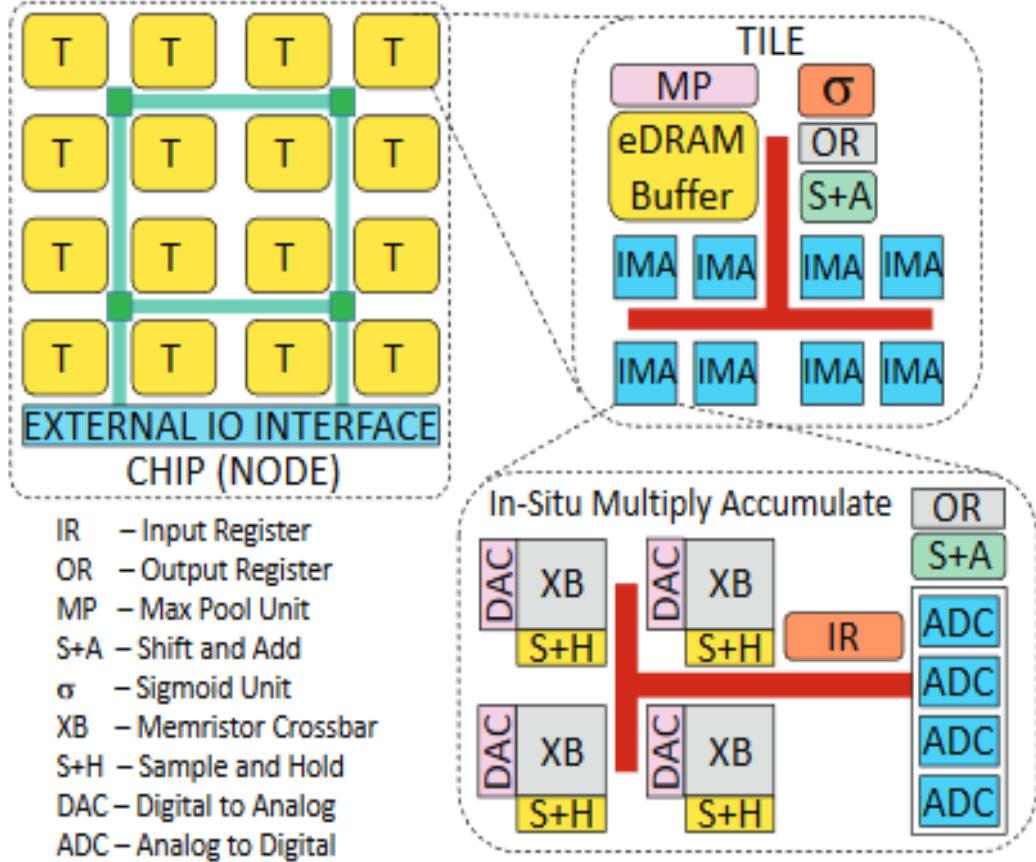
Mapping weights to ReRAMs



$$Y = \sum_{i=1}^4 16^{4-i} * y_i$$

- All bits of a weight are not mapped to same cell due to ADC size concerns
- Bits are distributed across multiple cells
- Output is obtained by accumulating partial outputs

ReRAM architecture



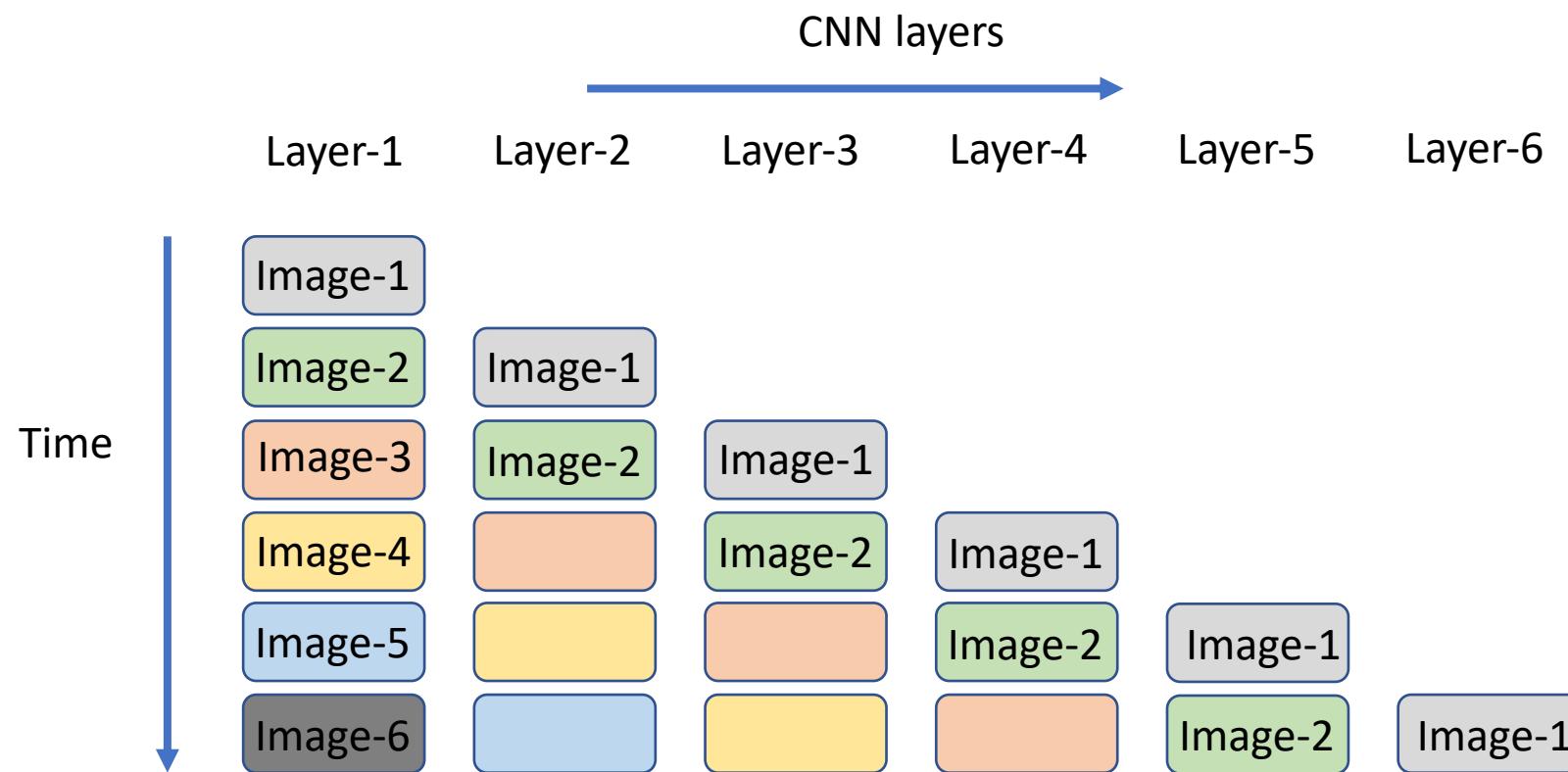
- Multiple ReRAM crossbars necessary for large CNNs
- Crossbars are grouped in to IMAs, multiple IMAs grouped in to tiles
- Peripherals include ADC, DAC, on-chip memory, etc.

Example ReRAM architecture

- ADC and Router introduce the most power and area
- Crossbar itself is area and power efficient
- ISAAC architecture has ~16k crossbars of shape 128*128

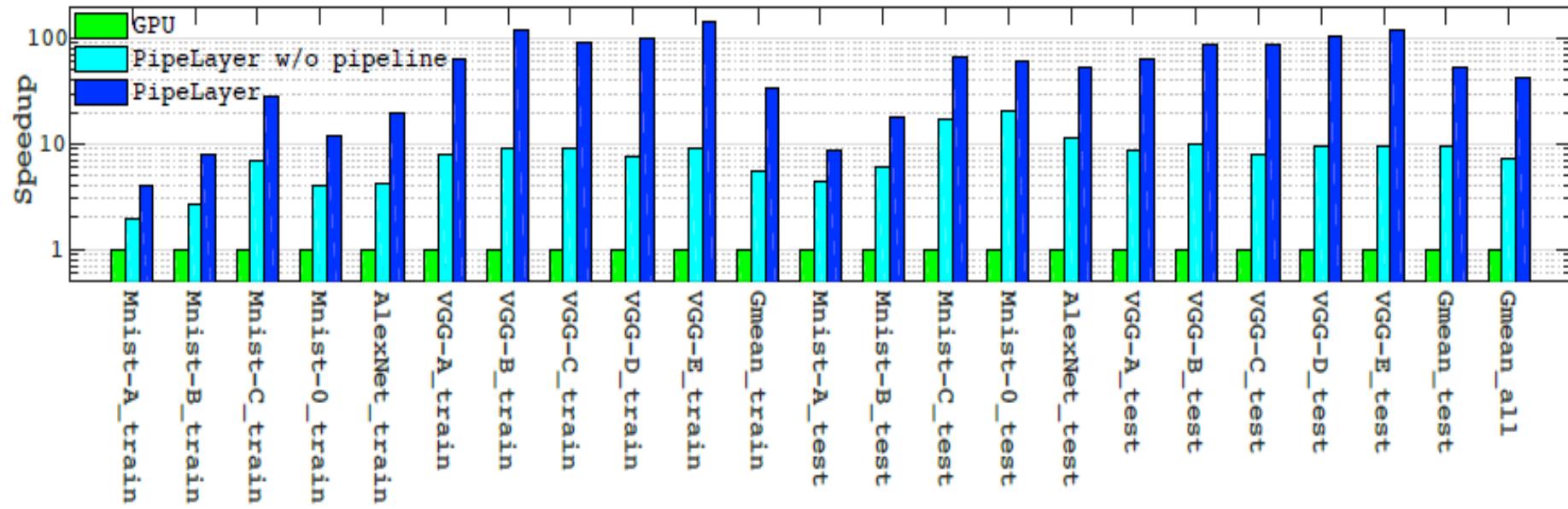
ISAAC Tile at 1.2 GHz, 0.37 mm ²				
Component	Params	Spec	Power	Area (mm ²)
eDRAM Buffer	size num_banks bus_width	64KB 2 256 b	20.7 mW	0.083
eDRAM -to-IMA bus	num_wire	384	7 mW	0.090
Router	flit size num_port	32 8	42 mW	0.151 (shared by 4 tiles)
Sigmoid S+A MaxPool OR	number number number size	2 1 1 3 KB	0.52 mW 0.05 mW 0.4 mW 1.68 mW	0.0006 0.00006 0.00024 0.0032
Total			40.9 mW	0.215 mm²
IMA properties (12 IMAs per tile)				
ADC	resolution frequency number	8 bits 1.2 GSps 8	16 mW	0.0096
DAC	resolution number	1 bit 8 × 128	4 mW	0.00017
S+H	number	8 × 128	10 uW	0.00004
Memristor array	number size bits per cell	8 128 × 128 2	2.4 mW	0.0002
S+A IR OR	number size size	4 2 KB 256 B	0.2 mW 1.24 mW 0.23 mW	0.00024 0.0021 0.00077
IMA Total	number	12	289 mW	0.157 mm²
1 Tile Total			330 mW	0.372 mm²
168 Tile Total			55.4 W	62.5 mm²
Hyper Tr	links/freq link bw	4/1.6GHz 6.4 GB/s	10.4 W	22.88
Chip Total			65.8 W	85.4 mm²

ReRAM execution pipeline



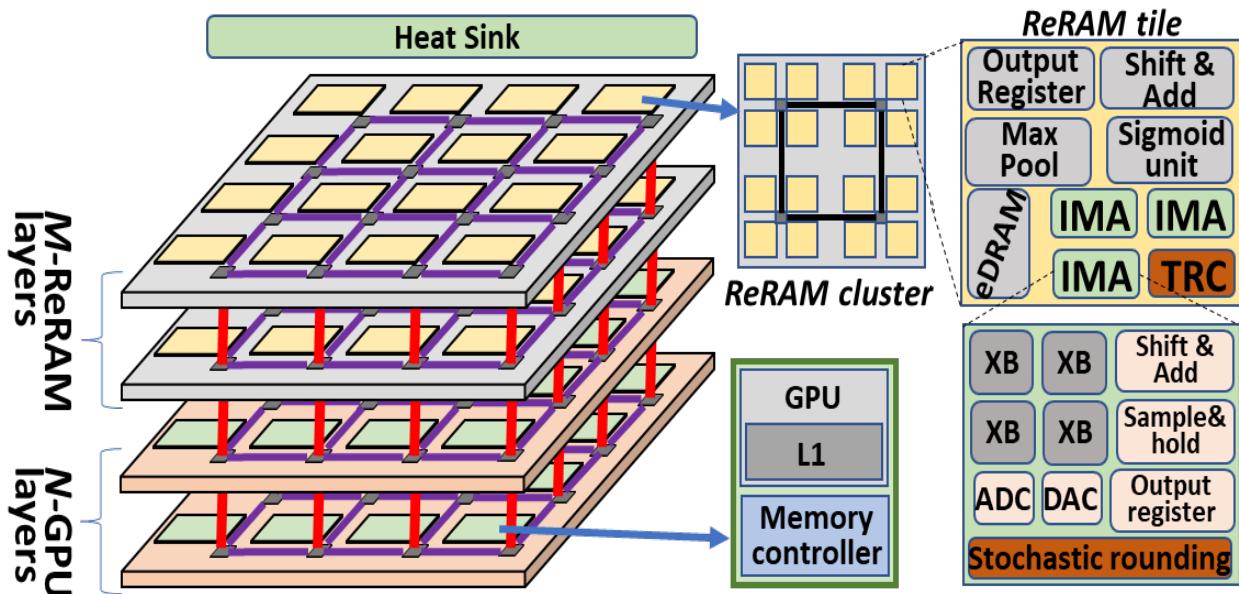
- Images are processed one after another in a pipeline
- Different from GPU/CPU execution
- Reduces number of write operations but requires more crossbars to implement

Performance of ReRAM accelerators



- Up to 100X speed-up achievable compared to GPU
- Area and power efficient as well

Why manycore ReRAM?



- Millions of weights in CNNs
- Pipelining requires many crossbars to implement
- Crossbars grouped into tiles (cores)
- Hence, ‘manycore’
- 3D allows for more cores

What about BN?

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

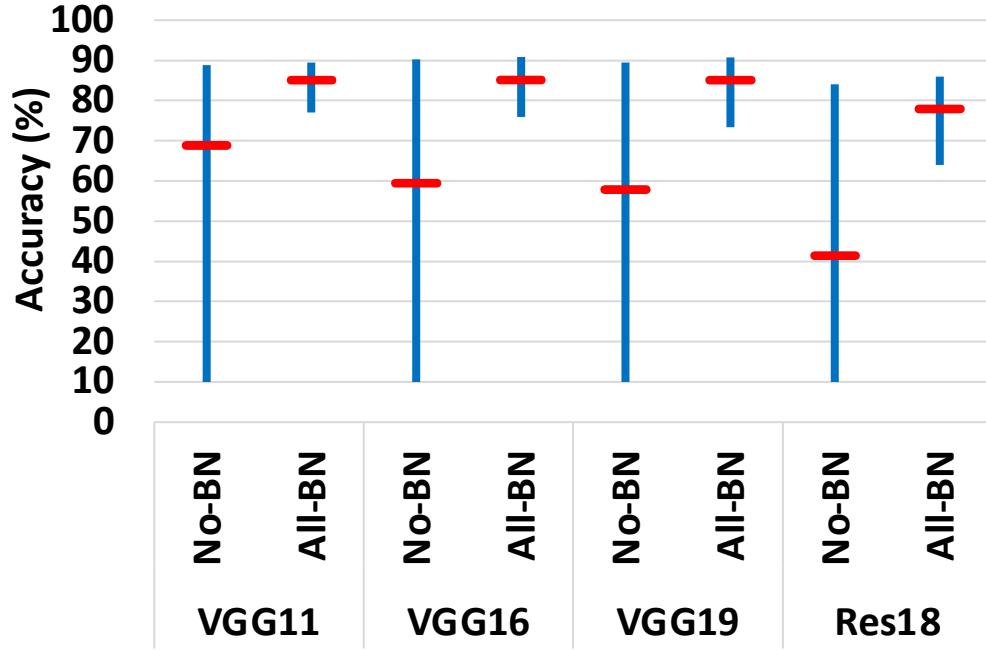
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

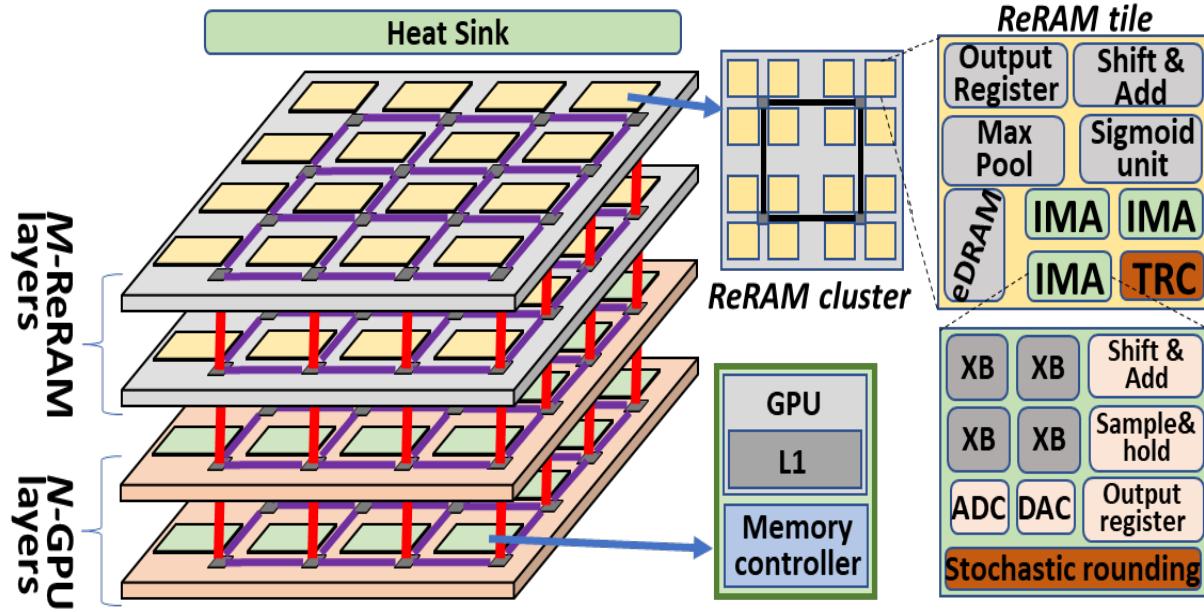
- Not every CNN layer has matrix multiplications
- BN layers are needed for Training deep CNNs
- Division, square-root needed for BN
- ReRAMs not good for these

Role of BN layers in CNN training



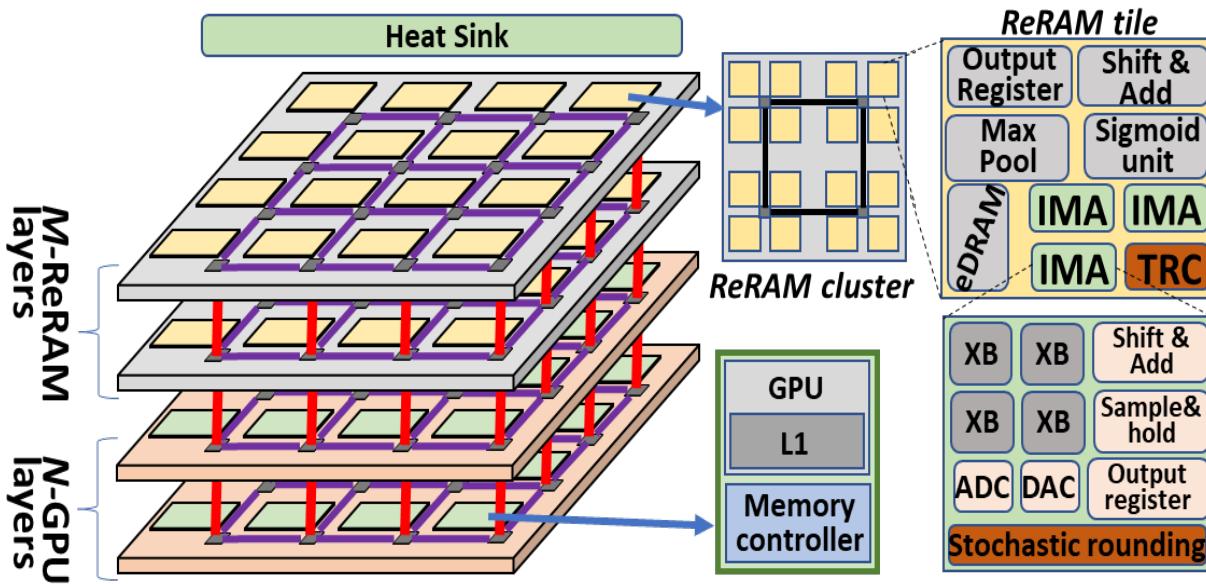
- Xavier initialization can replace BN layers
- However, they are not reliable always
- Sensitive to hyper-parameter choice

Heterogeneous ReRAM-based system



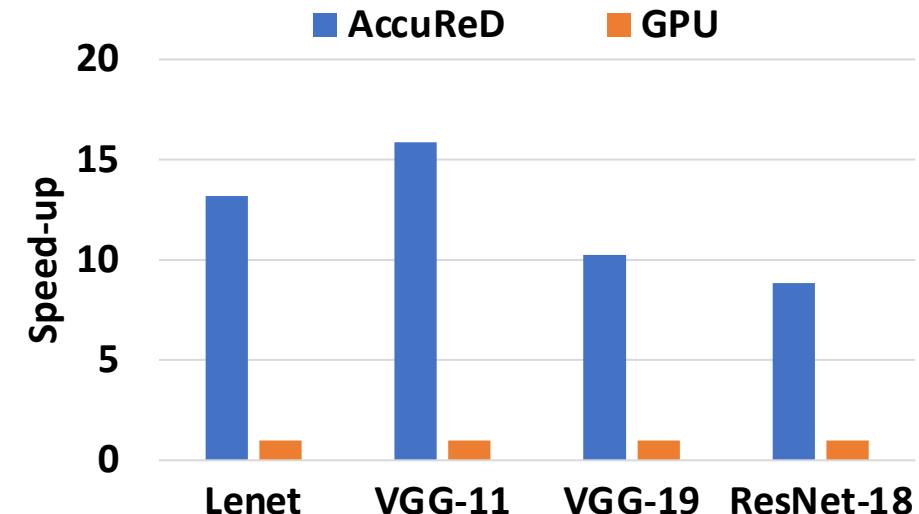
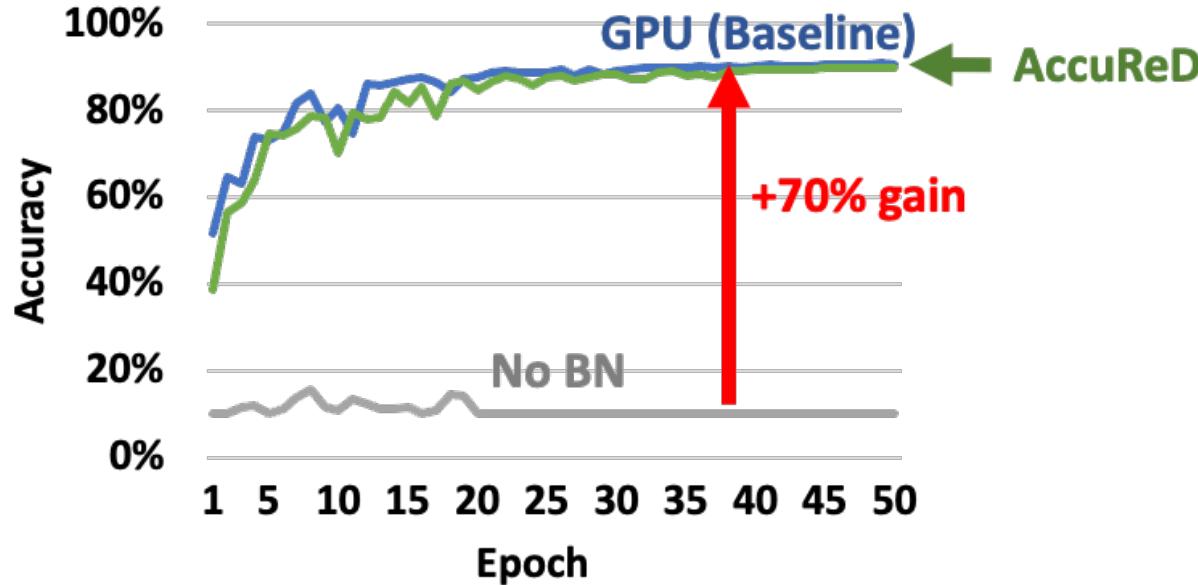
- Challenges with implementing BN layers
 - Full-precision
 - Division and square-root
 - Not suited for ReRAMs
- **Solution:** Heterogeneous architecture with GPU+ReRAM
- GPUs can do BN

Mapping CNN training to heterogeneous PIM



- Convolution and Fully connected layers on ReRAMs
- BN layers on GPUs
- 3D integration to allow fast data exchange

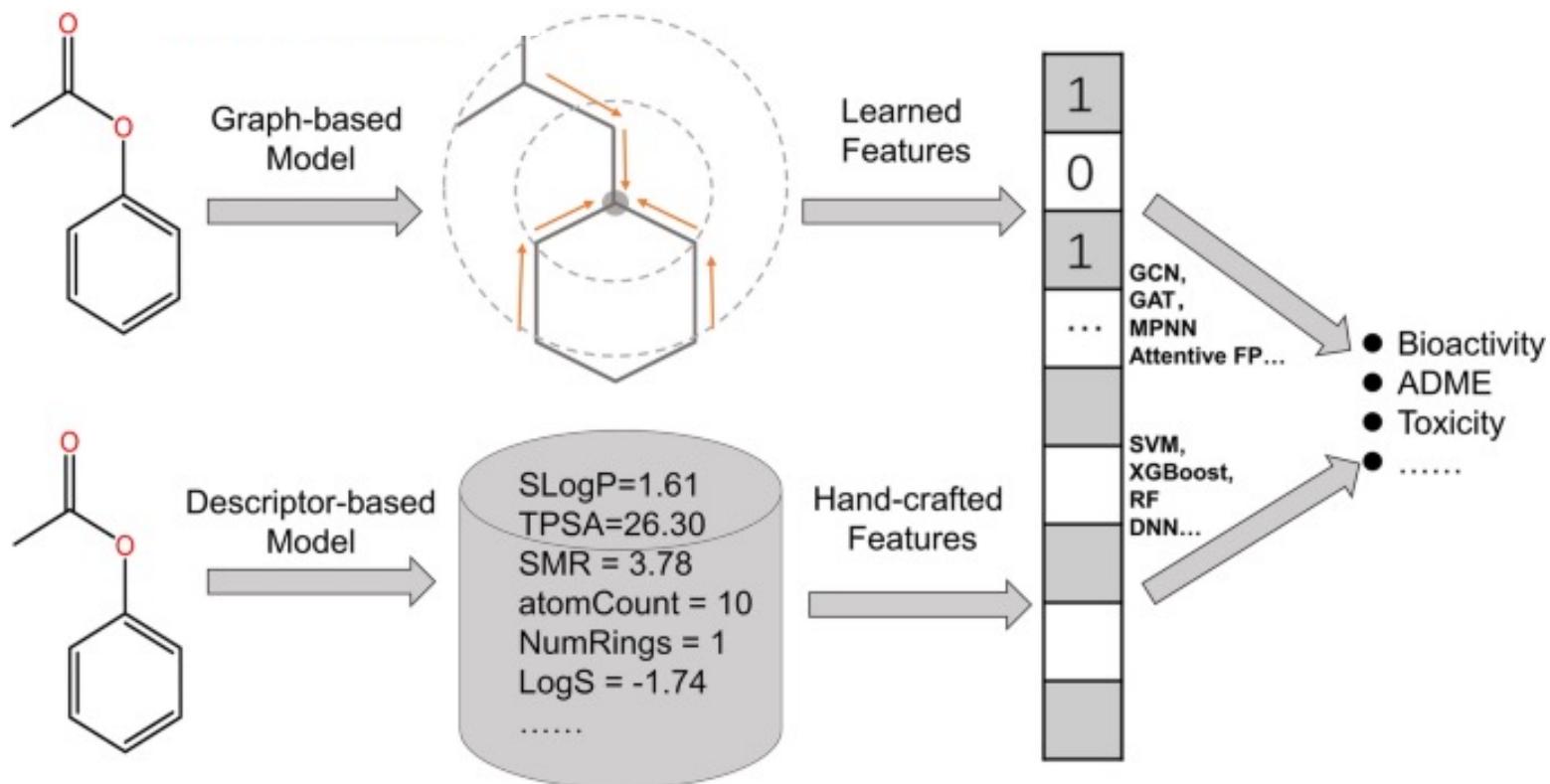
Performance of heterogeneous PIM



- Without BN, training of VGG11 with Cifar-10 failed
- With BN and heterogeneous PIM, accuracy is near-GPU

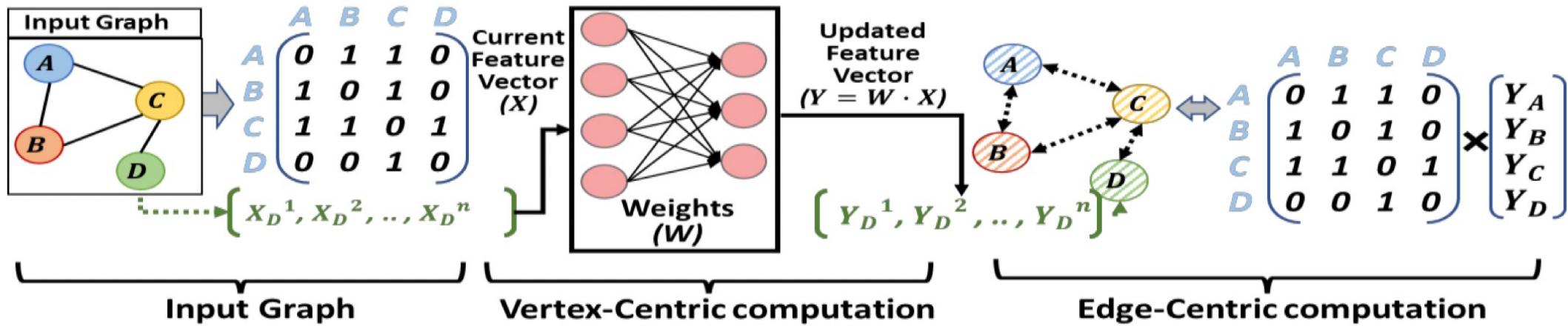
- Without BN, training of VGG11 with Cifar-10 failed
- With BN and heterogeneous PIM, accuracy is near-GPU

GNN training



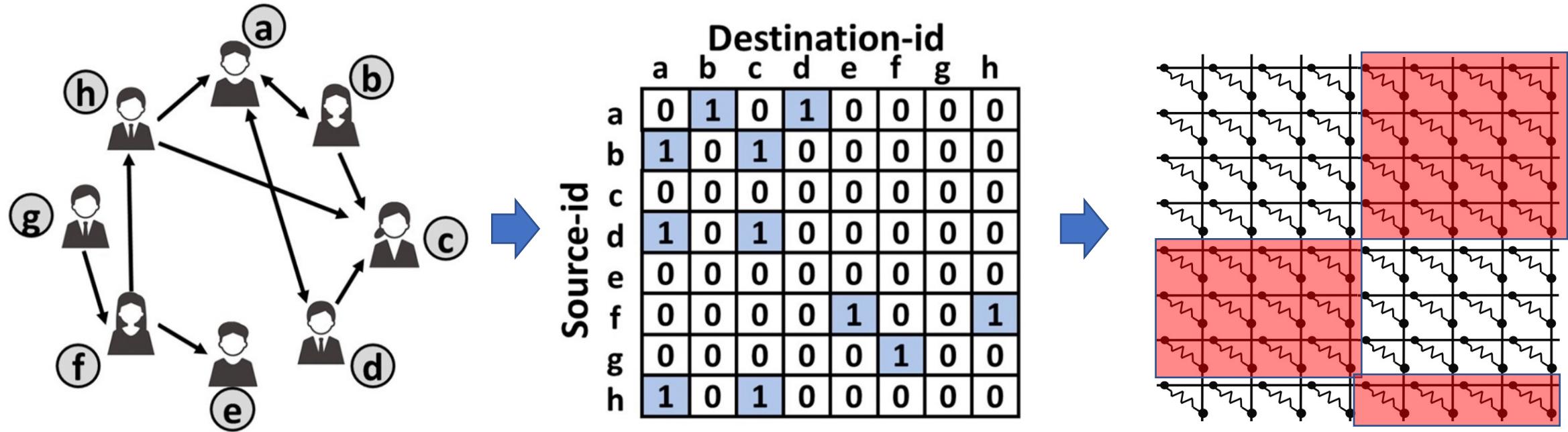
- DL on graphs
- Useful in recommendation systems, Drug discovery, etc.

GNN training on ReRAMs



- GNN layers also rely on matrix-vector multiplications
- GNNs usually not deep => BN layers not needed
- Can be implemented using ReRAMs

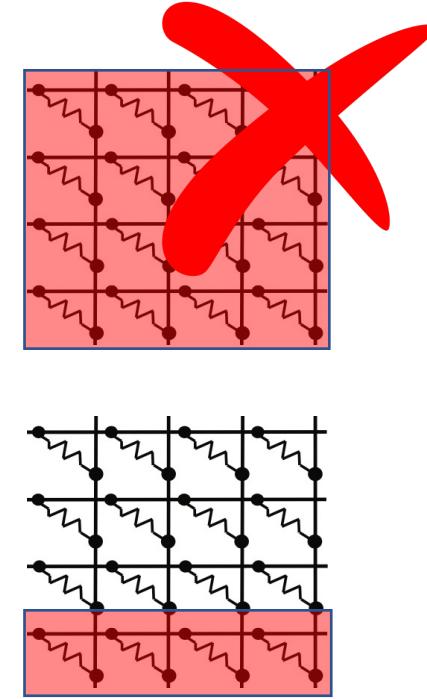
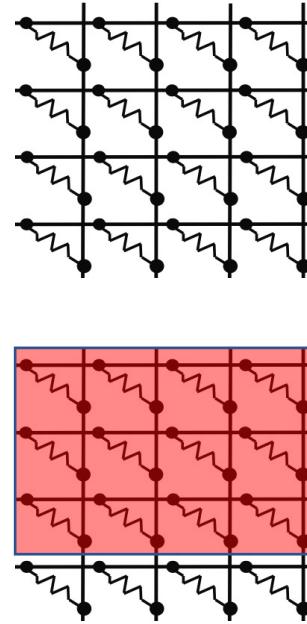
Storing sparse data on ReRAMs



- Graph adjacency matrices are very sparse
- >90% data is zero, multiplications with zero is redundant
- Storing on large ReRAM crossbars is wasteful

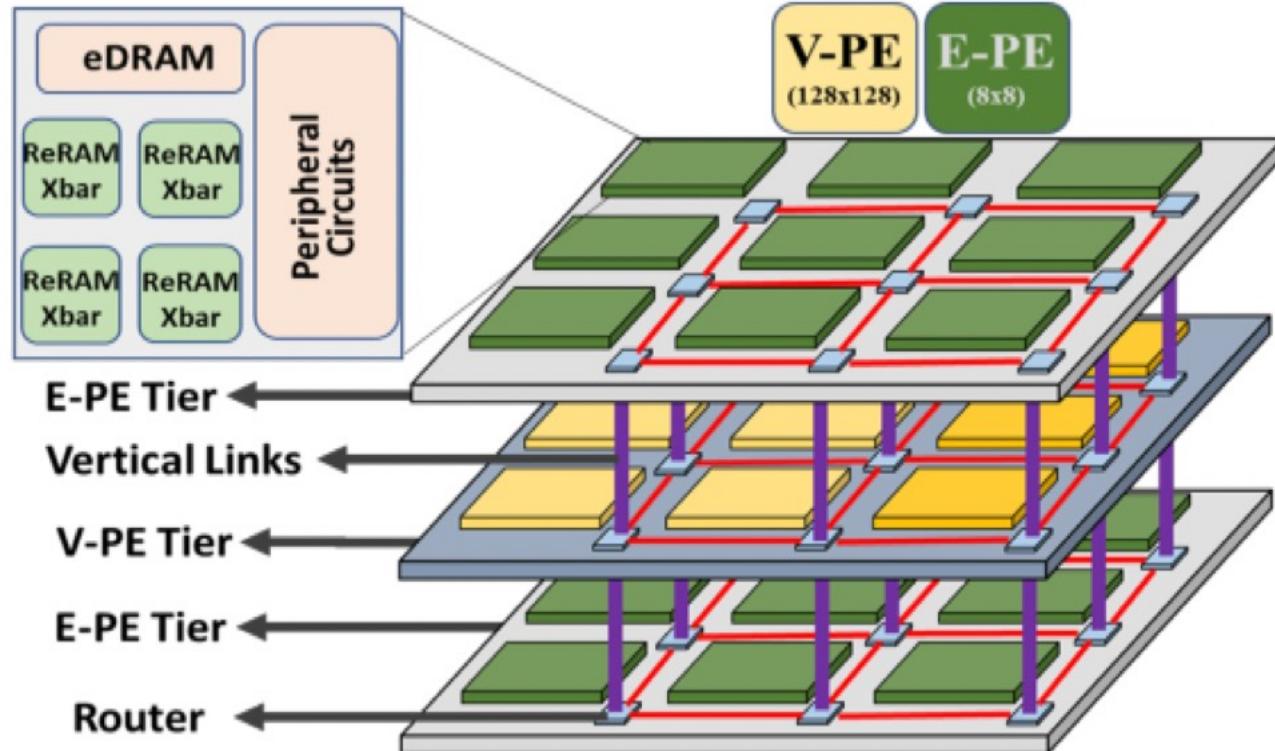
Using smaller crossbars

	Destination-id							
	a	b	c	d	e	f	g	h
a	0	1	0	1	0	0	0	0
b	1	0	1	0	0	0	0	0
c	0	0	0	0	0	0	0	0
d	1	0	1	0	0	0	0	0
e	0	0	0	0	0	0	0	0
f	0	0	0	0	1	0	0	1
g	0	0	0	0	0	1	0	0
h	1	0	1	0	0	0	0	0



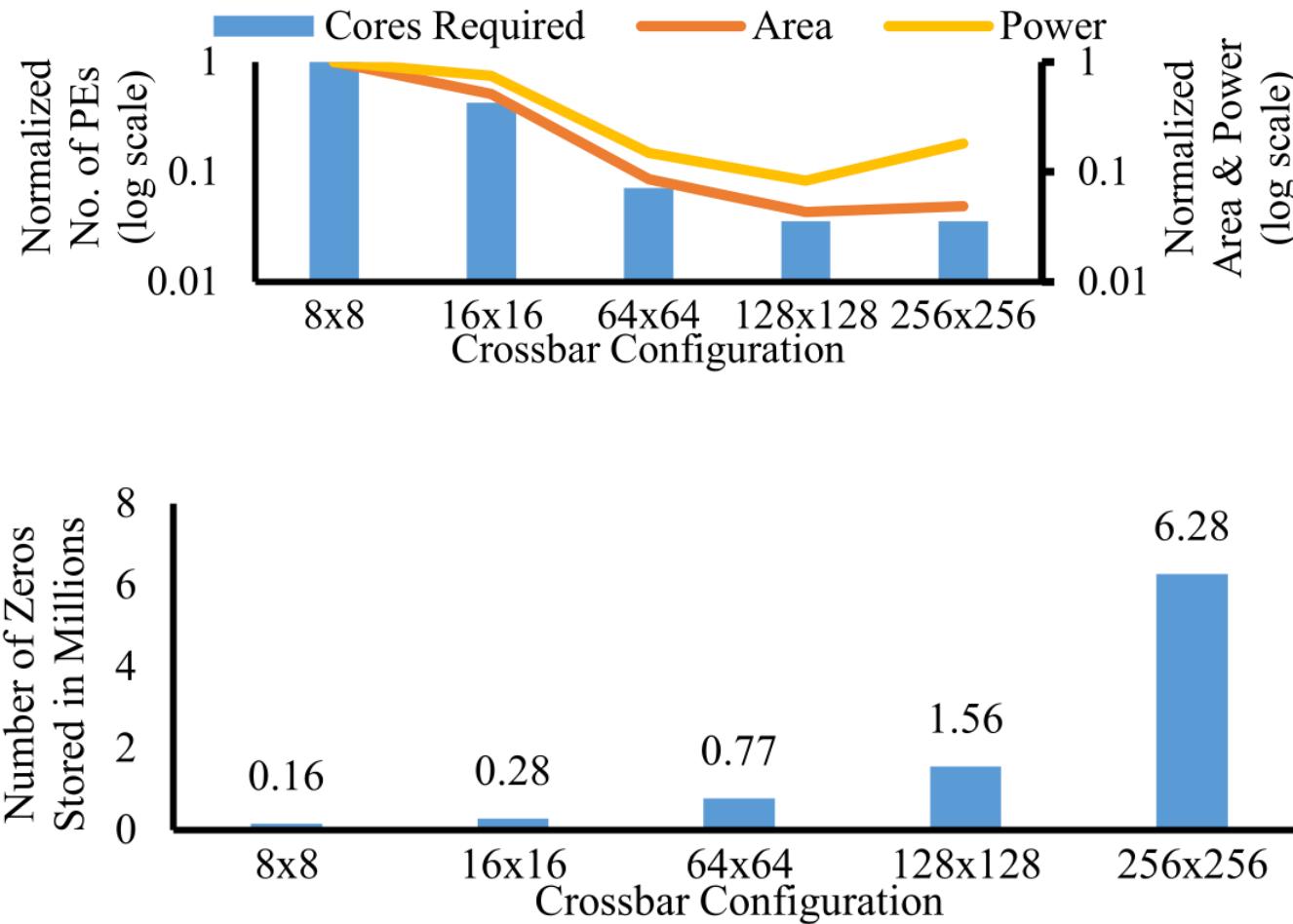
- Smaller crossbars are better for storing sparse data
- Some crossbars will be storing all zeros
- We do not need them at all => Area and power savings

Heterogeneous PIM for GNN training



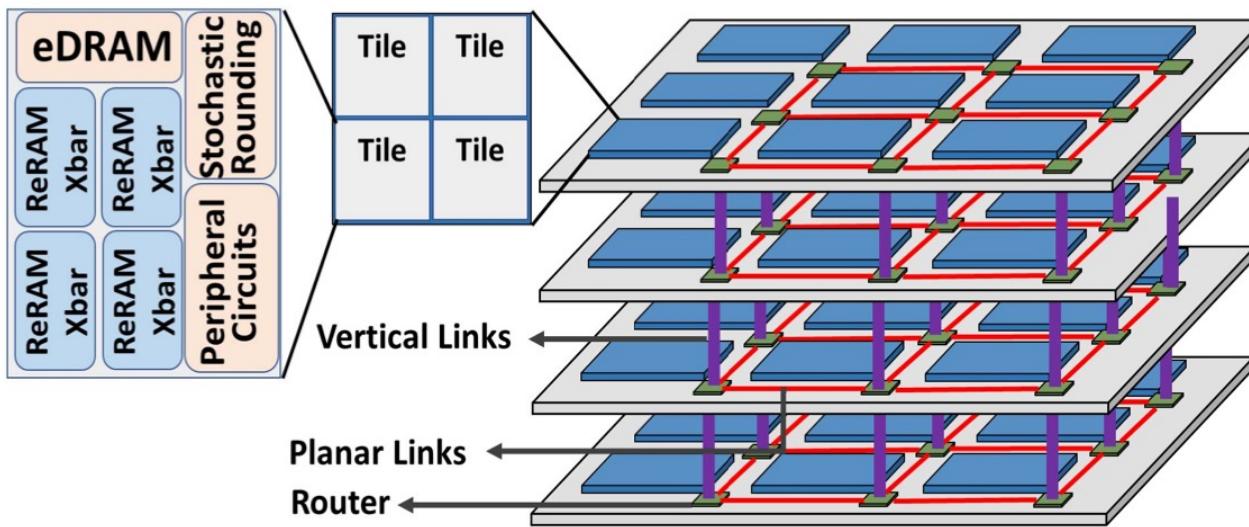
- GNNs have both dense data (weights) and sparse data (adjacency matrices)
- Dense weights => Large crossbars
- Sparse data => Small crossbars
- More efficient storage

Power-performance-area trade-offs



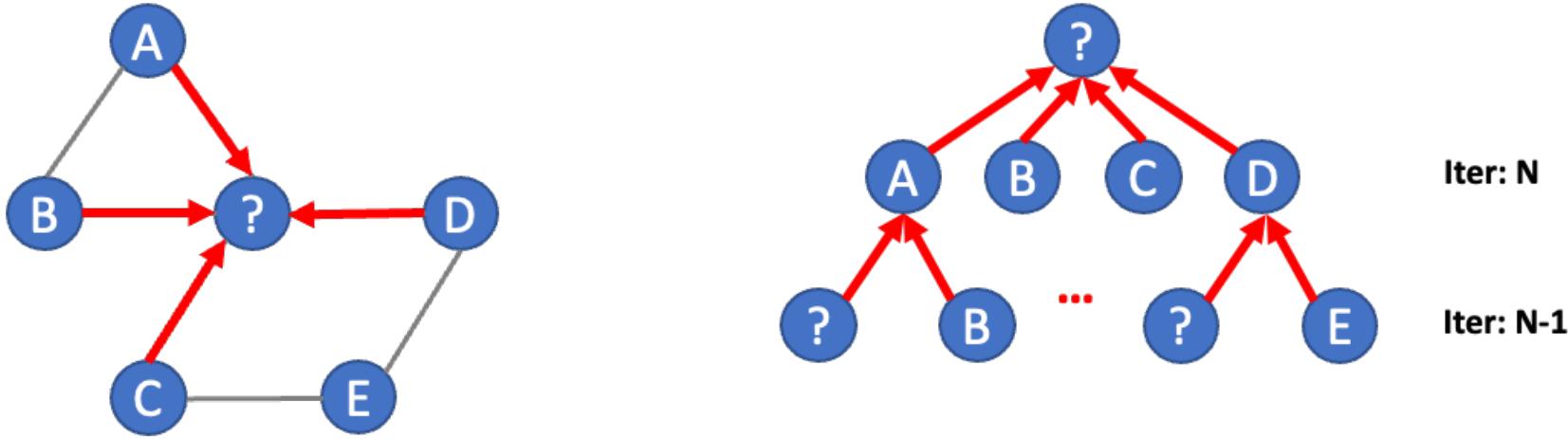
- Larger crossbars are inefficient for storing sparse data
- However, they are more area and power efficient
- Peripheral area >> Crossbar area
- Good design should reduce peripherals since crossbar area is relatively small

Homogeneous PIM for GNNs



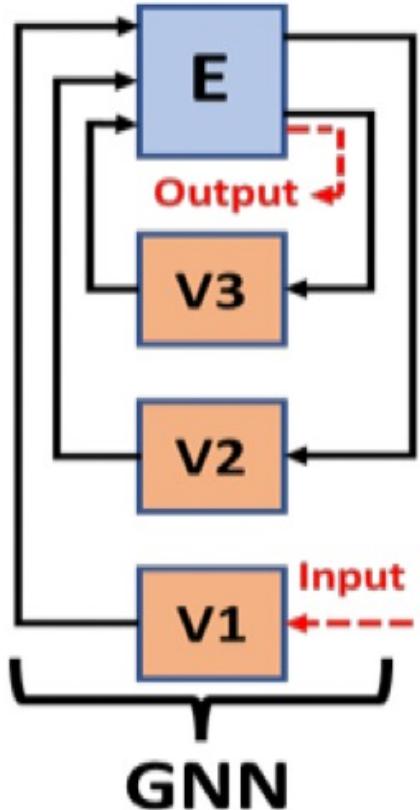
- ReRAM architecture with large crossbars only
- Both dense and sparse data stored on large crossbars
- Lower area and power overall
- 3D NoC for efficient communication

Message passing in hardware



- GNNs utilize neighbor information for predictions
 - $y_{current} = f(x_{current}, y_{neighbor})$
- Repeat for every GNN layer and iteration
 - Message passing => increased communication in the computing system

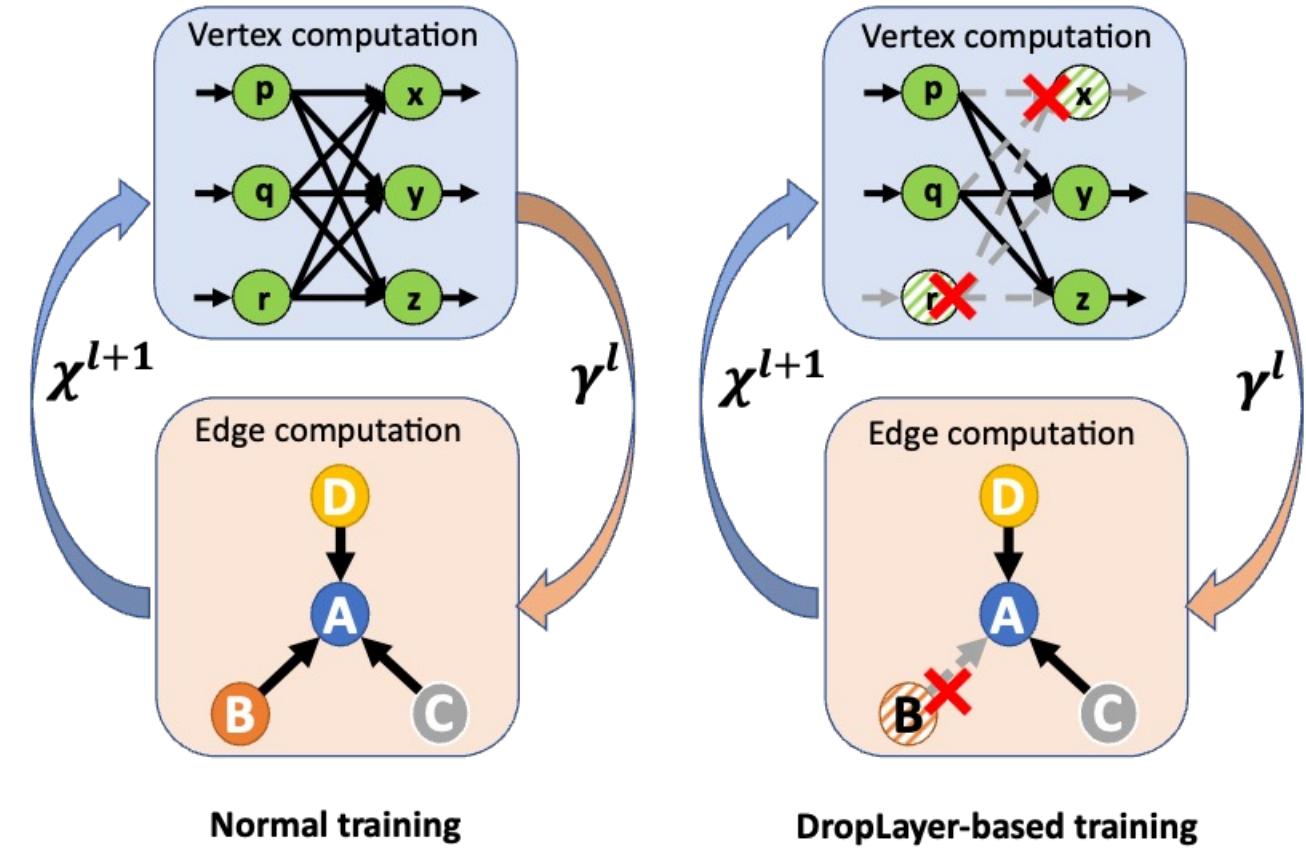
Many-to-few communication in ReRAMs



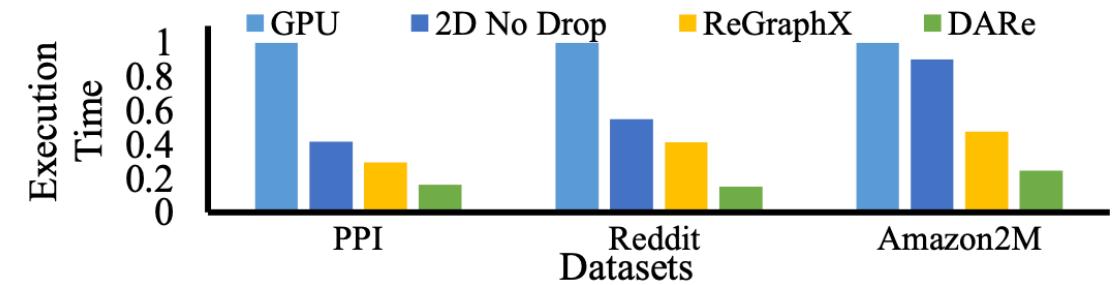
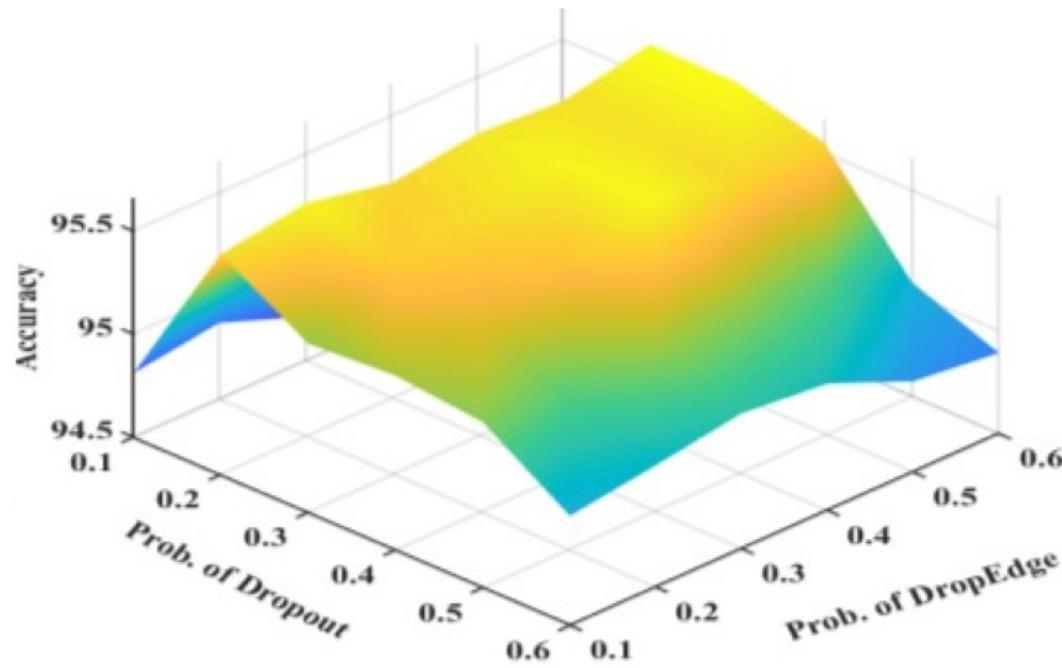
- Each GNN layer requires vertex (V) and edge-centric (E) computations
- Each V computation is associated with unique weights
- E computations require graph adjacency matrix (which is unique)
- Communication between multiple V's to one E => Many-to-few

Reducing communication using DropLayer

- Dropout is a regularization technique to improve accuracy
- DropLayer = Dropout + DropEdge
- Improved performance using DropLayer by reducing traffic



DropLayer performance

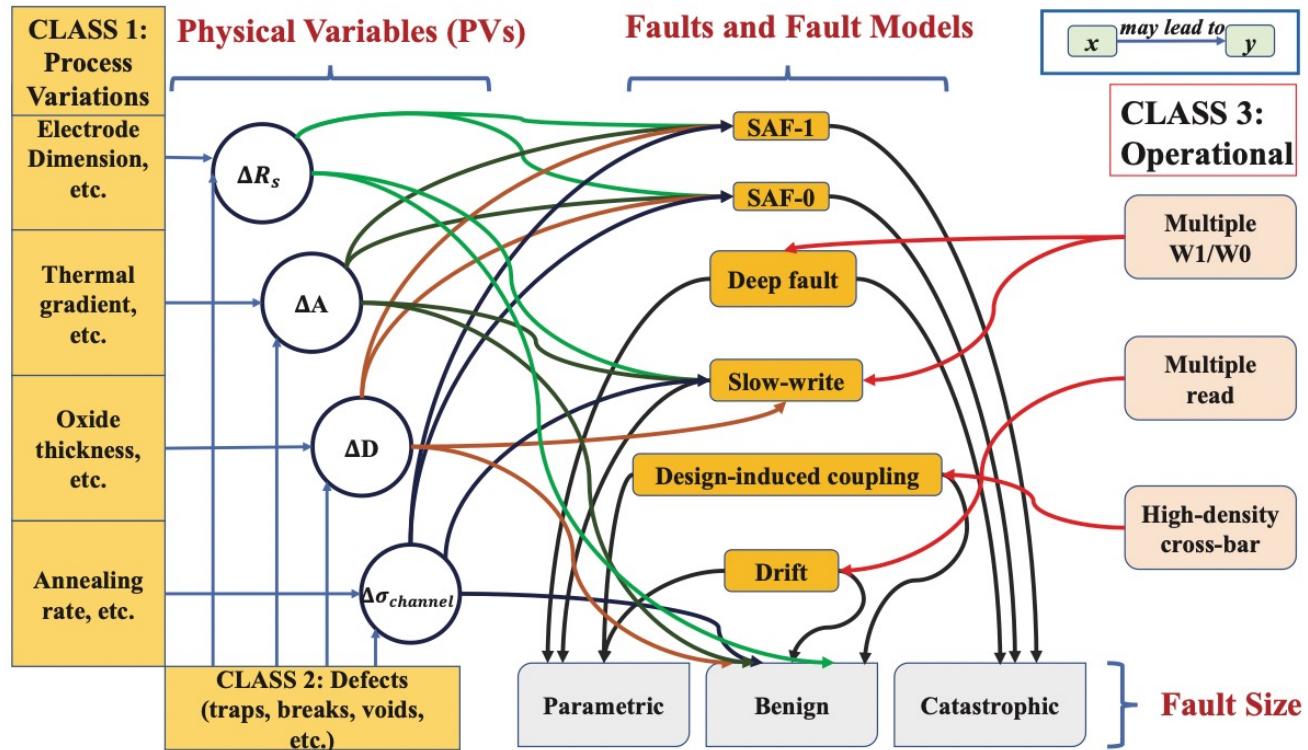


- Dropout improves accuracy
- Also reduces communication by ‘dropping’ data
- Improves execution time

Outline of Tutorial

- Exponential growth of Deep Learning and Hardware Challenges
- Introduction to Deep Learning
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- Methods to improve Reliability of ReRAMs
- Summary and Promising Directions

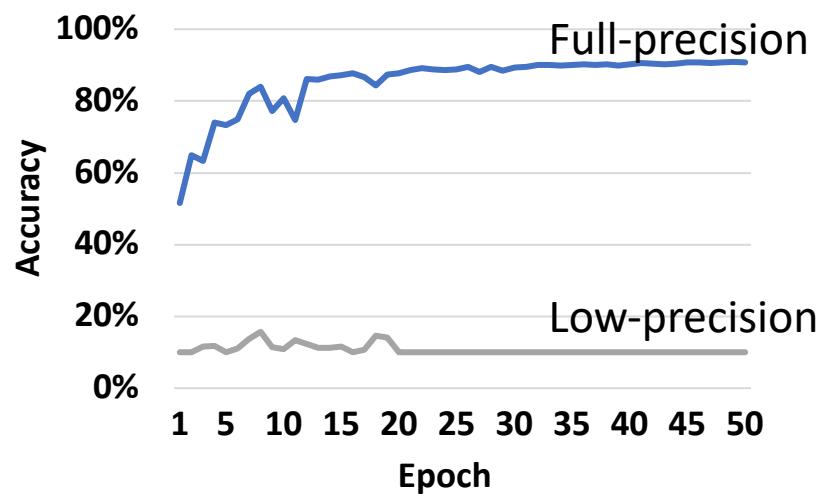
Reliability of ReRAMs



- ReRAMs have reliability issues
- Manufacturing defects, Process variations, write variance, etc.
- Not useful for real use

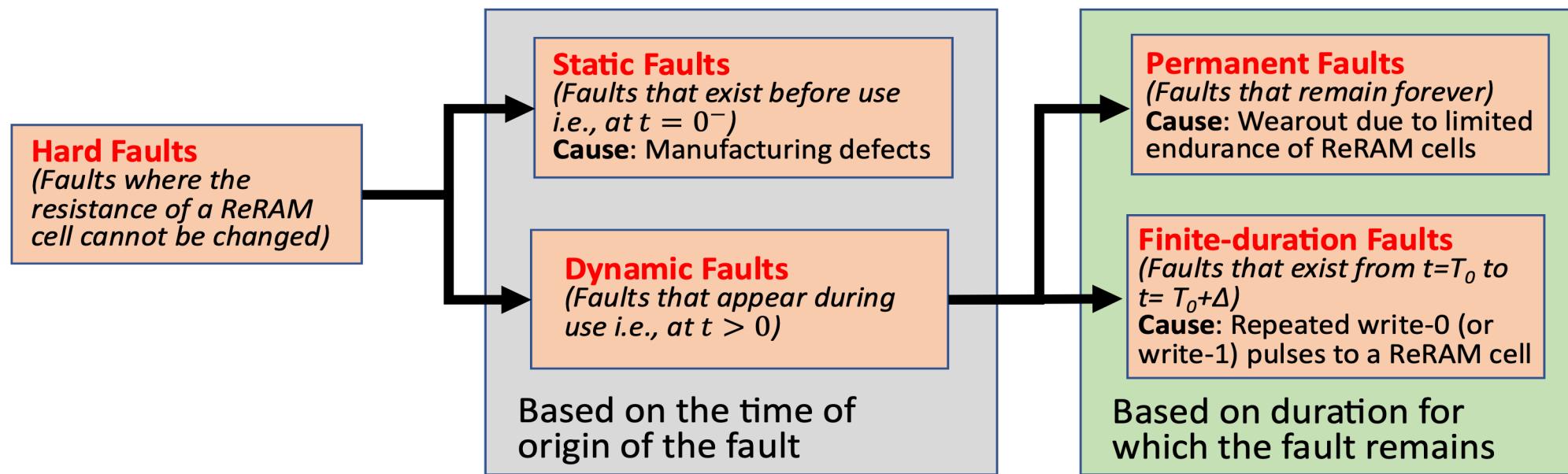
Fig. 4: Classification of fault origins and fault size.

Precision issue in ReRAMs



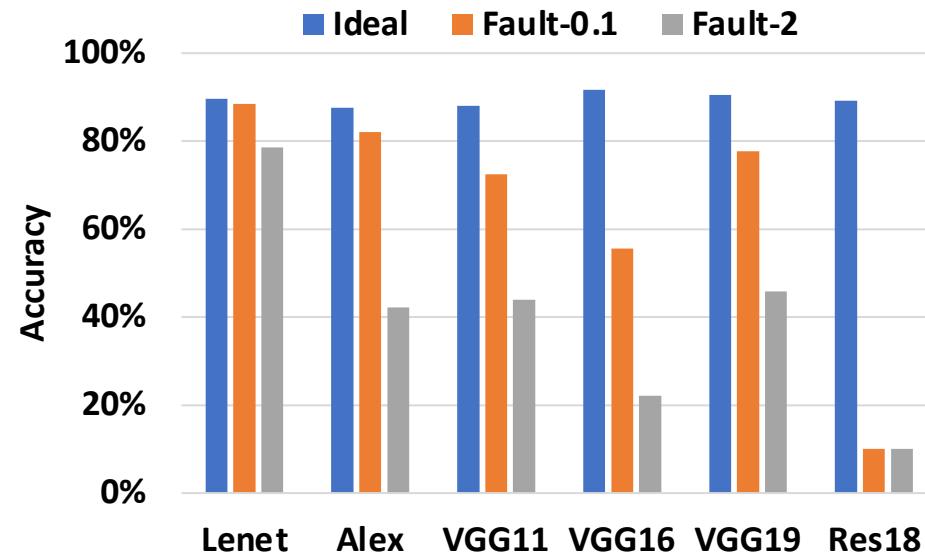
- ReRAMs use 16-bit fixed point precision
- GPUs use 32-bit floating-point precision
- Accuracy loss at low precision training

Different types of SAFs



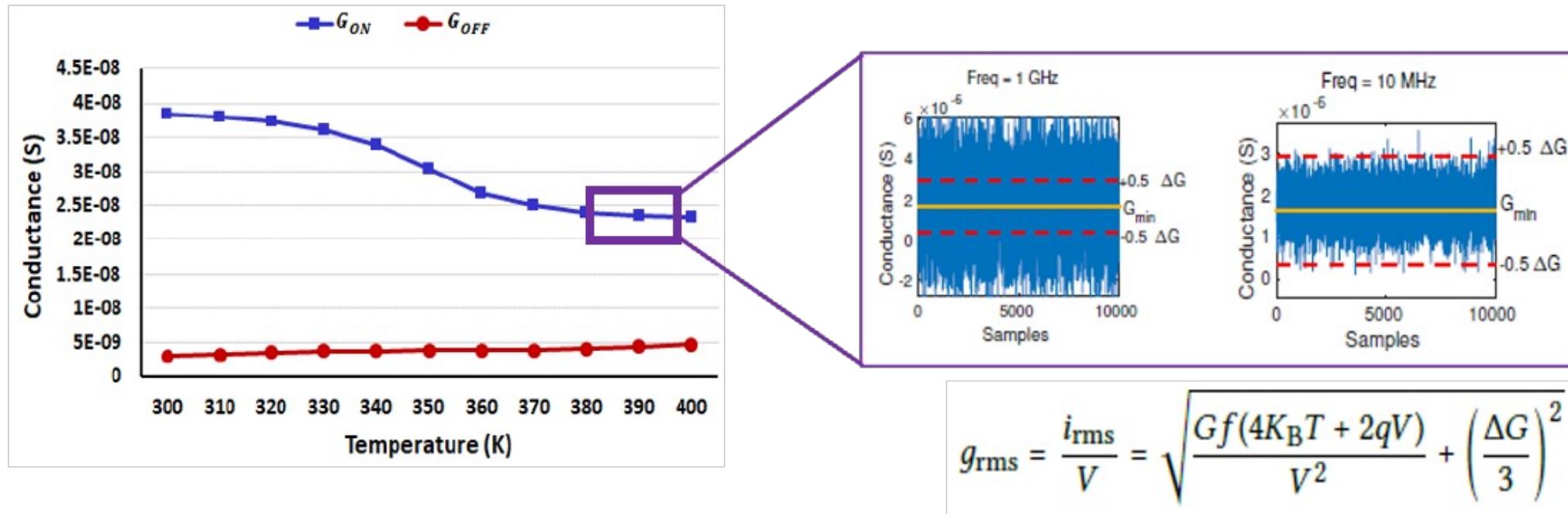
- Different types of faults in ReRAMs
- Some are permanent, some can be short-lived
- Some appear during use

Effect of faults



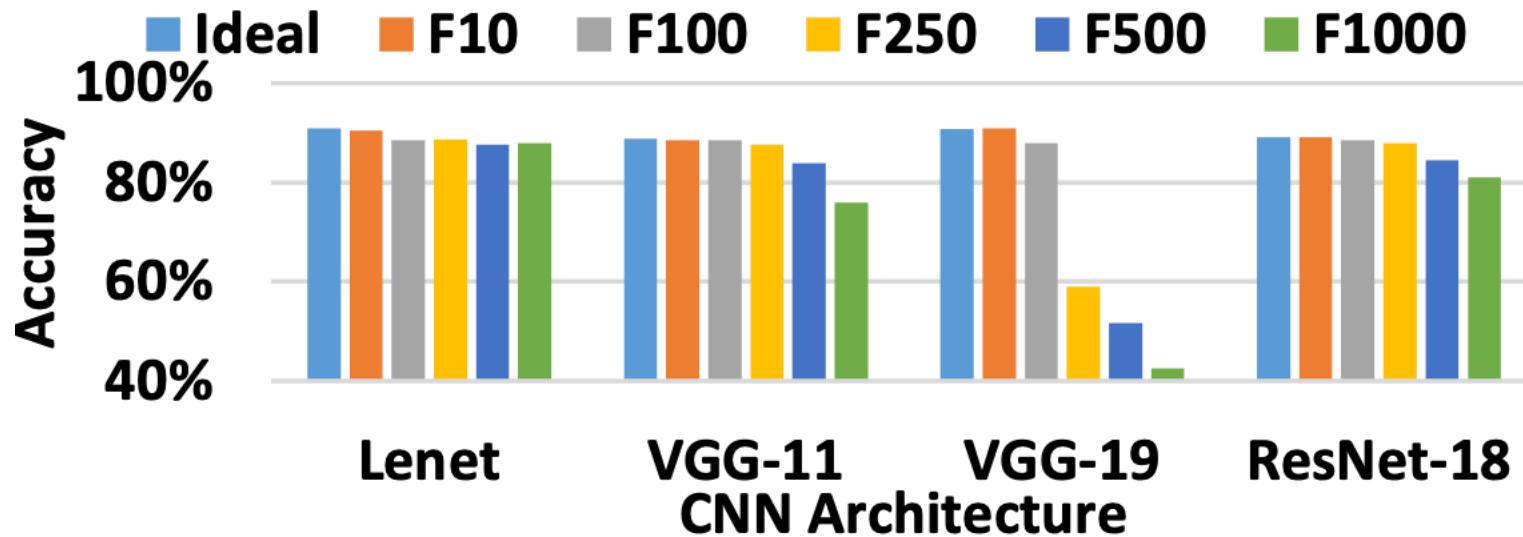
- Faults cause accuracy loss
- Some CNNs may fail to train completely due to faults
- Faults affect both weights and gradients

Thermal noise



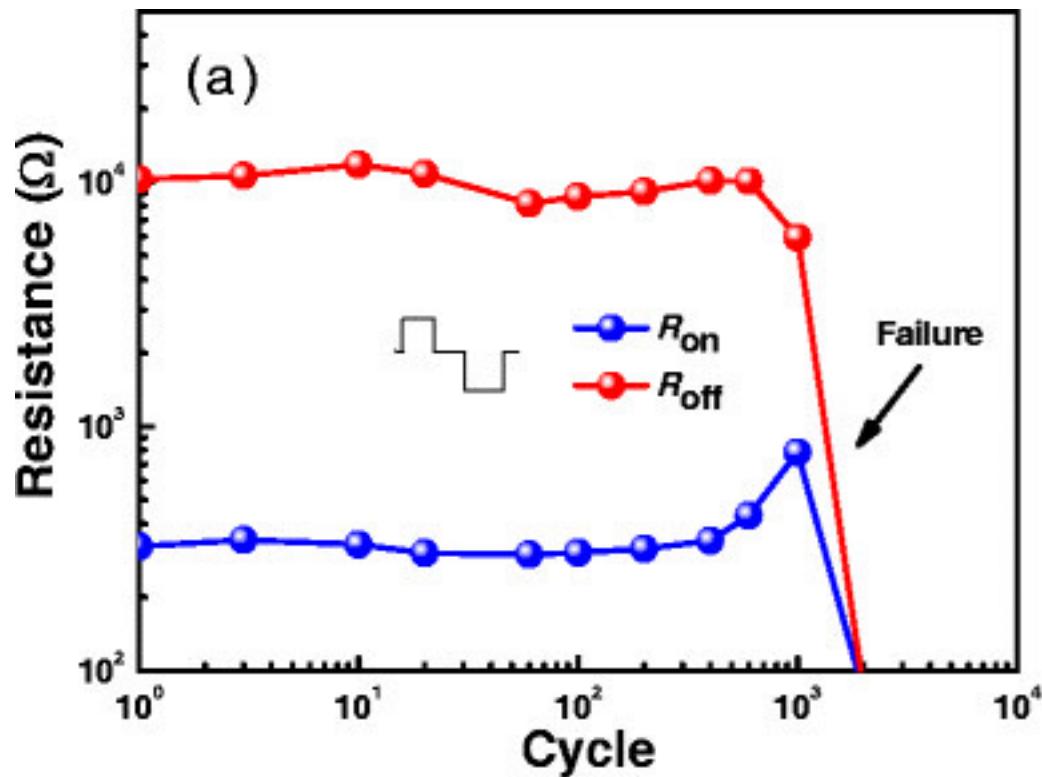
- Resistance changes with resistance
- Thermal noise can affect stored value
- Accuracy loss due to temperature

Accuracy loss due to noise



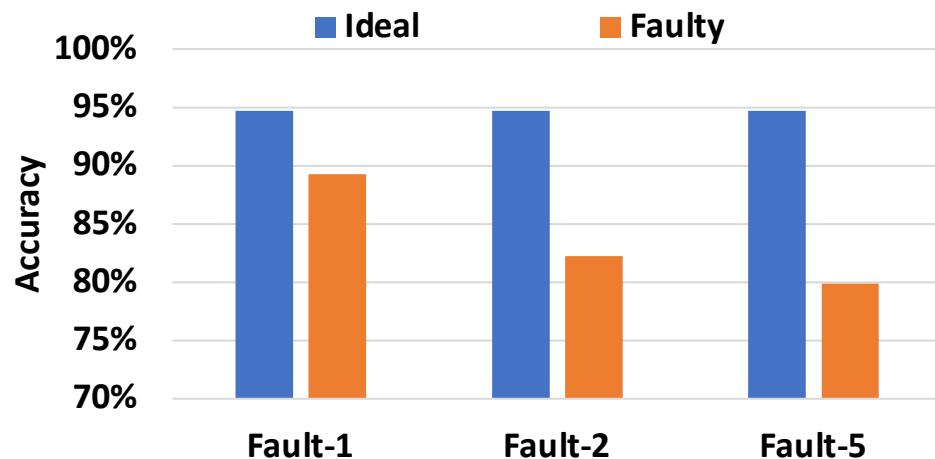
- Accuracy drop due to noise
- Higher accuracy loss at higher frequency

Write endurance of ReRAMs



- Repeated writes can damage ReRAM cells
- Cells stop behaving as usual
- CNN training involves many weight updates => writes

GNN training in presence of faults



- Faults are an issue for GNNs as well
- Accuracy drop at higher fault densities
- Must be addressed

Introduction

Robust and Efficient ReRAM-based System

High Inferencing Accuracy under Stochastic Noise

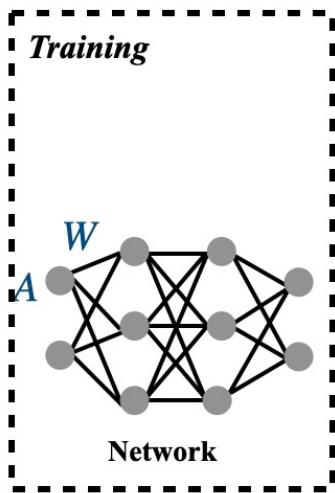
Area-, Energy-, Latency-efficient Designs

Multi-objective Optimization

Low-cost Optimization Algorithm

High-quality Pareto-front Designs

DNN Inferencing Process on ReRAM Crossbars



DNN
Parameters

Hardware
Configurations

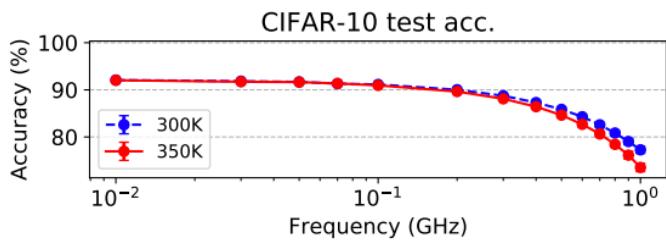
Stochastic
Noise

Design
Objectives

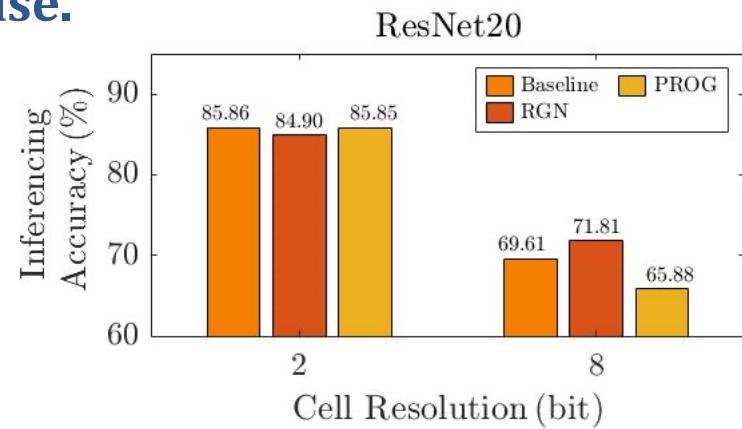
In need of hardware-aware training method

Challenges in Robust ReRAM design

(1) High-amplitude noise due to frequency and temperature.



(2) Aggregated noise due to the combination of stochastic noise.



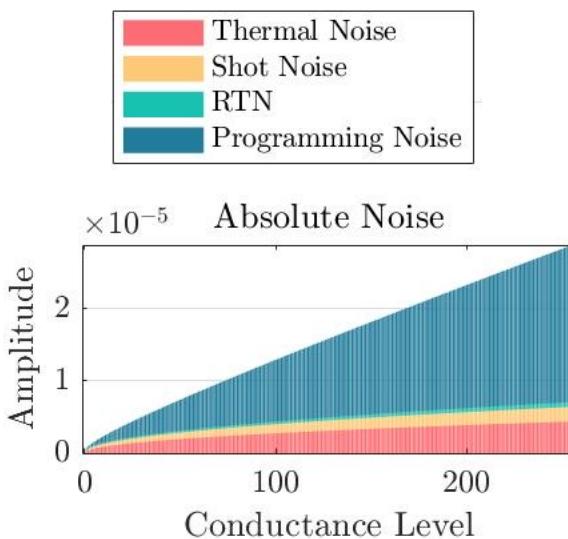
(3) Reduced noise margin due to high-resolution cell.

Figure 1 from [He, Zhezhi, et al. 2019]

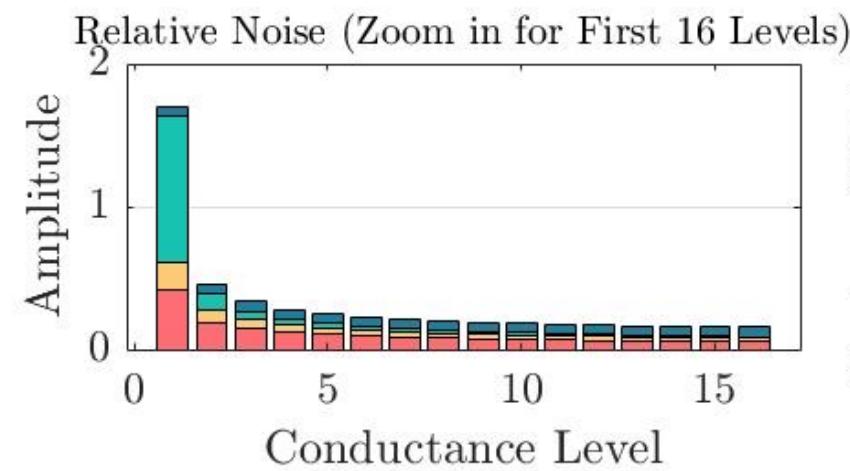
ReSNA Methodology

ReSNA: ReRAM-based Stochastic-Noise-Aware Training

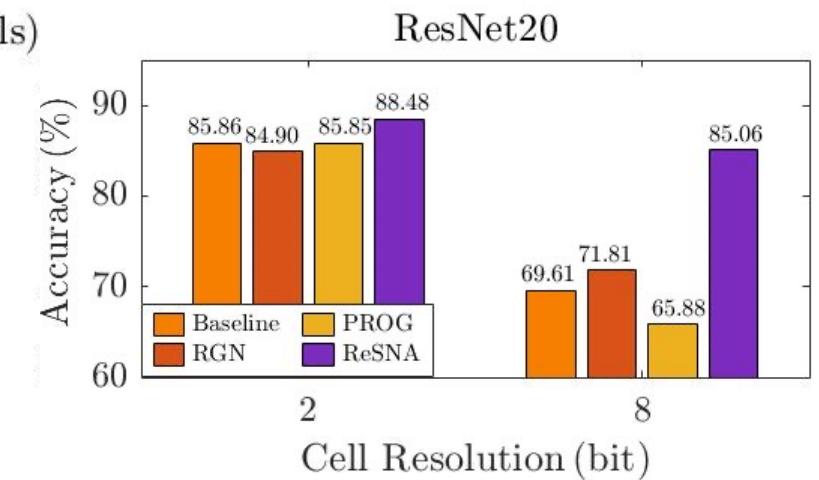
(1) Analysis the distribution of noise under frequency and temperature setting.



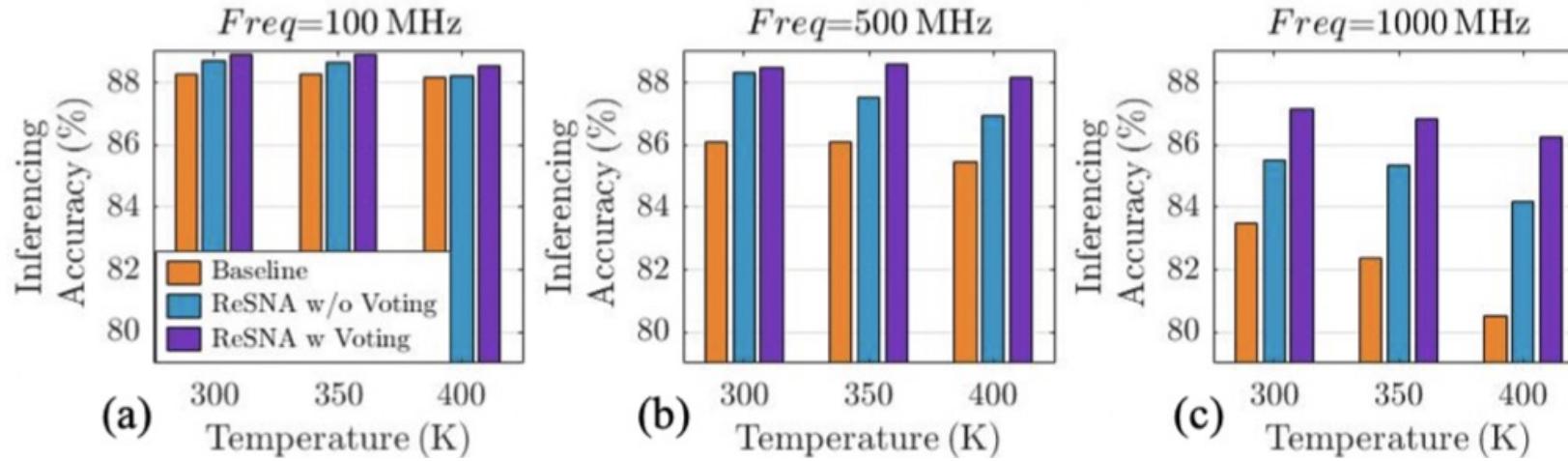
(2) Include the relative conductance change to the ReRAM cell during training.



(3) Apply the hardware settings to the quantization step.



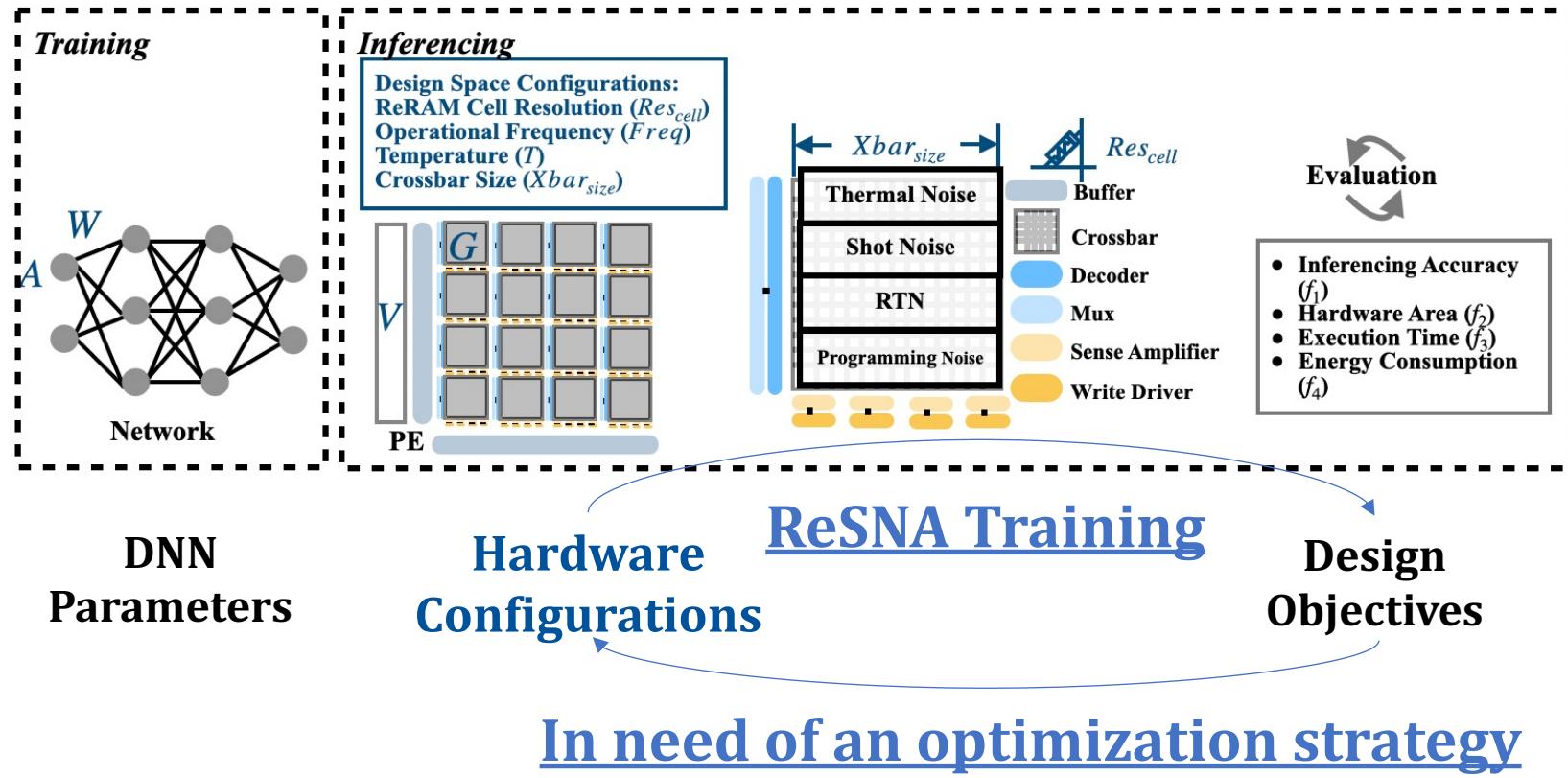
ReSNA Results



ReRAM inferencing accuracy under various temperature and frequency settings.
8-bit cell resolution, 64×64 crossbars. ResNet20 on the CIFAR-10 dataset.

- ✓ **Validates for the combination of stochastic noise.**
- ✓ **Works for various temperature and frequency setting, as well as the high-resolution (8-bit) cell setting.**
- ✓ **Shows promising accuracy improvement under the 1000MHz and 400K.**

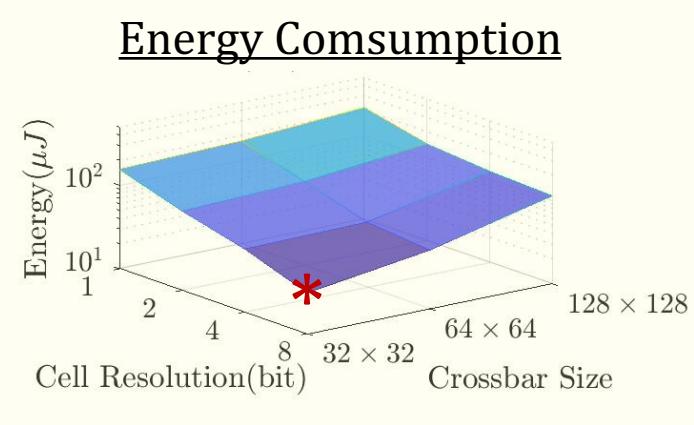
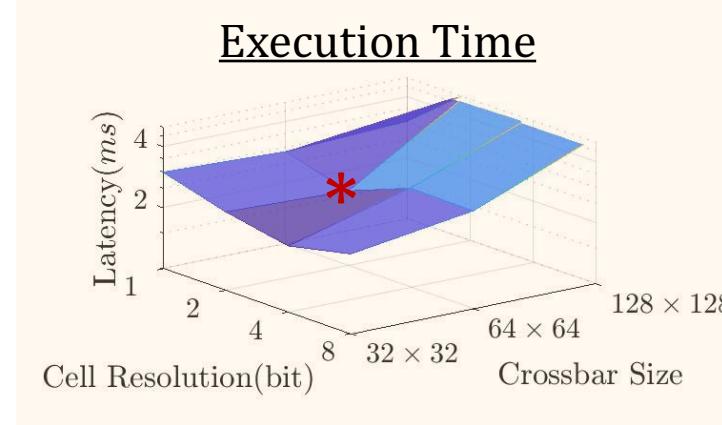
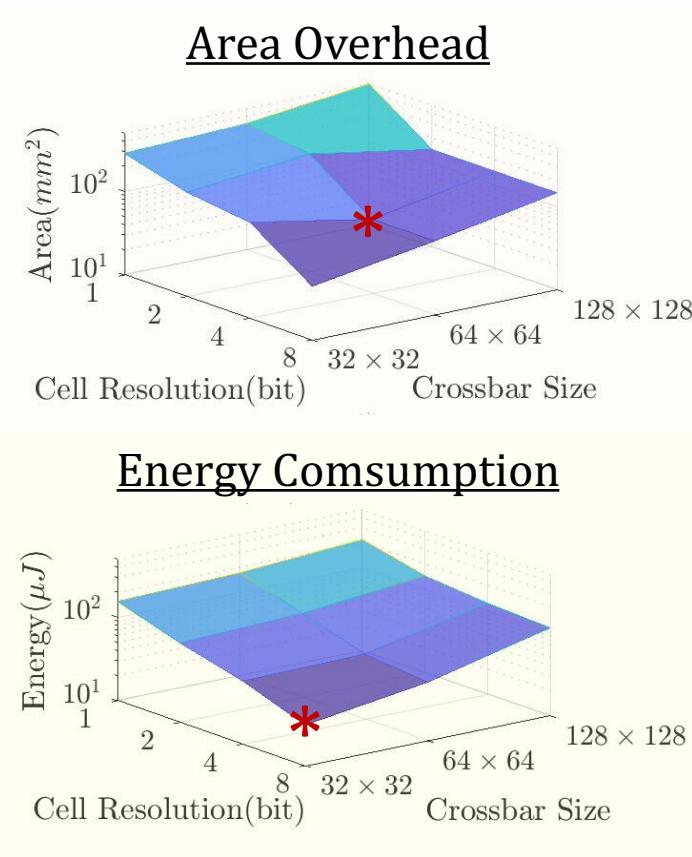
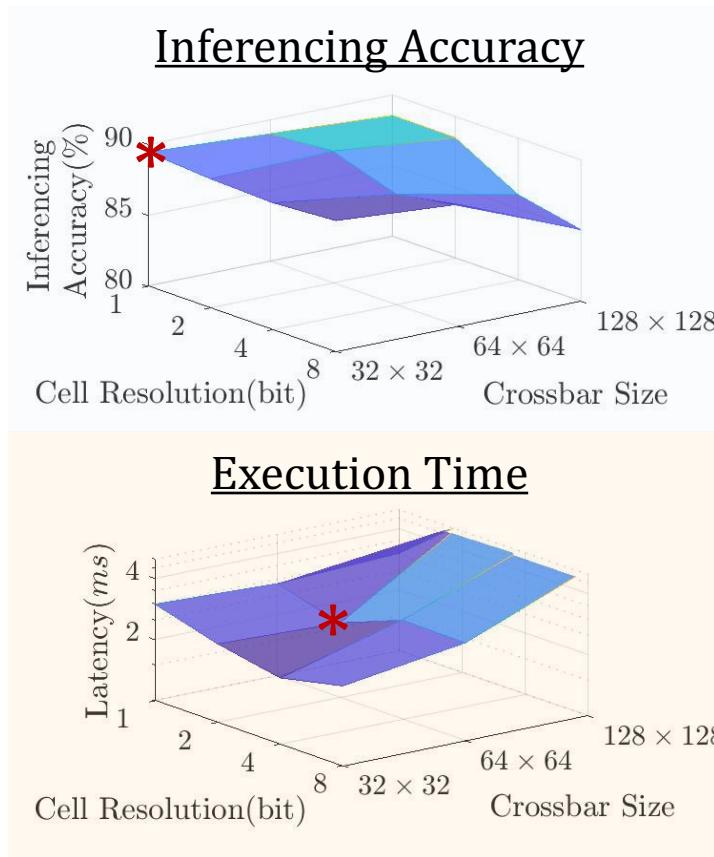
Robust and Efficient ReRAM-based System



Q1: The number of available choices is a combination problem.

Q2: Full hardware training process take non-negligible computation cost.

Design Trade-offs Considering Different Objectives



Observation: a global optimal design configuration is not achievable.

Goal: Find design configurations that lie in the Pareto set with the minimal cost.

MOO Algorithms for Hardware Design

- Inferencing Accuracy (f_1)
- Hardware Area (f_2)
- Execution Time (f_3)
- Energy Consumption (f_4)

Objective functions for ReRAM design.

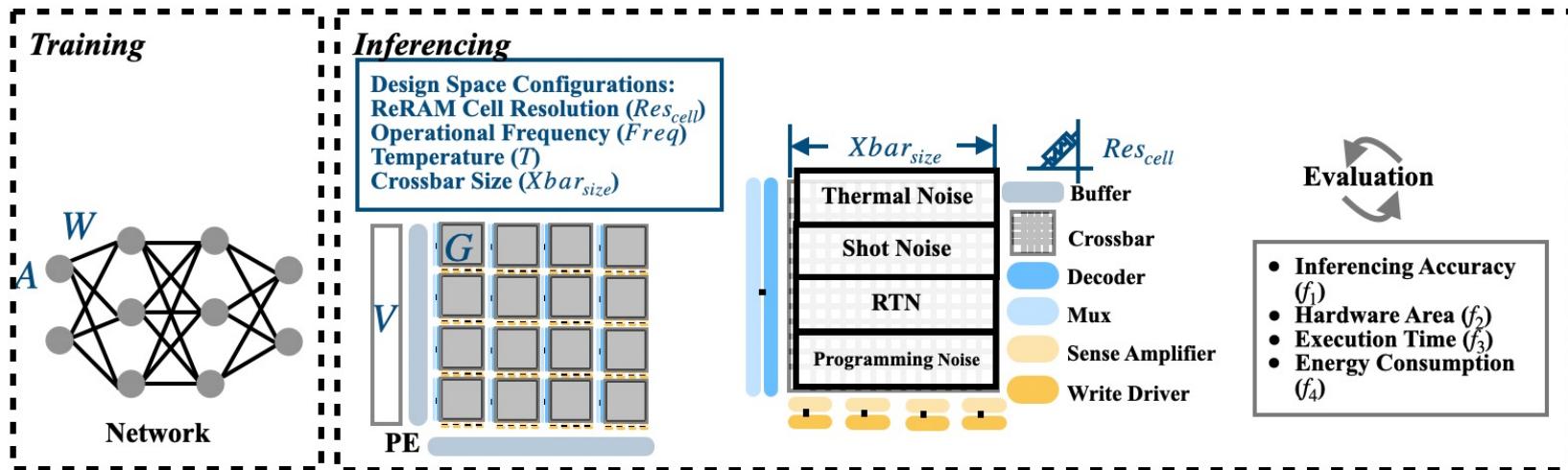
\$: the objective function is cheap to evaluate

e.g.: NSGA-II,
AMOSA

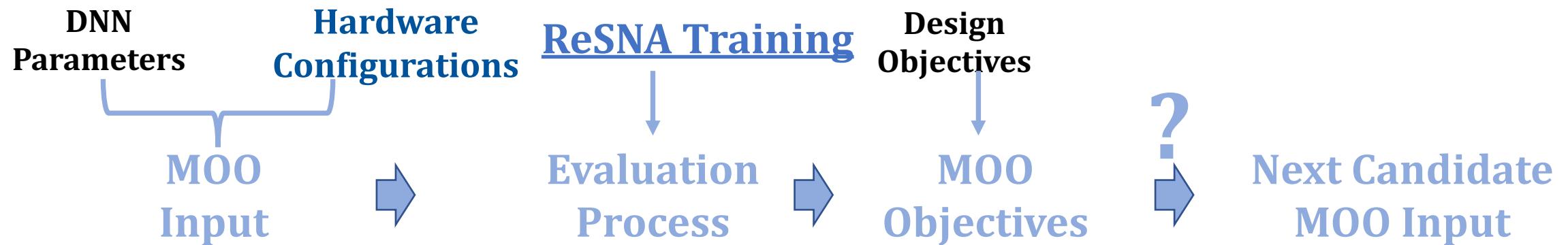
\$\$\$: the objective function is expensive to evaluate

e.g.: Bayesian Optimization, Max - entropy Search

MOO Steps and Goal



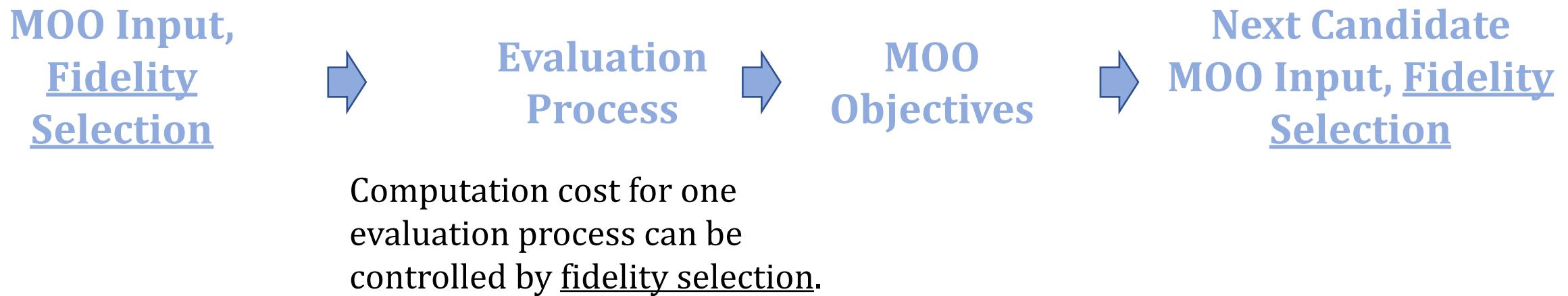
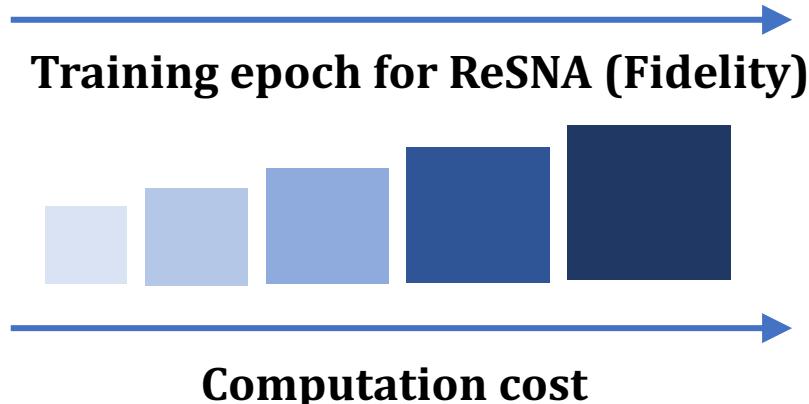
Find Pareto-Optimal ReRAM-based System with Minimal Cost



CF-MESMO: Continuous Fidelity

Definition for continuous fidelity :

- Vary the number of training epochs in ReSNA to trade-off computation cost and accuracy of objective function evaluations.



CF-MESMO: Max-Entropy Search

Surrogate model: Gaussian Processes

Sample the Pareto Front

Calculate the information gain per unit cost

Select next candidate ReRAM design and fidelity pair

Algorithm 1 CF-MESMO Algorithm

Input: ReRAM design space \mathcal{X} ; DNN π ; four objective functions f_j and their continuous approximations g_j using ReSNA training; total cost budget C_{total} .

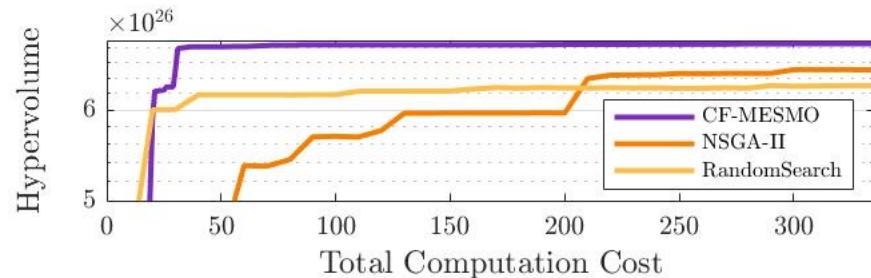
- 1: Initialize GP models $\mathcal{GP}_1, \dots, \mathcal{GP}_4$ via ReRAM design evaluations D
 - 2: **While** $C_t \leq C_{total}$ **and** not converged **do**
 - 3: for each sample $s \in 1, \dots, S$:
 - 4: Sample highest-fidelity functions $\tilde{f}_j \sim \mathcal{GP}_j(., z_j^*)$
 - 5: $\mathcal{F}_s^* \leftarrow$ Solve *cheap* MOO over $(\tilde{f}_1, \dots, \tilde{f}_K)$
 - 6: Select ReRAM design and fidelity pair:

$$(\mathbf{x}_t, \mathbf{z}_t) \leftarrow \arg \max_{\mathbf{x} \in \mathcal{X}, \mathbf{z} \in \mathcal{Z}} \alpha_t(\mathbf{x}, \mathbf{z}, \mathcal{F}^*)$$
 Equation (9)
 - 7: Perform ReSNA training of DNN π with ReRAM design and fidelity pair $(\mathbf{x}_t, \mathbf{z}_t)$
 - 8: Evaluate objectives f_1, f_2, f_3, f_4 for trained DNN on ReRAM design \mathbf{x}_t
 - 9: Update the total cost: $C_t \leftarrow C_t + C(\mathbf{x}_t, \mathbf{z}_t)$
 - 10: Aggregate training data: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t)\}$
 - 11: Update surrogate statistical models $\mathcal{GP}_1, \dots, \mathcal{GP}_4$
 - 12: $t \leftarrow t + 1$
 - 13: **end**
 - 14: **return** Pareto set and Pareto front of objective functions $f_1(x), \dots, f_4(x)$
-

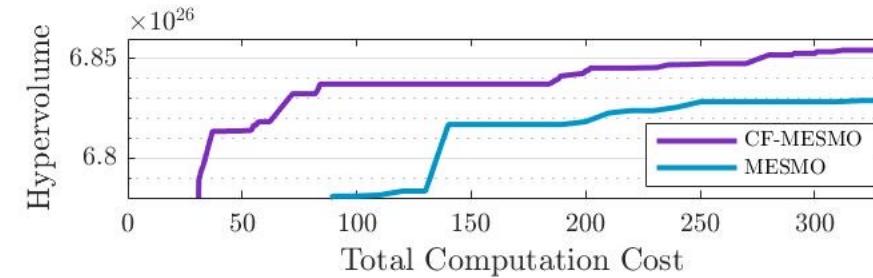
$$\alpha_t(\mathbf{x}, \mathbf{z}, \mathcal{F}^*) = \frac{1}{C(\mathbf{x}, \mathbf{z})S} \sum_{j=1}^4 \sum_{s=1}^S \frac{\gamma_s^{(g_j)} \phi(\gamma_s^{(g_j)})}{2\Phi(\gamma_s^{(g_j)})} - \ln(\Phi(\gamma_s^{(g_j)})).$$
(9)

Using CF-MESMO to Optimize ReRAM Crossbars

- CF-MESMO vs. NSGA-II and random search.



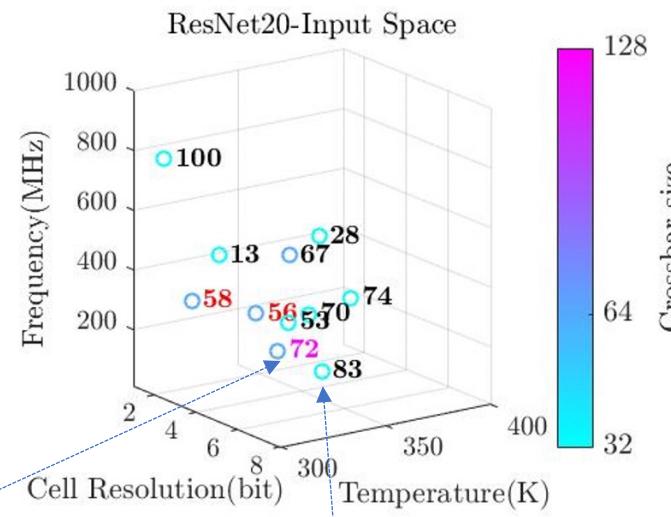
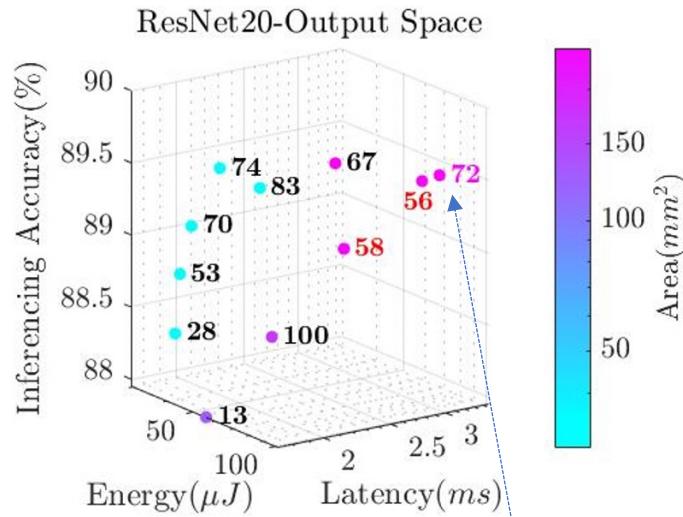
- CF-MESMO vs MESMO.



- ✓ CF-MESMO can achieve a higher-quality Pareto optimal set for the same total computation cost for ReRAM design evaluation;
- ✓ **Max-entropy based search** is highly efficient in terms of achieving high-quality Pareto Optimal.

- ✓ The **continuous-fidelity setting** in CF-MESMO can guarantee higher quality Pareto front with lower computation cost when compared to the single maximum fidelity algorithm MESMO.
- ✓ **Fidelity setting in CF-MESMO** makes the next candidate selection is based on the information gain per unit cost.

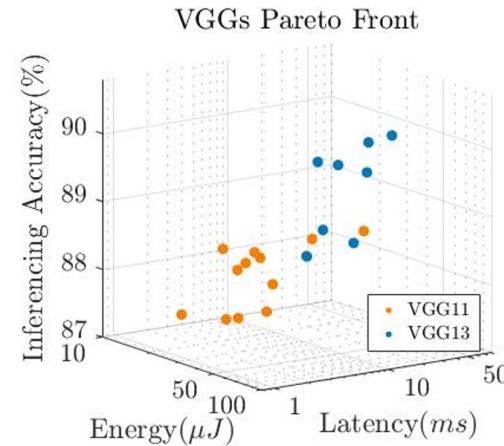
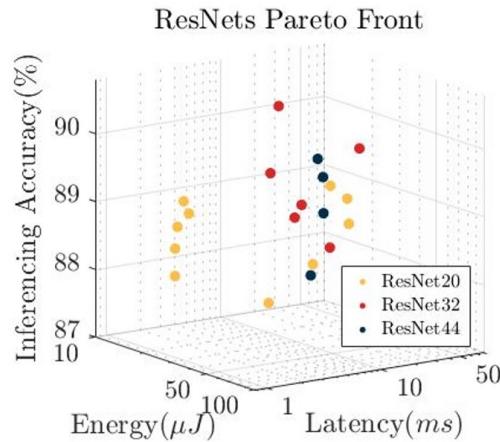
Pareto Optimal and Pareto Set



The number near the datapoint refers to the optimization iteration index.

- ✓ All the points shown on the figure lies on the Pareto front.
- ✓ We can avoid high-latency or high-energy design based on our criteria and budget.
- ✓ From the Pareto set, we can see that high-cell resolution setting or high-frequency setting appears in the Pareto front due to the ReSNA method.

More Results and Analysis



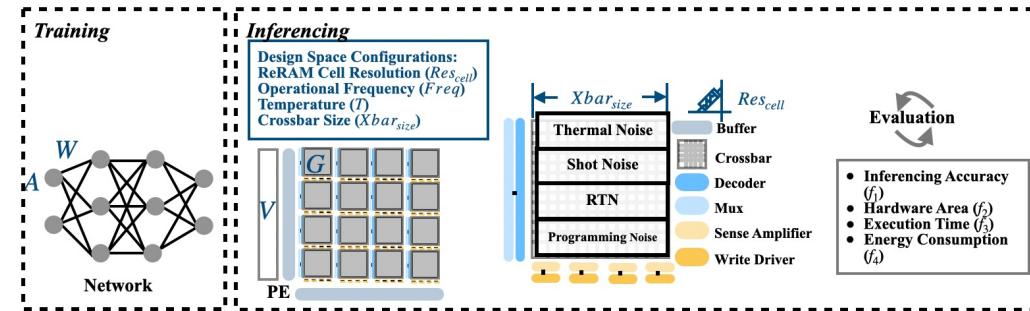
- ✓ There is a large overlap among the various clusters within the same network class.
- ✓ In designing ReRAM-based accelerators, we should first set the expected inferencing accuracy and hardware efficiency target and then choose the network using the Pareto front.

- ✓ The distribution of Pareto front for VGGs shows a different pattern.
- ✓ More details are discussed in our paper.

Summary: BO to configure ReRAM designs

ReSNA Training Method

Efficiently Find Pareto-Optimal Designs



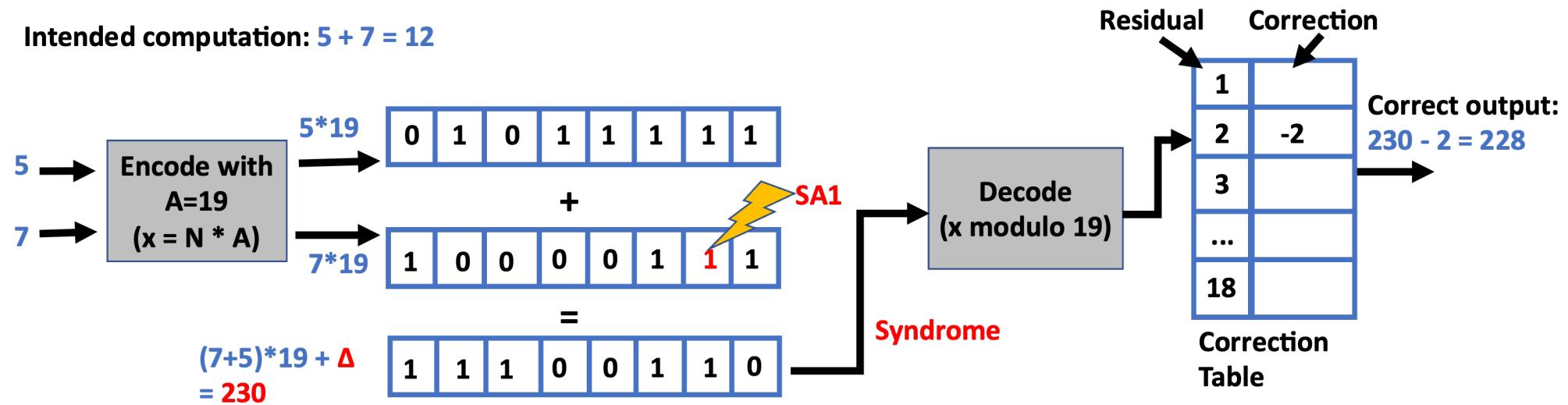
Build Robust and Efficient ReRAM-based System

CF-MESMO Method

Outline of Tutorial

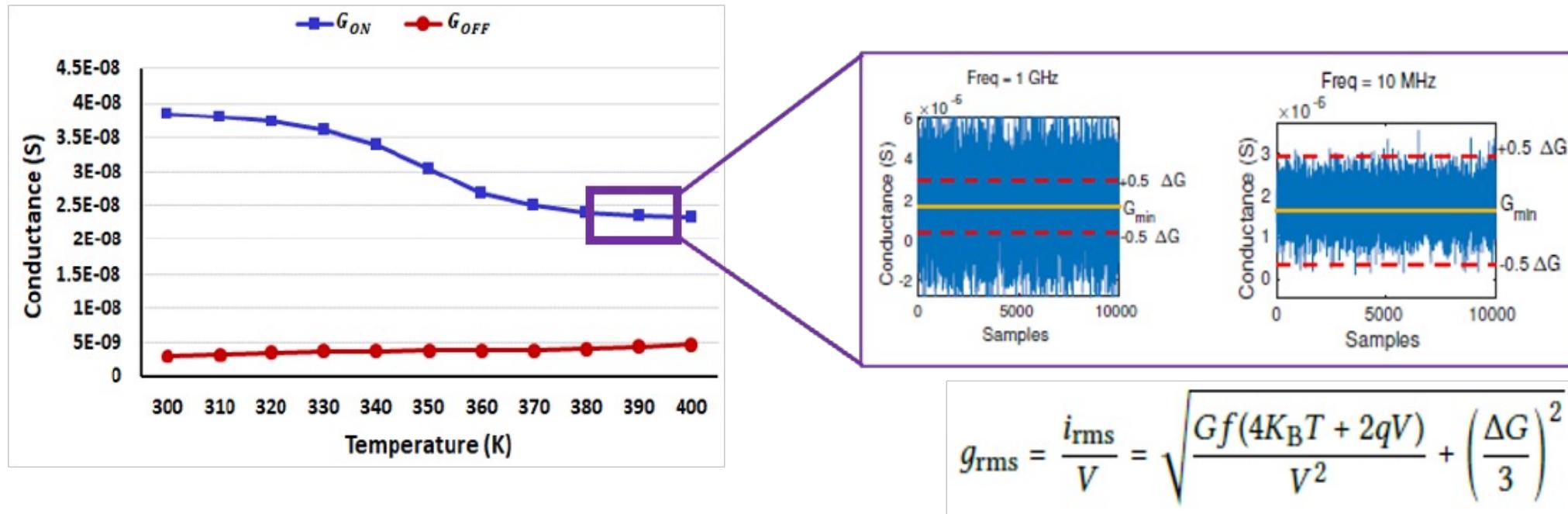
- Exponential growth of Deep Learning and Hardware Challenges
- Introduction to Deep Learning
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- **Methods to improve Reliability of ReRAMs**
- Summary and Promising Directions

Existing mitigation methods



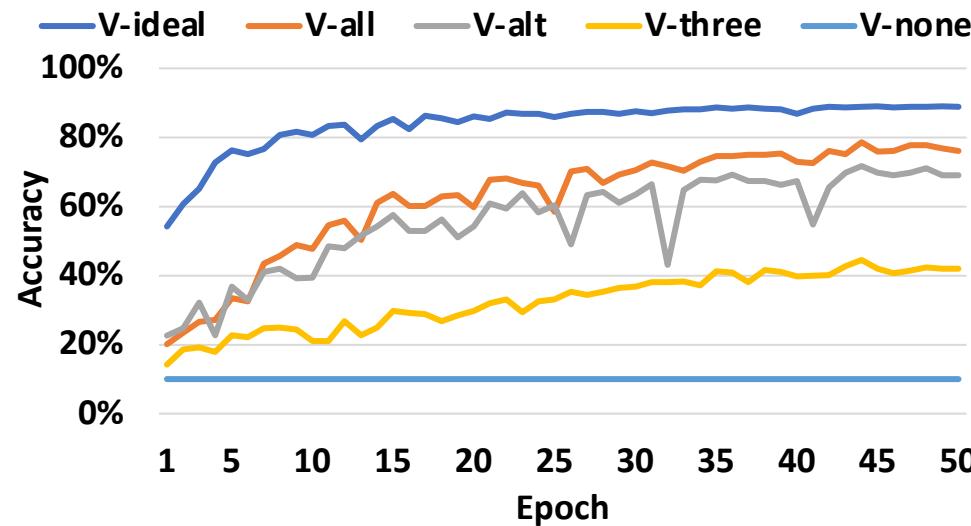
- Error correction codes, Use redundant hardware, retraining, etc.
- High area and power cost

Thermal noise



- Resistance changes with resistance
- Thermal noise can affect stored value
- Accuracy loss due to temperature

BN layers to reduce noise effect

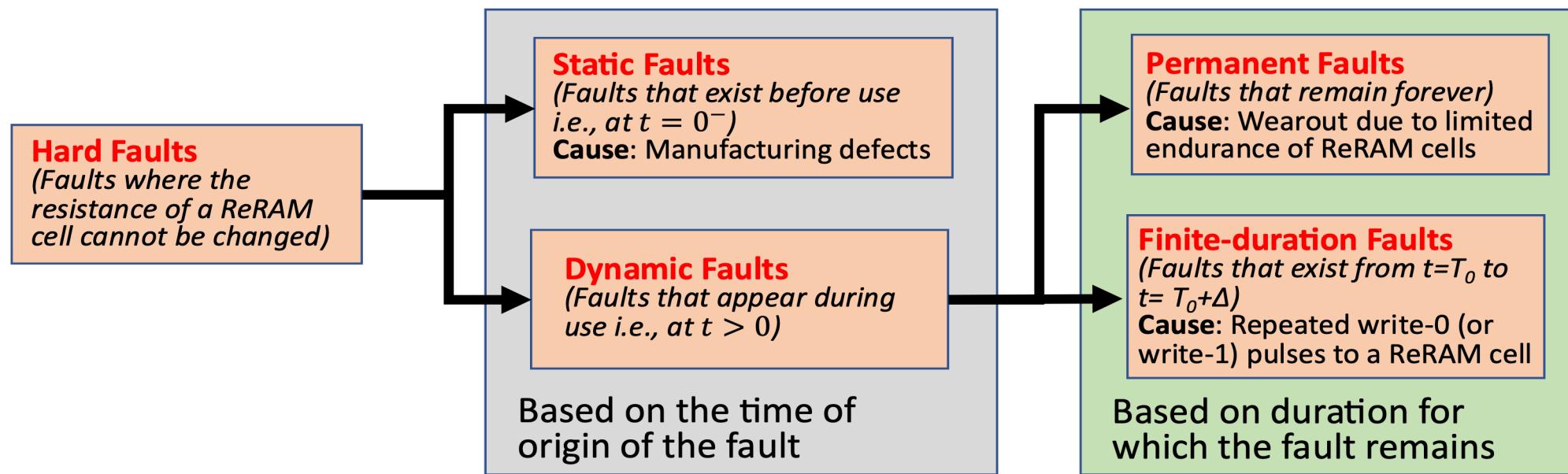


V-all: All layers are followed by normalization, V-alt: every alternate layer has normalization

V-three: every third layer has normalization, V-none: No layer has normalization

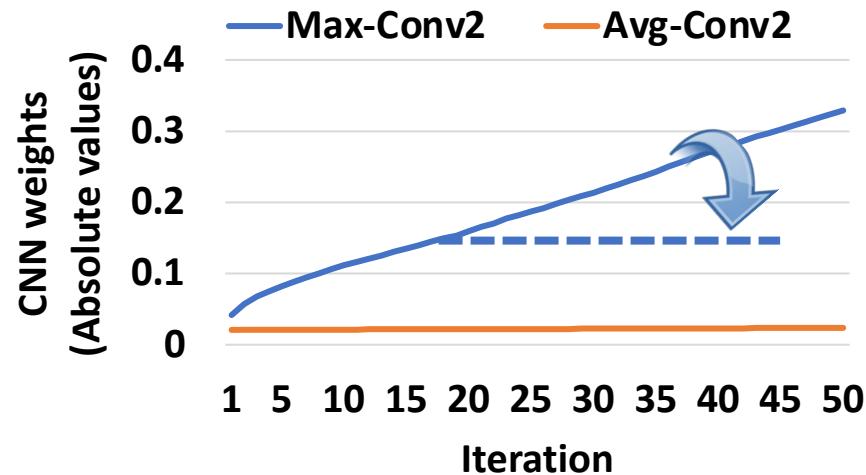
- Noise results in exploding weights and gradients
- Normalization layers can help reduce impact of noise
- Near-ideal accuracy even at 1GHZ, 100°C (worst case) when normalization is used

Different types of SAFs



- Different types of faults in ReRAMs
- Some are permanent, some can be short-lived
- Some appear during use

Weight clipping

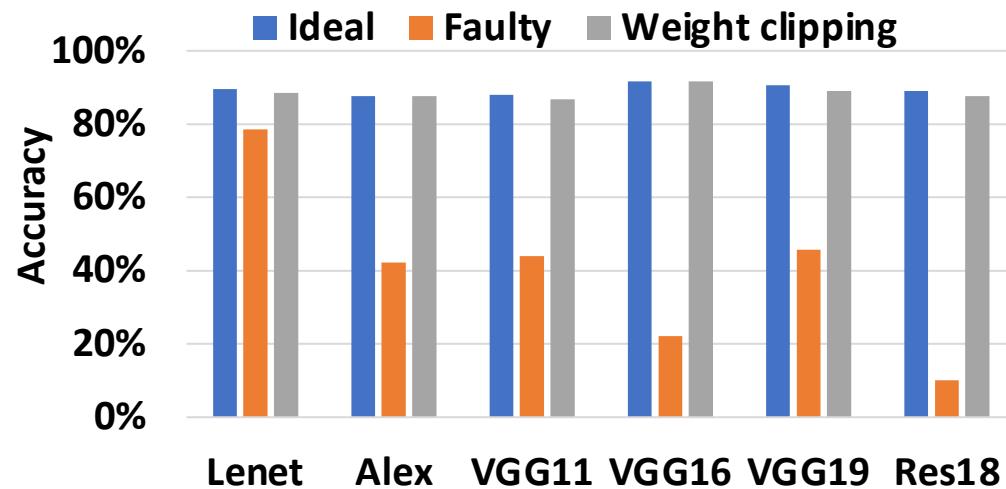


Clipping:

$$|w| = \begin{cases} |w|, & \text{if } |w| < \epsilon \\ \epsilon, & \text{otherwise} \end{cases}$$

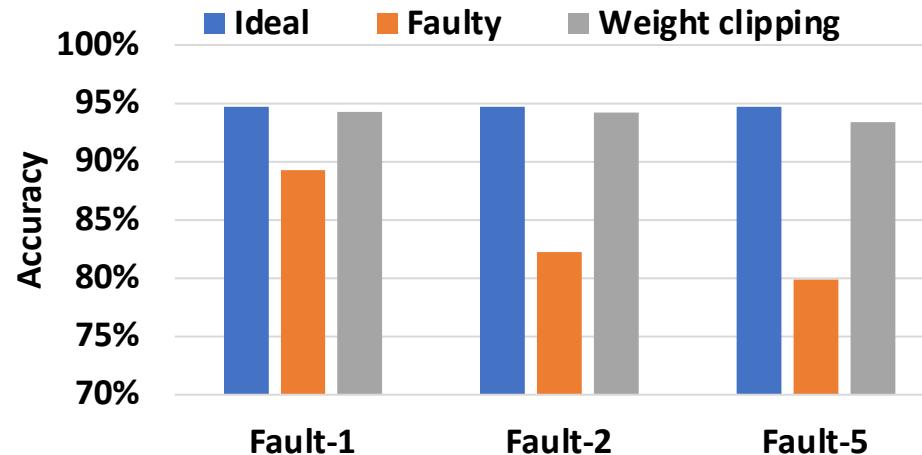
- Weight clipping acts as regularizer and reduces the sensitivity to various types of distortions
- Clipping can prevent large weights
 - Helps CNN training

Weight clipping for CNN



- Weight clipping can restore lost accuracy
- Prevent exploding gradients and activations
- Enables stable training

Weight clipping for GNNs



- Helps GNNs train in presence of faults
 - Near-ideal accuracy
- Hardware implementation
 - Mux and comparator needed
 - Low overhead implementation

Outline of Tutorial

- Exponential growth of Deep Learning and Hardware Challenges
- Introduction to Deep Learning
 - CNNs for images, RNNs for sequences, and GNNs for graph data
- ReRAM for Processing-in-Memory (PIM) to reduce data movement
- Heterogeneous GPU/ReRAM manycore systems for CNNs
- ReRAM based manycore systems for GNNs
- BO methods to configure ReRAM designs for improved Reliability
- Methods to improve Reliability of ReRAMs
- **Summary and Promising Directions**

Summary and Promising Directions

- ReRAMs have shown great success in accelerating DL workloads
- Significant progress on improving the reliability challenges of ReRAMs
- Much work needs to be done for ReRAM based systems for GNNs
- Exploring ReRAM based manycore systems for transformer architectures
- Exploring alternative non-volatile memories (e.g., Ferro-electric)
 - Understanding the power, performance, and reliability trade-offs w.r.t ReRAMs