Exploring Complexity Reduction for Learning in Deep Neural Networks

Sourya Dey

PhD Qualifying Exam

April 23rd, 2019

USC University of Southern California

Outline

- Introduction and Background
- Pre-Defined Sparsity
- Hardware Architecture
- Connection Patterns
- Dataset Engineering
- Model Search

Achieved Research Contributions

Proposed Research

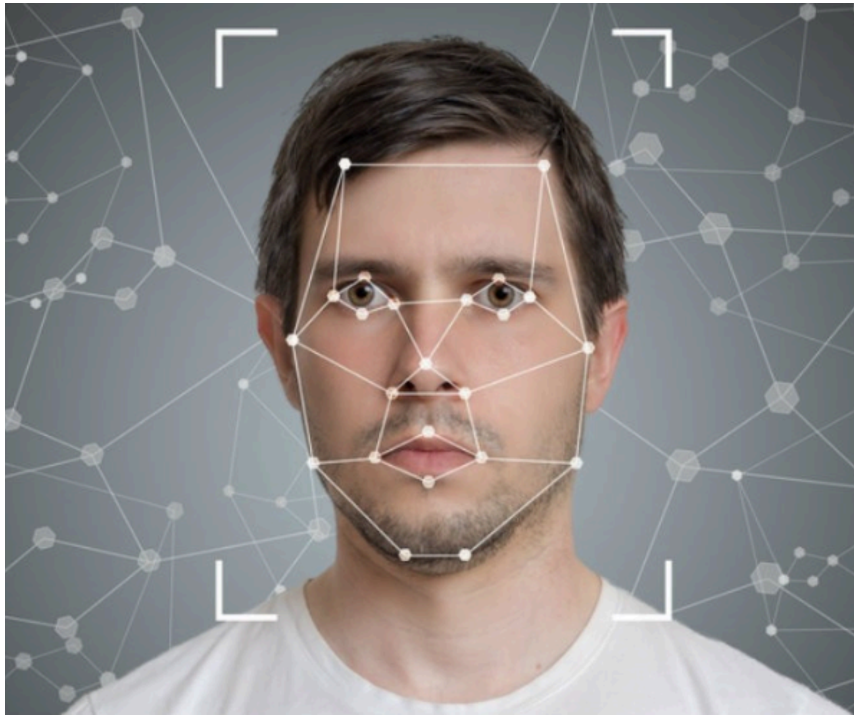
Outline

- Introduction and Background
- Pre-Defined Sparsity
- Hardware Architecture
- Connection Patterns
- Dataset Engineering
- Model Search

Introduction

Neural networks (NNs) are key machine learning technologies

- ➢ Artificial intelligence
- ➢ Self-driving cars
- ➢ Speech recognition
- ➢ Face ID
- ➢ and more smart stuff …



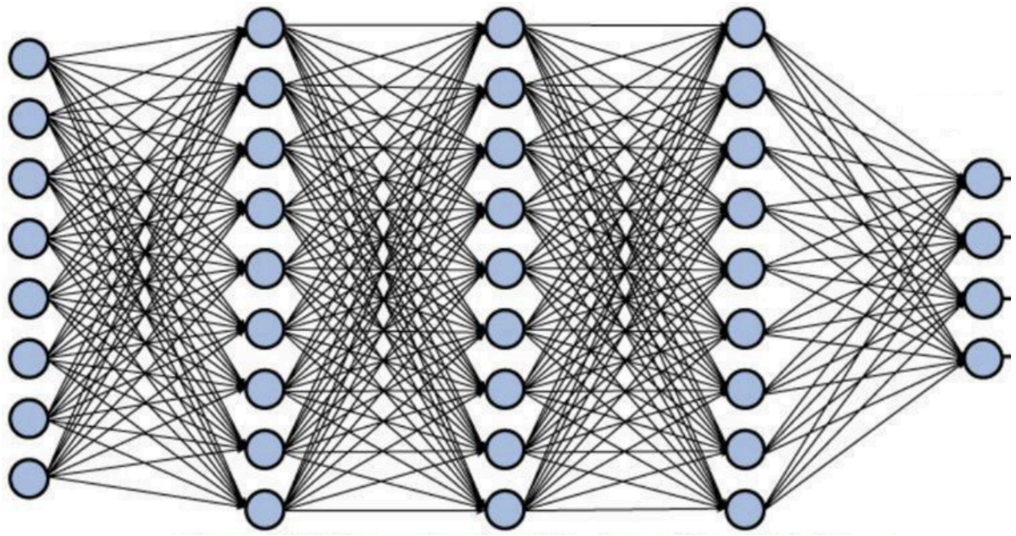
Overview

Modern neural networks suffer from parameter explosion

Training can take weeks on CPU

Cloud GPU resources are expensive

Our research reduces complexity of neural networks with minimal performance degradation



amazon web services

Microsoft Azure

Google Cloud Platform

Summary of Contributions

Achieved:

- **Pre-defined sparsity** to reduce complexity of neural networks
- **Hardware architecture** to leverage pre-defined sparsity
- Analyzing **connection patterns** and performance predicting measures
- Family of **synthetic datasets** on Morse code with tunable difficulty

Proposed:

- Better **pipelining** to improve hardware architecture
- **Architecture search** of low complexity neural networks

Notation

Multilayer Perceptron **(MLP)**

Junction 1

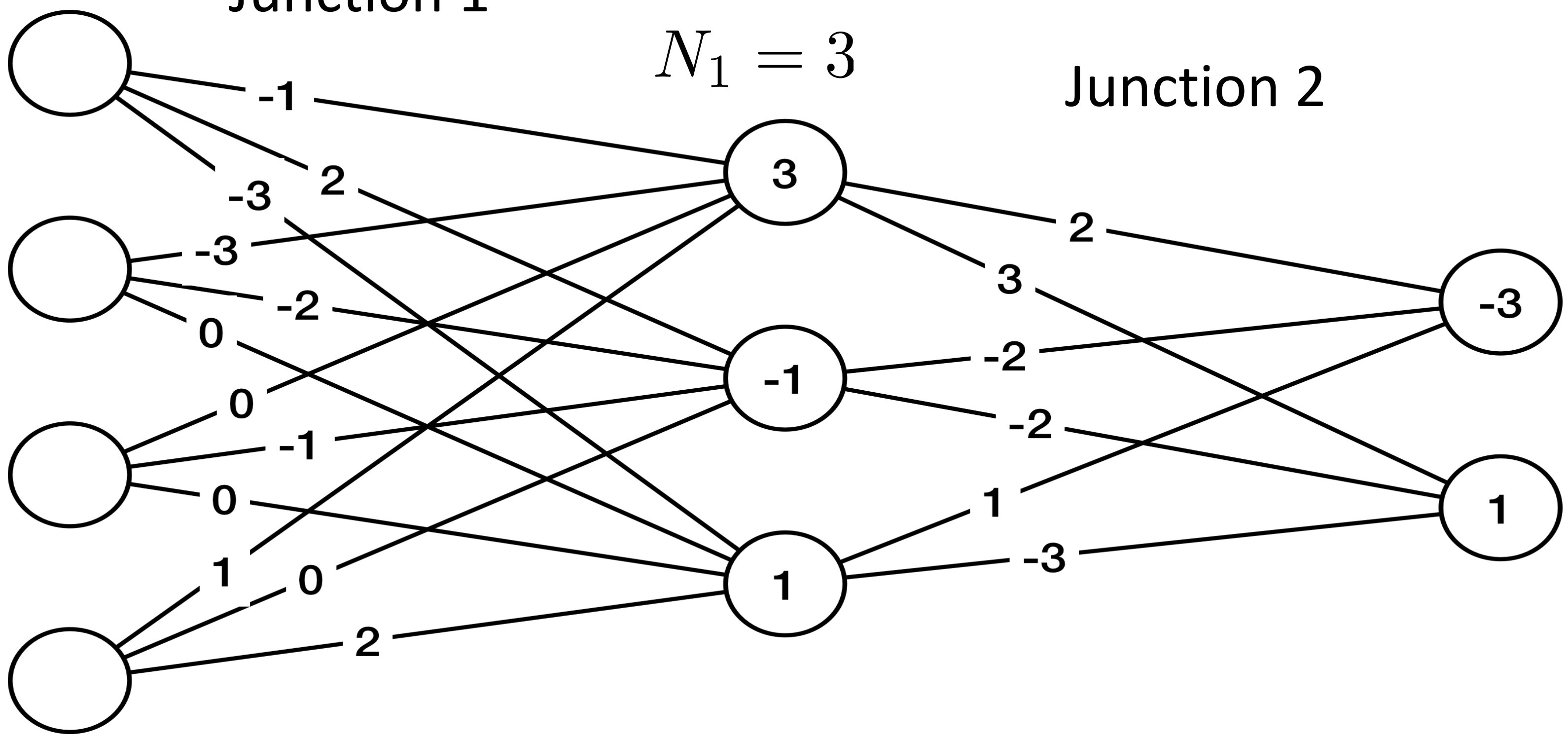
Layer 1

$N_1 = 3$

Junction 2

Layer 0

$N_0 = 4$



Layer L = 2

$N_2 = 2$

Trainable Parameters

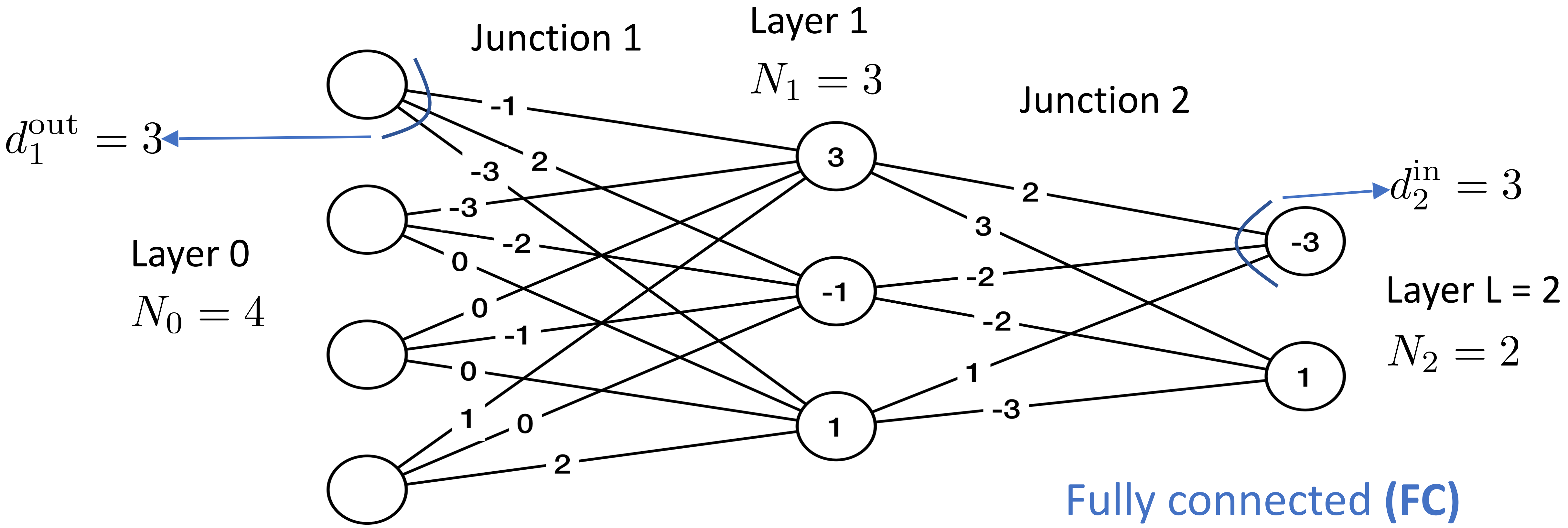
Weights
$$\boldsymbol{W}_1 = \begin{bmatrix} -1 & -3 & 0 & 1 \\ 2 & -2 & -1 & 0 \\ -3 & 0 & 0 & 2 \end{bmatrix}$$

Biases
$$\boldsymbol{b}_1 = \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}$$

$$\boldsymbol{W}_2 = \begin{bmatrix} 2 & -2 & 1 \\ 3 & -2 & -3 \end{bmatrix} \quad \boldsymbol{b}_2 = \begin{bmatrix} -3 \\ 1 \end{bmatrix}$$

Notation

Multilayer Perceptron **(MLP)**



Fully connected **(FC)**

Trainable Parameters

Weights $\boldsymbol{W}_1 = \begin{bmatrix} -1 & -3 & 0 & 1 \\ 2 & -2 & -1 & 0 \\ -3 & 0 & 0 & 2 \end{bmatrix}$ Biases $\boldsymbol{b}_1 = \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}$

$\boldsymbol{W}_2 = \begin{bmatrix} 2 & -2 & 1 \\ 3 & -2 & -3 \end{bmatrix}$ $\boldsymbol{b}_2 = \begin{bmatrix} -3 \\ 1 \end{bmatrix}$

Neural Networks Operations for Classification

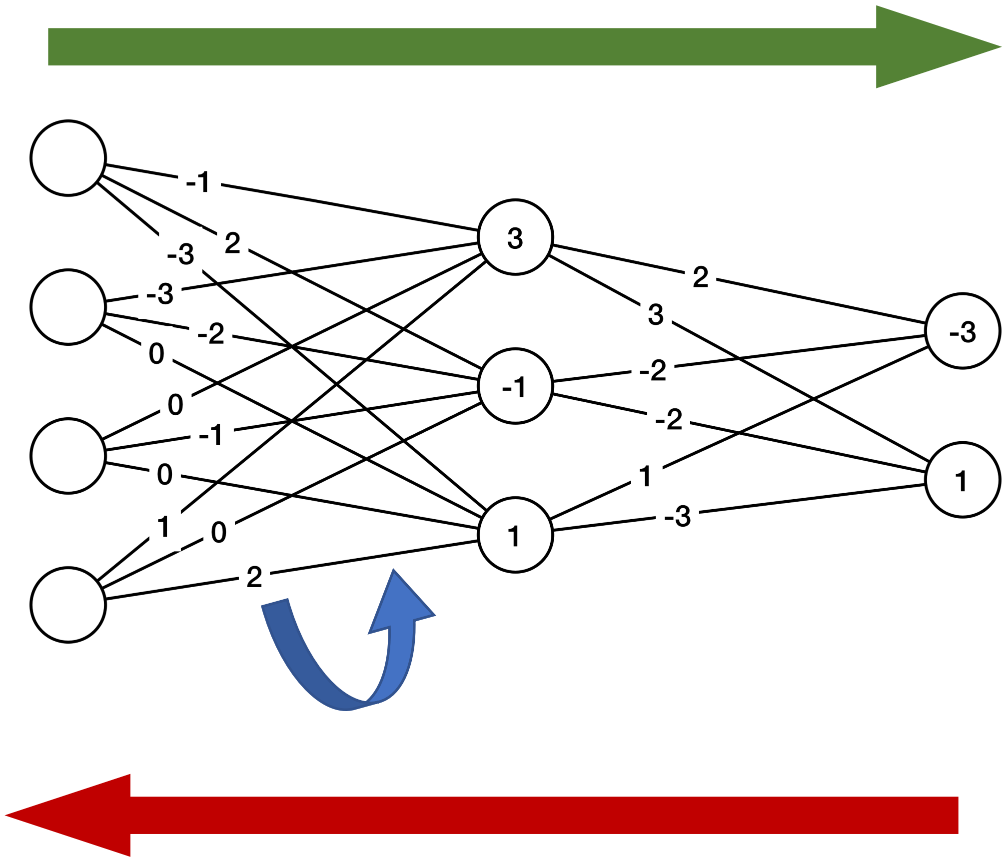
Training (*training data*)

- Feedforward (FF)
- Backpropagation (BP)
- Update parameters (UP)

Inference (*validation and test data*)

- Feedforward (FF) only

Test data performance used as metric for goodness of network



Feedforward (FF)

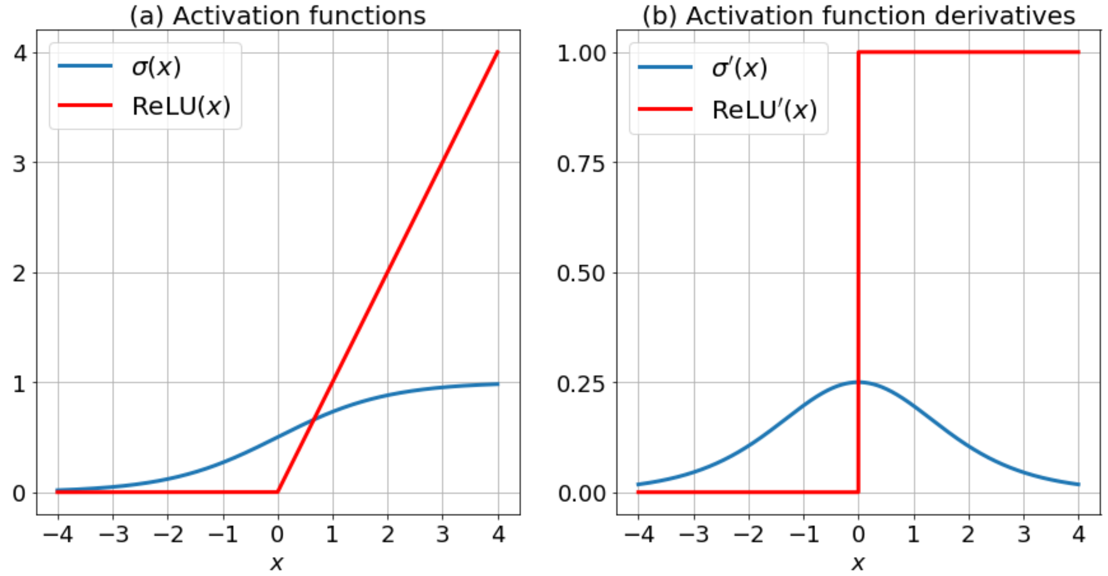
Previous layer activation
(Starts from input features $\boldsymbol{a}_0$)

Linear output $$\boldsymbol{s}_i = \boldsymbol{W}_i \boldsymbol{a}_{i-1} + \boldsymbol{b}_i$$

Activation output $$\boldsymbol{a}_i = \boldsymbol{h}\left(\boldsymbol{s}_i\right)$$

Non-linear activation function
ReLU / Sigmoid for hidden layers
Softmax for output layer

Activation derivative $$\boldsymbol{h}_i' = \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{s}_i}$$



Backpropagation (BP)

Ground truth labels
Typically one-hot for classification

Cross-entropy Cost

$$C = -\sum_{i=1}^{N_L} y^{(i)} \ln a_L^{(i)}$$

$$\boldsymbol{y} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Delta (output layer)

$$\boldsymbol{\delta}_L = \boldsymbol{a}_L - \boldsymbol{y}$$

Delta (intermediate layers)

$$\boldsymbol{\delta}_i = \left(\boldsymbol{W}_{i+1}^T \boldsymbol{\delta}_{i+1}\right) \circ \boldsymbol{h}_i'$$

Hadamard product
(element-wise multiplication)

Update (UP)

Training sample number

$$\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \frac{\eta}{M} \sum_{m=1}^{M} \left(\underbrace{\boldsymbol{\delta}_i \boldsymbol{a}_{i-1}^T}_{\text{Gradient of cost w.r.t weights}} \right)^{[m]}$$

Learning Rate Hyperparameter

$$\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i - \frac{\eta}{M} \sum_{m=1}^{M} \left(\boldsymbol{\delta}_i \right)^{[m]}$$

Batch size

Gradient of cost w.r.t biases

The Complexity Conundrum

➢ Storage Complexity - Dominated by weights

A typical fully connected MLP for classifying MNIST handwritten digits has ~10^5 weights

➢ Computational Complexity - Also dominated by weights

FF $$\sum_{\forall i,j} W_{ij} a_j$$

BP $$\sum_{\forall i,j} W_{ij} \delta_i$$

UP $$W_{ij} - \eta \nabla_{W_{ij}} C \quad \forall i,j$$

All the weights are used in all 3 operations

Existing methods to reduce Complexity

Algorithms

- Gong 2014 – Vector quantization
- Chen 2015 – HashedNets
- Sindhwani 2015 – Structured transforms
- Srinivas 2017 – Special regularizers
- Aghasi 2017 – Net-trim

ASIC Implementations

- Chen 2014 – Diannao
- Han 2016 – Efficient Inference Engine
- Reagen 2016 – Minerva
- Zhang 2016 – Cambricon-X
- Chen 2017 – Eyeriss

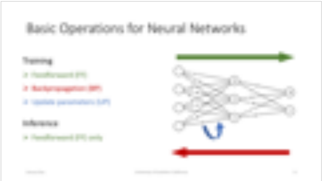
FPGA Implementations

- Courbariaux 2016 - Binarized nets
- Albericio 2016 – Cnvlutin
- Suda 2016 – Open-CL based
- Ma 2018 – ALAMO

Training Focused

- *Girones 2005* – Pipelined on-line BP
- Gomperts 2011 – Parametrized FPGA-based NNs
- Wang 2017 – DLAU

These reduce parameters during inference, but training complexity remains intensive



These focus on training, but do not delete parameters

Outline

- Introduction and Background
- Pre-Defined Sparsity
- Hardware Architecture
- Connection Patterns
- Dataset Engineering
- Model Search

Achieved Research Contributions

Our Work: Pre-defined Sparsity

Pre-define a sparse connection pattern prior to training

Use this sparse network for both training and inference

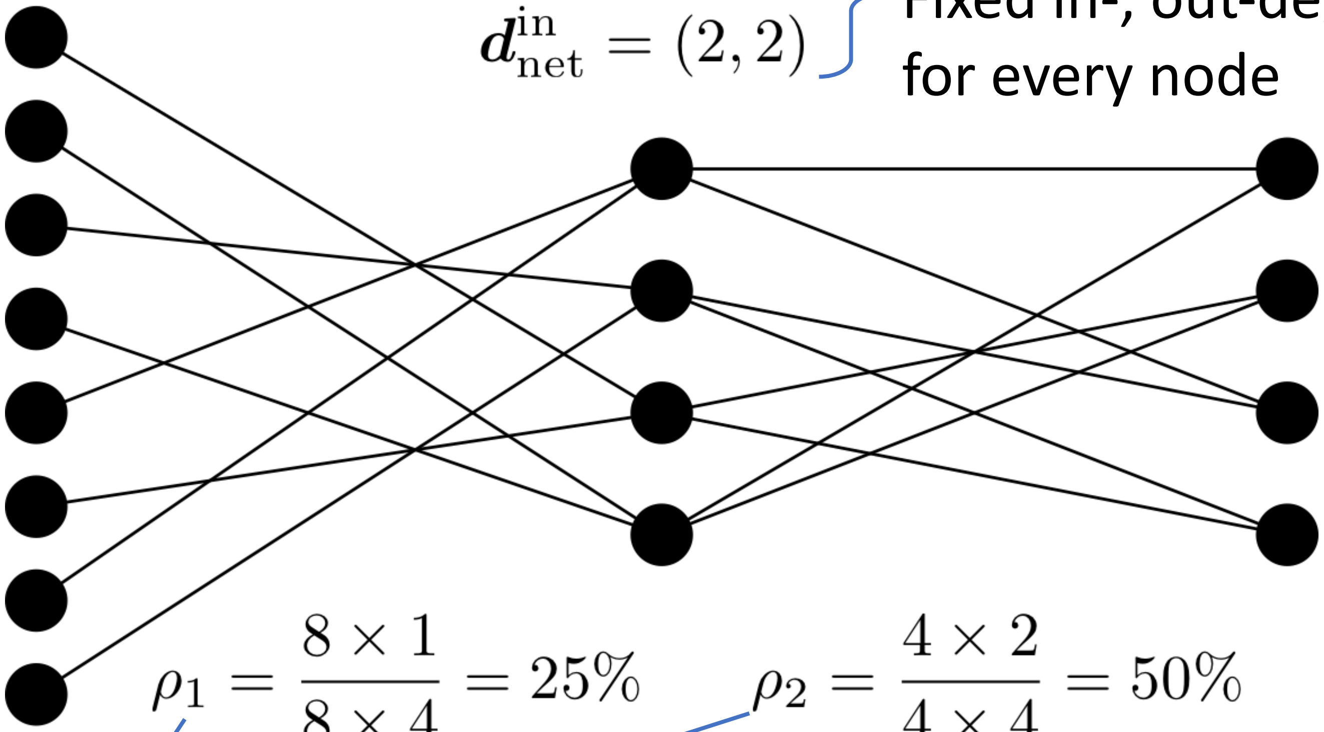
Reduced training and inference complexity

$$\boldsymbol{N}_{\text{net}} = (8, 4, 4)$$

$$\boldsymbol{d}^{\text{out}}_{\text{net}} = (1, 2)$$

$$\boldsymbol{d}^{\text{in}}_{\text{net}} = (2, 2)$$

Structured Constraints:
Fixed in-, out-degrees for every node



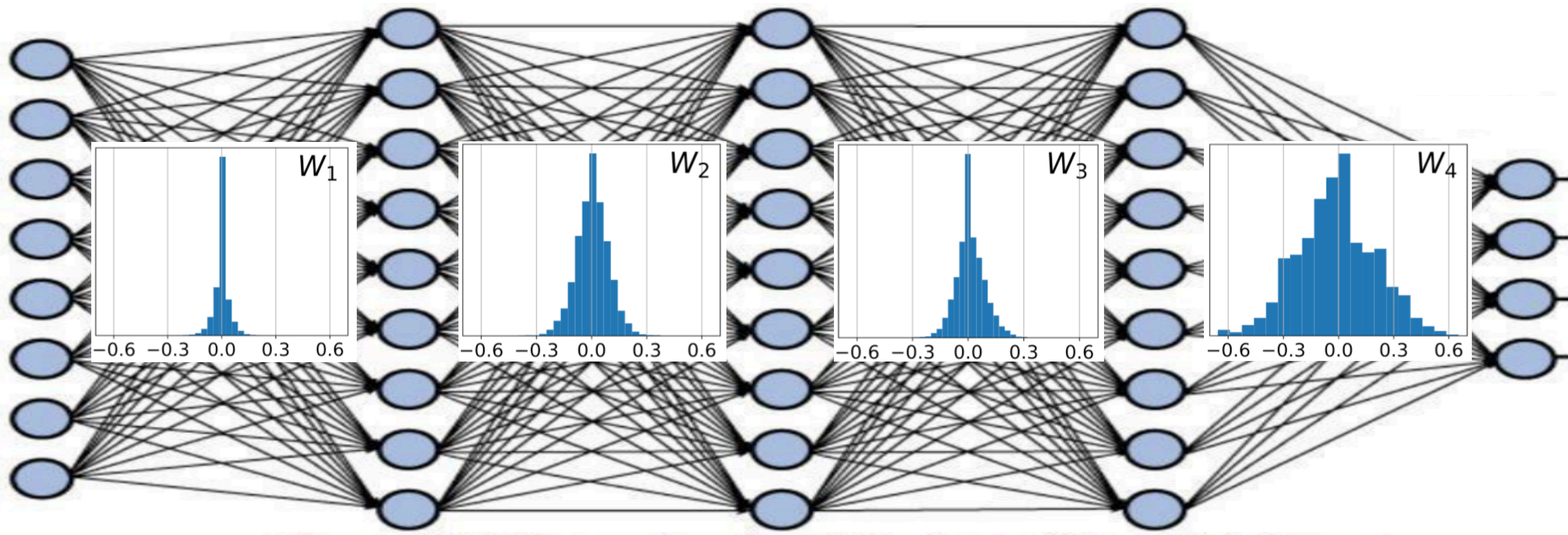
$$\rho_1 = \frac{8 \times 1}{8 \times 4} = 25\% \qquad \rho_2 = \frac{4 \times 2}{4 \times 4} = 50\%$$

Junction Densities

$$\rho_{\text{net}} = \frac{8 + 8}{32 + 16} = 33\%$$

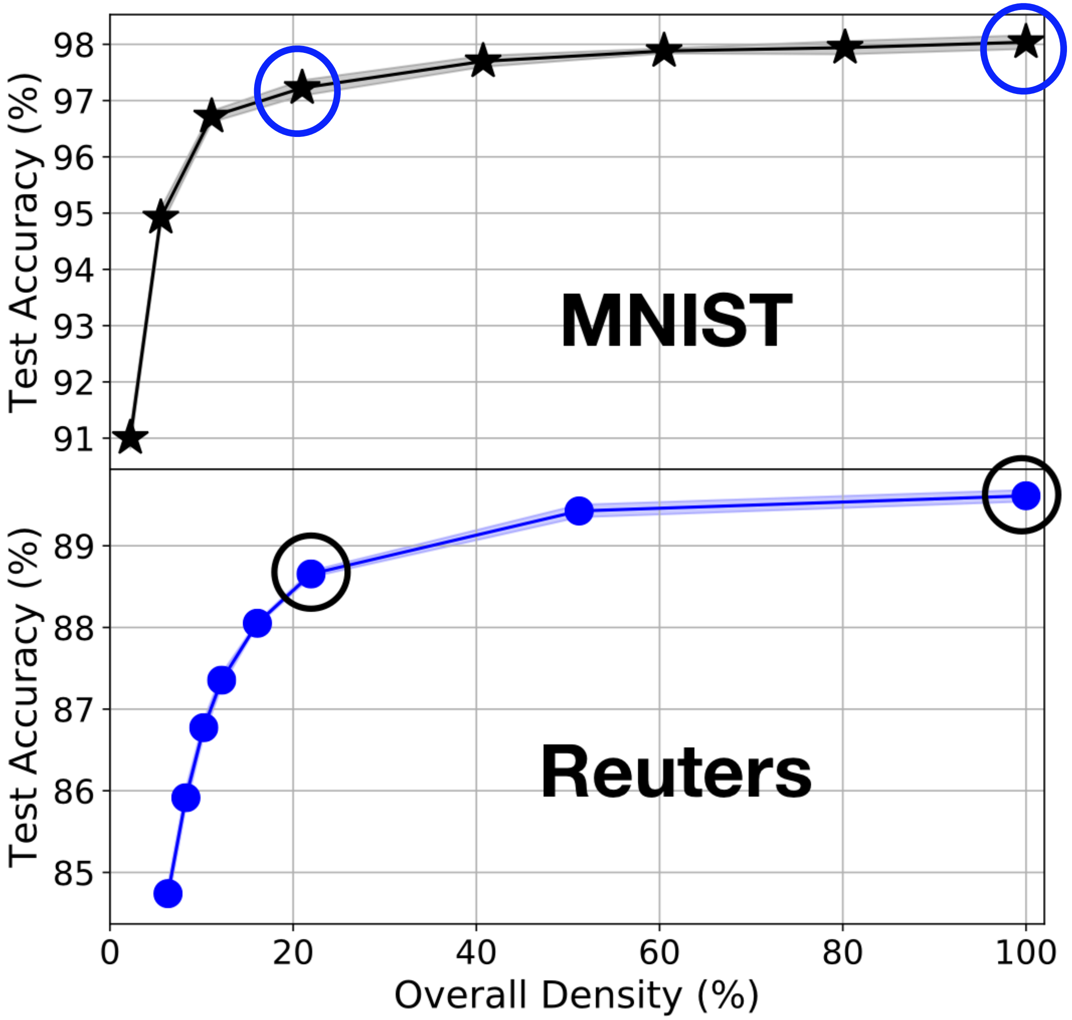
Overall Density

Motivation behind pre-defined sparsity



In a FC network, most weights are very small in magnitude after training

Performance of pre-defined sparsity

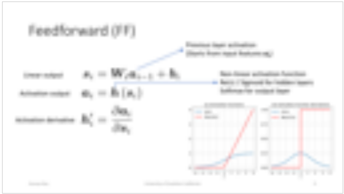


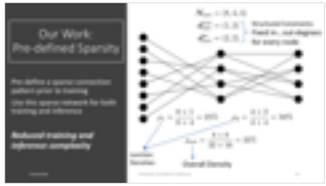
Starting with an MLP with only 20% of parameters compared to fully connected : Classification accuracy reduction on test data is <1%

Computational Savings

$$s_i^{(j)} = \sum_{f=1}^{d_i^{\text{in}}} W_i^{(j,k_f)} a_{i-1}^{(k_f)} + b_i^{(j)}$$

In-degree summations for each node in FF





$$\delta_i^{(j)} = h'^{(j)}_i \left(\sum_{f=1}^{d_i^{\text{out}}} W_{i+1}^{(k_f,j)} \delta_{i+1}^{(k_f)} \right)$$

Out-degree summations for each node in BP

$$W_i^{(j,k)} \leftarrow W_i^{(j,k)} - \eta a_{i-1}^{(k)} \delta_i^{(j)}$$

Only node pairs (j,k) which have weight connecting them in UP

For all 3 operations – FF, BP, UP – only use weights which are present

Designing pre-defined sparse networks

A pre-defined sparse connection pattern is a hyperparameter to be set prior to training

How can it be set?

Designing pre-defined sparse networks

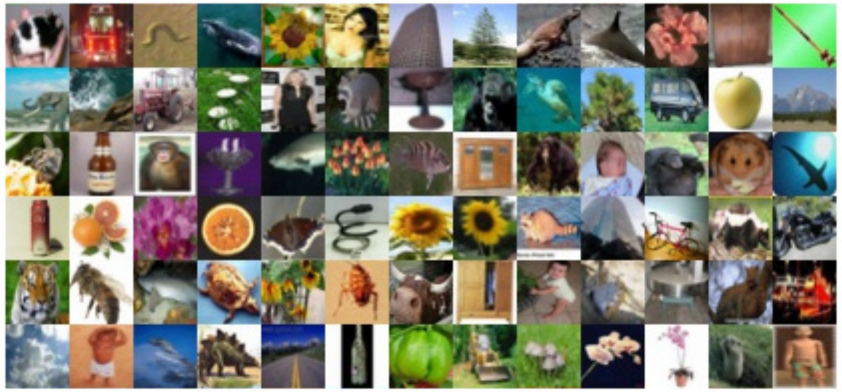
A pre-defined sparse connection pattern is a hyperparameter to be set prior to training

How can it be set?

We experimented on several datasets:

- ➢ MNIST handwritten digits
- ➢ Reuters RCV1 corpus of newswire articles
- ➢ TIMIT speech corpus (only MLP portion)
- ➢ CIFAR-10 and -100 images (CNN + MLP)
- ➢ Morse Code symbols (described later)

Topic	Description	Distribution
c15	Performance	149,359
c151	Accounts/earnings	81,201
c152	Comment/forecasts	72,910
ccat	Corporate/industrial	372,099
ecat	Economics	116,207
gcat	Government/social	232,032
m14	Commodity markets	84,085
mcat	Markets	197,813

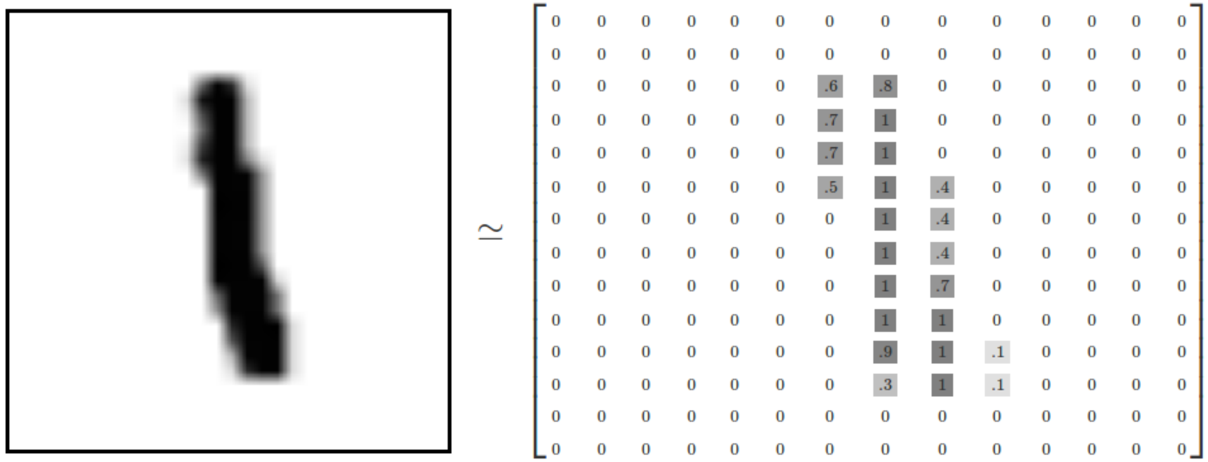


Pic courtesy: https://www.researchgate.net/publication/3454183_Hybrid_Neural_Document_Clustering_Using_Guided_Self-Organization_and_Wordnet/figures?lo=1

Dataset Redundancy

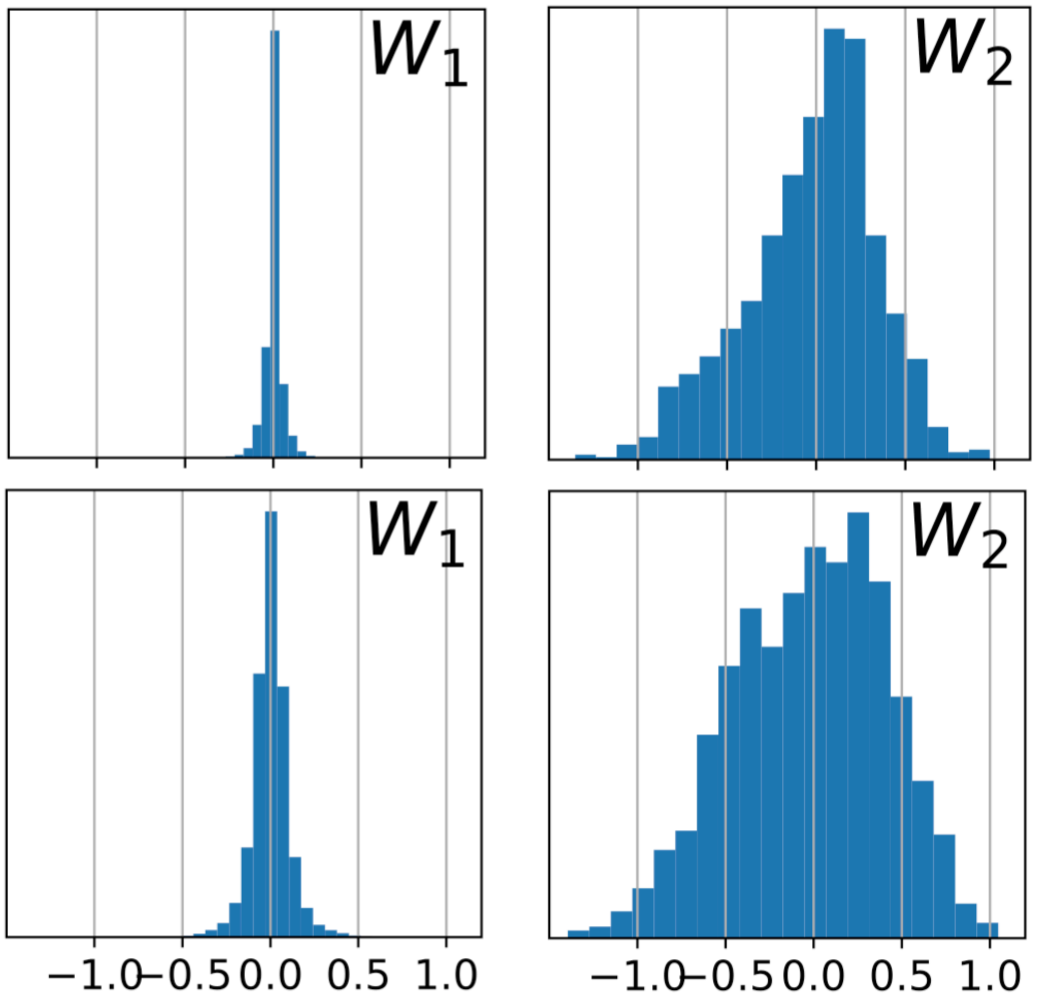
Most datasets have too many features => Can be reduced

- MNIST:
  - Default: 784 features (image pixels)
  - Principal component analysis to reduce to 200 => Less redundancy
- Reuters:
  - Default: Collect 2000 tokens (word snippets) as features from each article
  - Can be reduced to 400 => Less redundancy
- TIMIT:
  - Default: Collect 39 MFCCs as features
  - Decrease by 3x to 13 => Less redundancy
  - Increase by 3x to 117 => More redundancy
- CIFAR:
  - Default: Pre-process using a deep 9-layer CNN
  - Simplify to a 2-layer CNN => Less redundancy



Pic courtesy: https://tensorflow.rstudio.com/tensorflow/articles/tutorial_mnist_beginners.html

Effect of redundancy on sparsity



MNIST with default
784 features

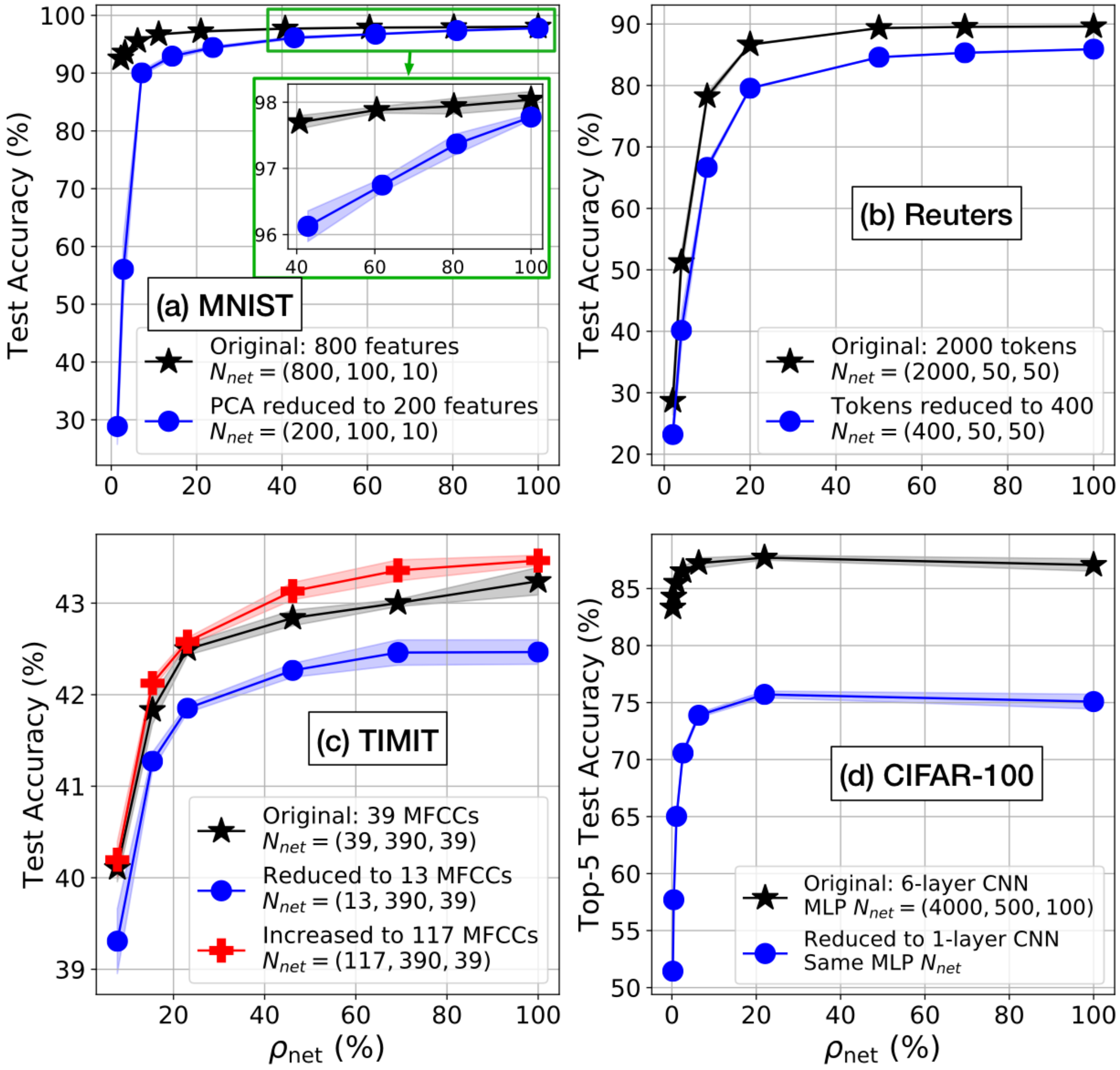
Less redundancy => Less sparsification possible

MNIST reduced to
200 features

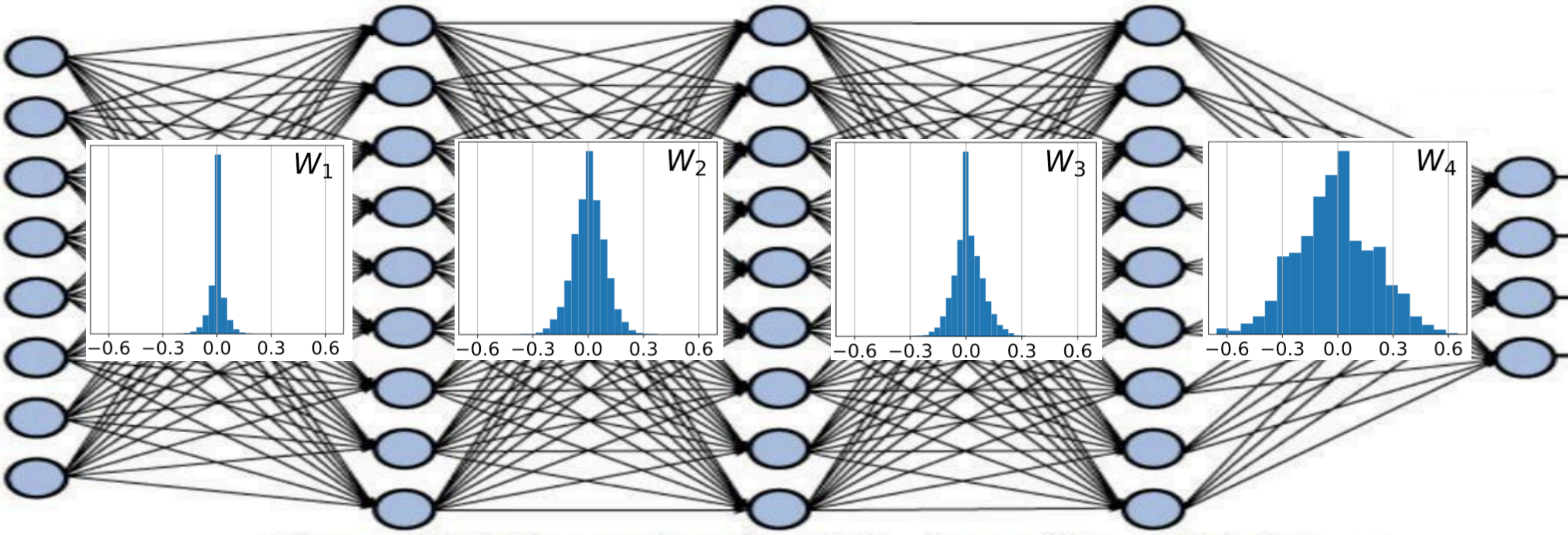
Wider spread

Effect of redundancy on sparsity

Reducing redundancy leads to performance starting to degrade at higher densities



Individual junction densities



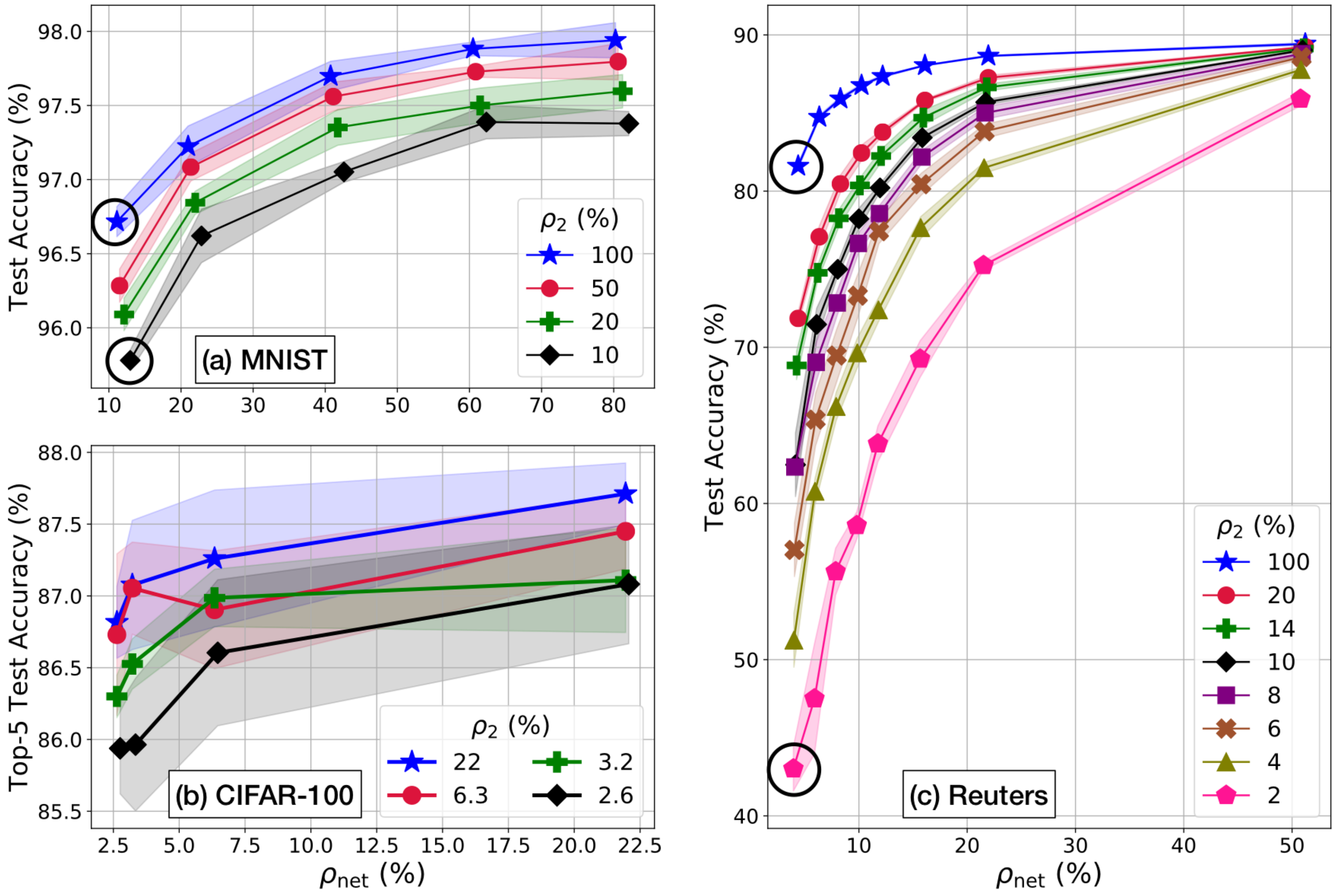
Latter junctions (closer to the output) need to be denser

Individual junction densities

Each curve keeps ρ_2 fixed and varies ρ_{net} by varying ρ_1

For the same ρ_{net}, $\rho_2 > \rho_1$ improves performance

Similar trends observed for deeper networks, with few exceptions



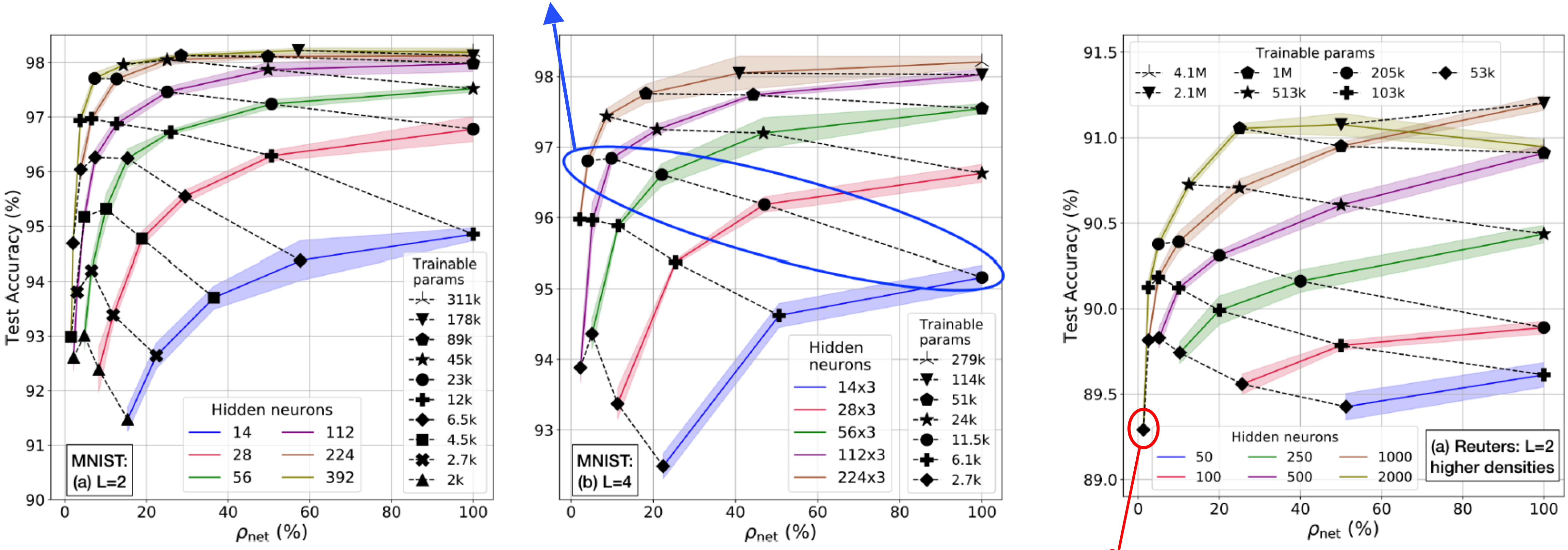
‘Large sparse’ vs ‘small dense’ networks

A sparser network with more nodes will outperform a denser network with less nodes, when both have same number of trainable parameters (weights+biases)

…unless density of the larger network goes lower than a critical density (problem dependent)

'Large sparse' vs 'small dense' networks

Networks with same number of parameters go from bad to good as #nodes in hidden layers is increased



Below critical density

Regularization

$$C(\boldsymbol{w}) = C_0(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2$$

Regularized cost

Original unregularized
cost (like cross-entropy)

Regularization term

Regularization

$$C(\boldsymbol{w}) = C_0(\boldsymbol{w}) + \lambda \left\| \boldsymbol{w} \right\|_2^2$$

Regularized cost

Original unregularized cost (like cross-entropy)

Regularization term

Overall Density	λ
100 %	1.1×10^{-4}
40 %	5.5×10^{-5}
11 %	0

Example for MNIST 2-junction networks

Pre-defined sparse networks need smaller λ (as determined by validation)

Pre-defined sparsity reduces the overfitting problem stemming from over-parametrization in big networks

Summary of pre-defined sparsity – Trends and design guidelines

Most networks can be significantly sparsified!

Exploits redundancy in dataset

Later junctions need more density

'Large and sparse' networks are better than 'small and dense' networks

Alternative to regularization

... these tie in with proposed research on model search

Outline

- Introduction and Background
- Pre-Defined Sparsity
- **Hardware Architecture**
- Connection Patterns
- Dataset Engineering
- Model Search

Achieved Research Contributions

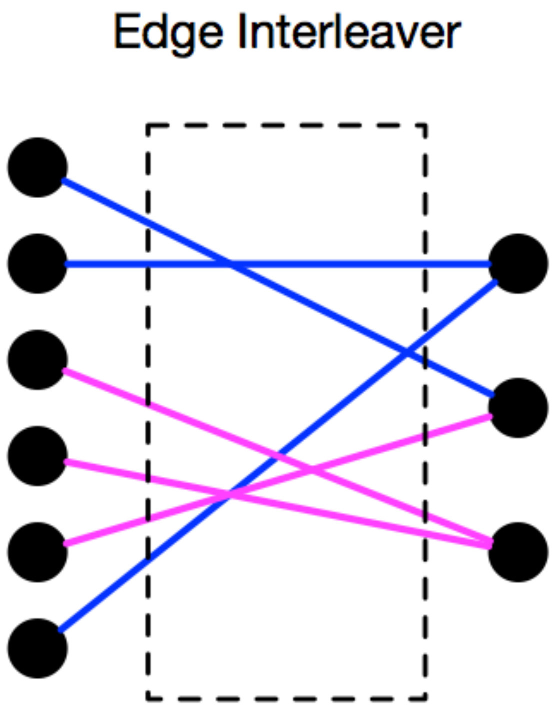
Hardware Architecture

We built a customized hardware architecture to leverage pre-defined sparsity

Key highlights:

- ➢ Edge-based
- ➢ Customizable amount of parallelism
- ➢ Clash free memory accesses
- ➢ Pipelined processing

Degree of parallelism z



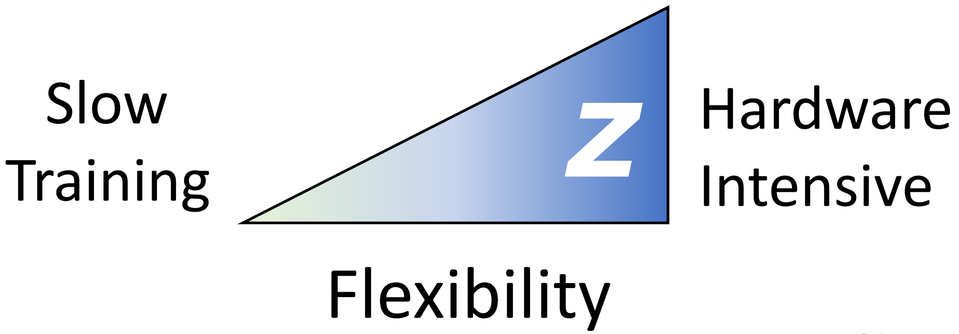
Example $z_i = 3$

z_i = #edges (weights) processed in parallel in junction i

$$\text{\#clock cycles } (C_i) \text{ to process junction i } = \frac{\text{\#weights } \left|\boldsymbol{W}_i\right|}{z_i}$$

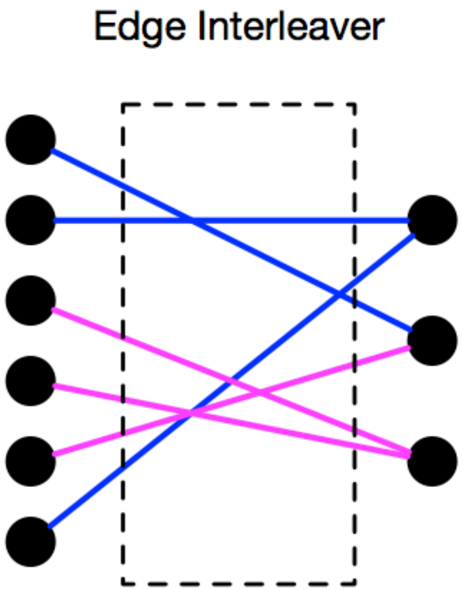
Computational complexity depends only on z_i

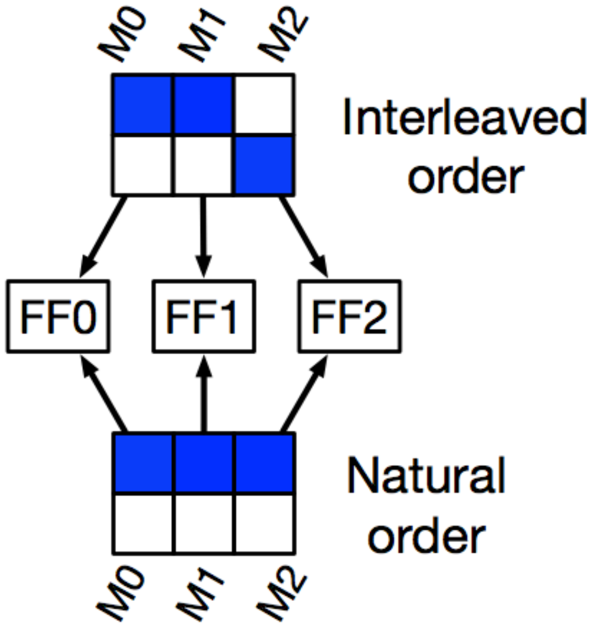
Decouple hardware required from network complexity



Memory organization and clash freedom

z_i memories for storing each variable – a, h′, δ, W, b – in each junction





Left side nodes are accessed in arbitrary order due to interleaving

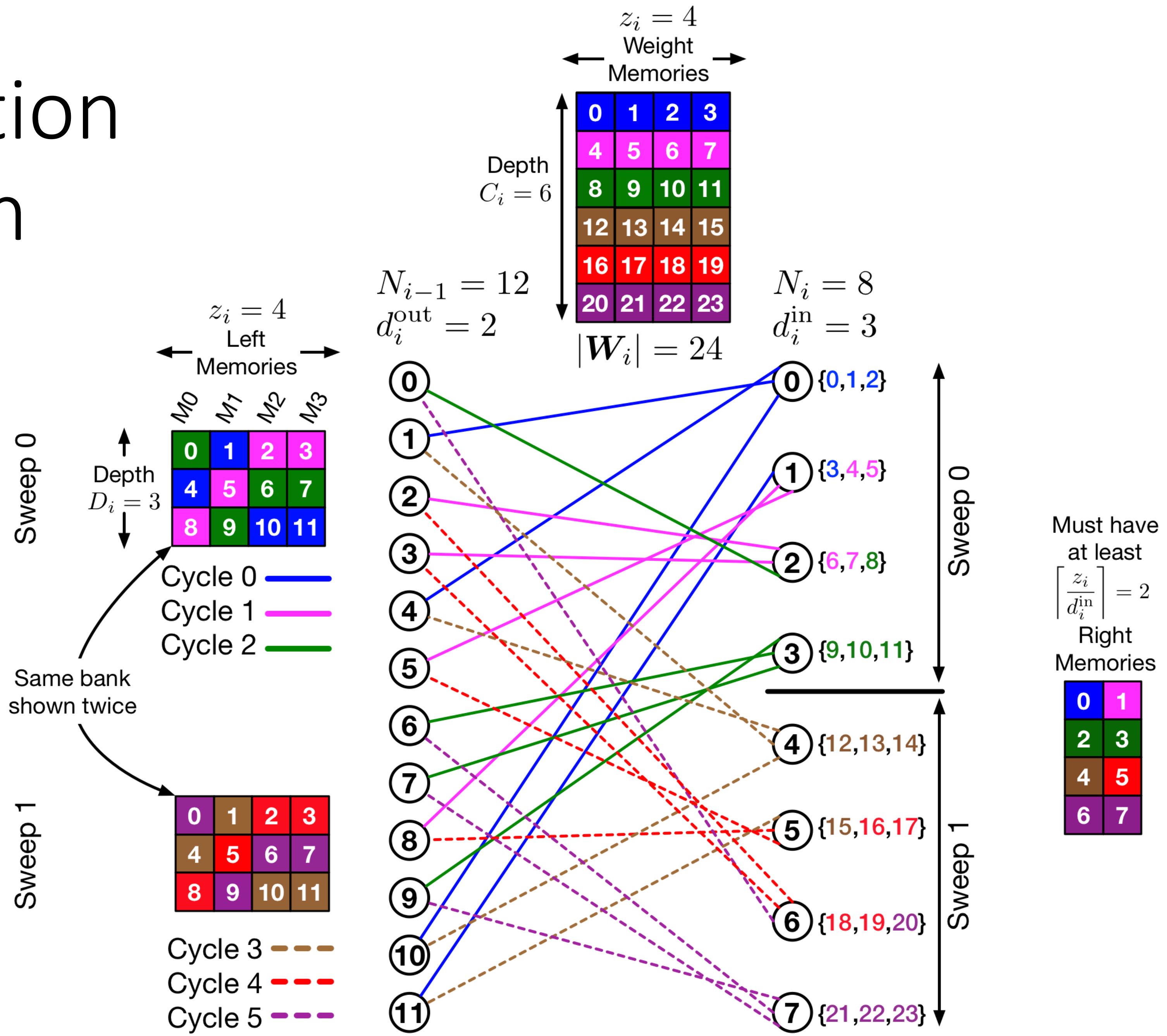
Weights are accessed one row at a time (natural order)

Must access each memory no more than once per clock cycle, otherwise clash => processing stall

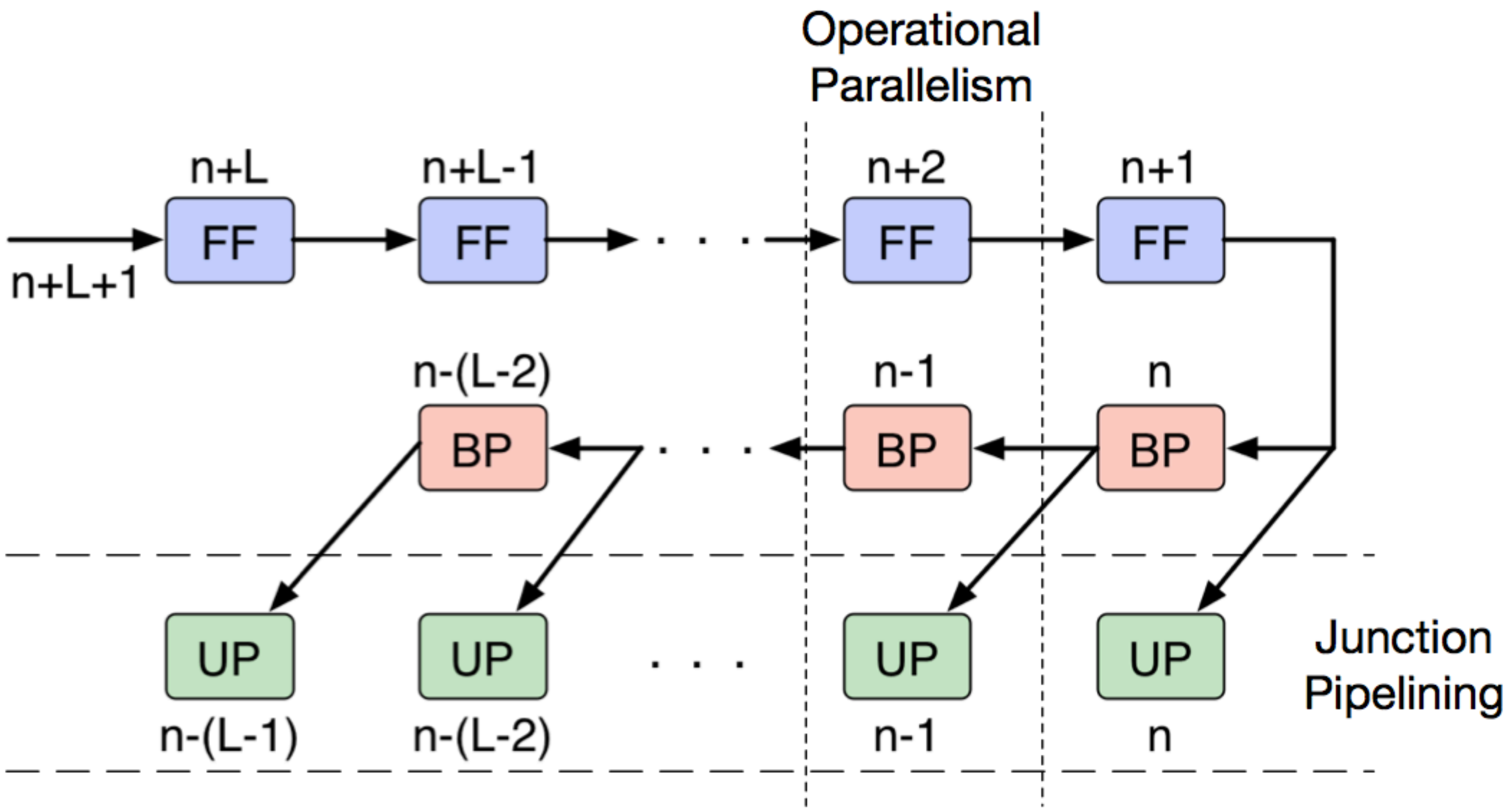
Example z_i = 3

Memory organization of a single junction

- ➢ z_i = 4 weights accessed per cycle
- ➢ Must access all 4 left memories exactly once per cycle for clash-freedom
- ➢ After D_i = 3 cycles, all left nodes are accessed once => 1 sweep
- ➢ Repeat for d_i^{out} = 2 sweeps to access all weights
- ➢ At most 2 right nodes accessed per cycle => At least 2 right memories required for clash-freedom



Parallel and Pipelined processing

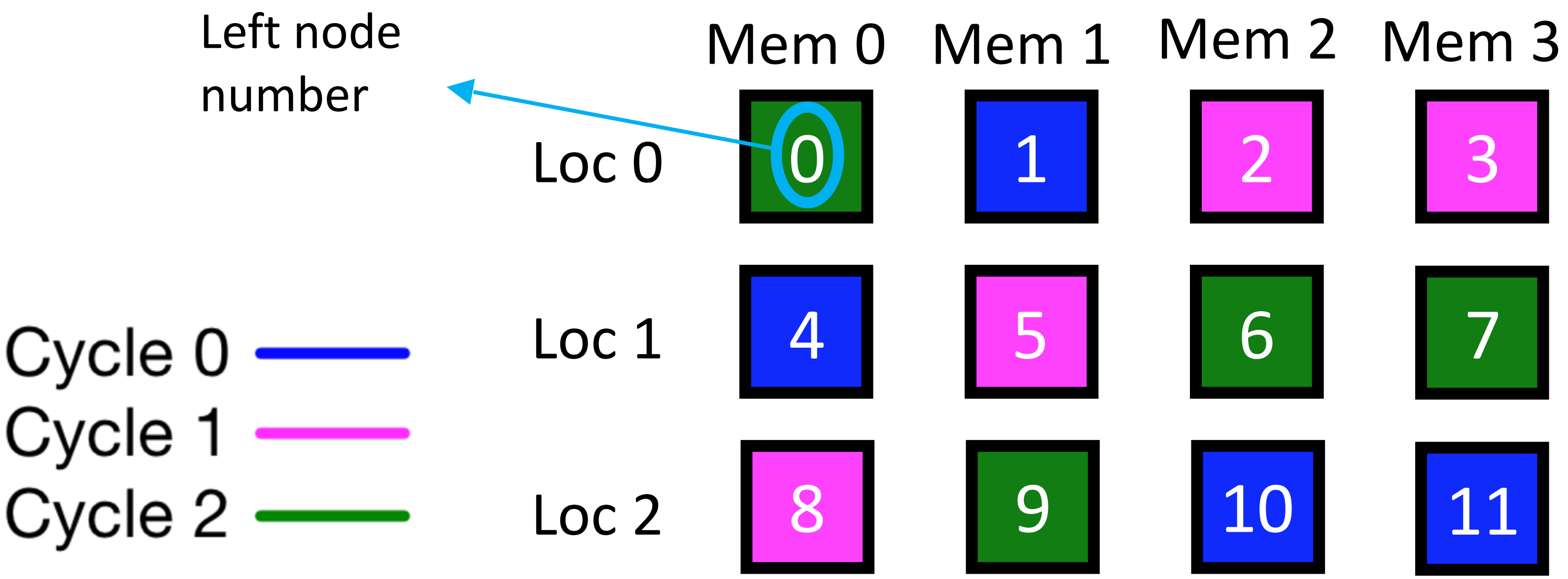


Operational parallelism: FF, BP, UP simultaneously inside a junction

Junction pipelining: Each operates on different inputs across junctions

Faster training @ more hardware and storage cost

Clash-free memory access patterns



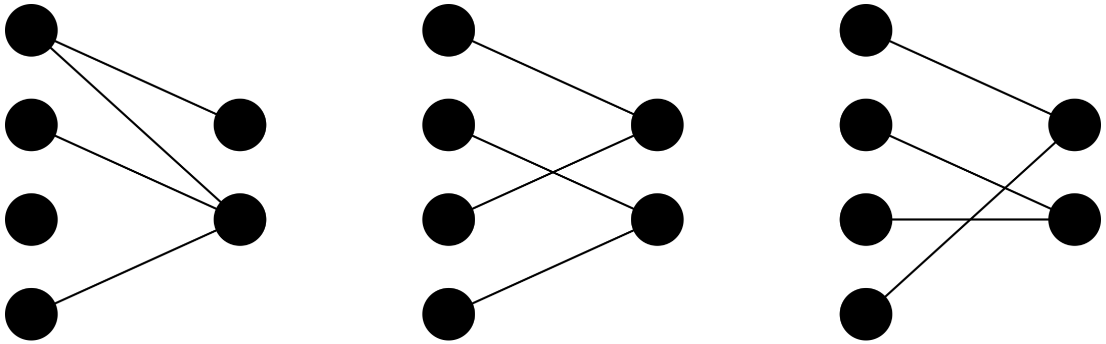
Storage: z_i-length seed vector

Computation: z_i incrementers

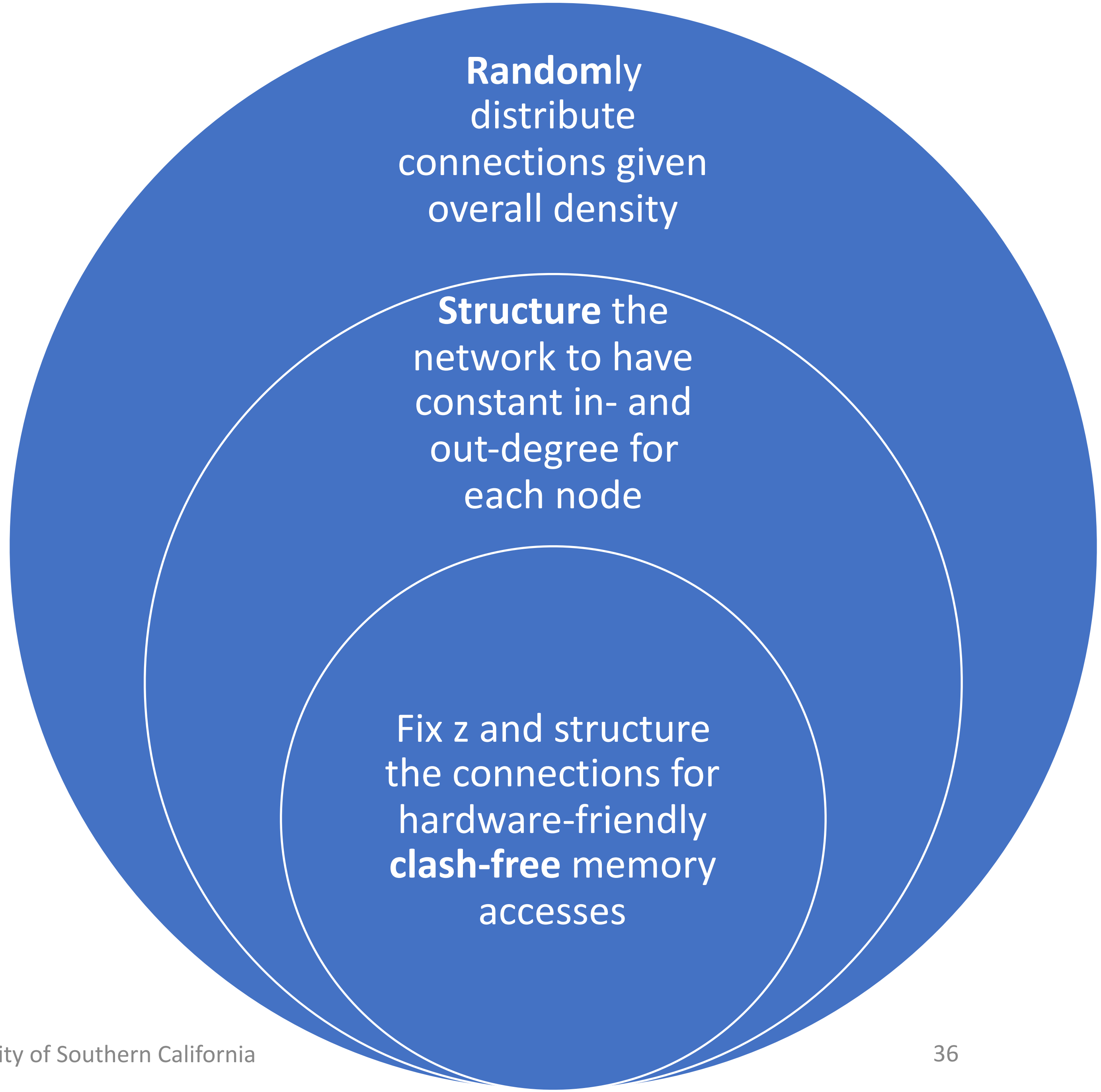
Can have richer classes of memory access patterns @ more hardware cost

N_{i-1} = 12 left side nodes arranged in z_i = 4 memories
Fix a seed vector = (1,0,2,2) for starting cycle 0 locations
For consecutive cycles, add 1 modulo memory depth

Types of pre-defined sparsity



Random → Structured → Clash-free progressively restricts the network



Performance Comparison

Hardware-friendly simple clash-free patterns can improve performance

Random sparsity can perform badly

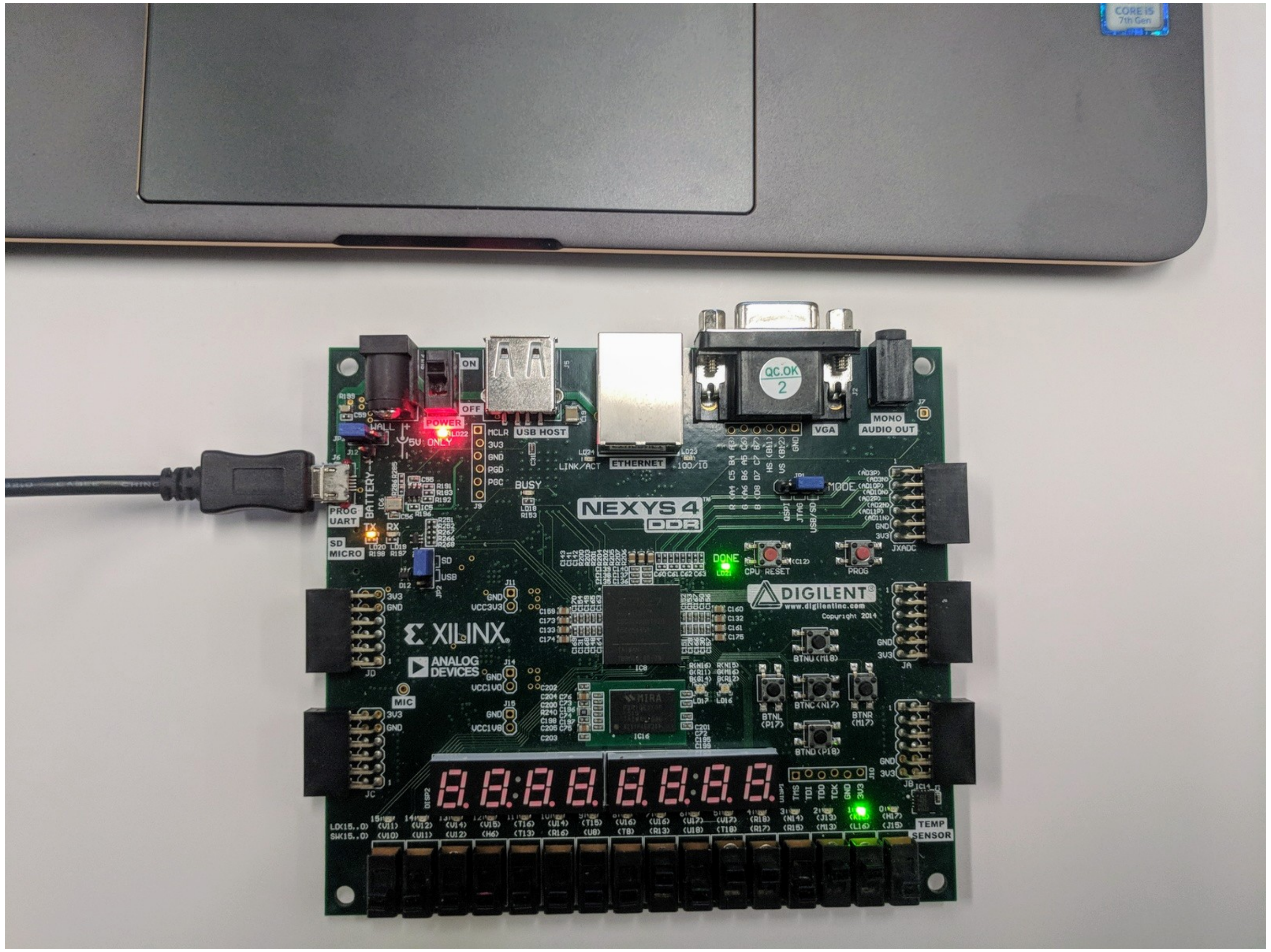
d_{net}^{out}	$\rho_{net}\%$	z_{net}	Test Accuracy Performance		
			Clash-free	Structured	Random
MNIST: $N_{net} = (800, 100, 100, 100, 10)$, FC test accuracy = 98 ± 0.1					
$(80, 80, 80, 10)$	80.2	$(200, 25, 25, 4)$	97.9 ± 0.2	97.9 ± 0.2	97.8 ± 0.2
$(60, 60, 60, 10)$	60.4	$(200, 25, 25, 4)$	97.6 ± 0.1	97.8 ± 0.1	97.6 ± 0.2
$(40, 40, 40, 10)$	40.6	$(200, 25, 25, 5)$	97.5 ± 0.1	97.7	97.6 ± 0.1
$(20, 20, 20, 10)$	20.8	$(200, 25, 25, 10)$	97.2 ± 0.2	97.2 ± 0.1	97.1 ± 0.1
$(10, 10, 10, 10)$	10.9	$(200, 25, 25, 25)$	96.7 ± 0.1	96.8 ± 0.2	96.7 ± 0.2
$(5, 10, 10, 10)$	6.9	$(100, 25, 25, 25)$	96.3 ± 0.1	96.3 ± 0.1	96.2 ± 0.1
$(2, 5, 5, 10)$	3.6	$(80, 25, 25, 50)$	95 ± 0.2	95.1 ± 0.1	95 ± 0.3
$(1, 2, 2, 10)$	2.2	$(80, 20, 20, 100)$	93.3 ± 0.3	93.1 ± 0.5	92 ± 0.3
Reuters: $N_{net} = (2000, 50, 50)$, FC test accuracy = 89.6 ± 0.1					
$(25, 25)$	50	$(1000, 25)$	89.4 ± 0.1	89.3	89.4
$(10, 10)$	20	$(400, 10)$	87 ± 0.1	86.7 ± 0.1	86.5 ± 0.1
$(5, 5)$	10	$(200, 5)$	78.5 ± 0.5	78.2 ± 0.7	77.5 ± 0.6
$(2, 2)$	4	$(80, 2)$	53.3 ± 1.8	51.2 ± 1.7	46.8 ± 2.9
$(1, 1)$	2	$(40, 1)$	28.4 ± 2.4	28.7 ± 2.3	28 ± 1.9
TIMIT: $N_{net} = (39, 390, 39)$, FC test accuracy = 43.2 ± 0.2					
$(270, 27)$	69.2	$(13, 13)$	43 ± 0.1	43	43 ± 0.1
$(180, 18)$	46.2		42.7 ± 0.1	42.8 ± 0.1	42.9 ± 0.1
$(90, 9)$	23.1		42.1 ± 0.1	42.5 ± 0.1	42.4 ± 0.1
$(60, 6)$	15.4		41.5 ± 0.1	41.8 ± 0.2	41.9 ± 0.1
$(30, 3)$	7.7		40.5 ± 0.2	40.1 ± 0.2	39.4 ± 0.8
CIFAR-100 : $N_{net} = (4000, 500, 100)$, FC top-5 test accuracy = 87.1 ± 0.6					
$(100, 100)$	22	$(2000, 250)$	87.5 ± 0.2	87.7 ± 0.2	87.4 ± 0.3
$(29, 29)$	6.4		86.8 ± 0.3	87.2 ± 0.5	87.1 ± 0.2
$(12, 12)$	2.6	$(400, 50)$	86.3 ± 0.2	86.5 ± 0.4	86.6 ± 0.4
$(5, 5)$	1.1		85.3 ± 0.5	85.5 ± 0.5	85.7 ± 0.3
$(2, 2)$	0.4	$(80, 10)$	84.1 ± 0.5	84.3 ± 0.3	83.8 ± 0.3
$(1, 1)$	0.2		83 ± 0.5	83.3 ± 0.4	81.7 ± 0.7

FPGA Implementation

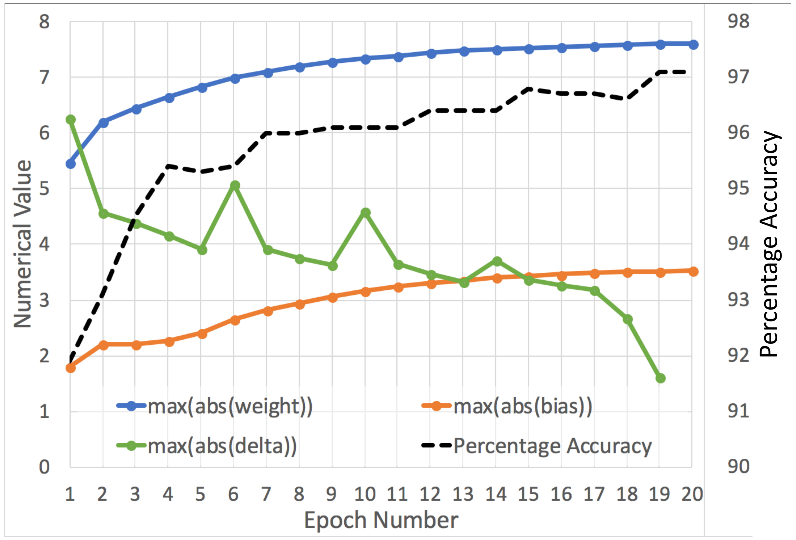
Initial hardware prototype of pre-defined sparse 2-junction network training on MNIST

- ➢ Nodes = 1120
- ➢ Weights = 5120
- ➢ Overall density = 7.5%
- ➢ Total parallelism = 160

Xilinx Artix-7 FPGA on Digilent Nexys4 board

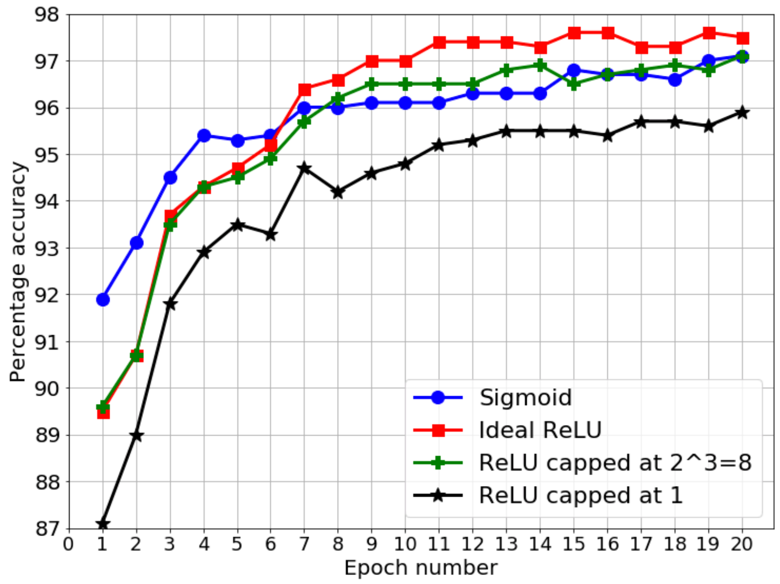


Some Findings and Considerations

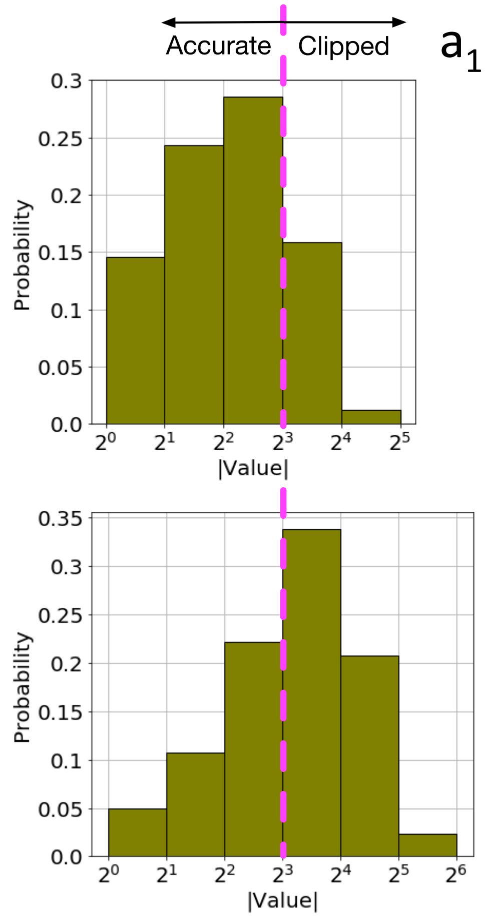


12-bit fixed point config:
(sign, integer, fractional) = (1,3,8)

Sigmoid activation works better in hardware than ReLU



a_1 histograms



(a) Sparse
Mean = 4.6
Stddev = 3.8
% Clipped = 17

(b) FC
Mean = 11
Stddev = 8.2
% Clipped = 57

Dynamic range is reduced due to pre-defined sparsity

Ongoing / Future Work in H/W Implementation

This dissertation:

➢ More pipelining to improve speed (current clock frequency = 15 MHz)

Other members of our team:

➢ Better memory interfacing and management protocols

➢ Leveraging cloud FPGA resources to support bigger networks

Outline

- Introduction and Background
- Pre-Defined Sparsity
- Hardware Architecture
- **Connection Patterns**
- Dataset Engineering
- Model Search

Achieved Research Contributions

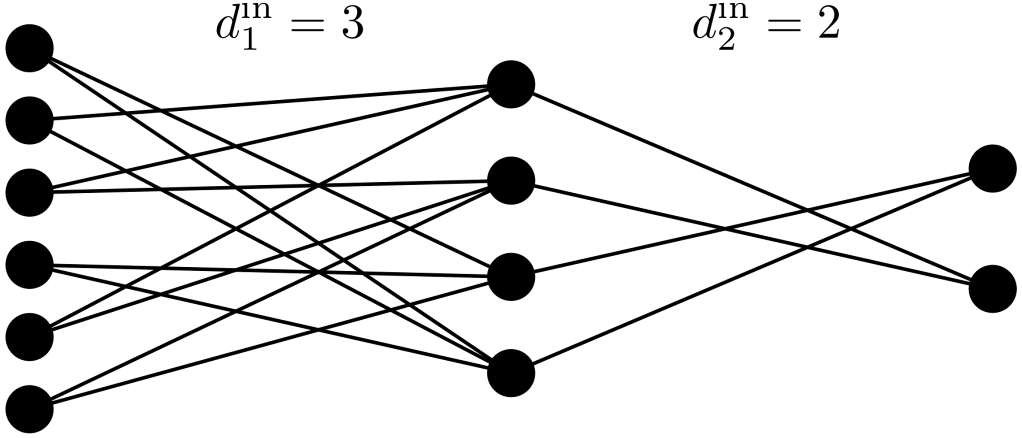
Biadjacency Matrices

Junction 1

$$d_1^{\text{out}} = 2$$
$$d_1^{\text{in}} = 3$$

Junction 2

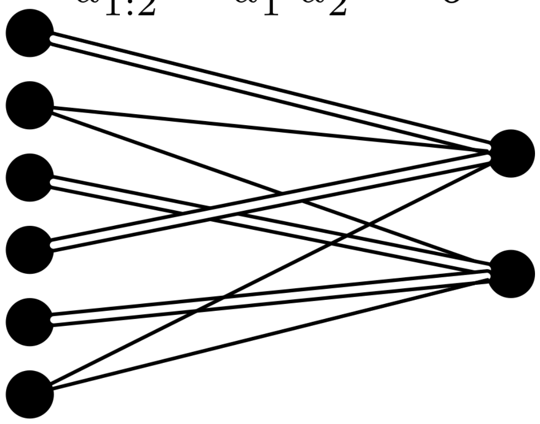
$$d_2^{\text{out}} = 1$$
$$d_2^{\text{in}} = 2$$



<=>

Equivalent Junction 1:2

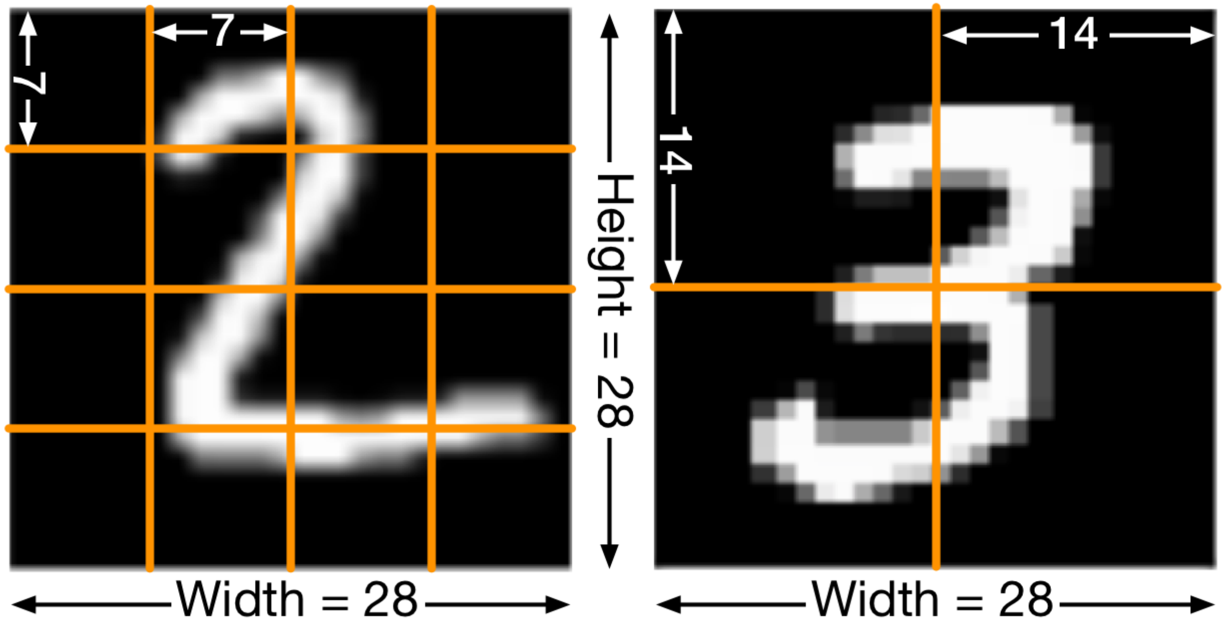
$$d_{1:2}^{\text{out}} = d_1^{\text{out}} d_2^{\text{out}} = 2$$
$$d_{1:2}^{\text{in}} = d_1^{\text{in}} d_2^{\text{in}} = 6$$



$$\boldsymbol{\mathcal{B}}_1 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{\mathcal{B}}_2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

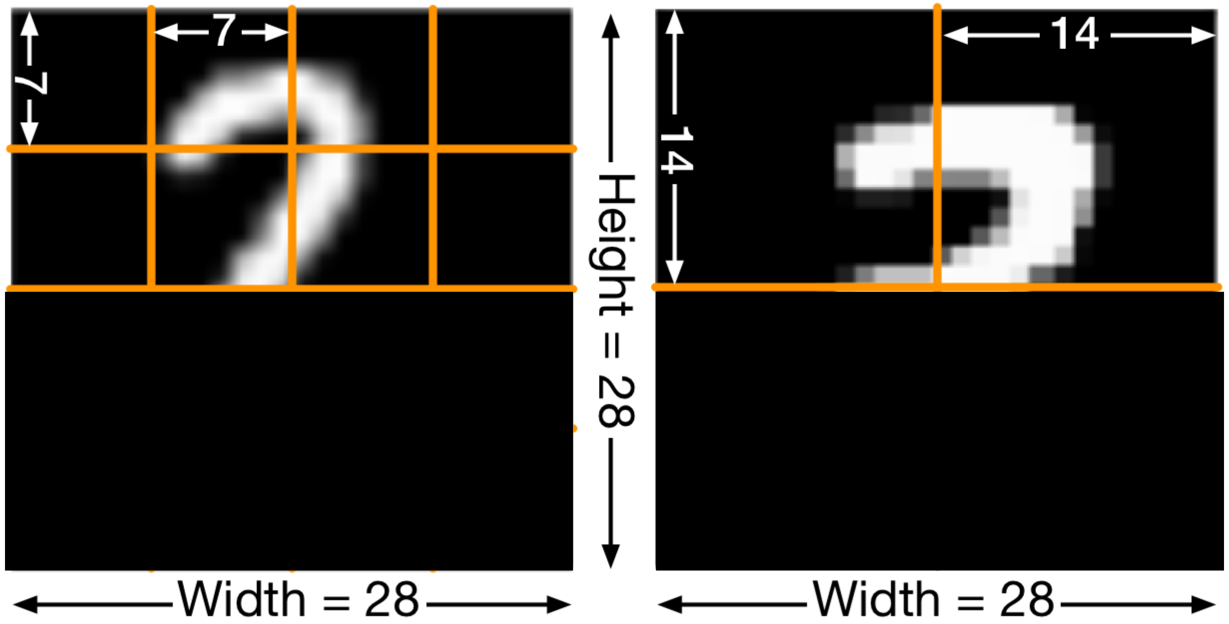
$$\boldsymbol{\mathcal{B}}_{1:2} = \boldsymbol{\mathcal{B}}_2 \boldsymbol{\mathcal{B}}_1 = \begin{bmatrix} 2 & 1 & 0 & 2 & 0 & 1 \\ 0 & 1 & 2 & 0 & 2 & 1 \end{bmatrix}$$

Windowed connection patterns



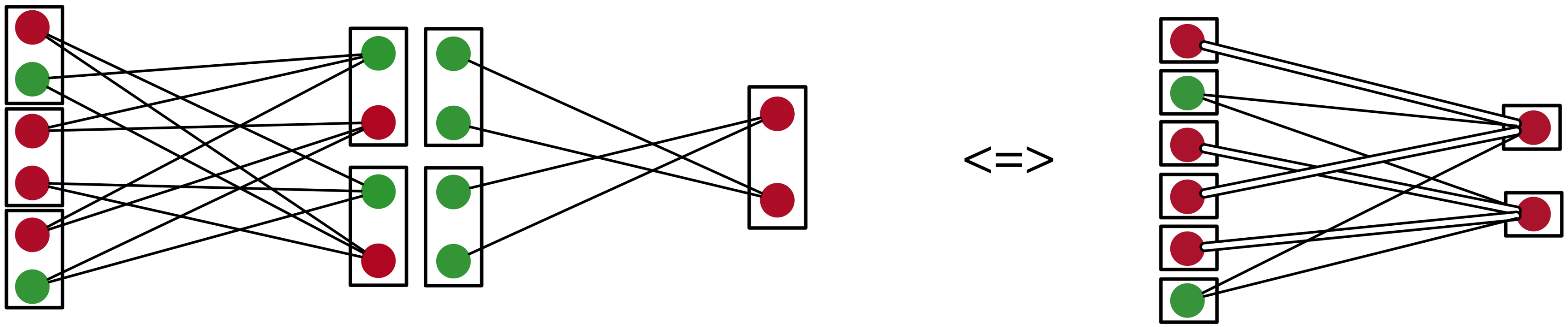
For best results, nodes should get information from all portions of adjacent layers
=> Define windows

Windowed connection patterns



For best results, nodes should get information from all portions of adjacent layers
=> Define windows

Windowed Biadjacency Matrices and Scatter



$$\mathcal{B}_1^{\mathrm{f}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{bmatrix} \qquad \mathcal{B}_2^{\mathrm{f}} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad \mathcal{B}_{1:2}^{\mathrm{f}} = \begin{bmatrix} 2 & 1 & 0 & 2 & 0 & 1 \\ 0 & 1 & 2 & 0 & 2 & 1 \end{bmatrix}$$

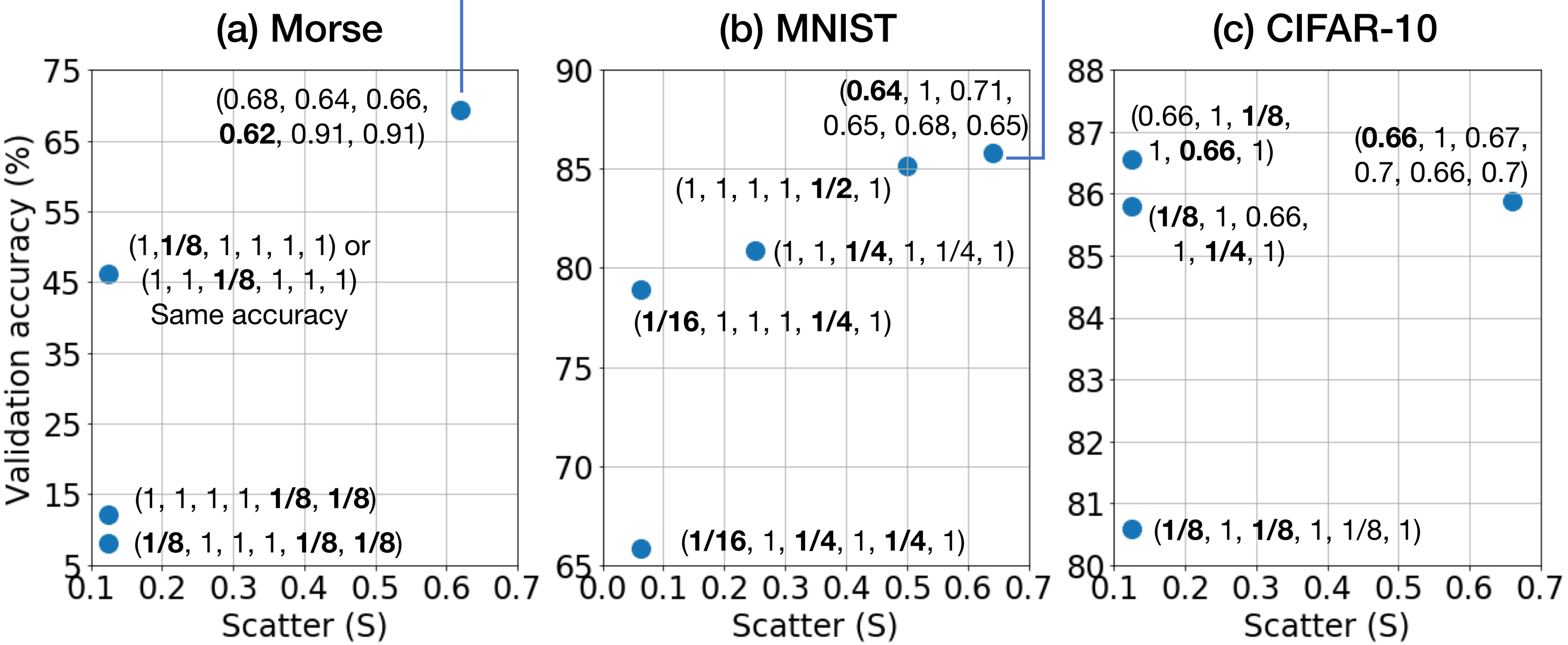
$$\mathcal{B}_1^{\mathrm{b}} = \begin{bmatrix} 0 & 1 & 2 & 0 & 2 & 1 \\ 2 & 1 & 0 & 2 & 0 & 1 \end{bmatrix} \qquad \mathcal{B}_2^{\mathrm{b}} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \qquad \mathcal{B}_{1:2}^{\mathrm{b}} = \begin{bmatrix} 2 & 1 & 0 & 2 & 0 & 1 \\ 0 & 1 & 2 & 0 & 2 & 1 \end{bmatrix}$$

$$\boldsymbol{S}^{\mathrm{net}} = \left(S_1^{\mathrm{f}} = 0.83, S_1^{\mathrm{b}} = 0.67, \quad S_2^{\mathrm{f}} = \mathbf{0.5}, S_2^{\mathrm{b}} = 1, \quad S_{1:2}^{\mathrm{f}} = 0.67, S_{1:2}^{\mathrm{b}} = 0.67 \right)$$

Scatter – Performance prediction before training

Not explicitly planning connections performs the best

… ties in with proposed research on model search



Scatter can help in filtering out bad networks before training … (work in progress)

Outline

- Introduction and Background
- Pre-Defined Sparsity
- Hardware Architecture
- Connection Patterns
- **Dataset Engineering**
- Model Search

Achieved Research Contributions

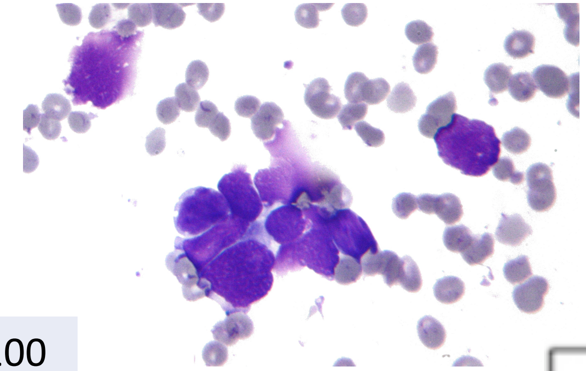
Data, data, everywhere,
Not quality enough to use

Real world data has challenges:

➢ Too few samples

➢ Incorrect labeling

➢ Missing entries



13.2	0.05		1200
10.9		A	
	0.78	B+	1400
11.4			1100

Synthetic data is generated using computer algorithms

➢ Very large quantities can be generated

➢ Mimic real-world data as desired

➢ Classification difficulty tweaking

Morse Code Datasets

Morse Code is a system of communication where letters, numbers and symbols are encoded using dots and dashes

Example:

\+ · – · – ·

Step 1:

Frame length: 64

Dot: 1-3
Dash: 4-9
Intermediate space: 1-3
Leading spaces: None
Trailing spaces: Remaining at end

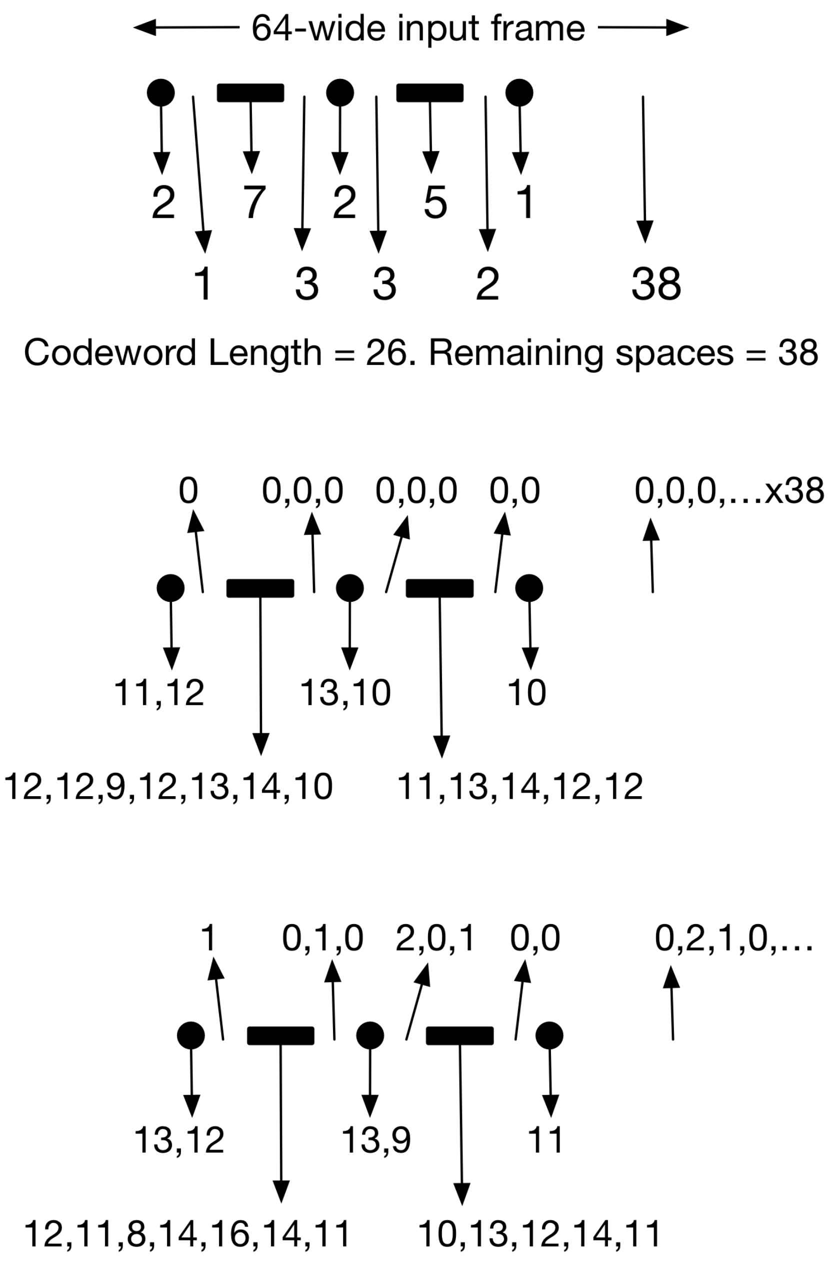
Step 2:

Expected value range = [0,16]

Dot, dash = *Normal*(12,4/3)
Space = 0

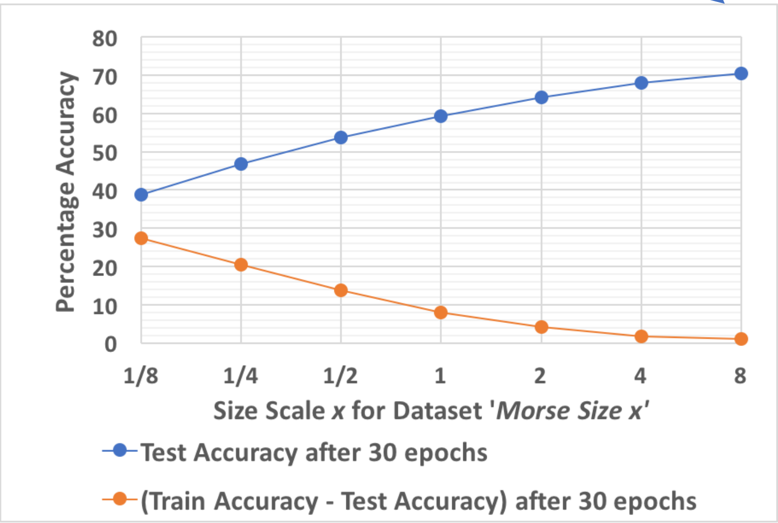
Step 3:

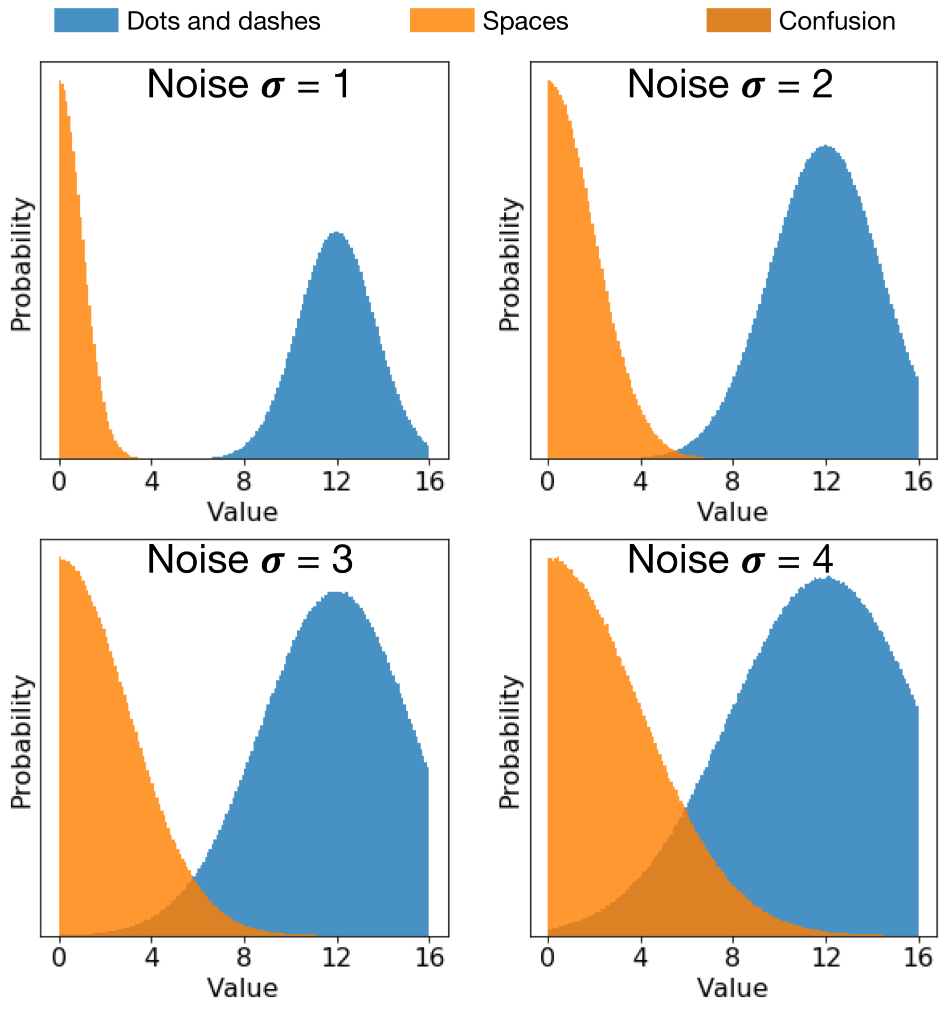
Additive Noise = *Normal*(0,σ)
(For this case, σ=1)



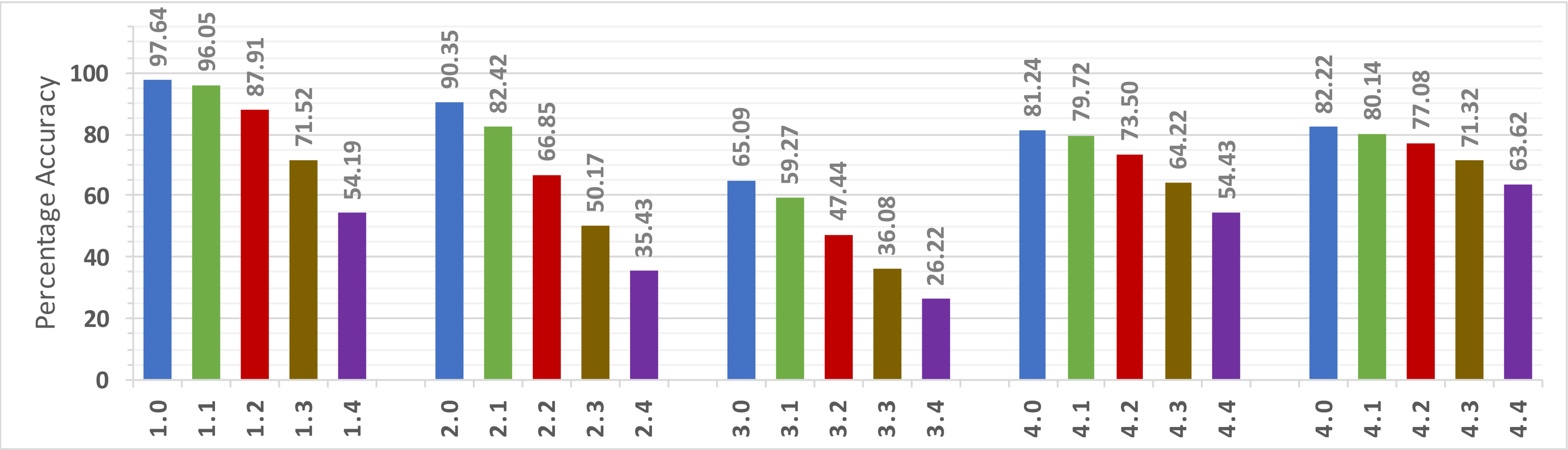
Variations and Difficulty Scaling

- ➢ More noise
- ➢ Leading and trailing spaces
- ➢ Confusing dashes with dots and spaces
- ➢ Dilating frame to size 256
- ➢ Increasing #samples in dataset





Neural network performance



Baseline + Noise

Noise +
Leading Spaces

Noise +
Leading Spaces +
Dash Length=3-9

Dilated by 4 : Noise +
Leading Spaces +
Dash Length=3-9

Using L2 reg = 10^-5
Dilated by 4 : Noise +
Leading Spaces +
Dash Length=3-9

Tunable dataset difficulty leads to a variety of benchmarks

Metrics to characterize dataset difficulty

$$V_{\text{lower}} = \sum_{m=1}^{N_L} P(m) Q \left(\sqrt{\frac{d_{\min}(m)^2}{4{\sigma_m}^2}} \right)$$

$$V_{\text{upper}} = \sum_{m=1}^{N_L} P(m) \sum_{\substack{j=1 \\ j \neq m}}^{N_L} Q \left(\sqrt{\frac{d(m,j)^2}{4{\sigma_m}^2}} \right)$$

$$V_{\text{dist}} = \frac{\sum_{m=1}^{N_L} \frac{\sigma_m}{d_{\min}(m)}}{N_L}$$

$$\boxed{V_{\text{thresh}}} = \sum_{m=1}^{N_L} \sum_{\substack{j=1 \\ j \neq m}}^{N_L} \mathbb{I} \left(\frac{\|c_m - c_j\|_1}{N_0} < 0.05 \right)$$

Metrics to characterize dataset difficulty

Probability of the mth class occurring

Gaussian Q-function

$$V_{\text{lower}} = \sum_{m=1}^{N_L} P(m) Q\left(\sqrt{\frac{d_{\min}(m)^2}{4{\sigma_m}^2}}\right)$$

Minimum distance between centroids of *m*th class and any other class

Average variance across all features in *m*th class

$$V_{\text{upper}} = \sum_{m=1}^{N_L} P(m) \sum_{\substack{j=1 \\ j\neq m}}^{N_L} Q\left(\sqrt{\frac{d(m,j)^2}{4{\sigma_m}^2}}\right)$$

Distance between centroids of *m*th and *j*th classes

$$V_{\text{dist}} = \frac{\sum_{m=1}^{N_L} \frac{\sigma_m}{d_{\min}(m)}}{N_L}$$

#classes

$$\boxed{V_{\text{thresh}}} = \sum_{m=1}^{N_L} \sum_{\substack{j=1 \\ j\neq m}}^{N_L} \mathbb{I}\left(\frac{\|c_m - c_j\|_1}{N_0} < 0.05\right)$$

#features

Goodness of the Metrics

Metric	r
V_{lower}	-0.59
V_{upper}	-0.64
V_{dist}	-0.63
V_{thresh}	-0.64

Pearson's correlation coefficient between metric and test set classification accuracy of Morse code datasets of varying difficulty (negative because metrics indicate difficulty)

Metrics can be used to understand the inherent difficulty of the classification problem on a dataset before applying any learning algorithm

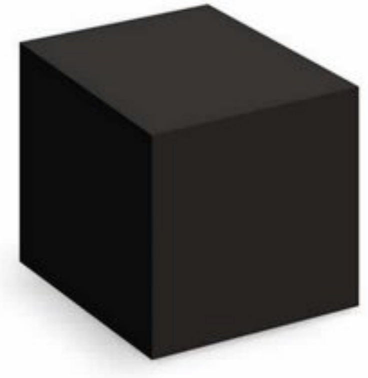
Outline

- Introduction and Background
- Pre-Defined Sparsity
- Hardware Architecture
- Connection Patterns
- Dataset Engineering
- **Model Search** — Proposed Research

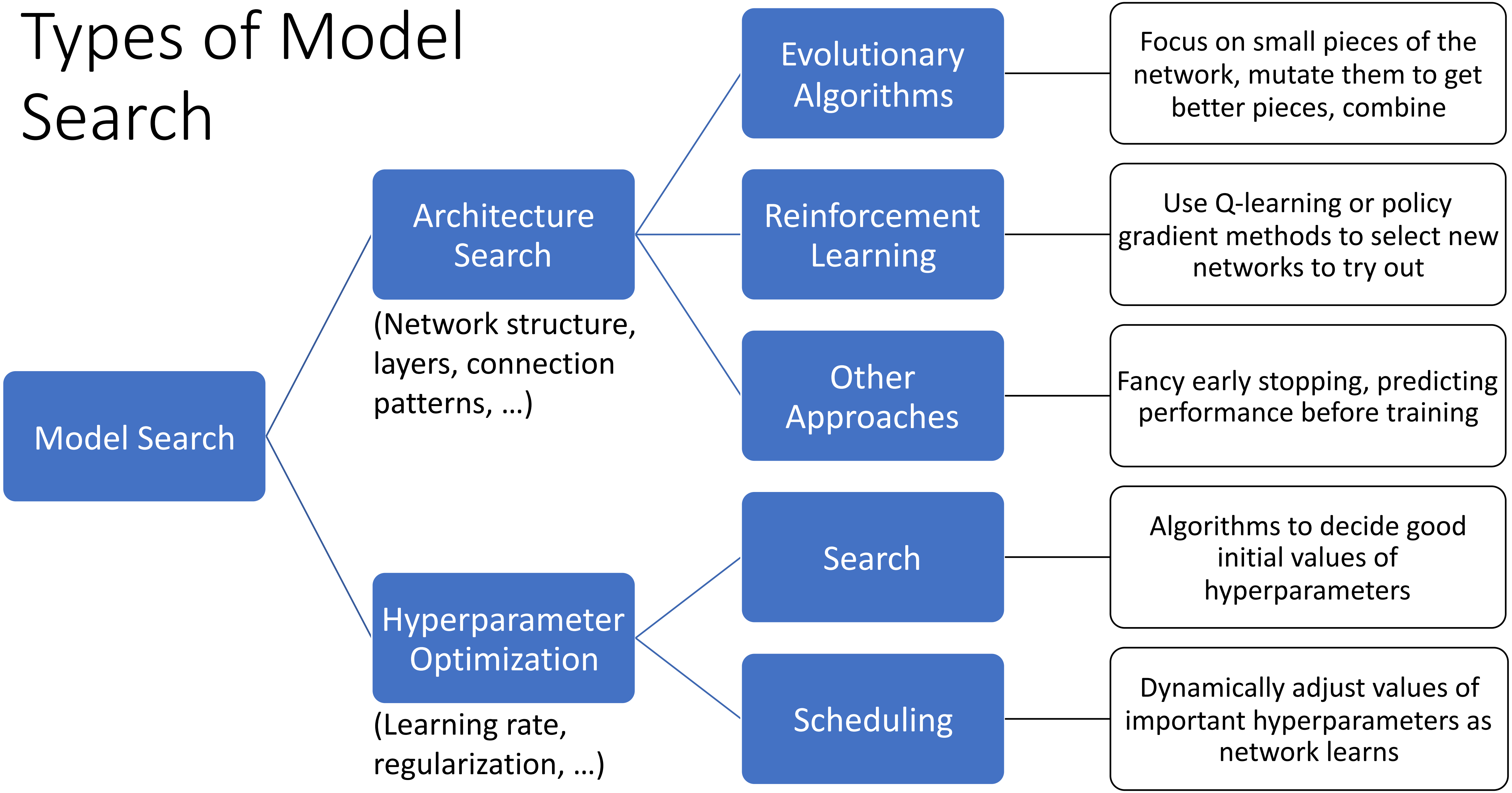
Introduction to Model Search?

Neural networks are largely black boxes

- How do they work?
- Are so many layers and neurons really needed?
- Which parts of a network are the most important?
- How should different layers be connected?
- What are good hyperparameter values?



Types of Model Search



Related Work

- Model Search
  - Architecture Search
    - Evolutionary Algorithms
      - Miikkulainen 2017 – Evolving DNNs
      - Liu 2018 – Hierarchical Representations
      - Real 2018 – Regularized Evolution
    - Reinforcement Learning
      - Zoph 2016 – NAS
      - Baker 2017 – Designing NN Architectures using RL
      - Pham 2018 – ENAS
    - Other Approaches
      - Liu 2017 – Progressive NAS
      - Baker 2018 – Accelerating NAS using Performance Prediction
      - Liu 2018 – DARTS
  - Hyperparameter Optimization
    - Search
      - Bergstra 2012 – Random search
      - Bergstra 2013 – Making a Science of Model Search
      - Snoek 2012 – Bayesian optimization
    - Scheduling
      - Smith 2017 – Cyclical learning rates
      - Loschilov 2017 – SGDR
      - Goyal 2017 – Accurate, large minibatch SGD

*Total '**GPU days**' for finding a good architecture can be >100*

Our Proposed Research

GOAL: Automate the process of designing well-performing, low complexity sparse neural networks for various applications

- Architecture search with focus on low complexity networks
  - Extend complexity reduction methods like pre-defined sparsity to other networks beyond MLP
  - Lower complexity networks can train faster (sparse libraries)
  - Democratize architecture search to entities without enormous finances
- Deeper understanding of neural networks
  - Build on trends and guidelines for sparsity
  - Which parts of a network are important – leverage evolutionary algorithms
  - Build on scatter-like methods to predict performance prior to training
  - More informed early stopping – software and hardware monitors

Summary of Contributions

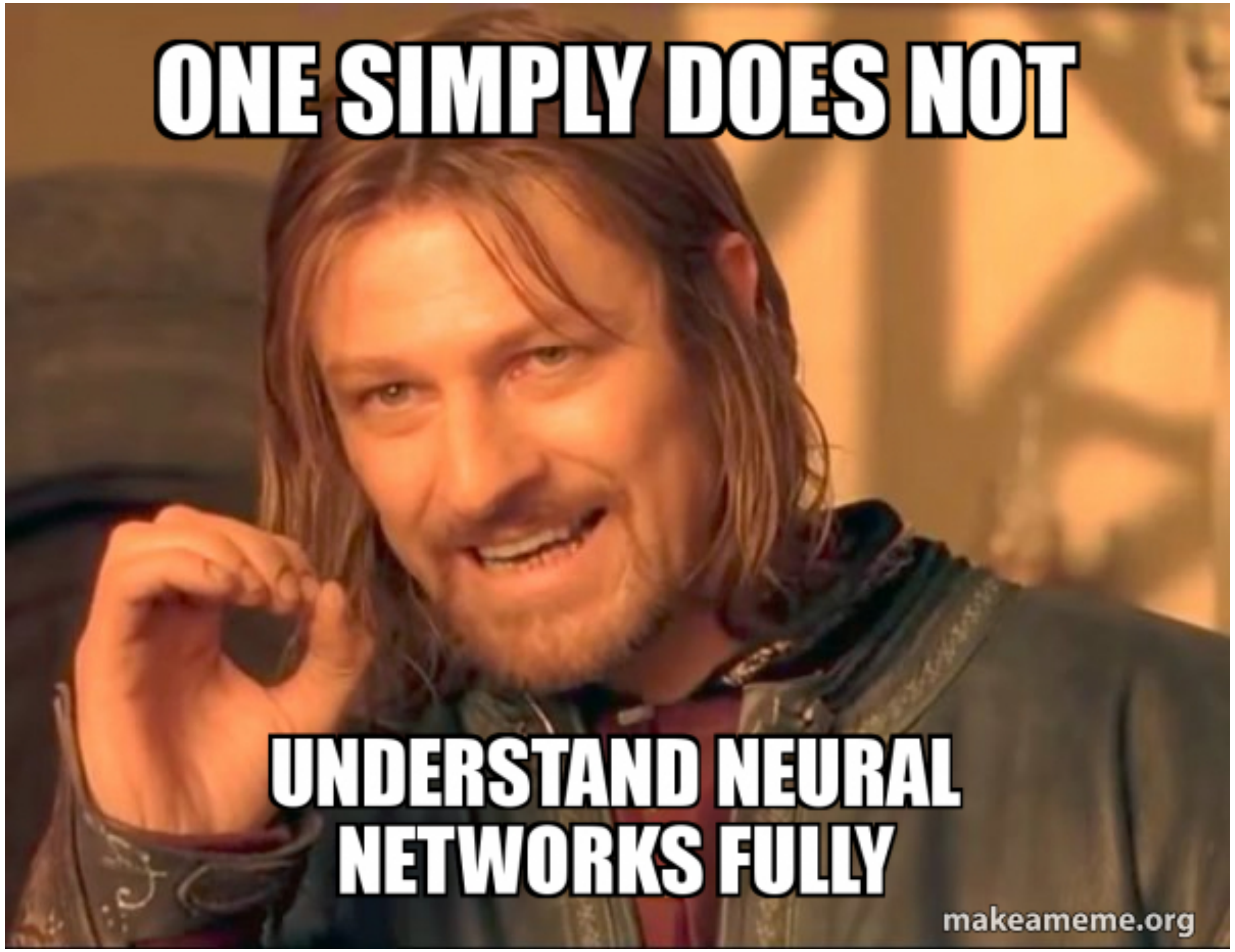
Achieved:

- Proposing and analyzing **pre-defined sparsity** to reduce NN complexity
- **Hardware architecture** to leverage pre-defined sparsity
- Analyzing **connection patterns** and performance predicting measures
- Family of **synthetic datasets** on Morse code with tunable difficulty

Proposed:

- Better **pipelining** to improve hardware architecture
- **Architecture search** and understanding of low complexity neural networks
- [*Time and resources permitting*] Hyperparameter search tuned to low complexity neural networks

Thank you!


ONE SIMPLY DOES NOT
UNDERSTAND NEURAL NETWORKS FULLY
makeameme.org