Interleaver Design for Deep Neural Networks

Sourya Dey, Peter Beerel, Keith Chugg

Asilomar Conference on Signals, Systems, and Computers

Oct-Nov 2017

USC University of Southern California

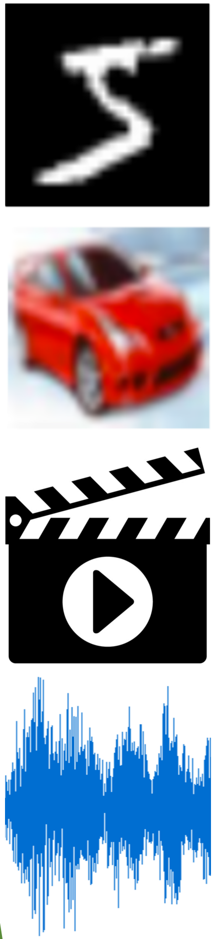
Overview of Current DNNs

- Key machine learning technologies

- Lot of parameters - **Memory intensive**
- Slow to train - **Computationally intensive**

- Training done **offline** in CPU/GPU
- Custom hardware used for **inference only**

Typical Supervised Network



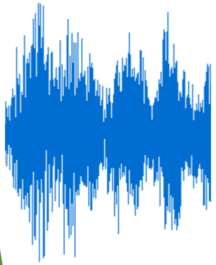
Convolutional Layers → Pooling and Normalization → Fully Connected Classification Layers (CLs) → **Predictions**

5% of total network connections

95% of total network connections

Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS-2012, pp. 1097–1105 (2012)
Zhang, C., Wu, D., Sun, J., Sun, G., Luo, G., Cong, J.: Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In: ISLPED-2016. pp. 326– 331. ACM, New York (2016)

Focus of our Approach



Convolutional Layers → Pooling and Normalization → Fully Connected CLs → Predictions

95% of total network connections

Overview of our Research

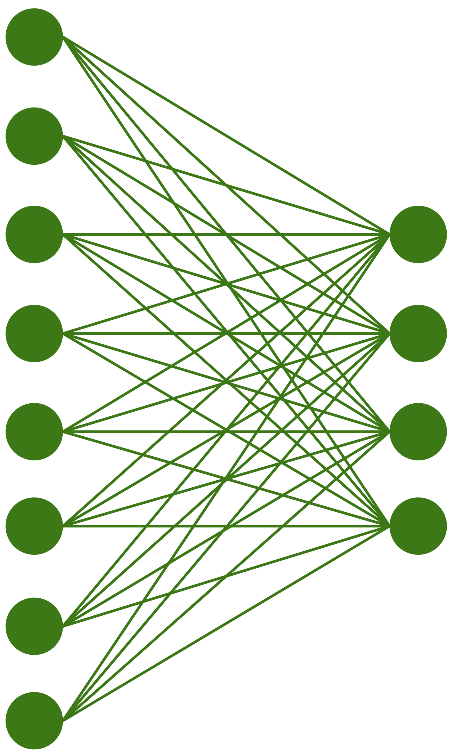
- Predefined sparsity - **Memory friendly**
  - *2-3x savings on CL only network parameters*
  - *2 orders of magnitude savings on CL parameters of CNNs **

- Edge-based processing - **Computationally flexible**
- Hardware optimizations - **Fast** training

- FPGA based architecture - **Online training** and inference

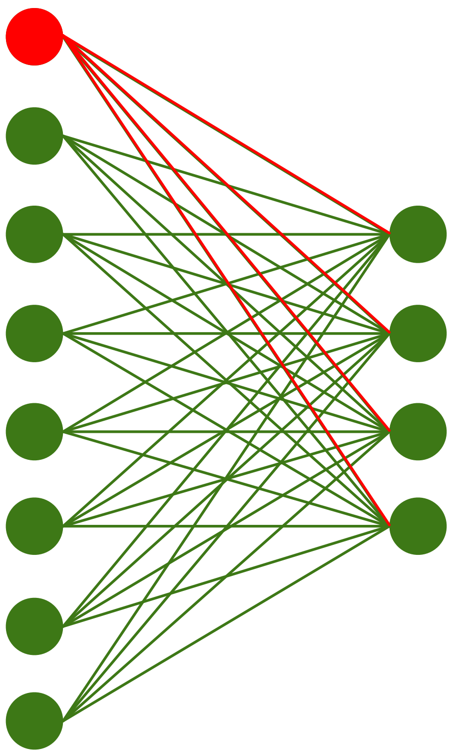
Dey, S., Shao, Y., Chugg, K.M., Beerel, P.A.: Accelerating Training of Deep Neural Networks via Sparse Edge Processing. In: Proc. ICANN-2017, pp. 273-280. LNCS (2017)
* Dey, S., Huang, K.W., Beerel, P.A., Chugg, K.M.: Characterizing Sparse Connectivity Patterns in Neural Networks. In: ICLR-2018 (submitted for publication)

Sparsity



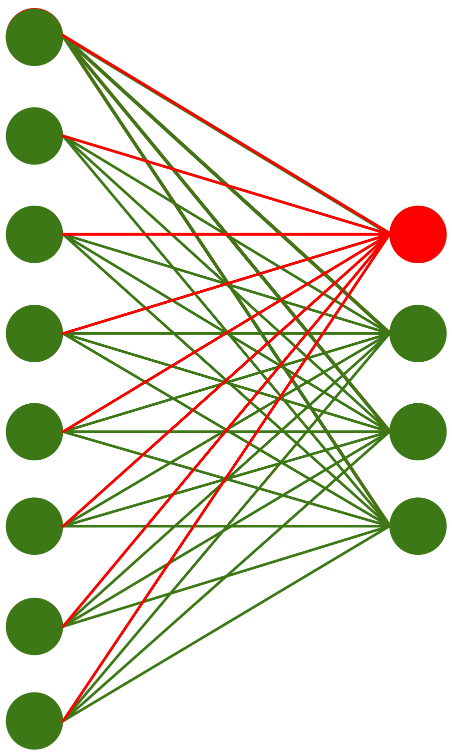
Fully connected (FC) network

Sparsity



Fully connected (FC) network
Fanout (fo) = 4

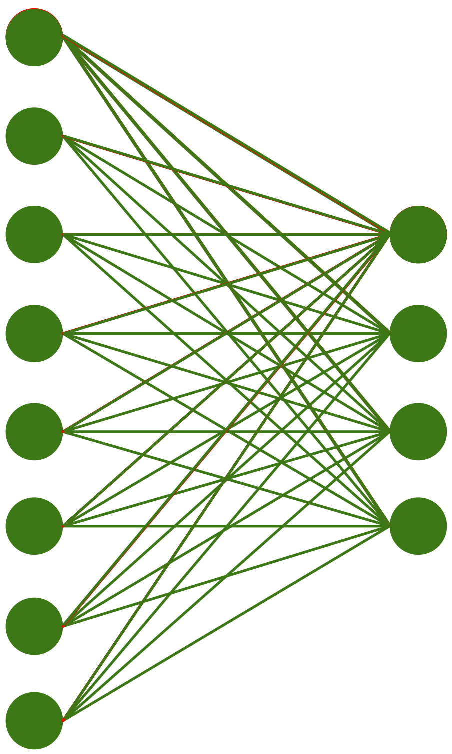
Sparsity



Fully connected (FC) network

Fanout (*fo*) = 4 Fanin (*fi*) = 8

Sparsity

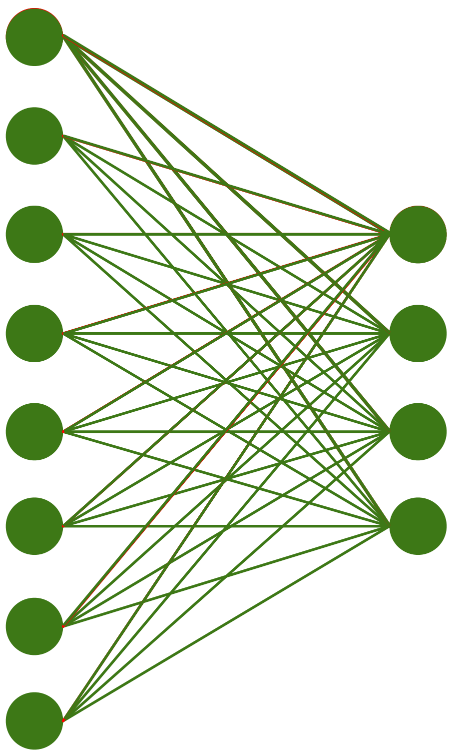


Fully connected (FC) network

Fanout (fo) = 4 Fanin (fi) = 8

Connectivity = 100%

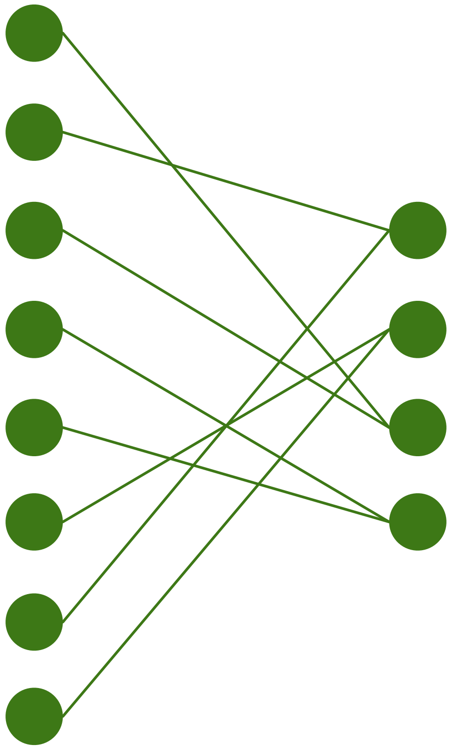
Sparsity



Fully connected (FC) network

Fanout (*fo*) = 4 Fanin (*fi*) = 8

Connectivity = 100%

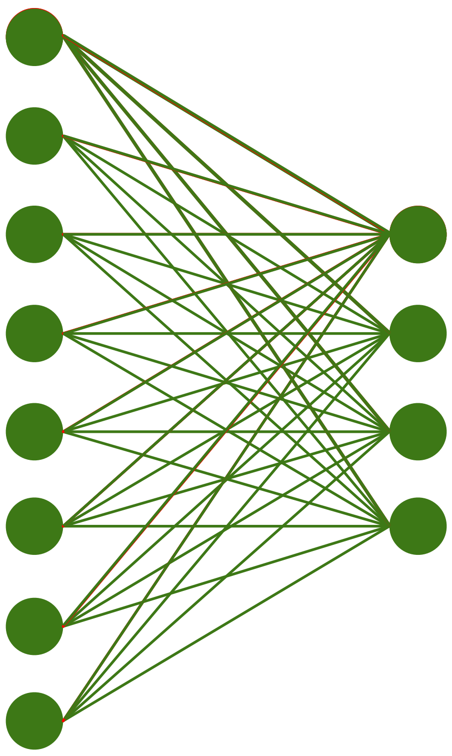


Sparse network

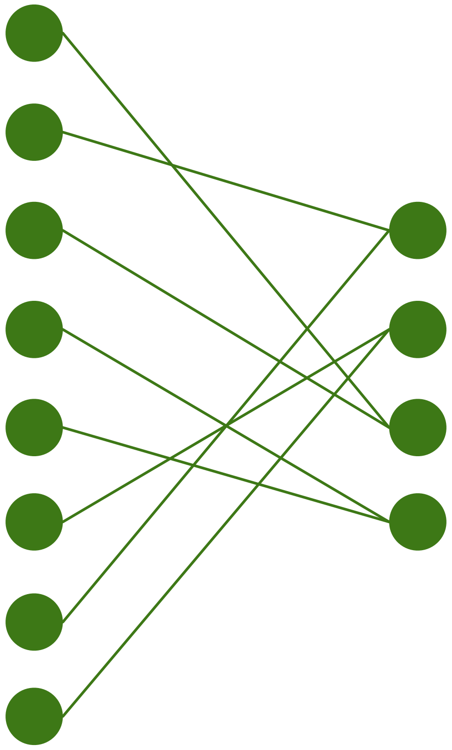
fo = 1, *fi* = 2

Connectivity = 25%

Sparsity – Predefined

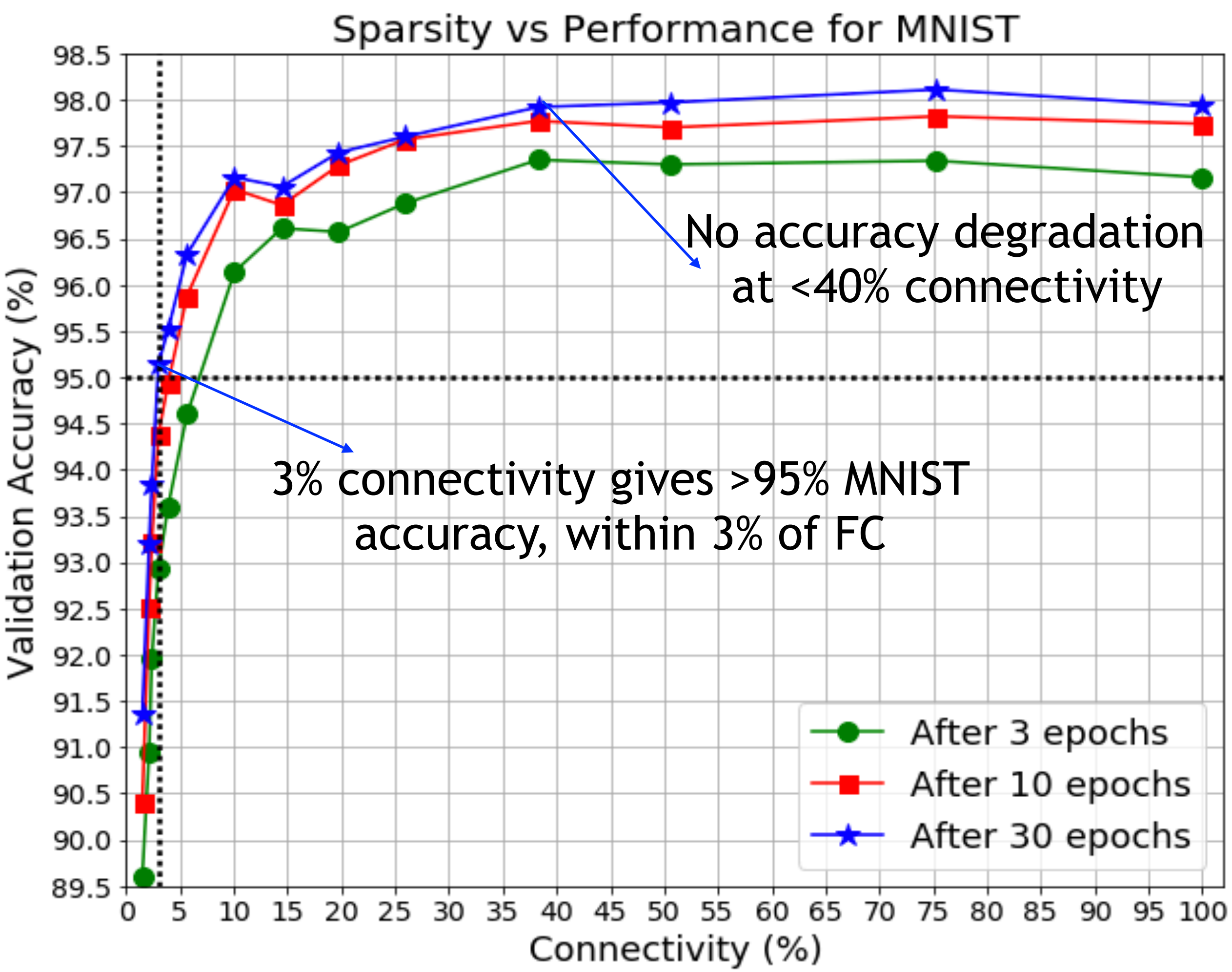


Fully connected (FC) network
Fanout (*fo*) = 4 Fanin (*fi*) = 8
Connectivity = 100%

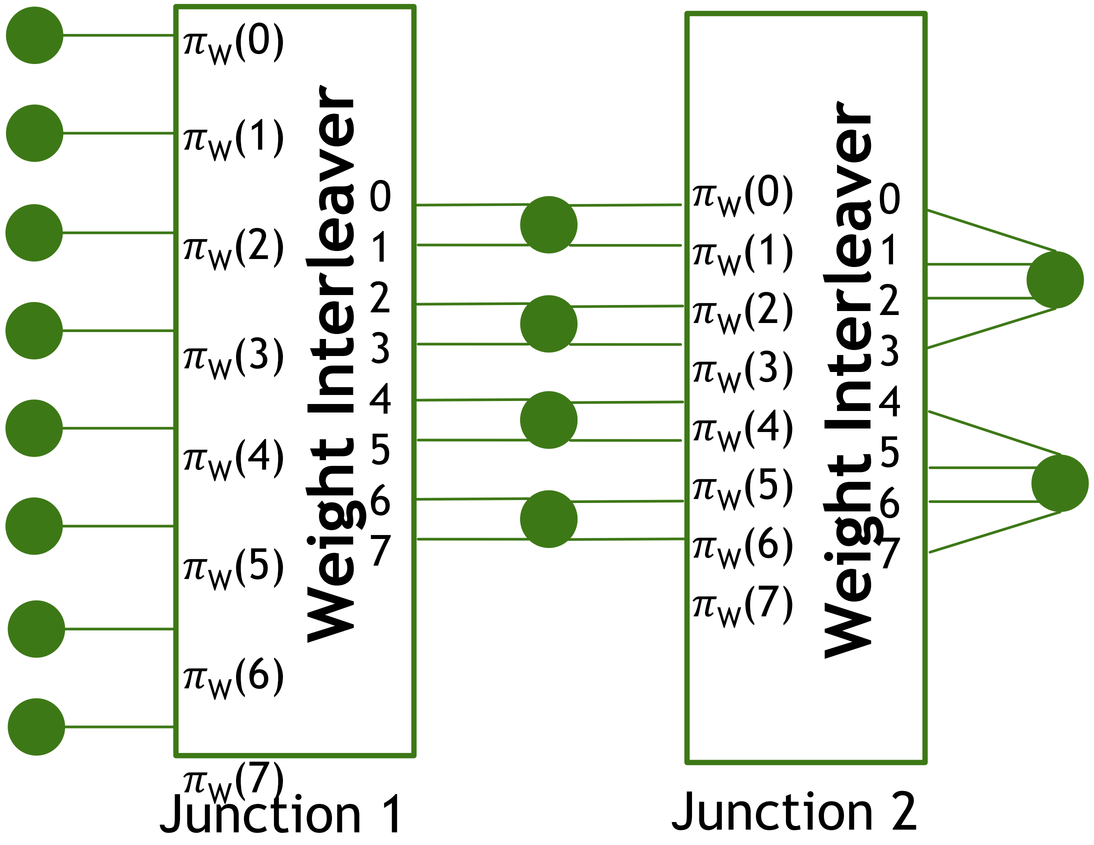


Sparse network
fo = 1, *fi* = 2
Connectivity = 25%

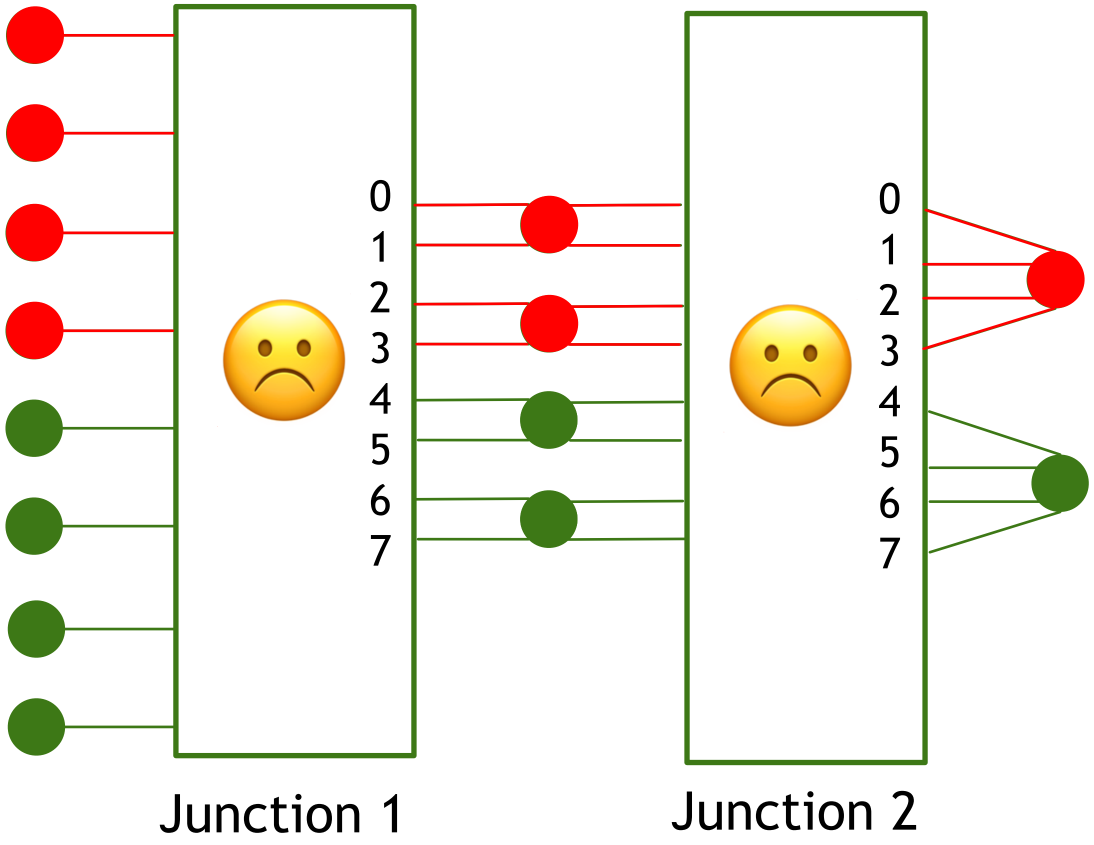
Example of Parameter Savings



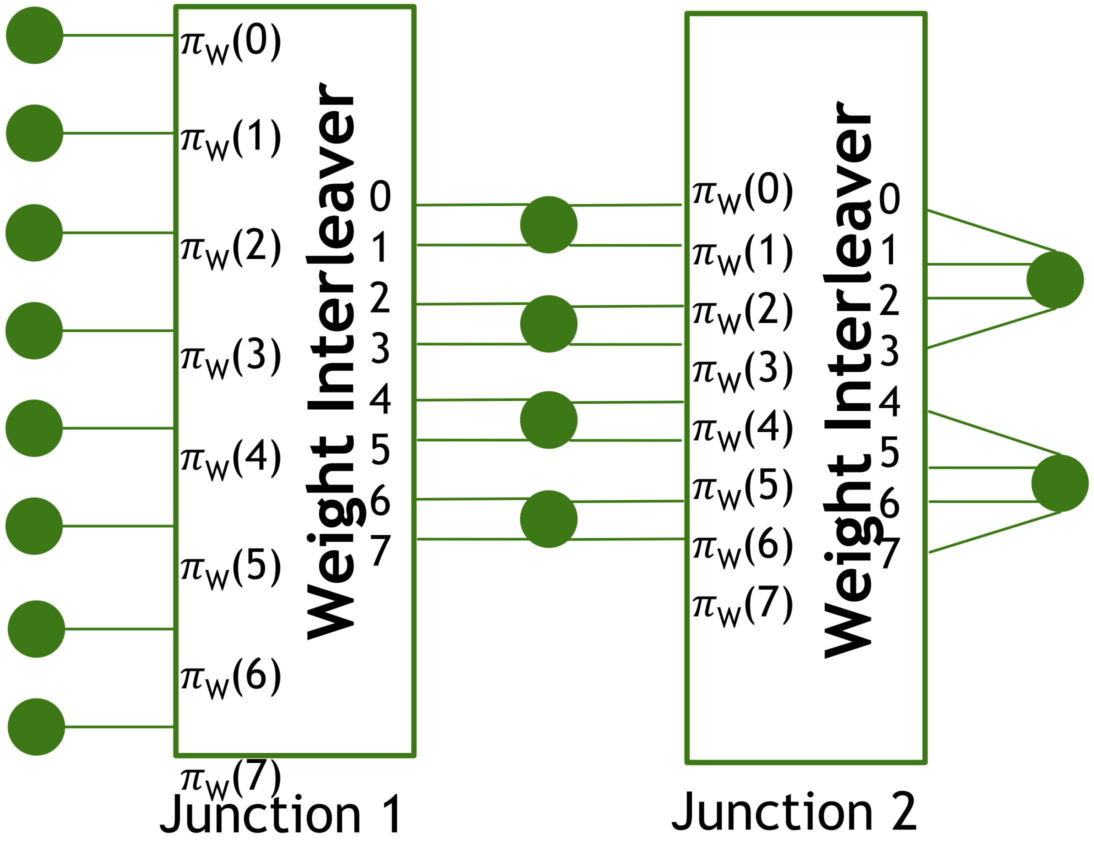
Present Work - Interleavers for Sparse Patterns



Present Work - Interleavers for Sparse Patterns



Present Work - Interleavers for Sparse Patterns



Interleaver algorithm ensures:

- Each output connected to a *good spatial chunk* of different inputs
- No neuron unconnected

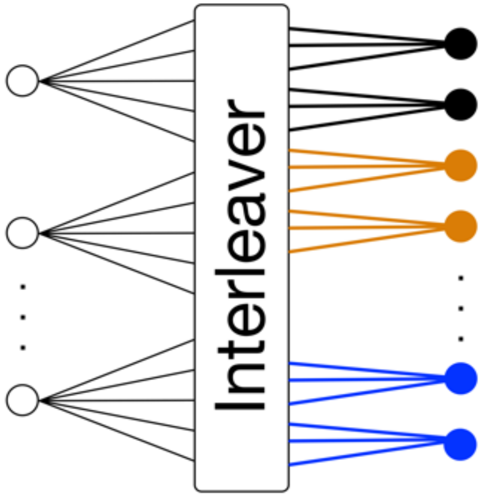
Interleaver Requirements

- Optimized for computational efficiency in hardware
- Optimized for on-chip storage
- High values for metrics which are performance indicators

Degree of Parallelism = z

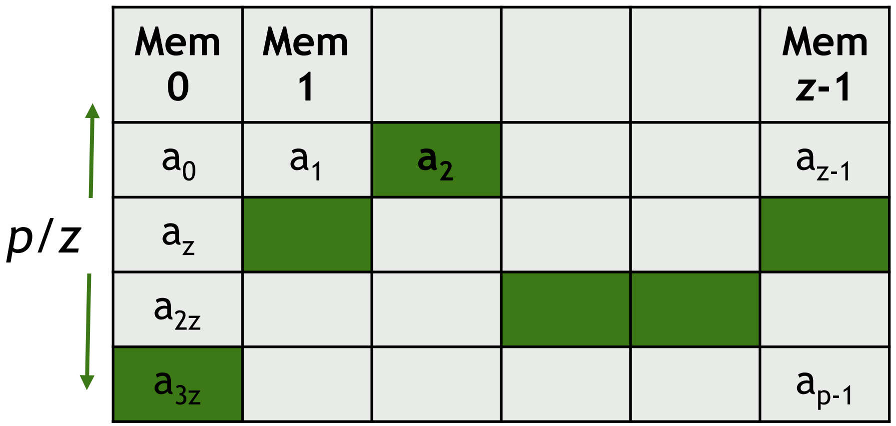
- z memories for all parameters of same type
- Process z parameters in 1 **cycle** => 1 from each mem
- **Process all parameters in a sweep**

Mem 0	**Mem 1**				**Mem z-1**
w_0	w_1	w_2			w_{z-1}
w_z					
w_{2z}					
w_{3z}					w_{W-1}

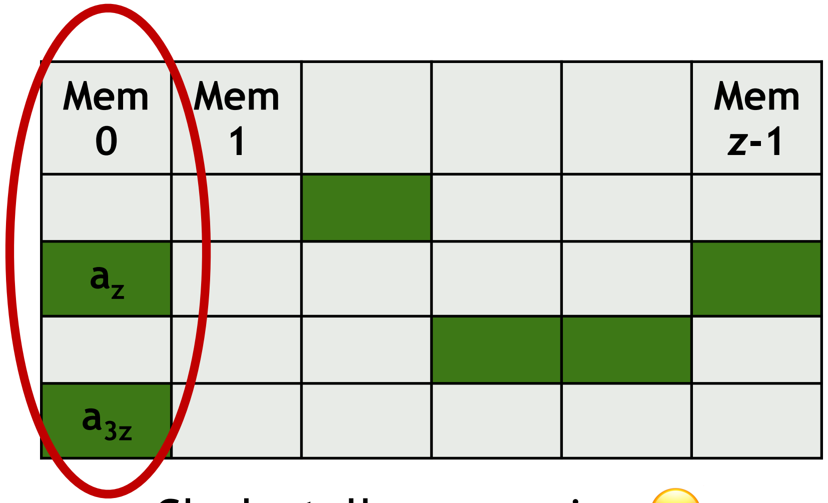


Clash Freedom

- E.g. *p* activations, so depth of each memory = *p*/*z*
- Accessed in interleaved (permuted order)



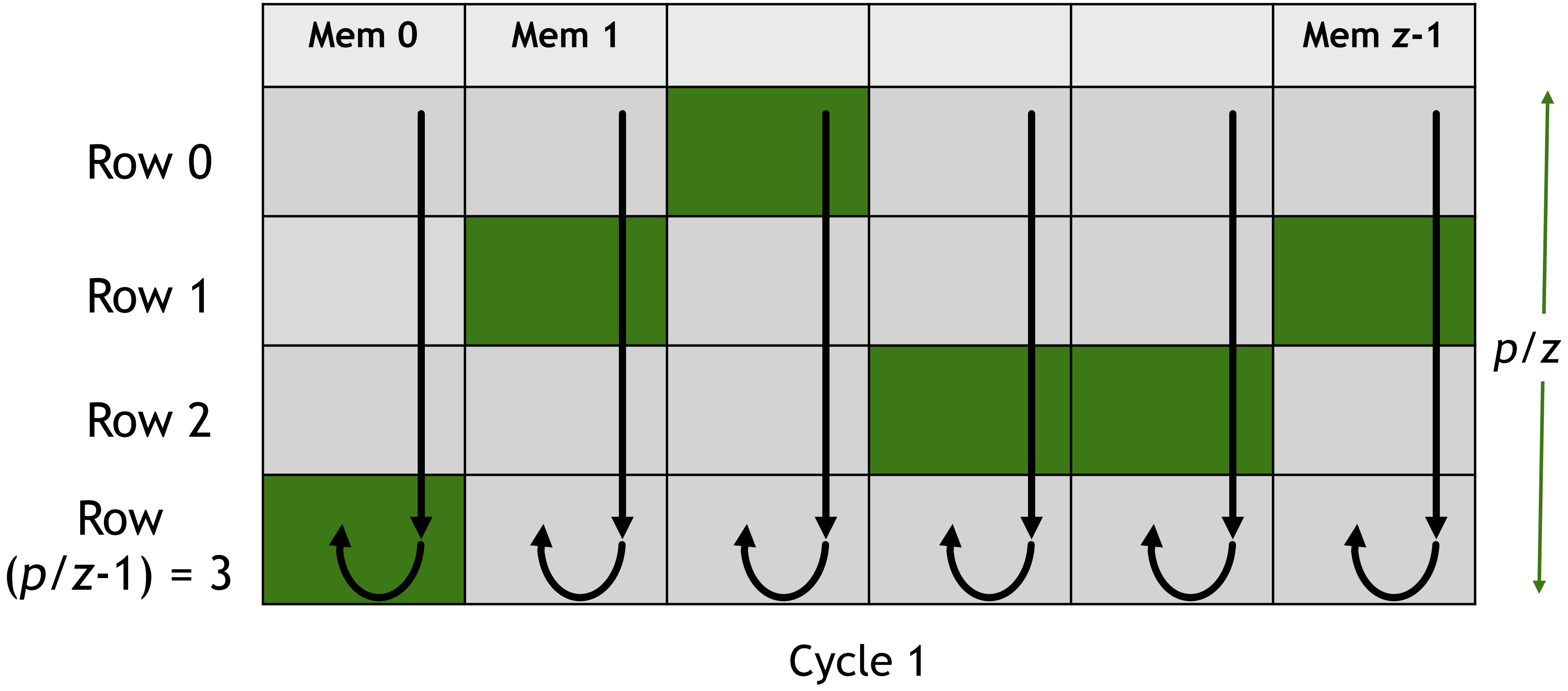
Clash-free access ☺



Clash stalls processing ☹

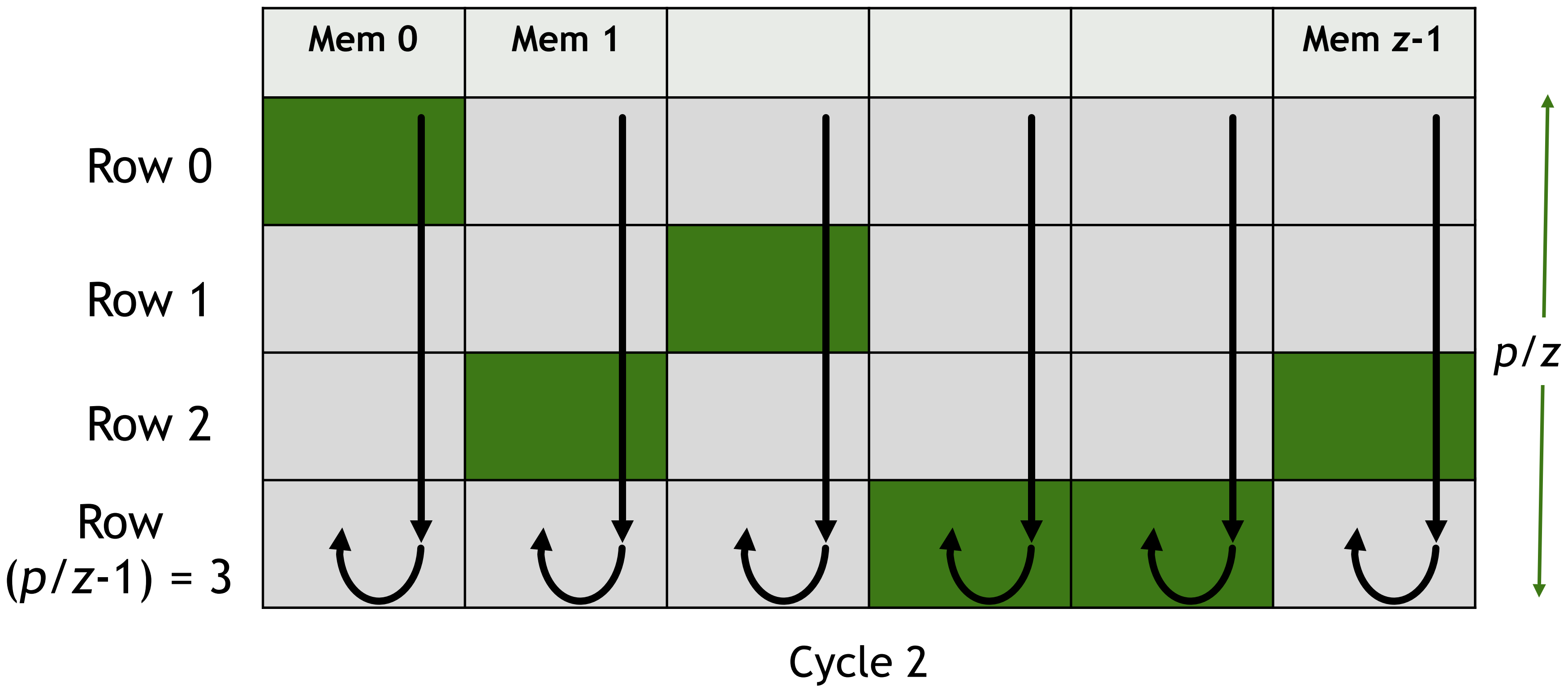
Interleaver must prevent clashes when accessing activations

Ease of Accesses



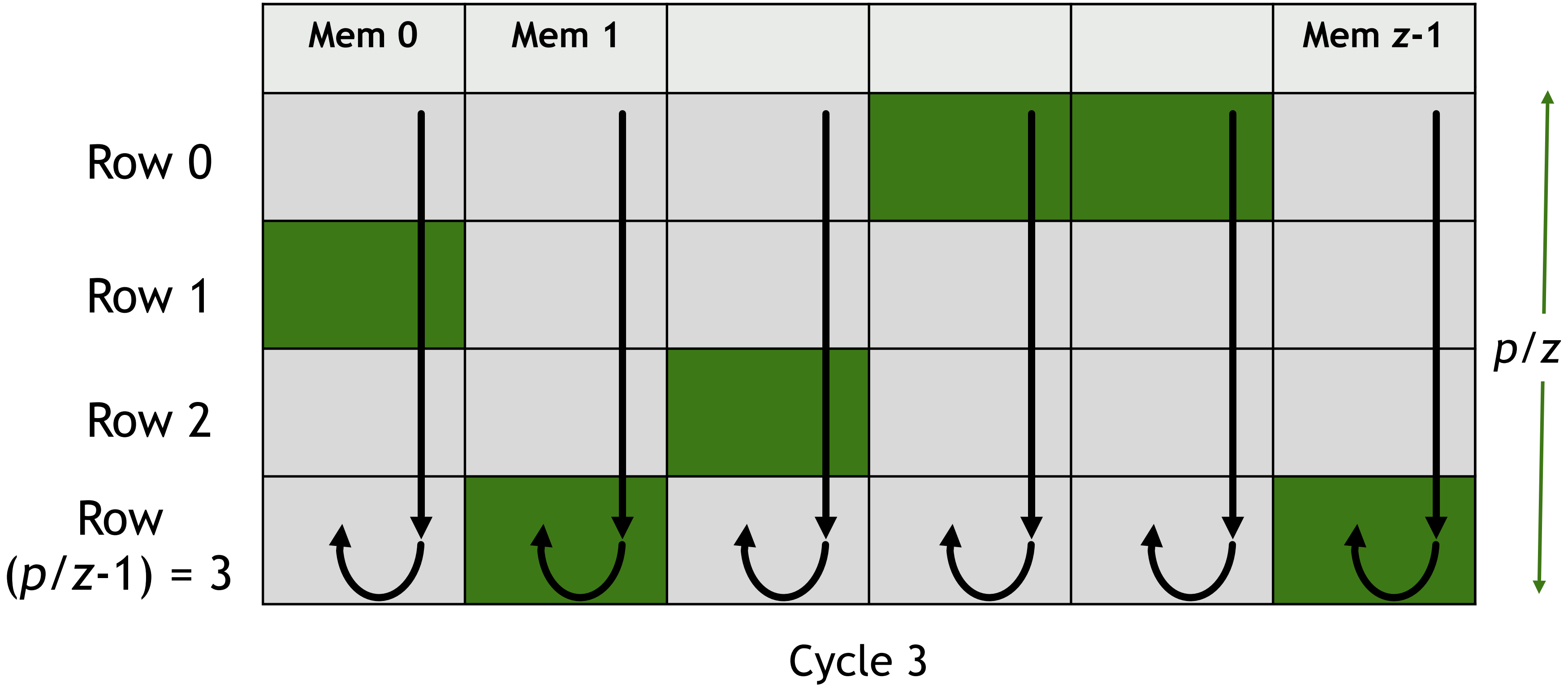
Design interleaver to have easy address computation

Ease of Accesses



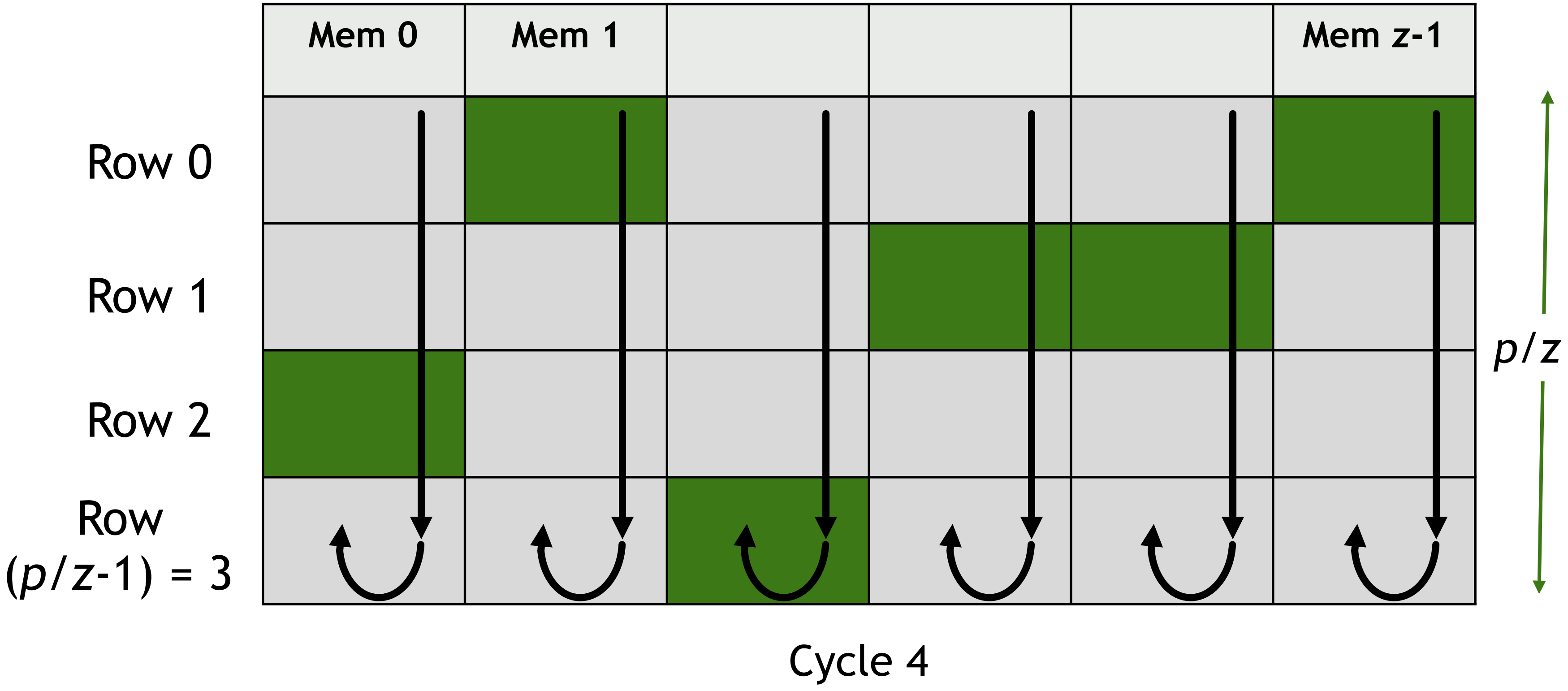
Design interleaver to have easy address computation

Ease of Accesses



Design interleaver to have easy address computation

Ease of Accesses



Design interleaver to have easy address computation

Interleaver Design Algorithm

- Let $\boldsymbol{r}$ be a random permutation of memory row index => Size p/z
- Replicate or partition r to form $\boldsymbol{s}$ of z elements => Starting indices of all mems
- $\boldsymbol{t}$ = {s, s+1, …, s+p/z-1}%(p/z) => All p indices for all mems in order

$$\pi_W(i) = \left(\boldsymbol{t}\left[\boldsymbol{i}\%\boldsymbol{p}\right] \times \boldsymbol{z} + \boldsymbol{i}\%\boldsymbol{z}\right) \times fo + \lfloor i/p \rfloor$$

Interleaver Design Algorithm

Example: p=32, fo=2, z=8 => $i \in [0,63]$. Say i = 45

$$\pi_W(i) = \left(\boldsymbol{t}\left[\boldsymbol{i}\%\boldsymbol{p}\right] \times \boldsymbol{z} + \boldsymbol{i}\%\boldsymbol{z} \right) \times fo + \lfloor i/p \rfloor$$

- Let ***r*** be a random permutation of memory row index => Size p/z
- Replicate or partition r to form ***s*** of z elements => Starting indices of all mems
- ***t*** = {s, s+1, …, s+p/z-1}%(p/z) => All p indices for all mems in order

Interleaver Design Algorithm

Example: p=32, fo=2, z=8 => $i\in[0,63]$. Say i = 45

$$\pi_W(i) = \left(\underbrace{\boldsymbol{t}\left[\boldsymbol{i\%p}\right]}_{\text{Activation Memory Bank Row = 45\%32 = 1}} \times \boldsymbol{z} + \boldsymbol{i\%z}\right) \times fo + \lfloor i/p \rfloor$$

Row1	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}

- Let ***r*** be a random permutation of memory row index => Size p/z
- Replicate or partition r to form ***s*** of z elements => Starting indices of all mems
- ***t*** = {s, s+1, …, s+p/z-1}%(p/z) => All p indices for all mems in order

Interleaver Design Algorithm

- Let ***r*** be a random permutation of memory row index => Size *p*/*z*
- Replicate or partition *r* to form ***s*** of *z* elements => Starting indices of all mems
- ***t*** = {*s*, *s*+1, …, *s*+*p*/*z*-1}%(p/z) => All *p* indices for all mems in order

Example: *p*=32, *fo*=2, *z*=8 => *i*∈[0,63]. Say *i* = 45

$$\pi_W(i) = \left(\boldsymbol{t}\left[\boldsymbol{i}\%\boldsymbol{p}\right] \times \boldsymbol{z} + \boldsymbol{i}\%\boldsymbol{z}\right) \times fo + \lfloor i/p \rfloor$$

Activation Memory Bank Row = 45%32 = 1

Row1	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}

a_{13} **Left side Neuron = 1x8+45%8 = 13**

Interleaver Design Algorithm

- Let ***r*** be a random permutation of memory row index => Size *p*/*z*
- Replicate or partition *r* to form ***s*** of *z* elements => Starting indices of all mems
- ***t*** = {*s*, *s*+1, …, *s*+*p*/*z*-1}%(p/z) => All *p* indices for all mems in order

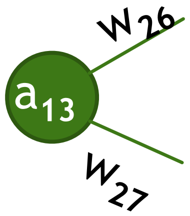
Example: *p*=32, *fo*=2, *z*=8 => $i \in [0,63]$. Say *i* = 45

$$\pi_W(i) = \left(\boldsymbol{t}\left[\boldsymbol{i}\%\boldsymbol{p}\right] \times \boldsymbol{z} + \boldsymbol{i}\%\boldsymbol{z}\right) \times fo + \lfloor i/p \rfloor$$

Activation Memory Bank Row = 45%32 = 1

Row1	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}

a_{13} **Left side Neuron = 1x8+45%8 = 13**



Left side Neuron's Weight = 13x2 = 26

Sourya Dey, USC

Interleaver Design Algorithm

- Let r be a random permutation of memory row index => Size p/z
- Replicate or partition r to form s of z elements => Starting indices of all mems
- $t = \{s, s+1, \ldots, s+p/z-1\}\%(p/z)$ => All p indices for all mems in order

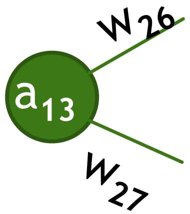
Example: p=32, fo=2, z=8 => $i \in [0,63]$. Say i = 45

$$\pi_W(i) = \left(t\left[i\%p \right] \times z + i\%z \right) \times fo + \lfloor i/p \rfloor$$

Activation Memory Bank Row = 45%32 = 1

Row1	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}

a_{13} **Left side Neuron = 1x8+45%8 = 13**



Left side Neuron's Weight = 13x2 = 26

Weight Offset = 1

Interleaver Design Algorithm

- Let r be a random permutation of memory row index => Size p/z
- Replicate or partition r to form s of z elements => Starting indices of all mems
- t = {s, s+1, …, s+p/z-1}%(p/z) => All p indices for all mems in order

Example: p=32, fo=2, z=8 => $i \in [0,63]$. Say i = 45

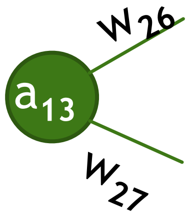
$$\pi_W(i) = \left(\boldsymbol{t}\left[\boldsymbol{i\%p}\right] \times \boldsymbol{z} + \boldsymbol{i\%z}\right) \times fo + \lfloor i/p \rfloor$$

Activation Memory Bank Row = 45%32 = 1

Row1	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}

a_{13} **Left side Neuron = 1x8+45%8 = 13**

Left side Neuron's Weight = 13x2 = 26



Weight Offset = 1

Finally: w_{27}

Meeting Requirements

- Easily generated – Proof in paper
  - All variables involved are powers of 2 (add extra neurons)
    - Modulo = Bit select
    - Multiply = Bit shift
  - Only store *r* for a new pattern
    - Create t by accumulating 1s

- Clash freedom – Proof in paper

Variations

- Start Vector Shuffle (SV)

Original: s = {2,0,3,1,2,0,3,1}

After SV: s = {2,0,3,1,**3,0,1,2**}

- Sweep Starter Shuffle (SS)

Original:
1st sweep s = {2,0,3,1,2,0,3,1}
2nd sweep s = {2,0,3,1,2,0,3,1}

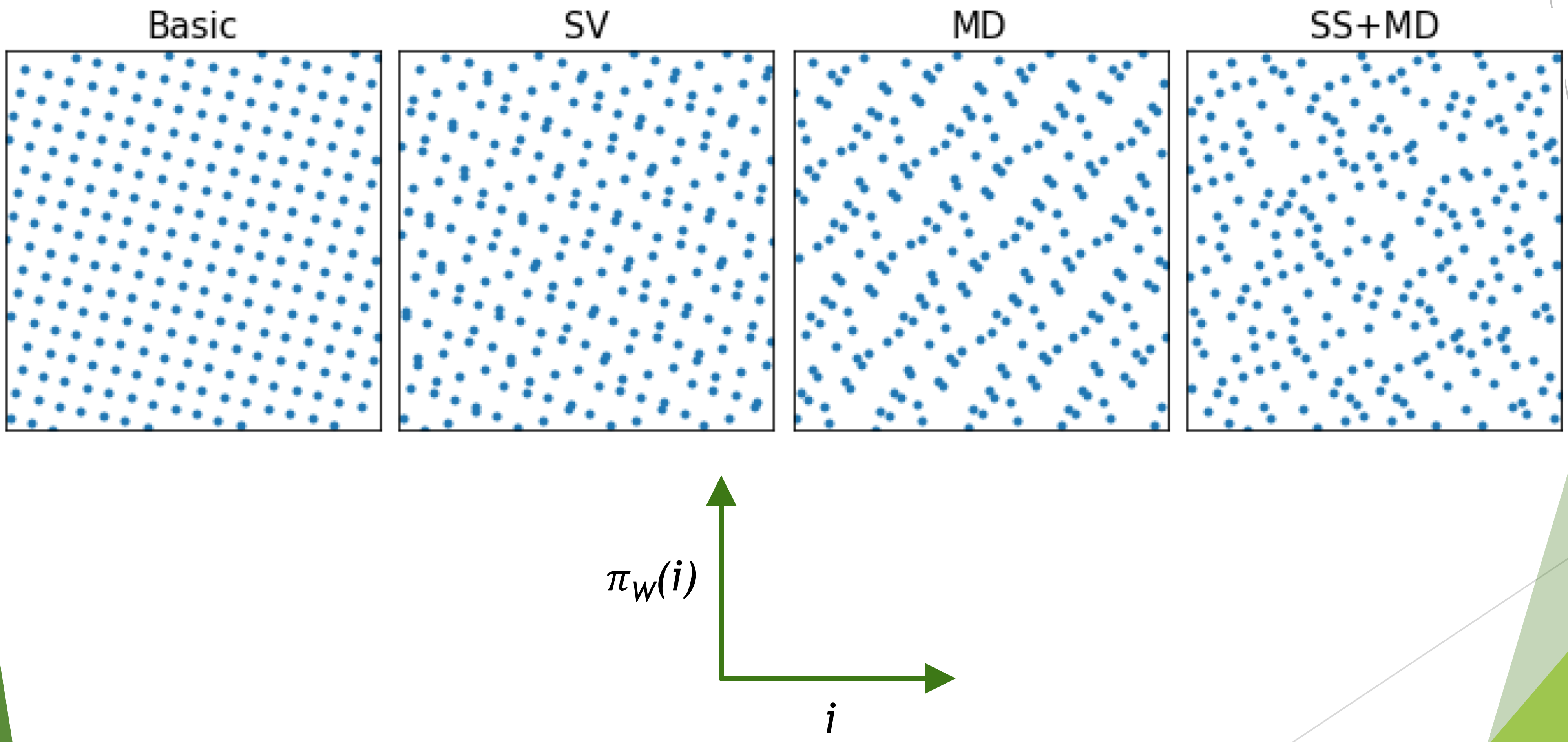
After SS:
1st sweep s = {2,0,3,1,3,0,1,2}
2nd sweep s = {**0,3,2,1,0,3,2,1**}

- Memory Dither (MD)

$$\pi_W(i) = \left(t\left[i\%p\right] \times z + \boldsymbol{v}[i\%z]\right) \times fo + \lfloor i/p \rfloor$$

v[.] = Permutation of [0,z-1]

Some Weight Interleaver Patterns

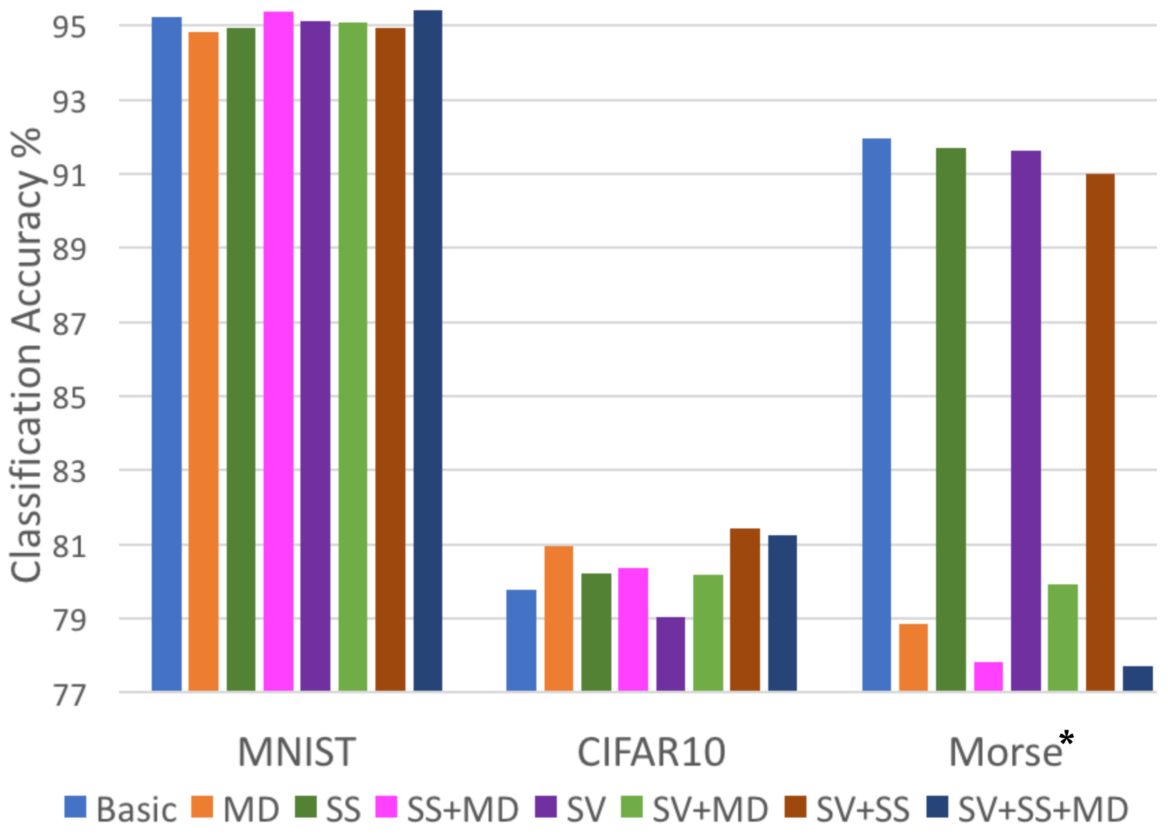


Spread and Dispersion

- Spread: Connections that are close on 1 side should be far away on other
- Dispersion: Connections should be irregular, i.e. no patterns or trends

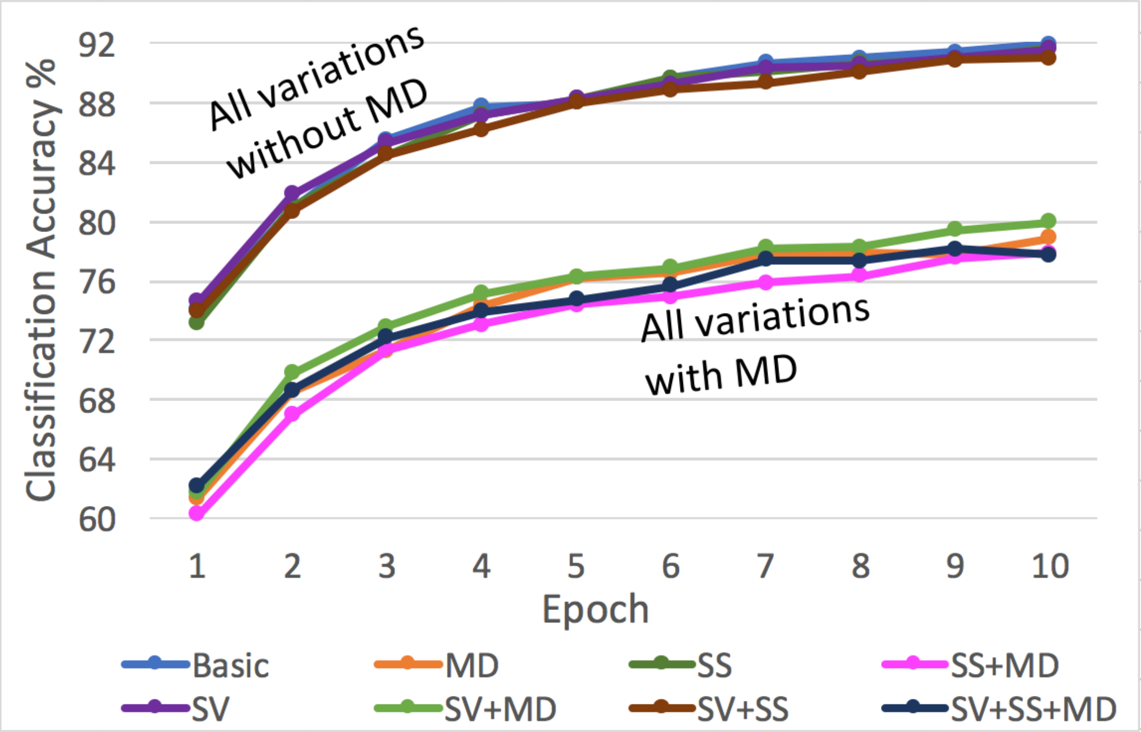
Interleaver Variant	Spread	Dispersion
Basic	**18.28**	0.04
MD	7.48	0.22
SS	9.7	0.07
SS + MD	6.5	0.37
SV	6.6	0.08
SV + MD	7.31	0.23
SV + SS	5.05	0.09
SV + SS + MD	5.7	**0.39**

Dataset Results



* Sourya Dey: https://cobaltfolly.wordpress.com/2017/10/15/morse-code-dataset-for-artificial-neural-networks/

Morse Dataset Trends



Morse has fewer inputs and low redundancy

Spread should be high, dispersion hurts

Summary and Ongoing Work

- Pre-defined sparse hardware architecture to lower memory and computational footprint
- Interleaver algorithm to guarantee clash freedom and ease of storage
- Interleaver variations and effects on performance

- Extension to multiple junctions - Adjacency matrices
- Measures to characterize network performance

Dey, S., Huang, K.W., Beerel, P.A., Chugg, K.M.: Characterizing Sparse Connectivity Patterns in Neural Networks. In: ICLR-2018 (submitted for publication)

Thank you!

Questions?

Contact: souryade@usc.edu