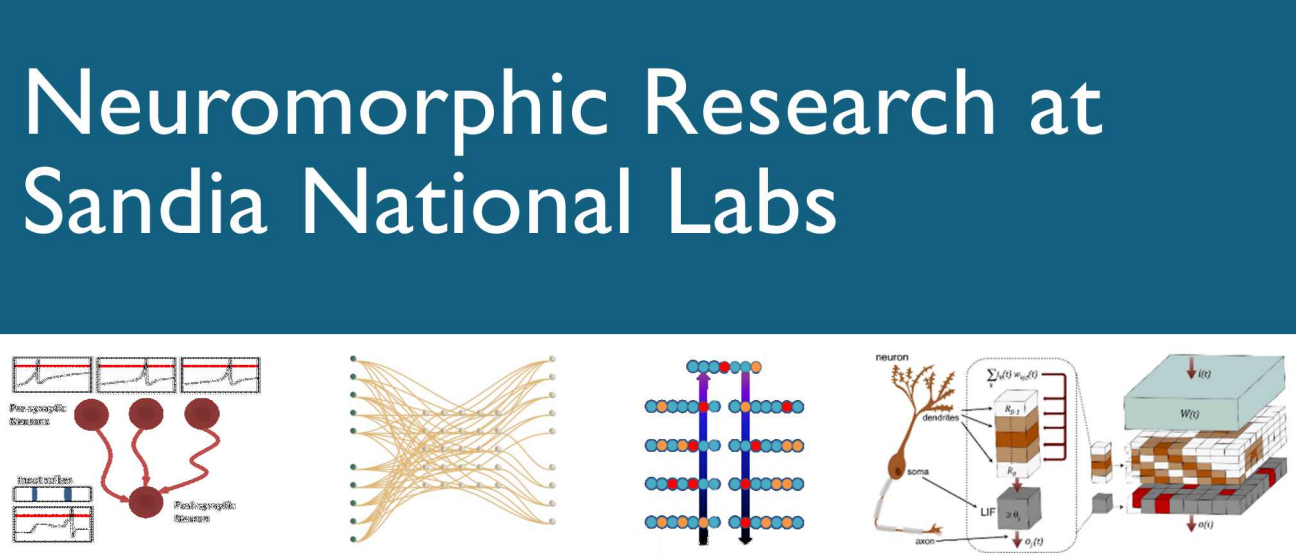
SAND2019-1532C

Neuromorphic Research at Sandia National Labs



PRESENTED BY

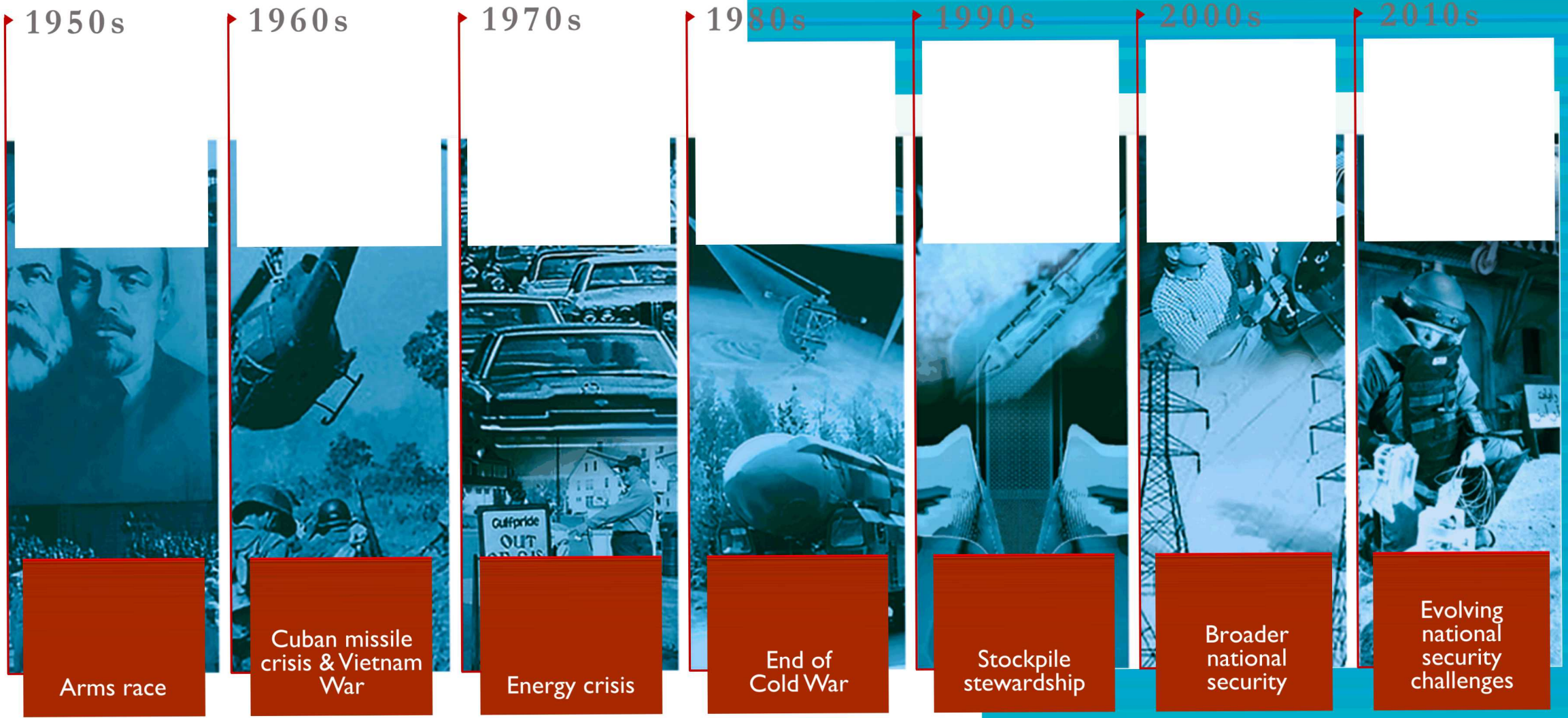
Craig M. Vineyard

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

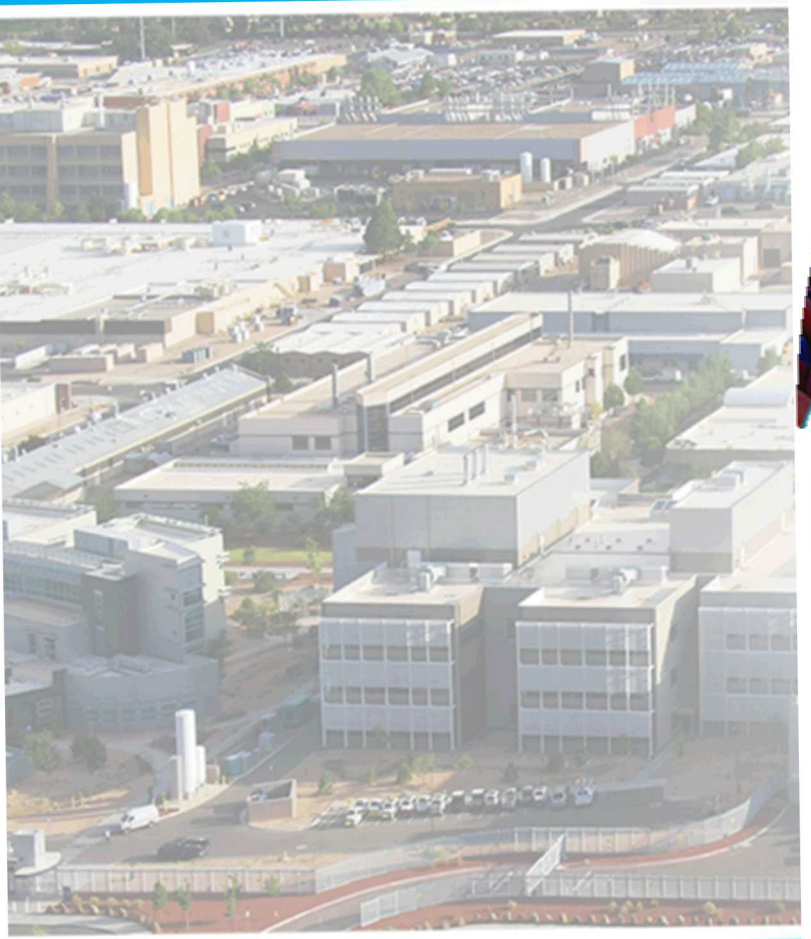
PURPOSE STATEMENT DEFINES WHAT WE DO

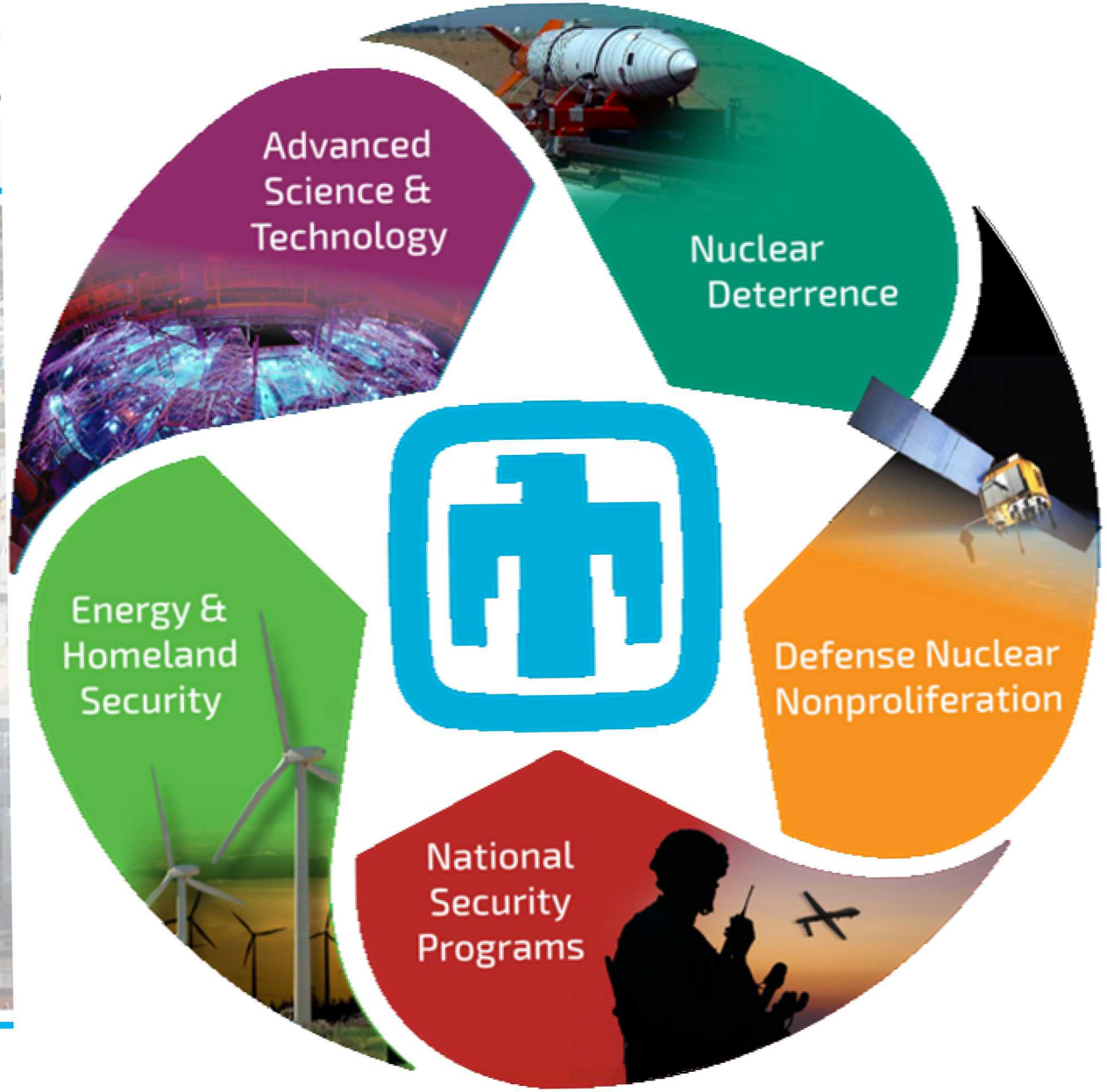
Sandia develops advanced technologies to ensure global peace

SANDIA ADDRESSES NATIONAL SECURITY CHALLENGES



SANDIA HAS FIVE MA
PROGRAM PORTFOL





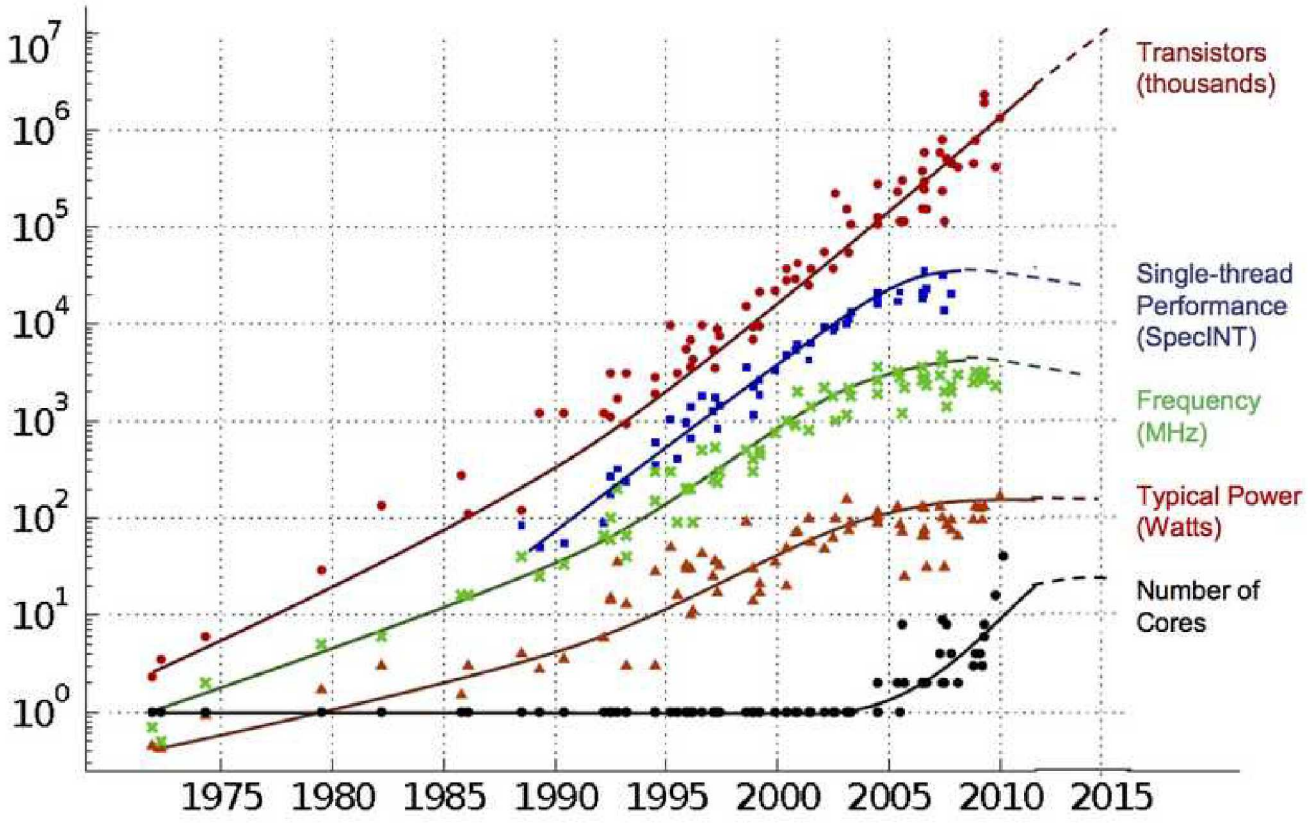
Dennard scaling

- As transistors get smaller, their power density remains constant

Unfortunately ended 10-15 years ago

- Cannot run CPUs at faster speeds
- Emphasis on multi-core

Need for new paradigm of computing

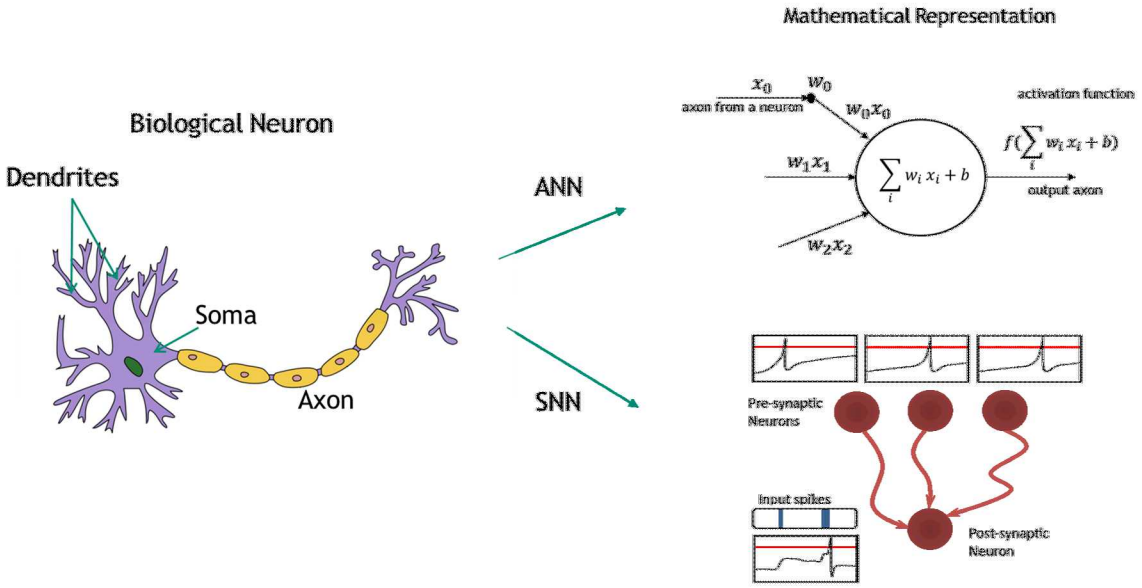


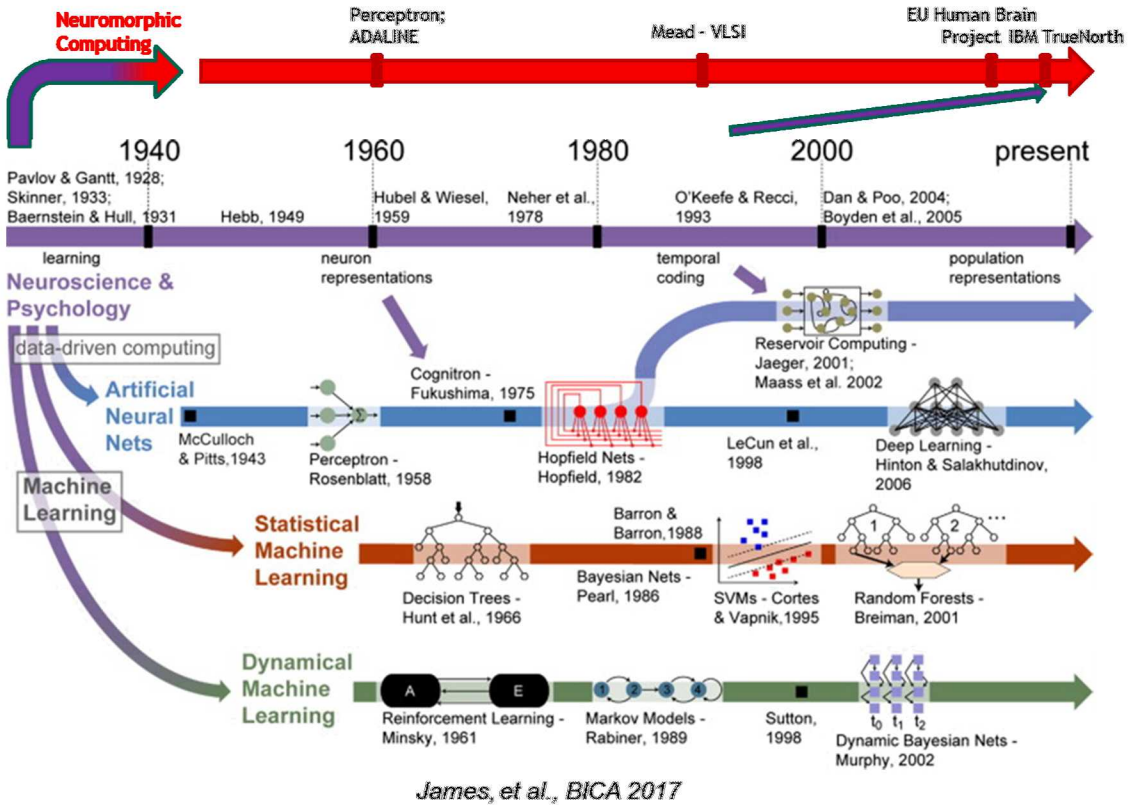
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Neural-Inspired Computing

What is neural-inspired, neuromorphic, brain-inspired computing?

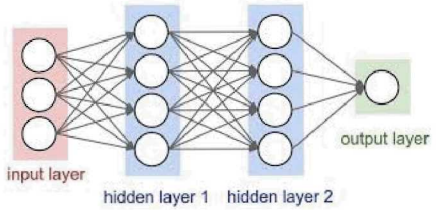
- Many terms
- Fundamental notion of taking inspiration from how the brain performs computation



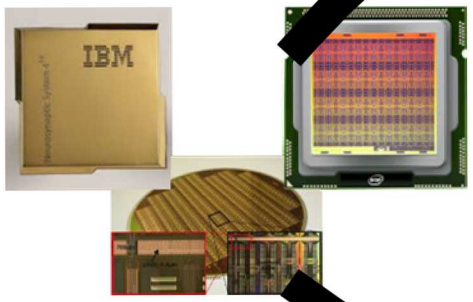


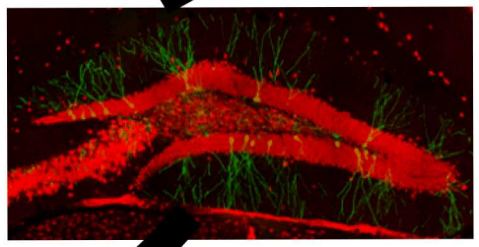
James, et al., BICA 2017

What is the best path from "neural" computing to *neural computing*?

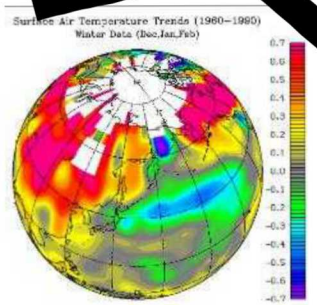


AI Path

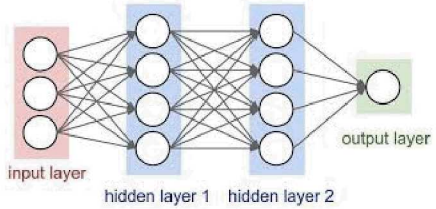




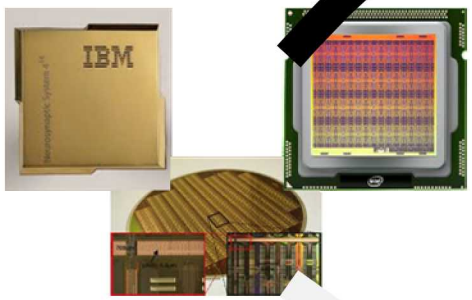
Scientific Computing Path

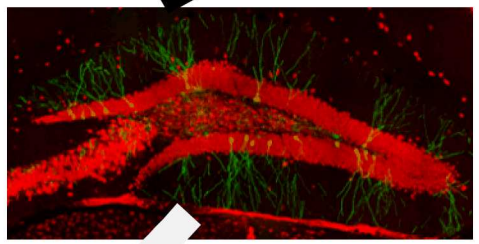


What is the best path from "neural" computing to *neural computing*?



AI Path





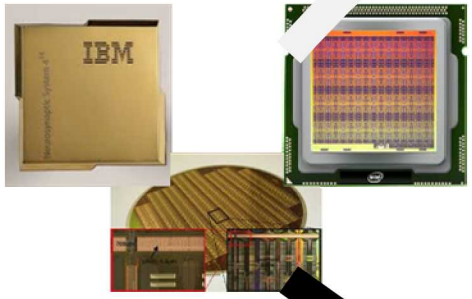
- Artificial neural networks "look" like neural hardware architectures and the brain
- ANNs emulate functions like image classification attributed to cognition
- Considerable attention on accelerating ANN function, especially inference

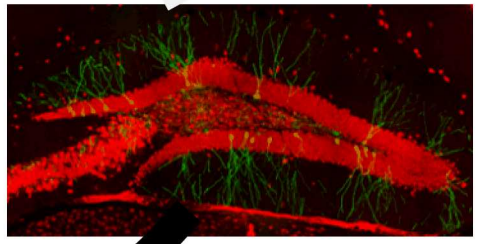
- ANNs impose many abstractions that are *not* brain-like, nor required in neural hardware
- ANN-brain functional similarity is shallow (1960s neuroscience…) and narrow (limited to sensory systems)
- ANN hardware acceleration is emphasizing features orthogonal to neural hardware value proposition

What is the best path from "neural" computing to *neural computing*?

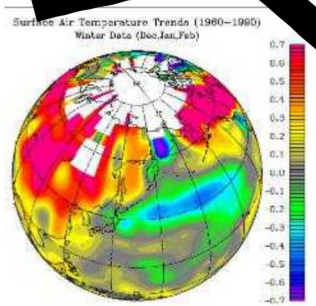
- Scientific computing requires significant precision
- General focus on tasks not performed cognitively
- Current scaling challenges are being addressed by hybrid CPU / GPU systems

- Neural hardware and algorithms can precise
- Neural logic is programmable just like any other logic
- Not all scientific computing tasks fit GPUs well

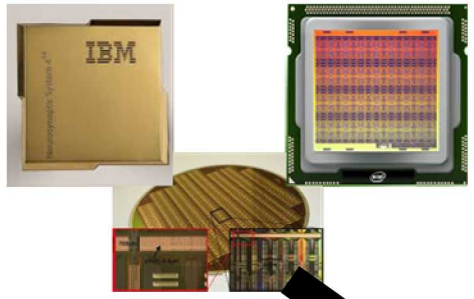


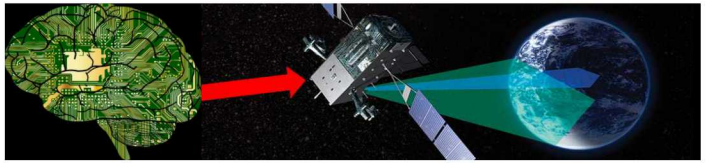


Scientific Computing Path



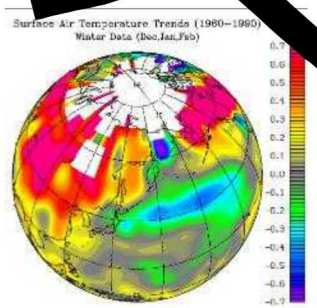
What is the best path from "neural" computing to *neural computing*?





Scientific Computing Path

Applications



WHETSTONE

Neural Algorithms

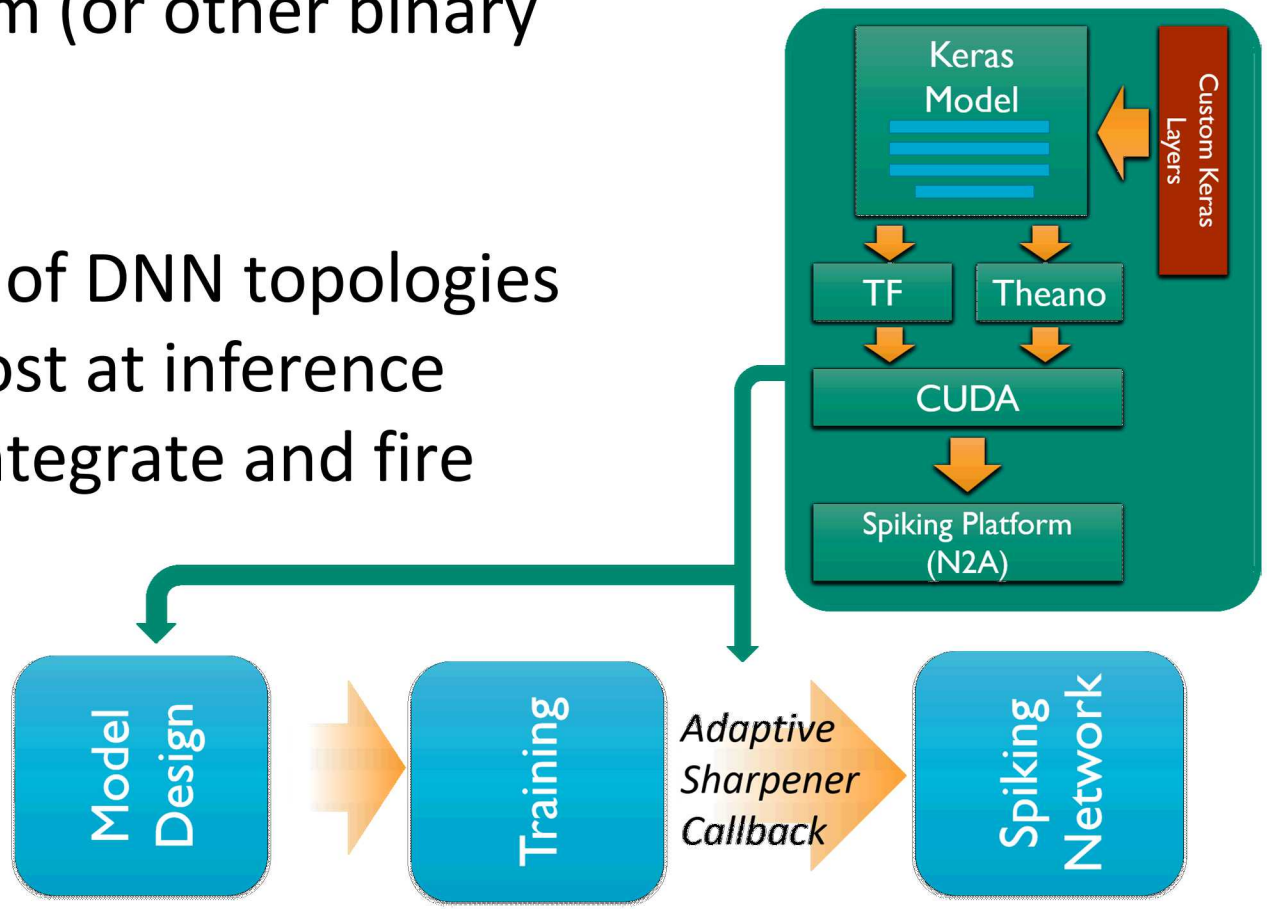
FUGU

Whetstone Overview

WHETSTONE

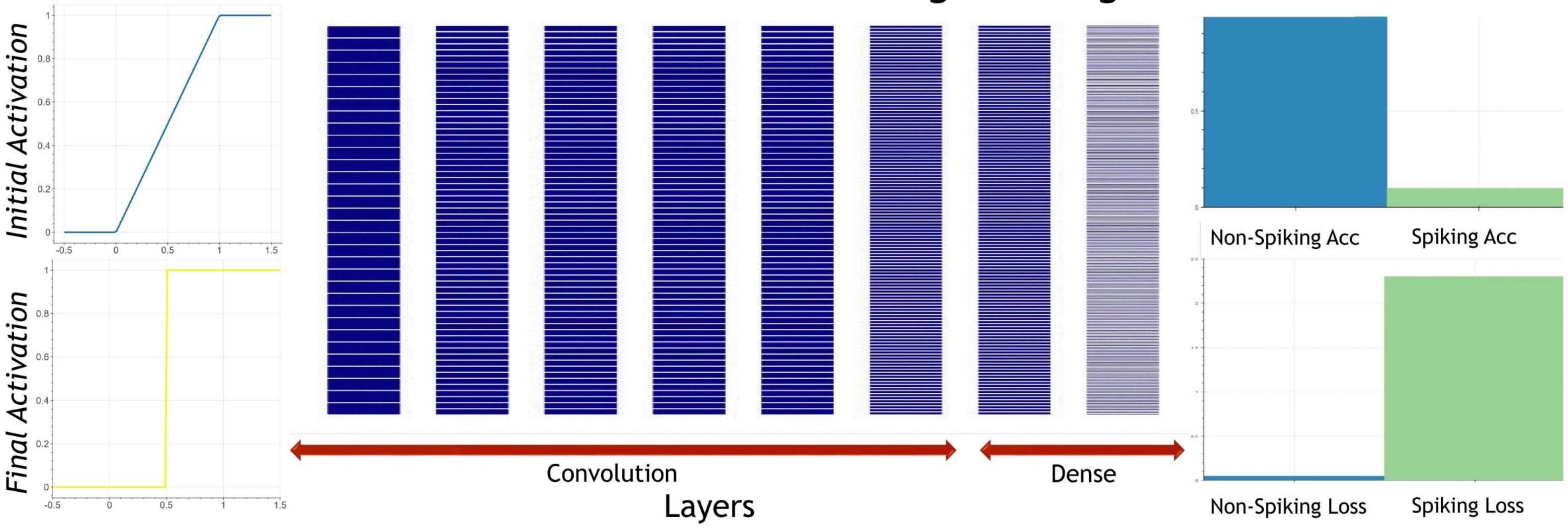
Whetstone provides a drop-in mechanism for tailoring a DNN to a spiking hardware platform (or other binary threshold activation platforms)

- Hardware platform agnostic
- Compatible with a wide variety of DNN topologies
- No added time or complexity cost at inference
- Simple neuron requirements: Integrate and fire



The real challenge for deep learning on spiking is the threshold activation function.

Using Whetstone, activation functions converge to a threshold activation *during training.*



Whetstone Overview

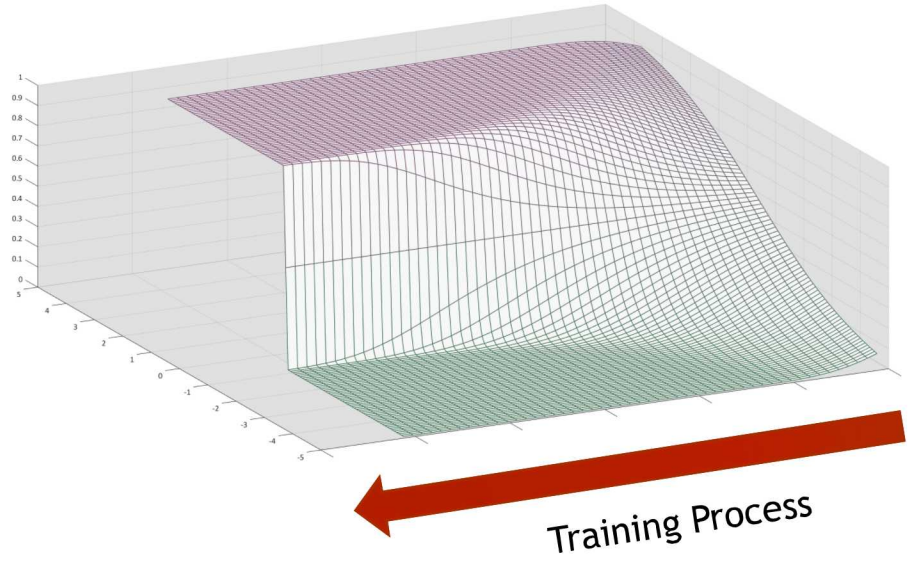
- Generally, gradient descent generates a sequence of weights A_i with the goal of

minimizing the error of $f(A_i x)$ in predicting the ground truth y.

- We generalize this by replacing the activation function f with a sequence f_k such that $f_k \to_{L_1} f$, where f is now the threshold activation function.
- Now, the optimizer must

minimize the error of $f_k(A_i x)$ in predicting y.

- Since the convergence in **neither i nor k is uniform**, this is a mathematically dangerous idea
- However, with a little care and a few tricks, the method reliably converges in many cases.



When/Where do we decide to ‘sharpen’ the activations?

1) Bottom-up Sharpening (The ‘toothpaste tube’ method)
   - Begin sharpening at the bottom layer
   - Wait until previous layer is fully sharpened
   - Increases stability of convergence
2) Adaptive Sharpening Callback
   - Hand-tuning sharpening rates is hard
   - Instead, use loss as a guide for an *adaptive sharpener*
   - Adaptive sharpener implemented as a callback automatically adjusts sharpening based on loss thresholds

Original Model Example

```
⋮
model.add(Dense(256))
model.add(Activation(‘relu’))
model.add(Dense(10))
model.add(Activation(‘softmax’))
⋮
model.fit(x,y)
⋮
```

Modified Model Example

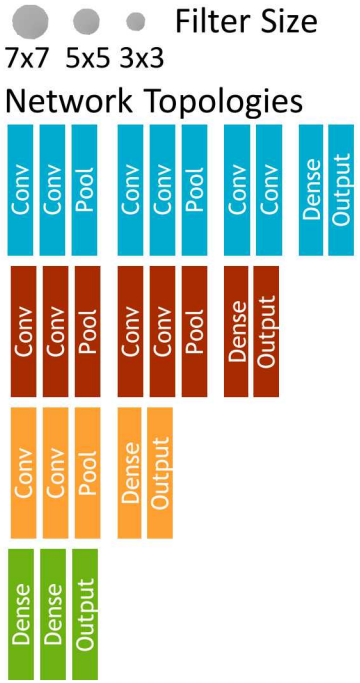
```
⋮
model.add(Dense(256))
model.add(Spiking_BRelu())
model.add(Dense(10))
model.add(Spiking_Brelu())
Model.add(Softmax_Decode(key))
⋮
sharpener = AdaptiveSharpener()
model.fit(x,y,callbacks=[sharpener])
⋮
```

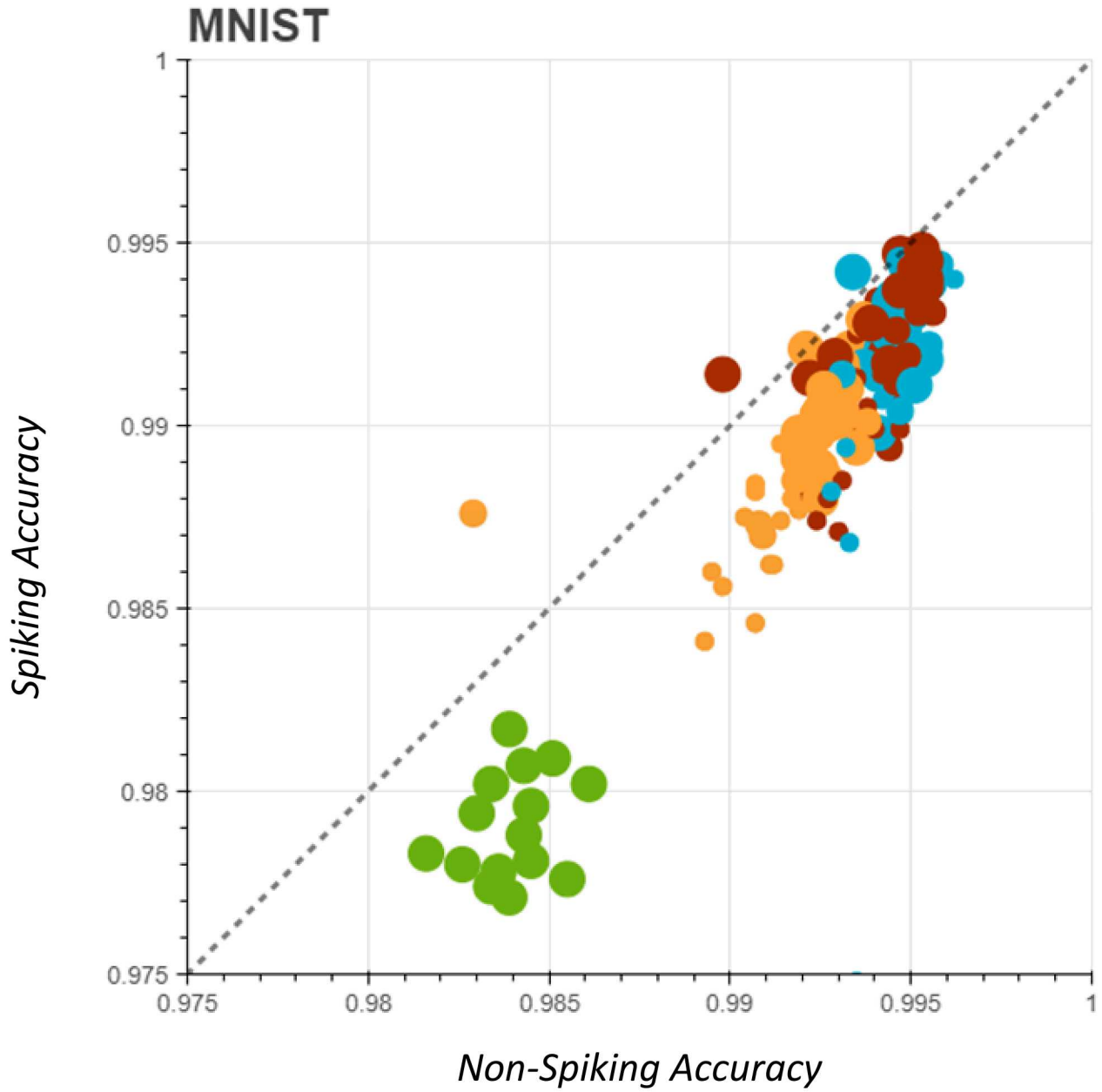
Results

Method	MNIST	CIFAR-10
Whetstone (VGG-like)	**0.9953**	**0.8467**
Whetstone (10-net ensemble)	**0.9953**	**0.8801**
Eliasmith, et al.	0.9912	0.8354
EEDN	0.9942	0.8932
Rueckauer, et al.	0.9944	0.9085
BinaryNet	0.9904	0.8985

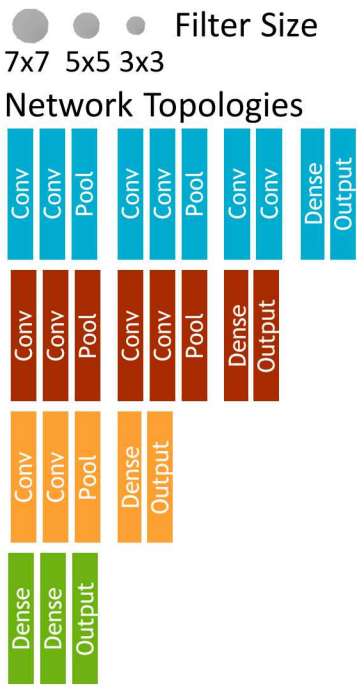
But direct comparisons are hard because each method uses different topologies, pre-processing, etc.

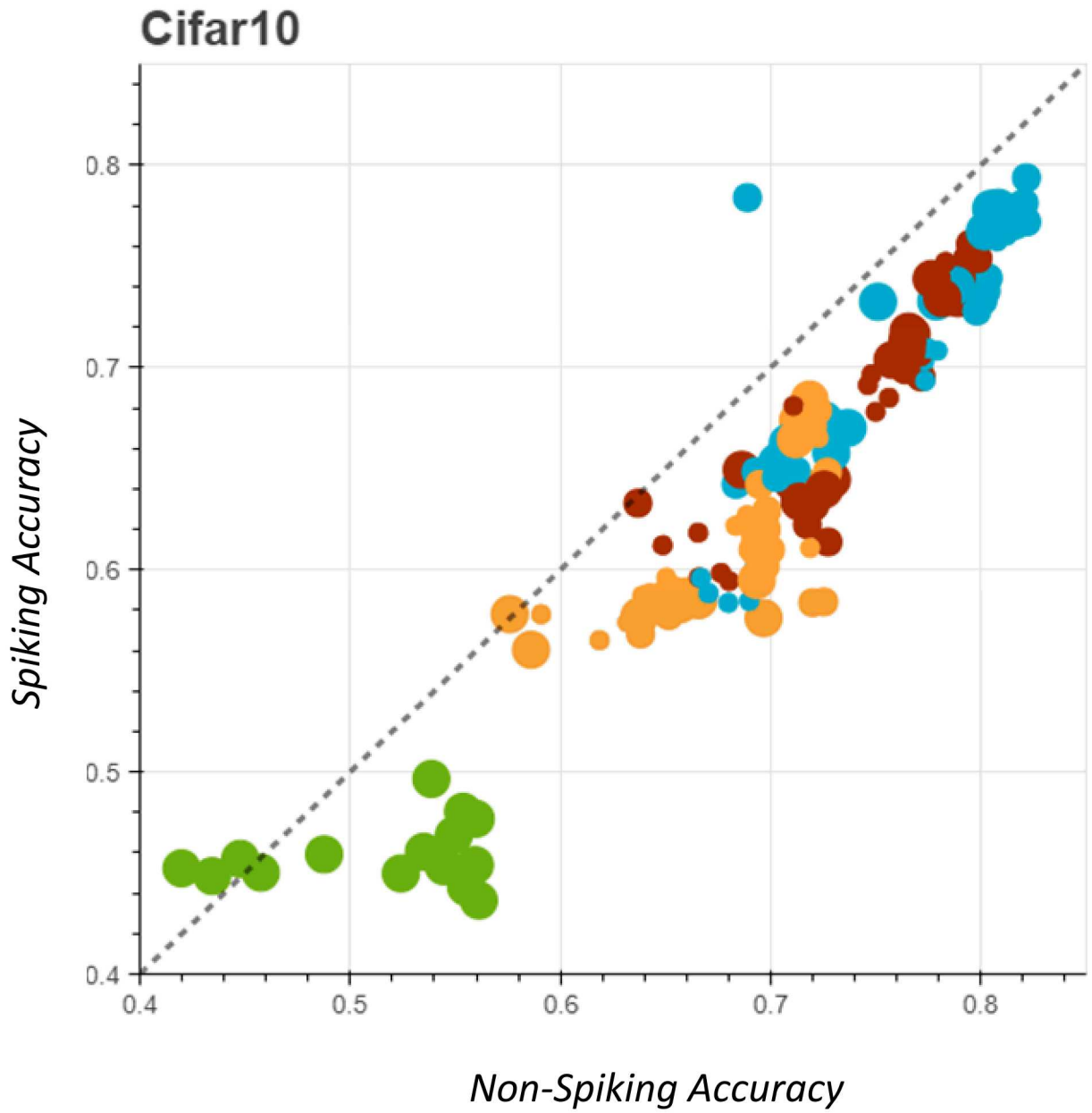
Results


Filter Size
7x7 5x5 3x3
Network Topologies
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Dense
Output
Conv
Conv
Pool
Conv
Conv
Pool
Dense
Output
Conv
Conv
Pool
Dense
Output
Dense
Dense
Output

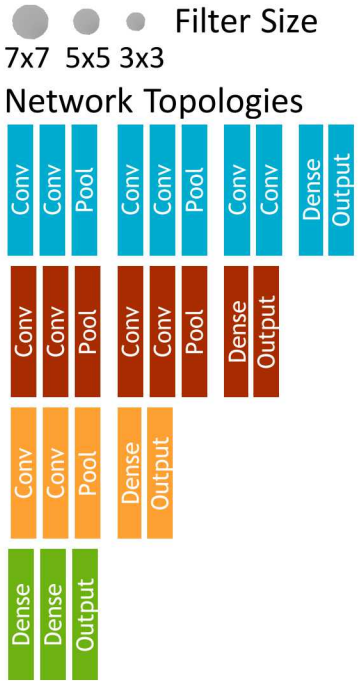

MNIST
Spiking Accuracy
Non-Spiking Accuracy
1
0.995
0.99
0.985
0.98
0.975
0.975
0.98
0.985
0.99
0.995
1

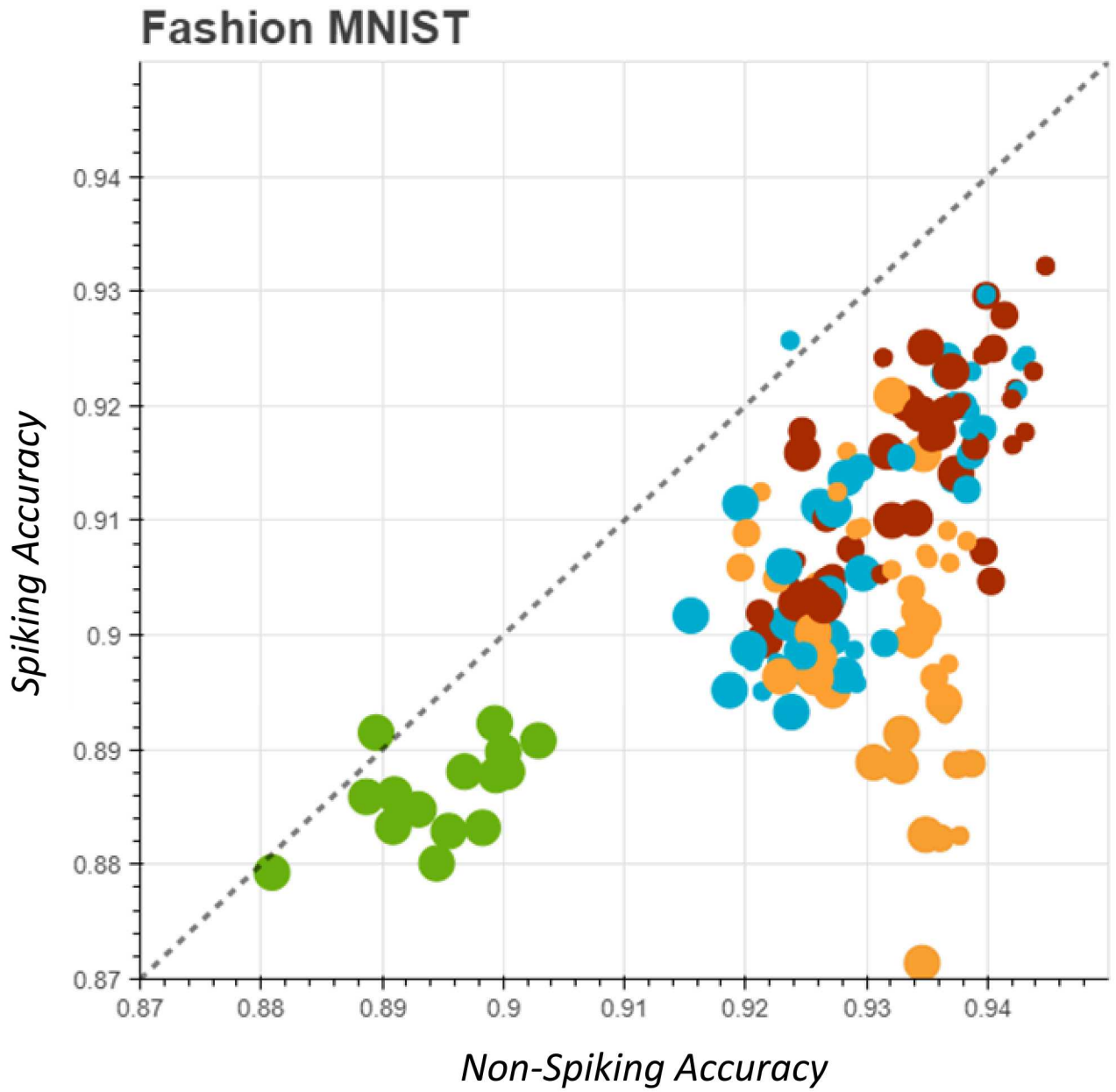
Results


Filter Size
7x7 5x5 3x3
Network Topologies
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Dense
Output
Conv
Conv
Pool
Conv
Conv
Pool
Dense
Output
Conv
Conv
Pool
Dense
Output
Dense
Dense
Output


Cifar10
Spiking Accuracy
Non-Spiking Accuracy
0.4
0.5
0.6
0.7
0.8

Results

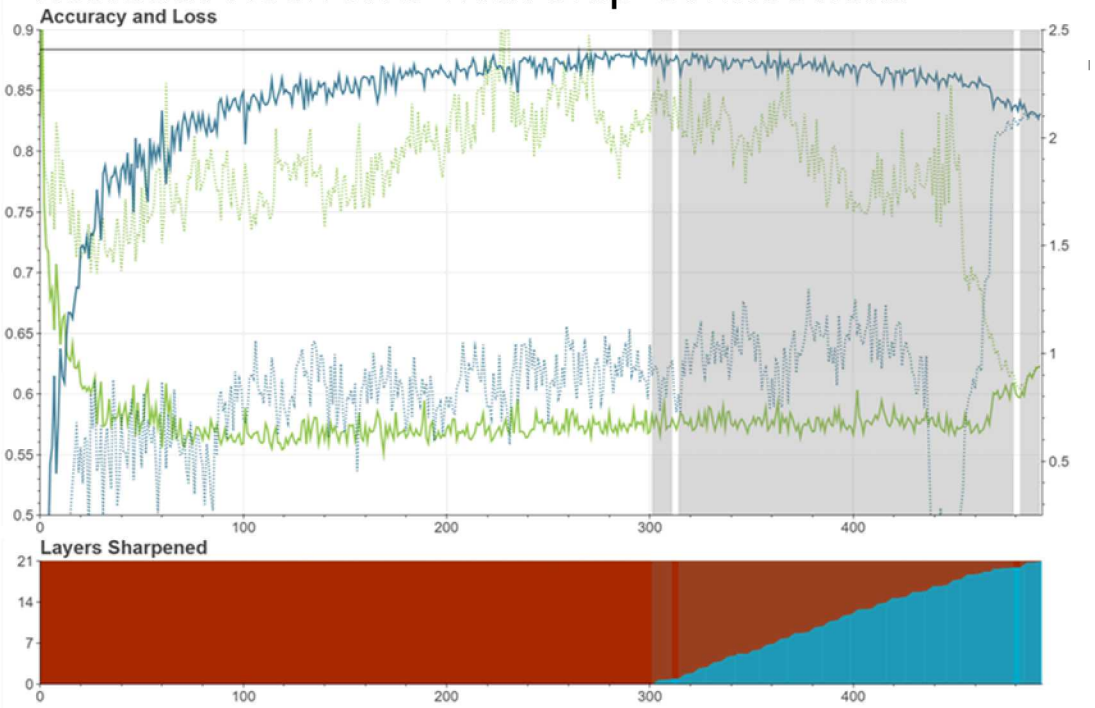

Filter Size
7x7 5x5 3x3
Network Topologies
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Dense
Output
Conv
Conv
Pool
Conv
Conv
Pool
Dense
Output
Conv
Conv
Pool
Dense
Output
Dense
Dense
Output


Fashion MNIST
Spiking Accuracy
Non-Spiking Accuracy
0.87
0.88
0.89
0.9
0.91
0.92
0.93
0.94

Effective Across Various Topologies, Datasets, and Tasks

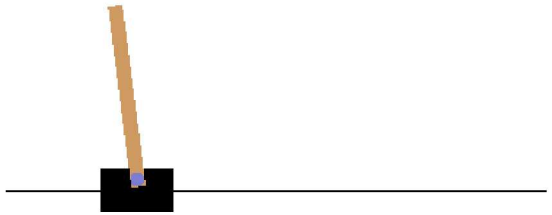
Semantic Segmentation (Trained on COCO Dataset; Videos from HMDB51 Dataset)

Residual Networks with Skip Connections



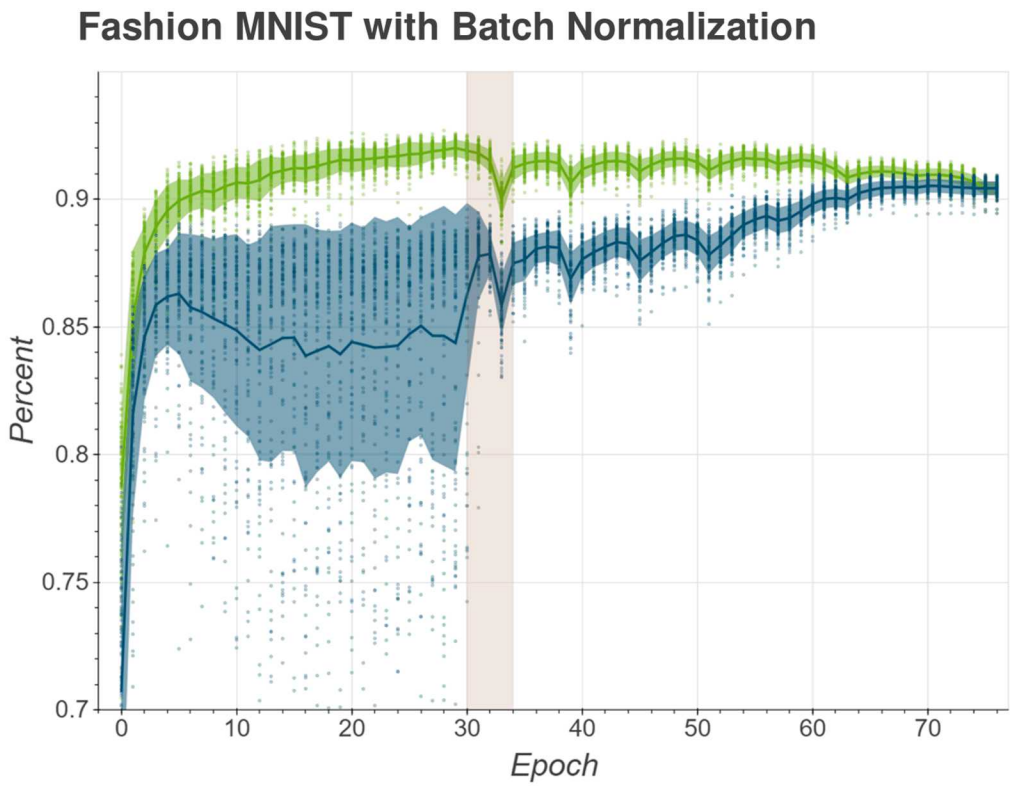
Autoencoders

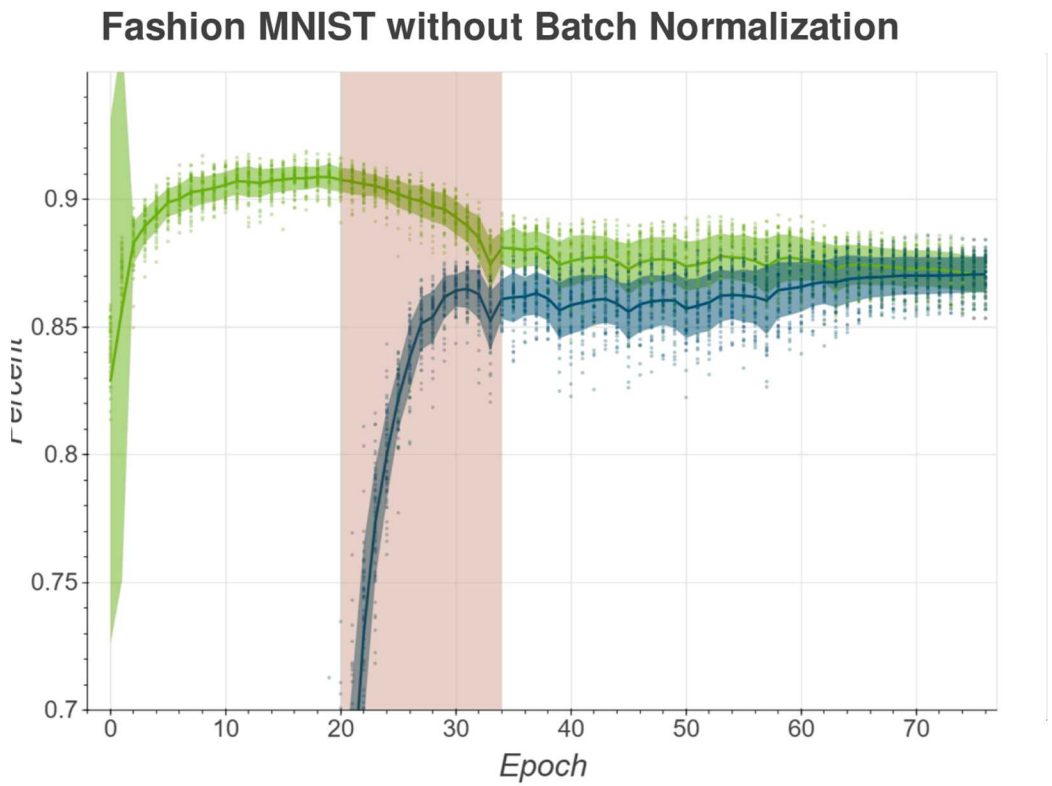
DQN



Established Deep Learning Techniques

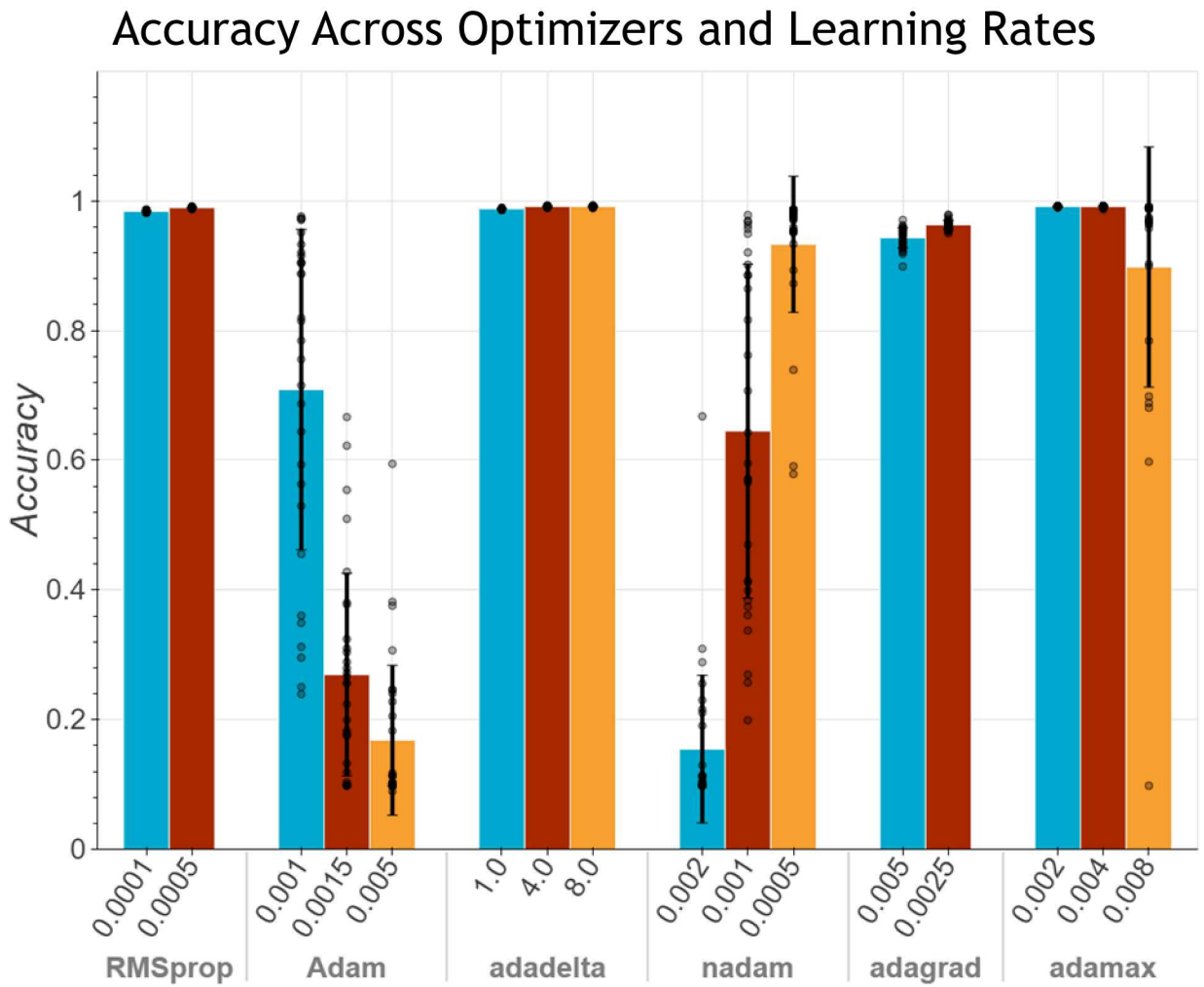
- Batch Normalization helps training stability and network performance
- Improvements across network sizes
- Sharpnening loss, particularly on first sharpening layer, is significantly less
- At inference time, bias (threshold) and weights are modulated according to stats collected during training





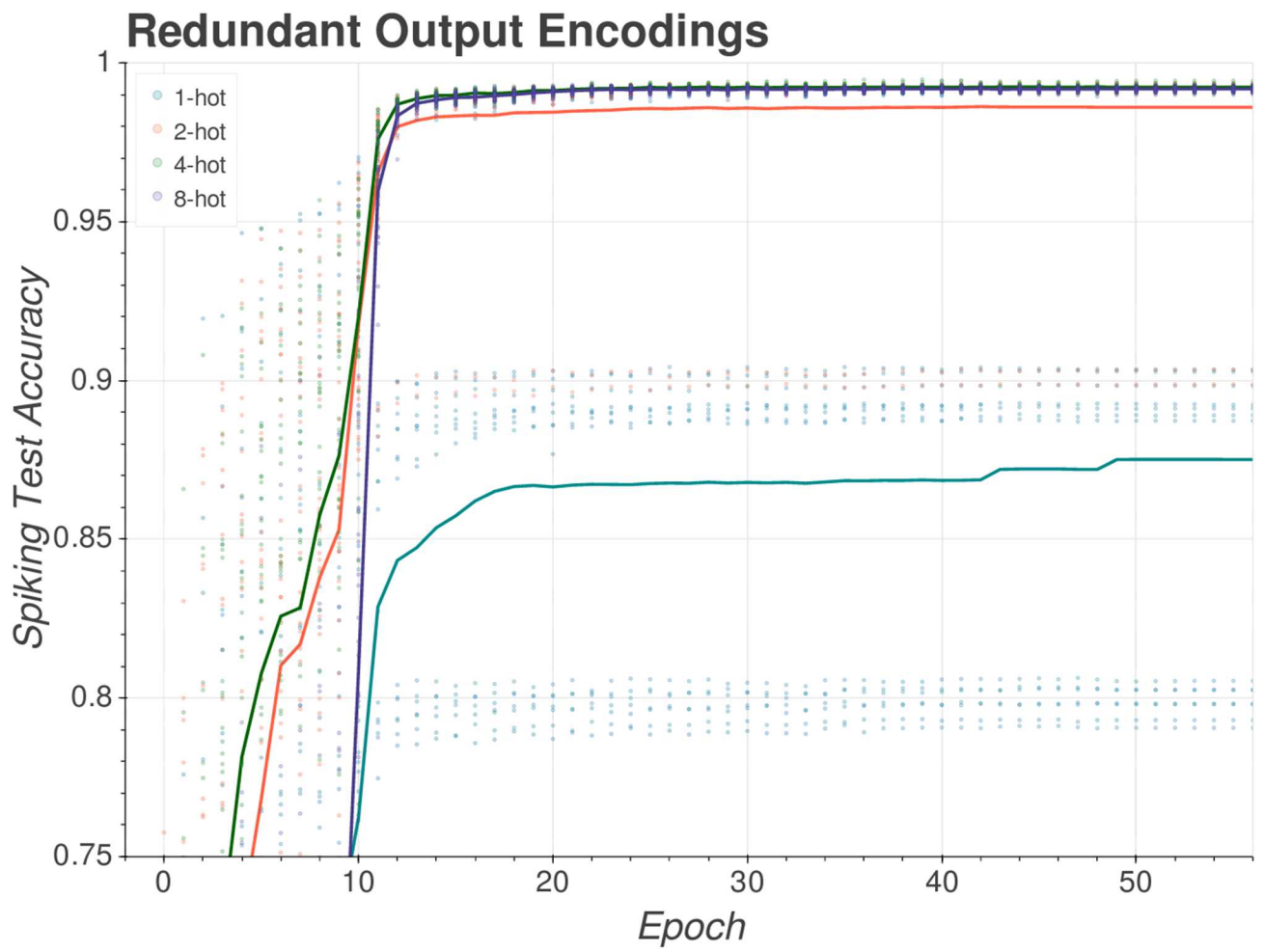
Established Deep Learning Techniques

- Sharpening process is sensitive to optimizer selection
- Adaptive optimizers often work better
- Learning rate modulation by moving average seems to help stability
- A custom Whetstone-aware optimizer is in early stages

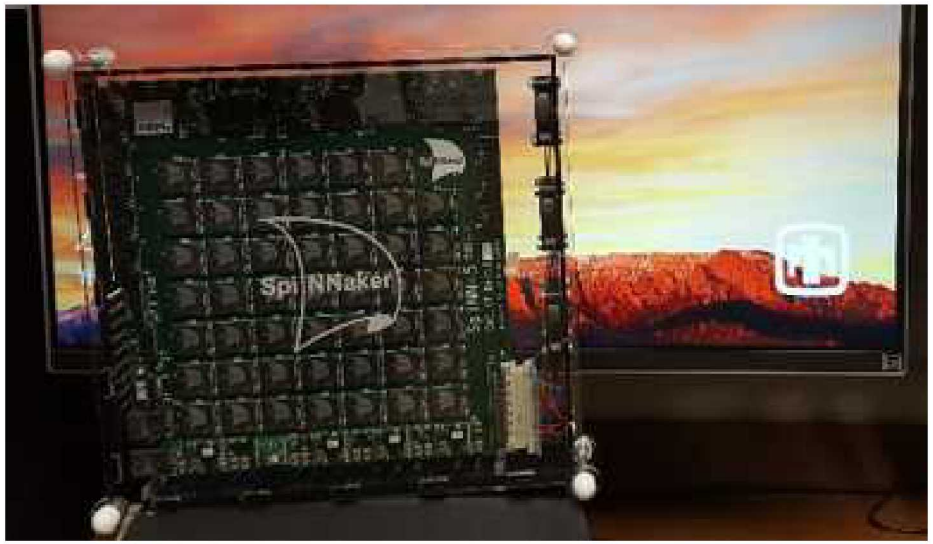


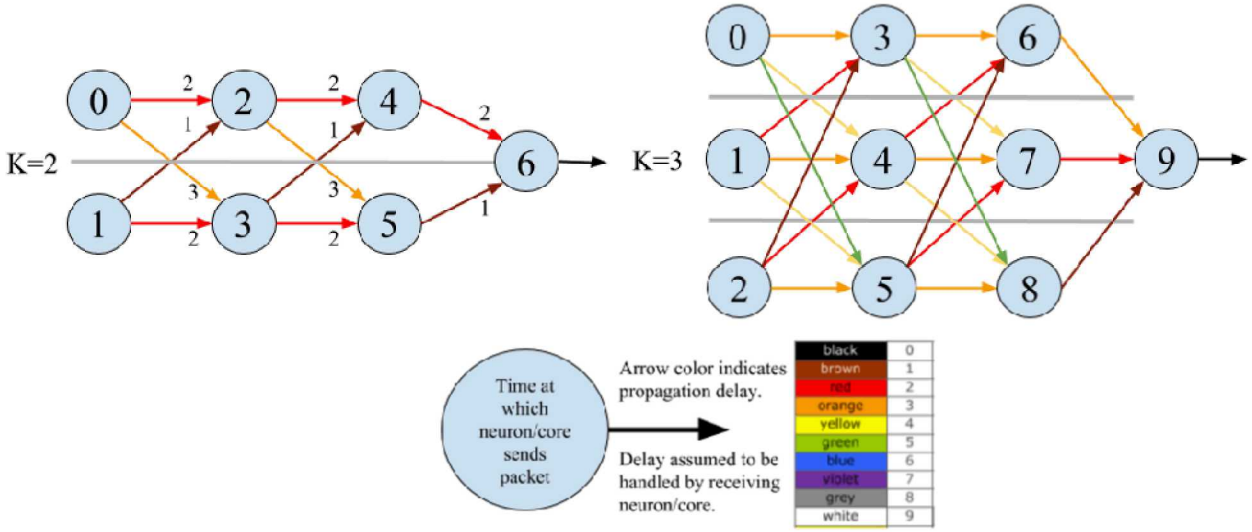
Established Deep Learning Techniques

- The trained neurons can be unreliable
- Redundant output encodings help mitigate this problem
- Similar to ensemble methods
- Reactive neurons feed into softmax during training (for classification)
- During inference, ‘best-matched’ group is used
- On simple datasets, 4-way redundancy is sufficient



Example Domain – Remote Sensing





Network	Small MLP	Medium MLP	Convolution Network
Total Neurons	57000	72500	47640
Total Cores	760	754	371
Total Chips Utilized	48	48	28
Network Tiles	190	29	1
Timescale Factor	5.0	6.0	14.0
Sample Delay (ms)	2	2	28
Throughput (frames/sec)	15317	2340	3.25
Accuracy	94%	97.7%	98.1%

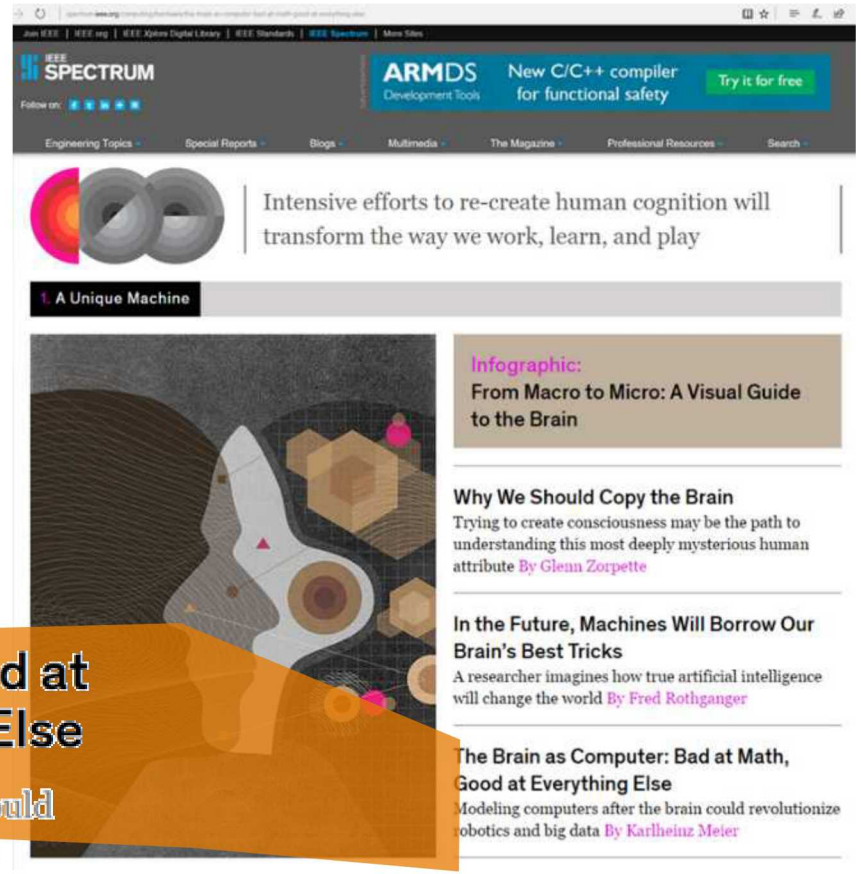
Question

Don’t we use computers because brains aren’t good at math?

Established conventional wisdom:
isn't neural-inspired computing bad at math?

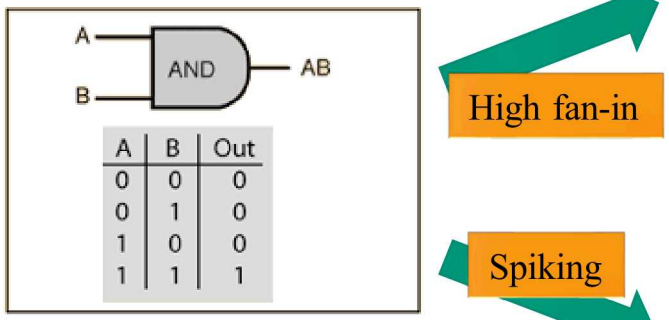
Why?

- It is a challenge to separate brains (cognitive capability) from neurons (low-energy mechanism)
- Belief that neurons are noisy
- Moore's Law – It has always been easier to wait for faster processors than to re-invent numerical computing on specialized parallel architecture



Spiking neurons are a more powerful version of classic logic gates

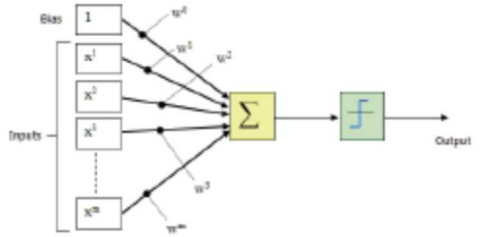
Spiking threshold gates provide high degree of parallelism at very low power



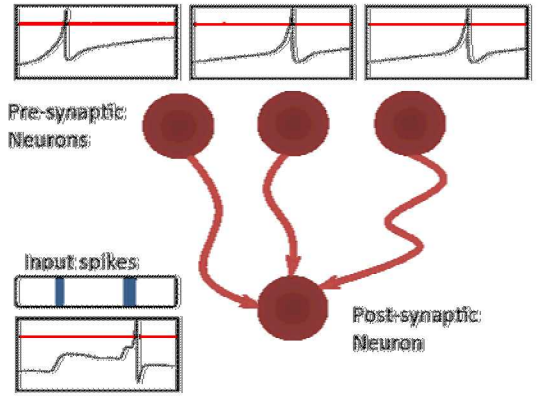
High fan-in

Spiking

Based on a simple McCullogh-Pitts model:



Outputs a 1 if and only if: $w_0 + \sum_{i>0} w_i x_i \geq 0$.



Compute more powerful logic functions

Incorporate time into logic

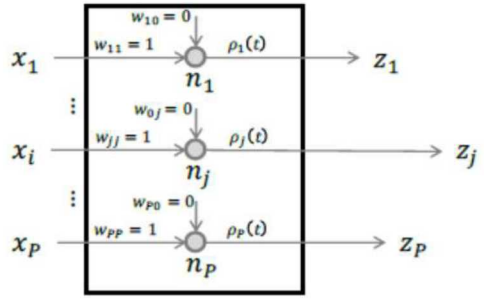
SNL has produced a number of spiking numerical algorithms

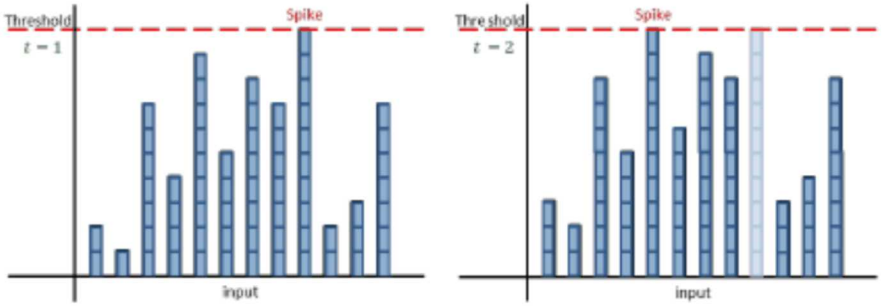
Cross-correlation
- Severa et al., *ICRC 2016*

SpikeSort, SpikeMin, SpikeMax, etc
- Verzi et al., *Neural Computation 2018*

SpikeOptimization
- Verzi et al., *IJCNN 2017*

Sub-cubic (i.e., Strassen) constant depth matrix multiplication
- Parekh et al., *SPAA 2018*





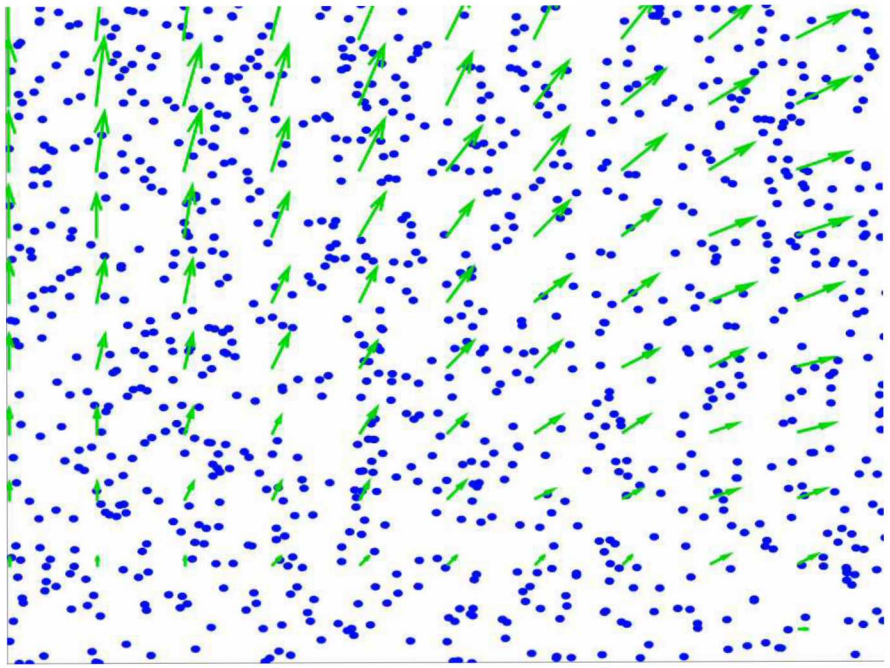
A Velocimetry Application

A motivating application is the determination of the local velocity in a flow field

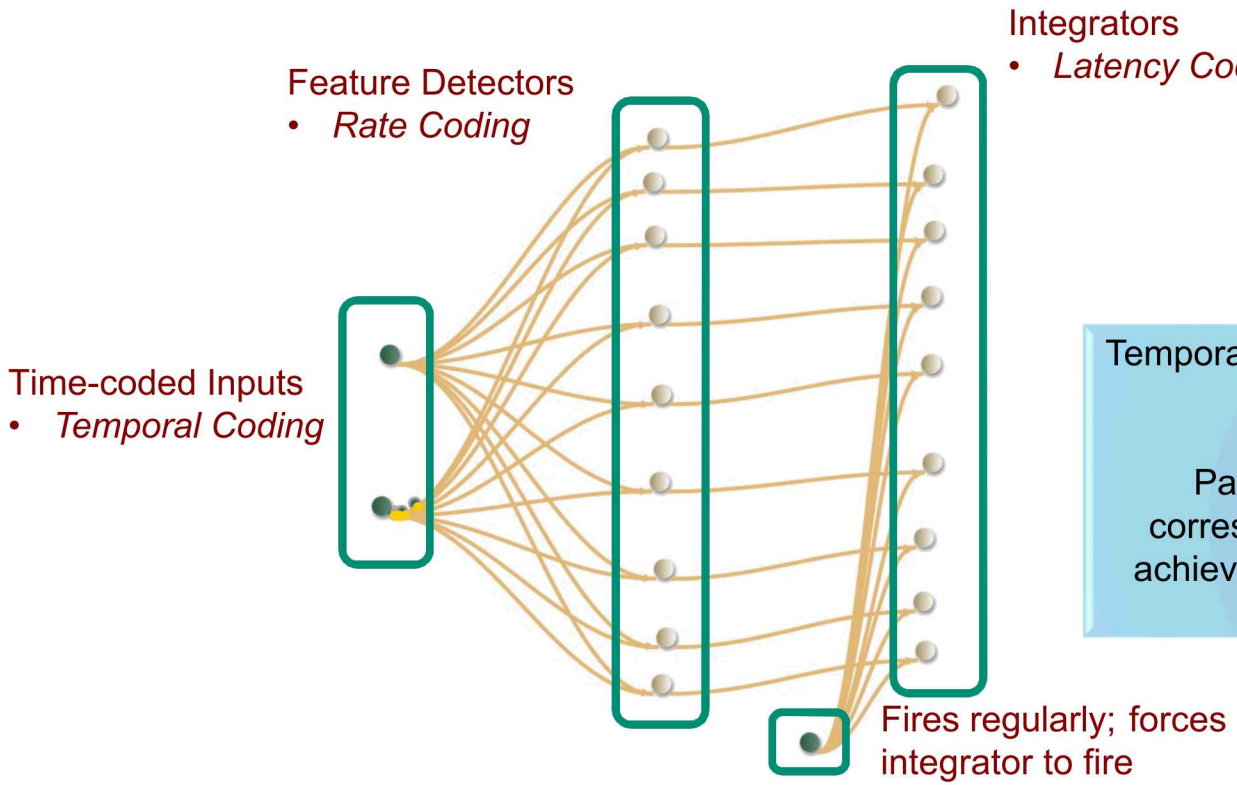
The maximal cross-correlation between two sample images provides a velocity estimate

SNN algorithms are straightforward; exemplify core concepts

- Highly parallel
- Different neural representations
- Modular, precise connectivity
- Time/Neuron tradeoff



Time Multiplexed Cross Correlation

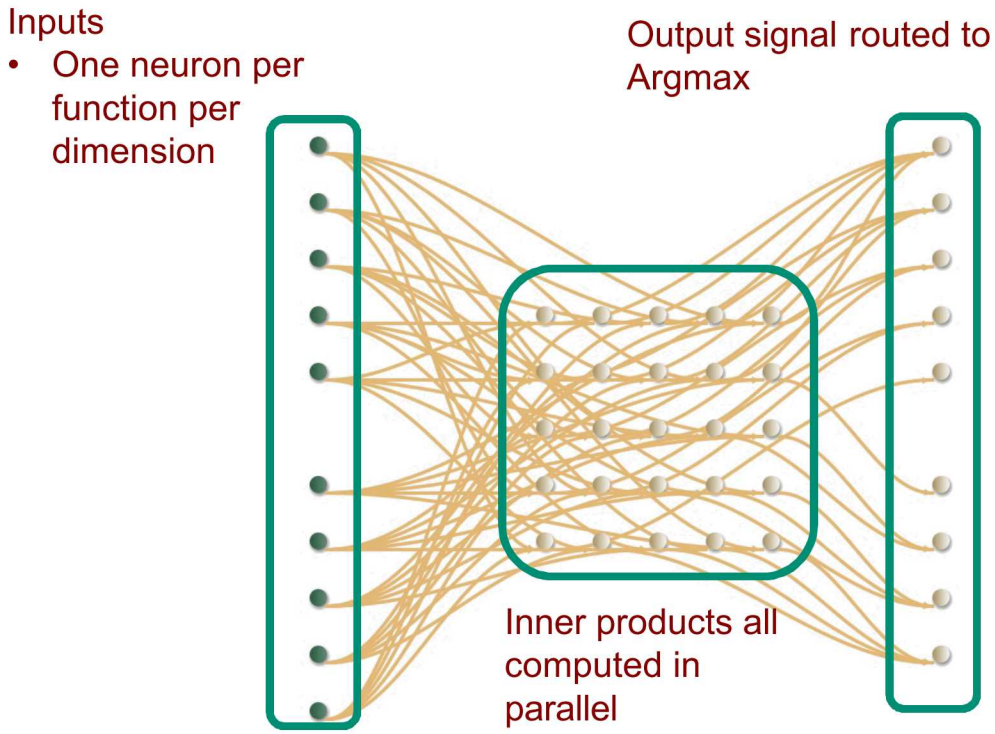


Temporal Coding: $O(n)$ neurons; $O(n)$ runtime

Parallelize inputs and corresponding timesteps to achieve $O(n^2)$ neurons; $O(1)$ runtime

Cross-Correlation Exhibits Time/Neuron Tradeoff

- Exchange Time Cost ↔ Neuron Cost
- Complexity is unchanged
- Neurons: $\boldsymbol{O(n^2)} \leftrightarrow \boldsymbol{O(n)}$
- Time: $\boldsymbol{O(1)} \leftrightarrow \boldsymbol{O(n)}$



Strassen's recursive algorithm for matrix multiplication

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \times \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

Näive
8 multiplications
4 additions

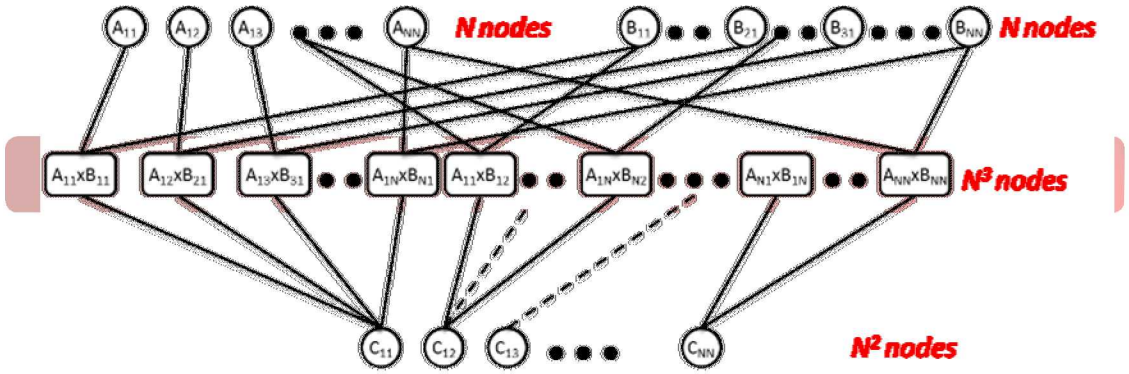
Strassen's
7 multiplications
18 additions/subtractions

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

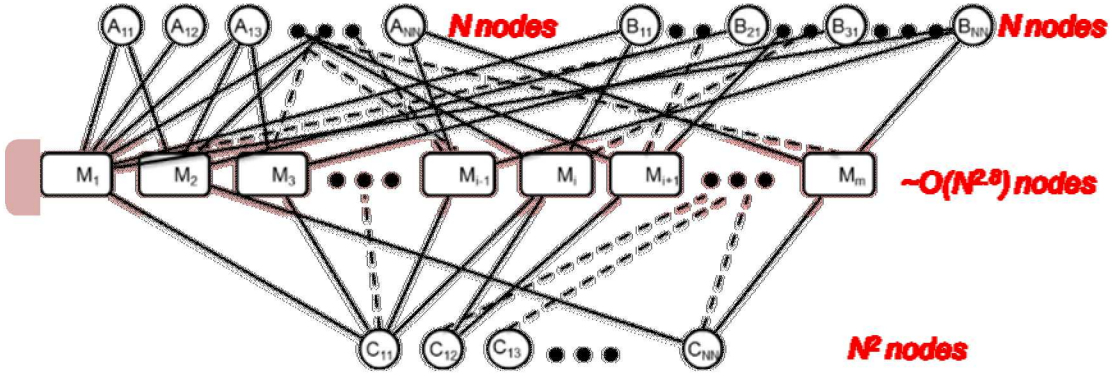
$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

"Neural" network for matrix multiplication

Standard:
8Ms, 4As → $O(N^3)$



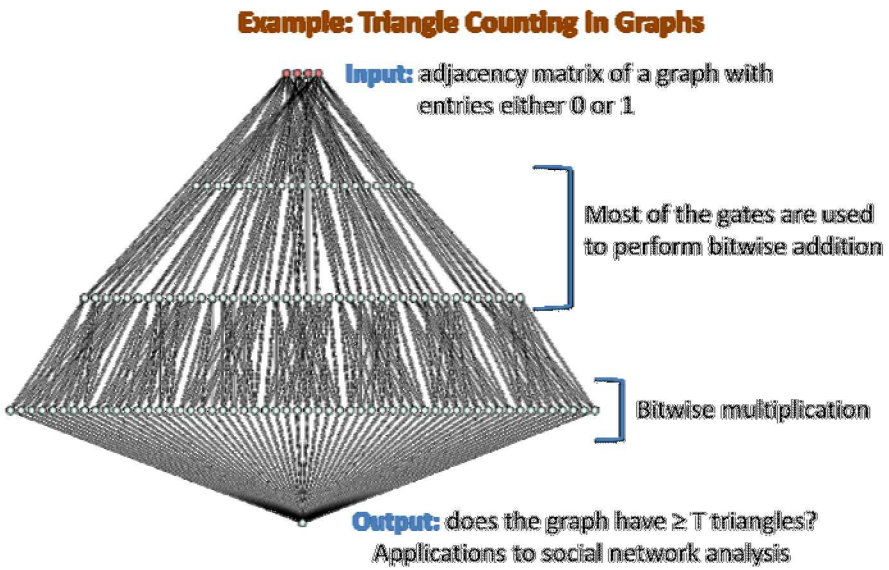
Strassen:
7Ms, 18A/Ss→
$O(N^{2+e})$

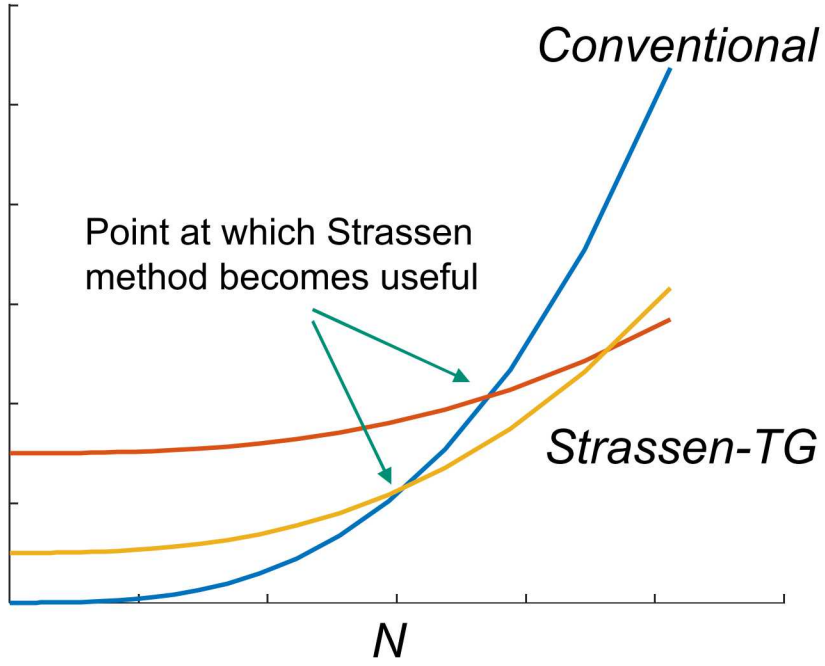


Strassen formulation of matrix multiply enables less than $O(N^3)$ neurons – resulting in less power consumption

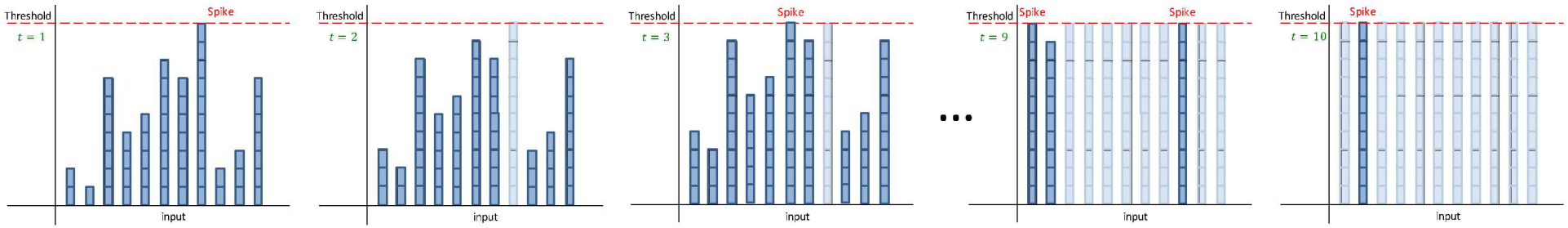
Strassen multiplication in neural hardware may show powerful advantages

	Depth	# Gates	Value of ε
Standard	3	$O(N^3)$	–
"Direct" Strassen	d	$O(N^{\omega+\varepsilon})$	1/d
Refined Strassen	d	$O(N^{\omega+\varepsilon})$	$O(1/c^d)$
Non-constant Depth	O(log log N)	$O(N^{\omega})$	–





Spiking Optimization Algorithms



time-step \ N	1	2	3	4	5	6	7	8	9	10	11
1								1			
2											
3						1					
4			1				1				1
5											
6					1						
7				1							
8										1	
9	1								1		
10		1									

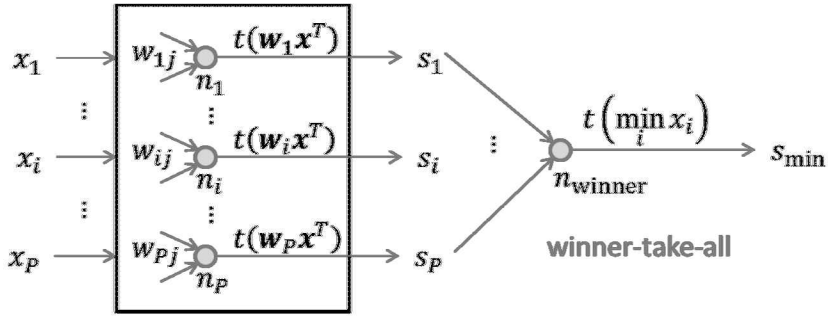
maximum

median

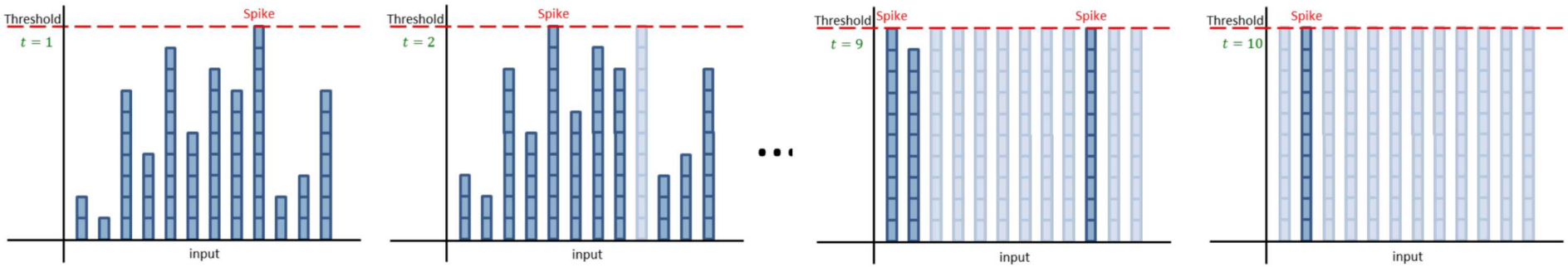
minimum

Sorted integers: {9, 7, 6, 6, 6, 5, 3, 2, 1, 1, 0}

Finding the min where $P \geq N$



Spiking Sort



Algorithm 1 spiking-sort

Input: set of integers, $\{x_1, x_2, \ldots, x_P\}$; k ▷ largest possible integer is $k-1$

Output: sparse bit matrix of spikes, $\mathbf{S}$

$w = 0$ ▷ initialize weight matrix to all zeros

for $j \leftarrow 1$ to P, in parallel **do**

  $w_{0j} = 1$ ▷ initialize bias weights

  $\theta_j = k$ ▷ set neuron threshold

  $u_j = x_j$ ▷ directly inject initial value as neuron potential

$x_0 = 1$ ▷ initialize bias input

$\mathbf{S} = 0$ ▷ initialize bit matrix to all zeros

for $j \leftarrow 1$ to P, in parallel **do**

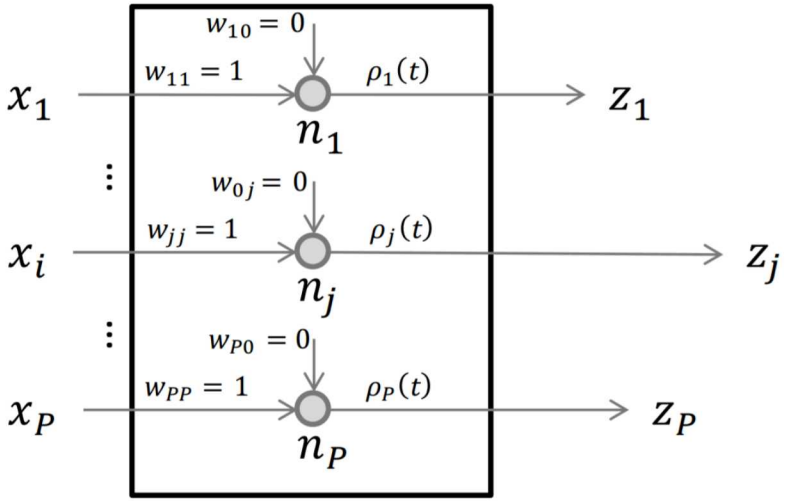
  for $\tau \leftarrow 1$ to k **do**

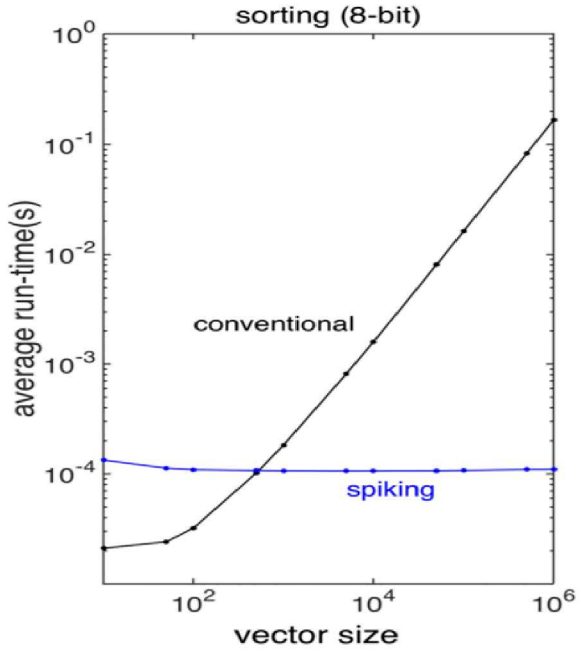
    $u_j = u_j + w_{0j}x_0$ ▷ neuron potential update (discretized LIF)

    if $u_j >= \theta_j$ **then** ▷ threshold check for spiking neuron

      $\mathbf{S}(\tau, j) = 1$

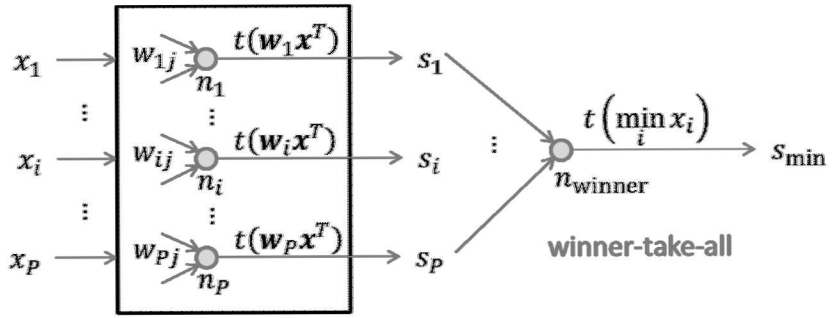
      $u_j = 0$ ▷ reset neuron potential after spike



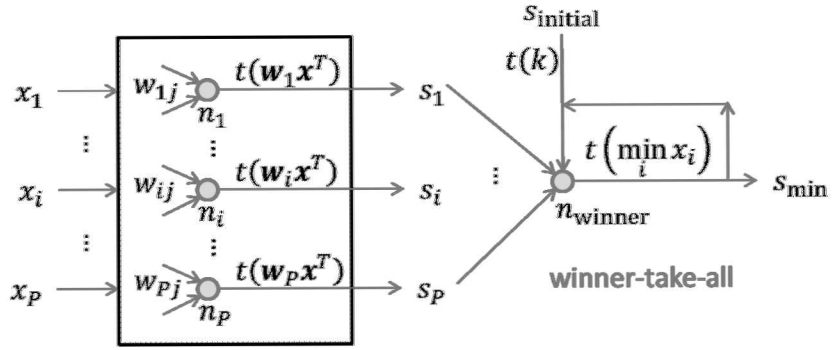


SpikeMin

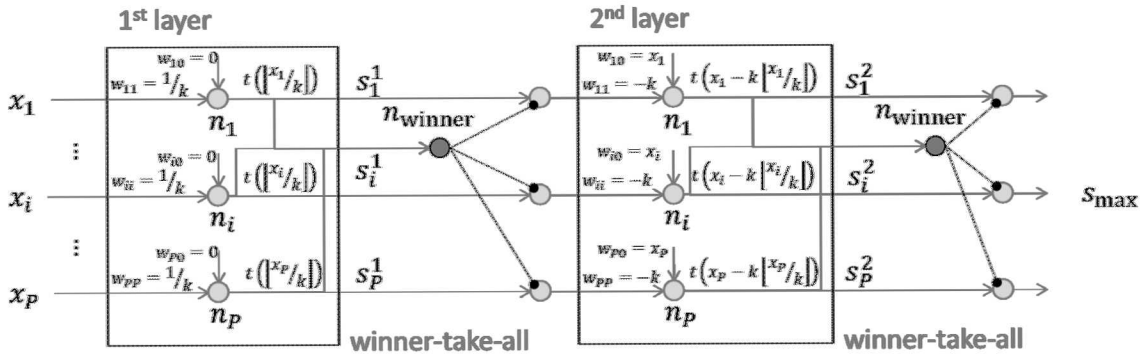
Finding the min where $P \geq N$



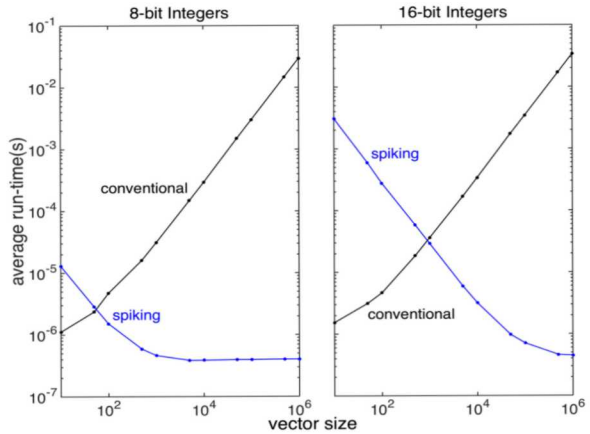
Finding the min where $P < N$



SpikeMax

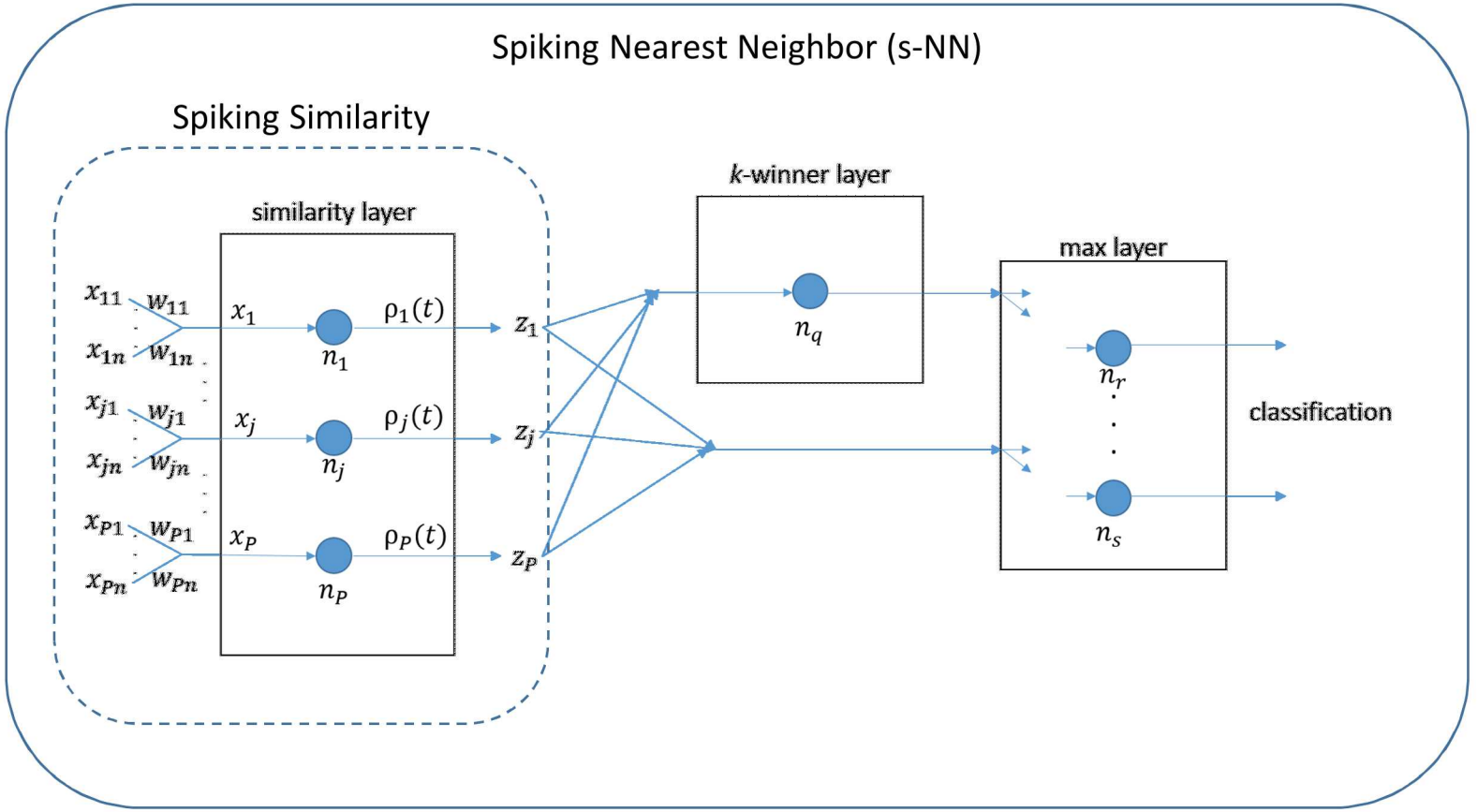


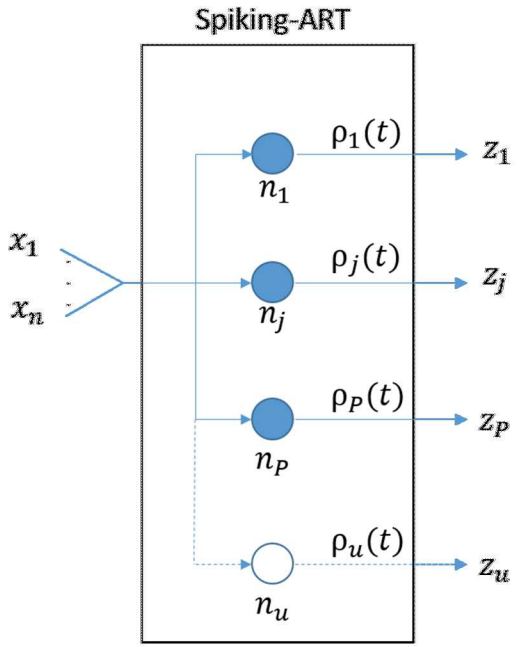
$$t\left(\max_i x_i\right) = ks_i^1 + s_i^2$$



Average runtimes for 10000 simulations of the spike-max neural spiking algorithm

Spiking Machine Learning Algorithms





Question

Can we extend this approach
to real-world scientific applications?

PURPOSE, GOALS AND APPROACH

Research Question

Can neuromorphic platforms be used for efficient and valid numerical computation critical to Sandia's mission?

*Emerging **low-power** computing platforms can potentially dramatically change how we approach our high-performance computing mission*

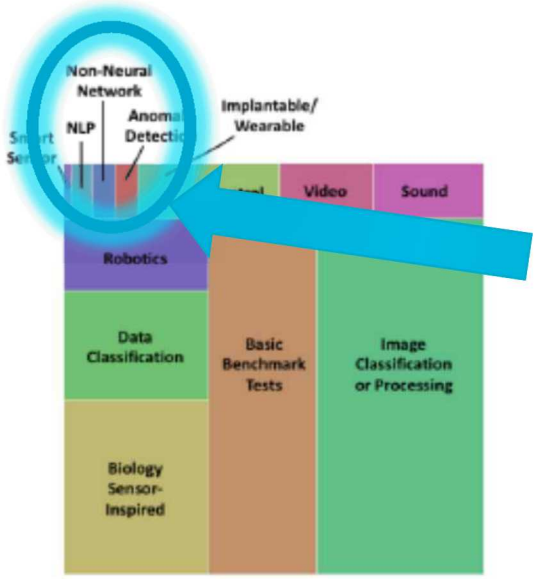


Fig. 13. Breakdown of applications to which neuromorphic systems have been applied. The size of the boxes corresponds to the number of works in which a neuromorphic system was developed for that application.

Only ~1% neuromorphic applications are not ANN-like

These aren't solving PDEs…

Katie Schuman, ORNL 2017

How is this different from previous research?

- ❑ Neuromorphic computing research has generally focused on AI applications
- ❑ Similarly, primary emphasis on application to scientific computing is on machine learning

Spiking random walk algorithms – diffusion is a pure scientific computing task for spiking algorithms

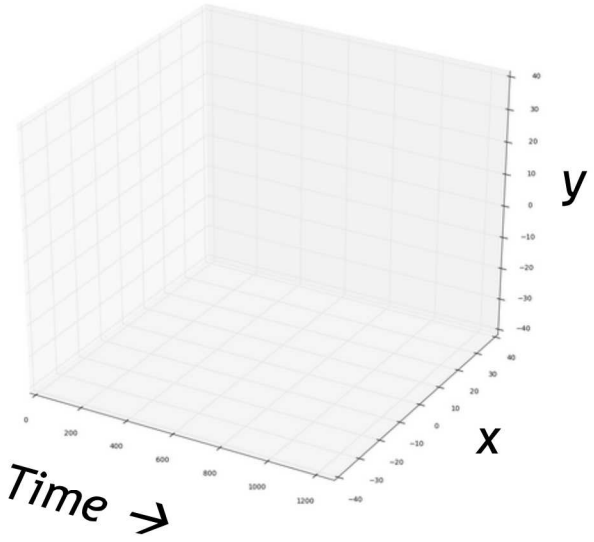
Diffusion can be modeled either as a deterministic PDE or a stochastic process

- For an initial distribution of particles, P_0, what is distribution of particles at time *t*?
- Diffusion can be modeled as the PDE

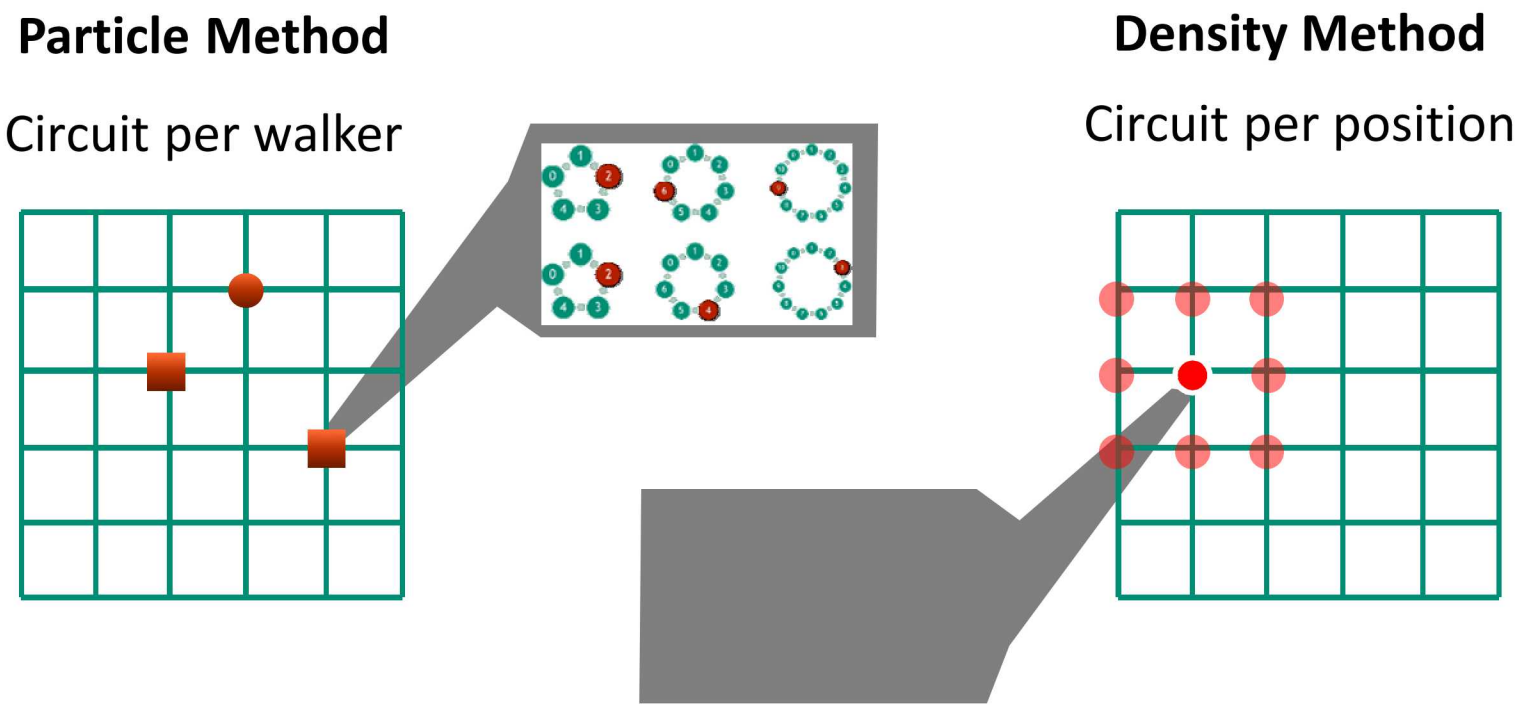
$$\frac{\partial C(x,t)}{\partial t} = D\frac{\partial^2 C(x,t)}{\partial x^2}$$

with B.C. + I.C.

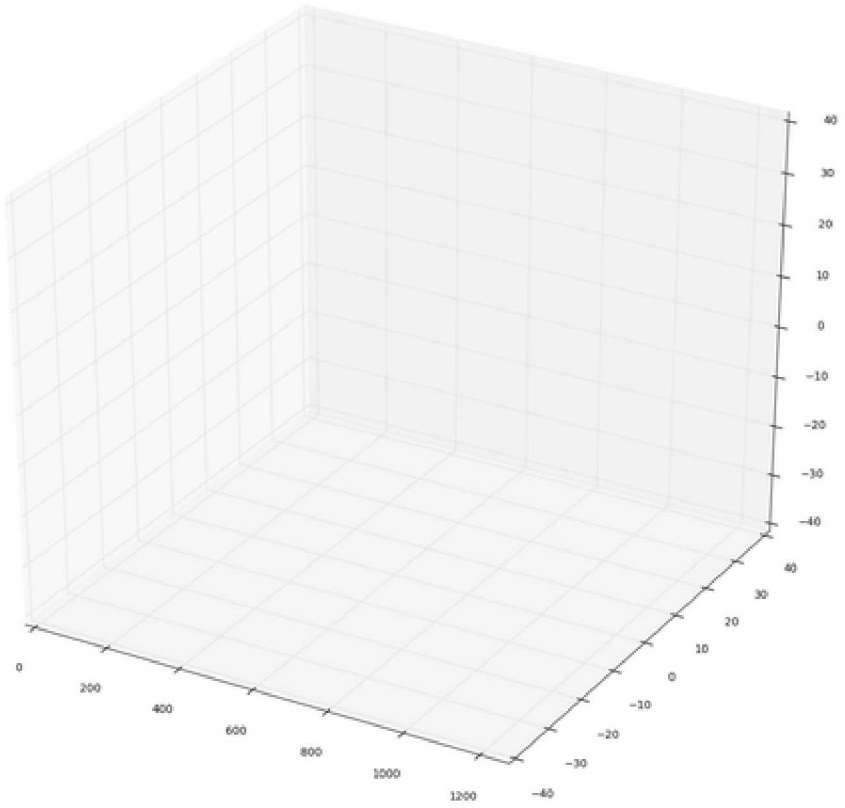
- Stochastic process implements many random walkers to statistically approximate a solution
  - Mean position of *N* walkers approaches expected mean of deterministic solution at rate of 1/sqrt(N)

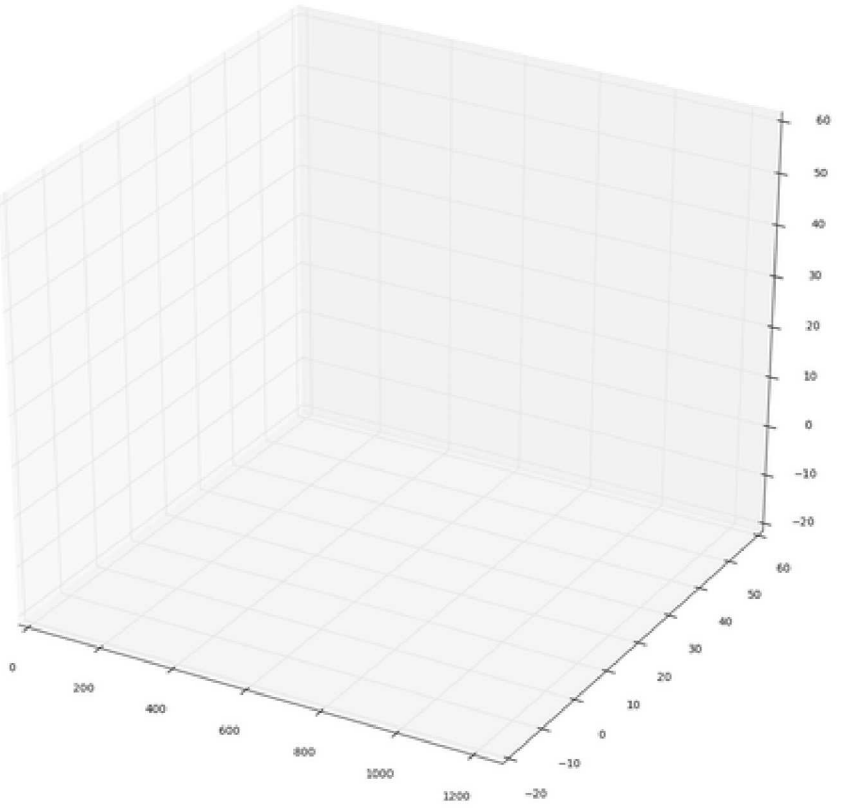


Two spiking algorithms for random walk with different costs and benefits



Neural random walkers evolve as expected

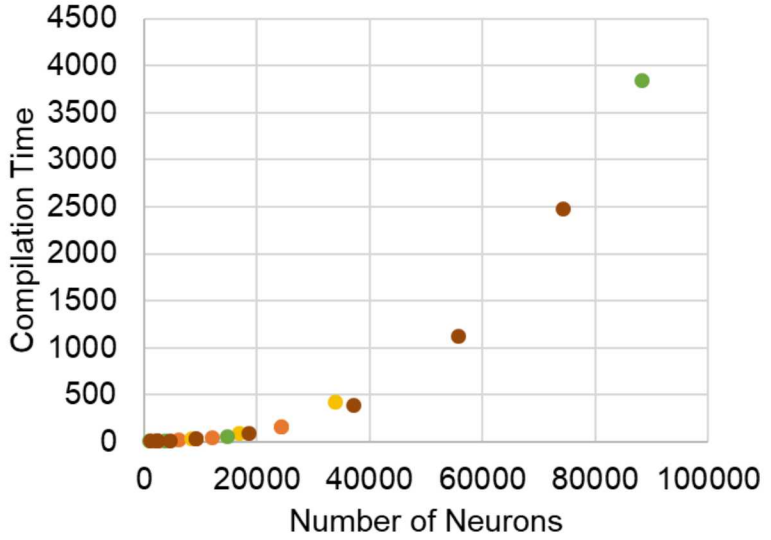


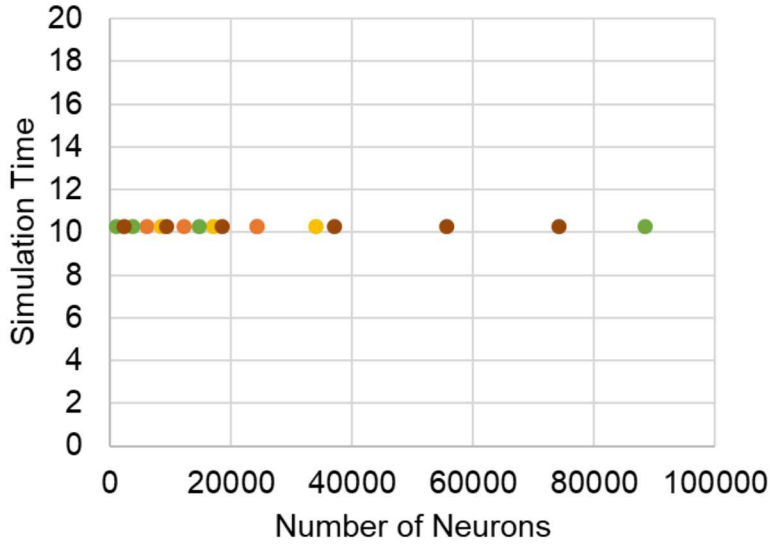


SpiNNaker 48 chip test board
250 walkers, 1250 time steps
with equal probability of moving (left) or strong bias (right)

Results of mapping particle-based RW to SpiNNaker

- Graph partitioning and compilation is currently prohibitive
  - Even though more walkers is simple repetition of circuit, software stack cannot readily account for it
  - In practice, mapping likely only needs to be performed once, and it is independent of simulation time (not shown)
- Simulation on SpiNNaker is truly constant time – more walkers requires more neurons and more cores, but is embarrassingly parallel
  - Simulation IS throttled back to mitigate implications of communication bottlenecks and is running quite slow

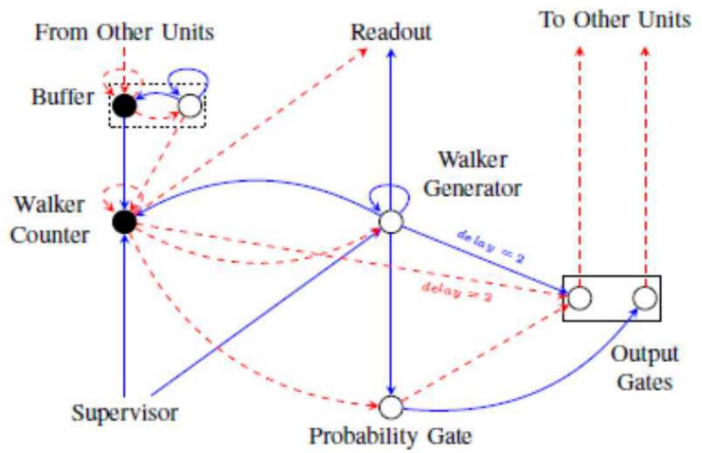




Density model repeats basic circuit at every vertex of mesh

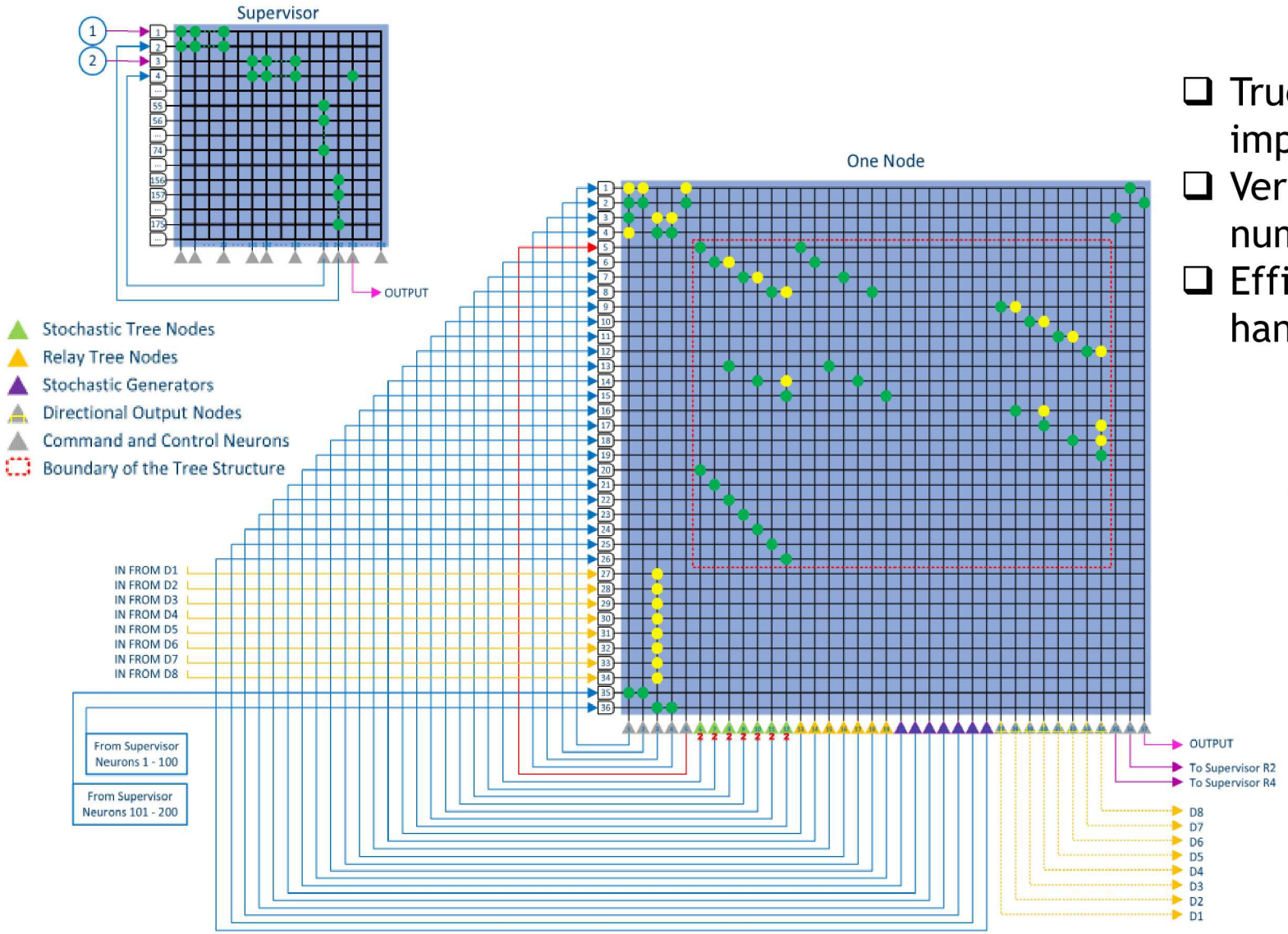
Each vertex encodes density of particles in the internal potential of certain nodes

Each time step "hands off" particles to connected vertices according to probabilistic maps

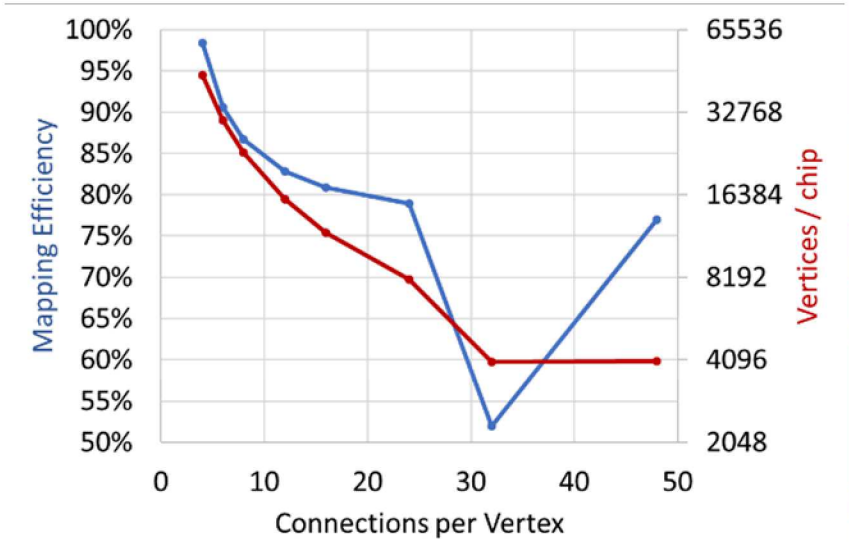


Measure	Cost (for k locations, simulating N walkers; 1-D case)
Walker memory	O(1)
Connection memory	O(k)
Total neurons	O(k)
Time per physical timestep	O(max(ρ_i)), where ρ_i is the density of walkers at each location
Position energy per timestep	O(N)
Update energy per timestep	O(N)

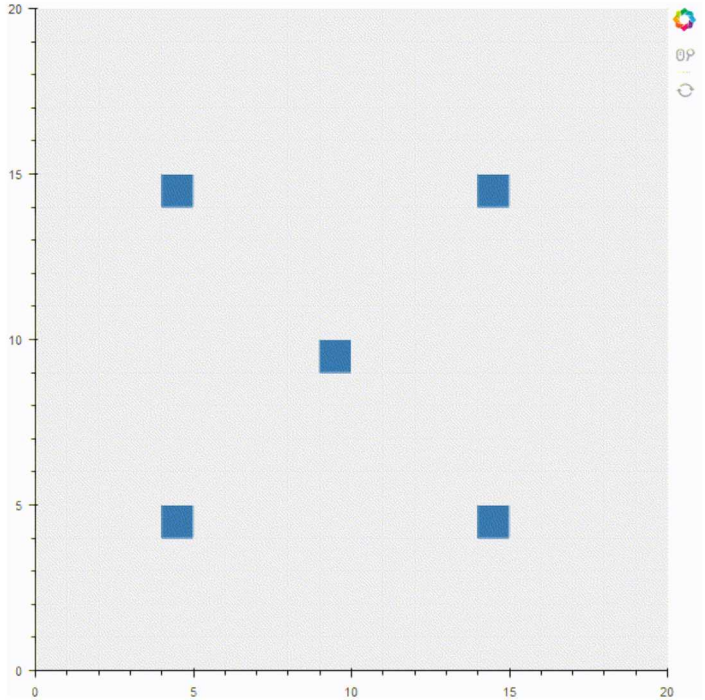
TrueNorth Corelet for ND-Random Walk Vertex



- TrueNorth is quite efficient at implementing density RW algorithm
- Vertex size scales linearly with number of outputs
- Efficiency can increase with by-hand mapping

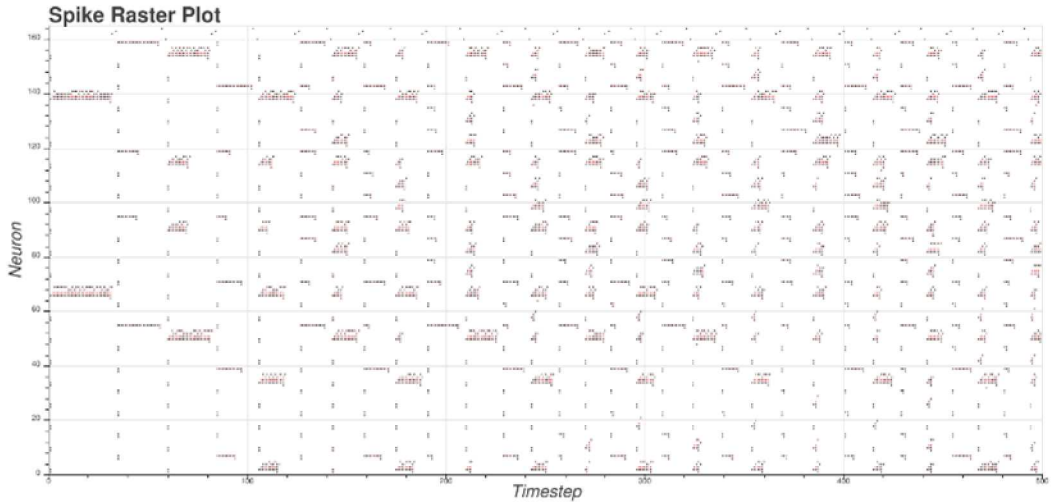


Speed of simulation depends on density distribution of walkers



Simulation on IBM TrueNorth hardware

- 20x20 grid
- 30 walkers initialized at 5 points
- 2000 timepoints

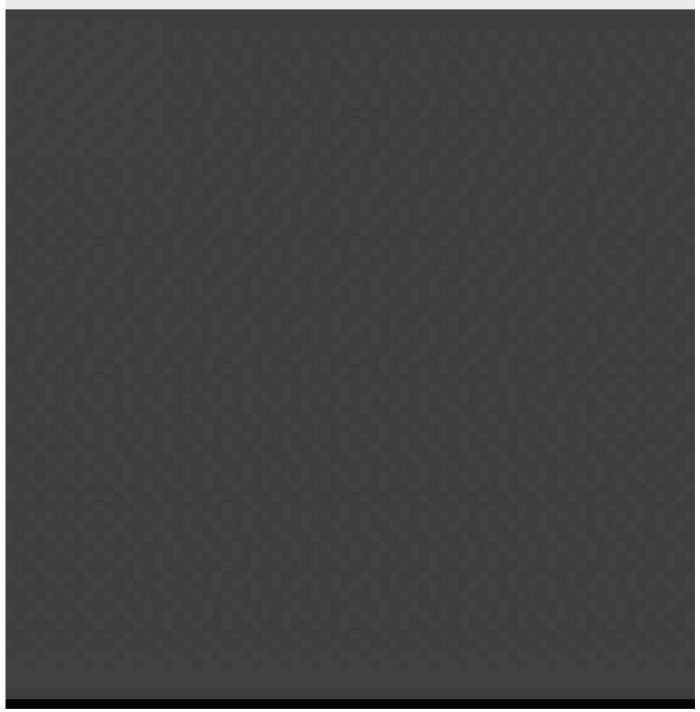


Density model can model arbitrary graphs, enabling complex behaviors

- This example uses a mesh with one-way edges that permit walkers to move into colored parts of the image, but not out

- Scale of simulation
  - 1600 vertices
  - 8000 walkers
  - 1000 timesteps
  - 19205 neurons
  - 60802 synapses

- This runs in ~1 second on TrueNorth



Density model can model arbitrary graphs, enabling complex behaviors

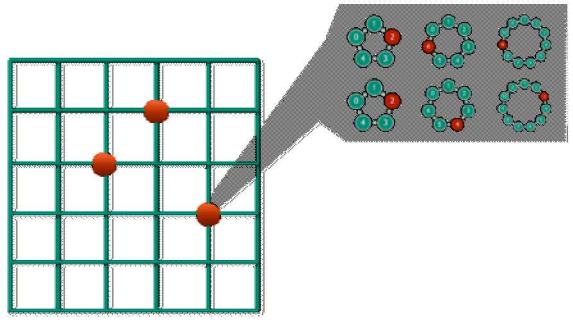
- Scaled up to 16 chips

- 640,000 vertex points
- 518,400,134 spike events
- 63,378 cores
- 100s for a 10s simulation

Each method offers unique advantages

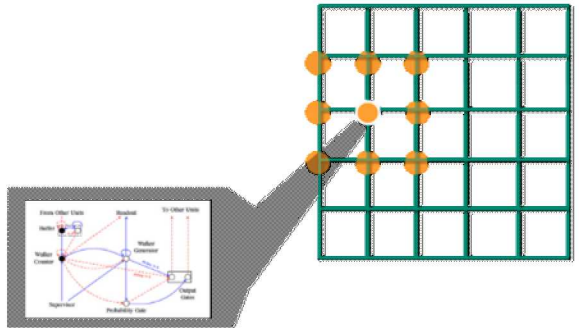
Particle method

- Path dependent behavior is readily available
- Communication is entirely local within particles (embarrassingly parallel)
- With unlimited neurons, can run in constant time
- **Ideal for sparse particles in large spaces**

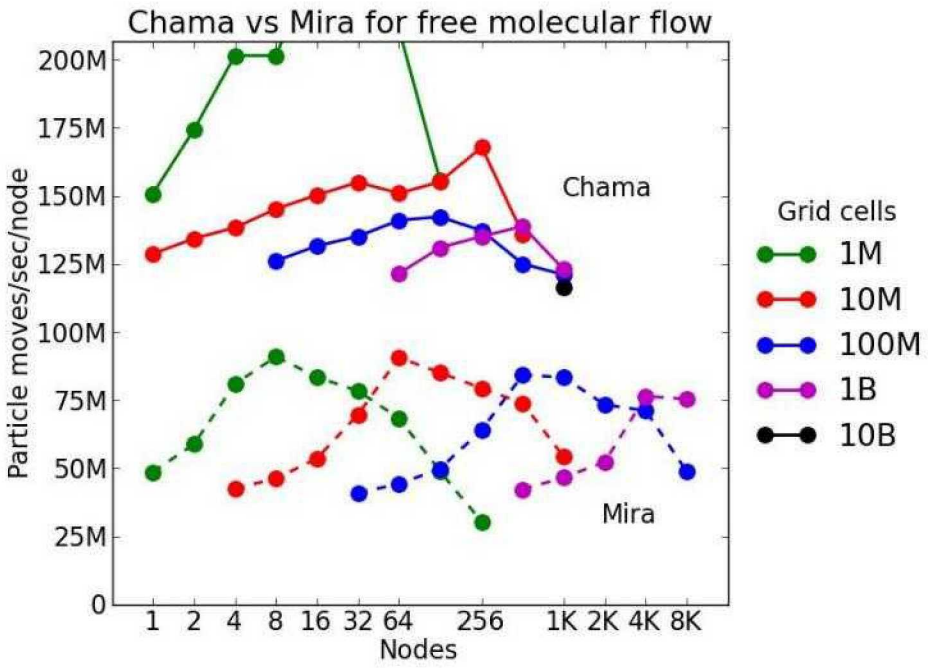


Density method

- Densities are readily available at all times
- Non-local or other complex graphs can easily be implemented
- With limited neurons, can tradeoff statistical approximation (i.e., number of walkers) with longer or shorter simulations
- **Ideal for dense particles in small spaces**



So can we say anything about mapping codes to a supercomputer?

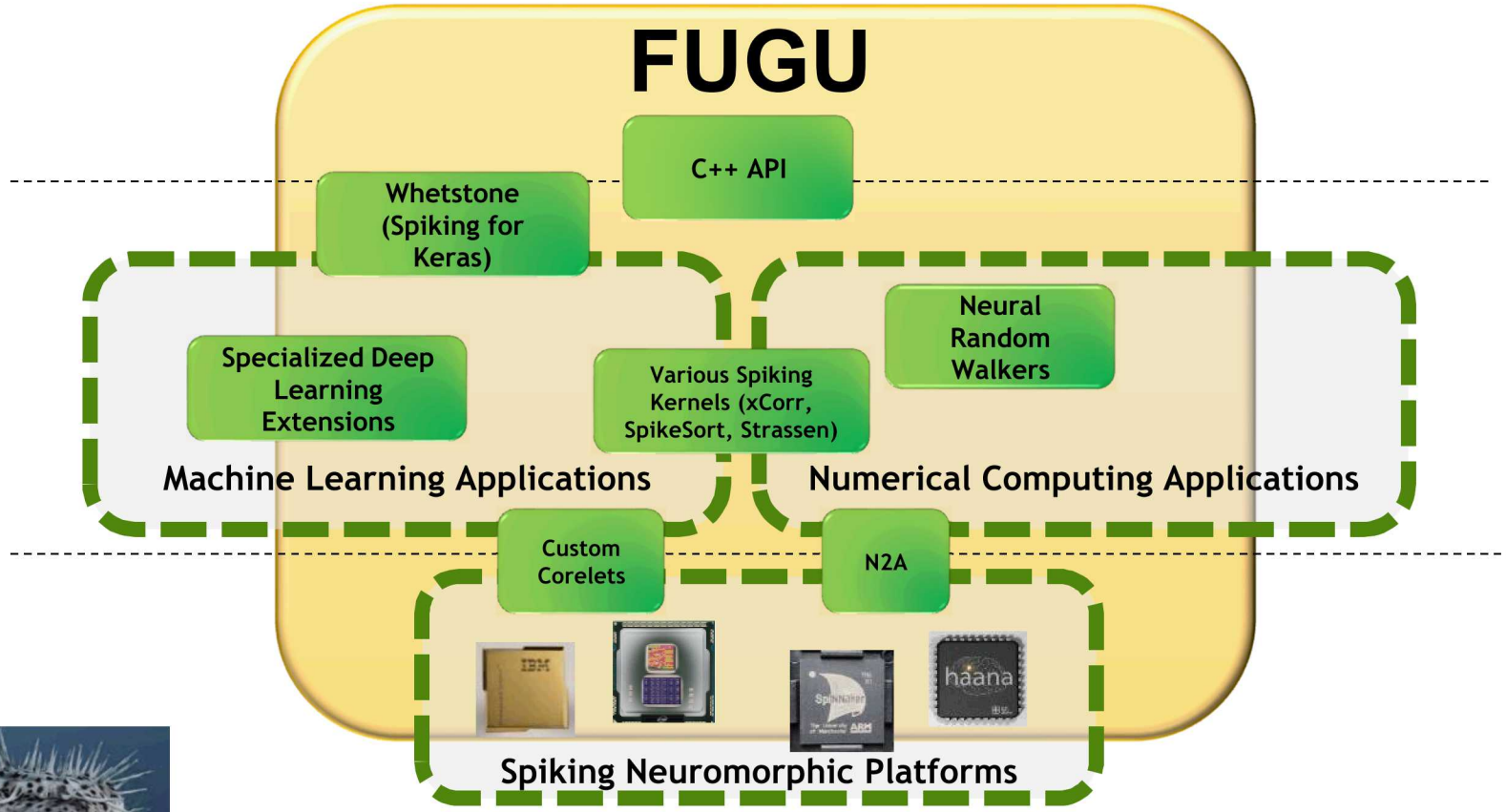


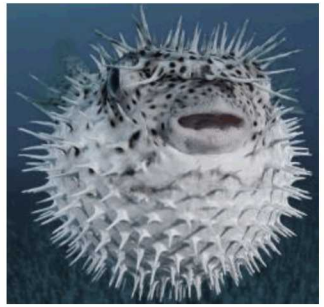
http://sparta.sandia.gov/bench.html

- Free molecular flow is a benchmark problem for HPC systems (see SPARTA Direct Simulation Monte Carlo benchmark ←)
  - Each Mira node ~80W
- Where would neural hardware end up?
  - *Most neural hardware is throttled back in speed (kHz vs GHz) to save power*
  - *Most HPC systems are tuned for speed, not power*
- 1M grid cells in 3D (6 possible movements) would require ~34 TrueNorth chips.
  - Ignoring chip-chip delays and neuron overhead, **could run up to 1,000M moves per second at ~4W**

- 10M walkers on SpiNNaker would require ~250,000 chips without custom neuron types

Emerging framework *Fugu* for neural codes

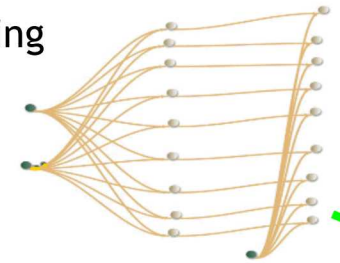




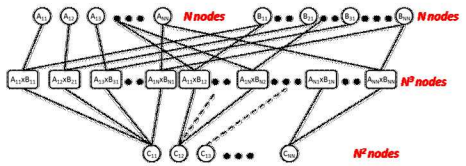
Fugu = pufferfish (why? Pufferfish have spikes…)

Impacting Broad Areas of Computation

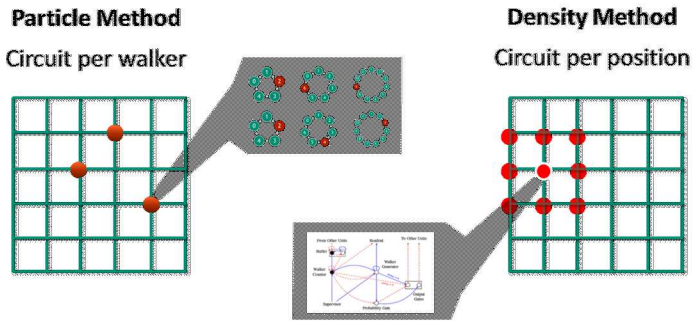
Pattern Matching



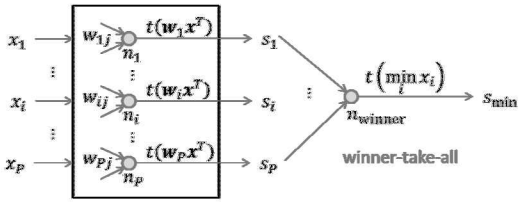
Linear Algebra



Scientific Computing



Optimizations



SNN

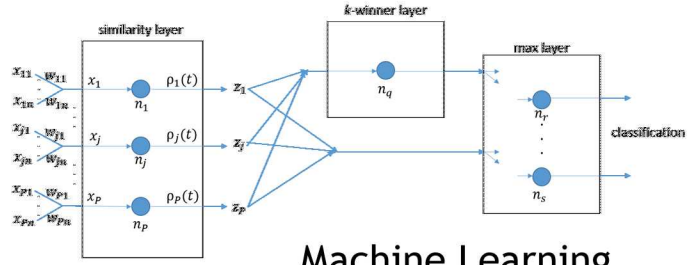
Neural Algorithms

NN

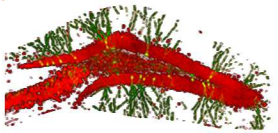
ANN

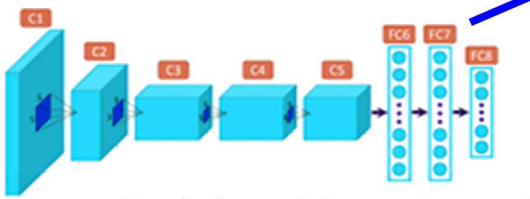
WHETSTONE

Machine Learning

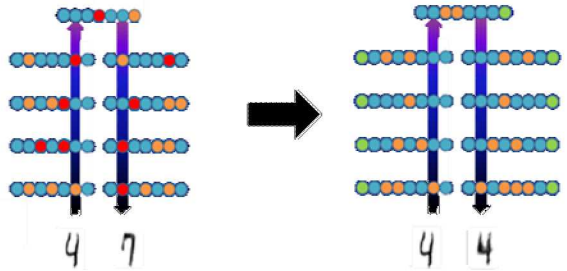


Intelligent Storage





Context Modulated Deep Learning



Adaptive Deep Learning

SNL Loihi Plans

Neuromorphic Random Walks on Loihi

- Implement and assess spiking neural algorithms for PDE computation
- Exploring the scalability and computational advantages Loihi affords has the potential to impact large scale scientific computing

Accelerator Enabled Proximal Computation

- We seek to explore which architectural features are most impactful in different computational paradigms – signal processing paradigms & data sources
- The flexibility of neurons and reconfigurable connectivity of the Loihi architecture will uniquely enable this landscape exploration

Conclusion

Sandia National Laboratories is pursuing neural-inspired computing as a transformative approach to computation

- Research driven by mission application
- Many opportunities at intersection of neuroscience, math, and computing
- Even loosely brain-inspired concepts have potential to be very impactful on computing applications

Thank you!