AN ALGORITHMIC THEORY OF BRAIN NETWORKS

Nancy Lynch, MIT EECS

Simons Institute

April 17, 2018

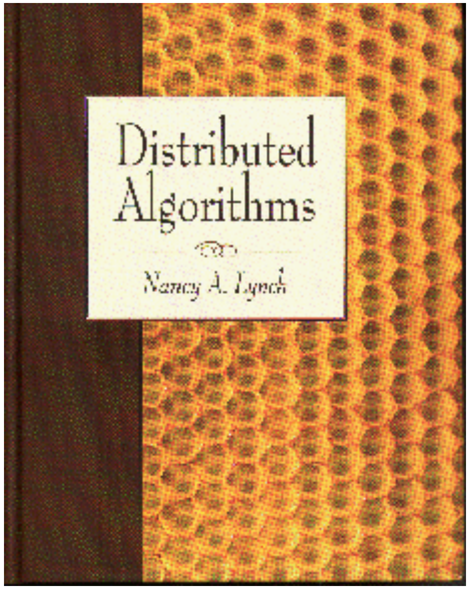
Joint work with

Cameron Musco (MIT) and

Merav Parter (Weizmann)

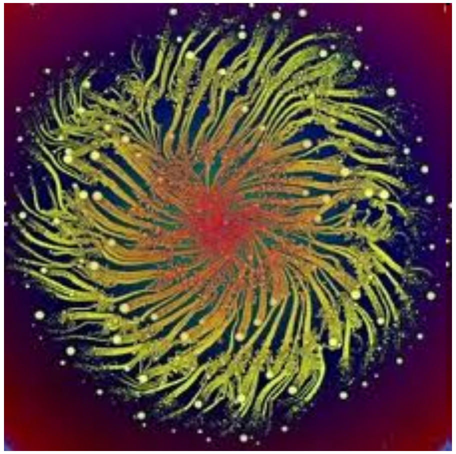
Distributed Algorithms

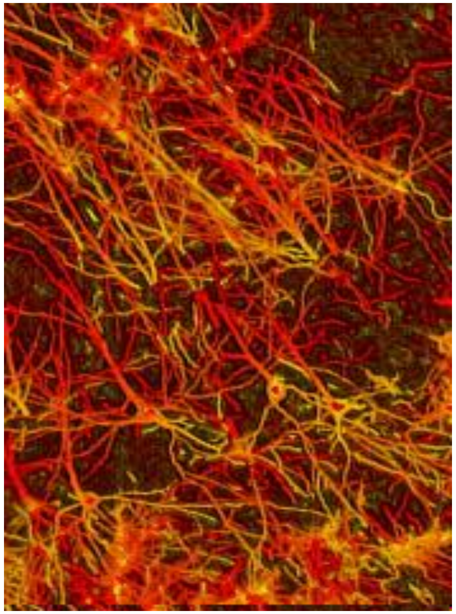
- My Theory of Distributed Systems research group works on distributed algorithms, concerned with communication in wired and wireless networks, network organization, distributed data management, consensus,…
- Also general concurrent systems theory, concerned with modeling systems and proving general theorems about how they can be constructed and proved correct.
- Many kinds of biological systems behave like distributed algorithms…

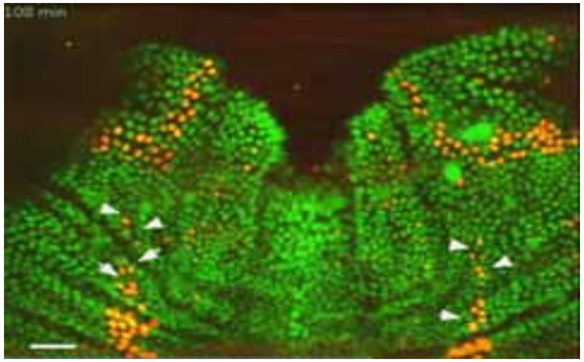


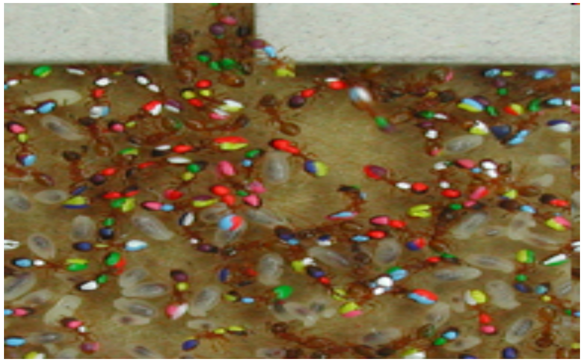
Biological Distributed Algorithms

- Many kinds of biological systems behave like distributed algorithms, e.g.:
  - Cells in developing organisms organize themselves into meaningful patterns.
  - Insect colonies cooperate to solve problems of cooperative exploration, task allocation, consensus.
  - Neurons cooperate to implement focus, learning, memory.



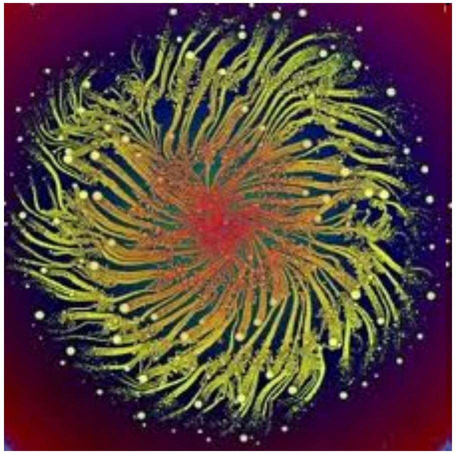


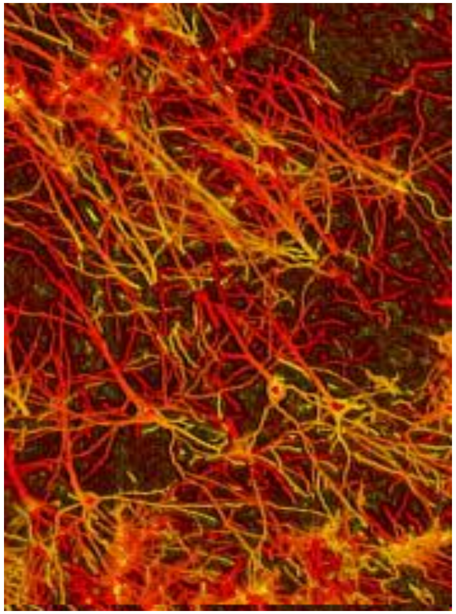


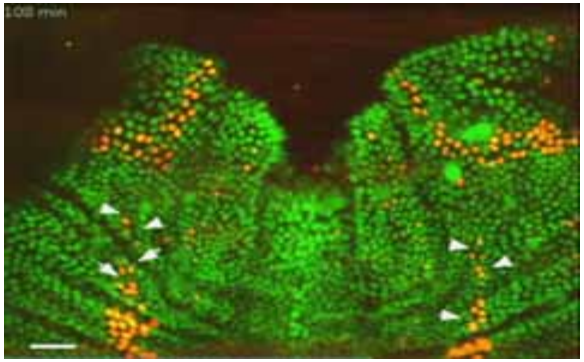


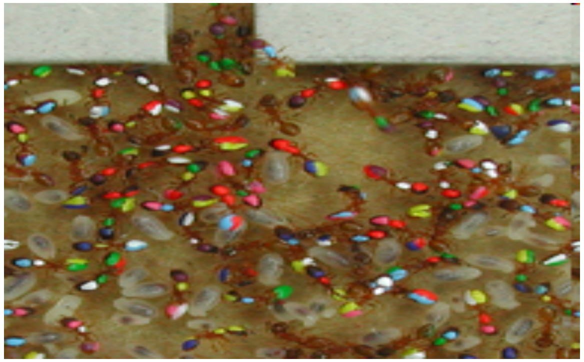
Biological Distributed Algorithms

- They have special characteristics:
  - Use simple chemical “messages”.
  - Components have simple “state”, follow simple rules.
  - Flexible, robust, adaptive.

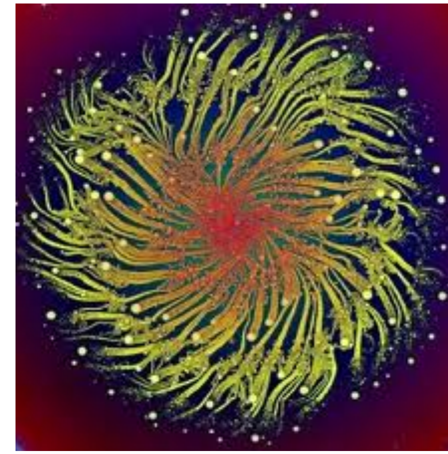






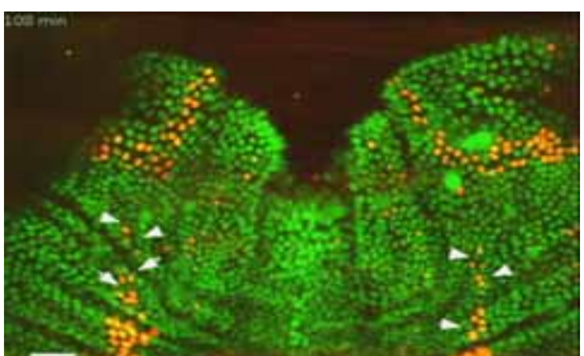


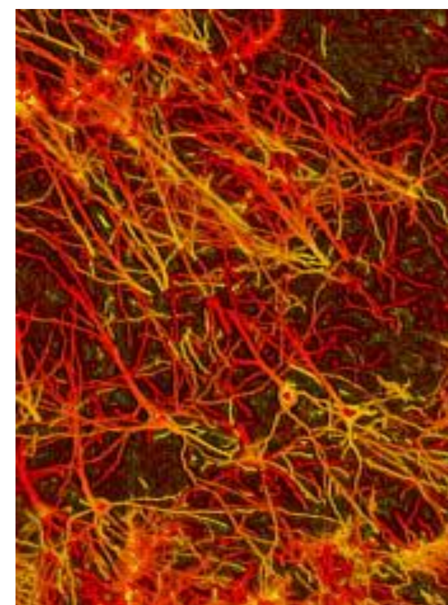
Biological Distributed Algorithms



Q: How can distributed algorithms help in understanding the behavior of biological systems?

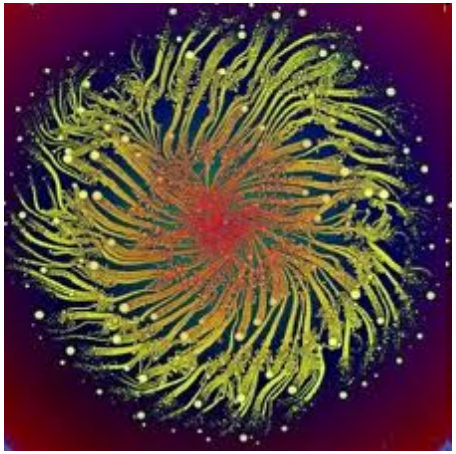
Q: How can understanding biological systems help in building better distributed network algorithms?

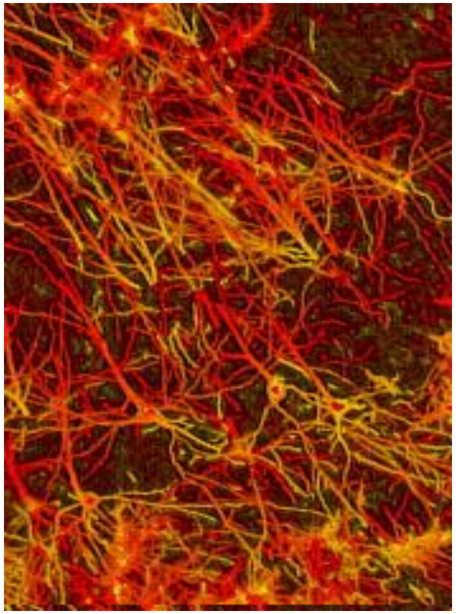


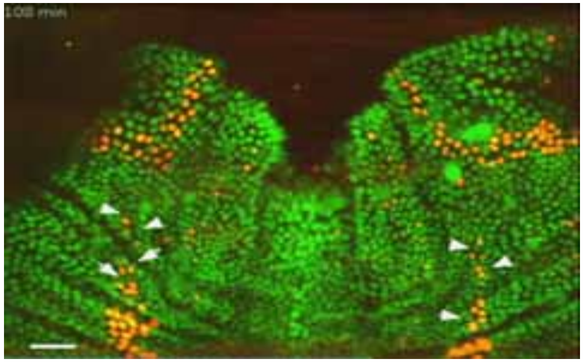


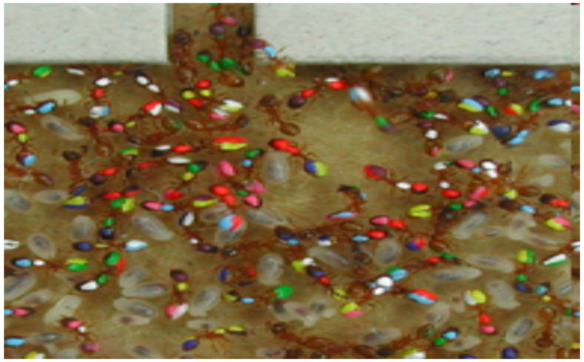
Some of our Results

- Insect colony behavior:
  - Cooperative foraging (resource discovery)
  - Task allocation
  - Nest relocation (consensus)
  - Density estimation
- May lead to the discovery of new kinds of distributed algorithms: flexible, robust, adaptive.



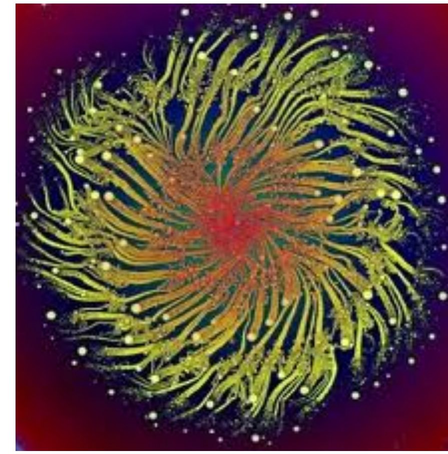


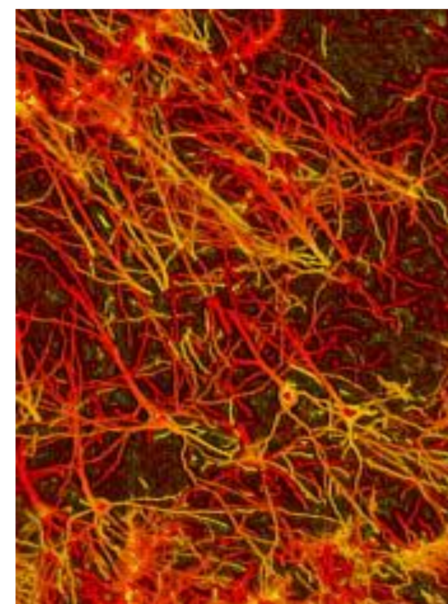


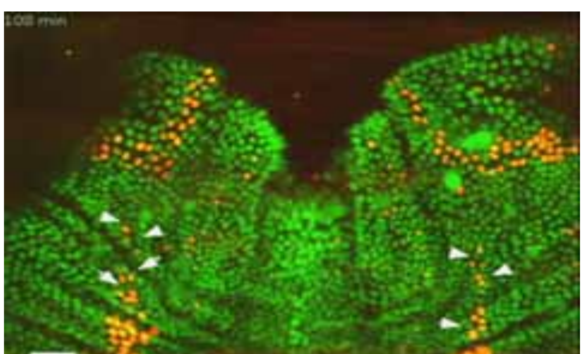


Some of our Results

- Brain network operation
  - Winner-Take-All (leader election)
  - Similarity Detection
  - Neural coding

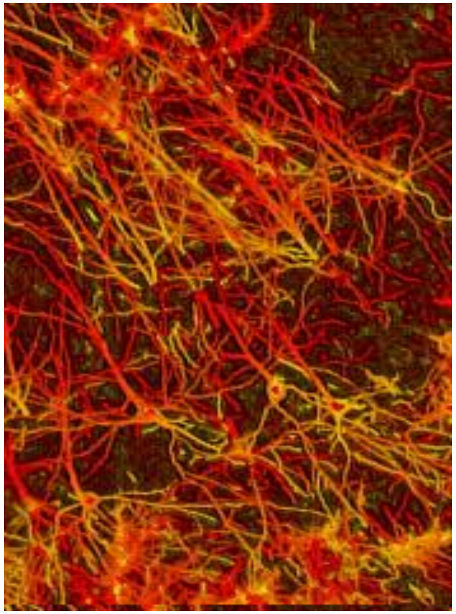






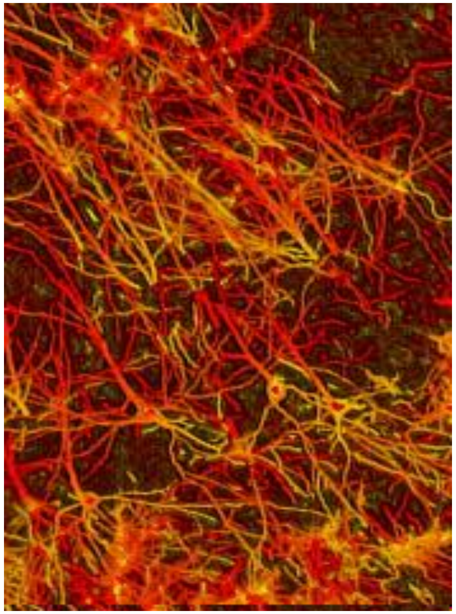
Our Work on Brain Algorithms

- Goal: Understand how computation is performed in biological neural networks, in terms of distributed algorithms.
- Biological features we consider: Spiking neurons, noisy firing thresholds, excitation and inhibition, restricted connectivity, synapse weights, synchronization.
- So far, we have focused on fixed, designed networks, rather than on how they are learned.
- Sample problems: Select one neuron from a set of firing neurons, test similarity of input patterns, neural coding, data compression,,…
- Basic abstract computational primitives, rather than complex real-world problems.



Guiding Questions

- How do the various biologically-inspired model features affect the solvability of particular problems? The costs of solving them? The design of algorithms?
- Is there interesting new theory beyond that for other well-studied models of computation, such as deterministic threshold circuits, Boltzmann machines, distributed graph networks?
- Can this theory say anything interesting about computation in real neural networks?
- E.g., clarify the role of noise and randomness; clarify the role of inhibition and excitation; identify recurring patterns,…
- Our starting point: Work by Maass et al. on theory of Spiking Neural Networks.

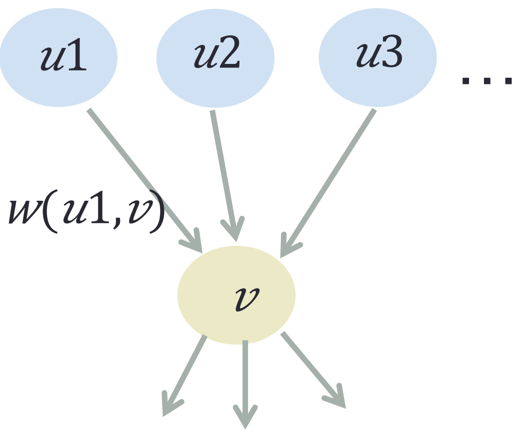


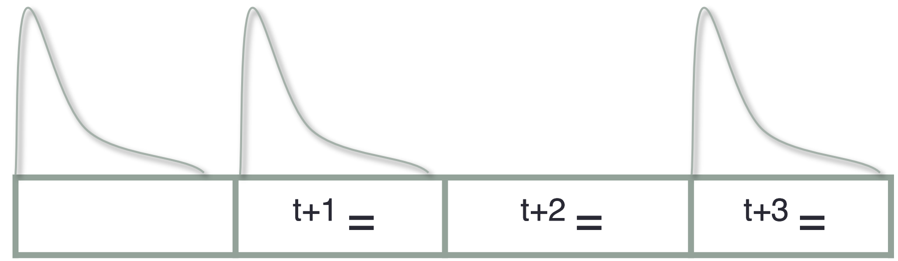
This talk

1. Introduction √
2. Our model: Stochastic Spiking Neural Networks
3. Winner-Take-All algorithms and lower bounds
4. Similarity Testing and Indexing
5. Composing Stochastic Spiking Neural Networks
6. Discussion

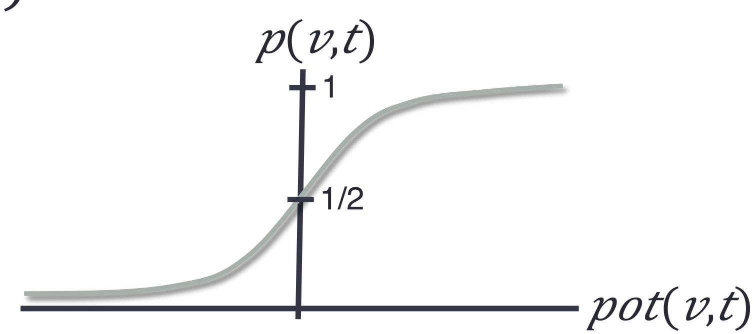
2. Stochastic Spiking Neural Networks

- $v^t = 1$ if and only if neuron v spikes at time t.





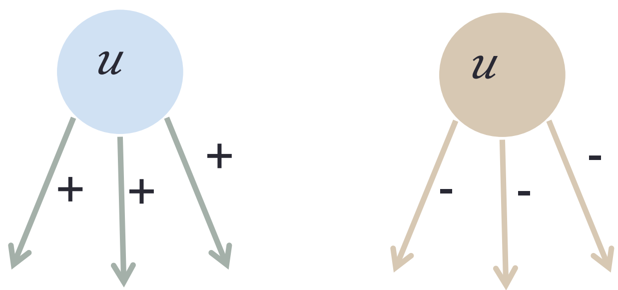
- $pot(v,t) = \Sigma_u \; u^{t-1} \; w(u,v) - b(v)$



- $\Pr[v^t = 1] = 1/(1 + e^{-pot(v,t)})$

Stochastic Spiking Neural Networks

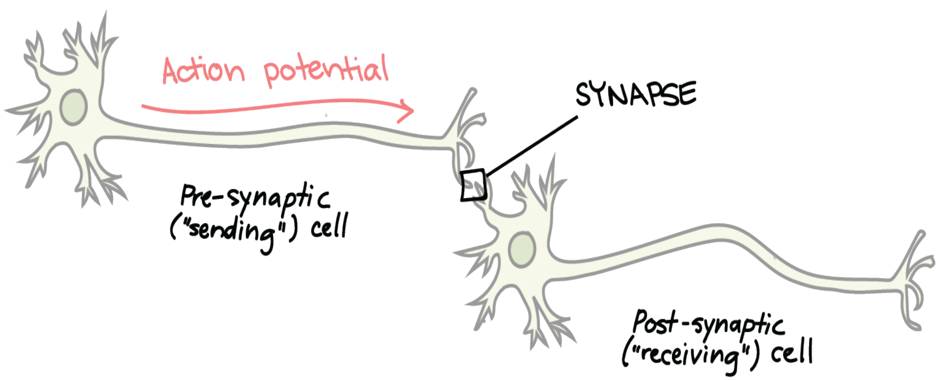
- All neurons are strictly inhibitory or strictly excitatory, i.e., $w(u,v) \geq 0$ for all v or $w(u,v) \leq 0$ for all v.

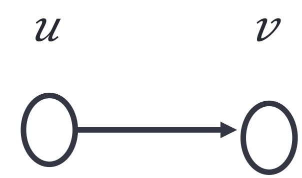


- We ignore many other biological features: Refractory period, spike propagation delay, history, noise on synapses,…
- Some can be simulated in our model.

Neural Network Model

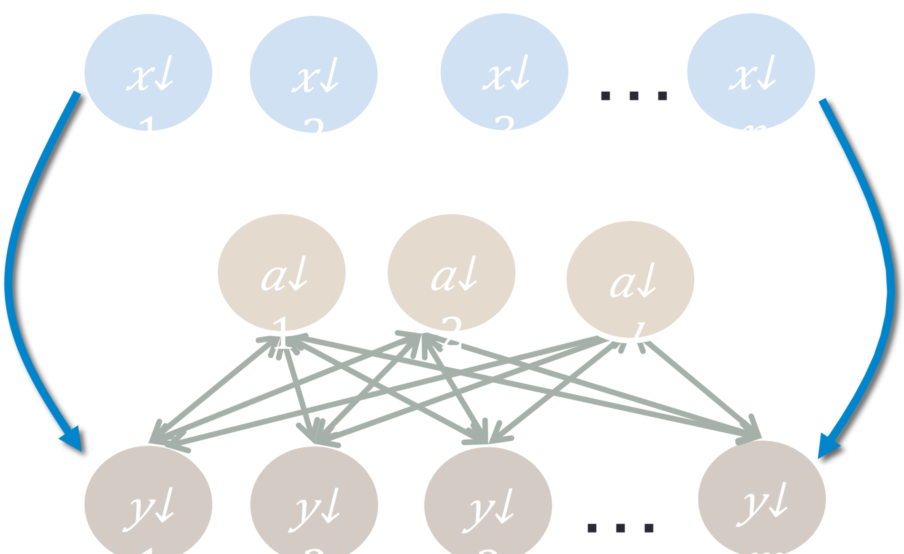
- A weighted directed graph, nodes represent neurons, edges represent synapses, weights indicate synaptic strength.
- Regard $weight = 0$ as absence of edge, $weight > 0$ as excitatory, $weight < 0$ as inhibitory.





Neural Network Model

- Neurons are either input neurons X, output neurons Y, or auxiliary neurons A.
- Input and output neurons must be excitatory.
- Auxiliary neurons may be either excitatory or inhibitory.



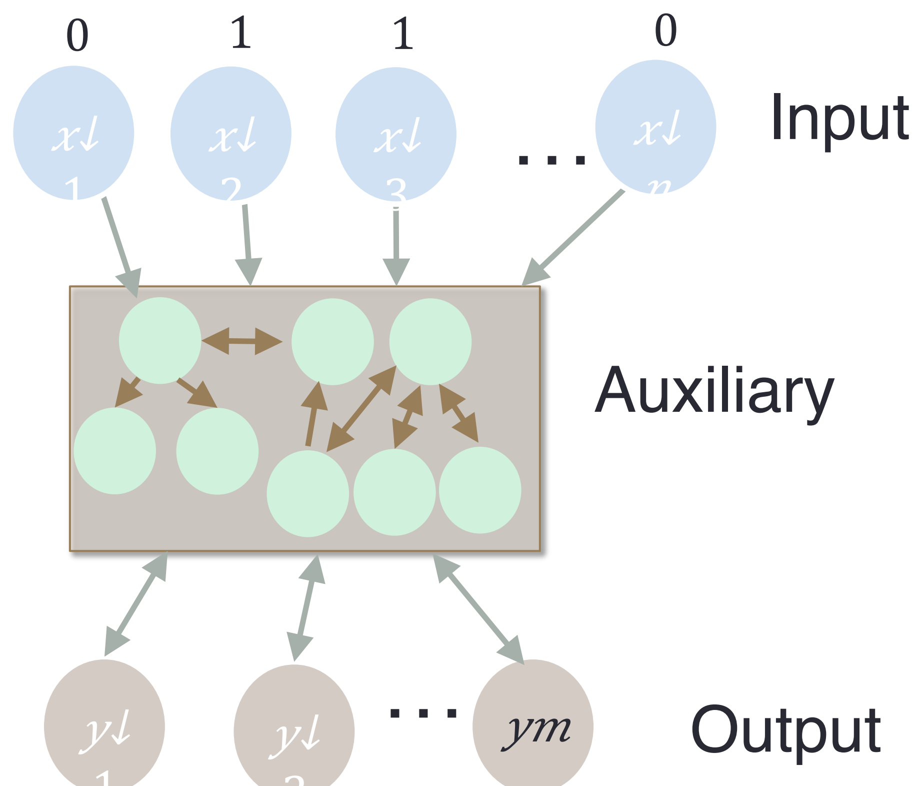
Network Dynamics

- Configuration C: Assigns a firing state, 0 or 1, to each neuron, where $C(u) = 1$ means it's firing and $= 0$ means it's not.

- Execution $\alpha = C^0, C^1, C^2, \ldots$, a sequence of configurations.
- $u^t = C^t(u)$ denotes the firing state of neuron u at time t.
- Input firing patterns may be arbitrary.
- Initial firing patterns for non-input (auxiliary and output) neurons are part of the network definition.
- For every infinite input execution, the network produces a probability distribution on infinite executions, by applying the stochastic firing dynamics for all non-input neurons at all rounds.

Computational Problems in our Model

- n input neurons X, each always firing or always not firing.
- m output neurons Y.
- Target function f: $\{0,1\}^n \rightarrow \{0,1\}^m$ (possibly multi-valued).

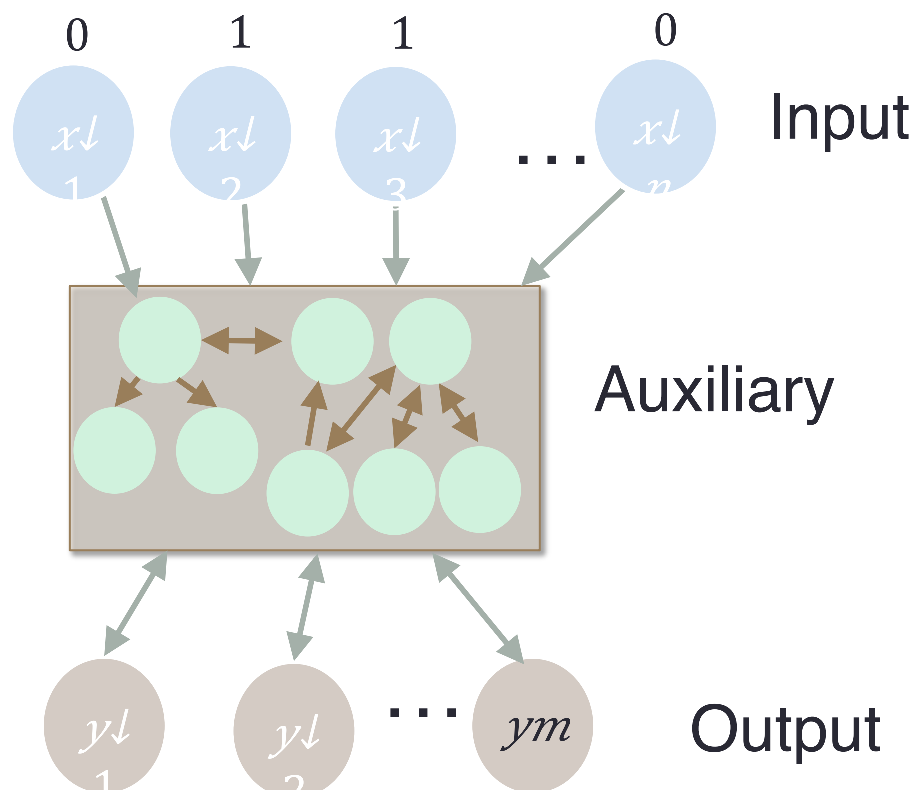


- Goal: Design a compact network that rapidly converges to some output firing pattern $Y \in f(X)$, with high probability.

Computational Problems in our Model

- n input neurons X, each always firing or always not firing.
- m output neurons Y.
- Target function $f: \{0,1\}^n \to \{0,1\}^m$ (possibly multi-valued).

- Complexity Measures:
  - Static measures, like number of neurons, number of auxiliary neurons, maximum weight used,...
  - Time (number of rounds to convergence).

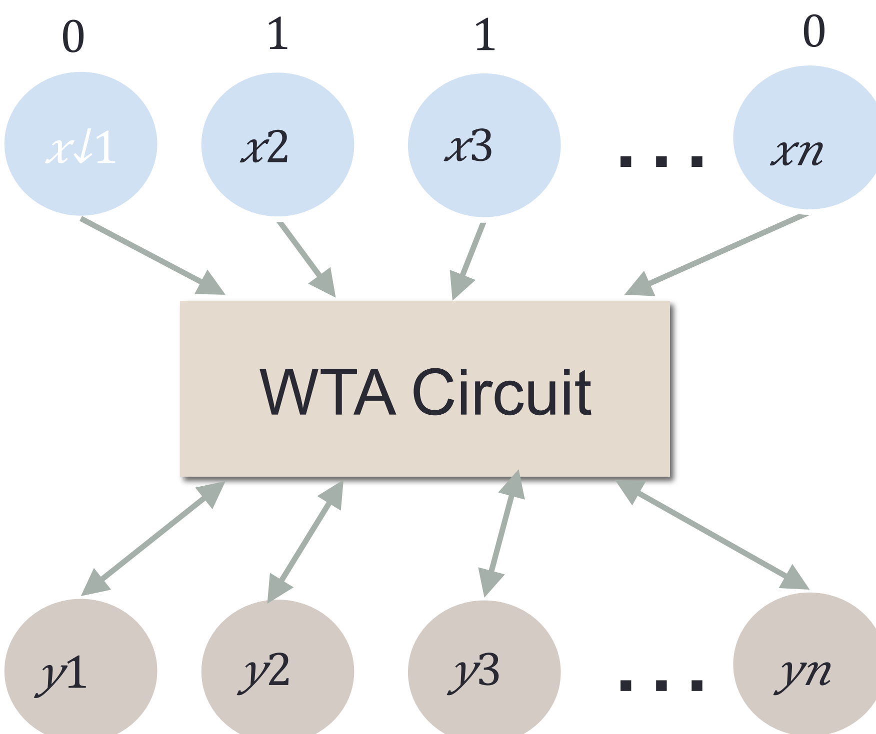


This talk

1. Introduction √
2. Stochastic SNNs √
3. Winner-Take-All algorithms and lower bounds
4. Similarity Testing and Indexing
5. Composing Stochastic SNNs
6. Discussion

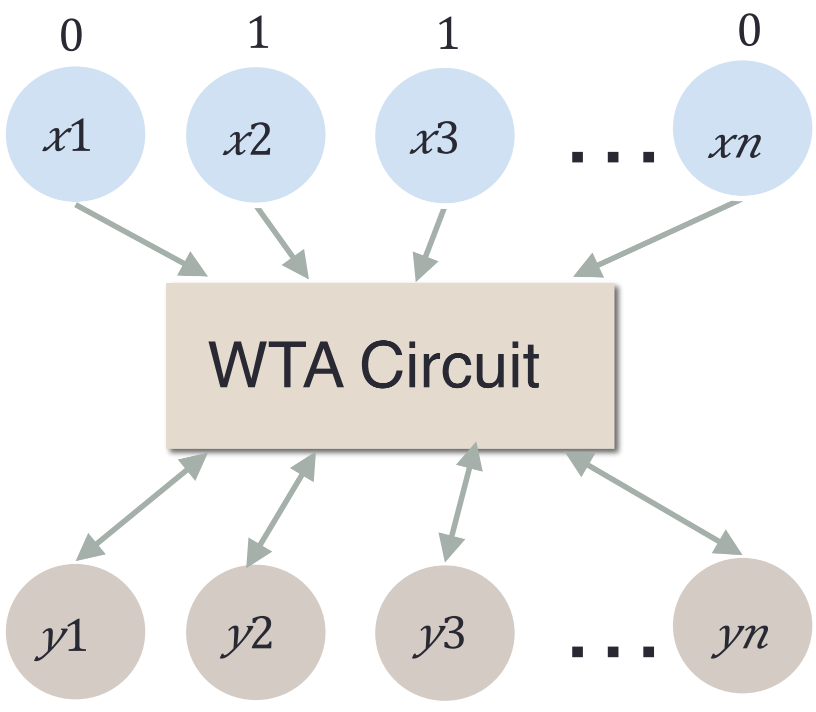
3. The Winner-Take-All Problem

- Nancy Lynch, Cameron Musco, Merav Parter. Computational Tradeoffs in Biological Neural Networks: Self-Stabilizing Winner-Take-All Networks. ITCS 2017.



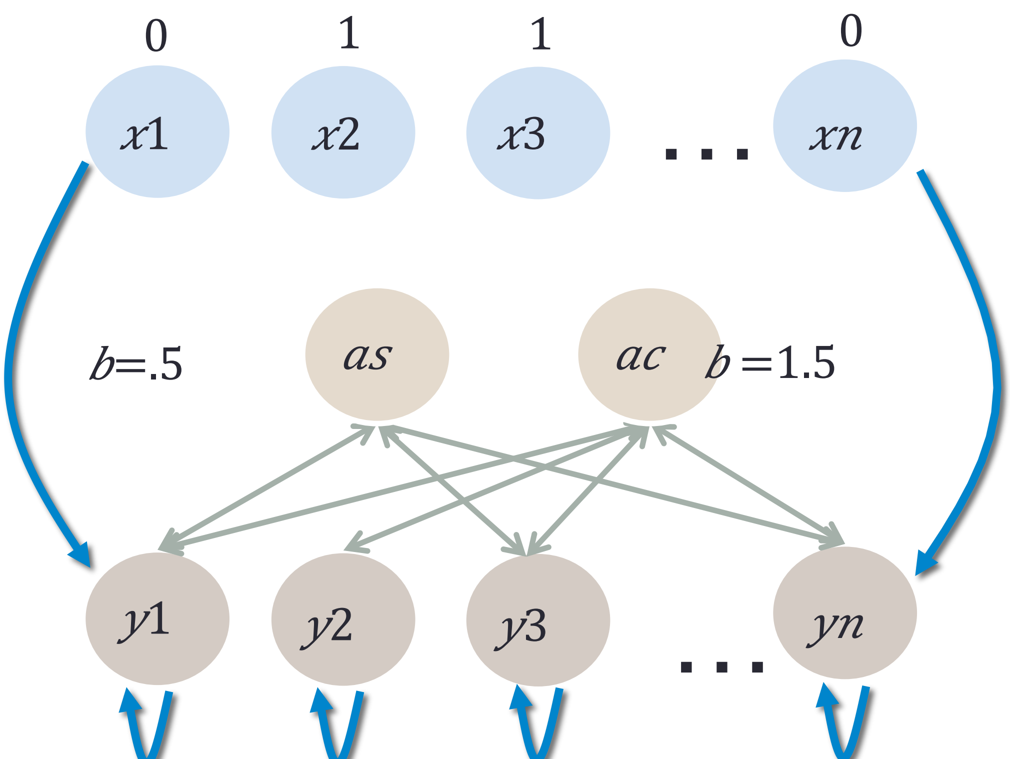
Winner-Take-All: $WTA(n, t_c, t_s, \delta)$

- n fixed inputs, each either always firing or always not firing.
- n corresponding outputs.
- Starting from any state, with probability $\geq 1 - \delta$, the network:
  - Converges, within a short time t_c, to a single firing output, which corresponds to a firing input, and then
  - Remains stable for a long time t_s.
- A neural leader election problem, studied in computational neuroscience.
- Used in perceptual attention, learning,…
- Powerful “nonlinear” primitive [Maass ‘99].



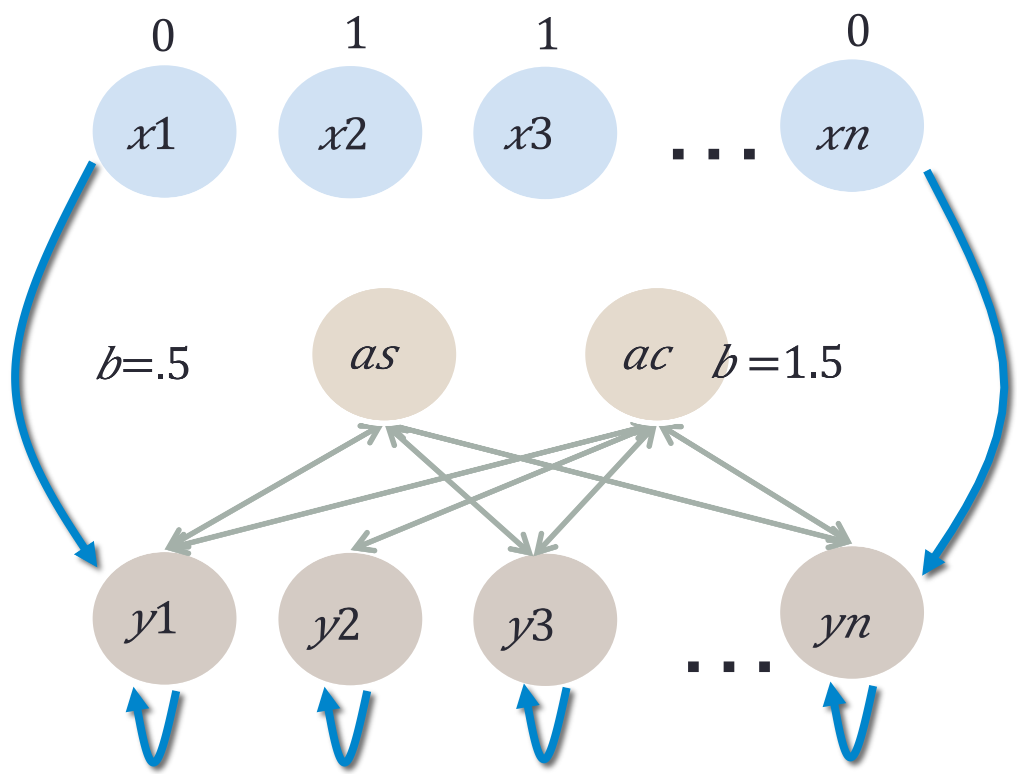
Simple Solution with Two Inhibitors

- Stability inhibitor a_s:
  - Fires with high probability whenever one or more outputs fire.
  - Prevents outputs that didn't fire at time t from firing at time $t+1$.
- Convergence inhibitor a_c:
  - Fires with high probability whenever two or more outputs fire.
  - Causes any output that fires at time t to fire at time $t+1$ with probability approximately ½.



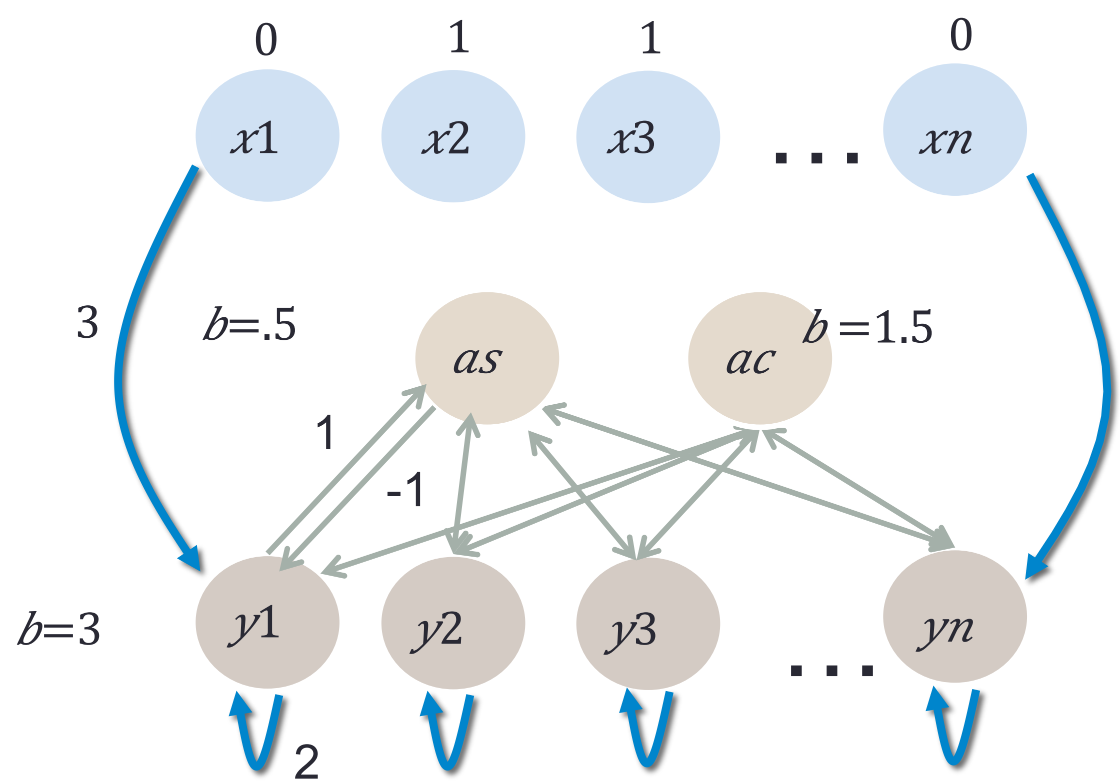
Simple Solution with Two Inhibitors

- Idea: Roughly half of the currently-firing outputs stop firing at each step.
- So with constant probability, there is some time $t_c \leq \log n$ such that exactly one output fires at time t.
- Moreover, after time t_c, with high probability, this selected output continues to fire for a long time t_s.
- Meanwhile, only inhibitor a_s fires, preventing all other outputs from firing.



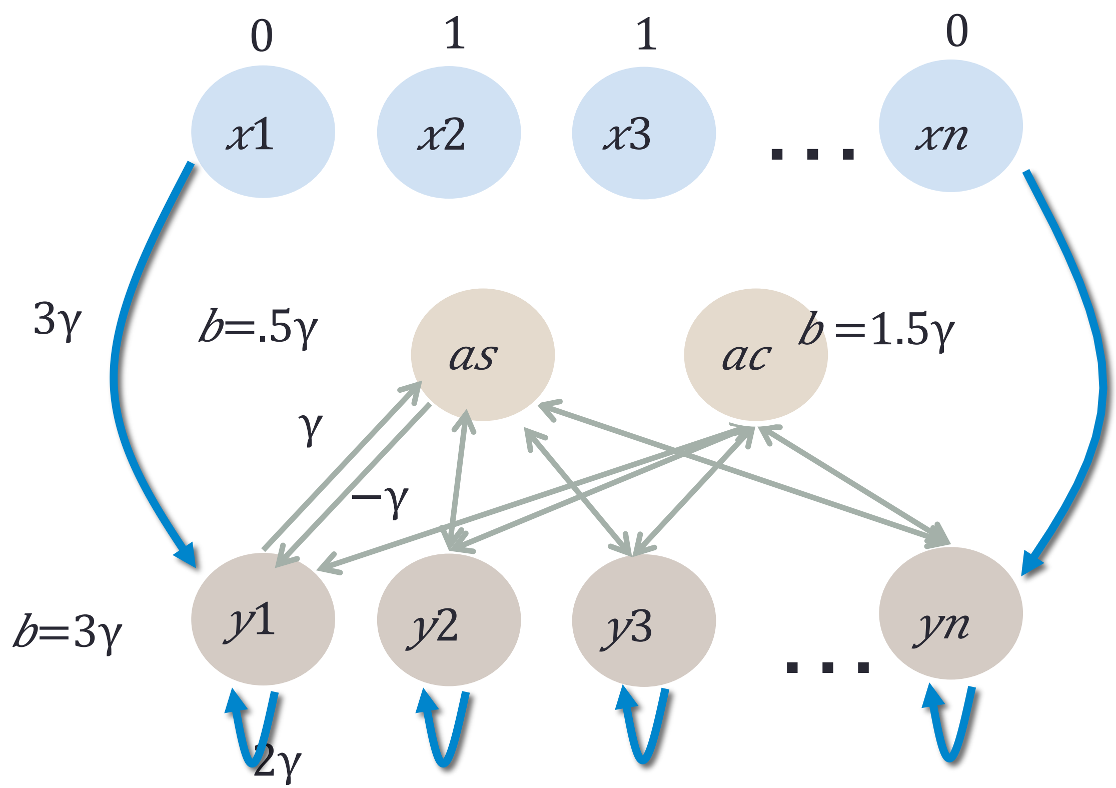
Simple Solution with Two Inhibitors

- Output neuron bias (threshold) = 3.
- Weights of (input, output) edges = 3.
- Weights of output self-loops = 2.
- Weights of (output, inhibitor) edges = 1.
- Weights of (inhibitor, output) edges = −1.



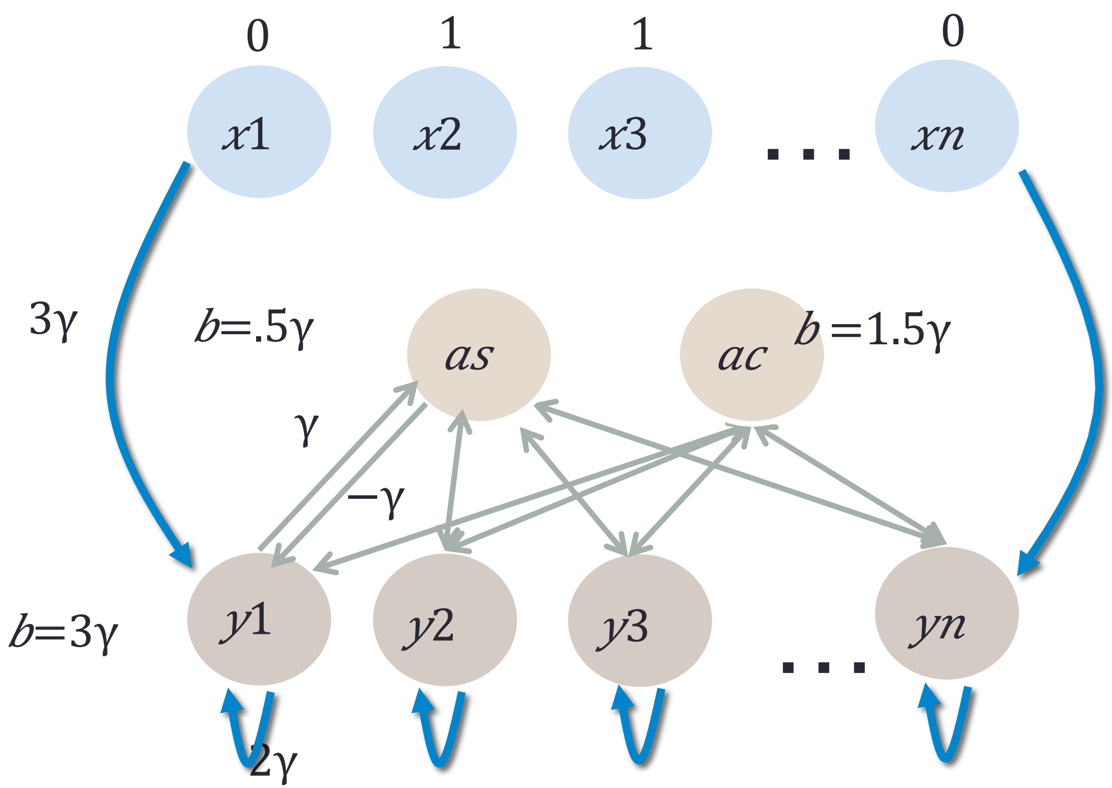
Include a Weighting Factor γ

- Multiply all weights and biases by a weighting factor γ, which must be sufficiently large with respect to n and stability time t_s.
- We can increase the stability time t_s by increasing γ; specifically, a linear increase in γ yields an exponential increase in t_s.
- Convergence time t_c is $O(\log n)$.



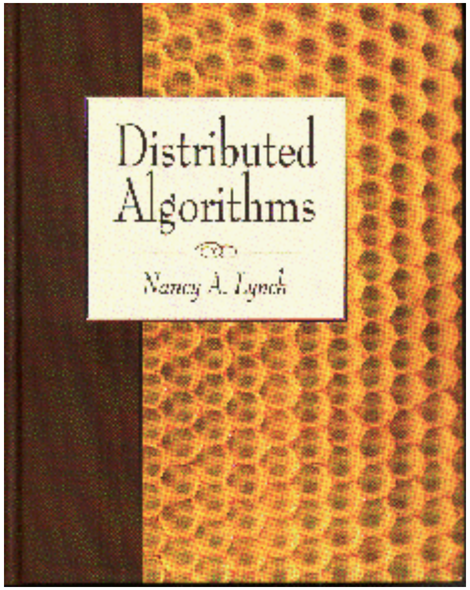
Our Main Theorem

- Theorem 1: Assume $\gamma \geq c \log(n\, t_s / \delta)$. Then starting from any state, with probability $\geq 1 - \delta$, the network converges, within time $t_c \approx c \log n \log(1/\delta)$, to a single firing output corresponding to a firing input, and remains stable for time t_s.
- Also:
  - Expected time result.
  - More than two inhibitors can be used to give faster convergence.
  - Can't solve the problem much faster with two inhibitors.
  - Can't solve it at all with one inhibitor.

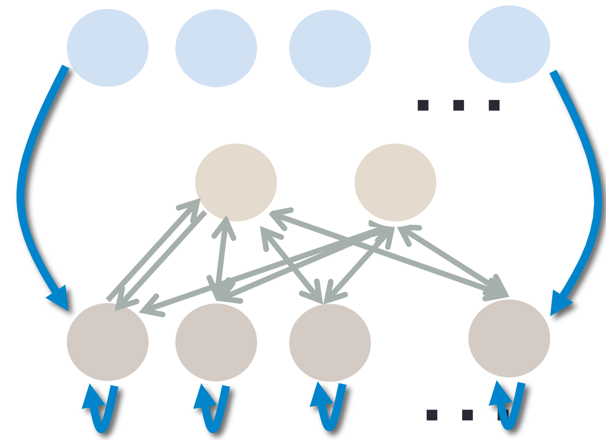


Proof of Correctness

- We must show that, with high probability:
  - (Convergence) System soon reaches a “valid WTA” configuration.
  - (Stability) Once it reaches such a configuration, it stays there for a long time.
- Similar properties are commonly proved for distributed algorithms; our proofs follow a style inspired by analysis of distributed algorithms:
  - Stability properties are proved using invariants.
  - Convergence properties proved by showing progress through a series of milestones.
- Except now we want high probability statements rather than absolute statements.



Stability Proof



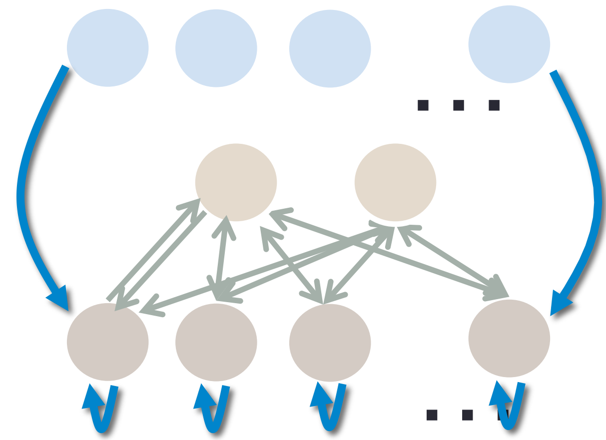
- Lemma 2 (Stability): Consider a "valid WTA" configuration C, in which:
  - Exactly one output, corresponding to a firing input, is firing,
  - a_s is firing, and
  - a_c is not firing.

  Then WHP, the next configuration is identical to C.

- Corollary 3: This situation persists for a long time (WHP):

  $C, C, C, \ldots$
- Duration t_s is exponential in the weighting factor γ.

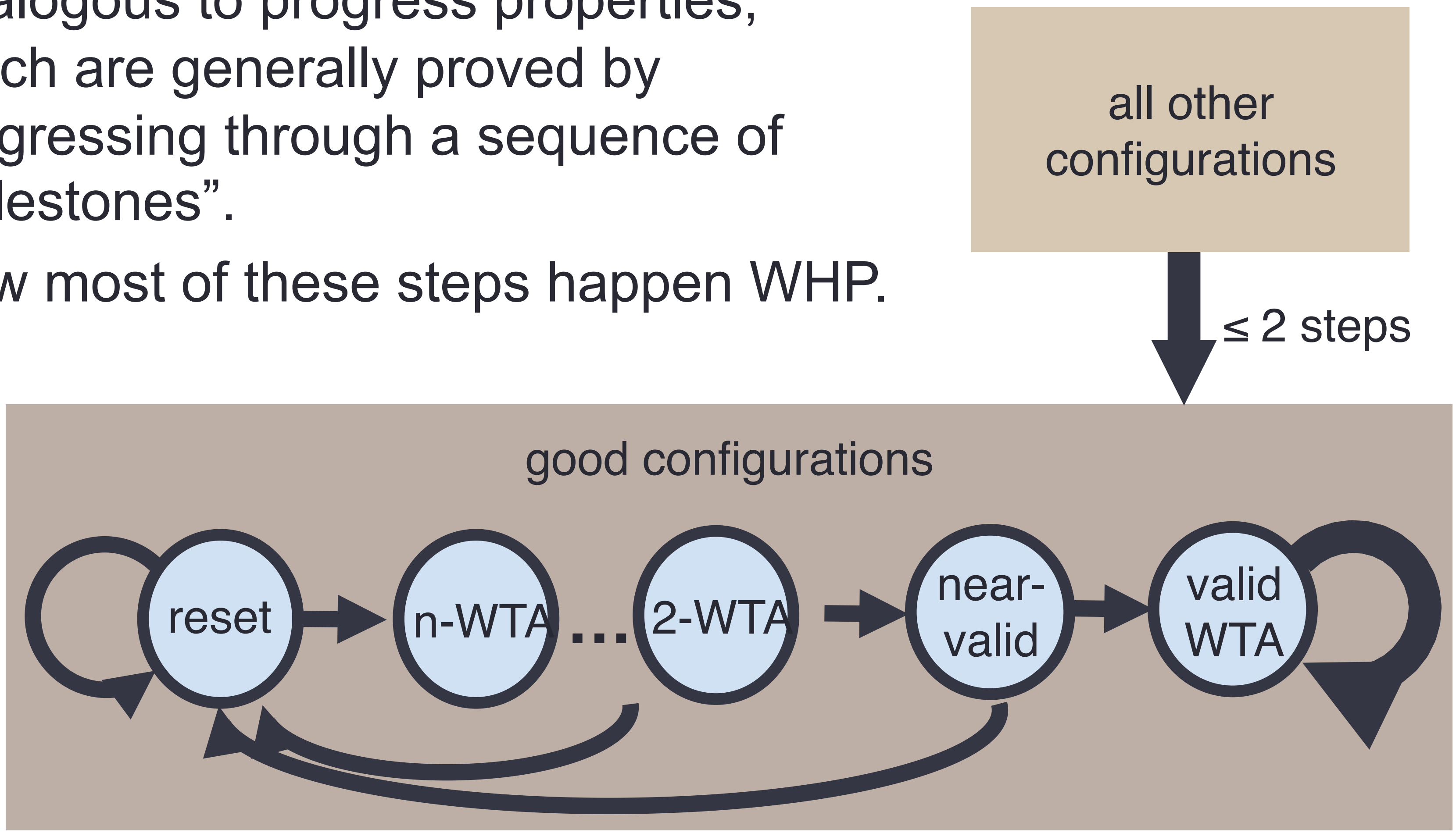
Stability Proof



- Follows typical style of invariant proofs.
- Uses a series of lemmas saying what is guaranteed (WHP) by each single transition.
- Namely, consider a configuration C, leading (probabilistically) to a new configuration C'. Then (WHP):
  - No output corresponding to a non-firing input fires in C'.
  - If a_s is the only inhibitor that fires in C, then the same outputs fire in C' as in C.
  - If there is exactly one firing output in C then a_s fires in C' and a_c does not.

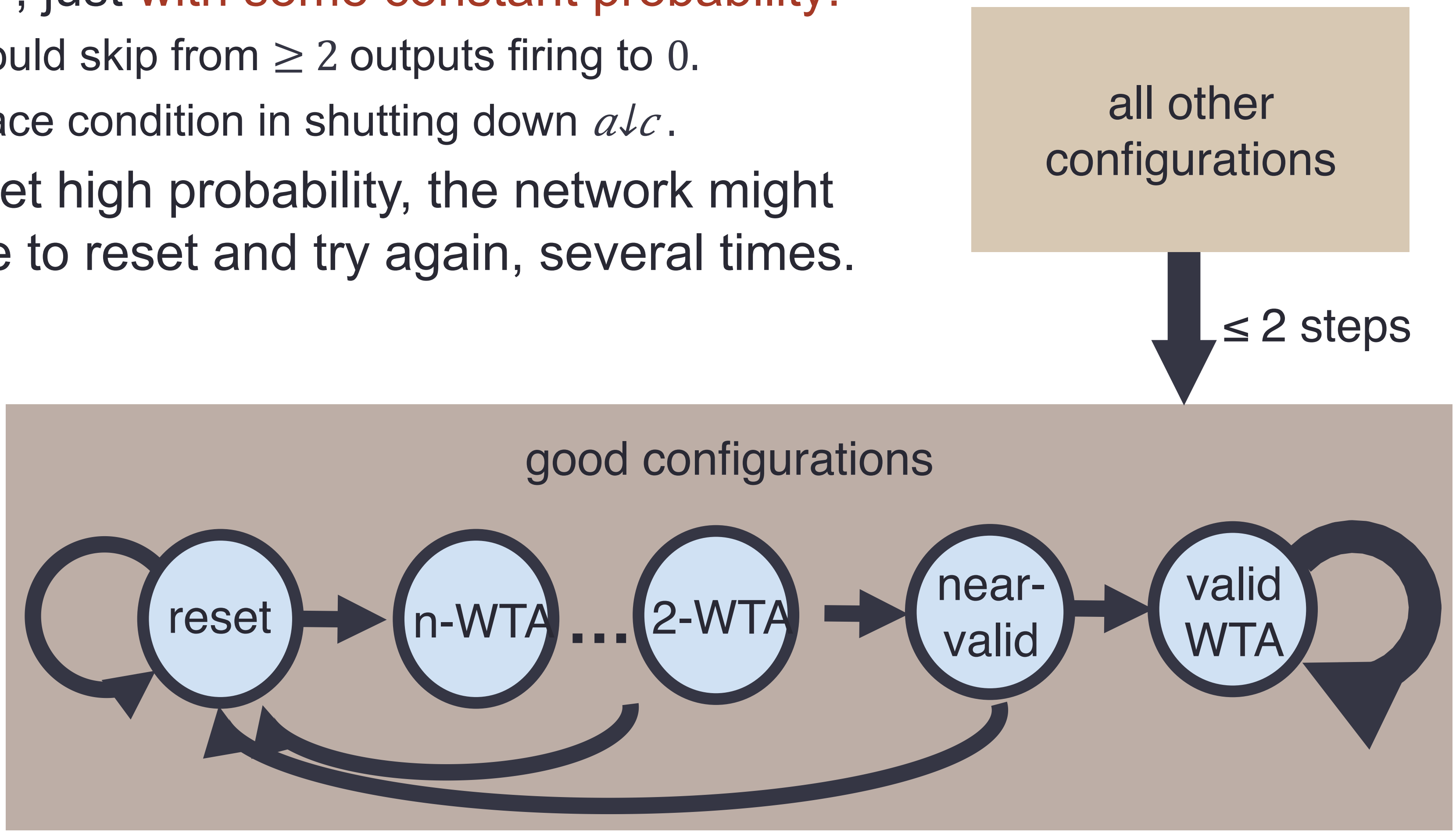
Convergence Proof

- Harder…Describes multi-step behavior.
- Analogous to progress properties, which are generally proved by progressing through a sequence of "milestones".
- Now most of these steps happen WHP.



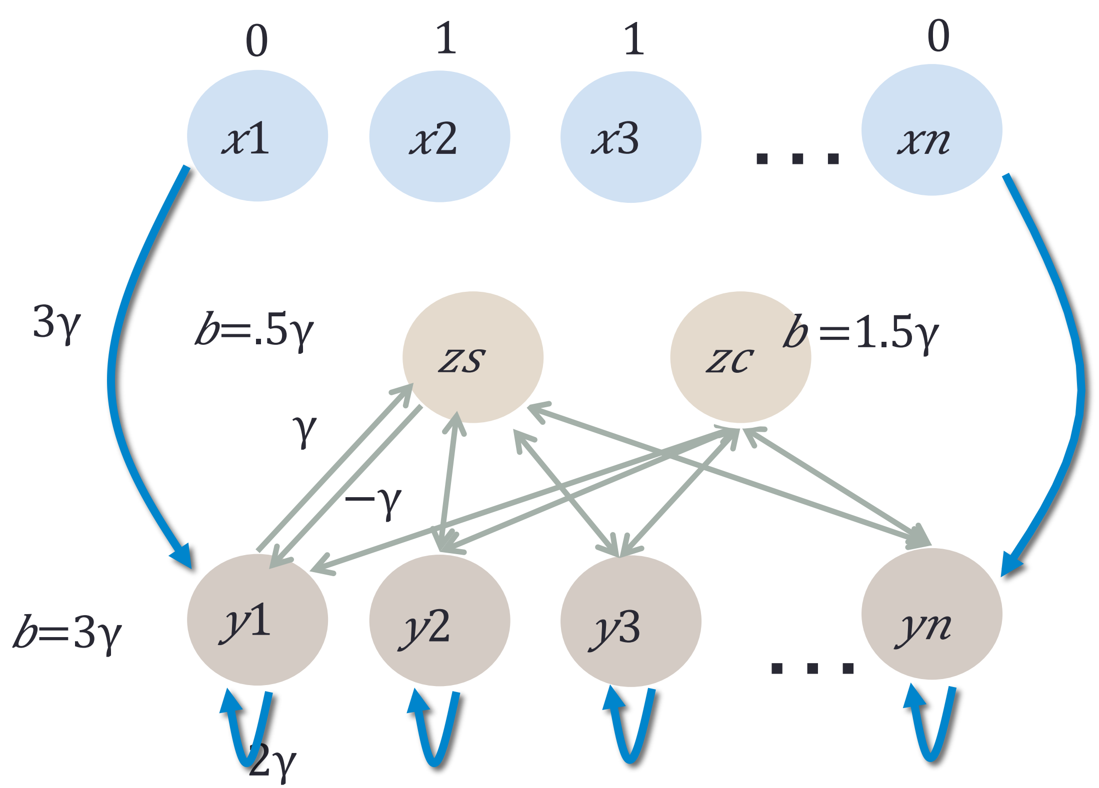
Convergence Proof

- The arrow from near-valid to valid WTA is not WHP, just with some constant probability:
  - Could skip from ≥ 2 outputs firing to 0.
  - Race condition in shutting down $a{\downarrow}c$.
- To get high probability, the network might have to reset and try again, several times.



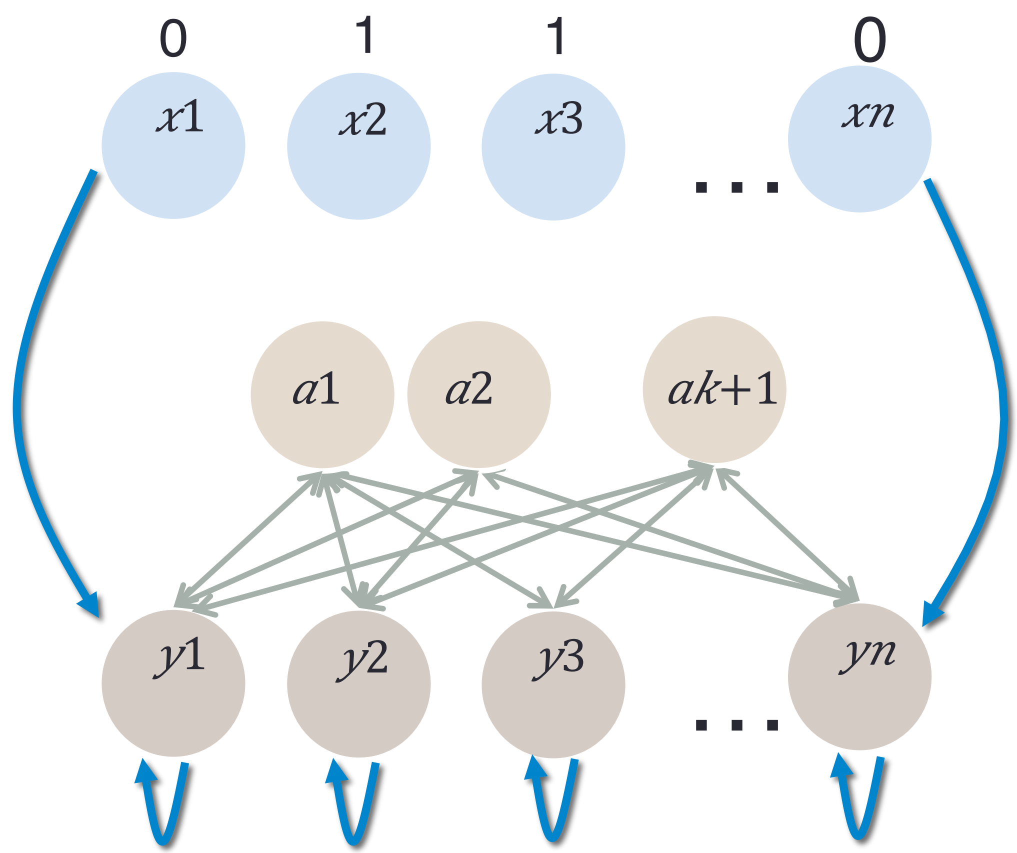
Convergence Proof

- Lemma 4 (Convergence): From any configuration C, the probability of reaching a valid WTA configuration within time $\approx c \log n$ is $\geq 1/18$.
- Theorem 1: If $\gamma \geq c \log(n\, t_s / \delta)$, then the network solves $WTA(n, t_c, t_s, \delta)$, with $t_c \approx c \log n \log(1/\delta)$.



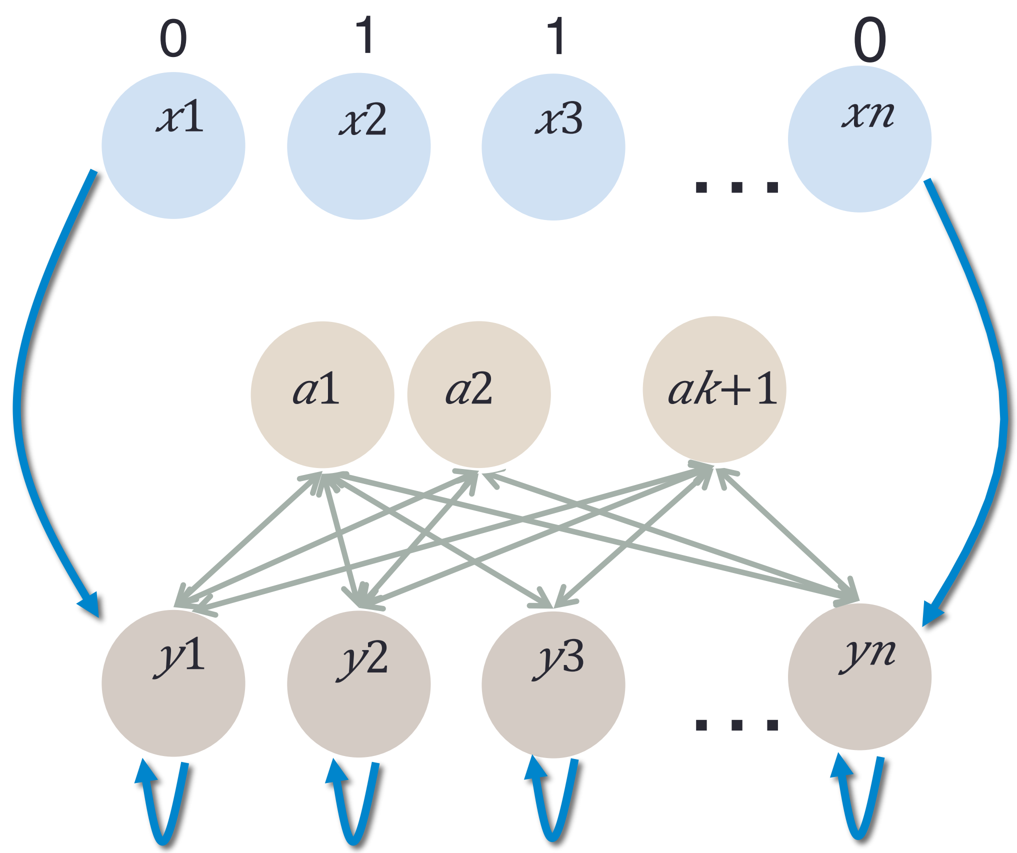
Faster Solution with More Inhibitors

- Uses one stability inhibitor, plus k convergence inhibitors, with exponentially-growing biases.
- These extra inhibitors speed up convergence when there are many firing outputs:
  - Higher-bias inhibitors trigger when more outputs fire.
  - Many firing inhibitors serve to decrease the output neurons' probability of continuing to fire.



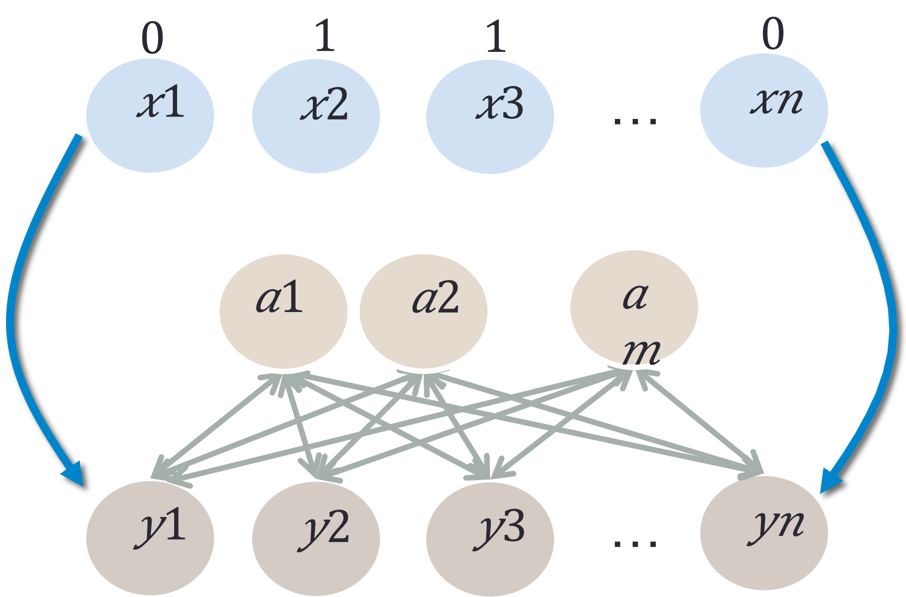
Faster Solution with More Inhibitors

- Theorem 5: Assume $\gamma \geq c \log(n t_s / \delta)$. Then from any state, with prob $\geq 1 - \delta$, the network converges, within time $t_c \approx c\, k \log^{1/k} n \log(1/\delta)$, to a single firing output corresponding to a firing input, and remains stable for time t_s.



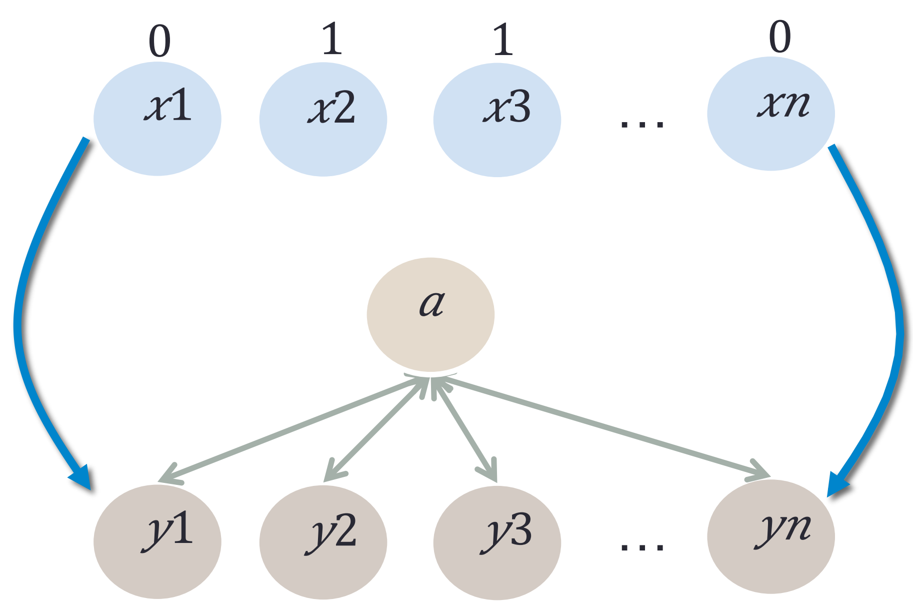
Lower Bounds

- Based on a slightly restricted version of the model:
  - Inputs connect to no outputs except their own.
  - Outputs do not connect to each other.
  - Auxiliary neurons are all inhibitors.
- These conditions are satisfied by our algorithms.
- Proofs depend on locality arguments as commonly used in distributed computing theory.



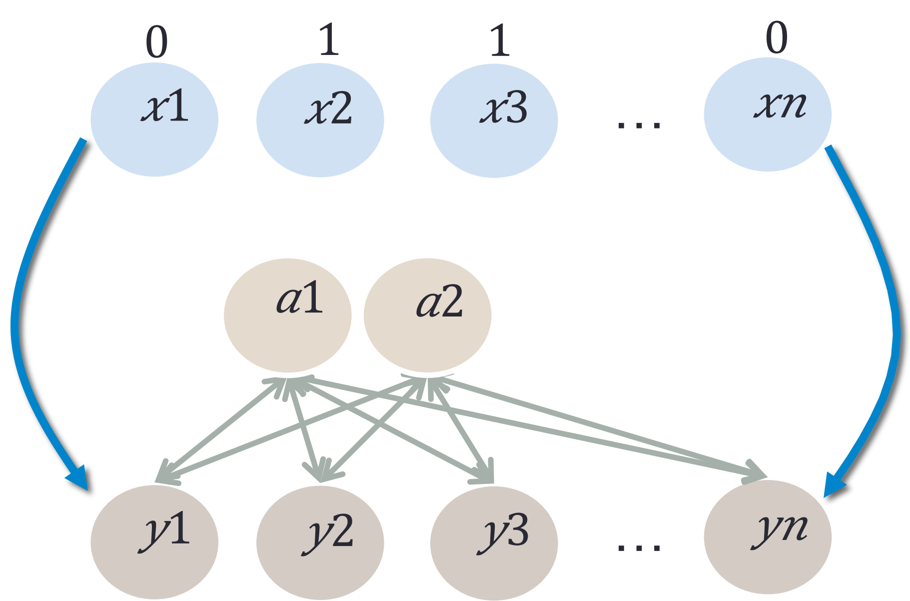
Lower Bounds

- Theorem 6: No SNN with just one auxiliary neuron can solve WTA with stability time $t_s \gg$ convergence time t_c.
- Proof idea: Assume a WTA network with one inhibitor a.
- Claim 1: If a is not firing, then any output with a firing input will fire (with good probability); this is needed to ensure that at least one output will fire by the required convergence time t_c.
- Claim 2: If a is firing, then any output that was firing will stop firing (with good probability); this is needed to ensure that at most one output is firing by t_c.
- This combination makes it hard to maintain stability.

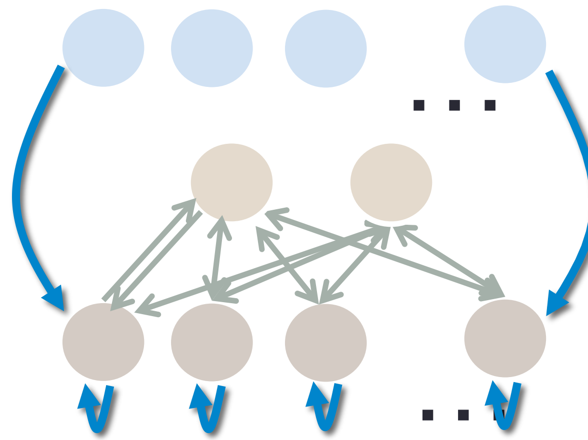


Lower Bounds

- Theorem 7: No SNN with two auxiliary neurons can improve on the convergence-time bound in our two-neuron solution by more than a factor of $O(\log\log n)$.
- Idea: Fast convergence requires a lot of inhibition when many outputs fire.
- This required inhibition is too high when a few outputs fire.



Discussion



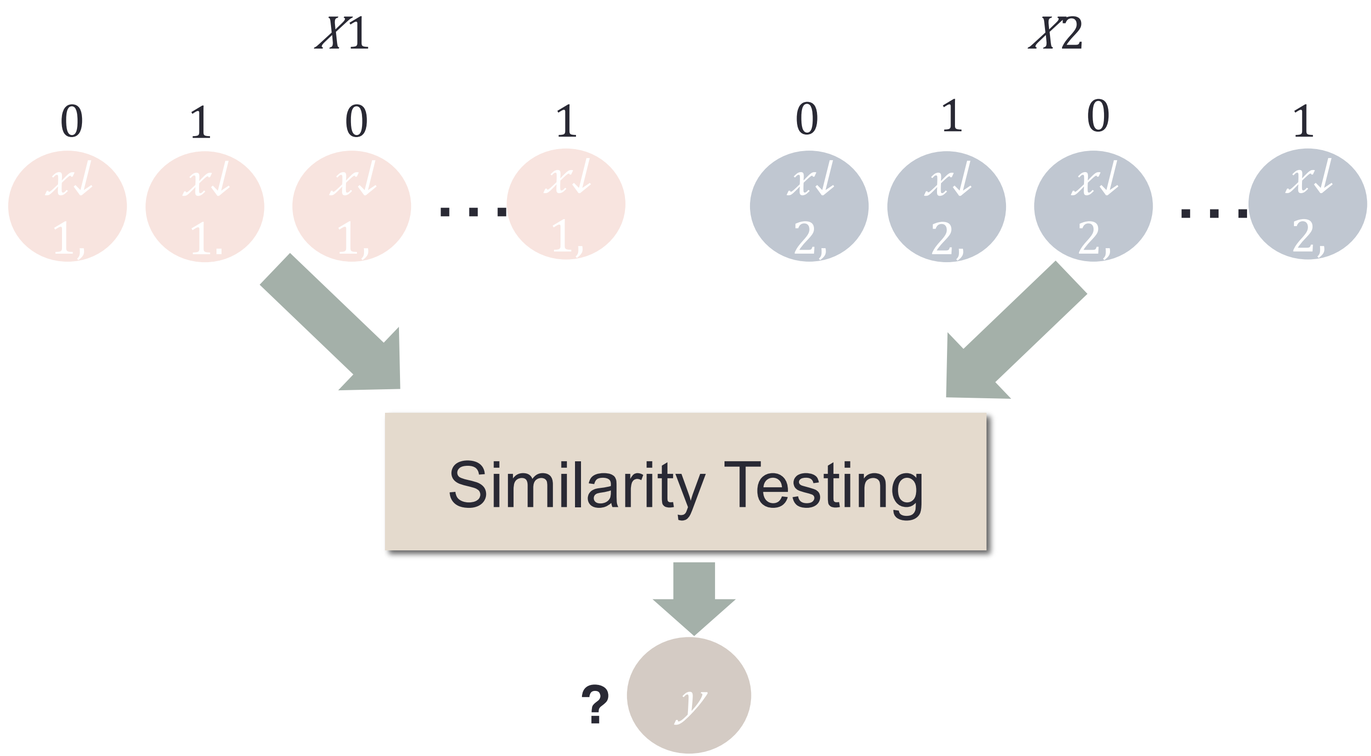
- Inhibition:
  - In our networks, inhibitors are used to achieve two goals: stability and convergence.
  - Inhibition is often viewed as a stability mechanism in the brain; in our networks, it also helps to drive computation toward convergence.
- Randomness:
  - Randomness is a source of noise that can introduce inaccuracies and slow down computation.
  - But it can also be a powerful computational resource.
  - Here, randomness is used to break symmetry among output neurons.

This talk

1. Introduction √
2. Stochastic SNNs √
3. Winner-Take-All algorithms and lower bounds √
4. Similarity Testing and Indexing
5. Composing Stochastic SNNs
6. Discussion

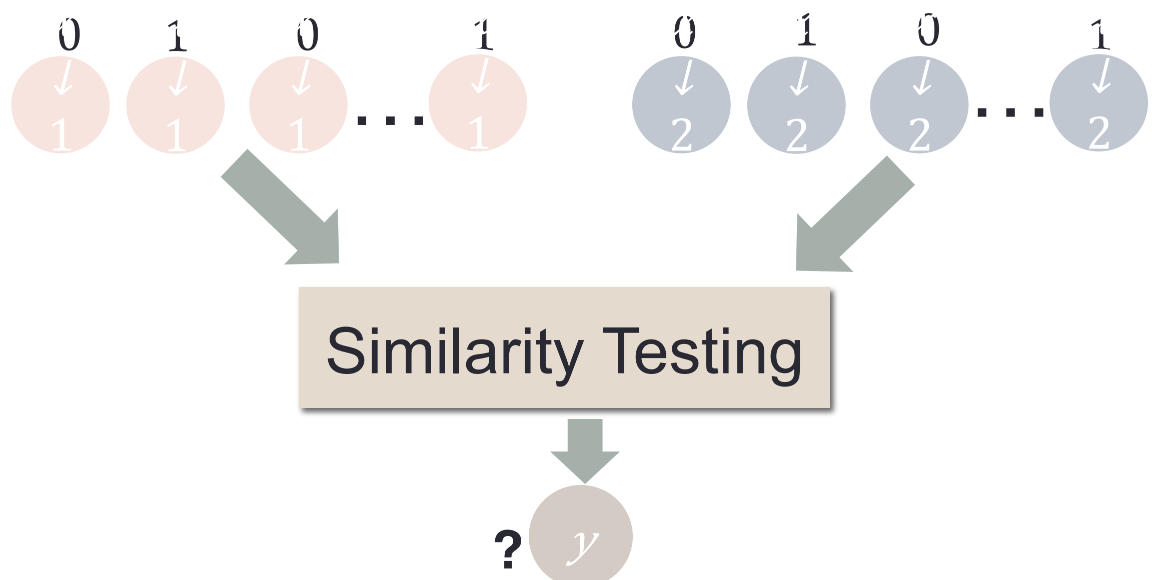
4. Similarity Testing and Indexing

- Nancy Lynch, Cameron Musco, Merav Parter. Neuro-RAM Unit with Applications to Similarity Testing and Compression in Spiking Neural Networks. DISC 2017.



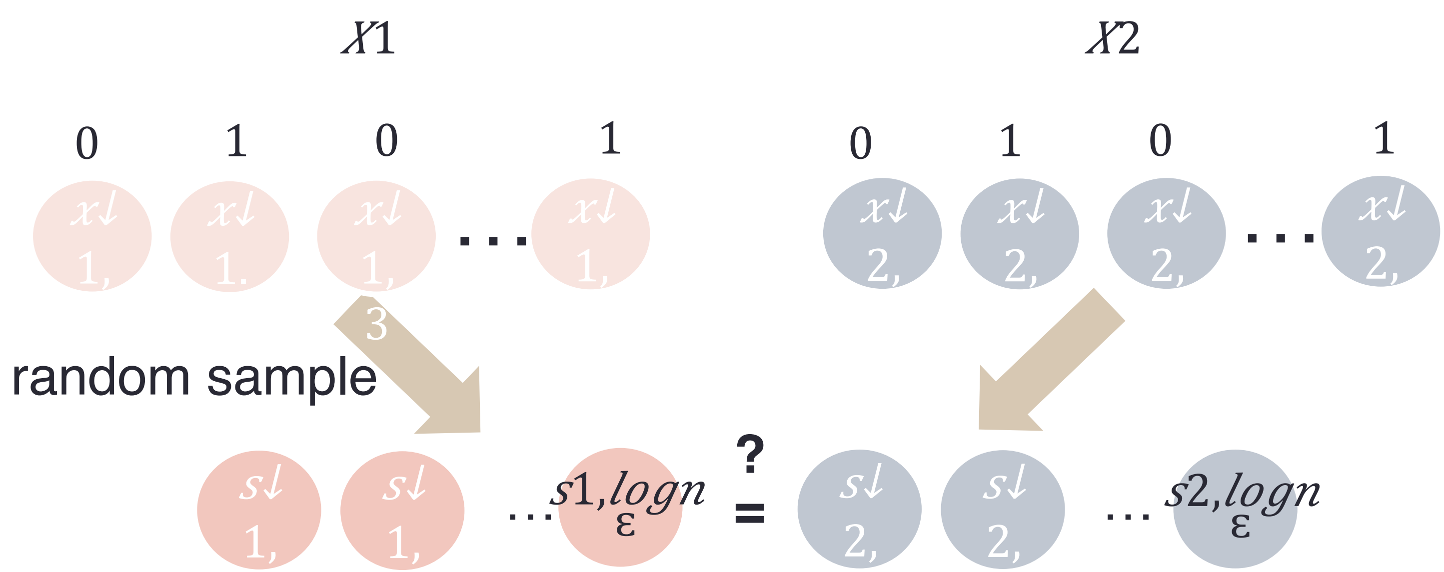
Similarity Testing

- Similarity testing problem: Given two input firing patterns X_1 and X_2, distinguish the case where $X_1 = X_2$ from the case where they are far from being equal, e.g., $d(X_1, X_2) \geq \epsilon n$.
- After convergence, the output neuron should fire continuously if the inputs are equal, and not fire if they are far from equal.
- A natural sub-problem for pattern recognition and other tasks.



Algorithm Idea

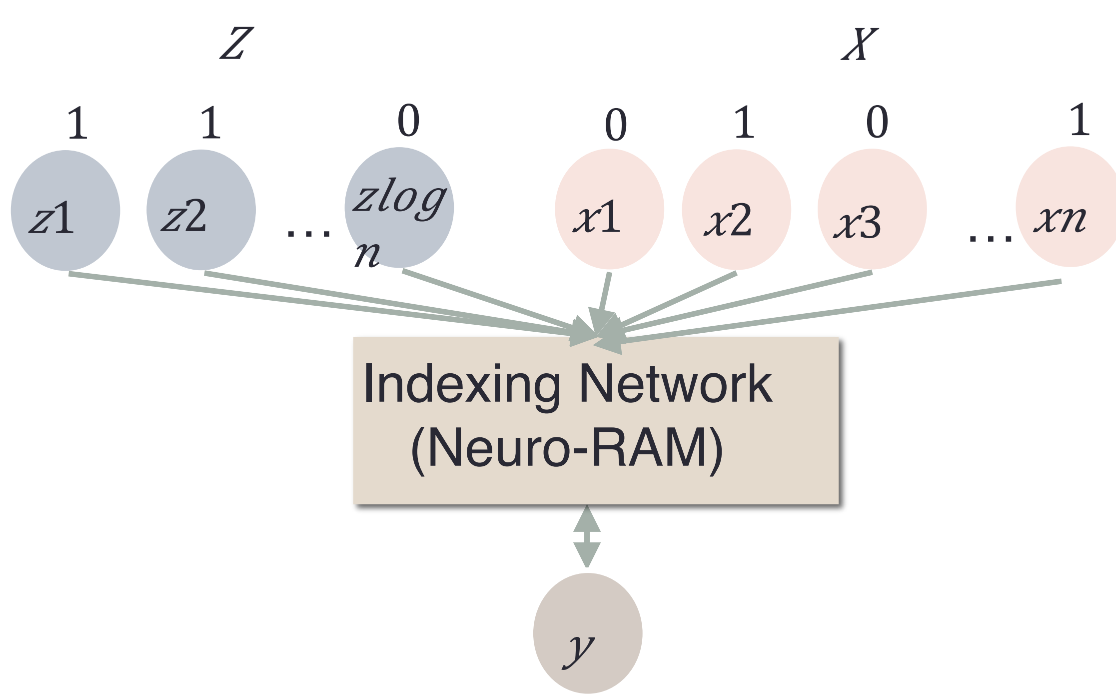
- Simple (non-neural) sublinear time algorithm: Sample $O(\log n / \epsilon)$ random positions and check whether X_1 and X_2 match at those positions.



- If $X_1 = X_2$, then $S_1 = S_2$. If $d(X_1, X_2) \geq \epsilon n$, then with high probability, $S_1 \neq S_2$.

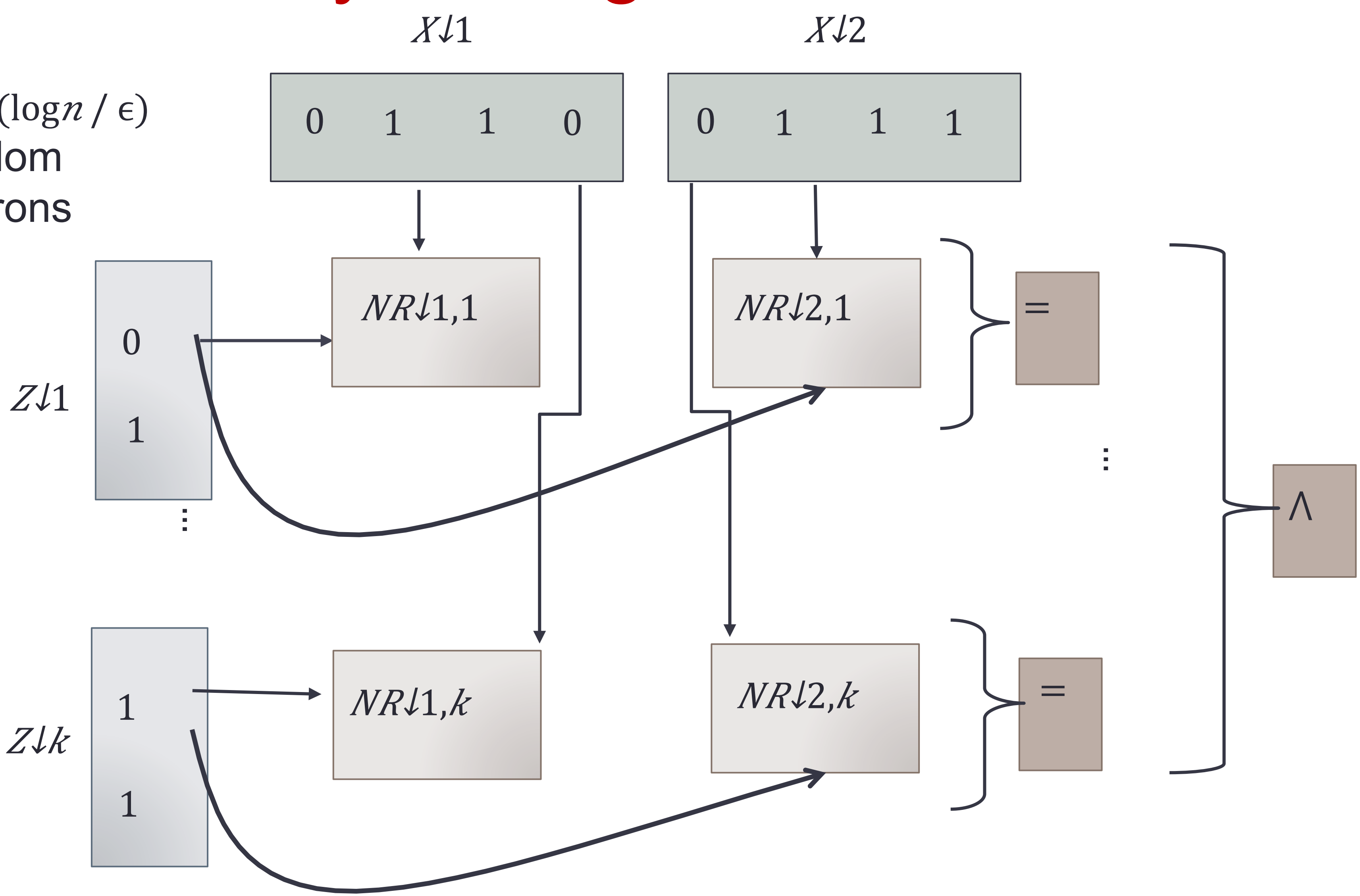
Implementing the Algorithm in an SNN

- Equality check between S_1 and S_2 is straightforward.
- For sampling random positions, we use an Indexing Module (Neuro-RAM): given an index encoded by the firing pattern of a set of neurons, select the appropriate value of X_1 or X_2.
- After convergence, the output neuron should fire continuously if and only if $X(Z)$ is firing.
- Simulates an excitatory connection from $X(Z)$ to y.



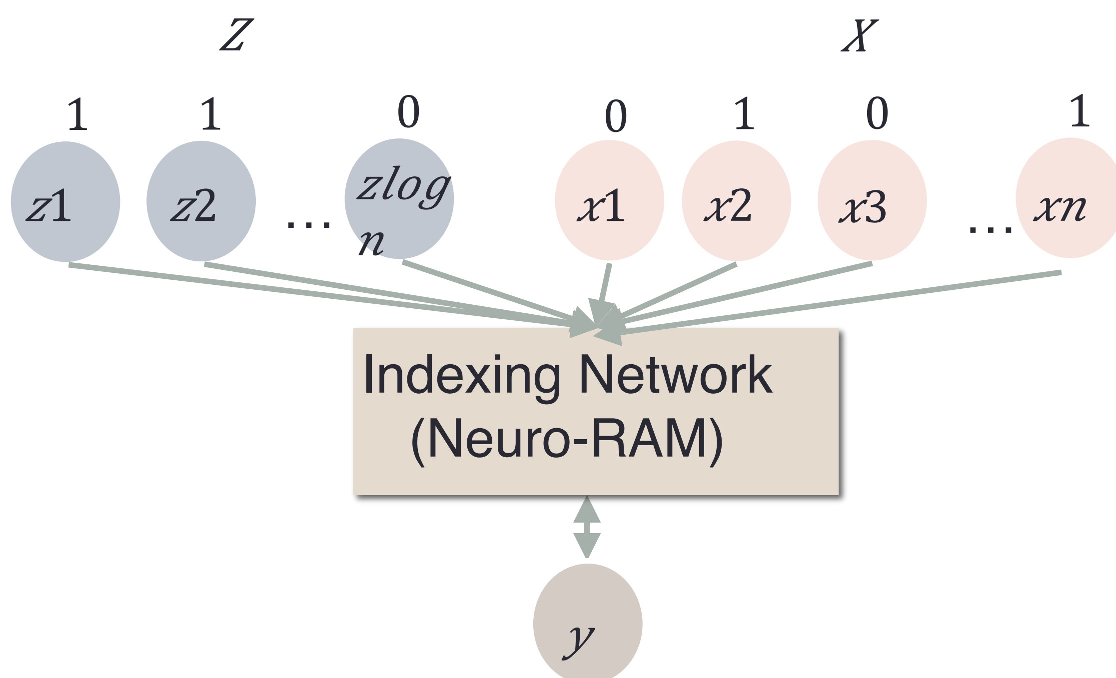
Similarity Testing with Neuro-RAM

$k = O(\log n / \epsilon)$
random
neurons



Indexing Problem (Neuro-RAM)

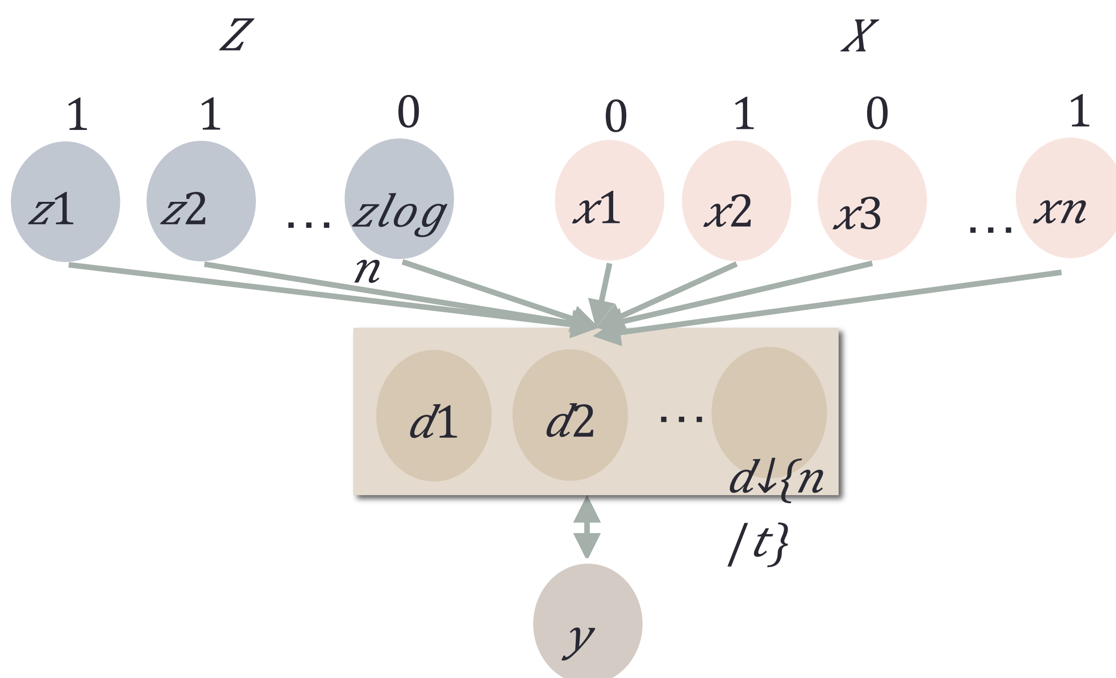
- Indexing might not seem very "neural".
- Some neural motivation: Uses information contained in a small set of neurons (the index) to access information from a much larger data store.
- This seems to be an important primitive for other applications besides similarity testing.
- E.g., a smell, or sight, or a word triggering a memory.



Main Algorithmic Result for Indexing

- Theorem 1: For any $t \le \sqrt{n}$, there is an SNN solving the indexing problem with $O(n/t)$ auxiliary neurons that converges by time t (WHP).

- Corollary: A sublinear-sized circuit for the similarity testing problem: $O(\sqrt{n}\log n/\epsilon)$ auxiliary neurons, running in time $O(\sqrt{n})$.



Implementation of Neuro-RAM Module

- Example: $O(\sqrt{n})$ auxiliary neurons implementing Neuro-RAM in $O(\sqrt{n})$ rounds.
- Divide n input neurons X into $\sqrt{n}$ buckets.
- Divide $\log n$ index neurons Z into two halves.

- Step 1: Select a bucket X_i using first half of Z.
- Step 2: Select the desired index inside the bucket X_i using the last half of Z.
- Various forms of trickiness involved in encoding and decoding, not very neural.

Our Main Results for Indexing

Theorem 1 (Upper Bound):
For any $t \leq \sqrt{n}$, there is an SNN solving the indexing problem with $O(n/t)$ auxiliary neurons that converges by time t time WHP.

Theorem 2 (Lower Bound):
Any circuit that solves the indexing problem and converges by t time WHP, requires $\Omega(n/t \log^2 n)$ auxiliary neurons.

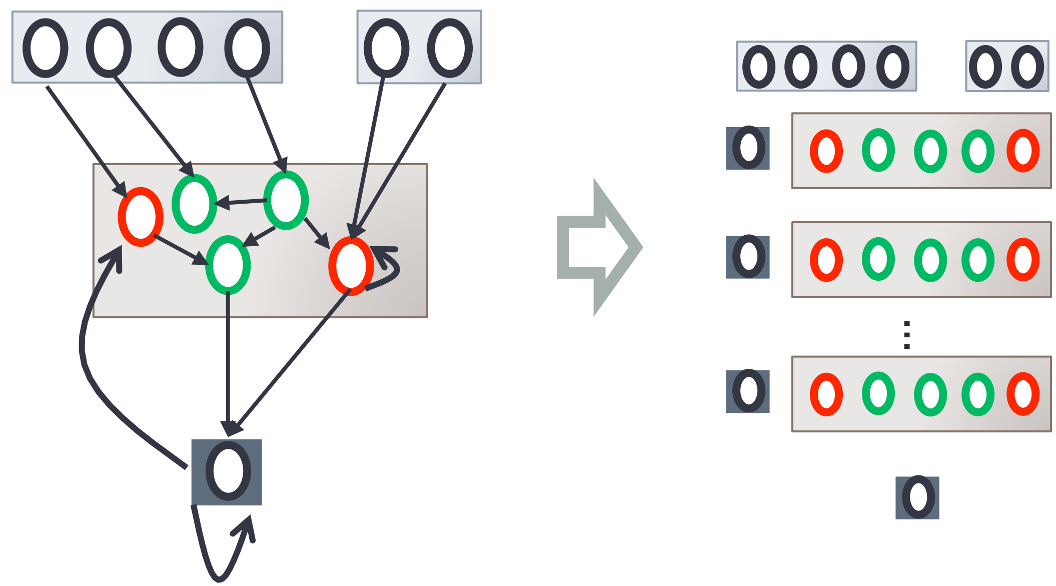
Lower Bound for Indexing

- This lower bound shows that our two-inhibitor network's convergence time cannot be improved by more than a $\log_2 n$ factor.

- We also get a stronger lower bound for Feed-Forward SNNs.
- This separates FF SNNs from FF circuits composed of sigmoidal gates with real-valued outputs; these can implement indexing with $O(\sqrt{n})$ neurons, in $O(\sqrt{n})$ time.

Proof of Lower Bound for Indexing

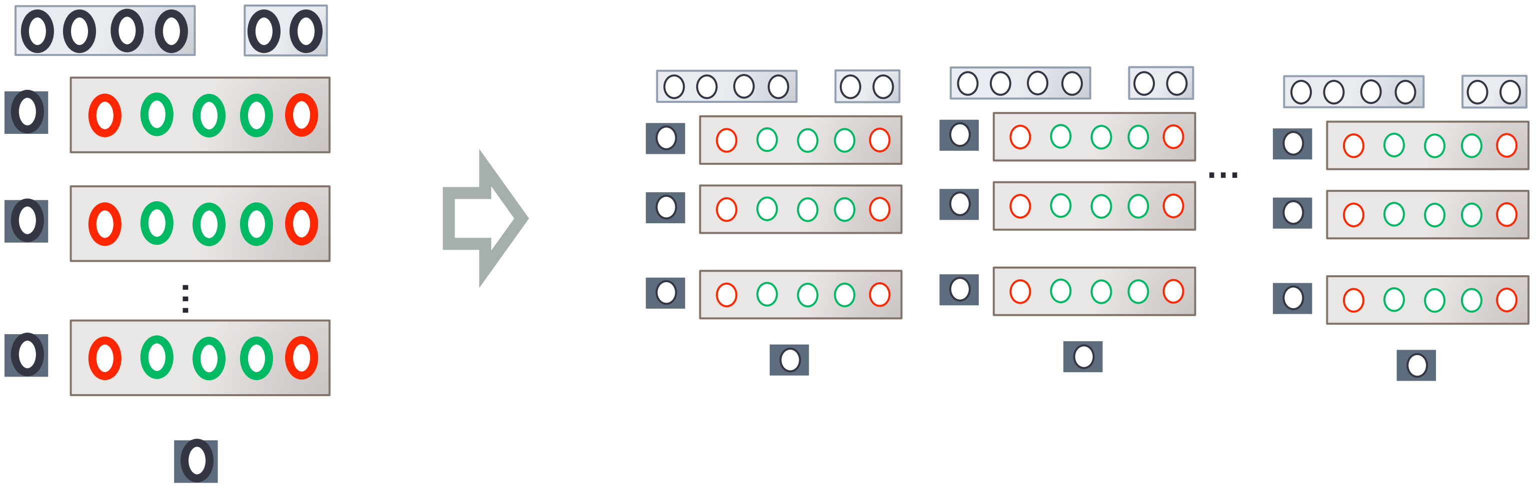
- Step 1: Transform an SNN for Indexing to an equivalent Feed-Forward SNN:



- Step 2: Convert the FF SNN to a probability distribution on deterministic circuits.
- Step 3: Identify a single circuit, and prove a lower bound for the circuit using a VC-dimension argument.

Proof of Lower Bound for Indexing

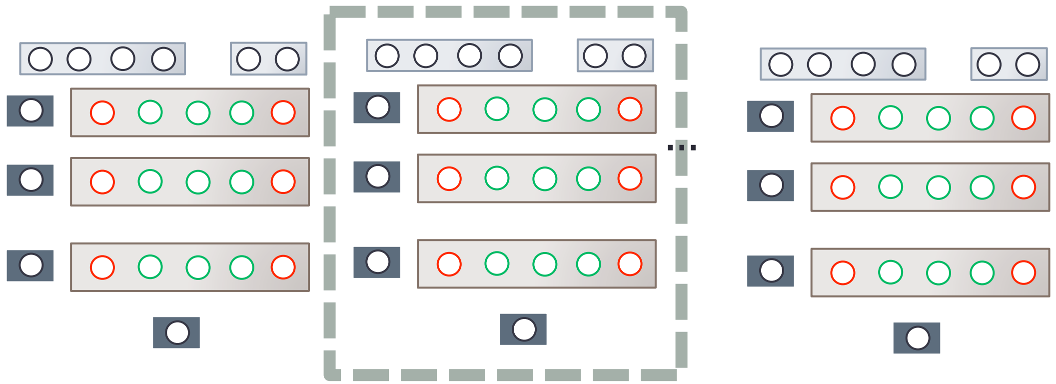
- Step 1: Transform an SNN for Indexing to an equivalent Feed-Forward SNN.
- Step 2: Convert the FF SNN to a probability distribution on deterministic circuits:



- Step 3: Identify a single circuit, and prove a lower bound for the circuit using a VC-dimension argument.

Proof of Lower Bound for Indexing

- Step 1: Transform an SNN for Indexing to an equivalent Feed-Forward SNN.
- Step 2: Convert the FF SNN to a probability distribution on deterministic circuits.
- Step 3: Identify a single circuit, and prove a lower bound for the circuit using a VC-dimension argument:



Our Main Results for Indexing

Theorem 1 (Upper Bound):
For any $t \leq \sqrt{n}$, there is an SNN solving the indexing problem with $O(n/t)$ auxiliary neurons that converges by time t time WHP.

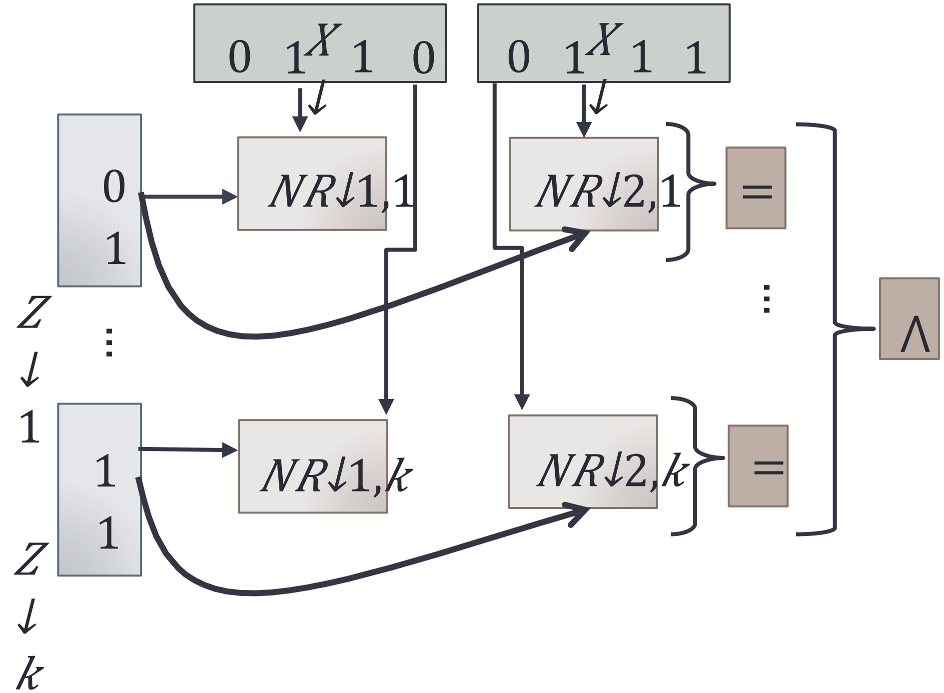
Theorem 2 (Lower Bound):
Any circuit that solves the indexing problem and converges by t time WHP, requires $\Omega(n/t \log^2 n)$ auxiliary neurons.

Our Main Results for Applications

Theorem 3 (Similarity Testing):
There is an SNN with $O(\sqrt{n}\log n/\epsilon)$ auxiliary neurons that solves ϵ-approx. equality in time $O(\sqrt{n})$.

Theorem 4 (Compression):
There is an SNN that implements random projection from dimension D to d, using $O(D/d)$ Neuro-RAM modules each with $O(\sqrt{D})$ neurons, in time $O(\sqrt{D})$.

Discussion



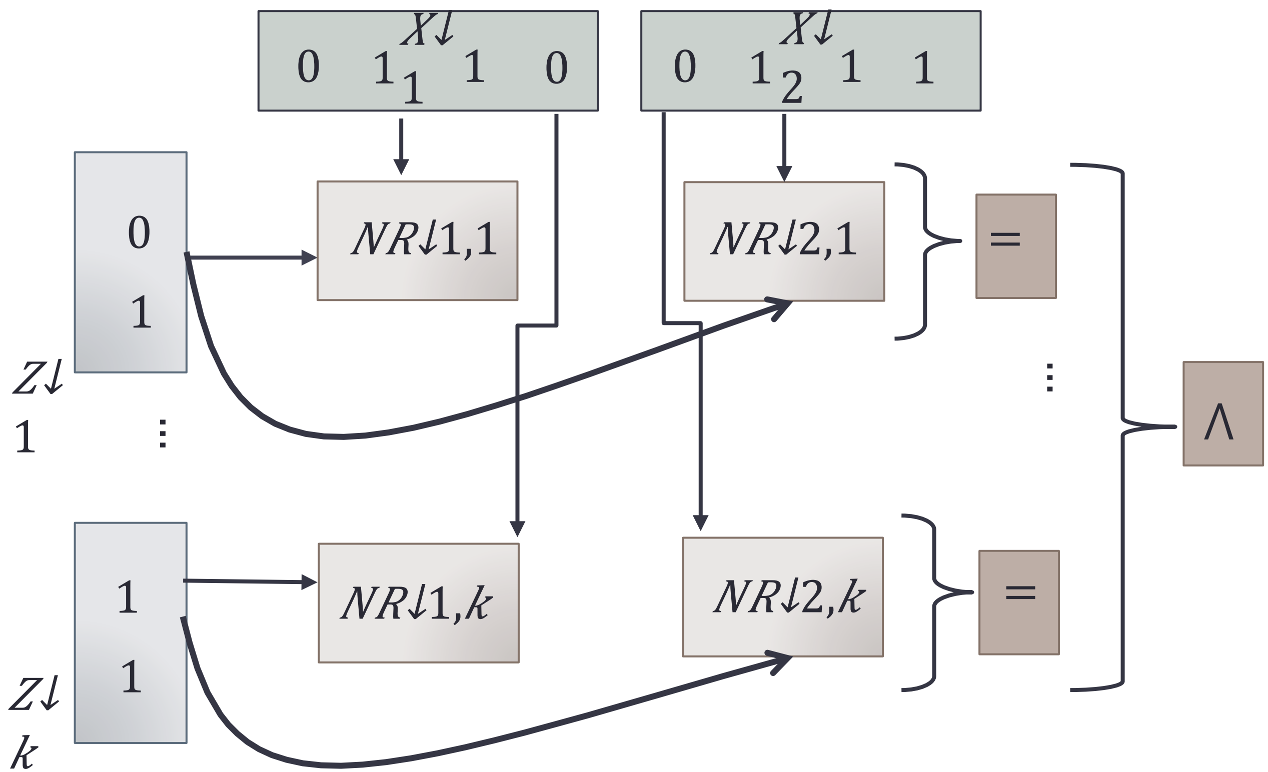
- Randomness:
  - Here used for random sampling.
  - Useful in the brain for compression, abstraction, and comparison.
- Indexing:
  - A “subroutine” that can be used in random-sampling networks.
  - Indexing can be implemented with a compact spiking network, but the network seems too complicated for a biological implementation. Are there other, more bio-like, indexing schemes?
  - Alternative approaches to random sampling might involve methods like Johnson-Lindenstrauss projection (multiplication by a random matrix), where randomness is used in the design of the network.

This talk

1. Introduction √
2. Stochastic SNNs √
3. Winner-Take-All algorithms and lower bounds √
4. Similarity Testing and Indexing √
5. Composing Stochastic SNNs
6. Discussion

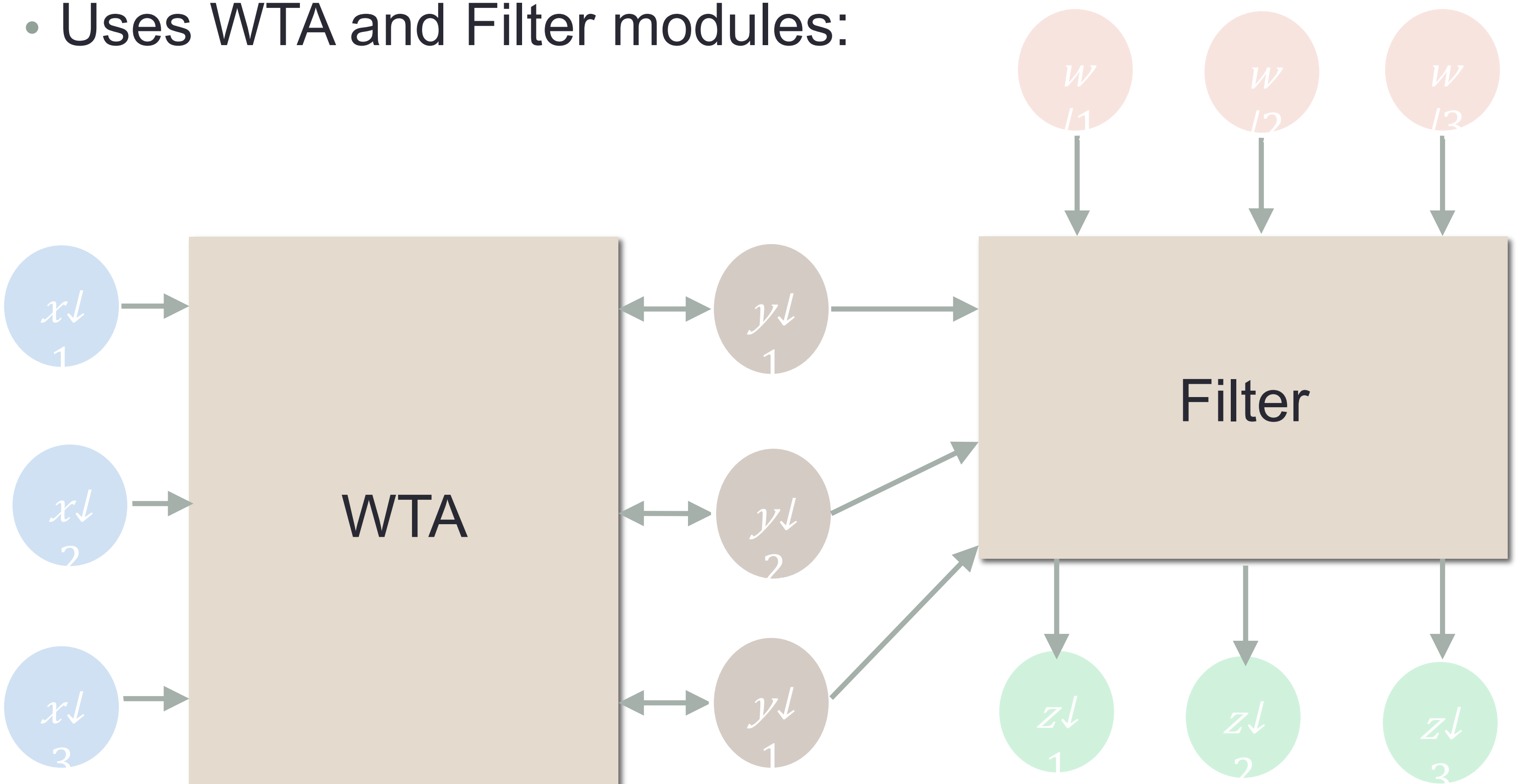
5. Composing Spiking Neural Networks

- Nancy Lynch, Cameron Musco. A Compositional Model for Spiking Neural Networks. In progress
- Combine networks that solve simple problems into larger networks that solve more complex problems.
- Example:
  Compose several Neuro-RAM modules (and logical gates) to solve Similarity Detection:



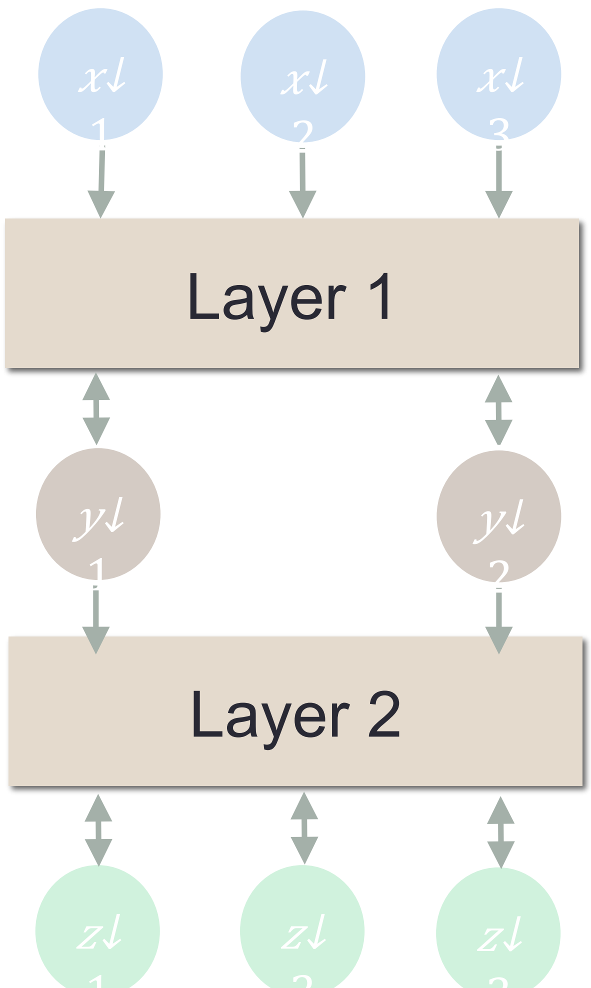
Composing Spiking Neural Networks

- Attention network: Processes a sequence of inputs and focuses attention on the "relevant" ones.
- Uses WTA and Filter modules:

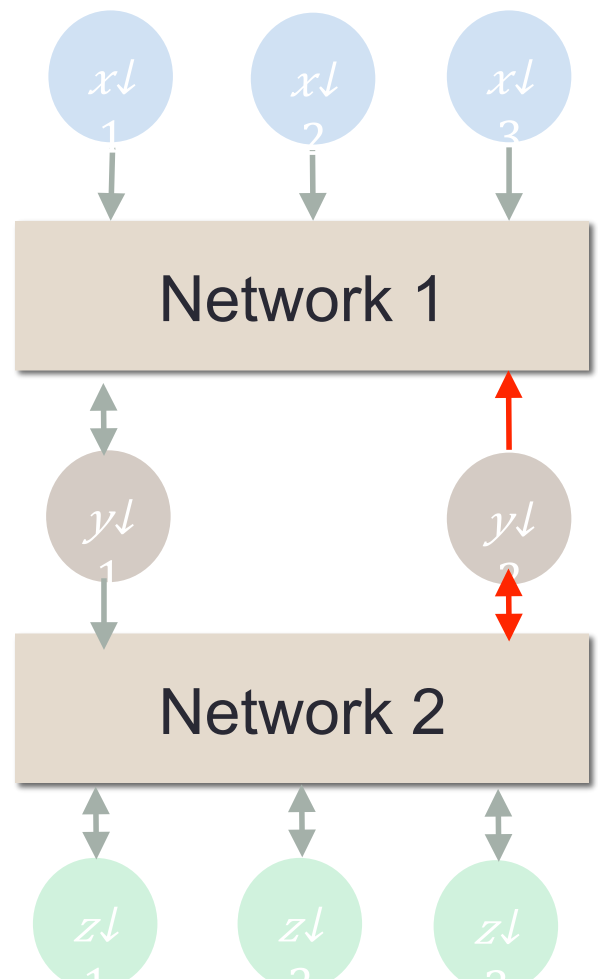


Composing Spiking Neural Networks

- Layered composition

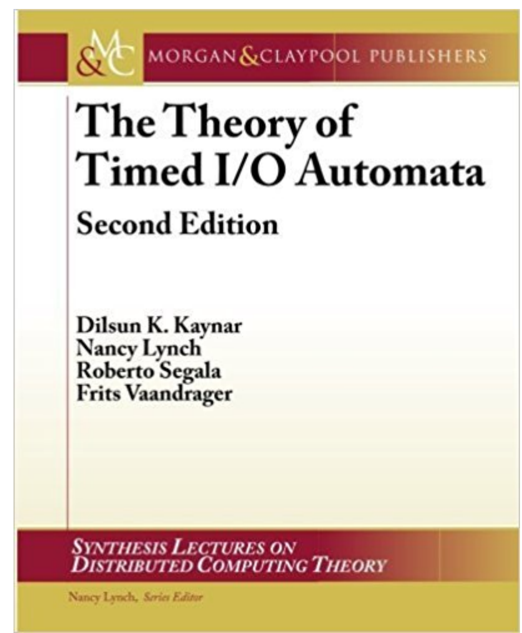


- Composition with feedback



Composing Spiking Neural Networks

- Following paradigms of Concurrency Theory, we are developing math foundations for composing SNNs.
- Define the external behavior of a network: a mapping from infinite sequences of input firing patterns to distributions on infinite sequences of output firing patterns.
- Define a problem to be solved by networks: a mapping from infinite sequences of input firing patterns to sets of distributions on infinite sequences of output firing patterns.
- Composition of networks $\mathcal{N}_1 \circ \mathcal{N}_2$.
- Composition of problems, $\mathcal{P}_1 \circ \mathcal{P}_2$.
- Theorem: If $\mathcal{N}_1$ solves $\mathcal{P}_1$ and $\mathcal{N}_2$ solves $\mathcal{P}_2$ then $\mathcal{N}_1 \circ \mathcal{N}_2$ solves $\mathcal{P}_1 \circ \mathcal{P}_2$.



6. Discussion

- Goal: Understand how computation is performed in biological neural networks, in terms of distributed algorithms.
- Progress so far:
  - Biologically-inspired Stochastic Spiking Neural Network model.
  - Winner-Take-All, networks and lower bounds.
  - Similarity Testing and Compression, networks and lower bounds.
  - Indexing (NeoroRAM) sub-network.
- Issues: Role of inhibition, randomness, indexing,…
- Interesting technical results, may say something about biology.

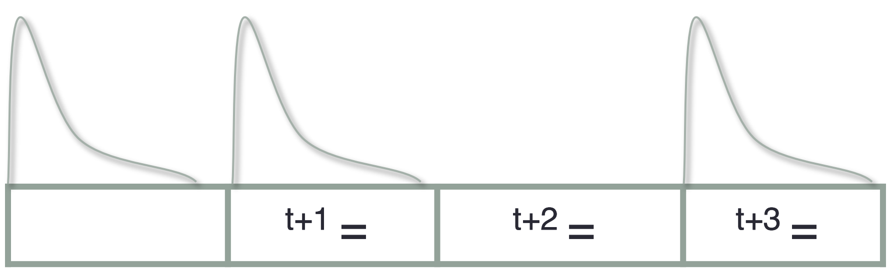
Future Work

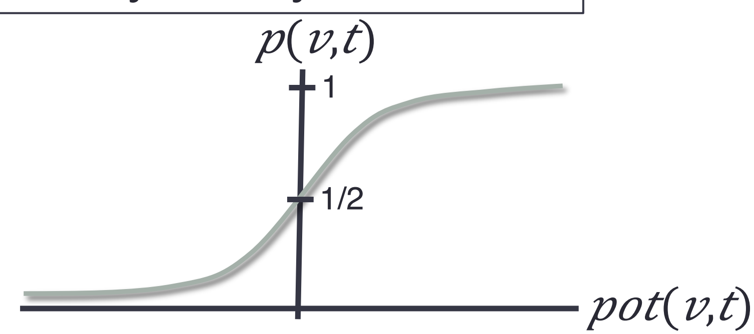
- Model:
  - Other biological features: Refractory period, cell memory, less synchrony, changing synapse weights,…
  - Theoretical variations: Other activation functions besides sigmoid, memory, firing rates,…
  - Comparative power of different models
  - Composition, levels of abstraction
- Algorithms:
  - WTA, sampling, indexing, and many other primitives
- Fault-tolerance
- Changing networks, learning
- Role of randomness in neural computation
- Building neural solutions for complex problems from solutions for simpler problems.

Future Work: The Model

- Consider other biologically-relevant features: Refractory period, cell memory, less synchrony, changing synapse weights,…
- Theoretical variations:
  - Consider other activation functions besides the sigmoid.
  - Consider more elaborate state than just firing status; firing rates.
  - Learning
- Comparative power of different models
- Compositional theory
- Levels of abstraction

Inspired by distributed algorithms and concurrency theory





Future Work: Algorithms

- Winner-Take-All:
  - $k-$WTA, electing k instead of just 1 output.
  - WTA with non-binary or varying inputs, selecting the strongest input, or the input with the highest firing rate.
  - Applications of WTA to solve other problems (attention, learning, neural coding…)
- Random sampling, indexing:
  - Simpler indexing circuits, more general lower bounds.
  - Applications of random sampling in solving other neural problems (estimating firing activity, estimating differences between firing patterns, exploring memories,…)
- Other primitives:
  - Other binary vector problems, computing functions, synchronization problems,…
  - Network designs, lower bounds, computational tradeoffs.

More Future Work

- Network changes and learning:
  - Hebbian-style modification of weights
  - Define model, study problems (memory formation, concept association, renaming and sparse coding, classification,…)
  - Do network mechanisms like ours arise via learning or are they preprogrammed? Or a combination?
- The role of randomness in neural computation:
  - Symmetry-breaking, similarity testing, compression,…
  - In general, where/how does randomness help?
- Connections with linear algebra
- Fault-tolerance
- Building solutions of complex problems from solutions for simple problems (compositional theory for computation in SNNs).

Thanks!