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.
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

• \( v^{\uparrow t} = 1 \) if and only if neuron \( v \) spikes at time \( t \).

\[
v^{\uparrow t} = 1 \\
v_{t+1} = 1 \\
v_{t+2} = 0 \\
v_{t+3} = 1
\]

• \( \text{pot}(v, t) = \sum_{u} u^{\uparrow t-1} w(u, v) - b(v) \)

• \( \Pr[v^{\uparrow t} = 1] = \frac{1}{1 + e^{\uparrow - \text{pot}(v, t)}} \)
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** $\mathcal{C}$: Assigns a firing state, 0 or 1, to each neuron, where $\mathcal{C}(u) = 1$ means it’s firing and $\mathcal{C}(u) = 0$ means it’s not.

- **Execution** $\alpha = \mathcal{C}^{\uparrow 0}, \mathcal{C}^{\uparrow 1}, \mathcal{C}^{\uparrow 2}, \ldots$, a sequence of configurations.
- $u^{\uparrow t} = \mathcal{C}^{\uparrow t}(u)$ 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 $f: \{0,1\}^n \rightarrow \{0,1\}^m$ (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


![WTA Circuit Diagram]
Winner-Take-All: $WTA(n, t_{\downarrow c}, t_{\downarrow s}, \delta)$

- $n$ 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 $t_{\downarrow c}$, to a single firing output, which corresponds to a firing input, and then
  - Remains stable for a long time $t_{\downarrow s}$.

- 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** $a\downarrow s$:
  - 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** $a\downarrow c$:
  - 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 $\frac{1}{2}$.
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 $t \leq \log n$ such that exactly one output fires at time $t$.
- Moreover, after time $t$, with high probability, this selected output continues to fire for a long time $t\downarrow s$.
- Meanwhile, only inhibitor $a\downarrow s$ 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 $\gamma$

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

$\begin{align*}
\gamma y_1 + \gamma y_2 + \gamma y_3 &= s, \\
(\gamma - \gamma) y_1 + (\gamma - \gamma) y_2 + (\gamma - \gamma) y_3 &= c, \\
2\gamma y_1 + 3\gamma y_2 + 3\gamma y_3 &= 0.
\end{align*}$

$\gamma b = 3\gamma$  $b = 0.5\gamma$  $b = 1.5\gamma$

$\gamma$  $-\gamma$  $\gamma$
Our Main Theorem

**Theorem 1:** Assume $\gamma \geq c \log(n \ t\downarrow s / \delta)$. Then starting from any state, with probability $\geq 1 - \delta$, the network converges, within time $t\downarrow c \approx c \log n \log (1/\delta)$, to a single firing output corresponding to a firing input, and remains stable for time $t\downarrow s$. 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):

  $C, C, C, ...$

• Duration $t↓s$ is exponential in the weighting factor $\gamma$. 
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 $\geq 2$ 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 $\gamma \geq c \log (nt \downarrow s / \delta)$, then the network solves $WTA(n, t \downarrow c, t \downarrow s, \delta)$, with $t \downarrow c \approx c \log n \log (1/\delta)$. 

---

The diagram illustrates the network configuration with labels $x_1, x_2, x_3, \ldots, x_n$, $y_1, y_2, y_3, \ldots, y_n$, $z_s$, and $z_c$. The labels $b = 0.5\gamma$, $b = 1.5\gamma$, and $b = 3\gamma$ are shown, along with directed edges representing the connectivity and influence between these nodes.
Faster Solution with More Inhibitors

• Uses one stability inhibitor, plus $\kappa$ 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.
Theorem 5: Assume $\gamma \geq c \log(n t \downarrow s / \delta)$. Then from any state, with prob $\geq 1 - \delta$, the network converges, within time $t \downarrow c \approx c k \log 1/k n \log (1/\delta)$, to a single firing output corresponding to a firing input, and remains stable for time $t \downarrow s$. 

Faster Solution with More Inhibitors

$\begin{align*}
\begin{array}{cccc}
0 & 1 & 1 & 0 \\
\cdots & \cdots & \cdots & \cdots \\
\end{array}
\end{align*}$
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

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 $X^\downarrow 1$ and $X^\downarrow 2$ match at those positions.

• If $X^\downarrow 1 = X^\downarrow 2$, then $S^\downarrow 1 = S^\downarrow 2$. If $d(X^\downarrow 1, X^\downarrow 2) \geq \epsilon n$, then with high probability, $S^\downarrow 1 \neq S^\downarrow 2$. 
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 $y$. 

![Diagram of the Indexing Network (Neuro-RAM)]
Similarity Testing with Neuro-RAM

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

\[ \begin{align*}
X_{\downarrow 1} & = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} \\
X_{\downarrow 2} & = \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix}
\end{align*} \]

\[ \begin{align*}
Z_{\downarrow 1} & = \begin{bmatrix} 0 & 1 \\ 1 & \vdots \end{bmatrix} \\
Z_{\downarrow k} & = \begin{bmatrix} 1 & 1 \end{bmatrix}
\end{align*} \]

\[ \begin{align*}
NR_{\downarrow 1,1} & = \ldots \\
NR_{\downarrow 2,1} & = \ldots \\
NR_{\downarrow 1,k} & = \ldots \\
NR_{\downarrow 2,k} & = \ldots 
\end{align*} \]

\[ = \ldots \]

\[ \wedge \]
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 $t \leq \sqrt{n}$, 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 \( X \downarrow i \) using first half of \( Z \).
- **Step 2:** Select the desired index inside the bucket \( X \downarrow i \) 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 $\Omega(n/t \log \log n)$ 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 $\log \sqrt[4]{2} n$ 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.
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
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 \( \Omega(n/t \log \sqrt{2} n) \) 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 $\epsilon$-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:

\[
\begin{align*}
X \downarrow_{2} & = X \downarrow_{1} \quad (1,1) = (2,1) \\
NR \downarrow_{1,1} & = NR \downarrow_{2,1} \\
Z \downarrow_{1} & = Z \downarrow_{k} \quad (1,1) = (k,1)
\end{align*}
\]
Composing Spiking Neural Networks

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

![Diagram with 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 \( \mathcal{N} \downarrow 1 \circ \mathcal{N} \downarrow 2 \).
- Composition of problems, \( \mathcal{P} \downarrow 1 \circ \mathcal{P} \downarrow 2 \).
- **Theorem:** If \( \mathcal{N} \downarrow 1 \) solves \( \mathcal{P} \downarrow 1 \) and \( \mathcal{N} \downarrow 2 \) solves \( \mathcal{P} \downarrow 2 \) then \( \mathcal{N} \downarrow 1 \circ \mathcal{N} \downarrow 2 \) solves \( \mathcal{P} \downarrow 1 \circ \mathcal{P} \downarrow 2 \).
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!