SPIKING NEURAL NETWORKS: AN ALGORITHMIC PERSPECTIVE

Nancy Lynch, **Cameron Musco** and Merav Parter

Massachusetts Institute of Technology, EECS.
Weizmann Institute of Science
BDA 2017.

Based on work in:

- *Computational Tradeoffs in Biological Neural Networks: Self-Stabilizing Winner-Take-All Networks*. ITCS 2017.
- *Neuro-RAM Unit with Applications to Similarity Testing and Compression in Spiking Neural Networks*. DISC 2017.

Based on work in:

- *Computational Tradeoffs in Biological Neural Networks: Self-Stabilizing Winner-Take-All Networks*. ITCS 2017.
- *Neuro-RAM Unit with Applications to Similarity Testing and Compression in Spiking Neural Networks*. DISC 2017.

Full versions are available at: `cameronmusco.com`

High Level Goal: Understand how computation is performed in biological neural networks at an algorithmic level.

High Level Goal: Understand how computation is performed in biological neural networks at an algorithmic level.

- **Biological Features:** Noisy threshold gates, spiking neurons, restricted connectivity structures/edge weights

NEURAL CIRCUITS FROM AN ALGORITHMIC VIEW

High Level Goal: Understand how computation is performed in biological neural networks at an algorithmic level.

- **Biological Features:** Noisy threshold gates, spiking neurons, restricted connectivity structures/edge weights
- **Tasks:** Select single neuron out of group with strongest output signal, test similarity of input patterns, etc.

High Level Goal: Understand how computation is performed in biological neural networks at an algorithmic level.

- **Biological Features:** Noisy threshold gates, spiking neurons, restricted connectivity structures/edge weights
- **Tasks:** Select single neuron out of group with strongest output signal, test similarity of input patterns, etc.

We focus on fixed networks and do not (yet) consider how they are learned. Our tasks are basic computational primitives rather than more complex pattern recognition goals.

GUIDING QUESTIONS

- How do biological features affect computability, runtime tradeoffs, and algorithm design?

GUIDING QUESTIONS

- How do biological features affect computability, runtime tradeoffs, and algorithm design?
- Is there interesting theory beyond what is known for other well studied models of computation. E.g. deterministic threshold circuits (perceptrons), Boltzmann machines, distributed networks w/ message passing, etc.?

GUIDING QUESTIONS

- How do biological features affect computability, runtime tradeoffs, and algorithm design?
- Is there interesting theory beyond what is known for other well studied models of computation. E.g. deterministic threshold circuits (perceptrons), Boltzmann machines, distributed networks w/ message passing, etc.?
- Can this theory say anything interesting about computation in real neural networks?

GUIDING QUESTIONS

- How do biological features affect computability, runtime tradeoffs, and algorithm design?
- Is there interesting theory beyond what is known for other well studied models of computation. E.g. deterministic threshold circuits (perceptrons), Boltzmann machines, distributed networks w/ message passing, etc.?
- Can this theory say anything interesting about computation in real neural networks?
    - E.g. role of noise and randomness, roll of inhibition and excitation, recurring design patterns.

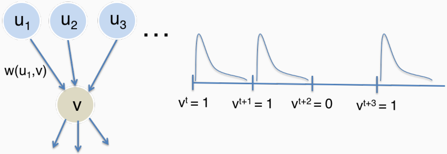
GUIDING QUESTIONS

- How do biological features affect computability, runtime tradeoffs, and algorithm design?
- Is there interesting theory beyond what is known for other well studied models of computation. E.g. deterministic threshold circuits (perceptrons), Boltzmann machines, distributed networks w/ message passing, etc.?
- Can this theory say anything interesting about computation in real neural networks?
  - E.g. role of noise and randomness, roll of inhibition and excitation, recurring design patterns.
- Significantly influenced by the work of Wolfgang Maass on the theory of spiking neural networks.

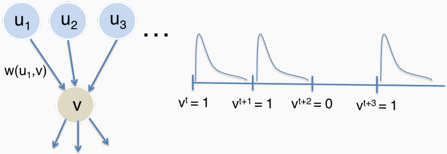
STOCHASTIC SPIKING NEURAL NETWORKS

$v^t = 1$ if neuron v spikes at time t, $v^t = 0$ otherwise.

$v^t = 1$ if neuron v spikes at time t, $v^t = 0$ otherwise.

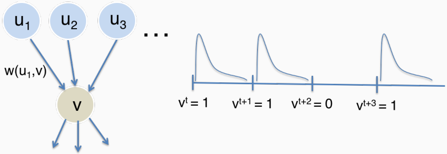


$v^t = 1$ if neuron v spikes at time t, $v^t = 0$ otherwise.



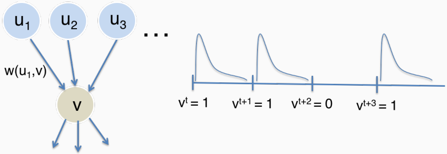
$$pot(v,t) = \sum_{u \in N} u^{t-1} \cdot w(u,v)$$

$v^t = 1$ if neuron v spikes at time t, $v^t = 0$ otherwise.



$$pot(v,t) = \sum_{u \in N} u^{t-1} \cdot w(u,v) - b(v)$$

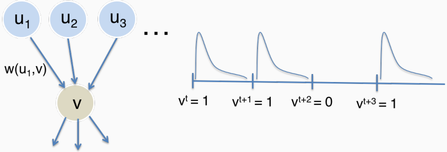
$v^t = 1$ if neuron v spikes at time t, $v^t = 0$ otherwise.



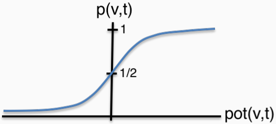
$$pot(v,t) = \sum_{u \in N} u^{t-1} \cdot w(u,v) - b(v) \quad \Pr\left[v^t = 1\right] = \frac{1}{1 + e^{-pot(v,t)}}$$

STOCHASTIC SPIKING NEURAL NETWORKS

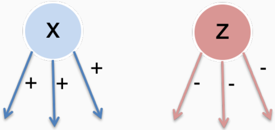
$v^t = 1$ if neuron v spikes at time t, $v^t = 0$ otherwise.



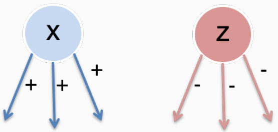
$$pot(v,t) = \sum_{u \in N} u^{t-1} \cdot w(u,v) - b(v) \qquad \Pr\left[v^t = 1\right] = \frac{1}{1 + e^{-pot(v,t)}}$$



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



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



Ignore many other biological features. E.g. refractory period, spike propagation delay, memory, noise on synapses etc. Some can be simulated in our model.

COMPUTATIONAL PROBLEMS IN OUR MODEL

- n input neurons X each either always firing or not firing.
- m output neurons Y.

COMPUTATIONAL PROBLEMS IN OUR MODEL

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

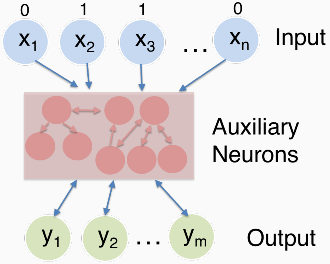
COMPUTATIONAL PROBLEMS IN OUR MODEL

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

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

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

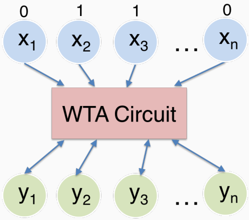
Goal: Design a compact network that rapidly converges to some output firing pattern $Y^t \in f(X)$ with high probability.



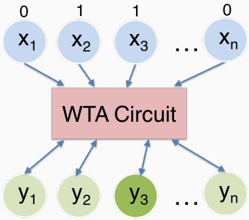
Questions so far?

Example Problem 1

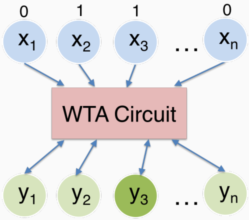
Binary WTA problem: Want to converge to a single firing output, which corresponds to a firing input.



Binary WTA problem: Want to converge to a single firing output, which corresponds to a firing input.



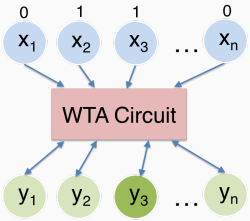
Binary WTA problem: Want to converge to a single firing output, which corresponds to a firing input.



- Neural leader election. Very heavily studied in computational neuroscience.

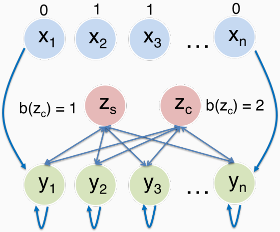
WINNER-TAKE-ALL (WTA)

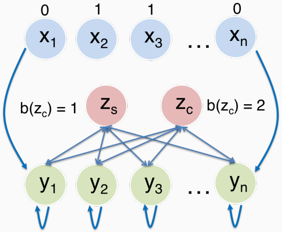
Binary WTA problem: Want to converge to a single firing output, which corresponds to a firing input.



- Neural leader election. Very heavily studied in computational neuroscience.
- Used in perceptual attention, competitive learning, etc. Powerful 'nonlinear' primitive [Maass '99]

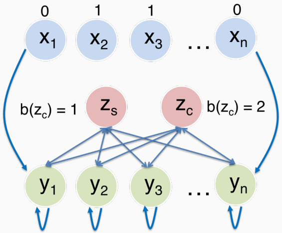
SIMPLE SOLUTION WITH TWO INHIBITORS



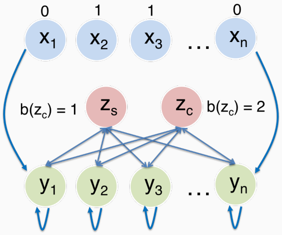


Main idea: Inhibitors facilitate competition (or lateral inhibition) between inputs, leading to a single 'winner'.

SIMPLE SOLUTION WITH TWO INHIBITORS

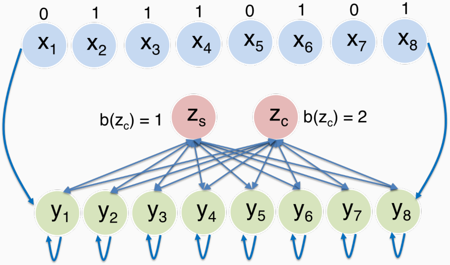


- Stability inhibitor z_s fires whenever there are ≥ 1 competing outputs and prevents any output that didn't fire at time t from firing at time $t + 1$.



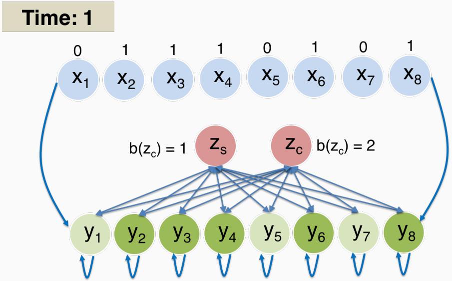
- Stability inhibitor z_s fires whenever there are ≥ 1 competing outputs and prevents any output that didn't fire at time t from firing at time $t+1$.
- Convergence inhibitor z_c fires whenever there are ≥ 2 competing outputs and causes any competing output to stop firing at time $t+1$ with probability $1/2$.

SIMPLE SOLUTION WITH TWO INHIBITORS



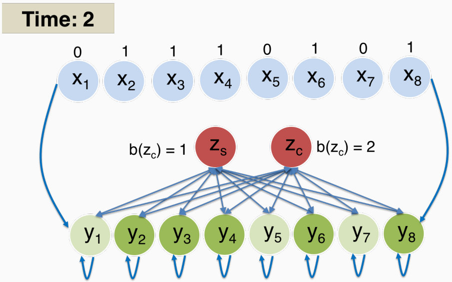
- Roughly 1/2 of competing outputs stop firing at each time step. With constant probability there is some time $t \leq \log n$ such that exactly one output fires at time t.
- After time t, this distinguished output continues to fire. Just z_s fires, preventing all other outputs from firing.

SIMPLE SOLUTION WITH TWO INHIBITORS



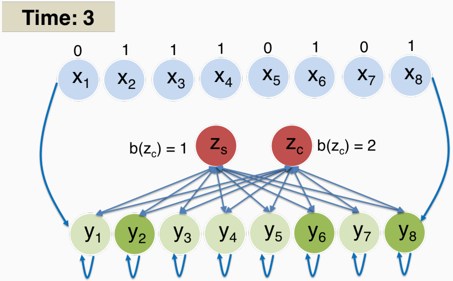
- Roughly 1/2 of competing outputs stop firing at each time step. With constant probability there is some time $t \leq \log n$ such that exactly one output fires at time t.
- After time t, this distinguished output continues to fire. Just z_s fires, preventing all other outputs from firing.

SIMPLE SOLUTION WITH TWO INHIBITORS



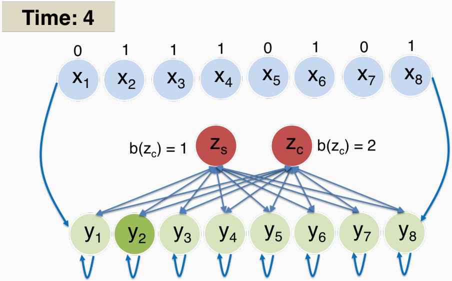
- Roughly 1/2 of competing outputs stop firing at each time step. With constant probability there is some time $t \leq \log n$ such that exactly one output fires at time t.
- After time t, this distinguished output continues to fire. Just z_s fires, preventing all other outputs from firing.

SIMPLE SOLUTION WITH TWO INHIBITORS

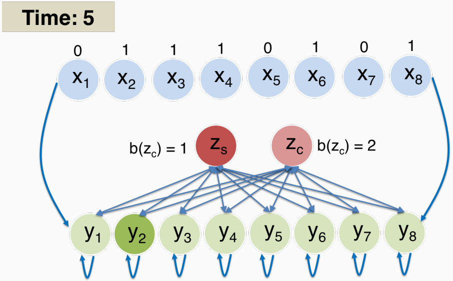


- Roughly 1/2 of competing outputs stop firing at each time step. With constant probability there is some time $t \leq \log n$ such that exactly one output fires at time t.
- After time t, this distinguished output continues to fire. Just z_s fires, preventing all other outputs from firing.

SIMPLE SOLUTION WITH TWO INHIBITORS



- Roughly 1/2 of competing outputs stop firing at each time step. With constant probability there is some time $t \leq \log n$ such that exactly one output fires at time t.
- After time t, this distinguished output continues to fire. Just z_s fires, preventing all other outputs from firing.



- Roughly 1/2 of competing outputs stop firing at each time step. With constant probability there is some time $t \leq \log n$ such that exactly one output fires at time t.
- After time t, this distinguished output continues to fire. Just z_s fires, preventing all other outputs from firing.

Upshot: Convergence to valid winner in $O(\log n)$ time in expectation.

SIMPLE SOLUTION WITH TWO INHIBITORS

Upshot: Convergence to valid winner in $O(\log n)$ time in expectation.

- More than two inhibitors can be used to give faster convergence (see full paper for results characterizing this tradeoff.)

Upshot: Convergence to valid winner in $O(\log n)$ time in expectation.

- More than two inhibitors can be used to give faster convergence (see full paper for results characterizing this tradeoff.)
- Can be used to solve non-binary WTA. Goal here is to select the input with the highest firing rate.

Two features of any near-optimal WTA circuit (show up in both upper and corresponding lower bounds).

Two features of any near-optimal WTA circuit (show up in both upper and corresponding lower bounds).

1. Inhibitors fall into two classes – convergence and stability neurons.

COMPUTATIONAL PRINCIPLES

Two features of any near-optimal WTA circuit (show up in both upper and corresponding lower bounds).

1. Inhibitors fall into two classes – convergence and stability neurons. Inhibition is often viewed as a stability mechanism in the brain. In our networks, it has two roles: maintaining stability and driving computation.

COMPUTATIONAL PRINCIPLES

Two features of any near-optimal WTA circuit (show up in both upper and corresponding lower bounds).

1. Inhibitors fall into two classes – convergence and stability neurons. Inhibition is often viewed as a stability mechanism in the brain. In our networks, it has two roles: maintaining stability and driving computation.
2. Inhibitors behave nearly deterministically. Randomness is used solely for symmetry breaking between outputs.

COMPUTATIONAL PRINCIPLES

Two features of any near-optimal WTA circuit (show up in both upper and corresponding lower bounds).

1. Inhibitors fall into two classes – convergence and stability neurons. Inhibition is often viewed as a stability mechanism in the brain. In our networks, it has two roles: maintaining stability and driving computation.
2. Inhibitors behave nearly deterministically. Randomness is used solely for symmetry breaking between outputs. Highlights dual nature of randomness – can be a powerful computational resource but can also slow down computation by leading to noisy behavior.

Example Problem 2

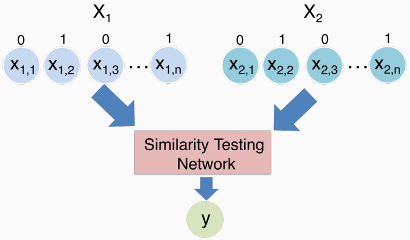
Similarity Testing: Given two input firing patterns X_1 and X_2, distinguish whether $X_1 = X_2$ or if they are far from being equal. I.e. if $d(X_1, X_2) \geq \epsilon n$.

Similarity Testing: Given two input firing patterns X_1 and X_2, distinguish whether $X_1 = X_2$ or if they are far from being equal. I.e. if $d(X_1, X_2) \geq \epsilon n$.

- After convergence, the output neuron should fire continuously if the inputs are equal and not fire if they are far from equal.

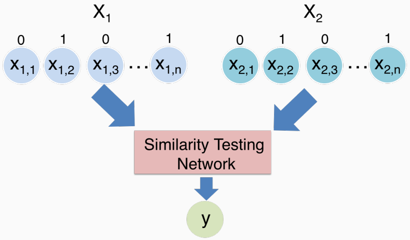
Similarity Testing: Given two input firing patterns X_1 and X_2, distinguish whether $X_1 = X_2$ or if they are far from being equal. I.e. if $d(X_1, X_2) \geq \epsilon n$.

- After convergence, the output neuron should fire continuously if the inputs are equal and not fire if they are far from equal.



Similarity Testing: Given two input firing patterns X_1 and X_2, distinguish whether $X_1 = X_2$ or if they are far from being equal. I.e. if $d(X_1, X_2) \geq \epsilon n$.

- After convergence, the output neuron should fire continuously if the inputs are equal and not fire if they are far from equal.

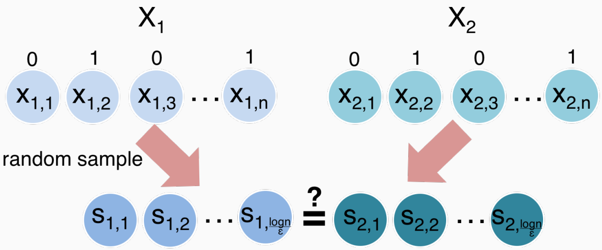


- Natural sub-problem for pattern recognition and other tasks.

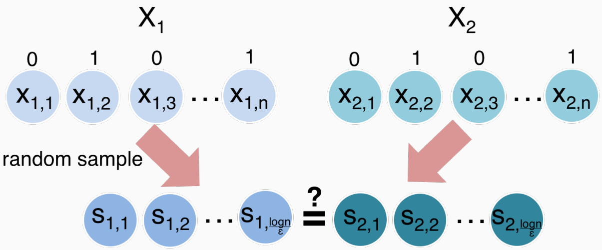
Simple (non-neural) sublinear time algorithm:

Simple (non-neural) sublinear time algorithm: Sample $O\left(\frac{\log n}{\epsilon}\right)$ random positions and check if X_1 and X_2 match at these positions.

Simple (non-neural) sublinear time algorithm: Sample $O\left(\frac{\log n}{\epsilon}\right)$ random positions and check if X_1 and X_2 match at these positions.



Simple (non-neural) sublinear time algorithm: Sample $O\left(\frac{\log n}{\epsilon}\right)$ random positions and check if X_1 and X_2 match at these positions.



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

HOW TO IMPLEMENT NEURALLY?

How to implement neurally?

- Equality check of S_1 and S_2 is straightforward.

HOW TO IMPLEMENT NEURALLY?

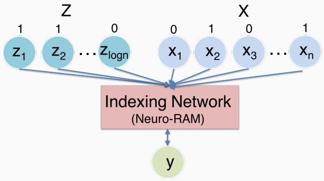
- Equality check of S_1 and S_2 is straightforward.
- Sampling random positions requires an indexing module

how to implement neurally?

- Equality check of S_1 and S_2 is straightforward.
- Sampling random positions requires an indexing module: given an index encoded by the firing pattern of a set of neurons, select the appropriate value of X_1 or X_2.

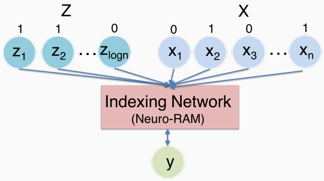
HOW TO IMPLEMENT NEURALLY?

- Equality check of S_1 and S_2 is straightforward.
- Sampling random positions requires an indexing module: given an index encoded by the firing pattern of a set of neurons, select the appropriate value of X_1 or X_2.



how to implement neurally?

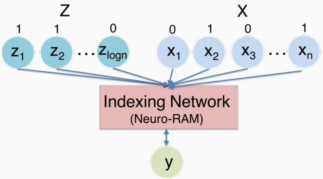
- Equality check of S_1 and S_2 is straightforward.
- Sampling random positions requires an indexing module: given an index encoded by the firing pattern of a set of neurons, select the appropriate value of X_1 or X_2.



- After convergence, the output neuron should fire continously if and only if $X(Z)$ is firing.

HOW TO IMPLEMENT NEURALLY?

- Equality check of S_1 and S_2 is straightforward.
- Sampling random positions requires an indexing module: 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 continously if and only if $X(Z)$ is firing.
- Simulates an excitatory connection from $X(Z)$ to y.

At first glance indexing may not seem very ‘neural.’

At first glance indexing may not seem very ‘neural.’

More neural motivation:

At first glance indexing may not seem very ‘neural.’

More neural motivation:

- Uses information contained in a small set of neurons (the index) to access information from a much larger data store X.

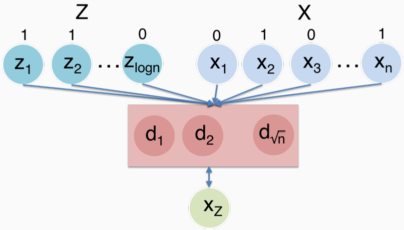
At first glance indexing may not seem very 'neural.'

More neural motivation:

- Uses information contained in a small set of neurons (the index) to access information from a much larger data store X.
- This seems to be an important primitive in many computations beyond our similarity testing application. E.g. a smell or sight triggering a memory.

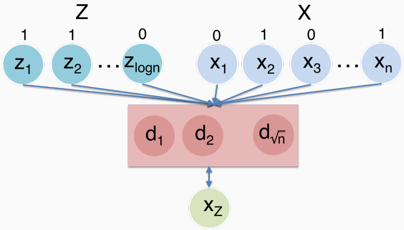
Theorem

For any $t \leq \sqrt{n}$, there is an SNN solving the indexing problem with $O(n/t)$ auxiliary neurons that converges in t time steps with high probability. For $t = \sqrt{n}$, the circuit uses $O(\sqrt{n})$ auxiliary neurons.



Theorem

For any $t \leq \sqrt{n}$, there is an SNN solving the indexing problem with $O(n/t)$ auxiliary neurons that converges in t time steps with high probability. For $t = \sqrt{n}$, the circuit uses $O(\sqrt{n})$ auxiliary neurons.



- Gives an $O\left(\frac{\sqrt{n}\log n}{\epsilon}\right)$ (i.e., sublinear) sized circuit for the similarity testing problem.

- Lower bound via VC-dimension-based arguments show that convergence time cannot be improved by more than a $\log^2 n$ factor.

- Lower bound via VC-dimension-based arguments show that convergence time cannot be improved by more than a $\log^2 n$ factor.
- The same upper bound can be achieved with linear threshold gates (i.e. perceptrons)

LOWER BOUND

- Lower bound via VC-dimension-based arguments show that convergence time cannot be improved by more than a $\log^2 n$ factor.
- The same upper bound can be achieved with linear threshold gates (i.e. perceptrons)
- So our spiking model, and importantly the availability of randomness, does not help much for this problem.

- Lower bound via VC-dimension-based arguments show that convergence time cannot be improved by more than a $\log^2 n$ factor.
- The same upper bound can be achieved with linear threshold gates (i.e. perceptrons)
- So our spiking model, and importantly the availability of randomness, does not help much for this problem.
- Also separates our model from sigmoidal gates with real valued outputs which can implement indexing with $O(\sqrt{n})$ neurons converging in $O(1)$ steps.

- Our result can be seen in two ways:

TAKEAWAYS

- Our result can be seen in two ways:
  - Indexing can be implemented with a compact spiking networks.
  - Any compact indexing network must converge slowly, and thus seems somewhat unlikely as a neural implementation.

TAKEAWAYS

- Our result can be seen in two ways:
  - Indexing can be implemented with a compact spiking networks.
  - Any compact indexing network must converge slowly, and thus seems somewhat unlikely as a neural implementation.
- Is general indexing machinery actually implemented in the brain?

TAKEAWAYS

- Our result can be seen in two ways:
  - Indexing can be implemented with a compact spiking networks.
  - Any compact indexing network must converge slowly, and thus seems somewhat unlikely as a neural implementation.
- Is general indexing machinery actually implemented in the brain?
- Our similarity testing algorithm is a simple application of randomized compression.

- Our result can be seen in two ways:
  - Indexing can be implemented with a compact spiking networks.
  - Any compact indexing network must converge slowly, and thus seems somewhat unlikely as a neural implementation.
- Is general indexing machinery actually implemented in the brain?
- Our similarity testing algorithm is a simple application of randomized compression.
  - Other randomized compression schemes like Johnson-Lindenstrauss projection have been considered as possible neural algorithms.
  - To what extent are these schemes implemented via random connectivity and to what extent do they require indexing operations?

Concrete Next Steps:

Concrete Next Steps:

- k-WTA, better understanding of WTA with non-binary inputs, sparse coding/renaming

Concrete Next Steps:

- *k*-WTA, better understanding of WTA with non-binary inputs, sparse coding/renaming
- More biologically plausible models: refractory period, history, asynchrony, learning and dynamic synapse weights

Concrete Next Steps:

- k-WTA, better understanding of WTA with non-binary inputs, sparse coding/renaming
- More biologically plausible models: refractory period, history, asynchrony, learning and dynamic synapse weights
- Theoretical abstractions that let us handle biological complexity.

Concrete Next Steps:

- k-WTA, better understanding of WTA with non-binary inputs, sparse coding/renaming
- More biologically plausible models: refractory period, history, asynchrony, learning and dynamic synapse weights
- Theoretical abstractions that let us handle biological complexity.
- What features of our model can be generalized? E.g. can we prove results for a wider class of activation functions beyond the sigmoid?

High-Level Directions

High-Level Directions

- How do networks for simple computational primitives arise? Can they be ‘learned’? Are they preprogramed in some way?

High-Level Directions

- How do networks for simple computational primitives arise? Can they be 'learned'? Are they preprogramed in some way?
- How do fixed network motifs such as WTA and similarity testing circuits interact with more flexible 'learning' networks?

High-Level Directions

- How do networks for simple computational primitives arise? Can they be 'learned'? Are they preprogramed in some way?
- How do fixed network motifs such as WTA and similarity testing circuits interact with more flexible 'learning' networks?
- More generally, would like to develop a theory for composing spiking neural networks to solve complex problems.

Thanks!