Neuro-RAM Unit with Applications to Similarity Testing and Compression in Spiking Neural Networks*

Nancy Lynch[1], Cameron Musco[2], and Merav Parter[3]

1 **CSAIL, MIT, Cambridge, USA**
`lynch@csail.mit.edu`
2 **CSAIL, MIT, Cambridge, USA**
`cnmusco@mit.edu`
3 **Weizmann Institute, Rehovot, Israel**
`merav.parter@weizmann.ac.il`

Abstract

We study distributed algorithms implemented in a simplified biologically inspired model for *stochastic spiking neural networks*. We focus on tradeoffs between computation time and network complexity, along with the role of noise and randomness in efficient neural computation.

It is widely accepted that neural spike responses, and neural computation in general, is inherently stochastic. In recent work, we explored how this stochasticity could be leveraged to solve the 'winner-take-all' leader election task. Here, we focus on using randomness in neural algorithms for similarity testing and compression. In the most basic setting, given two n-length patterns of firing neurons, we wish to distinguish if the patterns are equal or ϵ-far from equal.

Randomization allows us to solve this task with a very compact network, using $O\left(\frac{\sqrt{n}\log n}{\epsilon}\right)$ auxiliary neurons, which is sublinear in the input size. At the heart of our solution is the design of a t-round neural random access memory, or indexing network, which we call a *neuro-RAM*. This module can be implemented with $O(n/t)$ auxiliary neurons and is useful in many applications beyond similarity testing – e.g., we discuss its application to compression via random projection.

Using a VC dimension-based argument, we show that the tradeoff between runtime and network size in our neuro-RAM is near optimal. To the best of our knowledge, we are the first to apply these techniques to stochastic spiking networks. Our result has several implications – since our neuro-RAM can be implemented with deterministic threshold gates, it shows that, in contrast to similarity testing, randomness does not provide significant computational advantages for this problem. It also establishes a separation between feedforward networks whose gates spike with sigmoidal probabilities, and well-studied deterministic sigmoidal networks, whose gates output real number sigmoidal values, and which can implement a neuro-RAM much more efficiently.

1998 ACM Subject Classification F.1.1 Models of Computation – Self-modifying machines (e.g., neural networks), F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases spiking neural networks, biological distributed algorithms, circuit design

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.33

1 Introduction

Biological neural networks are arguably the most fascinating distributed computing systems in our world. However, while studied extensively in the fields of computational neuroscience

* Full version available at `https://arxiv.org/abs/1706.01382`

31st International Symposium on Distributed Computing (DISC 2017).
Editor: Andréa W. Richa; Article No. 33; pp. 33:1–33:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and artificial intelligence, they have received little attention from a distributed computing perspective. Our goal is to study biological neural networks through the lens of distributed computing theory. We focus on understanding tradeoffs between computation time, network complexity, and the use of randomness in implementing basic algorithmic primitives, which can serve as building blocks for high level pattern recognition, learning, and processing tasks.

Spiking Neural Network (SNN) Model. We work with biologically inspired *spiking neural networks* (SNNs) [18, 19, 12, 15], in which neurons fire in discrete pulses in synchronous rounds, in response to a sufficiently high membrane potential. This potential is induced by spikes from neighboring neurons, which can have either an excitatory or inhibitory effect (increasing or decreasing the potential). As observed in biological networks, neurons are either strictly inhibitory (all outgoing edge weights are negative) or excitatory. As we will see, this restriction can significantly affect the power of these networks.

A key feature of our model is stochasticity – each neuron is a probabilistic threshold unit, spiking with probability given by applying a sigmoid function to its potential. While a rich literature focuses on deterministic circuits [21, 13] we employ a stochastic model as it is widely accepted that neural computation is stochastic [1, 24, 9].

Computational Problems in SNNs. We consider an n-bit binary input vector X, which represents the firing status of a set of input neurons. Given a (possibly multi-valued) function $f : \{0,1\}^n \to \{0,1\}^m$, we seek to design a network of spiking neurons that converges to an output vector $Z = f(X)$ (or any $Z \in f(X)$ if f is multi-valued) as quickly as possible using few auxiliary (non-input or output) neurons.

The number of auxiliary neurons used corresponds to the "node complexity" of the network [14]. Designing circuits with small node complexity has received a lot of attention – e.g., the work of [10] on PARITY and [3] on AC_0. Much less is known, however, on what is achievable in spiking neural networks. For most of the problems we study, there is a trivial solution that uses $\Theta(n)$ auxiliary neurons for inputs of size n. Hence, we primarily focus on designing *sublinear* size networks – with n^{1-c} auxiliary neurons for some c.

Past Work: WTA. Recently, we studied the 'winner-take-all' (WTA) leader election task in SNNs [17]. Given a set of firing input neurons, the network is required to converge to a single firing output – corresponding to the 'winning' input. In that work, we critically leveraged the noisy behavior of our spiking neuron model: randomness is key in breaking the symmetry between initially identical firing inputs.

This Paper: Similarity Testing and Compression. In this paper, we study the role of randomness in a different setting: for similarity testing and compression. Consider the basic similarity testing problem: given $X_1, X_2 \in \{0,1\}^n$, we wish to distinguish the case when $X_1 = X_2$ from the case when the Hamming distance between the vectors is large – i.e., $\mathrm{d_H}(X_1, X_2) \geq \epsilon n$ for some parameter ϵ. This problem can be solved very efficiently using randomness – it suffices to sample $O(\log n/\epsilon)$ indices and compare X_1 and X_2 at these positions to distinguish the two cases with high probability. Beyond similarity testing, similar compression approaches using random input subsampling or hashing can lead to very efficient routines for a number of data processing tasks.

1.1 A Neuro-RAM Unit

To implement the randomized similarity testing approach described above, and to serve as a foundation for other random compression methods in spiking networks, we design a basic *indexing module*, or random access memory, which we call a *neuro-RAM*. This module solves:

▶ **Definition 1** (Indexing). Given $X \in \{0,1\}^n$ and $Y \in \{0,1\}^{\log n}$ which is interpreted as an integer in $\{0, ..., n-1\}$, the indexing problem is to output the value of the Y^{th} bit of X[1].

Our neuro-RAM uses a sublinear number of auxiliary neurons and solves indexing with high probability on any input. We focus on characterizing the trade-off between the convergence time and network size of the neuro-RAM, giving nearly matching upper and lower bounds.

Generally, our results show that a compressed representation (e.g., the index Y) can be used to access a much larger datastore (e.g., X), using a very compact neural network. While binary indexing is not very 'neural' we can imagine similar ideas extending to more natural coding schemes used, for example, for memory retrieval, scent recognition, or other tasks.

Relation to Prior Work. Significant work has employed random synaptic connections between neurons – e.g., the Johnson-Lindenstrauss compression results of [2] and the work of Valiant [26]. While it is reasonable to assume that the initial synapses are random, biological mechanisms for changing connectivity (functional plasticity) act over relatively large time frames and cannot provide a new random sample of the network for each new input. In contrast, stochastic spiking neurons do provide fresh randomness to each computation. In general, transforming of a network with m possible random edges to a network with fixed edges and stochastic neurons requires $\Omega(m)$ auxiliary neurons and thus fails to fulfill our sublinearity goal, as there is typically at least one possible outgoing edge from each input. Our neuro-RAM can be thought of as improving the naive simulation – by reading a random entry of an input, we simulate a random edge from the specified neuron. Beyond similarity testing, we outline how our result can be used to implement Johnson-Lindenstrauss compression similar to [2] without assuming random connectivity.

1.2 Our Contributions

1.2.1 Efficient Neuro-RAM Unit

Our primary upper bound result is the following:

▶ **Theorem 2** (t-round Neuro-RAM). *For every integer $t \leq \sqrt{n}$, there is a (*recurrent*) SNN with $O(n/t)$ auxiliary neurons that solves the indexing problem in t rounds with high probability. In particular, there exists a neuro-RAM unit that contains $O(\sqrt{n})$ auxiliary neurons and solves the indexing problem in $O(\sqrt{n})$ rounds.*

Above, and throughout the paper 'with high probability' or w.h.p. to denotes with probability at least $1 - 1/n^c$ for some constant c. Theorem 2 is proven in Section 3.

Neuro-RAM Construction. The main idea is to first 'encode' the firing pattern of the input neurons X into the potentials of t neurons. These encoding neurons will spike with some probability dependent on their potential. However, simply recording the firing rates of the neurons to estimate this probability is too inefficient. Instead, we use a 'successive decoding

[1] Here, and throughout, for simplicity we assume n is a power of 2 so $\log n$ is an interger.

strategy', in which the firing rates of the encoding neurons are estimated at finer and finer levels of approximation, and adjusted through recurrent excitation or inhibition as decoding progresses. The strategy converges in $O(n/t)$ rounds – the smaller t is the more information is contained in the potential of a single neuron, and the longer decoding takes.

Theorem 2 shows a significant separation between our networks and traditional feedforward circuits where significantly sublinear sized indexing units are not possible.

▶ **Fact 3** (See Lower Bounds in [16]). *A circuit solving the indexing problem that consists of AND/OR gates connected in a feedforward manner requires $\Theta(n)$ gates. A feedforward circuit using linear threshold gates requires $\Theta(n/\log n)$ gates.*

We note, however, that our indexing mechanism *does not exploit the randomness of the spiking neurons*, and in fact can also be implemented with deterministic linear threshold gates. Thus, the separation between Theorem 2 and Fact 3 is entirely due to the recurrent (non-feedforward) layout of our network. Since any recurrent network using $O(m)$ neurons and converging in t rounds can be 'unrolled' into a feedforward circuit using $O(mt)$ neurons, Fact 3 shows that the tradeoff between network size and runtime in Theorem 2 is optimal up to a $\log n$ factor, if we use our spiking neurons in this restricted way. However, it does not rule out improvements using more sophisticated randomized strategies.

1.2.2 Lower Bound for Neuro-RAM in Spiking Networks

Surprisingly, we are able to show that despite the restricted way in which we use our spiking neuron model, significant improvements are not possible:

▶ **Theorem 4** (Lower Bound for Neuro-RAM in SNNs). *Any SNN that solves indexing in t rounds with high probability in our model must use at least $\Omega\left(\frac{n}{t\log^2 n}\right)$ auxiliary neurons.*

Theorem 4, whose proof is in Section 4, shows that the tradeoff in Theorem 2 is within a $\log^2 n$ factor of optimal. It matches the lower bound of Fact 3 for deterministic threshold gates up to a $\log n$ factor, showing that there is not a significant difference in the power of stochastic neurons and deterministic gates in solving indexing.

Reduction from SNNs to Deterministic Circuits. We first argue that the output distribution of any SNN is identical to the output distribution of an algorithm that first chooses a deterministic threshold circuit from some distribution and then applies it to the input. This is a powerful observation as it lets us apply Yao's principle: an SNN lower bound can be shown via a lower bound for deterministic circuits on any input distribution [27].

Deterministic Circuit Lower Bound via VC Dimension. We next show that any deterministic circuit that succeeds with high probability on uniform random inputs cannot be too small. The bound is via a VC dimension-based argument, which extends the work of [16]. As far as we are aware, we are the first to give a VC dimension-based lower bound for probabilistic and biologically plausible networks and we hope our work significantly expands the toolkit for proving lower bounds in this area. In contrast to our lower bounds on the WTA problem [17], which rely on indistinguishability arguments based on network structure, our new techniques allow us to give more general bounds for any network architecture.

Separation of Network Models. Aside from showing that randomness does not give significant advantages in constructing a neuro-RAM (contrasting with its importance in WTA and similarity testing), our proof of Theorem 4 establishes a separation between feedforward spiking networks and deterministic *sigmoidal circuits*. Our neurons spike with probability computed as a sigmoid of their membrane potential. In sigmoidal circuits, neurons output real numbers, equivalent to our spiking probabilities. A neuro-RAM can be implemented very efficiently in these networks:

▶ **Fact 5** (See [16], along with [19] for similar bounds). *There is a feedforward sigmoidal circuit solving the indexing problem using $O(\sqrt{n})$ gates.*[2]

In contrast, via an unrolling argument, the proof of Theorem 4 shows that any feedforward spiking network requires $\Omega\left(\frac{n}{\log^2 n}\right)$ gates to solve indexing with high probability.

It has been shown that feedforward sigmoidal circuits can significantly outperform standard feedforward linear threshold circuits [20, 16]. However, previously it was not known that restricting gates to spike with a sigmoid probability function rather than output the real value of this function significantly affected their power. Our lower bound, along with Fact 5, shows that in some cases it does. This separation highlights the importance of modeling spiking neuron behavior in understanding complexity tradeoffs in neural computation.

1.2.3 Applications to Randomized Similarity Testing and Compression

As discussed, our neuro-RAM is widely applicable to algorithms that require random sampling of inputs. In Section 5 we discuss our main application, to similarity testing – i.e., testing if $X_1 = X_2$ or if $\mathrm{d_H}(X_1, X_2) \geq \epsilon n$. It is easy to implement an exact equality tester using $\Theta(n)$ auxiliary neurons. Alternatively, one can solve exact equality with three auxiliary neurons using mixed positive and negative edge weights for the outgoing edges of inputs. However this is not biologically plausible – neurons typically have either all positive (excitatory) or all negative (inhibitory) outgoing edges, a restriction included in our model. Designing sublinear sized exact equality testers under this restriction seems difficult – simulating the three neuron solution requires at least $\Theta(n)$ auxiliary neurons – $\Theta(1)$ for each input.

By relaxing to similarity testing and applying our neuro-RAM, we can achieve sublinear sized networks. We can use $\Theta(\log n/\epsilon)$ neuro-RAMs, each with $O(\sqrt{n})$ auxiliary neurons to check equality at $\Theta(\log n/\epsilon)$ random positions of X_1 and X_2 distinguishing if $X_1 = X_2$ or if $\mathrm{d_H}(X_1, X_2) \geq \epsilon n$ with high probability. This is the first sublinear solution for this problem in the spiking neural networks. In Section 5, we discuss possible additional applications of our neuro-RAM to Johnson-Lindenstrauss random compression, which amounts to multiplying the input by a sparse random matrix – a generalization of input sampling.

2 Computational Model and Preliminaries

2.1 Network Structure

We now give a formal definition of our computational model. A *Spiking Neural Network* (SNN) $\mathcal{N} = \langle X, Z, A, w, b\rangle$ consists of n input neurons $X = \{x_1, \ldots, x_n\}$, m output neurons $Z = \{z_1, \ldots, z_m\}$, and ℓ auxiliary neurons $A = \{a_1, ..., a_\ell\}$. The directed, weighted synaptic

[2] Note that [20] shows that general deterministic sigmoidal circuits can be simulated by our spiking model. However, the simulation blows up the size of the circuit size by $\sqrt{n}$, giving $\Theta(n)$ auxiliary neurons.

connections between X, Z, and A are described by the weight function $w : [X \cup Z \cup A] \times [X \cup Z \cup A] \to \mathbb{R}$. A weight $w(u, v) = 0$ indicates that a connection is not present between neurons u and v. Finally, for any neuron v, $\mathrm{b}(v) \in \mathbb{R}_{\geq 0}$ is the activation bias – as we will see, roughly, v's membrane potential must reach $\mathrm{b}(v)$ for a spike to occur with good probability.

The weight function defining the synapses in our networks is restricted in a few notable ways. The in-degree of every input neuron x_i is zero. That is, $w(u, x) = 0$ for all $u \in [X \cup Z \cup A]$ and $x \in X$. This restriction bears in mind that the input layer might in fact be the output layer of another network and so incoming connections are avoided to allow for the composition of networks in higher level modular designs. Additionally, each neuron is either inhibitory or excitatory: if v is inhibitory, then $w(v, u) \leq 0$ for every u, and if v is excitatory, then $w(v, u) \geq 0$ for every u. All input and output neurons are excitatory.

2.2 Network Dynamics

An SNN evolves in discrete, synchronous rounds as a Markov chain. The firing probability of every neuron at time t depends on the firing status of its neighbors at time $t - 1$, via a standard sigmoid function, with details given below.

For each neuron u, and each time $t \geq 0$, let $u^t = 1$ if u fires (i.e., generates a spike) at time t. Let u^0 denote the initial firing state of the neuron. Our results will specify the initial input firing states $x_j^0 = 1$ and assume that $u^0 = 0$ for all $u \in [Z \cup A]$. For each non-input neuron u and every $t \geq 1$, let $pot(u, t)$ denote the membrane potential at round t and $p(u, t)$ denote the corresponding firing probability ($\Pr[u^t = 1]$). These values are calculated as:

$$pot(u,t) = \sum_{v \in X \cup Z \cup A} w_{v,u} \cdot v^{t-1} - b(u) \text{ and } p(u,t) = \frac{1}{1 + e^{-pot(u,t)/\lambda}} \tag{1}$$

where $\lambda > 0$ is a *temperature parameter*, which determines the steepness of the sigmoid. It is easy to see that λ does not affect the computational power of the network. A network can be made to work with any λ simply by scaling the synapse weights and biases appropriately.

For simplicity we assume that $\lambda = \frac{1}{\Theta(\log n)}$. Thus by (1), if $pot(u, t) \geq 1$, then $u^t = 1$ w.h.p. and if $pot(u, t) \leq -1$, $u^t = 0$ w.h.p. (recall that w.h.p. denotes with probability at least $1 - 1/n^c$ for some constant c). Aside from this fact, the only other consequence of (1) we use in our constructions is that $pot(u, t) = 0 \implies p(u, t) = 1/2$. That is, we use our spiking neurons entirely as random threshold gates, which fire w.h.p. when the incoming potential from their neighbors' spikes exceeds $\mathrm{b}(u)$, don't fire w.h.p. when the potential is below $\mathrm{b}(u)$, and fire randomly when the input potential equals the bias. It is an open question if there are any problems which require using the full power of the sigmoidal probability function.

2.3 Additional Notation

For any vector x we let x_i denote the value at its i^{th} position, starting from x_0. Given binary $x \in \{0, 1\}^n$, we use $\mathrm{dec}(x)$ to indicate the integer encoded by x. That is, $\mathrm{dec}(x) = \sum_{i=0}^{n-1} x_i \cdot 2^i$. Given an integer x we use $\mathrm{bin}(x)$ to denote its binary encoding, where the number of digits used in the encoding will be clear from context. We will often think of the firing pattern of a set of neurons as a binary string. If $B = \{y_1, ..., y_m\}$ is a set of m neurons then $B^t \in \{0, 1\}^m$ is the binary string corresponding to their firing pattern at time t. Since the input is typically fixed for some number of rounds, we often just write X to refer to the n-bit string corresponding to the input firing pattern.

Boolean Circuits. We mention that SNNs are similar to boolean circuits, which have received enormous attention in theoretical computer science. A circuit consists of gates (e.g., threshold gates, probabilistic threshold gates) connected in a directed acyclic graph. This restriction means that a circuit does not have feedback connections or self-loops, which we do use in our SNNs. While we do not work with circuits directly, for our lower bound, we show a transformation from an SNN to a linear threshold circuit. We sometimes refer to circuits as *feedforward* networks, indicating that their connections are cycle-free.

3 Neuro-RAM Network

In this section we prove our main upper bound:

▶ **Theorem 6** (Efficient Neuro-RAM Network). *There exists an SSN with $O(\sqrt{n})$ auxiliary neurons that solves indexing in $5\sqrt{n}$ rounds. Specifically, given inputs $X \in \{0,1\}^n$, and $Y \in \{0,1\}^{\log n}$, which are fixed for all rounds $t \in \{0,...,5\sqrt{n}\}$, the output neuron z satisfies: if $X_{\mathrm{dec}(Y)} = 1$ then $z^{5\sqrt{n}} = 1$ w.h.p. Otherwise, if $X_{\mathrm{dec}(Y)} = 0$, $z^{5\sqrt{n}} = 0$ w.h.p.*

Theorem 6 easily generalizes to other network sizes, giving Theorem 2, which states the full size-time tradeoff. Here we discuss the intuition behind the basic construction. The full details and proof are given in Appendices A.1 and A.2 of our full paper.

We divide the n input neurons X into $\sqrt{n}$ buckets each containing $\sqrt{n}$ neurons[3]:

$$X_0 = \{x_0, ..., x_{\sqrt{n}-1}\}, ..., X_{\sqrt{n}-1} = \{x_{(\sqrt{n}-1)\sqrt{n}}, ..., x_{n-1}\}.$$

Throughout, all our indices start from 0. We encode the firing pattern of each bucket X_i via the potential of a *single* neuron e_i. Set $w(x_{i\sqrt{n}+j}, e_i) = 2^{\sqrt{n}-j}$ for all $i, j \geq 0$. Thus, for every round t, the total potential contributed to e_i by the firing of the inputs in bucket X_i is:

$$\sum_{j=0}^{\sqrt{n}-1} x_{i\sqrt{n}+j} \cdot 2^{\sqrt{n}-j} = 2 \cdot \mathrm{dec}(\bar{X}_i). \tag{2}$$

where $\bar{X}_i$ is the reversal of X_i and $\mathrm{dec}(\cdot)$ gives the decimal value of a binary string, as defined in the preliminaries. We set $\mathrm{b}(e_i) = 2^{\sqrt{n}+2} + 2^{\sqrt{n}} - 1$. We will see later why this is an appropriate value. We defer detailed discussion of the remaining connections to e_i for now, first giving a general description of the network construction.

In addition to the *encoding neurons* $e_0, ..., e_{\sqrt{n}-1}$, we have *decoding neurons* $d_{0,k}, ..., d_{\sqrt{n}-1,k}$ for $k = 1, 2, 3$ ($3\sqrt{n}$ neurons total). The idea is to select a bucket X_i (via e_i) using the first $\log\sqrt{n} = \frac{\log n}{2}$ bits in the index Y. Let $Y_1 \stackrel{\text{def}}{=} \{y_0, ..., y_{\frac{\log n}{2}-1}\}$ and $Y_2 \stackrel{\text{def}}{=} \{y_{\frac{\log n}{2}}, ..., y_{\log n-1}\}$ be the higher and lower order bits of Y respectively. It is not hard to see that using $O(\sqrt{n})$ neurons we can construct a network that processes Y_1 and uses it to select e_i with $i = \mathrm{dec}(Y_1)$. When a bucket is selected, the potential of any e_j with $j \neq \mathrm{dec}(Y_1)$ is significantly depressed compared to that of e_i and so after this selection stage, only e_i fires.

We then use the decoding neurons to 'read' *each bit of the potential encoded in* e_i. The final output is selected from each of these bits using the lower order bits Y_2, which can again be done efficiently with $O(\sqrt{n})$ neurons. We call this phase the decoding phase since e_i encodes the value (in decimal) of its bucket X_i, and we need to decode from that value the bit of the appropriate neuron inside that bucket.

[3] Throughout we assume for simplicity that $n = 2^{2m}$ for some integer m. This ensures that $\sqrt{n}$, $\log n$, and $\log\sqrt{n}$ are integers. It will be clear that if this is not the case, we can simply pad the input, which only affects our time and network size bounds by constant factors.

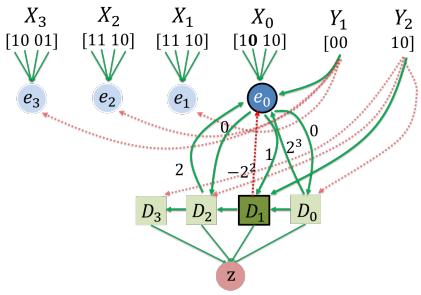

Figure 1 Illustration of the neuro-RAM module. D_i represents the set of 3 decoding neurons for each bit: $\{d_{i,1}, d_{i,2}, d_{i,3}\}$. The dotted lines from Y_1 and Y_2 represent connections to the buckets and decoding neurons which are not currently selected. The index encoded by Y is marked in bold and the selected encoding and decoding neurons are highlighted.

The decoding process works as follows: initially, e_i will fire only if the *first bit* of bucket i is on. Note that the weight from this bit to e_i is $2^{\sqrt{n}}$ and thus more than double the weight from any other input bit. Thus, by appropriately setting $\mathrm{b}(e_i)$, we can ensure that the setting of this single bit determines if e_i fires initially.

If the first bit is the correct bit to output (i.e., if the last $\frac{\log n}{2}$ bits of the index Y_2 encode position 0), this will trigger the output z to fire. Otherwise, we iterate. If e_i in fact fired, this triggers inhibition that cancels out the potential due to the first bit of bucket i. So e_i will now only fire if the *second bit* of X_i is on. If e_i did not fire, the opposite will happen. Further excitation will be given to e_i again ensuring that it can fire as long as the second bit of X_i is on. The network iterates in this way, successively reading each bit, until we reach the one encoded by Y_2 and the output fires. The first decoding neuron for position j, $d_{j,1}$, is responsible to triggering the output to fire if j is the correct bit encoded by Y_2. The second decoding neuron $d_{j,2}$ is responsible for providing excitation when e_i does not fire. Finally, the third decoding neuron $d_{j,3}$ provides inhibition when e_i does fire.

In Appendix A.1 of our full paper, we describe the first stage in which we use the first $\log n/2$ index bits to select the bucket to which the desired index belongs to.

In Appendix A.2, we discuss the second phase where we use the last $\log n/2$ bits of Y, to select the desired index inside the bucket i. Our successive decoding process is synchronized by a clock mechanism. This clock mechanism consists of chain of $\Theta(\sqrt{n})$ neurons that govern the timing of the $\Theta(\sqrt{n})$ steps of our decoding scheme. Roughly, traversing the $\sqrt{n}$ bits of the chosen i^{th} bucket from left to right, we spend $O(1)$ rounds checking if the current index is the one encoded by Y_2. If yes, we output the value at that index and if not, the clock "ticks" and we move to the next candidate.

Note that our model and the proof of Theorem 6 assume that no auxiliary neurons or the output neuron fire in round 0. However, in applications it will often be desirable to run the neuro-RAM for multiple inputs, with execution not necessarily starting at round 0. We can easily add a mechanism that 'clears' the network once it outputs, giving:

▶ **Observation 7** (Running Neuro-RAM for Multiple Inputs). *The neuro-RAM of Theorem 6 can be made to run correctly given a sequence of multiple inputs.*

4 Lower Bound for Neuro-RAM in Spiking Networks

In this section, we show that our neuro-RAM construction is nearly optimal. Specifically:

▶ **Theorem 8.** *Any SNN solving indexing with probability* $\geq 1 - \frac{1}{2n}$ *in* t *rounds must use* $\ell = \Omega\left(\frac{n}{t \log^2 n}\right)$ *auxiliary neurons.*

This result matches the lower bound for deterministic threshold gates of Fact 3 up to a $\log n$ factor, demonstrating that the use of randomness cannot give significant runtime advantages for the indexing problem. Even if one just desires a constant (e.g., 2/3) probability of success, a lower bound of $\Omega\left(\frac{n}{t \log^3 n}\right)$ applies: by replicating any network with success probability 2/3, $\Theta(\log n)$ times and taking the majority output (which can be computed with just a single additional auxiliary neuron), we obtain a network that solves the problem w.h.p.

4.1 High Level Approach and Intuition

The proof of Theorem 8 proceeds in a number of steps, which we overview here.

Reduction to Deterministic Indexing Circuit. We first observe that a network with ℓ auxiliary neurons solving the indexing problem in t rounds can be unrolled into a feedforward circuit with t layers and ℓ neurons per layer. We then show that the output distribution of a feedforward stochastic spiking circuit is identical to the output distribution if we first draw a deterministic linear threshold circuit (still with t layers and ℓ neurons per layer) from a certain distribution, and evaluate our input using this random circuit.

This equivalence is powerful since it allows us to apply Yao's principle [27]: assuming the existence of a feedforward SNN solving indexing with probability $\geq 1 - \frac{1}{2n}$, given any distribution of the inputs X, Y, there must be some deterministic linear threshold circuit $\mathcal{N}_D$ which solves indexing with probability $\geq 1 - \frac{1}{2n}$ over this distribution.

If we consider the uniform distribution over X, Y, this success probablity ensures via an averaging argument that for at least 1/2 of the 2^n possible values of X, $\mathcal{N}_D$ succeeds for at least a $1 - \frac{1}{2n}$ fraction of the possible Y inputs. Note, however, that the Y can only take on n possible values – thus this ensures that for 1/2 the possible values of X, $\mathcal{N}_D$ succeeds for *all possible values of the index* Y. Let $\mathcal{X}$ be the set of 'good inputs' for which $\mathcal{N}_D$ succeeds.

Lower Bound for Deterministic Indexing on a Subset of Inputs. We have now reduced our problem to giving a lower bound on the size of a deterministic linear threshold circuit which solves indexing on an arbitrary subset $\mathcal{X}$ of $\frac{1}{2} \cdot 2^n = 2^{n-1}$ inputs. We do this using VC dimension techniques inspired by the indexing lower bound of [16].

The key idea is to observe that if we fix some input $X \in \mathcal{X}$, then given Y, $\mathcal{N}_D$ evaluates the function $f_X : \{0,1\}^{\log n} \to \{0,1\}$, whose truth table is given by X. Thus $\mathcal{N}_D$ can be viewed as a circuit for evaluating any function $f_X(Y)$ for $X \in \mathcal{X}$, where the X inputs are 'programmable parameters', which effectively change the thresholds of some gates.

It can be shown that the VC dimension of the class of functions computable by a fixed a linear threshold circuit with m gates and variable thresholds is $O(m \log m)$. Thus for a circuit with t layers and ℓ gates per layer, the VC dimension is $O(\ell t \log(\ell t))$ [5]. Further, as a consequence of Sauer's Lemma [23, 25, 4], defining the class of functions $\mathcal{F} = \{f_X \text{ for any } X \in \mathcal{X}\}$, since $|\mathcal{F}| = |\mathcal{X}| = 2^{n-1}$, we have $VC(\mathcal{F}) = \Theta(n/\log n)$. These two VC dimension bounds, in combination with the fact that we know $\mathcal{N}_D$ can compute any function in $\mathcal{F}$ if its input bits are fixed appropriately, imply that $\ell t \cdot \log(\ell t) = \Omega(n/\log n)$. Rearranging gives $\ell = \Omega\left(\frac{n}{t \log^2 n}\right)$, completing Theorem 8.

4.2 Reduction to Deterministic Indexing Circuit

We now give the argument explained above in detail, first describing how any SNN that solves indexing w.h.p. implies the existence of a deterministic feedforward linear threshold circuit which solves indexing for a large fraction of possible inputs X.

▶ **Lemma 9** (Conversion to Feedforward Network). *Consider any SNN $\mathcal{N}$ with ℓ auxiliary neurons, which given input $X \in \{0,1\}^n$ that is fixed for rounds $\{0,...,t\}$, has output z satisfying $\Pr[z^t = 1] = p$. Then there is a feedforward SNN $\mathcal{N}_F$ (an SNN whose directed edges form an acyclic graph) with $(t-1)\cdot(\ell+1)$ auxiliary neurons also satisfying $\Pr[z^t = 1] = p$ when given X which is fixed for rounds $\{0,...,t\}$.*

Proof. Let $B = A \cup z$ – all non-input neurons. We produce $t-1$ duplicates of each auxiliary neuron $a \in A : \{a_1,...,a_{t-1}\}$ and of $z : \{z_1,...,z_{t-1}\}$, which are split into layers $B_1,...,B_{t-1}$. For each incoming edge from a neuron u to v and each $i \geq 2$ we add an identical edge from u_{i-1} to v_i. Any incoming edges from input neurons to u are added to each u_i for all $i \geq 1$. Finally connect z to the appropriate neurons in B_{t-1} (including z_{t-1} if there is a self-loop).

In round 1, the joint distribution of the spikes B_1^1 in $\mathcal{N}_F$ is identical to the distribution of B^1 in $\mathcal{N}$ since these neurons have identical incoming connections from the inputs, and since any incoming connections from other auxiliary neurons are not triggered in $\mathcal{N}$ since none of these neurons fire at time 0.

Assuming via induction that B_i^i is identically distributed to B^i, since B_{i+1} only has incoming connections from B_i and the inputs which are fixed, then the distribution of B_{i+1}^{i+1} identical to that of B^{i+1}. Thus B_{t-1}^{t-1} is identically distributed to B^{t-1}, and since the output in $\mathcal{N}_F$ is only connected to B_{t-1} its distribution is the same in round t as in $\mathcal{N}$. ◀

▶ **Lemma 10** (Conversion to Distribution over Deterministic Threshold Circuits). *Consider any spiking sigmoidal network $\mathcal{N}$ with ℓ auxiliary neurons, which given input $X \in \{0,1\}^n$ that is fixed for rounds $\{0,...,t\}$, has output neuron z satisfying $\Pr[z^t = 1] = p$. Then there is a distribution $\mathcal{D}$ over feedforward deterministic threshold circuits with $(t-1)\cdot(\ell+1)$ auxiliary gates that, for $\mathcal{N}_D \sim \mathcal{D}$ with output z, $\Pr_{\mathcal{D}}[z^t = 1] = p$ when presented input X.*

Proof. We start with $\mathcal{N}_F$ obtained from Lemma 9. This circuit has $t-1$ layers of $\ell+1$ neurons $B_1,...,B_{t-1}$. Given $X \in \{0,1\}^n$ that is fixed for rounds $\{0,...,t\}$, $\mathcal{N}_F$ has $\Pr[z^t = 1] = p$, which matches the firing probability of the output z in $\mathcal{N}$ in round t.

Let $\mathcal{D}$ be a distribution on deterministic threshold circuits that have identical edge weights to $\mathcal{N}_F$. Additionally, for any (non-input) neuron $u \in \mathcal{N}_F$, letting $\bar{u}$ be the corresponding neuron in the deterministic circuit, set the bias $b(\bar{u}) = \eta$, where η is distributed according to a logistic distribution with mean $\mu = b(u)$ and scale $s = \lambda$. The random bias is chosen independently for each u. It is well known that the cumulative density function of this distribution is equal to the sigmoid function. That is:

$$\Pr[\eta \leq x] = \frac{1}{1 + e^{-\frac{x - b(u)}{\lambda}}}. \tag{3}$$

Consider $\mathcal{N}_D \sim \mathcal{D}$ and any neuron u in the first layer B_1 of $\mathcal{N}_F$. u only has incoming edges from the input neurons X. Thus, its corresponding neuron $\bar{u}$ in $\mathcal{N}_D$ also only has incoming edges from the input neurons. Let $W = \sum_{x \in X} w(x,u) \cdot x^0$. Then we have:

$$\begin{aligned}
\Pr_{\mathcal{D}}[\bar{u}^1 = 1] = \Pr[W - \eta \geq 0] = \Pr[\eta \leq W] & \qquad \text{(Deterministic threshold)} \\
= \frac{1}{1 + e^{-\frac{W - b(u)}{\lambda}}} & \qquad \text{(Logistic distribution CDF (3))} \\
= \Pr[u^1 = 1]. & \qquad \text{(Spiking sigmoid dynamics (1))}
\end{aligned}$$

Let $\bar{B}_i$ denote the neurons in $\mathcal{N}_D$ corresponding to those in B_i. Since in round 1, all neurons in B_1 fire independently and since all neurons in $\bar{B}_1$ fire independently as their random biases are chosen independently, the joint firing distribution of B_1^1 is identical to that of $\bar{B}_1^1$.

By induction assume that $\bar{B}_i^i$ is identically distributed (over the random choice of deterministic network $\mathcal{N}_D \sim \mathcal{D}$) to B_i^i. Then for any $u \in B_{i+1}$ we have by the same argument as above, conditioning on some fixed firing pattern V of B_i in round i:

$$\Pr_{\mathcal{D}}[\bar{u}^{i+1} = 1 | \bar{B}_i^i = V] = \Pr[u^{i+1} = 1 | B_i^i = V].$$

Conditioned on $B_i^i = V$, the neurons in B_{i+1} fire independently in round $i+1$. So do the neurons of $\bar{B}_{i+1}$ due to their independent choices of random biases. Thus, the above implies that the distribution of $\bar{B}_{i+1}^{i+1}$ conditioned on $\bar{B}_i^i = V$ is identical to the distribution of B_{i+1}^{i+1}. This holds for all V, so, the full joint distribution of $\bar{B}_{i+1}^{i+1}$ is identical to that of B_{i+1}^{i+1}.

We conclude by noting that the same argument applies for the outputs of $\mathcal{N}_F$ and $\mathcal{N}_D$ since $\bar{B}_{t-1}^{t-1}$ is identically distributed to B_{t-1}^{t-1}. ◀

Lemma 10 is simple but powerful – it demonstrates that *the output distribution of a spiking sigmoid network is identical to the output distribution of a deterministic feedforward threshold circuit drawn from some distribution $\mathcal{D}$.* Thus, the performance of any SNN is equivalent to the performance of a randomized algorithm which first selects a linear threshold circuit using $\mathcal{D}$ and then applies this circuit to the input. This lets us show:

▶ **Lemma 11** (Application of Yao's Principle). *Assume there exists an SNN $\mathcal{N}$ with ℓ auxiliary neurons, which given any inputs $X \in \{0,1\}^n$ and $Y \in \{0,1\}^{\log n}$ which are fixed for rounds $\{0, ..., t\}$, solves indexing with probability $\geq 1 - \delta$ in t rounds. Then there exists a feedforward deterministic linear threshold circuit $\mathcal{N}_D$ with $(t-1) \cdot (\ell + 1)$ auxiliary gates which solves indexing with probability $\geq 1 - \delta$ given X, Y drawn uniformly at random.*

Proof. We use the idea of Yao's principle, employing an averaging argument to show that the existence of a randomized circuit succeeding with high probability implies the existence of a deterministic circuit succeeding with high probability on uniform random inputs. Specifically, given X, Y drawn uniformly at random, $\mathcal{N}$ solves indexing with probability $\geq 1 - \delta$ (since by assumption, it succeeds with this probability for any X, Y). By Lemma 10, $\mathcal{N}$ performs identically to an algorithm which selects a deterministic circuit from some distribution $\mathcal{D}$ and then applies it to the input. So at least one circuit in the support of $\mathcal{D}$ must succeed with probability $\geq 1 - \delta$ on X, Y drawn uniformly at random, since the success probability of $\mathcal{N}$ on the uniform distribution is just an average over the deterministic success probabilities. ◀

From Lemma 11 we have a corollary which concludes our reduction from our spiking sigmoid lower bound to a lower bound on deterministic indexing circuits.

▶ **Corollary 12** (Reduction to Deterministic Indexing on a Subset of Inputs). *Assume there exists an SNN $\mathcal{N}$ with ℓ auxiliary neurons, which, given inputs $X \in \{0,1\}^n$ and $Y \in \{0,1\}^{\log n}$ which are fixed for rounds $\{0, ..., t\}$, solves indexing with probability $\geq 1 - \frac{1}{2n}$ in t rounds. Then there exists some subset of inputs $\mathcal{X} \subseteq \{0,1\}^n$ with $|\mathcal{X}| \geq 2^{n-1}$ and a feedforward deterministic linear threshold circuit $\mathcal{N}_D$ with $(t-1) \cdot (\ell + 1)$ auxiliary gates which solves indexing given any $X \in \mathcal{X}$ and any index $Y \in \{0,1\}^{\log n}$.*

Proof. Applying Lemma 11 yields $\mathcal{N}_D$ which solves indexing on uniformly random X, Y with probability $1 - \frac{1}{2n}$. Let $\mathbb{I}(X, Y) = 1$ if $\mathcal{N}_D$ solves indexing correctly on X, Y and 0

otherwise. Then:

$$1 - \frac{1}{2n} \leq \frac{1}{n \cdot 2^n} \sum_{X \in \{0,1\}^n} \sum_{Y \in \{0,1\}^{\log n}} \mathbb{I}(X,Y) = \mathop{\mathrm{E}}_{X \text{ uniform from } \{0,1\}^n} \left[\frac{1}{n} \sum_{Y \in \{0,1\}^{\log n}} \mathbb{I}(X,Y) \right]$$

which in turn implies:

$$\mathop{\mathrm{E}}_{X \text{ uniform from } \{0,1\}^n} \left[\frac{1}{n} \sum_{Y \in \{0,1\}^{\log n}} (1 - \mathbb{I}(X,Y)) \right] \leq \frac{1}{2n}. \tag{4}$$

If $\frac{1}{n} \sum_{Y \in \{0,1\}^{\log n}} (1 - \mathbb{I}(X,Y)) \neq 0$ then $\frac{1}{n} \sum_{Y \in \{0,1\}^{\log n}} (1 - \mathbb{I}(X,Y)) \geq \frac{1}{n}$ just by the fact that the sum is an integer. Thus, for (4) to hold, we must have $\frac{1}{n} \sum_{Y \in \{0,1\}^{\log n}} (1 - \mathbb{I}(X,Y)) = 0$ for at least $\frac{1}{2}$ of the inputs $X \in \{0,1\}^n$. That is, $\mathcal{N}_D$ solves indexing for every input index on some subset $\mathcal{X}$ with $|\mathcal{X}| \geq \frac{1}{2}|\{0,1\}^n| \geq 2^{n-1}$. ◀

4.3 Lower Bound for Deterministic Indexing on a Subset of Inputs

With Corollary 12 in place, we now turn to lower bounding the size of a deterministic linear threshold circuit $\mathcal{N}_D$ which solves the indexing problem on some subset of inputs $\mathcal{X}$ with $|\mathcal{X}| \geq 2^{n-1}$. To do this, we employ VC dimension techniques first introduced for bounding the size of linear threshold circuits computing indexing on all inputs [16].

Consider fixing some input $X \in \mathcal{X}$, such that the output of $\mathcal{N}_D$ is just a function of the index Y. Specifically, with X fixed, $\mathcal{N}_D$ computes the function $f_X : \{0,1\}^{\log n} \to \{0,1\}$ whose truth table is given by X. Note that the output of $\mathcal{N}_D$ with X fixed is equivalent to the output of a feedforward linear threshold circuit $\mathcal{N}_D^X$ where each gate with an incoming edge from $x_i \in X$ has its threshold adjusting to reflect the weight of this edge if $x_i = 1$.

We define two sets of functions. Let $\mathcal{F} = \{f_X | X \in \mathcal{X}\}$ be all functions computable using some $\mathcal{N}_D^X$ as defined above. Further, let $\mathcal{G}$ be the set of all functions computabled by any circuit $\mathcal{N}'_D$ which is generated by removing the input gates of $\mathcal{N}_D$ and adjusting the threshold on each remaining gate to reflect the effects of any inputs with $x_i = 1$. We have $\mathcal{F} \subseteq \mathcal{G}$ and hence, letting $VC(\cdot)$ denote the VC dimension of a set of functions have: $VC(\mathcal{F}) \leq VC(\mathcal{G})$. We can now apply two results. The first gives a lower bound $VC(\mathcal{F})$:

▶ **Lemma 13** (Corollary 3.8 of [4] – Consequence of Sauer's Lemma [23, 25]). *For any set of boolean functions $\mathcal{H} = \{h\}$ with $h : \{0,1\}^{\log n} \to \{0,1\}$:*

$$VC(\mathcal{H}) \geq \frac{\log |\mathcal{H}|}{\log n + \log e}.$$

We next upper bound $VC(\mathcal{G})$. We prove in Appendix B of our full paper:

▶ **Lemma 14** (Linear Threshold Circuit VC Bound). *Let $\mathcal{H}$ be the set of all functions computed by a fixed feedforward linear threshold circuit with $m \geq 2$ gates (i.e., fixed edges and weights), where each gate has a variable threshold. Then: $VC(\mathcal{H}) \leq 3m \log m$.*

Applying the bounds of Lemmas 13 and 14 along with $VC(\mathcal{F}) \leq VC(\mathcal{G})$ gives:

▶ **Lemma 15** (Deterministic Circuit Lower Bound). *For any set $\mathcal{X} \subseteq \{0,1\}^n$ with $|\mathcal{X}| \geq 2^{n-1}$, any feedforward deterministic linear threshold circuit $\mathcal{N}_D$ with m non-input gates which solves indexing given any $X \in \mathcal{X}$ and any index $Y \in \{0,1\}^{\log n}$ must have $m = \Omega\left(\frac{n}{\log^2 n}\right)$.*

Proof. Let $\mathcal{F}$ and $\mathcal{G}$ be as defined in the beginning of the section. We have $VC(\mathcal{F}) \leq VC(\mathcal{G})$. At the same time, by Lemma 13 we have $VC(\mathcal{F}) \geq \frac{\log|\mathcal{F}|}{\log n + \log e} = \frac{\log|\mathcal{X}|}{\log n + \log e} \geq \frac{cn}{\log n}$ for some fixed constant c. By Lemma 14 we have $VC(\mathcal{G}) \leq 3m \log m$. We thus can conclude that $\frac{cn}{\log n} \leq 3m \log m$, and so $m = \Omega\left(\frac{n}{\log^2 n}\right)$. ◀

We conclude by proving our main lower bound:

Proof of Theorem 8. The existence of a spiking sigmoidal network with ℓ auxiliary neurons, solving indexing with probability $\geq 1 - \frac{1}{2n}$ in t rounds implies via Corollary 12 the existence of a feedforward deterministic linear threshold circuit with $(t-1)\ell + 1$ non-input gates solving indexing on some subset of inputs $\mathcal{X}$ with $|\mathcal{X}| \geq 2^{n-1}$. So by Lemma 15, $\ell \cdot t = \Omega\left(\frac{n}{\log^2 n}\right)$. ◀

5 Applications to Similarity Testing and Compression

5.1 Similarity Testing

▶ **Theorem 16** (Similarity Testing). *There exists an SNN with $O\left(\frac{\sqrt{n}\log n}{\epsilon}\right)$ auxilary neurons that solves the approximate equality testing problem in $O(\sqrt{n})$ rounds. Specifically, given inputs $X_1, X_2 \in \{0,1\}^n$ which are fixed for all rounds $t \in \{0, .., 5\sqrt{n}+2\}$, the output z satisfies w.h.p. $z^{5\sqrt{n}+2} = 1$ if $\mathrm{d_H}(X_1, X_2) \geq \epsilon n$. Further if $X_1 = X_2$ then $z^{5\sqrt{n}+2} = 0$ w.h.p.*

Our similarity testing network uses $K = \Theta\left(\frac{\log n}{\epsilon}\right)$ copies of our neuro-RAM network from Theorem 6, labeled $S_{1,k}$ and $S_{2,k}$ for all $k \in \{1, ..., K\}$. The idea is to employ $\log n$ auxiliary neurons $Y_k = y_{1,k}, ..., y_{\log n, k}$ whose values encode a *random index* $i \in \{0, ..., n-1\}$. By feeding the inputs (X_1, Y_k) and (X_2, Y_k) into S_1 and S_2, we can check whether X_1 and X_2 match at position i.

Checking $\Theta\left(\frac{\log n}{\epsilon}\right)$ different random indices suffices to identify if $\mathrm{d_H}(X_1, X_0) \geq \epsilon n$ w.h.p. Further, if $X_1 = X_0$, they will never differ at any of the checks, and so the output will never be triggered. We use:

▶ **Observation 17.** *Consider $X_1, X_2 \in \{0,1\}^n$ with $\mathrm{d_H}(X_1, X_0) \geq \epsilon n$. Let $i_1, ..., i_T$ be chosen independently and uniformly at random in $\{0, ..., n-1\}$. Then for $T = \frac{c \ln n}{\epsilon}$,*

$$\Pr[(X_1)_{i_t} = (X_2)_{i_t} \text{ for all } t \in 1, ..., T] \leq \frac{1}{n^c}.$$

Proof. For any fixed t, $\Pr[(X_1)_{i_t} = (X_2)_{i_t}] = 1 - \frac{\epsilon n}{n} = 1 - \epsilon$ as we select indices at random. Additionally, each of these events is independent since $i_1, ...i_T$ are chosen independently so: $\Pr[(X_1)_{i_t} = (X_2)_{i_t} \text{ for all } t \in 1, ..., T] \leq (1-\epsilon)^T = \left(1-\epsilon\right)^{1/\epsilon}{}^{c \ln n} \leq \frac{1}{e^{c \ln n}} \leq \frac{1}{n^c}$. ◀

5.1.1 Implementation Sketch

It is clear that the above strategy can be implemented in the spiking sigmoidal network model – we sketch the construction here. By Theorem 6, we require $O\left(\frac{\sqrt{n}\log n}{\epsilon}\right)$ auxiliary neurons for the $2K = \Theta\left(\frac{\log n}{\epsilon}\right)$ neuro-RAMs employed, which dominates all other costs.

It suffices to present a random index to each pair of neuro-RAMs S_1, k an S_2, k for $5\sqrt{n}$ rounds (the number of rounds required for the network of Theorem 6 to process an n-bit input). To implement this strategy, we need two simple mechanisms, described below.

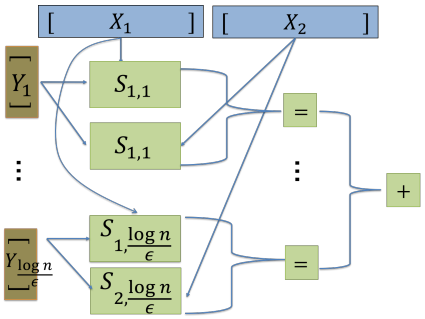

Figure 2 Illustration of our ϵ-approximate similarity testing network.

Random Index Generation: For each of the $\log n$ index neurons in Y_k we set $\mathrm{b}(y_i) = 0$ and add a self-loop $w(y_i, y_i) = 2$. In round 1, since they have no-inputs, each neuron has potential 0 and fires with probability 1/2. Thus, Y_k^1 represents a random index in $\{0, ..., n-1\}$. To propagate this index we can use a single auxiliary inhibitory neuron g, which has bias $\mathrm{b}(g) = 1$ and $w(x, g) = 2$ for every input neuron x. Thus, g fires w.h.p. in round 1 and continues firing in all later rounds, as long as at least one input fires.

We add an inhibitory edge from g to y_i for all i with weight $w(g, y_i) = -1$. The inhibitory edges from g will keep the random index 'locked' in place. The inhibitory weight of -1 prevents any y_i without an active self-loop from firing w.h.p. but allows any y_i with an active self-loop to fire w.h.p. since it will still have potential $\mathrm{b}(y_i) + w(y_i, y_i) - 1 = 1$.

If both inputs are 0, g will not fire w.h.p. However, here our network can just output 0 since $X_1 = X_2$ so it does not matter if the random indices stay fixed.

Comparing Outputs: We next handle comparing the outputs of $S_{1,k}$ and $S_{2,k}$. We use two neurons – $f_{1,k}$ and $f_{2,k}$. $f_{1,k}$ is excitatory and fires w.h.p. if at least one of $S_{1,k}$ or $S_{2,k}$ has an active output. $f_{2,k}$ is an inhibitor that fires only if *both* $S_{1,k}$ and $S_{2,k}$ have active outputs. We then connect $f_{1,k}$ to z with weight $w(f_{1,k}, z) = 2$ and connect $f_{2,k}$ with weight $w(f_2, z) = -2$ for all k. We set $\mathrm{b}(z) = 1$. Thus, z fires in round $5\sqrt{n} + 2$ w.h.p. if for some k, *exactly one* of $S_{1,k}$ or $S_{2,k}$ has an active output in round $5\sqrt{n}$ and hence an inequality is detected. Otherwise, z does not fire w.h.p. This gives the output condition of Theorem 16.

5.2 Randomized Compression

We conclude by discussing informally how our neuro-RAM can be applied beyond similarity testing to other randomized compression schemes. Consider the setting where we are given n input vectors $X_i \in \{0,1\}^d$. Let $\mathbf{X} \in \{0,1\}^{n \times d}$ denote the matrix of all inputs. Think of d as being a large ambient dimension, which we would like to reduce before further processing.

One popular technique is *Johnson-Lindenstrauss (JL) random projection*, where $\mathbf{X}$ is multiplied by a random matrix $\mathbf{\Pi} \in \mathbb{R}^{d \times d'}$ with $d' << d$ to give the compressed dataset $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{\Pi}$. *Regardless of the initial dimension* d, if d' is set large enough, $\tilde{\mathbf{X}}$ preserves significant information about $\mathbf{X}$. $d' = \tilde{O}(\log n)$ is enough to preserve the distances between all points, $d' = \tilde{O}(k)$ is enough to use $\tilde{\mathbf{X}}$ for approximate k-means clustering or k-rank approximation [6, 8], and $d' = \tilde{O}(n)$ preserves the full covariance matrix of the input and so $\tilde{\mathbf{X}}$ can be used for approximate regression and many other problems [7, 22].

JL projection has been suggested as a method for neural dimensionality reduction [2, 11], where $\mathbf{\Pi}$ is viewed as a matrix of random synapse weights, which connect the input neurons representing $\mathbf{X}$ to the output neurons representing $\tilde{\mathbf{X}}$. While this view is quite natural, we

often want to draw $\mathbf{\Pi}$ with *fresh randomness* for each input $\mathbf{X}$. This is not possible using changing synapse weights, which evolve over a relatively long time scale. Fortunately, it is possible to simulate these random connections using our neuro-RAM module.

Typically, $\mathbf{\Pi}$ is sparse so can be multiplied by efficiently. In the most efficient construction [7], it has just a single nonzero entry in each row which is a randomly chosen ± 1 placed in a uniform random position. Thus, computing a single bit of $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{\Pi}$ requires selecting on average d/d' random columns of $\mathbf{X}$, multiplying their entries by a random sign and summing them together. This can be done with a set of neuro-RAMS, each using $O(\sqrt{d})$ auxiliary neurons which select the random columns of $\mathbf{X}$. In total, we need $\tilde{O}(d/d')$ networks – the maximum column sparsity of $\mathbf{\Pi}$ with high probability, yielding $O(d^{3/2}/d')$ auxiliary neurons total. In contrast, a naive simulation of random edges using spiking neurons requires $\Theta(d)$ auxiliary neurons, which is less efficient whenever $d' > d^{1/2}$. Additionally, our neuro-RAMs can be reused to compute multiple entries of $\tilde{\mathbf{X}}$, which is not the case for the naive simulation.

Traditionally, the value of an entry of $\tilde{\mathbf{X}}$ is a real number, which cannot be directly represented in a spiking neural network. In our construction, the value of the entry is encoded in its potential, and we leave as an interesting open question how this potential should be decoded or otherwise used in downstream applications of the compression.

Acknowledgments. We thank Mohsen Ghaffari – the initial ideas regarding the importanace of the indexing module came up while Merav Parter was visiting him at ETH Zurich. We also thank Sergio Rajsbaum, Ron Rothblum, and Nir Shavit for helpful discussions.

References

1 Christina Allen and Charles F Stevens. An evaluation of causes for unreliability of synaptic transmission. *PNAS*, 1994.

2 Zeyuan Allen-Zhu, Rati Gelashvili, Silvio Micali, and Nir Shavit. Sparse sign-consistent Johnson–Lindenstrauss matrices: Compression with neuroscience-based constraints. *PNAS*, 2014.

3 Eric Allender. A note on the power of threshold circuits. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.

4 Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. Cambridge University Press, 2009.

5 Eric B Baum and David Haussler. What size net gives valid generalization? In *Advances in Neural Information Processing Systems 2 (NIPS)*, 1989.

6 Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. Random projections for k-means clustering. In *Advances in Neural Information Processing Systems 23 (NIPS)*, 2010.

7 Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, 2013.

8 Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, 2015.

9 A Aldo Faisal, Luc PJ Selen, and Daniel M Wolpert. Noise in the nervous system. *Nature Reviews Neuroscience*, 9(4):292–303, 2008.

10 Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Theory of Computing Systems*, 17(1):13–27, 1984.

11 Surya Ganguli and Haim Sompolinsky. Compressed sensing, sparsity, and dimensionality in neuronal information processing and data analysis. *Annual Review of Neuroscience*, 2012.

12 Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press, 2002.

13 John J Hopfield, David W Tank, et al. Computing with neural circuits- a model. *Science*, 233(4764):625–633, 1986.

14 Bill G Horne and Don R Hush. On the node complexity of neural networks. *Neural Networks*, 7(9):1413–1426, 1994.

15 Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, 2004.

16 Pascal Koiran. VC dimension in circuit complexity. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, 1996.

17 Nancy Lynch, Cameron Musco, and Merav Parter. Computational tradeoffs in biological neural networks: Self-stabilizing winner-take-all networks. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2017.

18 Wolfgang Maass. On the computational power of noisy spiking neurons. In *Advances in Neural Information Processing Systems 9 (NIPS)*, 1996.

19 Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

20 Wolfgang Maass, Georg Schnitger, and Eduardo D Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1991.

21 Marvin Minsky and Seymour Papert. Perceptrons. 1969.

22 Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

23 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

24 Michael N Shadlen and William T Newsome. Noise, neural codes and cortical organization. *Current Opinion in Neurobiology*, 4(4):569–579, 1994.

25 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

26 Leslie G Valiant. *Circuits of the Mind*. Oxford University Press on Demand, 2000.

27 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1977.