Counting to Ten with Two Fingers: Compressed Counting with Spiking Neurons

Yael Hitron
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel
yael.hitron@weizmann.ac.il

Merav Parter
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel
merav.parter@weizmann.ac.il

Abstract

We consider the task of measuring time with probabilistic threshold gates implemented by bio-inspired spiking neurons. In the model of *spiking neural networks*, network evolves in discrete rounds, where in each round, neurons fire in pulses in response to a sufficiently high membrane potential. This potential is induced by spikes from neighboring neurons that fired in the previous round, which can have either an excitatory or inhibitory effect.

Discovering the underlying mechanisms by which the brain perceives the duration of time is one of the largest open enigma in computational neuro-science. To gain a better algorithmic understanding onto these processes, we introduce the *neural timer* problem. In this problem, one is given a time parameter t, an input neuron x, and an output neuron y. It is then required to design a minimum sized neural network (measured by the number of auxiliary neurons) in which every spike from x in a given round i, makes the output y fire for the subsequent t consecutive rounds.

We first consider a deterministic implementation of a neural timer and show that $\Theta(\log t)$ (deterministic) threshold gates are both sufficient and necessary. This raised the question of whether randomness can be leveraged to reduce the number of neurons. We answer this question in the affirmative by considering neural timers with spiking neurons where the neuron y is required to fire for t consecutive rounds with probability at least $1-\delta$, and should stop firing after at most $2t$ rounds with probability $1-\delta$ for some input parameter $\delta \in (0,1)$. Our key result is a construction of a neural timer with $O(\log\log 1/\delta)$ spiking neurons. Interestingly, this construction uses only *one* spiking neuron, while the remaining neurons can be deterministic threshold gates. We complement this construction with a matching lower bound of $\Omega(\min\{\log\log 1/\delta, \log t\})$ neurons. This provides the first separation between deterministic and randomized constructions in the setting of spiking neural networks.

Finally, we demonstrate the usefulness of compressed counting networks for *synchronizing* neural networks. In the spirit of distributed synchronizers [Awerbuch-Peleg, FOCS'90], we provide a general transformation (or simulation) that can take any synchronized network solution and simulate it in an asynchronous setting (where edges have arbitrary response latencies) while incurring a small overhead w.r.t the number of neurons and computation time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases stochastic neural networks, approximate counting, synchronizer

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.57

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.10369`.

Funding *Merav Parter*: Supported in part by BSF-NSF grants.

Acknowledgements We are grateful to Cameron Musco, Renan Gross and Eylon Yogev for various useful discussions.

1 Introduction

Understanding the mechanisms by which brain experiences time is one of the major research objectives in neuroscience [26, 2, 9]. Humans measure time using a global clock based on standardized units of minutes, days and years. In contrast, the brain perceives time using specialized neural clocks that define their own time units. Living organisms have various other implementations of biological clocks, a notable example is the circadian clock that gets synchronized with the rhythms of a day.

In this paper we consider the algorithmic aspects of measuring *time* in a simple yet biologically plausible model of *stochastic spiking neural networks* (SNN) [23, 24], in which neurons fire in discrete pulses, in response to a sufficiently high membrane potential. This model is believed to capture the spiking behavior observed in real neural networks, and has recently received quite a lot of attention in the algorithmic community [18, 19, 20, 17, 15, 28, 6]. In contrast to the common approach in computational neuroscience and machine learning, the focus here is not on general computation ability or broad learning tasks, but rather on specific algorithmic implementation and analysis.

The SNN network is represented by a directed weighted graph $G = (V, A, W)$, with a special set of neurons $X \subset V$ called *inputs* that have no incoming edges, and a subset of *output* neurons[1] $Y \subset V$. The neurons in the network can be either deterministic threshold gates or probabilistic threshold gates. As observed in biological networks, and departing from many artificial network models, neurons are either strictly inhibitory (all outgoing edge weights are negative) or excitatory (all outgoing edge weights are positive). The network evolves in discrete, synchronous *rounds* as a Markov chain, where the firing probability of every neuron in round τ depends on the firing status of its neighbors in the preceding round $\tau - 1$. For probabilistic threshold gates this firing is modeled using a standard sigmoid function. Observe that an SNN network is in fact, a *distributed network*, every neuron responds to the firing spikes of its *neighbors*, while having no global information on the entire network.

Remark. In the setting of SNN, unlike classical distributed algorithms (e.g., LOCAL or CONGEST), the algorithm is fully specified by the *structure* of the network. That is, for a given network, its dynamic is fully determined by the model. Hence, the key complexity measure here is the size of the network measured by the number of auxiliary neurons[2]. For certain problems, we also care for the tradeoff between the size and the computation time.

1.1 Measuring Time with Spiking Neural Networks

We consider the algorithmic challenges of measuring time using networks of threshold gates and probabilistic threshold gates. We introduce the *neural timer* problem defined as follows:

> Given an input neuron x, an output neuron y, and a time parameter t, it is required to design a small neural network such that any firing of x in a given round invokes the firing of y for exactly the next t rounds.

In other words, it is required to design a succinct timer, activated by the firing of its input neuron, that alerts when exactly t rounds have passed.

[1] In contrast to the definition of *circuits*, we do allow output neurons to have outgoing edges and self loops. The requirement will be that the value of the output neurons converges over time to the desired solution.
[2] I.e., neurons that are not the input or the output neurons.

A trivial solution with t auxiliary neurons can be obtained by taking a directed chain of length t (Fig. 1): the head of the chain has an incoming edge from the input x, the output y has incoming edges from the input x, and all the other t neurons on the chain. All these neurons are simple OR-gates, they fire in round τ if at least one of their incoming neighbors fired in round $\tau - 1$. Starting with the firing of x in round 0, in each round i, exactly one neuron, namely the i^{th} neuron on the chain fires, which makes y keep on firing for exactly t rounds until the chain fades out. In this basic solution, the network spends one neuron that counts +1 and dies. It is noteworthy that the neurons in our model are very simple, they do not have any memory, and thus cannot keep track of the firing history. They can only base their firing decisions on the firing of their neighbors in the *previous* round.

With such a minimal model of computation, it is therefore intriguing to ask how to beat this linear dependency (of network size) in the time parameter t. Can we count to ten using only two (memory-less) neurons? We answer this question in the affirmative, and show that even with just simple deterministic threshold gates, we can measure time up to t rounds using only $O(\log t)$ neurons. It is easy to see that this bound is tight when using deterministic neurons (even when allowing some approximation). The reason is that $o(\log t)$ neurons encode strictly less than t distinct configurations, thus in a sequence of t rounds, there must be a configuration that re-occurs, hence locking the system into a state in which y fires forever.

▶ **Theorem 1** (Deterministic Timers). *For every input time parameter $t \in \mathbb{N}_{>0}$, (1) there exists a deterministic neural timer network $\mathcal{N}$ with $O(\log t)$ deterministic threshold gates, (2) any deterministic neural timer requires $\Omega(\log t)$ neurons.*

This timer can be easily adapted to the related problem of *counting*, where the network should output the number of spikes (by the input x) within a time window of t rounds.

Does Randomness Help in Time Estimation? Neural computation in general, and neural spike responses in particular, are inherently stochastic [16]. One of our broader scope agenda is to understand the power and limitations of randomness in neural networks. Does neural computation become *easier* or *harder* due to the stochastic behavior of the neurons?

We define a randomized version of the neural timer problem that allows some slackness both in the approximation of the time, as well as allowing a small error probability. For a given error probability $\delta \in (0, 1)$, the output y should fire for at least t rounds, and must stop firing after at most $2t$ rounds[3] with probability at least $1 - \delta$. It turns out that this randomized variant leads to a considerably improved solution for $\delta = 2^{-O(t)}$:

▶ **Theorem 2** (Upper Bound for Randomized Timers). *For every time parameter $t \in \mathbb{N}_{>0}$, and error probability $\delta \in (0, 1)$, there exists a probabilistic neural timer network $\mathcal{N}$ with $O(\min\{\log\log 1/\delta, \log t\})$ deterministic threshold gates plus* additional *random spiking neuron.*

Our starting point is a simple network with $O(\log 1/\delta)$ neurons, each firing independently with probability $1 - 1/t$. The key observation for improving the size bound into $O(\log\log 1/\delta)$ is to use the *time axis*: we will use a *single* neuron to generate random samples over time, rather than having *many* random neurons generating these samples in a *single* round. The deterministic neural counter network with time parameter of $O(\log 1/\delta)$ is used as a building block in order to gather the firing statistics of a single spiking neuron. In light of the $\Omega(\log t)$

[3] Taking $2t$ is arbitrary here, and any other constant would work as well.

lower bound for deterministic networks, we get the first separation between deterministic and randomized solutions for error probability $\delta = \omega(1/2^t)$. This shows that randomness can help, but up to a limit: Once the allowed error probability is exponentially small in t, the deterministic solution is the best possible. Perhaps surprisingly, we show that this behavior is tight:

▶ **Theorem 3** (Lower Bound for Randomized Timers). *Any SNN network for the neural timer problem with time parameter t, and error $\delta \in (0,1)$ must use $\Omega(\min\{\log\log 1/\delta, \log t\})$ neurons.*

Neural Counters. Spiking neurons are believed to encode information via their firing rates. This underlies the *rate coding* scheme [1, 30, 11] in which the spike-count of the neuron in a given span of time is interpreted as a *letter* in a larger alphabet. In a network of memory-less spiking neurons, it is not so clear how to implement this rate dependent behavior. How can a neuron convey a complicated message over time if its neighboring neurons remember only its recent spike? This challenge is formalized by the following neural counter problem: Given an input neuron x, a time parameter t, and $\Theta(\log t)$ output neurons represented by a vector $\bar{y}$, it is required to design a neural network such that the output vector $\bar{y}$ holds the binary representation of the number of times that x fired in a sequence of t rounds. As we already mentioned this problem is very much related to the neural timer problem and can be solved using $O(\log t)$ neurons. Can we do better?

The problem of maintaining a *counter* using a small amount of space has received a lot of attention in the *dynamic streaming* community. The well-known Morris algorithm [27, 10] maintains an approximate counter for t counts using only $\log\log t$ bits. The high-level idea of this algorithm is to increase the counter with probability of $1/2^{C'}$ where C' is the current read of the counter. The counter then holds the exponent of the number of counts. By following ideas of [10], carefully adapted to the neural setting, we show:

▶ **Theorem 4** (Approximate Counting). *For every time parameter t, and $\delta \in (0,1)$, there exists a randomized construction of approximate counting network using $O(\log\log t + \log(1/\delta))$ deterministic threshold gates plus an additional* single *random spiking neuron, that computes an $O(1)$ (multiplicative) approximation for the number of input spikes in t rounds with probability $1-\delta$.*

We note that unlike the deterministic construction of timers that could be easily adopted to the problem of neural counting, our optimized randomized timers with $O(\log\log 1/\delta)$ neurons cannot be adopted into an approximate counter network. We therefore solve the latter by adopting Morris algorithm to the neural setting.

Broader Scope: Lessons From Dynamic Streaming Algorithms. We believe that approximate counting problem provides just one indication for the potential relation between succinct neural networks and dynamic streaming algorithms. In both settings, the goal is to gather statistics (e.g., over time) using a small amount of space. In the setting of neural network there are additional difficulties that do not show up in the streaming setting. E.g., it is also required to obtain fast *update time*, as illustrated in our solution to the approximate counting problem.

1.2 Neural Synchronizers

The standard model of spiking neural networks assumes that all edges (synapses) in the network have a uniform response latency. That is, the electrical signal is passed from the presynaptic neuron to the postsynaptic neuron within a fixed time unit which we call a

round. However, in real biological networks, the response latency of synapses can vary considerably depending on the biological properties of the synapse, as well as on the distance between the neighboring neurons. This results in an asynchronous setting in which different edges have distinct response time. We formalize a simple model of spiking neurons in the asynchronous setting, in which the given neural network also specifies a *response latency* function $\ell : A \to \mathbb{R}_{\geq 1}$ that determines the number of rounds it takes for the signal to propagate over the edge. Inspired by the synchronizers of Awerbuch and Peleg [4], and using the above mentioned compressed timer and counter modules, we present a general simulation methodology (a.k.a synchronizers) that takes a network $\mathcal{N}_{\mathsf{sync}}$ that solves the problem in the synchronized setting, and transform it into an "analogous" network $\mathcal{N}_{\mathsf{async}}$ that solves the same problem in the asynchronous setting.

The basic building blocks of this transformation is the neural time component adapted to the asynchronous setting. The cost of the transformation is measured by the overhead in the number of neurons and in the computation time. Using our neural timers leads to a small overhead in the number of neurons.

▶ **Theorem 5** (Synchronizer, Informal). *There exists a synchronizer that given a network $\mathcal{N}_{\mathsf{sync}}$ with n neurons and maximum response latency*[4] *L, constructs a network $\mathcal{N}_{\mathsf{async}}$ that has an "analogous" execution in the asynchronous setting with a total number of $O(n + L \log L)$ neurons and a time overhead of $O(L^3)$.*

We note that although the construction is inspired by the work of Awerbuch and Peleg [4], due to the large differences between these models, the precise formulation and implementation of our synchronizers are quite different. The most notable difference between the distributed and neural setting is the issue of memory: in the distributed setting, nodes can aggregate the incoming messages and respond when all required messages have arrived. In strike contrast, our neurons can only respond (by either firing or not firing) to signals arrived in the *previous* round, and all signals from previous rounds cannot be locally stored. For this reason and unlike [4], we must assume a bound on the largest edge latency. In particular, in the full version we show that the size overhead of the transformed network $\mathcal{N}_{\mathsf{async}}$ must depend, at least logarithmically, on the value of the largest latency L.

▶ **Observation 1.** *The size overhead of any synchronization scheme is $\Omega(\log L)$.*

This provably illustrates the difference in the overhead of synchronization between general distributed networks and neural networks. We leave the problem of tightening this lower bound (or upper bound) as an interesting open problem.

Additional Related Work. To the best of our knowledge, there are two main previous theoretical work on asynchronous neural networks. Maass [22] considered a quite elaborated model for deterministic neural networks with *arbitrary* response *functions* for the edges, along with latencies that can be chosen by the network designer. Within this generalized framework, he presented a coarse description of a synchronization scheme that consists of various time modules (e.g., initiation and delay modules). Our work complements the scheme of [22] in the simplified SNN model by providing a rigorous implementation and analysis for size and time overhead. Khun et al. [14] analyzed the synchronous and asynchronous behavior under the stochastic neural network model of DeVille and Peskin [7]. Their model and framework is quite different from ours, and does not aim at building synchronizers.

[4] I.e., L correspond to the *length* of the longest round.

Turning to the setting of logical circuits, there is a long line of work on the asynchronous setting under various model assumptions [3, 12, 29, 5, 25] that do not quite fit the memory-less setting of spiking neurons.

Comparison with Concurrent Work [31]. Independently to our work, Wang and Lynch proposed a similar construction for the neural counter problem. Their work restricts attention to deterministic threshold gates and do not consider the neural timer problem and synchronizers which constitute the main contribution of our paper. We note that our approximate counter solution with $O(\log\log t + \log(1/\delta))$ neurons resolves the open problem stated in [31].

1.3 Preliminaries

We start by defining our model along with useful notation.

A Neuron. A *deterministic* neuron u is modeled by a *deterministic* threshold gate. Letting $b(u)$ to be the threshold value of u. Then it outputs 1 if the weighted sum of its incoming neighbors exceeds $b(u)$. A *spiking neuron* is modeled by a probabilistic threshold gate that fires with a sigmoidal probability $p(x) = \frac{1}{1+e^{-x}}$ where x is the difference between the weighted incoming sum of u and its threshold $b(u)$.

Neural Network Definition. A *Neural Network* (NN) $\mathcal{N} = \langle X, Z, Y, w, b\rangle$ consists of n input neurons $X = \{x_1, \ldots, x_n\}$, m output neurons $Y = \{y_1, \ldots, y_m\}$, and ℓ auxiliary neurons $Z = \{z_1, ..., z_\ell\}$. In a *deterministic* neural network (DNN) all neurons are deterministic threshold gates. In spiking neural network (SNN), the neurons can be either deterministic threshold gates or probabilistic threshold gates. The directed weighted synaptic connections between $V = X \cup Z \cup Y$ are described by the weight function $w : V \times V \to \mathbb{R}$. A weight $w(u, v) = 0$ indicates that a connection is not present between neurons u and v. Finally, for any neuron v, $b(v) \in \mathbb{R}_{\geq 0}$ is the threshold value (activation bias). The weight function defining the synapses is restricted in two ways. The in-degree of every input neuron x_i is zero, i.e., $w(u, x) = 0$ for all $u \in V$ and $x \in X$. 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.

Network Dynamics. The network evolves in discrete, synchronous rounds as a Markov chain. The firing probability of every neuron in round τ depends on the firing status of its neighbors in round $\tau - 1$, via a standard sigmoid function, with details given below. For each neuron u, and each round $\tau \geq 0$, let $u^\tau = 1$ if u fires (i.e., generates a spike) in round τ. Let u^0 denote the initial firing state of the neuron. The firing state of each input neuron x_j in each round is the input to the network. For each non-input neuron u and every round $\tau \geq 1$, let $pot(u, \tau)$ denote the membrane potential at round τ and $p(u, \tau)$ denote the firing probability ($\Pr[u^\tau = 1]$), calculated as:

$$pot(u,\tau) = \sum_{v \in V} w_{v,u} \cdot v^{\tau-1} - b(u) \text{ and } p(u,\tau) = \frac{1}{1 + e^{-\frac{pot(u,\tau)}{\lambda}}} \tag{1}$$

where $\lambda > 0$ is a *temperature parameter* which determines the steepness of the sigmoid. Clearly, λ does not affect the computational power of the network (due to scaling of edge weights and thresholds), thus we set $\lambda = 1$. In *deterministic* neural networks (DNN), each neuron u is a deterministic threshold gate that fires in round τ iff $pot(u, \tau) \geq 0$.

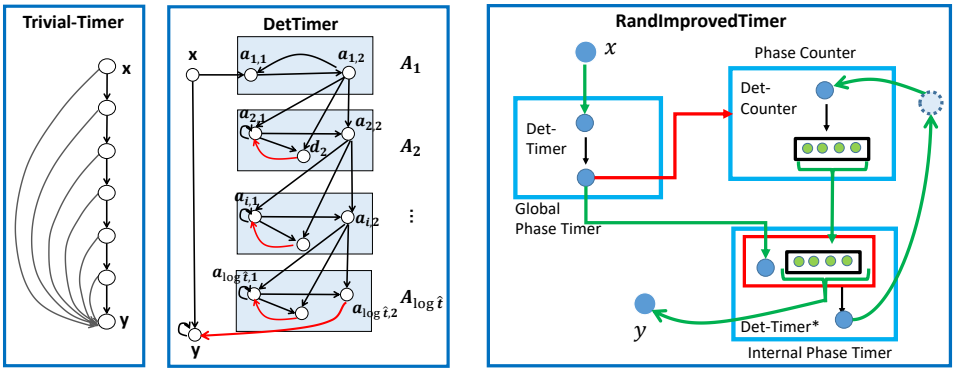

Figure 1 Illustration of timer networks with time parameter t. Left: The naïve timer with $\Theta(t)$ neurons. Mid: deterministic timer with $\Theta(\log t)$ neurons. Right: randomized timer with $O(\log\log 1/\delta))$ neurons, using the DetTimer modules with parameter $t' = \log 1/\delta$.

Network States (Configurations). Given a network $\mathcal{N}$ (either a DNN or SNN) with N neurons, the configuration (or *state*) of the network in time τ denoted as s_τ can be described as an N-length binary vector indicating which neuron fired in round τ.

The Memoryless Property. The neural networks have a memoryless property, in the sense that each state depends only on the state of the previous round. In a DNN network, the state $s_{\tau-1}$ fully determines s_τ. In an SNN network, for every fixed state s^* it holds $\Pr[s_\tau = s^* \mid s_1, \ldots s_{\tau-1}] = \Pr[s_\tau = s^* \mid s_{\tau-1}]$. Moreover for any $\tau, \tau', r > 0$, it holds that $\Pr[s_{\tau+r} = s^* \mid s_\tau] = \Pr[s_{\tau'+r} = s^* \mid s_{\tau'}]$.

Hard-Wired Inputs. We consider neural networks that *solve* a given parametrized problem (e.g., neural timer with time parameter t). The parameter to the problem can be either hard-wired in the network or alternatively be given as part of the input layer to the network. In most of our constructions, the time parameter is hard-wired. In some cases, we also show constructions with soft-wiring.

2 Deterministic Constructions of Neural Timer Networks

As a warm-up, we start by considering deterministic neural timers.

▶ **Definition 6** (Det. Neural Timer Network). *Given time parameter t, a* deterministic *neural timer network $\mathcal{DT}$ is a network of threshold gates, with an input neuron x, an output neuron y, and additional auxiliary neurons. The network satisfies that in every round τ, $y^\tau = 1$ iff there exists a round $\tau > \tau' \geq \tau - t$ such that $x^{\tau'} = 1$.*

Lower Bound (Pf. of Thm. 1(2)). For a given neural timer network $\mathcal{N}$ with N auxiliary neurons, recall that the *state* of the network in round τ is described by an N-length vector indicating the firing neurons in that round. Assume towards contradiction that there exists a neural timer with $N \leq \log t - 1$ auxiliary neurons. Since there are at most 2^N different states, by the pigeonhole principle, there must be at least two rounds $\tau, \tau' \leq t - 1$ in which the state of the network is identical, i.e., where $s_\tau = s_{\tau'} = s^*$ for some $s^* \in \{0,1\}^N$. By the correctness of the network, the output neuron y fires in all rounds $\tau'' \in [\tau + 1, \tau' + 1]$. By the memoryless property, we get that $s_{\tau''} = s^*$ for $\tau'' = \tau + i \cdot (\tau' - \tau)$ for every $i \in \mathbb{N}_{\geq 0}$.

Thus y continues firing forever, in contradiction that it stops firing after t rounds. Note that this lower bound holds even if y is allowed to stop firing in any finite time window.

A Matching Upper Bound (Pf. Thm. 1(1)). For ease of explanation, we will sketch here the description of the network assuming that it is applied only once (i.e., the input x fires once within a window of t rounds). Taking care of the general case requires slight adaptations[5], see the full version for complete details.

At the high-level, the network consists of $k = \Theta(\log t)$ layers $A_1, \ldots, A_k$ each containing two excitatory neurons $a_{i,1}, a_{i,2}$ denoted as *counting* neurons, and one inhibitory neuron d_i. Each layer A_i gets its input from layer A_{i-1} for every $i \geq 2$, and A_1 gets its input from x. The role of each layer A_i is to count *two* firing events of the neuron $a_{i-1,2} \in A_{i-1}$. Thus the neuron $a_{\log t,2}$ counts $2^{\log t}$ rounds.

Because our network has an update time of $\log t$ rounds (i.e., number of rounds to update the timer), for a given time parameter t, the construction is based on the parameter $\hat{t}$ where $\hat{t} + \log \hat{t} = t$. We assume that $\hat{t}$ is a power of 2, the general case can also be solved with $O(\log t)$ neurons, the analysis is deferred to the full version.

- The first layer A_1 consists of two neurons $a_{1,1}, a_{1,2}$. The first neuron $a_{1,1}$ has positive incoming edges from x and $a_{1,2}$ with weights $w(x, a_{1,1}) = 3$, $w(a_{1,2}, a_{1,1}) = 1$, and threshold $b(a_{1,1}) = 1$. The second neuron $a_{1,2}$ has an incoming edge from $a_{1,1}$ with weight $w(a_{1,1}, a_{1,2}) = 1$ and threshold $b(a_{1,2}) = 1$. Because we have a loop going from $a_{1,1}$ to $a_{1,2}$ and back, once x fired $a_{1,2}$ will fire every two rounds.
- For every $i = 2 \ldots \log \hat{t}$, the i^{th} layer A_i contains 3 neurons, two *counting* neurons $a_{i,1}$, $a_{i,2}$ and a reset neuron d_i. The first neuron $a_{i,1}$ has positive incoming edges from $a_{i-1,2}$, and a self loop with weight $w(a_{i-1,2}, a_{i,1}) = w(a_{i,1}, a_{i,1}) = 1$, a negative incoming edge from d_i with weight $w(d_i, a_{i,1}) = -1$, and threshold $b(a_{i,1}) = 1$. The second counting neuron $a_{i,2}$ has incoming edges from $a_{i-1,2}$ and $a_{i,1}$ with weight $w(a_{i-1,2}, a_{i,2}) = w(a_{i,1}, a_{i,2}) = 1$, and threshold $b(a_{i,2}) = 2$. The reset neuron d_i is an inhibitor copy of $a_{i-1,2}$ and therefore also has incoming edges from $a_{i-1,2}$ and $a_{i,1}$ with weight $w(a_{i-1,2}, d_i) = w(a_{i,1}, d_i) = 1$ and threshold $b(d_i) = 2$. As a result, $a_{i,1}$ starts firing after $a_{i-1,2}$ fires once, and $a_{i,2}$fires after $a_{i-1,2}$ fires twice. Then the neuron d_i inhibits $a_{i,1}$ and the layer is ready for a new count.
- The output neuron y has a positive incoming edge from x as well as a self-loop with weights $w(x, y) = 2$, $w(y, y) = 1$. In addition, it has a negative incoming edge from the last counting neuron $a_{\log \hat{t},2}$ with weight $w(a_{\log \hat{t},2}, y) = -1$ and threshold $b(y) = 1$. Hence, after x fires the output y continues to fire as long as $a_{\log \hat{t},2}$ did not fire.
- The last counting neuron $a_{\log \hat{t},2}$ also have negative outgoing edges to all counting neurons (neurons of the form $a_{i,j}$) with weight $w(a_{\log \hat{t},2}, a_{i,j}) = -2$. As a result, after the timer counts t rounds it is reset.

Timer with Time Parameter. In the full version we show a slight modified variant of neural timer denoted by $\mathsf{DetTimer}^*$ which also receives as input an additional set of $\log t$ neurons that encode the desired duration of the timer. This modified variant is used in our improved randomized constructions.

[5] I.e., whenever x fires again in a window of t rounds, one should reset the timer and start counting t rounds from that point on.

Neural Counters. We also show a modification of the timer into a counter network DetCounter that instead of counting the number of rounds, counts the number of input spikes in a time interval of t rounds. For complete details we defer the reader to the full version.

▶ **Lemma 7.** *Given time parameter t, there exists a deterministic* neural counter *network which has an input neuron x, a collection of $\log t$ output neurons represented by a vector $\bar{y}$, and $O(\log t)$ additional auxiliary neurons. In a time window of t rounds, for every round τ, if x fired r_τ times in the last τ rounds, the output $\bar{y}$ encodes r_τ by round $\tau + \log r_\tau + 1$.*

This extra-additive factor of $\log r_\tau$ is due to the update time of the counter. In addition, in the full version we revisit the neural counter problem and provide an *approximate* randomized solution with $O(\log\log t + \log(1/\delta))$ many neurons where δ is the error parameter. This construction is based on the well-known Morris algorithm (using the analysis of [10]) for approximate counting in the streaming model.

3 Randomized Constructions of Neural Timer Networks

We now turn to consider randomized implementations. The input to the construction is a time parameter t and an error probability $\delta \in (0,1)$, that are hard-wired into the network.

▶ **Definition 8** (Rand. Neural Timer Network)**.** *A randomized neural timer $\mathcal{RT}$ for parameters $t \in \mathbb{N}_{>0}$ and $\delta \in (0,1)$, satisfies the following for a time window of* $\mathsf{poly}(t)$ *rounds.*

- *For every fixed firing event of x in round τ, with probability $1-\delta$, y fires in each of the following t rounds.*
- *$y^{\tau'} = 0$ for every round τ' such that $\tau' - \mathsf{Last}(\tau') \geq 2t$ with probability $1-\delta$, where $\mathsf{Last}(\tau') = \max\{i \leq \tau' \mid x^i = 1\}$ is the last round τ in which x fired up to round τ'.*

Note that in our definition, we have a success guarantee of $1-\delta$ for any fixed firing event of x, on the event that y fires for t many rounds after this firing. In contrast, with probability of $1-\delta$ over the entire span of $\mathsf{poly}(t)$ rounds, y *does not* fire in cases where the last firing of x was $2t$ rounds apart. We start by showing a simple construction with $O(\log 1/\delta)$ neurons.

3.1 Warm Up: Randomized Timer with $O(\log 1/\delta)$ Neurons

The network $\mathsf{BasicRandTimer}(t,\delta)$ contains a collection of $\ell = \Theta(\log 1/\delta)$ spiking neurons $A = \{a_1, \ldots, a_\ell\}$ that can be viewed as a *time-estimator population*. Each of these neurons have a positive self loop, a positive incoming edge from the input neuron x, and a positive outgoing edge to the output neuron y. See Figure 2 for an illustration. Whereas these a_i neurons are probabilistic spiking neurons[6], the output y is simply a threshold gate. We next explain the underlying intuition. Assume that the input x fired in round 0. It is then required for the output neuron y to fire for at least t rounds $1, \ldots, t$, and stop firing after at most $2t$ rounds with probability $1-\delta$. By having every neuron a_i firing (independently) w.p $(1-1/t)$ in each round given that it fired in the previous round[7], we get that a_i fires for t *consecutive* rounds with probability $(1-1/t)^t = 1/e$. On the other hand, it fires for $2t$ consecutive rounds with probability $(1-1/t)^{2t} = 1/e^2$. Since we have $\Theta(\log 1/\delta)$ many neurons, by a simple application of Chernoff bound, the output neuron y (which simply counts the number of firing neurons in A) can distinguish between round t and round $2t$ with probability $1-\delta$, see the full version for the complete proof.

[6] A neuron that fires with a probability specified in Eq. (1)
[7] A neuron a_i that stops firing in a given round, drops out and would not fire again with good probability.

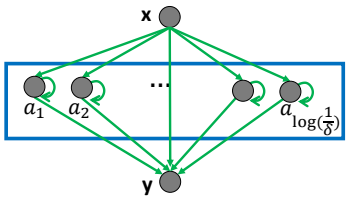

Figure 2 Illustration of the BasicRandTimer(t, δ) network. Each neuron a_i fires with probability $1 - 1/t$ in round τ given that it fired in the previous round $\tau - 1$, and therefore fires for t consecutive rounds with constant probability. The output y fires if at least $1/(2e)$ fraction of the a_i neurons fired in the previous round.

3.2 Improved Construction with $O(\log\log 1/\delta)$ Neurons

We next describe an optimal randomized timer RandImprovedTimer with an exponentially improved number of auxiliary neurons. This construction also enjoys the fact that it requires a *single* spiking neuron, while the remaining neurons can be deterministic threshold gates. Due to the tightness of Chernoff bound, one cannot really hope to estimate time with probability $1 - \delta$ using $o(\log(1/\delta))$ samples. Our key idea here is to generate the same number of samples by re-sampling one particular neuron over several rounds. Intuitively, we are going to show that for our purposes having $\ell = \log(1/\delta)$ neurons $a_1, \ldots, a_\ell$ firing with probability $1 - 1/t$ in a *given* round is *equivalent* to having a *single* neuron a^* firing with probability $1 - 1/t$ (independently) in a sequence of ℓ rounds.

Specifically, observe that the distinction between round t and $2t$ in the BasicRandTimer network is based only on the *number* of spiking neurons in a given round. In addition, the distribution on the number of times a^* fires in a span of ℓ rounds is equivalent to the distribution on the number of firing neurons $a_1, \ldots, a_\ell$ in a given round. For this reason, every phase of RandImprovedTimer simulates a single round of BasicRandTimer. To count the number of firing events in ℓ rounds, we use the deterministic neural counter module with $\log \ell = O(\log\log 1/\delta)$ neurons.

We now further formalize this intuition. The network RandImprovedTimer simulates each round of BasicRandTimer using a phase of $\ell' = \Theta(\log 1/\delta)$ rounds [8], but with only $O(\log\log 1/\delta)$ neurons. In the BasicRandTimer network each of the neurons a_i fires (independently) in each round w.p $1 - 1/t$. Once it stops firing in a given round, it basically drops out and would not fire again with good probability. Formally, consider an execution of the BasicRandTimer and let n_i be the number of neurons in A that fired in round i. In round $i + 1$ of this execution, we have n_i many neurons each firing w.p $1 - 1/t$ (while the remaining neurons in A fire with a very small probability). In the corresponding $i + 1$ phase of the network RandImprovedTimer, the chief neuron a^* fires w.p $1 - 1/t'$ where $t' = \frac{t}{\ell'}$ for $n'_i \leq \ell$ consecutive rounds[9] where n'_i is the number of rounds in which a^* fired in phase i.

The dynamics of the network RandImprovedTimer is based on discrete phases. Each phase has a fixed number of $\ell' = O(\ell)$ rounds, but has a possibly different number of *active rounds*, namely, rounds in which a^* attempts firing. Specifically, a phase i has an active part of n'_i rounds where n'_i is the number of rounds in which a^* fired in phase $i - 1$. In the remaining $\ell' - n'_i$ rounds of that phase, a^* is idle. To implement this behavior, the network should keep

[8] Due to tactical reasons each phase consists of $\ell' = \ell + \log \ell$ rounds instead of ℓ.

[9] Note that because each phase takes $\ell' = \Theta(\log 1/\delta)$ rounds, we will need to count $t' = \frac{t}{\ell'}$ many phases. Thus a^* fires with probability $1 - 1/t'$ rather then w.p $1 - 1/t$.

track of the number of rounds in which a^* fires in each phase, and supply it as an input to the next phase (as it determines the length of the active part of that phase). For that purpose we will use the deterministic modules of neural timers and counters. The module DetCounter with time parameter $\Theta(\log 1/\delta)$ is responsible for counting the number of rounds that a^* fires in a given phase i. The output of this module at the end of the phase is the input to a DetTimer* module[10] in the beginning of phase $i+1$. In addition, we also need a *phase timer* module DetTimer with time parameter $\Theta(\log 1/\delta)$ that "announces" the end of a phase and the beginning of a new one. Similarly to the network BasicRandTimer, the output neuron y fires as long as a^* fires for at least $(1/2e)$ fraction of the rounds in each phase (in an analogous manner as in the BasicRandTimer construction). See Fig. 1 for an illustration of the network. Note that since we only use deterministic modules with time parameter $\Theta(\log 1/\delta)$, the total number of neurons (which are all threshold gates) will be bounded by $O(\log\log 1/\delta)$. See the full version for the complete proof of Thm. 2.

Remark. We note that the fact that one can get a randomized upper bound of $O(\log\log 1/\delta)$ neurons has to do with the fact that our spiking neurons can be set to fire with probability $1-1/t$. Therefore the value of t is hard-wired in the network. We also note that in a more restrictive setting where neurons are simple fair coins that fire with probability half in each round, the size complexity might be dependent in t.

3.3 A Matching Lower Bound

We now turn to provide a proof sketch for Thm. 2. The full proof can be found in the full version. Assume towards contradiction that there exists a randomized neural timer $\mathcal{N}$ for a given time parameter t with $N = o(\log\log 1/\delta)$ neurons that succeeds with probability at least $1-\delta$. This implies that there exists some constant $c \geq 2$ such that y stops firing after $(c-1)\cdot t$ rounds w.p $1-\delta$. Since we have N many neurons, the number of distinct states (or configurations) is bounded by $S = 2^N = o(\log 1/\delta)$. In particular, we will adjust the constant in the number of neurons N such that $S \leq \frac{\log 1/\delta}{\log\log 1/\delta}$. Since $t > \log(1/\delta)$ [11], in every execution of $\mathcal{N}$ for at least t rounds, there must be a state that occurs at least *twice* during the execution. Moreover, the sequence of t rounds consists of at least $2S$ disjoint intervals each of length $t/3S$. We therefore get that in every execution of the network, there must be a state that occurs at least twice for some rounds t', t'' such that $t''-t' > t/3S$. In other words, we have the guarantee that there is always some state that reoccurs after a sufficiently large number of rounds. Since there are at most S configurations, we conclude that there must be one particular configuration s^* for which the probability to reoccur (after at least $t/(3S)$ rounds) is at least $1/S = \Omega(1/\log(1/\delta))$.

Let Π be the family of all $(c\cdot t)$-length executions of $\mathcal{N}$. We now restrict attention to all those executions in which s^* reoccurs within the first t rounds, with spacing of $\Omega(t/\log(1/\delta))$ rounds between its appearances[12]. We call the subset $\Pi^* \subseteq \Pi$ of those executions *special*. In addition, an execution in Π is *good* if y fires in *each* of the first t rounds, otherwise it is *bad*.

We next claim that the probability of an execution to be both special and good is at least $p^* = \Omega(1/\log(1/\delta) - \delta)$. First, by the definition of Π^*, the probability of an execution to be special is at least $1/S = \Omega(1/\log(1/\delta))$. In addition, by the success guarantee of $\mathcal{N}$, y fires in the first t rounds w.p. at least $1-\delta$. Thus by the union bound, we get that the probability of a special and good execution is $1/S - \delta$. This allows us to also conclude that given that s^*

[10] Here we use the variant of DetTimer in which the time is encoded in the input layer of the network.
[11] The lower bound is meaningful only when $\delta = 2^{-O(t)}$.
[12] We do allow s^* to appear several times within this interval.

appears in some round $t' < t - t/3S$, with probability of at least p^* the following happens: (1) s^* reoccurs in some round t'' such that $t'' - t \in [t/3S, t]$ and (2) y fires during the entire interval $[t', t'']$. We now use this argument to conclude that w.p. at least δ, the output y fires for at least $c \cdot t$ rounds, which will lead to a contradiction. To see this claim, note that w.p. at least $1/S$ there is a round $t' < t - t/3S$ in which s^* appears. With probability at least p^* there is a round $t'' > t + t/3S$ in which s^* appears again and y fires in each of the rounds in $[t', t'']$. A time window of $c \cdot t$ rounds contains at most $3c \cdot S$ intervals of length $t/3S$. Thus by the memory-less property, we get that the probability s^* reoccurs every $[t/3S, t]$ rounds and that y fires after ct rounds is at least $1/S \cdot (p^*)^{3c \cdot S} > \delta$, contradiction as y fires after $c \cdot t$ rounds w.p at most δ.

4 Applications to Synchronizers

The Asynchronous Setting. In this setting, the neural network $\mathcal{N} = \langle X, Z, Y, w, b \rangle$ also specifies a response latency function $\ell : A \to \mathbb{N}_{>0}$. For ease of notation, we normalize all latency values such that $\min_{e \in A} \ell(e) = 1$ and denote the maximum response latency by $L = \max_{e \in A} \ell(e)$. Supported by biological evidence [13], we assume that self-loop edges (a.k.a. autapses) have the minimal latency in the network, that $\ell((u,u)) = 1$ for self-edges (u,u). This assumption is crucial in our design[13]. Indeed the exceptional short latency of self-loop edges has been shown to play a critical role in biological network synchronization [21, 8]. The dynamics proceeds in synchronous rounds and phases. The length of a round corresponds to the minimum edge latency, this is why we normalize the latency values so that $\min_{e \in A} \ell(e) = 1$. If neuron u fires in round τ, its endpoint v receives u's signal in round $\tau + \ell(e)$. Formally, a neuron u fires in round τ with probability $p(u, \tau)$:

$$pot(u,\tau) = \sum_{v \in X \cup Z \cup Y} w_{v,u} \cdot v^{\tau - \ell(u,v)} - b(u) \text{ and } p(u,\tau) = \frac{1}{1 + e^{-\frac{pot(u,\tau)}{\lambda}}} \tag{2}$$

Synchronizer. A synchronizer ν is an algorithm that gets as input a network $\mathcal{N}_{\mathsf{sync}}$ and outputs a network $\mathcal{N}_{\mathsf{async}} = \nu(\mathcal{N}_{\mathsf{sync}})$ such that $V(\mathcal{N}_{\mathsf{sync}}) \subseteq V(\mathcal{N}_{\mathsf{async}})$ where $V(\mathcal{N})$ denotes the neurons of a network $\mathcal{N}$. The network $\mathcal{N}_{\mathsf{async}}$ works in the asynchronous setting and should have *similar execution* to $\mathcal{N}_{\mathsf{sync}}$ in the sense that for every neuron $v \in V(\mathcal{N}_{\mathsf{sync}})$, the firing pattern of v in the asynchronous network should be similar to the one in the synchronous network. The output network $\mathcal{N}_{\mathsf{async}}$ simulates each round of the network $\mathcal{N}_{\mathsf{sync}}$ as a *phase*.

▶ **Definition 9** (Pulse Generator and Phases). *A* pulse generator *is a module that fires to declare the end of each phase. Denote by $t(v,p)$ the (global) round in which neuron v receives the p^{th} spike from the pulse generator. We say that v is in phase p during all rounds $\tau \in [t(v,p-1), t(v,p)]$.*

▶ **Definition 10** (Similar Execution (Deterministic Networks)). *The* synchronous *execution Π_{sync} of a deterministic network $\mathcal{N}_{\mathsf{sync}}$ is specified by a list of states $\Pi_{\mathsf{sync}} = \{\sigma_1, \ldots, \}$ where each σ_i is a binary vector describing the firing status of the neurons in round i. The* asynchronous *execution of network $\mathcal{N}_{\mathsf{async}}$ denoted by Π_{async} is defined analogously only when applying the asynchronous dynamics (of Eq. (2)). The execution Π_{async} is divided into phases of fixed length. The networks $\mathcal{N}_{\mathsf{sync}}$ and $\mathcal{N}_{\mathsf{async}}$ have a* similar execution *if $V(\mathcal{N}_{\mathsf{sync}}) \subseteq V(\mathcal{N}_{\mathsf{async}})$, and in addition, a neuron $v \in V(\mathcal{N}_{\mathsf{sync}})$ fires in round p in the execution Π_{sync} iff v fires during phase p in Π_{async}.*

[13] In a follow-up work, we actually show that this assumption is necessary for the existence of syncrnoizers even when $L = 2$.

For simplicity of explanation, we assume that the network $\mathcal{N}_{\mathsf{sync}}$ is deterministic. However, our scheme can easily capture randomized networks as well (i.e., by fixing the random bits in the synchronized simulation and feeding it to the async. one). See the full version for more details.

The Challenge. Consider a network of a threshold gate z with two incoming inputs: an excitatory neuron x, and an inhibitory neuron y. The weights are set such that z computes $X \wedge \bar{Y}$ thus it fires in round τ if x fired in round $\tau - 1$ and y did not fire. Implementing an $X \wedge \bar{Y}$ gate in the asynchronous setting is quite tricky. In the case where both x and y fire in round τ, in the synchronous network, z should not fire in round $\tau + 1$. However, in the asynchronous setting, if $\ell(x, z) < \ell(y, z)$, then z will mistakenly fire in round $\tau + \ell(x, z)$. This illustrates the need of enforcing a *delay* in the asynchronous simulation: the neurons should attempt firing only after receiving *all* their inputs from the previous phase. We handle this by introducing a pulse-generator module, that announces when it is safe to attempt firing.

To illustrate another source of challenge, consider the asynchronous implementation of an AND-gate $X \wedge Y$. If both x and y fire in round τ, then z fires in round $\tau + 1$ in the synchronous setting. However, if the latencies of the edges $\ell(x, z)$ and $\ell(y, z)$ are distinct, z receives the spike from x and y in *different* rounds, thus preventing the firing of z. Recall, that z has no memory, and thus its firing decision is based only on the potential level in the previous round. To overcome this hurdle, in the transformed network, each neuron in the original synchronous network is augmented with 3 copy-neurons, some of which have self-loops. Since self-loops have latency 1, once a neuron with a self-loop fires, it fires in the next round as well. This will make sure that the firing states of x and y are kept on being presented to z for sufficiently many rounds, which guarantees the existence of a round where both spikes arrive.

While solving one problem, introducing self-loops into the system brings along other troubles. Clearly, we would not want the neurons to fire forever, and at some point, those neurons should get inhibition to allow the beginning of a new phase. This calls for a delicate *reset* mechanism that cleans up the old firing states at the end of each phase, only after their values have already being used. Our final solution consists of global synchronization modules (e.g., pulse-generator, reset modules) that are inter-connected to a modified version of the synchronous network. Before explaining those constructions, we start by providing a modified neural timer $\mathsf{DetTimer}_{\mathsf{async}}$ adapted to asynchronous setting. This timer will be the basic building block in our global synchronization infrastructures.

Asynchronous Analog of DetTimer. A basic building block in our construction is a variant of DetTimer to the asynchronous setting. Observe that the DetTimer implementation of Sec. 2 might fail miserably in the asynchronous setting, e.g., when the edges $(a_{i-1,2}, a_{i,2})$ have latency 2 for every $i \geq 2$, and the remaining edges have latency 1, the timer will stop counting after $\Theta(\log t)$ rounds, rather than after t rounds. In the full version we show:

▶ **Lemma 11.** *[Neural Timer in the Asynchronous Setting] For a given time parameter t, there exists a deterministic network $\mathsf{DetTimer}_{\mathsf{async}}$ with $O(L \cdot \log t)$ neurons, satisfying that in the asynchronous setting with maximum latency L, the output neuron fires at least $\Theta(t)$ rounds, and at most $\Theta(L \cdot t)$ rounds after each firing of the input neuron.*

Description of the Syncronizer. The construction has two parts: a global infrastructure, that can be used to synchronize many networks[14], and an adaptation of the given network $\mathcal{N}_{\mathsf{sync}}$ into a network $\mathcal{N}_{\mathsf{async}}$. The global infrastructures consists of the following modules:

- A *pulse generator* PG implemented by $\mathsf{DetTimer}_{\mathsf{async}}$ with time parameter $\Theta(L^3)$.
- A *reset* module R_1 implemented by a directed chain of $\Theta(L)$ neurons [15] with input from the output neuron of the PG module.
- A *delay* module D implemented by $\mathsf{DetTimer}_{\mathsf{async}}$ with time parameter $\Theta(L^2)$ and input from the output of of the PG module.
- Another *reset* module R_2 implemented by a chain of $\Theta(L)$ neurons with input from D.

The heart of the construction is the pulse-generator that fires once within a fixed number of $\ell \in [\Theta(L^3), \Theta(L^4)]$ rounds, and invokes a cascade of activities at the end of each phase. When its output neuron g fires, it activates the reset and the delay modules, R_1 and D. The second reset module R_2 will be activated by the delay module D. Both reset modules R_1 and R_2 are implemented by chains of length L, with the last neuron on these chains being an *inhibitor* neuron. The role of the reset modules is to *erase* the firing states of some neurons (in $\mathcal{N}_{\mathsf{async}}$) from the previous phase, hence their output neuron is an inhibitor. The timing of this clean-up is very delicate, and therefore the reset modules are separated by a delay module that prevents a premature operation. The total number of neurons in these global modules is $O(L \cdot \log L)$. We next consider the specific modifications to the synchronous network $\mathcal{N}_{\mathsf{sync}}$ (see Fig. 3).

Modifications to the Network $\mathcal{N}_{\mathsf{sync}}$. The input layer and output layer in N_{async} are *exactly* as in $\mathcal{N}_{\mathsf{sync}}$. We will now focus on the set of *auxiliary* neurons V in $\mathcal{N}_{\mathsf{sync}}$. In the network $\mathcal{N}_{\mathsf{async}}$, each $v \in V$ is augmented by three additional neurons $v_{\mathsf{in}}, v_{\mathsf{delay}}$ and v_{out}. The incoming (resp., outgoing) neighbors to v_{in} (resp., v_{out}) are the out-copies (resp., in-copies) of all incoming (resp., outgoing) neighboring neurons of v. The neurons $v_{\mathsf{in}}, v, v_{\mathsf{delay}}$ and v_{out} are connected by a directed chain (in this order). Both v_{delay} and v_{out} have self-loops.

In case where the original network $\mathcal{N}_{\mathsf{sync}}$ contains spiking neurons, the neuron v_{in} will be given the exact same firing function as v in Π_{sync}. That is, in phase p, v_{in} will be given the random coins[16] used by v in round p in Π_{sync}. The other neurons v, v_{delay} and v_{out} are deterministic threshold gates. The role of the *out-copy* v_{out} is to *keep on presenting* the firing status of v from the previous phase $p-1$ throughout the rounds of phase p. This is achieved through their self-loops. The role of the *in-copy* v_{in} is to simulate the firing behavior of v in phase p. We will make sure that v_{in} fires in phase p only if v fires in round p in Π_{sync}. For this reason, we set the incoming edge weights of v_{in} as well as its bias to be exactly the same as that of v in $\mathcal{N}_{\mathsf{sync}}$. The neuron v is an AND gate of its in-copy v_{in} and the PG output g. Thus, we will make sure that v fires at the *end* of phase p only if v_{in} fires in this phase as well. The role of the delay copy v_{delay} is to delay the update of v_{out} to the up-to-date firing state of v (in phase p). Since both neurons v_{delay} and v_{out} have self-loops, at the end of each phase, we need to carefully reset their values (through inhibition). This is the role of the reset modules R_1 and R_2. Specifically, the reset module R_1 operated by the pulse-generator inhibits v_{out}. The second reset module R_2 inhibits the delay neuron v_{delay} only after we can be certain that its value has already being "copied" to v_{out}. Finally, we describe the connections of the neuron v_{out}. The neuron v_{out} has an incoming edge from the reset module R_1 with a

[14] It is indeed believed that the neural brain has centers of synchronization.
[15] Each neuron in the chain has an incoming edge from its preceding neuron with weight 1 and threshold 1.
[16] I.e., the random coins that are used to simulate the firing decision of v.

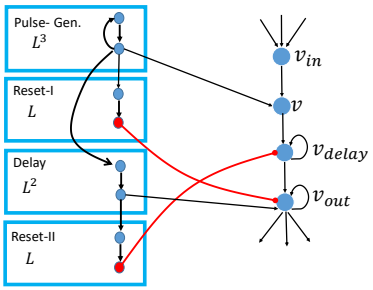

Figure 3 Illustration of the syncronizer modules. Left: global modules implemented by neural timers. Right: a neuron $v \in N_{\text{sync}}$ augmented by three additional neurons that interact with the global modules.

super-large weight. This makes sure that when the reset module is activated, v_{out} will be inhibited shortly after. In addition, it has a self-loop also of large weight (yet smaller than the inhibition edge) that makes sure that if v_{out} fires in a given round, and the reset module R_1 is not active, v_{out} also fires in the next round. Lastly, if v_{out} did not fire in the previous round, then it fires when receiving the spikes from *both* the delay module and from the delay copy v_{delay}. This will make sure that the firing state of v_{delay} will be copied to v_{out} only after the output of the delay module D fires. The complete analysis is given in the full version.

5 Open Problems

In this paper we introduce the problems of neural timer and neural counter in order to shed light into the way that neurons measure time in real biological neural networks. We believe that these timer and counting modules should be useful for many other computational tasks. The key application considered in this paper is for asynchronous computation. For that purpose we introduce a simplified asynchronous model. It would be interesting to delve into this setting and tighten the overhead in the number of neurons and computation time. Finally, exploring the connections between succinct neural networks and dynamic streaming algorithms is yet another promising research direction. The approximate counting problem already provides a positive indication for a potential relation between these models.

References

1 Edgar D Adrian. The impulses produced by sensory nerve endings. *The Journal of physiology*, 61(1):49–72, 1926.

2 Melissa J Allman, Sundeep Teki, Timothy D Griffiths, and Warren H Meck. Properties of the internal clock: first-and second-order principles of subjective time. *Annual review of psychology*, 65:743–771, 2014.

3 Douglas B Armstrong, Arthur D Friedman, and Premachandran R Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Transactions on Computers*, 100(12):1110–1120, 1969.

4 Baruch Awerbuch and David Peleg. Network Synchronization with Polylogarithmic Overhead. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 514–522, 1990.

5 Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1):1, 2006.

6 Chi-Ning Chou, Kai-Min Chung, and Chi-Jen Lu. On the Algorithmic Power of Spiking Neural Networks. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 26:1–26:20, 2019.

7 RE Lee DeVille and Charles S Peskin. Synchrony and asynchrony in a fully stochastic neural network. *Bulletin of mathematical biology*, 70(6):1608–1633, 2008.

8 Huawei Fan, Yafeng Wang, Hengtong Wang, Ying-Cheng Lai, and Xingang Wang. Autapses promote synchronization in neuronal networks. *Scientific reports*, 8(1):580, 2018.

9 Gerald T Finnerty, Michael N Shadlen, Mehrdad Jazayeri, Anna C Nobre, and Dean V Buonomano. Time in cortical circuits. *Journal of Neuroscience*, 35(41):13912–13916, 2015.

10 Philippe Flajolet. Approximate Counting: A Detailed Analysis. *BIT*, 25(1):113–134, 1985.

11 Wulfram Gerstner, Andreas K Kreiter, Henry Markram, and Andreas VM Herz. Neural codes: firing rates and beyond. *Proceedings of the National Academy of Sciences*, 94(24):12740–12741, 1997.

12 Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.

13 Kaori Ikeda and John M Bekkers. Autapses. *Current Biology*, 16(9):R308, 2006.

14 Fabian Kuhn, Joel Spencer, Konstantinos Panagiotou, and Angelika Steger. Synchrony and asynchrony in neural networks. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 949–964. SIAM, 2010.

15 Robert A. Legenstein, Wolfgang Maass, Christos H. Papadimitriou, and Santosh Srinivas Vempala. Long Term Memory and the Densest K-Subgraph Problem. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 57:1–57:15, 2018.

16 Benjamin Lindner. Some unsolved problems relating to noise in biological systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(01):P01008, 2009.

17 Nancy Lynch and Cameron Musco. A Basic Compositional Model for Spiking Neural Networks. *arXiv preprint*, 2018. `arXiv:1808.03884`.

18 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.

19 Nancy Lynch, Cameron Musco, and Merav Parter. Spiking Neural Networks: An Algorithmic Perspective. In *5th Workshop on Biological Distributed Algorithms (BDA 2017)*, July 2017.

20 Nancy A. Lynch, Cameron Musco, and Merav Parter. Neuro-RAM Unit with Applications to Similarity Testing and Compression in Spiking Neural Networks. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 33:1–33:16, 2017.

21 Jun Ma, Xinlin Song, Wuyin Jin, and Chuni Wang. Autapse-induced synchronization in a coupled neuronal network. *Chaos, Solitons & Fractals*, 80:31–38, 2015.

22 Wolfgang Maass. Lower Bounds for the Computational Power of Networks of Spiking Neurons. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(19), 1994. URL: `http://eccc.hpi-web.de/eccc-reports/1994/TR94-019/index.html`.

23 Wolfgang Maass. On the computational power of noisy spiking neurons. In *Advances in Neural Information Processing Systems 8 (NIPS)*, 1996.

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

25 Rajit Manohar and Yoram Moses. The eventual C-element theorem for delay-insensitive asynchronous circuits. In *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 102–109. IEEE, 2017.

26 Hugo Merchant, Deborah L Harrington, and Warren H Meck. Neural basis of the perception and estimation of time. *Annual review of neuroscience*, 36:313–336, 2013.

27 Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

28 Christos H Papadimitriou and Santosh S Vempala. Random projection in the brain and computation with assemblies of neurons. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

29 Jens Sparsø. Asynchronous circuit design-a tutorial. In *Chapters 1-8 in "Principles of asynchronous circuit design-A systems Perspective"*. Kluwer Academic Publishers, 2001.

30 Misha V Tsodyks and Henry Markram. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the national academy of sciences*, 94(2):719–723, 1997.

31 Barbeeba Wang and Nancy Lynch. Integrating Temporal Information to Spatial Information in a Neural Circuit. *arXiv preprint*, 2019. `arXiv:1903.01217`.