bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

Spike-based symbolic computations on bit strings and numbers

Ceca Kraišniković[1], Wolfgang Maass[1,*] Robert Legenstein[1,*]

September 13, 2021

[1]*Institute of Theoretical Computer Science, Graz University of Technology, Inffeldgasse 16b, Graz, Austria*
[*] *Joint last authors*
E-mail: {*ceca, maass, legi*}*@igi.tugraz.at*

Abstract

The brain uses recurrent spiking neural networks for higher cognitive functions such as symbolic computations, in particular, mathematical computations. We review the current state of research on spike-based symbolic computations of this type. In addition, we present new results which show that surprisingly small spiking neural networks can perform symbolic computations on bit sequences and numbers and even learn such computations using a biologically plausible learning rule. The resulting networks operate in a rather low firing rate regime, where they could not simply emulate artificial neural networks by encoding continuous values through firing rates. Thus, we propose here a new paradigm for symbolic computation in neural networks that provides concrete hypotheses about the organization of symbolic computations in the brain. The employed spike-based network models are the basis for drastically more energy-efficient computer hardware – neuromorphic hardware. Hence, our results can be seen as creating a bridge from symbolic artificial intelligence to energy-efficient implementation in spike-based neuromorphic hardware.

1 Introduction

Spiking neural networks (SNNs) constitute the third generation of neural network models (Maass, 1997). The most salient difference to standard artificial neural networks (ANNs) that are usually employed in deep learning is the means of communication between neurons in the network. While ANNs communicate real numbers in each computational step, spiking neurons in SNNs communicate events in an asynchronous manner. The output of a spiking neuron is zero most of the time, only occasionally turning to "1", which is referred to as spiking or sending a spike. This communication scheme is much closer to the way neurons communicate in the brain and it has some advantages that we will discuss below. In this chapter, we discuss symbolic computations with SNNs. In particular, since we concentrate on tasks that involve temporally sequential input which necessitates recurrent connections in order to combine temporally dispersed information, we consider here networks of recurrently connected spiking neurons (RSNNs). This also reflects the fact that neuronal networks in the brain are highly recurrent. We use the term RSNNs whenever we want to empahsize the recurrent architecture of the considered network. We review the current state of this research direction and in addition present new results which show that surprisingly small SNNs can perform symbolic computations on bit sequences and numbers.

Why should we bother about RSNNs when thinking about neuro-symbolic computation? There are at least two answers to this question. First, the brain is very likely a computing system that merges sub-symbolic connectionist computation with symbolic components. It has been argued many times that a pure subsymbolic system cannot achieve the generalization abilities of symbolic systems where variables can be used in various contexts (Marcus, 2001, 2018; Marcus et al., 2014). Hence, the superb generalization capabilities of the brain likely result from symbolic computations.

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

On the other hand, judged purely from the hardware, it is undisputed that the brain is a neural system. However, if we want to learn from the brain, we should take its basic computational units seriously. The event-based nature of spiking communication in the brain is in many aspects different from communication in ANNs with real-valued neurons. One can in principle quite easily emulate every ANN with an SNN since each real-valued output can be encoded by a spike rate, and several such and more advanced conversion methods exist (Rueckauer and Liu, 2018; Kim et al., 2019). On the other hand, due to the asynchronous and spike-timing based communication in SNNs, the emulation of an SNN with an ANN is much harder (Maass, 1997). Hence, it might not be straightforward to map the solution that evolution has found for symbolic computation to ANNs. Second, the brain is a network of a gigantic number of neurons, and the experience with deep learning has shown that bigger is usually better. Hence, we can expect that neuro-symbolic systems that approach at least some aspects of human intelligence will be huge. To make them practically useful, one should consider energy-efficient hardware solutions. One reason why the brain is extremely energy-efficient is its spike-based communication, where spikes are transmitted only when necessary. Energy-efficient neuromorphic hardware is based on this basic principle utilizing RSNNs (Maass, 2016; Davies et al., 2021; Furber, 2016).

Another lesson we learned from recent artificial intelligence (AI) research is that learning-based approaches are the most promising path to AI systems. Hence, we should not try to construct symbolic systems but rather equip neural networks with inductive biases and then train them to perform symbolic reasoning tasks. Since SNNs can exploit precise spike timing, some functions can be implemented with less resources by SNNs as compared to ANNs (Maass, 1997). In practice however, training of SNNs was problematic as the standard training algorithms for ANNs — Backprop for feed-forward networks and Backprop-through-time (BPTT) for recurrent networks — were not applicable due to the non-differentiability of the spiking output of the neurons. Recently however, several solutions to this problem have been proposed (Bellec et al., 2018, 2020) and it turned out that SNNs can also compete with ANNs in practice. Therefore, there is no longer a reason to resort to ANNs when studying neuro-symbolic computation. Here, we will consider a training algorithm for RSNNs called e-prop (Bellec et al., 2020). This algorithm does not only produce competitive results for RSNNs, it is also biologically plausible and can — as opposed to BPTT — easily be implemented in neuromorphic hardware.

In the following section, we will briefly discuss the basic principles of spike-based computations in the brain and introduce spiking neural network models. We will then touch on the main theories of symbolic computation in the brain. In the main part of this chapter, we will give an overview of the state-of-the-art of neuro-symbolic computation with RSNNs. We will show in particular, through a few examples, that biologically realistic RSNNs can perform symbolic tasks on bit strings and numbers. These results show that tasks in which symbols are manipulated can be learned by models of spiking neural networks and even trained using the biologically plausible learning rule e-prop (Bellec et al., 2020). These results are not only of interest from a biological perspective to understand the mechanisms used by our brain, but also as a computing paradigm for energy-efficient neuromorphic hardware. In that light, our results suggest bridging symbolic artificial intelligence to energy-efficient implementations in spike-based neuromorphic hardware.

2 Neural networks of the brain and spiking neural networks

Neurons are the basic computational units of the brain. They are cells specialized for information processing and information transmission via brief (approximately 1 ms) stereotyped electrical pulses, so-called action potentials or spikes. A spike — a short, sudden increase in voltage — has an amplitude of about 100 mV and lasts $1-2$ ms (Gerstner and Kistler, 2002). Since the spikes of a neuron are all alike, a spike itself (the form of the pulse) does not convey much information. Instead, information can be carried by the number of spikes in a certain time window, the precise timing of a spike in relation to the spikes of other neurons, and the temporal spike pattern. As the shape of the action potential is considered irrelevant, one can abstract over it and define the output of a neuron as a sequence of time points $t^{(1)}, t^{(2)}, t^{(3)}, \ldots$ where $t^{(i)}$ is the i-th spike of the neuron. Such a sequence of spike times from one neuron is also called a spike train.

The brain consist of large networks of such neurons, connected via synapses. Incoming spikes

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

to a neuron initiate changes in the membrane potential of the cell body (soma), and occasionally, if a certain voltage threshold is exceeded, the neuron generates a spike that is transmitted via synapses to other neurons.

This process is formalized mathematically in spiking neuron models. These models come at various granularities and abstraction levels, with the common characteristic that their outputs are spike trains.

2.1 The leaky integrate and fire (LIF) neuron model

One of the simplest spiking neuron models is the leaky integrate and fire (LIF) neuron model, see Figure 1 (left).

It captures the essential temporal dynamics of biological neurons, while abstracting over many — potentially important — aspects of its biological counterparts. The dynamics of the membrane potential V_j of a LIF neuron j is described by a first order linear differential equation:

$$\tau_m \dot{V}_j(t) = -V_j(t) + R_m I_j(t), \tag{1}$$

where τ_m is the membrane time constant (on the order of 10 to 30 ms), R_m is the resistance of the cell membrane, and I_j the input current. This equation describes a leaky integrator. The current I_j is integrated over time, however the integration is leaky, i.e., if there is no input current, the membrane potential decays to zero with time constant τ_m. The neuron outputs a spike at the time when its membrane potential V rises above the threshold v_{th}. After a spike is generated, the neuron dynamics is reset: the threshold value v_{th} is subtracted from the neuron's membrane potential, and the neuron enters a refractory period. During the refractory period, the neuron cannot spike.

We consider networks of such neurons where network neurons $1, \ldots, N$ are recurrently connected. We term these networks recurrent spiking neural networks (RSNNs). RSNNs reflect two important biological facts: first, that neurons in the brain are heavily recurrently connected (in contrast to the often used feed-forward architectures in artificial neural networks), and second, that these neurons communicate with spikes. The input current $I_j(t)$ to neuron j in the network is defined as the weighted sum of spikes from external inputs x and other neurons in the network z:

$$I_j(t) = \sum_i W^{\text{in}}_{ji} x_i(t - d^{\text{in}}_{ji}) + \sum_i W^{\text{rec}}_{ji} z_i(t - d^{\text{rec}}_{ji}), \tag{2}$$

where W^{in}_{ji} and W^{rec}_{ji} denote the input and the recurrent synaptic weights respectively, and d^{in}_{ji} and d^{rec}_{ji} the corresponding synaptic delays. Here, the weights model the efficacies of biological synapses and are similar to the connection weights in standard artificial neural networks. The delay d_{ji} models the time that passes between an action potential of the presynaptic neuron i and the voltage response at the soma of the postsynaptic neuron j.

Neurons are usually simulated in discrete time. Writing (1) in discrete time with time-step δt and incorporating the reset after a spike, we obtain

$$V_j(t + \delta t) = \alpha V_j(t) + (1 - \alpha) R_m I_j(t) - v_{\text{th}} z_j(t) \delta t, \tag{3}$$

where $\alpha = \exp(-\frac{\delta t}{\tau_m})$ and $z_j(t) \in \{0, \frac{1}{\delta t}\}$ represents the spike output of neuron j. The term $-v_{\text{th}} z_j(t) \delta t$ implements the voltage reset after each output spike. The output z_j of the neuron is obtained by comparison with a voltage threshold v_{th}

$$z_j(t) = H\left(V_j(t) - v_{\text{th}}\right) \frac{1}{\delta t}, \tag{4}$$

with $H(x) = 0$ if $x < 0$ and 1 otherwise being the Heaviside step function.

2.2 Enhancing temporal computing capabilities of RSNNs with Spike Frequency Adaptation (SFA)

Upon inspection of the membrane potential dynamics of a LIF neuron, (1), we observe that spiking neurons have some internal memory about previous inputs on the time scale of the membrane

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

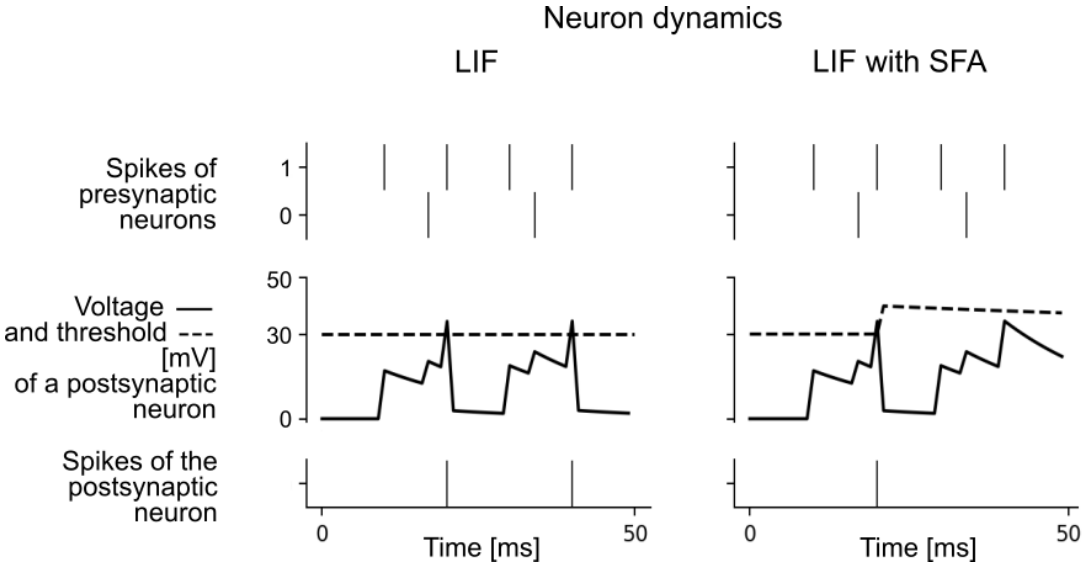

Figure 1: Neuron dynamics for LIF neuron without (left) and with SFA mechanism (right). Top to bottom: The same spike trains of two presynaptic neurons that were connected to a postsynaptic neuron, voltage traces (solid line) and threshold (dashed line) for the postsynaptic neuron without (left), and with spike frequency adaptation (SFA) mechanism (right), output spikes of the postsynaptic neuron. Due to the SFA, the postsynaptic neuron in the right panel outputs only one spike. Neuron parameters used: $\tau_m = 20$ ms, threshold $v_{\text{th}} = 30$ mV, $\beta = 1.7$ mV, $\tau_a = 100$ ms.

time constant τ_m, which is on the order of 10's of milliseconds. This is a difference from standard artificial neurons in deep learning where the output is an instantaneous function of the momentary input. A notable exception are long short-term memory (LSTM) units (Hochreiter and Schmidhuber, 1997), which turned out to be very powerful units in artificial recurrent neural networks. There, so-called memory cells can store their internal state arbitrarily long, which turned out to boost the memory capabilities of recurrent ANNs. The membrane time constant of spiking neurons is however too short to account for short-term memory at behaviorally relevant time scales of seconds in biological organisms. Instead, it has been shown that biological mechanisms at longer time scales can be utilized to boost temporal computing capabilities of RSNNs (Bellec et al., 2018; Salaj et al., 2021). One such mechanism is spike frequency adaptation (SFA), which can easily be incorporated in the standard LIF neuron model.

In contrast to LIF neurons, the threshold of a LIF neuron with SFA is dynamic, see Figure 1 (right). After each spike that the neuron produces, its firing threshold is increased by a fixed amount, and then it decays exponentially back to zero. The effective threshold $A_j(t)$ of a LIF neuron j with SFA is given by the equations

$$\begin{aligned} A_j(t) &= v_{\text{th}} + \beta a_j(t), \\ a_j(t+1) &= \rho_j a_j(t) + (1-\rho_j) z_j(t), \end{aligned} \tag{5}$$

where v_{th} is the constant baseline of the firing threshold and $\beta > 0$ scales the amplitude of the activity-dependent component $a(t)$. The parameter $\rho_j = \exp\left(-\frac{\delta t}{\tau_{a,j}}\right)$ controls the speed by which $a_j(t)$ decays back to 0, with $\tau_{a,j}$ as the adaptation time constant of neuron j. Experimental data show that many neurons in the brain exhibit SFA with time constants $\tau_{a,j}$ from hundreds of milliseconds up to tens of seconds (Pozzorini et al., 2013, 2015).

2.3 Training RSNNs with e-prop

Recurrently connected neural networks are trained on temporal sequences of input-output pairs via Backprop-through-time (BPTT) – an algorithm that unfolds the recurrent network in time and

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

operates in two phases. In the first phase, a forward pass is performed, that is, the network outputs are computed, and a cost function that measures the distance between the network's outputs and desired values is evaluated. In the second phase, by applying the chain rule of calculus, the derivatives of the cost function with respect to the parameters, i.e., synaptic weights, are evaluated for each time step, and the synaptic weights are updated. This approach requires the full history of the network activity to be stored, hence it is quite unlikely that the brain performs such an algorithm for learning. An alternative to the BPTT algorithm that is more biologically realistic is the e-prop learning rule (Bellec et al., 2020). Unlike BPTT, for updating the synaptic weights of the network this learning rule requires only information locally available to each synapse and neuron – an eligibility trace and a learning signal broadcasted across the network. Eligibility traces model preceding activity in neurons and synapses at the molecular level, whereas learning signals model top-down signals (e.g., dopamine, acetylcholine, and neural firing related to the error-related negativity) that target different populations of neurons and govern the synaptic plasticity, i.e., the process of adjusting the strengths of synaptic weights.

More precisely, consider a cost function E that should be minimized in the learning process. The gradient $\frac{dE}{dw_{ji}}$ of the cost function with respect to the synaptic weight w_{ji}, needed for the update of this weight can be calculated as follows:

$$\frac{dE}{dw_{ji}} = \sum_t L_j(t) e_{ji}(t), \tag{6}$$

with $L_j(t)$ representing the learning signal for neuron j, and $e_{ji}(t)$ the eligibility trace for the synapse from neuron i to neuron j. In e-prop, one uses an approximate learning signal

$$L_j(t) = \sum_k B_{jk}(y_k(t) - y_k^*(t)) \tag{7}$$

which captures errors occurring at the output neurons k at the current time step t, that is, deviations of the outputs $y_k(t)$ from the targets $y_k^*(t)$ are calculated, and fed back from output neuron k to every network neuron j via randomly chosen weights B_{jk}. The eligibility trace $e_{ji}(t)$ represents a transient memory for the synapse connecting the presynaptic neuron i and postsynaptic neuron j, that is, it integrates the history of the synapse up to time t based on locally available information. For a LIF neuron, it is based on a low-pass filtered version of the presynaptic spike train and the postsynaptic membrane potential.

The learning signal $L_j(t)$ only takes into account the current time step t, however, the eligibility trace $e_{ji}(t)$ accounts for the past of the neuron. Since they are readily available in every time step t of the forward phase, as soon as the forward phase is completed, the updates of synaptic weights can be performed, without performing the backward phase as required in BPTT. There is no need for unrolling the recurrent network in time for calculating the errors from previous time steps as in BPTT, and as such, e-prop is an online learning method for training models of recurrently connected neurons. For an illustrative, short summary of the e-prop learning rule, see (Manneschi and Vasilaki, 2020).

3 Symbolic computation and arithmetics in the brain

3.1 Representation of numbers in the brain

There exists experimental evidence for at least two types of neural codes for numbers in the cortex. It has been shown that individual neurons in the lateral intraparietal area (LIP) exhibit a Gaussian tuning for cardinality when macaque monkeys perform a number matching task (Nieder and Miller, 2003). A population of such neurons, each tuned to a different represented value, can collectively encode a number. To make this more precise, we describe here a simple mathematical model for this coding scheme which will also be used in later sections in our simulations. For a represented number z, each neuron i in such a population spikes with a firing rate of

$$f_i = f_{\max} \exp\Big(- \frac{(m_i - z)^2}{2\sigma^2} \Big), \tag{8}$$

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

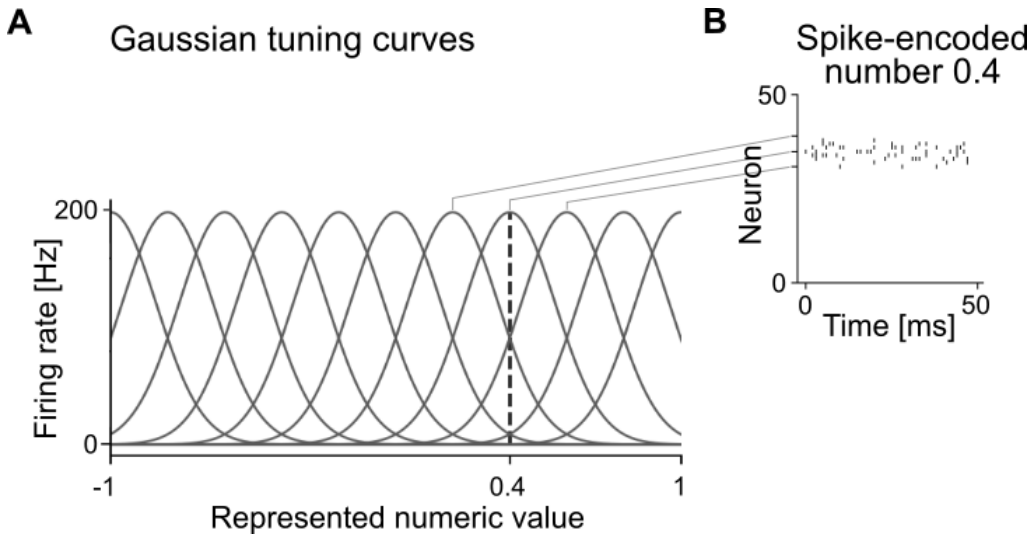

Figure 2: Population coding of numbers. This encoding was used in the middle panel of Figure 4C to encode numbers. (A) The numeric value from $[-1, 1]$ that is represented by a population of 50 neurons is indicated on the x-axis. The y-axis indicates the firing rate of 11 neurons out of this population in response to the corresponding number. Each neuron responds with its maximal firing rate to its preferred value (at the peak of its Gaussian tuning). When the represented value is off this preferred value, the firing rate decreases according to the indicated Gaussian profile. The dashed vertical line indicates one example value of 0.4. The neuron with its tuning curve centered on this value fires at 200 Hz. The two neurons with the shown neighboring tuning curves fire at approximately 80 Hz. (B) Spiking activity of 50 spiking neurons of this population when the number 0.4 was encoded, as indicated in panel A. Each row corresponds to one neuron of the population, where neurons are sorted by their preferred value from -1 to 1. Each tick indicates one spike. Gray lines connect the centers of tuning curves in A with the corresponding neurons in panel B.

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

where $f_{\max}$ is the maximum firing rate of neurons in the population, m_i is the value to which the neuron is tuned to, and σ is the standard deviation of the Gaussian tuning curve which is chosen here to be equal for all neurons in the population for simplicity. The preferred values m_i for these neurons are typically chosen equally spaced in the range of numbers that can be represented. Hence, each neuron responds with its maximal firing rate $f_{\max}$ if its preferred value m_i is represented. When the represented value is off this preferred value, the firing rate decreases according to a Gaussian profile. Such population coding of numeric values between -1 and 1 is illustrated in Fig. 2 for a population of 50 neurons. This coding scheme abstracts over input stimulus properties, as it is preserved over input modalities such as visual or auditory input (Nieder, 2012).

In other tasks, some neurons in LIP fire more strongly for larger numbers and others show the inverse behavior, hence exhibiting a spike frequency code (Roitman et al., 2007). It has been hypothesized that these two types of codes may separate the actual entity represented (i.e., the number represented with a Gaussian tuning) from the structure of the number space (a linear firing rate code which captures similarity between numbers) (Summerfield et al., 2020; Behrens et al., 2018). The authors also state that structure-disentangled codes are not unique to numbers and magnitude, rather this might be a general coding principle for structured information (e.g., space). There exists evidence for similar coding principles in humans. EEG activity varies smoothly with numerical distance when Arabic numbers are presented to humans (Spitzer et al., 2017; Teichmann et al., 2018).

3.2 Circuits for arithmetic in the brain

Little is known about the specific circuit mechanisms that implement arithmetic operations in the brain. Several cortical regions are activated during arithmetic tasks. Here, the strength of activation of specific areas often depends on the type of task (e.g. approximate versus exact), task difficulty (e.g. $2 + 3$ versus 27×32), and expert level of the subject (Zamarian et al., 2009). In general, frontal areas are more strongly involved in more complex calculations (Semenza et al., 1997). Interestingly, several lines of research indicate that growing expertise leads to the increased involvement of specific parietal areas whereas frontal areas become less strongly involved (Zamarian et al., 2009). This supports the theory that arithmetic operations can be performed by more general-purpose processing, but practice may train special purpose circuits which can perform the trained tasks with less effort. Our novel results below show that training of complex arithmetic operations can be performed on surprisingly small SNNs.

3.3 Symbolic computation

A symbol has meaning and significance; it refers to ideas, objects, relationships and concepts (Gazzaniga et al., 2009). There are a few important aspects to consider when studying the emergence of meaning from symbols, for example, the establishment of links between symbols and objects/concepts that they represent, learning symbolic meaning from context, or generalization over a range of instances of symbols. Also, different brain regions are involved in general meaning processing, but semantic integration mechanisms draw on higher cortical areas of the neocortex, such as prefrontal, posterior parietal, anterior, inferior, and posterior temporal cortex (Pulvermüller, 2013).

Unlike computations performed directly on values such as the activities of sensory receptors, symbolic computation addresses the question of the manipulation of symbols using rules. A symbolic expression can be expressed using variables that are instantiated by concrete values. In algebra, abstract variables x or y can be substituted by numbers; in language, terms *subject* or *verb* can be replaced by words. The abstract terms act as placeholders, and they are a higher-order representation of knowledge. Through the concept of variable binding — how the placeholders are instantiated by concrete values — one can study how computations performed by neurons give rise to cognitive processes, and eventually elucidate the underlying process in the brain which contribute to language and abstract thought (Marcus et al., 2014).

Symbols can represent categories (e.g., *cat*, *dog*), variables (e.g., x, y), computational operations (e.g., '+', '−', *concatenate*, *compare*), and individuals (e.g., *Felix*) (Marcus, 2001). The mind

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

is a system that represents variables, operations over variables, and structured representations, capable of distinguishing between categories and individuals. A possible physical implementation of such a system would be a set of buckets, with the bucket's content representing an instantiation of a category represented by the bucket. Digital computers already use binary registers that correspond to buckets being either empty or full (bits 0 or 1), and the operations over the registers are performed in parallel. Next, each variable should correspond to an anatomically defined region or register. For example, the symbol *"black"* in *"black cat"* or *"black sedan"* should apply equally, irrespective of the categories animals and cars that may be represented in widely separated regions of the cortex (Marcus et al., 2014).

Several neural network models have been proposed for the implementation of symbolic computation via variables and variable-binding mechanisms that bind specific content to a given variable. Pointer-based models (e.g., (Zylberberg et al., 2013)) assume that pointers are implemented by single neurons or co-active populations of neurons which are synaptically linked to content. Another class of models is based on the idea of indirect addressing. These models assume that a neural space for a variable encodes an address to another brain region where the corresponding content is represented (Kriete et al., 2013; O'Reilly, 2006). In anatomical binding models, different brain areas are regarded as distinct placeholders (Hayworth, 2012; Hayworth and Marblestone, 2018). In Dynamically Partionable Autoassociative Neural Networks (DPANN) (Hayworth, 2012), a given symbol corresponds to a global attractor state of the dynamics of a large-scale network that links many regions of the brain. Information flow between particular sets of registers is achieved through turning on and off subsets of synapses, i.e., gating. In neural blackboard architectures, one assumes that besides assemblies for content, there exist also assemblies for structural entities such as phrases, semantic roles, et cetera. Specialized circuits (so-called gating circuits and memory circuits) are then used to bind content to structure (van der Velde and de Kamps, 2006). This way, complex syntactic structures can be built. Vector-symbolic architectures (Plate, 1995) use high-dimensional vectors to represent the symbolic concepts encoded by activity patterns of a group of neurons and a defined set of mathematical operations. Both the variables and potential role filters, such as subject, object or verb, are represented as vectors. Compositional operations typical for symbol manipulation are then implemented through a mathematical transformation of these vectors. Activity patterns for binding, for example, a subject and an instance, are achieved through multiplication of two vectors (Eliasmith et al., 2012). Most of these models rely on specifically constructed circuitry that performs the binding operations. In contrast, a rather generic SNN model was proposed in (Müller et al., 2020). In this assembly pointer model, binding emerges through local Hebbian plasticity processes from the activation of assemblies in areas that contain variables to areas that contain content.

4 Prior work on symbolic computations on bit strings and numbers with SNNs

Some of the above mentioned neural network models for variable binding have been implemented in SNNs (e.g., (Eliasmith et al., 2012; Müller et al., 2020)). In this review, we concentrate more specifically on symbolic computations on numbers, and in contrast to the above mentioned models, we propose a more generic approach where SNNs are trained to perform specific tasks.

In a recent work (Salaj et al., 2021), a spiking neural network was trained on two tasks based on a rule that was given at the beginning of each trial. For the same input sequence, the task of the network was either to reproduce the sequence in the same order, if the rule was *duplicate*, or the sequence in reversed order if the rule was *reverse*. The SNN had to learn to observe data stimuli (symbol sequences) given as input, and two different rules on how to manipulate them, also given as an input, independent of each other, then to use both data stimuli and the given rule to produce an output. After training, the SNN was able to produce sequences with all correct symbols in 95.88% of test trials. Through an analysis of spiking activity on the test trials, groups of neurons that had specialized in encoding structural information of the task, such as symbol identity, position, and task identity, were identified. The analysis revealed that many neurons specialized in encoding non-linear combinations of conditions, indicating a non-linear mixed selectivity of neurons. In other words, while some neurons showed some preference in their tuning to specific aspects necessary to

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

solve the tasks, there was no clear-cut encoding of symbols by individual neurons in this trained circuit — different from what one would usually do to construct a neural network to perform these operations. Such mixed selectivity of neurons has also been observed in higher cortical areas (Rigotti et al., 2013).

In another experiment of (Salaj et al., 2021), it was shown that SNNs can learn to follow dynamic rules, i.e., make decisions, for example, choose the "Left" or "Right" button, depending on the currently active rule and specific subsequences of symbols relevant for that particular rule. In a long sequence of symbols, the currently active rule, relevant and distractor symbols appeared sequentially. Even for humans, the task is quite demanding, since it requires not only the maintenance and an occasional update of the active rule, but also maintaining relevant symbols and ignoring irrelevant ones until the target subsequence is complete. Such a task requires a hierarchical working memory – higher-level memory for maintenance of the active rule, and lower-level memory for maintenance of the symbols relevant in the context of the active rule, especially because the distractor symbols occur rather frequently. A generic SNN was able to solve this task almost perfectly without requiring a special-purpose architecture. In 97.79% of trials consisting of sequences of 90 symbols, all choices of the button "Left" or "Right" were correct. This result shows that cognitively demanding sequence processing tasks can be solved by generic SNNs.

5 New results on spike-based symbolic computation on bit sequences and numbers

In this section, we present new results on how SNNs can perform symbolic computations on bit sequences and numbers. Human infants can perform mental arithmetic using nonsymbolic, approximate number representations. Even before children learn algorithms for manipulating numerical symbols and formal mathematics, they are able to perform approximate addition and subtraction of large numbers. In such experiments, children performed well above chance – they were able to do so by comparing quantities instead of providing an exact solution (Gilmore et al., 2007). Certainly, we use non-symbolic representations before we master symbolic arithmetic, which takes years (Gilmore et al., 2010). Some authors refer to this capability as "the number sense" (Dehaene, 2011).

There are many open questions: How do humans perform arithmetic? How are the numbers encoded? What are the mental representations behind the comparison of two numbers or quantities? For the case of symbolic arithmetic, we face even more general questions – those of symbol manipulation and sequence processing.

Mastering symbolic representations and symbol processing, in general, helps us perform cognitively demanding tasks. Certainly, the human brain is capable of processing and manipulating symbols, or even sequences of symbols, thereby following the abstract relationships between mental concepts (symbols). These abstract relationships are known as rules, and the brain can process sequences of symbols even when rules dynamically change or when the stimulus is novel (Marcus, 2001).

Given a sequence of items, humans are able to infer the rules that generated the sequence and apply the same rule even on sequences of items that they had never encountered before (Liu et al., 2019). If we revisit the question of numbers, for example, Arabic numbers and their representation, we notice that single-digit numbers are represented by symbols 0 to 9. Multi-digit numbers are represented by sequences of symbols, where the position in the sequence is of particular relevance. Moreover, we learned the meaning of the most significant and least significant digit, and that the length of the sequence representing the number carries another crucial piece of information. To compare two numbers given in a symbolic form, we can first rely on the rule of comparing the lengths of sequences. In case the lengths are the same, we immediately proceed by applying the second rule - comparing the corresponding significant digits, iteratively if necessary, starting from the most significant digits towards the least significant ones, but we stop when we can unambiguously conclude which number is greater, or after we compared the least significant digit. In fact, we can perform such sophisticated sequence processing during the comparison of two symbolic numbers with little effort. We have mastered this processing during years of practice through schooling, and we are applying the learned rules rather automatically. This subjective

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

observation is backed up by experiments which suggest that specialized cortical circuits learn complex arithmetic operations during training (Zamarian et al., 2009), see Section 3.2.

We performed two experiments – both of them show that rather small, generic SNNs can be trained to perform demanding sequence processing tasks. In the first experiment, we considered the number comparison task described above, a task that has often been used to study number processing in cognitive science (Sheahan et al., 2021). For simplicity, we used binary numbers instead of previously described Arabic numbers, since the same sequence processing rules for comparison also apply when comparing numbers in any base. Note that, the rules describing how to compare the numbers were not explicitly given to the network, rather, the network learned to perform the task through training.

In the second experiment, an SNN was trained to perform arithmetic operations (addition, subtraction, and multiplication) on real numbers encoded into spiking activity using population coding with Gaussian tuning, a number encoding that has been observed in monkeys, see Section 3.1. Arithmetic expressions — the instructions indicating the rule to be applied on the data stream, i.e., real numbers — were given to the network in a separate stream, and sequentially. The challenge of this task was to incorporate pieces of information processed earlier, that is, to reuse the result of the computation from the previous step while following new instructions. Without explicitly being trained to do so, the network had also to solve the binding problem – between variables that appeared in the expression and the neurons encoding the number. The network learned successfully to process a sequence of arithmetic expressions, and similar to humans, to perform at a level of expertise.

5.1 Spike-based symbolic computation on bit strings

The comparison of two numbers is a task often considered in cognitive science (see e.g., (Sheahan et al., 2021) for a recent example). We wondered if an SNN could learn to perform the comparison task on expressions with numbers encoded as bit strings. We refer now to the numbers represented in a binary base as bit strings rather than sequences, since we use the term sequence to denote the (temporal) processing dimension and that symbols appear sequentially, one after the other.

The schematic network architecture is shown in Fig. 3A. The input to the network were binary strings of width 10 bits representing numbers in the binary base. Another input stream represented the relational operator to be applied, where we used *equals* '=', *less than* '<', and *greater than* '>'. Variables x_1, x_2, and the relational operator were presented to the network sequentially as Poisson spike trains. A Poisson spike train is a random sequence of spikes with a given rate (i.e., expected number of spikes per second), similar to spike trains observed in the brain. Mathematically, such a spike train is generated by a Poisson point process. Each presentation of a variable or relational operator lasted for 200 ms, with high firing activity coding for 1 and low firing activity coding for 0, see Fig. 3C, top. Only after the full sequence indicating the first number x_1, the second number x_2, and the relational operator rel_op has been shown, does the network have all the pieces of information required to determine whether the provided relation x_1 rel_op x_2 was correct. As the answer, the network was required to output "Yes" or "No".

We trained a recurrently connected spiking neural network consisting of 200 LIF neurons (100 LIF neurons with SFA, 100 without) on this task using the biologically plausible learning algorithm e-prop (Bellec et al., 2020), for which the accuracy over training iteration is shown in Fig. 3B. The curve was smoothed using bins of 20 values. Fig. 3C illustrates a trial where the network had to answer whether the binary number $x_1 = 1011111111$ was smaller than binary number $x_2 = 1011010100$, for which the final answer is "No". Tested on previously unseen instances x_1 rel_op x_2, that is, combinations of bit strings and relational operators that were not used during training, the proportion of correct answers (accuracy) achieved was 0.8723 ± 0.0235 (mean $\pm$ standard deviation), averaged over 5 different network initializations.

This result demonstrates that rather small SNNs can learn to perform arithmetic tasks using a biologically plausible learning algorithm. In this task, numbers were represented in a symbolic manner (i.e., using symbol strings as in Arabic numbers) rather than as magnitudes, and the network had no prior knowledge on the structure of this number encoding or on the structure of the task. Nevertheless, the network was able to generalize to novel task instances, indicating that explicitly designed symbolic computations are not necessary for generalization for this task.

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

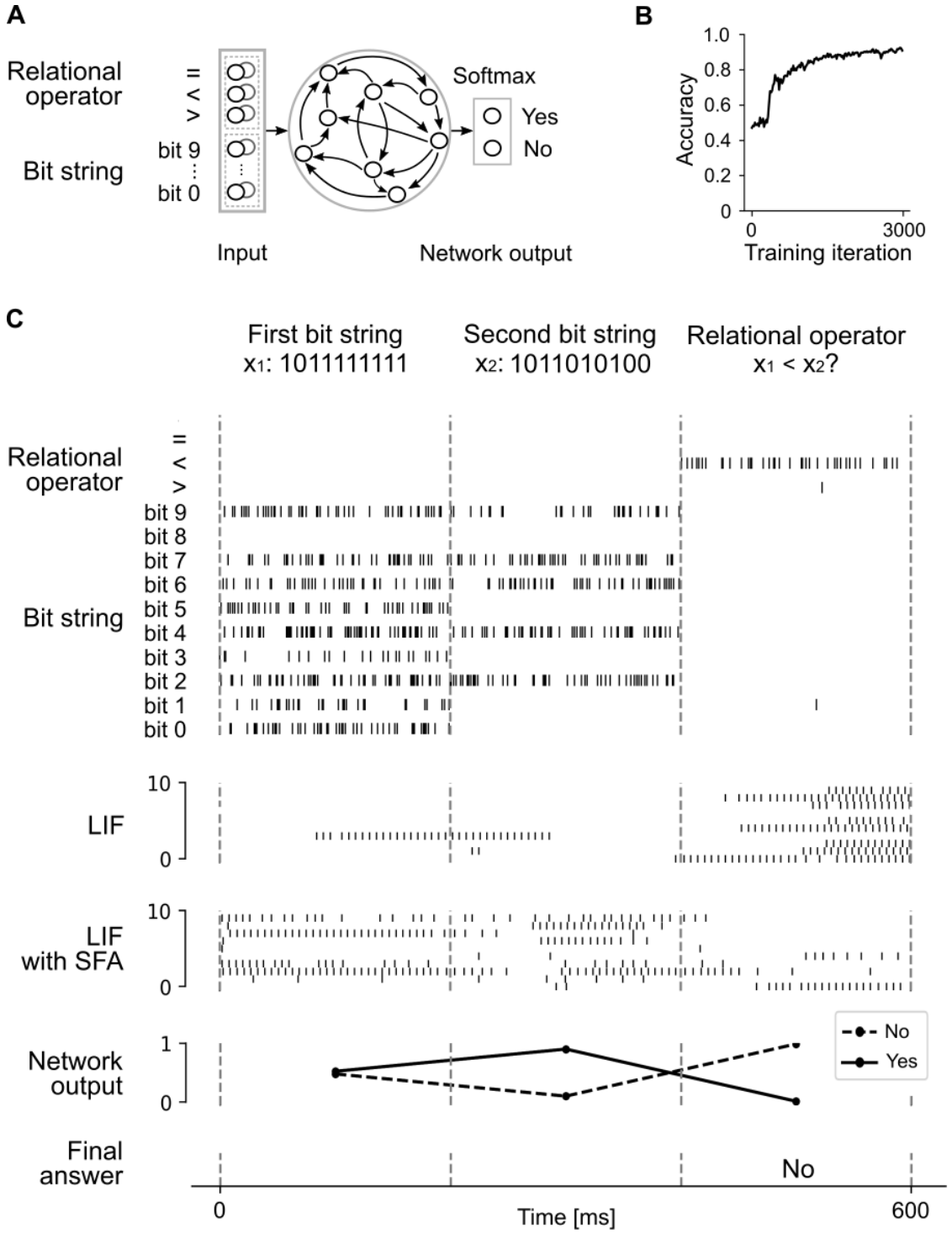

Figure 3: Comparison of two bit strings. (A) The schematic network architecture: input spiking neurons encoding relational operators and bit strings of width 10 (bit 9 the most significant, and bit 0 the least significant bit), recurrently connected LIF neurons, and two output neurons. The network output is the softmax of the ouput neurons producing "Yes/No" answers. (B) Training curve: Accuracy over training iteration. Test accuracy for novel trials was 0.8723. (C) Spiking activity of the network for a trial where the network compared if the first bit string $x_1 = 1011111111$ was smaller than the second bit string $x_2 = 1011010100$. Top to bottom: Spiking activity of neurons encoding the relational operator and bit strings (one neuron shown out of 2 for each symbol), 10 sample out of 100 LIF neurons, 10 sample out of 100 LIF neurons with SFA, network output – the softmax applied to readout neurons dedicated to producing the answer, the final (maximum value of those for "Yes" and "No" neurons) and correct answer "No" to the question "$x_1 < x_2$?".

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

5.2 Evaluation of nested arithmetic expressions

One of the most basic arithmetic tasks is the evaluation of an operation applied to two numbers such as $12 + 4$. One can replace numbers in such expressions by variables to obtain an expression such as $x_1 + x_2$. The solution to the expression can be obtained when the values of the variables are defined. For example, defining $x_1 = 12$ and $x_2 = 4$, the result for the above expression is 16. This calculation demands a basic type of computation where values need to be assigned to variables which then have to be combined by some arithmetic operation. We wondered whether such basic symbolic computations on numbers and variables could be learned by SNNs. However, we took a further step by considering long nested expressions built based on this basic template with three basic arithmetic operators addition '+', subraction '−', and multiplication '∗'. An example of such a complex expression is

$$(((((((((\underbrace{\underbrace{(x_2 * x_4)}_{=r_1} * x_5)}_{=r_2} * x_3) - x_3) + x_5) * x_4) - x_5) * x_4) * x_1) - x_4).$$

$$\cdots$$

The expressions we considered have a nested structure, where the result r_1 of the evaluation of the expression in the innermost nesting level, here, $(x_2 * x_4)$ is used to evaluate the result r_2 for the next nesting level, and so on, until the final result is obtained. We considered nested expressions of 10 operators combining 5 variables, leading to 9 intermediate results $r_1, \ldots, r_9$ and the final result r_{10} of the evaluation.

More precisely, we define the task as the sequential evaluation of expressions in 10 steps $i = 1, \ldots, 10$ as follows:

$$\begin{aligned}
i = 1: &\quad r_1 = (Operand_1 \;\; operator_1 \;\; Operand_2),\\
i = 2: &\quad r_2 = (r_1 \;\; operator_2 \;\; Operand_3),\\
i = 3: &\quad r_3 = (r_2 \;\; operator_3 \;\; Operand_4),\\
&\quad \vdots\\
i = 10: &\quad r_{10} = (r_9 \;\; operator_{10} \;\; Operand_{11}),
\end{aligned}$$

where in the first step ($i = 1$), $Operand_1$ and $Operand_2$ are drawn uniformly over $\{x_1, \ldots, x_5\}$ without replacement, and $operator_1$ was drawn uniformly over $\{+, *\}$ (the operator '−' was omitted in this step to avoid the need of encoding the order of $Operand_1$ and $Operand_2$). For the following steps, each of $operator_2, \ldots, operator_{10}$ was drawn uniformly from $\{+, -, *\}$, and each of $Operand_3, \ldots, Operand_{11}$ uniformly over $\{x_1, \ldots, x_5\}$. This describes the generation procedure for the symbolic part of the task, but numeric values are also required in order to evaluate such nested expressions. We allowed that the values of variables can change after each step i. This was done by drawing in each step i for each of the variables $x_1, \ldots, x_5$ a random value from the uniform distribution over $[-1, 1]$. An example value assignment and target results for the expression given above are shown in Table 1.

We show here that an SNN can be trained to evaluate such complex arithmetic expressions. The nested expression was given sequentially, starting from the expression in the innermost nesting level ($x_2 * x_4$ in the above example), together with the numeric assignments to all operands $x_1, \ldots, x_5$. These inputs were encoded by spiking input populations (see below for the encoding) and presented to the network for 50 ms, see Fig. 4A for a schematic of the input populations. The network was demanded to produce at its readout the intermediate result r_1. Then, the expression for the next nesting level ($*x_5$ in the above example) was provided for 50 ms and the result r_2 was demanded as the output, and so on, see Fig. 4C for the temporal sequence of inputs to the network.

Consequently, the challenges for the network were the processing of symbols with associated numeric values, i.e., it had to learn to bind a concrete value to a variable, and at the same time, carry out the demanded mathematical operations in an online fashion. An example of computations performed by a trained SNN on the previously mentioned nested expression is illustrated in Table 1.

The schematic architecture of a spiking network that we trained is illustrated in Fig. 4A. Symbolic inputs to the network were encoded by subpopulations of neurons producing Poisson

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

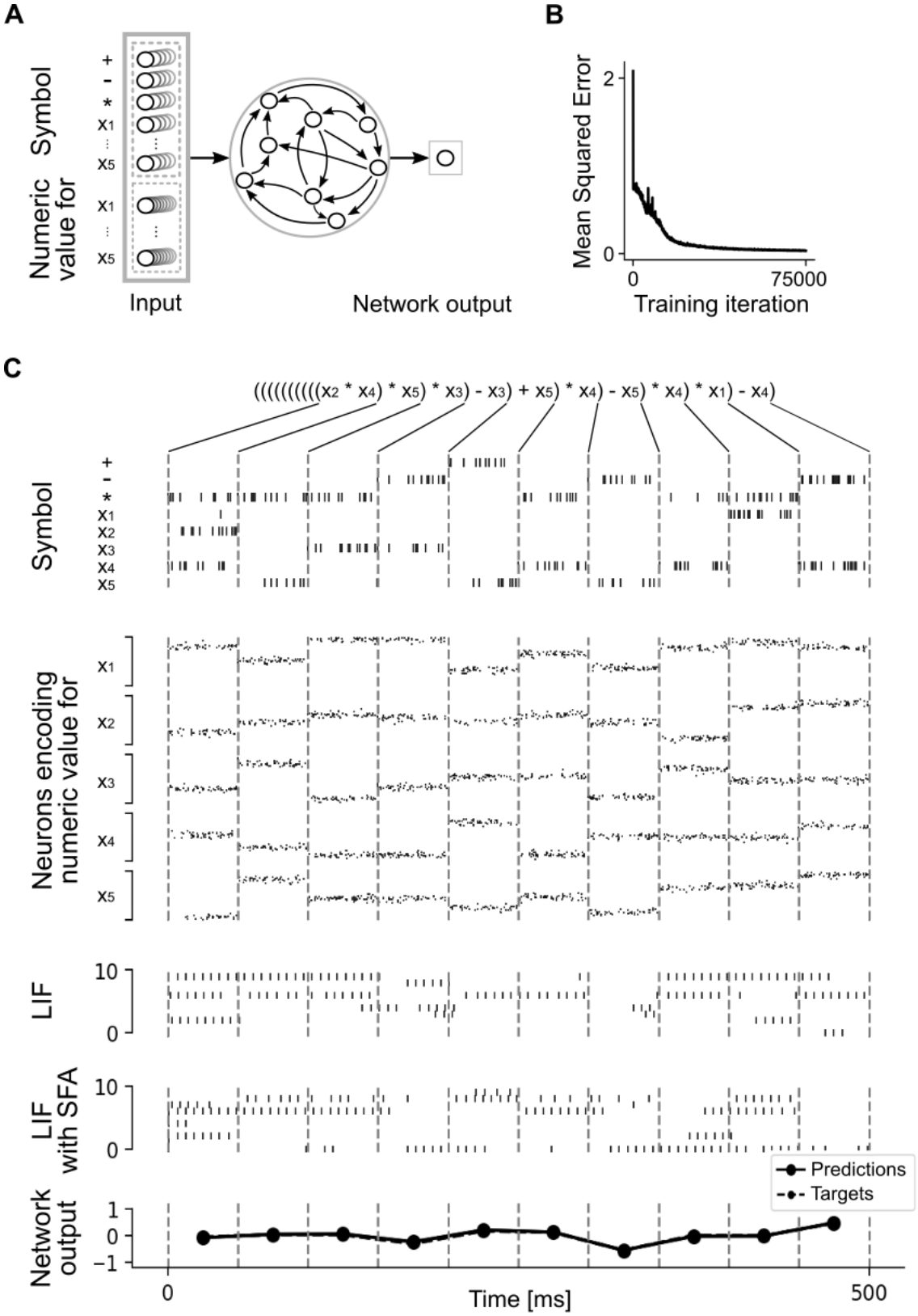

Figure 4: (Caption on the next page.)

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

Step i	Arith. expression	Numeric expression	Output r_i	Target	Absolute error
1.	x_2 * x_4	0.4966 * (-0.1113)	-0.0794	-0.0553	0.0241
2.	r_1 * x_5	(-0.0794) * (-0.7100)	0.0510	0.0564	0.0054
3.	r_2 * x_3	0.0510 * 0.8209	0.0629	0.0419	0.0210
4.	r_3 - x_3	0.0629 - 0.3314	-0.2308	-0.2685	0.0377
5.	r_4 + x_5	(-0.2308) + 0.4717	0.2096	0.2409	0.0313
6.	r_5 * x_4	0.2096 * 0.7014	0.1304	0.1470	0.0166
7.	r_6 - x_5	0.1304 - 0.6650	-0.5417	-0.5346	0.0071
8.	r_7 * x_4	(-0.5417) * (-0.0165)	-0.0222	0.0089	0.0311
9.	r_8 * x_1	(-0.0222) * (-0.8182)	-0.0128	0.0182	0.0310
10.	r_9 - x_4	(-0.0128) - (-0.4727)	0.4739	0.4599	0.0140

Table 1: Evaluation of nested arithmetic expressions with an SNN. One example input sequence with network outputs, targets, and errors. In each input step i — each step lasting for 50 ms — an arithmetic expression and numeric values were given to the network, for which the network produced the output r_i. Output values were very close to target values, that is, the absolute error was close to zero.

Figure 4: Spike-based arithmetic on numbers and variables. (A) The schematic network architecture consisting of input spiking neurons that encoded symbols used in arithmetic expressions (operators and variables) and numeric values for variables, recurrently connected LIF neurons, and a linear readout neuron that produced the result for each arithmetic expression. (B) Training curve (Mean Squared Error over training iteration) for the network trained to evaluate nested mathematical expressions. The curve was smoothed using bins of 50 values. (C) Spiking activity of the network for the nested arithmetic expression shown at the top. Solid lines from the expression to dashed lines of the spike raster indicate the time when this subexpression was presented to the network. Spike rasters from top to bottom: Spiking activity of neurons encoding the symbols appearing in the arithmetic expression (one neuron shown out of 5 for each symbol), encoding of numeric values that instantiate variables, 10 sample out of 150 LIF neurons (without SFA), 10 sample out of 150 LIF neurons with SFA, real-valued network outputs (predictions) that almost perfectly match targets (and consequently, the dashed line representing targets is difficult to see).

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

spike trains, dedicated to each symbol $\{+, -, *, x_1, \ldots, x_5\}$ (5 neurons per symbol) and indicating the presence (absence) of a symbol through a high (low) firing rate. Numeric values for each of variables $x_1, \ldots, x_5$ were encoded by another set of 5 subpopulations of neurons, each consisting of 50 neurons using a population coding scheme. See Fig. 2 for an illustration of population coding of a single number with a population of 50 neurons. Each neuron of this population was assigned a preferred value for which it exhibited its maximal firing rate of 200 Hz. When the represented number was off this preferred value, its rate decreased according to a Gaussian profile. The preferred values of the neurons in this population uniformly tiled the whole space of values between -1 and 1 (Fig. 2A). Hence, for a specific represented value, neurons with preferred tuning close to the value were active, while others remained mostly silent, see Fig. 2B.

As illustrated in Fig. 4A, the spiking activity of the input populations was projected to an RSNN. The output of the network was a single linear non-spiking neuron (a readout neuron), where the numeric value of its output (a real number) was interpreted as the result of the computation. We trained an RSNN consisting of 300 LIF neurons (150 with SFA and 150 without SFA) with all-to-all connectivity using e-prop for 75000 iterations on this task. The objective minimized was the Mean Squared Error (MSE) between the network outputs (predictions) and targets for all intermediate and the final result of the arithmetic expression. The evolution of the network error (MSE) over the training iterations is shown in Fig. 4B. The network achieved an MSE of 0.0341 and a mean absolute error (MAE) of 0.1307 in the mean over all intermediate results, i.e., all computational steps in all of 5000 test trials.

The column "Absolute error" of the example illustrated in Table 1 confirms that the network learned the task very well. The spiking input (encoding of symbols, and of numeric values for variables) and network spiking activity of LIF neurons for the same trial is shown in Fig. 4C, where the bottom panel shows an almost perfect match of outputs (predictions) and targets.

We note that we made sure that the nested symbolic expressions used in the testing trials were different from any nested expression used during training. Also, the used numeric values for the operands were drawn randomly, which, altogether, indicates good generalization capabilities of the trained SNN. This result shows that nested arithmetic expressions can be learned by rather small SNNs to high precision with a biologically plausible learning algorithm. Again, no explicit mechanisms for variable binding were necessary.

6 Conclusions

Most studies of computing capabilities of RSNNs have focused on sensory processing tasks. But the brain uses RSNNs also for higher cognitive functions such as symbolic computations, in particular, mathematical computations. We have reviewed recent results, and added two new ones, which show that RSNNs are very good at these tasks. Moreover, they are able to acquire these capabilities through a biologically plausible learning method – e-prop, and they are able to apply learned computing procedures to new instances of the task that never occurred during training. This is somewhat surprising, since RSNNs do not compute in a clocked mode, like digital computers or ANNs. Rather, they compute in an asynchronous mode where also the timing of spikes carries information. Also, the shown examples for such RSNNs operate in a rather low firing rate regime, where they could not simply emulate ANNs by encoding continuous values through firing rates. Thus we arrive here at a new paradigm for symbolic computation in neural networks that provides concrete hypotheses about the organization of symbolic computations in the brain. Furthermore, since RSNNs can be implemented more energy-efficiently in neuromorphic hardware than ANNs, the results that we have discussed also suggest new methods for improving the energy efficiency of computing hardware for symbolic computations.

7 Technical details to new results

7.1 Network architecture and neuron model

We trained a recurrent spiking neural network of LIF neurons with all-to-all connections. Since in both experiments that we demonstrated temporal integration of pieces of information was crucial

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

for a good performance, a fraction of LIF neurons (one half) was equipped with spike frequency adaptation (SFA). LIF neuron model was described in Section 2.1, and LIF neuron model with SFA in Section 2.2.

The firing threshold of LIF neurons was $v_{\text{th}} = 30$ mV. The same value of 30 mV was used as a baseline firing threshold v_{th} for LIF neurons with SFA, with an adaptation strength of $\beta = 1.7$ mV (see (5)). All LIF neurons had a synaptic delay of 1 ms, a refractory period of 5 ms, and a membrane time constant of $\tau_m = 20$ ms. The resistance of the cell membrane was $R_m = 1$ GΩ. A population of input neurons encoded inputs to the recurrently connected neurons in the form of spike trains. The network output was produced by linear readout neurons, from the average activity of recurrent neurons in the network during the time window representing one step of a trial.

7.2 Training method e-prop

In all tasks, input, recurrent, and readout weights were trained simultaneously. To train the weights of the network by the learning rule random e-prop (Bellec et al., 2020), we used the Adam optimizer with a learning rate of 0.001. The input and recurrent weights were initialized with values from a standard normal distribution, and then scaled with a factor $\frac{1}{\sqrt{n_{\text{in}}}}$, and $\frac{1}{\sqrt{n_{\text{rec}}}}$, respectively, with n_{in} being the total number of input neurons, and n_{rec} the number of recurrent neurons in the network. The readout weights (weights from recurrent to output neurons) were initialized with values from a normal distribution with mean zero, and standard deviation 1.05. The unit for all input and recurrent weights was pAs.

7.3 Tasks

Comparison of bit strings. An SNN consisting of 200 recurrently connected neurons was trained for 3000 iterations, with a batch size of 100, to answer the "Yes/No" questions of the form "Is x_1 *Equal to / Smaller than / Greater than* x_2?". Inputs to the network were two binary strings x_1 and x_2 of width 10 bits, and in fact, represented binary encoded numbers. Binary strings x_1 and x_2 were given successively in the first and the second step of each trial, then followed by a relational operator $=, <$ or $>$, given in the third step. In the third, final step, the network had to produce a "Yes/No" answer to the question that got complete, after all three important pieces of information were shown to the network. Each step lasted 200 ms, hence the total duration of the trial was 600 ms. 10-bit binary strings x_1 and x_2 were encoded into spike trains by a population of 20 spiking neurons – two neurons were dedicated to encoding each bit. Relational operators $\{=, <, >\}$ were encoded using 6 input spiking neurons (2 neurons per operator), in each trial exactly one operator being given in the third step. If a bit represented a value of 1, or if the operator was used in the expression, dedicated neurons fired with a high rate of 200 Hz. To encode a bit value of 0 or that the operator was not used in the expression, neurons fired with a low rate of 2 Hz. Such spike trains encoding the inputs were received by a population of recurrently connected LIF neurons – 100 LIF neurons with SFA and 100 without. Neurons with SFA had an adaptation time constant uniformly distributed from $[1, 600]$. The linear readout consisted of 2 neurons on which we applied softmax – one neuron was dedicated to "Yes" and one for "No" output. Final answer was determined from the maximum value after the softmax – the higher value indicated either "Yes" or "No".

The loss function minimized was cross-entropy function with an additional regularization term. The regularization term represented the squared difference between the average firing rate of the recurrent neurons and a target firing rate of 20 Hz, scaled by a factor of 10. On average, in 87.23% of, in total, 10000 testing trials, the network produced the correct answer. The average was calculated over 5 different initializations of networks with the same architecture and trained in the same way (the weights and adaptation time constants of LIF neurons with SFA differed in each run). The inputs in each test trial were novel, i.e., not used during training.

Evaluation of nested arithmetic expressions. An SNN consisting of 300 recurrent neurons was trained for 75000 iterations, with a batch size of 50 to perform symbolic computation on nested arithmetic expressions involving numbers and variables. Symbolic input to the network was an

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

arithmetic expression formed by the variables $x_1, \ldots, x_5$ and arithmetic operators $\{+, -, *\}$. In the first step of each trial of length 10, two different variables were drawn randomly to form a valid expression with two operands, and the possible operator was limited to either addition (+) or multiplication (∗), to avoid the problem of encoding the order of operands appearing in the expression, which a noncommutative operator subtraction (-) would require. In all other steps of a trial, a variable and an operator were drawn randomly over $\{x_1, \ldots, x_5\}$ and $\{+, -, *\}$ respectively – one variable was necessary, since the result of the previous step was considered as the value that always instantiated the first operand in the subexpression. If a variable $x_1, \ldots, x_5$ or operator $\{+, -, *\}$ used in an arithmetic expression appeared in a step of the trial, they were encoded by the "high" spiking activity of dedicated 5 neurons (200 Hz), otherwise by their "low"' activity (1 Hz). The numeric inputs to the network were analog values (numbers from $[-1, 1] \in \mathbb{R}$) and encoded by a subpopulation of 250 input spiking neurons. In each step, different numeric values that instantiated the variables $x_1, \ldots, x_5$ were given, but the network had to learn the binding ("variable - concrete number"), and also to ignore all unnecessary values if the variable did not appear in the current expression. No special modules that would help the binding of variables with numeric values were included. Each analog value used in arithmetic expressions was encoded using a population coding scheme with a population of 50 input neurons. Neurons in this population were tuned to numbers using Gaussian tuning curves where neuron i responded maximally with a firing rate of $f_{\max} = 200$ Hz if the represented number was equal to its preferred value m_i. The preferred values m_i for these 50 neurons were chosen equally spaced in the input domain $[-1, 1]$. The firing rate f_i of neuron i was given by

$$f_i = f_{\max} \exp\Big(- \frac{(m_i - z)^2}{2\sigma^2} \Big), \tag{9}$$

where z is the represented value and the standard deviation of $\sigma = 0.08$ was equal for all input neurons. Each step lasted 50 ms, and consequently, a duration of a trial was 500 ms. From 300 recurrently connected LIF neurons, a half (150 neurons) had an SFA mechanism. Adaptation time constant for each of the neurons with SFA was chosen randomly from a uniform distribution $[1, 500]$. The output was a linear readout – in each step, output weights were multiplied with the average firing activity of the recurrently connected neurons. This output per step was a single real value.

The network was trained by minimizing the Mean-Squared-Error (MSE) between the network outputs and targets for each step. The final test MSE of 0.0341, and mean absolute error (MAE) of 0.1307 were achieved, calculated for all computational steps in all of, in total, 5000 trials. All test trials had novel sequences of computational steps, that is, test trials were different from those used for training.

References

T. E. Behrens, T. H. Muller, J. C. Whittington, S. Mark, A. B. Baram, K. L. Stachenfeld, and Z. Kurth-Nelson. What is a cognitive map? organizing knowledge for flexible behavior. *Neuron*, 100(2):490–509, 2018.

G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *32nd Conference on Neural Information Processing Systems (NIPS 2018), Montreal, Canada*, 2018.

G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11 (1):1–15, 2020.

M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

S. Dehaene. *The number sense: How the mind creates mathematics*. OUP USA, 2011.

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, 2012.

S. Furber. Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5): 051001, 2016.

M. S. Gazzaniga, R. B. Ivry, and G. Mangun. Cognitive neuroscience: The biology of the mind, 2009.

W. Gerstner and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

C. Gilmore, S. E. McCarthy, and E. S. Spelke. Symbolic arithmetic knowledge without instruction, Jan 2007. URL `https://hdl.handle.net/2134/8760`.

C. K. Gilmore, S. E. McCarthy, and E. S. Spelke. Non-symbolic arithmetic abilities and mathematics achievement in the first year of formal schooling. *Cognition*, 115(3):394–406, June 2010. ISSN 00100277. doi: 10.1016/j.cognition.2010.02.002. URL `https://linkinghub.elsevier.com/retrieve/pii/S0010027710000363`.

K. J. Hayworth. Dynamically partitionable autoassociative networks as a solution to the neural binding problem. *Frontiers in computational neuroscience*, 6:73, 2012.

K. J. Hayworth and A. H. Marblestone. How thalamic relays might orchestrate supervised deep training and symbolic computation in the brain. *bioRxiv*, page 304980, 2018.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

R. Kim, Y. Li, and T. J. Sejnowski. Simple framework for constructing functional spiking recurrent neural networks. *Proceedings of the national academy of sciences*, 116(45):22811–22820, 2019.

T. Kriete, D. C. Noelle, J. D. Cohen, and R. C. O'Reilly. Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences*, 110 (41):16390–16395, 2013.

Y. Liu, R. J. Dolan, Z. Kurth-Nelson, and T. E. Behrens. Human replay spontaneously reorganizes experience. *Cell*, 178(3):640–652, 2019.

W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(97)00011-7. URL `http://www.sciencedirect.com/science/article/pii/S0893608097000117`.

W. Maass. Energy-efficient neural network chips approach human recognition capabilities. *Proceedings of the National Academy of Sciences*, 2016. ISSN 0027-8424. doi: 10.1073/pnas.1614109113. URL `http://www.pnas.org/content/early/2016/10/03/1614109113`.

L. Manneschi and E. Vasilaki. An alternative to backpropagation through time. *Nature Machine Intelligence*, 2(3):155–156, 2020.

G. Marcus. Deep learning: A critical appraisal. *CoRR*, abs/1801.00631, 2018. URL `https://arxiv.org/pdf/1801.00631.pdf`.

G. Marcus, A. Marblestone, and T. Dean. The atoms of neural computation. *Science*, 346 (6209):551–552, 2014. ISSN 0036-8075. doi: 10.1126/science.1261661. URL `https://science.sciencemag.org/content/346/6209/551`.

G. F. Marcus. *The Algebraic Mind - Integrating connectionism and cognitive science*. MIT Press, 2001.

M. G. Müller, C. H. Papadimitriou, W. Maass, and R. Legenstein. A model for structured information representation in neural networks of the brain. *Eneuro*, 7(3), 2020.

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

A. Nieder. Supramodal numerosity selectivity of neurons in primate prefrontal and posterior parietal cortices. *Proceedings of the National Academy of Sciences*, 109(29):11860–11865, 2012.

A. Nieder and E. K. Miller. Coding of cognitive magnitude: Compressed scaling of numerical information in the primate prefrontal cortex. *Neuron*, 37(1):149–157, 2003.

R. C. O'Reilly. Biologically based computational models of high-level cognition. *Science*, 314 (5796):91–94, 2006.

T. A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3): 623–641, 1995.

C. Pozzorini, R. Naud, S. Mensi, and W. Gerstner. Temporal whitening by power-law adaptation in neocortical neurons. *Nature neuroscience*, 16(7):942–948, 2013.

C. Pozzorini, S. Mensi, O. Hagens, R. Naud, C. Koch, and W. Gerstner. Automated high-throughput characterization of single neurons by means of simplified spiking models. *PLoS Comput Biol*, 11(6):e1004275, 2015.

F. Pulvermüller. How neurons make meaning: brain mechanisms for embodied and abstract-symbolic semantics. *Trends in Cognitive Sciences*, 17(9):458–470, 2013.

M. Rigotti, O. Barak, M. R. Warden, X.-J. Wang, N. D. Daw, E. K. Miller, and S. Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590, 2013.

J. D. Roitman, E. M. Brannon, and M. L. Platt. Monotonic coding of numerosity in macaque lateral intraparietal area. *PLoS Biol*, 5(8):e208, 2007.

B. Rueckauer and S.-C. Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

D. Salaj, A. Subramoney, C. Kraišniković, G. Bellec, R. Legenstein, and W. Maass. Spike frequency adaptation supports network computations on temporally dispersed information. *eLife*, 2021.

C. Semenza, L. Miceli, and L. Girelli. A deficit for arithmetical procedures: Lack of knowledge or lack of monitoring? *Cortex*, 33(3):483–498, 1997.

H. Sheahan, F. Luyckx, S. Nelli, C. Teupe, and C. Summerfield. Neural state space alignment for magnitude generalization in humans and recurrent networks. *Neuron*, 109(7):1214–1226, 2021.

B. Spitzer, L. Waschke, and C. Summerfield. Selective overweighting of larger magnitudes during noisy numerical comparison. *Nature Human Behaviour*, 1(8):1–8, 2017.

C. Summerfield, F. Luyckx, and H. Sheahan. Structure learning and the posterior parietal cortex. *Progress in neurobiology*, 184:101717, 2020.

L. Teichmann, T. Grootswagers, T. Carlson, and A. N. Rich. Decoding digits and dice with magnetoencephalography: evidence for a shared representation of magnitude. *Journal of cognitive neuroscience*, 30(7):999–1010, 2018.

F. van der Velde and M. de Kamps. Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, 29(01):37–70, 2006.

L. Zamarian, A. Ischebeck, and M. Delazer. Neuroscience of learning arithmetic—evidence from brain imaging studies. *Neuroscience & Biobehavioral Reviews*, 33(6):909–925, 2009.

A. D. Zylberberg, L. Paz, P. R. Roelfsema, S. Dehaene, and M. Sigman. A neuronal device for the control of multi-step computations. *Papers in Physics*, 5:050006, 2013.

bioRxiv preprint doi: https://doi.org/10.1101/2021.07.14.452347; this version posted September 13, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

Acknowledgements

General: We would like to thank Anand Subramoney for his help and numerous discussions about the experiments done during the master's thesis of Ceca Kraišniković. These experiments were partly the motivation for the new experiments presented here.
Funding: This research was partially supported by the Human Brain Project (Grant Agreement number 785907 and 945539), the SYNCH project (Grant Agreement number 824162) of the European Union, and under partial support by the Austrian Science Fund (FWF) within the ERA-NET CHIST-ERA programme (project SMALL, project number I 4670-N).