Peer Community Journal

Section: Neuroscience

RESEARCH ARTICLE

Published
2021-12-15

Cite as
Michele Nardin, James W. Phillips, William F. Podlaski and Sander W. Keemink (2021) *Nonlinear computations in spiking neural networks through multiplicative synapses*, Peer Community Journal, 1: e68.

Correspondence
michele.nardin@ist.ac.at
sander.keemink@donders.ru.nl

Peer-review
Peer reviewed and recommended by PCI Neuroscience, https://doi.org/10.24072/pci.cneuro.100003

Nonlinear computations in spiking neural networks through multiplicative synapses

Michele Nardin [1], James W. Phillips [2,3], William F. Podlaski [4], and Sander W. Keemink [5,6]

Volume **1** (2021), article e68

https://doi.org/10.24072/pcjournal.69

Abstract

The brain efficiently performs nonlinear computations through its intricate networks of spiking neurons, but how this is done remains elusive. While nonlinear computations can be implemented successfully in spiking neural networks, this requires supervised training and the resulting connectivity can be hard to interpret. In contrast, the required connectivity for any computation in the form of a linear dynamical system can be directly derived and understood with the spike coding network (SCN) framework. These networks also have biologically realistic activity patterns and are highly robust to cell death. Here we extend the SCN framework to directly implement any polynomial dynamical system, without the need for training. This results in networks requiring a mix of synapse types (fast, slow, and multiplicative), which we term multiplicative spike coding networks (mSCNs). Using mSCNs, we demonstrate how to directly derive the required connectivity for several nonlinear dynamical systems. We also show how to carry out higher-order polynomials with coupled networks that use only pair-wise multiplicative synapses, and provide expected numbers of connections for each synapse type. Overall, our work demonstrates a novel method for implementing nonlinear computations in spiking neural networks, while keeping the attractive features of standard SCNs (robustness, realistic activity patterns, and interpretable connectivity). Finally, we discuss the biological plausibility of our approach, and how the high accuracy and robustness of the approach may be of interest for neuromorphic computing.

[1]Institute of Science and Technology Austria, Klosterneuburg, Austria, [2]Current affiliation: UCL Department of Science, Technology, Engineering and Public Policy (STEaPP), University College London, London, UK, [3]Independent researcher, [4]Champalimaud Research, Champalimaud Centre for the Unknown, Lisbon, Portugal, [5]Artificial Intelligence, Donders Institute for Brain, Cognition and Behaviour, Radboud University,, [6]Nijmegen, the Netherlands

Peer Community Journal is a member of the Centre Mersenne for Open Scientific Publishing
http://www.centre-mersenne.org/

e-ISSN 2804-3871

Contents

1. Introduction

A central quest in neuroscience is to understand how the brain's neural networks are able to perform the computations needed to solve complex tasks. One promising hypothesis is that networks represent relatively low-dimensional signals (compared to network size) (Cunningham and Yu, 2014; Keemink and Machens, 2019), and in this lower-dimensional space implement nonlinear dynamical systems through recurrent connectivity (Larry F Abbott et al., 2016; Barak, 2017; Eliasmith, 2005; Mante et al., 2013; Mastrogiuseppe and Ostojic, 2018; Sussillo, 2014). The resulting networks usually achieve nonlinear computation through a basis-function approach: non-linearities at various levels (neural (Eliasmith, 2005; Jaeger, 2001; Maass et al., 2002; Mastrogiuseppe and Ostojic, 2018), synaptic (Thalmeier et al., 2016), or dendritic (Larry F Abbott et al., 2016; Alemi et al., 2018; Thalmeier et al., 2016)) are weighted to achieve a given computation through supervised training. The task of achieving nonlinear computation is then off-loaded to the basis-functions and the training method, and as a result the link between network connectivity and computation may be unclear. Additionally (unlike real biological systems), the resulting

models do not always exhibit robustness to perturbations (Li et al., 2016), and suffer from unrealistic activity levels (e.g. (Eliasmith, TC Stewart, et al., 2012)) compared to expected levels (Barth and Poulet, 2012). In contrast, the recurrent connectivity required for any *linear* dynamical system can be directly defined (i.e., without any training) for a spiking neural network through the theory of spike coding networks (SCNs) (Boerlin et al., 2013). SCNs are consistent with many features from biology, such as sparse and irregular activity, robustness to perturbations (such as cell death) (Barrett et al., 2016; Calaim et al., 2020), and excitation/inhibition-balance (Boerlin et al., 2013; Denève and Machens, 2016). Could we use a similar analytical approach to introduce nonlinear computations?

In SCN theory, fast recurrent connections are used to efficiently and accurately maintain a stable internal representation. Any linear dynamical system can then be directly implemented through the addition of slower recurrent connections (Boerlin et al., 2013), which will drift the internal representation according to the desired dynamics. While nonlinear dynamics have previously been implemented in SCNs, this was achieved through the aforementioned basis-function approach (Larry F Abbott et al., 2016; Alemi et al., 2018; Thalmeier et al., 2016). Here we extend the original SCN derivation for linear dynamics (Boerlin et al., 2013), by directly deriving the connectivity required for any polynomial dynamical system. The resulting networks require an additional set of slow connections with multiplicatively interacting synapses, and we thus term our model *multiplicative* SCNs, or mSCNs.

We demonstrate the capability of mSCNs through a precise implementation of the Lorenz system, as well as an implementation of a double pendulum. While polynomial systems can in principle approximate any other system (De Branges, 1959), this can quickly become infeasible, as higher-order polynomial systems require higher-order synapses (pair-wise, triplets, quadruplets, etc.), resulting in a dense and complicated all-to-all connectivity structure. We address the need for higher-order synapses by demonstrating how higher-order computations can be approximated by successive network layers with solely pair-wise synapses. Additionally, we demonstrate that the assumption of all-to-all connectivity can be loosened if each neuron is selective only for a subset of the relevant variables for the computation.

Our theory of mSCNs harnesses all the appealing properties of previous SCN implementations (in particular robustness to cell-death and irregular spiking activity), but now includes a directly derivable and more interpretable connectivity structure for a large class of nonlinear dynamical systems. Finally, the efficiency and accuracy of our networks might be of use for neuromorphic applications, especially for representing dynamical systems that are well-described by lower-order polynomials.

2. Spike coding networks

Here and in the following sections we present the main results and refer the reader to the Methods section for details. Consider a network of N spiking neurons, which emit spike trains of the form $s_i(t) = \sum_k \delta(t_k^i - t)$, where δ is the Dirac delta function and $\{t_k^i \geq 0\}$ is the set of discrete times at which a spike was emitted. The population spike train is described by the vector $\mathbf{s}(t) = [s_1(t), \ldots, s_N(t)]^\mathsf{T}$. Vectors will be denoted by lower case bold letters, and wherever possible we will omit the time index for the sake of text clarity.

2.1. Linear autoencoder.

Suppose that a given $K-$dimensional signal $\mathbf{x}(t) \in \mathbb{R}^K$ should be represented by the output activity of the network (Fig. 1A). How should the neurons then spike to accomplish this task? The theory of spike coding networks approaches this through two core assumptions (Barrett et al., 2016; Boerlin et al., 2013; Denève and Machens, 2016):

(a) Linear decoding: the network representation $\hat{\mathbf{x}}(t)$ is read out as

$$\hat{\mathbf{x}} = \mathbf{D}\mathbf{r}$$

where $\mathbf{D} \in \mathbb{R}^{K\times N}$ is the decoding matrix, and $\mathbf{r}(t) = [r_1(t), \dots, r_N(t)]^\mathsf{T}$ are filtered spike-trains

$$\dot{\mathbf{r}} = -\lambda\mathbf{r} + \mathbf{s},$$

where λ is the leak time-constant. The variable $\mathbf{r}$ can be seen as a neuron's time-dependent rate, or equivalently the effect of a neuron's spikes on the post-synaptic potential of other neurons.

(b) Efficient spiking: $\mathbf{D}_i$ (the $i-$th column vector of the matrix $\mathbf{D}$) represents the contribution that a spike from neuron i will have on each dimension of the read-out signal; more specifically, a spike at time t will update the current readout as $\hat{\mathbf{x}}(t) \to \hat{\mathbf{x}}(t) + \mathbf{D}_i$. Assumption (b) requires that this spike should only occur if it improves the read-out. Formally, we require that a spike reduces the ℓ^2-error between the readout and the signal. Thus, neuron i will fire at time t if and only if

$$\|\mathbf{x}(t) - (\hat{\mathbf{x}}(t) + \mathbf{D}_i)\|_2^2 < \|\mathbf{x}(t) - \hat{\mathbf{x}}(t)\|_2^2 .$$

After some algebra (see Methods 7.1), and defining the membrane potential of each neuron as $V_i = \mathbf{D}_i^\mathsf{T}(\hat{\mathbf{x}} - \mathbf{x})$, one finds an underlying dynamical description of the system where each neuron spikes whenever $V_i > T_i$, with $T_i = \mathbf{D}_i^\mathsf{T}\mathbf{D}_i/2$, and membrane potential dynamics

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}(\dot{\mathbf{x}} + \lambda\mathbf{x}) - \mathbf{D}^\mathsf{T}\mathbf{D}\mathbf{s}, \tag{1}$$

where $\mathbf{v} = (V_1, \dots, V_N)$. Thus, starting from the two core assumptions, we have derived a recurrently connected network of leaky integrate-and-fire neurons. Through recurrent connections (given by $\mathbf{D}^\mathsf{T}\mathbf{D}$) the network accurately tracks its input signal $\mathbf{x}$, (Fig. 1B+C top row).

2.2. Linear dynamics.

In the above derivation, the signal $\mathbf{x}$ was provided directly to the network, but this is not strictly necessary. If $\mathbf{x}$ follows some known linear dynamics $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ (with $A \in \mathbb{R}^{K\times K}$) then its trajectory can be computed by the network (Boerlin et al., 2013). Their derivation uses the fact that $\mathbf{x} \approx \hat{\mathbf{x}}$, so that (1) can be approximated as

$$\begin{aligned}\dot{\mathbf{v}} &= -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}\left(\mathbf{A}\hat{\mathbf{x}} + \lambda\hat{\mathbf{x}}\right) - \mathbf{D}^\mathsf{T}\mathbf{D}\mathbf{s} \\ &= -\lambda\mathbf{v} + \mathbf{\Omega}_f\mathbf{s} + \mathbf{\Omega}_s\mathbf{r},\end{aligned} \tag{2}$$

where so-called "fast" connections $\mathbf{\Omega}_f = -\mathbf{D}^\mathsf{T}\mathbf{D}$ keep the error constrained on a short timescale (Fig. 1B+C top row), and "slow" connections $\mathbf{\Omega}_s = \mathbf{D}^\mathsf{T}(\mathbf{A} + \lambda\mathbf{I})\mathbf{D}$ implement the dynamical computation using the filtered spikes $\mathbf{r}$ (Fig. 1B+C middle row). Here "fast" and "slow" refer to the rise-time of the synaptic PSPs (Fig. 1B). While technically an approximation, this implementation works well in practice and can closely reproduce a given linear dynamical system (Fig. 1B+C middle row).

3. Nonlinear dynamics

The approximation in (2) was originally conceived for linear systems, but can in principle be extended to any arbitrary dynamical system $\dot{\mathbf{x}} = F(\mathbf{x})$ (with $F : \mathbb{R}^K \to \mathbb{R}^K$). The full network dynamics then become

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}(F(\hat{\mathbf{x}}) + \lambda\hat{\mathbf{x}}) - \mathbf{D}^\mathsf{T}\mathbf{D}\mathbf{s}, \tag{3}$$

with the problem that the nonlinear function $F()$ has to be somehow computed by the network (or individual neurons). Previous work approximated this computation through a set of basis functions ((Alemi et al., 2018; Thalmeier et al., 2016); see Methods 7.6), which can be interpreted as dendritic nonlinearities. Here we take a different approach. We note that any smooth nonlinear function can in principle be approximated by a polynomial transformation (De Branges, 1959).

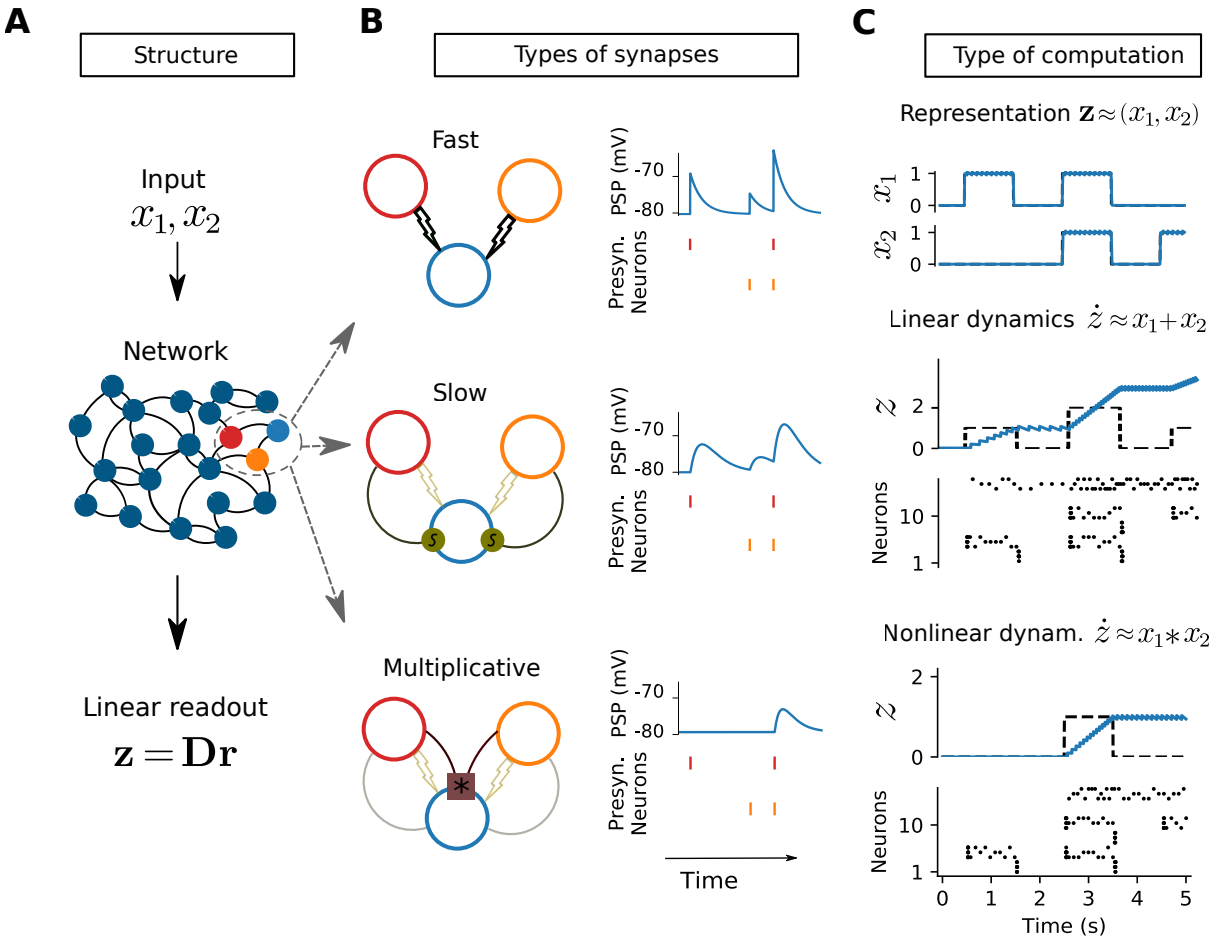


Figure 1 – Multiplicative Spike Coding Networks (mSCNs) can implement polynomial dynamics. **(A)** Schematic representation of the network. **(B)** The network has three types of synapses, illustrated for two neurons (red and orange) connecting to another (blue). The postsynaptic potential (PSP) of a cell endowed with multiplicative synapses will be affected only if the two presynaptic neurons fire very close in time to each other. Higher order multiplicative interactions can also be necessary, and are illustrated in Supp. Fig. S1. **(C)** Example computations enabled by the different types of synapses: (top) the network represents the two inputs (x_1 and x_2). The blue line represents the output of the network, the dashed black lines the real input. (Middle) A network which computes the dynamical system $\dot{z} = x_1 + x_2$. The black dashed line represents the real sum of the inputs, the blue line represents the output of the network. (Bottom) A network which computes the nonlinear dynamical system $\dot{z} = x_1 * x_2$, and thus integrates the product of x_1 and x_2. For both linear and nonlinear dynamics the fast synapses are also required.

Furthermore, any polynomial function $F : R^K \to R^K$ containing terms with maximum degree g can be written in the form

$$F(\mathbf{x}) = \sum_{d=0}^{g} \mathbf{A}_d \mathbf{x}^{\otimes d}, \tag{4}$$

where $\mathbf{A}_d \in \mathbb{R}^{K \times K^d}$ is the matrix of coefficients for the polynomials of degree d, and we define $\mathbf{M}^{\otimes d} = \mathbf{M} \otimes \mathbf{M} \otimes \cdots \otimes \mathbf{M}$ as the Kronecker product applied d times, with the convention that $\mathbf{M}^{\otimes 0} = 1$ and $\mathbf{M}^{\otimes 1} = \mathbf{M}$. The Kronecker product is closely related to the outer-product and computes all possible pair-wise multiplications between the elements of two matrices. For example, the Kronecker product of two vectors of length l is itself a new vector of length l^2 (see Methods 7.2 for a detailed explanation).

Using this notation, the connectivity and dynamics for the *multiplicative* SCN network implementing a polynomial function can be directly derived as

$$
\begin{aligned}
\dot{\mathbf{v}} &= -\lambda\mathbf{v} - \mathbf{D}^\mathsf{T}\mathbf{D}\mathbf{s} + \mathbf{D}^\mathsf{T}(\sum_{d=0}^{g}\mathbf{A}_d\mathbf{D}^{\otimes d}\mathbf{r}^{\otimes d} + \lambda\hat{\mathbf{x}}) \\
&= -\lambda\mathbf{v} + \mathbf{\Omega}_f\mathbf{s} + \mathbf{\Omega}_s^{m0} + \mathbf{\Omega}_s^{m1}\mathbf{r} + \mathbf{\Omega}_s^{m2}\mathbf{r}^{\otimes 2} + \cdots + \mathbf{\Omega}_s^{mg}\mathbf{r}^{\otimes d} \\
&= -\lambda\mathbf{v} + \mathbf{\Omega}_f\mathbf{s} + \sum_{d=0}^{g}\mathbf{\Omega}_s^{md}\mathbf{r}^{\otimes d},
\end{aligned}
\tag{5}
$$

where $\mathbf{\Omega}_f = -\mathbf{D}^\mathsf{T}\mathbf{D}$, $\mathbf{\Omega}_s^{m1} = \mathbf{D}^\mathsf{T}(\mathbf{A}_1 + \lambda\mathbf{I})\mathbf{D}$ and $\mathbf{\Omega}_s^{md} = \mathbf{D}^\mathsf{T}\mathbf{A}_d\mathbf{D}^{\otimes d}$ for $d \in \{0, 2, 3, \ldots, g\}$.

With this factorization we demonstrate how any given multiplicative interaction of state-variables can be accurately implemented through multiplicative synapses between neurons (Fig. 1 B+C bottom, Methods 7.4). The matrix $\mathbf{\Omega}_s^{md}$ then represents $d-$th degree multiplicative interactions between cells. In particular, $\mathbf{\Omega}_s^{m2}$ represents the connectivity required for each cell to multiply each pair of their inputs (Fig. 1 B,C bottom row), with the synapse essentially acting as a coincidence detector. Higher order synapses would behave similarly but with three or more coincident spikes (Supp. Fig. S1). While these higher order interactions are unlikely to be biologically feasible, lower order multiplicative interactions may indeed be possible in biology, and have been hypothesized before (Koch and Poggio, 1992), as we will discuss further in the discussion. In principle, increasingly complex nonlinear dynamics may be implemented through the inclusion of higher-order terms in eq. (5) ($\mathbf{\Omega}_{md}$ for $d > 2$), though this flexibility comes with increased cost on the number of synapses and neural interactions. In a later section we will show how to avoid interactions beyond pair-wise synapses, and we will derive the expected number of connections for each type of synapse.

Compared to linear dynamics, multiplicative synapses enable nonlinear computations such as AND gates (Fig. 1C bottom). Overall, the above derivation demonstrates that the presence of multiplicative synapses arises naturally in mSCNs from extending the spike coding framework to polynomial dynamical systems.

3.1. Lorenz attractor.

We illustrate the functionality of the mSCN formalism through an implementation of a simple dynamical system, the Lorenz attractor. The Lorenz attractor is a system of ordinary differential equations first studied by Edward Lorenz, which may lead to chaotic solutions (Lorenz, 1963; Strogatz, 2018). It is defined as

$$
\begin{aligned}
\dot{x} &= \sigma(y - x) \\
\dot{y} &= x(\rho - z) - y \\
\dot{z} &= xy - \beta z.
\end{aligned}
$$

Notably, this system contains pairwise multiplicative terms of the state-variables, thereby making it a polynomial (and nonlinear) dynamical system. In the following, we use the "classical" parameter values $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$, for which the system is in a chaotic regime. This is a useful case study as the resulting behavior is very sensitive to small representation errors, and has previously been used to test spiking network dynamical system implementations (e.g. (Thalmeier et al., 2016)).

We implemented the Lorenz system in three ways, each using networks of $N = 100$ neurons. First, we simulated the Lorenz system in a standard numerical simulation (Runge-Kutta method), and fed the dynamic variables $\mathbf{x}$ directly as input into an autoencoding network with only fast synapses. Note that the correct trajectory is thus continuously being fed into this network. This control acted as an upper-bound on the accuracy of representation with a spiking network of a given size and read-out weight magnitudes. As expected the network represented the system with high fidelity — only small deviations arose compared to the standard numerical simulation

due to the discrete spiking representation of the network (Fig. 2Aii). To have a better idea of the accuracy of the representation, we followed (Thalmeier et al., 2016) and compared the values of neighboring peaks in the dynamics of the z variable, which closely tracked a function defined by the pure Lorenz simulation (Fig. 2Aiv).

Next, we implemented the Lorenz system in an mSCN (following Eq. 5, see Methods 7.5). The resulting network is able to compute the Lorenz dynamics with high accuracy (Fig. 2B). The representation tracked the dynamics of the standard numerical simulation (dotted lines) for a reasonable amount of time, despite the attractor's chaotic nature (Fig. 2B i+ii), though this depends on the simulation time step (set to 0.1ms here). Additionally, despite the deviations from the 'true' trajectory, the peak analysis demonstrates that qualitatively the implementation is near perfect (Fig. 2B iv). Furthermore, the network simulation still displays the extreme robustness to cell death of traditional SCNs (Supp. Fig. S2).

Lastly, for comparison's sake, we implemented the same dynamical system using basis functions with trained weights (Methods 7.6) in order to understand the benefits and drawbacks of each approach. We used 500 basis functions per neuron. The Lorenz attractor is again qualitatively well reproduced (Fig. 2 top). But in contrast to the two previous schemes, the implementation with basis functions led to more inaccuracies (Fig. 2C), quickly resulting in missed shifts in the dynamics. This was likely due to the approximate nature of the basis function implementation, and we note that more precise simulations might be possible with more and different basis functions (see e.g. Fig. 3f-h in (Thalmeier et al., 2016); though even there, outliers are still present).

These results might suggest that the direct implementation of the Lorenz system with the mSCN is capable of more accurate dynamics than a basis function implementation. However, we note three caveats here. First, the accuracy of the dynamics depends on the nature of the underlying system — e.g., small inaccuracies would matter less for a system with stable fixed points. Second, the Lorenz attractor is perfectly described by a polynomial, and other dynamical systems might be better described by a basis-function implementation (of similar complexity as a given mSCN). Third, the differences in accuracy and scaling of the two implementations suggests that each may be more suitable depending upon the specific problem at hand (on the order of N^3 parameters are needed for the mSCN and bN^2 for the basis function implementation, where b is the number of basis functions).

4. Higher-order polynomials with sequential networks

While the Lorenz system is a good case study for demonstrating the power and accuracy of mSCNs, a core problem remains: higher order polynomials necessitate higher order multiplicative interactions. E.g., a polynomial of order 3 would require a $\mathbf{r}^{\otimes 3}$ term, with on the order of N^4 synapses. Such precise higher-order interactions may not always be feasible, either biologically or on a neuromorphic substrate. However, as we show here, this is not strictly necessary. Across populations, it is possible to combine many sequential pairwise interactions to achieve multiplications of any other order.

4.1. Input transformations.

In principle, an SCN can also represent a nonlinear transformation of a signal $G(\mathbf{x})$, where G is a smooth function $G : \mathbb{R}^K \to \mathbb{R}^M$, $M \geq 1$. For this, consider a new SCN with decoder $\mathbf{W}$, spikes $\boldsymbol{\sigma}$, and filtered spike trains $\boldsymbol{\rho}$. In that case $\mathbf{v}$ evolves according to

$$\dot{\mathbf{v}} = -\lambda \mathbf{v} + \mathbf{W}^\mathsf{T} \left(\mathbf{J}_G(\mathbf{x}) \dot{\mathbf{x}} + \lambda G(\mathbf{x}) \right) - \mathbf{W}^\mathsf{T} \mathbf{W} \boldsymbol{\sigma}, \tag{6}$$

where $\mathbf{J}_G$ is the Jacobian of G (Methods 7.1). The problem here is that the transformation function has to be either provided or computed by the network. However, if G is a polynomial function, the computation can be implemented using multiplicative synapses — e.g., for quadratic terms, we can use $G(\mathbf{x}) = \mathbf{x} \otimes \mathbf{x}$. Specifically, we need a network that takes as input the output

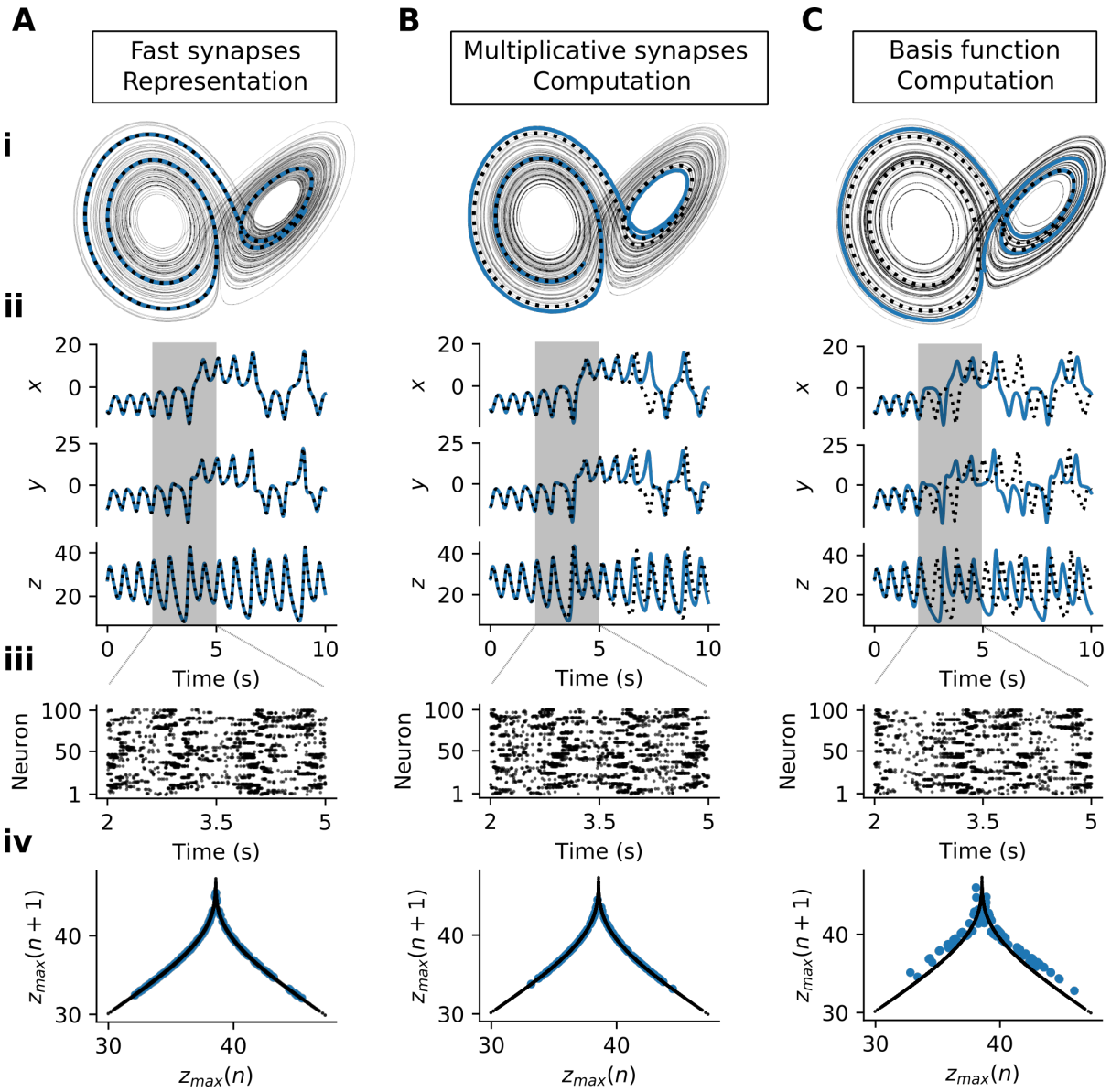


Figure 2 – Implementation of a Lorenz dynamical system. Across columns: **(A)** The numerical solution was found using an explicit Runge-Kutta method of order 4. The network contains only fast synapses and receives as input the output of the Lorenz simulation, which the network read-out tracks closely. **(B)** A network with multiplicative synapses (mSCN) computing the Lorenz attractor through its network dynamics. **(C)** A network with nonlinear basis functions computing the Lorenz attractor through its network dynamics. Across rows: **(i)** 3D view of the network readout for 100 sec (grey). The dotted line shows the 'true' simulation in the 2-5sec period, and the blue line shows the corresponding network output trajectory. **(ii)** Each network readout dimension (blue) across time vs the 'true' solution (black dotted). The gray region indicates the 2-5sec period used in panels i and iii. **(iii)** Raster plot with spikes emitted by the neurons in the time interval 2 - 5 sec. **(iv)** Peak analysis over 100 sec: blue = network output, black = 'true' Lorenz simulation.

of another network $\hat{\mathbf{x}} = \mathbf{D}\mathbf{r}$ (with spikes $\mathbf{s}$) and returns $\mathbf{W}\boldsymbol{\rho} \approx \hat{\mathbf{x}}^{\otimes 2}$ (Methods 7.3). Using the fact that $\dot{\mathbf{r}} = -\lambda\mathbf{r} + \mathbf{s}$, we obtain dynamics

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \boldsymbol{\Omega}_x(\mathbf{r}\otimes\mathbf{s} + \mathbf{s}\otimes\mathbf{r} + -\lambda\mathbf{r}\otimes\mathbf{r}) + \boldsymbol{\Omega}_f^W\boldsymbol{\sigma},$$

with $\boldsymbol{\Omega}_x = \mathbf{W}^\mathsf{T}(\mathbf{D}\otimes\mathbf{D})$ and $\boldsymbol{\Omega}_f^W = -\mathbf{W}^\mathsf{T}\mathbf{W}$. We illustrate the resulting network and its inputs and outputs in Supp. Fig. S3.

4.2. Combining networks.

Now, the combination of a standard mSCN (eq. 5) with a network which calculates the square of its inputs (Supp. Fig. S3), results in a system with the ability of computing third-order multiplications with only pairwise (second-order) synapses (Fig. 3A). Notably, one network computes the pairwise multiplications, and the other computes the desired third-order dynamic equation using another pairwise multiplication of the squared network output ($\mathbf{x}\otimes\mathbf{x}$) and $\mathbf{x}$.

More concretely, we consider a polynomial function $F : \mathbb{R}^K \to \mathbb{R}^K$ with maximum degree $g = 3$. We can write $F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{x}^{\otimes 2} + \mathbf{C}\mathbf{x}^{\otimes 3}$ using eq. (4). Naively, a network of neurons that approximates the solution to $\dot{\mathbf{x}} = F(\mathbf{x})$ can be written using eq. (5) as

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{\Omega}_f\mathbf{s} + \mathbf{\Omega}_s^{m1}\mathbf{r} + \mathbf{\Omega}_s^{m2}\mathbf{r}^{\otimes 2} + \mathbf{\Omega}_s^{m3}\mathbf{r}^{\otimes 3},$$

which contains the third-order synapses in the last term. However, we now reintroduce the first network (from Section 4.1) which takes $\mathbf{D}\mathbf{r}$ as input and outputs $W\boldsymbol{\rho} \approx \hat{\mathbf{x}}^{\otimes 2}$. This allows the term $\mathbf{\Omega}_s^{m3}\mathbf{r}^{\otimes 3}$ to be replaced by $\mathbf{D}^{\mathsf{T}}C(\mathbf{D}\otimes\mathbf{W})(\mathbf{r}\otimes\boldsymbol{\rho})$, yielding

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{\Omega}_f\mathbf{s} + \mathbf{\Omega}_s^{m1}\mathbf{r} + \mathbf{\Omega}_s^{m2}\mathbf{r}^{\otimes 2} + \mathbf{\Omega}_s^{ext}(\mathbf{r}\otimes\boldsymbol{\rho}), \tag{7}$$

where $\mathbf{\Omega}_s^{ext} = \mathbf{D}^{\mathsf{T}}\mathbf{C}(\mathbf{D}\otimes\mathbf{W})$. The same argument can be extended to higher order multiplications, at the cost of having $\lceil \log_2(g) - 1 \rceil$ support networks, where g is the maximum degree of $F()$ [1].

4.3. Example: approximating a double pendulum.

We illustrate the use of the higher-order polynomial implementation using the double pendulum as an example (Fig. 3B). Suppose that each pendulum has length l and mass m. We denote θ_1, θ_2 the angles of the first and second pendulum with respect to the vertical axis (i.e. $\theta_i = 0$ when the pendulum is pointing downwards), and p_{θ_1} and p_{θ_2} their momenta. The full double pendulum dynamics can be derived using the Lagrangian (Methods 7.7; e.g., (Levien and Tan, 1993)). For small angles one can consider the approximation $\sin\theta \approx \theta$ and $\cos\theta \approx 1$, which leads to the following approximated dynamics:

$$\begin{aligned}
\dot{\theta}_1 &= \frac{6}{7ml^2}\left(2p_{\theta_1} - 3p_{\theta_2}\right) \\
\dot{\theta}_2 &= \frac{6}{7ml^2}\left(8p_{\theta_2} - 3p_{\theta_1}\right) \\
\dot{p}_{\theta_1} &= -\tfrac{1}{2}ml^2\left(\dot{\theta}_1\dot{\theta}_2(\theta_1 - \theta_2) + 3\frac{g}{l}\theta_1\right) \\
\dot{p}_{\theta_2} &= -\tfrac{1}{2}ml^2\left(-\dot{\theta}_1\dot{\theta}_2(\theta_1 - \theta_2) + \frac{g}{l}\theta_2\right).
\end{aligned}$$

We denote $\mathbf{x} = (\theta_1, \theta_2, p_{\theta_1}, p_{\theta_2})^{\mathsf{T}}$, and rewrite the system as $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{C}\mathbf{x}^{\otimes 3}$ ($\mathbf{A}$ and $\mathbf{C}$ defined explicitly in Methods 7.7).

We implemented the first-order approximated double pendulum system in three distinct ways. As before for the Lorenz system, we first simulated a control network that was simply asked to autoencode the dynamics directly, which were computed externally. Next, we implemented two mSCNs — one network computed the dynamics through explicit third-order multiplicative synapses (as in eq. (5)). The other implementation utilized a support network (as explained in eq. (7)), allowing the main network to avoid explicit third-order interactions. We found that these two mSCN implementations produced accurate representations of the dynamics, closely following the autoencoder network which received the "true" solution directly.

[1] The maximum multiplication order that can be computed with k networks is 2^k - in fact, the first support network can compute $\hat{x}^{\otimes 2}$, the second can take as input the output of the first and compute $\hat{x}^{\otimes 4}$, and so on.

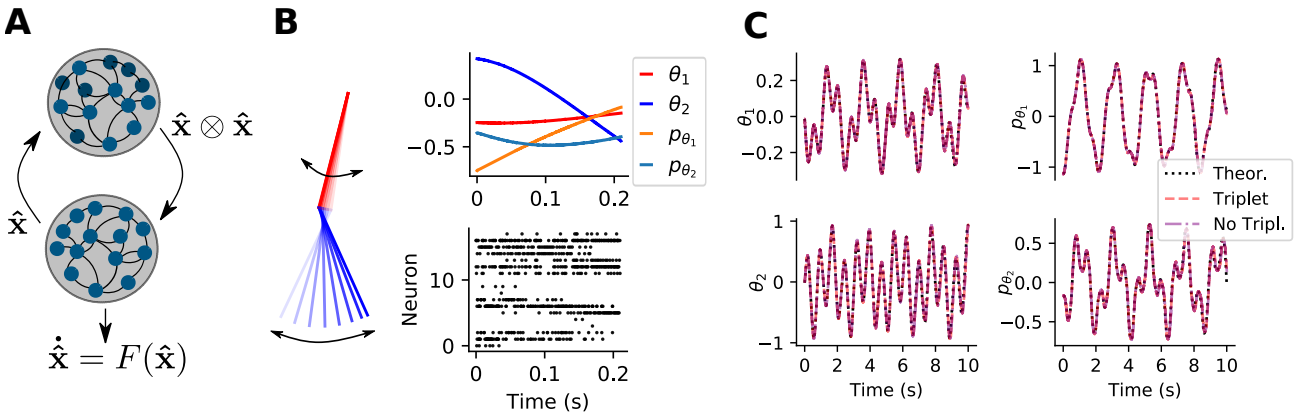


Figure 3 – Third order polynomial dynamics solved by sequential pairwise multiplications. **(A)** To avoid third order multiplications, another network can be used whose output will be the pairwise multiplication of any two input dimensions (which can be done through only pair-wise synapses). **(B)** Example output of a network computing the double pendulum and using an external network to avoid the third order multiplications. **(C)** Solution computed by employing a neural network with third order synapses (dashed line) or employing two neural networks to avoid the third order multiplications (dash-dotted line) compared to the numerical solution of dynamical system (dotted line). All solutions almost perfectly overlap.

5. On the number of required connections

As shown in the previous sections, mSCNs offer a powerful and intuitive way of implementing polynomial dynamical systems in spiking networks. However, though they may be efficient with respect to the number of neurons and spikes required, they can require dense synaptic connections (sometimes with several connections for each pair of neurons). In the standard SCN framework, any two neurons can be connected by fast and slow synapses. In mSCNs, additional connections are introduced with the multiplicative synapses. So far we have assumed full all-to-all connectivity of all types of connections (i.e., N^2 fast connections, N^2 slow connections, N^3 pair-wise multiplicative synapses, and so on). However, connectivity in the brain is known to be sparse (Lefort et al., 2009; Song et al., 2005). Given this constraint it is important to understand how mSCNs can be constructed with sparser connectivity, and the relationship between connectivity density and performance.

Consider networks of N neurons representing a K-dimensional signal space, with decoding matrix $\mathbf{D} \in \mathbb{R}^{K \times N}$. The $i-$th column vector of the matrix $\mathbf{D}$, denoted by $\mathbf{D}_i$, represents the weights associated to neuron i. Here and in the following we will say that "neuron i codes for dimension x" meaning that the $x-$th entry of $\mathbf{D}_i$ is $\neq 0$. We will define the connectivity density as the proportion of the all-to-all connectivity which is being used. Connectivity density is then determined by the decoder matrix $\mathbf{D}$ and some fixed matrices $\mathbf{A}_d$ given by the dynamical system, as explained in eq. (5). Thus far, we have considered that each neuron codes for all dimensions, meaning that $\mathbf{D}$ is dense and connectivity is all-to-all for each synapse type. If neurons instead only coded for a subset of the dimensions, how sparse would the connectivity be? We will make this explicit by giving each neuron a fixed probability p to code for each signal dimension (i.e. p is the probability that a given matrix entry in $\mathbf{D}$ is non-zero). If $p = 1$, then all neurons will code for all signal dimensions. As p approaches zero, neurons will code for progressively less dimensions.

Consider the three connectivity matrices required for second-order multiplicative computations: the fast connections $\mathbf{\Omega}_f = \mathbf{D}^\mathsf{T}\mathbf{D}$, the slow connections $\mathbf{\Omega}_s^{m1} = \mathbf{D}^\mathsf{T}(\mathbf{A} + \lambda\mathbf{I})\mathbf{D}$ and the multiplicative connections $\mathbf{\Omega}_s^{m2} = \mathbf{D}^\mathsf{T}\mathbf{B}\mathbf{D} \otimes \mathbf{D}$. In Fig. 4 we investigate the relationship between decoder density and final network connectivity density. We calculate theoretical upper-bounds on the expected connectivity and report here the asymptotic behavior in terms of density (see

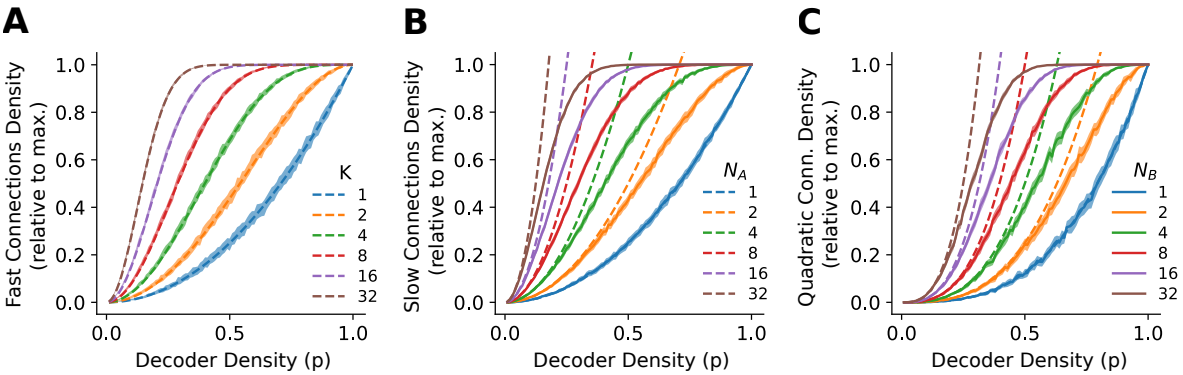


Figure 4 – Expected connectivity density relative to all-to-all connectivity for the different synapse types as a function of the decoder density p. Solid lines illustrate mean across iterations, whereas shaded areas represent ± 1 standard deviation from the average of the simulated connectivity (see Methods 7.8). Dashed lines correspond to theoretical upper bounds (Eq. 8-10). (**A**) The connectivity density for the fast synapses for different signal dimensions (K). (**B**) The connectivity density for the slow synapses due to linear computation (i.e. $\sim N_A p^2$), for different linear dynamical system densities (N_A). (**C**) The connectivity density for slow multiplicative (quadratic) synapses, for different quadratic dynamical system densities (N_B). N_A and N_B represent the number of non-zero entries of matrices $\mathbf{A}$ and $\mathbf{B}$ respectively.

Methods 7.8 for detailed derivations). Additionally, by generating random decoding matrices for different probabilities p, we measure the empirical expected connectivity density. The fast connection density (defined as the proportion of maximum number of connections used) depends on both p and the signal dimension K and behaves as

$$\mathbb{E}(\texttt{fast conn. dens.}) \sim 1 - (1 - p^2)^K. \tag{8}$$

Although the density rises sharply as the signal dimensionality increases (Fig. 4A), we see that for low to moderate p and K the connectivity density is far below all-to-all. The slow and multiplicative connection densities additionally depend on the 'density' of the dynamical system interactions, i.e. the number of non-zero elements in the matrices $\mathbf{A}$ and $\mathbf{B}$, denoted with N_A and N_B respectively. We find theoretical upper bounds on the expected densities

$$\mathbb{E}(\texttt{slow conn. dens.}) \lesssim \mathbb{E}(\texttt{fast conn. dens.}) + N_A p^2, \tag{9}$$

for slow connections (Fig. 4B) and

$$\mathbb{E}(\texttt{multiplicative conn. dens.}) \lesssim N_B p^3, \tag{10}$$

for multiplicative connections (Fig. 4C, see Methods 7.8 for details). Notably, the multiplicative connections have a much slower rise compared to the fast connections, which may push the network closer to a biologically-plausible regime.

Such reduced connectivity density does come at some cost to performance, in particular in robustness. We demonstrate this by considering the Lorenz attractor implementation with different values for p, and by considering different numbers of active neurons. We found that decreasing the decoder density made the network more prone to errors as the number of lost cells increased (Supp. Fig. S2C+D). Nonetheless, this increase in error was nearly nonexistent and became sizeable only after killing more than 80% of the cells.

6. Discussion

In this report, we investigated a new approach for implementing nonlinear dynamical systems in spiking neural networks. We extended the spike coding network (SCN) framework (Boerlin et al., 2013; Calaim et al., 2020; Denève and Machens, 2016) to implement arbitrary polynomial dynamics. We obtain the multiplicative SCN (mSCN), which requires fast, slow, and multiplicative connections. For second-order systems, the connectivity requires pair-wise multiplicative

synapses, and we demonstrated how higher-order multiplications can be implemented in several network stages with only pair-wise synapses. We demonstrated the accuracy and flexibility of this formalism by simulating the Lorenz system and a third order approximation of a double pendulum. Due to the rich flexibility of polynomials for approximating arbitrary nonlinear functions (Stone-Weierstrass Theorem or Taylor Expansion), this approach could in principle be extended to many other systems for which a lower-order polynomial approximation is sufficient. Lastly, we analyzed the relationship between the sparsity of signal coding per neuron and the connectivity sparsity (for fast, slow, and multiplicative connections), and showed how the need for all-to-all connectivity can be relaxed.

6.1. Related work.

The study of nonlinear computation has a long history in computational neuroscience, and has traditionally been studied in firing-rate networks, in which each neuron is represented by a continuous (or sometimes binary) variable (Dayan and Laurence F Abbott, 2001). When such neurons are endowed with a nonlinear input-output function, nonlinear computations are possible (Jaeger, 2001; Mante et al., 2013; Mastrogiuseppe and Ostojic, 2018; Rubin et al., 2015; Sussillo and Larry F Abbott, 2009).

In spiking neural networks (SNNs) nonlinear computations have previously been achieved by using random connectivity as a basis for complex dynamics (Maass et al., 2002), or using supervised training algorithms to optimize the networks (Neftci et al., 2019). Alternatively, more principled approaches to building spiking neural networks include the neural engineering framework (NEF) (Eliasmith and Anderson, 2004) and SCNs (Boerlin et al., 2013), the latter of which we studied here. Nonlinear dynamical system implementations in previous SNNs usually harnessed various nonlinearities (e.g. neural, dendritic, or synaptic) as basis functions, with the connectivity required for a given dynamical system implemented through supervised training (as we also implemented in Fig. 2C). Previous nonlinear computations in SCNs also relied on such basis functions (Alemi et al., 2018; Thalmeier et al., 2016). The basis function approach works very well, but does not allow for a direct derivation of the connectivity given a nonlinear dynamical system and thus lacks a well-defined mapping between computation and connectivity. In contrast, in mSCNs the connectivity is defined directly from the dynamical system, and is therefore more interpretable.

Finally, a more recent approach has emerged in which spiking networks are treated as piecewise linear functions, depending on which neurons in the network are active (Baker et al., 2020; Mancoo et al., 2020) (which may also explain the success of supervised training algorithms without explicit basis functions, e.g., (Zenke and Vogels, 2021)). These methods can be seen as complementary to our approach, in which case a nonlinear input-output function would be combined with nonlinear dynamics through additional slow connections.

6.2. Biological implications.

mSCNs represent a novel hypothesis of nonlinear computation in neural circuits, which can be seen as complementary to previous basis-function approaches. While previous theoretical studies have noted the wide range of nonlinear computations that multiplicative synapses could in principle enable (Koch and Poggio, 1992; Nezis and Rossum, 2011; Salinas and Laurence F Abbott, 1996), our work provides a precise formalization on how to use such synapses to implement any polynomial dynamical system efficiently in a spiking neural network. Additionally, the resulting networks inherit the attractive features from standard SCNs which match well to biological network activity (e.g. irregular and sparse activity, robustness, and E/I balance). But how biologically feasible are the the required synapse interactions and network connectivities?

First, the resulting network connectivity is extremely dense: by default, mSCN networks predict all-to-all connectivity. While local circuits in the cortex are indeed densely connected (Fino and Yuste, 2011; Harris and Mrsic-Flogel, 2013), connectivity is not all-to-all (Ko et al., 2011;

Perin et al., 2011; Song et al., 2005). However, as we showed in Fig. 4, full all-to-all connectivity is not required for mSCNs to function. The connectivity density is a function of the fraction of variables each neuron codes for, which is formalized by imposing sparsity of the decoder matrix. Consequently, according to SCN theory, if all neurons would code for all dimensions, this would be reflected in an all-to-all connectivity structure. Conversely, if neurons would code for a subset of the dimensions, this would be reflected in a less dense connectivity structure (but also, accordingly, a less robust code). This last property is not easily measured in the brain, where the amount of variables each neuron codes for is not clear. While current hypotheses emphasize that neurons exhibit mixed selectivity to many task variables (Rigotti et al., 2013), recent evidence also suggests that cortical representations are very high-dimensional (Stringer, Pachitariu, Steinmetz, Carandini, et al., 2019; Stringer, Pachitariu, Steinmetz, Reddy, et al., 2019). Therefore, it is plausible that neurons can display mixed selectivity while also only coding for a small subset of the overall coding space, leaving room for a more sparse implementation of our model.

Second, mSCNs require each pair of neurons to have three types of synapses connecting to each other (Fig. 1). The presence of several distinct synaptic connections between pairs of cells have indeed been observed in experimental circuit reconstruction studies (Kasthuri et al., 2015; Popov and MG Stewart, 2009). However, the feasibility of the precise multiplicative interactions is less clear. We can at the very least state that a form of multiplication must be performed at some level in biological networks. Several examples of effectively multiplicative computations have been characterized from experimental studies (Arandia-Romero et al., 2016; Gabbiani et al., 2002; Peña and Konishi, 2001; Zhou et al., 2007), and multiplicative interactions have been hypothesized to exist in the dendritic tree (London and Häusser, 2005). Indeed, experimental and theoretical work has long shown the computational advantages of nonlinear synaptic and dendritic interactions in single neurons (London and Häusser, 2005; Poirazi et al., 2003). Mechanisms such as dendritic calcium or NMDA spikes (Augustine et al., 2003; Schiller et al., 2000), synaptic clustering (Larkum and Nevian, 2008), and shunting inhibition (Mitchell and Silver, 2003; Zhang et al., 2013) are well established and could contribute to a code relying on multiplicative interactions (as detailed in (Koch and Poggio, 1992)). Finally, even if not fully feasible, for a given polynomial system mSCNs could be seen as defining the *ideal* set of interactions required for a given network of neurons, which would then have to be replicated by a given network either by a basis function approach or through non-optimal multiplicative interactions. Future work will have to show how well a more biological implementation of multiplicative interactions would allow for precise nonlinear computations.

Finally, while higher order multiplicative interactions are increasingly difficult to implement biologically, we demonstrated that by stacking networks with lower-order interactions one can achieve the same computations. This makes the concrete prediction that the connectivity between areas should be of the same dimensionality as the signal being transferred. This does indeed seem to be the case to some degree, with the communication between some areas being low-dimensional (Semedo et al., 2019). There is also a likely limit to how many networks can be effectively stacked in this way to perform higher-order interactions, as each stacked network introduces a delay.

6.3. Computational and neuromorphic applications.

Even given potential biological limitations, we contend that mSCNs offer two benefits. First, the fact that the implementation of polynomial dynamics is direct and explicit implies that this technique offers a useful comparative control when considering the possible computations that spiking networks can perform, as well as the limits of their accuracy. This model may therefore serve as a useful reference for future studies. Second, neuromorphic implementations of spiking neurons are becoming increasingly feasible (Young et al., 2019). As many of these rely on networks of integrate-and-fire type neurons (Davies et al., 2018; Indiveri et al., 2011; Merolla et al., 2014), it is in principle less constrained by biological plausibility. In this context, the need for dense pair-wise connectivity and precise multiplicative interactions may be an acceptable

cost for a precise connectivity recipe for a given nonlinear computation with fewer neurons. For example, a Lorenz attractor can be efficiently and precisely implemented with just 10 neurons (Supp. Fig. S2), in contrast to the 1600 neurons used in (Thalmeier et al., 2016).

6.4. Conclusion.

In sum, we have provided a proof of concept of direct and explicit polynomial dynamics implemented in spiking networks. Future directions include the application of this framework to other biologically-plausible and neuromorphic computations, a study of the efficiency of this framework, and the potential for biologically-plausible learning of the connectivity.

Author Contributions

MN contributed to discussions, initial and follow-up implementations, mathematical derivations and writing of the paper; JWP contributed to discussions and initial implementations; WFP contributed to discussions, supervision, and writing of the paper; SWK conceived the initial idea, contributed to supervision, discussions and initial implementation, and writing of the paper.

Supplementary material

Script and codes are available online: `https://github.com/michnard/mult_synapses`

7. Methods

7.1. General derivation of spike coding network.

We will show here a generalization of the derivation of spike-coding networks (SCNs) shown in Barrett et al., 2016, ignoring the constraint that neurons need to be either excitatory or inhibitory. Consider a network of N leaky integrate-and-fire neurons receiving time-varying inputs $\mathbf{x}(t) = (x_1(t), \dots, x_K(t))$, where K is the dimension of the input. For each neuron i we denote with $s_i(t) = \sum_k \delta(t_k^i - t)$ the spike train function, where δ represents the Dirac delta function and $\{t_k^i \geq 0\}$ is the set of discrete times at which a spike was emitted. The population spike train function is described by the vector $\mathbf{s}(t) = (s_1(t), \dots, s_N(t))^\mathsf{T}$. We define the filtered spike trains (loosely called firing rate) of neuron i as a convolution of the spike train with an exponentially decaying kernel

$$r_i(t) = \int_0^t \exp(-\lambda t') s_i(t - t') dt' = \sum_{t_k^i \leq t} \exp(-\lambda(t - t_k^i))) \tag{11}$$

with leak constant λ, or, equivalently, in the differential form

$$\dot{r_i}(t) = -\lambda r_i(t) + s_i(t).$$

We denote the firing rate for all neurons by the vector $\mathbf{r}(t) = (r_1(t), \dots, r_N(t))^\mathsf{T}$. Vectors will be denoted by bold letters, and wherever possible we will exclude the explicit dependence on time for the sake of text clarity.[2] A neuron i fires a spike whenever its membrane potential, V_i exceeds a spiking threshold, T_i, and is then reset to the value $V_i = R_i$.

Consider a generic smooth function $G : \mathbb{R}^K \to \mathbb{R}^M$, $M \geq 1$. Our goal is to derive dynamics and connectivity of the network so that its output activity provides an accurate representation of the modification of the incoming signal $\mathbf{y} = G(\mathbf{x}) \in \mathbb{R}^M$. Notice that, using the identity function, one can recover the same form considered in Barrett et al., 2016.

Following the assumptions made in the main text, we require the signals to be linearly decodable, so that the readout can be simply written as $\hat{\mathbf{y}} = \mathbf{D}\mathbf{r} \approx G(\mathbf{x})$. The matrix $\mathbf{D} \in \mathbb{R}^{M \times N}$ is called the decoding matrix, and its i-th column vector $\mathbf{D}_i \in \mathbb{R}^M$ is the fixed contribution of neuron i to the signal. The accuracy of the representation is measured using a squared error loss function, $E = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \|G(\mathbf{x}) - \hat{\mathbf{y}}\|_2^2$. The second assumption made in the main text requests

[2] Throughout the text, the input signals, the membrane voltages and the spike trains are all time-dependent quantities, whereas the thresholds, the decay constants, and the connection strengths are all constants.

the network to be efficient, and can be formalized by asking that a neuron fires a spike only if its effect on the readout will reduce the loss function:

$$E(\text{spike}) < E(\text{no spike}),$$

which is, noticing that a spike of neuron i changes the readout by $\hat{\mathbf{y}} \to \hat{\mathbf{y}} + \mathbf{D}_i$,

$$\|G(\mathbf{x}) - (\hat{\mathbf{y}} + \mathbf{D}_i)\|_2^2 < \|G(\mathbf{x}) - \hat{\mathbf{y}}\|_2^2 . \tag{12}$$

After expanding the squares and canceling equal terms we obtain

$$\|\mathbf{D}_i\|_2^2 - 2\mathbf{D}_i^\mathsf{T}(G(\mathbf{x}) - \hat{\mathbf{y}}) < 0, \tag{13}$$

which can be rearranged into

$$\mathbf{D}_i^\mathsf{T}(G(\mathbf{x}) - \hat{\mathbf{y}}) > \frac{\|\mathbf{D}_i\|_2^2}{2}. \tag{14}$$

This equation is crucial: it describes a spiking rule under which the loss function is reduced, and it offers an enticing geometric interpretation of the behavior of the network (Calaim et al., 2020). The right hand side of the equation is fixed, and can be interpreted as the spiking threshold of neuron i:

$$T_i = \frac{\|\mathbf{D}_i\|_2^2}{2}.$$

The left hand side of the equation, similarly to the derivation showed in (Barrett et al., 2016), is used to define the voltage of neuron i

$$V_i = \mathbf{D}_i^\mathsf{T}(G(\mathbf{x}) - \hat{\mathbf{y}}), \tag{15}$$

which, taking the derivative, yields,

$$\begin{aligned} \dot{V}_i &= \mathbf{D}_i^\mathsf{T}\left(\frac{dG(\mathbf{x})}{dt} - \frac{d\hat{\mathbf{y}}}{dt}\right) \\ &= \mathbf{D}_i^\mathsf{T}(\mathbf{J}_G(\mathbf{x})\dot{\mathbf{x}}) + \mathbf{D}_i^\mathsf{T}\lambda\hat{\mathbf{y}} - \sum_k \mathbf{D}_i^\mathsf{T}\mathbf{D}_k s_k, \end{aligned} \tag{16}$$

where we used $\mathbf{J}_G$ to indicate the Jacobian of the function G. Using (15), we have that $\mathbf{D}_i^\mathsf{T}\hat{\mathbf{y}} = -V_i + \mathbf{D}_i^\mathsf{T}(G(\mathbf{x})$, and substituting this into (16) we obtain:

$$\dot{V}_i = -\lambda V_i + \mathbf{D}_i^\mathsf{T}\left(\mathbf{J}_G(\mathbf{x})\dot{\mathbf{x}} + \lambda G(\mathbf{x})\right) - \sum_k \mathbf{D}_i^\mathsf{T}\mathbf{D}_k s_k. \tag{17}$$

This equation describes the dynamic behavior of the voltage of a neuron in a network that represents $G(\mathbf{x})$. We will use the vector form

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}\left(\mathbf{J}_G(\mathbf{x})\dot{\mathbf{x}} + \lambda G(\mathbf{x})\right) + \mathbf{\Omega}_f\mathbf{s}, \tag{18}$$

where $\mathbf{\Omega}_f = -\mathbf{D}^\mathsf{T}\mathbf{D}$ represents the fast connections among units, and also includes the reset terms on the diagonal.

7.2. The Kronecker product.

Throughout the text we make heavy use of the Kronecker product. $\otimes$ represents the Kronecker product, which is defined for any couple of matrices $\mathbf{A}, \mathbf{B}$ of any arbitrary size as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}.$$

We often use the mixed-product property, which states: If $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ are matrices of such size that one can form the matrix products $\mathbf{AC}$ and $\mathbf{BD}$, then

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}).$$

7.3. Representation of the multiplication of incoming inputs.

Consider the function $G : \mathbb{R}^K \to \mathbb{R}^{K^2}$ defined as $G(\mathbf{x}) = \mathbf{x} \otimes \mathbf{x}$, where $\otimes$ is the Kronecker product. Suppose that the input $\mathbf{x} = (x_1, \dots, x_K)$ is given as a linearly decodable input from an upstream network, such that $\mathbf{x} = \mathbf{D}\mathbf{r}$, where $\mathbf{r}$ describes the filtered spike-trains of the upstream neurons and $\mathbf{D}$ is their decoding matrix. This generalization requires us to keep track of $\dot{\mathbf{x}}$: if the upstream neurons follow equation (11), and we denote with $\mathbf{s}$ their spike trains, we will have that $\dot{\mathbf{x}} = \mathbf{D}\dot{\mathbf{r}} = \mathbf{D}(\mathbf{s} - \lambda\mathbf{r})$, where λ represents their leak constant. In order to use eq. (18), we need to compute the Jacobian of the function G. That's given by the $K^2 \times K$ matrix $\mathbf{J}_G(\mathbf{x})$, with column i given by $\frac{dG}{dx_i}$. Denote with $\mathbf{D}^i$ the $i-$th row of the matrix $\mathbf{D}$, and with $[\mathbf{J}_G(\mathbf{x})\dot{\mathbf{x}}]_{iK+j}$ the $(iK+j)-$th entry of the matrix-vector product $\mathbf{J}_G(\mathbf{x})\dot{\mathbf{x}} \in \mathbb{R}^{K^2}$, for $0 < i \leq j \leq K$. We have:

$$\begin{aligned}
[\mathbf{J}_G(\mathbf{x})\dot{\mathbf{x}}]_{iK+j} &= \frac{dG}{dx_i}\frac{dx_i}{dt} + \frac{dG}{dx_j}\frac{dx_j}{dt} \\
&= x_j\dot{x}_i + x_i\dot{x}_j \\
&= (\mathbf{D}^j\mathbf{r})(\mathbf{D}^i(\mathbf{s} - \lambda\mathbf{r})) + (\mathbf{D}^i\mathbf{r})(\mathbf{D}^j(\mathbf{s} - \lambda\mathbf{r}) \\
&= (\mathbf{D}^j \otimes \mathbf{D}^i + \mathbf{D}^i \otimes \mathbf{D}^j)(\mathbf{r} \otimes (\mathbf{s} - \lambda\mathbf{r})) \\
&= (\mathbf{D}^i \otimes \mathbf{D}^j)(\mathbf{r} \otimes \mathbf{s} + \mathbf{s} \otimes \mathbf{r} - 2\lambda\mathbf{r} \otimes \mathbf{r})
\end{aligned}$$

and

$$[G(x)]_{iK+j} = (\mathbf{D}^i\mathbf{r})(\mathbf{D}^j\mathbf{r}) = (\mathbf{D}^i \otimes \mathbf{D}^j)(\mathbf{r} \otimes \mathbf{r}).$$

We can now derive the voltage equations of a network of neurons that represents the product of any pair of input dimensions using the equation derived in the previous section. Denote with $\boldsymbol{\sigma}$ the spike train of the network, and with $\boldsymbol{\rho}$ their filtered spike train with leak constant α. Using eq. (18) we have

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \boldsymbol{\Omega}_x(\mathbf{r} \otimes \mathbf{s} + \mathbf{s} \otimes \mathbf{r} + (\alpha - 2\lambda)\mathbf{r} \otimes \mathbf{r}) + \boldsymbol{\Omega}_f^W\boldsymbol{\sigma},$$

with $\boldsymbol{\Omega}_x = \mathbf{W}^\mathsf{T}(\mathbf{D} \otimes \mathbf{D})$, $\boldsymbol{\Omega}_f^W = -\mathbf{W}^\mathsf{T}\mathbf{W}$ and $\mathbf{W}$ being their decoding matrix. An example of the output of such a network can be seen in Supp. Fig. S3. In that case the input was $3-$dimensional, and the $9-$dimensional output faithfully represented the product of each input dimension pair.

7.4. Implementing dynamical systems in spike coding networks.

By using the identity function $G(\mathbf{x}) = \mathbf{x}$ in (18) we obtain the "classical" equation

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}(\dot{\mathbf{x}} + \lambda\mathbf{x}) + \boldsymbol{\Omega}_f\mathbf{s}. \tag{19}$$

This will be the starting point to implement linear and nonlinear dynamical systems. Linear dynamical systems were already considered in (Boerlin et al., 2013). Here we will focus on a more general class of nonlinearities, namely polynomial nonlinearities, and show that the original formulation can be analytically extended to implement any polynomial nonlinearity.

Denote with $F : \mathbb{R}^K \to \mathbb{R}^K$ the dynamic under study, so that $\dot{\mathbf{x}} = F(\mathbf{x})$. Starting from (19) and knowing that $\mathbf{x} \approx \hat{\mathbf{x}}$ we can consider the following approximation:

$$\dot{\mathbf{v}} = -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}(F(\hat{\mathbf{x}}) + \lambda\hat{\mathbf{x}}) + \boldsymbol{\Omega}_f\mathbf{s}. \tag{20}$$

If F is a linear dynamic of the form $F(\mathbf{x}) = \mathbf{A}\mathbf{x}$, with the matrix $\mathbf{A} \in \mathbb{R}^{K\times K}$, we recover the same form considered in (Boerlin et al., 2013):

$$\begin{aligned}
\dot{\mathbf{v}} &= -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}\left(\mathbf{A}\hat{\mathbf{x}} + \lambda\hat{\mathbf{x}}\right) + \boldsymbol{\Omega}_f\mathbf{s} \\
&= -\lambda\mathbf{v} + \mathbf{D}^\mathsf{T}\left(\mathbf{A}\mathbf{D}\mathbf{r} + \lambda D\mathbf{r}\right) + \boldsymbol{\Omega}_f\mathbf{s} \\
&= -\lambda\mathbf{v} + \boldsymbol{\Omega}_s\mathbf{r} + \boldsymbol{\Omega}_f\mathbf{s},
\end{aligned} \tag{21}$$

where $\boldsymbol{\Omega}_f = -\mathbf{D}^\mathsf{T}\mathbf{D}$ and $\boldsymbol{\Omega}_s = \mathbf{D}^\mathsf{T}(\mathbf{A} + \lambda\mathbf{I})\mathbf{D}$ represent the fast and slow connections respectively.

If F is polynomial, we proceed as follows. Using Kronecker notation, any polynomial $F : R^K \to R^K$ with maximum degree g can be written in the form

$$F(\mathbf{x}) = \sum_{d=0}^{g} \mathbf{A}_d \mathbf{x}^{\otimes d}, \tag{22}$$

where $\mathbf{A}_d \in \mathbb{R}^{K \times K^d}$ is the matrix of coefficients for the polynomials of degree d, and we define $\mathbf{M}^{\otimes d} = \mathbf{M} \otimes \mathbf{M} \otimes \cdots \otimes \mathbf{M}$ as the Kronecker product applied d times, with the convention that $\mathbf{M}^{\otimes 0} = 1$ and $\mathbf{M}^{\otimes 1} = \mathbf{M}$. Once again replacing $\mathbf{x}$ by $\hat{\mathbf{x}}$, as well as using the notation introduced in (22) and the mixed-product property, we get

$$\begin{aligned} F(\hat{\mathbf{x}}) &= \sum_{d=0}^{g} \mathbf{A}_d \hat{\mathbf{x}}^{\otimes d} \\ &= \mathbf{A}_0 + \mathbf{A}_1 \hat{\mathbf{x}} + \mathbf{A}_2 \hat{\mathbf{x}}^{\otimes 2} + \mathbf{A}_3 \hat{\mathbf{x}}^{\otimes 3} + \dots \\ &= \mathbf{A}_0 + \mathbf{A}_1 \mathbf{D}\mathbf{r} + \mathbf{A}_2 (\mathbf{D}\mathbf{r})^{\otimes 2} + \mathbf{A}_3 (\mathbf{D}\mathbf{r})^{\otimes 3} + \dots \\ &= \mathbf{A}_0 + \mathbf{A}_1 \mathbf{D}\mathbf{r} + \mathbf{A}_2 (\mathbf{D}^{\otimes 2})(\mathbf{r}^{\otimes 2}) + \mathbf{A}_3 (\mathbf{D}^{\otimes 3})(\mathbf{r}^{\otimes 3}) + \dots \\ &= \sum_{d=0}^{g} \mathbf{A}_d \mathbf{D}^{\otimes d} \mathbf{r}^{\otimes d}. \end{aligned}$$

Inserting it into (20) one obtains the equations describing a network of integrate-and-fire neurons that approximate the solution of a polynomial dynamical system:

$$\begin{aligned} \dot{\mathbf{v}} &= -\lambda \mathbf{v} - \mathbf{D}^{\mathsf{T}} \mathbf{D} \mathbf{s} + \mathbf{D}^{\mathsf{T}} \Big(\sum_{d=0}^{g} \mathbf{A}_d \mathbf{D}^{\otimes d} \mathbf{r}^{\otimes d} + \lambda \hat{\mathbf{x}} \Big) \\ &= -\lambda \mathbf{v} + \mathbf{\Omega}_f \mathbf{s} + \mathbf{\Omega}_s^{m0} + \mathbf{\Omega}_s^{m1} \mathbf{r} + \mathbf{\Omega}_s^{m2} \mathbf{r}^{\otimes 2} + \cdots + \mathbf{\Omega}_s^{mg} \mathbf{r}^{\otimes d} \\ &= -\lambda \mathbf{v} + \mathbf{\Omega}_f \mathbf{s} + \sum_{d=0}^{g} \mathbf{\Omega}_s^{md} \mathbf{r}^{\otimes d}, \end{aligned} \tag{23}$$

where $\mathbf{\Omega}_f = -\mathbf{D}^{\mathsf{T}} \mathbf{D}$, $\mathbf{\Omega}_s^{m1} = \mathbf{D}^{\mathsf{T}} (\mathbf{A}_1 + \lambda \mathbf{I}) \mathbf{D}$ and $\mathbf{\Omega}_s^{md} = \mathbf{D}^{\mathsf{T}} \mathbf{A}_d \mathbf{D}^{\otimes d}$ for $d \in \{0, 2, 3, \dots, g\}$.

7.5. Implementing the Lorenz system.

Denoting $\mathbf{x} = (x, y, z)^{\mathsf{T}}$, the Lorenz attractor can be described in the form of eq. 22 as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{x}^{\otimes 2},$$

where

$$\mathbf{A} = \begin{bmatrix} -\sigma & \sigma & 0 \\ \rho & -1 & 0 \\ 0 & 0 & -\beta \end{bmatrix}, \tag{24}$$

and $\mathbf{B} \in \mathbb{R}^{3 \times 9}$ with $B_{23} = -1$, $B_{32} = 1$, and all other elements of $\mathbf{B}$ being zero.

Following eq. 23 the corresponding voltage dynamics in an mSCN are described by

$$\dot{\mathbf{v}} = -\lambda \mathbf{v} + \mathbf{\Omega}_f \mathbf{s} + \mathbf{\Omega}_s^{m1} \mathbf{r} + \mathbf{\Omega}_s^{m2} \mathbf{r}^{\otimes 2},$$

where $\mathbf{\Omega}_f = -\mathbf{D}^{\mathsf{T}} \mathbf{D}$, $\mathbf{\Omega}_s^{m1} = \mathbf{D}^{\mathsf{T}} (\mathbf{A} + \lambda \mathbf{I}) \mathbf{D}$ and $\mathbf{\Omega}_s^{m2} = \mathbf{D}^{\mathsf{T}} \mathbf{B} \mathbf{D}^{\otimes 2}$.

7.6. Learning nonlinear dynamics through basis functions.

In previous work the standard SCN derivation was extended to implement arbitrary nonlinear dynamical systems through weighted basis functions, meant to model nonlinear synapses or dendrites (Alemi et al., 2018; Thalmeier et al., 2016). We will use a similar approach to approximate the nonlinear part of a dynamical system of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + F(\mathbf{x}). \tag{25}$$

The basis-function approach derivation consists replacing the function $F(\mathbf{x})$ by a weighted set of L basis functions $\mathbf{g}(\mathbf{x}) = [g_0(\mathbf{x}), \ldots, g_L(\mathbf{x})]$, such that $\mathbf{Cg} \approx F(\mathbf{x})$ (where $\mathbf{C} \in \mathbb{R}^{K \times L}$ are the required weights). Eq. (20) can then be rewritten as

$$\dot{\mathbf{v}} = -\lambda \mathbf{v} + \mathbf{\Omega}_s \mathbf{r} + \mathbf{D}^{\mathsf{T}} \mathbf{C} \mathbf{g}(\mathbf{x}) - \mathbf{\Omega}_f \mathbf{s}. \tag{26}$$

In previous work the weights $\mathbf{C}$ were found through supervised local learning rules. For brevity and comparison's sake we will instead find the optimal weights through regression (following (Eliasmith and Anderson, 2004)).

We can find the weights by solving the following optimization problem

$$\mathtt{min}_{\mathbf{C}} ||F(\mathbf{X}) - \mathbf{CG}||_2^2, \tag{27}$$

where $\mathbf{X} \in \mathbb{R}^{K \times M}$ are M sampled inputs, $\mathbf{G} \in \mathbb{R}^{L \times M}$ are the resulting basis function outputs, and $F()$ is the target function. The ordinary least squares (OLS) solution is then

$$\mathbf{C}_{\mathtt{OLS}} = F(\mathbf{X})\mathbf{G}^{\mathsf{T}}(F(\mathbf{X})F(\mathbf{X})^{\mathsf{T}})^{-1}. \tag{28}$$

In previous work online learning rules were used to minimize the cost (Alemi et al., 2018; Thalmeier et al., 2016), but as learning rules are not the focus of this paper, we used the above solution. For the basis functions we used a simple rectification function ($g(x) = [bx + c]^+$, with randomly distributed $b \in [-1, 1]$ and $c \in [-90, 90]$), but many types of nonlinearities will work.

7.7. First order approximation of the double pendulum.

The equations describing the time evolution of the double pendulum with each length l and mass m can be derived using the Lagrangian (Levien and Tan, 1993). θ_1, θ_2 describe the angles of the first and second pendulum with respect to the vertical axis (i.e. $\theta_i = 0$ when the pendulum is pointing downwards). The position of the centers of mass can be written thanks to these two coordinates: assuming that the origin is at the point of suspension of the first pendulum, its center of mass will be at:

$$x_1 = \frac{l}{2}\sin\theta_1, \;\; y_1 = -\frac{l}{2}\cos\theta_1$$

and the center of mass of the second pendulum is at

$$x_2 = l\left(\sin\theta_1 + \tfrac{1}{2}\sin\theta_2\right), y_2 = -l\left(\cos\theta_1 + \tfrac{1}{2}\cos\theta_2\right).$$

The full dynamics can be described by a $4-$dimensional dynamical system representing the two angles and the two moments:

$$\dot{\theta}_1 = \frac{6}{ml^2}\frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)}$$
$$\dot{\theta}_2 = \frac{6}{ml^2}\frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}$$

$$\dot{p}_{\theta_1} = -\tfrac{1}{2}ml^2\left(\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + 3\frac{g}{l}\sin\theta_1\right)$$
$$\dot{p}_{\theta_2} = -\tfrac{1}{2}ml^2\left(-\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + \frac{g}{l}\sin\theta_2\right).$$

We will use a small angle approximation of the above equations: if $\theta \approx 0$, the functions $\sin, \cos$ are well approximated by $\theta, 1$ respectively. The introduction of this simplifying assumption turned

the above equations into these:

$$
\begin{aligned}
\dot{\theta}_1 &= \frac{6}{7ml^2}\left(2p_{\theta_1} - 3p_{\theta_2}\right) \\
\dot{\theta}_2 &= \frac{6}{7ml^2}\left(8p_{\theta_2} - 3p_{\theta_1}\right) \\
\dot{p}_{\theta_1} &= -\tfrac{1}{2}ml^2\left(\dot{\theta}_1\dot{\theta}_2(\theta_1-\theta_2) + 3\frac{g}{l}\theta_1\right) \\
\dot{p}_{\theta_2} &= -\tfrac{1}{2}ml^2\left(-\dot{\theta}_1\dot{\theta}_2(\theta_1-\theta_2) + \frac{g}{l}\theta_2\right).
\end{aligned}
\tag{29}
$$

These can be implemented using either equation (23) or (7) by considering $\mathbf{x} = (\theta_1, \theta_2, p_{\theta_1}, p_{\theta_2})$ and rewriting the dynamical system as $\dot{\mathbf{x}} = A\mathbf{x} + C\mathbf{x}^{\otimes 3}$, where

$$
\mathbf{A} = \begin{bmatrix} 0 & 0 & 2k & -3k \\ 0 & 0 & -3k & 8k \\ 3cg/l & 0 & 0 & 0 \\ 0 & cg/l & 0 & 0 \end{bmatrix},
\tag{30}
$$

and $\mathbf{C} \in \mathbb{R}^{4\times 64}$ with $C_{3,41} = -6ck^2$, $C_{3,42} = 6ck^2$,$C_{3,45} = 25ck^2$,$C_{3,46} = -25ck^2$,$C_{3,61} = -24ck^2$,$C_{3,62} = 24ck^2$, $\mathbf{C}_4 = -\mathbf{C}_3$ and all the other entries set to zero, with $k = 6/(7ml^2)$ and $c = -1/2ml^2$.

7.8. Connectivity density.

Here we discuss the expected amount of connections based on the sparsity of the decoding matrix $\mathbf{D}$ of a network implementing a generic dynamical system $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{y}\otimes\mathbf{y}$.

7.8.1. Fast connections $\mathbf{\Omega}_f$. For the fast connections, the connectivity matrix is given by $\mathbf{D}^\mathsf{T}\mathbf{D}$. For any pair of neurons m, n we will have that a (fast) connection exists if $\mathbf{D}_m^\mathsf{T}\mathbf{D}_n \neq 0$, which means that if these two neurons "share a dimension" (i.e. $\mathbf{D}_m, \mathbf{D}_n$ have nonzero entries in at least one common spot and they are not orthogonal) they will need a fast connection among them. Let's denote with $0 \leq p_d^n \leq 1$ the probability that a neuron n will participate in the representation of the $d-$th dimension (i.e. $p_d^n = P(\mathbf{D}_n^d \neq 0)$). Let's assume that they are all independent. Then the probability that any given pair of neurons n, m will need a connection is given by the probability that they both end up coding for at least one common dimension, given by

$$
p(\texttt{neurons } n, m \texttt{ code for common dimension}) = 1 - \prod_{d=1}^{K}(1 - p_d^n p_d^m).
$$

In the case where $p_d^n = p$ (such that neurons code for each possible dimension with equal probability) we can compute the expected number of fast connections for different neuron numbers and decoding densities as

$$
\mathbb{E}(\#\texttt{fast connections}) = \frac{N(N-1)}{2}\left(1 - (1-p^2)^K\right),
\tag{31}
$$

7.8.2. Slow connections $\mathbf{\Omega}_s$. Slow connections have the form $\mathbf{\Omega}_s = \mathbf{D}^\mathsf{T}(\mathbf{A} + \lambda\mathbf{I})\mathbf{D} = \mathbf{D}^\mathsf{T}\mathbf{A}\mathbf{D} + \lambda\mathbf{D}^\mathsf{T}\mathbf{D}$. The second term, $\lambda\mathbf{D}^\mathsf{T}\mathbf{D}$, has exactly the form of the already considered case of fast connections. We focus on the first term $\mathbf{D}^\mathsf{T}\mathbf{A}\mathbf{D}$, which will add further connections to allow the network to solve linear dynamical systems. Since $\mathbf{A}$ is not symmetric in general, $\mathbf{\Omega}_s$ can be non symmetric too, hence the total possible number of slow connections is N^2, and will be so when the decoding matrix $\mathbf{D}$ is not sparse. If the matrix $\mathbf{A}$ has a non-zero entry at a location d, e, all the neurons that code for dimension d will have to connect to all the neurons that code for dimension e. The probability that two neurons n, m will form a slow connection will be $p_d^n p_e^m$, or simply p^2 if the probability is uniform across dimensions and neurons. The expected number of slow connections (due to that single non-zero entry) is $\sum_{n=1}^{N}\sum_{m=1}^{N} p_d^n p_e^m = (Np)^2$, where the last equality holds only in case of uniform probability. In that case we also have

(32) $$\mathbb{E}(\#\texttt{slow connections}) \leq \mathbb{E}(\#\texttt{fast connections}) + N_A(Np)^2,$$

where N_A is the number of nonzero entries in $\mathbf{A}$.

7.8.3. Quadratic connections $\mathbf{\Omega}_{nl}$. The quadratic connections take the form $\mathbf{\Omega}_{nl} = \mathbf{D}^\mathsf{T}\mathbf{B}(\mathbf{D}\otimes\mathbf{D})$. If the decoding matrix is not sparse, the number of quadratic connections will be $\propto N^3$. In fact, the maximum possible number is given by $N^2(N-1)/2$, corresponding to each neuron (N) being connected to each possible pair ($\frac{N(N-1}{2}$). On the other hand, if $\mathbf{D}$ is sufficiently sparse, we can reason as follows. Denote with G_d the group of neurons that code for dimension d, i.e. $G_d = \{n \mid \mathbf{D}_n^d \neq 0\}$. Let's assume that our dynamical system depends nonlinearly on dimensions e and f, i.e. $\dot{x}_d \propto x_e x_f$, or equivalently $\mathbf{B}_{d,eK+f} \neq 0$. Then, each neuron in G_d needs to keep track of coincident firing of any neuron in G_e with any other in G_f. The probability that a neuron in G_d will need to take care of coincident spiking of the pair of neurons m, n is $1-(1-p_e^n p_f^m)(1-p_e^m p_f^n)$, corresponding to the probability that at least one of the two neurons codes for dimension e and the other for dimension f. In the case of uniform p this reduces to $2p^2 - p^4$, so each neuron in G_d will need an average of $\frac{N(N-1)}{2}(2p^2 - p^4)$ coincidence detectors, leading to an upper bound for the expected total number of multiplicative synapses

(33) $$\mathbb{E}(\#\texttt{multiplicative connections}) \leq N_B n_d \frac{N(N-1)}{2}(2p^2 - p^4) \approx N_B(Np)^3,$$

where $n_d = \#G_d \approx Np$ and N_B is the number of nonzero entries in $\mathbf{B}$. The equality sign holds only in the case $N_B \leq 1$.

7.8.4. Simulations. In order to simulate the connectivity we fixed a decoder density p and randomly filled the decoding matrix using a Bernoulli distribution $B(p)$ in each entry for 1000 times. For the fast connections we varied the size of the output signal - i.e. the size of the decoding matrix. For slow and multiplicative synapses the dimensionality of the signal K did not affect the density of the resulting connections (not shown). What influenced the amount of slow and multiplicative synapses was the number of non-zero entries in the matrices $\mathbf{A}$ and $\mathbf{B}$, respectively.

7.9. Code details.

Simulations were run in Ubuntu 20.04LTS on a Intel Core i5-6200U CPU with 32GB of RAM. The source code is available at `https://github.com/michnard/mult_synapses`.

Acknowledgements

A preprint version of this article has been peer-reviewed and recommended by *Peer Community In Neuroscience* (DOI link to the recommendation: https://doi.org/10.24072/pci.cneuro.100003). We thank Christian Machens and Nuno Calaim for useful discussions on the project. This report came out of a collaboration started at the CAJAL Advanced Neuroscience Training Programme in Computational Neuroscience in Lisbon, Portugal, during the 2019 summer. The authors would like to thank the participants, TAs, lecturers, and organizers of the summer school. SWK was supported by the Simons Collaboration on the Global Brain (543009). WFP was supported by FCT (032077). MN was supported by European Union Horizon 2020 (665385).

Conflict of interest disclosure

The authors of this work declare that they have no financial conflict of interest with the content of this article.

References

Abbott LF et al. (2016). *Building functional networks of spiking model neurons*. *Nature neuroscience* **19**, 350–355. https://doi.org/10.1038/nn.4241.

Alemi A et al. (2018). *Learning nonlinear dynamics in efficient, balanced spiking networks using local plasticity rules*. In: *Thirty-second aaai conference on artificial intelligence*. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11320.

Arandia-Romero I et al. (2016). *Multiplicative and additive modulation of neuronal tuning with population activity affects encoded information*. *Neuron* **89**, 1305–1316. https://doi.org/10.1016/j.neuron.2016.01.044.

Augustine GJ et al. (2003). *Local calcium signaling in neurons*. *Neuron* **40**, 331–346. https://doi.org/10.1016/s0896-6273(03)00639-1.

Baker C et al. (2020). *Nonlinear stimulus representations in neural circuits with approximate excitatory-inhibitory balance*. *PLoS computational biology* **16**, e1008192. https://doi.org/10.1371/journal.pcbi.1008192.

Barak O (2017). *Recurrent neural networks as versatile tools of neuroscience research*. *Current opinion in neurobiology* **46**, 1–6. https://doi.org/10.1016/j.conb.2017.06.003.

Barrett DG et al. (2016). *Optimal compensation for neuron loss*. *Elife* **5**, e12454. https://doi.org/10.7554/eLife.12454.

Barth AL, Poulet JFA (2012). *Experimental evidence for sparse firing in the neocortex*. *Trends in Neurosciences* **35**, 345–355. https://doi.org/10.1016/j.tins.2012.03.008.

Boerlin M et al. (2013). *Predictive coding of dynamical variables in balanced spiking networks*. *PLoS Comput Biol* **9**, e1003258. https://doi.org/10.1371/journal.pcbi.1003258.

Calaim N et al. (2020). *Robust coding with spiking networks: a geometric perspective*. *bioRxiv*. https://doi.org/10.1101/2020.06.15.148338.

Cunningham JP, Yu BM (2014). *Dimensionality reduction for large-scale neural recordings*. *Nature neuroscience* **17**, 1500–1509. https://doi.org/10.1038/nn.3776.

Davies M et al. (2018). *Loihi: A neuromorphic manycore processor with on-chip learning*. *IEEE Micro* **38**, 82–99. https://doi.org/10.1109/MM.2018.112130359.

Dayan P, Abbott LF (2001). *Theoretical neuroscience: computational and mathematical modeling of neural systems*. Computational Neuroscience Series. URL: https://mitpress.mit.edu/books/theoretical-neuroscience.

De Branges L (1959). *The stone-weierstrass theorem*. *Proceedings of the American Mathematical Society* **10**, 822–824. https://doi.org/10.1090/S0002-9939-1959-0113131-7.

Denève S, Machens CK (2016). *Efficient codes and balanced networks*. *Nature neuroscience* **19**, 375–382. https://doi.org/10.1038/nn.4243.

Eliasmith C (June 2005). *A Unified Approach to Building and Controlling Spiking Attractor Networks*. *Neural Computation* **17**, 1276–1314. ISSN: 0899-7667. https://doi.org/10.1162/0899766053630332.

Eliasmith C, Anderson CH (2004). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press. URL: https://mitpress.mit.edu/books/neural-engineering.

Eliasmith C, Stewart TC, et al. (2012). *A Large-Scale Model of the Functioning Brain*. *Science* **338**, 1202–1205. https://doi.org/10.1126/science.1225266.

Fino E, Yuste R (2011). *Dense inhibitory connectivity in neocortex*. *Neuron* **69**, 1188–1203. https://doi.org/10.1016/j.neuron.2011.02.025.

Gabbiani F et al. (Nov. 2002). *Multiplicative computation in a visual neuron sensitive to looming*. *Nature* **420**, 320–324. ISSN: 1476-4687. https://doi.org/10.1038/nature01190.

Harris KD, Mrsic-Flogel TD (2013). *Cortical connectivity and sensory coding*. *Nature* **503**, 51–58. https://doi.org/10.1038/nature12654.

Indiveri G et al. (2011). *Neuromorphic silicon neuron circuits*. *Frontiers in neuroscience* **5**, 73. https://doi.org/10.3389/fnins.2011.00073.

Jaeger H (2001). *The "echo state" approach to analysing and training recurrent neural networks-with an erratum note*. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* **148**, 13. URL: https://www.ai.rug.nl/minds/uploads/EchoStatesTechRep.pdf.

Kasthuri N et al. (2015). *Saturated reconstruction of a volume of neocortex*. *Cell* **162**, 648–661. https://doi.org/10.1016/j.cell.2015.06.054.

Keemink SW, Machens CK (2019). *Decoding and encoding (de) mixed population responses*. *Current Opinion in Neurobiology* **58**, 112–121. https://doi.org/10.1016/j.conb.2019.09.004.

Ko H et al. (2011). *Functional specificity of local synaptic connections in neocortical networks*. *Nature* **473**, 87–91. https://doi.org/10.1038/nature09880.

Koch C, Poggio T (1992). *Multiplying with Synapses and Neurons*. In: *Single Neuron Computation*. Elsevier, pp. 315–345. ISBN: 978-0-12-484815-3. https://doi.org/10.1016/B978-0-12-484815-3.50019-0.

Larkum ME, Nevian T (2008). *Synaptic clustering by dendritic signalling mechanisms*. *Current opinion in neurobiology* **18**, 321–331. https://doi.org/10.1016/j.conb.2008.08.013.

Lefort S et al. (2009). *The excitatory neuronal network of the C2 barrel column in mouse primary somatosensory cortex*. *Neuron* **61**, 301–316. https://doi.org/10.1016/j.neuron.2008.12.020.

Levien R, Tan S (1993). *Double pendulum: An experiment in chaos*. *American Journal of Physics* **61**, 1038–1044. https://doi.org/10.1119/1.17335.

Li N et al. (2016). *Robust neuronal dynamics in premotor cortex during motor planning*. *Nature* **532**, 459–464. https://doi.org/10.1038/nature17643. (Visited on 03/11/2021).

London M, Häusser M (2005). *Dendritic computation*. *Annu. Rev. Neurosci.* **28**, 503–532. https://doi.org/10.1146/annurev.neuro.28.061604.135703.

Lorenz EN (1963). *Deterministic nonperiodic flow*. *Journal of the atmospheric sciences* **20**, 130–141. https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

Maass W et al. (2002). *Real-time computing without stable states: A new framework for neural computation based on perturbations*. *Neural computation* **14**, 2531–2560. https://doi.org/10.1162/089976602760407955.

Mancoo A et al. (2020). *Understanding spiking networks through convex optimization*. *Advances in Neural Information Processing Systems* **33**. URL: https://proceedings.neurips.cc/paper/2020/file/64714a86909d401f8feb83e8c2d94b23-Paper.pdf.

Mante V et al. (2013). *Context-dependent computation by recurrent dynamics in prefrontal cortex*. *nature* **503**, 78–84. https://doi.org/10.1038/nature12742.

Mastrogiuseppe F, Ostojic S (2018). *Linking connectivity, dynamics, and computations in low-rank recurrent neural networks*. *Neuron* **99**, 609–623. https://doi.org/10.1016/j.neuron.2018.07.003.

Merolla PA et al. (2014). *A million spiking-neuron integrated circuit with a scalable communication network and interface*. *Science* **345**, 668–673. https://doi.org/10.1126/science.1254642.

Mitchell SJ, Silver RA (2003). *Shunting inhibition modulates neuronal gain during synaptic excitation*. *Neuron* **38**, 433–445. https://doi.org/10.1016/s0896-6273(03)00200-9.

Neftci EO et al. (2019). *Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks*. *IEEE Signal Processing Magazine* **36**, 51–63. https://doi.org/10.1109/MSP.2019.2931595.

Nezis P, Rossum MCWv (May 2011). *Accurate multiplication with noisy spiking neurons*. *Journal of Neural Engineering* **8**, 034005. ISSN: 1741-2552. https://doi.org/10.1088/1741-2560/8/3/034005.

Peña JL, Konishi M (Apr. 2001). *Auditory Spatial Receptive Fields Created by Multiplication*. *Science* **292**, 249–252. ISSN: 0036-8075, 1095-9203. https://doi.org/10.1126/science.1059201.

Perin R et al. (2011). *A synaptic organizing principle for cortical neuronal groups*. *Proceedings of the National Academy of Sciences* **108**, 5419–5424. https://doi.org/10.1073/pnas.1016051108.

Poirazi P et al. (2003). *Pyramidal neuron as two-layer neural network*. *Neuron* **37**, 989–999. https://doi.org/10.1016/s0896-6273(03)00149-1.

Popov VI, Stewart MG (2009). *Complexity of contacts between synaptic boutons and dendritic spines in adult rat hippocampus: Three-dimensional reconstructions from serial ultrathin sections in vivo*. *Synapse* **63**, 369–377. https://doi.org/10.1002/syn.20613.

Rigotti M et al. (2013). *The importance of mixed selectivity in complex cognitive tasks*. *Nature* **497**, 585–590. https://doi.org/10.1038/nature12160.

Rubin DB et al. (2015). *The stabilized supralinear network: a unifying circuit motif underlying multi-input integration in sensory cortex*. *Neuron* **85**, 402–417. https://doi.org/10.1016/j.neuron.2014.12.026.

Salinas E, Abbott LF (1996). *A model of multiplicative neural responses in parietal cortex*. *Proceedings of the national academy of sciences* **93**, 11956–11961. https://doi.org/10.1073/pnas.93.21.11956.

Schiller J et al. (2000). *NMDA spikes in basal dendrites of cortical pyramidal neurons*. *Nature* **404**, 285–289. https://doi.org/10.1038/35005094.

Semedo JD et al. (2019). *Cortical areas interact through a communication subspace*. *Neuron* **102**, 249–259. https://doi.org/10.1016/j.neuron.2019.01.026.

Song S et al. (2005). *Highly nonrandom features of synaptic connectivity in local cortical circuits*. *PLoS Biol* **3**, e68. https://doi.org/10.1371/journal.pbio.0030068.

Stringer C, Pachitariu M, Steinmetz N, Carandini M, et al. (2019). *High-dimensional geometry of population responses in visual cortex*. *Nature* **571**, 361–365. https://doi.org/10.1038/s41586-019-1346-5.

Stringer C, Pachitariu M, Steinmetz N, Reddy CB, et al. (2019). *Spontaneous behaviors drive multidimensional, brainwide activity*. *Science* **364**. https://doi.org/10.1126/science.aav7893.

Strogatz SH (2018). *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press. URL: https://www.routledge.com/Nonlinear-Dynamics-and-Chaos-with-Student-Solutions-Manual-With-Applications/Strogatz/p/book/9780813350844.

Sussillo D (2014). *Neural circuits as computational dynamical systems*. *Current opinion in neurobiology* **25**, 156–163. https://doi.org/10.1016/j.conb.2014.01.008.

Sussillo D, Abbott LF (2009). *Generating coherent patterns of activity from chaotic neural networks*. *Neuron* **63**, 544–557. https://doi.org/10.1016/j.neuron.2009.07.018.

Thalmeier D et al. (2016). *Learning universal computations with spikes*. *PLoS computational biology* **12**, e1004895. https://doi.org/10.1371/journal.pcbi.1004895.

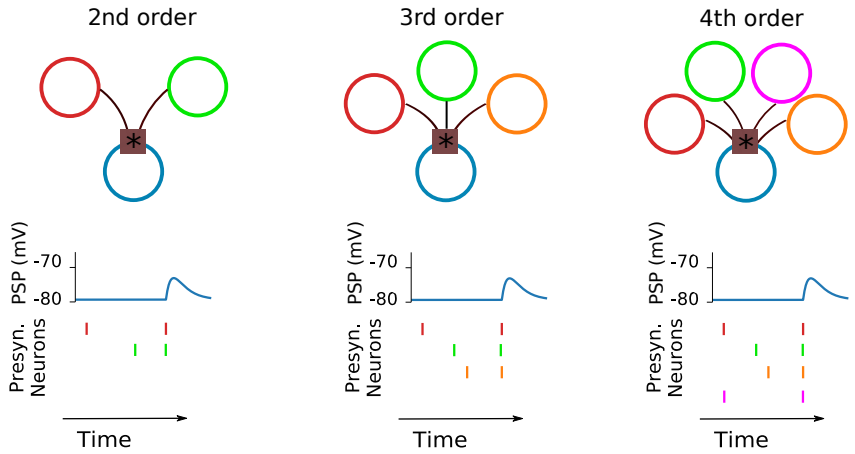
Young AR et al. (2019). *A Review of Spiking Neuromorphic Hardware Communication Systems*. *IEEE Access* **7**, 135606–135620. https://doi.org/10.1109/ACCESS.2019.2941772.

Zenke F, Vogels TP (2021). *The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks*. *Neural Computation* **33**, 899–925. https://doi.org/10.1162/neco_a_01367.

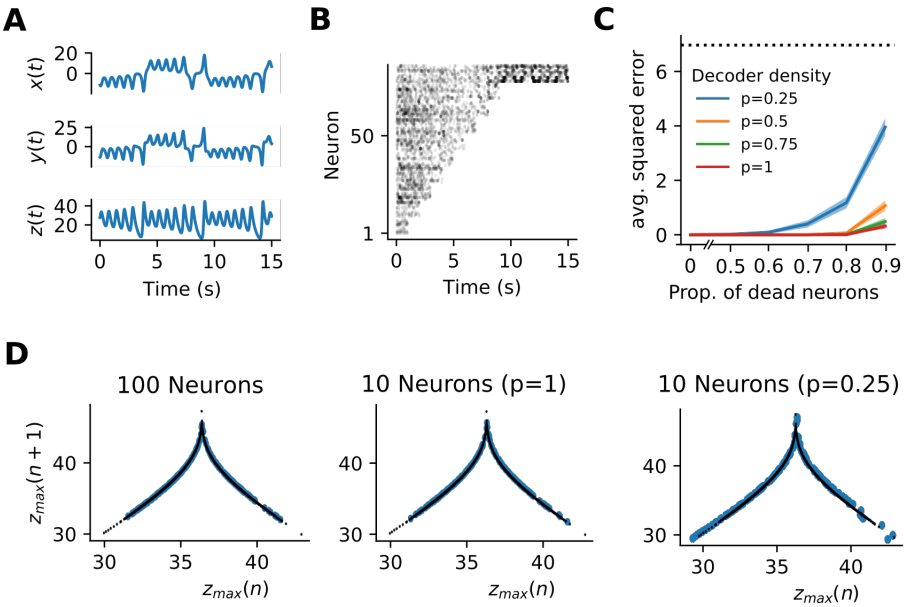
Zhang D et al. (2013). *Nonlinear multiplicative dendritic integration in neuron and network models*. *Frontiers in computational neuroscience* **7**, 56. https://doi.org/10.3389/fncom.2013.00056.

Zhou W et al. (Apr. 2007). *Multiplicative Computation in the Vestibulo-Ocular Reflex (VOR)*. *Journal of Neurophysiology* **97**, 2780–2789. ISSN: 0022-3077. https://doi.org/10.1152/jn.00812.2006.

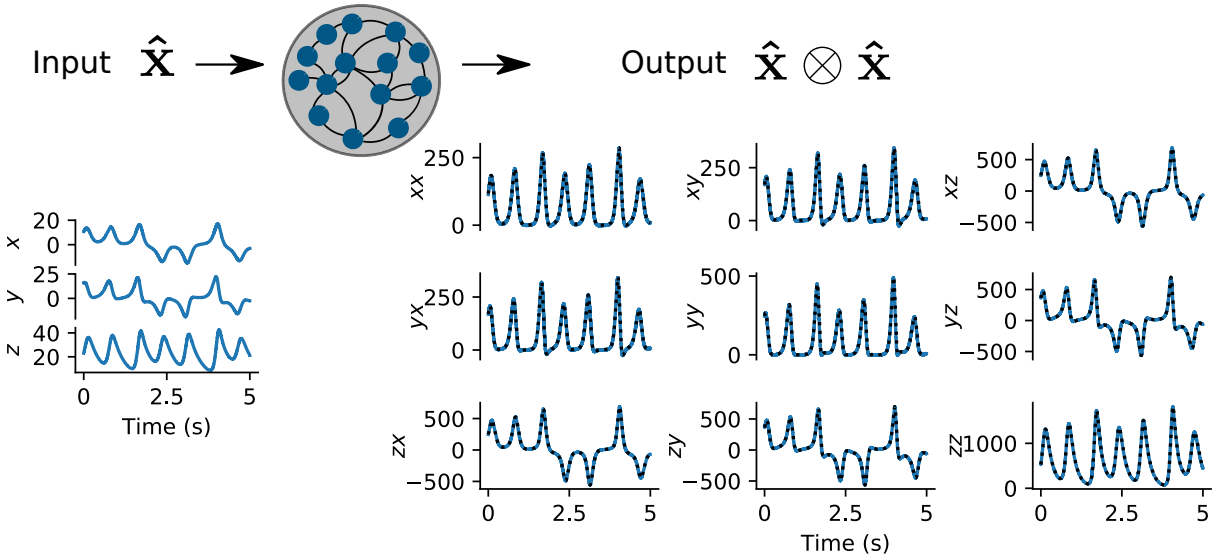
Supplementary Figures



Supp. Fig. S1 – *Higher order multiplicative interactions among cells.* Second (resp. third, fourth) order interactions require the post-synaptic cell to detect coincident activity in two (resp. three, four) pre-synaptic cells.



Supp. Fig. S2 – *Robustness of mSCNs.* **(A)** Example readout of a network of 100 neurons implementing a Lorenz dynamical system. For the first 9 seconds, 10 neurons were artificially killed every second. **(B)** Spike raster plot of the 100 neurons as a function of time. **(C)** Average squared error as a function of the proportion of neurons lost (out of 100 total) for different choices of decoder density. Shaded areas represent 95th confidence intervals. The error was computed by measuring the average squared distance of the network readout from the real solution of the Lorenz dynamical system (computed using a Runge-Kutta 4th order algorithm). The network was randomly initialized 1000 times and the solution was approximated for 1 second using $N = 100, 90, 80, \ldots, 10$ neurons, always starting from the same initial starting point. The dotted line represents the average squared error of an hypothetical constant readout center at the mean of the real solution in the $[0, 1]$ time interval. **(D)** Peak analysis on a 200 seconds network output using 100 cells (left), 10 cells and full decoder density (p=1, center), 10 cells and sparse decoder (p=0.25, right). Notice the loss of precision for the p=0.25 implementation.



Supp. Fig. S3 – *Representation of the Kronecker product of the input*. The Input $\hat{\mathbf{x}}$ was given by a network which computed a Lorenz system. The second network, using eq. (23), outputs a signal $\approx \hat{\mathbf{x}} \otimes \hat{\mathbf{x}}$. Blue lines represent network output, black dotted lines represent the real $\hat{\mathbf{x}} \otimes \hat{\mathbf{x}}$.