

Expressivity of Spiking Neural Networks

Manjot Singh*

Adalbert Fono*

Gitta Kutyniok

Ludwig-Maximilians-Universität München

SINGH@MATH.LMU.DE

FONO@MATH.LMU.DE

KUTYNIOK@MATH.LMU.DE

Abstract

The synergy between spiking neural networks and neuromorphic hardware holds promise for the development of energy-efficient AI applications. Inspired by this potential, we revisit the foundational aspects to study the capabilities of spiking neural networks where information is encoded in the firing time of neurons. Under the Spike Response Model as a mathematical model of a spiking neuron with a linear response function, we compare the expressive power of artificial and spiking neural networks, where we initially show that they realize piecewise linear mappings. In contrast to ReLU networks, we prove that spiking neural networks can realize both continuous and discontinuous functions. Moreover, we provide complexity bounds on the size of spiking neural networks to emulate multi-layer (ReLU) neural networks. Restricting to the continuous setting, we also establish complexity bounds in the reverse direction for one-layer spiking neural networks.

Keywords: Expressivity, Approximation Theory, Spiking Neural Networks, Deep (ReLU) Neural Networks, Temporal Coding, Linear Regions

1. Introduction

Spiking Neural Networks (SNNs), sometimes considered as the third generation of neural networks, have recently emerged as a notable paradigm in neural computing. In traditional artificial neural networks (ANNs), information is propagated synchronously through the network, whereas SNNs are based on asynchronous information transmission in the form of an *action-potential* or a *spike* (Gerstner et al., 2014). Spikes can be considered as point-like events in time, where incoming spikes received via a neuron’s synapses trigger new spikes in the outgoing synapses. Hence, a key difference between ANNs and SNNs lies in the significance of timing in the operation of SNNs. Moreover, the (typically) real-valued input information to an SNN needs to be encoded in the form of spikes, necessitating a spike-based encoding scheme.

Different encoding schemes enable spiking neurons to represent real-valued inputs, broadly categorized into rate and temporal coding (Gerstner and van Hemmen, 1993). Rate coding refers to the number of spikes in a given time period whereas in temporal coding, the precise timing of spikes matters (Maass, 2001). The notion of firing rate adheres to neurobiological experiments where it was observed that some neurons fire frequently in response to some external stimuli (Stein, 1967; Gerstner et al., 2014). However, the firing rate results in high latency and is computationally expensive due to an overhead related to temporal averaging. The latest experimental results indicate that the firing time of a neuron is essential for the system to respond fast to more complex sensory stimuli (Hopfield, 1995; Thorpe et al., 1996; Abeles, 1991). With the firing time, each spike carries a significant amount of information, thus the resulting signal can be quite sparse.

*. Equal Contribution

While there is no general consensus on the description of neural coding, in this work, we assume that the information is encoded exclusively in the firing time of a neuron. The event-driven, sparse information propagation, as seen in time-to-first-spike encoding (Gerstner and Kistler, 2002), facilitates system efficiency in terms of reduced computational power and improved energy efficiency in comparison to the substantial time and energy consumption associated with training and inferring on ANNs (Thompson et al., 2021). This concept is particularly relevant in the context of neuromorphic computing (Schuman et al., 2022), where a hardware architecture based on SNNs is designed to mimic the human brain’s structure and functioning to achieve efficient information processing.

It is clear that the differences in the processing of information between ANNs and SNNs should also lead to differences in the computations performed by these models. Several groups have analyzed the expressive power of ANNs (Yarotsky, 2017; Cybenko, 1989; Gühring et al., 2022; Petersen and Voigtlaender, 2018), and in particular provided explanations for the superior performance of deep networks over shallow ones (Daubechies et al., 2022; Yarotsky, 2017). In the case of ANNs with ReLU activation function, the number of linear regions into which the input space is partitioned is another property that highlights the advantages of deep networks over shallow ones. Unlike shallow networks, deep networks divide the input space into exponentially more linear regions (Goujon et al., 2024; Montúfar et al., 2014) enabling them to express more complex functions. There exist further approaches to characterize the expressiveness of ANNs, e.g., the concept of VC-dimension in the context of classification problems (Bartlett et al., 1999; Goldberg and Jerrum, 1995; Bartlett et al., 2019).

Few attempts have been made to understand the computational power of SNNs. The works by Maass (1996a,b) demonstrate the capability of spiking neurons to emulate Turing machines, arbitrary threshold circuits, and sigmoidal neurons in temporal coding. In Maass (1997), biologically relevant functions are depicted that can be emulated by a single spiking neuron but require complex ANNs to achieve the same task. Comsa et al. (2020), Maass (1995) showed that continuous functions can be approximated to arbitrary precision in temporal coding. A connection between SNNs and piecewise linear functions was noted in Mostafa (2018). The author showed that an SNN consisting of non-leaky integrate and fire neurons and temporal coding exhibits a piecewise linear input-output relation after a transformation of the time variable. A common theme is that the model of spiking neurons and the description of their dynamics varies, i.e., they are chosen and adjusted for a specific goal or task. Stöckl and Maass (2021) aims at generating high-performance SNNs for image classification using a modified spiking neuron model that limits the number of spikes emitted by each neuron while considering precise spike timing. In Zhang and Zhou (2022), the authors investigate self-connection SNNs, demonstrating their capacity to efficiently approximate discrete dynamical systems. Moraitis et al. (2021) showcases the ability of SNNs using short-term spike-timing-dependent-plasticity mechanism to model certain dynamic environments.

The primary challenge in advancing the field of SNNs has revolved around devising training methodologies. The typical approach is to either train from scratch (Lee et al., 2020; Wu et al., 2018; Comsa et al., 2020; Göltz et al., 2021) or convert trained ANNs into SNNs performing the same tasks (Rueckauer et al., 2017; Kim et al., 2018; Rueckauer and Liu, 2021, 2018; Stanojevic et al., 2022, 2023; Zhang et al., 2019; Yan et al., 2021). The latter works concentrate on the algorithmic construction of SNNs approximating or emulating given ANNs for various spike patterns, encoding schemes, and spiking neuron models. Therefore, we aim to extend the theoretical understanding of the differences and similarities in the expressive power between a network of spiking and artificial neurons employing a piecewise-linear activation function.

Contributions In this paper, to analyze SNNs, we employ the noise-free version of the Spike Response Model (SRM) (Gerstner, 1995). It describes the state of a neuron as a weighted sum of response and threshold functions. We assume a linear response function, where additionally each neuron spikes at most once to encode information through precise spike timing. The spiking networks based on the linear SRM are succinctly referred to as LSNNs. The main results are centered around the comparison of expressive power between LSNNs and ANNs:

- **Similarities between LSNNs and ReLU-ANNs:** We show that LSNNs are as expressive as ANNs with piecewise activation when expressing various functions.
 - We prove that the mapping generated by an LSNN is piecewise linear and under certain settings continuous, concave, and increasing.
 - We show that there exists an LSNN that emulates the ReLU non-linearity. Then, we extend the result to multi-layer neural networks and show that LSNNs have the capacity to effectively emulate any (ReLU) ANN. Furthermore, we present explicit complexity bounds for constructing an LSNN capable of realizing an equivalent ANN. We also provide insights into the influence of the encoding scheme and the impact of different parameters on the above expressivity results. These findings imply that LSNNs can approximate any function as accurately as deep ANNs with a piecewise linear activation function.
- **Differences between LSNNs and ReLU-ANNs:** We prove distinctive characteristics of LSNNs that distinguish them from ReLU-ANNs, thus illustrating differences in the structure of computations between LSNNs and ANNs.
 - We show that the mapping generated by LSNNs may be discontinuous which is in contrast to a ReLU-ANN, which outputs a continuous piecewise linear mapping. This suggests that LSNNs might be better suited for approximating / realizing discontinuous piecewise functions.
 - We demonstrate that the maximum number of linear regions that a one-layer LSNN generates scales exponentially with input dimension. Consequently, a shallow LSNN can be as expressive as a deep ReLU network in terms of the number of linear regions required to express certain types of continuous piecewise linear functions. Additionally, we give upper bounds on the size of ReLU-ANNs to emulate one-layer LSNNs.

Broader impact The findings presented herein deepen our understanding of the theoretical capabilities of SNNs and their differences from ANNs. Although we consider a simplified model of spiking dynamics within the LSNN framework, we obtain insights into event-driven computations where time plays a critical role. Moreover, our results further extend the understanding of the approximation properties of spiking neural networks, emphasizing their potential as an alternative computational model for handling complex tasks. By studying the theoretical power of SNNs, we aim to contribute to the realization of energy-efficient and low-power AI on neuromorphic hardware, providing viable options in contrast to established deep learning models.

Outline In Section 2, we introduce necessary definitions, including spiking neural networks and their realization under the Spike Response Model. We present our main results in Section 3. We conclude in Section 4 by summarizing the limitations and implications of our results. The proofs of all the results are provided in Appendix A.

2. Spiking neural networks

In neuroscience literature, several mathematical models exist that describe the generation and propagation of action-potentials. Action-potentials or spikes are short electrical pulses that are the result of electrical and biochemical properties of a biological neuron. We refer to Gerstner et al. (2014) for a comprehensive and detailed introduction to the dynamics of spiking neurons. To study the expressivity of SNNs, the main principles of a spiking neuron are condensed into a (simplified) mathematical model, where certain details about the biophysics of a biological neuron are neglected.

2.1. Spiking neurons under Spike Response Model

Following Maass (1996a), we consider the Spike Response Model (SRM) (Gerstner, 1995) as a formal model for a spiking neuron. It effectively captures the dynamics of the Hodgkin-Huxley model (Kistler et al., 1997; Gerstner et al., 2014), the most accurate model in describing neuronal dynamics, and is a generalized version of the leaky integrate and fire model (Gerstner, 1995). The SRM leads to the subsequent definition of an SNN (Maass, 1996b).

Definition 1 A spiking neural network Φ under the SRM is a (simple) finite directed graph (V, E) and consists of a finite set V of spiking neurons, a subset $V_{in} \subset V$ of input neurons, a subset $V_{out} \subset V$ of output neurons, and a set $E \subset V \times V$ of synapses. Each synapse $(u, v) \in E$ is associated with a synaptic weight $w_{uv} \geq 0$, a synaptic delay $d_{uv} \geq 0$, and a response function $\varepsilon_{uv} : \mathbb{R} \rightarrow \mathbb{R}$, which depends on the synaptic delay. Each neuron $v \in V \setminus V_{in}$ is associated with a firing threshold $\theta_v > 0$, and a membrane potential $P_v : \mathbb{R} \rightarrow \mathbb{R}$, which is given by

$$P_v(t) = \sum_{(u,v) \in E} \sum_{t_u^f \in F_u} w_{uv} \varepsilon_{uv}(t - t_u^f), \quad (1)$$

where $F_u = \{t_u^f : 1 \leq f \leq n \text{ for some } n \in \mathbb{N}\}$ denotes the set of firing times of a neuron u , i.e., times t whenever $P_u(t)$ reaches θ_u from below.

In general, the membrane potential also includes the *threshold function* $\Theta_v : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{>0}$, that models the refractoriness effect. That is, if a neuron v emits a spike at time t_v^f , v cannot fire again for some time interval immediately after t_v^f , regardless of how large its potential might be. However, we assume that each neuron fires at most once, i.e., information is encoded in the firing time of single spikes. Thus, in Definition 1, the refractoriness effect can be ignored, and the contribution of Θ_v is modeled by the constant θ_v . Moreover, the single spike condition simplifies (1) to

$$P_v(t) = \sum_{(u,v) \in E} w_{uv} \varepsilon_{uv}(t - t_u), \quad \text{where } t_u \text{ is the firing time of presynaptic neuron } u. \quad (2)$$

The response function ε_{uv} models the impact of a spike from a presynaptic neuron u on the membrane potential of a postsynaptic neuron v (Gerstner, 1995). A biologically realistic approximation

of ε_{uv} is a delayed α function (Gerstner, 1995), which is non-linear and leads to intractable problems when analyzing the propagation of spikes through an SNN. Hence, following Maass (1996a), we consider a simplified response and only require ε_{uv} to satisfy the following condition:

$$\varepsilon_{uv}(t) = \begin{cases} 0, & \text{if } t \notin [d_{uv}, d_{uv} + \delta], \\ s \cdot (t - d_{uv}), & \text{if } t \in [d_{uv}, d_{uv} + \delta], \end{cases} \quad \text{where } s \in \{+1, -1\} \text{ and } \delta > 0. \quad (3)$$

The parameter δ is some constant assumed to be the length of a linear segment of the response function. The variable s reflects the fact that biological synapses are either *excitatory* or *inhibitory* and the synaptic delay d_{uv} is the time required for a spike to travel from u to v . Inserting condition (3) in (2) and setting $w_{uv} := s \cdot w_{uv}$, i.e., allowing w_{uv} to take arbitrary values in \mathbb{R} , yields

$$P_v(t) = \sum_{(u,v) \in E} \mathbf{1}_{\{0 < t - t_u - d_{uv} \leq \delta\}} w_{uv}(t - t_u - d_{uv}) \quad (4)$$

Using (4) enables us to iteratively compute the firing time t_v of each neuron $v \in V \setminus V_{\text{in}}$ if we know the firing time t_u of each neuron $u \in V$ with $(u, v) \in E$ by solving for t in

$$\begin{aligned} \inf_{t \in \mathbb{R}} P_v(t) &= \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} \mathbf{1}_{\{0 < t - t_u - d_{uv} \leq \delta\}} w_{uv}(t - t_u - d_{uv}) = \theta_v, \\ \text{i.e., } t_v &= \frac{\theta_v + \sum_{(u,v) \in E} \mathbf{1}_{\{0 < t_v - t_u - d_{uv} \leq \delta\}} w_{uv}(t_u + d_{uv})}{\sum_{(u,v) \in E} \mathbf{1}_{\{0 < t_v - t_u - d_{uv} \leq \delta\}} w_{uv}}. \end{aligned} \quad (5)$$

Observe that t_v is a weighted sum (up to a positive constant) of the firing times of neurons u , $(u, v) \in E$, actually contributing to the firing of v . For instance, if $t_z + d_{zv} > t_v$ for some synapse $(z, v) \in E$, then z did not influence the firing of v since the spike from z arrived after v already fired. Depending on the firing time of the presynaptic neurons and the associated parameters (weights, delays, threshold), a specific subset of presynaptic neurons triggers the firing in v so that t_v changes accordingly. The dynamics of a neuron in this model are depicted in Figure 1.

Definition 2 We call an SNN based on the SRM with the additional assumptions (3) and (4) an LSNN and the corresponding spiking neurons LSNN neurons.

2.2. Realizations of LSNNs

A common representation of feedforward ANNs is based on a sequence of matrix-vector tuples (Berner et al., 2022), (Petersen and Voigtlaender, 2018), whereby a distinction between the network and the target function it realizes is established.

Definition 3 Let $L, N_0, \dots, N_L \in \mathbb{N}$. An artificial neural network Ψ is a sequence of matrix-vector tuples

$$\Psi = ((W^1, B^1), (W^2, B^2), \dots, (W^L, B^L)),$$

where each $W^\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell}$ and $B^\ell \in \mathbb{R}^{N_\ell}$. N_0 and N_L are the input and output dimension of Ψ . We call $N(\Psi) := \sum_{j=0}^L N_j$ the number of neurons of the network Ψ , $L(\Psi) := L$ the number of layers of Ψ and N_ℓ the width of Ψ in layer ℓ . The realization of Ψ with component-wise activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is defined as the map $\mathcal{R}_\Psi : \mathbb{R}^{N_0} \rightarrow \mathbb{R}^{N_L}$, $\mathcal{R}_\Psi(x) = y_L$, where y_L results from

$$y_0 = x, \quad y_\ell = \sigma((W^\ell)^T y_{\ell-1} + B^\ell), \quad \text{for } \ell = 1, \dots, L-1, \quad \text{and } y_L = (W^L)^T y_{L-1} + B^L. \quad (6)$$

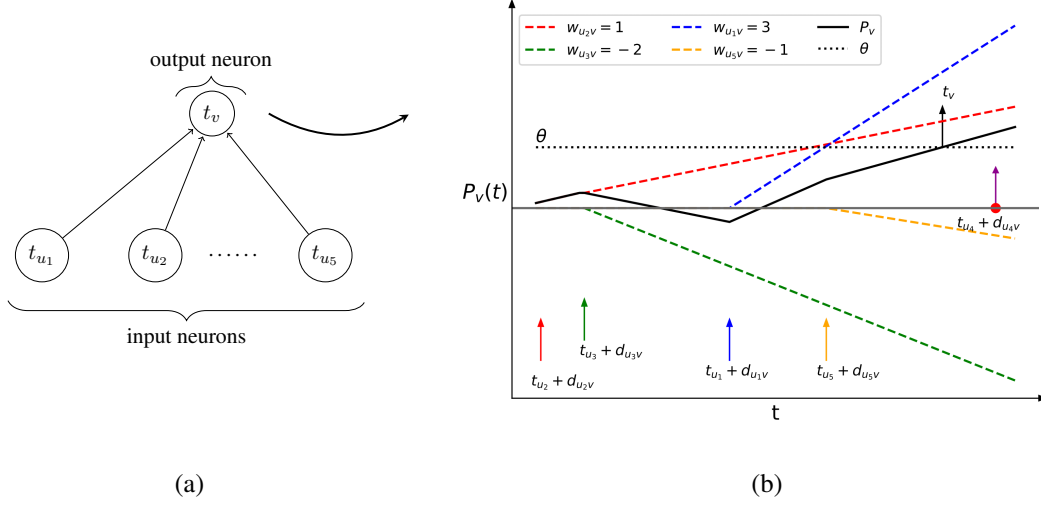


Figure 1: (a) An LSNN neuron v with five input neurons u_1, \dots, u_5 that fire at times t_{u_1}, \dots, t_{u_5} , respectively. (b) The trajectory in black shows the evolution of the membrane potential $P_v(t)$ of v as a result of incoming spikes (vertical arrows). Neurons u_1 and u_2 generate positive responses, whereas neurons u_3 and u_5 trigger negative responses, with the response magnitudes denoted by $w_{u_i v}$. The spike from neuron u_4 does not influence the firing time t_v of v since $t_v < t_{u_4} + d_{u_4 v}$.

Remark 4 Henceforth, $\sigma(x) = \max(0, x)$ denotes the ReLU activation.

An analogous framework can be derived for LSNNs by arranging the underlying graph in layers and equivalently representing LSNNs by a sequence of their parameters.

Definition 5 Let $L, N_0, \dots, N_L \in \mathbb{N}$. An LSNN Φ associated to the acyclic graph (V, E) is a sequence of matrix-matrix-vector tuples

$$\Phi = ((W^1, D^1, \Theta^1), (W^2, D^2, \Theta^2), \dots, (W^L, D^L, \Theta^L))$$

where each $W^\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell}$, $D^\ell \in \mathbb{R}_{\geq 0}^{N_{\ell-1} \times N_\ell}$, and $\Theta^\ell \in \mathbb{R}_{> 0}^{N_\ell}$. The matrix entries W_{uv}^ℓ and D_{uv}^ℓ represent the weight and delay value associated with the synapse $(u, v) \in E$, respectively, and the entry Θ_v^ℓ is the firing threshold associated with node $v \in V$ in layer ℓ . N_0 is the input dimension and N_L is the output dimension of Φ . We call $N(\Phi) := \sum_{j=0}^L N_j$ the number of neurons and $L(\Phi) := L$ denotes the number of layers of Φ .

Before turning to the realization of LSNNs, we highlight two assumptions we will rely on, which allow us to analyze the LSNN framework in the most basic setting:

Assumption I: The parameter δ describing the length of the linear segment of the response function introduced in (3) is assumed to be large or even infinite. Then the minimization problem in (5) to obtain the firing time t_v of a neuron v simplifies to

$$\inf_{t \in \mathbb{R}} P_v(t) = \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} \mathbf{1}_{\{t > t_u + d_{uv}\}} w_{uv}(t - t_u - d_{uv}) = \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} w_{uv} \sigma(t - t_u - d_{uv}) = \theta_v \quad (7)$$

so that

$$t_v = \frac{\theta_v + \sum_{(u,v) \in E: t_v > t_u + d_{uv}} w_{uv}(t_u + d_{uv})}{\sum_{(u,v) \in E: t_v > t_u + d_{uv}} w_{uv}}. \quad (8)$$

Informally, a large linear segment entails that spikes have a constant effect on postsynaptic neurons so that spikes do not act as point-like events in time. The obtained framework, which exhibits similarities to the integrate and fire model, enables us to assess the spiking dynamics of LSNN neurons and gain insights into what we can expect from more generalized models incorporating multi-spike responses and refractoriness effects. In contrast, a biologically more realistic small linear segment requires incoming spikes to have a correspondingly small time delay to jointly affect the potential of a neuron. Otherwise, the impact of the earlier spikes on the potential may already have vanished before the subsequent spikes arrive. The resulting model resembles the leaky integrate and fire model. In conclusion, incorporating δ as an additional parameter in the LSNN framework leads to additional complexity since the same firing patterns may result in different outcomes. However, an in-depth analysis of this effect is left as future work.

Assumption II: The sum of incoming weights of each neuron v in an LSNN Φ is assumed to be positive. The positivity ensures that each neuron in Φ emits a spike, in particular, it is a sufficient but not a necessary condition to guarantee that spikes are emitted by the output neurons. One can certainly treat LSNNs without requiring that neurons have to spike, which again leads to augmented complexity due to increased flexibility in the model.

Under the introduced conditions, the firing time of LSNN neurons can be considered as well-defined mappings in the following sense.

Definition 6 *Let Φ be an LSNN with input neurons u_1, \dots, u_d and output neurons v_1, \dots, v_n . For any firing time of the input neurons $(t_{u_1}, \dots, t_{u_d})^T \in \mathbb{R}^d$ and the corresponding firing times of the output neurons $(t_{v_1}, \dots, t_{v_n})^T \in \mathbb{R}^n$ determined via (7), we denote by $t_\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^n$, $(t_{u_1}, \dots, t_{u_d}) \mapsto t_\Phi(t_{u_1}, \dots, t_{u_d}) = (t_{v_1}, \dots, t_{v_n})^T$ the firing mapping of Φ .*

The key feature of any SNN is the asynchronous information propagation in the spiking domain due to variable firing times among neurons. Hence, to employ SNNs, the (typically real-valued) input information needs to be encoded in the firing times of the neurons in the input layer, and similarly, the firing times of the output neurons need to be translated back to an appropriate target domain. We will refer to this process as input encoding and output decoding. The applied encoding scheme certainly depends on the specific task at hand and the potential power and suitability of different encoding schemes is a topic that warrants separate investigation on its own. Our focus in this work lies on exploring the intrinsic capabilities of LSNNs, rather than the specifics of the encoding scheme.

Thus, we can formulate some guiding principles for establishing a reasonable encoding scheme. First, the firing times of input and output neurons should encode real-valued information in a consistent way so that different networks can be concatenated in a well-defined manner. This enables us to construct suitable subnetworks and combine them appropriately to solve more complex tasks. One can perform basic actions on neural networks such as concatenation and parallelization to construct larger networks from existing ones. Adapting a general approach for ANNs as defined in [Berner et al. \(2022\)](#); [Petersen and Voigtlaender \(2018\)](#), we formally introduce the concatenation and parallelization of networks of spiking neurons in the [Appendix A.1](#). Second, in the extreme case, the encoding scheme might directly contain the solution to a problem, underscoring the need for a sufficiently simple and broadly applicable encoding scheme to avoid this.

Definition 7 Let $[a, b]^d \subset \mathbb{R}^d$ and Φ be an LSNN with input neurons u_1, \dots, u_d and n output neurons. Fix reference times $T_{in} \in \mathbb{R}^d$ and $T_{out} \in \mathbb{R}^n$ via $T_{in} = t_{in}(1, \dots, 1)^T$ and $T_{out} = t_{out}(1, \dots, 1)^T$, respectively, where $t_{in}, t_{out} \in \mathbb{R}$ with $t_{out} > t_{in}$. For any $x \in [a, b]^d$, we set the firing times of the input neurons to $(t_{u_1}, \dots, t_{u_d})^T = T_{in} + x$. The corresponding firing times of the output neurons $t_\Phi(t_{u_1}, \dots, t_{u_d}) = T_{out} + y$ encode the target $y \in \mathbb{R}^n$. The realization of Φ is defined as the map $\mathcal{R}_\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^n$,

$$\mathcal{R}_\Phi(x) = -T_{out} + t_\Phi(t_{u_1}, \dots, t_{u_d}) = y.$$

Remark 8 A bounded input range ensures that appropriate reference times can be fixed. Note that the introduced encoding scheme translates real-valued information into input firing times in a continuous manner. Occasionally, we will point out the effect of adjusting the scheme.

3. Main results

Subsequently, we will employ the framework introduced in Section 2 to analyze the properties of LSNNs. First, we prove that LSNNs generate **C**ontinuous **P**iecewise **L**inear (CPWL) mappings under certain conditions on the weights. Next, we show that LSNNs can emulate the realization of any multi-layer ANN employing ReLU as an activation function. We analyze the number of linear regions generated by LSNNs and compare the arising pattern to the well-studied case of ReLU-ANNs. Lastly, our findings show that LSNNs can efficiently realize certain CPWL functions using fewer computational units and layers compared to ReLU-ANNs. If not stated otherwise, the encoding scheme introduced in Definition 7 is applied and the results need to be understood concerning this specific encoding.

3.1. Characterization of functions expressed by LSNNs

A broad class of ANNs based on a wide range of activation functions such as ReLU generate CPWL mappings (Dym et al., 2020; DeVore et al., 2021). In other words, these ANNs partition the input domain into regions, the so-called linear regions, on which an affine function represents the neural network's realization. Analyzing the firing mapping introduced in Definition 6, we find that LSNNs exhibit a similar behaviour although the continuity is not necessarily maintained. The proof of the statement can be found in Appendix A.2.

Theorem 9 Let $\Phi = ((W^1, D^1, \Theta^1), (W^2, D^2, \Theta^2), \dots, (W^L, D^L, \Theta^L))$ be an LSNN. The firing mapping t_Φ is PWL. If additionally

$$W_{uv}^\ell + \sum_{z: W_{zv}^\ell \leq 0} W_{zv}^\ell > 0 \quad \text{for all } \ell \text{ and } u, v \text{ with } W_{uv}^\ell > 0 \quad (9)$$

holds, i.e., each incoming positive synaptic weight of any neuron v is larger than the absolute value of the sum of its incoming negative synaptic weights, then t_Φ is a CPWL mapping.

Sketch of proof First, via (8) one can derive that the firing mapping of an LSNN neuron with arbitrarily many presynaptic neurons is PWL. Since Φ consists of LSNN neurons arranged in layers it immediately follows that the firing map of each layer is PWL. Thus, as a composition of PWL mappings t_Φ itself is PWL. Moreover, if all weights in Φ are positive, then the continuity of t_Φ at

the breakpoints of the linear regions can be directly verified. In contrast, negative weights can under certain circumstances create a plateau or decrease the potential, causing a delay in firing, hence, resulting in a (jump-)discontinuity in t_Φ . However, this effect can be excluded via (9) so that t_Φ is continuous if (9) holds. ■

The condition given in (9) is sufficient but not necessary to generate CPWL mappings; a corresponding example is provided in Appendix A.2. Under stronger assumptions, we can further characterize the properties of LSNNs. The properties can be verified mainly by repeated application of (7) and (8); the detailed computations are presented in Appendix A.2.

Proposition 10 *Let Φ be an LSNN with only positive weights. Then t_Φ is an increasing and concave function. Additionally, the firing time of a neuron v in Φ with corresponding parameter $(w, d, \theta) \in \mathbb{R}^d \times \mathbb{R}_{\geq 0}^d \times \mathbb{R}_{> 0}$ and firing times $t_{u_1}, \dots, t_{u_d} \in \mathbb{R}$ in the previous layer is given by*

$$t_v(t_{u_1}, \dots, t_{u_d}) = \inf_{\emptyset \neq I \subset [d]} \left\{ s^I = \frac{1}{\sum_{i \in I} w_i} \left(\theta + \sum_{i \in I} w_i (t_{u_i} + d_i) \right) : s^I > \max_{i \in I} t_{u_i} \right\}.$$

The properties described in Proposition 10 are in general not true if the positive weights assumption is dropped. An immediate follow-up question is if the above findings apply to the realization of LSNNs via input and output encoding on a bounded domain. It is immediate to verify that the realization of an LSNN Φ is CPWL, increasing or concave if the encoding scheme and t_Φ are CPWL, increasing or concave, respectively, which certainly holds for the encoding presented in Definition 7. However, even if t_Φ is not CPWL, increasing or concave, the corresponding feature can still arise in the realization due to the bounded input domain as the constructions in the subsequent results indicate. In particular, we show that LSNNs can realize the ReLU activation and as a consequence any multi-layer ReLU ANN. For the proof, please refer to Sections A.3 and A.4 in the Appendix.

Theorem 11 *Let $L, d \in \mathbb{N}$, $[a, b]^d \subset \mathbb{R}^d$ and let Ψ be an arbitrary ANN of depth L and fixed width d employing a ReLU non-linearity. Then, there exists an LSNN Φ with $N(\Phi) = N(\Psi) + L(2d + 3) - (2d + 2)$ and $L(\Phi) = 3L - 2$ that realizes \mathcal{R}_Ψ on $[a, b]^d$.*

Sketch of proof Any multi-layer ANN with ReLU activation is simply an alternating composition of affine functions A^ℓ determined by the weights W^ℓ and biases B^ℓ in layer ℓ and a non-linear function represented by σ . Thus, to construct an LSNN Φ that realizes \mathcal{R}_Ψ , we first construct LSNNs that realize affine-linear functions and the ReLU non-linearity. Subsequently, we compose these subnetworks to obtain Φ . Thereby, we realize A^ℓ by an LSNN with the same weights W^ℓ , which may be negative. Therefore, in the LSNN construction, we rely on the value of the threshold parameter, which may depend on $[a, b]$, and auxiliary neurons with appropriate weights that ensure the firing of the output neurons and the desired (continuous) realization. ■

Remark 12 *The result can be generalized to ANNs with varying widths that employ any type of PWL activation function. We note that the encoding scheme that converts the analog values into the time domain plays a crucial role. We construct a two-layer LSNN that realizes σ via the encoding scheme $(T_{in} + \cdot)$ and $(T_{out} + \cdot)$. At the same time, the encoding scheme $(T_{in} - \cdot)$ and $(T_{out} - \cdot)$ fails in the two-layer case, whereas utilizing an inconsistent input and output encoding enables us to construct a one-layer LSNN that realizes σ . This shows that not only the network but also the applied encoding scheme is highly relevant. For details, we refer to Appendix A.3.*

It is well known that ReLU-ANNs not only realize CPWL mappings but that every CPWL function can be represented by ReLU-ANNs (Arora et al., 2018). Theorem 11 implies that LSNNs are as expressive as any ReLU-ANN, i.e., LSNNs can represent every ReLU-ANN and thereby every CPWL function with similar complexity. However, in a hypothetical real-world implementation, which certainly includes some noise, the constructed LSNN is not necessarily robust with respect to input perturbation. Additionally, the complexity of an LSNN can be captured in other ways than in terms of the number of computational units and layers, e.g., the total number of spikes emitted in LSNNs is related to its energy consumption since emitting spikes consumes energy. Hence, the minimum number of spikes needed to realize a given function class may be a reasonable complexity measure with regard to energy efficiency for SNNs. Further research in these directions is necessary to analyze the behaviour under noise and provide error estimations as well as to evaluate the complexity of LSNNs via different measures with their benefits and drawbacks.

3.2. Bounds on the complexity of ReLU-ANNs for expressing LSNNs

In this section, we further explore the differences in the computational structure between LSNNs and ReLU-ANNs. An already observed major distinction is the ability of LSNNs to realize discontinuous functions. Aside from this fact, can we establish dissimilarities when restricted to continuous realizations? Since ReLU-ANNs can represent any CPWL mapping, they can realize any LSNN with a CPWL realization, in particular, LSNNs with positive weights and a CPWL encoding scheme. Hence, the key difference in the realization of arbitrary CPWL mappings may be the necessary size and complexity of the respective ANN and LSNN. To that end, we give upper bounds on the complexity of ReLU-ANNs needed to realize corresponding LSNNs. The first step in establishing the result is the study of the number of linear regions that LSNNs generate.

The number of linear regions can be seen as a measure of the flexibility and expressivity of the associated CPWL function. Similarly, we can measure the expressivity of an ANN by the number of linear regions of its realization. The connection of the depth, width, and activation function of an ANN to the maximum number of its linear regions is well-established, e.g., with increasing depth the number of linear regions can grow exponentially in the number of parameters of an ANN (Montúfar et al., 2014; Arora et al., 2018; Goujon et al., 2024). In the following, we observe a distinct scaling behaviour for LSNNs.

Lemma 13 *Let Φ be a one-layer LSNN with a single output neuron v and input neurons u_1, \dots, u_d . Then t_Φ partitions the input domain into at most $2^d - 1$ linear regions. The maximal number of linear regions is attained if and only if all synaptic weights are positive.*

Proof By Theorem 9, we observe that t_Φ is a PWL mapping. It can be inferred via (8) that each linear region corresponds to a subset of input neurons responsible for the firing of v on that specific domain. Hence, the number of regions is bounded by the number of non-empty subsets of $\{u_1, \dots, u_d\}$, i.e., $2^d - 1$. Now, observe that any subset of input neurons causes a spike in v if and only if the sum of their weights is positive. Otherwise, inputs from the corresponding region cannot trigger a spike in v since their net contribution is negative, i.e., the potential does not reach the threshold θ_v . Hence, the maximal number of regions is attained if and only if all weights are positive, and thereby the sum of weights of any subset of input neurons is positive as well. ■

A one-layer ReLU-ANN with one output neuron will partition the input domain into at most two linear regions, independent of the dimension of the input. In contrast, for a one-layer LSNN

with one output neuron, the maximum number of linear regions scales exponentially in the input dimension. In the case of LSNNs, non-linearity is an intrinsic property of the model and emerges from the subset of neurons that affect the firing time of the output neuron, whereas in ANNs a non-linear activation is directly applied to the output neuron. By shifting the non-linearity and applying it to the input, ANNs could exhibit the same exponential scaling of the linear regions as LSNNs. However, this change has rather a detrimental effect on the expressivity since the partitioning of the input domain is fixed and independent of the parameters of the ANN. The flexibility of LSNNs to generate arbitrary linear regions is to a certain extent limited, albeit not entirely restricted as in the adjusted ANN; this is exemplarily demonstrated for a two-dimensional input space in Appendix A.2.

Analogously to the result in Theorem 11, a natural question is: what is the complexity of the ReLU ANN needed to emulate a given LSNN? The full power of ANN comes into play with large numbers of layers, however, the result in Lemma 13 suggests that a shallow LSNN can be as expressive as a deep ReLU network in terms of the number of linear regions. In the following, we give upper bounds on depth and number of computational units required for a ReLU-ANN to express a one-layer LSNN with d -dimensional input.

Theorem 14 *For $d \geq 2$, $\ell := \lceil \log_2(d + 1) \rceil + 1$. Let Φ be a one-layer LSNN with one output neuron v and d input neurons u_1, \dots, u_d with $w_{u_i, v} \in \mathbb{R}_{>0}$ for $i \in [d]$. Then,*

- (a) t_Φ can be realized by a ReLU-ANN Ψ with $L(\Psi) = \ell$ and $N(\Psi) \in \mathcal{O}(\ell \cdot 2^{2d^3+3d^2+d})$.
- (b) t_Φ can be realized by a ReLU-ANN Ψ with $L(\Psi) \in \mathcal{O}(d)$ and $N(\Psi) \in \mathcal{O}(8^d)$.

Proof Since Φ consists of only positive weights, the firing map t_Φ is via Theorem 9 and Lemma 13 a CPWL mapping with $2^d - 1$ linear regions. Using upper bounds on the size of ReLU-ANNs to realize CPWL mappings with a fixed number of linear regions, we obtain the given bounds. Thereby, the result (a) follows from Theorem 9 in Hertrich et al. (2021), and (b) follows from Theorem 1 in Chen et al. (2022). ■

The complexity bounds in (a) and (b) of Theorem 14 are connected to the number of linear regions of the CPWL function that the given LSNN realizes. In (a), to represent this CPWL function with lower depth, a substantial number of units in each layer might be necessary. Conversely, in (b), deep but less wide networks can achieve the same function. To the best of our knowledge, we are not aware of any lower bounds on the depth of the ReLU-ANN in terms of the number of linear regions when expressing any CPWL function. Extending the bounds to multi-layer LSNNs is an important step and is left for future investigation.

Via Theorem 11 and Theorem 14, we provide upper bounds on the size of LSNNs to realize ReLU networks and vice versa. Although the bounds presented in Theorem 11 and Theorem 14 are not optimal, however, note that the bound on the size of ReLU-ANNs to emulate one-layer LSNNs grows exponentially in the input dimension whereas for LSNNs to emulate L -layer ReLU-ANNs, the bound scales linearly in the input dimension. These bounds (and their derivation) suggest that LSNNs and ReLU-ANNs offer distinct benefits for realizing certain types of CPWL functions. These observations highlight the need to establish the associated lower bounds. This will not only shed light on the types of functions for which LSNNs are better suited in terms of emulation / approximation capabilities than ReLU-ANNs but also provide valuable insights into their computational power.

4. Discussion

The central aim of this paper is to study and compare the expressive power of SNNs and ANNs employing any PWL activation function. Our expressivity result in Theorem 11 implies that LSNNs can approximate any function with the same accuracy and a certain complexity overhead as (deep) ANNs employing a piecewise linear activation function, given the response function satisfies some basic assumptions. Most related to Theorem 11 are the results in [Stanojevic et al. \(2023\)](#). Under certain assumptions, the authors define a one-to-one neuron mapping that converts a trained ReLU network to a corresponding SNN consisting of integrate and fire neurons by a non-linear transformation of parameters. However, significant distinctions exist between the approaches, particularly in terms of the chosen model, for instance, with the handling of the threshold parameter. In terms of methodology, we introduce an auxiliary neuron to ensure the firing of neurons even when a corresponding ReLU neuron exhibits zero activity. This diverges from their approach, which employs external current and a special parameter to achieve similar outcomes. We study the differences in the structure of computations between ANNs and SNNs, whereas in [Stanojevic et al. \(2023\)](#), only the conversion of ANNs to SNNs is examined and not vice versa.

Rather than approximating some function space by emulating a known construction for ReLU networks, one could also achieve optimal approximations by leveraging the intrinsic capabilities of LSNNs instead. The findings in Lemma 13 and Theorem 14 indicate that the latter approach may indeed be beneficial in terms of the complexity of the architecture in certain circumstances. However, we point out that finding optimal architectures for approximating different classes of functions is not the focal point of our work. The significance of our results lies in investigating theoretically the approximation and expressivity capabilities of SNNs, highlighting their potential as an alternative computational model for complex tasks. Extending the model of an LSNN neuron by incorporating, e.g., multiple spikes of a neuron, may yield an improvement in our results. However, by increasing the complexity of the model the analysis also tends to be more elaborate. In the aforementioned case of multiple spikes the threshold function becomes important so that additional complexity when approximating some target function is introduced since one would have to consider the coupled effect of response and threshold functions. Similarly, the choice of the response function and the frequency of neuron firings will surely influence the approximation results and we leave this for future work.

Limitations We study similarities and differences in the structure of computations between ANNs and LSNNs and theoretically show that LSNNs can be as expressive as ReLU-ANNs. However, achieving similar results in practice heavily relies on the effectiveness of the employed training algorithms. The implementation of efficient learning algorithms with weights, delays, and thresholds as programmable parameters is left for future work. In this work, our choice of model resides on theoretical considerations and not on practical considerations regarding implementation. However, there might be other models of spiking neurons that are more apt for implementation purposes — see e.g. [Stanojevic et al. \(2023\)](#) and [Comsa et al. \(2020\)](#). Furthermore, in reality, due to the ubiquitous sources of noise in the spiking neurons, the firing activity of a neuron is not deterministic. For mathematical simplicity, we perform our analysis in a noise-free case. Generalizing to the case of noisy spiking neurons is important (for instance concerning the aforementioned implementation in noisy environments) and may lead to further insights into the model.

Acknowledgments

The authors would like to thank Philipp Petersen for helpful discussion and feedback. M. Singh is supported by the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research.

G. Kutyniok acknowledges support from LMUexcellent, funded by the Federal Ministry of Education and Research (BMBF) and the Free State of Bavaria under the Excellence Strategy of the Federal Government and the Länder as well as by the Hightech Agenda Bavaria. Further, G. Kutyniok was supported in part by the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research. G. Kutyniok also acknowledges support from the Munich Center for Machine Learning (MCML) as well as the German Research Foundation under Grants DFG-SPP-2298, KU 1446/31-1 and KU 1446/32-1 and under Grant DFG-SFB/TR 109, Project C09 and the German Federal Ministry of Education and Research (BMBF) under Grant MaGriDo.

References

- M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, 1991.
- Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *ICLR 2018*, 2018.
- Peter Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear VC dimension bounds for piecewise polynomial networks. *Neural Computation*, 10, 1999.
- Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20(63):1–17, 2019.
- Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The Modern Mathematics of Deep Learning. In *Mathematical Aspects of Deep Learning*, pages 1–111. Cambridge University Press, 2022. doi: 10.1017/9781009025096.002.
- Kuan-Lin Chen, Harinath Garudadri, and Bhaskar D. Rao. Improved bounds on neural complexity for representing piecewise linear functions. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *NeurIPS 2022*, volume 35, pages 7167–7180. Curran Associates, Inc., 2022.
- Iulia M. Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020*, pages 8529–8533, 2020. doi: 10.1109/ICASSP40776.2020.9053856.
- George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.
- I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova. Nonlinear approximation and (deep) ReLU networks. *Constructive Approximation*, 55(1):127–172, 2022. doi: 10.1007/s00365-021-09548-z.

- Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021. doi: 10.1017/S0962492921000052.
- Nadav Dym, Barak Sober, and Ingrid Daubechies. Expression of fractals through neural network functions. *IEEE Journal on Selected Areas in Information Theory*, 1(1):57–66, 2020. doi: 10.1109/JSAIT.2020.2991422.
- Wulfram Gerstner. Time structure of the activity in neural network models. *Physical Review E*, 51: 738–758, 1995.
- Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.
- Wulfram Gerstner and J. van Hemmen. How to describe neuronal activity: Spikes, rates, or assemblies? In J. Cowan, G. Tesauro, and J. Alspector, editors, *NIPS 1993*, volume 6. Morgan-Kaufmann, 1993.
- Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.
- Paul W. Goldberg and Mark R. Jerrum. Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers. *Machine Learning*, 18(2):131–148, 1995. doi: 10.1007/BF00993408.
- Alexis Goujon, Arian Etemadi, and Michael Unser. On the number of regions of piecewise linear neural networks. *Journal of Computational and Applied Mathematics*, 441, 2024. doi: 10.1016/j.cam.2023.115667.
- J. Göltz, L. Kriener, Andreas Baumbach, S. Billaudelle, O. Breitwieser, B. Cramer, Dominik Dold, Akos Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and M. Petrovici. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature Machine Intelligence*, 3:823–835, 2021. doi: 10.1038/s42256-021-00388-x.
- Ingo Gühring, Mones Raslan, and Gitta Kutyniok. Expressivity of deep neural networks. In Philipp Grohs and Gitta Kutyniok, editors, *Mathematical Aspects of Deep Learning*, page 149–199. Cambridge University Press, 2022.
- Christoph Hertrich, Amitabh Basu, Marco Di Summa, and Martin Skutella. Towards lower bounds on the depth of relu neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *NeurIPS 2021*, volume 34, pages 3336–3348. Curran Associates, Inc., 2021.
- John Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:33–6, 1995. doi: 10.1038/376033a0.
- Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoun Choi. Deep neural networks with weighted spikes. *Neurocomputing*, 311:373–386, 2018.
- Werner M. Kistler, Wulfram Gerstner, and J. Leo van Hemmen. Reduction of the Hodgkin-Huxley equations to a single-variable threshold model. *Neural Computation*, 9(5):1015–1045, 1997. doi: 10.1162/neco.1997.9.5.1015.

- Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14, 2020. doi: 10.3389/fnins.2020.00119.
- Wolfgang Maass. An efficient implementation of sigmoidal neural nets in temporal coding with noisy spiking neurons. Technical report, Technische Universität Graz, 1995.
- Wolfgang Maass. Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In *NIPS 1996*, volume 9. MIT Press, 1996a.
- Wolfgang Maass. Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8(1):1–40, 1996b. doi: 10.1162/neco.1996.8.1.1.
- Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. doi: 10.1016/S0893-6080(97)00011-7.
- Wolfgang Maass. On the relevance of time in neural computation and learning. *Theoretical Computer Science*, 261(1):157–178, 2001. doi: 10.1016/S0304-3975(00)00137-7.
- Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *NIPS 2014*, page 2924–2932, Cambridge, MA, USA, 2014. MIT Press.
- Timoleon Moraitis, Abu Sebastian, and Evangelos Eleftheriou. Optimality of short-term synaptic plasticity in modelling certain dynamic environments. *arXiv:2009.06808*, 2021.
- Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3227–3235, 2018. doi: 10.1109/TNNLS.2017.2726060.
- Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018. doi: 10.1016/j.neunet.2018.08.019.
- Bodo Rueckauer and Shih-Chii Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *ISCAS 2018*, pages 1–5, 2018. doi: 10.1109/ISCAS.2018.8351295.
- Bodo Rueckauer and Shih-Chii Liu. Temporal pattern coding in deep spiking neural networks. In *IJCNN 2021*, pages 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9533837.
- Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11, 2017. doi: 10.3389/fnins.2017.00682.
- Catherine D. Schuman, Shruti R. Kulkarni, Maryam Parsa, J. Parker Mitchell, Prasanna Date, and Bill Kay. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, 2022.
- Ana Stanojevic, Evangelos Eleftheriou, Giovanni Cherubini, Stanisław Woźniak, Angeliki Pantazi, and Wulfram Gerstner. Approximating ReLU networks by single-spike computation. In *ICIP 2022*, pages 1901–1905, 2022. doi: 10.1109/ICIP46576.2022.9897692.

- Ana Stanojevic, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, and Wulfram Gerstner. An exact mapping from ReLU networks to spiking neural networks. *Neural Networks*, 168:74–88, 2023. doi: 10.1016/j.neunet.2023.09.011.
- R. B. Stein. The information capacity of nerve cells using a frequency code. *Biophysical journal*, 7(6):797–826, 1967.
- Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3:230–238, 2021. doi: 10.1038/s42256-021-00311-4.
- Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. Deep learning’s diminishing returns: The cost of improvement is becoming unsustainable. *IEEE Spectrum*, 58(10):50–55, 2021.
- Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.
- Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12, 2018. doi: 10.3389/fnins.2018.00331.
- Zhanglu Yan, Jun Zhou, and Weng-Fai Wong. Near lossless transfer learning for spiking neural networks. In *AAAI 2021*, volume 35, pages 10577–10584, 2021.
- Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017. doi: <https://doi.org/10.1016/j.neunet.2017.07.002>.
- Lei Zhang, Shengyuan Zhou, Tian Zhi, Zidong Du, and Yunji Chen. TDSNN: From deep neural networks to deep spike neural networks with temporal-coding. In *AAAI 2019*, volume 33, pages 1319–1326, 2019. doi: 10.1609/aaai.v33i01.33011319.
- Shao-Qun Zhang and Zhi-Hua Zhou. Theoretically provable spiking neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *NeurIPS 2022*, volume 35, pages 19345–19356. Curran Associates, Inc., 2022.

Appendix A. Proofs

Outline We start by introducing the spiking network calculus in Section A.1 to compose and parallelize different networks. In Section A.2, we characterize the firing maps of LSNNs. In particular, we show that the firing maps of LSNNs are PWL and under stronger assumptions continuous, increasing, and concave. In Section A.3, we construct an LSNN that emulates the ReLU non-linearity, and subsequently in Section A.4, we prove that an LSNN can realize the output of any ReLU network and simultaneously provide bounds on the required size of the LSNN.

A.1. Spiking neural network calculus

It can be observed from Definition 7 that both inputs and outputs of LSNNs are encoded in a unified format. This characteristic is crucial for concatenating/parallelizing two spiking network architectures that further enable us to attain compositions of network realizations.

We operate in the following setting: Let $L_1, L_2, d_1, d_2, d'_1, d'_2 \in \mathbb{N}$. Consider two LSNNs Φ_1, Φ_2 given by

$$\Phi_i = ((W_1^i, D_1^i, \Theta_1^i), \dots, (W_{L_i}^i, D_{L_i}^i, \Theta_{L_i}^i)), \quad i = 1, 2,$$

with input domains $[a_1, b_1]^{d_1} \subset \mathbb{R}^{d_1}$, $[a_2, b_2]^{d_2} \subset \mathbb{R}^{d_2}$ and output dimension d'_1, d'_2 , respectively. Denote the input neurons by u_1, \dots, u_{d_i} with respective firing times $t_{u_j}^i$. By Definition 6 and 7, we can express the firing times of the input neurons as

$$\begin{aligned} t_u^1(x) &:= (t_{u_1}^1, \dots, t_{u_{d_1}}^1)^T = T_{\text{in}}^1 + x \quad \text{for } x \in [a_1, b_1]^{d_1}, \\ t_u^2(x) &:= (t_{u_1}^2, \dots, t_{u_{d_2}}^2)^T = T_{\text{in}}^2 + x \quad \text{for } x \in [a_2, b_2]^{d_2} \end{aligned} \quad (10)$$

and the realization of the networks as

$$\begin{aligned} \mathcal{R}_{\Phi_1}(x) &= -T_{\text{out}}^1 + t_{\Phi_1}(t_u^1(x)) \quad \text{for } x \in [a_1, b_1]^{d_1}, \\ \mathcal{R}_{\Phi_2}(x) &= -T_{\text{out}}^2 + t_{\Phi_2}(t_u^2(x)) \quad \text{for } x \in [a_2, b_2]^{d_2} \end{aligned} \quad (11)$$

for some constants $T_{\text{in}}^1 \in \mathbb{R}^{d_1}, T_{\text{in}}^2 \in \mathbb{R}^{d_2}, T_{\text{out}}^1 \in \mathbb{R}^{d'_1}, T_{\text{out}}^2 \in \mathbb{R}^{d'_2}$.

We define the concatenation of the two networks in the following way.

Definition 15 (Concatenation) Let Φ_1 and Φ_2 be such that the input layer of Φ_1 has the same dimension as the output layer of Φ_2 , i.e., $d'_2 = d_1$. Then, the concatenation of Φ_1 and Φ_2 , denoted as $\Phi_1 \bullet \Phi_2$, represents the $(L_1 + L_2)$ -layer network

$$\Phi_1 \bullet \Phi_2 := ((W_1^2, D_1^2, \Theta_1^2), \dots, (W_{L_2}^2, D_{L_2}^2, \Theta_{L_2}^2), (W_1^1, D_1^1, \Theta_1^1), \dots, (W_{L_1}^1, D_{L_1}^1, \Theta_{L_1}^1)).$$

Lemma 16 Let $d'_2 = d_1$ and fix $T_{\text{in}} = T_{\text{in}}^2$ and $T_{\text{out}} = T_{\text{out}}^1$. If $T_{\text{out}}^2 = T_{\text{in}}^1$ and $\mathcal{R}_{\Phi_2}([a_2, b_2]^{d_2}) \subset [a_1, b_1]^{d_1}$, then

$$\mathcal{R}_{\Phi_1 \bullet \Phi_2}(x) = \mathcal{R}_{\Phi_1}(\mathcal{R}_{\Phi_2}(x)) \quad \text{for all } x \in [a, b]^{d_2}$$

with respect to the reference times $T_{\text{in}}, T_{\text{out}}$. Moreover, $\Phi_1 \bullet \Phi_2$ is composed of $N(\Phi_1) + N(\Phi_2) - d_1$ computational units.

Proof It is straightforward to verify via the construction that the network $\Phi_1 \bullet \Phi_2$ is composed of $N(\Phi_1) + N(\Phi_2) - d_1$ computational units. Moreover, under the given assumptions $\mathcal{R}_{\Phi_1} \circ \mathcal{R}_{\Phi_2}$ is well-defined so that (10) and (11) imply

$$\begin{aligned}\mathcal{R}_{\Phi_1 \bullet \Phi_2}(x) &= -T_{\text{out}} + t_{\Phi_1}(t_{\Phi_2}(T_{\text{in}} + x)) = -T_{\text{out}}^1 + t_{\Phi_1}(t_{\Phi_2}(T_{\text{in}}^2 + x)) \\ &= -T_{\text{out}}^1 + t_{\Phi_1}(t_{\Phi_2}(t_u^2(x))) = -T_{\text{out}}^1 + t_{\Phi_1}(T_{\text{out}}^2 + \mathcal{R}_{\Phi_2}(x)) \\ &= -T_{\text{out}}^1 + t_{\Phi_1}(T_{\text{in}}^1 + \mathcal{R}_{\Phi_2}(x)) = -T_{\text{out}}^1 + t_{\Phi_1}(t_u^1(\mathcal{R}_{\Phi_2}(x))) \\ &= \mathcal{R}_{\Phi_1}(\mathcal{R}_{\Phi_2}(x)) \quad \text{for } x \in [a_2, b_2]^{d_2}.\end{aligned}$$

■

In addition to concatenating networks, we also perform parallelization operation on LSNNs.

Definition 17 (*Parallelization*) Let Φ_1 and Φ_2 be such that they have the same depth and input dimension, i.e., $L_1 = L_2 =: L$ and $d_1 = d_2 =: d$. Then, the parallelization of Φ_1 and Φ_2 , denoted as $P(\Phi_1, \Phi_2)$, represents the L -layer network with d -dimensional input

$$P(\Phi_1, \Phi_2) := ((\tilde{W}_1, \tilde{D}_1, \tilde{\Theta}_1), \dots, (\tilde{W}_L, \tilde{D}_L, \tilde{\Theta}_L)),$$

where

$$\tilde{W}_1 = \begin{pmatrix} W_1^1 & W_1^2 \end{pmatrix}, \quad \tilde{D}_1 = \begin{pmatrix} D_1^1 & D_1^2 \end{pmatrix}, \quad \tilde{\Theta}_1 = \begin{pmatrix} \Theta_1^1 \\ \Theta_1^2 \end{pmatrix}$$

and

$$\tilde{W}_l = \begin{pmatrix} W_l^1 & 0 \\ 0 & W_l^2 \end{pmatrix}, \quad \tilde{D}_l = \begin{pmatrix} D_l^1 & 0 \\ 0 & D_l^2 \end{pmatrix}, \quad \tilde{\Theta}_l = \begin{pmatrix} \Theta_l^1 \\ \Theta_l^2 \end{pmatrix}, \quad \text{for } 1 < l \leq L.$$

Lemma 18 Let $d := d_2 = d_1$ and fix $T_{\text{in}} := T_{\text{in}}^1$, $T_{\text{out}} := (T_{\text{out}}^1, T_{\text{out}}^2)$, $a := a_1$ and $b := b_1$. If $T_{\text{in}}^2 = T_{\text{in}}^1$, $T_{\text{out}}^2 = T_{\text{out}}^1$ and $a_1 = a_2$, $b_1 = b_2$, then

$$\mathcal{R}_{P(\Phi_1, \Phi_2)}(x) = (\mathcal{R}_{\Phi_1}(x), \mathcal{R}_{\Phi_2}(x)) \quad \text{for } x \in [a, b]^d$$

with respect to the reference times $T_{\text{in}}, T_{\text{out}}$. Moreover, $P(\Phi_1, \Phi_2)$ is composed of $N(\Phi_1) + N(\Phi_2) - d$ computational units.

Proof The number of computational units is an immediate consequence of the construction. Since the input domains of Φ_1 and Φ_2 agree, (10) and (11) show that

$$\begin{aligned}\mathcal{R}_{P(\Phi_1, \Phi_2)}(x) &= -T_{\text{out}} + (t_{\Phi_1}(T_{\text{in}} + x), t_{\Phi_2}(T_{\text{in}} + x)) \\ &= (-T_{\text{out}}^1 + t_{\Phi_1}(T_{\text{in}}^1 + x), -T_{\text{out}}^2 + t_{\Phi_2}(T_{\text{in}}^2 + x)) \\ &= (-T_{\text{out}}^1 + t_{\Phi_1}(t_u^1(x)), -T_{\text{out}}^2 + t_{\Phi_2}(t_u^2(x))) \\ &= (\mathcal{R}_{\Phi_1}(x), \mathcal{R}_{\Phi_2}(x)) \quad \text{for } x \in [a, b]^d.\end{aligned}$$

■

Remark 19 Note that parallelization and concatenation can be straightforwardly extended (recursively) to a finite number of networks. Additionally, more general forms of parallelization and concatenations of networks, e.g., parallelization of networks with different depths, can be established. However, for the constructions presented in this work, the introduced notions suffice.

A.2. Characterization of functions expressed by LSNNs

A.2.1. SPIKING NEURON WITH TWO INPUTS

First, we provide a simple toy example to demonstrate the dynamics of an LSNN neuron. Let v be an LSNN neuron with two input neurons u_1, u_2 . Denote the associated weights and delays by $w_{u_i v} \in \mathbb{R}$ and $d_{u_i v} \geq 0$, respectively, and the threshold of v by $\theta_v > 0$. A spike emitted from v could then be caused by either u_1 or u_2 or a combination of both. Each possibility corresponds to a linear region in the input space \mathbb{R}^2 . We consider each case separately under Assumption I, i.e., δ in (3) is arbitrarily large, and we discuss the implications of this assumption in more detail after presenting the different cases.

Case 1: u_1 causes v to spike before a potential effect from u_2 reaches v . Note that this can only happen if $w_{u_1 v} > 0$ and

$$t_{u_2} + d_{u_2 v} \geq t_v = \frac{\theta_v}{w_{u_1 v}} + t_{u_1} + d_{u_1 v},$$

where we applied (7) and (8), and t_z represents the firing time of a neuron z . Solving for t_{u_2} leads to

$$t_{u_2} \geq \frac{\theta_v}{w_{u_1 v}} + t_{u_1} + d_{u_1 v} - d_{u_2 v}.$$

Case 2: An analogous calculation shows that

$$t_{u_2} \leq -\frac{\theta_v}{w_{u_2 v}} + t_{u_1} + d_{u_1 v} - d_{u_2 v},$$

whenever u_2 causes v to spike before a potential effect from u_1 reaches v .

Case 3: The remaining possibility is that spikes from u_1 and u_2 influence the firing time of v . Then, the following needs to hold: $w_{u_1 v} + w_{u_2 v} > 0$ and

$$\begin{aligned} t_{u_1} + d_{u_1 v} < t_v &= \frac{\theta_v}{w_{u_1 v} + w_{u_2 v}} + \sum_i \frac{w_{u_i v}}{w_{u_1 v} + w_{u_2 v}} (t_{u_i} + d_{u_i v}) \quad \text{and} \\ t_{u_2} + d_{u_2 v} < t_v &= \frac{\theta_v}{w_{u_1 v} + w_{u_2 v}} + \sum_i \frac{w_{u_i v}}{w_{u_1 v} + w_{u_2 v}} (t_{u_i} + d_{u_i v}). \end{aligned}$$

This yields

$$t_{u_2} \begin{cases} > -\frac{\theta_v}{w_{u_2 v}} + t_{u_1} + d_{u_1 v} - d_{u_2 v}, & \text{if } \frac{w_{u_2 v}}{w_{u_1 v} + w_{u_2 v}} > 0 \\ < -\frac{\theta_v}{w_{u_2 v}} + t_{u_1} + d_{u_1 v} - d_{u_2 v}, & \text{if } \frac{w_{u_2 v}}{w_{u_1 v} + w_{u_2 v}} < 0 \end{cases},$$

respectively

$$t_{u_2} \begin{cases} < \frac{\theta_v}{w_{u_1 v}} + t_{u_1} + d_{u_1 v} - d_{u_2 v}, & \text{if } \frac{w_{u_1 v}}{w_{u_1 v} + w_{u_2 v}} > 0 \\ > \frac{\theta_v}{w_{u_1 v}} + t_{u_1} + d_{u_1 v} - d_{u_2 v}, & \text{if } \frac{w_{u_1 v}}{w_{u_1 v} + w_{u_2 v}} < 0 \end{cases}.$$

Example 1 In a simple setting with $\theta_v = w_{u_1v} = d_{u_2v} = 1$ and $d_{u_1v} = 2$, the above considerations imply the following firing time of v on the corresponding linear regions (see Figure 2):

$$t_v = \begin{cases} t_{u_1} + 3, & \text{if } t_{u_2} \geq t_{u_1} + 2 \\ t_{u_2} + 2, & \text{if } t_{u_2} \leq t_{u_1} \\ \frac{1}{2}(t_{u_1} + t_{u_2}) + 2, & \text{if } t_{u_1} < t_{u_2} < t_{u_1} + 2 \end{cases}.$$

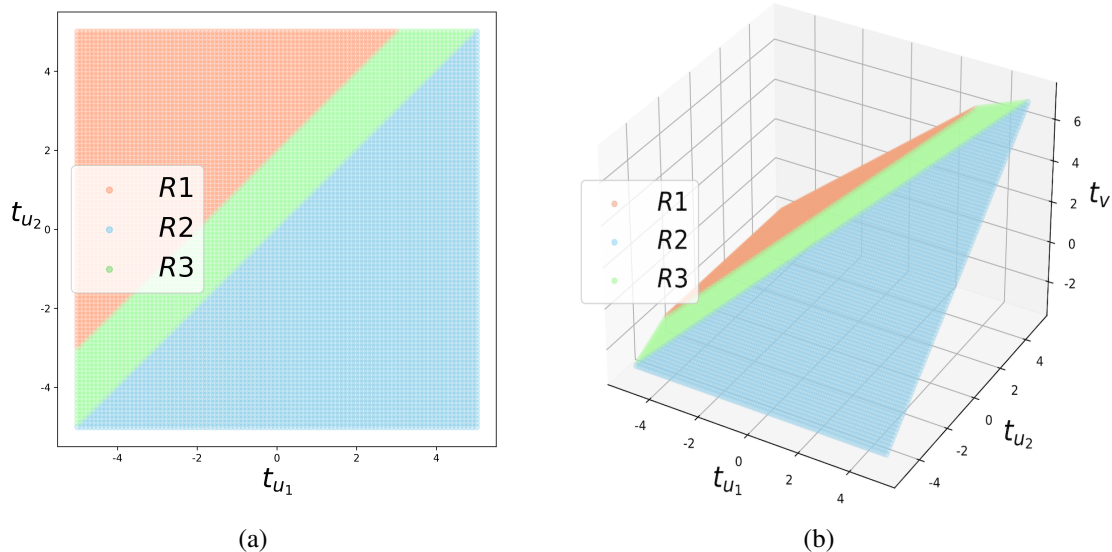


Figure 2: Illustration of Example 1. It shows that the output firing time $t_v(t_{u_1}, t_{u_2})$ as a function of inputs t_{u_1}, t_{u_2} is a CPWL mapping. (a) An illustration of the partitioning of the input space into three different regions. (b) Each region is associated with an affine-linear mapping.

Already this simple setting with two-dimensional inputs provides crucial insights. The actual number of linear regions in the input domain corresponds to the parameter of the LSNN neuron v . In particular, the maximum number of linear regions, i.e. three, can only occur if both weights w_{u_iv} are positive. Similarly, v does not fire at all if both weights are non-positive, which motivates the condition in Assumption II. The exact number of linear regions depends on the sign and magnitude of the weights. Furthermore, note that the linear regions are described by hyperplanes of the form

$$t_{u_2} \lesseqgtr t_{u_1} + C_{p,u}, \quad (12)$$

where $C_{p,u}$ is a constant depending on the parameter p corresponding to v , i.e., threshold, delays and weights, and the actual input neuron(s) causing v to spike. Hence, p has only a limited effect on the boundary of a linear region; depending on their exact value, the parameter only introduces an additive constant shift.

Remark 20 Dropping the assumption that δ is arbitrarily large in (3) yields an evolved model that is also biologically more realistic. The magnitude of δ describes the duration in which an incoming spike influences the membrane potential of a neuron. By setting δ arbitrarily large, we

generally consider an incoming spike to have a lasting effect on the membrane potential. Specifying a fixed δ increases the importance of the timing of the individual spikes as well as the choice of the parameter. For instance, inputs from certain regions in the input domain may not trigger a spike since the combined effect of multiple delayed incoming spikes is neglected. An in-depth analysis of the influence of δ is left as future work and we continue our analysis under the assumption that δ is arbitrarily large.

A.2.2. SPIKING NEURON WITH ARBITRARILY MANY INPUTS

A significant observation in the two-dimensional case is that the firing time $t_v(t_{u_1}, t_{u_2})$ as a function of the input t_{u_1}, t_{u_2} is a CPWL mapping. Indeed, each linear region is associated with an affine linear mapping and crucially these affine mappings agree at the breakpoints. This intuitively makes sense since a breakpoint marks the event when the effect of an additional neuron on the firing time of v needs to be taken into consideration or, equivalently, a neuron does not contribute to the firing of v anymore. However, in both circumstances, the effective contribution of this specific neuron is zero (and the contribution of the other neuron remains unchanged) at the breakpoint so that the crossing of a breakpoint and the associated change of a linear region does not result in a discontinuity.

Formally, the class of CPWL functions describes functions that are globally continuous and locally linear on each polytope in a given finite decomposition of \mathbb{R}^d into polytopes. We refer to the polytopes as linear regions. The insights obtained in the two-dimensional case do not straightforwardly extend to a d -dimensional input domain for $d > 2$. Crucially, continuity may be lost as the following simple example shows.

Example 2 *Let v be an LSNN neuron with threshold $\theta_v = 1$ and presynaptic neurons u_1, u_2, u_3 with corresponding weights $w_{u_1v} = 1, w_{u_2v} = -1, w_{u_3v} = 1$, delays $d_{u_1v} = d_{u_2v} = d_{u_3v} = 0$, and firing times $t_{u_1} = 0, t_{u_2} = 1 + \varepsilon, t_{u_3} = 2$, respectively. One easily verifies via (7) that*

$$t_v(t_{u_1}, t_{u_2}, t_{u_3}) = \begin{cases} 1, & \text{for } \varepsilon \geq 0 \\ 2 - \varepsilon, & \text{for } \varepsilon < 0 \end{cases}.$$

Hence, $t_v(t_{u_1}, t_{u_2}, t_{u_3})$ is not continuous since

$$\lim_{\varepsilon \uparrow 0} t_v(t_{u_1}, t_{u_2}, t_{u_3}) = \lim_{\varepsilon \uparrow 0} 2 - \varepsilon = 2 \neq 1 = \lim_{\varepsilon \downarrow 0} t_v(t_{u_1}, t_{u_2}, t_{u_3}).$$

However, we can show that an LSNN neuron generates a PWL mapping, which under certain conditions on its weights is in fact continuous.

Lemma 21 *Let v be a LSNN neuron with threshold $\theta_v > 0$ and presynaptic neurons u_1, \dots, u_d with corresponding weights $w_{u_iv} \in \mathbb{R}$, delays $d_{u_iv} \geq 0$, and firing times $t_{u_i} \in \mathbb{R}$, respectively. Then the firing time $t_v(t_{u_1}, \dots, t_{u_d})$ as a function of the firing times t_{u_1}, \dots, t_{u_d} is a PWL mapping, and additionally continuous provided that*

$$w_{u_iv} + \sum_{j:w_{u_jv} \leq 0} w_{u_jv} > 0 \quad \text{for all } i \text{ with } w_{u_iv} > 0. \quad (13)$$

Proof Recall that we operate under Assumption II, i.e., we presuppose that $\sum_{i=1}^d w_{u_iv} > 0$ so that any input firing time $(t_{u_1}, \dots, t_{u_d}) \in \mathbb{R}^d$ necessarily triggers a firing in v . In particular, the notion

of t_v as a PWL mapping on \mathbb{R}^d is well-defined. Moreover, for given t_{u_1}, \dots, t_{u_d} we can identify a subset $I \subset \{1, \dots, d\}$ such that all u_i with $i \in I$ contribute to the firing of v whereas spikes from u_j with $j \in I^c = \{1, \dots, d\} \setminus I$ do not influence the firing of v (since these spikes arrive after v already fired). Then $\sum_{i \in I} w_{u_i v}$ is required to be positive, so that by (7) and (8) the following holds:

$$t_{u_k} + d_{u_k v} \geq t_v = \frac{\theta_v}{\sum_{i \in I} w_{u_i v}} + \sum_{i \in I} \frac{w_{u_i v}}{\sum_{j \in I} w_{u_j v}} (t_{u_i} + d_{u_i v}) \quad \text{for all } k \in I^c \quad (14)$$

and

$$t_{u_k} + d_{u_k v} < t_v = \frac{\theta_v}{\sum_{i \in I} w_{u_i v}} + \sum_{i \in I} \frac{w_{u_i v}}{\sum_{j \in I} w_{u_j v}} (t_{u_i} + d_{u_i v}) \quad \text{for all } k \in I. \quad (15)$$

Rewriting yields

$$t_{u_k} \geq \frac{\theta_v}{\sum_{i \in I} w_{u_i v}} + \sum_{i \in I} \frac{w_{u_i v}}{\sum_{j \in I} w_{u_j v}} (t_{u_i} + d_{u_i v}) - d_{u_k v} \quad \text{for all } k \in I^c \quad (16)$$

and

$$t_{u_k} \begin{cases} < \frac{\theta_v}{\sum_{j \in I \setminus k} w_{u_j v}} + \sum_{i \in I \setminus k} \frac{w_{u_i v}}{\sum_{j \in I \setminus k} w_{u_j v}} (t_{u_i} + d_{u_i v}) - d_{u_k v}, & \text{if } \frac{\sum_{i \in I \setminus k} w_{u_i v}}{\sum_{i \in I} w_{u_i v}} > 0 \\ > \frac{\theta_v}{\sum_{j \in I \setminus k} w_{u_j v}} + \sum_{i \in I \setminus k} \frac{w_{u_i v}}{\sum_{j \in I \setminus k} w_{u_j v}} (t_{u_i} + d_{u_i v}) - d_{u_k v}, & \text{if } \frac{\sum_{i \in I \setminus k} w_{u_i v}}{\sum_{i \in I} w_{u_i v}} < 0 \end{cases} \quad \forall k \in I.$$

It is now clear that the firing time $t_v(t_{u_1}, \dots, t_{u_d})$ as a function of the input t_{u_1}, \dots, t_{u_d} is a piecewise linear mapping on polytopes decomposing \mathbb{R}^d . To show that the mapping is additionally continuous if (13) holds, we need to assess t_v on the breakpoints. Let $I, J \subset \{1, \dots, d\}$ be index sets corresponding to input neurons $\{u_i : i \in I\}, \{u_j : j \in J\}$ that cause v to fire on the input region $R^I \subset \mathbb{R}^d, R^J \subset \mathbb{R}^d$ respectively. Assume that it is possible to transition from R^I to R^J through a breakpoint $t^{I,J} = (t_{u_1}^{I,J}, \dots, t_{u_d}^{I,J}) \in \mathbb{R}^d$ without leaving $R^I \cup R^J$. Crossing the breakpoint is equivalent to the fact that the input neurons $\{u_i : i \in I \setminus J\}$ do not contribute to the firing of v anymore and the input neurons $\{u_i : i \in J \setminus I\}$ begin to contribute to the firing of v . Subsequently, we consider different cases concerning the relation of I and J . Thereby, we first require that all weights are positive.

Now, assume that $J \subset I$. Then, we observe that the breakpoint $t^{I,J}$ is necessarily an element of the linear region corresponding to the index set with smaller cardinality, i.e., $t^{I,J} \in R^J$. This is an immediate consequence of (15) and the fact that $t^{I,J}$ is characterized by

$$t_{u_k}^{I,J} + d_{u_k v} = t_v(t^{I,J}) \quad \text{for all } k \in I \setminus J. \quad (17)$$

Indeed, if $t_{u_k}^{I,J} + d_{u_k v} > t_v(t^{I,J})$, then there exists $\varepsilon_k > 0$ such that (16) also holds for $t_{u_k}^{I,J} \pm \varepsilon$, where $0 \leq \varepsilon < \varepsilon_k$, i.e., a small change in $t_{u_k}^{I,J}$ is not sufficient to change the corresponding linear region, contradicting our assumption that $t^{I,J}$ is a breakpoint.

The firing time $t_v(t^{I,J})$ is explicitly given by

$$t_v(t^{I,J}) = \frac{\theta_v}{\sum_{i \in J} w_{u_i v}} + \sum_{i \in J} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}} (t_{u_i}^{I,J} + d_{u_i v})$$

Using (17), we obtain

$$0 = -\frac{w_{u_k v}}{\sum_{j \in J} w_{u_j v}} (t_v(t^{I,J}) - (t_{u_k}^{I,J} + d_{u_k v})) \quad \text{for all } k \in I \setminus J$$

so that

$$t_v(t^{I,J}) = \frac{\theta_v}{\sum_{i \in J} w_{u_i v}} + \sum_{i \in J} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}} (t_{u_i}^{I,J} + d_{u_i v}) - \sum_{i \in I \setminus J} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}} (t_v(t^{I,J}) - (t_{u_i}^{I,J} + d_{u_i v})).$$

Solving for $t_v(t^{I,J})$ yields

$$\begin{aligned} t_v(t^{I,J}) &= \left(1 + \sum_{i \in I \setminus J} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}}\right)^{-1} \cdot \left(\frac{\theta_v}{\sum_{i \in J} w_{u_i v}} + \sum_{i \in I} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}} (t_{u_i}^{I,J} + d_{u_i v})\right) \\ &= \sum_{i \in J} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}} \cdot \left(\frac{\theta_v}{\sum_{i \in J} w_{u_i v}} + \sum_{i \in I} \frac{w_{u_i v}}{\sum_{j \in J} w_{u_j v}} (t_{u_i}^{I,J} + d_{u_i v})\right) \\ &= \frac{\theta_v}{\sum_{i \in I} w_{u_i v}} + \sum_{i \in I} \frac{w_{u_i v}}{\sum_{j \in I} w_{u_j v}} (t_{u_i}^{I,J} + d_{u_i v}), \end{aligned}$$

which is exactly the expression for the firing time on R^I . This shows that $t_v(t_{u_1}, \dots, t_{u_d})$ is continuous in $t^{I,J}$. Since the breakpoint $t^{I,J}$ was chosen arbitrarily, $t_v(t_{u_1}, \dots, t_{u_d})$ is continuous at any breakpoint.

The case $I \subset J$ follows analogously. It remains to check the case when neither $I \subset J$ nor $J \subset I$. To that end, let $i^* \in I \setminus J$ and $j^* \in J \setminus I$. Assume without loss of generality that $t^{I,J} \in R^I$ so that (14) and (15) imply

$$t_{u_{i^*}}^{I,J} + d_{u_{i^*} v} < t_v(t^{I,J}) \leq t_{u_{j^*}}^{I,J} + d_{u_{j^*} v}.$$

Hence, there exists $\varepsilon > 0$ such that

$$t_{u_{i^*}}^{I,J} + d_{u_{i^*} v} < t_{u_{j^*}}^{I,J} + d_{u_{j^*} v} - \varepsilon. \quad (18)$$

Moreover, due to the fact that $t^{I,J}$ is a breakpoint we can find $t^J \in R^J \cap \mathcal{B}(t^{I,J}; \frac{\varepsilon}{3})$, where $\mathcal{B}(t^{I,J}; \frac{\varepsilon}{3})$ denotes the open ball with radius $\frac{\varepsilon}{3}$ centered at $t^{I,J}$. In particular, this entails that

$$-\frac{\varepsilon}{3} < (t_{u_{i^*}}^J - t_{u_{i^*}}^{I,J}), (t_{u_{j^*}}^J - t_{u_{j^*}}^{I,J}) < \frac{\varepsilon}{3},$$

and therefore together with (18)

$$\begin{aligned} t_{u_{i^*}}^J + d_{u_{i^*} v} - (t_{u_{j^*}}^J + d_{u_{j^*} v}) &= (t_{u_{i^*}}^J - t_{u_{i^*}}^{I,J}) + (t_{u_{i^*}}^{I,J} + d_{u_{i^*} v} - (t_{u_{j^*}}^{I,J} + d_{u_{j^*} v})) + (t_{u_{j^*}}^{I,J} - t_{u_{j^*}}^J) \\ &< 0, \quad \text{i.e., } t_{u_{i^*}}^J + d_{u_{i^*} v} < t_{u_{j^*}}^J + d_{u_{j^*} v}. \end{aligned}$$

However, (14) and (15) require that

$$t_{u_{j^*}}^J + d_{u_{j^*} v} < t_v(t^J) \leq t_{u_{i^*}}^J + d_{u_{i^*} v}$$

since $t^J \in R^J$. Thus, $t^{I,J}$ can not exist, and the case when neither $I \subset J$ nor $J \subset I$ can not arise.

Finally, we observe that the previous analysis remains valid when dropping the positivity assumption on the weights, with the only exception being (17). In particular, for negative weights the behaviour demonstrated in Example 2 may arise so that (17) is not valid in general anymore. However, by restricting the feasible weights, i.e., prohibiting negative weights of large magnitude via the condition in (13), we ensure that (17) holds and the statement follows. ■

Remark 22 *The observations about the parameter δ in Remark 20 directly transfer from the two- to the d -dimensional setting. Additionally, note that (13) is a necessary and sufficient condition for the firing time of an LSNN neuron to be continuous in the input firing times. However, the corresponding condition for a whole multi-layer LSNN as provided in (9) is sufficient but not necessary.*

We want to highlight some similarities and differences between two- and d -dimensional inputs. In both cases, the actual number of linear regions depends on the choice of parameter, in particular, the synaptic weights. Thereby, the number of linear regions scales at most as $2^d - 1$ in the input dimension d of an LSNN neuron, and the number is indeed attained if all weights are positive (as observed in Lemma 13). However, the d -dimensional case allows for more flexibility in the structure of the linear regions. Recall that in the two-dimensional case, the boundary of any linear region is described by hyperplanes of the form (12). This does not hold if $d > 2$, see e.g. (16). Here, the weights also affect the shape of the linear region. Refining the connection between the boundaries of a linear region, its response function, and the specific choice of parameter requires further considerations. Similarly, obtaining a non-trivial upper bound on the number of linear regions for networks of LSNN neurons is not straightforward as the following example shows.

Example 3 *Let Φ be a one-layer LSNN with d_{in} input neurons and d_{out} output neurons. Via Proposition 13, we certainly can upper bound the number of linear regions generated by Φ by $(2^{d_{in}} - 1)^{d_{out}}$, i.e., the product of the number of linear regions generated by each output neuron. Unfortunately, the bound is far from optimal. Consider the case when $d_{in} = d_{out} = 2$. Then, the structure of the linear regions generated by the individual output neurons is given in (12). In particular, the boundary of the linear regions is described by a set of specific hyperplanes with a common normal vector, where the parameters of Φ only induce a shift of the hyperplanes. In other words, the hyperplanes separating the linear regions are parallel. Hence, each output neuron generates at most two parallel hyperplanes yielding three linear regions (see Figure 2). The number of linear regions generated by Φ is therefore given by the number of regions four parallel hyperplanes can decompose the input domain into, i.e., at most $5 < 9 = (2^{d_{in}} - 1)^{d_{out}}$.*

Finally, under even stronger conditions, i.e., all weights are positive, we can further characterize the firing map of an LSNN neuron.

Lemma 23 *Let v be a LSNN neuron with threshold $\theta_v > 0$ and presynaptic neurons u_1, \dots, u_d with corresponding weights $w_{u_i v} > 0$, delays $d_{u_i v} \geq 0$, and firing times $t_{u_i} \in \mathbb{R}$, respectively. Then the firing time $t_v(t_{u_1}, \dots, t_{u_d})$ as a function of the firing times t_{u_1}, \dots, t_{u_d} is a increasing and concave function. Moreover, the firing time of v is given by*

$$t_v(t_{u_1}, \dots, t_{u_d}) = \inf_{\emptyset \neq I \subset [d]} \left\{ s^I = \frac{1}{\sum_{i \in I} w_{u_i v}} (\theta_v + \sum_{i \in I} w_{u_i v} (t_{u_i} + d_{u_i v})) : s^I > \max_{i \in I} t_{u_i} \right\}. \quad (19)$$

Proof First, we show that the expression

$$s^*(t_{u_1}, \dots, t_{u_d}) = \inf_{\emptyset \neq I \subset [d]} \left\{ s^I = \frac{1}{\sum_{i \in I} w_{u_i v}} (\theta_v + \sum_{i \in I} w_{u_i v} (t_{u_i} + d_{u_i v})) : s^I > \max_{i \in I} t_{u_i} \right\}$$

is equivalent to the firing time $t_v(t_{u_1}, \dots, t_{u_d})$ of an LSNN neuron v . By (8) we immediately observe that

$$t_v(t_{u_1}, \dots, t_{u_d}) = \frac{1}{\sum_{i \in F} w_{u_i v}} (\theta_v + \sum_{i \in F} w_{u_i v} (t_{u_i} + d_{u_i v}))$$

for some $F \subset [d]$ such that

$$\max_{i \in F} t_{u_i} < t_v(t_{u_1}, \dots, t_{u_d}) \leq \min_{i \in [d] \setminus F} t_{u_i},$$

i.e., $t_v(t_{u_1}, \dots, t_{u_d}) \geq s^*(t_{u_1}, \dots, t_{u_d})$. Moreover, set

$$s^J = \frac{1}{\sum_{i \in J} w_{u_i v}} (\theta_v + \sum_{i \in J} w_{u_i v} (t_{u_i} + d_{u_i v})) \quad \text{for } J \subset [d]$$

and assume that $s^J > t_{u_j}$ for all $j \in J$. If $J \cap F^C \neq \emptyset$, then for some $k \in J \cap F^C$ we have $s^J > t_{u_k} \geq t_v(t_{u_1}, \dots, t_{u_d})$. Hence, assume that $J \cap F^C = \emptyset$, i.e., $J \subset F$. Then, either $s^J = t_v(t_{u_1}, \dots, t_{u_d})$ (if $F = J$) or (for $F \neq J$) we get

$$\begin{aligned} \sum_{i \in J} w_{u_i v} (s^J - t_{u_i} - d_{u_i v}) &= \theta = \sum_{i \in F} w_{u_i v} (t_v(t_{u_1}, \dots, t_{u_d}) - t_{u_i} - d_{u_i v}) \\ &= \sum_{i \in J} w_{u_i v} (t_v(t_{u_1}, \dots, t_{u_d}) - t_{u_i} - d_{u_i v}) + \sum_{i \in F \setminus J} w_{u_i v} (t_v(t_{u_1}, \dots, t_{u_d}) - t_{u_i} - d_{u_i v}) \\ &> \sum_{i \in J} w_{u_i v} (t_v(t_{u_1}, \dots, t_{u_d}) - t_{u_i} - d_{u_i v}), \quad \text{i.e., } s^J > t_v(t_{u_1}, \dots, t_{u_d}). \end{aligned}$$

Therefore $s^J \geq t_v(t_{u_1}, \dots, t_{u_d})$ so that $t_v(t_{u_1}, \dots, t_{u_d}) \leq s^*(t_{u_1}, \dots, t_{u_d})$ and (19) follows.

Next, we show that t_v is an increasing function. Let $\varepsilon \in \mathbb{R}^d$ such that $\varepsilon_i \geq 0$ for all $i = 1, \dots, d$. Denote the potential of v for inputs $(t_{u_1}, \dots, t_{u_d})$ and $(t_{u_1}, \dots, t_{u_d}) + \varepsilon$ by P_v and P_v^ε , respectively. Due to (7), we know that

$$\begin{aligned} \theta_v > P_v(t) &= \sum_{i \in [d]} w_{u_i v} \sigma(t - t_{u_i} - d_{u_i v}) \geq \sum_{i \in [d]} w_{u_i v} \sigma(t - t_{u_i} - \varepsilon_i - d_{u_i v}) \\ &= P_v^\varepsilon(t) \quad \text{for any } t < t_v(t_{u_1}, \dots, t_{u_d}). \end{aligned}$$

Hence, the potential $P_v^\varepsilon(t)$ does not reach θ_v for $t < t_v(t_{u_1}, \dots, t_{u_d})$. In other words, $t_v(t_{u_1} + \varepsilon_1, \dots, t_{u_d} + \varepsilon_d) \geq t_v(t_{u_1}, \dots, t_{u_d})$ so that t_v is indeed an increasing function.

Finally, we prove the concavity of t_v . Let $x, y \in \mathbb{R}^d$ denote two distinct firing times of u_1, \dots, u_d . Our goal is to show that

$$t_v(px + (1-p)y) \geq pt_v(x) + (1-p)t_v(y) \quad \text{for all } 0 < p < 1. \quad (20)$$

We already know that t_v is a CPWL function, i.e., t_v partitions \mathbb{R}^d into linear regions. Therefore, we consider the following possibilities: Either x and y are in the same or different linear regions. Since

the linear regions are defined by intersections of halfspaces (see (14) and (15)), they are convex so that in the former case it is immediate to verify that (20) holds. In the latter case, we need to further distinguish between two cases. Assume x and y are located in distinct linear regions R^I and R^J , respectively, whereby $I, J \subset [d]$ indicate the subsets of neurons u_1, \dots, u_d that trigger the firing of v on the given linear region. Then, either $px + (1-p)y \in R^I \cup R^J$ or $px + (1-p)y \in (R^I \cup R^J)^C$ for $0 < p < 1$. We will explicitly show the concavity of t_v for $px + (1-p)y \in R^I \cup R^J$, the other case follows along similar lines. Without loss of generality, we assume that $px + (1-p)y \in R^I$. Moreover, we denote the restriction of t_v to R^I and R^J by A^I and A^J , respectively. Since t_v is a CPWL function, A^I and A^J are affine functions so that $A^I(z^I) = (m^I)^T z^I + b^I$ and $A^J(z^J) = (m^J)^T z^J + b^J$ for $z^I \in R^I, z^J \in R^J$ and suitable parameter $m^I, m^J \in \mathbb{R}^d, b^I, b^J \in \mathbb{R}$. Using that A^I and A^J are affine, one derives that

$$A^I(px + (1-p)y) \geq pA^I(x) + (1-p)A^J(y)$$

is equivalent to

$$(m^I - m^J)^T y + (b^I - b^J) \geq 0, \quad (21)$$

i.e., verifying (21) suffices to obtain concavity of t_v . To that end, observe that m^I, b^I and m^J, b^J are determined by I and J , respectively, i.e., by the weights and delays associated with I and J via the firing time of v in (8). In particular, we have

$$m_i^I = \begin{cases} 0, & \text{if } i \notin I \\ \frac{w_{u_i v}}{\sum_{j \in I} w_{u_j v}}, & \text{if } i \in I \end{cases} \quad \text{and} \quad b^I = \frac{\theta_v + \sum_{i \in I} w_{u_i v} d_{u_i v}}{\sum_{j \in I} w_{u_j v}},$$

and the analogous expressions for m^J, b^J . Using these expressions and the properties of the firing time t_v one indeed obtains (21). \blacksquare

A.2.3. NETWORKS OF SPIKING NEURONS

Next, we want to extend the properties of an LSNN neuron to a network of LSNN neurons. Due to the layer-wise arrangement of neurons in an LSNN, this task is rather straightforward. Thus, the results in Theorem 9 and Proposition 10 follow.

Proof [of Theorem 9] In Lemma 21, we showed that the firing time of an LSNN neuron with arbitrarily many input neurons is a PWL function in the input firing times. Since Φ consists of LSNN neurons arranged in layers it immediately follows that the firing map of each layer is PWL. Thus, as a composition of PWL mappings t_Φ itself is PWL. Furthermore, (9) guarantees that (13) holds for each neuron in Φ , i.e., the firing time of each LSNN neuron continuously depends on its input firing times. Therefore, we conclude analogously as for the PWL property that t_Φ is continuous, i.e., t_Φ is a CPWL mapping, if (9) is satisfied. \blacksquare

As already indicated, the condition in Theorem 9 is not necessary to obtain a continuous firing map. Next, we specifically construct an LSNN demonstrating this observation.

Example 4 Consider a two-layer LSNN Φ defined by

$$W^1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{1}{2} \end{pmatrix}, W^2 = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \Theta^1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \Theta^2 = 1.$$

Without loss of generality, we assume the delays to be zero. We show that despite the negative weight, which violates (9), $t_\Phi(t_1, t_2)$ is a CPWL function for the input firing times $t_{u_1}, t_{u_2} \in \mathbb{R}$. Denote the input neurons by u_1, u_2 and the neurons in the hidden layer by v_1, v_2, v_3 . Note that the firing times of the neurons in the hidden layer depend on either u_1 or u_2 . Via (8), the firing times of the neurons in the hidden layer are

$$t_{v_1} = 1 + t_{u_1}, \quad t_{v_2} = 1 + t_{u_2}, \quad t_{v_3} = 2 + t_{u_2}.$$

Similarly, via (8), the firing time of the output neuron is

$$t_\Phi(t_{u_1}, t_{u_2}) = \begin{cases} 2 + t_{u_1}, & \text{if } t_{v_1} < t_{v_2} \\ 2 + t_{u_2}, & \text{if } t_{v_2} < t_{v_1} \\ \frac{3}{2} + \frac{1}{2}(t_{u_1} + t_{u_2}), & \text{if } |t_{v_1} - t_{v_2}| \leq 1 \end{cases}.$$

Hence, the breakpoints $(t_{u_1}^b - t_{u_2}^b) \in \mathbb{R}^2$ of the linear regions are determined by

$$|t_{u_1}^b - t_{u_2}^b| = 1 \Leftrightarrow t_{u_1}^b = \begin{cases} t_{u_2}^b + 1, & \text{if } t_{u_1}^b > t_{u_2}^b \\ t_{u_2}^b - 1, & \text{if } t_{u_1}^b < t_{u_2}^b \end{cases}.$$

Evaluating the firing time of the output neuron at the breakpoints gives

$$t_\Phi(t_{u_1}^b, t_{u_2}^b) = \begin{cases} 2 + t_{u_2}^b, & \text{if } t_{u_1}^b > t_{u_2}^b \\ 2 + t_{u_1}^b, & \text{if } t_{u_1}^b < t_{u_2}^b \end{cases},$$

which shows that t_Φ is indeed continuous.

With the same arguments as in the proof of Theorem 9, we also show Proposition 10.

Proof [of Proposition 10] First, the alternative expression for the firing time of an LSNN neuron with only positive weights is proven in Lemma 23. Hence, it is left to show that t_Φ is increasing and concave under the given condition. However, this is a direct consequence of Lemma 23 and the structure of Φ . In particular, one establishes that t_Φ is increasing as a composition of strictly increasing maps and concludes that t_Φ is also concave since the composition of non-decreasing and concave functions is again concave. \blacksquare

Finally, we construct specific LSNNs with locally decreasing and non-concave firing maps, highlighting the need for the positive weights condition.

Example 5 Let v be an LSNN neuron with threshold $\theta_v = 1$ and presynaptic neurons u_1, u_2, u_3 with corresponding weights $w_{u_1v} = -\frac{1}{2}, w_{u_2v} = 1, w_{u_3v} = 1$, delays $d_{u_1v} = d_{u_2v} = d_{u_3v} = 0$. Additionally, let $t_a = (0, 1, 1)^T$ and $t_b = (\frac{1}{2}, 1, 1)^T$ be two data points corresponding to the firing times of input neurons. Via (8), one can directly verify that $t_v(t_a) > t_v(t_b)$ when $t_a < t_b$, i.e., t_v is not increasing. For the non-concave part, let $t_x = (\frac{1}{2}, 0, 0)^T$ and $t_y = (\frac{1}{2}, 1, 1)^T$ as two data points corresponding to the firing times of input neurons. Setting $p = \frac{1}{2}$ and again via (8), one can directly verify that

$$pt_v(t_x, t_y) + (1 - p)t_v(t_x, t_y) \geq t_v(pt_x + (1 - p)t_y).$$

A.3. Realizing ReLU with LSNNs

In this section, we show that LSNNs can realize the ReLU function. To that end, we first analyze the ability of an LSNN neuron to express certain (simple) piecewise functions via its firing map.

Proposition 24 *Let $c_1, c_2 \in \mathbb{R}$, $c_3 \in (a, b) \subset \mathbb{R}$ and consider $f_1, f_2 : [a, b] \rightarrow \mathbb{R}$ given by*

$$f_1(x) = \begin{cases} x + c_1 & , \text{ if } x > c_3 \\ c_2 & , \text{ if } x < c_3 \end{cases} \quad \text{or} \quad f_2(x) = \begin{cases} x + c_1 & , \text{ if } x < c_3 \\ c_2 & , \text{ if } x > c_3 \end{cases},$$

where we do not fix the value at the breakpoint $x = c_3$. There does not exist an LSNN neuron v with input neuron u_1 such that $t_v(x) = f(x)$ on $[a, b]$, where $f \in \{f_1, -f_1, f_2\}$ and $t_v(x)$ denotes the firing time of v on input $t_{u_1} = x$.

Proof If u_1 is the only input neuron, then v fires if and only if $w_{u_1v} > 0$ and by (8) the firing time is given by

$$t_v(x) = \frac{\theta}{w_{u_1v}} + x + d_{u_1v} \quad \text{for all } x \in [a, b],$$

i.e., $t_v \neq f$. Therefore, we introduce auxiliary input neurons u_2, \dots, u_n and assume without loss of generality that $t_{u_i} + d_{u_iv} < t_{u_j} + d_{u_jv}$ for $j > i$. Here, the firing times t_{u_i} , $i = 2, \dots, n$, are suitable constants. We will show that even in this extended setting $t_v \neq f$ still holds and thereby also the claim.

For the sake of contradiction, assume that $t_v(x) = f_1(x)$ for all $x \in [a, b]$. This implies that there exists an index set $J \subset \{1, \dots, n\}$ with $\sum_{j \in J} w_{u_jv} > 0$ and a corresponding non-empty interval $(a_1, c_3) \subset [a, b]$ such that

$$c_2 = t_v(x) = \frac{1}{\sum_{i \in J} w_{u_iv}} \left(\theta_v + \sum_{i \in J} w_{u_iv} (t_{u_i} + d_{u_iv}) \right) \quad \text{for all } x \in (a_1, c_3), \quad (22)$$

where we have applied (8). Note that J is of the form $J = \{1, \dots, \ell\}$ for some $\ell \in \{2, \dots, n\}$ because $(t_{u_i} + d_{u_iv})_{i=2}^n$ is in ascending order, i.e., if the spike from u_ℓ has reached v before v fired, then so did the spikes from u_i , $2 \leq i < \ell$. In particular, we immediately observe that $1 \notin J$ since otherwise, due to the contribution from u_1 , t_v is non-constant on (a_1, c_3) , i.e., $t_v \neq c_3$ on (a_1, c_3) . Hence, the spike from u_1 with firing time $x \in (a_1, c_3)$ necessarily reaches v after the subset of neurons specified by J already caused v to fire. Therefore, we have

$$x + d_{u_1v} \geq t_v(x) = c_2 \quad \text{for all } x \in (a_1, c_3).$$

However, it immediately follows for any $y \geq c_3$ that

$$y + d_{u_1v} > x + d_{u_1v} \geq c_2 \quad \text{for all } x \in (a_1, c_3).$$

Thus, we know that a spike from u_1 with $t_{u_1} > c_3$ does not reach v before it emits a spike unless there are additional incoming spikes in v from neurons $\{u_i : i \in I \subset \{2, \dots, n\} \setminus J = \{\ell + 1, \dots, n\}\}$, which delay its firing. By the same reasoning as before, we conclude that $t_{u_{\ell+1}} + d_{u_{\ell+1}v} \geq c_2$ because otherwise (22) would be violated since spikes from neurons $\{u_i : i \in I\}$ contribute to the firing of v on (a_1, c_3) . Consequently, the firing of v can not be delayed, i.e.,

$t_v(x) = c_2$ for all $x \in (a_1, b]$, which contradicts $t_v = f_1$ on $[a, b]$. The same reasoning can be applied to derive that $-f_1$ can not be emulated by the firing map of an LSNN neuron.

We perform a similar analysis to show that f_2 can not be emulated. For the sake of contradiction, assume that $t_v(x) = f_2(x)$ for all $x \in [a, b]$. This implies that there exists an index set $I \subset \{1, \dots, n\}$ with $\sum_{i \in I} w_{u_i v} > 0$ and a corresponding non-empty interval $(a_1, c_3) \subset [a, b]$ such that

$$x + c_1 = t_v(x) = \frac{1}{\sum_{i \in I} w_{u_i v}} \left(\theta_v + w_{u_1 v}(x + d_{u_1 v}) + \sum_{i \in I \setminus \{1\}} w_{u_i v}(t_{u_i} + d_{u_i v}) \right) \quad \text{for } x \in (a_1, c_3), \quad (23)$$

where we have applied (8). Note that $1 \in I$ necessarily needs to hold, since otherwise t_v is constant on (a_1, c_3) . Hence, I is of the form $I = \{1, \dots, \ell\}$ for some $\ell \in \{1, \dots, n\}$. To satisfy $t_v(x) = f_2(x)$ for all $x \in [a, b]$, there additionally needs to exist an index set $J \subset \{1, \dots, n\}$ with $\sum_{j \in J} w_{u_j v} > 0$ and a corresponding non-empty interval $(c_3, b_1) \subset [a, b]$ such that $t_v = c_2$ on (c_3, b_1) . We conclude that $J = \{1, \dots, m\}$ or $J = \{2, \dots, m\}$ for some $m \in \{1, \dots, n\}$. In the former case, t_v is non-constant on (c_3, b_1) (due to the contribution from u_1), i.e., $t_v \neq c_2$ on (c_3, b_1) . Hence, it remains to consider the latter case. First, note that $c_1 \leq 0$ is not a admissible choice since (23) implies that for $x \in (a_1, c_3)$

$$t_v(x) = x + c_1 \leq x,$$

i.e., the spike from u_1 does not reach v before its firing, which contradicts the construction. Thus, there exists some $0 < \varepsilon < c_1$ such that $c_3 - \varepsilon \in (a_1, c_3)$ and

$$t_v(c_3 - \varepsilon) = c_3 - \varepsilon + c_1 > c_3.$$

This particularly entails that no spikes from neurons $\{u_j : j \in J \setminus I\}$ reach v before time $c_3 - \varepsilon + c_1$, i.e., $t_{u_{\ell+1}} + d_{u_{\ell+1}} \geq c_3 - \varepsilon + c_1 > c_3$. Therefore, we derive that $m \leq \ell$ needs to hold since otherwise u_1 with $t_{u_1} = x \in (c_3, c_3 - \varepsilon + c_1)$ contributes to the firing of v , contradicting the fact that t_v is constant on (c_3, b_1) . Moreover, $m = \ell$, i.e., $J = I \setminus \{1\}$, is not valid because either J is empty or by comparing the coefficients of x in (23) we find that

$$\frac{w_{u_1 v}}{\sum_{i \in I} w_{u_i v}} = 1 \Leftrightarrow \sum_{i \in I \setminus \{1\}} w_{u_i v} = 0, \quad (24)$$

which implies the contradiction

$$0 = \sum_{i \in I \setminus \{1\}} w_{u_i v} = \sum_{j \in J} w_{u_j v} > 0.$$

Finally, $m < \ell$ is also not feasible. This follows from (24) via the observation that $w_{u_1 v} > 0$, i.e., the contribution from u_1 to the potential of v is necessarily positive if the associated spike arrives in time. Consequently, when u_1 stops contributing to the firing of v on (c_3, b_1) the firing time t_v did not decrease (since by the assumption $m < \ell$ also no further incoming spikes arrive). Hence, the spikes from u_2, \dots, u_ℓ contribute to the firing of v on any input $x \in (c_3, b_1)$ contradicting $m < \ell$. Thus, J can not exist and thereby $t_v = f_2$ on $[a, b]$ can not be achieved. \blacksquare

Next, we show that $-f_2$ as defined in Proposition 24 can indeed be emulated by the firing map of an LSNN neuron under suitable conditions.

Proposition 25 Let $c_1, c_2 \in \mathbb{R}$, $c_3 \in (a, b) \subset \mathbb{R}$ and consider $f : [a, b] \rightarrow \mathbb{R}$

$$f(x) = \begin{cases} -x + c_1 & , \text{ if } x < c_3 \\ c_2 & , \text{ if } x > c_3 \end{cases},$$

where we do not fix the value at the breakpoint $x = c_3$. There exists a one-layer LSNN Φ with output neuron v and input neuron u_1 such that $t_v(x) = f(x)$ on $[a, b]$, where $t_v(x)$ denotes the firing time of v on input $t_{u_1} = x$, if and only if $c_1 - c_3 = c_2$ as well as $c_1 \geq 2c_3$.

Proof First, we show that t_v can not emulate f if the conditions $c_1 - c_3 = c_2$ and $c_1 \geq 2c_3$ are not met. The argument is essentially the same as in the proof of Proposition 24 and we only sketch the main steps. Assuming that $t_v(x) = f(x)$ for all $x \in [a, b]$ implies that there exists an index set $I = \{1\} \cup \{2, \dots, \ell\}$ for some $\ell \in \{2, \dots, n\}$ with $\sum_{i \in I} w_{u_i v} > 0$ and a corresponding non-empty interval $(a_1, c_3) \subset [a, b]$ such that

$$-x + c_1 = t_v(x) = \frac{1}{\sum_{i \in I} w_{u_i v}} \left(\theta_v + w_{u_1 v}(x + d_{u_1 v}) + \sum_{i \in I \setminus \{1\}} w_{u_i v}(t_{u_i} + d_{u_i v}) \right) \quad \text{for } x \in (a_1, c_3), \quad (25)$$

where we have applied (8). Due to $1 \in I$, we obtain

$$-x + c_1 = t_v(x) > x + d_{u_1 v} \Leftrightarrow c_1 > 2x + d_{u_1 v} \quad \text{for } x \in (a_1, c_3), \quad (26)$$

and in particular $c_1 \geq 2c_3$ needs to be satisfied. Hence, for $\varepsilon > 0$ such that $c_3 - \varepsilon \in (a_1, c_3)$ we find via (25) that

$$t_v(c_3 - \varepsilon) = -c_3 + \varepsilon + c_1 \geq c_3 + \varepsilon > c_3. \quad (27)$$

Moreover, to satisfy $t_v(x) = f_2(x)$ for all $x \in [a, b]$ to hold, there additionally needs to exist an index set $J = \{2, \dots, m\}$ for some $m \in \{2, \dots, n\}$ with $\sum_{j \in J} w_{u_j v} > 0$ and a corresponding non-empty interval $(c_3, b_1) \subset [a, b]$ such that $t_v = c_2$ on (c_3, b_1) . Due to (27) we conclude that no spikes from neurons $\{u_j : j \in J \setminus I\}$ reach v before time c_3 so that $m \leq \ell$ needs to be satisfied. Finally, assuming that $c_1 - c_3 \neq c_2$ entails that there is a jump at c_3 in f , i.e., f is discontinuous at c_3 . However, a jump discontinuity requires an incoming spike that delays the firing of v for $t_{u_1} = c_3$, which we already excluded. Thus, $t_v(x) = f(x)$ for all $x \in [a, b]$ can not be achieved given that $c_1 - c_3 \neq c_2$.

For the reverse direction, we will explicitly construct an LSNN neuron v that emulates f if the conditions $c_1 - c_3 = c_2$ and $c_1 \geq 2c_3$ are fulfilled. We introduce an auxiliary input neuron with constant firing time $t_{u_2} \in \mathbb{R}$ and specify the parameter of $\Phi = ((W, D, \Theta))$ in the following manner (see Figure 3a):

$$W = \begin{pmatrix} -\frac{1}{2} \\ 1 \end{pmatrix}, D = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \Theta = \theta,$$

where $\theta, d_1, d_2 > 0$ are to be specified. Note that either u_2 or u_1 together with u_2 can trigger a spike in v since $w_{u_1 v} < 0$. Therefore, applying (8) yields that u_2 triggers a spike in v under the following circumstances:

$$t_v(x) = \theta + t_{u_2} + d_2 \quad \text{if } t_v(x) \leq t_{u_1} + d_1 = x + d_1.$$

Hence, this case only arises when

$$\theta + t_{u_2} + d_2 \leq x + d_1 \Leftrightarrow \theta + t_{u_2} + d_2 - d_1 \leq x.$$

To emulate f the parameters need to satisfy

$$\theta + t_{u_2} + d_2 - d_1 \leq x \text{ for all } x \in (c_3, b] \quad \text{and} \quad \theta + t_{u_2} + d_2 - d_1 > x \text{ for all } x \in [a, c_3)$$

which simplifies to

$$\theta + t_{u_2} + d_2 - d_1 = c_3. \quad (28)$$

If the additional condition

$$\theta + t_{u_2} + d_2 = c_2 \quad (29)$$

is met, we can infer that for $d_1 = c_2 - c_3$ (which is a valid choice due to $c_2 - c_3 = c_1 - 2c_3 \geq 2c_3 - 2c_3 = 0$)

$$t_v(x) = \begin{cases} 2(\theta + t_{u_2} + d_2) - (x + d_1) & , \text{ if } x < c_3 \\ \theta + t_{u_2} + d_2 & , \text{ if } x \geq c_3 \end{cases} = \begin{cases} -x + c_1 & , \text{ if } x < 0 \\ c_2 & , \text{ if } x \geq 0 \end{cases}.$$

Finally, it is immediate to verify that the conditions (28) and (29) can be satisfied simultaneously by choosing $d_2 = d_1$ and $t_{u_2} = c_3 - \theta$. ■

Having established the capability or inability to emulate certain simple affine functions by the firing map of an LSNN neuron, we now turn to their realizations. To that end, recall that to realize a function $f : [a, b] \rightarrow \mathbb{R}$ we focus on encoding schemes of the type $T_{\text{in}}/T_{\text{out}} \pm \cdot$. Therefore, for an LSNN neuron v with input neuron u_1 and $x \in [a, b]$ we distinguish the following encoding schemes:

$$\begin{aligned} \text{a)} & \quad \begin{cases} t_{u_1} & = T_{\text{in}} + x =: z \\ t_v(z) & = T_{\text{out}} + \mathcal{R}_v(x) \Leftrightarrow \mathcal{R}_v(x) = -T_{\text{out}} + t_v(z) \end{cases} \\ \text{b)} & \quad \begin{cases} t_{u_1} & = T_{\text{in}} - x =: z \\ t_v(z) & = T_{\text{out}} - \mathcal{R}_v(x) \Leftrightarrow \mathcal{R}_v(x) = T_{\text{out}} - t_v(z) \end{cases} \\ \text{c)} & \quad \begin{cases} t_{u_1} & = T_{\text{in}} - x =: z \\ t_v(z) & = T_{\text{out}} + \mathcal{R}_v(x) \Leftrightarrow \mathcal{R}_v(x) = -T_{\text{out}} + t_v(z) \end{cases} \\ \text{d)} & \quad \begin{cases} t_{u_1} & = T_{\text{in}} + x =: z \\ t_v(z) & = T_{\text{out}} - \mathcal{R}_v(x) \Leftrightarrow \mathcal{R}_v(x) = T_{\text{out}} - t_v(z) \end{cases} \end{aligned}$$

Note that cases a) and b) describe consistent input and output encoding schemes, whereas c) and d) provide inconsistent in the sense that input and output formalism do not match. Consistency is an important property when stacking LSNNs because otherwise, the realization of the stacked network does not match the composition of the subnetworks in general; see Section A.1.

Now, consider the function $f_1 : [a, b] \rightarrow \mathbb{R}$ given by

$$f_1(x) = \begin{cases} x + c_1 & , \text{ if } x > c_3 \\ c_2 & , \text{ if } x < c_3 \end{cases}$$

for some constants $c_1, c_2 \in \mathbb{R}$ and $c_3 \in (a, b)$. To realize f_1 by an LSNN neuron v via scheme a) we need

$$f_1(x) = \mathcal{R}_v(x) = -T_{\text{out}} + t_v(z) \quad \text{for all } x \in [a, b],$$

i.e.,

$$t_v(z) = \begin{cases} z + T_{\text{out}} - T_{\text{in}} + c_1 & , \text{ if } z - T_{\text{in}} > c_3 \\ c_2 + T_{\text{out}} & , \text{ if } z - T_{\text{in}} < c_3 \end{cases} \Leftrightarrow t_v(z) = \begin{cases} z + c'_1 & , \text{ if } z > c'_3 \\ c'_2 & , \text{ if } z < c'_3 \end{cases}, \quad (30)$$

where $c'_1 = T_{\text{out}} - T_{\text{in}} + c_1$, $c'_2 = c_2 + T_{\text{out}}$, and $c'_3 = c_3 + T_{\text{in}}$. However, Proposition 24 implies that the sought function in (30) can not be expressed as the firing map of v . Hence, f_1 can not be realized by an LSNN neuron with encoding scheme a). Similar observations can be made for the other encoding schemes and the realization of the functions in Proposition 24 and 25. Our goal is to realize the ReLU activation function and it is now straightforward to verify that ReLU can not be realized by an LSNN neuron with consistent encoding. In contrast, for the inconsistent scheme c) we obtain for a given input domain $[a, b]$

$$\sigma(x) = \mathcal{R}_v(x) = -T_{\text{out}} + t_v(z),$$

i.e.,

$$t_v(z) = \begin{cases} -z + T_{\text{out}} + T_{\text{in}} & , \text{ if } z \leq T_{\text{in}} \\ T_{\text{out}} & , \text{ if } z > T_{\text{in}} \end{cases},$$

which can be expressed by the firing map of v for a suitable choice of T_{out} and T_{in} ; see Proposition 25. We summarize these observations in our next result.

Proposition 26 *An LSNN neuron can not realize ReLU on a given domain with consistent encoding, whereas ReLU can be realized when applying an inconsistent encoding.*

Is it possible to realize ReLU with a consistent encoding? To realize σ by an LSNN Φ via scheme a) we need

$$\sigma(x) = \mathcal{R}_\Phi(x) = -T_{\text{out}} + t_v(z) \quad \text{for all } x \in [a, b], \quad (31)$$

i.e.,

$$t_v(z) = \begin{cases} z + T_{\text{out}} - T_{\text{in}} & , \text{ if } z - T_{\text{in}} > 0 \\ T_{\text{out}} & , \text{ if } z - T_{\text{in}} \leq 0 \end{cases} \Leftrightarrow t_v(z) = \begin{cases} z + T_{\text{out}} - T_{\text{in}} & , \text{ if } z > T_{\text{in}} \\ T_{\text{out}} & , \text{ if } z \leq T_{\text{in}} \end{cases}. \quad (32)$$

We proceed by constructing a two-layer LSNN that indeed achieves this goal.

Proposition 27 *Let $c_1, c_2 \in \mathbb{R}$ be such that $c_2 > c_1$ and $c_1 + c_2 > 2b$. Consider $f : [a, b] \rightarrow \mathbb{R}$ defined as*

$$f(x) = \begin{cases} x + c_2 - c_1 & , \text{ if } x > c_1 \\ c_2 & , \text{ if } x \leq c_1 \end{cases}.$$

There exists a two-layer LSNN Φ with output neuron v and input neuron u_1 such that $t_v(x) = f(x)$ on $[a, b]$, where $t_v(x)$ denotes the firing time of v on input $t_{u_1} = x$.

Proof We introduce an auxiliary input neuron u_2 with constant firing time $t_{u_2} \in \mathbb{R}$ and specify the parameter of $\Phi = ((W^1, D^1, \Theta^1), (W^2, D^2, \Theta^2))$ in the following manner:

$$W^1 = \begin{pmatrix} -\frac{1}{2} & 0 \\ 1 & 1 \end{pmatrix}, D^1 = \begin{pmatrix} d & 0 \\ d & d \end{pmatrix}, \Theta^1 = \begin{pmatrix} \theta \\ \theta \end{pmatrix}, W^2 = \begin{pmatrix} -\frac{1}{2} \\ 1 \end{pmatrix}, D^2 = \begin{pmatrix} d \\ d \end{pmatrix}, \Theta^2 = \theta, \quad (33)$$

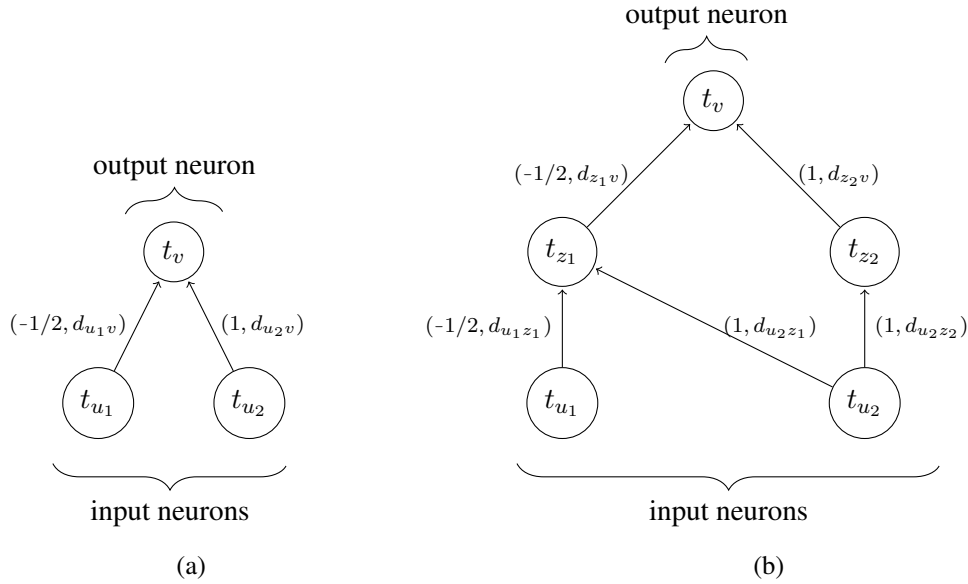


Figure 3: a) Computation graph associated with an LSNN with two input neurons and one output neuron that realizes f as defined in Proposition 25. (b) Stacking the network in (a) twice results in an LSNN that realizes the ReLU activation function.

where $d \geq 0$ and $\theta > 0$ is chosen such that $\theta + t_{u_2} > b$. We denote the input neurons by u_1, u_2 , the neurons in the hidden layer by z_1, z_2 , and the output neuron by v . Note that the firing time of z_1 depends on u_1 and u_2 . In particular, either u_2 or u_1 together with u_2 can trigger a spike in z_1 since $w_{u_1 z_1} < 0$. Therefore, applying (7) yields that u_2 triggers a spike in z_1 under the following circumstances:

$$t_{z_1}(x) = \theta + t_{u_2} + d \quad \text{if } t_{z_1}(x) \leq t_{u_1} + d = x + d.$$

Hence, this case only arises when

$$\theta + t_{u_2} + d \leq x + d \Leftrightarrow \theta + t_{u_2} \leq x. \quad (34)$$

However, by construction $\theta + t_{u_2} > b$, so that (34) does not hold for any $x \in [a, b]$. Thus, we conclude via (8) that

$$t_{z_1}(x) = 2(\theta + t_{u_2} + d) - (x + d) = 2(\theta + t_{u_2}) + d - x.$$

By construction, the firing time $t_{z_2} = \theta + t_{u_2} + d$ of z_2 is a constant which depends on the input only via u_2 . A similar analysis as in the first layer shows that

$$t_v(x) = \theta + t_{z_2} + d \quad \text{if } t_v(x) \leq t_{z_1} + d = 2(\theta + t_{u_2}) + d - x + d = 2(\theta + t_{u_2} + d) - x.$$

Hence, z_2 triggers a spike in v when

$$\theta + t_{z_2} + d = \theta + \theta + t_{u_2} + d + d \leq 2(\theta + t_{u_2} + d) - x \quad \Leftrightarrow \quad x \leq t_{u_2}.$$

If the additional condition

$$\theta + t_{z_2} + d = c_2 \quad \Leftrightarrow \quad 2(\theta + d) + t_{u_2} = c_2 \quad (35)$$

is met, we can infer that

$$\begin{aligned}
t_v(x) &= \begin{cases} 2(\theta + t_{z_2} + d) - (t_{z_1}(x) + d) & , \text{ if } x > t_{u_2} \\ \theta + t_{z_2} + d & , \text{ if } x \leq t_{u_2} \end{cases} \\
&= \begin{cases} 2c_2 - (2(\theta + t_{u_2}) + d - x + d) & , \text{ if } x > t_{u_2} \\ c_2 & , \text{ if } x \leq t_{u_2} \end{cases} \\
&= \begin{cases} x + c_2 - t_{u_2} & , \text{ if } x > t_{u_2} \\ c_2 & , \text{ if } x \leq t_{u_2} \end{cases}.
\end{aligned}$$

Setting $t_{u_2} = c_1$ gives

$$t_v(x) = \begin{cases} x + c_2 - c_1 & , \text{ if } x > c_1 \\ c_2 & , \text{ if } x \leq c_1 \end{cases},$$

and we observe that for $\theta = \frac{1}{2}(c_2 - c_1) - 2d$, which is a valid threshold provided that d is small, (35) as well as $\theta + t_{u_2} = \theta + c_1 > b - 2d > b$ is satisfied for a suitable choice of d . Hence, Φ emulates f as desired. \blacksquare

Finally, we state the implication of this proposition for the ReLU realization, which is a direct implication of (31) and (32).

Proposition 28 *There exists a two-layer LSNN that realizes ReLU on a given bounded domain with a consistent encoding scheme.*

A.4. Realizing ReLU networks by LSNNs

In this section, we show that an LSNN has the capability to reproduce the output of any ReLU network. Specifically, given access to the weights and biases of an ANN, we construct an LSNN and set the parameter values based on the weights and biases of the given ANN. This leads us to the desired result. The essential part of our proof revolves around choosing the parameters of an LSNN such that it effectively realizes the composition of an affine-linear map and the non-linearity represented by the ReLU activation. The realization of ReLU with LSNNs is proved in the previous Section A.3. To realize an affine-linear function using an LSNN neuron, it is necessary to ensure that the spikes from all the input neurons together result in the firing of an output neuron instead of any subset of the input neurons. We achieve that by appropriately adjusting the value of the threshold parameter. As a result, an LSNN neuron, which implements an affine-linear map, avoids partitioning of the input space.

Setup for the proof of Theorem 11 Let $d, L \in \mathbb{N}$ be the width and the depth of an ANN Ψ , respectively, i.e.,

$$\begin{aligned}
\Psi = ((A^1, B^1), (A^2, B^2), \dots, (A^L, B^L)), \text{ where } (A^\ell, B^\ell) \in \mathbb{R}^{d \times d} \times \mathbb{R}^d, 1 \leq \ell < L, \\
(A^L, B^L) \in \mathbb{R}^{d \times d} \times \mathbb{R}^d.
\end{aligned}$$

For a given input domain $[a, b]^d \subset \mathbb{R}^d$, we denote by $\Psi^\ell = ((A^\ell, B^\ell))$ the ℓ -th layer, where $y^0 \in [a, b]^d$ and

$$\begin{aligned}
y^\ell &= \mathcal{R}_{\Psi^\ell}(y^{\ell-1}) = \sigma((A^\ell)^T y^{\ell-1} + B^\ell), 1 \leq \ell < L, \\
y^L &= \mathcal{R}_{\Psi^L}(y^{L-1}) = (A^L)^T y^{L-1} + B^L
\end{aligned} \tag{36}$$

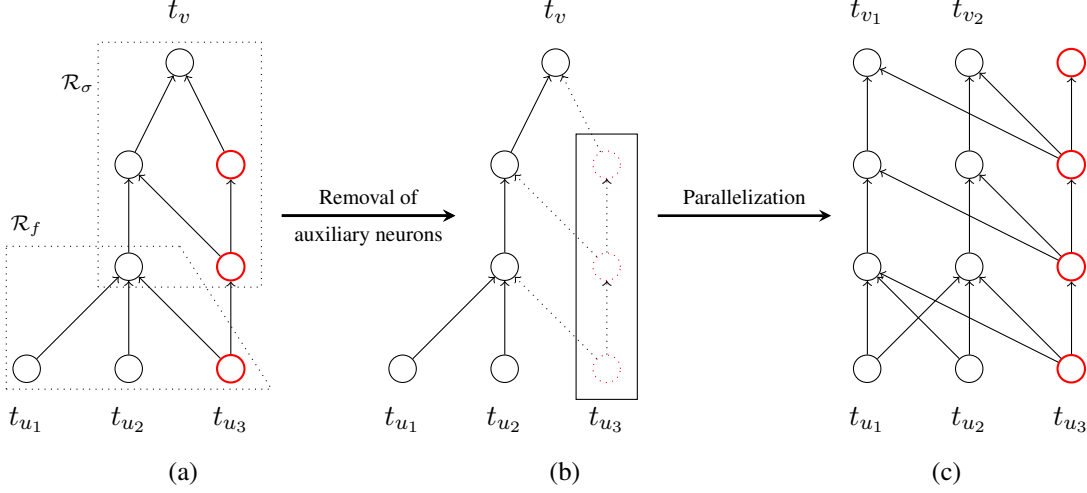


Figure 4: (a) Computation graph associated with an LSNN $\Phi^{\sigma \circ f}$ resulting from the concatenation of Φ^σ and Φ^f that realizes $\sigma(f(x_1, x_2))$, where f is an affine function and σ is the ReLU non-linearity. The auxiliary neurons are shown in red. (b) Same computation graph as in (a); when parallelizing two identical networks, the dotted auxiliary neurons can be removed and auxiliary neurons from (a) can be used for each network instead. (c) Computation graph associated with an LSNN as a result of the parallelization of two subnetworks $\Phi^{\sigma \circ f_1}$ and $\Phi^{\sigma \circ f_2}$. The auxiliary neuron in the output layer serves the same purpose as the auxiliary neuron in the input layer and is needed when concatenating two such subnetworks $\Phi^{\sigma \circ f}$.

so that $\mathcal{R}_\Psi = \mathcal{R}_{\Psi^L} \circ \dots \circ \mathcal{R}_{\Psi^1}$.

For the construction of the corresponding LSNN, we refer to the associated weights and delays between two LSNN neurons u and v by w_{uv} and d_{uv} , respectively.

Proof [of Theorem 11] Any multi-layer ANN Ψ with ReLU activation is simply an alternating composition of affine-linear functions $(A^l)^T y^{l-1} + B^l$ and a non-linear function represented by σ . To generate the mapping realized by Ψ , it suffices to realize the composition of affine-linear functions and the ReLU non-linearity and then extend the construction to the whole network using concatenation and parallelization operations. We prove the result via the following steps; see also Figure 4 for a depiction of the intermediate constructions.

Step 1: Realizing ReLU non-linearity.

Proposition 28 gives the desired result.

Step 2: Realizing affine-linear functions with one-dimensional range.

Let $f : [a, b]^d \rightarrow \mathbb{R}$ be an affine-linear function

$$f(x) = C^T x + s, \quad C^T = (c_1, \dots, c_d) \in \mathbb{R}^d, s \in \mathbb{R}. \quad (37)$$

Consider a one-layer LSNN that consists of an output neuron v and d input units u_1, \dots, u_d . Via (8) the firing time of v as a function of the input firing times on the linear region R^I corresponding to the index set $I = \{1, \dots, d\}$ is given by

$$t_v(t_{u_1}, \dots, t_{u_d}) = \frac{\theta_v}{\sum_{i \in I} w_{u_i v}} + \frac{\sum_{i \in I} w_{u_i v} (t_{u_i} + d_{u_i v})}{\sum_{i \in I} w_{u_i v}} \quad \text{provided that } \sum_{i \in I} w_{u_i v} > 0.$$

Introducing an auxiliary input neuron u_{d+1} with weight $w_{u_{d+1}v} = 1 - \sum_{i \in I} w_{u_i v}$ ensures that $\sum_{i \in I \cup \{d+1\}} w_{u_i v} > 0$ and leads to the firing time

$$t_v(t_{u_1}, \dots, t_{u_{d+1}}) = \theta_v + \sum_{i \in I \cup \{d+1\}} w_{u_i v} (t_{u_i} + d_{u_i v}) \quad \text{on } R^{I \cup \{d+1\}}.$$

Setting $w_{u_i v} = c_i$ for $i \in I$ and $d_{u_j v} = d' \geq 0$ for $j \in I \cup \{d+1\}$ yields

$$t_v(t_{u_1}, \dots, t_{u_{d+1}}) = \theta_v + w_{u_{d+1}v} \cdot t_{u_{d+1}} + d' + \sum_{i \in I} c_i t_{u_i} \quad \text{on } R^{I \cup \{d+1\}} \cap [a, b]^d.$$

Therefore, an LSNN $\Phi^f = (W, D, \Theta)$ with parameters

$$W = \begin{pmatrix} c_1 \\ \vdots \\ c_{d+1} \end{pmatrix}, D = \begin{pmatrix} d' \\ \vdots \\ d' \end{pmatrix}, \Theta = \theta > 0, \quad \text{where } c_{d+1} = 1 - \sum_{i \in I} c_i,$$

and the usual encoding scheme $T_{\text{in}}/T_{\text{out}} + \cdot$ and fixed firing time $t_{u_{d+1}} = t_{\text{in}}$, whereby we employ the notation from Definition 7, realizes

$$\mathcal{R}_{\Phi^f}(x) = -t_{\text{out}} + t_v(t_{\text{in}} + x_1, \dots, t_{\text{in}} + x_d, t_{\text{in}}) = -t_{\text{out}} + \theta + t_{\text{in}} + d' + \sum_{i \in I} c_i x_i \quad (38)$$

$$= -t_{\text{out}} + \theta + t_{\text{in}} + d' + f(x_1, \dots, x_d) - s \quad \text{on } R^{I \cup \{d+1\}} \cap [a, b]^d. \quad (39)$$

Choosing a large enough threshold θ ensures that a spike in v is necessarily triggered after all the spikes from u_1, \dots, u_{d+1} reached v so that $[a, b]^d \subset R^{I \cup \{d+1\}}$ holds. It suffices to set

$$\theta \geq \sup_{x \in [a, b]^d} \sup_{x_{\min} \leq t - t_{\text{in}} - d' \leq x_{\max}} P_v(t),$$

where $x_{\min} = \min\{x_1, \dots, x_d, 0\}$ and $x_{\max} = \max\{x_1, \dots, x_d, 0\}$, since this implies that the potential $P_v(t)$ is smaller than the threshold to trigger a spike in v on the time interval associated to feasible input spikes, i.e., v emits a spike after the last spike from an input neuron arrived at v . Applying (7) shows that for $x \in [a, b]^d$ and $t \in [x_{\min} + t_{\text{in}} + d', x_{\max} + t_{\text{in}} + d']$

$$\begin{aligned} P_v(t) &= \sum_{i \in I} w_{u_i v} (t - (t_{\text{in}} + x_i) - d_{u_i v}) + w_{u_{d+1}v} (t - t_{\text{in}} - d_{u_{d+1}v}) = t - d' - t_{\text{in}} + \sum_{i \in I} c_i x_i \\ &\leq x_{\max} + d \|C\|_{\infty} \|x\|_{\infty} \leq (1 + d \|C\|_{\infty}) \max\{|a|, |b|\}. \end{aligned}$$

Hence, we set

$$\theta = (1 + d \|C\|_{\infty}) \max\{|a|, |b|\} + s + |s| \quad \text{and} \quad t_{\text{out}} = \theta - s + t_{\text{in}} + d'$$

to obtain via (38) that

$$\mathcal{R}_{\Phi^f}(x) = -t_{\text{out}} + t_v(t_{\text{in}} + x_1, \dots, t_{\text{in}} + x_d, t_{\text{in}}) = f(x) \quad \text{for } x \in [a, b]^d. \quad (40)$$

Note that the reference time $t_{\text{out}} = (1 + d \|C\|_{\infty}) \max\{|a|, |b|\} + |s| + t_{\text{in}} + d'$ is independent of the specific parameters of f in the sense that only upper bounds $\|C\|_{\infty}, |s|$ on the parameters

are relevant. Therefore, t_{out} (with the associated choice of θ) can be applied for different affine linear functions as long as the upper bounds remain valid. This is necessary for the composition and parallelization of subnetworks in the subsequent construction.

Step 3: Realizing compositions of affine-linear functions with one-dimensional range and ReLU.

The next step is to realize the composition of ReLU σ with an affine linear mapping f defined in (37). To that end, we want to concatenate the networks Φ^σ and Φ^f constructed in Step 1 and Step 2, respectively, via Lemma 16. To employ the concatenation operation we need to perform the following steps:

1. Find an appropriate input domain $[a', b'] \subset \mathbb{R}$, that contains the image $f([a, b]^d)$ so that parameters and reference times of Φ^σ can be fixed appropriately (see Proposition 28 for the detailed conditions on how to choose the parameter).
2. Ensure that the output reference time t_{out}^f of Φ^f equals the input reference time t_{in}^σ of Φ^σ .
3. Ensure that the number of neurons in the output layer of Φ^f is the same as the number of input neurons in Φ^σ .

For the first point, note that

$$|f(x)| = |C^T x + s| \leq d \|C\|_\infty \cdot \|x\|_\infty + |s| \leq d \|C\|_\infty \cdot \max\{|a|, |b|\} + |s| \text{ for all } x \in [a, b]^d.$$

Hence, we can use the input domain

$$[a', b'] = [-d \|C\|_\infty \cdot \max\{|a|, |b|\} + |s|, d \|C\|_\infty \cdot \max\{|a|, |b|\} + |s|]$$

and specify the parameters of Φ^σ accordingly. Additionally, recall from Proposition 28 that t_{in}^σ can be chosen freely, so we may fix $t_{\text{in}}^\sigma = t_{\text{out}}^f$, where t_{out}^f is established in Step 2. It remains to consider the third point. To realize ReLU, an additional auxiliary neuron in the input layer of Φ^σ with constant input t_{in}^σ was introduced. Hence, we also need to add an auxiliary output neuron in Φ^f with (constant) firing time $t_{\text{out}}^f = t_{\text{in}}^\sigma$ so that the corresponding output and input dimension and their specification match. This is achieved by introducing a single synapse from the auxiliary neuron in the input layer of Φ^f to the newly added output neuron and by specifying the parameters of the newly introduced synapse and neuron suitably. Formally, the adapted network $\Phi^f = (W, D, \Theta)$ is given by

$$W = \begin{pmatrix} c_1 & 0 \\ \vdots & \vdots \\ c_d & 0 \\ c_{d+1} & 1 \end{pmatrix}, D = \begin{pmatrix} d' & 0 \\ \vdots & \vdots \\ d' & 0 \\ d' & d' \end{pmatrix}, \Theta = \begin{pmatrix} \theta \\ t_{\text{out}}^f - t_{\text{in}}^f - d' \end{pmatrix},$$

where the values of the parameters are specified in Step 2.

Then the realization of the concatenated network $\Phi^{\sigma \circ f}$ is the composition of the individual realizations. This is exemplarily demonstrated in Figure 4a for the two-dimensional input case. By analyzing $\Phi^{\sigma \circ f}$, we conclude that a three-layer LSNN with

$$N(\Phi^{\sigma \circ f}) = N(\Phi^\sigma) - N_0(\Phi^\sigma) + N(\Phi^f) = 5 - 2 + d + 3 = d + 6$$

computational units can realize $\sigma \circ f$ on $[a, b]^d$, where $N_0(\Phi^\sigma)$ denotes the number of neurons in the input layer of Φ^σ .

Step 4: Realizing layer-wise computation of Ψ .

The computations performed in a layer Ψ^ℓ of Ψ are described in (6). Hence, for $1 \leq \ell < L$ the computation can be expressed as

$$\mathcal{R}_{\Psi^\ell}(y^{\ell-1}) = \sigma((A^\ell)^T y^{\ell-1} + B^\ell) = \begin{pmatrix} \sigma(\sum_{i=1}^d (A_{1,i}^\ell)^T y_i^{\ell-1} + B_1^\ell) \\ \vdots \\ \sigma(\sum_{i=1}^d (A_{d,i}^\ell)^T y_i^{\ell-1} + B_d^\ell) \end{pmatrix} =: \begin{pmatrix} \sigma(f_1(y^{\ell-1})) \\ \vdots \\ \sigma(f_d(y^{\ell-1})) \end{pmatrix},$$

where $f_1^\ell, \dots, f_d^\ell$ are affine linear functions with one-dimensional range on the same input domain $[a^{\ell-1}, b^{\ell-1}]^d \subset \mathbb{R}^d$, where $[a^0, b^0] = [a, b]$ and $[a^\ell, b^\ell]$ is the range of

$$(\sigma \circ f_1^{\ell-1}, \dots, \sigma \circ f_d^{\ell-1})([a^{\ell-1}, b^{\ell-1}]^d).$$

Thus, via Step 3, we construct LSNNs $\Phi_1^\ell, \dots, \Phi_d^\ell$ that realize $\sigma \circ f_1^\ell, \dots, \sigma \circ f_d^\ell$ on $[a^{\ell-1}, b^{\ell-1}]$. Note that by choosing appropriate parameters in the construction performed in Step 2 (as described below (40)), e.g., $\|A^\ell\|_\infty$ and $\|B^\ell\|_\infty$, we can employ the same input and output reference time for each $\Phi_1^\ell, \dots, \Phi_d^\ell$. Consequently, we can parallelize $\Phi_1^\ell, \dots, \Phi_d^\ell$ (see Lemma 18) and obtain networks $\Phi^\ell = P(\Phi_1^\ell, \dots, \Phi_d^\ell)$ realizing \mathcal{R}_{Ψ^ℓ} on $[a^{\ell-1}, b^{\ell-1}]^d$. Finally, Ψ^L can be directly realized via Step 2 by an LSNN Φ^L (as in the last layer no activation function is applied and the output is d-dimensional). Although Φ^ℓ already performs the desired task of realizing \mathcal{R}_{Ψ^ℓ} we can slightly simplify the network. By construction in Step 3, each Φ_i^ℓ contains two auxiliary neurons in the hidden layers. Since the input and output reference time is chosen consistently for $\Phi_1^\ell, \dots, \Phi_d^\ell$, we observe that the auxiliary neurons in each Φ_i^ℓ perform the same operations and have the same firing times. Therefore, without changing the realization of Φ^ℓ we can remove the auxiliary neurons in $\Phi_2^\ell, \dots, \Phi_d^\ell$ and introduce synapses from the auxiliary neurons in Φ_1^ℓ accordingly. This is exemplarily demonstrated in Figure 4b for the case $d = 2$. After this modification, we observe that $L(\Phi^\ell) = L(\Phi_i^\ell) = 3$ and

$$\begin{aligned} N(\Phi^\ell) &= N(\Phi_1^\ell) + \sum_{i=2}^d (N(\Phi_i^\ell) - 2 - N_0(\Phi_i^\ell)) = dN(\Phi_1^\ell) - (d-1)(2 + N_0(\Phi_1^\ell)) \\ &= d(d+6) - 2(d-1) - (d-1)(d+1) = 4d+3 \quad \text{for } 1 \leq \ell < L, \end{aligned}$$

whereas $L(\Phi^L) = 1$ and $N(\Phi^L) = 2d+1$.

Step 5: Realizing compositions of layer-wise computations of Ψ .

The last step is to compose the realizations $\mathcal{R}_{\Phi^1}, \dots, \mathcal{R}_{\Phi^L}$ to obtain the realization

$$\mathcal{R}_{\Phi^L} \circ \dots \circ \mathcal{R}_{\Phi^1} = \mathcal{R}_{\Psi^L} \circ \dots \circ \mathcal{R}_{\Psi^1} = \mathcal{R}_\Psi.$$

As in Step 3, it suffices again to verify that the concatenation of the networks $\mathcal{R}_{\Phi^1}, \dots, \mathcal{R}_{\Phi^L}$ is feasible. First, note that for $\ell = 1, \dots, L$ the input domain of \mathcal{R}_{Φ^ℓ} is given by $[a^{\ell-1}, b^{\ell-1}]^d$ so that, we can fix the suitable output reference time $T_{\text{out}}^{\Phi^\ell} = t_{\text{out}}^{\Phi^\ell}(1, \dots, 1)^T \in \mathbb{R}^d$ based on the parameters of the network, the domain $[a^{\ell-1}, b^{\ell-1}]$, and some input reference time $T_{\text{in}}^{\Phi^\ell} = t_{\text{in}}^{\Phi^\ell}(1, \dots, 1)^T \in \mathbb{R}^d$. By construction in Steps 2 - 4 $T_{\text{in}}^{\Phi^\ell}$ can be chosen freely. Hence setting $T_{\text{in}}^{\Phi^{\ell+1}} = T_{\text{out}}^{\Phi^\ell}$ ensures

that the reference times of the corresponding networks agree. It is left to align the input dimension of $\Phi^{\ell+1}$ and the output dimension of Φ^ℓ for $\ell = 1, \dots, L-1$. Due to the auxiliary neuron in the input layer of $\Phi^{\ell+1}$, we also need to introduce an auxiliary neuron in the output layer of Φ^ℓ (see Figure 4c) with the required firing time $t_{\text{in}}^{\Phi^{\ell+1}} = t_{\text{out}}^{\Phi^\ell}$. Similarly, as in Step 3, it suffices to add a single synapse from the auxiliary neuron in the previous layer to obtain the desired firing time.

Thus, we conclude that $\Phi = \Phi^L \bullet \dots \bullet \Phi^1$ realizes \mathcal{R}_Ψ on $[a, b]$, as desired. The complexity of Φ in the number of layers and neurons is given by

$$L(\Phi) = \sum_{\ell=1}^L L(\Phi^\ell) = 3L - 2 = 3L(\Psi) - 2$$

and

$$\begin{aligned} N(\Phi) &= N(\Phi^1) + \sum_{\ell=2}^L (N(\Phi^\ell) - N_0(\Phi^\ell)) + (L-1) \\ &= 4d + 3 + (L-2)(4d + 3 - (d+1)) + (2d + 1 - (d+1)) + (L-1) \\ &= 3L(d+1) - (2d+1) \\ &= N(\Psi) + L(2d+3) - (2d+2) \end{aligned}$$

■

Remark 29 *Note that the delays play no significant role in the proof of the above theorem. Nevertheless, they can be employed to alter the timing of spikes, consequently impacting the firing time and the resulting output. However, the exact function of delays requires further investigation. The primary objective is to present a construction that proves the existence of an LSNN capable of accurately reproducing the output of any ReLU network.*