Ludwig-Maximilians-Universität at München
Department “Institut für Mathematik”
Lehr- und Forschungseinheit Mathematische Grundlagen der künstlichen Intelligenz
Prof. Dr. Gitta Kutyniok

Bachelor’s Thesis

Understanding the Representational Power of Recurrent Spiking Neural Networks in Discrete Time

Valentin Herrmann
plus+official@valentin-herrmann.de

Bearbeitungszeitraum:	15.8.2025 bis 31.10.2025
Betreuer:	Adalbert Fono
Verantw. Hochschullehrer:	Prof. Dr. Gitta Kutyniok

Contents

1 Introduction

In recent years, the capabilities of AI have improved significantly. Especially large language models have found their ways into most people's lives. But even though LLMs (see [Jones and Bergen, 2025]) already pass a simple Turing Test, it is still unclear whether they are fundamentally able of reasoning. While they are able to solve many problems with lots of compute, the process of finding a solution quite often involves far more try and error than it does for a human (see TODO). Human intuition on the other hand does not require as many attempts and quite often leads to a better solution (see TODO). Another limitation seem to be hallucinations, that appear to be an intrinsic property of LLMs (see TODO).

A further problem with current models is their incredible energy inefficiency compared to a human brain. Of course one single human is not an expert in every field, but still, even if we account for this

We conclude therefore that other models might be better fitted to the task of reasoning. Since many human technological advances have been inspired by the longest ongoing optimization progress in history — the evolution of life — like airplanes or sonars, we propose that neural networks with more similar inner workings to human brains might be smarter and more efficient. In particular we will investigate a kind of spiking neural network.

While the idea behind SNNs is quite old, they have not been researched as much , since it is comparatively hard to train them. Therefore there still remain a lot of open questions about them. In this paper we shall extend on the work done in [Nguyen et al., 2025]. We will add a decaying factor to the input of the neurons and allow recursive connections between neurons in a layer.

We will roughly follow the structure of [Nguyen et al., 2025]: In Section 2 we will motivate and formally introduce recurrent discrete time leaky-integrate-and-fire SNN, in short r. LIF-SNNs. In Section 3 we will show that some continuously differentiable functions can be approximated arbitrarily well by r. LIF-SNNs. While this has already been show in [Nguyen et al., 2025] more generally for continuous functions, we present a construction using far less neurons, using the internal linear structure of r. LIF-SNNs. In the following, in Section 4, we analyze the landscape of a r. LIF-SNN regarding the shape of constant output regions. While we were not able to generalize the upper bound on the number of constant regions of d.t. LIF-SNN that [Nguyen et al., 2025] established to r. LIF-SNN, we show some promising experimental results in Section 5. We additionally explain how we implemented the algorithms

2 Definition of r. LIF-SNN

2.1 Notations and Conventions

First a few words about the notation we will use in the paper:

We write $\{a,...,b\} := \{a, a+1, ..., b\}$ for the range of integers from a to b and in particular $[n] := \{1,...,n\}$ for the integers from 1 to n and $[n]_0 := \{0,...,n\}$ for the integers from 0 to n. Moreover $\vee$ is used to mean the logical "or", $\wedge$ to mean the logical "and".

We write $[x,y) := \{z \in \mathbb{R} \mid x \leq z < y\}$ for the half-open interval between $x \in \mathbb{R}$ and $y \in \mathbb{R}$ and further use $[\![x,y)\!)$ with $x,y \in \mathbb{R}^n$ to write half-open cuboids $\prod_{i=1}^n [x_i, y_i)$. Similarly $[\![x,y]\!]$ is the closed cuboid $\prod_{i=1}^n [x_i, y_i]$. A cube is a cuboid such that all sides have the same length. We further take $[x,y)$ to be empty for $x,y \in \mathbb{R}$ with $x \geq y$, and $[-\infty, x)]$ to mean $(-\infty, x)$. This means in particular, that $[\![x,y)\!) = \emptyset$ holds exactly when $y_i \leq x_i$ for a $i \in [n]$.

We further use $\operatorname{diam}_p(U) := \sup_{x,y \in U} \|x-y\|_p$ to mean the diameter of $U \subset \mathbb{R}^m$ regarding the p-norm $\|\cdot\|_p$.

Continuing, $e_i := (0,...,1,...,0) \in \mathbb{R}^n$ is the i-th standard basis vector, $\mathbf{0}_n := (0,...,0) \in \mathbb{R}^n$, $\mathbf{1}_n := (1,...,1) \in \mathbb{R}^n$ the vectors containing 0 and 1s in every component, respectively and $I_n \in \mathbb{R}^{n \times n}$ the identity matrix of size $n \times n$. Is moreover $W \in \mathbb{R}^{n \times m}$ a matrix, then we will write $w_i \in \mathbb{R}^{1 \times n}$ for the i-th row vector and $W_{i,j} \in \mathbb{R}$ for the value of the cell at location (i,j). We add an ordering $\leq$ to vectors: $x \leq y$ holds exactly for $x,y \in \mathbb{R}^n$ if $\forall_{i \in [n]} x_i \leq y_i$. Further $x < y$ is canonically defined, so $x < y$ holds exactly if $x \leq y$ but not $x = y$.

Is further $f : U \to \mathbb{R}^n$ a function with arbitrary domain U, then $f_i := \pi_i \circ f$ is the i-th component function. $\pi_i : \mathbb{R}^n \to \mathbb{R}$ is the projection to the i-th component. We also write f^{-1} for the inverse of f and $f^{-1}(W) := \{x \in U \mid f(x) \in W\}$ for the preimage of W under f. The set of continuous functions of type $U \to W$ will be written by $\mathcal{C}^0(U,W)$. Correspondingly, the set of k-times continuously differentiable functions will be written $\mathcal{C}^k(U,W)$. The total derivative of a differentiable function $f : U \to W$ is written as $df : U \to \operatorname{Hom}_{\mathbb{R}}(W,W)$.

We moreover write $\max U := (\max_{u \in U} u_i)_{i \in [n]}$ with $U \subset (\mathbb{R} \cup \{\pm\infty\})^n$ for the maximum by component of U, we similarly define $\min U$, $\inf U$ and $\sup U$. In addition we use the conventions $\inf(\emptyset) = \infty$ and $\sup(\emptyset) = -\infty$.

We finally write

$$\chi_M(x) := \begin{cases} 1 & x \in M \\ 0 & x \notin M \end{cases}$$

for the characteristic function of a set M and

$$1_A := \begin{cases} 1 & A \\ 0 & \neg A \end{cases}$$

for a formula A, such that in particular $\chi_M(x) = 1_{(x \in M)}$.

2.2 Motivation

A neuron in the human brain has connections to other neurons, which may be weaker or stronger. A neuron further has an internal membrane potential that is increased or decreased by the input from other neurons. If that membrane potential reaches a certain point, the neuron will spike and the connected neurons will receive an update. This update is delivered by chemical transmitters or electric signals. Further this signal won't just be a binary spike, but will decay over time after the spike, e.g. because the chemical transmitters have some chance of taking a longer path to the next neuron. The membrane potential may also decay over time.

From these biologic observations we can derive the following differential equations for a single neuron input $I : \mathbb{R} \to \mathbb{R}$ and membrane potential $U : \mathbb{R} \to \mathbb{R}$ (compare e.g. [Gerstner et al., 2014]):

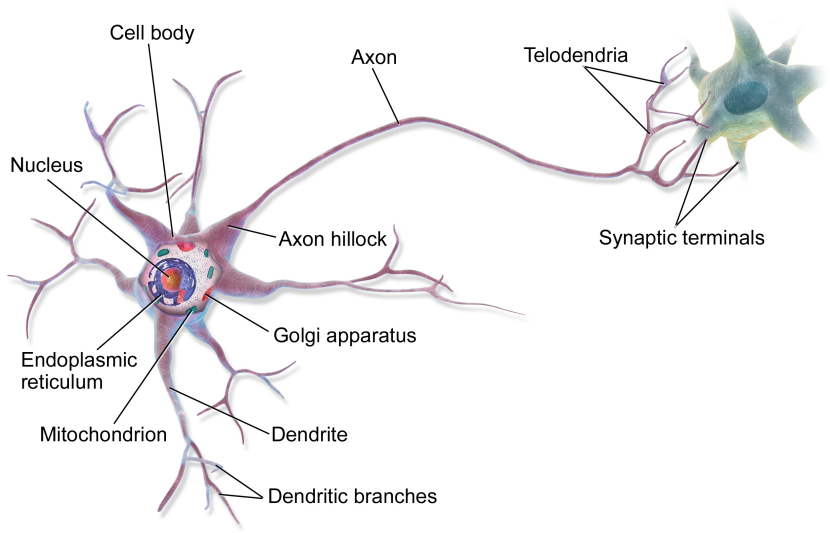


Figure 2.1: Structure of a neuron.[1]

$$I'(t) := -\tau_\alpha I(t) + \sum_{j=1}^{n} w_j s_j(t-\Delta t)$$
$$U'(t) := -\tau_\beta U(t) + I(t) + b - \vartheta s(t)$$

Here $s_j : \mathbb{R} \to \{0,1\}$ represents the j-th connection of the given neuron spiking at time t; the variables $w_j \in \mathbb{R}$ represent the weights of the connections. Since we might get unresolvable dependencies between neurons otherwise, the connections need to have some latency $\Delta t \in \mathbb{R}$. The variables τ_α, τ_β specify the decay rate.

Since we are in an analog scenario, it is quite reasonable to assume that I and U describe smoothly differentiable functions. We can therefore use the first-order exponential integrator method to obtain a discretization of I and U from the differential equations. Let $t_0, h \in \mathbb{R}$ be arbitrary and $t_{n+1} := t_n + h$. By using the fundamental theorem of calculus we get

$$\begin{aligned}
&e^{\tau_\alpha t_{n+1}} I(t_{n+1}) - e^{\tau_\alpha t_n} I(t_n) \\
&= \int_{t_n}^{t_{n+1}} \frac{d}{dt} \left(e^{\tau_\alpha t} I(t)\right) dt \\
&= \int_{t_n}^{t_{n+1}} e^{\tau_\alpha t} (I'(t) + \tau_\alpha I(t)) dt \\
&= \int_{t_n}^{t_{n+1}} e^{\tau_\alpha t} \left(\sum_{j=1}^{n} w_j s_j(t-\Delta t) \right) dt
\end{aligned}$$

If we assume the input from the spikes of other neurons to be constant during $[t_n, t_{n+1}]$, we get

$$\begin{aligned}
&= \frac{1}{\tau_\alpha} (e^{\tau_\alpha t_{n+1}} - e^{\tau_\alpha t_n}) \sum_{j=1}^{n} w_j s_j(t-\Delta t) \\
&= \frac{1}{\tau_\alpha} e^{\tau_\alpha t_{n+1}} (1 - e^{\tau_\alpha h}) \sum_{j=1}^{n} w_j s_j(t-\Delta t)
\end{aligned}$$

So we get

$$I(t_{n+1}) = e^{-\tau_\alpha h} I(t_n) + \frac{1}{\tau_\alpha} (1 - e^{\tau_\alpha h}) \sum_{j=1}^{n} w_j s_j(t-\Delta t)$$

[1] Source: By BruceBlaus - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=28761830

Let us now use $h = 1$ and absorb $\frac{1-e^{\tau_\alpha}}{\tau_\alpha}$ into the weights $(w_j)_{j\in[n]}$, such that we have

$$I(t_{n+1}) = \alpha I(t_n) + \sum_{j=1}^{n} w_j s_j(t - \Delta t) \tag{1}$$

by using $\alpha := e^{-\tau_\alpha}$. We similarly obtain

$$U(t_{n+1}) = \beta U(t_n) + I(t) + b - \vartheta s(t) \tag{2}$$

by using the first-order exponential integrator method, defining $\beta := e^{\tau_\beta}$ and absorbing $\frac{1-\beta}{\tau_\beta}$ into $I(0)$, $(w_j)_{j\in[n]}$, b and ϑ.

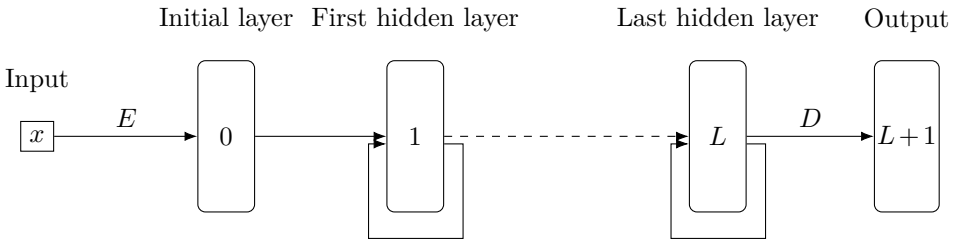


Figure 2.2: High-level network layout

We will now arrange those neurons into layers like shown in Fig. 2.2, such that neurons are only connected to neurons from the previous layer or their own layer. Of course, since we allow recurrent connections we can always just merge all layers into the first one, but the layers do give us more control over the network. For connections between different layers, we can use $\Delta t = 0$ in (1) since there can be no interdependent connections between neurons in different layers. For connections between neurons in the same layer we use $\Delta t > 0$ to prevent those interdependencies. Since we have previously chosen $h = 1$, it is natural to further choose $\Delta t = 1$.

Since we want to work with arbitrary data and not just spike trains, we further need to encode our data to spikes trains and decode it from spike trains. Like [Nguyen et al., 2025] we choose a simple direct encoding and membrane potential outputs.

2.3 Definitions

Our type of SNN should be thought of as a composition of an initial input layer, a number of hidden spiking layers with internal state and an affine-linear layer mapping spikes activations over time, so called spike trains, to the value of the output layer.

We first define the structure of the hidden layers:

Definition 2.1. The **input vector** $i^{[l]}(t) \in \{0,1\}^{n_l}$, the **spike vector** $s^{[l]}(t) \in \{0,1\}^{n_l}$, the **pre-spike membrane potential vector** $p^{[l]}(t)$ and the **post-spike membrane potential vector** $u^{[l]}(t)$ of a hidden layer $\lambda = (W^{[l]}, b^{[l]}, u^{[l]}(0), i^{[l]}(0), \alpha^{[l]}, \beta^{[l]}, \vartheta^{[l]})$ with index $l \in [L]$, are recursively defined as

$$i^{[l]}(t) := \alpha^{[l]} i^{[l]}(t-1) + W^{[l]} s^{[l-1]}(t) + V^{[l]} s^{[l]}(t-1), \tag{3}$$
$$p^{[l]}(t) := \beta^{[l]} u^{[l]}(t-1) + i^{[l]}(t) + b^{[l]}, \tag{4}$$
$$s^{[l]}(t) := H(p^{[l]}(t) - \vartheta \mathbf{1}_{n_l}), \tag{5}$$
$$u^{[l]}(t) := p^{[l]}(t) - \vartheta s^{[l]}(t), \tag{6}$$

with $\forall_{l\in[L]} s^{[l]}(0) = 0$ and given

- **initial membrane potential**: $u^{[l]}(0) \in \mathbb{R}^{n_l}$,

- **initial input**: $i^{[l]}(0) \in \mathbb{R}^{n_l}$,
- **weight matrices**: $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$, $V^{[l]} \in \mathbb{R}^{n_l \times n_l}$,
- **bias vectors**: $b^{[l]} \in \mathbb{R}^{n_l}$,
- **leaky terms**: $\alpha^{[l]}, \beta^{[l]} \in [0,1]$,
- **threshold**: $\vartheta^{[l]} \in (0,\infty)$,

where $H := \mathbb{1}_{[0,\infty)}$ is a step function, $T \in \mathbb{N}$ is the number of **simulated time steps** and $L \in \mathbb{N}$ the total **number of hidden layers**.

Remark 2.1. While it is suppressed in the notation, $i^{[l]}$, $p^{[l]}$, $s^{[l]}$ and $u^{[l]}$ clearly not only depend on t, but by recursion on $s^{[0]}$.

Moreover we will also use $s^{[l]}$ as if it was an element of $\{0,1\}^{n_l \times T}$.

We further define recurrent d.t. LIF-SNN and the function the network realizes:

Definition 2.2. A **recurrent discrete-time LIF-SNN**, also called r. LIF-SNN, of **depth** L with **layer-widths** $(n_0, ..., n_{L+1})$ and $T \in \mathbb{N}$ time-steps is given by

$$\Phi := ((W^{[l]}, b^{[l]}, V^{[l]}, u^{[l]}(0), i^{[l]}(0), \alpha^{[l]}, \beta^{[l]}, \vartheta^{[l]})_{l \in [L]}, T, (E, D))$$

where the **input encoder** $E : \mathbb{R}^{n_0} \to \mathbb{R}^{n_0 \times T}$ maps a vector $x \in \mathbb{R}^{n_0}$ to a corresponding first layer spike activation $\forall_{t \in [T]} s^{[0]}(t) = E(x)(t)$ and the **output decoder** $D : \{0,1\}^{n_L \times T} \to \mathbb{R}^{n_{L+1}}$ maps the spike activations of the last hidden layer to real values.

Definition 2.3. A recurrent discrete-time LIF-SNN ϕ **realizes** the function $R(\Phi) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_{L+1}}$:

$$R(\Phi)(x) = D((s^{[L]}(t))_{t \in [T]}) \quad \text{with } s^{[0]} := E(x)$$

Definition 2.4. A recurrent discrete-time LIF-SNN employs **direct encoding** if we have

$$\forall_{t \in [T]} E(x)(t) = x$$

for the input encoder and has **membrane potential outputs** if the output decoder can be written as

$$D((s(t))_{t \in [T]}) = \sum_{t=1}^{T} a_t (W^{[L+1]} s^{[L]}(t) + b^{[L+1]})$$

for some $(a_t)_{t \in [T]} \in \mathbb{R}^T$, $b^{[L+1]} \in \mathbb{R}^{n_{L+1}}$ and $W^{[L+1]} \in \mathbb{R}^{n_{L+1} \times n_L}$.

Remark 2.2. We will only consider recurrent discrete-time LIF-SNN with direct encoding and membrane potential outputs. In fact, we will use "recurrent discrete-time LIF-SNN" to mean "r. LIF-SNN with direct encoding and membrane potential".

Remark 2.3. Our definition of r. LIF-SNN breaks now down to the definition d.t. LIF-SNN of [Nguyen et al., 2025] if we require $\alpha^{[l]} := 0$ and $V^{[l]}$ for all layers $l \in [L]$. So r. LIF-SNNs can have recursive dependencies inside of the layers and decaying input vectors in contrast to the more simpler d.t. LIF-SNNs.

Let us now take a look at some simple examples of r. LIF-SNN:

Example 2.1. *Let $T, L \in \mathbb{N}$. Then there exists a r. LIF-SNN with $\forall_{t \in [T]} s^{[L]}(t) = s^{[0]}(t)$ for any $s^{[0]} \in \{0,1\}^{n_0 \times T}$.*

Let us use constant width $n_l = n$, weights $W^{[l]} = I_n$, $V^{[l]} = \mathbf{0}_{n \times n}$, biases $b^{[l]} = 0$, initial input $i^{[l]}(0) = 0$, initial membrane potential $u^{[l]}(0) = 0$, leaky terms $\alpha^{[l]} = \beta^{[l]} = 0$ and threshold $\vartheta^{[l]} = 1$ for all $l \in [L]$.

It follows from the definitions that $i^{[l]}(t) = s^{[l-1]}(t)$ and therefore further

$$s^{[l]}(t) = H(s^{[l-1]}(t) - \mathbf{1}_{n_l}) = s^{[l-1]}(t).$$

So by induction we indeed get $\forall_{t \in [T]} s^{[L]}(t) = s^{[0]}(t)$.

Example 2.2. *Let $T, L \in \mathbb{N}$. Then there exists a r. LIF-SNN with $\forall_{t\in T, i\in[n]} s_i^{[L]}(t) = 1 \Leftrightarrow \exists_{t'\in[t-1]}(s_i^{[0]}(t')) = 1$ for any $s^{[0]} \in \{0,1\}^{n_0\times T}$, i.e. an output neuron switches on once the corresponding input neurons fires.*

Let us use constant width $n_l = n$, weights $W^{[l]} = I_n$, $V^{[l]} = I_n$, biases $b^{[l]} = 0$, initial input $i^{[l]}(0) = 0$, initial membrane potential $u^{[l]}(0) = 0$, leaky terms $\alpha^{[l]} = 0$, $\beta^{[l]} = 0$ and threshold $\vartheta^{[l]} = 1$ for all $l \in [L]$.

We then get by definition

$$i^{[l]}(t) = s^{[l-1]}(t) + s^{[l]}(t-1)$$

and therefore

$$s^{[l]}(t) = H(s^{[l-1]}(t) + s^{[l]}(t-1) - \mathbf{1}_{n_l}).$$

So if and only if $s_i^{[l-1]}(t) = 1$ or $s_i^{[l]}(t-1) = 1$, then $s_i^{[l]}(t) = 1$. We therefore have $\forall_{t\in T} s_i^{[l]}(t) = 1 \Leftrightarrow \exists_{t'\in[t]}(s_i^{[l-1]}(t')) = 1$ for all $i \in [n], l \in [L]$. Thus $\forall_{t\in T, i\in[n]} s_i^{[L]}(t) = 1 \Leftrightarrow \exists_{t'\in[t-1]}(s_i^{[0]}(t')) = 1$ by induction.

There also is a d.t. LIF-SNN construction achieving the same behavior: We use the same parameters as before, but with $V^{[l]} = \mathbf{0}_{n\times n}$, $W^{[l]} = T \cdot I_n$ and $\beta^{[l]} = 1$ for all layers $l \in [L]$.

We then get by definition

$$i^{[l]}(t) = T s^{[l-1]}(t)$$

and therefore

$$\begin{aligned} s^{[l]}(t) &= H(p^{[l]}(t) - \mathbf{1}_{n_l}) \\ p^{[l]}(t) &= \quad p^{[l]}(t-1) + T s^{[l-1]}(t) - s^{[l]}(t-1) \end{aligned}$$

By induction over t (see also Lemma 2.1) we obtain

$$p^{[l]}(t) = \sum_{k=1}^{t} \left(T s^{[l-1]}(k) - s^{[l]}(k-1)\right)$$

Let now $i \in [n]$. Since $s^{[l]}(0) = 0$, we have $\sum_{k=1}^{t} s_i^{[l]}(k-1) \leq t-1$. Now if and only if there is any $t_0 \in [T]$ with $s_i^{[l-1]}(t_0) = 1$, we get

$$p_i^{[l]}(t') = T \sum_{k=1}^{t'} s_i^{[l-1]}(k) - \sum_{k=1}^{t'} s_i^{[l]}(k-1) \geq 1$$

for $t' \geq t_0$ and hence $s_i^{[l]}(t') = 1$. Just as before we now get the required property for the whole network by induction over the layers.

For a clearer construction of our networks we will additionally define neurons in our network:

Definition 2.5. The ***i*-th neuron** of a hidden layer $\lambda = (W^{[l]}, b^{[l]}, V^{[l]}, u^{[l]}(0), i^{[l]}(0), \alpha^{[l]}, \beta^{[l]}, \vartheta^{[l]})$ of a r. LIF-SNN is a tuple (w, b, v, u_0, i_0) with $w \in \mathbb{R}^{n_{l-1}}$, $v \in \mathbb{R}^{n_l}$ and $b, u_0, i_0 \in \mathbb{R}$, such that b, u_0, i_0 are the i-th component of $b^{[l]}$, $u^{[l]}(0)$, $i^{[l]}(0)$ respectively and w, v are the i-th row vector of $W^{[l]}$, $V^{[l]}$ respectively.

2.4 Basic properties

In the following we present some technical but helpful notations and lemmas for writing proofs about SNN. Of particular importance are Definition 2.6 and Lemma 2.1, which introduce non-recursive formulas for the defining equations of r. LIF-SNNs.

Definition 2.6. Let $t \in [T]$, $l \in [L]$ and spike train families $\sigma = (\sigma^{[l']})_{l' \in [l]_0}$, $\sigma' = (\sigma'^{[l']})_{l' \in [l]_0}$ be given, such that $\forall_{l \in [l-1]_0} \sigma^{[l']} \in \{0,1\}^{n_{l'} \times t}$ and $\sigma^{[l]} \in \{0,1\}^{n_l \times t}$ as well as $\forall_{l' \in [l]_0} \sigma'^{[l']} \in \{0,1\}^{n_{l'} \times t}$, i.e. σ, σ' have to be chosen such that the terms in the following definitions are well-defined. We define

$$i^{[l]}(t;\sigma) := (\alpha^{[l]})^t i^{[l]}(0) + \sum_{k=1}^{t} (\alpha^{[l]})^{t-k} \left(W^{[l]} \sigma^{[l-1]}(k) + V^{[l]} \sigma^{[l]}(k-1) \right), \tag{7}$$

$$p^{[l]}(t;\sigma) := (\beta^{[l]})^t u^{[l]}(0) + \sum_{k=1}^{t} (\beta^{[l]})^{t-k} \left(i^{[l]}(k;\sigma) + b^{[l]} \right) - \vartheta \sum_{k=1}^{t-1} (\beta^{[l]})^{t-k} \sigma^{[l]}(k), \tag{8}$$

$$s^{[l]}(t;\sigma) := H(p^{[l]}(t;\sigma) - \vartheta \mathbf{1}_{n_l}), \tag{9}$$

$$u^{[l]}(t;\sigma') := (\beta^{[l]})^t u^{[l]}(0) + \sum_{k=1}^{t} (\beta^{[l]})^{t-k} \left(i^{[l]}(k;\sigma') + b^{[l]} - \vartheta \sigma'^{[l]}(k) \right). \tag{10}$$

Remark 2.4. The spike train families σ, σ' in Definition 2.6 are chosen such that they only include the data that is actually needed in the definition of $i^{[l]}, p^{[l]}, s^{[l]}, u^{[l]}$. This is also the reason why we have to use σ' for $u^{[l]}$; its definition requires in contrast to the definitions of $i^{[l]}, p^{[l]}, s^{[l]}$ the newest spikes from the current layer layer.

We will also allow using $i^{[l]}, p^{[l]}, s^{[l]}, u^{[l]}$ with extensions of the by the definition necessary spike train families, in particular we will use the functions with "complete" spike train families $(\sigma^{[l]})_{l \in [L]}$ with $\sigma^{[l]} \in \{0,1\}^{n_l \times T}$.

The notation for the previous definitions is justified due to

Lemma 2.1. *The non-recursive formulas from Definition 2.6 are equivalent to the recursive definitions for $l \in [L]$, $t \in [T]$ assuming previous spikes are equal: $\forall_{l' \in [l-1]_0} \sigma^{[l']} = s^{[l']}$ and $\forall_{t' \in [t-1]} \sigma^{[l]}(t') = s^{[l]}(t')$ as well as $\forall_{l' \in [l]_0} \sigma'^{[l']} = s^{[l']}$.*

Proof. We first proof $\forall_{t \in [T]} i^{[l]}(t;\sigma) = i^{[l]}(t)$ and $\forall_{t \in [T]} u^{[l]}(t;\sigma) = u^{[l]}(t)$ for $\sigma^{[l]} = s^{[l]}$ by induction: Let $t = 1$. We have

$$\begin{aligned} i^{[l]}(1;\sigma) &= \alpha^{[l]} i^{[l]}(0) + W^{[l]} \sigma^{[l-1]}(1) + V^{[l]} \sigma^{[l]}(0) \\ &= \alpha^{[l]} i^{[l]}(0) + W^{[l]} s^{[l-1]}(1) + V^{[l]} s^{[l]}(0) \\ &= i^{[l]}(1) \end{aligned}$$

and

$$\begin{aligned} u^{[l]}(1;\sigma') &= \beta^{[l]} u^{[l]}(0) + i^{[l]}(1;\sigma') + b^{[l]} - \vartheta \sigma'^{[l]}(1) \\ &= \beta^{[l]} u^{[l]}(0) + i^{[l]}(1) + b^{[l]} - \vartheta s^{[l]}(1) \\ &= p^{[l]}(1) - \vartheta s^{[l]}(1) \\ &= u^{[l]}(1) \end{aligned}$$

by using the definitions and using our assumption $\sigma^{[l]} = s^{[l]}$. We have $i^{[l]}(1;\sigma') = i^{[l]}(1)$, since σ' is an "extension" of σ.

Let further $t > 1$. We may assume $i^{[l]}(t-1;\sigma) = i^{[l]}(t-1)$ and $u^{[l]}(t-1;\sigma') = u^{[l]}(t-1)$ and obtain

$$\begin{aligned} i^{[l]}(t;\sigma) &= (\alpha^{[l]})^t i^{[l]}(0) + \sum_{k=1}^{t} (\alpha^{[l]})^{t-k} \left(W^{[l]} \sigma^{[l-1]}(k) + V^{[l]} \sigma^{[l]}(k-1) \right) \\ &= \alpha^{[l]} i^{[l]}(t-1;\sigma) + \left(W^{[l]} \sigma^{[l-1]}(t) + V^{[l]} \sigma^{[l]}(t-1) \right) \\ &= \alpha^{[l]} i^{[l]}(t-1) + W^{[l]} s^{[l-1]}(t) + V^{[l]} s^{[l]}(t-1) \\ &= i^{[l]}(t) \end{aligned}$$

and similarly

$$\begin{aligned}
u^{[l]}(t;\sigma') &= (\beta^{[l]})^t u^{[l]}(0) + \sum_{k=1}^{t} (\beta^{[l]})^{t-k} \left(i^{[l]}(k;\sigma') + b^{[l]} - \vartheta \sigma'^{[l]}(k) \right) \\
&= \beta^{[l]} u^{[l]}(t-1;\sigma') + \left(i^{[l]}(t;\sigma') + b^{[l]} - \vartheta \sigma'^{[l]}(t) \right) \\
&= \beta^{[l]} u^{[l]}(t-1) + i^{[l]}(t) + b^{[l]} - \vartheta s^{[l]}(t) \\
&= p^{[l]}(t) - \vartheta s^{[l]}(t) \\
&= u^{[l]}(t).
\end{aligned}$$

By substituting $u^{[l]}$ in $p^{[l]}$ we further obtain

$$\begin{aligned}
p^{[l]}(t;\sigma) &= (\beta^{[l]})^t u^{[l]}(0) + \sum_{k=1}^{t} (\beta^{[l]})^{t-k} \left(i^{[l]}(k;\sigma) + b^{[l]} \right) - \vartheta \sum_{k=1}^{t-1} (\beta^{[l]})^{t-k} \sigma^{[l]}(k), \\
&= \beta^{[l]} u^{[l]}(t-1;\sigma) + \left(i^{[l]}(t;\sigma) + b^{[l]} \right) \\
&= \beta^{[l]} u^{[l]}(t-1) + i^{[l]}(t) + b^{[l]} \\
&= p^{[l]}(t).
\end{aligned}$$

By using the above we obtain

$$s^{[l]}(t;\sigma) = H(p^{[l]}(t;\sigma) - \vartheta \mathbf{1}_{n_l}) = H(p^{[l]}(t) - \vartheta \mathbf{1}_{n_l}) = s^{[l]}(t).$$

□

Lemma 2.2. *Let $a, x \in \mathbb{R}$, $a \neq 0$ and $b \in \mathbb{Z}$. We then have $\lfloor \frac{x}{a} \rfloor = b \Leftrightarrow 0 \leq x - ab < a$.*

Proof. $0 \leq x - ab < a$ is equivalent to $b \leq \frac{x}{a} < b + 1$, which is yet again equivalent to $\lfloor \frac{x}{a} \rfloor = b$ by definition of $\lfloor \cdot \rfloor$. □

Lemma 2.3. *Let $t_0, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_\omega$. If $\beta = 1$, then $0 \leq u_j^{[l]}(t_\omega) < \vartheta$ holds if and only if*

$$\left\lfloor \frac{1}{\vartheta} \left(u_j^{[l]}(t_0 - 1) + \sum_{t=t_0}^{t_\omega} (i_j^{[l]}(t) + b_j^{[l]}) \right) \right\rfloor = \sum_{t=t_0}^{t_\omega} s_j^{[l]}(t). \tag{11}$$

Proof. We have

$$u_j^{[l]}(t_\omega) = u_j^{[l]}(t_0 - 1) + \sum_{k=t_0}^{t_\omega} \left(i_j^{[l]}(k) + b_j^{[l]} - \vartheta s_j^{[l]}(k) \right)$$

by Lemma 2.1. So (11) is equal to $0 \leq u_j^{[l]}(t_\omega) < \vartheta$ by Lemma 2.2. □

Lemma 2.4. *Let $t \in [T]$, $l \in [L]$ and $j \in [n_l]$. We then have $u_j^{[l]}(t) \geq 0 \Leftrightarrow p_j^{[l]}(t) \geq 0$.*

Proof. Let $u_j^{[l]}(t) \geq 0$. Then $p_j^{[l]}(t) = u_j^{[l]}(t) + \vartheta s_j^{[l]}(t) \geq 0$.

If we know $p_j^{[l]}(t) \geq 0$ instead, then suppose $u_j^{[l]}(t) < 0$. Since $p_j^{[l]}(t) \neq u_j^{[l]}(t)$, $s_j^{[l]}(t) = 1$ and therefore $p_j^{[l]}(t) \geq \vartheta$. But this means $u_j^{[l]}(t) = p_j^{[l]}(t) - \vartheta \geq 0$. □

Lemma 2.5. *Let $t_0, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_\omega$. If $\forall_{t \in \{t_0+1 \dots t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \geq 0$ and $u_j^{[l]}(t_0) \geq 0$, then $u_j^{[l]}(t_\omega) \geq 0$.*

Proof. Suppose there is a t, $t_0 \leq t \leq t_\omega$ with $u_j^{[l]}(t) < 0$. W.l.o.g. we can assume t to be minimal. Clearly $t \neq t_0$, since this contradicts our assumption. So we have $u_j^{[l]}(t-1) \geq 0$ and $i_j^{[l]}(t) + b_j^{[l]} \geq 0$. So from

$$0 > u_j^{[l]}(t) = p_j^{[l]}(t) - \vartheta s^{[l]}(t) = \beta^{[l]} u_j^{[l]}(t-1) + \beta(i_j^{[l]}(t) + b_j^{[l]}) - \vartheta s_j^{[l]}(t).$$

we conclude $s_j^{[l]}(t) = 1$ and $p_j^{[l]}(t) \geq \vartheta$. But this means $u_j^{[l]}(t) \geq 0$ by Lemma 2.4. □

Lemma 2.6. *Let $t_0, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_\omega$. If $\forall_{t \in \{t_0, \dots, t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \leq \vartheta$ and $u_j^{[l]}(t_0 - 1) < \vartheta$, then*

$$\forall_{t \in \{t_0 - 1, \dots, t_\omega\}} u_j^{[l]}(t) < \vartheta. \tag{12}$$

Proof. We proof (12) by induction over $t \in \{t_0 - 1, \dots, t_\omega\}$. The base case is given by assumption. Let further $t \in \{t_0 \dots t_\omega\}$. By definition of $p^{[l]}$, the given assumptions and the induction hypothesis we get

$$p_j^{[l]}(t) = \beta^{[l]} u_j^{[l]}(t-1) + i_j^{[l]}(t) + b_j^{[l]} \leq u_j^{[l]}(t-1) + i_j^{[l]}(t) + b_j^{[l]} < 2\vartheta$$

We further get $u_j^{[l]}(t) < \vartheta$ by definition of $u^{[l]}$ and $s^{[l]}$. □

Lemma 2.7. *Let $t_0, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_\omega$. If $\forall_{t \in \{t_0 + 1 \dots t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \leq 0$ and $u_j^{[l]}(t_\omega) \geq \vartheta$, then $\forall_{t \in \{t_0 \dots t_\omega\}} u_j^{[l]}(t) \geq \vartheta$.*

Proof. Suppose there is a t, $t_0 \leq t \leq t_\omega$, with $u_j^{[l]}(t) < \vartheta$. Let t be maximal with this property. Clearly $t \neq t_\omega$ by assumption. So we have a contradiction by

$$\vartheta \leq u_j^{[l]}(t+1) - \beta^{[l]}(i_j^{[l]}(t+1) + b_j^{[l]}) + \vartheta s_j^{[l]}(t) = u_j^{[l]}(t).$$

□

Lemma 2.8. *Let $t_0, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_\omega$. If*

$$0 \leq (\beta^{[l]})^{t_\omega - t_0} u_j^{[l]}(t_0) + \sum_{t=t_0+1}^{t_\omega} (\beta^{[l]})^{t_\omega - t} \left(i_j^{[l]}(t) + b_j^{[l]} \right)$$

and either $\forall_{t \in \{t_0+1, \dots, t_\omega\}} s_j^{[l]}(t) = 0$ or $\forall_{t \in \{t'+1, \dots, t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \geq 0$, where t' is the maximal time $\leq t_\omega$ such that $s_j^{[l]}(t') = 1$, then $u_j^{[l]}(t_\omega) \geq 0$.

Proof. If $\forall_{t \in \{t_0 \dots t_\omega\}} s_j^{[l]}(t) = 0$, we get by assumption

$$u_j^{[l]}(t_\omega) = u_j^{[l]}(t_\omega) + \sum_{t=t_0+1}^{t_\omega} (\beta^{[l]})^{t_\omega - t} s_j^{[l]}(t) = (\beta^{[l]})^{t_\omega - t_0} u_j^{[l]}(t_0) + \sum_{t=t_0+1}^{t_\omega} (\beta^{[l]})^{t_\omega - t} \left(i_j^{[l]}(t) + b_j^{[l]} \right) \geq 0.$$

Is there on the other hand a t' with $s_j^{[l]}(t') = 1$, then let t' be maximal $\leq t_\omega$. By assumption we now have $\forall_{t \in \{t'+1 \dots t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \geq 0$. Since $s_j^{[l]}(t') = 1$ we further have $p_j^{[l]}(t') \geq \vartheta$, so by Lemma 2.4 $u_j^{[l]}(t') \geq 0$ such that we can conclude by Lemma 2.5. □

Proposition 2.1. *Let $\beta = 1$, $t_0, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_\omega$. Suppose $u_j^{[l]}(t_0 - 1) < \vartheta$ and further $\forall_{t \in \{t_0, \dots, t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \leq \vartheta$ and $0 \leq u_j^{[l]}(t_0 - 1) + \sum_{t=t_0}^{t_\omega} \left(i_j^{[l]}(t) + b_j^{[l]} \right)$. If moreover either $\forall_{t \in \{t_0, \dots, t_\omega\}} s_j^{[l]}(t) = 0$ or $\forall_{t \in \{t'+1, \dots, t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} \geq 0$, where t' is the last time $\leq t_\omega$ such that $s_j^{[l]}(t') = 1$, then*

$$\sum_{t=t_0}^{t_\omega} s_j^{[l]}(t) = \left\lfloor \frac{1}{\vartheta} \left(u_j^{[l]}(t_0 - 1) + \sum_{t=t_0}^{t_m} \left(i_j^{[l]}(t) + b_j^{[l]} \right) \right) \right\rfloor.$$

Proof. It suffices to show $0 \leq u_j^{[l]}(t_\omega) < \vartheta$ due to Lemma 2.3. We get $0 \leq u_j^{[l]}(t_\omega)$ due to Lemma 2.8 and $u_j^{[l]}(t_\omega) < \vartheta$ due to Lemma 2.6.

□

Proposition 2.2. *Let $\beta = 1$, $t_0, t_m, t_\omega \in [T]$ and $j \in [n_l]$ for an $l \in [L]$ such that $t_0 \leq t_m \leq t_\omega$. If $\forall_{t \in \{t_m+1,\dots,t_\omega\}} i_j^{[l]}(t) + b_j^{[l]} = 0$, $u_j^{[l]}(t_m) \geq 0$ and $0 \leq u_j^{[l]}(t_0 - 1) + \sum_{t=t_0}^{t_m} \left(i_j^{[l]}(t) + b_j^{[l]}\right) \leq \vartheta(t_\omega - t_m + 1)$, then*

$$\sum_{t=t_0}^{t_\omega} s_j^{[l]}(t) = \left\lfloor \frac{1}{\vartheta} \left(u_j^{[l]}(t_0 - 1) + \sum_{t=t_0}^{t_m} \left(i_j^{[l]}(t) + b_j^{[l]}\right) \right) \right\rfloor.$$

Proof. It suffices to show $0 \leq u_j^{[l]}(t_\omega) < \vartheta$ due to Lemma 2.3. We get $u_j^{[l]}(t_\omega) \geq 0$ from Lemma 2.5 using our assumptions, in particular $u_j^{[l]}(t_m) \geq 0$. Furthermore, $u_j^{[l]}(t_\omega) < \vartheta$; otherwise, if $u_j^{[l]}(t_\omega) \geq \vartheta$, then we had $\forall_{t \in \{t_m \dots t_\omega\}} u_j^{[l]}(t) \geq \vartheta$ by Lemma 2.7 and therefore in particular $\forall_{t \in \{t_m \dots t_\omega\}} s_j^{[l]}(t) = 1$, from which we get

$$\begin{aligned}
u_j^{[l]}(t_\omega) &= u_j^{[l]}(t_0 - 1) + \sum_{k=t_0}^{t_\omega} \left(i^{[l]}(k) + b^{[l]} - \vartheta s^{[l]}(k)\right) \\
&= u_j^{[l]}(t_0 - 1) + \sum_{k=t_0}^{t_m} \left(i^{[l]}(k) + b^{[l]}\right) - \vartheta \sum_{k=t_0}^{t_\omega} s^{[l]}(k) \\
&\leq \vartheta(t_\omega - t_m + 1) - \vartheta(t_\omega - t_m + 1) - \vartheta \sum_{k=t_0}^{t_m - 1} s^{[l]}(k) \\
&\leq 0
\end{aligned}$$

contradicting $u_j^{[l]}(t_\omega) \geq \vartheta$. □

3 Structure of computations in r. LIF-SNNs

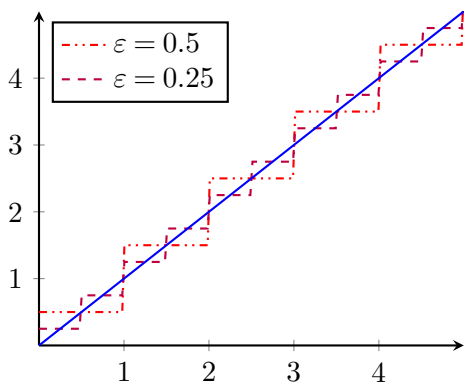
When working with neural networks a fundamental question is how well they are able to approximate functions. Towards that end the following theorem was proved in [Nguyen et al., 2025].

Theorem 3.1. *Let f be a continuous function on a compact set $\Omega \subset \mathbb{R}^{n_0}$. For all $\varepsilon > 0$, there exists a d.t. LIF-SNN Φ with direct encoding, membrane potential output, $L = 2$ and $T = 1$ such that*

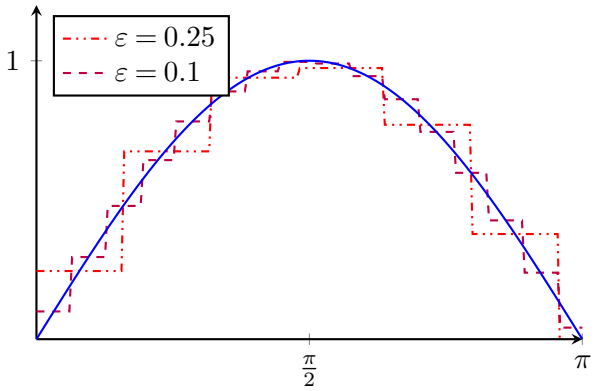
$$\|(R(\Phi) - f)|_\Omega\|_\infty \leq \varepsilon$$

Moreover, if f is Γ-Lipschitz, then Φ can be chosen with width parameter $n = (n_1, n_2)$ given by

$$n_1 = \left(\max\left\{\left\lceil \frac{\operatorname{diam}_\infty(\Omega)}{\varepsilon}\Gamma \right\rceil, 1\right\} + 1\right) n_0,$$
$$n_2 = \max\left\{\left\lceil \frac{\operatorname{diam}_\infty(\Omega)}{\varepsilon}\Gamma \right\rceil^{n_0}, 1\right\}.$$



(a) A d.t. LIF-SNN approximating the identity



(b) d.t. LIF-SNNs approximating a sinus wave

The proof of Theorem 3.1 first shows that a continuous function can be arbitrarily approximated by step functions, in particular by step functions constant on hypercubes in Ω. Then a d.t. LIF-SNN is constructed by using the first layer to partition the input space along hyperplanes into cubes and the second layer to assign values to the hypercubes.

While quite simple, this construction does not use the unique feature of d.t. LIF-SNNs/r. LIF-SNNs, the ability of neurons to accumulate state over time. It therefore needs quite a lot more neurons than actually needed for many functions with (almost) linear segments, like a sinus wave. E.g. in Fig. 3.1b a neuron is needed for every constant region of the graphs in the first and second layer each.

We will now show a more efficient construction for r. LIF-SNN using the fact that r. LIF-SNN can quite efficiently approximate linear segments. The general intuition behind it is to use piece-wise linear functions to approximate continuously differentiable functions and then construct a r. LIF-SNN approximating the piece-wise linear function by discretizing the input dimensions into spike trains in the first layer that are consumed by groups of neurons in the second layer, one group for each almost linear segment.

To state our theorem we first need to define the notions of "modulus of uniform continuity" and "generalized inverse of a modulus of uniform continuity" and proof some simple properties:

Definition 3.1. Let M, N be metric spaces. A modulus of uniform continuity of a uniformly continuous function $f : M \to N$ is a function $\omega : [0, \infty] \to [0, \infty]$, such that it vanishes at 0, i.e. $\lim_{x \to 0} \omega(x) = 0$, and

$$\forall_{x,y \in M} d_N(f(x), f(y)) \leq \omega(d_M(x, y)).$$

The generalized inverse of ω is defined as

$$\omega^\dagger(s) := \inf\{t \in [0,\infty] \mid \omega(t) > s\}.$$

Lemma 3.1. *Let $\omega : [0,\infty] \to [0,\infty]$ be a modulus of uniform continuity of a uniformly continuous function $f : M \to N$, where M, N are metric spaces.*

We have the following properties

1. $\forall_{x,y\in M, s\in[0,\infty]} d_M(x,y) \leq \omega^\dagger(s) \Rightarrow d_N(f(x),f(y)) \leq s.$
2. $\forall_{s\in[0,\infty]} s = 0 \Leftrightarrow w^\dagger(s) = 0.$

Proof.

1. Let $x, y \in M$ and $s \in [0,\infty]$ be given such that $d_M(x,y) \leq \omega^\dagger(s)$. By definition of $\omega^\dagger$, this means $\omega(d_M(x,y)) \leq s$. Since ω is a modulus of uniform continuity of f, we have $d_N(f(x),f(y)) \leq \omega(d_M(x,y))$ and therefore overall $d_N(f(x),f(y)) \leq s$.
2. Since ω is a modulus of uniform continuity, it is by definition continuous at 0. Let us choose an arbitrary sequence $(t_n)_{n\in\mathbb{N}}$ with $t_n \to 0$. Then $\omega(t_n) \to 0$ and therefore $\omega^\dagger(0) \leq \inf_{n\in\mathbb{N}} t_n = 0$.

   Is on the other hand $\omega^\dagger(s) = 0$, then there is a sequence $(t_n)_{n\in\mathbb{N}}$ with $t_n \to 0$ and therefore $\omega(t_n) \to 0$. By definition of $\omega^\dagger(s)$, we have $\omega(t_n) > s$, so we get $s = 0$.

□

Let us now state our theorem:

Theorem 3.2. *Let $f \in \mathcal{C}^0(C,\mathbb{R}^m)$ be defined on a half-open cube $C \neq \emptyset$, such that $f|_{C^\circ} \in \mathcal{C}^1(C^\circ,\mathbb{R}^m)$ is continuously differentiable with bounded differential, i.e. $\|d(f|_{C^\circ})\|_{\infty,2} < \infty$. For all $\varepsilon,\mu,\nu > 0$, $\varepsilon = \mu + \nu$, there exists a r. LIF-SNN Φ with $L = 2$ and*

$$\begin{aligned} T &= (K(\mu)+1)T_r(\nu)+2 \\ n_1 &= n+1 \\ n_2 &= K(\mu)^n(n+1)+3 \end{aligned}$$

such that

$$\|R(\Phi)|_C - f\|_{\infty,2} \leq \varepsilon.$$

Where we use

$$\begin{aligned} T_r := T_r(\nu) &:= \max\left(2, \left\lceil \sqrt{n}\frac{\mathrm{diam}_\infty(C)}{K}\frac{\|d(f|_{C^\circ})\|_{\infty,2}}{\nu}\right\rceil\right), \\ K := K(\mu) &:= \min_{\substack{\xi,\theta>0 \\ \xi\theta=\mu}} \left\{ \left\lceil \frac{\mathrm{diam}_\infty(C)}{\frac{2}{\sqrt{n}}\min(\omega^\dagger(\xi),\theta)} \right\rceil \right\}. \end{aligned}$$

Here $\omega^\dagger$ is the generalized inverse of a modulus of uniform continuity with regard to $\|\cdot\|_2$ of the total derivative $d(f|_{C^\circ})$. $d(f|_{C^\circ})$ is uniformly continuous since it is bounded. Since $\xi \neq 0$, we have $\omega^\dagger(\xi) > 0$ by Lemma 3.1. Further there are ξ,θ such that the minimum in the definition of K^n is obtained, since we the minimum is taken over the set of natural numbers, so K is well-defined. Moreover T_r is well-defined, since $C \neq 0$ and therefore $\mathrm{diam}_\infty(C) \neq 0$ and $K \neq 0$.

Remark 3.1. In our construction μ and ν determine whether to optimize the number of neurons or the number of time-steps. As we see later, K^n corresponds to the number of subcubes we will split C into such that f is almost linear on each of them. From the definition it is clear that K and therefore the number of neurons only depend on f through $\omega^\dagger(\xi)$, which is essentially a measure of how strongly the slope of f is changing and therefore into how many subcubes we need to split C to get sufficiently almost affine linear regions of f.

We will further see that T_r corresponds to the number of constant intervals with which we approximate f on the almost affine linear regions. It is therefore to be expected that T_r depends on the width $\frac{\operatorname{diam}_\infty(C)}{K}$ of the subcubes and the maximal slope $\|d(f|_{C^\circ})\|_{\infty,2}$.

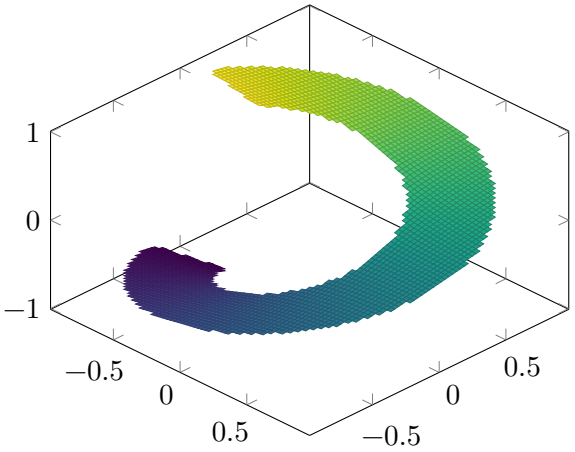


Figure 3.2: A spiral staircase function

Remark 3.2. Theorem 3.2 cannot be generalized to arbitrary input spaces like compact sets: Consider a spiral staircase function like shown in Fig. 3.2 defined on a compact set $\Omega := \{(r\cos(\varphi), r\sin(\varphi)) \mid 0.5 \leq r \leq 1,\ |\varphi - \pi| \geq 0.2\}$. Such a function is clearly continuous and continuously differentiable on its interior Ω°, such that in particular the differential is bounded, $\|d(f|_{\Omega^\circ})\|_{\infty,2} < \infty$.

For our construction of the SNN we now need an encompassing half-open rectangle of Ω, let e.g. $C = [0,2)^2$. Now in Lemma 3.2 we will partition C into sub-cubes, such that f is almost linear on those w

We need the half-open cube C with $\Omega \subset C \subset U$ to correctly choose T_r and K: Suppose we regard the function $f : ((0,1) \cup (2,3)) \to \mathbb{R}$, $x \mapsto \chi_{[0,1]}(x)$. This function is certainly continuously differentiable, in particular, all derivatives are zero. In our construction of the r. LIF-SNN we use a cube in the input space as our canvas, Suppose, K would depend on the modulus of uniform continuity of df on U and T_r on $\|df|_U\|_{\infty,2}$, without the a half-open cube C with $\Omega \subset C \subset U$

Remark 3.3. If $f \in \mathcal{C}^1(U, \mathbb{R}^m)$ is given with $\emptyset \neq U \subset \mathbb{R}^n$ and a compact subset $\Omega \subset \mathbb{R}^n$, we can extend $f|_\Omega$ to a function $df \in \mathcal{C}^1(\mathbb{R}^n, \mathbb{R}^m)$: There is a partition of one $\varphi_1, \varphi_2 \in \mathcal{C}^\infty(\mathbb{R}^n)$ subordinate to U and $\mathbb{R}^n \setminus \Omega$. We get $\varphi_1|_\Omega = 1$ and therefore $(\varphi_1 f)|_\Omega = f|_\Omega$. $f' := \varphi_1 f$ is further clearly $\mathcal{C}^1(U, \mathbb{R}^m)$.

Remark 3.4. If $f \in \mathcal{C}^1(U, \mathbb{R}^m)$ is given with $\emptyset \neq U \subset \mathbb{R}^n$ and a compact $\Omega \subset \mathbb{R}^n$, but with no half-open cube C such that $\Omega \subset \overline{C} \subset U$, we can extend $f|_\Omega$ to a function $df \in \mathcal{C}^1(\mathbb{R}^n, \mathbb{R}^m)$: There is a partition of one $\varphi_1, \varphi_2 \in \mathcal{C}^\infty(\mathbb{R}^n)$ subordinate to U and $\mathbb{R}^n \setminus \Omega$. We get $\varphi_1|_\Omega = 1$ and therefore $(\varphi_1 f)|_\Omega = f|_\Omega$. $f' := \varphi_1 f$ is further clearly $\mathcal{C}^1(U, \mathbb{R}^m)$.

We first proof that continuous differentiable functions with uniform continuous differential can be efficiently approximated by piece-wise linear function. Compare e.g. Section 3 to Fig. 3.1b

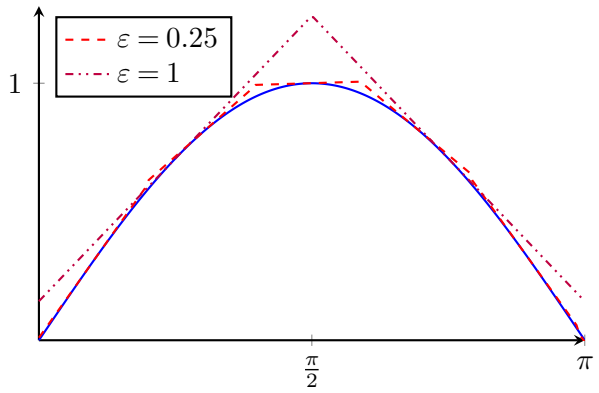


Figure 3.3: Piece-wise affine linear functions approximating the sinus

Lemma 3.2. *Let $f \in \mathcal{C}^0(C, \mathbb{R}^m)$ be a continuous function defined on half-open cube C, such that $f|_{C^\circ} \in \mathcal{C}^1(C^\circ, \mathbb{R}^m)$ is continuously differentiable and $d(f|_{C^\circ})$ is uniformly continuous. Let further $K(\mu)$ be defined as in Theorem 3.2.*

For every $\mu > 0$ we can compose C into $K^n := K(\mu)^n$ half-open subcubes $(C^{(j)})_{j \in [K^n]}$ such that affine linear functions $g^{(j)} : C^{(j)} \to \mathbb{R}^m$ exist with $\left\|d(g^{(j)})\right\|_{\infty,2} \leq \|df\|_{\infty,2}$ and $\|f - g\|_{\infty,2} < \mu$, where $g := \sum_{i=1}^m g^{(j)} \chi_{C^{(j)}}$.

Proof of Lemma 3.2. Let $\mu > 0$ be given. Let ω be a modulus of uniform continuity of $df|_{C^\circ}$. We will now partition C in K^n half-open subcubes with K defined as in Theorem 3.2. Let ξ, Θ be given, such that the minium is attained. Then the subcubes have width $w := \frac{\operatorname{diam}_\infty(C)}{K} \leq \frac{2}{\sqrt{n}} \min(\omega^\dagger(\xi), \theta)$.

Let us further define $g^{(j)} : C^{(j)} \to \mathbb{R}^m$ by $g^{(j)}(x) := f(c^{(j)}) + df_{c^{(j)}}(x - c^{(j)})$ where $c^{(j)}$ is the center of $C^{(j)}$, so in particular for all $x \in C^{(j)}$

$$\left\|x - c^{(j)}\right\|_2 = \sqrt{\sum_{i=1}^n \left|(x - c^{(j)})_i\right|^2} \leq \frac{w}{2}\sqrt{n} \leq \min(\omega^\dagger(\xi), \theta). \tag{13}$$

Now by definition of $g^{(j)}$ we already have $\left\|d(g^{(j)})\right\|_{\infty,2} = \|df_{c^{(j)}}\| \leq \|df\|_{\infty,2}$.

It suffices now to show $\left\|f|_{C^{(j)}} - g^{(j)}\right\|_\infty < \mu$. Let $x \in C^{(j)}$ and $h(t) := f(t(x - c^{(j)}) + c^{(j)})$. We then have

$$h'(t) = (df_{(x-c^{(j)})t+c^{(j)}} \circ d(t \mapsto t(x - c^{(j)}) + c^{(j)})_t)(1) = df_{(x-c^{(j)})t+c^{(j)}}(x - c^{(j)}).$$

We obtain by the fundamental theorem of calculus:

$$\begin{aligned}
\left\|f(x) - g^{(j)}(x)\right\|_2 &= \left\|f(x) - f(c^{(j)}) - df_{c^{(j)}}(x - c^{(j)})\right\|_2 \\
&= \left\|h(1) - h(0) - df_{c^{(j)}}(x - c^{(j)})\right\|_2 \\
&= \left\|\int_0^1 df_{(x-c^{(j)})t+c^{(j)}}(x - c^{(j)})dt - df_{c^{(j)}}(x - c^{(j)})\right\|_2
\end{aligned}$$

Due to the generalized Minkowski-Inequality we can move the norm inside the integral:

$$\begin{aligned}
&\leq \int_0^1 \left\| df_{(x-c^{(j)})t+c^{(j)}}(x-c^{(j)}) - df_{c^{(j)}}(x-c^{(j)}) \right\|_2 dt \\
&= \int_0^1 \left\| (df_{(x-c^{(j)})t+c^{(j)}} - df_{c^{(j)}})(x-c^{(j)}) \right\|_2 dt \\
&\leq \int_0^1 \left\| df_{(x-c^{(j)})t+c^{(j)}} - df_{c^{(j)}} \right\|_2 \left\| x-c^{(j)} \right\|_2 dt \\
&\leq \int_0^1 \xi \left\| x-c^{(j)} \right\|_2 dt \\
&= \xi \left\| x-c^{(j)} \right\|_2 \\
&\leq \xi\theta \\
&= \mu
\end{aligned}$$

In the fourth step we use $\left\| df_{(x-c^{(j)})t+c^{(j)}} - df_{c^{(j)}} \right\| \leq \xi$, which holds due to $\forall_{t\in[0,1]}(x-c^{(j)})t+c^{(j)} \in C^{(j)}$, (13) and Lemma 3.1. □

Proof of Theorem 3.2.. Let there be $\varepsilon, \mu, \nu > 0$ with $\varepsilon = \mu + \nu$. By Lemma 3.2 we have a composition K^n of C into half-open subcubes $(C^{(j)})_{i=1..K^n}$ and linear functions $g^{(j)} : C^{(j)} \to \mathbb{R}^m$, such that $\left\| d(g^{(j)}) \right\|_{\infty,2} \leq \|df\|_{\infty,2}$ and $\|f-g\|_\infty < \mu$ for $g := \sum_{i=1}^m g^{(j)} \chi_{C^{(j)}}$.

We will now define a r. LIF-SNN Φ with direct input encoding and membrane-potential outputs such that $\|R(\Phi)|_C - g\|_\infty < \nu$.

Let us first set the basic parameters $i^{[l]}(0) = 0$, $\alpha^{[l]} = 0$ and $\beta^{[l]} = \vartheta^{[l]} = 1$ for all layers.

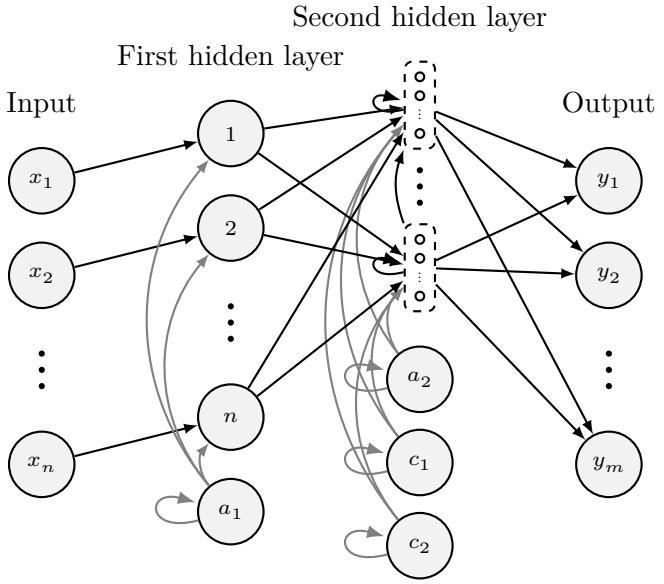


Figure 3.4: Structure of the whole network

The intuitive idea for the construction of the network is the following: We have five phases. We define

$$\begin{aligned}
T_1 &:= \{1, ..., KT_r\}, \qquad T_2 := \{KT_r\}, \qquad T_3 := \{KT_r + 1\}, \\
T_4 &:= \{KT_r + 2\}, \qquad T_5 := \{KT_r + 3, ..., T\}.
\end{aligned}$$

For ease of notation, will also use T_2, T_3, T_4 as if they were numbers.

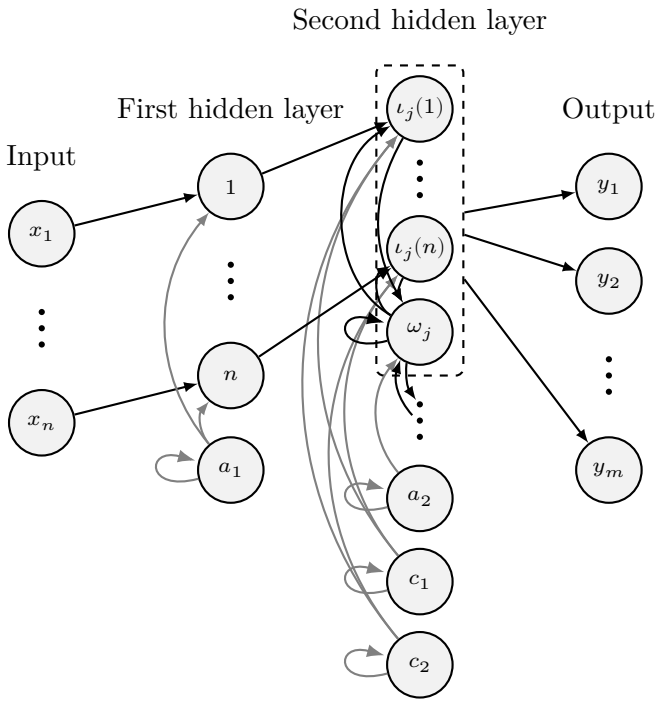
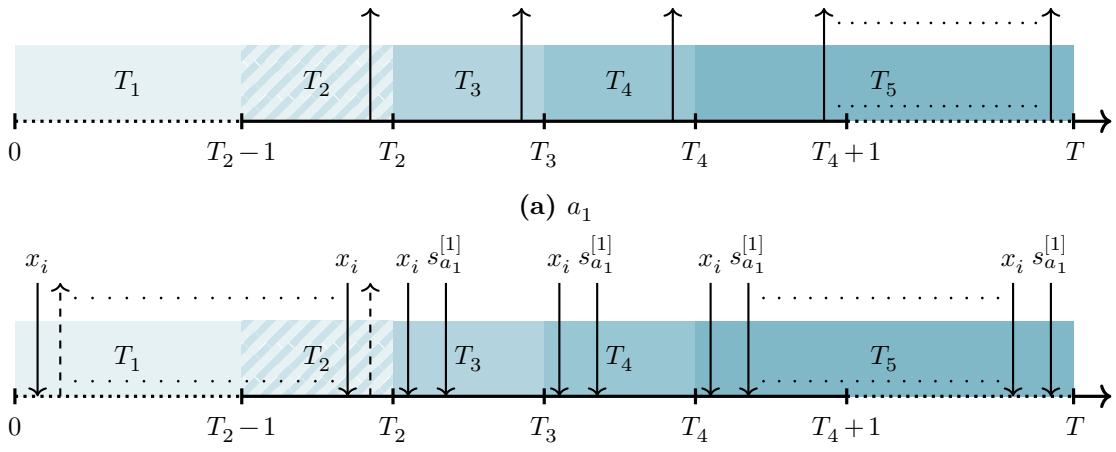


Figure 3.5: Structure of the network, focused on the j-th group of the second layer

The first layer is only active during the first phase, T_1. It is composed of $n+1$ neurons, where the first n neuron convert the input vector regarding its position in C into spike trains. The last neuron, the "alarm clock", shuts down the first layer after T_1 ends.

The second layer only accumulates state without spike during $T_1 \setminus T_2$. Then during $T_2 \cup T_3 \cup T_4$ it is decided in which region $C^{(j)}$ the input x is located. Further during $T_4 \cup T_5$ the location in $C^{(j)}$ is encoded through spikes.

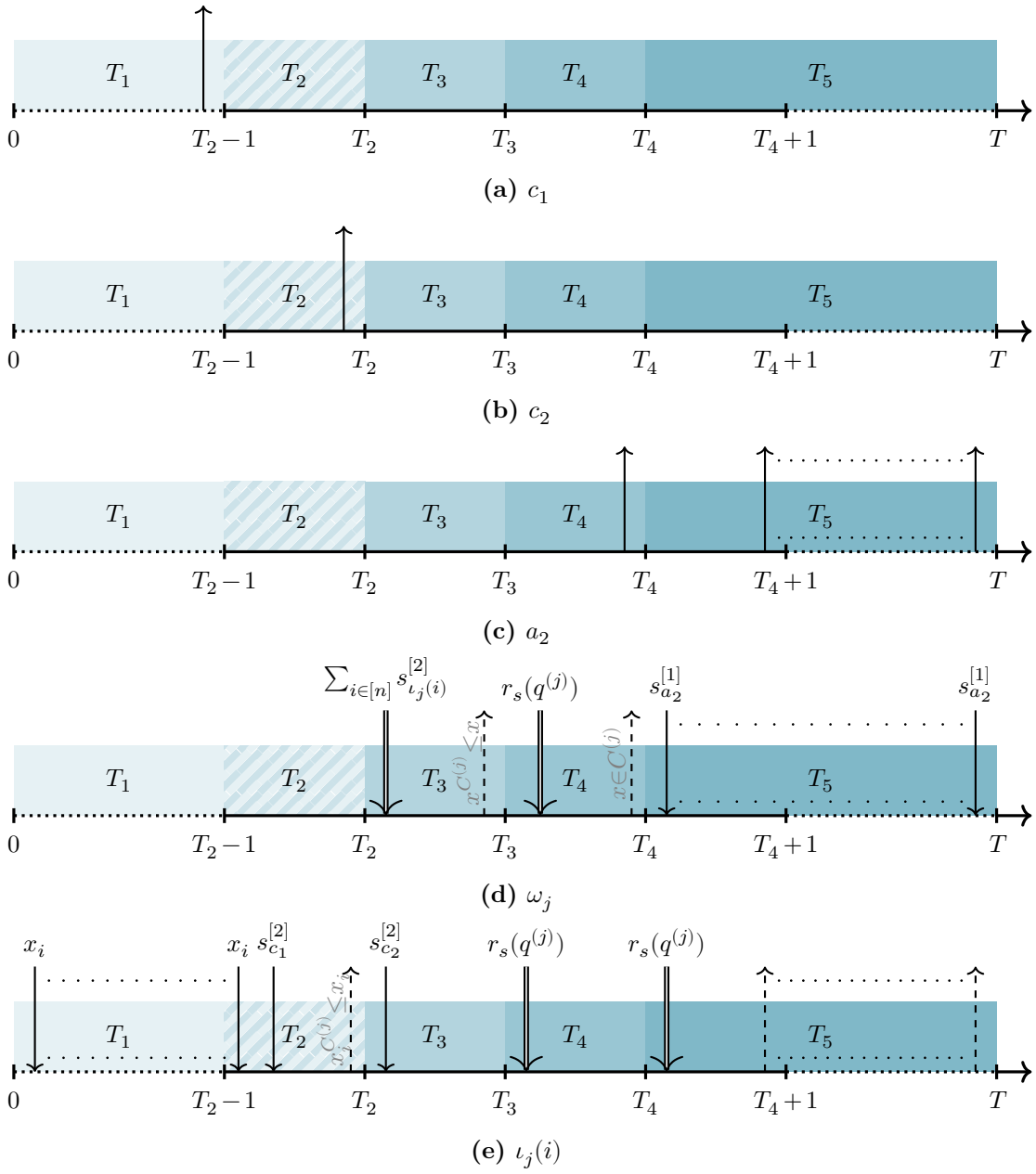
For each region $C^{(j)}$ we have $n+1$-neurons in the second layer. Each of the first n neurons encodes a commponent of the linear part of $g^{(j)}$. They are also used to inform the $n+1$-th neuron of the group if the x has at least as big as the base point of $C^{(j)}$. The $n+1$-th deactivates all other neurons of regions with smaller base point and encodes the constant part of $g^{(j)}$. The last 3 neurons act as "clock neurons" enabling and disabling the other ones.



(a) a_1

(b) i

Figure 3.6: Timelines of first layer neurons



(a) c_1 **(b)** c_2 **(c)** a_2 **(d)** ω_j **(e)** $\iota_j(i)$

Figure 3.7: Timelines of second layer neurons

To obtain the normalized location of a value in C we will often us

$$o_i(z) = \frac{z_i - x_i^C}{y_i^C - x_i^C}$$

with $i \in [n_1]$ in the following.

1. **First layer**: We define the **i-th neuron** of the n neurons of the first layer by parameters

$$w = \frac{1}{y_i^C - x_i^C} e_i, \quad b = -\frac{x_i^C}{y_i^C - x_i^C}, \quad v = -e_{a_1}, \quad u_0 = 0, \quad i_0 = 0. \tag{i}$$

The **"alarm neuron"** of the first layer, with index $a_1 := n+1$, is defined by:

$$w = 0, \quad b = \frac{1}{T_2}, \quad v = e_{a_1}, \quad u_0 = 0, \quad i_0 = 0. \tag{a_1}$$

2. **Second layer**: Let us now construct the second layer in the following way: For each of the K^n subcubes in C we define $n+1$ neurons like so: Let $C^{(j)} = [\![x^{C^{(j)}}, y^{C^{(j)}}]\!)$ be one such subcube with position $q^{(j)} \in ([K-1]_0)^{n_1}$ in C, i.e.

$$\forall_{i\in[n_1]} q_i^{(j)} = Ko_i(x^{C^{(j)}}).$$

We will write $\iota_j(i) \coloneqq j(n+1)+i$ to index the first n neurons in the layer and $\omega_j \coloneqq (j+1)(n+1)$ to index the last neuron of each group.

The ***i*-th neuron** of the first n neurons (of the j-th group), with index $\iota_j(i)$ in the second layer, has the parameters

$$\begin{gathered} w = e_i, \quad b = 0, \quad v = T(e_{c_1} - 2e_{c_2} + r(q^{(j)})), \\ u_0 = -q_i^{(j)} T_r - T + 1, \quad i_0 = 0. \end{gathered} \tag{$\iota_j(i)$}$$

where "the switch" is

$$r(q) \coloneqq e_{\omega_{j(q)}} - \sum_{\substack{q'\in([K-1]_0)^{n_1} \\ q<q'}} e_{\omega_{j(q')}}.$$

with the index $j(q)$ of the subcube at position q. We further define the applied variant

$$r_s(q;t) \coloneqq \langle r(q), s^{[2]}(t)\rangle = s^{[2]}_{\omega_{j(q)}}(t) - \sum_{\substack{q'\in([K-1]_0)^{n_1} \\ q<q'}} s^{[2]}_{\omega_{j(q')}}(t).$$

The **final neuron** of the group, with index ω_j in its layer, has the parameters

$$w = 0, \quad b = 0, \quad v = \frac{1}{n}\sum_{i=1}^{n} e_{\iota_j(i)} - 2e_{a_2} + r(q^{(j)}), \quad u_0 = 0, \quad i_0 = 0. \tag{ω_j}$$

We also define the two **"clock neurons"**, with index $c_1 \coloneqq (n+1)K^n + 1$ and $c_2 \coloneqq (n+1)K^n + 2$ with parameters:

$$w = 0, \quad b = b_{c_i}, \quad v = -(T-1)e_{c_i}, \quad u_0 = 0, \quad i_0 = 0. \tag{c_1, c_2}$$

where $b_{c_1} = \frac{1}{T_2 - 1}$ and $b_{c_2} = \frac{1}{T_2}$. We further define the **"alarm neuron"**, with index $a_2 \coloneqq (j+1)K^n(\mu) + 3$, by

$$w = 0, \quad b = \frac{1}{T_4}, \quad v = e_{a_2}, \quad u_0 = 0, \quad i_0 = 0. \tag{a_2}$$

3. Output decoder: We further define the parameters of the output decoder by $a_t = 0$, for $t \le T_3$ and otherwise $a_t = 1$. We further set $b^{[L+1]} = 0$ and

$$(W^{[L+1]})_{k,\iota_j(i)} = d(g^{(j)})_k((y_i^{C^{(j)}} - x_i^{C^{(j)}})\frac{1}{T_r}e_i)$$

for $k \in [m]$, $j \in K^n$ and $i \in [n]$. Here $d(g^{(j)})$ is not only the total derivative, but also the linear part of $g^{(j)}$, i.e. $\forall_{x\in C^{(j)}} g^{(j)}(x) = d(g^{(j)})(x - x^{C^{(j)}}) + g(x^{C^{(j)}})$. We further set

$$(W^{[L+1]})_{k,\omega_j} = g_k(x^{C^{(j)}})$$

for $k \in [m]$ and $j \in K^n$. We finally define $W^{[L+1]}_{k,c_i} = 0$ for $i \in \{2..4\}$.

We will now proof that this construction indeed approximates g well enough. It will be helpful to consider the following, by choice of $i^{[l]}, \alpha^{[l]}, \beta^{[l]}, \vartheta^{[l]}$, simplified equations:

$$\begin{aligned} i^{[l]}(t) &= W^{[l]} s^{[l-1]}(t) + V^{[l]} s^{[l]}(t-1) \\ p^{[l]}(t) &= u^{[l]}(t-1) + i^{[l]}(t) + b^{[l]} \\ s^{[l]}(t) &= H(p^{[l]}(t) - \mathbf{1}_{n_l}) \\ u^{[l]}(t) &= p^{[l]}(t) - s^{[l]}(t) \end{aligned}$$

and in particular by Lemma 2.1

$$p^{[l]}(t) = u^{[l]}(0) + \sum_{k=1}^{t} \left(i^{[l]}(t) + b^{[l]} \right) - \sum_{k=1}^{t-1} s^{[l]}(k).$$

Let now $s^{[0]}(t) = x \in C$. We will proof $\|R(\Phi)(x) - g(x)\|_{\infty,2} \leq \nu$ in steps, by first characterizing the behavior of the first layer:

1. Characterization of the "alarm neuron" a_1:
   Let us first regard the neuron in the first layer: By choice of parameters we get:

   $$p_{a_1}^{[1]}(t) = \frac{t}{T_2} + \sum_{k=1}^{t} s_{a_1}^{[1]}(k-1) - \sum_{k=1}^{t-1} s_{a_1}^{[1]}(k) = \frac{t}{T_2}$$

   So $s_{a_1}^{[1]}(t) = 1 \Leftrightarrow t \geq T_2$.

2. Characterization of i-th neuron, $i \in [n]$:
   We have

   $$i_i^{[1]}(t) + b_i^{[1]} = \frac{x_i - x_i^C}{y_i^C - x_i^C} - s_{a_1}^{[1]}(t-1) = o_i(x) - s_{a_1}^{[1]}(t-1).$$

   So in particular

   $$0 \leq i_i^{[1]}(t) + b_i^{[1]} = o_i(x) \leq 1$$

   for $t \in T_1$, since $x_i^C \leq x_i < y_i^C$. Because further $0 \leq u_i^{[1]}(0) = 0 < 1$ we can use Proposition 2.1 with $t_0 := 0$ and $t_\omega := T_2$ to obtain

   $$\lfloor T_2 o_i(x) \rfloor = \left\lfloor u_i^{[1]}(0) + \sum_{t=1}^{T_2} (i_i^{[1]}(t) + b_i^{[1]}) \right\rfloor = \sum_{t=1}^{T_2} s_i^{[1]}(t).$$

   We further have

   $$i_i^{[1]}(t) + b_i^{[1]} = o_i(x) - s_{a_1}^{[1]}(t-1) = o_i(x) - 1 < 0$$

   for $t > T_2$, so we get

   $$p_i^{[1]}(t) = u_i^{[1]}(T_2) + \sum_{k=T_3}^{t} \left(i_i^{[1]}(k) + b_i^{[1]} \right) - \sum_{k=T_3}^{t-1} s^{[1]}(k) < 1$$

   and therefore $s_i^{[1]}(t) = 0$ for $t > T_2$.

We will now continue with characterizing the second layer. We start with "clock neurons" and the "alarm" neuron:

1. Characterization of the "clock neurons":
   In contrast to the alarm neuron of the first layer, the two "clock" neurons only fire once:

$$p^{[2]}_{c_i}(t) = tb_{c_i} - (T-1)\sum_{k=1}^{t} s^{[2]}_{c_i}(k-1) - \sum_{k=1}^{t-1} s^{[2]}_{c_i}(k) = tb_{c_i} - T\sum_{k=1}^{t-1} s^{[2]}_{c_i}(k)$$

   Let us first consider c_1: We clearly have $p^{[2]}_{c_1}(t) < 1$ for $t < T_2 - 1$, but $p^{[2]}_{c_1}(T_2-1) = 1$. Since by definition $K \geq 1$ and $T_r \geq 2$, so $T_2 = KT_r \geq 2$, we have $t \leq T \leq T(T_2-1)$. Therefore $p^{[2]}_{c_1}(t) < 1$ for $t > T_2 - 1$ due to $s^{[2]}_{c_1}(T_2-1) = 1$. So $\forall_{t\in[T]} s^{[2]}_{c_1}(t) = \chi_{\{T_2-1\}}(t)$.

   We similarly obtain $\forall_{t\in[T]} s^{[2]}_{c_2}(t) = \chi_{T_2}(t)$.

2. Characterization of the "alarm neuron":
   Just like for a_1, we also get $s^{[1]}_{a_2}(t) = 1 \Leftrightarrow t \geq T_4$.

We now proof the behavior of the remaining neurons in the second layer step by step throughout the phases.

1. Phase 1
   We will show $\forall_{i\in[n]} s^{[2]}_{\iota_j(i)}(t) = 0$ and $s^{[2]}_{\omega_j}(t) = 0$ for all $j \in [K^n]$ and $t \in [T_2-1]_0$ by induction over t. Let $t = 0$. We then get $s^{[2]}_{\iota_j(i)}(0) = s^{[2]}_{\omega_j}(0) = 0$ by definition. Let further $1 \leq t \leq T_2 - 1$. First notice that by induction hypothesis, we have $\forall_{i\in[n]} s^{[2]}_{\iota_j(i)}(t') = 0$ and $r_s(q^{(j)}; t') = 0$ for $t' < t$. It follows that

$$i^{[2]}_{\iota_j(i)}(t') + b^{[2]}_{\iota_j(i)} = s^{[1]}_i(t') + T(s^{[2]}_{c_1}(t'-1) - 2s^{[2]}_{c_2}(t'-1) + r_s(q^{(j)}; t'-1)) = s^{[1]}_i(t'),$$
$$i^{[2]}_{\omega_j}(t') + b^{[2]}_{\omega_j} = \frac{1}{n}\sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(t'-1) - 2s^{[2]}_{a_2}(t'-1) + r_s(q^{(j)}; t'-1) = 0.$$

   for all $t' \leq t$. Thus, we further get

$$p^{[2]}_{\iota_j(i)}(t) = -q^{(j)}_i T_r - T + 1 + \sum_{k=1}^{t} s^{[1]}_i(k) - \sum_{k=1}^{t-1} s^{[2]}_{\iota_j(i)}(k) = -q^{(j)}_i T_r - T + 1 + \sum_{k=1}^{t} s^{[1]}_i(k),$$
$$p^{[2]}_{\omega_j}(t) = \sum_{k=1}^{t} 0 - \sum_{k=1}^{t-1} s^{[2]}_{\omega_j}(k) = 0.$$

   Since $\sum_{k=1}^{t} s^{[1]}_i(k) \leq T_2 - 1 < T - 1$ and $s^{[2]}_{c_1}(t) = \chi_{\{T_2-1\}}(t)$, we in particular get $p^{[2]}_{\iota_j(i)}(t) \leq 0$. So we have prooven $\forall_{i\in[n]} s^{[2]}_{\iota_j(i)}(t) = 0$, $s^{[2]}_{\omega_j}(t) = 0$.

2. Phase 2
   Just as in phase 1, we have

$$i^{[2]}_{\omega_j}(T_2) + b^{[2]}_{\omega_j} = \frac{1}{n}\sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(T_2 - 1) - 2s^{[2]}_{a_2}(T_2 - 1) + r_s(q^{(j)}; T_2 - 1) = 0.$$

   So we also get $p^{[2]}_{\omega_j}(T_2) = 0$ and $s^{[2]}_{\omega_j}(T_2) = 0$. It is different for $\iota_j(i)$ due to the dependence on c_1. Let $j \in [K^n]$ and $i \in [n]$ be given. The neuron with index $\iota_j(i)$ fires exactly then at T_2, if $x_i \geq x_i^{C^{(j)}}$: First notice that

$$i^{[2]}_{\iota_j(i)}(T_2) + b^{[2]}_{\iota_j(i)} = s^{[1]}_i(T_2) + T(s^{[2]}_{c_1}(T_2 - 1) - 2s^{[2]}_{c_2}(T_2 - 1) + r_s(q^{(j)}; T_2 - 1)) = s^{[1]}_i(T_2) + T,$$

thus, we conclude that

$$\begin{aligned} p^{[2]}_{\iota_j(i)}(T_2) &= p^{[2]}_{\iota_j(i)}(T_2-1) + i^{[2]}_{\iota_j(i)}(T_2) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(T_2-1) \\ &= -q_i^{(j)} T_r + 1 + \sum_{k=1}^{T_2} s_i^{[1]}(k) \\ &= -q_i^{(j)} T_r + 1 + \lfloor T_2 o_i(x) \rfloor \end{aligned}$$

holds using the characterization of layer 1. Therefore we have $s^{[2]}_{\iota_j(i)}(T_2) = 1$ exactly if $q_i^{(j)} T_r \leq T_2 o_i(x)$, which is equivalent to $\frac{q_i^{(j)}}{K}(y_i^C - x_i^C) + x_i^C \leq x_i$ by definition of o_i. Further $\frac{q_i^{(j)}}{K}(y_i^C - x_i^C) + x_i^C$ is equal to $x^{C^{(j)}}$ by definition of $q^{(j)}$. So $s^{[2]}_{\iota_j(i)}(T_2) = 1$ holds exactly if $x_i^{C^{(j)}} \leq x_i$.

3. Phase 3
   The "Activator neuron" ω_j fires at T_3 if and only if $x^{C^{(j)}} \leq x$: First notice

$$i^{[2]}_{\omega_j}(T_3) + b^{[2]}_{\omega_j} = \frac{1}{n} \sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(T_2) - 2 s^{[2]}_{a_2}(T_2) + r_s(q^{(j)}; T_2) = \frac{1}{n} \sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(T_2).$$

   from which we derive

$$p^{[2]}_{\omega_j}(T_3) = p^{[2]}_{\omega_j}(T_2) + i^{[2]}_{\omega_j}(T_3) + b^{[2]}_{\omega_j} - s^{[2]}_{\omega_j}(T_2) = \frac{1}{n} \sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(T_2).$$

   So $0 \leq p^{[2]}_{\omega_j}(T_3) \leq 1$ and we get $p^{[2]}_{\omega_j}(T_3) = 1$, as well as $s^{[2]}_{\omega_j}(T_3) = 1$ exactly if $\forall_{i \in [n_1]} x_i^{C^{(j)}} \leq x_i$, so if $x^{C^{(j)}} \leq x$.

   Let further $i \in [n_1]$. We get

$$i^{[2]}_{\iota_j(i)}(T_3) + b^{[2]}_{\iota_j(i)} = s_i^{[1]}(T_3) + T(s^{[2]}_{c_1}(T_2) - 2 s^{[2]}_{c_2}(T_2) + r_s(q^{(j)}; T_2)) = -2T$$

   and therefore

$$\begin{aligned} p^{[2]}_{\iota_j(i)}(T_3) &= p^{[2]}_{\iota_j(i)}(T_2) + i^{[2]}_{\iota_j(i)}(T_3) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(T_2) \\ &= -q_i^{(j)} T_r + 1 + \lfloor T_2 o_i(x) \rfloor - 2T - s^{[2]}_{\iota_j(i)}(T_2). \end{aligned}$$

   So $p^{[2]}_{\iota_j(i)}(T_3) \leq -T$ and $s^{[2]}_{\iota_j(i)}(T_3) = 0$, since $\lfloor T_2 o_i(x) \rfloor \leq T_2 \leq T - 1$.

4. Phase 4
   Let $i \in [n]$. The neuron $\iota_j(i)$ stays inactive at T_4, since

$$i^{[2]}_{\iota_j(i)}(T_4) + b^{[2]}_{\iota_j(i)} = s_i^{[1]}(T_4) + T(s^{[2]}_{c_1}(T_3) - 2 s^{[2]}_{c_2}(T_3) + r_s(q^{(j)}; T_3)) = T r_s(q^{(j)}; T_3)$$

   and hence

$$\begin{aligned} p^{[2]}_{\iota_j(i)}(T_4) &= p^{[2]}_{\iota_j(i)}(T_3) + i^{[2]}_{\iota_j(i)}(T_4) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(T_3) \\ &= p^{[2]}_{\iota_j(i)}(T_3) + T r_s(q^{(j)}; T_3). \end{aligned}$$

   Since $p^{[2]}_{\iota_j(i)}(T_3) \leq -T$, we conclude $p^{[2]}_{\iota_j(i)}(T_4) \leq 0$ and $s^{[2]}_{\iota_j(i)}(T_4) = 0$.

   Further the "activator neuron" ω_j fires at T_4 exactly if $x \in C^{(j)}$: First notice

$$i^{[2]}_{\omega_j}(T_4) + b^{[2]}_{\omega_j} = \frac{1}{n} \sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(T_3) - 2 s^{[2]}_{a_2}(T_3) + r_s(q^{(j)}; T_3) = r_s(q^{(j)}; T_3).$$

So we get

$$\begin{aligned} p^{[2]}_{\omega_j}(T_4) &= p^{[2]}_{\omega_j}(T_3) + i^{[2]}_{\iota_j(i)}(T_4) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(T_3) \\ &= p^{[2]}_{\omega_j}(T_3) + r_s(q^{(j)};T_3) - s^{[2]}_{\iota_j(i)}(T_3). \end{aligned}$$

By definition of q, $q_i^{(j)} < q_i^{(j')}$ holds exactly if $x_i^{C^{(j)}} < x_i^{C^{(j')}}$ holds for all $i \in [n_1]$ and $j, j' \in [K^n]$, so $\forall_{j,j' \in [K^n]} q^{(j)} < q^{(j')} \Leftrightarrow x^{C^{(j)}} < x^{C^{(j')}}$. We further have shown $\forall_{j' \in K^n} s^{[2]}_{\omega_{j'}}(T_3) = 1 \Leftrightarrow x^{C^{(j')}} \leq x$ before.

$$\begin{aligned} r_s(q^{(j)};T_3) &= s^{[2]}_{\omega_j}(T_3) - \sum_{\substack{q' \in ([K-1]_0)^{n_1} \\ q^{(j)} < q'}} s^{[2]}_{\omega_{j(q')}}(T_3) \\ &= s^{[2]}_{\omega_j}(T_3) - \sum_{\substack{j' \in [K^n] \\ x^{C^{(j)}} < x^{C^{(j')}} \leq x}} 1 \end{aligned}$$

Now if $x \in C^{(j)}$, i.e. $x^{C^{(j)}} \leq x < y^{C^{(j)}}$, then $s^{[2]}_{\omega_j}(T_3) = 1$ and if there was a $s^{[2]}_{\omega_{j'}}(T_3) = 1$ with $x^{C^{(j)}} < x^{C^{(j')}} \leq x$, we would further get $x^{C^{(j)}} < x^{C^{(j(q'))}} < y^{C^{(j)}}$ due to $x < y^{C^{(j)}}$. But this contradicts the construction of the subcubes $(C^{(j)})_{j \in [K^n]}$.

We have also shown $p^{[2]}_{\omega_j}(T_3) = 1$ and $s^{[2]}_{\omega_j}(T_3) = 1$ for this case, so we can conclude $p^{[2]}_{\omega_j}(T_4) = 1$ as well as $s^{[2]}_{\omega_j}(T_4) = 1$.

Now suppose $x \notin C^{(j)}$. Since we assumed $x \in C$ and constructed the subcubes $(C^{(j)})_{j \in [K^n]}$ as a partition of C, we have $x \in C^{(j')}$, $j \neq j'$. Now if $x^{C^{(j)}} < x^{C^{(j')}}$, then $r_s(q^{(j)};T_3) \leq 0$. We further have $p^{[2]}_{\omega_j}(T_3) = 1$ and $s^{[2]}_{\omega_j}(T_3) = 1$ in this case and therefore $p^{[2]}_{\omega_j}(T_4) \leq 0$ and $s^{[2]}_{\omega_j}(T_4) = 0$.

Is on the other hand $x^{C^{(j)}} \not< x^{C^{(j')}}$, then there is a component $i \in [n_1]$, such that $x_i^{C^{(j')}} < x_i^{C^{(j)}}$ and therefore $x_i < y_i^{C^{(j')}} \leq x_i^{C^{(j)}}$. This implies $x^{C^{(j)}} \not\leq x$ and we also get $r_s(q^{(j)};T_3) \leq 0$. We further have have $p^{[2]}_{\omega_j}(T_3) < 1$ and $s^{[2]}_{\omega_j}(T_3) = 0$ in this case and therefore $p^{[2]}_{\omega_j}(T_4) < 1$ and $s^{[2]}_{\omega_j}(T_4) = 0$.

To summarize, we $s^{[2]}_{\omega_j}(T_4) = 1$ exactly if $x \in C^{(j)}$ just as we claimed, as well as $p^{[2]}_{\omega_j}(T_4) \leq 1$ in general.

5. Phase 5
The "activator neuron" ω_j is inactive during T_5, since for $t > T_4$

$$i^{[2]}_{\omega_j}(t) + b^{[2]}_{\omega_j} = \frac{1}{n} \sum_{i=1}^{n} s^{[2]}_{\iota_j(i)}(t-1) - 2s^{[2]}_{a_2}(t-1) + r_s(q^{(j)};t-1) \leq 0.$$

and

$$\begin{aligned} p^{[2]}_{\omega_j}(t) &= u^{[2]}_{\omega_j}(T_4) + \sum_{k=T_4+1}^{t} \left(i^{[2]}_{\omega_j}(k) + b^{[2]}_{\omega_j} \right) - \sum_{k=T_4+1}^{t-1} s^{[2]}_{\omega_j}(k) \\ &\leq u^{[2]}_{\omega_j}(T_4). \end{aligned}$$

Further $u^{[2]}_{\omega_j}(T_4) < 1$, since $p^{[2]}_{\omega_j}(T_4) \leq 1 < 2$. So $\forall_{t > T_4} s^{[2]}_{\omega_j}(t) = 0$. In particular the "switch" $r_s(q^{(j)};t) = 0$ is off for all $j \in [K^n]$ and $t > T_4$.

Let $i \in [n]$. During T_5 the neuron $\iota_j(i)$ captures the position of x in $C^{(j)}$ regarding the i-th dimension if $x \in C^{(j)}$ and stays inactive otherwise.

Let us first assume $x \notin C^{(j)}$. We have previously shown $s^{[2]}_{\omega_j}(T_4) = 0$ and just now $\forall_{t>T_4} s^{[2]}_{\omega_j}(t) = 0$. So $\forall_{t \geq T_4} r_s(q^{(j)};t) = 0$ and therefore for all $t > T_4$

$$i^{[2]}_{\iota_j(i)}(t) + b^{[2]}_{\iota_j(i)} = s^{[1]}_i(t) + T(s^{[2]}_{c_1}(t-1) - 2s^{[2]}_{c_2}(t-1) + r_s(q^{(j)};t-1)) \leq 0$$

as well as

$$\begin{aligned} p^{[2]}_{\iota_j(i)}(t) &= p^{[2]}_{\iota_j(i)}(T_4) + \sum_{k=T_4+1}^{t} \left(i^{[2]}_{\iota_j(i)}(k) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(k-1) \right) \\ &= p^{[2]}_{\iota_j(i)}(T_4) + \sum_{k=T_4+1}^{t} \left(i^{[2]}_{\iota_j(i)}(k) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(k-1) \right) \\ &\leq p^{[2]}_{\iota_j(i)}(T_4) \\ &\leq 0. \end{aligned}$$

So in particular $\forall_{t>T_4} s^{[2]}_{\iota_j(i)}(t) = 0$.

Suppose now $x \in C^{(j)}$.

Let j be given with $x \in C^{(j)}$. As we have seen before, we have $r_s(q^{(j)};T_4) = 1$ and $\forall_{t>T_4} r_s(q^{(j)};t) = 0$. We therefore get

$$i^{[2]}_{\iota_j(i)}(T_4+1) + b^{[2]}_{\iota_j(i)} = s^{[1]}_i(T_4+1) + T(s^{[2]}_{c_1}(T_4) - 2s^{[2]}_{c_2}(T_4) + r_s(q^{(j)};T_4)) = T$$

and for all $t > T_4 + 1$

$$i^{[2]}_{\iota_j(i)}(t) + b^{[2]}_{\iota_j(i)} = s^{[1]}_i(t) + T(s^{[2]}_{c_1}(t-1) - 2s^{[2]}_{c_2}(t-1) + r_s(q^{(j)};t-1)) = 0.$$

Using previous results and in particular $s^{[2]}_{\iota_j(i)}(T_2) = 1 \Leftrightarrow x^{C^{(j)}}_i \leq x_i$, we obtain

$$\begin{aligned} p^{[2]}_{\iota_j(i)}(T_4+1) &= p^{[2]}_{i_j(i)}(T_4) + i^{[2]}_{\iota_j(i)}(T_4+1) + b^{[2]}_{\iota_j(i)} - s^{[2]}_{\iota_j(i)}(T_4) \\ &= p^{[2]}_{\iota_j(i)}(T_3) + T r_s(q^{(j)};T_3) + T \\ &= -q^{(j)}_i T_r + 1 + \lfloor T_2 o_i(x) \rfloor - 2T - s^{[2]}_{\iota_j(i)}(T_2) + 2T \\ &= -KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor. \end{aligned}$$

We now have $KT_r o_i(x^{C^{(j)}}) \leq KT_r o_i(x)$, since $x^{C^{(j)}}_i \leq x_i$, so $p^{[2]}_{\iota_j(i)}(T_4+1) \geq 0$.

We further have

$$o_i(y^{C^{(j)}}) - o_i(x^{C^{(j)}}) = \frac{y^{C^{(j)}}_i - x^{C^{(j)}}_i}{y^C_i - x^C_i} = \frac{1}{K}.$$

and $o_i(x) < o_i(y^{C^{(j)}}) = o_i(x^{C^{(j)}}) + \frac{1}{K}$ for all $x \in C^{(j)}$. We can deduce

$$\begin{aligned} p^{[2]}_{\iota_j(i)}(T_4+1) &= -KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor \\ &\leq -KT_r o_i(x^{C^{(j)}}) + KT_r o_i(y^{C^{(j)}}) \\ &\leq -KT_r o_i(x^{C^{(j)}}) + T_r + KT_r o_i(x^{C^{(j)}}) \\ &\leq T_r. \end{aligned}$$

So we have shown

$$0 \leq p^{[2]}_{\iota_j(i)}(T_4+1) = u^{[2]}_{i_j(i)}(T_4) + i^{[2]}_{\iota_j(i)}(T_4+1) + b^{[2]}_{\iota_j(i)} \leq T_r = T - (T_4+1).$$

By Lemma 2.8 we now get $u^{[2]}_j(T_4+1) \geq 0$. We can therefore use Proposition 2.2 with $t_0 := T_4+1$, $t_m := T_4+1$ and $t_\omega := T$, such that we obtain

$$\begin{aligned}\sum_{t=T_4+1}^{T} s^{[2]}_{\iota_{j(i)}}(t) &= \left\lfloor u^{[2]}_{\iota_{j(i)}}(T_4) + i^{[2]}_{\iota_{j(i)}}(T_4+1) + b^{[2]}_{\iota_{j(i)}} \right\rfloor \\ &= -KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor .\end{aligned}$$

We have now reached the final step, where we will show that the spikes actually approximate g. Let us consolidate our results. For j with $x \in C^{(j)}$ we have

$$\begin{aligned}\sum_{t=T_4}^{T} s^{[2]}_{\iota_{j(i)}}(t) &= \sum_{t=T_4+1}^{T} s^{[2]}_{\iota_{j(i)}}(t) = -KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor , \\ \sum_{t=T_4}^{T} s^{[2]}_{\omega_j}(t) &= s^{[2]}_{\omega_j}(T_4) = 1.\end{aligned}$$

And therefore for all $k \in [m]$

$$\begin{aligned}W^{[L+1]}_{k,\omega_j} \sum_{t=T_4}^{T} s^{[2]}_{\omega_j}(t) &= g_k(x^{C^{(j)}}), \\ W^{[L+1]}_{k,\iota_{j(i)}} \sum_{t=T_4}^{T} s^{[2]}_{\iota_{j(i)}}(t) &= \left(-KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor\right) \left(d(g^{(j)})_k((y_i^{C^{(j)}} - x_i^{C^{(j)}}) \frac{1}{T_r} e_i) \right).\end{aligned}$$

Let now j be such that $x \notin C^{(j)}$, we have shown

$$\begin{aligned}\sum_{t=T_4}^{T} s^{[2]}_{\iota_{j(i)}}(t) &= 0, \\ \sum_{t=T_4}^{T} s^{[2]}_{\omega_j}(t) &= 0.\end{aligned}$$

So this group of neurons does not contribute to the output of the network:

$$\begin{aligned}W^{[L+1]}_{k,\omega_j} \sum_{t=T_4}^{T} s^{[2]}_{\omega_j}(t) &= 0, \\ W^{[L+1]}_{k,\iota_{j(i)}} \sum_{t=T_4}^{T} s^{[2]}_{\iota_{j(i)}}(t) &= 0.\end{aligned}$$

Finally note that by choice of $W^{[L+1]}$, the neurons c_1, c_2, a_2 don't contribute to the output as well. For any $j \in \{c_1, c_2, a_2\}$

$$W^{[L+1]}_{k,j} \sum_{t=T_4}^{T} s^{[2]}_{j}(t) = 0.$$

Let now $j \in [K^n]$ be such that $x \in C_j$ again. We can consequently have

$$R(\Phi)_k(x) = \sum_{t=T_4}^{T} (W^{[L+1]} s^{[2]}(t))_k = W^{[L+1]}_{k,\omega_j} \sum_{t=T_4}^{T} s^{[2]}_{\omega_j}(t) + W^{[L+1]}_{k,\iota_{j(i)}} \sum_{t=T_4}^{T} s^{[2]}_{\iota_{j(i)}}(t).$$

Which is further equal to

$$g_k(x^{C^{(j)}}) + \sum_{i\in[n_1]} \left(d(g^{(j)})_k((y_i^{C^{(j)}} - x_i^{C^{(j)}})\frac{1}{T_r}e_i) \right) \left(-KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor \right)$$
$$= g_k(\underbrace{x^{C^{(j)}} + \sum_{i\in[n_1]} \frac{y_i^{C^{(j)}} - x_i^{C^{(j)}}}{T_r} \left(-KT_r o_i(x^{C^{(j)}}) + \lfloor KT_r o_i(x) \rfloor \right) e_i}_{=:x'})$$

Now by assumption, we have

$$T_r = \sqrt{n_1}\frac{y_i^C - x_i^C}{K}\frac{\|df\|_{\infty,2}}{2\nu} \geq \sqrt{n_1}(y_i^{C^{(j)}} - x_i^{C^{(j)}})\frac{\left\|d(g^{(j)})\right\|_{\infty,2}}{2\nu}$$

for any $i \in [n_1]$ and hence

$$\xi_i := \frac{1}{T_r}\left(\lfloor KT_r o_i(x) \rfloor - KT_r o_i(x)\right) \leq \frac{1}{\sqrt{n_1}(y_i^{C^{(j)}} - x_i^{C^{(j)}})}\frac{2\nu}{\left\|d(g^{(j)})\right\|_{\infty,2}}.$$

We now get the following inequality by definition of ξ_i and $(y_i^{C^{(j)}} - x_i^{C^{(j)}})K = (y_i^C - x_i^C)$

$$\begin{aligned}
\|x' - x\|_2^2 &= \sum_{i\in[n_1]} (x' - x_i)^2 \\
&= \sum_{i\in[n_1]} \left((y_i^{C^{(j)}} - x_i^{C^{(j)}}) \left(\tfrac{1}{T_r} \lfloor KT_r o_i(x) \rfloor - K o_i(x^{C^{(j)}}) \right) - (x_i - x_i^{C^{(j)}}) \right)^2 \\
&= \sum_{i\in[n_1]} \left((y_i^{C^{(j)}} - x_i^{C^{(j)}}) \left(\xi_i + K\frac{x_i - x_i^{C^{(j)}}}{y_i^C - x_i^C} \right) - (x_i - x_i^{C^{(j)}}) \right)^2 \\
&= \sum_{i\in[n_1]} \left(\left(\xi_i (y_i^{C^{(j)}} - x_i^{C^{(j)}}) + (x_i - x_i^{C^{(j)}}) \right) - (x_i - x_i^{C^{(j)}}) \right)^2 \\
&= \sum_{i\in[n_1]} \xi_i^2 (y_i^{C^{(j)}} - x_i^{C^{(j)}})^2 \\
&\leq \frac{\nu^2}{\left\|d(g^{(j)})\right\|_{\infty,2}^2}.
\end{aligned}$$

So we can conclude

$$\|g(x) - R(\Phi)_k(x)\|_2 = \|g(x) - g(x')\|_2 = \left\|d(g^{(j)})(x - x')\right\|_2 \leq \left\|d(g^{(j)})\right\|_{\infty,2}\|x - x'\|_2 \leq \nu$$

□

Sadly the size of the network in this construction is not always smaller than the one from Theorem 3.1. A concrete counter example is a sinus wave with high frequency and small ampitude, like $f(x) := \frac{\sin(nx)}{n}$ with $n \in \mathbb{N}$ on $C = [0, 2\pi)$, $\Omega = [0, 2\pi]$. Since $f'(x) = \cos(nx)$ and $\left\|f'|_{(0,2\pi)}\right\|_\infty = 1$, $\Gamma := 1$ is the optimal Lipschitz-constant for f. At the same time, since $f''(x) = n\sin(nx)$ and $\left\|f''|_{(0,2\pi)}\right\|_\infty = n$, the biggest possible modulus of uniform continuity on C° we can give for f' is $\delta(\varepsilon) := \frac{\varepsilon}{n}$. So we get

$$\max_{\substack{\xi,\theta>0 \\ \xi\theta=\varepsilon}} \min(\delta(\xi), \theta) = \max_{\xi>0} \min(\frac{\xi}{n}, \frac{\varepsilon}{\xi}) = \sqrt{\varepsilon n}$$

So we get $K(\varepsilon) := \lfloor \frac{\pi}{\sqrt{n\varepsilon}} \rfloor$. We therefore get for the layer sizes:

Theorem 3.1	*Theorem* 3.2
$n_1 = \lceil \frac{2\pi}{\varepsilon} \rceil + 1$	$n_1 = 2$
$n_2 = \lceil \frac{2\pi}{\varepsilon} \rceil$	$n_2 = 2\lfloor \frac{\pi}{\sqrt{n\varepsilon}} \rfloor + 3$

While the first and second layer of Theorem 3.2 are clearly arbitrarily smaller than the first layer of the other construction for small ε, the second layer of Theorem 3.2 is arbitrarily bad for $n \to \infty$ compared to the second layer of Theorem 3.1.

On the other hand, even for large $n \in \mathbb{N}$, the second layer of Theorem 3.2 is arbitrarily more efficient for $\varepsilon \to 0$, since the size of the second layer of Theorem 3.2 only grows proportionally to $\frac{1}{\sqrt{\varepsilon}}$ and not $\frac{1}{\varepsilon}$.

We generalize this observation with the following theorem.

Theorem 3.3. *Let $C \neq \emptyset$ be a cube in $\mathbb{R}^n$, $\Omega \subset \overline{C}$ a compact subset with $\operatorname{diam}_\infty(\Omega) \neq 0$ and $f \in \mathcal{C}^0(\overline{C}, \mathbb{R})$ a continuous function that is Γ'-Lipschitz, such that $f|_{C^\circ} \in \mathcal{C}^1(C^\circ, \mathbb{R}^m)$ is continuously differentiable and $d(f|_{C^\circ})$ is Γ-Lipschitz.*

We get

$$\lim_{\varepsilon \to 0} \frac{n_2(\varepsilon)}{n_2'(\varepsilon)} = 0.$$

Where $n_2'(\varepsilon)$ is $n_2(\varepsilon)$ as defined in Theorem 3.1 for Ω, $f|_\Omega$ with Lipschitz-constant Γ' and ε; and where $n_2(\varepsilon)$ is as defined in Theorem 3.2 for C, $f|_C$ with modulus of continuity $\omega(x) := \Gamma x$ of $d(f|_{C^\circ})$ and $\mu := \frac{\varepsilon}{2}$.

Proof. First notice, that since

$$n_2'(\varepsilon) = \max\left\{ \left\lceil \frac{\operatorname{diam}_\infty(\Omega)}{\varepsilon} \Gamma' \right\rceil^{n_0}, 1 \right\}$$

we get

$$\lim_{\varepsilon \to 0} n_2'(\varepsilon)\varepsilon = \operatorname{diam}_\infty(\Omega)\Gamma'.$$

We further have

$$n_2(\varepsilon) = \min_{\substack{\xi,\theta>0 \\ \xi\theta = \frac{\varepsilon}{2}}} \left\{ \left\lceil \frac{\operatorname{diam}_\infty(C)}{\frac{2}{\sqrt{n}} \min(\omega^\dagger(\xi), \theta)} \right\rceil \right\} (n+1) + 3.$$

Let now $(\xi_\varepsilon)_{\varepsilon>0}$ and $(\theta_\varepsilon)_{\varepsilon>0}$ be given such that $\forall_{\varepsilon>0} \xi_\varepsilon \theta_\varepsilon = \frac{\varepsilon}{2}$ and the minimum in $n_2(\varepsilon)$ is obtained, i.e.

$$\min(\omega^\dagger(\xi_\varepsilon), \theta_\varepsilon) = \max_{\substack{\xi,\theta>0 \\ \xi\theta = \frac{\varepsilon}{2}}} \min(\omega^\dagger(\xi), \theta).$$

Note moreover that $\omega^\dagger(s) = \inf\{t \in [0,\infty] \mid \Gamma t > s\} = \frac{s}{\Gamma}$ for $\Gamma \neq 0$ and $\omega^\dagger(s) = \infty$ otherwise, since $\omega(x) = \Gamma x$. We now get

$$\min(\omega^\dagger(\xi_\varepsilon), \theta_\varepsilon) = \min\left(\frac{\xi_\varepsilon}{\Gamma}, \frac{\varepsilon}{\xi_\varepsilon 2} \right) = \sqrt{\frac{\Gamma\varepsilon}{2}}$$

and therefore

$$n_2(\varepsilon) = \left\lceil \sqrt{\frac{n}{2\Gamma\varepsilon}} \operatorname{diam}_\infty(C) \right\rceil (n+1) + 3$$

for $\Gamma \neq 0$. Is $\Gamma = 0$, then $\min(\omega^\dagger(\xi_\varepsilon), \theta_\varepsilon) = \infty$ (TODO) With this simplified definition, we get

$$\lim_{\varepsilon \to 0} n_2(\varepsilon)\sqrt{\varepsilon} = \sqrt{\frac{n}{2\Gamma}}\,\mathrm{diam}_\infty(C)(n+1).$$

We can conclude

$$\lim_{\varepsilon \to 0} \frac{n_2(\varepsilon)}{n_2'(\varepsilon)} \frac{1}{\sqrt{\varepsilon}} = \lim_{\varepsilon \to 0} \frac{n_2(\varepsilon)\varepsilon}{n_2'(\varepsilon)\sqrt{\varepsilon}} = \frac{\sqrt{\frac{n}{2\Gamma}}\,\mathrm{diam}_\infty(C)(n+1)}{\mathrm{diam}_\infty(\Omega)\Gamma'} \in \mathbb{R}$$

So the proof is finished, since we can deduce

$$\lim_{\varepsilon \to 0} \frac{n_2(\varepsilon)}{n_2'(\varepsilon)} = 0.$$

□

4 Complexity of input partitions

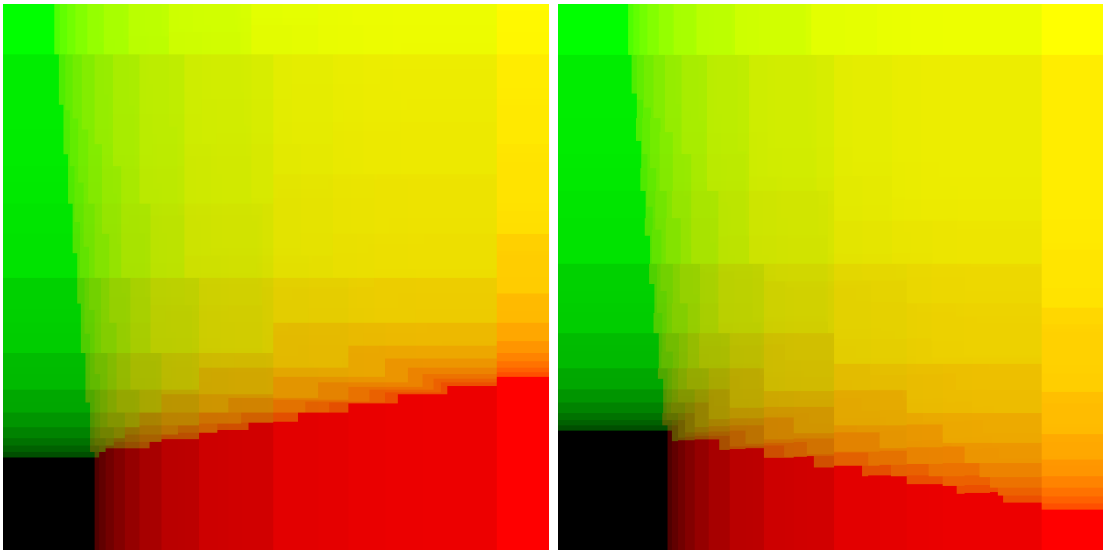


Figure 4.1: The landscape of the first layer of two different r. LIF-SNN

In the following we will analyze what shape the graph of a r. LIF-SNN has. Of particular interest to us is the question how many different values a r. LIF-SNN can produce. Since the number of different values of the following layers only depends on the spike trains of the first hidden layer, we can get an upper bound on that number by analyzing how many different spike trains the first hidden layer can produce. We will therefore only study the output landscape of the first hidden layer. We will further generally assume $W^{[1]} = I_{n_1}$. Let Φ be a r. LIF-SNN with $W^{[1]}$ arbitrary and Φ' the same one, but with $W^{[1]} = I_{n_1}$. We then have $R(\Phi)(x) = R(\Phi')(W^{[1]}x)$ since we are using direct encoding. So $W^{[1]}$ just corresponds to a pre-transformation on the input vector and can therefore (1) only decrease the number of different output values and further the key to understanding the output landscape of a general r. LIF-SNN is to understand the landscape of one with $W^{[1]} = I_{n_1}$.

We can further simplify the notation in this section by writing $W, b, V, \alpha, \beta, \vartheta, i, u, s, n$ for $W^{[1]}, b^{[1]}, V^{[1]}, \alpha^{[1]}, \beta^{[1]}, \vartheta^{[1]}, i^{[1]}, u^{[1]}, s^{[1]}, n_1$ respectively, since we will only be working on the first layer anyways. Further write $x := s^{[0]}$. Since we are using direct encoding, we may assume $\forall_{t\in[T]} s^{[0]}(t) = x$. We now get the following simplified defining equations:

$$i(t) = \alpha i(t-1) + Wx + Vs(t-1) \tag{14}$$
$$p(t) = \beta u(t-1) + i(t) + b \tag{15}$$
$$s(t) = H(p(t) - \vartheta \mathbf{1}_n) \tag{16}$$
$$u(t) = p(t) - \vartheta s(t) \tag{17}$$

Since the definitions recursively depend on x, we will sometimes also write $i(t;x)$, $p(t;x)$, $s(t;x)$ and $u(t;x)$ to make the dependency explicit.

By using the simplified notation, we obtain the following definitions from Definition 2.6 for

the input vector $x \in \mathbb{R}^{n_0}$ and the first hidden layer spike-train σ:

$$i(t;x;\sigma) = \alpha^t i(0) + \sum_{k=1}^{t} \alpha^{t-k} \left(Wx + V\sigma(k-1)\right), \tag{18}$$

$$p(t;x;\sigma) = \beta^t u(0) + \sum_{k=1}^{t} \beta^{t-k} \left(i(k;x;\sigma) + b\right) - \vartheta \sum_{k=1}^{t-1} \beta^{t-k} \sigma(k), \tag{19}$$

$$s(t;x;\sigma) = H(p(t;x;\sigma) - \vartheta \mathbf{1}_{n_1}) \tag{20}$$

$$u(t;x;\sigma) = \beta^t u(0) + \sum_{k=1}^{t} \beta^{t-k} \left(i(k;x;\sigma) + b - \vartheta\sigma(k)\right), \tag{21}$$

Let us first introduce some preliminary definitions.

Definition 4.1. The set of constant regions of a r. LIF-SNN Φ is defined as the partition

$$C_\Phi := \{R(\Phi)^{-1}(\{y\}) \mid y \in \mathrm{im}(R(\Phi))\}$$

of $\mathbb{R}^{n_0}$. A constant region with spike train $s' \in \{0,1\}^{n_1 \times T}$, of the first layer of a r. LIF-SNN Φ is defined as

$$C_{s'} := \{x \in \mathbb{R}^{n_0} \mid \forall_{t\in[T]} s(t;x) = s'(t)\}$$

We further notate the set of such non-empty regions by $C_{\Phi,1} := \{C_{s'} \mid s' \in \{0,1\}^{n_1 \times T}, C_{s'} \neq \emptyset\}$.

While quite technical, g is the key to proper rigorous proofs about the landscape of r. LIF-SNNs.

Lemma 4.1. *There is a function* $g : \dot{\bigcup}_{t\in[T]} \{\sigma \mid \sigma \in \{0,1\}^{n_1 \times (t-1)}\} \to \mathbb{R}$ *such that* $\forall_{i\in[n_1]} s_i(t;x;\sigma) = 1 \Leftrightarrow \langle w_i, x\rangle \geq g_i(t;\sigma)$ *defined by*

$$g(t;\sigma) := -\frac{\sum_{k=1}^{t} \beta^{t-k} \left(\alpha^k i(0) + b + \sum_{l=1}^{k} \alpha^{k-l} V\sigma(l-1)\right) + \beta^t u(0) - \vartheta\left(1 + \sum_{k=1}^{t-1} \beta^{t-k}\sigma(k)\right)}{\sum_{k=1}^{t} \beta^{t-k} \sum_{l=1}^{k} \alpha^{k-l}}$$

Where w_i *denotes the* i*-th row vector of* W*.*

Proof of Lemma 4.1. We compute for $i \in [n_1]$ using Lemma 2.1:

$$\begin{aligned}
&H(p_i(t;x;\sigma) - \vartheta) \\
&= H\left(\sum_{k=1}^{t} \beta^{t-k} \left(i_i(k;x;\sigma) + b_i\right) + \underbrace{\beta^t u_i(0) - \vartheta\left(1 + \sum_{k=1}^{t-1} \beta^{t-k}\sigma_i(k)\right)}_{(*)}\right) \\
&= H\left(\sum_{k=1}^{t} \beta^{t-k} \left(\alpha^k i_i(0) + \sum_{l=1}^{k} \alpha^{k-l} \left(\langle w_i, x\rangle + (V\sigma(l-1))_i\right) + b_i\right) + (*)\right) \\
&= H\left(\sum_{k=1}^{t} \beta^{t-k} \sum_{l=1}^{k} \alpha^{k-l} \langle w_i, x\rangle + \sum_{k=1}^{t} \beta^{t-k} \left(\alpha^k i_i(0) + b_i + \sum_{l=1}^{k} \alpha^{k-l} (V\sigma(l-1))_i\right) + (*)\right) \\
&= H\left(\langle w_i, x\rangle + \frac{\sum_{k=1}^{t} \beta^{t-k} \left(\alpha^k i_i(0) + b_i + \sum_{l=1}^{k} \alpha^{k-l} (V\sigma(l-1))_i\right) + (*)}{\sum_{k=1}^{t} \beta^{t-k} \sum_{l=1}^{k} \alpha^{k-l}}\right) \\
&= H(\langle w_i, x\rangle - g_i(t;\sigma))
\end{aligned}$$

□

Remark 4.1. Like with $i(t;x;\sigma)$, $p(t;x;\sigma)$, etc. we will allow supplying g with a extension of the spiketrain that is required. In that case the value of g should be understood as the value at the prefix of that spiketrain.

Proposition 4.1. *The constant regions of the first layer of a r. LIF-SNN Φ (with $W = I_{n_1}$) are half-open cuboids. In particular, let $s' \in \{0,1\}^{n_1 \times T}$ be a spike train and $C_{s'} = [\![x^{s'}, y^{s'}]\!]$ be the corresponding constant region. Then*

$$x_i^{s'} = \sup_{\substack{t \in [T] \\ s_i'(t)=1}} g_i(t;s') \qquad y_i^{s'} = \inf_{\substack{t \in [T] \\ s_i'(t)=0}} g_i(t;s')$$

using the conventions $\inf \emptyset = \infty$ and $\sup \emptyset = -\infty$ here.

Remark 4.2. Note that by Proposition 4.1, we have $x_i^{s'} = -\infty$ exactly if $\forall_{t \in [T]} s_i'(t) = 0$; and $y_i^{s'} = \infty$ exactly if $\forall_{t \in [T]} s_i'(t) = 1$. So in particular we cannot have s' with an component $i \in [n_1]$, such that both $x_i^{s'} = -\infty$ and $y_i^{s'} = \infty$.

Lemma 4.2. *For every $s' \in \{0,1\}^{n_1 \times T}$ we have*

$$C_{s'} = \left(\bigcap_{\substack{i \in [n_1], t \in [T] \\ s_i'(t)=0}} \pi_i^{-1}([-\infty, g_i(t;s'))) \right) \bigcap \left(\bigcap_{\substack{i \in [n_1], t \in [T] \\ s_i'(t)=1}} \pi_i^{-1}([g_i(t;s'), \infty)) \right).$$

Proof of Lemma 4.2. Let $s' \in \{0,1\}^{n_1 \times T}$ be given. We then get

$$\begin{aligned}
C_{s'} &= \bigcap_{i \in [n_1], t \in [T]} \{x \mid s_i'(t) = s_i(t;x)\} \\
&= \bigcap_{i \in [n_1], t \in [T]} \{x \mid s_i'(t) = s_i(t;x;s')\} \\
&= \left(\bigcap_{\substack{i \in [n_1], t \in [T] \\ s_i'(t)=0}} \{x \mid s_i(t;x;s') = 0\} \right) \bigcap \left(\bigcap_{\substack{i \in [n_1], t \in [T] \\ s_i'(t)=1}} \{x \mid s_i(t;x;s') = 1\} \right) \\
&= \left(\bigcap_{\substack{i \in [n_1], t \in [T] \\ s_i'(t)=0}} \pi_i^{-1}([-\infty, g_i(t;s'))) \right) \bigcap \left(\bigcap_{\substack{i \in [n_1], t \in [T] \\ s_i'(t)=1}} \pi_i^{-1}([g_i(t;s'), \infty)) \right)
\end{aligned}$$

by Lemma 2.1 for the second equality and Lemma 4.1 for the last one. □

Lemma 4.3. $\bigcap_{j \in J} \prod_{i \in [n]} M_{i,j} = \prod_{i \in [n]} \bigcap_{j \in J} M_{i,j}$ *for an index set J and sets $(M_{i,j})_{i \in [n], j \in J}$.*

Proof of Lemma 4.3. For every $x = (x_i)_{i \in [n]} \in M := \prod_{i \in [n]} \bigcup_{j \in J} M_{i,j}$ holds:

$$x \in \bigcap_{j \in J} \prod_{i \in [n]} M_{i,j} \Leftrightarrow \left(\forall_{j \in J} x \in \prod_{i \in [n]} M_{i,j} \right) \Leftrightarrow \forall_{j \in J} \forall_{i \in [n]} x_i \in M_{i,j}.$$

On the other hand, we have

$$x \in \prod_{i \in [n]} \bigcap_{j \in I} M_{i,j} \Leftrightarrow \left(\forall_{i \in [n]} x_i \bigcap_{j \in I} M_{i,j} \right) \Leftrightarrow \forall_{i \in [n]} \forall_{j \in J} x_i \in M_{i,j}.$$

In total we get

$$\bigcap_{j \in J} \prod_{i \in [n]} M_{i,j} = M \cap \bigcap_{j \in J} \prod_{i \in [n]} M_{i,j} = M \cap \prod_{i \in [n]} \bigcap_{j \in I} M_{i,j} = \prod_{i \in [n]} \bigcap_{j \in I} M_{i,j}.$$

□

Lemma 4.4. *The intersection $C_1 \cap C_2$ of half-open cuboids $C_1, C_2 \subset \mathbb{R}^n$ is a half-open cuboid. In particular, if $C_i := \prod_{j\in[n]}[c_{i,j}, d_{i,j})$, then*

$$C_1 \cap C_2 = \prod_{j\in[n]} ([\sup(c_{1,j}, c_{2,j}), \inf(d_{1,j}, d_{2,j})))$$

We use inf/sup *instead of* min/max *to highlight that the components of the vertices might be* $\pm\infty$.

Proof of Lemma 4.4. Let us first regard $n = 1$. For $c_1, c_2, d_1, d_2 \in \mathbb{R} \cup \{\pm\infty\}$ we get

$$\begin{aligned} & x \in [c_1, d_1) \cap [c_2, d_2) \\ \Leftrightarrow\ & c_1, c_2 \leq x < d_1, d_2 \\ \Leftrightarrow\ & x \in [\sup(c_{1,j}, c_{2,j}), \inf(d_{1,j}, d_{2,j})) \end{aligned}$$

for $x \in \mathbb{R}$ and therefore

$$C_1 \cap C_2 = \prod_{j\in[n]} \left([c_{1,j}, d_{1,j}) \cap [c_{2,j}, d_{2,j})\right) = \prod_{j\in[n]} [\sup(c_{1,j}, c_{2,j}), \inf(d_{1,j}, d_{2,j}))$$

by Lemma 4.3. □

Proof of Proposition 4.1. Let $C_{s'} \in C_{\Phi,1}$ be a constant region. We then get

$$\begin{aligned} C_{s'} &= \left(\bigcap_{\substack{i\in[n_1], t\in[T] \\ s'_i(t)=0}} \pi_i^{-1}([-\infty, g_i(t; s'))) \right) \bigcap \left(\bigcap_{\substack{i\in[n_1], t\in[T] \\ s'_i(t)=1}} \pi_i^{-1}([g_i(t; s'), \infty)) \right) \\ &= \prod_{i\in[n]} [\sup_{t\in[T],\ s'_i(t)=1} g_i(t; s'), \inf_{t\in[T],\ s'_i(t)=0} g_i(t; s')) \end{aligned}$$

by Lemma 4.2 and Lemma 4.4, since

$$\pi_i^{-1}([c, d)) = [-\infty, \infty) \times \ldots \times [c, d) \times \ldots \times [-\infty, \infty).$$

□

We will now add an ordering to spike trains based on lexicographical ordering.

Definition 4.2. Let $s', s'' \in \{0,1\}^{n_1 \times T}$, we then have $s' \leq_l s''$ exactly if either $s' = s''$ or there exists a $t \in [T]$ such that $s'(t) < s''(t)$ and $\forall_{t'<t} s'(t') = s''(t')$. We further define $s' <_l s''$ as $s' \leq_l s''$ and $s' \neq s''$.

Lemma 4.5. *Let $x, y \in \mathbb{R}^{n_0}$. We then have $x \leq y \Rightarrow s(\cdot; x) \leq_l s(\cdot; y)$.*

Is on the other hand $s(\cdot; x) <_l s(\cdot; y)$ and $i \in [n_1]$ such that $s_i(t; x) \neq s_i(t; y)$ holds at the minimal time t at which $s(\cdot; x)$ and $s(\cdot; y)$ are different, then $x_i < y_i$ holds.

Proof. First notice that for a fixed σ the function $i_i(t; x; \sigma)$ is growing montonically in x_i for all $i \in [n]$, since $\alpha \geq 0$. We similarly get that $p_i(t; x; \sigma)$ and $s_i(t; x; \sigma)$ are growing monotonically in x_i, since $H = \chi_{[0,\infty)}$ is monotonically growing.

Let now $x \leq y$ and suppose $s(\cdot; x) \neq s(\cdot; y)$. We then have a minimal $t \in [T]$ such $s(t; x) \neq s(t; y)$. Now due to $\forall_{t'<t} s(t'; x) = s(t'; y)$ and $x \leq y$ we have $\forall_{i\in[n_1]} s_i(t; x) \leq s_i(t; y)$ using the previous remark with $\sigma(t) := s(t; x)$. We therefore get $s(t; x) <_l s(t; y)$.

Let on the other hand $s(\cdot; x) <_l s(\cdot; y)$. Then there is a smallest time t such that $\exists_{i\in[n_1]} s_i(t; x) \neq s_i(t; y)$. By definition of the ordering on spike trains we get $s_i(t; x) <_l s_i(t; y)$. Now $s_i(t; x)$ is growing monotonically in x_i with $\sigma(t) := s(t; x)$ by choice of t. We therefore have $x_i < y_i$. □

We will now proof some theorems about specific kinds of spike trains. We first have to define them.

Definition 4.3. We call a spike train $s' \in \{0,1\}^{n_1 \times T}$ constant in component $i \in [n_1]$, if $\forall_{t,t' \in [T]} s'_i(t) = s'_i(t')$. A spike train s' is therefore constant in every component, if $\forall_{i \in [n_1]} \forall_{t,t' \in [T]} s'_i(t) = s'_i(t')$. We further call s' non-constant in every component, if $\forall_{i \in [n_1]} \exists_{t,t' \in [T]} s'_i(t) \neq s'_i(t')$.

We also need some geometric definitions.

Definition 4.4. We call a point $p \in (\mathbb{R} \cup \{\pm\infty\})^n$ vertex of a non-empty region $[\![x,y)\!)$, $[\![x,y]\!]$ or $(\!(x,y)\!)$, if $\forall_{i \in [n]} p_i \in \{x_i, y_i\}$. We call a vertex p finite, if $p \in \mathbb{R}^n$.

Lemma 4.6. *Let $s' \in \{0,1\}^{n_1 \times T}$ be a spike train that is constant in every component. Then $C_{s'}$ is non-empty.*

Proof. Suppose s' is constant in every component. Let further $i \in [n_1]$. We now either have $\forall_{t \in [T]} s'_i(t) = 0$ or $\forall_{t \in [T]} s'_i(t) = 1$. In the first case, we get $x_i^{s'} = -\infty$ and $y_i^{s'} \in \mathbb{R}$; in the second case $x_i^{s'} \in \mathbb{R}$ and $y_i^{s'} = \infty$ by construction. So $C_{s'}$ is clearly non-empty. □

Lemma 4.7. *Let $s' \in \{0,1\}^{n_1 \times T}$ be a spike train such that $C_{s'} \neq \emptyset$ and s' is constant in k components. Then $C_{s'}$ has $2^{n_1 - k}$ finite vertices.*

Remark 4.3. So in particular, every region $C_{s'} \in C_{\Phi,1}$ has at least one finite vertex.

Proof. First note that if s' is constant in component $i \in [n_1]$, then either $x_i^{s'} = -\infty$ and $y_i^{s'} \in \mathbb{R}$; or $x_i^{s'} \in \mathbb{R}$ and $y_i^{s'} = \infty$. In either case $\left|\mathbb{R} \cap \{x_i^{s'}, y_i^{s'}\}\right| = 1$. Is on the other hand s' non-constant in i, then $x_i^{s'}, y_i^{s'} \in \mathbb{R}$. We therefore get

$$\left|\mathbb{R}^{n_1} \cap \prod_{j \in [n_1]} \{x_j^{s'}, y_j^{s'}\}\right| = \left|\prod_{j \in [n_1]} \mathbb{R} \cap \{x_j^{s'}, y_j^{s'}\}\right| = 2^{n_1 - k}$$

for the set of finite vertices, $\mathbb{R}^{n_1} \cap \prod_{j \in [n_1]} \{x_j^{s'}, y_j^{s'}$. □

Lemma 4.8. *Let $s' \in \{0,1\}^{n_1 \times T}$ be a spike train such that $C_{s'} \neq \emptyset$. Then the following conditions,*

(a) s' is constant in at least one component,
(b) $C_{s'}$ has $\leq 2^{n_1 - 1}$ finite vertices,
(c) $C_{s'}$ has a non-finite vertix,
(d) $C_{s'}$ is unbounded,

are equivalent.

Proof. Let first $x^{s'}, y^{s'}$ be defined as in Proposition 4.1, such that we have $C_{s'} = [\![x^{s'}, y^{s'})\!)$.

- $(a) \Leftrightarrow (b)$: Follows directly by Lemma 4.7.
- $(b) \Rightarrow (c)$: A non-empty cube $[\![x^{s'}, y^{s'})\!)$ has

  $$\left|\prod_{j \in [n_1]} \{x_j^{s'}, y_j^{s'}\}\right| = 2^{n_1}$$

  vertices, so assuming $C_{s'}$ has $\leq 2^{n_1 - 1}$ finite vertices, it must have a non-finite one.
- $(c) \Rightarrow (b)$: If $C_{s'}$ has a non-finite vertice, then the number of finite vertices is smaller than 2^{n_1}. On the other hand the number of finite vertices must be a power of 2 by Lemma 4.7, so the number must be smaller or equal $2^{n_1 - 1}$.

- $(c) \Leftrightarrow (d)$: If all vertices are finite, then in particular $x^{s'}, y^{s'} \in \mathbb{R}^{n_1}$ and $[\![x^{s'}, y^{s'}]\!]$ is clearly bounded. Is one vertice non-finite, then either $x^{s'}$ or $y^{s'}$ has a component that is $\pm\infty$. Suppose we have $x_i^{s'} = -\infty$. Since $C_{s'} \neq \emptyset$ we further have $z \in C_{s'}$. By definition of $[\![x^{s'}, y^{s'}]\!]$, we have now $z' \in [\![x^{s'}, y^{s'}]\!]$ with $\forall_{i \neq j} z'_j = z_j$ and $z'_i \in [-\infty, z_i]$ arbitrary. So $C_{s'}$ is not bounded. The proof for $y_i^{s'} = \infty$ is similar.

□

Proposition 4.2. *Let P be the set of all finite vertices of regions $C_{s'} \in C_{\Phi,1}$.*
We then have $P \subset [\![x^C, y^C]\!]$ for

$$x_i^C := \min_{\substack{t \in [T],\ \sigma \in \{0,1\}^{n_1 \times t} \\ \forall_{t \in [T]} \sigma_i(t) = 0}} g(t; \sigma),$$
$$y_i^C := \max_{\substack{t \in [T],\ \sigma \in \{0,1\}^{n_1 \times t} \\ \forall_{t \in [T]} \sigma_i(t) = 1}} g(t; \sigma).$$

In fact every component in x_i^C is maximal and every component in y_i^C minimal with $P \subset [\![x^C, y^C]\!]$. We further have in particular $x_i^C = \min_{t \in [T]} g_i(t; s^{x_i^C})$ and $y_i^C = \max_{t \in [T]} g_i(t; s^{y_i^C})$, where

$$s_j^{x_i^C}(t) := \begin{cases} 1 & (i \neq j) \wedge (v_{ij} > 0) \\ 0 & (i = j) \vee (v_{ij} \leq 0) \end{cases}$$
$$s_j^{y_i^C}(t) := \begin{cases} 1 & (i = j) \vee (v_{ij} < 0) \\ 0 & (i \neq j) \wedge (v_{ij} \geq 0) \end{cases}$$

Remark 4.4. From Proposition 4.2 we obtain in particular, that we $C_{s'} \subset [\![x^C, y^C]\!]$ for all regions $C_{s'} \in C_{\Phi,1}$ with non-constant spiketrain s': By Lemma 4.8 all vertices of $C_{s'} = [\![x^{s'}, y^{s'}]\!]$ are finite, so $x^{s'}, y^{s'} \in P \subset [\![x^C, y^C]\!]$ and therefore $[\![x^{s'}, y^{s'}]\!] \subset [\![x^C, y^C]\!]$.

Remark 4.5. Proposition 4.2 is in particular useful, when computing $\left|C_{\Phi,1}\right|$: We know that every region $C_{s'} \in C_{\Phi,1}$ has at least one finite vertex by Lemma 4.7 and by Proposition 4.2 we know where to search for those vertices.

Proof. We will first proof $P \subset [\![x^C, y^C]\!]$: Let $v \in P$ be a finite vertix of region $C_{s'} = [\![x^{s'}, y^{s'}]\!]$. Now $\forall_{i \in [n_1]} v_i \in \{x_i^{s'}, y_i^{s'}\} \cap \mathbb{R}$, so in particular in the case of $v_i = x_i^{s'}$ for a $i \in [n_1]$ there is time-step $t_1 \in [T]$ such that $s'(t_1) = 1$; is instead $v_i = y_i^{s'}$ then there is time-step $t_0 \in [T]$ such that $s'(t_0) = 0$. Let now $i \in [n_1]$ and $v_i = x_i^{s'}$ be given. Let further t_1 be minimal with $s'(t_1) = 1$. Let us now define $s^1 \in \{0,1\}^{n_1 \times t_1}$ by $s^0(t) := \mathbf{0}_{n_1}$. Now by choice of t_1, we have $\forall_{t < t_1} s_i^1(t) = 0$. Further, since $g(t; \sigma)$ only consumes the first $t-1$ time-steps of the given spike train σ (see Remark 4.1), we have $g_i(t_1; s^0) = g_i(t_1; s')$. We therefore get

$$x_i^C = \min_{\substack{t \in [T],\ \sigma \in \{0,1\}^{n_1 \times t} \\ \forall_{t \in [T]} \sigma_i(t) = 0}} g_i(t; \sigma) \leq \sup_{\substack{t \in [T] \\ s'_i(t) = 1}} g_i(t; s')$$

since $g_i(t_1; s^1)$ appears on the left hand and $g_i(t_1; s')$ on the right hand. We can similarly construct s^0 and obtain

$$y_i^C = \max_{\substack{t \in [T],\ \sigma \in \{0,1\}^{n_1 \times t} \\ \forall_{t \in [T]} \sigma_i(t) = 1}} g_i(t; \sigma) \geq \inf_{\substack{t \in [T] \\ s'(t) = 0}} g_i(t; s').$$

So we get $v \in [\![x^C, y^C]\!]$.

Further x^C, y^C are in fact chosen optimal: Let $i \in [n_1]$. Now the spike-train $s' := s^{x_i^C}$ is constant in every component, so $C_{s'} = [\![x^{s'}, y^{s'}]\!]$ is non-empty by Lemma 4.6 and has a finite

vertex $v \in \mathbb{R}^{n_1}$ (exactly one) by Lemma 4.7. This vertex must have $v_i = y_i^{s'}$, since $\forall_{t\in[T]} s_i'(t;x) = 0$ by definition and therefore $x_i^{s'} = -\infty$. Further

$$v_i = y_i^{s'} = \inf_{\substack{t\in[T]\\ s_i'(t)=0}} g_i(t;s') = \min_{t\in[T]} g_i(t;s') = x_i^C$$

by definition. We similarly get a vertex $v \in P$ with $v_i = x_i^{s^{y_i^C}} = y_i^C$.

Let us now show that it suffices to compute g on $s_j^{x_i^C}$ and $s_j^{y_i^C}$ to obtain the boundaries. For that purpose we have to take another look at the definition of g. Since the definition is quite clunky we have split it up into the following functions

$$g_i^r(t;\sigma) := -\frac{\sum_{k=1}^t \beta^{t-k}\left(\alpha^k i_i(0) + b + \sum_{l=1}^k \alpha^{k-l}\langle v_i, \sigma(l-1)\rangle\right) + \beta^t u_i(0)}{\sum_{k=1}^t \beta^{t-k}\sum_{l=1}^k \alpha^{k-l}},$$

$$g_i^\vartheta(t;\sigma) := \frac{\vartheta\left(1 + \sum_{k=1}^{t-1}\beta^{t-k}\sigma_i(k)\right)}{\sum_{k=1}^t \beta^{t-k}\sum_{l=1}^k \alpha^{k-l}}$$

such that $g(t;\sigma) = g^r(t;\sigma) + g^\vartheta(t;\sigma)$. Now fixing t and requiring $\forall_{t\in[T]}\sigma_i(t) = 0$, the function $g_i^\vartheta(t;\sigma)$ constant and $g_i^r(t;\sigma)$ is minimal for $\sigma_j(t) = 1_{((v_i)_j>0)}$ for $i \neq j$, so in particular for $\sigma = s^{x_i^C}$. Hence we indeed have $x_i^C = \min_{t\in[T]} g_i(t; s^{x_i^C})$. We similarly get $x_i^C = \min_{t\in[T]} g_i(t; s^{y_i^C})$. □

Theorem 4.3 of [Nguyen et al., 2025] states the following:

Theorem 4.1. *Consider a discrete-time LIF-SNN Φ with T time steps, input dimension n_0 and n_1 neurons in the first hidden layer. Then the number of constant regions is bounded by*

$$|C_\Phi| \leq \left|C_{\Phi,1}\right| \leq \begin{cases} \sum_{i=0}^{n_0}\left(\frac{T^2+T}{2}\right)^i \binom{n_1}{i} & n_1 > n_0, \\ \left(\frac{T^2+T+2}{2}\right)^{n_1} & \text{otherwise.} \end{cases} \tag{22}$$

While we were able to get promising empirical results in Section 5 that seem to confirm thatTheorem 4.1 also holds for r. LIF-SNN, we were not able to proof them. We will therefore first give a simplified proof for the case of $W = I_{n_1}$ and $V = \mathbf{0}_{n_1\times n_1}$ and then showcase why the generalization to arbitrary V fails. Let us first state the stripped down version:

Theorem 4.2. *Consider a r. LIF-SNN Φ with T time steps, input dimension n_0, trivial weights, i.e. $W = I_{n_1}$ and no recurrent connections, i.e. $V = \mathbf{0}_{n_1\times n_1}$. Then the number of constant regions is bounded by*

$$|C_\Phi| \leq \left|C_{\Phi,1}\right| \leq \left(\frac{T^2+T+2}{2}\right)^{n_1} \tag{23}$$

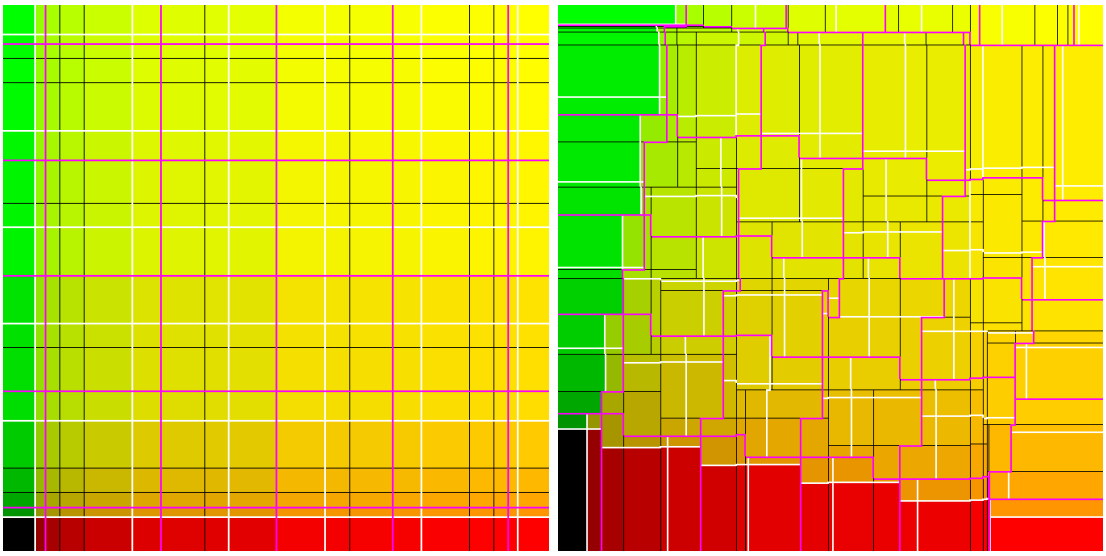
Proof. As we have stated before, $|C_\Phi| \leq \left|C_{\Phi,1}\right|$ follows directly from the fact that the output of the whole network only depends on the binary spike trains of the first layer.

Let us further proof $\left|C_{\Phi,1}\right| \leq \left(\frac{T^2+T+2}{2}\right)^{n_1}$. In the following we will use Φ_τ to mean the r. LIF-SNN that only differ from Φ by $T_{\Phi_\tau} = \tau$. It suffices now to show $\left|C_{\Phi_\tau,1}\right| \leq \left(\frac{\tau^2+\tau+2}{2}\right)^{n_1}$ inductively. Let first □

We will now try to give a high-level overview on why this is the case. We will assume $W = I_{n_1}$ again. The proof in [Nguyen et al., 2025] makes use of two major facts:

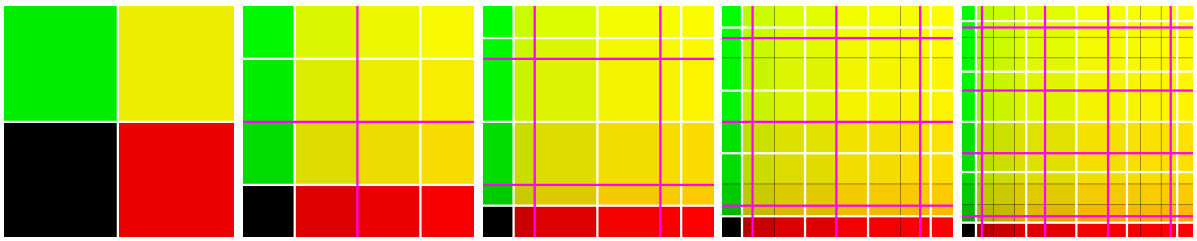
While we were able to get promising empirical results in Section 5 that seem to confirm thatTheorem 4.1 also holds for r. LIF-SNN, we were not able to proof them.

We will now try to give a high-level overview on why this is the case. We will assume $W = I_{n_1}$ again. The proof in [Nguyen et al., 2025] makes use of two major facts:



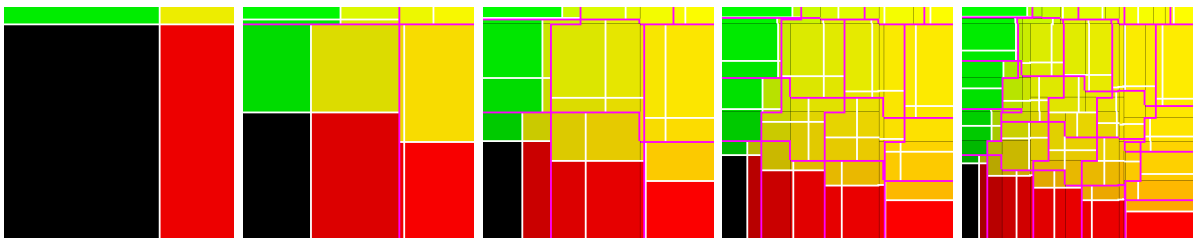
(a) The landscape of a d.t. LIF-SNN. **(b)** The landscape of a r. LIF-SNN.

Figure 4.2: Comparison of d.t. LIF-SNN vs. r. LIF-SNN; White lines separate regions with a different spike count; Pink lines separate regions with a different spike count disregarding the latest spikes from the last times-step



(a) $T = 1$ **(b)** $T = 2$ **(c)** $T = 3$ **(d)** $T = 4$ **(e)** $T = 5$

Figure 4.3: TODO



(a) $T = 1$ **(b)** $T = 2$ **(c)** $T = 3$ **(d)** $T = 4$ **(e)** $T = 5$

Figure 4.4: TODO

- For $W = I_{n_1}$, $V = \mathbf{0}_{n_1 \times n_1}$ and $\alpha = 0$ we get equations

$$i(t) = x \tag{24}$$
$$p(t) = \beta u(t-1) + i(t) + b \tag{25}$$
$$s(t) = H(p(t) - \vartheta \mathbf{1}_n) \tag{26}$$
$$u(t) = p(t) - \vartheta s(t). \tag{27}$$

  and in particular

$$p(t;x) = \beta^t u(0) + (x+b) \sum_{k=1}^{t} \beta^{t-k} - \vartheta \sum_{k=1}^{t-1} \beta^{t-k} s(k) \tag{28}$$

  Notice that the components i_i, p_i, s_i, u_i are computed in parallel, without affecting each other, in particular due to $V = 0$.

  , the neurons of the first layer don't influence each other in d.t. LIF-SNN and the proof can therefore just focus on how many different spikestrains a single neuron can possibly emit. Due to, each neurons input has its own independent input component. So the number of different values the whole first hidden layer can emit is just the product of the maximal number of spikes each neuron can emit. Since the parameters can be arbitrary, that maximal number is the same for each neuron. This is why the upper bound in Theorem 4.1 has an exponent n_1.

- Let us regard the definition of g for $\alpha = 0$ and $V = 0$:

$$g(t;\sigma) = -b - \frac{\beta^t u(0) - \vartheta \left(1 + \sum_{k=1}^{t-1} \beta^{t-k} \sigma(k)\right)}{\sum_{k=1}^{t} \beta^{t-k}}$$

  Let us further regard a time-step $t \in [T]$. If we regard the spiketrains $(s_i(t';x))_{t' \in [t]}$ we get for $x \in \mathbb{R}^{n_1}$, we see that $s_i(\cdot;x)$ only changes depending on x_i, since $V = 0$. Further, if we have $x, y \in \mathbb{R}^{n_1}$ such that $\forall_{i \neq j} x_j = y_j$ with $x_i < y_i$ and $s_i(\cdot;x) \neq s_i(\cdot;y)$, then $s_i(\cdot;x) < s_i(\cdot;y)$ lexiographically by Lemma 4.5. In fact, not only lexiographically, but even stronger, but every spike appears earlier in $s_i(\cdot;y)$ than in $s_i(\cdot;x)$.

  If we now regard g, we therefore see that $g_i(t; s_i(\cdot;x)) \geq g_i(t; s_i(\cdot;y))$. So the ordering has been reversed. But since the points returned by g define the points where the regions are split (compare Proposition 4.1), this means that only one region among all of those that share the same number of spikes will be split in time-step t. Since there are only $t+1$ possible counts of spikes at t, we therefore only get $t+1$ possible splits when going from t to $t+1$, wherefore we get $\approx \sum_{t=1}^{T} t = \frac{T^2+T}{2}$ regions in this neuron.

Now since the neurons of the first layer of a r. LIF-SNN can influence each other, it does not suffice to look at the behavior of a single neuron. We instead have to look at all the neurons together. One approach that seemed quite promising was to focus on partitioning the input space into regions with the same number of spikes in every component. This gives some control onto the next regions, but still not enough to have the same reversed orderings of regions in $C_{\Phi,1}$ and points induced by g: Consider … Further, even for $\beta = 1$, where we clearly get that g is constant on the whole region with constant number of spike trains, we don't quite have enough enough control on the behavior of the new regions.

5 Experimental results

To better understand the landscape of the input of a r. LIF-SNN with the goal of prooving Theorem 4.1 in mind, we have created the following programs.

5.1 Computing the number of regions

We use $W = I_n$ yet again in the following section, since it simplifies the following algorithm considerably and we hope to quite easily proof the situation for arbitrary W, once we have done it for $W = I_n$.

Algorithm 1 Compute regions similar to Proposition 4.1

function SubRegions(Φ, t, st, x^C, y^C)
  $c \leftarrow g_\Phi(t+1; st)$
  $subRegions \leftarrow \{\}$
  for $sp \in \{0,1\}^{n_1}$ **do**
    $x' \leftarrow (\text{if } sp_i = 0 \text{ then } x_i^C \text{ else } \max(x_i^C, c_i))_{i \in [(n_\Phi)_1]}$
    $y' \leftarrow (\text{if } sp_i = 1 \text{ then } y_i^C \text{ else } \min(y_i^C, c_i))_{i \in [(n_\Phi)_1]}$
    if $x' < y'$ **then**
      $subRegions \leftarrow subRegions \cup \{(x', y', st +\!\!+ [sp])\}$
    end if
  end for
  return *subRegions*
end function
function RegionsWithST(Φ, t, st, x^C, y^C)
  if $t \geq T_\Phi$ **then**
    return $\{(st, x^C, y^C)\}$
  end if
  $regions \leftarrow \{\}$
  for $(x', y', st) \in$ SubRegions(Φ, t, st, x^C, y^C) **do**
    $newRegions \leftarrow$ RegionsWithST($\Phi, t+1, st, x', y'$)
    $regions \leftarrow regions \cup newRegions$
  end for
  return *regions*
end function
function CompRegions(Φ)
  return RegionsWithST($\Phi, 0, [(0,...,0)], (-\infty,...,-\infty), (\infty,...,\infty)$)
end function

Algorithm 1 is motivated by Proposition 4.1, but instead of iterating over all spike trains $s' \in \{0,1\}^{n_1 \times T}$ and checking whether $C_{s'} \neq \emptyset$ by evaluating $x_i^{s'} < y_i^{s'}$, we use a more efficient algorithm.

A few words on notation and representation in Algorithm 1: We represent spike trains as lists, including the initial and trivial spike $s^{[l]}(0) = \mathbf{0}_{n_l}$. So e.g. $s' \in \{0,1\}^{n_1 \times T}$ with $\forall_{t \in [T]} s_1'(t) = 1$ and $\forall_{t \in [T]} s_i'(t) = 1$ for all $i \neq 1$ is represented by $st = [(0,...,0),(1,0,...,0),...,(1,0,...,0)]$. Further a region $C_{s'} = [\![x^{s'}, y^{s'}]\!]$ is represented as a tuple $(st, x^{s'}, y^{s'})$, where st is the list corresponding to s'. We also use g_Φ for g as defined in Lemma 4.1, used parameters from Φ. Similarly T_Φ is T from Φ.

Lemma 5.1. *Let Φ be a r. LIF-SNN. With $W^{[1]} = I_{n_1}$. The algorithm* `CompRegions`(Φ) *from Algorithm 1 computes $C_{\Phi,1}$.*

Proof. Let us write $st[t]$ for the t-th element of st, where st is indexed starting with 0.

In the following we will use Φ_τ to mean the r. LIF-SNN that only differ from Φ by having $T_{\Phi_\tau} = \tau$.

Let us now show that given $C_{s'} = [\![x^{s'}, y^{s'}]\!] \in C_{\Phi_{t-1},1}$, the algorithm $\texttt{SubRegions}(\Phi, t, s', x^{C_{s'}}, y^{C_{s'}})$ computes all regions $C_{s''} = [\![x^{s''}, y^{s''}]\!] \in C_{\Phi_t,1}$, where s'' is an extension of s' with non-empty region $C_{s''}$, i.e. $\forall_{t' \in [t-1]} s'(t') = s''(t')$ and $x^{C_{s''}} < y^{C_{s''}}$.

Notice first, that by Proposition 4.1, we have $x_i^{s''} = \sup(x_i^{s'}, g_i(t; s''))$ and $y_i^{s''} = y_i^{s'}$ if $s_i''(t) = 1$ as well as $x_i^{s''} = x_i^{s'}$ and $y_i^{s''} = \inf(y_i^{s'}, g_i(t; s''))$ otherwise.

So we have

$$\begin{aligned} C_{s''} &= \prod_{i \in [n_1]} [x_i^{s''}, y_i^{s''}) \\ &= \prod_{i \in [n_1]} \begin{cases} [x_i^{s'},\ \inf(y_i^{s'}, g_i(t; s''))) & s''(t) = 0 \\ [\sup(x_i^{s'}, g_i(t; s''))_i,\ y_i^{s'}) & s''(t) = 1 \end{cases}. \end{aligned}$$

Comparing with the code of `SubRegions`, it is clear that given a region $C_{s'} \in C_{\Phi_{t-1},1}$, the algorithm computes the set of all extensions s'' of s' such that $C_{s''}$ is non-empty.

We similar get that $\texttt{SubRegions}(\Phi, 0, [(0, ..., 0)], (-\infty, ..., -\infty), (\infty, ..., \infty))$ computes all regions $C_{s''} \in C_{\Phi_1,1}$ with spikes $s'' \in \{0,1\}^{n_1 \times 1}$. First note that again by Proposition 4.1, $x_i^{s''} = g_i(t; s'')$ and $y_i^{s''} = \infty$ if $s_i''(t) = 1$ as well as $x_i^{s''} = -\infty$ and $y_i^{s''} = g_i(t; s'')$ otherwise. So we have

$$C_{s''} = \prod_{i \in [n_1]} \begin{cases} [-\infty,\ g_i(t; s'')) & s''(t) = 0 \\ [g_i(t; s'')_i,\ \infty) & s''(t) = 1 \end{cases}.$$

Which is clearly what $\texttt{SubRegions}(\Phi, 0, [(0, ..., 0)], (-\infty, ..., -\infty), (\infty, ..., \infty))$ will compute.

Since `RegionsWithST` just repeatedly maps `SubRegions` over an initial value of $\{(\Phi, 0, [(0, ..., 0)], (-\infty, ..., -\infty), (\infty, ..., \infty))\}$, and since `SubRegions` computes the regions of $C_{\Phi_\tau,1}$ from $C_{\Phi_{\tau-1},1}$; as well as $C_{\Phi_1,1}$ from the initial value, `RegionsWithST` computes $C_{\Phi,1}$. □

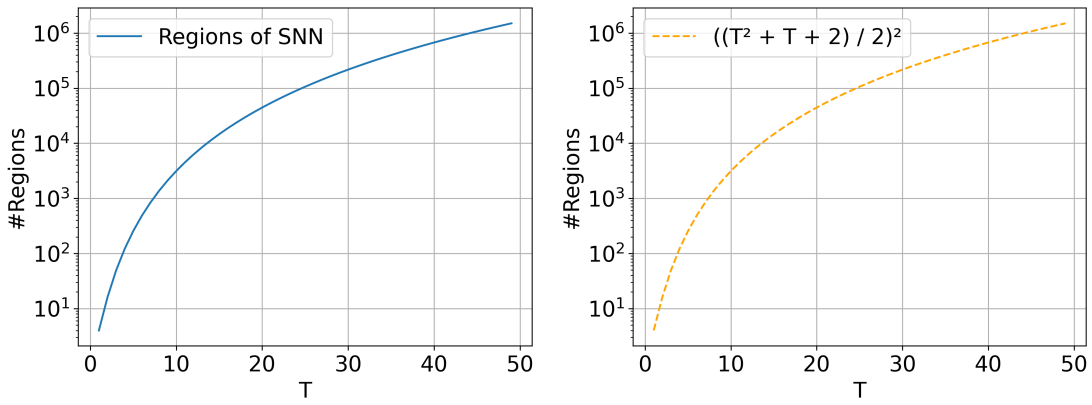
Algorithm 1 has been implemented in Python for reference; we have also implemented a more efficient version in C++. Due to the usage of floating point numbers, the results are only accurate up to a certain degree. In particular the C++ and Python implementations sometimes differ slightly, even for a low iteration number. The problem lies in the numerical instability of the algorithm. The landscape quite often includes empty regions $C_{s'} = [\![x^{s'}, y^{s'}]\!]$, i.e. $\exists_i x_i^{s'} = y_i^{s'}$, such that an implementation of g might just coincidentally return a slightly lower value for $x_i^{s'}$ than for $y_i^{s'}$.

Just skipping very slim regions does not seem correct either, since proper regions might just as well be very slim. Which brings us to the second issue of the implementations: Undercounting of regions. In Theorem B.15 of [Nguyen et al., 2025] it was proven, that the bound in Theorem 4.1 is tight, i.e. for every $T \in \mathbb{N}$ and $n_0 = n_1 \in \mathbb{N}$ there exists a d.t. LIF-SNN $\Phi_{\max,T,n_0}$ such that the bound $(\frac{T^2+T+2}{2})^{n_1}$ is reached. Suppose we compute the number of regions for this d.t. LIF-SNN. In our algorithm we used 64-bit doubles for each vector component, so we can represent at most 2^{64} values. So at the very latest at $T \approx 2^{32}\sqrt{2}$ the algorithm will start to undercount the regions, though we have reason to think that the problem might appear far earlier.

Now $\Phi_{\max,T,n_0}$ is essentially just a d.t. LIF-SNN with canonical parameters in the first layer, so $\beta = \vartheta = 1$, $W = I_{n_1}$ and $b = i(0) = \mathbf{0}_{n_1}$, but with a small positive number for every component $u_i(0) > 0$ of $u(0)$.

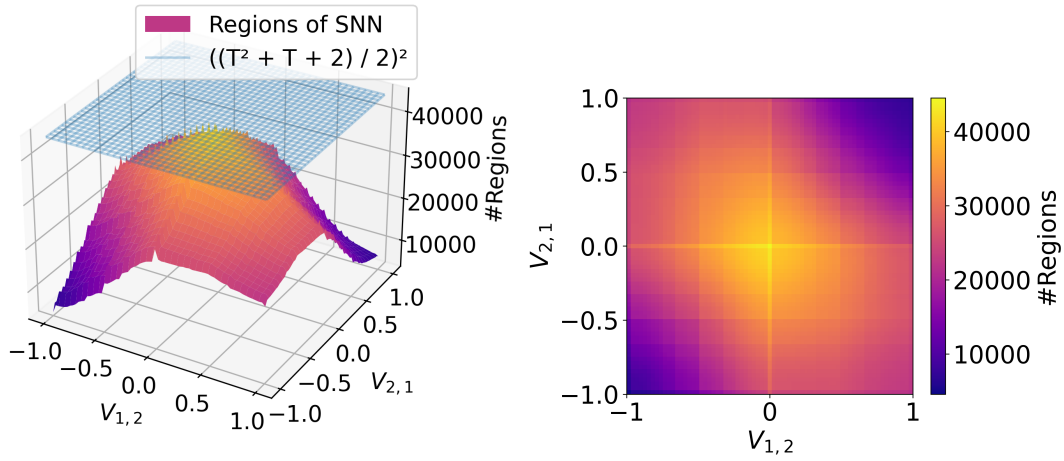
Let us consider such a d.t. LIF-SNN and take a look at the maximum number of regions in Fig. 5.1, computed by Algorithm 1.

Clearly the plot in Fig. 5.1a of the number of regions of $\Phi_{\max,T,n_0}$ corresponds to the theoretical limit in Fig. 5.1b.



(a) $\left|C_{\Phi_{\max,T,n_0},1}\right|$ for different values of T **(b)** Theoretical limit for d.t. LIF-SNN

Figure 5.1: Optimal Φ vs. theoretical limit



(a) 3D Plot **(b)** Color Plot

Figure 5.2: $\left|C_{\Phi,1}\right|$ for $T = 20$ and different values of $V_{1,2}$ and $V_{2,1}$

Let us now consider a r. LIF-SNN Φ that only differs in α or V to $\Phi_{\max,T,n_0}$. Changing $V_{1,2}$ or $V_{2,1}$ only decreases $\left|C_{\Phi_{0,V},1}\right|$ as can be seen in Fig. 5.2. If we instead just change α or even β, we also only find a decrease in the number of regions, as one can see in Fig. 5.3.

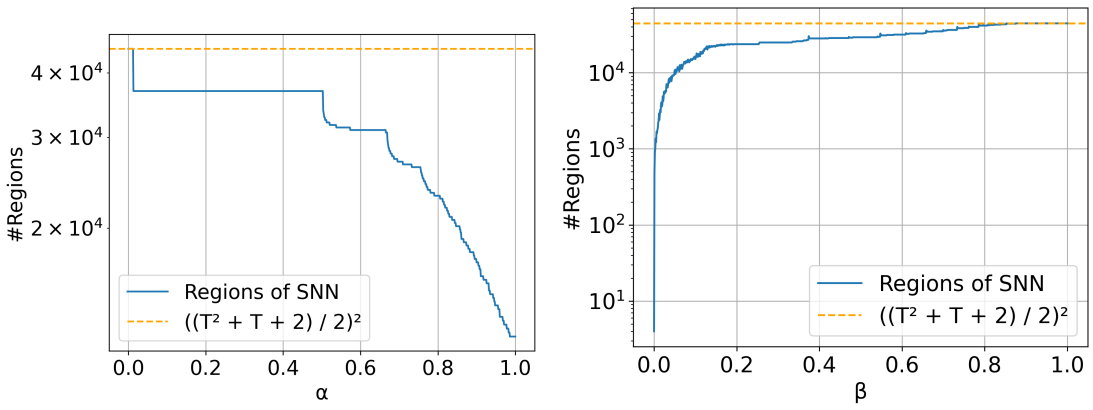
We have furthermore found, that in our experience even changes in multiple parameters do not increase the number of regions past the threshold.

Still, our hypothesis might be wrong, since our algorithms are insufficient and unable to compute the number of regions for high enough iterations efficiently; or the usage of floating point numbers might lead to too much undercounting of regions.

The only change that consistently pushes the number of regions above the upper bound is increasing β past its limit of 1, see Fig. 5.4 in contrast to Fig. 5.3b. This is also consistent with the proof of Theorem 4.1 in [Nguyen et al., 2025], since $\beta \in [0,1]$ is a critical assumption it has to make. Of course this bothers us not too much, since exponential growth instead of decay would not make a good model of the neurological realities.

5.2 Visualization of landscape

While we can use the previous algorithm to compute all regions for a 2-neuron network and draw them on the screen, the algorithm is quite slow for higher iterations due to its exponential



(a) Changing α decreases the number of regions **(b)** Changing β decreases the number of regions

Figure 5.3: No parameter seems to be able to push the number of regions beyond the upper bound for $T = 20$

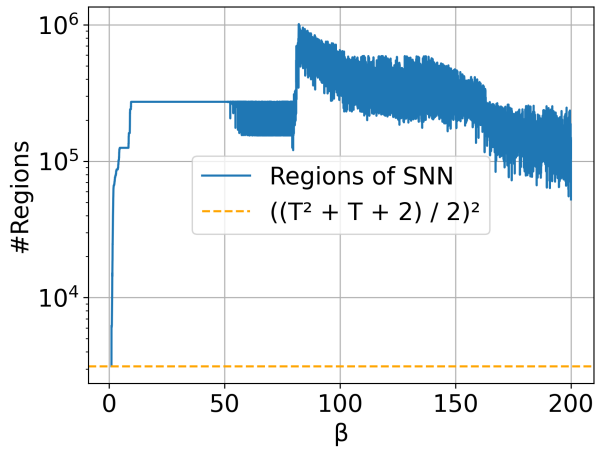


Figure 5.4: Changing β past the limit pushes the number of regions above the threshold for $T = 20$

nature. A previous iteration of our algorithms does not have that limitation:

We just compute $s^{[1]}$ on a grid in the input space and then compute the set of unique spike trains. Executed on the CPU this algorithm would be quite slow, however it is natural to implement it on the gpu instead.

But this has following disadvantages: Since there are no dynamically-sized lists in GLSL, the GPU programming language that we implemented the algorithm in, we used integers as fixed-sized lists of bits to represent the spike trains. Furthermore since there are no arbitrary precision integers on the gpu, we have choosen to use two 32-bit integers to represent spike trains with a length up to 64 time-steps.

Moreover, since we just evaluate $s^{[1]}$ on a grid, we might miss very slim regions located between nodes. Finally GLSL is also not flexible enough to allow arbitrary sized matrizes/vectors. We have therefore fixed ourselves to the simple and easily visualizable case of 2 neurons in the first layer.

Another limitation is the storage requirement of the algorithm: If we want to make sure not to miss regions with width of smaller than $\text{diam}_\Omega(C) \cdot 0.00025 \approx \text{diam}_\Omega(C) \cdot 2^{-12}$ in a direction, the grid needs to have at least a width of 2^{12}, so in total it needs 2^{24} nodes. For each of those we compute 8 bytes (64 bits), so we need $2^{27} = 128\text{MiB}$ of storage. Further, since we are using 2 neurons in layer one, the storage requirement grows quadratically in the width of the grid, so

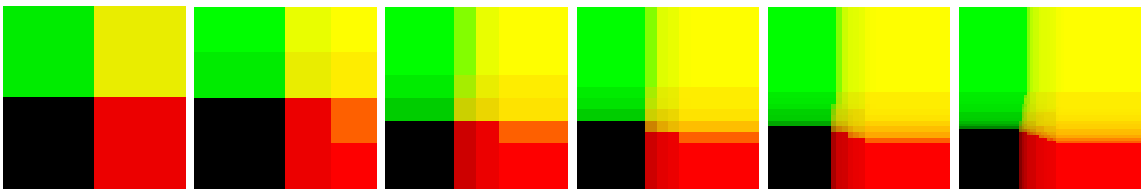
we quite quickly reach the limits of consumer hardware in storage.

On the other hand, this approach has the big advantage of allowing us to easily create visualizations of the landscape of a r. LIF-SNN by represent the pixels of an image with the grid. It is furthermore quite simple to port the program into the web browser using WebGL (TODO: see here). This allowed us to create a simple user interface for changing the parameters of the r. LIF-SNN, the coloring algorithm, etc. without too much fuss, which allows users to obtain a much better intuition for the problem at hand through its interactive visualization.

We also implemented region counting for this type of algorithm, but sadly due to the limited architecture of WebGL, we were not able to implement it on the web. The concrete problem is that we are not able to read out data from gpu buffers. This will change with the new WebGPU API, but sadly that API is not yet completely supported in most browsers at the time of writing.

We have instead implemented the region counting in python. The obvious way to implement it, to just read out the buffer of spike-trains from the gpu and use a standard-algorithm to determine the unique elements has proved to be a major bottleneck. As before, we probably want to be able to quickly compute our result for a width of 2^{12} for the grid, so for 2^{24} nodes in total. It is further well-known, that algorithms filtering out the unique elements of a list are $\Theta(n \log(n))$, compare e.g. [Ben-Or, 1983].

We have therefore implemented a different algorithm for finding the unique elements in this particular case, that is just $\Theta(n)$. We will assume $W = I_n$, though it should be possible to generalize it. Since we know due to Proposition 4.1 that all regions in $C_{\Phi,1}$ are half-open rectangles, it suffices to count all lower-left corners. So we just count all pixels such that the lower and left neighboring pixels are different (or don't exist since the pixel is located at the left/lower border of the grid).



(a) 2x2 **(b)** 4x4 **(c)** 8x8 **(d)** 16x16 **(e)** 32x32 **(f)** 64x64

Figure 5.5: Doubling the grid repeatedly to improve the accuracy

We further utilize the regions having rectangular shapes by first executing the algorithm on a narrow grid and then doubling the size of the grid repeatedly. During the next iteration we can then just use the previous result if it is the same on the surrounding four nodes from the previous iteration. A nice side-effect is that we are additionally getting results for smaller grid sizes, such that we can get some feel for how much our algorithm is undercounting the regions due to too narrow grids.

6 Conclusion

R. LIF-SNN are a natural extension of d.t. LIF-SNN that allows more compact and efficient approximation of functions. On the other hand it still remains to be seen if they are indeed a better suited model for fitting data and in particular reasoning. Especially our analysis in Section 4 and Section 5 might show that they are actually not so much stronger than d.t. LIF-SNN.

Bibliography

References

[Ben-Or, 1983] Ben-Or, M. (1983). Lower bounds for algebraic computation trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 80–86, New York, NY, USA. Association for Computing Machinery.

[Gerstner et al., 2014] Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

[Herrmann, 2025a] Herrmann, V. (2025a). dt-lif-snn-compute-regions - compute constant regions of r. lif-snn. GitHub repository.

[Herrmann, 2025b] Herrmann, V. (2025b). dt-lif-snn-compute-regions-depthsearch - compute constant regions of r. lif-snn. GitHub repository.

[Herrmann, 2025c] Herrmann, V. (2025c). dt-lif-snn-visualizer - visualize the constant regions of r. lif-snn. GitHub repository.

[Jones and Bergen, 2025] Jones, C. R. and Bergen, B. K. (2025). Large language models pass the turing test.

[Love, 2024] Love, F. (2024). Nntikz - tikz diagrams for deep learning and neural networks. GitHub repository.

[Nguyen et al., 2025] Nguyen, D. A., Araya, E., Fono, A., and Kutyniok, G. (2025). Time to spike? understanding the representational power of spiking neural networks in discrete time.

[Ou, 2019] Ou, C. (2019). lmu-thesis-latex - lmu thesis latex template. GitHub repository.

7 Appendix

Find the source code for our programs of Section 5 below.