On the geometry of recurrent spiking networks

Josue Casco-Rodriguez JC135@RICE.EDU
Rice University, Houston, TX

Editors: List of editors' names

Abstract

Biological neural networks are recurrent and transmit information via discrete spikes. However, neural network theories largely focus on deep feedforward architectures with continuous activations, and it is not clear to what extent these theories are relevant to the analysis and optimization of recurrent spiking neural networks (SNNs). We propose to study the geometry of multi-layer recurrent SNNs as piecewise-constant functions, which we visualize with our new algorithm, SplineCam-SNN. We first study how the parameters of SNNs affect their input-output geometry. In contrast to deep networks with continuous activations, recurrent (synaptic) weights appear to have a more limited influence on the geometry than skip connections, leakages, and delays. With these insights, we compare two plasticity methods: gradient descent aligns input-output geometry around data, while STDP, being focused on recurrent weights, cannot. Our findings emphasize the importance of skip connections, leakages, and delays for training SNNs, and suggest that these parameters may be promising targets for future plasticity algorithms.

Keywords: spiking, recurrent, neural, network, geometry, theory, spline, visualization

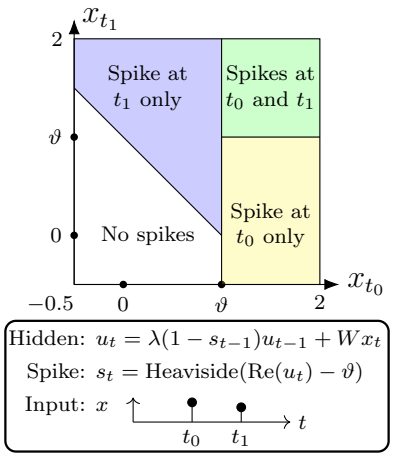

Figure 1: **Spiking neurons are piecewise-constant functions with discontinuous partitions.** Shown here are the four modes of activity from a single spiking neuron receiving scalar input x_t at times t_0 and t_1; each mode is a region of the unrolled input space with a fixed sequence of output spikes. The first region boundary ($x_{t_0} = \vartheta$) is where $u_{t_0} = \vartheta$, while the next two ($x_{t_0} + x_{t_1} = \vartheta$ and $x_{t_1} = \vartheta$) are where $u_{t_1} = \vartheta$ (depending on s_{t_0}). Figure 2 investigates how these partitions are affected by various SNN parameters, Figures 4 and 6 illustrate the specific influence of recurrent weights, and Figures 3 and 7 use our algorithm, SplineCam-SNN, to observe how plasticity changes these partitions in deep multi-neuron SNNs.

1. Introduction

Two longstanding challenges with neural networks are understanding them and connecting them to neuroscience. Sometimes, these two aims are aligned (e.g., convolutional networks (1; 2; 3; 4)), but often they are not. For example, much of deep learning theory rests on a foundation of continuous feedforward networks (5; 6; 7), even though biological neural networks are recurrent and communicate through binary signals (spikes). In particular,

spiking neural networks (SNNs) (8; 9; 10; 11; 12; 13), which resemble biological neurons, are frustrating targets of deep learning theory: there is little consensus about how best to parameterize or train them, and few works try interpreting them (14; 15; 16). SNNs could bring human-scale energy efficiency to deep networks and newfound insights to neuroscience (17; 18; 19; 20), but without a comprehensive theory for SNNs, this remains elusive.

Contributions. We propose a theory of SNNs as piecewise-constant functions: at each timestep, each neuron partitions the input space into regions where it does or does not spike. We also extend existing region computation methods (21; 22; 23) to SNNs via a new algorithm, SplineCam-SNN. With this framework, we reveal how SNN parameters affect the geometry of these regions: (1) skip connections are required for hidden layers to draw any new partitions, and (2) leakages, delays, input weights, and even resets can rotate partition boundaries, while recurrent weights can only shift them. Then we describe plasticity affecting region geometry: gradient descent aligns region boundaries around input data (like in continuous networks), while local learning via spike-timing-dependent plasticity can only shift certain region boundaries based on which regions contain the most data points. Our findings emphasize the importance of skip connections, leakages, and delays, and suggest that these parameters are promising targets for future plasticity algorithms.

2. Background and Related Work

2.1. Spiking Neural Networks

Leaky integrate-and-fire networks. Spiking neural networks (SNNs) are biologically inspired recurrent networks with hidden states that communicate only through binary signals (spikes). They often employ leaky integrate-and-fire (LIF) dynamics: at time t, hidden states (membrane potentials) $\boldsymbol{u}_t$ decay exponentially, with leakage decays $\boldsymbol{\lambda}$, and receive weighted inputs $\boldsymbol{W}_x\boldsymbol{x}_t$ and previous spikes $\boldsymbol{W}_r\boldsymbol{s}_{t-1}$. Neurons emit spikes $\boldsymbol{s}_t$ when $\boldsymbol{u}_t$ meets the firing thresholds $\boldsymbol{\vartheta}$; immediately after spiking, neurons' hidden states are reset to 0:

$$\boldsymbol{u}_t = \boldsymbol{\lambda} \odot (1 - \boldsymbol{s}_{t-1}) \odot \boldsymbol{u}_{t-1} + \boldsymbol{W}_x\boldsymbol{x}_t + \boldsymbol{W}_r\boldsymbol{s}_{t-1}, \quad \boldsymbol{s}_t = \text{Heaviside}(\text{Re}(\boldsymbol{u}_t) - \boldsymbol{\vartheta}) \tag{1}$$

We probe a variety of biologically inspired SNN parameters: (a) subthreshold oscillation (complex $\boldsymbol{\lambda}$) (24; 25; 26; 27; 28), soft and hard resets (either subtraction by the threshold or multiplication by 0 after spiking) (29; 30; 31; 32; 33; 34), dendritic delays (weighted input delays) (10; 35; 19; 36; 37; 38; 26; 39; 40; 41; 42; 43), (d) recurrent weights $\boldsymbol{W}_r$ (44; 45; 46; 47), (e) skip connections between layers (48; 49; 50; 51; 52; 11; 12; 53; 54; 55), and local plasticity methods like spike-timing-dependent plasticity (STDP) (56; 17; 34; 57; 58).

2.2. Continuous Piecewise-Linear Networks

Definition. Deep networks are cascades of linear and nonlinear functions. For example, an L-layer feedforward network could be a composition of L layers, where the ℓ-th layer has weights $\boldsymbol{W}_\ell$, biases $\boldsymbol{b}_\ell$, a nonlinearity σ_ℓ, and the form $f_\ell(\boldsymbol{x}) = \sigma_\ell(\boldsymbol{W}_\ell f_{\ell-1}(\boldsymbol{x}) + \boldsymbol{b}_\ell)$, where $f_0(\boldsymbol{x}) = \boldsymbol{x}$. If all the nonlinearities in a network are continuous piecewise-linear (CPWL), the network is CPWL and thus partitions its input space into a parameter-dependent set of regions Ω. Within each region $\omega \in \Omega$, the network is linear: $f(\boldsymbol{x}) = \boldsymbol{W}_{\omega(\boldsymbol{x})}\boldsymbol{x} + \boldsymbol{b}_{\omega(\boldsymbol{x})}$.

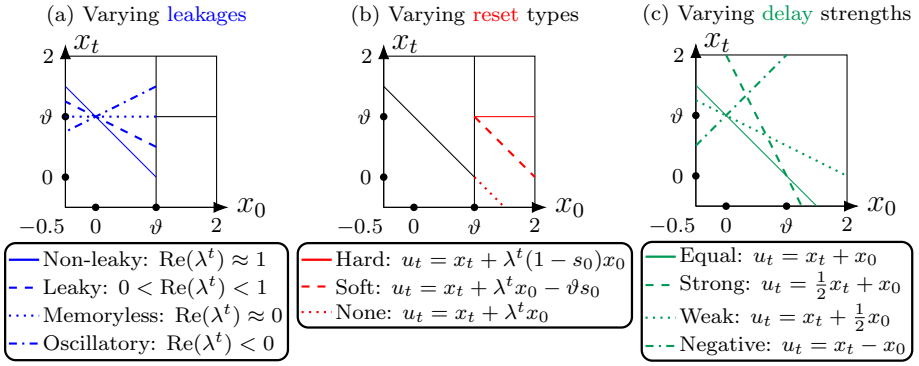

Figure 2: **Leakages, resets, and delays can rotate partition boundaries.** Multiple instances of Figure 1, but with varying parameters.

Applications. CPWL theories of neural networks (59; 60; 61), have yielded insights into deep network properties like normalization and adversarial robustness (6; 62; 63; 64). Three phenomena therein are relevant: (1) weights rotate the boundaries of partitions, while biases shift them; (2) hidden layers divide existing linear regions by drawing new boundaries; and (3) training a network orients its partitions around data. Additionally, the linear regions of CPWL networks can be exactly computed (21; 22; 23), but existing methods do not support nonlinear recurrent networks or discontinuous activations (i.e., SNNs). A few works have discussed the piecewise behavior of SNNs, but have done so either via spike *timings* (with restrictive assumptions) (65; 66) or in the *hidden state space* of SNNs (16; 67; 68), while we show how various factors affect the input-output geometry of SNNs in their *input space*.

3. Results

3.1. Parameters and Partitions

Partition calculation algorithm. We inspect *constant regions* of an SNN, portions of input space[1] with fixed spike patterns. We calculate them by iterating through each existing region for each neuron for each timestep. Each region, if applicable, is divided along lines where hidden states meet spike thresholds. For simple cases, we do this analytically, but for deep networks, we made an algorithm, SplineCam-SNN (see Appendices B and C).

Leakages and delays rotate partition boundaries; recurrent weights shift them. Training continuous networks requires fitting their partitions to data: weights rotate boundaries, and biases shift them. As for SNNs, Figure 2 reveals that several parameters rotate boundaries: leakages, delays, input weights, and resets, with the first two being the most direct[2]. However, partition boundaries from delays and complex leakages are less constrained than those from real leakages. Meanwhile, Figures 4 and 9 show that recurrent weights only shifts boundaries (since they transmit binary signals), unlike weights in continuous networks, which rotate boundaries (Figure 5). These findings imply that recurrent weights may be less useful than leakages and delays for fitting partitions to data.

1. Our input space is over *unrolled* sequences, e.g., $\mathbb{R}^{T\times d}$ for a d-dimensional sequence of T elements.
2. Input weights can rotate partitions, but only by scaling axes of input space rather than directly rotating any specific boundaries per se. As for resets, only hard ones can rotate boundaries (Figure 2).

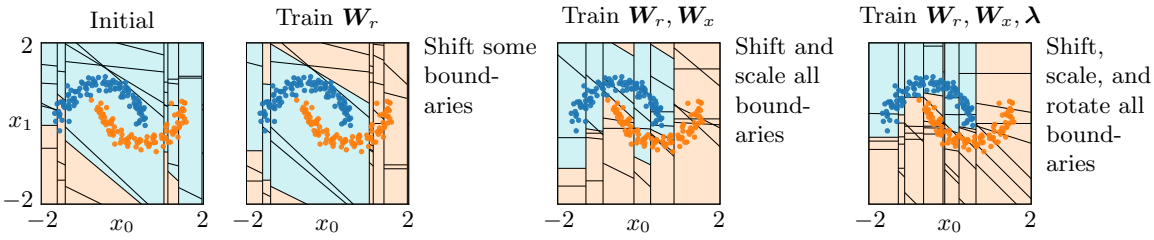

Figure 3: **Future plasticity algorithms should train all parameters.** We use SplineCam-SNN to compare instances of gradient descent in a deep SNN classifying a 2-timestep toy dataset. Colors denote true class labels (of data points) and the network's classification decisions (of input regions). Feedforward weights to hidden layers are always trained. Recurrent weights shift some, input weights scale all, and leakages rotate some boundaries. These differences help explain the limited expressivity of plasticity algorithms like STDP, which at most train input and recurrent weights (Figure 8).

Hidden layers need input skip connections to draw new partitions. In continuous networks, hidden layers draw new partitions from those of previous layers, but in SNNs, they cannot, since previous layers' outputs are constant within each region. Instead, hidden layers map their own spike patterns to each existing constant region (e.g., assign a decision to each region in a classification task, like in Figures 3 and 7). However, hidden layers can draw new partitions if and only if they observe the original input via skip connections.

3.2. Plasticity and Partitions

SNNs align partitions around data. Continuous networks are known to align their partitions to data during training. We use SplineCam-SNN in Figures 3 and 7 to confirm that SNNs trained via gradient descent also align their partitions around data. If SNNs align partitions to data, then they may be compatible with additional findings and techniques from spline theory concerning batch normalization and adversarial robustness (62; 63; 64).

Local plasticity methods need to train boundary-rotating parameters. Finally, we compare two popular SNN plasticity algorithms: surrogate gradient descent (global) and STDP (local). The former rotates and shifts partitions to align with data, while the latter, focused on recurrent weights, only shifts the boundaries of existing regions based on where input data is common (Figures 6 and 3). Under the right conditions, this expansion of data-heavy partitions could improve robustness to perturbations (69; 70), as in continuous networks (63; 64). However, plasticity algorithms focused on recurrent weights cannot rotate partition boundaries to align them with data—new plasticity algorithms should also train boundary-rotating parameters like leakages and delays.

4. Conclusions and Future Work

We have introduced a theory of SNNs as piecewise-constant functions, and an algorithm to visualize them as such. Our findings show the relative importance of skip connections, leakages, and delays, rather than recurrent weights, and suggests they are lucrative targets

of future plasticity algorithms (71). While this message is not wholly unique, our perspective is uniquely based on a geometric theory of SNNs as functions and unites disparate findings based on gradients or empirical observations (38; 35; 54; 53). Future work could examine how adaptation (26; 27), initialization (45; 72), normalization (73; 74; 75; 76; 77; 78; 79), and regularization (80; 30) affect partition geometry.

References

[1] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), 1980.

[2] Grace W Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of cognitive neuroscience*, 33(10), 2021.

[3] Alessia Celeghin, Alessio Borriero, Davide Orsenigo, Matteo Diano, Carlos Andrés Méndez Guerrero, Alan Perotti, Giovanni Petri, and Marco Tamietto. Convolutional neural networks for vision neuroscience: significance, developments, and outstanding issues. *Frontiers in Computational Neuroscience*, 17, 2023.

[4] David Daniel Cox and Thomas Dean. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 24(18), 2014.

[5] Daniel A Roberts, Sho Yaida, and Boris Hanin. *The principles of deep learning theory*, volume 46. Cambridge University Press, 2022.

[6] Randall Balestriero, Ahmed Imtiaz Humayun, and Richard G Baraniuk. On the geometry of deep learning. *Notices of the American Mathematical Society*, 72(4), 2025.

[7] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1), 1998.

[8] Kai Malcolm and Josue Casco-Rodriguez. A comprehensive review of spiking neural networks: Interpretation, optimization, efficiency, and best practices. *arXiv preprint arXiv:2303.10780*, 2023.

[9] Jason K Eshraghian, Max Ward, Emre O Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 2023.

[10] Chenxiang Ma, Xinyi Chen, Yanchen Li, Qu Yang, Yujie Wu, Guoqi Li, Gang Pan, Huajin Tang, Kay Chen Tan, and Jibin Wu. Spiking neural networks for temporal processing: Status quo and future prospects. *arXiv preprint arXiv:2502.09449*, 2025.

[11] Chenlin Zhou, Han Zhang, Liutao Yu, Yumin Ye, Zhaokun Zhou, Liwei Huang, Zhengyu Ma, Xiaopeng Fan, Huihui Zhou, and Yonghong Tian. Direct training high-performance deep spiking neural networks: a review of theories and methods. *Frontiers in Neuroscience*, 18, 2024.

[12] Yufei Guo, Xuhui Huang, and Zhe Ma. Direct learning-based deep spiking neural networks: a review. *Frontiers in Neuroscience*, 17, 2023.

[13] Guoqi Li, Lei Deng, Huajin Tang, Gang Pan, Yonghong Tian, Kaushik Roy, and Wolfgang Maass. Brain-inspired computing: A systematic survey and future trends. *Proceedings of the IEEE*, 2024.

[14] Youngeun Kim and Priyadarshini Panda. Visual explanations from spiking neural networks using inter-spike intervals. *Scientific reports*, 11(1), 2021.

[15] Elisa Nguyen, Meike Nauta, Gwenn Englebienne, and Christin Seifert. Feature attribution explanations for spiking neural networks. In *2023 IEEE 5th International Conference on Cognitive Machine Intelligence (CogMI)*, 2023.

[16] Nuno Calaim, Florian A Dehmelt, Pedro J Gonçalves, and Christian K Machens. The geometry of robustness in spiking neural networks. *Elife*, 11, 2022.

[17] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. *Brain Sciences*, 12(7), 2022.

[18] Anthony Zador, Sean Escola, Blake Richards, Bence Ölveczky, Yoshua Bengio, Kwabena Boahen, Matthew Botvinick, Dmitri Chklovskii, Anne Churchland, Claudia Clopath, et al. Catalyzing next-generation Artificial Intelligence through NeuroAI. *Nature Communications*, 14(1), 2023.

[19] Kwabena Boahen. Dendrocentric learning for synthetic intelligence. *Nature*, 612(7938), 2022.

[20] Dhireesha Kudithipudi, Catherine Schuman, Craig M Vineyard, Tej Pandit, Cory Merkel, Rajkumar Kubendran, James B Aimone, Garrick Orchard, Christian Mayr, Ryad Benosman, et al. Neuromorphic computing at scale. *Nature*, 637(8047), 2025.

[21] Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard G. Baraniuk. SplineCam: Exact visualization and characterization of deep network geometry and decision boundaries. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[22] Josue Casco-Rodriguez, Tyler Burley, CJ Barberan, Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Visualizing linear RNNs through unrolling. In *LatinX in AI @ NeurIPS 2024*, 2024.

[23] Arturs Berzins. Polyhedral complex extraction from relu networks using edge subdivision. In *International Conference on Machine Learning (ICML)*, 2023.

[24] Eugene M Izhikevich. Resonate-and-fire neurons. *Neural networks*, 14(6-7), 2001.

[25] Saya Higuchi, Sebastian Kairat, Sander Bohte, and Sebastian Otte. Balanced resonate-and-fire neurons. In *International Conference on Machine Learning (ICML)*, 2024.

[26] Lucas Deckers, Laurens Van Damme, Werner Van Leekwijck, Ing Jyh Tsang, and Steven Latré. Co-learning synaptic delays, weights and adaptation in spiking neural networks. *Frontiers in Neuroscience*, 18, 2024.

[27] Maximilian Baronig, Romain Ferrand, Silvester Sabathiel, and Robert Legenstein. Advancing spatio-temporal processing through adaptation in spiking neural networks. *Nature Communications*, 16(1), 2025.

[28] Wei Fang, Zhaofei Yu, Zhaokun Zhou, Ding Chen, Yanqi Chen, Zhengyu Ma, Timothée Masquelier, and Yonghong Tian. Parallel spiking neurons with high efficiency and ability to learn long-term dependencies. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[29] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In *European Conference on Computer Vision*, 2020.

[30] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. RMP-SNN: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2020.

[31] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-YOLO: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 2020.

[32] Yuhang Li, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu. A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration. In *International Conference on Machine Learning (ICML)*, 2021.

[33] Yufei Guo, Yuanpei Chen, Liwen Zhang, YingLei Wang, Xiaode Liu, Xinyi Tong, Yuanyuan Ou, Xuhui Huang, and Zhe Ma. Reducing information loss for spiking neural networks. In *European Conference on Computer Vision*, 2022.

[34] Eimantas Ledinauskas, Julius Ruseckas, Alfonsas Juršėnas, and Giedrius Buračas. Training deep spiking neural networks. *arXiv preprint arXiv:2006.04436*, 2020.

[35] Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. In *International Conference on Learning Representations (ICLR)*, 2024.

[36] Melika Payvand, Simone D'Agostino, Filippo Moro, Yigit Demirag, Giacomo Indiveri, and Elisa Vianello. Dendritic computation through exploiting resistive memory as both delays and weights. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, 2023.

[37] Simone D'Agostino, Filippo Moro, Tristan Torchet, Yiğit Demirağ, Laurent Grenouillet, Niccolò Castellani, Giacomo Indiveri, Elisa Vianello, and Melika Payvand. DenRAM: neuromorphic dendritic architecture with RRAM for efficient temporal processing with delays. *Nature Communications*, 15(1), 2024.

[38] Karim G. Habashy, Benjamin D. Evans, Dan F. M. Goodman, and Jeffrey S. Bowers. Adapting to time: Why nature may have evolved a diverse set of neurons. *PLOS Computational Biology*, 20(12), 2024.

[39] Balázs Mészáros, James C Knight, and Thomas Nowotny. Efficient event-based delay learning in spiking neural networks. *arXiv preprint arXiv:2501.07331*, 2025.

[40] Prajna G Malettira, Shubham Negi, Wachirawit Ponghiran, and Kaushik Roy. TSkips: Efficiency through explicit temporal delay connections in spiking neural networks. *arXiv preprint arXiv:2411.16711*, 2024.

[41] Julian Göltz, Jimmy Weber, Laura Kriener, Sebastian Billaudelle, Peter Lake, Johannes Schemmel, Melika Payvand, and Mihai A. Petrovici. DelGrad: Exact event-based gradients for training delays and weights on spiking neuromorphic hardware. *arXiv preprint arXiv:2404.19165*, 2025.

[42] Balázs Mészáros, James C Knight, and Thomas Nowotny. Learning delays through gradients and structure: emergence of spatiotemporal patterns in spiking neural networks. *Frontiers in Computational Neuroscience*, 18, 2024.

[43] Mario Chacón-Falcón, Alberto Patiño-Saucedo, Luis Camuñas-Mesa, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. BAM-SLDK: biologically inspired attention mechanism with spiking learnable delayed kernel synapses. *Neuromorphic Computing and Engineering*, 5(2), 2025.

[44] Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, 55(1), 2022.

[45] Aurora Micheli, Olaf Booij, Jan van Gemert, and Nergis Tömen. Deep activity propagation via weight initialization in spiking neural networks. In *2025 Neuro Inspired Computational Elements (NICE) Conference*, 2025.

[46] Jun-nosuke Teramae and Tomoki Fukai. Computational implications of lognormally distributed synaptic weights. *Proceedings of the IEEE*, 102(4), 2014.

[47] György Buzsáki and Kenji Mizuseki. The log-dynamic brain: how skewed distributions affect network operations. *Nature Reviews Neuroscience*, 15(4), 2014.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[49] Muhammad Shafiq and Zhaoquan Gu. Deep residual learning for image recognition: A survey. *Applied Sciences*, 12(18), 2022.

[50] Bobby He and Thomas Hofmann. Simplifying transformer blocks. In *International Conference on Learning Representations (ICLR)*, 2024.

[51] Marcus Kaiser, Matthias Goerner, and Claus C Hilgetag. Criticality of spreading dynamics in hierarchical cluster networks without inhibition. *New Journal of Physics*, 9(5), 2007.

[52] Rafael Lorente De N. Analysis of the activity of the chains of internuncial neurons. *Journal of Neurophysiology*, 1(3), 1938.

[53] Yifan Hu, Lei Deng, Yujie Wu, Man Yao, and Guoqi Li. Advancing spiking neural networks toward deep residual learning. *IEEE Transactions on Neural Networks and Learning Systems*, 36(2), 2024.

[54] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[55] Yimeng Shan, Xuerui Qiu, Rui-jie Zhu, Jason K Eshraghian, Malu Zhang, and Haicheng Qu. SynA-ResNet: Spike-driven ResNet achieved through OR residual connection. *arXiv preprint arXiv:2311.06570*, 2023.

[56] Alexander G Ororbia. Brain-inspired machine intelligence: A survey of neurobiologically-plausible credit assignment. *arXiv preprint arXiv:2312.09257*, 2023.

[57] Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: a Hebbian learning rule. *Annual Review of Neuroscience*, 31(1), 2008.

[58] Daniel E Feldman. The spike-timing dependence of plasticity. *Neuron*, 75(4), 2012.

[59] Randall Balestriero et al. A spline theory of deep learning. In *International Conference on Machine Learning (ICML)*, 2018.

[60] Randall Balestriero and Richard G Baraniuk. Mad max: Affine spline insights into deep learning. *Proceedings of the IEEE*, 109(5), 2020.

[61] Justin Sahs, Ryan Pyle, Aneel Damaraju, Josue Ortega Caro, Onur Tavaslioglu, Andy Lu, Fabio Anselmi, and Ankit B Patel. Shallow univariate ReLU networks as splines: initialization, loss surface, Hessian, and gradient flow dynamics. *Frontiers in Artificial Intelligence*, 5, 2022.

[62] Randall Balestriero and Richard G Baraniuk. Batch normalization explained. *arXiv preprint arXiv:2209.14778*, 2022.

[63] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Deep networks always grok and here is why. *arXiv preprint arXiv:2402.15555*, 2024.

[64] Thomas Walker, Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Grokalign: Geometric characterisation and acceleration of grokking. *arXiv preprint arXiv:2506.12284*, 2025.

[65] Manjot Singh, Adalbert Fono, and Gitta Kutyniok. Expressivity of spiking neural networks through the spike response model. In *UniReps: the First Workshop on Unifying Representations in Neural Models*, 2023.

[66] Manjot Singh, Adalbert Fono, and Gitta Kutyniok. Are spiking neural networks more expressive than artificial neural networks?, 2024.

[67] William F Podlaski and Christian K Machens. Storing overlapping associative memories on latent manifolds in low-rank spiking networks. In *NeurIPS 2024 Workshop on Symmetry and Geometry in Neural Representations*.

[68] William F Podlaski and Christian K Machens. Approximating nonlinear functions with latent boundaries in low-rank excitatory-inhibitory spiking networks. *Neural Computation*, 36(5), 2024.

[69] Yihui Cui, Ilya Prokin, Alexandre Mendes, Hugues Berry, and Laurent Venance. Robustness of STDP to spike timing jitter. *Scientific Reports*, 8(1), 2018.

[70] Karl Lindblad and Axel Nilsson. Adversarial robustness of STDP-trained spiking neural networks, 2023.

[71] Marissa Dominijanni, Alexander Ororbia, and Kenneth W Regan. Extending spike-timing dependent plasticity to learning synaptic delays. *arXiv preprint arXiv:2506.14984*, 2025.

[72] Julian Rossbroich, Julia Gygax, and Friedemann Zenke. Fluctuation-driven initialization for spiking neural network training. *Neuromorphic Computing and Engineering*, 2(4), 2022.

[73] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.

[74] Yufei Guo, Yuhan Zhang, Yuanpei Chen, Weihang Peng, Xiaode Liu, Liwen Zhang, Xuhui Huang, and Zhe Ma. Membrane potential batch normalization for spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[75] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[76] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[77] Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in Neuroscience*, 15, 2021.

[78] Chaoteng Duan, Jianhao Ding, Shiyan Chen, Zhaofei Yu, and Tiejun Huang. Temporal effective batch normalization in spiking neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 35, 2022.

[79] Haiyan Jiang, Vincent Zoonekynd, Giulia De Masi, Bin Gu, and Huan Xiong. TAB: Temporal accumulated batch normalization in spiking neural networks. In *International Conference on Learning Representations (ICLR)*, 2024.

[80] Yufei Guo, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Yinglei Wang, Xuhui Huang, and Zhe Ma. IM-Loss: information maximization loss for spiking neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 35, 2022.

Appendix A. Additional figures

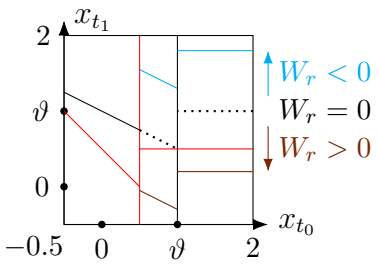

Figure 4: **Recurrent weights can only shift partition boundaries.** We examine two neurons in a two-timestep input plane. If the recurrent weight between them $W_r = 0$, their partitions are a simply two variants of Figure 1 superimposed. Neuron 0 (red) has a lower threshold and slower decay ($\vartheta = 0.5, \lambda = 1$) than neuron 1 ($\vartheta = 1, \lambda = 0.5$), so it always spikes first if $W_r = 0$. However, if $W_r \neq 0$, then a spike from neuron 0 at time t_0 will shift the threshold for a neuron 1 spike at time t_1 by W_r, thus shifting some region boundaries of neuron 1.

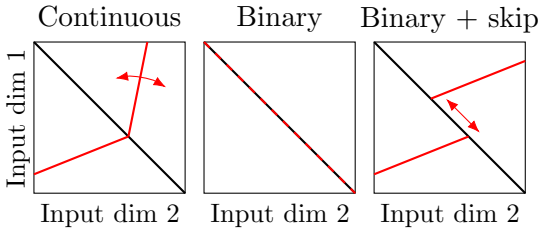

Figure 5: **Continuous networks rotate partition boundaries, binary networks can only shift them.** Neuron 0 (black) feeds neuron 1 (red). The latter draws new rotated boundaries if neuron 0 is continuous or new shifted boundaries if it observes inputs via skip connections and neuron 0 is binary.

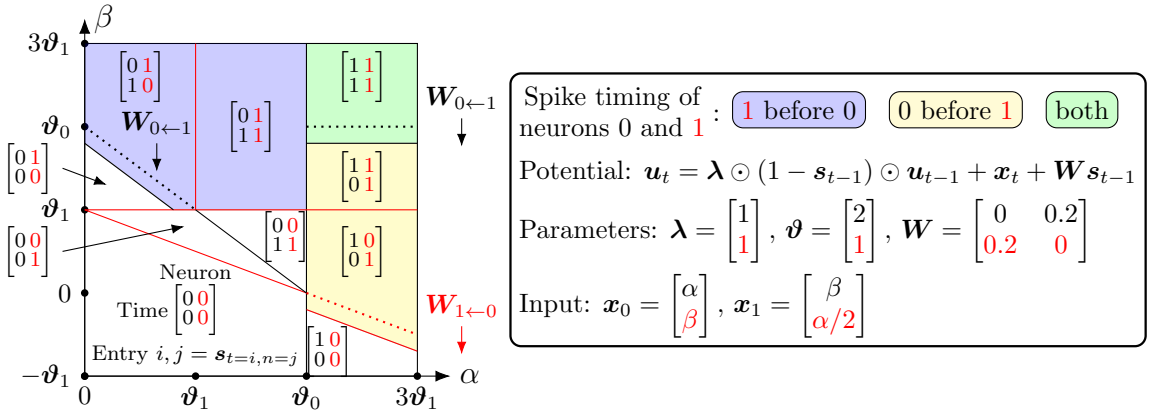

Figure 6: **STDP shifts partition boundaries based on where data is concentrated.** Like in Figure 4, we have two neurons, connected via excitatory recurrent weights $\boldsymbol{W}$, stimulated at two timesteps. However, now both dimensions of our input space affect both timesteps: α controls how much neuron 0 (black) is stimulated before neuron 1 (red), while β does the opposite. This projection shows how STDP affects partitions: if training sequences largely satisfy $\beta > \alpha$, then neuron 1 frequently spikes before neuron 0, and the ensuing changes to recurrent weights would expand regions where neuron 1 spikes before neuron 0. The opposite would happen if training sequences largely satisfied $\alpha > \beta$.

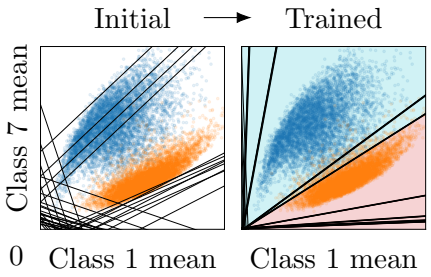

Figure 7: **Training aligns partitions around data.** We use SplineCam-SNN on a deep (15 hidden + 1 output neuron, all spiking) SNN randomly initialized and then trained (via gradient descent) to classify repeated of MNIST digits over 3 timesteps. Partition geometry aligns itself to data points (blue and orange are classes 7 and 1, respectively) so that the final layer can assign classification decisions (instead of drawing new partitions, since there are no skip connections).

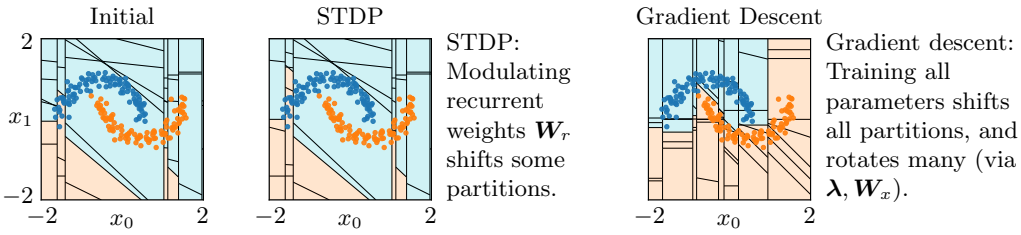

Figure 8: **Gradient descent is better at orienting partitions around data than STDP.** We use SplineCam-SNN to compare gradient descent and STDP in a deep SNN classifying a 2-timestep toy dataset. Gradient descent aligns boundaries around data, while STDP only shifts some boundaries (here, horizontal and diagonal ones). Colors denote true class labels (of data points) and the network's classification decisions (of input regions).

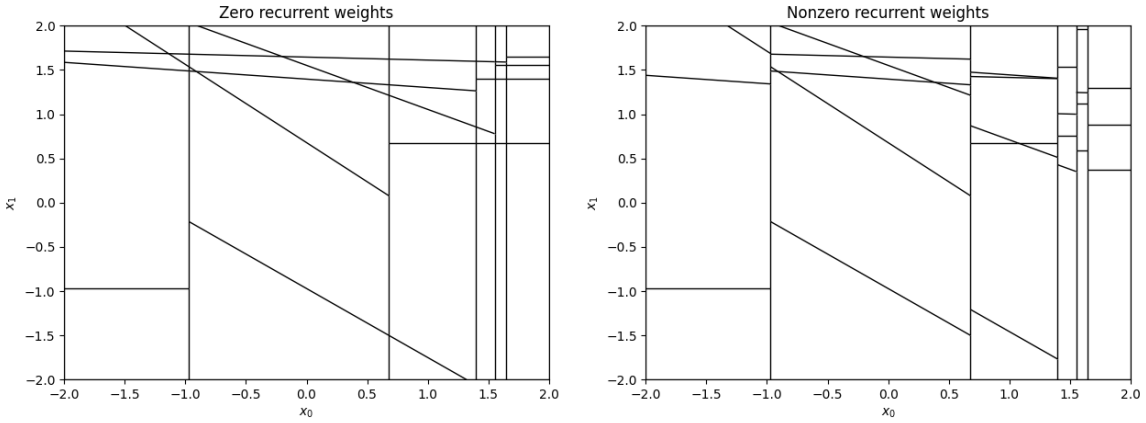

Figure 9: **Recurrent weights shift, but do not rotate, partition boundaries.** Above are two observations of a two-timestep five-neuron SNN without (left) and with (right) recurrent weights. In the former case, the partitions simply correspond to shifted or flipped versions of the diagram established in Figure 1. This is because the activity of the SNN here without recurrent weights is simply the simultaneous activity of five different two-timestep single-input SNNs. However, once we introduce recurrent weights (sampled from a normal distribution), we can see that the nature of the partitions changes.

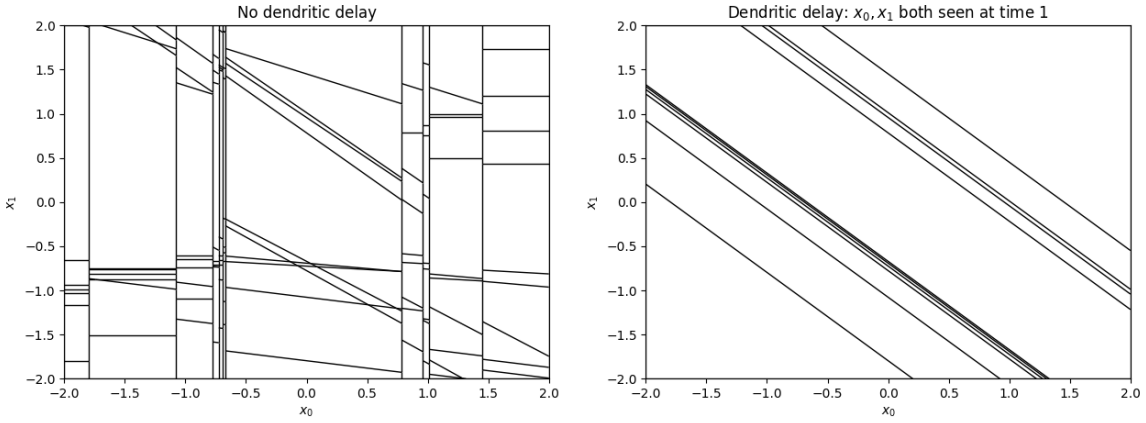

Figure 10: **Dendritic delays facilitate coincidence processing.** Here we have an SNN of 10 hidden neurons that again observes two inputs, x_0 and x_1, sequentially. Without dendritic delays, the SNN evidently exhibits complex behavior with respect to the choices of x_0 and x_1. However, with dendritic delays, the SNN views x_0 and x_1 both at timestep 1, and is thus much better able to fire in a response that depends equally on x_0 and x_1. Since the SNN has a static input weight W_x that is shared between x_0 and x_1, all lines in the case with dendritic delay have the same slope, but if the SNN had weighted delays (i.e., different weights for x_0 and x_1), then the slopes would vary, which would be good for plasticity.

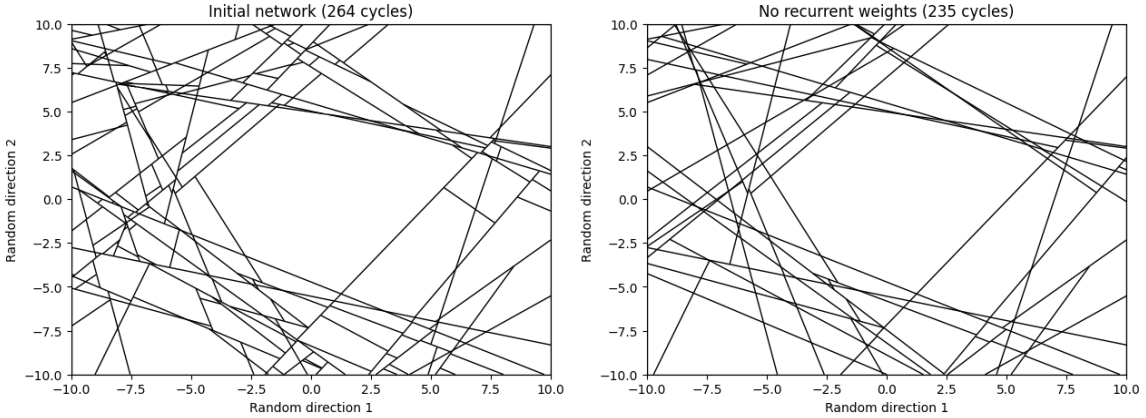

Figure 11: **Initial linear partitions are largely independent of recurrent weights.** Unlike all previous plots, here we observe a 10-timestep 5-neuron SNN that observes a 5-dimensional input at each timestep. We take a 2D random Gaussian slice of the $5 \times 10 = 50$-dimensional input space of the SNN, and observe what happens when we remove the recurrent weights from the SNN. Surprisingly, the overall shape of the partitions remains largely intact, suggesting that the boundaries of linear behavior in SNNs are strongly dependent on parameters like input weight matrices and leakage decay rates.

Appendix B. SplineCam-SNN Overview

Goal. The fundamental goal of our algorithm is to take a linear 2D projection from the input space of a SNN and calculate the boundaries of linear behavior therein, in a similar fashion as previous works have done for feedforward networks (21; 23) and linear RNNs (22). The primary challenge in doing so is that in SNNs, outputs are discontinuous, so the partition boundaries of SNN behavior are also discontinuous—all previous methods have assumed partition boundaries are continuous (since ReLUs are continuous). All code is available at https://colab.research.google.com/drive/1kBOu9ho2GD-2uegj9G5Dg63k7G2o3ZWw and https://colab.research.google.com/drive/1E0B7-y9ZgyQ7IqJsrtrZnTveWDwaOuwb?usp=sharing.

Conceptual overview. Like previous works, our algorithm begins with a single rectangular area comprised of four 2D vertices, and then progressively subdivides this area according to lines where a single piecewise-linear neuron transitions between two modes of linear behavior. However, due to the limitations of previous works for SNNs, our algorithm uses a new method to compute these lines in a given linear region. While a recent algorithm (23) showed that parallel evaluation of all vertices is massively beneficial for runtime complexity, the discontinuity of SNN activations forces us to perform calculations separately on each linear region, as done in SplineCam (21; 22). However, SplineCam uses slow CPU graph cycle discovery operations to divide any given linear region, so we avoid these graph operations by keeping track of the cycles, or regions, of our overall graph as we are mutating it. Our algorithm conceptually lies between SplineCam and a recently proposed alternative (23). Starting from the restriction that the input time series to our given SNN is a linear function of 2D coordinates, our algorithm conceptually works as follows:

1. Our list of regions Ω starts with a single rectangular linear region of 4 vertices, $\boldsymbol{V}$.
2. For each timestep $t \in [0, 1, \ldots, T-1]$:
   (a) For each neuron $n \in [0, 1, \ldots, N-1]$:
   (b) $\Omega_{old} \leftarrow \Omega$ before adding any new regions from neuron n at time t.
      i. For each linear region $\omega \in \Omega_{old}$:
         A. Compute $\boldsymbol{W}_{t,n,\omega} \in \mathbb{R}^2, b_{t,n,\omega} \in \mathbb{R}$, the affine parameters for the n-th hidden state at time t from any 2D point within ω.
         B. Using $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$, find or calculate the two 2D vertices along the boundary of ω where the n-th hidden state at time t equals the threshold voltage ($\boldsymbol{u}_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0$).
         C. Append any newly created vertices to $\boldsymbol{V}$.
         D. Halve ω along the line formed by connecting the two 2D vertices where $\boldsymbol{u}_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0$. Both halves share the new line. $\omega \leftarrow$ one half of ω, $\Omega \leftarrow \Omega +$ the other half of ω.

For a detailed explanation of the algorithm, see Appendix C.

Appendix C. SplineCam-SNN Details

C.1. Calculating the Affine Parameters of a SNN

Goal. Here we compute the affine parameters of the hidden state of neuron n at time t as a function of any 2D vertex within a given region ω. First, before any other part of our algorithm can run, we must calculate the contribution from any 2D vertex to all hidden states at all times, in the subthreshold regime. This is necessary because each 2D vertex represents a different T-timestep D-dimensional time series, which is fed into the SNN sequentially, but the reset mechanism of SNNs directly modulates how the hidden state at time t depends on the inputs at previous timesteps.

Impulse responses. We begin by calculating the impulse response of each neuron's hidden state, which are simply exponential functions of time with decay rates $\boldsymbol{\lambda}$ (since neurons do not communicate with each other unless they spike):

$$h_n(t) = \boldsymbol{\lambda}_n^t \;\; \forall \;\; t \in [0, \ldots, T-1], \;\; n \in [0, \ldots, N-1] \tag{2}$$

Contribution of a 2D vertex from time t_{in} to time t_{out}. The hidden state of neuron n at time t_{out} is a linear function of its inputs at times $[0, \ldots, t_{out}]$, but because of the reset mechanism of SNNs, we need to characterize exactly how the n-th state at t_{out} depends on any single prior time t_{in}. We express this dependence for each neuron as a 2D function $\boldsymbol{H}_n(t_{in}, t_{out}) : \mathbb{N}^2 \to \mathbb{R}^2$. $\boldsymbol{H}_n(t_{in}, t_{out})$ returns a vector describing the weight of any 2D vertex on the n-th neuron at time t_{out} from the input at time t_{in} prescribed by the 2D vertex.

Characterizing $\boldsymbol{H}_n(t_{in}, t_{out})$. At any time t_{out}, contribution to the n-th hidden state from an input at time t_{in} is necessarily proportional to $h_n(t_{out} - t_{in})$. However, the input at time t_{in} depends on the input time series projection matrix $\boldsymbol{P} \in \mathbb{R}^{T \times D \times 2}$—specifically, its entry $\boldsymbol{P}_{t_{in}} \in \mathbb{R}^{D \times 2}$. The D-dimensional input at time t_{in} gets fed to the neurons via $\boldsymbol{W}_x \in \mathbb{R}^{N \times D}$, so to get the input to neuron n prescribed by a 2D vertex at time t_{in}, one need only multiply be the vertex by $\boldsymbol{W}_{x,n}\boldsymbol{P}_{t_{in}} \in \mathbb{R}^2$. Therefore, the final expression for $\boldsymbol{H}_n(t_{in}, t_{out})$ is:

$$\boldsymbol{H}_n(t_{in}, t_{out}) = h_n(t_{out} - t_{in})\boldsymbol{W}_{x,n}\boldsymbol{P}_{t_{in}} \tag{3}$$

In practice, we precompute $\boldsymbol{H}_n(t_{in}, t_{out})$ as a 4D tensor of shape $T \times T \times D \times 2$ for simplicity.

Computing the affine parameters $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$. We now describe the hidden state of neuron n at time t as a (region-dependent) linear function of any given 2D vertex (without using any autodifferentiation). If we ignored spike resets and recurrent spikes, this would simply be $\sum_{\tau=0}^{t} \boldsymbol{H}_n(\tau, t)$. However, we must account for resets and recurrent spikes. We do so by iterating through the SNN update equations (Equation 1) for all neurons, up to time t; along the way, we update $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$ according to resets and recurrent spikes:

1. Inputs: SNN with parameters $\boldsymbol{W}_r \in \mathbb{R}^{N \times N}, \boldsymbol{W}_x \in \mathbb{R}^{N \times D}, \boldsymbol{\lambda} \in [0,1]^N, \boldsymbol{\vartheta} \in \mathbb{R}^N$; 2D linear projection tensor $P \in \mathbb{R}^{T \times D \times 2}$

2. Initialize hidden states, spikes, and affine parameters: $\boldsymbol{u}_{-1} = \boldsymbol{s}_{-1} = \boldsymbol{0}, \boldsymbol{W}_{t,n,\omega} \leftarrow \boldsymbol{0}, b_{t,n,\omega} \leftarrow 0$

3. For $\tau \in [0, 1, \ldots, t]$:
   (a) $\boldsymbol{u}_\tau = W_x P_\tau \boldsymbol{V} + \boldsymbol{W}_h \boldsymbol{s}_{\tau-1}$ (feedforward and recurrent inputs)
   (b) Incorporate leakage and spike resets
      i. If hard reset: $\boldsymbol{u}_\tau \leftarrow \boldsymbol{u}_\tau + \boldsymbol{\lambda} \odot (1 - \boldsymbol{s}_{\tau-1}) \odot \boldsymbol{u}_{\tau-1}$
      ii. If soft reset: $\boldsymbol{u}_\tau \leftarrow \boldsymbol{u}_\tau + \boldsymbol{\lambda} \odot (\boldsymbol{u}_{\tau-1} - \boldsymbol{\vartheta} \odot \boldsymbol{s}_{\tau-1})$
   (c) $\boldsymbol{s}_\tau = \text{Heaviside}(\boldsymbol{u}_\tau - \boldsymbol{\vartheta})$ (spikes)
   (d) $\boldsymbol{W}_{t,n,\omega} \leftarrow \boldsymbol{W}_{t,n,\omega} + \boldsymbol{H}_n(\tau, t)$ (add the contribution onto time t from inputs at time τ)
   (e) $b_{t,n,\omega} \leftarrow \boldsymbol{\lambda}_n b_{t,n,\omega} + \boldsymbol{W}_{h,n} \boldsymbol{s}_{\tau-1}$ (exponential moving average of recurrent inputs)
   (f) If $\tau < t$ and neuron n spiked ($\boldsymbol{s}_{\tau,n} = 1$), then update $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$:
      i. If hard reset: $\boldsymbol{W}_{t,n,\omega} \leftarrow \boldsymbol{0}, b_{t,n,\omega} \leftarrow 0$
      ii. If soft reset: $b_{t,n,\omega} \leftarrow b_{t,n,\omega} - \boldsymbol{\vartheta}_n$
4. Return $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$.

C.2. Dividing an Edge

To divide an input region ω with affine parameters $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$, we must calculate the line within ω where the n-th neuron has a hidden state equal to $\boldsymbol{\vartheta}_n$ at time t. To do this for a piecewise-linear function, we need only find or calculate the two points along the boundary of ω where $\boldsymbol{u}_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0$. Given an adjacent pair of vertices (i.e., and edge) $(\boldsymbol{V}_1, \boldsymbol{V}_2)$ where $\text{sign}(\boldsymbol{u}_{t,n,\omega} - \boldsymbol{\vartheta}_n)$ changes between -1 and +1, we seek α such that the linear interpolation $\boldsymbol{V}_\alpha = \alpha \boldsymbol{V}_1 + (1 - \alpha)\boldsymbol{V}_2$ satisfies $\text{preact}(\boldsymbol{V}_\alpha) := \boldsymbol{V}_\alpha \cdot \boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0$. The solution for α is in Equation 8.

$$\boldsymbol{V}_\alpha \cdot \boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0 \tag{4}$$

$$(\alpha \boldsymbol{V}_1 + (1 - \alpha)\boldsymbol{V}_2) \cdot \boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0 \tag{5}$$

$$\alpha(\boldsymbol{V}_1 \cdot \boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega} - \boldsymbol{\vartheta}_n) + (1 - \alpha)(\boldsymbol{V}_2 \cdot \boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega} - \boldsymbol{\vartheta}_n) = 0 \tag{6}$$

$$\alpha \cdot \text{preact}_{t,n,\omega}(\boldsymbol{V}_1) + (1 - \alpha) \cdot \text{preact}_{t,n,\omega}(\boldsymbol{V}_2) = 0 \tag{7}$$

$$\alpha = \frac{\text{preact}_{t,n,\omega}(\boldsymbol{V}_2)}{\text{preact}_{t,n,\omega}(\boldsymbol{V}_2) - \text{preact}_{t,n,\omega}(\boldsymbol{V}_1)} \tag{8}$$

Without loss of generality, one could absorb $-\boldsymbol{\vartheta}_n$ into $b_{t,n,\omega}$ to recover similar zero-finding methods used by previous works with ReLU neurons (23; 21; 22).

C.3. Detailed Algorithm

1. Inputs: SNN with parameters $\boldsymbol{W}_r \in \mathbb{R}^{N \times N}, \boldsymbol{W}_x \in \mathbb{R}^{N \times D}, \boldsymbol{\lambda} \in [0, 1]^N, \boldsymbol{\vartheta} \in \mathbb{R}^N$; 2D linear projection tensor $P \in \mathbb{R}^{T \times D \times 2}$
2. Start with an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{4 \times 4}$ connecting 4 vertices $\boldsymbol{V} \in \mathbb{R}^{4 \times 2}$ in a 2D rectangle centered around 0. Ω is a list representing the cycle basis of $\boldsymbol{A}$, i.e., the regions of linear behavior in our 2D input projection space; at the start, it only contains one cycle, the 2D rectangle.

3. For each timestep $t \in [0, 1, \ldots, T-1]$: For each neuron $n \in [0, 1, \ldots, N-1]$:
   (a) Compute the center (mean vertex) of each linear region (cycle): $\boldsymbol{V}_c \in \mathbb{R}^{|\Omega| \times 2}$
   (b) Calculate $\{\boldsymbol{W}_{t,n,\omega}\}_{\omega \in \Omega}, \{b_{t,n,\omega}\}_{\omega \in \Omega}$ around each $\boldsymbol{V}_c$, the affine parameters of each linear region ω for the n-th hidden state at time t as a function of any 2D vertex within ω. Each $\boldsymbol{W}_{t,n,\omega}, b_{t,n,\omega}$ are 2D and 1D vectors, respectively.
   (c) $\Omega_{old} \leftarrow \Omega$ before adding any new regions from neuron n at time t
   (d) For each region (cycle) $\omega \in \Omega_{old}$:
      i. $\boldsymbol{V}_\omega$ = all 2D vertices in $\boldsymbol{V}$ that form the boundary of ω
      ii. $\boldsymbol{u}_{t,n,\omega} = \boldsymbol{V}_\omega \boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega}$
      iii. If $-1, +1$ are not both in $\boldsymbol{q}_{t,n,\omega} = \text{sign}(\boldsymbol{u}_{t,n,\omega} - \boldsymbol{\vartheta}_n)$:
         A. continue
      iv. $\boldsymbol{V}_{new\_edge}$ = any vertices in $\boldsymbol{V}_\omega$ where $\boldsymbol{q}_{t,n,\omega} = 0$
      v. For each adjacent vertex pair $(\boldsymbol{V}_1, \boldsymbol{V}_2)$ where $\boldsymbol{q}_{t,n,\omega}$ changes between -1 and $+1$:
         A. Calculate α such that $(\alpha \boldsymbol{V}_1 + (1-\alpha)\boldsymbol{V}_2)\boldsymbol{W}_{t,n,\omega} + b_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0$, from Equation 8
         B. Append the new vertex $\boldsymbol{V}_\alpha = (\alpha \boldsymbol{V}_1 + (1-\alpha)\boldsymbol{V}_2)$ to $\boldsymbol{V}$ and to $\boldsymbol{V}_{new\_edge}$.
         C. Update $\boldsymbol{A}$ and Ω: all instances of the edge $(\boldsymbol{V}_1, \boldsymbol{V}_2)$ are replaced by $(\boldsymbol{V}_1, \boldsymbol{V}_\alpha), (\boldsymbol{V}_\alpha, \boldsymbol{V}_2)$.
      vi. There should be exactly 2 vertices in $\boldsymbol{V}_{new\_edge}$. Draw a new edge between them in $\boldsymbol{A}$.
      vii. Halve ω along the new edge where $\boldsymbol{u}_{t,n,\omega} - \boldsymbol{\vartheta}_n = 0$. Both halves share the new line. $\omega \leftarrow$ one half of ω, $\Omega \leftarrow \Omega +$ the other half of ω.
4. Return vertices $\boldsymbol{V}$, adjacency matrix $\boldsymbol{A}$, and linear regions (cycle basis of $\boldsymbol{A}$) Ω.