Training Spiking Neural Networks Using Lessons from State Space Models

Andrew Smith[(1)], Taylor Kergan[(1)], Assel Kembay[(1)], Kimia Gholami[(1)], Vincent Harkins[(1)], Binh Nguyen[(1)], Ruhai Lin[(1)], Ridger Zhu[(1)], and Jason K. Eshraghian[(1)]

[(1)] University of California, Santa Cruz

Anyone who has trained a spiking neural network (SNN) knows they can be difficult to work with: discrete spike events, surrogate gradients, subtle mismatch with deployment hardware, and the list goes on. A more fundamental issue lurks underneath all of this: most SNNs are still trained by unrolling their dynamics *sequentially* in time. As sequence length grows, wall-clock time grows with it. This makes training increasingly expensive and limits the ability of SNNs to scale to the long-context workloads now common in modern deep learning.

In this paper, we show how to overcome that limitation by accelerating the core SNN primitives themselves. We work around the sequential nature of SNNs by borrowing ideas from state space models (SSMs), which re-express recurrent computations as parallel convolutions or scans over sequence length. We first outline a practical taxonomy of recurrence-parallelization strategies relevant to SNNs and SSMs (iterative rollout, convolution, prefix scan, and general associative scans) and clarify which are directly compatible with PyTorch and SNNTORCH.

Guided by this taxonomy, we introduce two new spiking neuron models in SNNTORCH. *StateLeaky* is a Generation 1 spiking SSM obtained by relaxing the reset nonlinearity in Leaky/LIF dynamics during training, yielding a diagonal linear recurrence that can be evaluated by convolution over the full sequence. While the convolutional formulation increases activation memory, simple batch-wise chunking with gradient accumulation substantially mitigates this overhead. *AssociativeLeaky* is a Generation 2 spiking SSM with a matrix-valued hidden state, input-dependent decay, and rank-1 key-value writes, enabling prefix-scan evaluation and associative-memory dynamics. Across systems benchmarks, both models improve training-time scaling with sequence length relative to standard sequential Leaky dynamics. On TinyStories language modeling, *StateLeaky* reaches lower validation perplexity sooner in wall-clock time than the sequential baseline, while *AssociativeLeaky* improves sequence modeling in the unit-matched regime and decisively outperforms diagonal recurrences on Neural Turing Machine-style associative recall. Throughout, we keep the focus on practical use: both constructions are implemented in SNNTORCH and remain compatible with iterative, event-driven inference at deployment.

Table 1: Tools for parallelizing recurrent/state-space updates in PyTorch/SNNTORCH.

Tool	**Concept**	**Best for**	**snnTorch support**
Iterative rollout	Sequential recurrence: $x_{t+1} = f(x_t, u_t)$	Exact dynamics; neuromorphic deployment	✓ Baseline
Convolution	Time-invariant linear filter: $y = h * u$	Fast parallel training with fixed decay	✓ `StateLeaky`
Prefix scan	Associative multiply–add: $x_t = \mathrm{diag}(\alpha_t)x_{t-1} + u_t$	Input-dependent or time-varying decay	✓ `AssociativeLeaky`
Associative scan	General associative operator: $(x,u) \oplus (x',u')$	Nonlinear recurrences	× Needs JAX/Triton