Training

ar006 · 14 May 2026

This is the shared training recipe every model on the ladder uses. It runs, in order, through how gradients flow through time, the surrogate that lets them pass through spikes, the one flag that stops them exploding (its own article, Gradient Stabilisation), the loss and optimiser, the readout options, the firing-rate regulariser, weight initialisation, and the tasks the networks are trained on.

Backpropagation through time

Every model here is a recurrent system run forward in time, so gradients come from Backpropagation Through Time (BPTT): unroll the recurrence into a deep feedforward graph — one layer per timestep, all sharing the same weights — and backpropagate through it.

Take a hidden state h^t that evolves as

$$h^t = f(h^{t-1}, x^t; \theta), \quad y^t = g(h^t; \theta)$$

with input x^t, output y^t, and parameters θ shared across time. Running T steps gives a chain $h^0 \to h^1 \to \cdots \to h^T$, which for gradients we treat as a depth-T feedforward network with tied weights.

The gradient with respect to a parameter θ_k then sums over every timestep it touched:

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial h^t} \cdot \frac{\partial h^t}{\partial \theta_k}$$

The catch is the product of per-step Jacobians $\partial h^{t+1}/\partial h^t$ running through the chain: if their norms sit above 1 the product explodes in T, if below 1 it vanishes.

SNNs fit BPTT naturally — one simulation step is one step of the recursion, with the hidden state holding membrane potentials, synaptic conductances, and refractory counters. A 200 ms trial at $\Delta t = 0.1$ ms unrolls to $T = 2000$ steps. Because the state variables carry physical units (mV, µS), the per-step Jacobians are wildly scaled: voltage updates carry tiny factors like $\Delta t / C_m$ while surrogate gradients through spikes are $O(1)$. That mismatch is exactly what the gradient-stabilisation flag exists to fix — derived in full in Gradient Stabilisation.

Surrogate gradients

The spike function $S = \mathbf{1}[U \geq \theta]$ has zero gradient almost everywhere, so the backward pass substitutes a smooth surrogate. Pinglab uses the fast-sigmoid surrogate everywhere. Forward is the hard step; backward is

$$\frac{\partial \tilde{S}}{\partial U} = \frac{k}{(1 + k\,|U - \theta|)^2}$$

This matches snntorch's *FastSigmoid*, so equal-k comparisons against the snntorch reference test the update rule, not the surrogate.

It takes its slope from *SURROGATE_SLOPE = 5.0*, overridable per-run with *–surrogate-slope*.

Gradient stabilisation

Conductance-based networks (COBA, PING) need one extra ingredient to train: the recurrent E↔I loop makes the backpropagated gradient explode during BPTT, and a single flag, `--v-grad-dampen`, tames it. The full derivation — why the gradient diverges once per gamma cycle, and why per-step voltage damping fixes it without touching the forward pass — has its own article: **Gradient Stabilisation**.

The training loop

Logits from the readout go into cross-entropy loss:

$$L_{\text{CE}} = -\frac{1}{B}\sum_{b=1}^{B} \log \frac{\exp\left(\hat{y}_{b,c_b}\right)}{\sum_k \exp\left(\hat{y}_{b,k}\right)}$$

with batch size B, logit vector $\hat{y}_b$, and true class c_b; chance-level loss on a 10-class problem is $\ln 10 \approx 2.30$. The optimiser is Adam, with gradients clipped to unit norm (`GRAD_CLIP = 1.0`) before each step. The saved `weights.pth` is the *best-epoch* state by test accuracy, not the final epoch.

Readout

The readout collapses the last hidden layer's activity into class logits; `--readout` picks how. Four modes:

- `spike-count` — sum last-hidden spikes over the trial and project linearly, $\hat{y} = \left(\sum_t s_t^{\text{hid}}\right) W_{\text{out}} + b_{\text{out}}$. Equivalent to spike-rate up to a constant.
- `mem-mean` — pass spikes through a final non-resetting LIF and average its membrane potential over the trial. Default for the COBA/PING recipes; used by exp025 and the streaming entries.
- `li` — leaky integrator: a non-spiking LIF whose final-step membrane potential is the logit.
- `rate` — softmax over per-trial spike rates.

The choice matters because it sets where the gradient enters the network: `mem-mean` lets it flow through the output LIF's membrane at every timestep, while `spike-count` only sees the aggregate.

Firing-rate regularisation

Many recipes penalise too much or too little hidden firing via `--fr-reg-upper-theta`, `--fr-reg-upper-strength`, and the matching `lower` pair:

$$\mathcal{L}_{\text{fr}} = s_u \cdot \text{ReLU}(\bar{r} - \theta_u) + s_l \cdot \text{ReLU}(\theta_l - \bar{r})$$

where $\bar{r}$ is the per-layer mean firing rate (per-neuron or population, set by `--fr-reg-mode`). This is the mechanism behind the θ_u sweeps in exp025 and the rate-floor framing in ar009.

Weight init

Feedforward weights are sampled fan-in-normalised, either half-normal (Dale's law) or normal (signed):

$$W \sim \mathcal{N}(\mu, \sigma^2), \quad W \leftarrow W / N_{\text{pre}}$$

with optional sparsity $s \in [0, 1)$: a fraction s of entries are zeroed and the survivors rescaled by $1/(1-s)$, so the expected synaptic input per post-neuron is preserved.

Dale's law under Adam

When Dale's law is on, the feedforward matrices W_{ff} are clamped to $W \geq 0$ when they are read by the forward pass and every trainable constrained matrix is projected back into the non-negative cone by `project_dales()` after each optimiser step. The recurrent conductance matrices W_{ee}, W_{ei}, W_{ie}, and W_{ii} are not forward-clamped: they are initialised non-negative and, when trainable, kept non-negative by the post-step projection. Their entries are conductance magnitudes; pathway-specific reversal potentials, rather than a negative stored W_{ie}, determine whether a synapse is excitatory or inhibitory.