K-prop: Deep Online Learning by Backpropagating Temporal Kernels

Bingkun Liu*, Paul Haider, Federico Benitez, Mihai A. Petrovici, Walter Senn

Department of Physiology
University of Bern
Bern, Switzerland

Abstract

Self-recurrence represents an important component for temporal processing in neural networks, both artificial and biological, where it appears naturally in the form of leaky integration. However, due to the complex temporal dependencies of the gradient, an efficient and effective method of online training is still lacking. Here we propose K-prop, a novel online learning method for leaky neural networks. It dramatically reduces the computational and memory cost of real time recurrent learning (RTRL) by propagating kernel gains. K-prop also outperforms existing RTRL approximations and achieves performance comparable to backpropagation through time in sequence learning, classification, and reinforcement learning tasks. We further analyze how its sole approximation affects gradient quality and identify conditions under which K-prop remains accurate and effective.

1 Introduction

Temporal processing is fundamental to both biological and artificial intelligence, requiring systems to transform continuous inputs into structured internal representations. Effective training for temporal dynamics relies on accurate credit assignment over time, commonly addressed by backpropagation through time (BPTT, Werbos [2002]). However, BPTT requires storing and replaying full trajectories, limiting its applicability to online settings. In contrast, many natural and real-world systems operate continuously, where learning must occur in real time without access to stored past states, motivating the need for efficient online learning mechanisms. Prior research [Irie et al., 2024] suggests that online learning can also improve sample efficiency in reinforcement learning settings.

Artificial neural networks typically employ instantaneous neurons, whose activations depend only on the current input. In contrast, *leaky (integrating) neurons* evolve according to continuous dynamics of the type $\tau\dot{u} = -u + Wr$, where the membrane potential u decays with time constant τ while integrating presynaptic input r through weights W. These dynamics emulate biological membrane potentials and have therefore been widely adopted in computational neuroscience [Dayan and Abbott, 2005]. Beyond biological inspiration, leaky neurons offer an intrinsic mechanism for temporal processing: each neuron implements a low-pass filter, and networks of such units naturally construct rich temporal kernels in an *online* manner. Unlike Temporal Convolutional Networks (TCN, Bai et al. [2018]), which require access to full input sequences, leaky networks have the potential to process streaming data continuously while maintaining a distributed, multi-timescale memory, a power that has recently been demonstrated in models [Spieler et al., 2024, Ebato et al., 2024]. Moreover, their dynamics align naturally with analog hardware, where voltage-based circuits exhibit leaky integration [Stoliar et al., 2017, Yang et al., 2020].

*bingkun.liu@unibe.ch

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Despite these advantages, online training of feedforward leaky networks remains fundamentally challenging. Due to temporal integration, activity at each layer influences future downstream responses, making the gradient depend jointly on spatial pathways and temporal dynamics, analogous to recurrent networks. In principle, exact gradients can be computed by BPTT or real-time recurrent learning (RTRL, Williams and Zipser [1989]). RTRL, while online, incurs prohibitive computational complexity of order $\mathcal{O}(n^4)$, rendering it impractical beyond very small networks. Existing RTRL approximations either reduce complexity at the expense of gradient quality or fail to scale to deep architectures. As a result, investigating online learning in large and deep leaky networks remains challenging.

In this work, we introduce *Kernel propagation* (K-prop), a novel learning algorithm for feedforward leaky networks that achieves efficient, online, and accurate gradient approximation. The central idea is to express gradient contributions as a decomposition over exponential temporal kernels, and to propagate only their gain coefficients backward through the network. This transforms the combinatorial explosion of possible temporal pathways into a tractable, layer-wise computation analogous to standard backpropagation.

K-prop offers the following key contributions:
Efficiency and effectiveness: K-prop significantly reduces the quartic complexity of RTRL to near-quadratic scaling, while maintaining a close approximation to the exact gradient. It achieves best-in-class task performance, close to the exact RTRL/BPTT solutions, in deep architectures.
Continuous formulation: Derived directly in continuous time, K-prop is robust to arbitrarily small time steps and naturally suited for analog systems and for dealing with continuous or sparse data.
Layer-wise locality: K-prop remains layer-wise local during training, facilitating practical implementation without requiring global information pathways.

2 Related Work

A large body of work has focused on approximations to RTRL. Truncated BPTT [Williams and Zipser, 2013] reduces memory by limiting the temporal horizon, but introduces bias toward short-term dependencies. Methods that are tailored to standard recurrent networks with dense global recurrence do not align well with leaky neurons, whose dominant temporal effect arises from strong *self-recurrence* (diagonal dynamics). For example, low-rank approximations such as unbiased online recurrent optimization (UORO, Tallec and Ollivier [2018]) fail to capture the inherently high-rank diagonal structure, while Kronecker-based methods [Mujika et al., 2018, Benzing et al., 2019] do not exploit the naturally separable temporal kernels. Moreover, these approaches are formulated in discrete time and are not directly suited to continuous-time or analog dynamics.

Alternatively, methods such as random feedback local online learning (RFLO, Murray [2019]) and e-prop [Bellec et al., 2020] are specifically designed for leaky neurons and emphasize biological plausibility. These methods rely on local eligibility traces but discard inter-neuron dependencies, which prevents effective credit assignment in deep layers. Zucchet et al. [2023] derived online learning for complex-valued leaky-like memory, and is also limited to a single memory layer. Online spatio-temporal learning (OSTL, Bohnstingl et al. [2022]), partially addresses this by propagating instantaneous errors to deep layers whose temporal dependencies are still missing. Following an energy-based approach, generalized latent equilibrium (GLE, Ellenberger et al. [2025]) assumes a sufficiently smooth error signal ϵ, so that the prospective error $\breve{\epsilon} = \epsilon + \tau\dot{\epsilon}$ can be used to estimate temporal gradients. Beyond previous online methods, K-prop preserves the temporal structure of the deep layer gradient by deploying a novel decompositions of both temporal kernels and spatial paths.

3 Theory

3.1 K-prop: Propagating kernel gains

We consider a feedforward leaky neural network with nonlinear activation $r = \rho(u)$. From the output loss function $\mathcal{C}(t)$, gradients propagate backward to each hidden neuron through the downstream network along all possible directed paths (as shown in Fig. 1a). The gradient of each path constains a *temporal term* accounting for its temporal contribution. For a postsynaptic neuron i in the last leaky layer L, this temporal term is represented by a vector of synaptic eligibility traces $\boldsymbol{z}_i = \rho_i' \overline{\boldsymbol{r}}^{\tau_i}_{[L-1]}$,

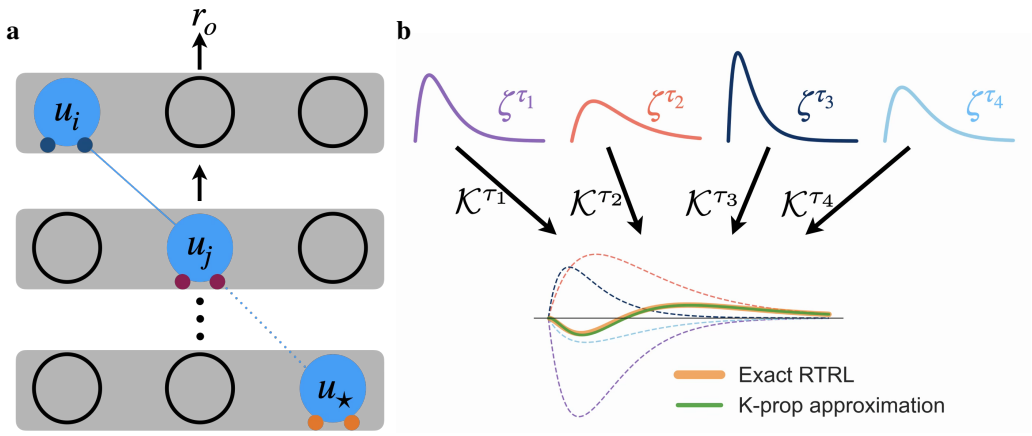

Figure 1: Kernel propagation. (a) Calculating gradients involves hierarchical filters along paths of neuron connections. (b) Illustration of kernel decomposition. The complicated temporal structure in the true gradient can be approximated by a sum over ζ weighted by τ-dependent coefficients $\mathcal{K}$. Every ζ (solid) is weighted by corresponding $\mathcal{K}$ as a component (dashed in the same color). The sum of all components aligns well with the exact term computed by RTRL.

where $\overline{\boldsymbol{x}}^{\tau} = \kappa^{\tau} * \boldsymbol{x}$ denotes a low-pass filtering of the vector-valued temporal variable $\boldsymbol{x}_t$ with the exponential kernel $\kappa^{\tau} = \Theta(t)\frac{1}{\tau}\exp(-\frac{t}{\tau})$, where Θ is the Heaviside function. Moving one layer upstream to neuron j introduces an additional filtering $\rho_i'(\kappa^{\tau_i} * \boldsymbol{z}_j)$, and an arbitrary neuron $\star$ in deeper layers generates more iterative hierarchical filters $\rho_i'(\kappa^{\tau_i} * \rho_j'(\kappa^{\tau_j} * ... * \boldsymbol{z}_\star))$. Therefore, each path induces a *hierarchical temporal kernel* characterized by the leaky dynamics of all neurons along the path, and the exact gradient requires summing over all combinations of such path-dependent kernels.

K-prop simplifies this involved computation in three steps.

Step 1: Separating nonlinearity from filtering

K-prop approximates ρ' by a surrogate $\hat{\rho}'$ ($= \overline{\rho_\star'}^{\tau_\star}$ as default), such that it can be moved outside of the filter: $\overline{\rho' x}^{\tau} \approx \hat{\rho}' \overline{x}^{\tau}$. We can therefore approximate a hierarchical filter by a nested filter: $\rho_i'(\kappa^{\tau_i} * \rho_j'(\kappa^{\tau_j} * \rho_k'(\kappa^{\tau_k} * ...))) \approx (\rho_i'\hat{\rho}_j'\hat{\rho}_k'...)(\kappa^{\tau_i} * \kappa^{\tau_j} * \kappa^{\tau_k} * ...)$. The assumption that ρ' varies slowly relative to the filtering time scale τ is the only approximation required by K-prop.

Step 2: Collapsing nested filters into basis kernels

Lemma 1: Nested exponential filters can be rewritten as a linear combination of single filters. Υ is a set of N mutually distinct τ. A proof is given in Appendix A.

$$\kappa^{\Upsilon}(x) = \kappa^{\tau_0} * \kappa^{\tau_1} * \ldots \kappa^{\tau_N} * x \tag{1}$$

$$= \sum_{\tau_i \in \Upsilon} c_i(\kappa^{\tau_i} * x) \text{ with } c_i = \prod_{\tau_j \in \Upsilon, j \neq i} \frac{\tau_i}{\tau_i - \tau_j}. \tag{2}$$

In this work, we treat identical τ across layers as slightly perturbed, which yields an equivalent temporal horizon while avoiding singularities in Eq. (2). Consequently, each τ appears once in a path. Applying the lemma, each *path-specific nested kernel* is converted into a *linear combination of basis kernels*. Together with Step 1, the path gradient can then be approximated as $\frac{\partial \mathcal{C}(t)}{\partial W_\star}\Big|_p \approx w_p \sum_{i \in p \setminus \star} c_i(\kappa^{\tau_i} * \boldsymbol{z}_\star)$, which sums over all neurons i in the path p except the initial $\star$. The term w_p is the **path weight**, a product of all synaptic weights, nonlinear surrogates $\hat{\rho}'$, and the output error along path p (see Eq. (37) in Supplement).

Step 3: Propagating kernel gains through layers

Despite the reduction above, explicitly summing over paths remains intractable given: (1) exponential growth in the number of paths; (2) dependence on detailed network topology; (3) dependence on

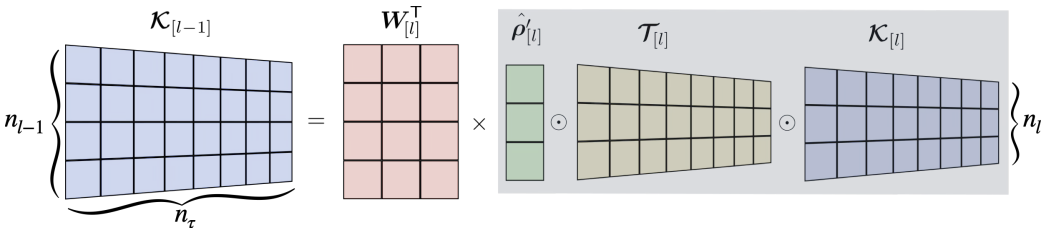

Figure 2: Kernel propagation operation. Example propagation from layer l with 3 neurons to layer $l-1$ with 4 neurons. The gray box works as a bracket.

global downstream variables (weights, activities, errors). The key observation is that the number of distinct time constants is much smaller than the number of paths. Let $\mathcal{P}$ denote the set of all downstream paths and Υ_{down} represent the set of all distinct time constants in downstream neurons. We can rearrange the summation over paths and group coefficients by their shared time constant, $\sum_{p\in\mathcal{P}_\star} w_p \sum_{i\in p} c_i \left(\kappa^{\tau_i} * \boldsymbol{z}_\star\right) = \sum_{\tau\in\Upsilon^\star_{\text{down}}} \mathcal{K}^\tau_\star \left(\kappa^\tau * \boldsymbol{z}_\star\right)$, where $\mathcal{K}^\tau_\star$ is a **kernel gain** that aggregates *all path contributions* associated with time constant τ. This transformation converts the sum over *all combinations of paths and their neurons* to a sum over *a small set of kernels*.

Another central insight of K-prop is that $\mathcal{K}$, which implicitly depends on all downstream paths, satisfies a *layer-wise recursive relation*, i.e., $\mathcal{K}^\tau_{[l-1]}$ can be derived from $\mathcal{K}^\tau_{[l]}$, in a way analogous to backpropagation. For a compact notation, we define Υ_{net} as the set of all distinct τ in the network, and define τ-related transformation factors at neuron j:

$$\mathcal{T}^\tau_j = \begin{cases} 1 & \tau_j = \tau \\ 0 & \tau \text{ in the same layer as neuron } j \text{ and } \tau \neq \tau_j \\ \frac{\tau}{\tau-\tau_j} & \text{otherwise}\,. \end{cases} \tag{3}$$

This definition for $\mathcal{T}$ is based on fully connected feedforward networks, and is further elaborated and extended in Appendix A. With $n_\tau = |\Upsilon_{\text{net}}|$, let $\boldsymbol{\mathcal{K}}_{[l]} \in \mathbb{R}^{n_l \times n_\tau}$ denote the kernel gain at layer l, $\boldsymbol{\mathcal{T}}_{[l]} \in \mathbb{R}^{n_l \times n_\tau}$ the corresponding transformation matrix, and $\hat{\boldsymbol{\rho}}'_{[l]} \in \mathbb{R}^{n_l}$. The external error captured by the loss enters from the output layer as the gain $\boldsymbol{\mathcal{K}}^\tau_o = \frac{\partial \mathcal{C}}{\partial \boldsymbol{r}_o}$, and $\boldsymbol{\mathcal{K}}$ propagates backward according to (Fig. 2)

$$\boldsymbol{\mathcal{K}}^\tau_{[l-1]} = \boldsymbol{W}^{\mathsf{T}}_{[l]}(\hat{\boldsymbol{\rho}}'_{[l]} \odot \boldsymbol{\mathcal{T}}^\tau_{[l]} \odot \boldsymbol{\mathcal{K}}^\tau_{[l]})\,. \tag{4}$$

We consider the eligibility matrix $\boldsymbol{\mathcal{Z}}_{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ for layer l, with the eligibility vector $\boldsymbol{z}^{\mathsf{T}}_{[l],i}$ appearing in the i'th row. With this we define **deep eligibility traces** $\boldsymbol{\zeta}^{\tau_d}_{[l]} = \overline{\boldsymbol{\mathcal{Z}}_{[l]}}^{\tau_d} \in \mathbb{R}^{n_l \times n_{l-1}}$ for all downstream time constants $\tau_d \in \Upsilon^l_{\text{down}}$. Each synapse thus tracks a number of eligibility traces equal to the number of different neuronal time constants that appear in the downstream network. This can be done locally and online through simple leaky integrator dynamics in a matrix form:

$$\tau_d \dot{\boldsymbol{\zeta}}^{\tau_d}_{[l]} = -\boldsymbol{\zeta}^{\tau_d}_{[l]} + \boldsymbol{\mathcal{Z}}_{[l]}\,. \tag{5}$$

Finally, the total gradient sums up the contribution from all kernel components

$$\dot{\boldsymbol{W}}_{[l]} \propto \sum_{\tau\in\Upsilon^l_{\text{down}}} \text{diag}(\boldsymbol{\mathcal{K}}^\tau_{[l]})\, \boldsymbol{\zeta}^\tau_{[l]}\,. \tag{6}$$

K-prop replaces explicit path enumeration by propagating kernel coefficients. Each neuron tracks local temporal features $\boldsymbol{\zeta}$, while $\boldsymbol{\mathcal{K}}$ carries error information across layers (Fig. 1b). This yields a fully online and continuous time learning rule that avoids combinatorial complexity while preserving the structure of the path-based gradient (see pseudocode in Algorithm 1). In order to enable this mechanism, global information is used at initialization to construct $\boldsymbol{\mathcal{T}}$ and Υ_{down}. During training, all computations remain layer-wise local. Consequently, K-prop can be straightforward to implement by integrating feed-forward network techniques, treating $\boldsymbol{\mathcal{K}}$ as the gradient in standard backpropagation. Normalization or dropout simply scales or drops out $\boldsymbol{\mathcal{K}}$, and convolutional or residual architectures only change the matrix operation during propagation.

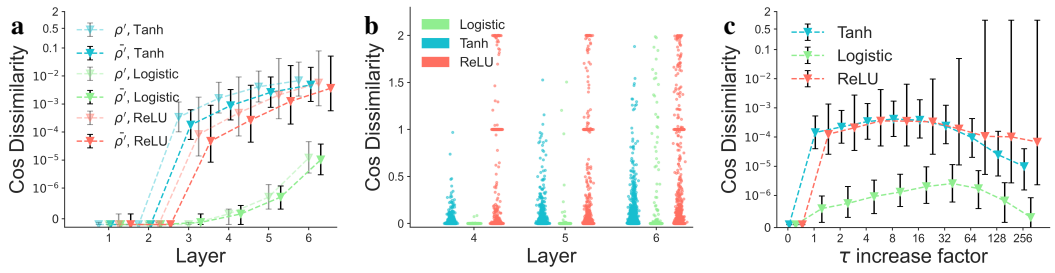

Figure 3: Gradient cosine dissimilarity between K-prop and exact (BPTT/RTRL) gradients in randomly initialized networks, measured with respect to each neuron given a sample of the MNIST1D dataset (see Section 4.2). Dissimilarity ranges from 0 to 2 with 0 indicating complete alignment, 1 orthogonality, and 2 complete anti-alignment. Error bars show median, 1st, and 3rd quartiles. (a) Gradient mismatch increases towards earlier layers, as an increasing number of ρ' need to be approximated as constant. The mismatch depends on the type of nonlinearity present in the network, and $\bar{\rho}'$ works better than ρ' as surrogate $\hat{\rho}'$. (b) Scatter plots show the distribution of outliers. ReLU leads to more outlier neurons than sigmoid nonlinearities. (c) Gradient mismatch increases, then decreases for larger τ. All plots show sigmoid functions performing better than ReLU.

3.2 Quality of the gradient approximation

The accuracy of K-prop is governed by its sole approximation $\frac{\rho'(t'<t)}{\hat{\rho}'(t)} \approx 1$, which assumes that ρ' varies slowly over the relevant timescales determined by the inputs. Deviations from this assumption introduce a mismatch between the true gradient and its K-prop estimate. Understanding how this mismatch depends on network and model properties is key to improving K-prop performance.

We quantify gradient quality using the cosine dissimilarity between K-prop and BPTT gradients:

$$1 - \frac{\Delta W_{\mathrm{KP}} \cdot \Delta W_{\mathrm{BP}}}{\|\Delta W_{\mathrm{KP}}\| \, \|\Delta W_{\mathrm{BP}}\|}.$$

Across configurations, the median dissimilarity remains low ($< 10^{-2}$), indicating that most neurons receive well-aligned gradients.

Depth. As shown in Fig. 3a, gradient dissimilarity increases with distance from the output layer. The last layers are computed exactly, while upstream layers accumulate approximation errors through repeated propagation. This is consistent with the layer-wise structure of K-prop, where small local mismatches compound along depth.

Time constants. The approximation error is directly linked to the timescales of the input (and thus the network) variation. When all $\tau = 0$, K-prop reduces to standard backpropagation and is exact. As τ increases, dissimilarity grows due to longer temporal integration windows. In the limit of very large τ, network dynamics become slower, reducing temporal variation in ρ' and improving the K-prop approximation. Note that this limit only serves as a cross-check for our intuition: while gains ρ' become more constant over time, networks also lose functionality as they become incapable of meaningfully processing incoming signals.

Nonlinearity. The choice of activation function determines the smoothness of ρ'. Functions with slowly varying derivatives (e.g., logistic sigmoid) yield lower dissimilarity. Although $\tanh$ is an offset and scaled logistic function, its derivative has larger magnitude and varies more rapidly, leading to increased mismatch. ReLU presents a distinct regime: ρ' is piecewise constant but discontinuous at zero. As a result, gradients are either highly accurate or highly inaccurate, leading to low median error but significantly higher variance, as reflected by the outlier distribution.

Surrogate derivative. The design of $\hat{\rho}'$ directly controls the approximation. For example, a low-pass-filtered version $\bar{\rho}'^{\tau_\rho}(t)$ improves gradient alignment compared to using the instantaneous $\rho'(t)$, and the τ_ρ used to filter ρ' shifts the gradient alignment tradeoff across network depth (Appendix C.2). Exploring surrogate replacements is a promising direction for further improving K-prop.

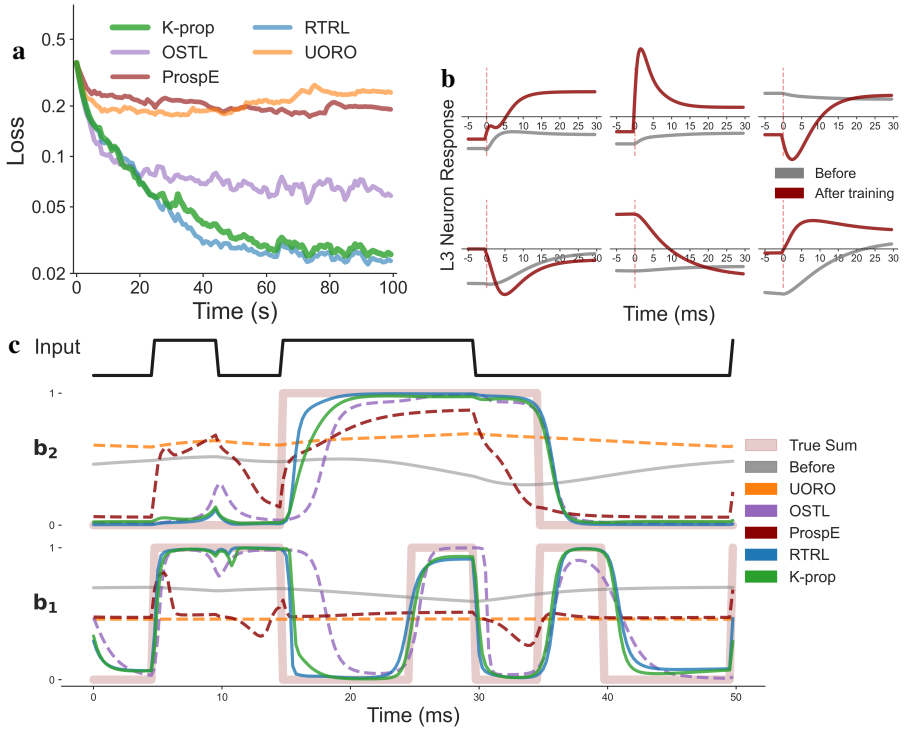

Figure 4: N-sum task evaluated with various online learning methods. ProspE only considers prospective errors $\check{\epsilon}$ in GLE [Ellenberger et al., 2025], without further augmentations. (a) Training loss over time. K-prop achieves substantially lower error than other approximation methods and closely matches RTRL. (b) Temporal response of hidden neurons to a step input before and after learning. K-prop develops more structured and contrastive temporal kernels. (c) Example input and corresponding outputs. Before learning, all models produce non-informative outputs. After learning, K-prop and RTRL closely align with the target sequence, while other methods deviate in both timing and amplitude.

Overall, these results show that gradient quality in K-prop is determined by the interaction between temporal dynamics and nonlinearity, and can be systematically improved through appropriate choices of network timescales, activation functions, and surrogate design.

4 Experiments

4.1 Continuous signal processing with memory

We first evaluate K-prop on a clearly defined non-episodic task, the N-sum problem with $N = 3$. The model receives a continuous binary stream (e.g., [0, 1, 1, 1, 0, ...]) without truncation and must output the sum of the most recent three bits in binary form using two output units ($b_2 b_1 \in \{00, 01, 10, 11\}$). Each bit is presented for 5 ms, requiring the model to maintain a memory of at least $5N = 15$ ms. Note that the time units are, in principle, arbitrary, and what matters is the ratio between time constants and bit duration.

Despite its simple formulation, the task presents two key challenges: (1) the model is not informed of the temporal horizon N or the bit duration, and must infer the relevant memory timescale purely from delayed error signals; (2) the least significant output bit b_1 effectively implements a parity function over recent inputs, which is a class of problems known to be difficult for gradient-based learning [Abbe and Boix-Adsera, 2022, Shalev-Shwartz et al., 2017].

The loss is defined as the mean squared error and is delivered 2 ms after the onset of each input bit (reaction latency). Gradients are computed online at every time step, accumulated into ΔW, and

Table 1: Test accuracy (%) of BPTT, OSTL, and K-Prop across the evaluated datasets. Results are reported as mean ± standard deviation.

Method	MNIST1d-Lk4	MNIST1d-Lk5	GSC
BPTT Werbos [2002]	96.60 ± 0.20	97.69 ± 0.21	92.47 ± 0.21
OSTL Bohnstingl et al. [2022]	96.08 ± 0.30	97.03 ± 0.16	90.61 ± 0.20
K-prop (Ours)	$\mathbf{96.53 \pm 0.54}$	$\mathbf{97.56 \pm 0.19}$	$\mathbf{91.52 \pm 0.10}$

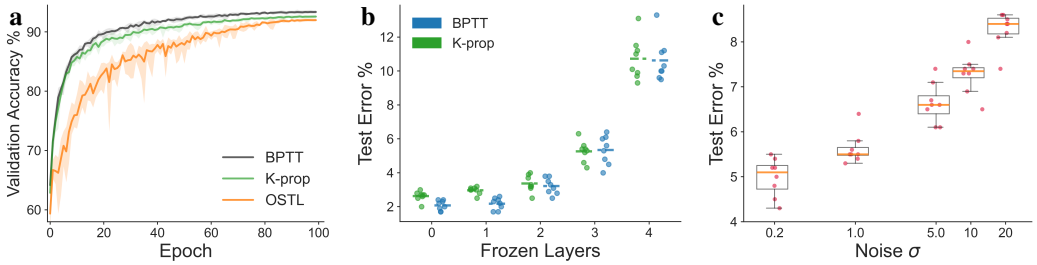

Figure 5: Sequential Classification Benchmark learning. (a) Mean, minimum, and maximum validation accuracy during training on GSC Warden [2018] 20-command task. K-prop outperforms OSTL Bohnstingl et al. [2022] in final accuracy, convergence speed, and training stability. (b) Depth ablation by freezing early layers shows decreasing accuracy with less trained layers. The accuracy loss is similar between K-prop and BPTT, indicating that K-prop provides effective gradients for deep layers. (c) Noise robustness evaluation of the Lk4 in MNIST1d (see **??** and B.6). K-prop maintains high accuracy under a broad range of noise levels σ.

applied periodically, after which the accumulator is reset (see weight update period in Section 5.3 and Supplement Table 3). This update scheme is shared across all methods.

To emphasize learning under constrained capacity, we use a small architecture (6-6-6-2) with logistic output units. In this setting, exact RTRL remains tractable, allowing for a controlled comparison where all methods share identical initialization and learning rates. K-prop achieves performance nearly indistinguishable from RTRL, while significantly outperforming other online approximation methods. The advantage becomes more pronounced on the more difficult parity component b_1, where K-prop better matches the target in both timing and amplitude (Fig. 4). In N-sum, the correct output depends on integrating information over a precise temporal window ($\sim$15 ms) while simultaneously performing a nonlinear transformation (parity-like mapping). This requires assigning credit not only locally in time, but also across combinations of temporal filters.

Beyond output accuracy, we analyze the temporal responses of hidden neurons (layer L3) in Fig. 4. During training, neurons develop diverse temporal kernels that extract task-relevant features. Compared to initialization, the learned responses to step inputs become both more expressive and temporally structured. For instance, neurons exhibiting sign changes around 10 ms, effectively compute differences between recent and earlier inputs, resembling temporal filtering operations. Such structured temporal responses have been widely observed in biological systems, including retina [Trenholm and Krishnaswamy, 2020], visual cortex [Zhaoping, 2014], and auditory system [Lesica and Grothe, 2008, Zhao and Zhaoping, 2011], suggesting convergent strategies for temporal feature extraction.

4.2 Sequential Classification Tasks Benchmark

We next verify K-prop in deeper and wider networks on benchmark sequential classification task MNIST1D [Greydanus and Kobak, 2024]. Prior benchmarks [Beebe et al., 2026, Ellenberger et al., 2025] report accuracies of approximately 94% for temporal convolutional networks, 91% for GRU, and 96% for human performance. For comparable leaky neural networks trained with GLE [Ellenberger et al., 2025], performance reaches 91% with 15K parameters and 93% with 42K parameters. Using K-prop, we achieve $96.5 \pm 0.5\%$ with a 4-layer leaky network (Lk4, 14K parameters) and $97.5 \pm 0.3\%$ with a 5-layer configuration (Lk5, 21K parameters), outperforming

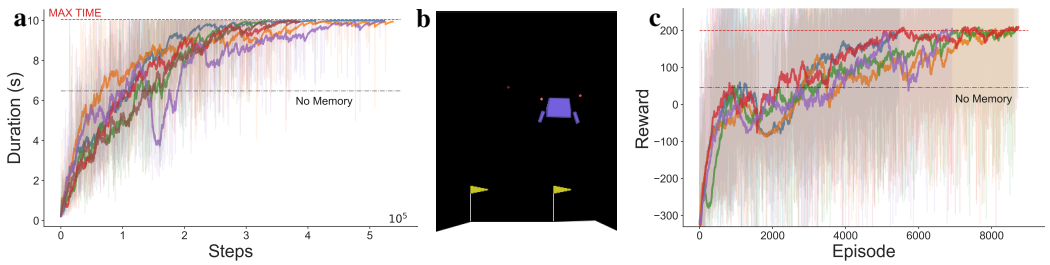

Figure 6: Motor-skill acquisition through online reinforcement learning. Learning curves for (a) CartPole and (c) LunarLander. Faded traces in the background represent individual runs, while solid curves show the corresponding smoothed learning trajectories. Red dashed lines indicate the task-specific success thresholds, and black dashed lines show the performance of an instantaneous, memoryless network. (b) Animation of LunarLander.

previously reported leaky network baselines while remaining parameter-efficient. Moreover, K-prop closely matches the BPTT loss, whereas OSTL converges to a substantially higher loss (Fig. 8).

To evaluate K-prop at a larger scale and on a more realistic dataset, we perform classification on Google Speech Commands (GSC) [Warden, 2018]. This task requires substantially longer temporal dependencies to distinguish acoustically similar words such as 'eight' and 'right', making long-range credit assignment critical. On the 20-command task, K-prop achieves an accuracy comparable to that reported for GLE on the 10-command task, despite using approximately half as many parameters (0.55M, Table 1). K-prop also outperforms OSTL in accuracy, convergence speed, and training stability. These results suggest that K-prop more effectively propagates credit across long temporal dependencies than OSTL instantaneous error signals (Fig. 5a).

To assess whether K-prop effectively trains deep layers, we perform a layer ablation study on Lk5 (Fig. 5b). Progressively freezing early layers degrades performance, indicating that K-prop provides meaningful training signals throughout the network depth. Together, these results demonstrate that K-prop scales to deeper architectures and larger, more realistic sequential-learning tasks, supporting its use as an online alternative for training leaky models.

4.3 Motor learning through real-time reinforcement learning

Motor adaptation in biological systems relies heavily on continuous, online learning, where behavior is adjusted in real time. With online learning, we can adapt faster to new environments and make efficient explorations based on learning. Such scenarios often provide observations that fall into the regime of a partially observable Markov decision process [Kaelbling et al., 1998], requiring agents to construct memory for internal state representations.

To emulate this setting, we use the CartPole and LunarLander tasks from OpenAI Gymnasium [Towers et al., 2024]. We partially mask the observations by removing both positional variables in CartPole and the horizontal position in LunarLander, which makes the instantaneous observation insufficient for optimal control. Although the missing positions can, in principle, be estimated by integrating the observed velocities over time, the agents need to collect historical information correctly without any supervision signal for reconstructing hidden variables. The temporal state inference thus becomes an implicit prerequisite of the reward-maximization task.

We employ an online actor–critic framework similar to [Bellec et al., 2020], using TD(λ) for value estimation. Both actor and critic are implemented as leaky neural networks trained with K-prop. Across multiple runs, the agents consistently succeed in reward maximizing, demonstrating that K-prop supports stable policy and value function learning under partial observability of the system and online updates. Unlike conventional episodic training [Bellec et al., 2020], our setting emphasizes online adaptation during ongoing interaction without batched updates across episodes. In CartPole, in particular, the weights are updated every 3s of interaction. These results show that K-prop provides effective online optimization beyond supervised learning.

5 Discussion

5.1 Computational and memory efficiency

K-prop significantly reduces both memory and computation cost compared to RTRL, whose memory cost scales with the number of neurons n as $\mathcal{O}(n^3)$ and computational cost is $\mathcal{O}(n^4)$. K-prop maintains deep eligibility traces $\boldsymbol{\zeta}$ with size $\mathcal{O}(n_\tau n^2)$. In practice, we divide neurons within each layer into "fast" and "slow" groups (Table 2), leading to $n_\tau = 2n_{\text{layers}}$. Since $n_\tau \ll n$, this yields a substantial reduction in memory requirement.

The computational cost of K-prop consists of two parts: (i) updating local eligibility traces $\boldsymbol{\zeta}$ and (ii) propagating $\boldsymbol{\mathcal{K}}$. Both operations scale as $\mathcal{O}(n_\tau n^2)$ per time step, resulting in an overall complexity of the same order. Except for GLE with an extremely efficient memory footprint of $\mathcal{O}(n)$, approximation methods such as RFLO, e-prop, UORO, and OSTL achieve $\mathcal{O}(n^2)$ complexity in both memory and computation (Fig. 7). K-prop introduces only a modest additional factor n_τ, while providing a more accurate gradient that translates into improved empirical performance.

5.2 Robustness against noise in learning

Because K-prop implements low-pass filters at all dynamical stages, both for neurons and synaptic eligibility traces, it has a built-in protection against temporal noise. This is essential for physical networks, both biological and artificial (neuromorphic), where such noise is near-ubiquitous. We evaluate learning under noise by adding Gaussian noise to $\boldsymbol{\mathcal{Z}}$. K-prop retains an average MNIST1D accuracy of 94.5% at noise variance $\sigma = 1$ and 92.3% at $\sigma = 10$ (Fig. 5), which are strong noise levels relative to $\boldsymbol{\mathcal{Z}}$ (bounded within $[-1, 1]$). These results demonstrate that K-prop remains robust even under substantial perturbations.

5.3 Weight updating during online learning

In practice, we observe that applying weight updates at discrete intervals, rather than continuously at every time step, is important for stable learning. This observation reflects several fundamental challenges in online gradient-based learning, including exact RTRL.

First, online gradients implicitly assume that past contributions are evaluated using the current weights, whereas the weights evolve over time. This mismatch introduces inconsistencies between the accumulated eligibility traces and the current parameter state. Second, continuously updating weights interferes with the forward dynamics: the network is simultaneously required to maintain a stable temporal memory while its parameters are being modified, which can disrupt the memory dynamics that generate useful network output. Third, instantaneous gradients are inherently noisy, as they depend on transient activity and error signals; aggregating gradients over a temporal window can reduce variance, analogous to batch averaging in standard stochastic gradient descent.

6 Conclusion

We introduced K-prop, a novel online learning algorithm for feedforward networks of leaky neurons. K-prop significantly reduces the computational and memory cost of RTRL while maintaining a close approximation to the exact gradient, as evidenced by strong directional alignment. Empirically, K-prop achieves performance comparable to RTRL and BPTT on temporal tasks such as N-sum and MNIST1D, and further demonstrates its applicability in reinforcement learning through continuous control in the Cartpole balancing task. K-prop outperforms other approximation methods, which we attribute to more accurate temporal credit assignment across multiple time scales and layers.

From an engineering perspective, K-prop serves as a practical surrogate for RTRL, enabling the study and deployment of online gradient learning in deeper and wider networks. By avoiding the prohibitive cost of RTRL and the severe performance degradation of prior approximations, which potentially conceals the intrinsic limitations, it allows a clearer assessment of online gradient learning methodological properties. Its layer-wise structure and compatibility with standard architectures and techniques, such as residual connections, convolutions, normalization and drop-out, make it a viable candidate for integrating online temporal learning into modern deep learning pipelines.

From a neuroscience perspective, K-prop provides insight into how eligibility traces may operate in deeper neural circuits. In particular, it suggests a framework in which multiple eligibility traces coexist at each synapse, each selectively interacting with different error-related signals, with reward signals provided by distinct neuro-modulators. Such a mechanism could support more flexible and expressive temporal credit assignment as compared to simple three-factor learning rules Gerstner et al. [2018]. Meanwhile, K-prop can be extended to other neuron models that include, for example, multiple dendritic compartments or adaptive mechanisms by adding structured auxiliary kernels.

Finally, our analysis highlights a key direction for future improvement: the approximation of the activation derivative through $\hat{\rho}'$. The accuracy of K-prop critically depends on the assumption that ρ' varies slowly in time. Relaxing this assumption, especially in regimes with discontinuous or highly non-smooth activations, remains an open challenge. In particular, extending K-prop to spiking neural networks, where ρ' is inherently impulsive, will require more refined surrogate formulations. Addressing this limitation could further broaden the applicability of K-prop to a wider class of temporal and biologically grounded models.

Acknowledgements

We gratefully acknowledge funding from the European Union under grant agreement #101147319 (EBRAINS 2.0). WS and BL acknowledge the Swiss National Science Foundation (SNF) for Project funding (Project #10004971). We would like to express particular gratitude for the ongoing support from the Manfred Stärk Foundation.

Competing interests

The authors declare there are no competing interests.

References

Emmanuel Abbe and Enric Boix-Adsera. On the non-universality of deep learning: quantifying the cost of symmetry. *Advances in Neural Information Processing Systems*, 35:17188–17201, 2022.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Michael Beebe, GodsGift Uzor, Manasa Chepuri, Divya Sree Vemula, and Angel Ayala. Exploring the performance of ml/dl architectures on the mnist-1d dataset. *arXiv preprint arXiv:2602.13348*, 2026.

Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

Frederik Benzing, Marcelo Matheus Gauy, Asier Mujika, Anders Martinsson, and Angelika Steger. Optimal kronecker-sum approximation of real time recurrent learning. In *International Conference on Machine Learning*, pages 604–613. PMLR, 2019.

Thomas Bohnstingl, Stanisław Woźniak, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8894–8908, 2022.

Simon Brandt, Mihai Alexandru Petrovici, Walter Senn, Katharina Anna Wilmes, and Federico Benitez. Prospective and retrospective coding in cortical neurons. *arXiv preprint arXiv:2405.14810*, 2024.

Claudia Clopath, Lorric Ziegler, Eleni Vasilaki, Lars Büsing, and Wulfram Gerstner. Tag-trigger-consolidation: a model of early and late long-term-potentiation and depression. *PLoS computational biology*, 4(12):e1000248, 2008.

Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005.

Alain Destexhe, Michael Rudolph, and Denis Paré. The high-conductance state of neocortical neurons in vivo. *Nature reviews neuroscience*, 4(9):739–751, 2003.

Yudai Ebato, Sou Nobukawa, Yusuke Sakemi, Haruhiko Nishimura, Takashi Kanamaru, Nina Sviridova, and Kazuyuki Aihara. Impact of time-history terms on reservoir dynamics and prediction accuracy in echo state networks. *Scientific Reports*, 14(1):8631, 2024.

Benjamin Ellenberger, Paul Haider, Federico Benitez, Jakob Jordan, Kevin Max, Ismael Jaras, Laura Kriener, and Mihai A Petrovici. Backpropagation through space, time and the brain. *Nature Communications*, 2025.

Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

Samuel James Greydanus and Dmitry Kobak. Scaling down deep learning with MNIST-1d. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=n9pru4bJU9`.

Paul Haider, Benjamin Ellenberger, Laura Kriener, Jakob Jordan, Walter Senn, and Mihai A Petrovici. Latent equilibrium: a unified learning theory for arbitrarily fast computation with arbitrarily slow neurons. *Advances in neural information processing systems*, 34:17839–17851, 2021.

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*, pages 770–778, 2016.

Kazuki Irie, Anand Gopalakrishnan, and Jürgen Schmidhuber. Exploring the promise and limits of real-time recurrent learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=V2cBKtdC3a`.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

Nicholas A Lesica and Benedikt Grothe. Efficient temporal processing of naturalistic sounds. *PLoS One*, 3(2):e1655, 2008.

Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. *Advances in neural information processing systems*, 31, 2018.

James M Murray. Local online learning in recurrent networks with random feedback. *eLife*, 8:e43299, may 2019. ISSN 2050-084X. doi: 10.7554/eLife.43299. URL `https://doi.org/10.7554/eLife.43299`.

John D Murray, Alberto Bernacchia, David J Freedman, Ranulfo Romo, Jonathan D Wallis, Xinying Cai, Camillo Padoa-Schioppa, Tatiana Pasternak, Hyojung Seo, Daeyeol Lee, et al. A hierarchy of intrinsic timescales across primate cortex. *Nature neuroscience*, 17(12):1661–1663, 2014.

Cian O'Donnell. Nonlinear slow-timescale mechanisms in synaptic plasticity. *Current opinion in neurobiology*, 82:102778, 2023.

Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

Walter Senn, Dominik Dold, Akos F Kungl, Benjamin Ellenberger, Jakob Jordan, Yoshua Bengio, João Sacramento, and Mihai A Petrovici. A neuronal least-action principle for real-time learning in cortical circuits. *ELife*, 12:RP89674, 2024.

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pages 3067–3075. PMLR, 2017.

Aaron Spieler, Nasim Rahaman, Georg Martius, Bernhard Schölkopf, and Anna Levina. The expressive leaky memory neuron: an efficient and expressive phenomenological neuron model can solve long-horizon tasks. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vE1e1mLJ0U.

Pablo Stoliar, Julien Tranchant, Benoit Corraze, Etienne Janod, Marie-Paule Besland, Federico Tesler, Marcelo Rozenberg, and Laurent Cario. A leaky-integrate-and-fire neuron analog realized with a mott insulator. *Advanced Functional Materials*, 27(11):1604740, 2017.

Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJQDjk-0b.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Stuart Trenholm and Arjun Krishnaswamy. An annotated journey through modern visual neuroscience. *Journal of Neuroscience*, 40(1):44–53, 2020.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 2002.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Backpropagation*, pages 433–486. Psychology Press, 2013.

Jia-Qin Yang, Ruopeng Wang, Zhan-Peng Wang, Qin-Yuan Ma, Jing-Yu Mao, Yi Ren, Xiaoyang Yang, Ye Zhou, and Su-Ting Han. Leaky integrate-and-fire neurons based on perovskite memristor for spiking neural networks. *Nano Energy*, 74:104828, 2020.

Lingyun Zhao and Li Zhaoping. Understanding auditory spectro-temporal receptive fields and their changes with input statistics by efficient coding principles. *PLoS computational biology*, 7(8): e1002123, 2011.

Li Zhaoping. *Understanding Vision: Theory, Models, and Data*. Oxford University Press, Oxford, UK, 2014.

Nicolas Zucchet, Robert Meier, Simon Schug, Asier Mujika, and Joao Sacramento. Online learning of long-range dependencies. *Advances in Neural Information Processing Systems*, 36:10477–10493, 2023.

A Mathematical proofs

Hierarchical exponential kernels are equivalent to linear combinations of kernels

Before analyzing gradients in the network, we first derive a fundamental lemma for this theory. Hierarchical exponential kernels can be expressed as a linear summation of exponential kernels. This result can be derived using Laplace transforms; however, here we provide a derivation based on elementary calculus. Consider a double exponential kernel over a temporal variable x:

$$\overline{\overline{x}^{\tau_i}}^{\tau_j}(t) = \int_{t_0}^{t} dt' \int_{t_0}^{t'} dt'' \left(\frac{1}{\tau_j} e^{\frac{t'-t}{\tau_j}} \frac{1}{\tau_i} e^{\frac{t''-t'}{\tau_i}} x(t'') \right) \tag{7}$$

$$= \int_{t_0}^{t} dt'' x(t'') \int_{t''}^{t} dt' \left(\frac{1}{\tau_j} e^{\frac{t'-t}{\tau_j}} \frac{1}{\tau_i} e^{\frac{t''-t'}{\tau_i}} \right) \tag{8}$$

$$= \int_{t_0}^{t} dt'' x(t'') \frac{1}{\tau_i \tau_j} e^{\frac{t''}{\tau_i} - \frac{t}{\tau_j}} \int_{t''}^{t} dt' e^{\frac{t'}{\tau_j} - \frac{t'}{\tau_i}} \tag{9}$$

$$= \int_{t_0}^{t} dt'' x(t'') \frac{1}{\tau_j - \tau_i} \left(e^{\frac{t''-t}{\tau_j}} - e^{\frac{t''-t}{\tau_i}} \right) \tag{10}$$

$$= \frac{\tau_i}{\tau_i - \tau_j} \overline{x}^{\tau_i} + \frac{\tau_j}{\tau_j - \tau_i} \overline{x}^{\tau_j} . \tag{11}$$

This rule can be applied iteratively to additional exponential kernels. For example, for three exponential kernels:

$$\overline{\overline{\overline{x}^{\tau_i}}^{\tau_j}}^{\tau_k}(t) = \frac{\tau_i}{\tau_i - \tau_j} \overline{\overline{x}^{\tau_i}}^{\tau_k} + \frac{\tau_j}{\tau_j - \tau_i} \overline{\overline{x}^{\tau_j}}^{\tau_k} \tag{12}$$

$$= \frac{\tau_i}{\tau_i - \tau_j} \left(\frac{\tau_i}{\tau_i - \tau_k} \overline{x}^{\tau_i} + \frac{\tau_k}{\tau_k - \tau_i} \overline{x}^{\tau_k} \right) + \frac{\tau_j}{\tau_j - \tau_j} \left(\frac{\tau_k}{\tau_k - \tau_i} \overline{x}^{\tau_k} + \frac{\tau_j}{\tau_j - \tau_k} \overline{x}^{\tau_j} \right) \tag{13}$$

$$= \frac{\tau_i}{\tau_i - \tau_j} \frac{\tau_i}{\tau_i - \tau_k} \overline{x}^{\tau_i} + \frac{\tau_j}{\tau_j - \tau_i} \frac{\tau_j}{\tau_j - \tau_k} \overline{x}^{\tau_j} + \frac{\tau_k}{\tau_k - \tau_i} \frac{\tau_k}{\tau_k - \tau_j} \overline{x}^{\tau_k} . \tag{14}$$

Then, this lemma can be generalized to an arbitrary number of exponential kernels (a set of distinct time constants Υ), where $\mathcal{T}_j^{\tau_i} = 1$ when $i = j$, which allows a more compact representation:

$$\boldsymbol{\kappa}(x, \Upsilon, t) = \sum_{\tau_i \in \Upsilon} c_i \, \overline{x}^{\tau_i}(t) \ , \quad \text{with } c_i = \prod_{\tau_j \in \Upsilon, j \neq i} \frac{\tau_i}{\tau_i - \tau_j} = \prod_{\tau_j \in \Upsilon} \mathcal{T}_j^{\tau_i} \ . \tag{15}$$

The limitation arises when multiple time constants are identical, in which case the expression reduces to the special case of the alpha function, which cannot be written as a simple linear combination. We therefore assume $\tau_i \neq \tau_j$ for any $i \neq j$.

Exact gradient for 2-leaky-layer networks

To begin with, we transform the ODE $\tau \dot{\boldsymbol{u}} = -\boldsymbol{u} + \boldsymbol{W}\boldsymbol{r}$ into its solution form $\boldsymbol{u}(t) = \boldsymbol{W}\overline{\boldsymbol{r}}^{\tau}(t)$ with the initial condition $u(t_0) = 0$. We use bold symbols to denote vectors and matrices. Consider a network architecture with two hidden layers (and widths of the input, hidden and output layer denoted by n_{in}, n_1, n_2 and n_{out}), where neurons have exponential filters with different time constants τ, except for the instantaneous output neurons (formally, we only need that neurons from different layers have different time constants). For the last leaky layer, the gradient for minimizing the objective function can be written as:

$$\dot{\boldsymbol{W}}_{[2]}(t) \propto -\frac{d\boldsymbol{u}_{[2]}(t)}{dW_{[2]}} \frac{\partial \boldsymbol{r}_{[2]}(t)}{\partial \boldsymbol{u}_{[2]}(t)} \frac{\partial \mathcal{C}(t)}{\partial \boldsymbol{r}_{[2]}(t)} \tag{16}$$

$$= -\bar{\boldsymbol{r}}_{[1]}(t) \odot \boldsymbol{\rho}'_{[2]}(t) \frac{\partial \mathcal{C}(t)}{\partial \boldsymbol{r}_{[2]}(t)} . \tag{17}$$

Here $\bar{\boldsymbol{r}}_l$ without superscript is a matrix of shape $(n_l \times n_{l+1})$ where each column is filtered by corresponding post-synaptic τ. Since the output layer is instantaneous, we can directly compute the error arriving at the last leaky layer: $\boldsymbol{\epsilon}(t) = -\frac{\partial \mathcal{C}(t)}{\partial \boldsymbol{r}_{[2]}(t)} = -\boldsymbol{W}_{\text{out}}^{\mathsf{T}} \frac{\partial \mathcal{C}(t)}{\partial \boldsymbol{r}_o(t)}$, which can be readily

obtained. The exact gradient can then be computed online $\propto \boldsymbol{z}_{[2]} \odot \boldsymbol{\epsilon}$. This reduces to RFLO Murray [2019] and E-prop Bellec et al. [2020] for the last hidden layer, which is in fact equivalent to $\zeta_{[2]}^{\tau=0}$ considering downstream instantaneous neurons.

For the deeper layer L1, the simple eligibility traces with local τ are no longer enough, since the temporal gradient $\frac{\partial \mathcal{C}(t)}{\partial \boldsymbol{r}_{[1]}(t'<t)} \neq 0$. The gradient must integrate over temporal contributions $\frac{\partial \boldsymbol{u}_{[2]}(t)}{\partial \boldsymbol{r}_{[1]}(t'<t)}$, which are scaled by the decay term $\frac{1}{\tau} e^{\frac{t'-t}{\tau}}$:

$$\dot{\boldsymbol{W}}_{[1]}(t) \propto -\int_{t_0}^{t} \bar{\boldsymbol{r}}_{\text{in}}(t') \boldsymbol{\rho}'_{[1]}(t') \frac{\partial \boldsymbol{u}_{[2]}(t)}{\partial \boldsymbol{r}_{[1]}(t')} \frac{\partial \mathcal{C}(t)}{\partial \boldsymbol{u}_{[2]}(t)} \, dt' \tag{18}$$

$$= \int_{t_0}^{t} \bar{\boldsymbol{r}}_{\text{in}}(t') \boldsymbol{\rho}'_{[1]}(t') \boldsymbol{W}_{[2]}^{\mathsf{T}} \left(\frac{1}{\boldsymbol{\tau}} e^{\frac{t'-t}{\boldsymbol{\tau}}} \boldsymbol{\epsilon}(t) \right) dt' \, . \tag{19}$$

Since we aim to rearrange these terms according to time constants, it becomes difficult to proceed directly with matrix operations. To compute this explicitly, we replace matrix operations with summations. We start by focusing on an arbitrary neuron $\star$ in L1, denote its input weight vector by $\boldsymbol{W}_{\text{in},\star} \in \mathbb{R}^{n_{\text{in}}}$, the corresponding eligibility trace by $\boldsymbol{z}_\star \in \mathbb{R}^{n_{\text{in}}}$ and the weight to neuron i in L2 by $W_{i\star}$. We then calculate

$$\dot{\boldsymbol{W}}_{\text{in},\star}(t) \propto \int_{t_0}^{t} \bar{\boldsymbol{r}}_{\text{in}}^{\tau_\star}(t') \rho'_\star(t') \sum_i W_{i\star} \frac{1}{\tau_i} e^{\frac{t'-t}{\tau_i}} \epsilon_i(t) dt' \tag{20}$$

$$= \sum_i \epsilon_i(t) \int_{t_0}^{t} \boldsymbol{z}_\star(t') \rho'_\star(t') W_{i\star} \frac{1}{\tau_i} e^{\frac{t'-t}{\tau_i}} dt' \tag{21}$$

$$= \sum_i W_{i\star} \epsilon_i(t) \overline{\boldsymbol{z}_\star}^{\tau_i}(t) \, . \tag{22}$$

Equation 21 can be considered as the closed-form solution of the RTRL sensitivity in continuous time, which accounts for the contribution of parameters in $\boldsymbol{W}_{\text{in}}$ to each L2 neuron where the error arrives. We next group L2 neurons that share the same time constant and collect all distinct τ's into the set Υ_{L2}. This allows us to merge the coefficients within each group as the kernel gain $\mathcal{K}$ and rewrite the expression as:

$$\dot{\boldsymbol{W}}_{\text{in},\star}(t) \propto \sum_{\Upsilon_{L2}} \mathcal{K}_\star^\tau \boldsymbol{\zeta}_\star^\tau \, , \quad \mathcal{K}_\star^\tau = \sum_{i|\tau_i=\tau} W_{i\star} \epsilon_i(t) = \sum_i \epsilon_i(t) W_{i\star} \mathcal{T}_i^\tau \, , \tag{23}$$

where $\mathcal{T}$ compactly replaces the selection condition $\tau_i = \tau$. Therefore, we construct a specialized version of K-prop in which the exact gradient can be computed in two layers networks more efficiently than RTRL.

To three and more leaky layers

In the same manner, we can compute the gradient for networks with more leaky layers. To maintain consistent layer indexing for the back-propagation process, we use $[-\text{index}]$ to denote layers by their distance from the output, and indices $i, j, k, \ldots$ follow this order, with ϵ still as the error signal arrives at the last leaky layer. We write the chain rule in an order that facilitates the later transformation into summation form:

$$\dot{\boldsymbol{W}}_{[-3],\star}(t) \propto -\frac{\partial \mathcal{C}(t)}{\partial u_{[-1]}(t)} \int_{t_0}^{t} dt' \left(\frac{\partial u_{[-1]}(t)}{\partial u_{[-2]}(t')} \int_{t_0}^{t'} dt'' \frac{\partial u_{[-2]}(t')}{\partial r_{[-3]}(t'')} \boldsymbol{z}_\star(t'') \right) \tag{24}$$

$$= \sum_i \epsilon_i(t) \rho'_i(t) \int_{t_0}^{t} dt' \sum_j W_{ij} \frac{1}{\tau_i} e^{\frac{t'-t}{\tau_i}} \rho'_j(t') \int_{t_0}^{t'} dt'' W_{j\star} \frac{1}{\tau_j} e^{\frac{t''-t'}{\tau_j}} \boldsymbol{z}_\star(t'') \tag{25}$$

$$= \sum_i \sum_j W_{ij} W_{j\star} \epsilon_i(t) \rho'_i(t) \int_{t_0}^{t} dt' \left(\frac{1}{\tau_i} e^{\frac{t'-t}{\tau_i}} \rho'_j(t') \int_{t_0}^{t'} dt'' \frac{1}{\tau_j} e^{\frac{t''-t'}{\tau_j}} \boldsymbol{z}_\star(t'') \right) \tag{26}$$

$$= \sum_i \sum_j W_{ij} W_{j\star} \epsilon_i(t) \rho'_i(t) \overline{\rho'_j \overline{\boldsymbol{z}_\star(t'')}^{\tau_j}}^{\tau_i}(t) \, . \tag{27}$$

To reduce this triple exponential kernel to a double exponential form, we can follow the computation in Equation 9. However, in this case, we encounter an intractable expression due to the temporally varying $\rho'_j(t')$.

$$\overline{\rho'_j \overline{\boldsymbol{z}_\star}^{\tau_j}}^{\tau_i}(t) = \int_{t_0}^{t} dt' \frac{1}{\tau_i} e^{\frac{t'-t}{\tau_i}} \rho'_j(t') \int_{t_0}^{t'} dt'' \frac{1}{\tau_j} e^{\frac{t''-t'}{\tau_j}} \boldsymbol{z}_\star(t'') \tag{28}$$

$$= \int_{t_0}^{t} dt'' \boldsymbol{z}_\star(t'') \frac{1}{\tau_j \tau_i} e^{\frac{t''}{\tau_j} - \frac{t}{\tau_i}} \left[\int_{t''}^{t} dt' \rho'_j(t') e^{\frac{t'}{\tau_i} - \frac{t'}{\tau_j}} \right] . \tag{29}$$

This expression can be approximately handled by extracting ρ'_j from the integral and introducing a surrogate variable $\hat{\rho}'$, such that $\frac{\rho'_j(t')}{\hat{\rho}'_j(t)} \approx 1$ for recent times $t - t' < T\epsilon$. This approximation effectively requires ρ' to vary slowly over time, which is better satisfied by smooth activation functions.

$$\overline{\rho_j \overline{\boldsymbol{z}_\star}^{\tau_j}}^{\tau_i}(t) = \hat{\rho}'_j(t) \overline{\frac{\rho'_j}{\hat{\rho}'_j(t)} \overline{\boldsymbol{z}_\star}^{\tau_j}}^{\tau_i}(t) \tag{30}$$

$$\approx \hat{\rho}'_j(t) \overline{\overline{\boldsymbol{z}_\star}^{\tau_j}}^{\tau_i}(t) . \tag{31}$$

Then, according to Equation 11, we can convert the double exponential kernel into a summation:

$$\dot{\boldsymbol{W}}_{[-3],\star}(t) \propto \sum_i \sum_j W_{ij} W_{j\star} \epsilon_i(t) \rho'_i(t) \hat{\rho}'_j(t) \left(\frac{\tau_j}{\tau_j - \tau_i} \overline{\boldsymbol{z}_\star}^{\tau_j}(t) + \frac{\tau_i}{\tau_i - \tau_j} \overline{\boldsymbol{z}_\star}^{\tau_i} \right) \tag{32}$$

$$= \sum_i \sum_j W_{ij} W_{j\star} \epsilon_i(t) \rho'_i(t) \hat{\rho}'_j(t) \left(\mathcal{T}_i^{\tau_j} \boldsymbol{\zeta}_\star^{\tau_j}(t) + \mathcal{T}_j^{\tau_i} \boldsymbol{\zeta}_\star^{\tau_i}(t) \right) . \tag{33}$$

In the same sense, we can derive a similar expression for the -4 hidden layer by introducing an additional summation over the k-th neuron in the additional layer.

$$\dot{\boldsymbol{W}}_{[-4],\star}(t) \propto \sum_i \sum_j \sum_k W_{ij} W_{jk} W_{k\star} \epsilon_i \rho'_i(t) \overline{\rho'_j \overline{\rho'_k \overline{\boldsymbol{z}_\star}^{\tau_k}}^{\tau_j}}^{\tau_i}(t) \tag{34}$$

$$\approx \sum_{i,j,k} \sum_k W_{ij} W_{jk} W_{k\star} \epsilon_i(t) \rho'_i \hat{\rho}'_j \hat{\rho}'_k(t) \overline{\overline{\boldsymbol{z}_\star}^{\tau_k}}^{\tau_j}}^{\tau_i}(t) \tag{35}$$

$$= \sum_{i,j,k} \sum_k W_{ij} W_{jk} W_{k\star} \epsilon_i \rho'_i \hat{\rho}'_j \hat{\rho}'_k \left(\mathcal{T}_j^{\tau_k} \mathcal{T}_i^{\tau_k} \boldsymbol{\zeta}_\star^{\tau_k} + \mathcal{T}_i^{\tau_j} \mathcal{T}_k^{\tau_j} \boldsymbol{\zeta}_\star^{\tau_j} + \mathcal{T}_j^{\tau_i} \mathcal{T}_k^{\tau_i} \boldsymbol{\zeta}_\star^{\tau_i} \right)(t) . \tag{36}$$

To generalize this result to arbitrary depth, we define a path p as synaptic chain passing from a initial neuron $i_{[l]}$ at layer l to a next neuron $j_{[l+1]}$ at layer $l+1$ until we end a neuron $v_{[L]}$ in the output layer, $p = (i_{[l]}, j_{[l+1]} ..., v_{[L]})$. The path weight w_p is defined as the product of all synaptic weights, the rate derivatives ρ', and the error ϵ_p entering the path p,

$$w_p = \epsilon_p \prod_{i \in p} W_{p(i+1),p(i)} \; \hat{\rho}'_{i+1} \tag{37}$$

where i runs through the components of the path p up to the second last.

We next define the set of paths $\boldsymbol{\mathcal{P}}_\star$ as set of paths that start at neuron $\star$ in a hidden layer l. By $\boldsymbol{\mathcal{P}}_\star^\tau$ we denote the subset of paths starting at $\star$ and containing along its path a neuron that low-pass filters the input with τ, i.e. $\tau \in \Upsilon_p$. Using these path sets, we can rewrite the hierarchical summation over layers $\sum_{i,j,k}$ as a single summation over paths:

$$\dot{\boldsymbol{W}}_\star(t) \propto \sum_{p \in \boldsymbol{\mathcal{P}}_\star} w_p \sum_{\tau \in \Upsilon_p} \prod_{j \in p^{\setminus \star}} \mathcal{T}_j^\tau \boldsymbol{\zeta}_\star^\tau . \tag{38}$$

Thus, we obtain an approximation of the gradient for all neurons in a network of arbitrary depth, expressed in terms of paths. The remaining question is how to compute this efficiently, since the number of paths in $\boldsymbol{\mathcal{P}}_\star$ grows exponentially $|\boldsymbol{\mathcal{P}}_\star| = \prod_{l'=l+1}^{L} n_{l'}$, with l' running across all layers downstream of neuron $\star$.

Kernel gain by regrouping summation over τ

The first observation is that a single τ can participate in multiple paths, and this multiplicity becomes even larger when the same τ is shared within a layer. This suggests that maximizing the reuse of $\zeta_\star^\tau$ is beneficial. Therefore, we aim to group the coefficients with respect to τ by changing the order of summation. We define the set $\Upsilon_\star$ of all downstream time constants and rewrite Eq. (38) by

$$\dot{\boldsymbol{W}}_\star(t) \propto \sum_{\tau\in\Upsilon_\star} \sum_{p\in\boldsymbol{\mathcal{P}}_\star^\tau} w_p \prod_{j\in p^{\setminus\star}} \mathcal{T}_j^\tau \boldsymbol{\zeta}_\star^\tau = \sum_{\tau\in\Upsilon_\star} \mathcal{K}_\star^\tau \boldsymbol{\zeta}_\star^\tau \tag{39}$$

$$\text{with } \mathcal{K}_\star^\tau = \sum_{p\in\boldsymbol{\mathcal{P}}_\star^\tau} w_p \prod_{j\in p^{\setminus\star}} \mathcal{T}_j^\tau \,. \tag{40}$$

Three challenges arise when computing $\mathcal{K}$: 1. The number of paths in $\boldsymbol{\mathcal{P}}_\star^\tau$ grows exponentially with increasing depth. 2. Using w_p requires neurons to access distant downstream information, such as feedback error, synaptic weights, and activations. 3. Neurons must be aware of the precise downstream network structure, i.e., which neurons belong to a given $p^{\setminus\star}$. Since ζ can be tracked locally in an online manner, it is crucial to determine how $\mathcal{K}$ can be computed efficiently.

Layer-wise propagation of $\mathcal{K}$

To address the above challenges, we note that $\mathcal{K}$ can be propagated from layer $l+1$ to l, and that this leads to an efficient computational scheme. In a feedforward architecture, each hidden neuron inherits and extends all paths from layer $l+1$. Define $p_+ = p^{\setminus\star}$ as the path obtained from p by removing neuron $\star$ in layer l and instead starting from neuron $\star\star$ in layer $l+1$. Then we have $w_p = W_{\star\star,\star}\,\hat{\rho}'_{\star\star}\,w_{p_+}$ and

$$\mathcal{K}_\star^\tau = \sum_{p\in\boldsymbol{\mathcal{P}}_\star^\tau} w_p \prod_{j\in p^{\setminus\star}} \mathcal{T}_j^\tau \tag{41}$$

$$= \sum_{p\in\boldsymbol{\mathcal{P}}_\star^\tau} \mathcal{T}_{\star\star}^\tau \hat{\rho}'_{\star\star} W_{\star\star,\star} \left(w_{p'} \prod_{j\in p_+^{\setminus\star\star}} \mathcal{T}_j^\tau \right) . \tag{42}$$

Since many paths share the same node in layer $l+1$, we can rearrange the summation to first sum over neuron i in the next layer:

$$\mathcal{K}_\star^\tau = \sum_{i\in l+1} \hat{\rho}'_i \mathcal{T}_i^\tau W_{i\star} \sum_{p\in\boldsymbol{\mathcal{P}}_{i,\tau}} w_p \prod_{j\in p} \mathcal{T}_j^\tau \tag{43}$$

$$= \sum_{i\in l+1} \hat{\rho}'_i \mathcal{T}_i^\tau W_{i\star}\, \mathcal{K}_i^\tau \,. \tag{44}$$

Note that $\mathcal{K}_\star^\tau$ must be defined for every τ and every neuron in the network to support this iterative computation. This equation can be written in matrix form as in Equation 4, which highlights its strong similarity to instantaneous gradient backpropagation.

$\mathcal{T}$ in other network architectures

Under a rigorous definition, $\mathcal{T}_j^\tau = 0$ when τ does not occur on any path through neuron j. In a fully connected feedforward network, this applies only to time constants associated with other neurons in the same layer as j and distinct from τ_j. Other architectures can also contain time constants that do not occur on any path through j in different situations.

Residual networks [He et al., 2016] provide one example where skip connections can provide pathways with faster temporal responses than the residual pathway. To apply K-prop to a residual network, we propagate each kernel gain $\mathcal{K}^\tau$ along both the skip and residual pathways, then sum the contributions at the merge point for each τ. The eligibility traces ζ^τ are unchanged. Along the skip pathway, set $\mathcal{T}_j^\tau = 0$ for time constants that occur only along the residual pathway, since they do not occur on any path through a neuron on the skip pathway.

More generally, consider a direct connection matrix from a source layer m to target layer n, with neuron j in layer m. Kernels contained by any other, more indirect paths between layers m and n (via

other intermediate layers) will neither affect nor be affected by kernels along the direct connection from m to n, so their contribution to $\mathcal{T}^{\tau}_{j,[mn]}$ should be zero. More formally, let $\Upsilon_{[l<m]}$ denote the set of time constants associated with layers whose indices satisfy $l < m$, and let $\Upsilon_{[l\geq n]}$ denote the set associated with layers whose indices satisfy $l \geq n$. The transformation factor for this connection is

$$\mathcal{T}^{\tau}_{j,[mn]} = \begin{cases} \frac{\tau}{\tau-\tau_j} & \tau \in \Upsilon_{[l<m]} \text{ or } \tau \in \Upsilon_{[l\geq n]} \\ 1 & \tau_j = \tau \\ 0 & \text{otherwise.} \end{cases} \tag{45}$$

This allows us to create the transformation matrices $\mathcal{T}$ efficiently before training without having to evaluate all possible paths.

Identical or zero τ

A special case arises when identical time constants appear across multiple layers. As discussed in the main text, this leads to singularities in the kernel decomposition. The singularity can be resolved either by grouping identical τ into a single basis component or by introducing infinitesimal perturbations. The grouping approach suggests replacing $\boldsymbol{\zeta}$ with higher-order kernels built from nested identical τ, such as the alpha function $\frac{t}{\tau}e^{-t/\tau}$. The transformation factor $\mathcal{T}$ and the propagation rule would then need to be modified accordingly to accommodate the new basis. While the grouping approach is theoretically valid, it introduces additional complexity in both the kernel representation and the propagation rules. To prevent grouping, we consider infinitesimal perturbations for simplicity and consistency with the main formulation.

Biologically, different time constants across layers are consistent with the hierarchical increase of time constants in the primate cortex [Murray et al., 2014]. Empirically, we do not observe any benefit from using identical or near-identical time constants across layers. In our experiments, time constants are chosen to be distinct across layers. We further verify that small perturbations (e.g., differences on the order of 0.1) do not affect numerical stability or performance.

Another special case is $\tau = 0$. This case is taken into account because $\mathcal{T}_j^{\tau=0} = 0$ for any neuron j including $\tau_j = 0$. An instantaneous pathway with co-existence of $\tau = 0$ neurons across layers does not cause any singularity, which collapses into simple backpropagation.

Practical implementation of K-prop

A few notes for practical implementation: 1. We define $\zeta^{\tau_{\text{up}}} = 0$, so each layer can omit tracking the upstream components of $\boldsymbol{\zeta}$. 2. A special case arises for the first leaky layer: its τ does not need to be included in Υ_{net} (since there are no upstream neurons), and its nonlinearity ρ_{L1} does not affect the approximation. 3. $\mathcal{K}$ propagation is only required when an external error signal is present (otherwise all $\mathcal{K} = 0$). In many tasks, including real world scenarios, error signals are sparse in time, allowing $\mathcal{K}$ propagation to be triggered in an event-based manner. Altogether, in practical implementations, K-prop has both computational and memory costs on the order of half of $\mathcal{O}(n_\tau n^2)$. Under this efficient implementation, we can plot the cost comparison:

This reduction is well illustrated in the CartPole task, where K-prop reduces RTRL memory cost by $28\times$ and computational cost by $1{,}684\times$, turning an otherwise intractable method into a practical one. Although BPTT is computationally efficient, it incurs a memory cost that scales with sequence length T, as it requires storing the full trajectory. In comparison, K-prop reduces memory usage by up to $57\times$, at the expense of a moderate ($\sim 4\times$) increase in computation relative to BPTT, demonstrating a classical space-time trade-off. Simulations are programmed in Python with PyTorch. Code is available through `https://github.com/Baskin-LIU/Kernel-Propagation#`

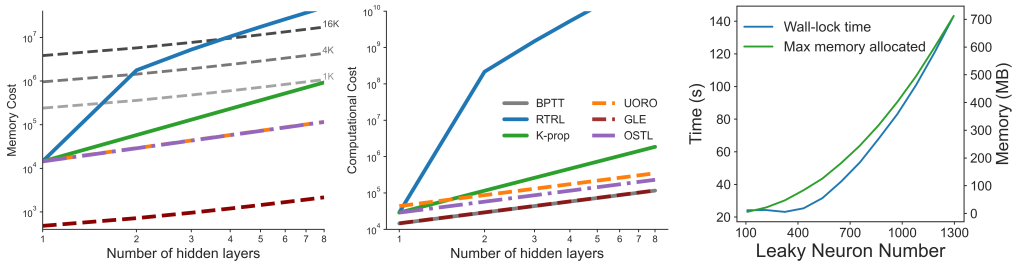

Figure 7: Compare the cost under multi-layer architecture. **Left**: Memory cost **Middle**: Computational cost per time step. GLE has $\mathcal{O}(n)$ memory and $\mathcal{O}(n^2)$ computational cost. All costs are computed in optimal layer-wise implementations instead of directly from scaling law. **Right**: Wall-clock release time and maximum memory allocation per epoch on the MNIST1D dataset, measured across an increasing number of leaky neurons in 5-layer networks. Experiments were conducted on a local machine equipped with an NVIDIA RTX 4060 Laptop GPU (8 GB). Overall running time reflects additional latency introduced by temporal loop overhead.

Algorithm 1 Kernel Propagation (K-prop) for Online Learning in Leaky Networks

Initialization: Network with L layers, time constants τ, step size dt
Initialize weights $\{\boldsymbol{W}, \boldsymbol{b}\}$ randomly ▷ bias is treated as weight for $r = 1$
Construct transformation factor matrices $\{\mathcal{T}\}$
for each iteration / episode **do**
  Reset $\bar{\boldsymbol{r}}_{[l]} \leftarrow 0$, $\boldsymbol{\zeta}_{[l]} \leftarrow 0$, $\Delta \boldsymbol{W}_{[l]} \leftarrow 0$
  for each time step t **do**
    for layer $l = 1$ to L **do** ▷ Forward dynamics
      $\bar{\boldsymbol{r}}_{[l]} \leftarrow (1 - \frac{dt}{\tau_l})\bar{\boldsymbol{r}}_{[l]} + \frac{dt}{\tau_l}\boldsymbol{r}_{[l]}$
      $\boldsymbol{u}_{[l]} \leftarrow \boldsymbol{W}_{[l]}\bar{\boldsymbol{r}}_{[l-1]} + \boldsymbol{b}_{[l]}$
      $\boldsymbol{r}_{[l]} \leftarrow \rho(\boldsymbol{u}_{[l]})$
      $\boldsymbol{\zeta}_{[l]} \leftarrow (1 - \frac{dt}{\boldsymbol{\tau}_{\text{net}}}) \odot \boldsymbol{\zeta}_{[l]} + \frac{dt}{\boldsymbol{\tau}_{\text{net}}} \left(\rho'_{[l]} \otimes \bar{\boldsymbol{r}}_{[l-1]}\right)$ ▷ Deep eligibility trace update
    end for
    if $\frac{\partial \mathcal{C}}{\partial \boldsymbol{r}_L} \neq 0$ **then** ▷ Check if error signal is available
      $\mathcal{K}^{\tau}_{L} \leftarrow \frac{\partial \mathcal{C}}{\partial \boldsymbol{r}_L} \quad \forall \tau \in \boldsymbol{\Upsilon}_{\text{net}}$ ▷ Initialize $\mathcal{K}$ at output layer
      for layer $l = L$ down to 1 **do** ▷ Backward kernel propagation
        $\mathcal{K}^{\tau}_{[l-1]} \leftarrow \boldsymbol{W}^{\top}_{[l]} \left(\hat{\boldsymbol{\rho}}'_{[l]} \odot \mathcal{K}^{\tau}_{[l]} \odot \mathcal{T}^{\tau}_{[l]}\right)$
        $\dot{\boldsymbol{W}}_{[l]} \leftarrow \sum_{\tau \in \boldsymbol{\Upsilon}^{l}_{\text{down}}} \text{diag}(\mathcal{K}^{\tau}_{[l]})\, \boldsymbol{\zeta}^{\tau}_{[l]}$
        $\Delta \boldsymbol{W}_{[l]} \leftarrow \Delta \boldsymbol{W}_{[l]} + \dot{\boldsymbol{W}}_{[l]}$ ▷ Gradient accumulation
      end for
    end if
    if Update weight **then**
      for layer $l = 1$ to L **do**
        $\boldsymbol{W}_{[l]} \leftarrow \boldsymbol{W}_{[l]} - \eta\, \Delta \boldsymbol{W}_{[l]}$
        $\Delta \boldsymbol{W}_{[l]} \leftarrow 0$
      end for
    end if
  end for
end for

B Experimental details

B.1 MNIST1D dataset

MNIST1D converts digit patterns analogous to the handwritten images in MNIST [LeCun et al., 2002] into one-dimensional temporal sequences. Each sample spans 360ms and contains a digit pattern of approximately 90ms embedded at a random temporal position, requiring both temporal

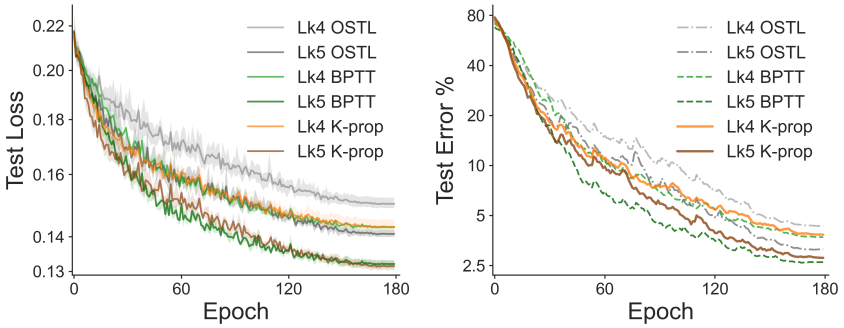

Figure 8: **Learning curves on MNIST1D.** Left: Mean, minimum, and maximum training loss. K-prop closely matches the loss achieved by BPTT, whereas OSTL converges to a substantially higher loss. Right: Mean test error for the Lk4 and Lk5 networks. Curves are smoothed for visualization.

localization and pattern recognition. Apart from increasing the number of time steps, we use the default sample-generation settings of Greydanus and Kobak [2024], including the default training/test split of 4000/1000 samples.

B.2 GSC dataset

The Google Speech Commands (GSC) dataset [Warden, 2018] comprises utterances lasting approximately 0.8-1s, recorded from multiple speakers. We evaluate the 20-class command-label subset, rather than the 12-class subset used in Ellenberger et al. [2025]. The selected commands are ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'nine', 'eight','yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go']. We use the default training, validation, and test splits provided by the PyTorch dataset. The original audio waveforms are converted into 80-channel log-Mel spectrograms and interpolated to 500 time frames, corresponding to a time step of approximately 2ms. We apply data augmentation based on SpecAugment [Park et al., 2019], including frequency masking over four channels, temporal shifts of up to 25ms, and temporal warping of up to 25ms.

B.3 Loss and Optimizer

In all experiments, we use $r = \tanh(\alpha u)$ with $\alpha \approx 0.5$, making it behave similarly to a centered sigmoid. The learning rate decays exponentially in N-sum and follows a cosine annealing schedule (180 epochs total) in MNIST1D. All methods share the same learning rate scheduler. We use standard gradient descent for N-sum and the Adam optimizer for all other tasks. For Adam, (β_1, β_2) are set to (0.9, 0.999) for BPTT and (0.8, 0.995) for K-prop to reduce momentum and encourage stronger averaging. We have verified that (0.8, 0.995) on BPTT yields worse performance, excluding the advantage brought by the optimizer setup.

The loss is defined online in all experiments. For N-sum, we use logistic output units with a mean-squared-error loss. For MNIST1D and GSC, cross-entropy loss is evaluated at every time step within a designated response period: the entire sequence for MNIST1D and the final 600,ms for GSC. The total loss is therefore the sum over all answering time steps T, $\sum_{t \in T} \mathcal{L}(u_o(t), y)$.

We use a set of random seeds 0, 42, 428, 781, 3407, 115, 256, 666 for training.

B.4 Reinforcement learning tasks

For the Cartpole environment, we extend the termination boundaries to $\pm 36^\circ$ (from $\pm 24^\circ$) for pole angle and ± 3.6 (from ± 2.4) for cart position, allowing longer and more variable episode. To obtain more continuous inputs, observations from each environment step (20 ms) are interpolated by inserting 3 intermediate frames to obtain $dt = 5ms$. Actor has two output neurons with softmax activation, producing the probability of Left and Right. The action choice at each step is sampled by a Bernoulli distribution with the possibility.

In LunarLander, the agent must smoothly land a spacecraft within a target zone using four discrete actions: `do nothing`, `fire left engine`, `fire main engine`, and `fire right engine`. The environment follows the default OpenAI Gymnasium configuration, except that the horizontal position is masked from the observation. Frame interpolation is not required, and actions are sampled in the same manner as in CartPole. The task is considered successfully learned when the recent average reward exceeds 205.

The reinforcement learning algorithm follows Bellec et al. [2020], which transforms TD(λ) $\dot{W} \propto \bar{\delta}\frac{\partial V}{\partial W} \rightarrow \delta\overline{\frac{\partial V}{\partial W}}$. The critic propagates a scalar value signal, while the actor propagates a cross-entropy loss using the chosen action as the target label. The propagated gradient is first accumulated in an auxiliary term that decays with λ, and is then scaled by $\delta(t) = R(t) + \gamma V(t + dt) - V(t)$, where $\gamma = 0.99$ is the discount rate. $V(t + dt)$ is computed by a target critic network that is softly updated toward the critic with a factor of 0.05. We devices a special learning rate scheduler: After each episode, the learning rate is decreased by a factor of $\times 0.997$ if the episodic reward is lower than the running average, and increased by $+0.001$ when episodic reward is larger. otherwise. Due to the inherent instability of actor–critic methods and the absence of batch learning, we clip both δ and the gradient norm to prevent value explosion.

B.5 Implementation configurations

Model	dt (ms)	Leaky layers	L1	L2	L3	L4	L5
N-sum	0.25	3	[0, 6.28]	0.25, 3, 6	2, 4, 10		
Lk4	1	4	[0, 10]	3, 6	2, 7	2.5, 8	
Lk5	1	5	[0, 10]	3, 6	2, 7	1, 8	2.5, 6.6
GSC	2	4	[0, 40]	3,12, 24	4, 16, 36	50, 400	
Cartpole	5	3	[0, 40]	5, 10	11, 24		
Lunarlander	5	3	[0, 20]	5, 20	6, 10		

Table 2: Network time constants (ms). Since the first layer does not need to be included in Υ_{net}, it is preferable to use a wider range of time constants in the first layer and fewer in downstream layers. Accordingly, we assign the first-layer time constants τ to be evenly spaced in log space. In the other layers, we use two distinct τ values per layer, divided into two equal groups. We find that introducing more distinct τ values within a layer does not yield notable improvements, as long as both a group of "fast" and a group of "slow" neurons are present. Square brackets indicate the log-space range for L1.

The issues from online weight updating Section 5.3 can be mitigated either by using very small learning rates or by accumulating gradients over a time interval and applying periodic updates. We find the latter more effective, as it preserves learning speed while improving stability. This design choice is also consistent with biological observations, where long-term synaptic plasticity operates on slower timescales rather than instantaneously [Clopath et al., 2008, O'Donnell, 2023].

Meanwhile, our experiments indicates a necessity of "fast" neurons with much shorter τ than typical neuron membrane potential time constants. Such range of time constants could be achieved through modulation of the membrane conductance [Destexhe et al., 2003] or prospective coding [Haider et al., 2021, Senn et al., 2024, Brandt et al., 2024].

Both N-sum, Cartpole, and LunarLander simulations can run on CPU hardware within 20 minutes. MNIST1D and GSC benefits from GPU acceleration due to larger matrix operations. A mid-range GPU such as a GTX 4060 reduces MNIST1D training time from approximately 4 hours on CPU to around 1.5 hours. Additional experimental details are provided below.

Model	Network width	α	Learning rate	Update Period	Batch size
N-sum	6-6-6-2	0.6	0.1	50 ms	1
Lk4	36-54-60-60-72-10	0.6	0.01	180 ms	100
Lk5	54-60-60-60-60-72-10	0.5	0.01	180 ms	100
GSC	240-360-360-400-400-20	0.5	0.001	1s	256
Cartpole	32-40-30-20-2	0.6	0.01	3 s	1
Lunarlander	120-120-80-20-4	0.6	0.0002	-	1

Table 3: Network configurations

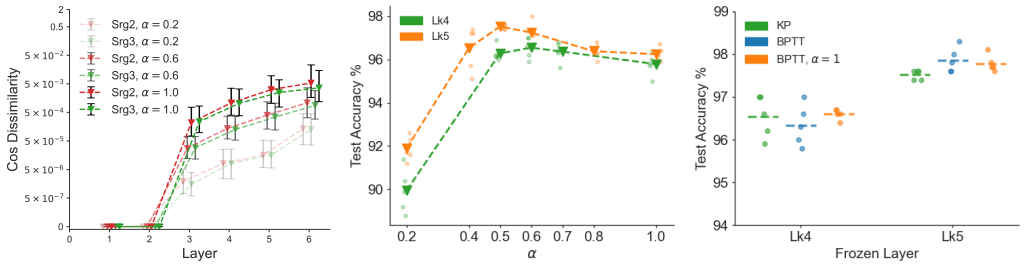

Figure 9: Further investigation of how the surrogate and nonlinearity affect K-prop approximation. (Left) Gradient cosine similarity check with BPTT like Fig. 3 on α and another surrogate. The dissimilarity increased when bigger α enhance nonlinearity. Here $\hat{\rho}' = \bar{\rho}'^{\tau_\rho}$ while Srg2 uses the neurons' local τ for τ_ρ and Srg3 uses τ_ρ as a summation of downstream layers' τ. Srg3 shows an advantage in early layers while slightly degrading layers close to the output. (Mid) Investigating the practical tradeoff between nonlinearity and K-prop gradient quality by tuning α in MNIST1D, which shows the different sweet spots on Lk4 and Lk5. (Right) Comparing with networks with $\alpha = 1$ and trained with BPTT. Enhanced nonlinearity does not boost final performance.

B.6 Learning under noise

To investigate the sensitivity of K-prop to noise in its eligibility dynamics, we inject an independent Gaussian perturbation into each integration step of Eq. (5), with $\mathfrak{n} \sim \mathcal{N}(0, \sigma^2)$. We evaluate the Lk4 network on MNIST1D using noise levels $\sigma \in 0.2, 1.0, 5.0, 10, 20$. K-prop remains effective across a broad range of noise levels. Given that z is bounded within $[-1, 1]$, the results indicate that the perturbations must be large relative to the driving signal before learning performance deteriorates substantially.

We attribute this robustness to two properties of K-prop:

1. Independent Gaussian perturbations are smoothed by the leaky integration dynamics. When driven by white noise, these dynamics produce an Ornstein–Uhlenbeck-like process in which the gradient signal contributes to the drift.

2. The resulting noisy gradient estimates are further averaged over time through gradient accumulation and the optimizer dynamics.

These results provide preliminary evidence that K-prop can tolerate the noisy dynamics expected in physical computing substrates.

C Robustness of the approximation

C.1 Tuning the non-linearity

In terms of nonlinearity, the K-prop approximation works better for smoother and flatter functions such as logistic and tanh. Scaling down the input $\rho(\alpha x), 0 < \alpha < 1$ slows down the variation and thus improves the approximation. However, a smaller α reduces the expressiveness of the nonlinearity, which may degrade performance. Therefore, there is a trade-off between approximation quality and nonlinearity, and a "sweet spot" typically exists that yields the best performance. The sweet spot depends on the network depth and τ configuration, as illustrated in the MNIST1D experiment.

In Lk4, the test accuracy increases significantly from $\alpha = 0.2$ and reaches an optimum around $\alpha = 0.5, 0.6$. The performance slightly decreases as α approaches 1, while still maintaining an accuracy of 94%, suggesting that K-prop preserves a usable gradient quality. In Lk5, the sweet spot shifts to $\alpha = 0.5$. We therefore choose $\alpha = 0.6$ for Lk4 and $\alpha = 0.5$ for Lk5. Meanwhile, models with $\alpha = 1$ trained with BPTT do not show significant improvement, indicating that α scaling is an effective hyperparameter that can be tuned for K-prop.

C.2 Surrogate choice

The choice of surrogate derivative has an important effect on K-prop performance. In all experiments, we use the low-pass-filtered surrogate $\hat{\rho}' = \bar{\rho'}^{\tau_\rho}$ which consistently improves performance over the instantaneous choice $\hat{\rho}' = \rho'$.

For all tasks except GSC, we set the filtering time constant τ_ρ to the local time constant τ of each neuron. For GSC, τ_ρ at each layer is instead set to the sum of the time constants in its downstream layers. This produces stronger smoothing in earlier layers, reflecting the cumulative temporal filtering along their longer paths to the output. The gradient-alignment analysis in Fig. 9 shows that this cumulative surrogate improves gradient quality in the earlier layers.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We strove to provide an accurate description of our work in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In the paper, we explicitly investigate how the K-prop gradient approximation quality is degraded by different factors.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The detailed mathematical proofs for K-prop are provided in the appendix. In the main text we provide an intuitive proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the code through GitHub links, where we include all hyperparameters including random seeds needed to reproduce all figures.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide open source code through the GitHub link.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We report all training details either in the main text or in the specific appendix on experimental details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We show error bar with median, 1st and 3rd quartiles in our plots.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We described the training hardware and duration in the appendix section on experimental details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We strictly follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is a highly theoretical work. While it can lead to further developments that can have societal impact, such an analysis would be extremely tentative at this stage and is not worth including.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This work uses standard methods or dataset. There is no risk of misuse that we could assess.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite all datasets used to test our method: GSC, MNIST, MNIST1D datasets, and the OpenAI Gymnasium framework.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not introduce new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We used public LLM agents for issue debugging and paper polishing.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.