NEURAL REAL-TIME RECURRENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent neural networks (RNNs) are commonly trained using backpropagation through time. However, its memory cost grows with sequence length, and it prevents online updates of the RNN parameters. Real-Time Recurrent Learning (RTRL) enables online parameter updates at a fixed memory cost. It maintains an *influence matrix* describing how parameter changes affect the state of the network. Despite these benefits, both its memory and per-step computational costs scale poorly with network width. Prior work has addressed these limitations either by restricting the recurrent architecture to make RTRL tractable or by approximating the influence matrix, notably through low-rank tensor representations. In this work, we introduce Neural Real-Time Recurrent Learning (NRTRL), which learns a compact representation of the influence matrix. NRTRL uses neural networks to update this representation over time and reconstruct the corresponding influence estimate, allowing the approximation mechanism itself to be learned. We instantiate this approach by reusing the RNN cell along a short *virtual trajectory*, with learned low-rank perturbations to its parameters and learned inputs. We show that this construction learns influence estimates that are accurate enough to enable learning. We evaluate NRTRL on pixel-by-pixel MNIST and character-level Penn Treebank.

1 INTRODUCTION

Recurrent neural networks (RNNs) have been successful across a broad range of tasks (Sutskever et al., 2014; Hafner et al., 2019). Recent work has brought substantial advances in recurrent architectures and their training efficiency (Orvieto et al., 2023; Lim et al., 2024). Despite this progress, several difficulties identified decades ago remain unresolved. RNNs are typically trained using Backpropagation Through Time (BPTT; Werbos, 1990), which applies reverse-mode differentiation to recurrent computations unfolded through time. BPTT presents two limitations for learning from long sequences. First, the need to propagate gradients backward through the sequence delays parameter updates, complicating *online learning* (Jaderberg et al., 2017). Second, storing past states for the backward pass makes memory use grow linearly with sequence length (Gruslys et al., 2016).

A common way to address both limitations is Truncated Backpropagation Through Time (TBPTT; Williams & Peng, 1990), which *truncates* the backward computation. However, discarding gradient contributions beyond this horizon introduces bias and limits learning of long-term dependencies (Tallec & Ollivier, 2017; Jaderberg et al., 2017).

Real-Time Recurrent Learning (RTRL; Williams & Zipser, 1989b) offers an alternative that addresses both limitations without truncating the gradient history. It applies forward-mode differentiation to the recurrent computation, accumulating parameter sensitivities in an *influence matrix*. Maintaining this matrix allows parameters to be updated online without a backward pass through the sequence and with memory requirements independent of sequence length. However, its memory use scales cubically and its per-step computational cost scales quartically with network width, making exact RTRL prohibitively expensive for most practical network sizes.

One particular line of work reduces these costs by introducing compact approximations of the influence matrix (Marschall et al., 2020). Unbiased Online Recurrent Optimization (UORO; Tallec & Ollivier, 2018) reduces memory and computation while preserving unbiasedness, but introduces variance that can slow learning. Kronecker-Factored RTRL (KF-RTRL; Mujika et al., 2018) and Optimal Kronecker-Sum Approximation (OK; Benzing et al., 2019) reduce this variance at greater

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

computational cost. SnAp-1 (Menick et al., 2021) instead sparsifies the influence matrix, retaining only sensitivities between parameters and directly affected hidden units.

In this work, we introduce Neural Real-Time Recurrent Learning (NRTRL), which treats influence approximation as a learning problem. Inspired by Sobolev training (Czarnecki et al., 2017), we represent the influence matrix as the Jacobian of a learned function. We propose a tractable architecture that reuses the RNN cell with learned inputs and perturbations to its parameters. These constitute a compact representation that can be several orders of magnitude smaller than the full influence matrix while maintaining estimates that are accurate enough to enable learning. We evaluate NRTRL on pixel-by-pixel MNIST and character-level Penn Treebank.

2 Background and related works

2.1 Background

We consider a recurrent network defined by $\mathbf{h}^{(t)} = F_{\mathbf{w}}(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$, with parameters $\mathbf{w} \in \mathbb{R}^P$, inputs $\mathbf{x}^{(t)} \in \mathbb{R}^{D_x}$, and state $\mathbf{h}^{(t)} \in \mathbb{R}^{D_h}$. The initial state $\mathbf{h}^{(0)}$ is fixed. We denote the loss at time t by $\ell^{(t)}(\mathbf{h}^{(t)})$ and seek to minimize the total loss $\mathcal{L} = \sum_{t=1}^{T} \ell^{(t)}(\mathbf{h}^{(t)})$ with respect to $\mathbf{w}$. We define the *transition Jacobian* $\mathbf{J}^{(t)} := \partial \mathbf{h}^{(t)} / \partial \mathbf{h}^{(t-1)}$, the *immediate influence* $\mathbf{B}^{(t)} := \partial \mathbf{h}^{(t)} / \partial \mathbf{w}^{(t)}$, and the *immediate credit* $\overline{\mathbf{c}}^{(t)} := \partial \ell^{(t)} / \partial \mathbf{h}^{(t)}$. Following Marschall et al. (2020) and Menick et al. (2021), we denote the application of the parameters at time t by $\mathbf{w}^{(t)}$.

Backpropagation through time. BPTT (Werbos, 1990) applies reverse-mode differentiation to the unfolded recurrent computation. It computes the gradient from the *credit assignment vectors* $\mathbf{c}^{(t)} := \partial \mathcal{L} / \partial \mathbf{h}^{(t)}$ and immediate influences as $\partial \mathcal{L} / \partial \mathbf{w} = \sum_{t=1}^{T} (\partial \mathcal{L} / \partial \mathbf{h}^{(t)})(\partial \mathbf{h}^{(t)} / \partial \mathbf{w}^{(t)}) = \sum_{t=1}^{T} \mathbf{c}^{(t)} \mathbf{B}^{(t)}$. A backward recurrence combines immediate and backpropagated credit,

$$\mathbf{c}^{(t)} = \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t+1)}} \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} = \overline{\mathbf{c}}^{(t)} + \mathbf{c}^{(t+1)} \mathbf{J}^{(t+1)}, \tag{1}$$

starting from $\mathbf{c}^{(T)} = \overline{\mathbf{c}}^{(T)}$. For dense recurrent cells, BPTT over T steps scales as $\mathcal{O}(TP)$ in computation and $\mathcal{O}(TD_h)$ in memory to store recurrent states. TBPTT (Williams & Peng, 1990) limits gradient propagation to a fixed number of time steps, reducing memory use but discarding gradient contributions beyond that window. Further details and derivations are provided in Appendix B.

Real-time recurrent learning. RTRL (Williams & Zipser, 1989b) applies forward-mode differentiation to compute the gradient of the current loss for online parameter updates. It maintains the *influence matrix* $\mathbf{M}^{(t)} := \mathrm{d}\mathbf{h}^{(t)} / \mathrm{d}\mathbf{w} \in \mathbb{R}^{D_h \times P}$, which tracks how changes in the parameters affect the hidden state. The gradient of the loss at time t is given by $\frac{\partial \ell^{(t)}}{\partial \mathbf{w}} = \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\mathrm{d}\mathbf{h}^{(t)}}{\mathrm{d}\mathbf{w}} = \overline{\mathbf{c}}^{(t)} \mathbf{M}^{(t)}$, where the influence matrix follows the forward recurrence

$$\mathbf{M}^{(t)} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \frac{\mathrm{d}\mathbf{h}^{(t-1)}}{\mathrm{d}\mathbf{w}} + \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}} = \mathbf{J}^{(t)} \mathbf{M}^{(t-1)} + \mathbf{B}^{(t)} =: \mathcal{T}^{(t)}(\mathbf{M}^{(t-1)}), \tag{2}$$

with $\mathbf{M}^{(0)} = \mathbf{0}$, where $\mathcal{T}^{(t)}(\mathbf{X}) := \mathbf{J}^{(t)} \mathbf{X} + \mathbf{B}^{(t)}$ denotes the affine RTRL update operator. For dense recurrent cells of hidden width H, RTRL requires $\mathcal{O}(H^3)$ memory and $\mathcal{O}(H^4)$ computation per step, independently of sequence length. Derivations and scaling are discussed in Appendix C.

2.2 Related works

Biological plausibility. The need for online updates and a fixed memory cost also arises in biologically plausible learning. Although some motivations differ, e-prop (Bellec et al., 2020) and Random Feedback Local Online Learning (RFLO; Murray, 2019) maintain eligibility traces computed locally at each connection, instead of tracking each weight's influence on every hidden coordinate.

Synthetic gradients. TBPTT allows online updates at a fixed memory cost, but discards gradient contributions beyond the truncation boundary. Decoupled Neural Interfaces (Jaderberg et al., 2017) use a neural network to predict the credit assignment vector $\mathbf{c}^{(t)}$ from the current state. Backpropagating these *synthetic gradients* allows learning beyond this boundary.

Efficient recurrent architectures. One solution to reduce the cost of RTRL is to restrict recurrent connections so that each parameter influences only one hidden coordinate or a small group of coordinates. The transition Jacobian $\mathbf{J}^{(t)}$ is then diagonal or block diagonal and the exact influence matrix $\mathbf{M}^{(t)}$ sparse. Only its nonzero entries need to be stored and updated (Mozer, 1989; Irie et al., 2024; Elelimy et al., 2024).

Approximations of RTRL. Another solution is to approximate the influence matrix using fewer values and update this representation online (Tallec & Ollivier, 2018; Marschall et al., 2020).

3 Approximations to RTRL

3.1 Problem setup

In this work, we extend the family of methods that approximate RTRL. Inspired by the unified perspective of Marschall et al. (2020), we describe the methods used as baselines in this work through a common set of operators. Approximations to RTRL seek to compress the influence matrix $\mathbf{M}^{(t)} \in \mathbb{R}^{D_h \times P}$ from Equation 2 using

$$\widehat{\mathbf{M}}^{(t)} = \mathcal{R}^{(t)}(\mathbf{z}^{(t)}), \qquad \mathbf{z}^{(t)} = \mathcal{U}^{(t)}(\mathbf{z}^{(t-1)}). \tag{3}$$

Here $\mathbf{z}^{(t)} \in \mathcal{Z}$ is an implicit representation of $\mathbf{M}^{(t)}$, which we call the *influence code*. The set $\mathcal{Z}$ of admissible codes is the *code class*. The *decoder* $\mathcal{R}^{(t)}$ maps the code to $\widehat{\mathbf{M}}^{(t)}$, and the *update operator* $\mathcal{U}^{(t)}$ computes the new code from the previous one. The matrices that can be obtained by varying the code form the *decoded class*, $\mathcal{M}^{(t)} := \{\mathcal{R}^{(t)}(\mathbf{z}) : \mathbf{z} \in \mathcal{Z}\}$.

To be consistent with RTRL, updating the code and then decoding it must approximately match applying RTRL to the previous estimate. Using the RTRL operator $\mathcal{T}^{(t)}$ from Equation 2, we write

$$\widehat{\mathbf{M}}^{(t)} = \big(\mathcal{R}^{(t)} \circ \mathcal{U}^{(t)}\big)(\mathbf{z}^{(t-1)}) \approx \mathcal{T}^{(t)}\big(\widehat{\mathbf{M}}^{(t-1)}\big) = \big(\mathcal{T}^{(t)} \circ \mathcal{R}^{(t-1)}\big)(\mathbf{z}^{(t-1)}). \tag{4}$$

The meaning of $\approx$ depends on the method. We discuss the main cases below. The recurrent gradient is estimated as $\partial \ell^{(t)}/\partial \mathbf{w} \approx \bar{\mathbf{c}}^{(t)} \mathcal{R}^{(t)}(\mathbf{z}^{(t)})$, generally without explicitly computing $\widehat{\mathbf{M}}^{(t)}$.

3.2 Existing approximations

Several existing methods can be described using the template introduced in the previous section. We briefly review the main ones below and compare them in Figure 1(a). Further details on their derivations, implementations, and scaling are given in Appendices E and F.

Unbiased stochastic approximations. Unbiased Online Recurrent Optimization (UORO; Tallec & Ollivier, 2018) approximates the influence matrix using a rank-one factorization. It carries two vectors, $\widetilde{\mathbf{h}}^{(t)}$ and $\widetilde{\mathbf{w}}^{(t)}$, as its influence code, with decoder $\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) = \widetilde{\mathbf{h}}^{(t)} \widetilde{\mathbf{w}}^{(t)\top}$. Kronecker-Factored Real-Time Recurrent Learning (KF-RTRL; Mujika et al., 2018) instead uses a Kronecker-product factorization. It carries a vector $\widetilde{\mathbf{u}}^{(t)}$ and a matrix $\widetilde{\mathbf{A}}^{(t)}$, with decoder $\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t)}) = \widetilde{\mathbf{u}}^{(t)\top} \otimes \widetilde{\mathbf{A}}^{(t)}$, where $\otimes$ denotes the Kronecker product. Both methods update the factors directly, without forming the full influence matrix. The new factors are random, but the matrix they represent matches the RTRL update in expectation,

$$\mathbb{E}_{\boldsymbol{\xi}^{(t)}}\big[\mathcal{R}^{(t)}\big(\mathcal{U}^{(t)}\big(\mathbf{z}^{(t-1)}; \boldsymbol{\xi}^{(t)}\big)\big)\big] = \mathcal{T}^{(t)}\big(\mathcal{R}^{(t-1)}\big(\mathbf{z}^{(t-1)}\big)\big), \tag{5}$$

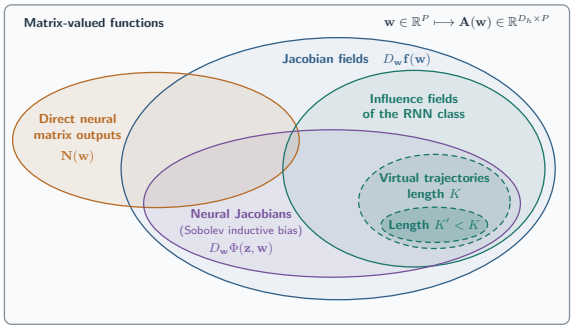
where $\boldsymbol{\xi}^{(t)}$ denotes the randomness used in the update. Each stochastic update can introduce new variance. KF-RTRL reduces this variance by exploiting the recurrent layer's structure, at greater computational cost.

Sparse influence approximation. Sparse 1-step Approximation (SnAp-1; Menick et al., 2021) exploits the sparsity of the immediate influence matrix $\mathbf{B}^{(t)}$. For some dense recurrent layers followed by an elementwise nonlinearity, each weight directly affects only the hidden coordinate it enters, giving $B^{(t)}_{ijk} = \partial h^{(t)}_i / \partial W^{(t)}_{jk} = 0$ for $i \neq j$. SnAp-1 sets those same entries to zero after applying the RTRL recurrence. The retained entries form its influence code. This approximation retains paths of any length within coordinate j but discards paths through other coordinates. Random Feedback

Local Online Learning (RFLO; Murray, 2019) is a biologically motivated method that also enables online updates by relying on a sparse approximation of the influence matrix. We view it as a special case of SnAp-1 and present it in Appendix F.2.

Method	Additional mem. per sequence	Per-step cost	Unbiased
RTRL	$\mathcal{O}(H^3)$	$\mathcal{O}(H^4)$	Yes
UORO	$\mathcal{O}(H^2)$	$\mathcal{O}(H^2)$	Yes
KF-RTRL	$\mathcal{O}(H^2)$	$\mathcal{O}(H^3)$	Yes
OK	$\mathcal{O}(rH^2)$	$\mathcal{O}(rH^3)$	Yes
SnAp-1	$\mathcal{O}(H^2)$	$\mathcal{O}(H^2)$	No
RFLO	$\mathcal{O}(H^2)$	$\mathcal{O}(H^2)$	No
NRTRL	$\mathcal{O}(H)$	$\mathcal{O}(H^2)$	No

(a)



(b)

Figure 1: (a) Costs for a dense recurrent layer with elementwise nonlinearity, hidden width H, and input dimension $\mathcal{O}(H)$. OK uses $r \leq H$ Kronecker terms. The shared learned parameters of NRTRL are excluded from its memory count. See Appendix G.2. (b) Function classes for influence estimation. NRTRL takes the Jacobian of a short virtual trajectory of the recurrent cell.

4 Neural Real-Time Recurrent Learning

The previous section identified three ingredients for approximating RTRL and the influence matrix: 1) a carried representation $\mathbf{z}^{(t)}$, 2) a mechanism to decode that representation using $\mathcal{R}^{(t)}$, and 3) an update operator $\mathcal{U}^{(t)}$ consistent with RTRL. Neural Real-Time Recurrent Learning (NRTRL) treats each of these as a learning problem and uses one neural network for decoding and another for the update operator.

Representing influence through a neural Jacobian Our goal is to approximate the influence matrix using a neural network that decodes a compact influence code $\mathbf{z}^{(t)}$. We denote this network by f_θ and call it the *influence network*. The resulting estimate is $\widehat{\mathbf{M}}^{(t)} = f_\theta(\mathbf{z}^{(t)}, \mathbf{w}) \in \mathbb{R}^{D_h \times P}$. A naive implementation of the influence network would have $D_h P$ outputs, one for each matrix entry. This approach has two limitations. First, both the parameter count and computational cost of a dense output layer scale linearly with the number of outputs, making direct prediction prohibitively expensive. Second, such a matrix-valued network does not necessarily represent the Jacobian of a single function. The influence matrix, however, is precisely a Jacobian, where for a fixed initial state and input history, $\mathbf{M}^{(t)}(\mathbf{w}) = \partial \mathbf{h}^{(t)}(\mathbf{w}) / \partial \mathbf{w}$. Directly predicting the entries produces the right shape without enforcing this property (Czarnecki et al., 2017; Chaudhari et al., 2024).

An alternative is to construct the influence network from the Jacobian of another neural network. This is in the spirit of *Sobolev training*, which trains neural networks by supervising their derivatives (Czarnecki et al., 2017). Let $\Phi_\theta(\mathbf{z}^{(t)}, \boldsymbol{\omega}) \in \mathbb{R}^{D_h}$ take the influence code and the recurrent parameters as inputs. Writing the second input as $\boldsymbol{\omega} \in \mathbb{R}^P$, we define the influence network as the Jacobian with respect to $\boldsymbol{\omega}$, evaluated at $\mathbf{w}$,

$$\widehat{\mathbf{M}}^{(t)} = f_\theta(\mathbf{z}^{(t)}, \mathbf{w}) := \frac{\partial \Phi_\theta(\mathbf{z}^{(t)}, \boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\bigg|_{\boldsymbol{\omega}=\mathbf{w}}. \tag{6}$$

This Jacobian is differentiable and can be trained to approximate $\mathbf{M}^{(t)}$. Following Section 3.1, the Jacobians that can be represented by varying the code form the decoded class $\mathcal{M}_\theta := \{f_\theta(\mathbf{z}, \mathbf{w}) : \mathbf{z} \in \mathcal{Z}\}$. The remaining question is how Φ_θ processes the recurrent parameters $\mathbf{w}$, as naively using a dense input layer would again be prohibitively expensive.

Constructing the influence network from the recurrent cell We seek a network Φ_θ that processes the recurrent parameters while keeping both the number of learned parameters θ and the computational cost small. Additionally, a small influence code $\mathbf{z}^{(t)}$ should be enough to represent accurate

approximations of the true influence matrices. What could a neural network that takes $\boldsymbol{\omega}$ as input and exploits its structure look like? One possible answer is to use the recurrent cell $F_{\boldsymbol{\omega}}$, as it already performs this computation. Hence, reusing it introduces no additional network weights.

Once $\boldsymbol{\omega}$ is supplied to the cell $F_{\boldsymbol{\omega}}(\mathbf{h}, \mathbf{x})$, only the hidden state and input remain to be learned. We can therefore learn a *virtual state* $\mathbf{q}^{(t)}$ and a *virtual input* $\widetilde{\mathbf{x}}^{(t)}$, using them to change the Jacobian of the cell. To give the cell more flexibility, we also add low-rank perturbations to its weight matrices. To keep the code small, we parameterize the perturbation of a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ as

$$\widetilde{\mathbf{W}}^{(t)}(\mathbf{W}) = \mathbf{W} + \mathbf{U}\mathbf{R}^{(t)}\mathbf{V}, \qquad \mathbf{U} \in \mathbb{R}^{m \times r}, \quad \mathbf{R}^{(t)} \in \mathbb{R}^{r \times r}, \quad \mathbf{V} \in \mathbb{R}^{r \times n}. \tag{7}$$

To further reduce the code size, we learn $\mathbf{U}$ and $\mathbf{V}$ as part of $\boldsymbol{\theta}$ and share them across sequences. Thus, the code only needs to store r^2 values for each perturbed weight matrix. Similar constructions have been introduced in prior work (Bershatsky et al., 2024; Bałazy et al., 2024). When biases are present, we also use $\widetilde{\mathbf{b}}^{(t)} = \mathbf{b} + \Delta\mathbf{b}^{(t)}$, with the offsets $\Delta\mathbf{b}^{(t)}$ included in the influence code.

Finally, a single recurrent step may not be sufficient to model the true influence matrix, which accumulates effects from many recurrent steps. We therefore extend the construction to a *virtual trajectory* of K steps, starting from $\widetilde{\mathbf{h}}_0^{(t)} = \mathbf{q}^{(t)}$ and learning an input and perturbation coefficients at each step $\ell = 1, \ldots, K$. The network $\Phi_{\boldsymbol{\theta}}$ returns the final virtual state,

$$\widetilde{\mathbf{h}}_\ell^{(t)} = F_{\widetilde{\boldsymbol{\omega}}_\ell^{(t)}(\boldsymbol{\omega})}\left(\widetilde{\mathbf{h}}_{\ell-1}^{(t)}, \widetilde{\mathbf{x}}_\ell^{(t)}\right), \qquad \Phi_{\boldsymbol{\theta}}(\mathbf{z}^{(t)}, \boldsymbol{\omega}) = \widetilde{\mathbf{h}}_K^{(t)}. \tag{8}$$

The virtual initial state, virtual inputs, and perturbation coefficients form the influence code $\mathbf{z}^{(t)} = \left(\mathbf{q}^{(t)}, (\widetilde{\mathbf{x}}_\ell^{(t)}, \mathbf{R}_\ell^{(t)}, \Delta\mathbf{b}_\ell^{(t)})_{\ell=1}^K\right)$. Figure 1(b) summarizes these choices. Taking the Jacobian of a neural network guarantees that the estimate is a Jacobian. Reusing the recurrent cell makes it an influence matrix of the same architecture.

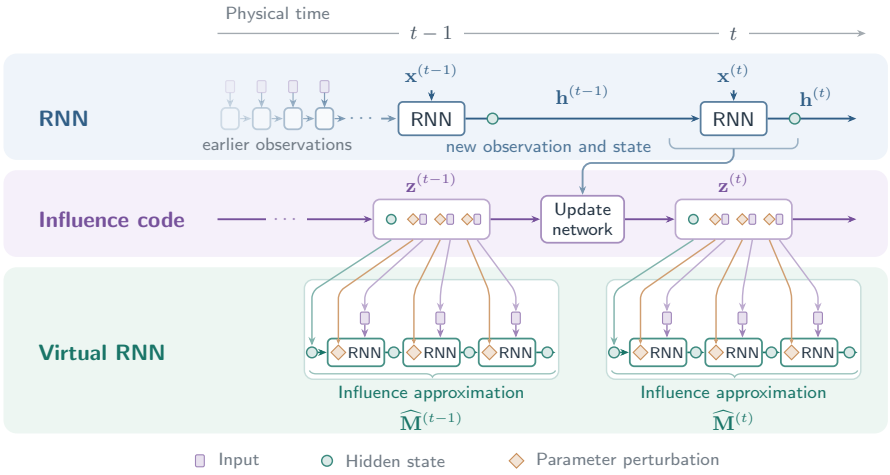


Figure 2: The update network carries the influence code forward. The code is unpacked and used to condition a short virtual trajectory whose parameter Jacobian approximates the influence matrix.

Matching the RTRL recurrence Consistency with RTRL requires the new influence estimate $\widehat{\mathbf{M}}^{(t)}$ to approximately match $\mathcal{T}^{(t)}(\widehat{\mathbf{M}}^{(t-1)})$. We measure their difference using the squared Frobenius norm. However, directly evaluating this loss would require forming the full influence matrices. Following Czarnecki et al. (2017), we instead compare their products with a random vector. For fixed codes and i.i.d. probes $\mathbf{v} \in \mathbb{R}^P$ with $\mathbb{E}[\mathbf{v}\mathbf{v}^\top] = \mathbf{I}_P$, the loss satisfies

$$\ell_{\text{cons}}^{(t)} = \left\| \widehat{\mathbf{M}}^{(t)} - \mathcal{T}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\right) \right\|_F^2 = \mathbb{E}_{\mathbf{v}}\left[\left\| \widehat{\mathbf{M}}^{(t)}\mathbf{v} - \mathbf{J}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\mathbf{v}\right) - \mathbf{B}^{(t)}\mathbf{v} \right\|_2^2 \right]. \tag{9}$$

Averaging over probes gives an unbiased estimate of this loss. The required computations can be performed efficiently using Jacobian–vector products (JVPs) and vector–Jacobian products (VJPs), without forming any full Jacobian (Bradbury et al., 2018).

Amortized solution. In order to avoid repeated iterative optimization of the loss in Equation 9, we amortize this computation using a neural network $U_{\boldsymbol{\eta}}$, which we call the *update network*. At each time step, it predicts the new code as $\mathbf{z}^{(t)} = U_{\boldsymbol{\eta}}(\mathbf{z}^{(t-1)}, \mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}, \mathbf{h}^{(t)})$, implementing $\mathcal{U}^{(t)}$. We jointly train the influence and update networks by minimizing the sampled loss using N_{probe} independent probes $\mathbf{v}_a$, with sg denoting stop-gradient,

$$\underset{\boldsymbol{\theta},\boldsymbol{\eta}}{\arg\min} \ \frac{1}{N_{\text{probe}}} \sum_{a=1}^{N_{\text{probe}}} \left\| f_{\boldsymbol{\theta}}\left(\mathcal{U}_{\boldsymbol{\eta}}^{(t)}(\mathbf{z}^{(t-1)}), \mathbf{w}\right) \mathbf{v}_a - \text{sg}\left(\mathcal{T}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\right) \mathbf{v}_a\right) \right\|_2^2. \tag{10}$$

The stop-gradient makes this a bootstrapped objective, as in temporal-difference learning (Sutton, 1988). As in semi-gradient temporal-difference methods (Sutton & Barto, 2018), gradients do not flow through the target.

Algorithm. NRTRL learns to approximate the influence matrix of the RNN and uses this approximation to train the RNN. The influence network, update network, and RNN are all trained online. Each sequence in a batch carries its own hidden state $\mathbf{h}^{(t)}$ and influence code $\mathbf{z}^{(t)}$, while the parameters $\mathbf{w}, \boldsymbol{\theta}, \boldsymbol{\eta}$ are shared across the batch. At each time step, the RNN processes the new input and the update network predicts a new code, which the influence network decodes into $\widehat{\mathbf{M}}^{(t)}$. Figure 2 illustrates this loop. We train the influence and update networks using the consistency loss in Equation 10, averaged over the batch. Whenever a task loss is observed, we use the approximate gradient $\overline{\mathbf{c}}^{(t)}\widehat{\mathbf{M}}^{(t)}$ to update the recurrent parameters $\mathbf{w}$. Pseudocode is given in Appendix G. Appendix G.2 shows that the influence code only involves the virtual initial state and, at each virtual step, an input, perturbation coefficients, and bias offsets. It thus scales as $\mathcal{O}(H)$ per sequence, at least a factor of H below the other approximations in Figure 1(a), while each time step costs $\mathcal{O}(H^2)$.

5 Analysis and Theory

Error decomposition and recurrence. Compressing the influence matrix and updating its representation over time can introduce errors. The decoder may be unable to represent the true influence matrix exactly, and the update network may not predict a code that minimizes the consistency loss. Fix the recurrent parameters $\mathbf{w}$ and the input sequence. The following decomposition and recurrence apply to all the influence approximations discussed above. For stochastic methods, we consider one realization. Each step introduces a local error between the new estimate and its recurrence target, which we write as $\boldsymbol{\varepsilon}^{(t)} := \widehat{\mathbf{M}}^{(t)} - \mathcal{T}^{(t)}(\widehat{\mathbf{M}}^{(t-1)})$. To distinguish these two sources of error, let $\mathbf{z}_\star^{(t)} \in \arg\min_{\mathbf{z}\in\mathcal{Z}} \|\mathcal{R}^{(t)}(\mathbf{z}) - \mathcal{T}^{(t)}(\widehat{\mathbf{M}}^{(t-1)})\|_F^2$. Then

$$\boldsymbol{\varepsilon}^{(t)} = \left(\mathcal{R}^{(t)}(\mathbf{z}_\star^{(t)}) - \mathcal{T}^{(t)}(\widehat{\mathbf{M}}^{(t-1)})\right) + \left(\widehat{\mathbf{M}}^{(t)} - \mathcal{R}^{(t)}(\mathbf{z}_\star^{(t)})\right) = \boldsymbol{\varepsilon}_{\mathcal{M}}^{(t)} + \boldsymbol{\varepsilon}_{\mathcal{U}}^{(t)}. \tag{11}$$

The first term measures the error that remains between the target and the best prediction allowed by the decoder. A more expressive influence network can reduce this error. The second compares that optimal estimate with the one obtained from the predicted code. Improving the update network or making the code easier to optimize can reduce this error. These errors can accumulate with each new step of the RNN as the influence code is updated. To see this, define the influence error at time t as $\mathbf{E}^{(t)} := \widehat{\mathbf{M}}^{(t)} - \mathbf{M}^{(t)}$. Using the definition of the local error gives the forward recurrence

$$\mathbf{E}^{(t)} = \mathbf{J}^{(t)}\mathbf{E}^{(t-1)} + \boldsymbol{\varepsilon}^{(t)}. \tag{12}$$

Contraction of the RTRL operator. NRTRL minimizes the RTRL consistency loss introduced in Equation 9. A natural question is whether minimizing this loss can recover the true influence matrices, even though its targets are computed from previous estimates. To study this question, we analyze the RTRL operator without assuming a particular decoded class $\mathcal{M}^{(t)}$. For a candidate sequence $\mathbf{X} = (\mathbf{X}^{(0)}, \ldots, \mathbf{X}^{(T)})$, define the RTRL recurrence operator over sequences by $(\mathscr{T}\mathbf{X})^{(t)} := \mathcal{T}^{(t)}(\mathbf{X}^{(t-1)})$ for $t = 1, \ldots, T$, with $(\mathscr{T}\mathbf{X})^{(0)} = \mathbf{X}^{(0)} = \mathbf{0}$. The true influence sequence $\mathbf{M}$ satisfies $\mathscr{T}\mathbf{M} = \mathbf{M}$. We measure differences between sequences using $\|\mathbf{X} - \mathbf{Y}\|_{2,T}^2 := \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{X}^{(t)} - \mathbf{Y}^{(t)}\|_F^2$.

The following proposition shows that repeated applications of $\mathscr{T}$ become contractive under an assumption on the RNN. This assumption concerns its multi-step Jacobians $\mathbf{J}^{(t\leftarrow s)} := \mathbf{J}^{(t)} \cdots \mathbf{J}^{(s+1)}$.

Proposition 1. *Suppose that* $\|\mathbf{J}^{(t\leftarrow s)}\|_2 \le \kappa\rho^{t-s}$ *for all* $0 \le s \le t \le T$*, with* $\kappa \ge 1$ *and* $0 < \rho < 1$*. For any candidate sequences* $\mathbf{X}, \mathbf{Y}$ *initialized at zero, and any integer* $n \ge 0$*,*

$$\|\mathscr{T}^n\mathbf{X} - \mathscr{T}^n\mathbf{Y}\|_{2,T} \le \kappa\rho^n\|\mathbf{X} - \mathbf{Y}\|_{2,T}. \tag{13}$$

Thus, $\mathscr{T}^n$ *is contractive whenever* $\kappa\rho^n < 1$*. Moreover, if* $\mathbf{E}^{(0)} = \mathbf{0}$*, then*

$$\frac{1}{T}\sum_{t=1}^{T}\|\mathbf{E}^{(t)}\|_F^2 \le \frac{\kappa^2}{(1-\rho)^2}\frac{1}{T}\sum_{t=1}^{T}\ell_{\text{cons}}^{(t)}. \tag{14}$$

The proof is given in Appendix H. Under this assumption on the RNN, repeated RTRL updates make the influence estimates converge to the true influence sequence at a rate independent of sequence length. The average consistency loss also bounds the average squared influence error up to a factor independent of sequence length. This gives one setting in which the consistency loss in Equation 9 is a principled training objective when the true influence matrices are unavailable.

What the influence network can represent. To examine the structure of the influence estimate, we consider a simple recurrent layer. Let $\mathbf{W}$ be its parameter matrix, such that

$$\mathbf{g}^{(t)} = \mathbf{W}\mathbf{s}^{(t)} = [\,\mathbf{W}_h\ \mathbf{W}_x\ \mathbf{b}_h\,]\mathbf{s}^{(t)}, \quad \mathbf{h}^{(t)} = \sigma(\mathbf{g}^{(t)}), \quad \mathbf{s}^{(t)} = [\,\mathbf{h}^{(t-1)\top}\ \mathbf{x}^{(t)\top}\ 1\,]^\top. \tag{15}$$

Here H is the hidden size, σ is a nonlinearity applied elementwise, and $\mathbf{W} \in \mathbb{R}^{H\times(D+1)}$, with $D = H + D_x$. Flattening $\mathbf{W}$ by columns gives $\mathbf{w} \in \mathbb{R}^P$, with $P = H(D+1)$. More details are given in Appendix A.

Proposition 2. *Consider the virtual trajectory of Section 4 for the layer above, with additive parameter perturbations. At virtual step* ℓ*, let* $\widetilde{\mathbf{s}}_\ell^{(t)}$ *denote the augmented input formed from the previous virtual state and the virtual input. Let* $\widetilde{\mathbf{g}}_\ell^{(t)}$ *denote the preactivation at that step, and let the Jacobian matrix* $\mathbf{A}_\ell^{(t)} := \partial\Phi_\theta/\partial\widetilde{\mathbf{g}}_\ell^{(t)} \in \mathbb{R}^{H\times H}$ *describe how the final virtual state depends on this preactivation. The influence estimate decomposes as*

$$\widehat{\mathbf{M}}^{(t)} = \sum_{\ell=1}^{K}\widetilde{\mathbf{s}}_\ell^{(t)\top} \otimes \mathbf{A}_\ell^{(t)}. \tag{16}$$

The proof is given in Appendix H.2. Each virtual step contributes one Kronecker product. This connects NRTRL to Optimal Kronecker-Sum Approximation (OK; Benzing et al., 2019), which represents the influence matrix using a sum of r Kronecker products. OK carries the vectors and matrices forming these products, whereas NRTRL represents them implicitly through its influence code and influence network. With the parameters of this network shared across sequences, the goal of NRTRL is to obtain accurate influence estimates using as few code dimensions as possible.

6 Experiments

6.1 Setup

We compare NRTRL with UORO (Tallec & Ollivier, 2018), KF-RTRL (Mujika et al., 2018), and SnAp-1 (Menick et al., 2021) using the GRU (Cho et al., 2014) variant described in Appendix I. All methods share the same architecture and data within each experiment. We optimize all parameters using Adam (Kingma & Ba, 2015). All methods are implemented in JAX (Bradbury et al., 2018).

As NRTRL learns both the update and influence networks, temporal correlations between training samples can make training less stable (Mnih et al., 2016). We therefore include examples from different sequence positions in each minibatch by processing sequences in parallel with temporal offsets. All methods update the recurrent parameters as soon as task losses are observed, including terminal losses, without waiting for the other sequences in the batch to finish.

For NRTRL we use perturbations of rank $r = 2$ and $N_{\text{probe}} = 5$ probes to estimate the consistency loss. The update network is a one-layer transformer described in Appendix G.3.1. Appendix G.3 gives the implementation details and Table 3 summarizes the hyperparameters.

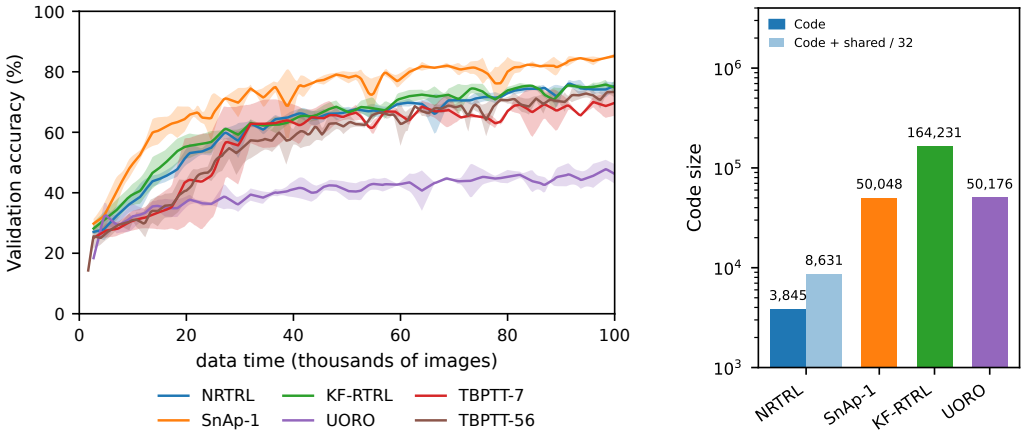


Figure 3: (Left) Validation accuracy on sequential MNIST as a function of the number of training images. The online methods use TBPTT over windows of 7 steps. Curves show the mean over three seeds, and shaded regions show the standard deviation across seeds. (Right) Code size of each influence approximation, on a logarithmic scale. For NRTRL, the lighter bar includes its shared parameters (influence network and update network) divided by the batch size of 32, in addition to the code size.

6.2 Sequential MNIST

Sequential MNIST presents each image one pixel at a time (Arjovsky et al., 2016). Training minimizes the terminal cross-entropy, $\ell^{(784)} = -\log p(y \mid \mathbf{x}^{(1:784)})$, where y is the label.

Figure 3 shows the accuracy on 10,000 validation images, as a function of the number of training images. The influence approximation methods (NRTRL, SnAp-1, KF-RTRL, and UORO) are combined with TBPTT over windows of 7 steps (Appendix I), and we compare them with TBPTT over 7 and 56 steps, two window lengths that divide the 784 steps of each sequence. We use a GRU of width 128, trained with a learning rate of 10^{-4} for all methods.

Despite its simplicity, SnAp-1 learns fastest and reaches the highest accuracy. Notably, NRTRL and KF-RTRL perform similarly, above TBPTT-7 and comparable to TBPTT-56, although NRTRL uses far fewer code dimensions. UORO learns slowest, consistent with the high variance of its rank-one estimates. These results should be read together with the code sizes in Figure 3 (right). The per-sequence memory of SnAp-1 and the other approximations grows quadratically with the hidden size, whereas the code of NRTRL grows linearly. For a fixed memory budget, this could allow larger batch sizes for larger models.

6.3 Character-level Penn Treebank

We evaluate character-level language modelling on the Penn Treebank corpus (Marcus et al., 1993), using a setting similar to Mujika et al. (2018). The network predicts the next character at every step and minimizes the cross-entropy $\ell^{(t)} = -\log p(x^{(t+1)} \mid x^{(1:t)})$.

Figure 4 shows validation bits per character (BPC) as a function of the number of training characters. We compare NRTRL, SnAp-1, and KF-RTRL with TBPTT over 25 steps. We do not include UORO, whose training diverged in this setting. We use a GRU of width 256 but keep the perturbation rank of NRTRL at 2, so the code sizes of SnAp-1 and KF-RTRL grow much faster than that of NRTRL. The GRU is trained with a learning rate of $5 \cdot 10^{-4}$ for all methods, and results are averaged over three seeds.

NRTRL and SnAp-1 reduce BPC much faster than TBPTT-25 early in training. The online methods update the parameters after every character, whereas TBPTT-25 updates them once every 25 characters, so they make many more updates for the same data. Again, NRTRL is competitive with SnAp-1 and outperforms KF-RTRL, despite using far fewer code dimensions.

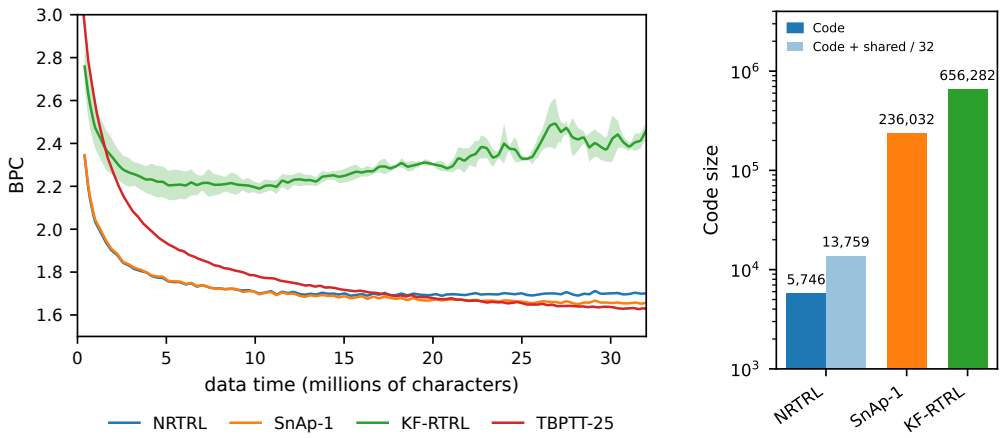


Figure 4: (Left) Validation bits per character (BPC) on character-level Penn Treebank as a function of the number of training characters. Curves show the mean over three seeds, and shaded regions the standard deviation across seeds. (Right) Code size of each influence approximation, on a logarithmic scale. For NRTRL, the lighter bar includes its shared parameters (influence network and update network) divided by the batch size of 32, in addition to the code size.

7 Conclusion

In this work, we introduced NRTRL, a method that allows online training of an RNN by learning to approximate its influence matrix using a compact code. To do so, it reuses the recurrent cell over a short virtual trajectory with learned inputs and low-rank perturbations to the recurrent parameters. The Jacobian of the final virtual state with respect to these parameters provides the influence matrix estimate. An update network maintains the code by matching the RTRL recurrence.

Future work could explore low-rank transformations beyond the additive perturbations used here. A virtual trajectory is one way to compose the recurrent cell, and other constructions may provide more expressive influence networks. Another possibility is to modify only the backward computation through straight-through estimators, while leaving the forward computation unchanged (Bengio et al., 2013). We could also make the influence matrices easier to approximate by choosing suitable recurrent architectures or regularizing the RNN. The RTRL consistency objective is blind to how the influence approximation is used, as it does not account for the credit assigned to the hidden state by the task loss. Using this credit could prioritize accuracy where it most affects the task gradient.

Our formulation leaves open how the influence network processes the recurrent parameters. Developing efficient architectures for this purpose could draw on work on learning from neural network weights (Zhou et al., 2023; Herrmann et al., 2024). NRTRL is biased. Future work could develop unbiased variants or combine it with the unbiased methods discussed here. Scaling to larger models and longer sequences is another direction.

AI Use Statement

We used large language models to assist with formulating and refining mathematical claims, developing proofs, and discussing research hypotheses. We used them to discuss experimental methodology and assist with implementation. They also assisted with identifying and interpreting related work, revising the manuscript, and preparing figures. The authors take responsibility for the final content.

References

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Re-*

search, pp. 1120–1128. PMLR, 2016. URL https://proceedings.mlr.press/v48/arjovsky16.html.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. In *ECAI 2025: 28th European Conference on Artificial Intelligence 25-30 October 2025, Bologna, Italy-Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025) Proceedings*, pp. 3194–3201. SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2024.

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.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Frederik Benzing, Marcelo Matheus Gauy, Asier Mujika, Anders Martinsson, and Angelika Steger. Optimal Kronecker-sum approximation of real time recurrent learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 604–613. PMLR, 2019. URL https://proceedings.mlr.press/v97/benzing19a.html.

Daniel Bershatsky, Daria Cherniuk, Talgat Daulbaev, Aleksandr Mikhalev, and Ivan Oseledets. Lotr: Low tensor rank weight adaptation. *arXiv preprint arXiv:2402.01376*, 2024.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Shreyas Chaudhari, Srinivasa Pranav, and José MF Moura. Gradient networks. *IEEE Transactions on Signal Processing*, 73:324–339, 2024.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179/.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *Advances in neural information processing systems*, 30, 2017.

Esraa Elelimy, Adam White, Michael Bowling, and Martha White. Real-time recurrent learning using trace units in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 17006–17043. Curran Associates, Inc., 2024. doi: 10.52202/079017-0541. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1e616bde0438cb10cb6adf076ae7d336-Paper-Conference.pdf.

Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. *Advances in neural information processing systems*, 29, 2016.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2026. URL http://github.com/google/flax.

Vincent Herrmann, Francesco Faccio, and Jürgen Schmidhuber. Learning useful representations of recurrent neural network weight matrices. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 18205–18227. PMLR, 2024. URL https://proceedings.mlr.press/v235/herrmann24a.html.

Kazuki Irie, Anand Gopalakrishnan, and Jürgen Schmidhuber. Exploring the promise and limits of real-time recurrent learning. In *International Conference on Learning Representations*, volume 2024, pp. 27575–27598, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/74aec30590e07dbe2e29879f9df14fb2-Paper-Conference.pdf.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1627–1635. PMLR, 2017. URL https://proceedings.mlr.press/v70/jaderberg17a.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah. Parallelizing non-linear sequential models over the sequence length. In *International Conference on Learning Representations*, volume 2024, pp. 55334–55360, 2024.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL https://aclanthology.org/J93-2004/.

Owen Marschall, Kyunghyun Cho, and Cristina Savin. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of machine learning research*, 21(135): 1–34, 2020.

Jacob Menick, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves. Practical real time recurrent learning with a sparse approximation. In *International Conference on Learning Representations*, 2021.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mniha16.html.

Michael C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3(4):349–381, 1989. URL https://content.wolfram.com/uploads/sites/13/2018/02/03-4-4.pdf.

Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/dba132f6ab6a3e3d17a8d59e82105f4c-Abstract.html.

James M Murray. Local online learning in recurrent networks with random feedback. *Elife*, 8: e43299, 2019.

Yann Ollivier, Corentin Tallec, and Guillaume Charpiat. Training recurrent networks online without backtracking. *arXiv preprint arXiv:1507.07680*, 2015.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International conference on machine learning*, pp. 26670–26698. PMLR, 2023.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. Technical Report ICS Report 8506, Institute for Cognitive Science, University of California, San Diego, 1985.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988. doi: 10.1007/BF00115009. URL https://doi.org/10.1007/BF00115009.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018. ISBN 978-0-262-03924-6. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Corentin Tallec and Yann Ollivier. Unbiased Online Recurrent Optimization. In *International Conference on Learning Representations*, 2018. URL https://mlanthology.org/iclr/2018/tallec2018iclr-unbiased/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990. doi: 10.1162/neco.1990.2.4.490.

Ronald J. Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989a. doi: 10.1080/09540098908915631.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989b. doi: 10.1162/neco.1989.1.2.270.

Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. In *Advances in Neural Information Processing Systems*, volume 36, pp. 24966–24992. Curran Associates, Inc., 2023. doi: 10.52202/075280-1085. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/4e9d8aeeab6120c3c83ccf95d4c211d3-Paper-Conference.pdf.

Appendix

A A canonical recurrent layer **14**

A.1 Costs of elementary operations 14

B Backpropagation Through Time (BPTT) **15**

B.1 Backward recurrence 15

B.2 Scaling 15

B.3 Truncated Backpropagation Through Time (TBPTT) 16

C Real-Time Recurrent Learning (RTRL) **16**

C.1 Forward recurrence 16

C.2 Structure of the influence matrix 17

C.3 Scaling 18

D RTRL Approximations as Tensor Factorizations **18**

E Stochastic approximation baselines **19**

E.1 A unifying view 19

E.2 Base results and definitions 20

E.3 Unbiased Online Recurrent Optimization (UORO) 22

E.4 Kronecker-Factored RTRL (KF-RTRL) 27

E.5 Optimal Kronecker-Sum Approximation (OK) 32

F Deterministic approximation baselines **35**

F.1 Sparse n-step Approximation (SnAp) 35

F.2 Random Feedback Local Online Learning (RFLO) 38

G Neural Real-Time Recurrent Learning (NRTRL) **39**

G.1 Implementation 40

G.2 Memory and computation 40

G.3 Empirical details 42

H Theoretical results **44**

H.1 Contraction of RTRL operator 44

H.2 Factorization of the influence matrix 47

I Experiments **48**

A A CANONICAL RECURRENT LAYER

Inspired by the setup of Marschall et al. (2020), we use a single recurrent layer parameterized by one weight matrix, including its bias, for the derivations below. Let $\mathbf{h}^{(t)} \in \mathbb{R}^H$ be the hidden state and $\mathbf{x}^{(t)} \in \mathbb{R}^{D_x}$ the external input. We collect the previous hidden state, current input, and a constant bias coordinate into

$$\mathbf{s}^{(t)} := \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \\ 1 \end{bmatrix} \in \mathbb{R}^{D+1}, \qquad D := H + D_x. \tag{17}$$

We collect the recurrent weights $\mathbf{W}_h \in \mathbb{R}^{H\times H}$, input weights $\mathbf{W}_x \in \mathbb{R}^{H\times D_x}$, and biases $\mathbf{b}_h \in \mathbb{R}^H$ into $\mathbf{W} = [\mathbf{W}_h\ \mathbf{W}_x\ \mathbf{b}_h] \in \mathbb{R}^{H\times(D+1)}$, with $P = H(D+1)$ parameters. The layer is written as

$$\mathbf{g}^{(t)} = \mathbf{W}\mathbf{s}^{(t)} = \mathbf{W}_h\mathbf{h}^{(t-1)} + \mathbf{W}_x\mathbf{x}^{(t)} + \mathbf{b}_h, \qquad \mathbf{h}^{(t)} = \sigma\left(\mathbf{g}^{(t)}\right), \tag{18}$$

where σ is applied element-wise.

A.1 COSTS OF ELEMENTARY OPERATIONS

We discuss the costs of the basic differentiation operators needed for the different baselines in this work, using the notation of Section 2. Table 1 summarizes the results for the recurrent layer above. The forward quantities $\mathbf{s}^{(t)}$ and $\mathbf{g}^{(t)}$ are assumed available.

Table 1: Costs of elementary differentiation operations for the recurrent layer in Eq. 18.

Operation	Product	Cost
State JVP	$\mathbf{J}^{(t)}\mathbf{v}$	$\mathcal{O}(H^2)$
Multiplication by the state Jacobian	$\mathbf{J}^{(t)}\mathbf{A}$	$\mathcal{O}(kH^2)$
State VJP	$\mathbf{J}^{(t)\top}\mathbf{u}$	$\mathcal{O}(H^2)$
Parameter JVP	$\mathbf{B}^{(t)}\mathbf{v}$	$\mathcal{O}(P)$
Parameter VJP	$\mathbf{B}^{(t)\top}\mathbf{u}$	$\mathcal{O}(P)$

State JVP. For a state direction $\mathbf{v} \in \mathbb{R}^H$,

$$\mathbf{J}^{(t)}\mathbf{v} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}}\mathbf{v} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{g}^{(t)}}\frac{\partial \mathbf{g}^{(t)}}{\partial \mathbf{h}^{(t-1)}}\mathbf{v} = \operatorname{diag}\left(\sigma'(\mathbf{g}^{(t)})\right)\mathbf{W}_h\mathbf{v} = \sigma'(\mathbf{g}^{(t)}) \odot (\mathbf{W}_h\mathbf{v}). \tag{19}$$

The matrix–vector product costs $\mathcal{O}(H^2)$ and the elementwise operations cost $\mathcal{O}(H)$, giving $\mathcal{O}(H^2)$ overall.

Multiplication by the state Jacobian. For $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_k] \in \mathbb{R}^{H\times k}$,

$$\mathbf{J}^{(t)}\mathbf{A} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}}\mathbf{A} = \begin{bmatrix} \mathbf{J}^{(t)}\mathbf{a}_1 & \cdots & \mathbf{J}^{(t)}\mathbf{a}_k \end{bmatrix}. \tag{20}$$

Each column is a state JVP costing $\mathcal{O}(H^2)$, so the k columns cost $\mathcal{O}(kH^2)$ in total.

State VJP. For an output weighting vector $\mathbf{u} \in \mathbb{R}^H$,

$$\mathbf{J}^{(t)\top}\mathbf{u} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}}^{\top}\mathbf{u} = \frac{\partial \mathbf{g}^{(t)}}{\partial \mathbf{h}^{(t-1)}}^{\top}\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{g}^{(t)}}^{\top}\mathbf{u} = \mathbf{W}_h^{\top}\operatorname{diag}\left(\sigma'(\mathbf{g}^{(t)})\right)\mathbf{u} = \mathbf{W}_h^{\top}\left(\sigma'(\mathbf{g}^{(t)}) \odot \mathbf{u}\right). \tag{21}$$

The elementwise operations cost $\mathcal{O}(H)$, followed by a matrix–vector product costing $\mathcal{O}(H^2)$. The total is again $\mathcal{O}(H^2)$.

Parameter JVP. For a parameter direction $\mathbf{v} \in \mathbb{R}^P$, let $\mathbf{V} \in \mathbb{R}^{H\times(D+1)}$ satisfy $\operatorname{vec}(\mathbf{V}) = \mathbf{v}$, using the same vectorization as for $\mathbf{w} = \operatorname{vec}(\mathbf{W})$. Holding $\mathbf{s}^{(t)}$ fixed gives

$$\mathbf{B}^{(t)}\mathbf{v} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}}\mathbf{v} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{g}^{(t)}}\frac{\partial \mathbf{g}^{(t)}}{\partial \mathbf{w}^{(t)}}\mathbf{v} = \operatorname{diag}\left(\sigma'(\mathbf{g}^{(t)})\right)\mathbf{V}\mathbf{s}^{(t)} = \sigma'(\mathbf{g}^{(t)}) \odot (\mathbf{V}\mathbf{s}^{(t)}). \tag{22}$$

The matrix–vector product costs $\mathcal{O}(H(D+1)) = \mathcal{O}(P)$, and the elementwise operations cost $\mathcal{O}(H)$. The total is $\mathcal{O}(P)$.

Parameter VJP. For an output weighting vector $\mathbf{u} \in \mathbb{R}^H$, holding $\mathbf{s}^{(t)}$ fixed gives

$$\mathbf{B}^{(t)\top}\mathbf{u} = \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}}\right)^\top \mathbf{u} = \left(\frac{\partial \mathbf{g}^{(t)}}{\partial \mathbf{w}^{(t)}}\right)^\top \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{g}^{(t)}}\right)^\top \mathbf{u} = \mathrm{vec}\Big[\big(\sigma'(\mathbf{g}^{(t)}) \odot \mathbf{u}\big)\mathbf{s}^{(t)\top}\Big]. \tag{23}$$

The elementwise operations cost $\mathcal{O}(H)$ and the outer product produces P entries at cost $\mathcal{O}(P)$. The total is $\mathcal{O}(P)$.

B Backpropagation Through Time (BPTT)

Backpropagation Through Time (BPTT; Rumelhart et al., 1985; Werbos, 1990) is the canonical method for computing gradients in recurrent neural networks. It applies reverse-mode differentiation to the recurrent computation unfolded through time, which scales well to the common setting where the gradient of a scalar loss is required with respect to a large number of parameters.

B.1 Backward recurrence

Consider the recurrent dynamics

$$\mathbf{h}^{(t)} = F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right), \qquad \mathbf{h}^{(t)} \in \mathbb{R}^{D_h}, \qquad \mathbf{w} \in \mathbb{R}^P. \tag{24}$$

The notation $\mathbf{w}^{(t)}$ denotes the application of the shared recurrent parameters at physical time t. For the total loss $\mathcal{L} = \sum_t \ell^{(t)}$, the gradient with respect to the shared parameters decomposes over their applications,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_t \frac{\partial \mathcal{L}}{\partial \mathbf{w}^{(t)}} = \sum_t \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}}. \tag{25}$$

The two factors are the *credit assignment vector* and the *immediate influence*,

$$\mathbf{c}^{(t)} := \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t)}} \in \mathbb{R}^{1\times D_h}, \qquad \mathbf{B}^{(t)} := \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}} \in \mathbb{R}^{D_h \times P}. \tag{26}$$

The total recurrent gradient is therefore

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_t \mathbf{c}^{(t)}\mathbf{B}^{(t)}.} \tag{27}$$

The hidden state $\mathbf{h}^{(t)}$ affects the total loss directly through $\ell^{(t)}$ and through the next hidden state $\mathbf{h}^{(t+1)}$. Applying the chain rule gives

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t)}} = \underbrace{\frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}}}_{\bar{\mathbf{c}}^{(t)}} + \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t+1)}}}_{\mathbf{c}^{(t+1)}} \underbrace{\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}}}_{\mathbf{J}^{(t+1)}}. \tag{28}$$

Identifying the left-hand side as $\mathbf{c}^{(t)}$ gives the BPTT recurrence

$$\boxed{\mathbf{c}^{(t)} = \bar{\mathbf{c}}^{(t)} + \mathbf{c}^{(t+1)}\mathbf{J}^{(t+1)}.} \tag{29}$$

The *immediate credit assignment vector* $\bar{\mathbf{c}}^{(t)}$ contains the contribution of the current loss, while the second term propagates future credit backward through the recurrent transition.

B.2 Scaling

Stored trajectory. Equation (29) is evaluated backward through the sequence and requires the local recurrent transition at each timestep. The forward states $\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(T)}$ and the corresponding

intermediate activations must therefore remain available until their backward steps are evaluated, giving $\mathcal{O}(TD_h)$ memory per sequence.

Per-step computation. For the recurrent layer in Section A, each backward step computes the state VJP $\mathbf{c}^{(t)}\mathbf{J}^{(t)}$ and the parameter VJP $\mathbf{c}^{(t)}\mathbf{B}^{(t)}$, costing $\mathcal{O}(H^2)$ and $\mathcal{O}(P)$, respectively, as derived in Section A.1. Including credit addition and gradient accumulation, the per-step cost is $\mathcal{O}(H^2+P) = \mathcal{O}(P)$, since $P = H(D+1)$ and $D \geq H$. Across T steps, the total backward computation is $\mathcal{O}(TP)$. For $D = \mathcal{O}(H)$, the storage for recurrent states and total backward computation scale as $\mathcal{O}(TH)$ and $\mathcal{O}(TH^2)$, respectively.

B.3 Truncated Backpropagation Through Time (TBPTT)

Truncated Backpropagation Through Time (TBPTT), introduced by Williams & Peng (1990), limits the backward propagation of credit to a finite temporal window. Repeatedly applying Equation (29) expresses the credit assignment vector as

$$\begin{aligned}\mathbf{c}^{(t)} &= \overline{\mathbf{c}}^{(t)} + \overline{\mathbf{c}}^{(t+1)}\mathbf{J}^{(t+1)} + \overline{\mathbf{c}}^{(t+2)}\mathbf{J}^{(t+2)}\mathbf{J}^{(t+1)} + \cdots \\ &= \sum_{\tau=0}^{T-t} \overline{\mathbf{c}}^{(t+\tau)}\mathbf{J}^{(t+\tau)}\mathbf{J}^{(t+\tau-1)}\cdots\mathbf{J}^{(t+1)},\end{aligned} \tag{30}$$

where the product is the identity for $\tau = 0$ and T denotes the final timestep.

For a truncation window of L timesteps, the credit assignment vector is approximated by

$$\mathbf{c}_L^{(t)} := \sum_{\tau=0}^{\min(L-1,T-t)} \overline{\mathbf{c}}^{(t+\tau)}\mathbf{J}^{(t+\tau)}\mathbf{J}^{(t+\tau-1)}\cdots\mathbf{J}^{(t+1)}. \tag{31}$$

The resulting recurrent gradient approximation is

$$\boxed{\widetilde{\mathbf{g}}_{\mathrm{TBPTT},w} = \sum_t \mathbf{c}_L^{(t)}\mathbf{B}^{(t)}.} \tag{32}$$

With $L = 1$, only $\overline{\mathbf{c}}^{(t)}$ is retained. Larger windows include longer recurrent paths, recovering full BPTT when they cover the sequence. The BPTT costs above apply with T replaced by L, giving $\mathcal{O}(LD_h)$ storage for recurrent states and $\mathcal{O}(LP)$ backward computation per window, or $\mathcal{O}(LH^2)$ computation when $D = \mathcal{O}(H)$.

C Real-Time Recurrent Learning (RTRL)

Real-Time Recurrent Learning (RTRL) was introduced by Williams & Zipser (1989b) as an exact online method for differentiating recurrent neural networks, with a companion experimental study by Williams & Zipser (1989a). Rather than storing and revisiting the recurrent trajectory, RTRL carries forward the sensitivity of the current hidden state to the shared recurrent parameters.

C.1 Forward recurrence

Consider the recurrent dynamics

$$\mathbf{h}^{(t)} = F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right), \qquad \mathbf{h}^{(t)} \in \mathbb{R}^{D_h}, \qquad \mathbf{w} \in \mathbb{R}^P. \tag{33}$$

The same recurrent parameters $\mathbf{w}$ are applied at every physical timestep, and we denote their particular application at time t' by $\mathbf{w}^{(t')}$. For the total loss $\mathcal{L} = \sum_t \ell^{(t)}$, the gradient with respect to the

shared recurrent parameters decomposes as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \sum_t \frac{\partial \ell^{(t)}}{\partial \mathbf{w}} = \sum_t \sum_{t'} \frac{\partial \ell^{(t)}}{\partial \mathbf{w}^{(t')}} = \sum_t \sum_{t' \le t} \frac{\partial \ell^{(t)}}{\partial \mathbf{w}^{(t')}} \\ &= \sum_t \sum_{t' \le t} \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t')}} = \sum_t \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}} \left(\sum_{t' \le t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t')}} \right) \\ &= \sum_t \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}}. \end{aligned} \tag{34}$$

The two factors in the last expression are the *immediate credit* and the *influence matrix*,

$$\bar{\mathbf{c}}^{(t)} := \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}} \in \mathbb{R}^{1 \times D_h}, \qquad \mathbf{M}^{(t)} := \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}} \in \mathbb{R}^{D_h \times P}. \tag{35}$$

The total recurrent gradient is therefore

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_t \bar{\mathbf{c}}^{(t)} \mathbf{M}^{(t)}. \tag{36}$$

The current hidden state depends on $\mathbf{w}$ through the previous hidden state and through the current parameter application $\mathbf{w}^{(t)}$. Applying the chain rule and using $\partial \mathbf{w}^{(t)} / \partial \mathbf{w} = \mathbf{I}_P$ gives

$$\begin{aligned}\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}} &= \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{w}} + \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}} \frac{\partial \mathbf{w}^{(t)}}{\partial \mathbf{w}} \\ &= \underbrace{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}}}_{\mathbf{J}^{(t)}} \underbrace{\frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{w}}}_{\mathbf{M}^{(t-1)}} + \underbrace{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}}}_{\mathbf{B}^{(t)}}. \end{aligned} \tag{37}$$

Here $\mathbf{J}^{(t)} \in \mathbb{R}^{D_h \times D_h}$ is the *transition Jacobian* and $\mathbf{B}^{(t)} \in \mathbb{R}^{D_h \times P}$ is the *immediate influence*. Identifying the left-hand side as $\mathbf{M}^{(t)}$ gives the RTRL recurrence

$$\boxed{\mathbf{M}^{(t)} = \mathbf{J}^{(t)} \mathbf{M}^{(t-1)} + \mathbf{B}^{(t)}.} \tag{38}$$

The first term propagates the previously accumulated parameter sensitivities through the current recurrent transition, while the second adds the sensitivity introduced by the current parameter application.

At each physical timestep, the recurrent gradient contribution is obtained directly from the carried influence matrix,

$$\boxed{\mathbf{g}_w^{(t)} = \frac{\partial \ell^{(t)}}{\partial \mathbf{w}} = \bar{\mathbf{c}}^{(t)} \mathbf{M}^{(t)}.} \tag{39}$$

C.2 Structure of the influence matrix

The rows of $\mathbf{M}^{(t)}$ correspond to hidden-state coordinates and its columns correspond to recurrent parameters,

$$\mathbf{M}^{(t)} = \begin{bmatrix} \frac{\partial h_1^{(t)}}{\partial w_1} & \frac{\partial h_1^{(t)}}{\partial w_2} & \cdots & \frac{\partial h_1^{(t)}}{\partial w_P} \\ \frac{\partial h_2^{(t)}}{\partial w_1} & \frac{\partial h_2^{(t)}}{\partial w_2} & \cdots & \frac{\partial h_2^{(t)}}{\partial w_P} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_{D_h}^{(t)}}{\partial w_1} & \frac{\partial h_{D_h}^{(t)}}{\partial w_2} & \cdots & \frac{\partial h_{D_h}^{(t)}}{\partial w_P} \end{bmatrix} \in \mathbb{R}^{D_h \times P}. \tag{40}$$

Each column therefore records how one recurrent parameter affects the complete current hidden state.

This column organization directly reflects any decomposition of the recurrent parameters. Suppose the recurrent cell contains N_{modules} parameterized submodules and write

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{N_{\text{modules}}} \end{bmatrix}, \qquad \mathbf{w}_k \in \mathbb{R}^{P_k}, \qquad P = \sum_{k=1}^{N_{\text{modules}}} P_k. \tag{41}$$

The influence matrix decomposes into the corresponding column blocks,

$$\boxed{\mathbf{M}^{(t)} = \left[\begin{array}{c|c|c|c} \underbrace{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}_1}}_{\mathbf{M}_1^{(t)} \in \mathbb{R}^{D_h \times P_1}} & \underbrace{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}_2}}_{\mathbf{M}_2^{(t)} \in \mathbb{R}^{D_h \times P_2}} & \cdots & \underbrace{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}_{N_{\text{modules}}}}}_{\mathbf{M}_{N_{\text{modules}}}^{(t)} \in \mathbb{R}^{D_h \times P_{N_{\text{modules}}}}} \end{array}\right].} \tag{42}$$

The immediate influence has the same decomposition,

$$\mathbf{B}^{(t)} = \left[\mathbf{B}_1^{(t)} \mid \mathbf{B}_2^{(t)} \mid \cdots \mid \mathbf{B}_{N_{\text{modules}}}^{(t)}\right]. \tag{43}$$

The parameter decomposition of the recurrent cell therefore induces the same column decomposition of the influence matrix. Each parameter block follows

$$\mathbf{M}_k^{(t)} = \mathbf{J}^{(t)} \mathbf{M}_k^{(t-1)} + \mathbf{B}_k^{(t)}, \qquad k = 1, \ldots, N_{\text{modules}}. \tag{44}$$

The same transition Jacobian $\mathbf{J}^{(t)}$ acts on every block, so parameters from any submodule may affect all hidden coordinates.

C.3 Scaling

Persistent state. Exact RTRL carries $\mathbf{M}^{(t)} \in \mathbb{R}^{D_h \times P}$, requiring $D_h P$ additional scalars per sequence. For the recurrent layer in Section A, $P = H(D+1)$, giving $\mathcal{O}(H^3)$ storage when $D = \mathcal{O}(H)$.

Per-step computation. Each step requires three operations. 1) *Influence propagation*, $\mathbf{J}^{(t)}\mathbf{M}^{(t-1)}$, applies P state JVPs at a cost of $\mathcal{O}(H^2 P)$, as derived in Section A.1. 2) *Immediate influence addition*, $\mathbf{M}^{(t)} = \mathbf{J}^{(t)}\mathbf{M}^{(t-1)} + \mathbf{B}^{(t)}$, costs $\mathcal{O}(HP)$ once $\mathbf{B}^{(t)}$ is available. 3) *Gradient computation*, $\mathbf{g}_w^{(t)} = \bar{\mathbf{c}}^{(t)}\mathbf{M}^{(t)}$, multiplies $\bar{\mathbf{c}}^{(t)}$ by $\mathbf{M}^{(t)}$ at cost $\mathcal{O}(HP)$. The total per-step computation is therefore $\mathcal{O}(H^2 P)$, or $\mathcal{O}(H^4)$ when $D = \mathcal{O}(H)$.

D RTRL Approximations as Tensor Factorizations

The stochastic approximations in Section E and the deterministic approximations in Section F can be compared more directly by exposing the natural matrix indices of the recurrent parameters. We follow the tensor decomposition introduced by Marschall et al. (2020), adapting their notation to the recurrent-layer convention used throughout this appendix. This section defines the common unit of discussion.

Exposing the parameter indices. Consider the recurrent layer defined in Section A, with $\mathbf{W} \in \mathbb{R}^{H \times (D+1)}$ and $P = H(D+1)$. Let $\mathbf{w}$ denote the vectorization of $\mathbf{W}$. The scalar entries W_{jk} and w_p refer to the same parameter under the chosen flattening order; the three-index influence $M_{ijk}^{(t)}$ retains the capital letter of its matrix view. Its RTRL influence matrix is

$$\mathbf{M}^{(t)} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}} \in \mathbb{R}^{H \times P}. \tag{45}$$

Following the decomposition of Marschall et al. (2020), we expose the two parameter indices of $\mathbf{W}$ and write

$$\boxed{M_{ijk}^{(t)} := \frac{\partial h_i^{(t)}}{\partial W_{jk}}.} \tag{46}$$

This is the same influence matrix with its parameter dimension reshaped,

$$\mathbf{M}^{(t)} \in \mathbb{R}^{H\times H(D+1)} \qquad \longleftrightarrow \qquad \left[M^{(t)}_{ijk}\right] \in \mathbb{R}^{H\times H\times (D+1)}. \tag{47}$$

The index i identifies the current hidden coordinate, while j and k identify respectively the output and input coordinates of $\mathbf{W}$, with $k = D + 1$ corresponding to the bias.

For the layer in Equation (18), the immediate influence of the current application of $\mathbf{W}$ is

$$\boxed{B^{(t)}_{ijk} = \frac{\partial h^{(t)}_i}{\partial W^{(t)}_{jk}} = \delta_{ij}\sigma'\left(g^{(t)}_i\right) s^{(t)}_k.} \tag{48}$$

E Stochastic approximation baselines

This section presents the stochastic RTRL approximations used as baselines in this work. We begin with a unifying view adapted from Benzing et al. (2019) in Section E.1, followed by the definitions and results shared by these methods in Section E.2. We then present UORO (Tallec & Ollivier, 2018) in Section E.3, KF-RTRL (Mujika et al., 2018) in Section E.4, and OK (Benzing et al., 2019) in Section E.5. Each presentation includes supporting results, a derivation of the update rule, and an analysis of memory and computational costs.

E.1 A unifying view

As observed by Benzing et al. (2019), some unbiased stochastic methods can be decomposed into an encoding step (their Ap) followed by a combination step (their Mix). This allows us to discuss the unbiasedness of these approximations as well as their variance. Here we reintroduce these ideas using our notation, which will be useful for understanding the methods presented below.

Encoding and combination. Using the previous influence estimate $\widehat{\mathbf{M}}^{(t-1)} = \mathcal{R}^{(t-1)}(\mathbf{z}^{(t-1)})$ in the affine RTRL recurrence gives

$$\mathcal{T}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\right) = \mathbf{J}^{(t)}\widehat{\mathbf{M}}^{(t-1)} + \mathbf{B}^{(t)}. \tag{49}$$

This target need not belong to the decoded class $\mathcal{M}^{(t)}$. There may be no code $\mathbf{z} \in \mathcal{Z}$ that decodes to it. First, the immediate influence $\mathbf{B}^{(t)}$ is encoded in the same format as the previous estimate, exactly or through an unbiased approximation. This *encoding step* (Encode) usually leaves the propagated term unchanged. Their sum may still lie outside $\mathcal{M}^{(t)}$. The *combination step* $(\mathrm{Combine})$ gives an unbiased estimate of the sum in the same format. Either step may use randomness, denoted by ξ_{enc} for encoding and ξ_{comb} for combination. Thus,

$$\begin{aligned}\widehat{\mathbf{M}}^{(t)} &= \left(\mathcal{R}^{(t)} \circ \mathcal{U}^{(t)}\right)(\mathbf{z}^{(t-1)}) \\ &= \mathrm{Combine}\left(\mathrm{Encode}\left(\mathbf{J}^{(t)}\mathcal{R}^{(t-1)}(\mathbf{z}^{(t-1)})\right), \mathrm{Encode}\left(\mathbf{B}^{(t)}\right)\right).\end{aligned} \tag{50}$$

The new influence code $\mathbf{z}^{(t)}$ is the representation produced by the combination step.

Unbiasedness. As pointed out in UORO (Tallec & Ollivier, 2018), unbiased gradient estimates are a key assumption behind convergence guarantees for stochastic gradient descent. Unbiasedness follows when the encoding and combination steps are unbiased for each contribution and their sum, respectively.

$$\mathbb{E}_{\xi_{\mathrm{enc}}}[\mathrm{Encode}(\mathbf{X})] = \mathbf{X}, \qquad \mathbb{E}_{\xi_{\mathrm{comb}}}[\mathrm{Combine}(\mathbf{X}, \mathbf{Y})] = \mathbf{X} + \mathbf{Y}. \tag{51}$$

The first expectation averages over encoding randomness while $\mathbf{X}$ stays fixed. The second averages over combination randomness while $\mathbf{X}$, $\mathbf{Y}$, and their stored representations stay fixed. For fixed incoming code $\mathbf{z}$ and current matrices $\mathbf{J}^{(t)}$ and $\mathbf{B}^{(t)}$, averaging first over combination randomness and then over encoding randomness gives

$$\begin{aligned}\mathbb{E}_{\xi^{(t)}}\left[\left(\mathcal{R}^{(t)} \circ \mathcal{U}^{(t)}\right)(\mathbf{z})\right] &= \mathbf{J}^{(t)}\mathcal{R}^{(t-1)}(\mathbf{z}) + \mathbf{B}^{(t)} \\ &= \left(\mathcal{T}^{(t)} \circ \mathcal{R}^{(t-1)}\right)(\mathbf{z}),\end{aligned} \tag{52}$$

where $\xi^{(t)} = (\xi^{(t)}_{\text{enc}}, \xi^{(t)}_{\text{comb}})$ collects all random draws used for encoding and combination at step t. This is the stochastic consistency condition from Equation 5.

Variance. Once unbiasedness is ensured, the core design objective of all stochastic methods is to minimize estimator variance for a fixed representation budget, since high variance slows learning.

Here, matrix variance is defined as the sum of the variances of its entries (Tallec & Ollivier, 2018; Mujika et al., 2018; Benzing et al., 2019). For a fixed incoming code $\mathbf{z}$, writing $\widehat{\mathbf{M}}^{(t)} = (\mathcal{R}^{(t)} \circ \mathcal{U}^{(t)})(\mathbf{z})$, this gives

$$\begin{aligned}\operatorname{Var}_{\xi^{(t)}}\left[\widehat{\mathbf{M}}^{(t)}\right] &= \sum_{i=1}^{D_h}\sum_{p=1}^{P}\operatorname{Var}_{\xi^{(t)}}\left[\widehat{m}^{(t)}_{ip}\right] \\ &= \mathbb{E}_{\xi^{(t)}}\left[\left\|\widehat{\mathbf{M}}^{(t)} - \mathbb{E}_{\xi^{(t)}}\left[\widehat{\mathbf{M}}^{(t)}\right]\right\|_F^2\right] \\ &= \mathbb{E}_{\xi^{(t)}}\left[\left\|\left(\mathcal{R}^{(t)} \circ \mathcal{U}^{(t)}\right)(\mathbf{z}) - \left(\mathcal{T}^{(t)} \circ \mathcal{R}^{(t-1)}\right)(\mathbf{z})\right\|_F^2\right].\end{aligned} \tag{53}$$

The last equality follows from the unbiasedness condition, with the incoming code and current transition held fixed.

E.2 Base results and definitions

Definition E.1 (Outer product)**.** Let

$$\mathbf{u} \in \mathbb{R}^m, \qquad \mathbf{v} \in \mathbb{R}^n.$$

The *outer product* of $\mathbf{u}$ and $\mathbf{v}$ is the matrix

$$\mathbf{u}\mathbf{v}^\top := \begin{bmatrix} u_1v_1 & \cdots & u_1v_n \\ \vdots & \ddots & \vdots \\ u_mv_1 & \cdots & u_mv_n \end{bmatrix} \in \mathbb{R}^{m\times n}.$$

Equivalently,

$$(\mathbf{u}\mathbf{v}^\top)_{ij} = u_iv_j.$$

Definition E.2 (Kronecker product)**.** Let

$$\mathbf{A} \in \mathbb{R}^{m\times n}, \qquad \mathbf{B} \in \mathbb{R}^{p\times q}.$$

The *Kronecker product* of $\mathbf{A}$ and $\mathbf{B}$, denoted by $\mathbf{A} \otimes \mathbf{B}$, is the block matrix

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} A_{11}\mathbf{B} & \cdots & A_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ A_{m1}\mathbf{B} & \cdots & A_{mn}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{mp\times nq}.$$

Equivalently,

$$(\mathbf{A} \otimes \mathbf{B})_{(i-1)p+r,\,(j-1)q+s} = A_{ij}B_{rs}.$$

The Kronecker product is bilinear. In particular, for compatible matrices $\mathbf{A}_i, \mathbf{B}_j$ and scalars α_i, β_j,

$$\left(\sum_i \alpha_i\mathbf{A}_i\right) \otimes \left(\sum_j \beta_j\mathbf{B}_j\right) = \sum_i\sum_j \alpha_i\beta_j\,(\mathbf{A}_i \otimes \mathbf{B}_j).$$

Definition E.3 (Frobenius norm)**.** Let $\mathbf{A} \in \mathbb{R}^{m\times n}$. The *Frobenius norm* of $\mathbf{A}$ is

$$\|\mathbf{A}\|_F := \left(\sum_{i=1}^m\sum_{j=1}^n A_{ij}^2\right)^{1/2} = \sqrt{\operatorname{Tr}(\mathbf{A}^\top\mathbf{A})}.$$

The corresponding *Frobenius inner product* of $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$ is

$$\langle\mathbf{A}, \mathbf{B}\rangle_F := \sum_{i=1}^m\sum_{j=1}^n A_{ij}B_{ij} = \operatorname{Tr}(\mathbf{A}^\top\mathbf{B}),$$

so that

$$\|\mathbf{A}\|_F^2 = \langle\mathbf{A}, \mathbf{A}\rangle_F.$$

Lemma E.1 (Frobenius inner product of Kronecker products)**.** Let $\mathbf{A}, \mathbf{C} \in \mathbb{R}^{m\times n}$ and $\mathbf{B}, \mathbf{D} \in \mathbb{R}^{p\times q}$. Then

$$\langle \mathbf{A}\otimes\mathbf{B},\, \mathbf{C}\otimes\mathbf{D}\rangle_F = \langle \mathbf{A},\mathbf{C}\rangle_F \langle \mathbf{B},\mathbf{D}\rangle_F. \tag{54}$$

Proof.

$$\begin{aligned}
\langle \mathbf{A}\otimes\mathbf{B},\, \mathbf{C}\otimes\mathbf{D}\rangle_F &= \mathrm{Tr}\big((\mathbf{A}\otimes\mathbf{B})^\top(\mathbf{C}\otimes\mathbf{D})\big) && \big(\langle \mathbf{X},\mathbf{Y}\rangle_F = \mathrm{Tr}(\mathbf{X}^\top\mathbf{Y})\big)\\
&= \mathrm{Tr}\big((\mathbf{A}^\top\otimes\mathbf{B}^\top)(\mathbf{C}\otimes\mathbf{D})\big) && \big((\mathbf{X}\otimes\mathbf{Y})^\top = \mathbf{X}^\top\otimes\mathbf{Y}^\top\big)\\
&= \mathrm{Tr}\big((\mathbf{A}^\top\mathbf{C})\otimes(\mathbf{B}^\top\mathbf{D})\big) && ((\mathbf{X}\otimes\mathbf{Y})(\mathbf{W}\otimes\mathbf{Z}) = \mathbf{X}\mathbf{W}\otimes\mathbf{Y}\mathbf{Z})\\
&= \mathrm{Tr}(\mathbf{A}^\top\mathbf{C})\,\mathrm{Tr}(\mathbf{B}^\top\mathbf{D}) && (\mathrm{Tr}(\mathbf{X}\otimes\mathbf{Y}) = \mathrm{Tr}(\mathbf{X})\,\mathrm{Tr}(\mathbf{Y}))\\
&= \langle \mathbf{A},\mathbf{C}\rangle_F\langle \mathbf{B},\mathbf{D}\rangle_F && \big(\langle \mathbf{X},\mathbf{Y}\rangle_F = \mathrm{Tr}(\mathbf{X}^\top\mathbf{Y})\big).
\end{aligned} \tag{55}$$

□

Definition E.4 (Rademacher random variable)**.** A *Rademacher random variable* ε is a random variable taking values in $\{-1,+1\}$ with equal probability,

$$\mathbb{P}(\varepsilon = +1) = \mathbb{P}(\varepsilon = -1) = \frac{1}{2}.$$

It satisfies

$$\mathbb{E}[\varepsilon] = 0, \qquad \mathbb{P}(\varepsilon^2 = 1) = 1, \qquad \mathbb{E}[\varepsilon^2] = 1.$$

Definition E.5 (Rademacher random vector)**.** A *Rademacher random vector*

$$\boldsymbol{\varepsilon} = (\varepsilon_1,\ldots,\varepsilon_d)^\top \in \{-1,+1\}^d$$

is a random vector whose entries $\varepsilon_1,\ldots,\varepsilon_d$ are independent Rademacher random variables. It satisfies

$$\mathbb{E}[\varepsilon_i\varepsilon_j] = \delta_{ij} = \begin{cases} 1, & i = j,\\ 0, & i \neq j, \end{cases}$$

or equivalently,

$$\mathbb{E}\left[\boldsymbol{\varepsilon}\,\boldsymbol{\varepsilon}^\top\right] = \mathbf{I}_d.$$

Definition E.6 (Unbiased rank-r matrix approximation)**.** Let $\mathbf{A} \in \mathbb{R}^{H\times P}$. A random matrix $\widehat{\mathbf{A}}_r \in \mathbb{R}^{H\times P}$ is an *unbiased rank-r approximation* of $\mathbf{A}$ if

$$\mathrm{rank}(\widehat{\mathbf{A}}_r) \le r, \qquad \mathbb{E}[\widehat{\mathbf{A}}_r] = \mathbf{A}.$$

It can be written as

$$\widehat{\mathbf{A}}_r = \sum_{q=1}^{r} \mathbf{u}_q\mathbf{v}_q^\top, \qquad \mathbf{u}_q \in \mathbb{R}^H, \quad \mathbf{v}_q \in \mathbb{R}^P.$$

Definition E.7 (Variance of a random matrix, following Ollivier et al. (2015))**.** Let $\widehat{\mathbf{A}} \in \mathbb{R}^{m\times n}$ be a random matrix. Its variance is defined as the sum of the variances of its entries,

$$\begin{aligned}
\mathrm{Var}[\widehat{\mathbf{A}}] &:= \sum_{i=1}^{m}\sum_{j=1}^{n} \mathrm{Var}[\widehat{A}_{ij}]\\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \Big(\mathbb{E}[\widehat{A}_{ij}^2] - \mathbb{E}[\widehat{A}_{ij}]^2\Big)\\
&= \mathbb{E}\Big[\|\widehat{\mathbf{A}}\|_F^2\Big] - \Big\|\mathbb{E}[\widehat{\mathbf{A}}]\Big\|_F^2\\
&= \mathbb{E}\Big[\|\widehat{\mathbf{A}}\|_F^2\Big] - 2\Big\|\mathbb{E}[\widehat{\mathbf{A}}]\Big\|_F^2 + \Big\|\mathbb{E}[\widehat{\mathbf{A}}]\Big\|_F^2\\
&= \mathbb{E}\Big[\|\widehat{\mathbf{A}}\|_F^2 - 2\Big\langle \widehat{\mathbf{A}}, \mathbb{E}[\widehat{\mathbf{A}}]\Big\rangle_F + \Big\|\mathbb{E}[\widehat{\mathbf{A}}]\Big\|_F^2\Big]\\
&= \mathbb{E}\left[\Big\|\widehat{\mathbf{A}} - \mathbb{E}[\widehat{\mathbf{A}}]\Big\|_F^2\right].
\end{aligned} \tag{56}$$

Lemma E.2 (Randomized rank-one representation)**.** Let $\mathbf{A} \in \mathbb{R}^{H\times P}$ and let $\boldsymbol{\varepsilon} \in \{-1,+1\}^H$ be a Rademacher random vector. Then

$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\boldsymbol{\varepsilon}\left(\boldsymbol{\varepsilon}^\top \mathbf{A}\right)\right] = \mathbf{A}. \tag{57}$$

The matrix $\boldsymbol{\varepsilon}(\boldsymbol{\varepsilon}^\top \mathbf{A})$ is therefore an unbiased rank-one approximation of $\mathbf{A}$.

Proof. Since

$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\right] = \mathbf{I}_H,$$

it follows that

$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\boldsymbol{\varepsilon}\left(\boldsymbol{\varepsilon}^\top \mathbf{A}\right)\right] = \mathbb{E}_{\boldsymbol{\varepsilon}}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\right]\mathbf{A} = \mathbf{A}. \tag{58}$$

The estimator is a single outer product and therefore has rank at most one. □

E.3 Unbiased Online Recurrent Optimization (UORO)

Unbiased Online Recurrent Optimization (UORO) was introduced in Tallec & Ollivier (2018) as a method for compressing the influence matrix with a rank-one approximation. Its update operator applies a stochastic rank-one projection twice while preserving the unbiasedness of the influence estimate. Section E.3.1 provides some background results to help understand the method. We put these results together in Section E.3.2 to explain how UORO works. Section E.3.3 describes our implementation. Algorithm 1 gives the pseudocode.

E.3.1 Background and results

UORO carries an influence code consisting of two vectors,

$$\mathbf{z}^{(t)} = (\widetilde{\mathbf{h}}^{(t)}, \widetilde{\mathbf{w}}^{(t)}) \in \mathcal{Z}_{\mathrm{UORO}} := \mathbb{R}^{D_h} \times \mathbb{R}^P. \tag{59}$$

Its decoder forms their outer product,

$$\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) = \widetilde{\mathbf{h}}^{(t)}\widetilde{\mathbf{w}}^{(t)\top}. \tag{60}$$

The decoded class is therefore the set of matrices of rank at most one,

$$\mathcal{M}_{\mathrm{UORO}} = \left\{\mathbf{X} \in \mathbb{R}^{D_h\times P} : \mathrm{rank}(\mathbf{X}) \le 1\right\}. \tag{61}$$

The approximation is unbiased,

$$\mathbb{E}\Big[\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t-1)})\Big] = \mathbf{M}^{(t-1)}. \tag{62}$$

The result of applying the RTRL recurrence to the previous estimate is

$$\begin{aligned}(\mathcal{T}^{(t)} \circ \mathcal{R}_{\mathrm{UORO}})(\mathbf{z}^{(t-1)}) &= \mathbf{J}^{(t)}\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t-1)}) + \mathbf{B}^{(t)} \\ &= \left(\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\right)\widetilde{\mathbf{w}}^{(t-1)\top} + \mathbf{B}^{(t)}.\end{aligned} \tag{63}$$

As the update is affine in the previous influence estimate, its unbiasedness implies

$$\begin{aligned}\mathbb{E}\Big[(\mathcal{T}^{(t)} \circ \mathcal{R}_{\mathrm{UORO}})(\mathbf{z}^{(t-1)})\Big] &= \mathbf{J}^{(t)}\mathbb{E}\Big[\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t-1)})\Big] + \mathbf{B}^{(t)} \\ &= \mathbf{J}^{(t)}\mathbf{M}^{(t-1)} + \mathbf{B}^{(t)} \\ &= \mathbf{M}^{(t)}.\end{aligned} \tag{64}$$

This estimate therefore remains unbiased before its reduction to the next rank-one estimate $\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)})$.

The affine update may leave the decoded class. The result of applying the RTRL recurrence in Equation (63) is the sum of two contributions, a propagated term and an immediate one. The first remains in the decoded class, since it is itself a single outer product,

$$\mathbf{J}^{(t)}\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t-1)}) = \left(\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\right)\widetilde{\mathbf{w}}^{(t-1)\top} \in \mathcal{M}_{\mathrm{UORO}}. \tag{65}$$

However, the second term $\mathbf{B}^{(t)}$ is generally not rank one. Even when $\mathbf{B}^{(t)}$ is rank one, adding it to the propagated term can increase the rank unless one pair of vectors defining the two outer products is collinear.

For example, take $D_h = P = 2$ and

$$\mathbf{J}^{(t)} = \mathbf{I}_2, \qquad \widetilde{\mathbf{h}}^{(t-1)} = \begin{bmatrix}1\\0\end{bmatrix}, \qquad \widetilde{\mathbf{w}}^{(t-1)} = \begin{bmatrix}1\\0\end{bmatrix}, \qquad \mathbf{B}^{(t)} = \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}. \tag{66}$$

Both contributions in Equation (63) have rank one, while their sum is

$$\widehat{\mathbf{M}}^{(t)}_{\mathrm{UORO}} = \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}, \qquad \operatorname{rank}\left(\widehat{\mathbf{M}}^{(t)}_{\mathrm{UORO}}\right) = 2. \tag{67}$$

UORO addresses this issue using an unbiased stochastic rank-one reduction based on Proposition 1 from Tallec & Ollivier (2018), reproduced below.

Proposition E.1 (Randomized rank-one compression, Proposition 1 from Tallec & Ollivier (2018))**.** Suppose a matrix $\mathbf{A} \in \mathbb{R}^{D_h \times P}$ admits the decomposition

$$\mathbf{A} = \sum_{i=1}^{k} \mathbf{p}_i \mathbf{q}_i^\top, \qquad \mathbf{p}_i \in \mathbb{R}^{D_h}, \quad \mathbf{q}_i \in \mathbb{R}^P. \tag{68}$$

Let $\nu_1, \ldots, \nu_k$ be independent Rademacher random variables and let $\rho_1, \ldots, \rho_k > 0$. Define

$$\widetilde{\mathbf{p}} := \sum_{i=1}^{k} \rho_i \nu_i \mathbf{p}_i, \qquad \widetilde{\mathbf{q}} := \sum_{i=1}^{k} \frac{\nu_i}{\rho_i} \mathbf{q}_i. \tag{69}$$

Then

$$\widehat{\mathbf{A}} := \widetilde{\mathbf{p}}\,\widetilde{\mathbf{q}}^\top \tag{70}$$

is an unbiased rank-one approximation of $\mathbf{A}$

$$\operatorname{rank}(\widehat{\mathbf{A}}) \le 1, \qquad \mathbb{E}_\nu[\widehat{\mathbf{A}}] = \mathbf{A}. \tag{71}$$

Proof. Expanding the outer product in Equation (70) gives

$$\begin{aligned}\widehat{\mathbf{A}} &= \left(\sum_{i=1}^{k} \rho_i \nu_i \mathbf{p}_i\right)\left(\sum_{j=1}^{k} \frac{\nu_j}{\rho_j}\mathbf{q}_j\right)^\top \\ &= \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{\rho_i}{\rho_j}\nu_i\nu_j \mathbf{p}_i \mathbf{q}_j^\top.\end{aligned} \tag{72}$$

Since the ν_i are independent Rademacher random variables,

$$\mathbb{E}_\nu[\nu_i\nu_j] = \delta_{ij} = \begin{cases}1, & i = j,\\ 0, & i \ne j.\end{cases} \tag{73}$$

Taking the expectation of Equation (72) therefore eliminates all terms with $i \ne j$

$$\begin{aligned}\mathbb{E}_\nu[\widehat{\mathbf{A}}] &= \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{\rho_i}{\rho_j}\mathbb{E}_\nu[\nu_i\nu_j]\mathbf{p}_i\mathbf{q}_j^\top \\ &= \sum_{i=1}^{k} \mathbf{p}_i\mathbf{q}_i^\top \\ &= \mathbf{A}.\end{aligned} \tag{74}$$

Finally, $\widehat{\mathbf{A}} = \widetilde{\mathbf{p}}\,\widetilde{\mathbf{q}}^\top$ is a single outer product and therefore has rank at most one. □

Proposition E.2 (Variance-minimizing scaling, from Ollivier et al. (2015))**.** Consider the randomized rank-one approximation

$$\widehat{\mathbf{A}} = \left(\sum_{i=1}^{k} \rho_i \nu_i \mathbf{p}_i\right)\left(\sum_{i=1}^{k} \frac{\nu_i}{\rho_i}\mathbf{q}_i\right)^\top, \tag{75}$$

where the ν_i are independent Rademacher random variables and $\rho_i > 0$. Among all choices of the scaling coefficients ρ_i, the variance

$$\mathrm{Var}[\widehat{\mathbf{A}}] = \mathbb{E}_\nu\left[\left\|\widehat{\mathbf{A}} - \mathbb{E}_\nu[\widehat{\mathbf{A}}]\right\|_F^2\right] = \mathbb{E}_\nu\left[\left\|\widehat{\mathbf{A}} - \mathbf{A}\right\|_F^2\right] \tag{76}$$

is minimized by

$$\boxed{\rho_i = \sqrt{\frac{\|\mathbf{q}_i\|_2}{\|\mathbf{p}_i\|_2}}.} \tag{77}$$

Proof. The proof is deferred to Appendix A of Ollivier et al. (2015). □

E.3.2 METHOD

The encoding step ($\mathrm{Encode}_{\mathrm{UORO}}$) replaces the immediate influence $\mathbf{B}^{(t)}$ by an unbiased rank-one approximation. Let $\boldsymbol{\varepsilon} \in \{-1, +1\}^{D_h}$ be a Rademacher random vector. Applying Lemma E.2 gives

$$\begin{aligned} \mathrm{Encode}_{\mathrm{UORO}}(\mathbf{B}^{(t)}) &= \boldsymbol{\varepsilon}(\boldsymbol{\varepsilon}^\top \mathbf{B}^{(t)}) \in \mathcal{M}_{\mathrm{UORO}}, \\ \mathbb{E}_{\boldsymbol{\varepsilon}}\left[\mathrm{Encode}_{\mathrm{UORO}}(\mathbf{B}^{(t)})\right] &= \mathbf{B}^{(t)}. \end{aligned} \tag{78}$$

The propagated influence term has rank at most one. Adding this estimate to the propagated term therefore gives an unbiased estimate of rank at most two

$$\underbrace{\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}}_{\mathbf{p}_1}\underbrace{\widetilde{\mathbf{w}}^{(t-1)\top}}_{\mathbf{q}_1^\top} + \underbrace{\boldsymbol{\varepsilon}}_{\mathbf{p}_2}\underbrace{\boldsymbol{\varepsilon}^\top \mathbf{B}^{(t)}}_{\mathbf{q}_2^\top}. \tag{79}$$

The combination step ($\mathrm{Combine}_{\mathrm{UORO}}$) applies Proposition E.1 to these two rank-one terms, compressing their sum into a single unbiased rank-one estimate. Using the variance-minimizing scaling from Proposition E.2,

$$\rho_1 = \sqrt{\frac{\|\mathbf{q}_1\|_2}{\|\mathbf{p}_1\|_2}} = \sqrt{\frac{\left\|\widetilde{\mathbf{w}}^{(t-1)}\right\|_2}{\left\|\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\right\|_2}}, \qquad \rho_2 = \sqrt{\frac{\|\mathbf{q}_2\|_2}{\|\mathbf{p}_2\|_2}} = \sqrt{\frac{\left\|\mathbf{B}^{(t)\top}\boldsymbol{\varepsilon}\right\|_2}{\|\boldsymbol{\varepsilon}\|_2}}. \tag{80}$$

Let ν_1, ν_2 be independent Rademacher signs, also independent of $\boldsymbol{\varepsilon}$. The combination step gives

$$\begin{aligned} \mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) &= (\rho_1\nu_1\mathbf{p}_1 + \rho_2\nu_2\mathbf{p}_2)\left(\frac{\nu_1\mathbf{q}_1}{\rho_1} + \frac{\nu_2\mathbf{q}_2}{\rho_2}\right)^\top \\ &= \left(\rho_1\nu_1\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)} + \rho_2\nu_2\boldsymbol{\varepsilon}\right)\left(\frac{\nu_1\widetilde{\mathbf{w}}^{(t-1)}}{\rho_1} + \frac{\nu_2\mathbf{B}^{(t)\top}\boldsymbol{\varepsilon}}{\rho_2}\right)^\top \\ &= \nu_1^2\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\widetilde{\mathbf{w}}^{(t-1)\top} + \nu_2^2\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\mathbf{B}^{(t)} \\ &\quad + \frac{\rho_1}{\rho_2}\nu_1\nu_2\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\boldsymbol{\varepsilon}^\top\mathbf{B}^{(t)} + \frac{\rho_2}{\rho_1}\nu_1\nu_2\boldsymbol{\varepsilon}\widetilde{\mathbf{w}}^{(t-1)\top}. \end{aligned} \tag{81}$$

The first two terms simplify because $\nu_1^2 = \nu_2^2 = 1$. Define $\boldsymbol{\varepsilon}' = \nu_1\nu_2\boldsymbol{\varepsilon}$, so that $\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon}'^\top = \boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top$.

$$\begin{aligned} \mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) &= \mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\widetilde{\mathbf{w}}^{(t-1)\top} + \boldsymbol{\varepsilon}'\boldsymbol{\varepsilon}'^\top\mathbf{B}^{(t)} \\ &\quad + \frac{\rho_1}{\rho_2}\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}\boldsymbol{\varepsilon}'^\top\mathbf{B}^{(t)} + \frac{\rho_2}{\rho_1}\boldsymbol{\varepsilon}'\widetilde{\mathbf{w}}^{(t-1)\top}. \end{aligned} \tag{82}$$

The product $\nu_1\nu_2$ either keeps $\boldsymbol{\varepsilon}$ or flips its sign. Both vectors are equally likely under the Rademacher distribution, and a sign flip leaves the norms in ρ_2 unchanged. Thus $\boldsymbol{\varepsilon}'$ has the same

distribution as $\boldsymbol{\varepsilon}$, with the same scaling coefficients. We can therefore draw $\boldsymbol{\varepsilon}'$ directly, without drawing ν_1, ν_2 separately. Thus, we have

$$\begin{aligned}\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) &= \left(\rho_1 \mathbf{J}^{(t)} \widetilde{\mathbf{h}}^{(t-1)} + \rho_2 \boldsymbol{\varepsilon}\right) \left(\frac{\widetilde{\mathbf{w}}^{(t-1)}}{\rho_1} + \frac{\mathbf{B}^{(t)\top} \boldsymbol{\varepsilon}}{\rho_2}\right)^{\top} \\ &= \widetilde{\mathbf{h}}^{(t)} \widetilde{\mathbf{w}}^{(t)\top}.\end{aligned} \tag{83}$$

Thus, the influence code is updated as

$$\mathbf{z}^{(t)} = \mathcal{U}^{(t)}_{\mathrm{UORO}}(\mathbf{z}^{(t-1)}) = (\widetilde{\mathbf{h}}^{(t)}, \widetilde{\mathbf{w}}^{(t)}), \tag{84}$$

with

$$\widetilde{\mathbf{h}}^{(t)} = \rho_1 \mathbf{J}^{(t)} \widetilde{\mathbf{h}}^{(t-1)} + \rho_2 \boldsymbol{\varepsilon}, \tag{85}$$

$$\widetilde{\mathbf{w}}^{(t)} = \frac{\widetilde{\mathbf{w}}^{(t-1)}}{\rho_1} + \frac{\mathbf{B}^{(t)\top} \boldsymbol{\varepsilon}}{\rho_2}. \tag{86}$$

Tensor factorization. For the recurrent layer from Section A, the parameter vector $\mathbf{w}$ is the vectorization of $\mathbf{W}$. Reshaping the UORO factor $\widetilde{\mathbf{w}}^{(t)} \in \mathbb{R}^P$ accordingly gives $\widetilde{\mathbf{W}}^{(t)} \in \mathbb{R}^{H \times (D+1)}$. The rank-one approximation

$$\widetilde{M}^{(t)}_{\mathrm{UORO},ip} = \widetilde{h}^{(t)}_i \widetilde{w}^{(t)}_p \qquad \longleftrightarrow \qquad \widetilde{M}^{(t)}_{\mathrm{UORO},ijk} = \widetilde{h}^{(t)}_i \widetilde{W}^{(t)}_{jk} \tag{87}$$

therefore separates the current hidden index i from the two parameter indices (j, k). In the tensor view of Marschall et al. (2020), UORO consequently has the form $a^{(t)}_i C^{(t)}_{jk}$.

E.3.3 Implementation and scaling

Implementation. The UORO gradient estimate and the updates of the two factors of its rank-one approximation can be computed without explicitly forming the decoded estimate $\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)})$ or the local Jacobians $\mathbf{J}^{(t)}$ and $\mathbf{B}^{(t)}$. The required actions of these Jacobians are instead obtained through Jacobian–vector products (JVPs) and vector–Jacobian products (VJPs), which we compute using JAX Bradbury et al. (2018).

The recurrent gradient estimate is

$$\widehat{\mathbf{g}}^{(t)}_w = \overline{\mathbf{c}}^{(t)} \mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) = \left(\overline{\mathbf{c}}^{(t)} \widetilde{\mathbf{h}}^{(t)}\right) \widetilde{\mathbf{w}}^{(t)\top}. \tag{88}$$

This product can be evaluated directly from the two carried factors.

The factor updates derived above in Equations (85) and (86) require only the two Jacobian products

$$\mathbf{J}^{(t)} \widetilde{\mathbf{h}}^{(t-1)} \qquad \text{and} \qquad \mathbf{B}^{(t)\top} \boldsymbol{\varepsilon}. \tag{89}$$

The first quantity is a Jacobian–vector product (JVP) through the recurrent transition,

$$\mathbf{J}^{(t)} \widetilde{\mathbf{h}}^{(t-1)} = \frac{\partial F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right)}{\partial \mathbf{h}^{(t-1)}} \widetilde{\mathbf{h}}^{(t-1)}. \tag{90}$$

In the original UORO implementation, Tallec & Ollivier (2018) approximate this JVP by a directional finite difference along $\widetilde{\mathbf{h}}^{(t-1)}$,

$$\mathbf{J}^{(t)} \widetilde{\mathbf{h}}^{(t-1)} \approx \frac{F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)} + \alpha \widetilde{\mathbf{h}}^{(t-1)}, \mathbf{x}^{(t)}\right) - F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right)}{\alpha}, \tag{91}$$

for a small α. Using JAX, the same quantity is computed directly as a JVP Bradbury et al. (2018).

The second quantity is obtained by reverse-mode differentiation. Since $\mathbf{B}^{(t)} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}}\big|_{\mathbf{h}^{(t-1)}}$, backpropagating the Rademacher vector $\boldsymbol{\varepsilon}$ through the recurrent transition with respect to $\mathbf{w}$ gives $\mathbf{B}^{(t)\top} \boldsymbol{\varepsilon}$, which we compute as a vector–Jacobian product (VJP) in JAX (Bradbury et al., 2018).

Algorithm 1 One step of UORO, adapted from Tallec & Ollivier (2018)

Input: Recurrent parameters $\mathbf{w} \in \mathbb{R}^P$, previous state $\mathbf{h}^{(t-1)} \in \mathbb{R}^{D_h}$, input $\mathbf{x}^{(t)}$, and loss $\ell^{(t)}$; previous code $\mathbf{z}^{(t-1)} = (\widetilde{\mathbf{h}}^{(t-1)}, \widetilde{\mathbf{w}}^{(t-1)})$, with factors in $\mathbb{R}^{D_h}$ and $\mathbb{R}^P$, respectively.

1: **Advance the recurrent state:**

$$\mathbf{h}^{(t)} \leftarrow F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right).$$

2: **Propagate the previous rank-one estimate:**

$$\mathbf{p}_1 \leftarrow \mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}, \qquad \mathbf{q}_1 \leftarrow \widetilde{\mathbf{w}}^{(t-1)}.$$

3: **Approximate the immediate influence.** Draw $\boldsymbol{\varepsilon} \in \mathbb{R}^{D_h}$ with independent, equally likely ± 1 entries, and set

$$\mathbf{p}_2 \leftarrow \boldsymbol{\varepsilon}, \qquad \mathbf{q}_2 \leftarrow \mathbf{B}^{(t)\top}\boldsymbol{\varepsilon}.$$

4: **Choose the scaling coefficients** that minimize the combination variance:

$$\rho_1 \leftarrow \sqrt{\frac{\|\mathbf{q}_1\|_2}{\|\mathbf{p}_1\|_2}}, \qquad \rho_2 \leftarrow \sqrt{\frac{\|\mathbf{q}_2\|_2}{\|\mathbf{p}_2\|_2}}.$$

5: **Combine the two contributions** into the new factors:

$$\widetilde{\mathbf{h}}^{(t)} \leftarrow \rho_1\mathbf{p}_1 + \rho_2\mathbf{p}_2, \qquad \widetilde{\mathbf{w}}^{(t)} \leftarrow \frac{\mathbf{q}_1}{\rho_1} + \frac{\mathbf{q}_2}{\rho_2}.$$

6: **Update $\mathbf{w}$** using the approximate task gradient

$$\left(\overline{\mathbf{c}}^{(t)}\widetilde{\mathbf{h}}^{(t)}\right)\widetilde{\mathbf{w}}^{(t)\top}, \qquad \overline{\mathbf{c}}^{(t)} = \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}}.$$

7: **Carry the new state and code** to the next step:

$$\mathbf{h}^{(t)}, \qquad \mathbf{z}^{(t)} = \left(\widetilde{\mathbf{h}}^{(t)}, \widetilde{\mathbf{w}}^{(t)}\right).$$

Consequently, the two factors can be updated without constructing either local Jacobian. Using the scaling coefficients from Equation (80), the updates are

$$\widetilde{\mathbf{h}}^{(t)} = \rho_1 \underbrace{\mathrm{JVP}_{\mathbf{h}^{(t-1)}}\left[F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right), \widetilde{\mathbf{h}}^{(t-1)}\right]}_{J^{(t)}\widetilde{h}^{(t-1)}} + \rho_2\boldsymbol{\varepsilon}, \tag{92}$$

$$\widetilde{\mathbf{w}}^{(t)} = \frac{\widetilde{\mathbf{w}}^{(t-1)}}{\rho_1} + \frac{1}{\rho_2}\underbrace{\mathrm{VJP}_w\left[F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right), \boldsymbol{\varepsilon}\right]^{\top}}_{\mathbf{B}^{(t)\top}\boldsymbol{\varepsilon}}. \tag{93}$$

Algorithm 1 summarizes the resulting one-step update.

Scaling.

Persistent state. Each sequence carries the influence code $\mathbf{z}^{(t)} = (\widetilde{\mathbf{h}}^{(t)}, \widetilde{\mathbf{w}}^{(t)})$, requiring $D_h + P$ scalars in addition to the hidden state. For the recurrent layer in Section A, $P = H(D+1)$, giving $\mathcal{O}(H^2)$ additional storage when $D = \mathcal{O}(H)$.

Per-step computation. Each step requires three groups of operations. 1) **State JVP**, $\mathbf{J}^{(t)}\widetilde{\mathbf{h}}^{(t-1)}$, propagates the previous factor at cost $\mathcal{O}(H^2)$. 2) **Parameter VJP**, $\mathbf{B}^{(t)\top}\boldsymbol{\varepsilon}$, supplies the parameter factor for the encoding step at cost $\mathcal{O}(P)$. Both costs follow from Section A.1. 3) **Factor rescaling, factor updates, and gradient computation** require norms, vector additions, and scalar multiplications on the two factors. In particular, $\widehat{\mathbf{g}}_w^{(t)} = \overline{\mathbf{c}}^{(t)}\mathcal{R}_{\mathrm{UORO}}(\mathbf{z}^{(t)}) = (\overline{\mathbf{c}}^{(t)}\widetilde{\mathbf{h}}^{(t)})\widetilde{\mathbf{w}}^{(t)\top}$ requires a dot product between two vectors with H entries followed by scaling a vector with P entries. These

operations together cost $\mathcal{O}(H + P)$. The total additional per-step computation is therefore $\mathcal{O}(P)$, or $\mathcal{O}(H^2)$ when $D = \mathcal{O}(H)$.

E.4 KRONECKER-FACTORED RTRL (KF-RTRL)

Kronecker-Factored RTRL (KF-RTRL), introduced by Mujika et al. (2018), extends UORO by replacing its rank-one factorization with a Kronecker factorization to reduce the variance of the stochastic approximation. Like UORO, it maintains an unbiased approximation of the influence matrix. KF-RTRL is restricted to recurrent cells where each parameter directly affects only one coordinate of the new recurrent state, although each new coordinate may still depend on the entire previous hidden state. The process, as in UORO, proceeds in two steps. First, under this restriction, the immediate Jacobian admits an exact Kronecker factorization. Second, the propagated previous factorization and the immediate Jacobian factorization form a sum of two Kronecker products, which is itself reduced through a stochastic approximation to a single Kronecker product. Section E.4.1 provides some background results to help understand the method. We put these results together in Section E.4.2 to explain how KF-RTRL works. Section E.4.3 describes our implementation. Algorithm 2 gives the pseudocode.

E.4.1 BACKGROUND AND RESULTS

Lemma E.3 (Kronecker factorization of the immediate influence, Lemma 1 from Mujika et al. (2018))**.** Let $\mathbf{s}^{(t)} \in \mathbb{R}^{D+1}$ be the augmented input defined in Section A, and suppose that the recurrent parameters consist of matrices

$$\mathbf{W}_1, \dots, \mathbf{W}_{N_{\text{blocks}}} \in \mathbb{R}^{H \times (D+1)}.$$

Form $\mathbf{w}$ by stacking $\mathbf{W}_1, \dots, \mathbf{W}_{N_{\text{blocks}}}$ vertically and concatenating the columns of the resulting matrix. For each parameter matrix $\mathbf{W}_k$, define the corresponding linear transformation

$$\mathbf{g}_k^{(t)} := \mathbf{W}_k \mathbf{s}^{(t)} \in \mathbb{R}^H, \qquad k = 1, \dots, N_{\text{blocks}}. \tag{94}$$

Assume that each coordinate of the next hidden state is computed as

$$h_j^{(t)} = f\left(g_{1,j}^{(t)}, \dots, g_{N_{\text{blocks}},j}^{(t)}\right), \qquad j = 1, \dots, H. \tag{95}$$

This is the architectural restriction underlying KF-RTRL. Each new hidden coordinate may still depend on the entire previous hidden state through the linear transformations in Equation (94), but after these transformations, $h_j^{(t)}$ depends only on their j-th coordinates.

For each of the N_{blocks} linear transformations, define the diagonal matrix of local derivatives

$$\mathbf{D}_k^{(t)} := \operatorname{diag}\left(\frac{\partial h_1^{(t)}}{\partial g_{k,1}^{(t)}}, \dots, \frac{\partial h_H^{(t)}}{\partial g_{k,H}^{(t)}}\right) \in \mathbb{R}^{H \times H}, \qquad k = 1, \dots, N_{\text{blocks}}. \tag{96}$$

Collect these derivative matrices into the block matrix

$$\mathbf{D}^{(t)} := \begin{bmatrix} \mathbf{D}_1^{(t)} & \cdots & \mathbf{D}_{N_{\text{blocks}}}^{(t)} \end{bmatrix} \in \mathbb{R}^{H \times N_{\text{blocks}} H}. \tag{97}$$

Then the immediate influence matrix admits the exact Kronecker factorization

$$\boxed{\mathbf{B}^{(t)} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{w}^{(t)}} = \mathbf{s}^{(t)\top} \otimes \mathbf{D}^{(t)}.} \tag{98}$$

Proof. Holding $\mathbf{s}^{(t)}$ fixed, consider an individual parameter $(\mathbf{W}_k)_{ji}$, where i indexes the input coordinate and j the output coordinate. From Equation (94), the j-th coordinate of $\mathbf{g}_k^{(t)}$ is

$$g_{k,j}^{(t)} = \sum_{a=1}^{D+1} (\mathbf{W}_k)_{ja}\, s_a^{(t)}, \tag{99}$$

and therefore

$$\frac{\partial g_{k,j}^{(t)}}{\partial (\mathbf{W}_k)_{ji}} = s_i^{(t)}. \tag{100}$$

Given that $h_\ell^{(t)}$ is assumed to depend only on $g_{1,\ell}^{(t)}, \ldots, g_{N_{\text{blocks}},\ell}^{(t)}$, it depends on $g_{k,j}^{(t)}$ only when $j = \ell$. Hence

$$\begin{aligned}\frac{\partial h_\ell^{(t)}}{\partial (\mathbf{W}_k)_{ji}} &= \frac{\partial h_\ell^{(t)}}{\partial g_{k,\ell}^{(t)}} \frac{\partial g_{k,\ell}^{(t)}}{\partial (\mathbf{W}_k)_{ji}} \\ &= s_i^{(t)} \frac{\partial h_\ell^{(t)}}{\partial g_{k,\ell}^{(t)}} \delta_{\ell j} \\ &= s_i^{(t)} \left(\mathbf{D}_k^{(t)}\right)_{\ell j}. \end{aligned} \tag{101}$$

For fixed k and i, collecting these derivatives over $j = 1, \ldots, H$ gives the Jacobian with respect to the i-th column of $\mathbf{W}_k$,

$$\frac{\partial \mathbf{h}^{(t)}}{\partial (\mathbf{W}_k)_{:,i}} = s_i^{(t)} \mathbf{D}_k^{(t)}. \tag{102}$$

Collecting these Jacobians across the N_{blocks} parameter matrices gives

$$\begin{aligned}\frac{\partial \mathbf{h}^{(t)}}{\partial \left((\mathbf{W}_1)_{:,i}, \ldots, (\mathbf{W}_{N_{\text{blocks}}})_{:,i}\right)} &= s_i^{(t)} \begin{bmatrix} \mathbf{D}_1^{(t)} & \cdots & \mathbf{D}_{N_{\text{blocks}}}^{(t)} \end{bmatrix} \\ &= s_i^{(t)} \mathbf{D}^{(t)}. \end{aligned} \tag{103}$$

Finally, concatenating these blocks over $i = 1, \ldots, D+1$ gives

$$\begin{aligned}\mathbf{B}^{(t)} &= \begin{bmatrix} s_1^{(t)} \mathbf{D}^{(t)} & \cdots & s_{D+1}^{(t)} \mathbf{D}^{(t)} \end{bmatrix} \\ &= \mathbf{s}^{(t)\top} \otimes \mathbf{D}^{(t)}, \end{aligned} \tag{104}$$

which proves the result. □

Proposition E.3 (Randomized Kronecker compression, Lemma 3 from Mujika et al. (2018))**.** Suppose a matrix $\mathbf{A}$ admits a decomposition as a sum of Kronecker products,

$$\mathbf{A} = \sum_{i=1}^{k} \mathbf{P}_i \otimes \mathbf{Q}_i, \tag{105}$$

where the matrices $\mathbf{P}_1, \ldots, \mathbf{P}_k$ have the same dimensions and the matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ have the same dimensions. Let $\nu_1, \ldots, \nu_k$ be independent Rademacher random variables and let $\rho_1, \ldots, \rho_k > 0$. Define the two randomized projections

$$\widetilde{\mathbf{P}} := \sum_{i=1}^{k} \rho_i \nu_i \mathbf{P}_i, \qquad \widetilde{\mathbf{Q}} := \sum_{i=1}^{k} \frac{\nu_i}{\rho_i} \mathbf{Q}_i. \tag{106}$$

Take their Kronecker product

$$\widehat{\mathbf{A}} := \widetilde{\mathbf{P}} \otimes \widetilde{\mathbf{Q}}. \tag{107}$$

The approximation $\widehat{\mathbf{A}}$ compresses the sum of k Kronecker products into a single Kronecker product while remaining unbiased,

$$\mathbb{E}_\nu[\widehat{\mathbf{A}}] = \mathbf{A}. \tag{108}$$

Proof. Expanding the Kronecker product of the two randomized projections gives

$$\begin{aligned}\widehat{\mathbf{A}} &= \left(\sum_{i=1}^{k} \rho_i \nu_i \mathbf{P}_i\right) \otimes \left(\sum_{j=1}^{k} \frac{\nu_j}{\rho_j} \mathbf{Q}_j\right) \\ &= \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{\rho_i}{\rho_j} \nu_i \nu_j \mathbf{P}_i \otimes \mathbf{Q}_j. \end{aligned} \tag{109}$$

Since the ν_i are independent Rademacher random variables,

$$\mathbb{E}_\nu\left[\nu_i \nu_j\right] = \delta_{ij}. \tag{110}$$

Therefore,

$$\begin{aligned}\mathbb{E}_\nu[\widehat{\mathbf{A}}] &= \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{\rho_i}{\rho_j}\mathbb{E}_\nu\left[\nu_i\nu_j\right]\mathbf{P}_i \otimes \mathbf{Q}_j \\ &= \sum_{i=1}^{k} \mathbf{P}_i \otimes \mathbf{Q}_i \\ &= \mathbf{A}, \end{aligned} \tag{111}$$

where the last equality follows from Equation (105). □

Proposition E.4 (Variance-minimizing Kronecker scaling, variance result from Lemma 3 of Mujika et al. (2018))**.** Under the notation of Proposition E.3, the variance $\mathrm{Var}[\widehat{\mathbf{A}}]$ is minimized by

$$\boxed{\rho_i = \sqrt{\frac{\|\mathbf{Q}_i\|_F}{\|\mathbf{P}_i\|_F}}.} \tag{112}$$

Proof. The proof is deferred to Lemma 3 of Mujika et al. (2018). □

E.4.2 METHOD

As in UORO, KF-RTRL maintains a factorized approximation of the influence matrix. At each time step, it combines a factorization of the propagated term with a factorization of the immediate influence term $\mathbf{B}^{(t)}$. The difference is that UORO first requires a randomized rank-one approximation of $\mathbf{B}^{(t)}$, whereas for the class of recurrent architectures considered by KF-RTRL, Lemma E.3 provides the exact Kronecker factorization used by the encoding step ($\mathrm{Encode}_{\mathrm{KF}}$)

$$\mathbf{B}^{(t)} = \mathbf{s}^{(t)\top} \otimes \mathbf{D}^{(t)}. \tag{113}$$

The influence code carried from the previous time step is $\mathbf{z}^{(t-1)} = (\widetilde{\mathbf{u}}^{(t-1)}, \widetilde{\mathbf{A}}^{(t-1)}) \in \mathcal{Z}_{\mathrm{KF}}$, with $\mathcal{Z}_{\mathrm{KF}} = \mathbb{R}^{D+1} \times \mathbb{R}^{H \times N_{\mathrm{blocks}} H}$. The corresponding approximation is

$$\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t-1)}) = \widetilde{\mathbf{u}}^{(t-1)\top} \otimes \widetilde{\mathbf{A}}^{(t-1)}. \tag{114}$$

The decoded class is $\mathcal{M}_{\mathrm{KF}} = \{\mathcal{R}_{\mathrm{KF}}(\mathbf{z}) : \mathbf{z} \in \mathcal{Z}_{\mathrm{KF}}\}$. The propagated term remains a single Kronecker product and therefore belongs to $\mathcal{M}_{\mathrm{KF}}$,

$$\mathbf{J}^{(t)}\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t-1)}) = \widetilde{\mathbf{u}}^{(t-1)\top} \otimes \left(\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}\right). \tag{115}$$

Adding the exact factorization of $\mathbf{B}^{(t)}$ gives the result of applying the RTRL recurrence

$$\left(\mathcal{T}^{(t)} \circ \mathcal{R}_{\mathrm{KF}}\right)(\mathbf{z}^{(t-1)}) = \underbrace{\widetilde{\mathbf{u}}^{(t-1)\top}}_{\mathbf{P}_1} \otimes \underbrace{\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}}_{\mathbf{Q}_1} + \underbrace{\mathbf{s}^{(t)\top}}_{\mathbf{P}_2} \otimes \underbrace{\mathbf{D}^{(t)}}_{\mathbf{Q}_2}. \tag{116}$$

The combination step ($\mathrm{Combine}_{\mathrm{KF}}$) then applies Proposition E.3 to these two Kronecker products, reducing their sum to a single Kronecker product while preserving unbiasedness. Using the variance-minimizing scaling from Proposition E.4,

$$\rho_1 = \sqrt{\frac{\|\mathbf{Q}_1\|_F}{\|\mathbf{P}_1\|_F}} = \sqrt{\frac{\left\|\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}\right\|_F}{\left\|\widetilde{\mathbf{u}}^{(t-1)}\right\|_2}}, \qquad \rho_2 = \sqrt{\frac{\|\mathbf{Q}_2\|_F}{\|\mathbf{P}_2\|_F}} = \sqrt{\frac{\left\|\mathbf{D}^{(t)}\right\|_F}{\left\|\mathbf{s}^{(t)}\right\|_2}}. \tag{117}$$

Let ν_1 and ν_2 be independent Rademacher random variables. The resulting single-Kronecker-product approximation can be written as

$$\begin{aligned}\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t)}) &= (\rho_1\nu_1\mathbf{P}_1 + \rho_2\nu_2\mathbf{P}_2) \otimes \left(\frac{\nu_1}{\rho_1}\mathbf{Q}_1 + \frac{\nu_2}{\rho_2}\mathbf{Q}_2\right) \\ &= \left(\rho_1\nu_1\widetilde{\mathbf{u}}^{(t-1)\top} + \rho_2\nu_2\mathbf{s}^{(t)\top}\right) \otimes \left(\frac{\nu_1}{\rho_1}\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)} + \frac{\nu_2}{\rho_2}\mathbf{D}^{(t)}\right) \\ &= \widetilde{\mathbf{u}}^{(t)\top} \otimes \widetilde{\mathbf{A}}^{(t)}. \end{aligned} \tag{118}$$

Algorithm 2 One step of KF-RTRL, adapted from Mujika et al. (2018)

Input: Recurrent parameters $\mathbf{w} \in \mathbb{R}^P$, previous state $\mathbf{h}^{(t-1)} \in \mathbb{R}^H$, input $\mathbf{x}^{(t)}$, and loss $\ell^{(t)}$; previous code $\mathbf{z}^{(t-1)} = (\widetilde{\mathbf{u}}^{(t-1)}, \widetilde{\mathbf{A}}^{(t-1)})$, with factors in $\mathbb{R}^{D+1}$ and $\mathbb{R}^{H\times q}$, where $q = N_{\text{blocks}}H$.

1: **Advance the recurrent state:**

$$\mathbf{h}^{(t)} \leftarrow F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right), \qquad \mathbf{s}^{(t)} \leftarrow \begin{bmatrix}\mathbf{h}^{(t-1)\top} & \mathbf{x}^{(t)\top} & 1\end{bmatrix}^{\top}.$$

2: **Propagate the previous Kronecker estimate:**

$$\mathbf{P}_1 \leftarrow \widetilde{\mathbf{u}}^{(t-1)\top}, \qquad \mathbf{Q}_1 \leftarrow \mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}.$$

3: **Use the exact factorization of the immediate influence:**

$$\mathbf{P}_2 \leftarrow \mathbf{s}^{(t)\top}, \qquad \mathbf{Q}_2 \leftarrow \mathbf{D}^{(t)}, \qquad \mathbf{B}^{(t)} = \mathbf{P}_2 \otimes \mathbf{Q}_2.$$

4: **Choose the scaling coefficients** that minimize the combination variance:

$$\rho_1 \leftarrow \sqrt{\frac{\|\mathbf{Q}_1\|_F}{\|\mathbf{P}_1\|_F}}, \qquad \rho_2 \leftarrow \sqrt{\frac{\|\mathbf{Q}_2\|_F}{\|\mathbf{P}_2\|_F}}.$$

5: **Sample two random signs** $\nu_1, \nu_2 \in \{-1, +1\}$, independently and with equal probability.

6: **Combine the two contributions** into the new factors:

$$\widetilde{\mathbf{u}}^{(t)} \leftarrow \left(\rho_1\nu_1\mathbf{P}_1 + \rho_2\nu_2\mathbf{P}_2\right)^{\top}, \qquad \widetilde{\mathbf{A}}^{(t)} \leftarrow \frac{\nu_1}{\rho_1}\mathbf{Q}_1 + \frac{\nu_2}{\rho_2}\mathbf{Q}_2.$$

7: **Update $\mathbf{w}$** when a task loss is observed, using the approximate gradient

$$\widetilde{\mathbf{u}}^{(t)\top} \otimes \left(\overline{\mathbf{c}}^{(t)}\widetilde{\mathbf{A}}^{(t)}\right), \qquad \overline{\mathbf{c}}^{(t)} = \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}}.$$

8: **Carry the new state and code** to the next step:

$$\mathbf{h}^{(t)}, \qquad \mathbf{z}^{(t)} = \left(\widetilde{\mathbf{u}}^{(t)}, \widetilde{\mathbf{A}}^{(t)}\right).$$

This directly yields the two update equations for the Kronecker factors carried by KF-RTRL,

$$\widetilde{\mathbf{u}}^{(t)} = \rho_1\nu_1\widetilde{\mathbf{u}}^{(t-1)} + \rho_2\nu_2\mathbf{s}^{(t)}, \tag{119}$$

$$\widetilde{\mathbf{A}}^{(t)} = \frac{\nu_1}{\rho_1}\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)} + \frac{\nu_2}{\rho_2}\mathbf{D}^{(t)}. \tag{120}$$

Thus, the influence code is updated as $\mathbf{z}^{(t)} = \mathcal{U}^{(t)}_{\text{KF}}(\mathbf{z}^{(t-1)}) = (\widetilde{\mathbf{u}}^{(t)}, \widetilde{\mathbf{A}}^{(t)})$.

Tensor factorization. For the recurrent layer from Section A, $N_{\text{blocks}} = 1$. The vector factor $\widetilde{\mathbf{u}}^{(t)}$ is indexed by the input coordinate of $\mathbf{W}$, while the matrix factor $\widetilde{\mathbf{A}}^{(t)}$ is indexed by the current hidden coordinate and the output coordinate of $\mathbf{W}$. The Kronecker approximation

$$\mathcal{R}_{\text{KF}}(\mathbf{z}^{(t)}) = \widetilde{\mathbf{u}}^{(t)\top} \otimes \widetilde{\mathbf{A}}^{(t)} \qquad \longleftrightarrow \qquad \widetilde{M}^{(t)}_{\text{KF},ijk} = \widetilde{u}^{(t)}_k \widetilde{A}^{(t)}_{ij} \tag{121}$$

therefore separates the input parameter index k from the current-hidden and output indices (i, j). In the tensor view of Marschall et al. (2020), KF-RTRL consequently has the form $a^{(t)}_k C^{(t)}_{ij}$.

E.4.3 Implementation and scaling

Algorithm 2 summarizes one step for recurrent cells satisfying Lemma E.3.

Implementation. As for UORO, KF-RTRL does not require explicitly constructing the full influence approximation $\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t)})$. Using its Kronecker representation,

$$\widehat{\mathbf{g}}_w^{(t)} = \overline{\mathbf{c}}^{(t)}\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t)}) = \overline{\mathbf{c}}^{(t)}\left(\widetilde{\mathbf{u}}^{(t)\top} \otimes \widetilde{\mathbf{A}}^{(t)}\right) = \widetilde{\mathbf{u}}^{(t)\top} \otimes \left(\overline{\mathbf{c}}^{(t)}\widetilde{\mathbf{A}}^{(t)}\right). \tag{122}$$

The recurrent gradient estimate can therefore be evaluated directly from the two carried factors.

The KF-RTRL updates require computing the two quantities

$$\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)} \qquad \text{and} \qquad \mathbf{D}^{(t)}. \tag{123}$$

Both are computed once and reused in the variance-minimizing scaling coefficients,

$$\rho_1 = \sqrt{\frac{\left\|\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}\right\|_F}{\left\|\widetilde{\mathbf{u}}^{(t-1)}\right\|_2}}, \qquad \rho_2 = \sqrt{\frac{\left\|\mathbf{D}^{(t)}\right\|_F}{\left\|\mathbf{s}^{(t)}\right\|_2}}, \tag{124}$$

and in the update of $\widetilde{\mathbf{A}}^{(t)}$.

The first quantity is the matrix analogue of the single JVP required by UORO. The matrix factor has $q = N_{\mathrm{blocks}}H$ columns. Writing

$$\widetilde{\mathbf{A}}^{(t-1)} = \begin{bmatrix}\mathbf{a}_1 & \cdots & \mathbf{a}_q\end{bmatrix}, \tag{125}$$

gives

$$\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)} = \begin{bmatrix}\mathbf{J}^{(t)}\mathbf{a}_1 & \cdots & \mathbf{J}^{(t)}\mathbf{a}_q\end{bmatrix}. \tag{126}$$

Each column can therefore be computed as

$$\mathbf{J}^{(t)}\mathbf{a}_j = \mathrm{JVP}_{\mathbf{h}^{(t-1)}}\left[F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right), \mathbf{a}_j\right], \tag{127}$$

and the q JVPs can be evaluated in parallel using `vmap` in JAX Bradbury et al. (2018). Thus, whereas UORO propagates one vector direction through $\mathbf{J}^{(t)}$, KF-RTRL propagates an entire matrix of directions. The original implementation of Mujika et al. (2018) instead explicitly forms $\mathbf{J}^{(t)}$ and computes the matrix product $\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}$.

The second quantity follows from the architectural restriction in Lemma E.3, where each hidden coordinate depends only on the corresponding coordinates of the preactivations $\mathbf{g}_k^{(t)} = \mathbf{W}_k\mathbf{s}^{(t)}$. For each k, the local derivative matrix is

$$\mathbf{D}_k^{(t)} := \mathrm{diag}\left(\frac{\partial h_1^{(t)}}{\partial g_{k,1}^{(t)}}, \ldots, \frac{\partial h_H^{(t)}}{\partial g_{k,H}^{(t)}}\right), \tag{128}$$

and

$$\mathbf{D}^{(t)} := \begin{bmatrix}\mathbf{D}_1^{(t)} & \cdots & \mathbf{D}_{N_{\mathrm{blocks}}}^{(t)}\end{bmatrix}. \tag{129}$$

Thus, computing $\mathbf{D}^{(t)}$ only requires the local derivatives of the hidden state with respect to the preactivations. These derivatives may be obtained analytically or by automatic differentiation. They do not require constructing or applying the full parameter Jacobian $\mathbf{B}^{(t)}$. The architectural restriction is precisely what makes these local derivatives sufficient to obtain the exact factorization

$$\mathbf{B}^{(t)} = \mathbf{s}^{(t)\top} \otimes \mathbf{D}^{(t)}. \tag{130}$$

Consequently, unlike UORO, KF-RTRL does not require a VJP with respect to the parameters such as $\mathbf{B}^{(t)\top}\boldsymbol{\varepsilon}$. Its main Jacobian propagation instead consists of the batched JVPs required to compute $\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}$, together with the local derivatives forming $\mathbf{D}^{(t)}$.

Scaling.

The costs below concern the recurrent layer in Section A, with $N_{\mathrm{blocks}} = 1$ and $P = H(D+1)$.

Persistent state. Each sequence carries the influence code $\mathbf{z}^{(t)} = (\widetilde{\mathbf{u}}^{(t)}, \widetilde{\mathbf{A}}^{(t)})$, with $\widetilde{\mathbf{u}}^{(t)} \in \mathbb{R}^{D+1}$ and $\widetilde{\mathbf{A}}^{(t)} \in \mathbb{R}^{H\times H}$. This requires $D + 1 + H^2$ scalars in addition to the hidden state, giving $\mathcal{O}(H^2)$ additional storage when $D = \mathcal{O}(H)$.

Per-step computation. Each step requires three groups of operations.

1) **Multiplication by the state Jacobian**, $\mathbf{J}^{(t)}\widetilde{\mathbf{A}}^{(t-1)}$, propagates the previous matrix factor using H state JVPs $\mathbf{J}^{(t)}\mathbf{a}_j$, one for each column $\mathbf{a}_j$ of $\widetilde{\mathbf{A}}^{(t-1)}$. Each costs $\mathcal{O}(H^2)$, giving $\mathcal{O}(H^3)$ overall, as derived in Section A.1.

2) **Local derivatives forming the immediate Kronecker factor**, $\mathbf{D}^{(t)} = \mathrm{diag}(\sigma'(\mathbf{g}^{(t)}))$, require computing its H diagonal entries at cost $\mathcal{O}(H)$.

3) **Factor rescaling, factor updates, and gradient computation**. Computing ρ_1, ρ_2 and updating $\widetilde{\mathbf{u}}^{(t)}, \widetilde{\mathbf{A}}^{(t)}$ require norms, additions, and scalar multiplications on the two factors. In particular, $\widehat{\mathbf{g}}_w^{(t)} = \overline{\mathbf{c}}^{(t)}\mathcal{R}_{\mathrm{KF}}(\mathbf{z}^{(t)}) = \widetilde{\mathbf{u}}^{(t)\top} \otimes (\overline{\mathbf{c}}^{(t)}\widetilde{\mathbf{A}}^{(t)})$ requires multiplying $\overline{\mathbf{c}}^{(t)}$ by $\widetilde{\mathbf{A}}^{(t)}$ at cost $\mathcal{O}(H^2)$ followed by computing the P entries of the gradient at cost $\mathcal{O}(P)$. These operations together cost $\mathcal{O}(H^2 + P)$.

The total additional per-step computation is therefore $\mathcal{O}(H^3 + P)$, or $\mathcal{O}(H^3)$ when $D = \mathcal{O}(H)$.

E.5 Optimal Kronecker-Sum Approximation (OK)

Optimal Kronecker (OK) (Benzing et al., 2019) extends KF-RTRL (Mujika et al., 2018). It uses the same architectural restriction from Lemma E.3, which allows the immediate influence to be factorized exactly. It improves upon KF-RTRL by maintaining a sum of r Kronecker products and using a different combination rule to reduce $r + 1$ terms to r, whereas KF-RTRL combines two terms into one. Section E.5.1 provides some background results to help understand the method. We put these results together in Section E.5.2 to explain how OK works.

E.5.1 Background and results

Lemma E.4 (Reduction of Kronecker compression to low-rank matrix compression, Lemma A.3 from Benzing et al. (2019))**.** Let $\mathbf{A}$ be a matrix expressed as a sum of $r + 1$ Kronecker products,

$$\mathbf{A} = \sum_{i=1}^{r+1} \mathbf{p}_i \otimes \mathbf{Q}_i, \tag{131}$$

where $\mathbf{p}_i \in \mathbb{R}^{1\times m}$ and $\mathbf{Q}_i \in \mathbb{R}^{n\times q}$, so that $\mathbf{A} \in \mathbb{R}^{n\times mq}$. Assume orthonormal bases $\mathbf{v}_1, \ldots, \mathbf{v}_{r+1}$ and $\mathbf{B}_1, \ldots, \mathbf{B}_{r+1}$ have been chosen for the subspaces spanned by $\mathbf{p}_1, \ldots, \mathbf{p}_{r+1}$ and $\mathbf{Q}_1, \ldots, \mathbf{Q}_{r+1}$, respectively.

Define $\mathbf{L}, \mathbf{R} \in \mathbb{R}^{(r+1)\times(r+1)}$ to contain the coefficients of the factors $\mathbf{p}_i$ and $\mathbf{Q}_i$ in these orthonormal bases,

$$L_{ai} := \langle \mathbf{v}_a, \mathbf{p}_i \rangle, \qquad R_{bi} := \langle \mathbf{B}_b, \mathbf{Q}_i \rangle_F. \tag{132}$$

Each factor can then be written in its corresponding basis as

$$\mathbf{p}_i = \sum_{a=1}^{r+1} L_{ai}\mathbf{v}_a, \qquad \mathbf{Q}_i = \sum_{b=1}^{r+1} R_{bi}\mathbf{B}_b. \tag{133}$$

Define the corresponding coefficient matrix

$$\mathbf{C} := \mathbf{L}\mathbf{R}^\top \in \mathbb{R}^{(r+1)\times(r+1)}. \tag{134}$$

Suppose there exists a random matrix $\widehat{\mathbf{C}} \in \mathbb{R}^{(r+1)\times(r+1)}$ that is an unbiased approximation of $\mathbf{C}$ with rank at most r and minimum variance. More precisely,

$$\widehat{\mathbf{C}} \in \arg\min_{\mathbf{Y}} \mathbb{E}\big[\|\mathbf{Y} - \mathbf{C}\|_F^2\big] \qquad \text{subject to} \qquad \mathbb{E}[\mathbf{Y}] = \mathbf{C}, \quad \mathrm{rank}(\mathbf{Y}) \le r. \tag{135}$$

Since $\mathrm{rank}(\widehat{\mathbf{C}}) \le r$, it admits a rank factorization with at most r outer-product components. Thus, it can be written as

$$\widehat{\mathbf{C}} = \widehat{\mathbf{L}}\widehat{\mathbf{R}}^\top, \qquad \widehat{\mathbf{L}}, \widehat{\mathbf{R}} \in \mathbb{R}^{(r+1)\times r}. \tag{136}$$

We now use $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{R}}$ to construct a new set of factors in the same bases as the original factors. Each column j gives the coefficients of one new factor in the corresponding basis,

$$\widehat{\mathbf{p}}_j := \sum_{a=1}^{r+1} \widehat{L}_{aj}\mathbf{v}_a \in \mathbb{R}^{1\times m}, \qquad \widehat{\mathbf{Q}}_j := \sum_{b=1}^{r+1} \widehat{R}_{bj}\mathbf{B}_b \in \mathbb{R}^{n\times q}, \qquad j = 1,\dots,r. \tag{137}$$

Combining these r pairs gives the approximation

$$\widehat{\mathbf{A}} := \sum_{j=1}^{r} \widehat{\mathbf{p}}_j \otimes \widehat{\mathbf{Q}}_j. \tag{138}$$

Then $\widehat{\mathbf{A}}$ is an unbiased approximation of $\mathbf{A}$, represented as a sum of r Kronecker products, and has minimum variance among all such approximations. Thus, finding an optimal unbiased sum of r Kronecker products for $\mathbf{A}$ reduces to finding an optimal unbiased rank-r approximation of the smaller matrix $\mathbf{C}$.

Proof. See Lemma A.3 and its proof in Section A.1.2 of the supplementary material of Benzing et al. (2019). □

Proposition E.5 (Optimal unbiased low-rank matrix approximation, Theorem 2 from Benzing et al. (2019))**.** Let $\mathbf{C} \in \mathbb{R}^{(r+1)\times(r+1)}$ be the matrix to approximate. There exists a random matrix $\widehat{\mathbf{C}}$ satisfying

$$\widehat{\mathbf{C}} \in \arg\min_{\mathbf{Y}} \mathbb{E}\left[\|\mathbf{Y}-\mathbf{C}\|_F^2\right] \qquad \text{subject to} \qquad \mathbb{E}[\mathbf{Y}] = \mathbf{C}, \quad \operatorname{rank}(\mathbf{Y}) \le r. \tag{139}$$

Following Algorithm 3 of Benzing et al. (2019), this approximation can be constructed as follows.

1. Compute the singular value decomposition.

$$\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top, \qquad \mathbf{\Sigma} = \operatorname{diag}(\sigma_1,\dots,\sigma_{r+1}), \qquad \sigma_1 \ge \cdots \ge \sigma_{r+1} \ge 0. \tag{140}$$

2. Choose which singular values to retain. Define the index m where compression begins and the rank k used for the approximation,

$$m := \min\left\{ i : (r-i+1)\sigma_i \le \sum_{j=i}^{r+1}\sigma_j \right\}, \qquad k := r-m+1, \tag{141}$$

and denote the sum of the singular values to be compressed by $s_1 := \sum_{i=m}^{r+1}\sigma_i$. We retain $\sigma_1,\dots,\sigma_{m-1}$ and approximate the matrix $\operatorname{diag}(\sigma_m,\dots,\sigma_{r+1})$, containing the remaining $k+1$ singular values, with a matrix of rank at most k.

3. Construct an orthonormal basis. Define the vector

$$\mathbf{q}_0 := \begin{bmatrix} \sqrt{1-\dfrac{k\sigma_m}{s_1}} & \cdots & \sqrt{1-\dfrac{k\sigma_{r+1}}{s_1}} \end{bmatrix}^\top \in \mathbb{R}^{k+1}, \tag{142}$$

and choose $\mathbf{q}_1,\dots,\mathbf{q}_k$ so that $\mathbf{q}_0,\mathbf{q}_1,\dots,\mathbf{q}_k$ form an orthonormal basis of $\mathbb{R}^{k+1}$.

4. Apply random signs to the basis vectors. Let $\boldsymbol{\varepsilon} \in \{-1,+1\}^{k+1}$ be a Rademacher random vector and define

$$\mathbf{Z} := \sqrt{\frac{s_1}{k}}\begin{bmatrix} \boldsymbol{\varepsilon}\odot\mathbf{q}_1 & \cdots & \boldsymbol{\varepsilon}\odot\mathbf{q}_k \end{bmatrix} \in \mathbb{R}^{(k+1)\times k}, \tag{143}$$

where $\odot$ denotes the element-wise product. As in the combination step of Proposition E.3, the independent signs cancel the undesired cross-terms in expectation. The construction of Lemmas 5–6 of Benzing et al. (2019) gives

$$\mathbb{E}\left[\mathbf{Z}\mathbf{Z}^\top\right] = \operatorname{diag}\left(\sigma_m,\dots,\sigma_{r+1}\right). \tag{144}$$

Thus, although $\mathbf{Z}\mathbf{Z}^\top$ has rank at most k, its expectation recovers exactly the $k+1$ singular values in the compressed tail.

5. Assemble the approximation to $\boldsymbol{\Sigma}$. Define the two factor matrices

$$\mathbf{L}_0 = \mathbf{R}_0 := \mathrm{blockdiag}\left(\sqrt{\sigma_1}, \ldots, \sqrt{\sigma_{m-1}}, \mathbf{Z}\right) \in \mathbb{R}^{(r+1)\times r}. \tag{145}$$

Their product has rank at most $(m-1)+k = r$, and Equation (144) shows that it is an unbiased approximation of $\boldsymbol{\Sigma}$,

$$\mathbb{E}\left[\mathbf{L}_0\mathbf{R}_0^\top\right] = \boldsymbol{\Sigma}. \tag{146}$$

6. Construct the approximation to $\mathbf{C}$. Define the new factors using the singular vector matrices $\mathbf{U}$ and $\mathbf{V}$,

$$\widehat{\mathbf{L}} := \mathbf{U}\mathbf{L}_0, \qquad \widehat{\mathbf{R}} := \mathbf{V}\mathbf{R}_0, \tag{147}$$

and form their product to obtain the approximation

$$\widehat{\mathbf{C}} := \widehat{\mathbf{L}}\widehat{\mathbf{R}}^\top. \tag{148}$$

Since $\mathbb{E}[\mathbf{L}_0\mathbf{R}_0^\top] = \boldsymbol{\Sigma}$, taking the expectation of this product gives

$$\mathbb{E}[\widehat{\mathbf{C}}] = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top = \mathbf{C}, \tag{149}$$

while $\mathrm{rank}(\widehat{\mathbf{C}}) \le r$. Theorem 2 of Benzing et al. (2019) establishes that this construction satisfies the minimum-variance objective in Equation (139). Its runtime is asymptotically the same as computing the singular value decomposition of $\mathbf{C}$.

Proof. The reduction from $\mathbf{C}$ to its singular-value matrix $\boldsymbol{\Sigma}$ follows from Lemma 4 of Benzing et al. (2019). The minimum achievable variance and the corresponding optimality conditions are established by Theorem 6, while Theorem 7, together with Lemmas 5–6, establishes the construction used by Algorithm 3. We defer the full proof to Appendix 6.3, pp. 13–17 of Benzing et al. (2019). □

E.5.2 METHOD

OK extends KF-RTRL by maintaining a sum of r Kronecker products instead of a single one, and by replacing the combination step of Proposition E.3 with a compression that minimizes variance while preserving unbiasedness (Benzing et al., 2019). As in KF-RTRL, the architectural restriction from Lemma E.3 provides the exact factorization used by the encoding step ($\mathrm{Encode}_{\mathrm{OK}}$),

$$\mathbf{B}^{(t)} = \mathbf{s}^{(t)\top} \otimes \mathbf{D}^{(t)}. \tag{150}$$

The previous influence code contains r pairs, $\mathbf{z}^{(t-1)} = \left((\widetilde{\mathbf{u}}_i^{(t-1)}, \widetilde{\mathbf{A}}_i^{(t-1)})\right)_{i=1}^r \in \mathcal{Z}_{\mathrm{OK}}$, with $\mathcal{Z}_{\mathrm{OK}} = \left(\mathbb{R}^{D+1} \times \mathbb{R}^{H\times N_{\mathrm{blocks}}H}\right)^r$. Here r counts Kronecker terms and N_{blocks} counts weight matrices. The corresponding approximation is

$$\mathcal{R}_{\mathrm{OK}}(\mathbf{z}^{(t-1)}) = \sum_{i=1}^{r} \widetilde{\mathbf{u}}_i^{(t-1)\top} \otimes \widetilde{\mathbf{A}}_i^{(t-1)}. \tag{151}$$

The decoded class is $\mathcal{M}_{\mathrm{OK}} = \{\mathcal{R}_{\mathrm{OK}}(\mathbf{z}) : \mathbf{z} \in \mathcal{Z}_{\mathrm{OK}}\}$. Each propagated term remains a single Kronecker product, so their sum still belongs to $\mathcal{M}_{\mathrm{OK}}$,

$$\mathbf{J}^{(t)}\mathcal{R}_{\mathrm{OK}}(\mathbf{z}^{(t-1)}) = \sum_{i=1}^{r} \widetilde{\mathbf{u}}_i^{(t-1)\top} \otimes \left(\mathbf{J}^{(t)}\widetilde{\mathbf{A}}_i^{(t-1)}\right). \tag{152}$$

Multiplying by $\mathbf{J}^{(t)}$ and adding $\mathbf{B}^{(t)}$ introduce no new approximation. Adding the immediate influence from Equation (150) to the propagated contribution gives a sum of $r+1$ Kronecker products,

$$\left(\mathcal{T}^{(t)} \circ \mathcal{R}_{\mathrm{OK}}\right)(\mathbf{z}^{(t-1)}) = \sum_{i=1}^{r} \underbrace{\widetilde{\mathbf{u}}_i^{(t-1)\top}}_{\mathbf{p}_i} \otimes \underbrace{\mathbf{J}^{(t)}\widetilde{\mathbf{A}}_i^{(t-1)}}_{\mathbf{Q}_i} + \underbrace{\mathbf{s}^{(t)\top}}_{\mathbf{p}_{r+1}} \otimes \underbrace{\mathbf{D}^{(t)}}_{\mathbf{Q}_{r+1}}. \tag{153}$$

The remaining problem is therefore to compress the sum

$$\big(\mathcal{T}^{(t)} \circ \mathcal{R}_{\mathrm{OK}}\big)(\mathbf{z}^{(t-1)}) = \sum_{i=1}^{r+1} \mathbf{p}_i \otimes \mathbf{Q}_i \tag{154}$$

from $r+1$ Kronecker products back to r. This is the combination step ($\mathrm{Combine}_{\mathrm{OK}}$). Unlike KF-RTRL, which applies the combination step of Proposition E.3 directly to the terms it must combine, OK first reduces this Kronecker compression to a small low-rank matrix approximation problem.

Following Lemma E.4, let $\{\mathbf{v}_a\}_{a=1}^{r+1}$ and $\{\mathbf{B}_b\}_{b=1}^{r+1}$ be orthonormal bases of $\mathrm{span}\{\mathbf{p}_i\}_{i=1}^{r+1}$ and $\mathrm{span}\{\mathbf{Q}_i\}_{i=1}^{r+1}$, respectively. Define the coefficient matrices $\mathbf{L}$ and $\mathbf{R}$ in these bases by

$$L_{ai} := \langle \mathbf{v}_a, \mathbf{p}_i \rangle, \qquad R_{bi} := \langle \mathbf{B}_b, \mathbf{Q}_i \rangle_F, \tag{155}$$

and multiply $\mathbf{L}$ by $\mathbf{R}^\top$ to form the smaller matrix

$$\mathbf{C} := \mathbf{L}\mathbf{R}^\top \in \mathbb{R}^{(r+1)\times(r+1)}. \tag{156}$$

Lemma E.4 shows that unbiasedness and variance are preserved by this reduction. Thus, the original compression problem is equivalent to finding an unbiased approximation of $\mathbf{C}$ of rank at most r that minimizes variance.

Proposition E.5 constructs this approximation,

$$\widehat{\mathbf{C}} = \widehat{\mathbf{L}}\widehat{\mathbf{R}}^\top, \qquad \widehat{\mathbf{L}}, \widehat{\mathbf{R}} \in \mathbb{R}^{(r+1)\times r}. \tag{157}$$

Lemma E.4 then maps the two factors of $\widehat{\mathbf{C}}$ back to the original Kronecker spaces, yielding the update equations for the r Kronecker factors carried by OK. For $j = 1, \ldots, r$,

$$\widetilde{\mathbf{u}}_j^{(t)\top} = \sum_{a=1}^{r+1} \widehat{L}_{aj} \mathbf{v}_a, \qquad \widetilde{\mathbf{A}}_j^{(t)} = \sum_{b=1}^{r+1} \widehat{R}_{bj} \mathbf{B}_b. \tag{158}$$

Thus, the influence code is updated as $\mathbf{z}^{(t)} = \mathcal{U}_{\mathrm{OK}}^{(t)}(\mathbf{z}^{(t-1)}) = \big((\widetilde{\mathbf{u}}_j^{(t)}, \widetilde{\mathbf{A}}_j^{(t)})\big)_{j=1}^{r}$. The approximation carried to the next time step is therefore $\mathcal{R}_{\mathrm{OK}}(\mathbf{z}^{(t)}) = \sum_{j=1}^{r} \widetilde{\mathbf{u}}_j^{(t)\top} \otimes \widetilde{\mathbf{A}}_j^{(t)}$. By Lemma E.4 and Proposition E.5, this compression preserves unbiasedness and has minimum variance among unbiased approximations of the quantity in Equation (153) represented as sums of r Kronecker products.

For $r = 1$, OK and KF-RTRL therefore start from the same two-term Kronecker quantity. KF-RTRL reduces these two terms to a single Kronecker product using the combination step of Proposition E.3, whereas 1-OK minimizes variance while preserving unbiasedness using Lemma E.4 and Proposition E.5.

F Deterministic approximation baselines

F.1 Sparse n-step Approximation (SnAp)

The Sparse n-step Approximation (SnAp) was introduced by Menick et al. (2021) as another method for approximating RTRL. Its update is deterministic and generally biased, as discussed below. We describe it for the recurrent layer in Section A, with influence matrix $\mathbf{M}^{(t)} \in \mathbb{R}^{H\times P}$ and $P = H(D+1)$. Section F.1.1 explains how the sparsity mask defines the approximation and its update. Section F.1.2 describes our implementation. Algorithm 3 gives the pseudocode for SnAp-1.

F.1.1 Method

Immediate recurrence case. The RTRL recurrence for each coordinate is

$$M_{ijk}^{(t)} = \Big[\mathcal{T}^{(t)}\Big(\mathbf{M}^{(t-1)}\Big)\Big]_{ijk} = \sum_{m=1}^{H} J_{im}^{(t)} M_{mjk}^{(t-1)} + B_{ijk}^{(t)}. \tag{159}$$

Looking more closely at $\mathbf{B}^{(t)}$, Menick et al. (2021) observe that even for a fully connected layer, the current application of weight W_{jk} affects only hidden coordinate j,

$$B_{ijk}^{(t)} = \frac{\partial h_i^{(t)}}{\partial W_{jk}^{(t)}} = \delta_{ij} \sigma'(g_j^{(t)}) s_k^{(t)}. \tag{160}$$

Each column of $\mathbf{B}^{(t)}$ corresponds to a parameter W_{jk}, with j and k fixed. Since $\delta_{ij} = 0$ for $i \neq j$, that column has at most one nonzero entry, in row j. However, the full influence matrix can become dense as a weight affects other hidden coordinates through recurrent transitions.

Using sparsity in $\mathbf{B}^{(t)}$. SnAp-1 keeps only entries $\widehat{M}^{(t)}_{ijk}$ whose corresponding immediate influence $B^{(t)}_{ijk}$ can be nonzero. For the layer considered here, this means keeping entries with $i = j$. We describe this choice by a binary mask $\mathbf{S}_1 \in \{0,1\}^{H\times P}$,

$$S_{1,ijk} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{161}$$

Assume that the previous estimate satisfies $\widehat{M}^{(t-1)}_{ijk} = 0$ for $i \neq j$. Applying the RTRL operator gives

$$\left[\mathcal{T}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\right)\right]_{ijk} = \sum_{m=1}^{H} J^{(t)}_{im}\widehat{M}^{(t-1)}_{mjk} + B^{(t)}_{ijk} = J^{(t)}_{ij}\widehat{M}^{(t-1)}_{jjk} + B^{(t)}_{ijk}. \tag{162}$$

The sum reduces to the term $m = j$ because all other entries of the previous estimate are zero. This result can still be nonzero for $i \neq j$. SnAp-1 then applies the mask to retain only $i = j$,

$$\begin{aligned} \widehat{M}^{(t)}_{ijk} &= S_{1,ijk}\left[\mathcal{T}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\right)\right]_{ijk} \\ &= \begin{cases} J^{(t)}_{jj}\widehat{M}^{(t-1)}_{jjk} + B^{(t)}_{jjk}, & i = j, \\ 0, & i \neq j. \end{cases} \end{aligned} \tag{163}$$

Starting from $\widehat{\mathbf{M}}^{(0)} = \mathbf{0}$, the mask therefore ensures that $\widehat{M}^{(t)}_{ijk} = 0$ for $i \neq j$ at every step.

Interpretation. The immediate effect of W_{jk} is on hidden coordinate j and can spread to $m \neq j$ at the next step, $W_{jk} \longrightarrow j \longrightarrow m$. By applying the mask $\mathbf{S}_1$, SnAp-1 discards this effect on m, losing its contribution to the current loss. Any later contribution through $W_{jk} \longrightarrow j \longrightarrow m \longrightarrow j$ is also lost, because the effect on m was already discarded.

SnAp-1 retains the immediate influence of each application of W_{jk} at time s,

$$B^{(s)}_{jjk} = \sigma'(g^{(s)}_j)s^{(s)}_k. \tag{164}$$

At the next step, this contribution becomes $J^{(s+1)}_{jj}B^{(s)}_{jjk}$. Each later step multiplies it by another diagonal Jacobian entry, retaining the path

$$W^{(s)}_{jk} \longrightarrow h^{(s)}_j \longrightarrow h^{(s+1)}_j \longrightarrow \cdots \longrightarrow h^{(t)}_j \longrightarrow \ell^{(t)}. \tag{165}$$

Identifying the code, decoded class, and update operator. SnAp-1 carries a sparse approximation of the influence matrix, retaining one entry per column. Since each column corresponds to a parameter, the influence code contains one value per parameter. In the notation of Section 3.1, the influence code is

$$\mathbf{z}^{(t)} \in \mathcal{Z}_{\text{SnAp-1}} := \mathbb{R}^{H\times(D+1)}, \qquad z^{(t)}_{jk} = \widehat{M}^{(t)}_{jjk}. \tag{166}$$

The influence estimate contains the code entries at the positions retained by the mask and zeros elsewhere. Thus, the decoder places each code entry in its corresponding matrix position and sets all other entries to zero. The decoded class contains the matrices allowed by the mask,

$$\left[\mathcal{R}_{\text{SnAp-1}}(\mathbf{z})\right]_{ijk} = \delta_{ij}z_{jk}, \qquad \mathcal{M}_{\text{SnAp-1}} = \left\{\mathbf{X} \in \mathbb{R}^{H\times P} : \mathbf{X} = \mathbf{S}_1 \odot \mathbf{X}\right\}, \tag{167}$$

where $\odot$ denotes elementwise multiplication. Since the retained entries form the code, applying the RTRL recurrence then masking as in Equation 163 defines the update operator,

$$\mathbf{z}^{(t)} = \mathcal{U}^{(t)}_{\text{SnAp-1}}(\mathbf{z}^{(t-1)}), \qquad \left[\mathcal{U}^{(t)}_{\text{SnAp-1}}(\mathbf{z})\right]_{jk} = J^{(t)}_{jj}z_{jk} + \sigma'(g^{(t)}_j)s^{(t)}_k, \tag{168}$$

starting from $\mathbf{z}^{(0)} = \mathbf{0}$. The gradient estimate follows by multiplying the immediate credit by the decoded influence,

$$\widehat{\mathbf{g}}^{(t)}_w = \overline{\mathbf{c}}^{(t)}\mathcal{R}_{\text{SnAp-1}}(\mathbf{z}^{(t)}). \tag{169}$$

For each parameter W_{jk},

$$\widehat{g}^{(t)}_{w,jk} = \overline{\mathbf{c}}^{(t)} \widehat{\mathbf{M}}^{(t)}_{:,jk} = \sum_{i=1}^{H} \overline{c}^{(t)}_i \widehat{M}^{(t)}_{ijk} = \sum_{i=1}^{H} \overline{c}^{(t)}_i \delta_{ij} z^{(t)}_{jk} = \overline{c}^{(t)}_j z^{(t)}_{jk}. \tag{170}$$

This product can be computed directly from the code without forming the influence matrix.

General n-step case. Menick et al. (2021) generalize this approach to $n > 1$ by retaining the influence of each weight W_{jk} on hidden coordinates i that it can affect within n recurrent steps. For the fully connected layer considered here, $\mathbf{S}_2$ retains every entry, so applying the mask leaves the RTRL recurrence unchanged. The paper uses sparse recurrent weights to limit how quickly a weight can affect other hidden coordinates, allowing SnAp to use larger n while keeping its influence approximation sparse. We refer to the original paper for more details.

F.1.2 Implementation and scaling

Implementation. The mask depends on the recurrent layer, as it is built from the positions where the immediate influence can be nonzero. For the layer considered here, $S_{1,ijk} = 1$ if $i = j$ and 0 otherwise. Algorithm 3 summarizes the resulting update.

Algorithm 3 One step of SnAp-1 for the layer in Section A, adapted from Menick et al. (2021)

Input: Recurrent weights $\mathbf{W} \in \mathbb{R}^{H\times(D+1)}$, previous state $\mathbf{h}^{(t-1)} \in \mathbb{R}^H$, input $\mathbf{x}^{(t)} \in \mathbb{R}^{D_x}$, and previous code $\mathbf{z}^{(t-1)} \in \mathbb{R}^{H\times(D+1)}$.

1: **Advance the recurrent state:**

$$\mathbf{s}^{(t)} \leftarrow [\,\mathbf{h}^{(t-1)\top}\ \mathbf{x}^{(t)\top}\ 1\,]^\top, \quad \mathbf{g}^{(t)} \leftarrow \mathbf{W}\mathbf{s}^{(t)}, \quad \mathbf{h}^{(t)} \leftarrow \sigma(\mathbf{g}^{(t)}).$$

2: **Update the retained influences** for $j = 1, \ldots, H$ and $k = 1, \ldots, D+1$:

$$z^{(t)}_{jk} \leftarrow \sigma'(g^{(t)}_j)\left(W_{h,jj} z^{(t-1)}_{jk} + s^{(t)}_k\right).$$

3: **Update $\mathbf{W}$** when a task loss is observed, using the approximate gradient

$$\widehat{g}^{(t)}_{w,jk} = \overline{c}^{(t)}_j z^{(t)}_{jk}, \qquad \overline{\mathbf{c}}^{(t)} = \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}}.$$

4: **Carry the new state and code** $\mathbf{h}^{(t)}$ and $\mathbf{z}^{(t)}$ to the next step.

Scaling.

The costs below concern SnAp-1 for the recurrent layer in Section A, with $P = H(D+1)$.

Persistent state. Each sequence carries $P = H(D+1)$ code entries in addition to its hidden state. The additional memory cost is therefore $\mathcal{O}(P)$, or $\mathcal{O}(H^2)$ when $D = \mathcal{O}(H)$.

Per-step computation. Each step requires three groups of operations.

1) **Local derivatives and diagonal entries of the transition Jacobian**. For this layer, $J^{(t)}_{jj} = \sigma'(g^{(t)}_j) W_{jj}$, since the first H coordinates of $\mathbf{s}^{(t)}$ contain the previous hidden state. Computing $\sigma'(\mathbf{g}^{(t)})$ and the H diagonal entries costs $\mathcal{O}(H)$.

2) **Influence code update**. Computing $z^{(t)}_{jk} = J^{(t)}_{jj} z^{(t-1)}_{jk} + \sigma'(g^{(t)}_j) s^{(t)}_k$ requires a constant number of operations for each of the P retained entries, giving $\mathcal{O}(P)$ computation.

3) **Gradient computation**. Equation 169 gives $\widehat{g}^{(t)}_{w,jk} = \overline{c}^{(t)}_j z^{(t)}_{jk}$, requiring one multiplication per parameter at cost $\mathcal{O}(P)$.

The total additional per-step computation is therefore $\mathcal{O}(H + P) = \mathcal{O}(P)$, or $\mathcal{O}(H^2)$ when $D = \mathcal{O}(H)$.

F.2 Random Feedback Local Online Learning (RFLO)

F.2.1 Method

Random Feedback Local Online Learning (RFLO) was introduced by Murray (2019) as a biologically motivated method for online weight updates. The original method makes two approximations, one to the influence matrix and one to the gradient of the current loss with respect to the hidden state.

Recurrent layer. We use a slight modification of the layer in Section A, adding a skip connection from the previous hidden state. Following Marschall et al. (2020), we write α for the coefficient denoted by $1/\tau$ in Murray (2019). With fixed $0 < \alpha \leq 1$, the layer is

$$\mathbf{g}^{(t)} = \mathbf{W}\mathbf{s}^{(t)}, \qquad \mathbf{h}^{(t)} = (1-\alpha)\mathbf{h}^{(t-1)} + \alpha\sigma(\mathbf{g}^{(t)}). \tag{171}$$

Here $\mathbf{s}^{(t)}$ and $\mathbf{W} = [\mathbf{W}_h \ \mathbf{W}_x \ \mathbf{b}_h]$ are defined as in Section A. We retain our bias parameters, which are absent from the original RFLO cell. Setting $\alpha = 1$ recovers our usual layer.

Local approximation of the influence matrix. For this layer, the RTRL recurrence is

$$\mathbf{M}^{(t)} = (1-\alpha)\mathbf{M}^{(t-1)} + \mathbf{B}^{(t)} + \alpha \operatorname{diag}\Big(\sigma'(\mathbf{g}^{(t)})\Big)\,\mathbf{W}_h\mathbf{M}^{(t-1)} = \mathcal{T}^{(t)}(\mathbf{M}^{(t-1)}), \tag{172}$$

where $\mathbf{J}^{(t)} = (1-\alpha)\mathbf{I}_H + \alpha\operatorname{diag}(\sigma'(\mathbf{g}^{(t)}))\mathbf{W}_h$ and $B^{(t)}_{ijk} = \alpha\delta_{ij}\sigma'(g^{(t)}_j)s^{(t)}_k$. To distinguish local and non-local contributions for weight W_{jk}, consider its influence on hidden coordinate j. Separating the contribution through the self-connection $W_{h,jj}$ gives four terms,

$$\begin{aligned} M^{(t)}_{jjk} = {} & \underbrace{(1-\alpha)M^{(t-1)}_{jjk}}_{\text{retention}} + \underbrace{\alpha\sigma'(g^{(t)}_j)s^{(t)}_k}_{\text{immediate influence}} \\ & \underbrace{+\,\alpha\sigma'(g^{(t)}_j)\sum_{a\neq j} W_{h,ja}M^{(t-1)}_{ajk}}_{\text{non-local term}} + \underbrace{\alpha\sigma'(g^{(t)}_j)W_{h,jj}M^{(t-1)}_{jjk}}_{\text{self-connection term}}. \end{aligned} \tag{173}$$

The non-local term requires the influence of W_{jk} on other hidden coordinates a and their recurrent connections to j. The self-connection term only involves coordinate j. RFLO discards both terms by removing the entire contribution through $\mathbf{W}_h$ in Equation 172. Its influence update is therefore

$$\widehat{M}^{(t)}_{ijk} = (1-\alpha)\widehat{M}^{(t-1)}_{ijk} + \alpha\delta_{ij}\sigma'(g^{(t)}_j)s^{(t)}_k. \tag{174}$$

Interpretation. Starting from $\widehat{\mathbf{M}}^{(0)} = \mathbf{0}$, this update keeps $\widehat{M}^{(t)}_{ijk} = 0$ for $i \neq j$. For the retained entries, expanding the recurrence gives

$$\widehat{M}^{(t)}_{jjk} = \sum_{s=1}^{t}(1-\alpha)^{t-s}\underbrace{\alpha\sigma'(g^{(s)}_j)s^{(s)}_k}_{B^{(s)}_{jjk}}. \tag{175}$$

Each application of W_{jk} contributes its immediate influence, which is propagated through the skip connections along the path

$$W^{(s)}_{jk} \longrightarrow h^{(s)}_j \xrightarrow{1-\alpha} h^{(s+1)}_j \longrightarrow \cdots \xrightarrow{1-\alpha} h^{(t)}_j \longrightarrow \ell^{(t)}. \tag{176}$$

By keeping this record of past activity, the trace makes earlier weight applications eligible for an update based on the current error. This accumulated quantity is therefore an *eligibility trace* (Murray, 2019).

Connection with SnAp-1. Applying SnAp-1 to the modified layer with a skip connection also keeps only entries with $i = j$, but propagates them using the full diagonal entry $J^{(t)}_{jj}$. The two updates are

$$\begin{aligned} \text{SnAp-1} \qquad & \widehat{M}^{(t)}_{jjk} = \big(1-\alpha+\alpha\sigma'(g^{(t)}_j)W_{h,jj}\big)\widehat{M}^{(t-1)}_{jjk} + \alpha\sigma'(g^{(t)}_j)s^{(t)}_k, \\ \text{RFLO} \qquad & \widehat{M}^{(t)}_{jjk} = (1-\alpha)\widehat{M}^{(t-1)}_{jjk} + \alpha\sigma'(g^{(t)}_j)s^{(t)}_k. \end{aligned} \tag{177}$$

SnAp-1 retains the self-connection term, whereas RFLO removes it. At $\alpha = 1$, RFLO retains only the immediate influence, while SnAp-1 can still propagate past influences through the self-connection. RFLO therefore relies entirely on the skip connection to retain past influences, weighting a contribution from time s by $(1-\alpha)^{t-s}$.

Identifying the code, decoded class, and update operator. RFLO carries one retained influence entry per parameter. In the notation of Section 3.1, its influence code is

$$\mathbf{z}^{(t)} \in \mathcal{Z}_{\mathrm{RFLO}} := \mathbb{R}^{H\times(D+1)}, \qquad z_{jk}^{(t)} = \widehat{M}_{jjk}^{(t)}. \tag{178}$$

The decoder places each code entry in the corresponding position with $i = j$ and sets all other entries to zero. RFLO therefore has the same decoded class as SnAp-1,

$$[\mathcal{R}_{\mathrm{RFLO}}(\mathbf{z})]_{ijk} = \delta_{ij} z_{jk}, \qquad \mathcal{M}_{\mathrm{RFLO}} = \left\{\mathbf{X} \in \mathbb{R}^{H\times P} : \mathbf{X} = \mathbf{S}_1 \odot \mathbf{X}\right\}. \tag{179}$$

Since the retained entries form the code, Equation 174 defines the update operator,

$$\mathbf{z}^{(t)} = \mathcal{U}_{\mathrm{RFLO}}^{(t)}(\mathbf{z}^{(t-1)}), \qquad \left[\mathcal{U}_{\mathrm{RFLO}}^{(t)}(\mathbf{z})\right]_{jk} = (1-\alpha) z_{jk} + \alpha\sigma'(g_j^{(t)}) s_k^{(t)}, \tag{180}$$

starting from $\mathbf{z}^{(0)} = \mathbf{0}$. Thus, the two methods use the same code and decoded classes but different update operators.

The gradient estimate is obtained by multiplying the exact immediate credit by the decoded influence,

$$\widehat{\mathbf{g}}_w^{(t)} = \overline{\mathbf{c}}^{(t)} \mathcal{R}_{\mathrm{RFLO}}(\mathbf{z}^{(t)}), \qquad \widehat{g}_{w,jk}^{(t)} = \overline{c}_j^{(t)} z_{jk}^{(t)}. \tag{181}$$

This product can be computed directly from the code without forming the influence matrix.

Random feedback. The original RFLO method also uses fixed random weights to send the output error back to the hidden coordinates, avoiding the need for feedback weights to match the output weights (Murray, 2019). For a linear output with weights $\mathbf{W}_{\mathrm{out}}$ and squared loss, the exact credit $\overline{\mathbf{c}}^{(t)} = \mathbf{e}^{(t)\top}\mathbf{W}_{\mathrm{out}}$ is replaced by $\widehat{\overline{\mathbf{c}}}^{(t)} = \mathbf{e}^{(t)\top}\mathbf{F}^\top$, where $\mathbf{e}^{(t)}$ is the prediction minus the target and $\mathbf{F}$ is sampled once and held fixed. Following Marschall et al. (2020), we use exact feedback to compare how RFLO approximates the RTRL influence matrix.

F.2.2 Implementation and scaling

Scaling.

The costs below concern the recurrent layer in Equation 171, with $P = H(D+1)$ parameters and exact immediate credit, as in the other baselines.

Persistent state. Each sequence carries P code entries in addition to its hidden state. The additional memory cost is therefore $\mathcal{O}(P)$, or $\mathcal{O}(H^2)$ when $D = \mathcal{O}(H)$.

Per-step computation. Each step requires three groups of operations.

1) **Local derivatives**. Computing $\sigma'(\mathbf{g}^{(t)})$ costs $\mathcal{O}(H)$.

2) **Influence code update**. Computing $z_{jk}^{(t)} = (1-\alpha) z_{jk}^{(t-1)} + \alpha\sigma'(g_j^{(t)}) s_k^{(t)}$ requires a constant number of operations for each of the P code entries, giving $\mathcal{O}(P)$ computation.

3) **Gradient computation**. Equation 181 gives $\widehat{g}_{w,jk}^{(t)} = \overline{c}_j^{(t)} z_{jk}^{(t)}$, requiring one multiplication per parameter at cost $\mathcal{O}(P)$.

The total additional per-step computation is therefore $\mathcal{O}(H+P) = \mathcal{O}(P)$, or $\mathcal{O}(H^2)$ when $D = \mathcal{O}(H)$.

G Neural Real-Time Recurrent Learning (NRTRL)

We explain how to train the influence and update networks using Jacobian products, then account for the memory and computation required by NRTRL. Section G.3 gives the architecture and optimization choices.

G.1 Implementation

Computing the RTRL target. The sampled consistency loss in Equation 10 is

$$\widehat{\ell}^{(t)}_{\text{cons}} = \frac{1}{N_{\text{probe}}} \sum\nolimits_{a=1}^{N_{\text{probe}}} \left\| f_{\theta}\left(\mathcal{U}^{(t)}_{\eta}(\mathbf{z}^{(t-1)}), \mathbf{w}\right) \mathbf{v}_a - \text{sg}\Big(\mathcal{T}^{(t)}(\widehat{\mathbf{M}}^{(t-1)})\mathbf{v}_a\Big) \right\|_2^2. \tag{182}$$

For each probe $\mathbf{v}_a \in \mathbb{R}^P$, the RTRL target is

$$\mathbf{y}^{(t)}_a = \text{sg}\Big[\mathcal{T}^{(t)}(\widehat{\mathbf{M}}^{(t-1)})\mathbf{v}_a\Big] = \text{sg}\Big[\mathbf{J}^{(t)}\Big(\widehat{\mathbf{M}}^{(t-1)}\mathbf{v}_a\Big) + \mathbf{B}^{(t)}\mathbf{v}_a\Big]. \tag{183}$$

We first multiply the previous influence estimate by $\mathbf{v}_a$. This amounts to computing a JVP through the previous virtual trajectory,

$$\mathbf{u}^{(t-1)}_a = \widehat{\mathbf{M}}^{(t-1)}\mathbf{v}_a = f_{\theta}(\mathbf{z}^{(t-1)}, \mathbf{w})\mathbf{v}_a = \frac{\partial \Phi_{\theta}(\mathbf{z}^{(t-1)}, \boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\Big|_{\boldsymbol{\omega}=\mathbf{w}} \mathbf{v}_a. \tag{184}$$

We compute the two contributions to the target together using a JVP through the current RNN step. Its derivatives with respect to the previous hidden state and the parameters are $\mathbf{J}^{(t)}$ and $\mathbf{B}^{(t)}$. Applying them to the corresponding vectors gives

$$\begin{bmatrix}\mathbf{J}^{(t)} & \mathbf{B}^{(t)}\end{bmatrix} \begin{bmatrix}\mathbf{u}^{(t-1)}_a \\ \mathbf{v}_a\end{bmatrix} = \mathbf{J}^{(t)}\mathbf{u}^{(t-1)}_a + \mathbf{B}^{(t)}\mathbf{v}_a. \tag{185}$$

Applying stop-gradient to this result gives $\mathbf{y}^{(t)}_a$.

Training the influence and update networks. Evaluating the loss requires three JVPs per probe, one through each virtual trajectory and one through the current RNN step. We then backpropagate through the prediction to update θ and η, keeping the target fixed. This backward pass uses VJPs without forming any full Jacobian.

Task gradient. After updating θ and η, we recompute $\mathbf{z}^{(t)}$ using the updated η. When a task loss is observed, a VJP through Φ_{θ} gives the approximate gradient

$$\overline{\mathbf{c}}^{(t)}\widehat{\mathbf{M}}^{(t)} = \overline{\mathbf{c}}^{(t)}\frac{\partial \Phi_{\theta}(\mathbf{z}^{(t)}, \boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\Big|_{\boldsymbol{\omega}=\mathbf{w}}. \tag{186}$$

Algorithm 4 summarizes one time step for a single sequence.

G.2 Memory and computation

Size of the influence code $\mathbf{z}^{(t)}$. As defined in Section 4, the influence code $\mathbf{z}^{(t)}$ contains the virtual initial state $\mathbf{q}^{(t)}$ and, at each of its K virtual steps, an input $\widetilde{\mathbf{x}}^{(t)}_{\ell}$, a matrix $\mathbf{R}^{(t)}_{\ell,b}$ for each perturbed weight matrix b, and an offset $\Delta\mathbf{b}^{(t)}_{\ell,j}$ for each bias vector j. Write $H = D_h$ for the hidden width. The initial state has H entries, each input has D_x, each $\mathbf{R}^{(t)}_{\ell,b} \in \mathbb{R}^{r_b \times r_b}$ has r_b^2, and each bias offset has D_j. Thus,

$$\dim \mathcal{Z} = H + K\left(D_x + \sum_b r_b^2 + \sum_j D_j\right). \tag{187}$$

For the recurrent cells considered here, the bias vectors have $\mathcal{O}(H)$ entries. With the number of weight matrices and bias vectors, the virtual length K, the perturbation ranks, and D_x fixed, the code requires $\mathcal{O}(H)$ scalars per sequence. This storage does not grow with the length of the physical sequence.

Size of the shared parameters θ and η. For $\mathbf{W}_b \in \mathbb{R}^{m_b \times n_b}$, the matrices $\mathbf{U}_b$ and $\mathbf{V}_b$ in Equation 7 contribute $r_b(m_b + n_b)$ parameters to θ. The update network has parameters η. Both sets of parameters are stored once and shared across sequences.

Cost of the virtual trajectory. At each virtual step, the weight matrices are perturbed as $\widetilde{\mathbf{W}} = \mathbf{W} + \mathbf{URV}$, giving

$$\widetilde{\mathbf{W}}\widetilde{\mathbf{s}} = \mathbf{W}\widetilde{\mathbf{s}} + \mathbf{U}\big(\mathbf{R}(\mathbf{V}\widetilde{\mathbf{s}})\big). \tag{188}$$

Algorithm 4 One time step of Neural Real-Time Recurrent Learning

Input: Parameters $\mathbf{w} \in \mathbb{R}^P$, $\boldsymbol{\theta}$, and $\boldsymbol{\eta}$; previous state $\mathbf{h}^{(t-1)} \in \mathbb{R}^{D_h}$ and code $\mathbf{z}^{(t-1)} \in \mathcal{Z}$; current input $\mathbf{x}^{(t)} \in \mathbb{R}^{D_x}$ and loss $\ell^{(t)}$.

1: **Update the recurrent state:**

$$\mathbf{h}^{(t)} \leftarrow F_{\mathbf{w}}\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right).$$

2: **Sample** N_{probe} probes $\mathbf{v}_a \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_P / N_{\text{probe}})$.

3: **Compute the RTRL targets** for each probe, with $m^{(t)} = 0$ at sequence starts and 1 otherwise:

$$\mathbf{y}_a^{(t)} \leftarrow \operatorname{sg}\left[m^{(t)} \mathbf{J}^{(t)} \left(f_{\boldsymbol{\theta}}(\mathbf{z}^{(t-1)}, \mathbf{w}) \mathbf{v}_a\right) + \mathbf{B}^{(t)} \mathbf{v}_a\right].$$

4: **Predict the new code** with the update network:

$$\mathbf{z}^{(t)} \leftarrow U_{\boldsymbol{\eta}}\left(\mathbf{z}^{(t-1)}, \mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}, \mathbf{h}^{(t)}\right).$$

5: **Train the influence and update networks** by a gradient descent step using

$$\nabla_{\boldsymbol{\theta},\boldsymbol{\eta}} \frac{1}{N_{\text{probe}}} \sum_{a=1}^{N_{\text{probe}}} \left\| f_{\boldsymbol{\theta}}(\mathbf{z}^{(t)}, \mathbf{w}) \mathbf{v}_a - \mathbf{y}_a^{(t)} \right\|_2^2.$$

6: **Recompute** $\mathbf{z}^{(t)}$ as in line 4, using the updated $\boldsymbol{\eta}$.

7: **Update** $\mathbf{w}$ using the approximate task gradient

$$\bar{\mathbf{c}}^{(t)} f_{\boldsymbol{\theta}}(\mathbf{z}^{(t)}, \mathbf{w}), \qquad \bar{\mathbf{c}}^{(t)} = \frac{\partial \ell^{(t)}}{\partial \mathbf{h}^{(t)}}.$$

8: **Carry** $\mathbf{h}^{(t)}$ **and** $\mathbf{z}^{(t)}$ to the next step.

For $\mathbf{W} \in \mathbb{R}^{m \times n}$, the successive multiplications by $\mathbf{V}$, $\mathbf{R}$, and $\mathbf{U}$ cost $\mathcal{O}(nr)$, $\mathcal{O}(r^2)$, and $\mathcal{O}(mr)$, respectively. The perturbation therefore adds $\mathcal{O}((m+n)r + r^2)$ operations to the usual $\mathcal{O}(mn)$ cost of applying $\mathbf{W}$. For an $H \times H$ matrix and fixed r, the total remains $\mathcal{O}(H^2)$.

Cost of a time step. We count the cost of one time step by following the computations it performs, in order. Let C_{phys} be the cost of one physical recurrent step, C_{virt} the cost of one virtual step, and C_U the cost of evaluating and training the update network.

Advancing the RNN. The physical step computes $\mathbf{h}^{(t)}$ from $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$, at cost C_{phys}.

Evaluating the consistency loss. For each of the N_{probe} probes $\mathbf{v}_a$, three Jacobian–vector products are needed:

1. A JVP through the virtual trajectory of the previous code, giving $f_{\boldsymbol{\theta}}(\mathbf{z}^{(t-1)}, \mathbf{w})\mathbf{v}_a$, at cost $\mathcal{O}(KC_{\text{virt}})$.
2. A joint JVP through the physical step, giving the target $\mathbf{J}^{(t)}\mathbf{u}_a + \mathbf{B}^{(t)}\mathbf{v}_a$, at cost $\mathcal{O}(C_{\text{phys}})$.
3. A JVP through the virtual trajectory of the new code, giving the prediction $f_{\boldsymbol{\theta}}(\mathbf{z}^{(t)}, \mathbf{w})\mathbf{v}_a$, at cost $\mathcal{O}(KC_{\text{virt}})$.

Backpropagating the squared error through the last JVP, to train $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$, multiplies its cost by a constant only. Each probe therefore costs $\mathcal{O}(KC_{\text{virt}} + C_{\text{phys}})$.

Updating the code. The update network predicts the new code, takes one optimizer step, and predicts the code again with its updated parameters. Together, these cost C_U.

Computing the task gradient. When a task loss is observed, one vector–Jacobian product through a virtual trajectory, and one through the physical step, give the recurrent-parameter gradient. This costs $\mathcal{O}(KC_{\text{virt}} + C_{\text{phys}})$, the same as one additional probe.

Adding these parts gives

$$C_{\mathrm{NRTRL}} = \mathcal{O}\big((N_{\mathrm{probe}} + 1)(KC_{\mathrm{virt}} + C_{\mathrm{phys}}) + C_U\big). \tag{189}$$

Scaling with the hidden width. We now express each term as a function of H. For a dense GRU with fixed input dimension, one physical step multiplies $H \times H$ matrices, so $C_{\mathrm{phys}} = \mathcal{O}(H^2)$. The previous paragraph showed that the low-rank perturbations keep a virtual step at the same order, $C_{\mathrm{virt}} = \mathcal{O}(H^2)$. For the update transformer, the number of tokens depends only on K and on the number of perturbed parameter blocks, not on H. If its depth is fixed and its token width grows at most linearly with H, each token costs at most $\mathcal{O}(H^2)$, and so does C_U. With K and N_{probe} fixed, every term in Equation 189 is therefore $\mathcal{O}(H^2)$.

G.3 Empirical details

G.3.1 Transformer update network

We implement the update network $U_{\boldsymbol{\eta}}$ using a transformer (Vaswani et al., 2017) with a SwiGLU feed-forward block (Shazeer, 2020). Each object in the previous influence code $\mathbf{z}^{(t-1)}$ becomes a token: one for the virtual initial state and, at each virtual step, one for its input and each parameter perturbation. With TBPTT, the context $\chi^{(t)}$ adds one token for the hidden state at the start of the window and one for each input in that window. Table 2 gives the architecture settings.

The tokens are embedded using learned linear maps. The GRU has three gates, each with one weight matrix applied to the input and one applied to the hidden state. It also has four bias vectors. Each matrix and bias vector has its own encoder for its perturbation. The virtual inputs and initial state have separate encoders. We reuse the encoders across virtual steps and add learned embeddings to identify each parameter and virtual step. All tokens then attend to one another.

The transformer predicts candidate values $\widetilde{\mathbf{z}}^{(t)}$ and retention coefficients for every part of the new influence code. Two learned linear maps produce these values from the outputs of the transformer, one for the candidates and one for the retention logits $\mathbf{a}^{(t)}$. The three GRU matrices applied to inputs reuse the same pair of maps. The three matrices applied to the hidden state reuse another pair, and the four bias vectors reuse a third. The virtual inputs share one pair across steps, and the initial state has another. The maps share parameters, but each matrix, bias vector, and virtual step receives separate predicted values. The new code is

$$\begin{aligned} \boldsymbol{\rho}^{(t)} &= \sigma\left(\mathbf{a}^{(t)}\right), \\ \mathbf{z}^{(t)} &= \boldsymbol{\rho}^{(t)} \odot \mathbf{z}^{(t-1)} + \left(\mathbf{1} - \boldsymbol{\rho}^{(t)}\right) \odot \widetilde{\mathbf{z}}^{(t)}. \end{aligned} \tag{190}$$

A retention coefficient near one preserves its previous value, while one near zero replaces it with the candidate. The output projection weights start at zero, and their retention biases initialize the coefficients near 0.1. At the start of a sequence, retention is set to zero.

Table 2: Hyperparameters of the transformer update network.

Transformer setting	Value
Layers	1
Attention heads	4
Token width	64
Feed-forward block	SwiGLU, width 171
Normalization	Post-norm LayerNorm
Attention value projection	Zero-initialized

G.3.2 Optimization and hyperparameters

Probes. The consistency loss uses $N_{\mathrm{probe}} = 5$ probes in all experiments. Each probe has independent Gaussian entries, $\mathbf{v}_a \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_P/N_{\mathrm{probe}})$, so that $\sum_a \mathbf{v}_a \mathbf{v}_a^\top$ is an unbiased estimate of $\mathbf{I}_P$ and the summed squared residual is an unbiased estimate of the loss in Equation 9. Probes are redrawn for every sequence and every window, and the same probes are used for the target and the prediction.

Training. At every window, the influence network and the update network each take one Adam step ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) on the consistency loss, without gradient clipping or schedules. Table 3 summarizes these settings.

Table 3: NRTRL settings, shared by all experiments unless stated.

Setting	Value
Influence network f_{θ}	
Virtual trajectory length K	7 (MNIST), 5 (PTB)
Perturbation rank r	2
Perturbed parameters	6 GRU weight matrices (rank r), 4 bias vectors (additive)
Virtual inputs	$\sigma(\mathbf{z}_\ell)$; $\mathrm{softmax}(\mathbf{z}_\ell)$ for PTB
Virtual initial state	$\tanh(\mathbf{q})$
Consistency loss	
Number of probes N_{probe}	5
Probe distribution	Gaussian, variance $1/N_{\mathrm{probe}}$
Probe redraw	every sequence and window
Optimization	
Influence network θ	Adam, learning rate 10^{-3}
Update network η	Adam, learning rate 10^{-3}, weight decay 0.01
Steps per window	1 for each network

H THEORETICAL RESULTS

H.1 CONTRACTION OF RTRL OPERATOR

Proposition H.1 (Contraction of the RTRL sequence operator)**.** If $\|\mathbf{J}^{(t)}\|_2 \leq \rho < 1$ at every timestep, then, for any candidate sequences $\mathbf{X}, \mathbf{Y}$ initialized at zero,

$$\|\mathscr{T}\mathbf{X} - \mathscr{T}\mathbf{Y}\|_{2,T} \leq \rho \|\mathbf{X} - \mathbf{Y}\|_{2,T}. \tag{191}$$

Its unique fixed point is the true influence sequence $\mathbf{M} = (\mathbf{M}^{(0)}, \ldots, \mathbf{M}^{(T)})$, and repeated applications of $\mathscr{T}$ converge to $\mathbf{M}$ from any candidate sequence initialized at zero.

Proof. Starting from $(\mathscr{T}\mathbf{X})^{(t)} = \mathbf{J}^{(t)}\mathbf{X}^{(t-1)} + \mathbf{B}^{(t)}$ and the corresponding update for $\mathbf{Y}$, subtracting cancels the immediate influence $\mathbf{B}^{(t)}$. Therefore, using $\|\mathbf{AB}\|_F \leq \|\mathbf{A}\|_2 \|\mathbf{B}\|_F$ and the assumption $\|\mathbf{J}^{(t)}\|_2 \leq \rho < 1$, we get

$$\begin{aligned}
\|\mathscr{T}\mathbf{X} - \mathscr{T}\mathbf{Y}\|_{2,T}^2 &= \frac{1}{T}\sum_{t=1}^{T} \left\| (\mathscr{T}\mathbf{X})^{(t)} - (\mathscr{T}\mathbf{Y})^{(t)} \right\|_F^2 \\
&= \frac{1}{T}\sum_{t=1}^{T} \left\| \mathbf{J}^{(t)} \left(\mathbf{X}^{(t-1)} - \mathbf{Y}^{(t-1)} \right) \right\|_F^2 \\
&\leq \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{J}^{(t)}\|_2^2 \left\| \mathbf{X}^{(t-1)} - \mathbf{Y}^{(t-1)} \right\|_F^2 \\
&\leq \frac{\rho^2}{T}\sum_{t=1}^{T} \left\| \mathbf{X}^{(t-1)} - \mathbf{Y}^{(t-1)} \right\|_F^2 \\
&= \frac{\rho^2}{T}\sum_{s=0}^{T-1} \left\| \mathbf{X}^{(s)} - \mathbf{Y}^{(s)} \right\|_F^2 && (s = t-1) \\
&= \frac{\rho^2}{T}\sum_{s=1}^{T-1} \left\| \mathbf{X}^{(s)} - \mathbf{Y}^{(s)} \right\|_F^2 && (\mathbf{X}^{(0)} = \mathbf{Y}^{(0)} = \mathbf{0}) \\
&\leq \frac{\rho^2}{T}\sum_{s=1}^{T} \left\| \mathbf{X}^{(s)} - \mathbf{Y}^{(s)} \right\|_F^2 \\
&= \rho^2 \|\mathbf{X} - \mathbf{Y}\|_{2,T}^2.
\end{aligned} \tag{192}$$

Taking square roots gives

$$\|\mathscr{T}\mathbf{X} - \mathscr{T}\mathbf{Y}\|_{2,T} \leq \rho \|\mathbf{X} - \mathbf{Y}\|_{2,T}. \tag{193}$$

By definition, $\mathscr{T}\mathbf{M} = \mathbf{M}$. Taking $\mathbf{Y} = \mathbf{M}$ and repeatedly applying this inequality gives

$$\|\mathscr{T}^n\mathbf{X} - \mathbf{M}\|_{2,T} \leq \rho^n \|\mathbf{X} - \mathbf{M}\|_{2,T} \longrightarrow 0. \tag{194}$$

Thus, every candidate sequence initialized at zero converges to $\mathbf{M}$ under repeated applications of $\mathscr{T}$.

If $\mathbf{Y}$ is another fixed point, then

$$\|\mathbf{Y} - \mathbf{M}\|_{2,T} = \|\mathscr{T}\mathbf{Y} - \mathscr{T}\mathbf{M}\|_{2,T} \leq \rho \|\mathbf{Y} - \mathbf{M}\|_{2,T}. \tag{195}$$

Since $\rho < 1$, this implies $\mathbf{Y} = \mathbf{M}$. Thus, $\mathbf{M}$ is the unique fixed point. □

Proposition H.2 (Multistep contraction and control by the consistency loss)**.** Suppose that $\|\mathbf{J}^{(t \leftarrow s)}\|_2 \leq \kappa \rho^{t-s}$ for all $0 \leq s \leq t \leq T$, with $\kappa \geq 1$ and $0 < \rho < 1$. For any candidate sequences $\mathbf{X}, \mathbf{Y}$ initialized at zero, and any integer $n \geq 0$,

$$\|\mathscr{T}^n\mathbf{X} - \mathscr{T}^n\mathbf{Y}\|_{2,T} \leq \kappa \rho^n \|\mathbf{X} - \mathbf{Y}\|_{2,T}. \tag{196}$$

Thus, $\mathscr{T}^n$ is contractive whenever $\kappa\rho^n < 1$. Moreover, if $\mathbf{E}^{(0)} = \mathbf{0}$, then

$$\frac{1}{T}\sum\nolimits_{t=1}^{T} \|\mathbf{E}^{(t)}\|_F^2 \leq \frac{\kappa^2}{(1-\rho)^2} \frac{1}{T}\sum\nolimits_{t=1}^{T} \ell_{\text{cons}}^{(t)}. \tag{197}$$

Proof.

Contraction under repeated applications. Each application of $\mathscr{T}$ computes every entry from the previous candidate sequence. Since $\mathbf{X}^{(0)} = \mathbf{M}^{(0)} = \mathbf{0}$, the first application makes the entry at time 1 exact. The second then makes the entry at time 2 exact,

$$\begin{aligned}(\mathscr{T}\mathbf{X})^{(1)} &= \mathbf{J}^{(1)}\mathbf{X}^{(0)} + \mathbf{B}^{(1)} = \mathbf{J}^{(1)}\mathbf{M}^{(0)} + \mathbf{B}^{(1)} = \mathbf{M}^{(1)},\\ (\mathscr{T}^2\mathbf{X})^{(2)} &= \mathbf{J}^{(2)}(\mathscr{T}\mathbf{X})^{(1)} + \mathbf{B}^{(2)} = \mathbf{J}^{(2)}\mathbf{M}^{(1)} + \mathbf{B}^{(2)} = \mathbf{M}^{(2)}.\end{aligned} \tag{198}$$

Thus, if the first n entries are exact, applying $\mathscr{T}$ preserves them and makes entry $n+1$ exact. Since the initial entry is fixed at $\mathbf{X}^{(0)} = \mathbf{M}^{(0)} = \mathbf{0}$, after T applications the entire sequence is exact, giving $\mathscr{T}^T\mathbf{X} = \mathbf{M}$.

Now take $1 \le n < T$ and $t > n$. To compare two candidate sequences at time t after n applications, we substitute the RTRL recurrence n times,

$$\begin{aligned}(\mathscr{T}^n\mathbf{X})^{(t)} - (\mathscr{T}^n\mathbf{Y})^{(t)} &= \mathbf{J}^{(t)}(\mathscr{T}^{n-1}\mathbf{X})^{(t-1)} + \mathbf{B}^{(t)} - \mathbf{J}^{(t)}(\mathscr{T}^{n-1}\mathbf{Y})^{(t-1)} - \mathbf{B}^{(t)}\\ &= \mathbf{J}^{(t)}\left[(\mathscr{T}^{n-1}\mathbf{X})^{(t-1)} - (\mathscr{T}^{n-1}\mathbf{Y})^{(t-1)}\right]\\ &= \cdots\\ &= \mathbf{J}^{(t)}\cdots\mathbf{J}^{(t-n+1)}\left(\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right)\\ &= \mathbf{J}^{(t\leftarrow t-n)}\left(\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right).\end{aligned} \tag{199}$$

Each substitution moves one timestep back and contributes one transition Jacobian, giving n factors in total. After n substitutions, the difference is expressed in terms of the original values of the candidates, $\mathbf{X}^{(t-n)}$ and $\mathbf{Y}^{(t-n)}$. As shown above, after n applications both sequences equal the true influence sequence at times $0, \ldots, n$. Combining this fact with the preceding calculation gives, for $0 \le n < T$,

$$(\mathscr{T}^n\mathbf{X})^{(t)} - (\mathscr{T}^n\mathbf{Y})^{(t)} = \begin{cases}\mathbf{0}, & t \le n,\\ \mathbf{J}^{(t\leftarrow t-n)}\left(\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right), & t > n.\end{cases} \tag{200}$$

Using $\|\mathbf{AB}\|_F \le \|\mathbf{A}\|_2\|\mathbf{B}\|_F$ and the assumption $\|\mathbf{J}^{(t\leftarrow s)}\|_2 \le \kappa\rho^{t-s}$, we get

$$\begin{aligned}\|\mathscr{T}^n\mathbf{X} - \mathscr{T}^n\mathbf{Y}\|_{2,T}^2 &= \frac{1}{T}\sum_{t=1}^{T}\left\|(\mathscr{T}^n\mathbf{X})^{(t)} - (\mathscr{T}^n\mathbf{Y})^{(t)}\right\|_F^2\\ &\le \frac{1}{T}\sum_{t=n+1}^{T}\left\|\mathbf{J}^{(t\leftarrow t-n)}\left(\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right)\right\|_F^2\\ &\le \frac{1}{T}\sum_{t=n+1}^{T}\left\|\mathbf{J}^{(t\leftarrow t-n)}\right\|_2^2\left\|\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right\|_F^2\\ &\le \frac{1}{T}\sum_{t=n+1}^{T}\left(\kappa\rho^{\,t-(t-n)}\right)^2\left\|\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right\|_F^2\\ &= \frac{\kappa^2\rho^{2n}}{T}\sum_{t=n+1}^{T}\left\|\mathbf{X}^{(t-n)} - \mathbf{Y}^{(t-n)}\right\|_F^2\\ &= \frac{\kappa^2\rho^{2n}}{T}\sum_{s=1}^{T-n}\left\|\mathbf{X}^{(s)} - \mathbf{Y}^{(s)}\right\|_F^2 \qquad (s = t-n)\\ &\le \frac{\kappa^2\rho^{2n}}{T}\sum_{s=1}^{T}\left\|\mathbf{X}^{(s)} - \mathbf{Y}^{(s)}\right\|_F^2\\ &= \kappa^2\rho^{2n}\|\mathbf{X} - \mathbf{Y}\|_{2,T}^2.\end{aligned} \tag{201}$$

Taking square roots gives

$$\|\mathscr{T}^n\mathbf{X} - \mathscr{T}^n\mathbf{Y}\|_{2,T} \le \kappa\rho^n\|\mathbf{X} - \mathbf{Y}\|_{2,T}. \tag{202}$$

For $n \ge T$, both iterates equal $\mathbf{M}$, so the inequality also holds. Thus, $\mathscr{T}^n$ is contractive whenever $\kappa\rho^n < 1$.

Bound on the average influence error. The assumption $\mathbf{E}^{(0)} = \mathbf{0}$ ensures that $\widehat{\mathbf{M}}$ is initialized at zero. As shown above, applying $\mathscr{T}$ T times gives the true influence sequence, so $\mathscr{T}^T\widehat{\mathbf{M}} = \mathbf{M}$. Adding and subtracting the intermediate sequences gives

$$\begin{aligned}\widehat{\mathbf{M}} - \mathbf{M} &= \widehat{\mathbf{M}} - \mathscr{T}^T\widehat{\mathbf{M}} \\ &= \left(\widehat{\mathbf{M}} - \mathscr{T}\widehat{\mathbf{M}}\right) + \left(\mathscr{T}\widehat{\mathbf{M}} - \mathscr{T}^2\widehat{\mathbf{M}}\right) + \cdots + \left(\mathscr{T}^{T-1}\widehat{\mathbf{M}} - \mathscr{T}^T\widehat{\mathbf{M}}\right) \\ &= \sum_{n=0}^{T-1}\left(\mathscr{T}^n\widehat{\mathbf{M}} - \mathscr{T}^{n+1}\widehat{\mathbf{M}}\right).\end{aligned} \tag{203}$$

Using the triangle inequality and the bound from the first part, we get

$$\begin{aligned}\|\widehat{\mathbf{M}} - \mathbf{M}\|_{2,T} &= \left\|\sum_{n=0}^{T-1}\left(\mathscr{T}^n\widehat{\mathbf{M}} - \mathscr{T}^{n+1}\widehat{\mathbf{M}}\right)\right\|_{2,T} \\ &\le \sum_{n=0}^{T-1}\left\|\mathscr{T}^n\widehat{\mathbf{M}} - \mathscr{T}^{n+1}\widehat{\mathbf{M}}\right\|_{2,T} \\ &= \sum_{n=0}^{T-1}\left\|\mathscr{T}^n\widehat{\mathbf{M}} - \mathscr{T}^n(\mathscr{T}\widehat{\mathbf{M}})\right\|_{2,T} \\ &\le \sum_{n=0}^{T-1}\kappa\rho^n\|\widehat{\mathbf{M}} - \mathscr{T}\widehat{\mathbf{M}}\|_{2,T} \\ &= \kappa\left(\sum_{n=0}^{T-1}\rho^n\right)\|\widehat{\mathbf{M}} - \mathscr{T}\widehat{\mathbf{M}}\|_{2,T} \\ &= \frac{\kappa(1-\rho^T)}{1-\rho}\|\widehat{\mathbf{M}} - \mathscr{T}\widehat{\mathbf{M}}\|_{2,T} \\ &\le \frac{\kappa}{1-\rho}\|\widehat{\mathbf{M}} - \mathscr{T}\widehat{\mathbf{M}}\|_{2,T}.\end{aligned} \tag{204}$$

Squaring the preceding bound gives

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\|\mathbf{E}^{(t)}\|_F^2 &= \frac{1}{T}\sum_{t=1}^{T}\|\widehat{\mathbf{M}}^{(t)} - \mathbf{M}^{(t)}\|_F^2 \\ &= \|\widehat{\mathbf{M}} - \mathbf{M}\|_{2,T}^2 \\ &\le \frac{\kappa^2}{(1-\rho)^2}\|\widehat{\mathbf{M}} - \mathscr{T}\widehat{\mathbf{M}}\|_{2,T}^2 \\ &= \frac{\kappa^2}{(1-\rho)^2}\frac{1}{T}\sum_{t=1}^{T}\left\|\widehat{\mathbf{M}}^{(t)} - (\mathscr{T}\widehat{\mathbf{M}})^{(t)}\right\|_F^2 \\ &= \frac{\kappa^2}{(1-\rho)^2}\frac{1}{T}\sum_{t=1}^{T}\left\|\widehat{\mathbf{M}}^{(t)} - \mathcal{T}^{(t)}\left(\widehat{\mathbf{M}}^{(t-1)}\right)\right\|_F^2 \\ &= \frac{\kappa^2}{(1-\rho)^2}\frac{1}{T}\sum_{t=1}^{T}\ell_{\mathrm{cons}}^{(t)}.\end{aligned} \tag{205}$$

□

H.2 FACTORIZATION OF THE INFLUENCE MATRIX

We prove Proposition 2.

We give the factorization of the influence approximation using a general weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$. The main-text statement follows for the layer in Equation 15, where $D_h = m = H$, $n = D+1$, and $\mathbf{M}_W^{(t)} = \mathbf{M}^{(t)}$.

Suppose that the only uses of $\mathbf{W}$ in the virtual trajectory are $\widetilde{\mathbf{g}}_\ell^{(t)} = \widetilde{\mathbf{W}}_\ell^{(t)}\widetilde{\mathbf{s}}_\ell^{(t)}$, with $\widetilde{\mathbf{W}}_\ell^{(t)} = \mathbf{W} + \mathbf{U}\mathbf{R}_\ell^{(t)}\mathbf{V}$ and $\widetilde{\mathbf{s}}_\ell^{(t)} \in \mathbb{R}^n$. Let $\widehat{\mathbf{M}}_W^{(t)}$ contain the columns of the influence estimate corresponding to $\mathbf{W}$, ordered by columns of $\mathbf{W}$. The code and shared parameters are held fixed, and all derivatives are evaluated at the recurrent parameters.

Proposition H.3 (Factorization of the influence approximation)**.** Let the Jacobian matrix $\mathbf{A}_\ell^{(t)} := \partial\Phi_\theta/\partial\widetilde{\mathbf{g}}_\ell^{(t)} \in \mathbb{R}^{D_h\times m}$ describe how the final virtual state depends on the preactivation at step ℓ. The influence estimate decomposes as

$$\widehat{\mathbf{M}}_W^{(t)} = \sum\nolimits_{\ell=1}^{K} \widetilde{\mathbf{s}}_\ell^{(t)\top} \otimes \mathbf{A}_\ell^{(t)}. \tag{206}$$

Proof.

The influence estimate is the Jacobian of Φ_θ with respect to the recurrent parameters. Write $\mathbf{W}^{(\ell)}$ for the application of the shared matrix $\mathbf{W}$ at virtual step ℓ. Summing the contributions from these applications gives

$$\widehat{\mathbf{M}}_W^{(t)} = \frac{\partial\Phi_\theta}{\partial\mathbf{W}} = \sum\nolimits_{\ell=1}^{K} \frac{\partial\Phi_\theta}{\partial\mathbf{W}^{(\ell)}}. \tag{207}$$

For each application, $\partial\widetilde{\mathbf{W}}_\ell^{(t)}/\partial\mathbf{W}^{(\ell)} = \mathbf{I}_{mn}$. Applying the chain rule gives

$$\frac{\partial\Phi_\theta}{\partial\mathbf{W}^{(\ell)}} = \frac{\partial\Phi_\theta}{\partial\widetilde{\mathbf{W}}_\ell^{(t)}}\frac{\partial\widetilde{\mathbf{W}}_\ell^{(t)}}{\partial\mathbf{W}^{(\ell)}} = \frac{\partial\Phi_\theta}{\partial\widetilde{\mathbf{W}}_\ell^{(t)}}. \tag{208}$$

We now compute this contribution entry by entry. Recall that the layer gives $\widetilde{g}_{\ell,a}^{(t)} = \sum_{b=1}^{n}\widetilde{W}_{\ell,ab}^{(t)}\widetilde{s}_{\ell,b}^{(t)}$. Applying the chain rule gives

$$\begin{aligned}\left[\frac{\partial\Phi_\theta}{\partial\widetilde{\mathbf{W}}_\ell^{(t)}}\right]_{ijk} &= \sum\nolimits_{a=1}^{m}\left[\frac{\partial\Phi_\theta}{\partial\widetilde{\mathbf{g}}_\ell^{(t)}}\right]_{ia}\left[\frac{\partial\widetilde{\mathbf{g}}_\ell^{(t)}}{\partial\widetilde{\mathbf{W}}_\ell^{(t)}}\right]_{ajk}\\ &= \sum\nolimits_{a=1}^{m} A_{\ell,ia}^{(t)}\delta_{aj}\widetilde{s}_{\ell,k}^{(t)}\\ &= A_{\ell,ij}^{(t)}\widetilde{s}_{\ell,k}^{(t)}.\end{aligned} \tag{209}$$

The derivative $\mathbf{A}_\ell^{(t)}$ includes every later operation through which $\widetilde{\mathbf{g}}_\ell^{(t)}$ affects the final virtual state. Summing over the applications of $\mathbf{W}$ yields

$$\widehat{M}_{W,ijk}^{(t)} = \left[\frac{\partial\Phi_\theta}{\partial\mathbf{W}}\right]_{ijk} = \sum\nolimits_{\ell=1}^{K}\left[\frac{\partial\Phi_\theta}{\partial\mathbf{W}^{(\ell)}}\right]_{ijk} = \sum\nolimits_{\ell=1}^{K} A_{\ell,ij}^{(t)}\widetilde{s}_{\ell,k}^{(t)}. \tag{210}$$

For a fixed column k of $\mathbf{W}$, collecting these derivatives over i, j gives

$$\left[\frac{\partial\Phi_\theta}{\partial\mathbf{W}}\right]_{:,:,k} = \sum\nolimits_{\ell=1}^{K}\widetilde{s}_{\ell,k}^{(t)}\mathbf{A}_\ell^{(t)}. \tag{211}$$

Collecting these Jacobians over all columns of $\mathbf{W}$, then taking the sum outside the block matrix, gives

$$\begin{aligned}\widehat{\mathbf{M}}_W^{(t)} &= \begin{bmatrix}\sum_{\ell=1}^{K}\widetilde{s}_{\ell,1}^{(t)}\mathbf{A}_\ell^{(t)} & \cdots & \sum_{\ell=1}^{K}\widetilde{s}_{\ell,n}^{(t)}\mathbf{A}_\ell^{(t)}\end{bmatrix}\\ &= \sum\nolimits_{\ell=1}^{K}\begin{bmatrix}\widetilde{s}_{\ell,1}^{(t)}\mathbf{A}_\ell^{(t)} & \cdots & \widetilde{s}_{\ell,n}^{(t)}\mathbf{A}_\ell^{(t)}\end{bmatrix}\\ &= \sum\nolimits_{\ell=1}^{K}\widetilde{\mathbf{s}}_\ell^{(t)\top}\otimes\mathbf{A}_\ell^{(t)}.\end{aligned} \tag{212}$$

□

I EXPERIMENTS

Gated recurrent unit (GRU). The GRU (Cho et al., 2014) uses a *reset gate* $\mathbf{r}^{(t)}$ and an *update gate* $\mathbf{g}^{(t)}$ to compute a candidate state $\mathbf{n}^{(t)}$ and update the hidden state,

$$\begin{aligned}
\mathbf{r}^{(t)} &= \sigma\Big(\mathbf{W}_{r,h}\mathbf{h}^{(t-1)} + \mathbf{W}_{r,x}\mathbf{x}^{(t)} + \mathbf{b}_r\Big), \\
\mathbf{g}^{(t)} &= \sigma\Big(\mathbf{W}_{g,h}\mathbf{h}^{(t-1)} + \mathbf{W}_{g,x}\mathbf{x}^{(t)} + \mathbf{b}_g\Big), \\
\mathbf{n}^{(t)} &= \tanh\Big(\mathbf{W}_{n,h}\big(\mathbf{r}^{(t)} \odot \mathbf{h}^{(t-1)}\big) + \mathbf{W}_{n,x}\mathbf{x}^{(t)} + \mathbf{b}_n\Big), \\
\mathbf{h}^{(t)} &= \mathbf{g}^{(t)} \odot \mathbf{h}^{(t-1)} + \big(\mathbf{1} - \mathbf{g}^{(t)}\big) \odot \mathbf{n}^{(t)}.
\end{aligned} \tag{213}$$

Here σ is the sigmoid, both nonlinearities act elementwise, and $\odot$ denotes elementwise multiplication. For $a \in \{r, g, n\}$, the parameters have shapes $\mathbf{W}_{a,h} \in \mathbb{R}^{H\times H}$, $\mathbf{W}_{a,x} \in \mathbb{R}^{H\times D_x}$, and $\mathbf{b}_a \in \mathbb{R}^H$. The state dimension is $D_h = H$.

As stated in Lemma E.3, KF-RTRL (Mujika et al., 2018) and OK (Benzing et al., 2019) require the cell to first compute weighted sums of the previous state and current input, then apply nonlinearities and gates separately to each hidden coordinate. In the formulation above, $\mathbf{W}_{n,h}$ mixes the reset-gate coordinates, so a reset-gate parameter can directly affect several hidden coordinates. To meet this requirement, we apply the reset gate after the recurrent matrix multiplication,

$$\mathbf{n}^{(t)} = \tanh\Big(\mathbf{r}^{(t)} \odot \big(\mathbf{W}_{n,h}\mathbf{h}^{(t-1)} + \mathbf{b}_{n,h}\big) + \mathbf{W}_{n,x}\mathbf{x}^{(t)} + \mathbf{b}_{n,x}\Big), \tag{214}$$

where $\mathbf{b}_{n,h}, \mathbf{b}_{n,x} \in \mathbb{R}^H$. This allows the immediate influence of each weight matrix to be factorized as required by KF-RTRL and OK. This variant is implemented in Flax (Heek et al., 2026), whose GRU implementation we use throughout this work.

Task gradient and TBPTT. When TBPTT is used, the sequence is divided into non-overlapping windows of L steps. Within each window, the observed task losses are backpropagated, stopping at the hidden state that entered the window. For the influence approximations other than SnAp-1, we treat those L steps as one recurrent transition, so the carried representation is updated once per window. SnAp-1 instead updates its sparse influence estimate at every physical step.

For a window covering steps $s, \dots, t$, let $\mathcal{L}_{s:t} = \sum_{u=s}^{t} \ell^{(u)}$. The gradient used to update the recurrent parameters is

$$\widehat{\mathbf{g}}_w^{(s:t)} = \left.\frac{\partial \mathcal{L}_{s:t}}{\partial \mathbf{w}}\right|_{\mathbf{h}^{(s-1)} \text{ fixed}} + \frac{\partial \mathcal{L}_{s:t}}{\partial \mathbf{h}^{(s-1)}} \widehat{\mathbf{M}}^{(s-1)}. \tag{215}$$

This also describes our implementation when $L = 1$: the current step contributes $\overline{\mathbf{c}}^{(t)}\mathbf{B}^{(t)}$, while earlier steps contribute $\overline{\mathbf{c}}^{(t)}\mathbf{J}^{(t)}\widehat{\mathbf{M}}^{(t-1)}$.