Model-Free Multiplexed Gradient Descent: Neuromorphic Learning in Recurrent Networks

1st Ryan O'Loughlin
Physical Measurements Laboratory
Natl. Inst. of Standards and Technology
Department of Computer Science
University of Colorado Boulder
Boulder, United States
ryan.oloughlin@nist.gov

2nd Nick Skuda
Physical Measurements Laboratory
Natl. Inst. of Standards and Technology
Boulder, United States
nicholas.skuda@nist.gov

3rd Bakhrom Oripov
Physical Measurements Laboratory
Natl. Inst. of Standards and Technology
Boulder, United States
bakhrom.oripov@nist.gov

4th Bradley Hayes
Deparment of Computer Science
University of Colorado Boulder
Boulder, United States
bradley.hayes@colorado.edu

5th Adam McCaughan
Physical Measurements Laboratory
Natl. Inst. of Standards and Technology
Boulder, United States
adam.mccaughan@nist.gov

6th Sonia Buckley
Physical Measurements Laboratory
Natl. Inst. of Standards and Technology
Boulder, United States
sonia.buckley@nist.gov

Abstract*—The brain implements recurrent neural networks (RNNs) efficiently. Modern computing hardware does not. Although specialized neuromorphic systems are well suited for recurrent implementations in the inference phase, training RNNs on-chip is a challenge and not amenable to backpropagation through time (BPTT). To mitigate this mismatch of RNNs and hardware, we propose the application of Multiplexed Gradient Descent (MGD) for model-free learning in recurrent networks with simple operations realizable by a number of neuromorphic systems. MGD does not require analytic differentiation, weight transport, diverse computations, or even a hardware model. Nevertheless, MGD can estimate the true gradient of a loss landscape for any differentiable objective function and is able to do so with multiplexed perturbations. Considerable time advantages are accessible by minimizing the resolution of gradient estimation required for gradient descent. The only operations needed for MGD are those canonically associated with an RNN along with additional mechanisms for global broadcast, local memory, and pseudorandom value generation. In this paper, we demonstrate that MGD converges to the true gradient in agreement with BPTT for a recurrent state space model (SSM), independent of learning task and network initialization. MGD is found to perform gradient descent with efficient gradient approximations on real-world time series data, obtaining a 99% test accuracy on the ECG anomaly dataset and 93% on the full imbalanced multiclass ECG-5000, with similar results over a wide variety of network sizes, initialization, and hyperparameter settings. We show that learning in the recurrent layer is self-regulating for stability by analyzing its ongoing spectral radius. The separability of learned representations by this method are visually and quantitatively analyzed with k-means clustering, principal component analysis, and distance measures. Finally, prospects for neuromorphic hardware implementations are proposed given the modest operation and topological requirements to learn in RNNs with MGD *in-situ*, thereby offering a viable path toward the full neuromorphic advantage of learning in RNNs on-chip.

Work made possible by The National Institute of Standards and Technology and The University of Colorado Boulder

***Index Terms*—neuromorphic computing, neuromorphic algorithms, neuromorphic hardware, recurrent neural networks, model-free, multiplexed gradient descent, perturbative methods**

I. Introduction

TABLE I
A CASE FOR ON-CHIP RNNS TRAINED WITH MGD.

System	Efficient RNNs	BPTT	MGD
Brain	✓	✗	?
CPU/GPU	✗	✓	✓
Simulated Neuromorphics	✗	✓	✓
Physical Neuromorphics	✓	✗	✓

Recurrent neural networks (RNNs) are the natural choice for temporally dependent computing. Certainly, RNNs are nature's choice, as the vast majority of all network topologies in the brain are recurrent [1]–[3]. From a theoretical standpoint, cycles in a network create temporal offsets that are capable of producing higher-dimensional reconstructions according to Taken's theorem [4]. Intuitively, allowing information from previous inputs to persist in a network through recurrence, rather than being immediately extruded through an output layer, readily paints the picture of a memory window for input-history-informed network activity. Indeed, RNNs are the established choice for brain-inspired computing in the time domain [5]–[9] , and have moreover seen a resurgence with state-of-the-art results in sequence processing by way of Legendre Memory Units [10], [11] and later State Space Models (SSMs, Fig. 1) [12]–[14] with strong performance even in the large language model (LLM) domain. While modern SSM results leverage GPU-trainable *approximations* of recurrence, efficient and scalable in-hardware training of true RNNs remains to be seen.

DOI 10.1109/ICONS69015.2025.00025

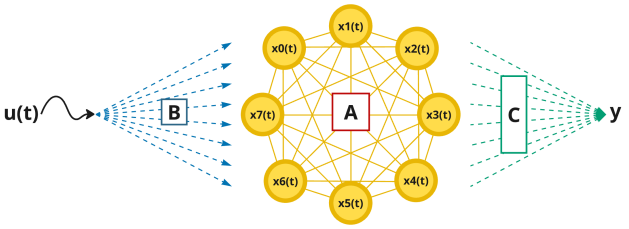

Fig. 1. Input signal $u(t)$ is projected to state $x(t) \in \mathbb{R}^N$ according to projection matrix $B \in \mathbb{R}^N$ where N is the size of the state. $x(t)$ is moreover subject to the recurrent transition matrix $A \in \mathbb{R}^{N \times N}$ at ever step in time. Here, only the final state $x(t = T)$ is mapped by readout C to some output y, which is associated with some objective function to guide learning.

Generally, the motivation to realize temporal computing with brain-inspired efficiency using RNNs is at odds with modern computing hardware. Although iterative matrix multiplication can instantiate a recurrent system in a way well suited to both GPU parallelism [15] and the backpropagation-through-time (BPTT) learning algorithm, it fails to take advantage of in-memory computation with dedicated neural circuit elements as is observed in the brain. Even basic field programmable array (FPGA) implementations of RNNs have been shown to outperform CPUs and GPUs [16], despite that FPGAs are not uniquely designed for RNNs. Though GPUs are the most powerful computing hardware for multilayer perceptrons (MLPs) and convolutional architectures, they still struggle to implement RNNs directly with commensurate efficiency, and this is all the more reason to consider brain-inspired methods on brain-inspired hardware.

Neuromorphic hardware can implement neuronal activations and recurrence through native properties of physical circuit elements, resulting in massive efficiency gains that enable small-footprint, real-time devices that enjoy inherent low power through asynchronous parallelism, in-memory compute, and physical activation functions [17]. However, these extremely efficient implementations of neural networks do not follow the von Neumann architectural scheme and are therefore less poised to implement the diverse and sophisticated computational requirements of backpropagation, and moreover BPTT, *on-chip*. This is because BPTT requires a variety of computation types (including differentiation on well-defined activation functions), weight transport [18] (the topological constraint of backward passes requiring every new downstream weight to be routed to every learnable parameter), and unrolling the recurrent graph in time – none of which are suitable for specialized hardware that involve noisy, unconventional, and even non-differentiable activation functions, dedicated topologies, and in-time in-memory compute.

An alternative solution is to train entirely off-chip with simulated hardware models before transferring weights to the neuromorphic device, or even cycling through on-chip forward passes, off-chip backward calculations, and on-chip updates. These approaches suffer from susceptibility to inaccurate hardware models, imperfect weight transfer techniques, device defects, and noise. The motivation for on-chip, *in-situ* training methods is therefore great because learning on the hardware itself, and around its associated noise and defects, circumvents much of the aforementioned susceptibilities by including these properties in the learning procedure. The limiting factor thus becomes what learning algorithms can be instantiated with the limited (but efficient) operation scope of neuromorphic hardware? A variety of choices have been explored [19]–[21], but a definitive solution to *in-situ* training in neuromorphic systems, and especially those implementing RNNs, is not yet salient. This claim is evidenced by the sparsity of experimental papers that demonstrate recurrent on-chip neuromorphic learning, though promising efforts have been made [22]–[25].

We here propose an existing learning algorithm, multiplexed gradient descent (MGD) [26], [27], as a viable solution to the mismatch between the potential neuromorphic advantage of implementing RNNs and the neuromorphic disadvantage of on-chip learning in RNNs, thus advancing toward a caveat-free neuromorphic advantage in modern computing (see Table I). MGD requires only simple operations and topologies in the form of random number generation, global broadcast, and local memory, all of which are commonly found in neuromorphic systems (as we will examine in Sec. V), and are present in the brain in the form of stochasticity, neuro-modulation, and synaptic storage respectively.

Perturbative methods are well known to reliably estimate the true gradient with respect to an objective function for sufficiently small perturbation sizes and have strong theoretical grounds for this guarantee [28]. The mechanics of perturbative learning are most obviously understood by the finite difference method [29], whereby a single parameter of a system is randomly perturbed by some value ϵ, whose contribution to the change in global cost is perfectly isolated. By iterating over all parameters, the method converges to the true gradient in the limit of $\epsilon \to 0$ and is guaranteed to work for *any* differentiable function with a tractable gradient. Importantly, perturbative methods of this sort are model-free (i.e., the function-to-differentiate need not be known). MGD introduces an approach to multiplex (parallelize) these perturbations in a time-efficient manner by employing random $\pm\epsilon$ perturbations at every learnable parameter simultaneously, and by moderating three key hyperparameters:

1) τ_θ: Number of iterations to integrate into a gradient estimation before making an update.
2) τ_p: Number of iterations per set of perturbation values.
3) τ_x: Number of iterations per training example.

To execute gradient descent in a neural network using MGD, it is often sufficient to batch a set of training examples ($\tau_\theta =$ batch size), make a random $\pm\epsilon$ perturbation to all parameters ($\tau_p =$ batch size) only once before making an update ($\tau_x = 1$) [27], and this happens to be on the upper end of operation efficiency for perturbative gradient descent, roughly $\mathcal{O}(N_\Theta \cdot iter)$, where N_Θ is the total number of parameters and $iter$ is the total number of training iterations. The method thus benefits from efficient multiplexing of perturbations and updates.

The key insight as to why MGD works is that over time the history of random perturbation signs for a given parameter becomes increasingly unique to that parameter, and likewise updates to that parameter become increasingly uniquely correlated to its true contribution to the cost. According to this insight, and by [28], this method is guaranteed to approximate the true gradient for any differentiable system, regardless of system topology, activation functions, hardware defects, and/or noise (given an appropriate noise-to-ϵ ratio). This then should apply (as will be demonstrated) to recurrent topologies with operations confined to those afforded by specialized neuromorphic hardware implementations.

The MGD learning algorithm starts by accumulating an estimate of the gradient,

$$\nabla_i \leftarrow \Delta C \theta_i \tag{1}$$

where ∇_i is the i^{th} component of the gradient, ΔC is the change in cost before and after multiplexed perturbations are applied, and θ_i is the sign and magnitude of the perturbation to the learnable parameter associated with the i^{th} component of the gradient. The estimation is accumulated over a number of iterations designated by the hyperparameter τ_θ, and stored in a distributed component-wise fashion before the simple update

$$\Theta \leftarrow -\eta \nabla / \tau_\theta \tag{2}$$

is applied to the set of all learnable parameters Θ according to learning rate η.

The work presented here is the first application of MGD toward learning in RNNs, and we therefore make contact with where RNNs are currently seeing the greatest advantages in modern computing. Legendre Memory Units (LMUs), and later State Space Models (SSMs), utilize principled structuring of an input-to-hidden projection matrix, and linearly recurrent hidden layer, to recoup coefficients to the Legendre polynomials in the form of neuronal activations for a specified sliding window of time over a signal or sequence. These dynamics accommodate improved history-informed representations at any given moment in the recurrent state, and therefore promote more effective temporal learning and smoother gradient initializations. Adopting the common notation of SSM literature, we describe (and depict, in Fig. 1) the associated system of equations accordingly,

$$x'(t) = Ax(t) + Bu(t) \tag{3}$$

$$y(t) = Cx(t) + Du(t), \tag{4}$$

where B projects temporal input $u(t)$ to state space $x(t)$, which is itself subject to the recurrent transition matrix A. The readout map C projects $x(t)$ to some output $y(t)$. Note, that A and B are discretized as a function of input length and simulation time step (according to [12]). As is typical in SSMs, $D := 0$ and C can represent any sort of readout mapping (linear, nonlinear, multilayer, etc.), which often uses only the final state for classification, and in which case we refer to $y(t)$ instead as y. Dropping D, Equation 4 becomes $y = Cx(t = T)$, where T is the length of input. Learning can take place on any of these active parameters (A, B, and C) with the objective function being associated only with the final output y. We indeed show that MGD can train the entire system, (projection B, recurrence A, and readout C). Moreover, MGD is found to increase machine learning performance and quantitatively improve separability in state representations of input.

When considering the relationship of contemporary SSM literature to an operation-limited learning algorithm for training an SSM in its *recurrent* form, it is important to note that although SSM architectures like S4 [13] employ efficient approximations of recurrent dynamics according to a convolutional kernel on unrolled time, they do not instantiate learning in physical in-memory recurrence. Specialized, naturally recurrent hardware (like the brain), stands to gain from efficient implementations of RNNs rather than kernel approximations, which are non-physical and require sophisticated training techniques that would not readily transfer to specialized hardware. This work offers a proof of concept for on-chip training of physical RNNs that should in theory have access to some of the many impressive results of modern SSMs on GPUs. We show that MGD converges to the true recurrent gradient, independent of parameter count and network topology, therefore making a strong suggestion that these findings should scale to the greater scopes implementable in specialized neuromorphic hardware.

We thus propose a method for performing gradient descent in RNNs using operations that are conducive to on-chip implementation in neuromorphic systems. In Section II, we demonstrate empirically that MGD converges to the true gradient for the entire state space architecture, including the recurrent layer. In Section III, we apply this finding toward machine learning performance on real data and investigate the learning dynamics of these results according to the ongoing spectral radius of the recurrent matrix. The class separability of state representations is scored with k-means clustering and visualized with principle component analysis (PCA). Finally, appropriate candidates for hardware implementations and suggestions for future work are proposed in Section V.

II. Gradient Convergence

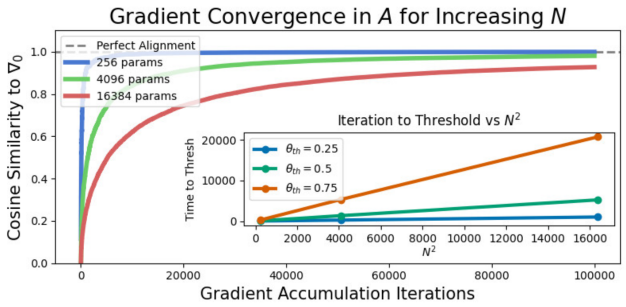

Fig. 2. Alignment of estimated gradient with true gradient over time for increasing state size N. A grows with N^2. The inset shows time-to-threshold over increasing parameter count (N^2), for different selected thresholds.

The first step in demonstrating the efficacy of MGD gradient estimation for recurrent systems is to examine the alignment of estimated gradients with the true gradient ∇_0, which is here computed with BPTT on random samples from the ECG-5000 dataset. In every case ∇_0 is corroborated in the perturbative context with the finite difference (FD) method in the limit of $\epsilon \to 0$. It is found that BPTT and FD are always in alignment. Although the FD is an excellent gradient estimator, it requires an entire forward pass *per parameter*, so at best it may here serve as a watermark for perturbation-derived gradient estimation, and not as a viable hardware-efficient learning algorithm itself.

Fig.2 shows the MGD gradient approximation method converging to the true backpropagation-derived gradient for all parameters associated with the recurrent matrix A, which scale with the state size N according to N^2. Note, the full A-B-C architecture is used in the forward pass and the gradient is estimated specifically for A through multiplexed perturbations on its respective parameters only. Again, referring the reader to [27], it is important to realize that gradient descent *does not require the true gradient*, but rather only an approximation that is in positive alignment (the same direction) as the true gradient. This insight is also the nature of stochastic gradient descent (SGD) [30], one of the most common variants on pure backpropagation in machine learning. We see here that MGD indeed is immediately in positive alignment, even after just one iteration. Therefore it is sufficient to set a low threshold for alignment convergence to benefit from improved scaling (see Fig. 2 inset). The MGD estimated gradient should always be in alignment with the true gradient, so long as a perturbation is not made exactly orthogonal to the gradient. This is because a perturbation that either increases or decreases the cost equivalently gives the same amount of information. Even for a single parameter, a perturbation that is orthogonal to the cost is extremely unlikely. A smooth, descending loss landscape, for example, would have only exactly two directions that could be orthogonal to the gradient. *En mass* it would be virtually impossible that perturbation information in the cost would be outweighed by orthogonal noise.

It is also found that the B and C parameter groups converge to the true gradient according to MGD, including in the presence of up-or-downstream recurrence. This finding prompts an investigation into a layerwise training scheme (collecting the gradient for each layer at every iteration separately), and it is found that all parameters converge together. Simultaneous gradient estimation across all layers does result in a reduced final alignment with respect to the total gradient, which is likely due to weight diffusion, but the alignment is still always positive and therefore sufficient for effective gradient descent, which we will see in Sec. III. All of these findings extend to batching in an SGD fashion (i.e., $\tau_\theta = \tau_p =$ batch size, and $\tau_x = 1$). Like SGD with backpropagation, batch size does negatively correlate with gradient accuracy (linearly), and like with backpropagation, this is found to be an asset for more regularized learning. In the case of RNNs trained with MGD, this may even result in better recurrent stability, as we will also explore in Sec. III.

Having seen that MGD accurately converges on the true gradient, and is already in positive alignment with the true gradient in early iterations, we may now apply these findings to recurrent learning in RNNs on real data.

III. Results and Analysis

To give insight on training RNNs with MGD, we here consider a number of architectures, analytics, and visualizations. Precise details on datasets and implementation techniques are available in section IV and moreover through the public git repository associated with this work.

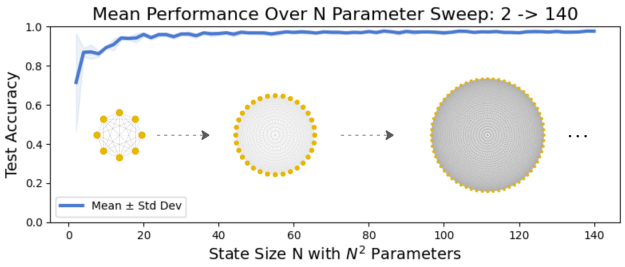

Fig. 3. ECG-anomaly performance for increasing N over multiple initializations (five seeds each with randomness in the readout and perturbation draw). It can be seen that the variance is fairly low, and the performance is fairly high, even for small state sizes. This suggests a robustness to random variations in the model.

Please refer to Fig. 1 for a diagrammatic overview of the general architecture used in the results below, adopted from state space model literature. For the entire sequence length input is projected to the state $x \in \mathbb{R}^N$ according to $B \in \mathbb{R}^N$ and is subject to the recurrent transition matrix $A \in \mathbb{R}^{N,N}$, both of which are pre-derived and discretized according to the HiPPO initialization [12]. Once the input phase is complete, the final state $x(t = T)$ is mapped to some output y according to readout C and subsequently to an objective function (e.g. cross-entropy loss). C can stand for any number of readouts, such as a multilayer perceptron, or a linear readout, all of which are suitable for training with MGD. We refer to the trainable parameters in a learning regime accordingly (e.g. an ABC-training model, or a C-only model).

A. Learning

As a first look into training RNNs with MGD, we consider the 5000-sample ECG-Anomaly detection task, which involves natural data in the time domain. We see this dataset is handily solved for a range of state sizes in Fig. 3, implicating trainability of both small and large recurrent networks with MGD. Fig. 4 visualizes the system as it learns to separate heartbeat classes with a test accuracy of 99%. In all cases, $A, B,$ and C are trained together, and it is found that the model could utilize either a linear single-layer readout map or a nonlinear MLP with to a roughly equivalent effect.

Importantly, the temperament of the recurrent layer proves to be fairly self-regulating when perturbed with MGD. Fig. 4 demonstrates the stable behavior A over the course of learning

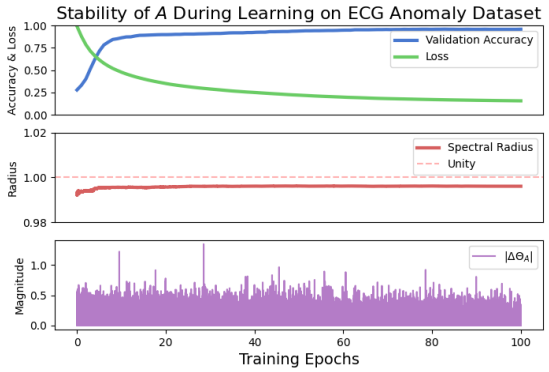

Fig. 4. (Top) Validation accuracy and loss during training. (Middle) Ongoing spectral radius of A during training, which proves to be stable. (Bottom) The magnitude of updates being made to A, implying stability is not due to a placid update regime.

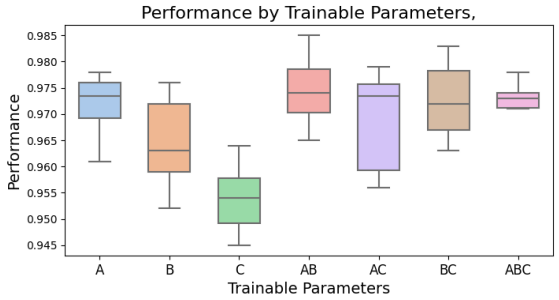

Fig. 5. Hyperparameter sweep over all possible sets of trainable parameters with state size limited to $N = 32$ (to better tease out performance and for faster run time), with ten random seeds rerun for each configuration. Models all use a two-layer [N,N,1] MLP as their readout map C and learning rate is always normalized according to the parameter count and batch size (see IV). Generally, all configurations perform well and always tested over 93%. This is a typical box plot where the box center line corresponds to median, the upper and lower box bounds to the upper and lower quartiles, the whiskers to a 1.5x interquartile range, and circle-markers to outliers.

according to its spectral radius (its absolute maximum eigenvalue). When training RNNs, this is not always guaranteed, as subtle changes in the recurrent layer can result in runaway dynamics that disable learning. Even BPTT often requires optimizers to maintain such balance. Generally, it is advisable to enforce the spectral radius of a fixed recurrent system to unity [6], as anything else should compound recurrent updates on themselves, resulting in runoff dynamics. Conversely, a small spectral radius can result in system dissipation that is too rapid to use for a learning scheme. However, it is observed with the MGD learning algorithm that the spectral radius tends to hover near unity, either above or below, without running off. This is an unexpected and convenient result which may follow from MGD's noisy estimations of the true gradient providing natural regularization. If the parameters make contact with a runoff region in the gradient, there would likely be several update iterations available to recognize this threat to cost minimization and move toward more stable regions. This reasoning is supported by how the gradient alignment becomes increasingly noisy as performance rises and the perturbation-to-loss ratio $\epsilon/cost$ increases. The resulting stability is observed over a wide range of parameter settings, loss functions, and readout maps (both linear and nonlinear), suggesting a rich parameter space for which MGD training responds smoothly to input.

In support of observed stability in A, we can refer to Fig. 5 and see that under a variety of conditions and initializations, training that involves the recurrent layer consistently performs well. A proves to be the most powerful single-trainable layer configuration, and A is involved in the top three highest performance means and the highest absolute performance. Training A, B, and C together results in the most stable performance with the least variance. Training C alone is the weakest classifier (though this still performs well), even given that it takes the pure HiPPO initialized SSM outputs as input. Notably, training B and C together results in strong performance, though B is associated to the objective function only through A, so the recurrent component is still integral. A is also stable for increasing state size N, and does improve performance with size, as seen in Table II. Given these results, and robustness seen to state size and random seeds in Figs. 3 and 5, we can expect on an empirical basis that training A with MGD is prone to stability. Importantly, this same stability *is not* found using BPTT, which only was able to produce similar performance when using the Adam optimizer [31], a method unlikely to match hardware using dedicated circuits for computation.

TABLE II
A FEW HIGHLIGHTED PERFORMANCES.

dataset	N	C	loss	Accuracy
ECG-Anomaly	128	Linear	MSE	99.0%
ECG-Anomaly	128	MLP	CE	99.0 %
ECG-Anomaly	32	MLP	CE	98.5 %
ECG-5000	256	MLP	CE	93%

B. Separability

From a theoretical standpoint, for a given input signal over a specified sliding window in time, a state space is recouping the Legendre polynomial coefficients associated with that signal in its state activations to an order corresponding to the number of neurons in the state. This encodes history-related temporal information in the state vector at any given moment in time. While this is precisely the case at initialization, once training begins, state space dynamics will have no such guarantees. It becomes a question of interest then what sort of states are produced and what is their quality with respect to the learning task? The latter requires a metric for success in terms of state quality, which is readily available for estimation by a number of proxy metrics, all of which may be cast as statements of *separability*, by which we mean how easy is it to separate different states produced by the RNN with respect to their associated input class?

A first metric, as we have already seen, is of course raw classification performance. States more amenable to the classification task will be more classifiable because they are more separable. To isolate this metric from the readout map C, which is itself trained by MGD, we may also employ a simple logistic regression classifier over all states generated in response to a dataset to estimate the linear separability of the states produced before and after the MGD training procedure. We find that having trained A during a full ABC-training round essentially always improves logistic regression performance, and by as much as 5%, which we here take as a metric for improved linear separability.

Another method to assess state quality in terms of the separability of class representations is by way of K-means clustering, an unsupervised learning algorithm that clusters inputs optimally over a given number of centroids by ascertaining the centroid locations that produce the minimum distances to every data point. A classification accuracy can furthermore be produced by determining which centroid-to-label pairings result in the best classification. So as not to conflate this with our MGD model performances (which are better), we will refer to this as a separability score, because this is a good metric for determining how well separated class-associated data points are in any high dimensional space. Fig. 6 well illustrates the improved separability of state space representations for the *multiclass* ECG-5000 dataset (details in Sec. IV). We see a considerable boost in separability score (over 10% increase) as a function of MGD training, which itself produces a training accuracy of 93% with 256 states.

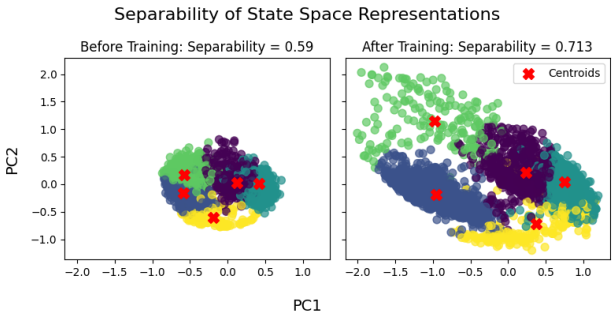

Fig. 6. Final state vectors for every input in the multiclass ECG-5000 test set projected onto their first two principle components, and grouped according to the k-means clustering algorithm. The left plot is before the recurrent matrix A was subject to MGD training and the plot on the right is after. It can be qualitatively seen that training improves and quantitatively measured by the improved separability score (k-means accuracy).

C. Dynamics

Beyond separability of the final state representation, a question also arises as to what sorts of dynamics are at play in higher-dimensions over time. PCA enables us to project these higher dimensions onto their components of maximum variance, and if this variance is taken across all states over all time and all inputs, we can appropriately compare the trajectories of high-dimensional input-driven state representations over classes. Referring to Fig. 7, we see that indeed states take on class-specific dynamics after being trained with MGD, and moreover that class separability across time is improved by training with MGD. Before training, class representations encircle each other throughout input duration and are often in close proximity (including in the final state). After training, it can be seen that class trajectories immediately separate from their common starting point and continue to separate throughout the input duration.

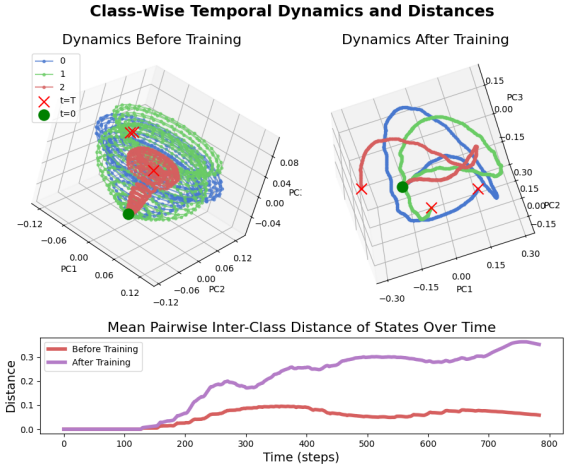

Fig. 7. (Top) Mean state trajectories by class before and after training with MGD on sequential MNIST digits. Qualitatively, it is clear that paths are better separated post training. (Bottom) Quantifying temporal separability with mean pairwise inter-class distances of state representations over time. This is a proxy measure for separability and we see it is improved by training with MGD.

IV. Methods

A. Simulations

Because training state space models in their recurrent (non-convolutional) form is not common practice, a codebase has been developed for this express purpose, and is publicly available at https://github.com/ryangitsit/recurrent-mgd, with directions to replicate primary results of this paper.

Simulations of recurrent neural networks can be executed in a number of ways, with the forward Euler method being the most straightforward choice and requiring the fewest steps of approximation. This, and the requirement to broadcast functions over non-trivial structures, make the JAX programming language a natural choice. A single simulation step directly follows equations 3 and 4, which can be batched in parallel over many examples simply by adding another dimension to input u and output y. Time is iterated over with the *jax.lax.scan* function and parameter updates are broadcast with *jax.tree.map* to all parameters at once.

B. Training and Testing

The objective function is applied to the final output of $y(t = T)$ as mapped according to readout C from the

TABLE III
MGD-RELATED OPERATION SCOPE OF SELECTED NEUROMORPHIC HARDWARE

Hardware	On-Chip	Local Memory	Analog	Recurrent	RNG	Broadcast	MGD-RNN Potential
Intel Loihi 2	✓	✓	✗	✓	✓	On/Off Chip	✓
BrainScaleS-2	✓	✓	✓	✓	✓	On/Off Chip	✓
SpiNNaker (1 & 2)	✗	✓	✗	✓	✓	Off Chip	✓

final internal RNN state $x(t = T)$. Therefore, the temporal components of the system are the input signal, the state to which the input signal is projected, and the state transitions. This is a typical scheme in SSMs and a reasonable approach for hardware implementations.

C. PCA and K-Means Clustering

K-Means clustering is used in a typical unsupervised fashion and accuracy on the test set is derived by optimally assigning classes to each cluster post-hoc. This is all done as a proxy for estimating how separable states are in high-dimensional space in order to quantitatively complement the visual separability of plotting PCA trajectories, which are themselves averaged over all state-space-paths for each sample with respect to class for the entire sequential MNIST dataset.

D. Datasets

The ECG-Anomaly dataset has 5000 samples, each of length 140 ms (and 140 data points), with a roughly even split of normal and anomalous heartbeats. The full multiclass ECG-5000 includes more resolved classification over the anomalies for a total of five classes, which are steeply unevenly distributed. The training set has only 500 samples, and the test set, 4500. Few shot learning is required on the least-common classes.

E. Learning Rate

Explicit learning rates, and potentially decay, are the most straightforward to implement in specialized hardware, and we therefore constrain ourselves to only these techniques for deriving η. While it is expected that performance should improve with optimizers borrowed from machine learning, it is unreasonable to expect these multifaceted computations to sit well into dedicated circuits. We instead use the normalization equation from [27], except for that we count the number of parameters K according to how many times each weight is touched for one sample through time. Thus, we set

$$\Gamma = c \cdot \delta^2 \sqrt{K} \sqrt{1 + (K-1)/\tau_\theta} \tag{5}$$

$$\eta := 1/\Gamma \tag{6}$$

where δ is the magnitude of perturbation size being used globally, τ_θ is essentially batch size, and c is a variable coefficient, but is almost always set to the number of classes in the dataset.

V. DISCUSSION AND HARDWARE

Having seen that SSM-RNNs can be trained with MGD using only a simple set of operations, minimal iterations, varying state sizes, and for a large range of hyperparameters, the question then becomes *what tasks can this method scale to?* Given that MGD has been shown to estimate the true end-to-end gradient for an SSM, it is reasonable to aim for many of the impressive results and scales produced in SSM literature, culminating in competitive LLM performance. However, such results involve kernalized approximations of RNNs (reducing $N^2 \rightarrow N$) and are heavily optimized for GPUs, which is another case of adapting AI to modern hardware. However, there are an increasing number of efforts to develop hardware for the explicit purpose of intelligence, and especially brain-inspired intelligence, in the form of neuromorphic computing.

MGD generally offers a viable path for training unconventional systems, and the work presented here extends this viability to RNNs, where neuromorphic computing stands to shine most over conventional hardware by virtue of natural recurrence, in-memory compute, and physically instantiated native activation functions. Table III includes just some of the forerunners in the neuromorphic hardware field [32]–[34], all of which host the operations necessary to implement MGD toward the training of recurrent networks. This list is not necessarily comprehensive, but rather an example of how to map hardware features to MGD requirements. Candidate in-development hardware also exists, such as superconducting optoelectronic networks [35], [36]. All of the above offer spiking communication (but not exclusively) and it is therefore of interest that spiking-SSMs have been shown to outperform even transformers on long-range dependency sequence tasks [37], and given that for the selected spiking dynamics surrogate functions were sufficient to train the system with BPTT, it should follow that this spiking architecture has sufficiently smooth gradients to be trained with MGD on-chip.

A natural extension for future work is to make these suggested hardware implementations a reality. Even prior to hardware, further extensions in the simulation domain can be accessed. Simulated spiking RNN implementations will advance the coupling of MGD to existing hardware efforts. Deep SSMs in the large language model domain would also strengthen the case for scalability. However, as neither RNNs nor MGD are best suited for simulations, it is of paramount interest to implement these methods on neuromorphic systems first and foremost. This may grant access to speeds and scales previously unattainable, and perhaps not foreseeable. The combination of *in-situ* training, RNNs, and specialized hardware,

has not yet been realized and cannot be well mimicked on conventional computers. Thus, we outline an opportunity to train RNNs with MGD on-chip, and to potentially fulfill the promise of neuromorphic advantage with novel forms of machine intelligence.

ACKNOWLEDGMENT

This work was made possible by the institutional support from the National Institute of Standards and Technology and the University of Colorado Boulder.

DISCLAIMER

Commercial hardware are mentioned to contextualize algorithm applicability, but this does not imply endorsement of any product or service by NIST.

REFERENCES

[1] Rodney J Douglas and Kevan AC Martin. Recurrent neuronal circuits in the neocortex. *Current biology*, 17(13):R496–R500, 2007.

[2] Xiao-Jing Wang. Synaptic reverberation underlying mnemonic persistent activity. *Trends in neurosciences*, 24(8):455–463, 2001.

[3] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

[4] Floris Takens. Lecture notes in mathematics. In *DA Rand and LS Young Springer, Berlin*, volume 898, page 366. 1981.

[5] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[6] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. *Advances in neural information processing systems*, 15, 2002.

[7] Xiaoran He, Ting Liu, Fatemeh Hadaeghi, and Herbert Jaeger. Reservoir transfer on analog neuromorphic hardware. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 1234–1238. IEEE, 2019.

[8] Alejandra Patiño-Saucedo, Hugo Rostro-González, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. Liquid state machine on spinnaker for spatio-temporal classification tasks. *Frontiers in Neuroscience*, 16:819063, 2022.

[9] Rishabh Patel, Vivek Saraswat, and Umesh Ganguly. Liquid state machine on loihi: Memory metric for performance prediction. In *International Conference on Artificial Neural Networks*, pages 692–703. Springer Nature Switzerland, 2022.

[10] Aaron Voelker, Ilja Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

[11] Rahul Gaurav, Terrence C Stewart, and Yiran C Yi. Spiking reservoir computing for temporal edge intelligence on loihi. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 526–530. IEEE, 2022.

[12] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

[13] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[14] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[15] Jeremy Appleyard, Tomas Kocisky, and Phil Blunsom. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946*, 2016.

[16] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016.

[17] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.

[18] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

[19] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

[20] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

[21] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[22] Charlotte Frenkel and Giacomo Indiveri. Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[23] Uicheol Shin, Masatoshi Ishii, Atsuya Okazaki, Megumi Ito, Malte J Rasch, Wanki Kim, Akiyo Nomura, Wonseok Choi, Dooyong Koh, Kohji Hosokawa, et al. Pattern training, inference, and regeneration demonstration using on-chip trainable neuromorphic chips for spiking restricted boltzmann machine. *Advanced Intelligent Systems*, 4(8):2200034, 2022.

[24] Qingtian Zhang, Huaqiang Wu, Peng Yao, Wenqiang Zhang, Bin Gao, Ning Deng, and He Qian. Sign backpropagation: An on-chip learning algorithm for analog rram neuromorphic computing systems. *Neural Networks*, 108:217–223, 2018.

[25] ERW Van Doremaele, X Ji, J Rivnay, and Y Van De Burgt. A retrainable neuromorphic biosensor for on-chip learning and classification. *Nature Electronics*, 6(10):765–770, 2023.

[26] Adam N McCaughan et al. Multiplexed gradient descent: Fast online training of modern datasets on hardware neural networks without backpropagation. *APL Machine Learning*, 1(2), 2023.

[27] Bakhrom G Oripov, Andrew Dienstfrey, Adam N McCaughan, and Sonia M Buckley. Scaling of hardware-compatible perturbative training algorithms. *arXiv preprint arXiv:2501.15403*, 2025.

[28] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[29] Lewis Fry Richardson. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 210(459-470):307–357, 1911.

[30] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[33] Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning-keynote presentation. In *Communicating Process Architectures 2017 & 2018*, pages 277–280. IOS Press, 2019.

[34] Christian Pehle, Sebastian Billaudelle, Benedikt Cramer, Jan Kaiser, Katharina Schreiber, Yannik Stradmann, et al. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16:795876, 2022.

[35] Saeed Khan, Bryce A Primavera, Jeff Chiles, Adam N McCaughan, Sonia M Buckley, Alexander N Tait, Adriana Lita, John Biesecker, Anna Fox, David Olaya, et al. Superconducting optoelectronic single-photon synapses. *Nature electronics*, 5(10):650–659, 2022.

[36] Bryce A Primavera, Saeed Khan, Samuel R Adler, and Jeffrey M Shainline. Programmable synapses and dendritic circuits for superconducting optoelectronic neuromorphic computing. In *2024 International Conference on Neuromorphic Systems (ICONS)*, pages 277–281. IEEE, 2024.

[37] Matei-Ioan Stan and Oliver Rhodes. Learning long sequences in spiking neural networks. *Scientific Reports*, 14(1):21957, 2024.