LETTER Communicated by Terrence Sejnowski

Learning Fixed Points of Recurrent Neural Networks by Reparameterizing the Network Model

Vicky Zhu
vzhu@babson.edu
Babson College, Mathematics, Analytics, Science, and Technology Division, Wellesley, MA 02481, U.S.A.

Robert Rosenbaum
Robert.Rosenbaum@nd.edu
University of Notre Dame, Department of Applied and Computational Mathematics and Statistics, Notre Dame, IN 46556, U.S.A.

In computational neuroscience, recurrent neural networks are widely used to model neural activity and learning. In many studies, fixed points of recurrent neural networks are used to model neural responses to static or slowly changing stimuli, such as visual cortical responses to static visual stimuli. These applications raise the question of how to train the weights in a recurrent neural network to minimize a loss function evaluated on fixed points. In parallel, training fixed points is a central topic in the study of deep equilibrium models in machine learning. A natural approach is to use gradient descent on the Euclidean space of weights. We show that this approach can lead to poor learning performance due in part to singularities that arise in the loss surface. We use a reparameterization of the recurrent network model to derive two alternative learning rules that produce more robust learning dynamics. We demonstrate that these learning rules avoid singularities and learn more effectively than standard gradient descent. The new learning rules can be interpreted as steepest descent and gradient descent, respectively, under a non-Euclidean metric on the space of recurrent weights. Our results question the common, implicit assumption that learning in the brain should be expected to follow the negative Euclidean gradient of synaptic weights.

1 Introduction

Recurrent neural network models (RNNs) are widely used in machine learning and computational neuroscience. In machine learning, they are typically used to learn dynamical responses to time series inputs. In computational neuroscience, RNNs are sometimes used to model dynamical responses of neurons to dynamical stimuli (Durstewitz et al., 2000; Vogels et al., 2005; Sussillo & Abbott, 2009; Sussillo, 2014; Rajan et al., 2016; Song

Neural Computation **36, 1568–1600** (2024)
https://doi.org/10.1162/neco_a_01681

et al., 2017; DePasquale et al., 2018; Yang et al., 2019; Pyle & Rosenbaum, 2019; Yang & Wang, 2020; Dubreuil et al., 2022; Márton et al., 2022; Brenner et al., 2022; Valente et al., 2022; Durstewitz et al., 2023), but they are also often used to model stationary, fixed-point neural responses to static inputs. For example, many phenomena observed in visual cortical circuits, such as surround suppression, are widely modeled by stationary states of computational models in which recurrent connections model lateral, intralaminar connectivity (Ferster & Miller, 2000; Ozeki et al., 2009; Rubin et al., 2015; Ebsch & Rosenbaum, 2018; Curto et al., 2019; Baker et al., 2020).

A natural approach to learning fixed points of RNNs is to use direct gradient descent on the recurrent weight matrix after the network has converged toward a fixed point. A direct application of this approach, called "truncated backpropagation through time," can be computationally expensive because it requires the application of backpropagation on a computational graph unrolled over many time steps. Moreover, backpropagation through time is difficult to implement or approximate with biologically plausible models of learning (Lillicrap & Santoro, 2019).

Alternative approaches use the implicit equation for fixed points to derive the exact gradients of the loss with respect to the weight matrix at the fixed point, or some approximations to this quantity (Pineda, 1987; Almeida, 1990; Williams & Peng, 1990; Ollivier et al., 2015; Liao et al., 2018). Studies of deep equilibrium models have generalized the training of fixed points beyond fully connected RNNs to more general classes of models with implicitly defined fixed points (Bai et al., 2019, 2020; Winston & Kolter, 2020). These approaches can also be computationally expensive and difficult to implement in biologically plausible models because the gradient derived from the implicit equation involves matrix inverses, which either need to be computed directly or approximated using, for example, iterative methods.

In this work, we additionally show that gradient descent on the recurrent weight matrix can lead to poor learning performance because the associated loss landscape has singularities and implicit biases that make it poorly conditioned for gradient-based learning.

In mentioning "gradient descent" above, we were implicitly referring to the Euclidean gradient on weights, which is standard practice. However, several authors have argued that the default use of the Euclidean gradient in gradient descent is not necessarily optimal for studying artificial or biological learning. In machine learning, non-Euclidean gradients informed by information theory, such as the natural gradient, are superior in some settings (Amari, 1998; Amari & Douglas, 1998; Martens, 2020). Moreover, in computational neuroscience, where synaptic weights describe physical properties, the use of a Euclidean gradient implicitly assumes a specific choice of physical units and parameterization of the model (Surace et al., 2020; Kreutzer et al., 2022; Pogodin et al., 2023). Different units or different parameterizations of a biological model will yield different gradients

and ultimately different learning dynamics. For example, synaptic weights in the brain can be parameterized in units of conductance, current, the integral of current across time, voltage, the integral of voltage across time, the probability of vesicle release, or more detailed biophysical parameters. Each of these parameterizations potentially produces gradients pointing in different directions. These issues were explored in more depth by Surace et al. (2020). Hence, gradient descent using the Euclidean gradient of the loss with respect to synaptic weights under a specific choice of parameterization might not capture learning dynamics or learned representations in biological neuronal networks.

In this work, we derive two new learning rules for fixed points of recurrent neural networks by reparameterizing the network model. The first learning rule can be viewed as steepest descent with respect to a non-Euclidean metric. The second rule approximates the first one, but it is more efficient and can be interpreted as gradient descent on a non-Euclidean geometry. We demonstrate empirically that these learning rules exhibit more robust and efficient learning dynamics than standard, Euclidean gradient descent. Robustness is achieved by avoiding singularities that plague standard gradient descent. Efficiency is achieved by avoiding the expensive computation of matrix inverses to compute weight updates. We also find that the parameter updates produced by the new learning rules point in substantially different directions in parameter space than the negative Euclidean gradient.

Our results provide a new, more effective learning rule for training fixed points in RNNs with potential applications for improving the state of the art for deep equilibrium models (Bai et al., 2019). Moreover, the learning rules we derive could be combined with standard gradient descent to improve the training of models with slowly changing input features and long time dependencies, since these properties are approximated by fixed points. In addition, our results question the common, implicit assumption in computational neuroscience that learning should follow the negative Euclidean gradient of synaptic weights.

Code to apply the proposed learning rules and produce all figures in this letter can be found at https://github.com/RobertRosenbaum/LearningFixedPointsInRNNs.

2 Background and Theory

2.1 Model Description. We consider a recurrent neural network (RNN) model of the form (Dayan & Abbott, 2001; Gerstner et al., 2014; Sussillo & Abbott, 2009; Sussillo, 2014)

$$\tau \frac{d\boldsymbol{r}}{dt} = -\boldsymbol{r} + f(W\boldsymbol{r} + \boldsymbol{x}), \tag{2.1}$$

where $\boldsymbol{r}(t) \in \mathbb{R}^N$ is a vector of model firing rates, $\tau > 0$ is a time constant, $W \in \mathbb{R}^{N \times N}$ is a recurrent connectivity matrix, $\boldsymbol{x} \in \mathbb{R}^N$ models external input to the network, and $f : \mathbb{R} \to \mathbb{R}$ is a nonnegative, nondecreasing activation function or f-I curve, which is applied pointwise. For a time-constant input, $\boldsymbol{x}(t) = \boldsymbol{x}$, fixed points are defined by setting $d\boldsymbol{r}/dt = 0$ in equation 2.1 to obtain an implicit equation,

$$\boldsymbol{r} = f(W\boldsymbol{r} + \boldsymbol{x}). \tag{2.2}$$

The stability of fixed-point firing rates from equation 2.1 is determined by the eigenvalues of the Jacobian matrix,

$$\mathcal{J} = \frac{1}{\tau}[-I + GW], \tag{2.3}$$

where $G = \mathrm{diag}(f'(\boldsymbol{z}))$ is a diagonal matrix with entries

$$G_{jj} = f'(\boldsymbol{z}_j)$$

and $\boldsymbol{z} = [W\boldsymbol{r} + \boldsymbol{x}]$ is the vector of neural inputs or preactivations evaluated at their fixed points. Specifically, a fixed point is hyperbolically stable if all eigenvalues of $\mathcal{J}$ have a negative real part.

We can alternatively consider a recurrent neural network model in discrete time of the form

$$\boldsymbol{r}(n+1) = f(W\boldsymbol{r}(n) + \boldsymbol{x}(n)). \tag{2.4}$$

Equation 2.1 is more common in computational neuroscience, while equation 2.4 is more common in machine learning, but they are closely related. Equation 2.4 has the same fixed points as equation 2.2, but hyperbolic stability is obtained when eigenvalues of GW have a magnitude less than 1. Hence, if a fixed point is stable for equation 2.4, it is also stable for equation 2.1, but the converse is not true. In this work, we focus on the continuous system in equation 2.1, but our approach and learning rules can also be applied to the discrete system in equation 2.4.

In machine learning applications, RNNs are often used to learn mappings from input time series, $\boldsymbol{x}(t)$, to output time series, $\boldsymbol{r}(t)$, and they are often trained using backpropagation through time. In computational neuroscience, RNNs of the form in equation 2.1 are often studied for their fixed-point properties—for example, to study orientation selectivity and surround suppression among other phenomena (Ferster & Miller, 2000; Ozeki et al., 2009; Rubin et al., 2015; Ebsch & Rosenbaum, 2018; Curto et al., 2019; Baker et al., 2020), but the weights in these studies are often chosen by hand, not learned. As a combination of these perspectives, we are

interested in learning mappings from static inputs, $\boldsymbol{x}(t) = \boldsymbol{x}$, to their associated fixed points, $\boldsymbol{r}$, given by equation 2.2.

Specifically, consider a supervised learning task with a cost function of the form

$$J(W) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{r}^i, \boldsymbol{y}^i),$$

where $\boldsymbol{x}^i$ is an input, $\boldsymbol{y}^i$ is a label, and $\boldsymbol{r}^i = f(W\boldsymbol{r}^i + \boldsymbol{x}^i)$ is the fixed point that the network converges to under input $\boldsymbol{x}^i$. The loss function, $L(\boldsymbol{r}, \boldsymbol{y})$, could in principle be any real-valued function. In examples below, we consider mean-squared error and categorical cross-entropy loss functions. This learning task presents unique challenges because fixed points are defined implicitly by equation 2.2 instead of explicitly as a function of $\boldsymbol{x}^i$, and also because we only wish to learn stable fixed points. The data set $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_i$ can be the entire data set in the case of full-batch learning or a mini-batch in the case of stochastic learning. Updates to W during learning can be written as

$$W \leftarrow W + \Delta W$$

where

$$\Delta W = \frac{1}{m}\sum_{i=1}^{m} \Delta W^i.$$

Here, ΔW^i is an update rule that can depend on $\boldsymbol{x}^i$, $\boldsymbol{y}^i$, W^i, and $\boldsymbol{r}^i$. Below, we derive and compare three different update rules, ΔW_1^i, ΔW_2^i, and ΔW_3^i, for minimizing J.

2.2 Gradient Descent on the Recurrent Weight Matrix. The first learning rule we consider is direct gradient descent of the loss surface with respect to W using the Euclidean gradient,

$$\Delta W_1^i = -\eta_W \nabla_W L(\boldsymbol{r}^i, \boldsymbol{y}^i), \tag{2.5}$$

where $\eta_W > 0$ is a learning rate and ∇_W refers to the standard, Euclidean gradient with respect to W. If the fixed point, $\boldsymbol{r}^i$, is hyperbolically stable, then the Jacobian matrix from equation 2.3 has eigenvalues with negative real part, so $I - G^i W = -\tau \mathcal{J}$ is invertible and we have (see appendix A.1)

$$\Delta W_1^i = -\eta_W G^i \left[I - G^i W\right]^{-T} (\nabla_{\boldsymbol{r}^i} L) \left(\boldsymbol{r}^i\right)^T, \tag{2.6}$$

where $G^i = \text{diag}(f'(\mathbf{z}^i))$ evaluated at the fixed point and U^{-T} denotes the inverse transpose of a matrix, U. If $G^i_{jj} \neq 0$ for all j, then G^i is invertible so equation 2.6 can be simplified to get

$$\Delta W_1^i = -\eta_W \left[\left[G^i \right]^{-1} - W \right]^{-T} (\nabla_{\mathbf{r}^i} L) \left(\mathbf{r}^i \right)^T . \tag{2.7}$$

Evaluating equations 2.6 and 2.7 directly is computationally expensive because they require the calculation of a matrix inverse. Truncated backpropagation through time and other methods provide alternative approaches to approximating ΔW_1 (Pineda, 1987; Almeida, 1990; Williams & Peng, 1990; Ollivier et al., 2015; Liao et al., 2018), but note that truncated backpropagation through time requires the storage of a large computational graph, making it memory inefficient. Moreover, we show in the examples that follow that using ΔW_1 to update weights can lead to poor learning performance. We next propose an alternative update rule based on a nonlinear reparameterization of the model.

2.3 A New Learning Rule from Reparameterizing the RNN. To motivate the reparameterized model, first consider the special case of a linear network defined by

$$f(z) = z.$$

In this case, $G = I$ is the identity matrix, and equation 2.2 for the fixed point can be written as

$$\mathbf{r} = [I - W]^{-1}\mathbf{x}.$$

This is a linear model in the sense that $\mathbf{r}$ is a linear function of $\mathbf{x}$, but the nonlinear dependence of the cost on W (especially a nonlinearity involving matrix inverses) produces complicated and computationally expensive updates from equation 2.6.

Instead of performing gradient descent with respect to W, we propose instead to first apply a nonlinear change of coordinates to obtain new parameters,

$$A = F(W) := [I - W]^{-1}. \tag{2.8}$$

If we parameterize the model in terms of A instead of W, then fixed points satisfy the standard linear model,

$$\mathbf{r} = A\mathbf{x}, \tag{2.9}$$

which is linear in the input, $\mathbf{x}$, *and* the parameters, A. Gradient descent of the loss with respect to A gives the standard update rule for a linear, single-layer neural network,

$$\begin{aligned}\Delta A^i &= -\eta_A \nabla_A L(\boldsymbol{r}^i, \boldsymbol{y}^i) \\ &= -\eta_A \left(\nabla_{\boldsymbol{r}^i} L\right) \left(\boldsymbol{x}^i\right)^T \\ &= -\eta_A \left(\nabla_{\boldsymbol{r}^i} L\right) \left(\boldsymbol{r}^i\right)^T A^{-T}, \end{aligned} \tag{2.10}$$

where we distinguish between the learning rate, η_A, used for the reparameterized model and the learning rate, η_W, used for the original parameterization. Equation 2.10 gives a gradient-based update to the new parameter, A, but our original RNN model is parameterized by W. To update our original parameters, we need to change the ΔA from equation 2.10 back to W coordinates. To do this, note that we want to find a value for ΔW that satisfies $A + \Delta A = F(W + \Delta W)$ whenever $A = F(W)$ and ΔA comes from equation 2.10. In other words, the update to W is given by

$$\begin{aligned}\Delta W_2^i &= F^{-1}(F(W) + \Delta A^i)) - W \\ &= -\left[[I - W]^{-1} - \eta_A \left(\nabla_{\boldsymbol{r}^i} L\right) \left(\boldsymbol{r}^i\right)^T [I - W]^T\right]^{-1} + I - W, \end{aligned} \tag{2.11}$$

where $F^{-1}(A) = I - A^{-1}$ is the inverse of $F(W)$.

To summarize this approach, if equation 2.11 is used to update parameters, W, under the linear fixed-point model, $\mathbf{r} = f(W\mathbf{r} + \mathbf{x})$ with $f(z) = z$, then the learning dynamics will be identical to standard linear regression of parameters, A, on the model $\mathbf{r} = A\mathbf{x}$.

Since gradient descent with respect to A in equation 2.10 represents the steepest descent of the loss surface in the new parameter space of A and since equation 2.11 gives the same updates in the original parameter space of W, the learning rule in equation 2.5 corresponds to the steepest descent of the cost, $J(W)$, using a non-Euclidean metric defined by

$$d(W_1, W_2) = \|F(W_1) - F(W_2)\|, \tag{2.12}$$

where $\|B\| = \sqrt{\mathrm{Tr}(BB^T)}$ is the Euclidean or Frobenius norm on matrices. Note that $d(\cdot, \cdot)$ is a metric when restricted to the space of all matrices, W, for which $I - W$ is invertible. Hence, if we restrict to W that yields hyperbolically stable fixed points, equation 2.5 corresponds to the steepest descent with respect to a non-Euclidean metric. However, the metric d is not necessarily generated by an inner product, so equation 2.11 cannot be called *gradient* descent since the notion of a gradient requires a metric induced by an inner product. In section 2.4, we show that an approximation to ΔW_2^i produces gradient descent with a non-Euclidean gradient. Moreover, in

section 3, we present examples showing that ΔW_2 is better suited to learning fixed points than the standard approach to gradient descent represented by ΔW_1. But first, we need to generalize the derivation of ΔW_2 to arbitrary activation functions.

Equation 2.11 was derived for the specific case $f(z) = z$, but we can extend it to a model with arbitrary $f(z)$. To do so, we first linearize equation 2.2 to obtain a linearized fixed-point equation,

$$\boldsymbol{r} = G[W\boldsymbol{r} + \boldsymbol{x}], \tag{2.13}$$

which has a closed-form solution given by

$$\mathbf{r} = [I - GW]^{-1}G\mathbf{x}. \tag{2.14}$$

Note, again, that $I - GW$ is invertible whenever $\boldsymbol{r}$ is a hyperbolically stable fixed point.

Given equation 2.14, a natural choice of new parameters would be

$$A = [I - GW]^{-1}G, \tag{2.15}$$

because it would again produce a (linearized) model of the form $\boldsymbol{r} = A\boldsymbol{x}$. Note that under the linear model $f(z) = z$, we have $G = I$, and recover the parameterization in equation 2.8, so equation 2.15 is a generalization of equation 2.8.

However, the update rule to W derived from gradient descent on A from the parameterization in equation 2.15 is susceptible to blow-up or singularities when some values of $G_{jj} = f'(\boldsymbol{z}_j)$ become small in magnitude or zero. This occurs because small changes to A would require large changes to W when G_{jj} is small. And in the extreme case that $G_{jj} = 0$ for some j, updates to W do not affect $\mathbf{r}$ (i.e., $\Delta\mathbf{r}_j = 0$ for any ΔW under the linear approximation $\mathbf{r} = G[W\mathbf{r} + \mathbf{x}]$), so we cannot derive a ΔW to match a given ΔA, that is, the reparameterization in equation 2.15 is ill posed. See appendix A.2 for more details on these problems with this particular reparameterization.

To circumvent these problems, we instead take the parameterization

$$A = F(W) := [G - GWG]^{-1} \tag{2.16}$$

in place of equation 2.15. Under the linearized fixed-point equation in equation 2.13, we then obtain the linear model

$$\mathbf{r} = GAG\mathbf{x},$$

which generalizes equation 2.9. This equation is linear in $\boldsymbol{x}$ and in the new parameters, A. Hence, learning A is again a linear regression problem, albeit

with the extra G terms. These extra G terms prevent singularities and blow up when G_{jj} terms become small or zero (see appendix A.2). Under the simple linear model $f(z) = z$, we have $G = I$ and recover the parameterization in equation 2.8, so that equation 2.16 (like equation 2.15) is a generalization of equation 2.8.

Note that each input (i.e., each i) will potentially have a different gain matrix, $G^i = \mathrm{diag}(f'(\mathbf{z}^i))$, so each sample will have a potentially different value of $A^i = [G^i - G^i W G^i]^{-1}$ as well. The gradient-based update of the loss, $L(\boldsymbol{r}^i, \boldsymbol{y}^i)$, with respect to A^i for each sample becomes

$$\begin{aligned}\Delta A^i &= -\eta_A \nabla_{A^i} L(\boldsymbol{r}^i, \boldsymbol{y}^i) \\ &= -\eta_A G^i \left(\nabla_{\boldsymbol{r}^i} L\right) (\boldsymbol{r}^i)^T [G^i]^{-1} A^{-T}.\end{aligned}$$

Using the same approach for deriving equation 2.11, we can again derive an update to W given by

$$\begin{aligned}\Delta W_2^i &= F^{-1}(F(W) + \Delta A^i) - W \\ &= -\left[\left[I - G^i W\right]^{-1} G^i - \eta_A \left[G^i\right]^2 \left(\nabla_{\boldsymbol{r}^i} L\right) (\boldsymbol{r}^i)^T \left[I - G^i W\right]^T G^i\right]^{-1} \\ &\quad + \left[[G^i]^{-1} - W\right]. \end{aligned} \tag{2.17}$$

This update can only be evaluated directly in the situation where $G^i_{jj} \neq 0$ for all j so that the inverse of the gain matrix, G, exists. However, note that $[W_2^i]_{jk} \to 0$ as $G^i_{jj} \to 0$, as expected, so in situations where $G^i_{jj} = 0$, it is consistent to take $[W_2^i]_{jk} = 0$. Note also that equation 2.17 is equivalent to equation 2.11 whenever $G = I$, as expected, since equation 2.17 generalizes equation 2.11 to the case of arbitrary f.

We additionally remark that we considered another reparameterization given by

$$A = F(W) = [I - WG]^{-1}, \tag{2.18}$$

which gives the linearized model

$$\boldsymbol{r} = GA\boldsymbol{x}.$$

The update rule derived from equation 2.18 avoids the blow-up of updates when G_{jj} is small or zero (similar to equation 2.17 derived from equation 2.16). Moreover, equation 2.18 has a nice interpretation because linearizing the fixed-point equation for $\boldsymbol{z} = W\boldsymbol{r} + \boldsymbol{x} = W f(\boldsymbol{z}) + \boldsymbol{x}$ gives the simple expression $\boldsymbol{z} = A\boldsymbol{x}$ under equation 2.18. However, we found empirically that the learning rules derived from equation 2.18 tend to underperform those derived from equation 2.16, so we use equation 2.16.

2.4 A Simpler Learning Rule from Linearizing the Reparameterized Rule. The reparameterized rule in equation 2.17 is a rather complicated learning rule, and the matrix inverses can be computationally expensive to compute or approximate. If we assume that $\eta_A > 0$ is small, then we can approximate equation 2.17 by applying Taylor expansion to linear order in η_A. This gives the linearized parameterized rule (see appendix A.3 for details),

$$\Delta W_3^i = -\eta_A \left[I - WG^i\right] G^i \left(\nabla_{\mathbf{r}^i} L\right) (\boldsymbol{r}^i)^T \left[I - G^i W\right]^T [I - G^i W]. \tag{2.19}$$

In contrast to equations 2.6 and 2.17 for ΔW_1^i and ΔW_2^i, equation 2.19 for ΔW_3^i does not require the computation of matrix inverses. Like ΔW_1^i and ΔW_2^i, ΔW_3^i satisfies $\Delta W_{jk} \to 0$ whenever $G_{jj} \to 0$, but unlike equation 2.17 for ΔW_2^i, equation 2.19 for ΔW_3^i can be evaluated directly when $G_{jj} = 0$ for some j.

The update rule ΔW_3^i converges to ΔW_2^i as in the limit of small learning rate, $\eta_A \to 0$, and the remainder terms are quadratic in η_A. Therefore, for sufficiently small learning rates, the two learning rules should behave similarly (which we will verify empirically below). At larger learning rates, their behavior might differ. At the end of appendix A.3, we use the Taylor coefficients to derive a more precise condition on how small η_A must be for the learning rules to agree.

Notably, ΔW_3^i can be interpreted as gradient descent of the loss function with a non-Euclidean gradient. To see why this is the case, first note that ΔW_3^i is related to ΔW_1^i according to

$$\Delta W_3^i = B^i \Delta W_1^i C^i, \tag{2.20}$$

where

$$B^i = [I - WG^i][I - WG^i]^T$$

and

$$C^i = [I - G^i W]^T [I - G^i W].$$

Here and for the remainder of this section, we take $\eta_A = \eta_W = \eta$ to highlight the relationship between the two update rules, but constant scalar coefficients do not affect these results.

Using equation 2.20, we may conclude that ΔW_3^i is equivalent to gradient descent of the loss with respect to W using a non-Euclidean gradient. To explain this statement in more detail, note that the gradient of $L(\mathbf{r}^i, \mathbf{y}^i)$ with respect to W depends on the choice of metric or geometry (Surace et al., 2020). Given an inner product, $\langle \cdot, \cdot \rangle_a$, on $\mathbb{R}^{N \times N}$, the gradient of a function, $F : \mathbb{R}^{N \times N} \to \mathbb{R}$, on the geometry imposed by $\langle \cdot, \cdot \rangle_a$ evaluated at $W \in \mathbb{R}^{N \times N}$ is

the unique matrix $\nabla^a_W F \in \mathbb{R}^{N\times N}$ such that for every $U \in \mathbb{R}^{N\times N}$ (Spivak, 2018; Surace et al., 2020),

$$\langle \nabla^a_W F, U\rangle_a = \lim_{\epsilon\to 0} \frac{F(W+\epsilon U) - F(W)}{\epsilon}.$$

The standard Euclidean gradient, $\nabla = \nabla^E$, on matrices is given by taking the geometry produced by the Euclidean or Frobenius inner product,

$$\langle U, V\rangle_E = \sum_{jk} U_{jk}V_{jk} = \mathrm{Tr}(UV^T).$$

Recall that ΔW^i_1 is defined by the Euclidean gradient,

$$\Delta W^i_1 = -\eta \nabla^E_W L(\mathbf{r}^i, \mathbf{y}^i),$$

where $L(\mathbf{r}^\mathbf{i}, \mathbf{y}^i)$ is interpreted as a function of W. We claim that

$$\Delta W^i_3 = -\eta \nabla^B_W L(\mathbf{r}^i, \mathbf{y}^i), \tag{2.21}$$

where ∇^B_W is the gradient under the geometry defined by the inner product,

$$\begin{aligned}\langle U, V\rangle_B &= \mathrm{Tr}(B^{-1}UC^{-1}V^T)\\ &= \langle B^{-1}U, VC^{-1}\rangle_E.\end{aligned}$$

Now note that we can use the cyclic property of the trace operator to write

$$\begin{aligned}\langle U, V\rangle_B &= \mathrm{Tr}(B^{-1}UC^{-1}V^T)\\ &= \mathrm{Tr}\left([I-WG]^{-T}[I-WG]^{-1}U[I-GW]^{-1}[I-GW]^{-T}V^T\right)\\ &= \mathrm{Tr}\left([I-WG]^{-1}U[I-GW]^{-1}[I-GW]^{-T}V^T[I-WG]^{-T}\right)\\ &= \langle \mathcal{L}U, \mathcal{L}V\rangle_E,\end{aligned}$$

where $\mathcal{L} : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ is a linear operator on $N \times N$ matrices defined by

$$\mathcal{L}(U) = [I-WG]^{-1}U[I-GW]^{-1}.$$

Hence, $\langle\cdot,\cdot\rangle_B$ can be viewed as a Euclidean inner product on linearly transformed coordinates. This confirms that $\langle\cdot,\cdot\rangle_B$ defines an inner product on square matrices whenever $[I-WG]$ and $[I-GW]$ are nonsingular.

For notational convenience here and below, we do not write the explicit dependence of B, C, or $\mathcal{L}$ on i, but they *do* depend on i through G^i. In other

words, there are distinct matrices, B and C, and therefore distinct inner products, $\langle \cdot, \cdot \rangle_B$, at each gradient descent iteration.

Given equation 2.20, we can prove equation 2.21 by showing that

$$\nabla^B_W L = B \left[\nabla^E_W L \right] C. \tag{2.22}$$

To show equation 2.22, first define the $N \times N$ standard basis matrices $\mathbf{1}^{jk} \in \mathbb{R}^{N \times N}$ entrywise by

$$\mathbf{1}^{jk}_{j'k'} = \begin{cases} 1 & j = j' \text{ and } k = k' \\ 0 & \text{otherwise} \end{cases}$$

for $j, k = 1, \ldots, N$. Now compute the inner product of the gradient with $\mathbf{1}^{jk}$,

$$\begin{aligned} \left\langle \left[\nabla^B L \right], \mathbf{1}^{jk} \right\rangle_B &= \left\langle B^{-1} \left[\nabla^B L \right], \mathbf{1}^{jk} C^{-1} \right\rangle_E \\ &= \operatorname{Tr} \left(B^{-1} \left[\nabla^B L \right] C^{-1} \left[\mathbf{1}^{jk} \right]^T \right) \\ &= \sum_{n=1}^{N} \left[B^{-1} \left[\nabla^B L \right] C^{-1} \mathbf{1}^{kj} \right]_{n,n} \\ &= \sum_{n,m=1}^{N} \left[B^{-1} \left[\nabla^B L \right] C^{-1} \right]_{n,m} \left[\mathbf{1}^{kj} \right]_{m,n} \\ &= \left[B^{-1} \left[\nabla^B L \right] C^{-1} \right]_{jk}, \end{aligned} \tag{2.23}$$

where the last line follows from the fact that $\mathbf{1}^{kj}_{n,m} = 1$ when $n = k$ and $m = j$, and it is equal to zero for all other j, k. But we also have, from the definition of a gradient,

$$\left\langle \left[\nabla^B L \right], \mathbf{1}^{jk} \right\rangle_B = \lim_{\epsilon \to 0} \frac{J(W + \epsilon \mathbf{1}^{jk}) - J(W)}{\epsilon} = \frac{\partial J}{\partial W_{jk}} = \left[\nabla^E L \right]_{jk}. \tag{2.24}$$

Since equations 2.23 and 2.24 apply for all indices j, k, we may conclude that

$$B^{-1} \left[\nabla^B L \right] C^{-1} = \left[\nabla^E L \right]$$

and therefore that

$$\left[\nabla^B L \right] = B \left[\nabla^E L \right] C,$$

which concludes our proof.

In summary, if W is updated according to ΔW_3^i from equation 2.19, then this is equivalent to performing gradient descent on the loss with respect to the weight matrix under the geometry defined by the new inner product, $\langle U, V \rangle_B$. We next present examples showing that this geometry is better suited to learning W than gradient descent with respect to the standard Euclidean geometry. Specifically, ΔW_3^i learns more robustly than ΔW_1^i.

3 Experiments and Results

We next evaluate and interpret each of the learning rules already derived on two different learning tasks.

3.1 Learning Fixed Points in a Linear Model. For demonstrative purposes, we first consider an example of linear regression with mean-squared error loss. Specifically, we consider $f(z) = z$ with

$$L(\boldsymbol{r}, \boldsymbol{y}) = \|\boldsymbol{r} - \boldsymbol{y}\|^2,$$

where $\|\cdot\|$ is the Euclidean norm on $\mathbb{R}^N$. Note that $G = I$ is the identity in this case. We define the $N \times m$ matrices, $X = \begin{bmatrix}\mathbf{x}^1 & \mathbf{x}^2 & \dots \mathbf{x}^m\end{bmatrix}$, $Y = \begin{bmatrix}\mathbf{y}^1 & \mathbf{y}^2 & \dots \mathbf{y}^m\end{bmatrix}$, and $R = \begin{bmatrix}\mathbf{r}^1 & \mathbf{r}^2 & \dots \mathbf{r}^m\end{bmatrix} = [I - W]^{-1}X$. The cost function can be written as

$$J(W) = \frac{1}{m} \left\| [I - W]^{-1}X - Y \right\|^2. \tag{3.1}$$

It is useful to also write the cost in terms of the parameters $A = [I - W]^{-1}$ to get a standard quadratic cost function,

$$J_A(A) = \frac{1}{m} \|AX - Y\|^2. \tag{3.2}$$

For this problem, minimizers of $J(W)$ and $J_A(A)$ can be found explicitly. Before continuing to empirical examples, we derive and discuss these explicit minimizers.

3.1.1 Computing Explicit Minimizers in a Linear Model. In the underparameterized case ($N \leq m$ when all matrices full rank), $J_A(A)$ in 3.2 has a unique minimizer defined by

$$A^* = YX^+,$$

where $X^+ = X^T(XX^T)^{-1}$ is the Moore-Penrose pseudo-inverse of X when $N \leq m$. Therefore, $J(W)$ has a unique minimizer at

$$W^* = I - [A^*]^{-1} = XX^T(YX^T)^{-1}$$

under the assumption that A^* is invertible.

The overparameterized case ($N > m$ when matrices are full rank) is more relevant and interesting. In this case, there are infinitely many choices of W and A for which $J(W) = 0$ and $J_A(A) = 0$. The problem of choosing a solution to $J_A(A) = 0$ is a standard least squares problem, and a common approach is to take

$$A^* = YX^+,$$

where $X^+ = (X^T X)^{-1} X^T$ is the Moore-Penrose pseudo-inverse of X when $N > m$. It is tempting to use this value of A^* and then take $W^* = I - [A^*]^{-1}$. However, note that A^* is the solution to $AX = Y$ that minimizes the Frobenius norm of A, that is,

$$A^* = \underset{A}{\operatorname{argmin}} \|A\| \text{ s.t. } AX = Y.$$

Therefore, $W^* = I - [A^*]^{-1}$ represents a solution, W, that minimizes the norm of $A = [I - W]^{-1}$. Since the Jacobian matrix is given by $\mathcal{J} = (-I + W)/\tau = -A^{-1}/\tau$, stability is promoted by W having a small spectral radius (all eigenvalues of W must have a real part less than 1 for stability). Hence, $W^* = I - [A^*]^{-1}$ is a poor choice for W^*. Minimizing the Frobenius norm of A will tend to push the eigenvalues of A toward zero, which can lead to large eigenvalues of $W = I - A^{-1}$ and $\mathcal{J} = -A^{-1}/\tau$, promoting unstable fixed points. Instead, to find a good optimizer, W^*, we can find solutions that minimize the norm of W instead of A. To this end, we can solve

$$W^* = \underset{W}{\operatorname{argmin}} \|W\| \text{ s.t. } [I - W]^{-1} X = Y.$$

To solve this problem, we rewrite it in a more standard form:

$$W^* = \underset{W}{\operatorname{argmin}} \|W\| \text{ s.t. } WY = Y - X.$$

This problem has the solution

$$W^* = [Y - X]Y^+, \tag{3.3}$$

where $Y^+ = (Y^T Y)^{-1} Y^T$ is the Moore-Penrose pseudo-inverse of Y when $N > m$. This is the solution with minimal Frobenius norm on W and is therefore more likely than $I - [A^*]^{-1}$ to have a small spectral radius and therefore more likely to give stable fixed points. Hence, equation 3.3 provides a good optimizer in the overparameterized case ($N > m$).

3.1.2 Visualizing the Loss Landscape of a Linear Model. For empirical examples, we first generated inputs, $\mathbf{x}^i$, independently from a gaussian distribution and generated targets $\mathbf{y}^i$ using a ground-truth weight matrix, $\hat{W}$, and adding noise. Specifically, we define

$$\begin{aligned} X &\sim \sigma_x Z_{N\times m}, \\ Y &\sim \left[I - \hat{W}\right]^{-1} X + \sigma_y Z_{N\times m}, \end{aligned} \tag{3.4}$$

where $\sigma_x = 0.1$ controls the magnitude of the inputs, $\sigma_y = 0.01$, and each $Z_{N\times m}$ represents an $N \times m$ matrix of independent, standard, gaussian random variables. The ground-truth weight matrix is generated by

$$\hat{W} \sim \frac{\sigma_w}{\sqrt{N}} Z_{N\times N}.$$

Following Girko's circular law, the eigenvalues of $\hat{W}$ lie approximately within a circle of radius σ_w with high probability (Girko, 1985). Hence, we take $\sigma_w = 0.5 < 1$ to control the spectral radius of the circle to be less than 1, so that all eigenvalues of the Jacobian matrix, $\mathcal{J} = (-I + \hat{W})/\tau$, are negative and fixed-point firing rates are stable under the ground-truth parameters, $\hat{W}$.

The cost landscape, $J(W)$, cannot easily be visualized as a function of W for $N > 1$ because W has N^2 dimensions, so even $N = 2$ would be difficult to visualize. To help visualize $J(W)$, we first plotted it on a random line segment passing through W^* in $\mathbb{R}^{N\times N}$. Specifically, we defined the parameterized function

$$W(t) = W^* + \frac{ct}{\sqrt{N}} Z_{N\times N}, \tag{3.5}$$

where $c = 2.5$ scales the maximum magnitude of the perturbation and t was varied from -1 to 1 to create the visualization of $J(W(t))$ (see Figure 1A). This corresponds to plotting $J(W)$ along a one-dimensional slice of the space $\mathbb{R}^{N\times N}$ on which W lives. Note that the true minimizer, $W = W^*$, is sampled when $t = 0$. The values of W sampled by the process can produce stable or unstable fixed points. Making the approximation $W^* \approx \hat{W}$, we have that $\rho(W(t)) \approx \sqrt{\sigma_w^2 + c^2 t^2}$ and therefore an approximate stability condition is given by $|t| < \sqrt{1 - \sigma_w^2}/c \approx 0.346$.

Figure 1A shows the resulting cost curve for five random values of Z, with the blue dashed lines marking the approximate stability boundary. The cost is relatively well behaved within the boundary but poorly conditioned outside the boundary because of the singularities produced by the matrix inverses in equation 3.1. Specifically, outside of the stability region, the spectral radius of W is larger than 1, so some eigenvalues are near 1 in

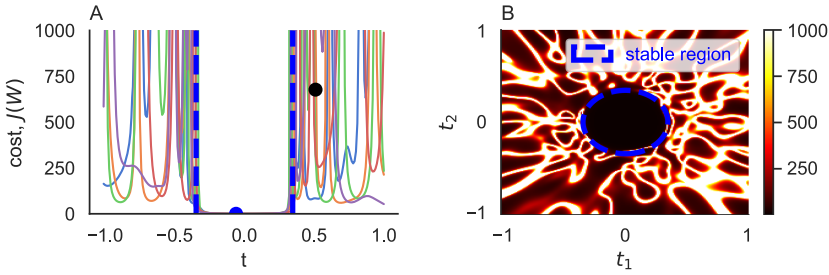

Figure 1: Visualizing the cost landscape for a linear model. (A) The cost function $J(W(t))$ as a function of t from equation 3.5. This represents the cost evaluated along five random line segments in $\mathbb{R}^{N\times N}$, each passing through W^* at $t = 0$. Two blue dashed lines show the stability boundary, $|t| = 0.346$. The vertical axis is cut off at $J = 1000$ to better visualize the curves. Blue and black circles show stable and unstable initial conditions used for learning. (B) The cost function $J(W(t_1, t_2))$ from equation 3.6. This represents the cost evaluated on a randomly oriented square with center at W^*. The color axis is cut off at $J = 1000$.

magnitude. As a result, the $[I - W]^{-1}$ in equation 3.1 can lead to very large values of $J(W)$.

To further visualize the loss landscape, we repeated the procedure above in two dimensions by sampling values of W from a random plane passing through W^*. Specifically, we defined the parameterized function,

$$W(t_1, t_2) = W^* + \frac{c}{\sqrt{N}}(t_1 Z_1 + t_2 Z_2), \tag{3.6}$$

where $Z_1, Z_2 \sim N_{N\times N}(0, 1)$, t_1 and t_2 were each varied from -1 to 1 to create the visualization of $J(W(t))$, and $c = 2.5$ scales the perturbation (see Figure 1B). Note that $W(t_1, t_2) = W^*$ when $t_1 = t_2 = 0$, so the center of the square corresponds to the minimum cost, $J = 0$. The approximate stability condition becomes $\sqrt{t_1^2 + t_2^2} < \sqrt{1 - \sigma_w^2}/c = 0.346$, so the approximate stability boundary is a circle (see Figure 1B, dashed blue curve). Singularities create intricate ridges of large cost outside of the stability boundary (see Figure 1B).

In summary, Figure 1 shows that the cost landscape, $J(W)$, is extremely poorly conditioned outside of the stability region, that is, when W has a spectral radius larger than 1. Note, however, that the effective cost landscape, $J_A(A)$, of the reparameterized model is a simple quadratic landscape, given by equation 3.2. Gradient-based learning according to ΔW_1 must traverse the poorly conditioned landscape from Figure 1. But the learning dynamics of the reparameterized rule, ΔW_2, are equivalent to those produced by A traversing a comparatively well-behaved quadratic landscape.

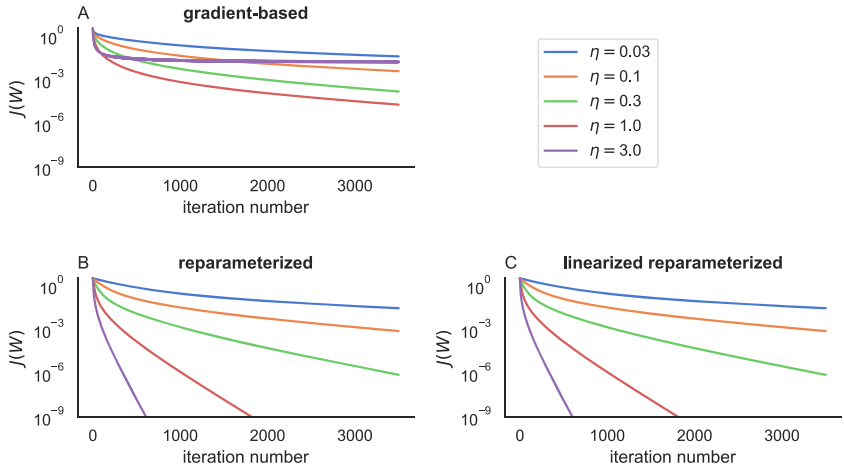

Figure 2: Performance of three different learning rules for a linear regression problem. (A) The cost function, $J(W)$, from equation 3.1 for five different learning rates, $\eta = \eta_W$, using direct gradient descent on the weight matrix, ΔW_1^i from equation 3.7. (B) Same as panel A, but for the reparameterized learning rule, ΔW_2^i, from equation 3.8 with $\eta = \eta_A$. (C) Same as panel B, but for the linearized rule, ΔW_3^i from equation 3.10.

We show in empirical examples below that this difference helps ΔW_2 and its linear approximation, ΔW_3, perform more robustly than ΔW_1.

3.1.3 Gradient Descent on the Recurrent Weight Matrix in a Linear Model. We first perform direct gradient descent on J with respect to W using ΔW_1. The gradient-based update rule from equation 2.7 can be written as

$$\begin{aligned}\Delta W_1 &= \frac{1}{m}\sum_{i=1}^{m}\Delta W_1^i \\ &= \frac{-2\eta_W}{m}\left[I - W^T\right]^{-1}[R - Y]\,R^T. \end{aligned} \tag{3.7}$$

Empirical simulations show relatively poor learning performance (see Figure 2A). Learning is slow for small learning rates, but larger learning rates fail to converge to good minima. Recall that the true minimum is zero because the model is overparameterized. We next show that the linearized approximation to ΔW_2 performs similarly well.

3.1.4 Learning Using the Reparameterized Rule in a Linear Model. For this linear example, the reparameterized learning rule from equations 2.11 and

2.17 can be written as

$$\Delta W_2 = [I - W] - \left([I - W]^{-1} - \frac{2\eta_A}{m}(R - Y)X^T \right)^{-1}. \tag{3.8}$$

Recall that the learning dynamics produced by equation 3.8 are equivalent to those produced by learning the standard quadratic cost function, $J_A(A)$, along with the standard gradient-based update rule,

$$\Delta A = -\frac{2\eta_A}{m}(AX - Y)X^T, \tag{3.9}$$

which is often called the "delta rule."

The behavior of the learning dynamics under equation 3.9 in the overparameterized case is well understood (Gunasekar et al., 2017, 2018; Soudry et al., 2018; Zhang et al., 2021). Specifically, A tends toward solutions to $AX = Y$ that minimize the distance, $\|A - A_0\|$, of A from its initial condition under the Frobenius norm. As a result, ΔW_2 finds solutions, W, to $[I - W]^{-1}X = Y$ that minimize the distance, $d(W, W_0)$, of W from its initial condition under the metric, d, defined in equation 2.12. In addition, since the Jacobian matrix is given by $\mathcal{J} = -A^{-1}/\tau$, we may conclude that ΔW_2 finds solutions that minimize the distance, $\left\| \mathcal{J}_0^{-1} - \mathcal{J}^{-1} \right\|$, between the inverse Jacobian and its initial condition under the Frobenius norm.

In comparison to the gradient-based update, ΔW_1, from see equation 3.7; see Figure 2A, we see that ΔW_2 from equation 3.8 performs much more robustly (see Figure 2B). The cost reliably converges toward zero with increasing rates of convergence at larger learning rates.

3.1.5 Learning Using the Linearized Reparameterized Rule in a Linear Model. The linearized, reparameterized update rule from equation 2.19 for this linear model can be written as

$$\Delta W_3 = -\frac{2\eta_A}{m}[I - W](R - Y)X^T[I - W]. \tag{3.10}$$

This learning rule gives a simpler equation that is more efficient to compute, but still shows excellent agreement with the reparameterized rule, ΔW_2 (see Figure 2C; compare to panel B).

Note that equation 3.10 does not require any explicit computation of matrix inverses with the exception of the computation of firing rates $R = [I - W]^{-1}X$. However, note that expanding the $(R - Y)$ term in equation 3.10 and then substituting $R = [I - W]^{-1}X$ allows this inverse to cancel, and we obtain an expression without inverses:

$$\Delta W_3 = -\frac{2\eta_A}{m}\left(XX^T[I - W] - [I - W]YX^T[I - W] \right). \tag{3.11}$$

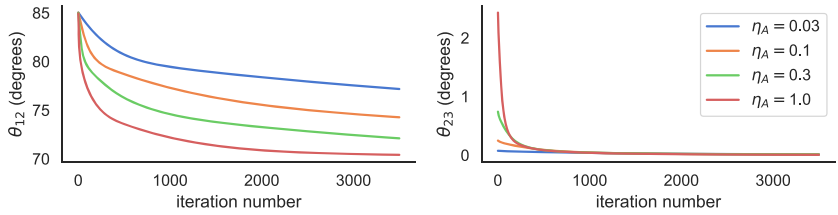

Figure 3: Angles and correlations between weight updates. (A) Angle (θ_{12}) between the weight updates for the gradient-based and reparameterized learning rules. (B) Same as panel A, but comparing the parameterized rule with its linearization.

Note that the equation $R = [I - W]^{-1}X$, and therefore equation 3.11, are specific to the linear case, $f(z) = z$. When using a nonlinear $f(z)$, fixed-point firing rates, R, cannot generally be computed in closed form, but must be approximated by directly simulating equation 2.1 until convergence.

3.1.6 Comparing the Direction of Updates. To check the similarity between the updates from each learning rule, we calculated the angle between the updates at each iteration, defined by

$$\theta_{\alpha\beta} = \cos^{-1}\left(\frac{\Delta W_\alpha \cdot \Delta W_\beta}{\sqrt{(\Delta W_\alpha \cdot \Delta W_\alpha)(\Delta W_\beta \cdot \Delta W_\beta)}}\right)$$

for $\alpha, \beta \in \{1, 2, 3\}$, where $A \cdot B = \mathrm{Tr}(A^T B)$ is the Frobenius inner product. For sufficiently small learning rates, any update, ΔW, that decreases the cost must satisfy $\Delta W \cdot \Delta W_1 > 0$ where ΔW_1 is the gradient-based update (Richards & Kording, 2023) since the change in cost can be written as

$$\begin{aligned}\Delta J &= \Delta W \cdot \nabla_W J + \mathcal{O}(\eta^2) \\ &= -\frac{\eta_W}{m}\Delta W \cdot \Delta W_1 + \mathcal{O}(\eta^2).\end{aligned} \tag{3.12}$$

Additionally, $\Delta W_2 \to \Delta W_3$ as $\eta_A \to 0$. Hence, we should expect that $\theta_{\alpha\beta} < 90^\circ$ for all pairs, α and β.

Figure 3 shows the angles θ_{12} and θ_{23} during learning. The angles θ_{13} were virtually identical to θ_{23}, so they are not shown. In each example, we used the same update, ΔW_β, to update W throughout learning. Hence, the two updates, dW_α and dW_β, were compared starting at the same initial W at each learning step.

The angle θ_{12} between the gradient-based updates and the reparameterized updates is relatively close to 90° (see Figure 3A), indicating that they point in different directions, nearly as different as possible under the

condition that they both decrease the cost. Unsurprisingly, θ_{23} is near zero (see Figure 3B), indicating that the reparameterized rule is similar to its linearization.

3.2 Training Fixed Points on a Nonlinear Categorization Task. So far, for demonstrative purposes, we considered only simple examples of linear regression in which closed equations for optima are known. We next consider an example of image categorization using the MNIST hand-written digit benchmark.

The learning goal is to minimize a categorical cross-entropy loss on $C = 10$ classes using one-hot encoded labels. Specifically,

$$L(\boldsymbol{y}^i, \boldsymbol{s}^i) = -\mathbf{y}^i \cdot \log(\boldsymbol{s}^i)$$

where $\boldsymbol{y}^i$ is a "one-hot" encoded label for digit i,

$$\boldsymbol{s}_l^i = \frac{e^{z_l^i}}{\sum_{k=1}^{C} e^{z_k^i}},$$

is the softmax output, and $\boldsymbol{z}^i \in \mathbb{R}^C$ is a logit computed from a random projection of fixed-point rates of a recurrent network. Specifically,

$$\mathbf{z}^i = W_{\text{out}}\mathbf{r}^i,$$

where $W_{\text{out}} \in \mathbb{R}^{C\times N}$ is a fixed, random readout matrix and $\mathbf{r}^i = f(W\mathbf{r}^i + \mathbf{x}^i)$ is the fixed point from an $N \times N$ recurrent network with input i. Inputs are flattened 28×28 MNIST images, $\mathbf{p}^i \in \mathbb{R}^M$, where $M = 28 * 28 = 784$ and we multiply them by a fixed, random read-in matrix to form the input to the network,

$$\mathbf{x}^i = W_{\text{in}}\mathbf{p}^i,$$

where $W_{\text{in}} \in \mathbb{R}^{N\times M}$ and $N = 300$ is the number of neurons in the network. We did not train W_{out} or W_{in} because we wanted to focus on the effectiveness of learning the recurrent weight matrix, W. We used a hyperbolic target activation function, $f(z) = \tanh(z)$. To compute $\mathbf{r}$, we simulated equation 2.1 using a forward Euler scheme for 500 time steps with $\tau = 100dt$ where dt is the step size used in the Euler method. The model was trained on three epochs of the MNIST data set using a batch size of 512.

For this learning task, the gradient-based update rule from equation 2.7 can be written as

$$\Delta W_1 = -\frac{\eta_W}{m}\sum_{i=1}^{m}\left[\left[G^i\right]^{-1} - W^T\right]^{-1} W_{\text{out}}^T\left[\boldsymbol{s}^i - \boldsymbol{y}^i\right]\left[\boldsymbol{r}^i\right]^T.$$

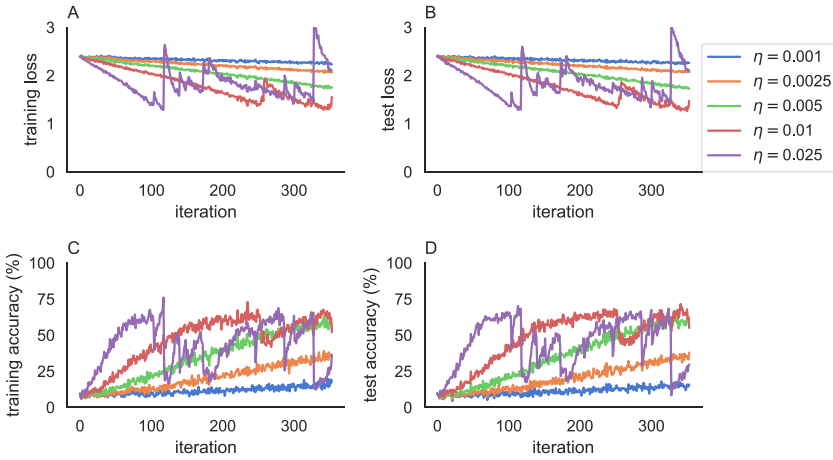

Figure 4: Gradient-based learning on a nonlinear classification task. Results from training the fixed points of a recurrent network to categorize MNIST digits using the gradient-based update rule, ΔW_1. (A–D) Training and testing losses and accuracies evaluated at each step over the course of three epochs.

We found that this gradient-based learning rule performed poorly (see Figure 4). Small learning rates learned slowly, as expected, while larger learning rates produced instabilities that caused the loss and accuracy to jump erratically during learning. Indeed, analysis of the Jacobian matrices showed that fixed points became unstable for the two largest learning rates considered in Figure 4.

We next tested the linearized, reparameterized update rule, ΔW_3. We did not include results for ΔW^2 because, as in the linear examples already considered, they are very similar to ΔW_3, and they are computationally more expensive to calculate. For this learning task, ΔW_3 can be written as

$$\Delta W_3 = -\frac{\eta_A}{m} \sum_{i=1}^{m} \left[I - WG^i\right] G^i W_{\text{out}}^T \left[\boldsymbol{s}^i - \boldsymbol{y}^i\right] \left[\boldsymbol{r}^i\right]^T \left[I - G^i W^T\right] \left[I - G^i W\right].$$

Using this linearized, reparameterized update rule significantly improved learning performance (see Figure 5). Learning performance improved consistently with increasing learning rates, and higher accuracy was achieved without instabilities. Analysis of the Jacobian matrices showed that fixed points were stable for all of the learning rates considered in Figure 5. We conclude that the linearized, reparameterized learning rule can improve the learning of fixed points in nonlinear recurrent neural network models.

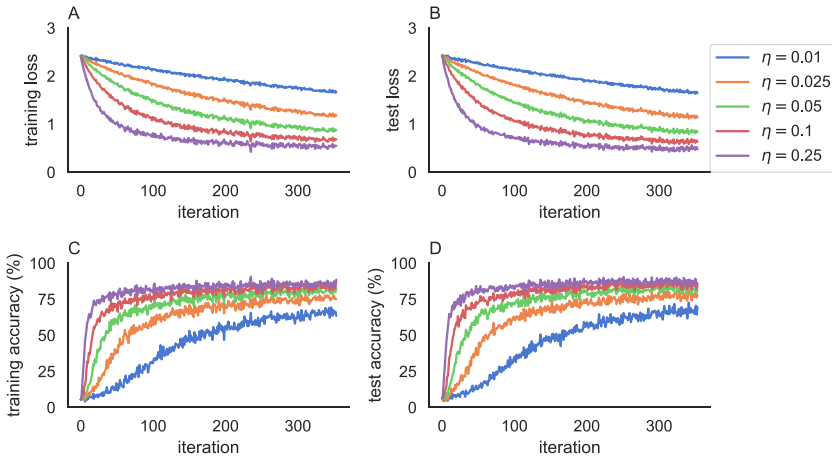

Figure 5: A reparameterized learning rule on a nonlinear classification task. Results from training the fixed points of a recurrent network to categorize MNIST digits using the gradient-based update rule, ΔW_3. (A–D) Training and testing losses and accuracies evaluated at each step over the course of three epochs. Compare to Figure 4.

4 Discussion

In summary, we have shown that when learning fixed points of recurrent neural network models, the direct application of gradient descent with respect to the recurrent weight matrix under the Euclidean geometry is computationally expensive and not robust. Badly conditioned loss surfaces can cause ineffective learning. Moreover, matrix inverses in the equations for the gradients are expensive to evaluate or approximate.

We derived two alternative learning rules derived from a reparameterization of the recurrent network model. These learning rules perform more robustly than the standard gradient descent approach. Moreover, one of the two learning rules is simpler and more computationally efficient. The learning rules can be interpreted as steepest descent and gradient descent on the recurrent weight matrix under a non-Euclidean metric. Our results support recent calls to reconsider the default use of Euclidean gradients on parameters in machine learning (Amari, 1998; Amari & Douglas, 1998; Martens, 2020) and computational neuroscience (Surace et al., 2020; Pogodin et al., 2023).

We showed that the learning rules we derived prevent weights from passing through a boundary beyond which singularities arise in the loss surface. For randomly initialized weight matrices, we showed that this boundary approximately corresponds to the boundary of criticality, beyond

which fixed points lose stability, typically to chaotic dynamics. A large body of prior work has studied networks near criticality and demonstrated that they can exhibit a variety of beneficial properties and can often provide more accurate models of biological data (Levina et al., 2009; Cramer et al., 2020; Morales & Muñoz, 2021; Zeraati et al., 2021; Velikanov & Yarotsky, 2021; Bahri et al., 2021; Liang & Zhou, 2022; Xie & Marsili, 2022; Safavi et al., 2023). It might seem paradoxical, then. that our learning rules improve learning performance by pushing the network away from criticality. This apparent paradox is potentially explained by the fact that criticality in recurrent networks is perhaps most beneficial for performing dynamical tasks in which information about stimuli and task context must be remembered and processed across a range of timescales. On the other hand, we focused on learning fixed points with time-constant inputs, a task in which dynamics and rich timescales are not needed, but stability is more important. Consistent with this distinction, sensory cortical areas exhibit faster timescales, and prefrontal areas exhibit slower timescales (Murray et al., 2014). Slower timescales (mixed with faster ones) are expected in networks near criticality, whereas faster timescales are expected in networks farther from criticality since they converge quickly to their steady states or quasi-steady states. Prefrontal areas participate in dynamical tasks such as the generation of movement, whereas sensory areas participate in tasks more similar to those considered in our study.

Recently, authors have argued that the use of Euclidean gradients for modeling learning in the brain is justified because any learning rule that takes small steps and reduces the loss must be positively correlated with the negative Euclidean gradient (Richards & Kording, 2023). Put another way, the angle between the parameter updates and the negative Euclidean gradient must be less than 90° (see equation 3.12 and surrounding discussion). While this is true of the learning rules that we studied, the angle is very close to 90° in practice, indicating only a weak correlation. Hence, our work shows that the Euclidean gradient is not always strongly correlated with effective learning rules.

We focused on a single, fully connected recurrent layer, which limits the ease with which our model can be applied to larger data sets. Partly for this reason, we only considered the relatively simple MNIST data set as a benchmark. Future work could extend our results to multilayer recurrent networks in which read-in and read-out matrices are trained and in which at least some fully connected layers are replaced by convolutional connectivity. These extensions will allow our approach to be applied to larger and more challenging data sets.

Fixed points of recurrent neural networks are widely used in computational neuroscience to model static neural responses to static stimuli (Ferster & Miller, 2000; Ozeki et al., 2009; Rubin et al., 2015; Ebsch & Rosenbaum, 2018; Curto et al., 2019; Baker et al., 2020), and our results could be useful for these modeling approaches. On the other hand, recurrent

neural networks in machine learning are almost exclusively used for time-varying inputs. And models with time-varying input are widely used in computational neuroscience to model tasks with time-varying stimuli (Vogels et al., 2005; Sussillo & Abbott, 2009; Sussillo, 2014; Rajan et al., 2016; Song et al., 2017; DePasquale et al., 2018; Yang et al., 2019; Pyle & Rosenbaum, 2019; Yang & Wang, 2020; Dubreuil et al., 2022; Márton et al., 2022; Valente et al., 2022). Our results rely on the assumption of a time-constant input, $\mathbf{x}(t) = \mathbf{x}$, which limits their direct application to many machine learning problems. Moreover, even in neuroscience, the assumption of a static stimulus is only an approximation. Natural stimuli are dynamical. However, if fixed points are approached faster than the stimulus changes (i.e., τ is faster than $\mathbf{x}(t)$) then the response, $\mathbf{r}(t)$, is approximated by the fixed point in equation 2.2 and our results provide an approximation. Moreover, a combination of our fixed-point learning rules with dynamical learning rules, such as backpropagation through time, could improve learning in situations where some components of the input are static and others are dynamical. Future work should test whether our learning rules can be combined with backpropagation through time to improve performance on tasks with multiple timescales.

Appendix

A.1 Derivation of the Direct Gradient Descent Update, ΔW_1. Here, we derive equation 2.6 for direct gradient descent on W. To derive this equation, it is sufficient to show that

$$\nabla_W L = \left(\boldsymbol{r} \left[\nabla_{\boldsymbol{r}} L \right]^T \left[I - GW \right]^{-1} G \right)^T .$$

Here we are considering a single input, label, and fixed point—$\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{r}$—so we can omit the i superscripts that appear in equation 2.6. Note that $\nabla_W L(\boldsymbol{r}(W))$ is a matrix with elements

$$\frac{\partial L}{\partial W_{jk}} = \left[\nabla_{\boldsymbol{r}} L(\boldsymbol{r}, \boldsymbol{y}) \right] \cdot \frac{\partial \boldsymbol{r}}{\partial W_{jk}}. \tag{A.1}$$

To derive $\frac{\partial \boldsymbol{r}}{\partial W_{jk}}$, we first derive the change of firing rate, $\Delta \boldsymbol{r}$, to linear order in an update ΔW_{jk}. Consider an initial $\boldsymbol{r}_0$ satisfying $\boldsymbol{r}_0 = f(W_0 \boldsymbol{r}_0 + \boldsymbol{x})$ and an update to W defined by $W = W_0 + \Delta W$ for some ΔW. The new fixed point satisfies $\boldsymbol{r} = f(W\boldsymbol{r} + \boldsymbol{x})$, and we wish to compute $\Delta \boldsymbol{r} = \boldsymbol{r} - \boldsymbol{r}_0$ to linear order in ΔW. Define $\boldsymbol{z}_0 = W_0 \boldsymbol{r}_0 + \boldsymbol{x}$ and $\boldsymbol{z} = W\boldsymbol{r} + \boldsymbol{x}$. Then

$$\begin{aligned} \Delta \boldsymbol{r} &= f(\boldsymbol{z}) - f(\boldsymbol{z}_0) \\ &= f(\boldsymbol{z}_0) + f'(\boldsymbol{z}_0)(\boldsymbol{z} - \boldsymbol{z}_0) - f(\boldsymbol{z}_0) + O(\boldsymbol{z} - \boldsymbol{z}_0)^2 \\ &= G(\boldsymbol{z} - \boldsymbol{z}_0) + O(\boldsymbol{z} - \boldsymbol{z}_0)^2. \end{aligned}$$

To linear-order in $\Delta \boldsymbol{r}$, we have

$$\begin{aligned}
\Delta \boldsymbol{r} &= G(\boldsymbol{z} - \boldsymbol{z}_0) \\
&= G\left((W\boldsymbol{r} + \boldsymbol{x}) - (W_0\boldsymbol{r}_0 + \boldsymbol{x}_0)\right) \\
&= G\left((W_0 + \Delta W)\boldsymbol{r} - W_0\boldsymbol{r}_0\right) \\
&= G(W_0\boldsymbol{r} + \Delta W\boldsymbol{r} - W_0\boldsymbol{r}_0) \\
&= G(W_0\Delta\boldsymbol{r} + \Delta W\boldsymbol{r}) \\
\Delta \boldsymbol{r} - GW_0\Delta\boldsymbol{r} &= G\Delta W\boldsymbol{r} \\
[I - GW_0]\Delta\boldsymbol{r} &= G\Delta W\boldsymbol{r}.
\end{aligned}$$

As a result, we have that

$$\frac{\partial \boldsymbol{r}}{\partial W_{jk}} = [I - GW]^{-1} G\mathbf{1}^{jk}\boldsymbol{r},$$

which is interpreted as a column vector. Here, $\mathbf{1}^{jk}$ is the matrix with all entries equal to zero except for element (j, k), which is equal to 1. Equation 2.6 then follows from lemma 1.

Lemma 1.

$$[I - GW]^{-1} G\mathbf{1}^{jk}\boldsymbol{r} = \boldsymbol{r}_k \left[[I - GW]^{-1} G\right]_{(:,\, j)} \tag{A.2}$$

where $\boldsymbol{r}_k$ is the kth element of $\boldsymbol{r}$ and $B_{(:,\, j)}$ denotes the jth column of a matrix, B.

Proof. We first calculate $\mathbf{1}^{11}\boldsymbol{r}$, $\mathbf{1}^{12}\boldsymbol{r}$, and $\mathbf{1}^{21}\boldsymbol{r}$:

$$\mathbf{1}^{11}\boldsymbol{r} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \cdot \\ \cdot \\ \cdot \\ \boldsymbol{r}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} = \boldsymbol{r}_1 I(:, 1),$$

$$\mathbf{1}^{12}\boldsymbol{r} = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \cdot \\ \cdot \\ \cdot \\ \boldsymbol{r}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_2 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} = \boldsymbol{r}_2 I(:, 1),$$

$$\mathbf{1}^{21}\boldsymbol{r} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \cdot \\ \cdot \\ \cdot \\ \boldsymbol{r}_M \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{r}_1 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} = \boldsymbol{r}_1 I(:,2).$$

Denote $A := [I - GW]^{-1}G$, so $A\mathbf{1}^{11}\boldsymbol{r} = \boldsymbol{r}_1 A_{(:,1)}$, $A\mathbf{1}^{12}\boldsymbol{r} = \boldsymbol{r}_2 A_{(:,1)}$, and $A\mathbf{1}^{21}\boldsymbol{r} = \boldsymbol{r}_1 A_{(:,2)}$. Notice that they are column vectors. WLOG, $A\mathbf{1}^{jk}\boldsymbol{r} = \boldsymbol{r}_k A_{(:,j)}$,

$$LHS = \nabla_w L(\boldsymbol{r}(W)) = \begin{bmatrix} \frac{dL}{dW_{11}} & \frac{dL}{dW_{12}} & \dots & \dots & \frac{dL}{dW_{1M}} \\ \frac{dL}{dW_{21}} & \frac{dL}{dW_{22}} & \dots & \dots & \frac{dL}{dW_{2M}} \\ \frac{dL}{dW_{j1}} & \dots & \frac{dL}{dW_{jk}} & \dots & \frac{dL}{dW_{jM}} \\ \frac{dL}{dW_{M1}} & \frac{dL}{dW_{M2}} & \dots & \dots & \frac{dL}{dW_{MM}} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,1)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,1)} & \dots & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,1)} \\ \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,2)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,2)} & \dots & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,2)} \\ \dots & \dots & \boldsymbol{r}_k[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,j)} & \dots \\ \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,M)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,M)} & \dots & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,M)} \end{bmatrix},$$

$$RHS = \left(\boldsymbol{r}[\nabla_r L(\boldsymbol{r})]^T [I - GW]^{-1} G\right)^T = \left(\boldsymbol{r}[\nabla_r L(\boldsymbol{r})]^T A\right)^T,$$

$$= \left(\begin{bmatrix} \boldsymbol{r}_1 \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_1} & \boldsymbol{r}_1 \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_2} & \dots & \boldsymbol{r}_1 \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_M} \\ \boldsymbol{r}_2 \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_1} & \boldsymbol{r}_2 \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_2} & \dots & \boldsymbol{r}_2 \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_M} \\ \dots & \dots & \dots & \dots \\ \boldsymbol{r}_M \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_1} & \boldsymbol{r}_M \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_2} & \dots & \boldsymbol{r}_M \frac{\partial L(\boldsymbol{r})}{\partial \boldsymbol{r}_M} \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1M} \\ A_{21} & A_{22} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots \\ A_{M1} & A_{M2} & \dots & A_{MM} \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})]^T A_{(:,1)} & \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})]^T A_{(:,2)} & \dots & \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})]^T A_{(:,M)} \\ \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})]^T A_{(:,1)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})]^T A_{(:,2)} & \dots & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})]^T A_{(:,M)} \\ \dots & \dots & \dots & \dots \\ \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})]^T A_{(:,1)} & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})]^T A_{(:,2)} & \dots & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})]^T A_{(:,M)} \end{bmatrix}^T$$

$$= \begin{bmatrix} \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,1)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,1)} & \dots & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,1)} \\ \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,2)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,2)} & \dots & \boldsymbol{r}_M[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,2)} \\ \dots & \dots & \boldsymbol{r}_k[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,j)} & \dots \\ \boldsymbol{r}_1[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,M)} & \boldsymbol{r}_2[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,M)} & \dots & r_M[\nabla_r L(\boldsymbol{r})] \cdot A_{(:,M)} \end{bmatrix}$$

$= LHS.$

□

Combining equation A.1 with equation A.2 gives

$$\nabla_W L = \left(\boldsymbol{r} \left[\nabla_{\boldsymbol{r}} L(\boldsymbol{r}) \right]^T \left[I - GW \right]^{-1} G \right)^T$$

which can be simplified to get equation 2.6 for ΔW_1.

A.2 Analysis of a Natural Reparameterization and Its Linear Approximation. We now consider the updates given by the reparameterization $A = \left[G^{-1} - W \right]^{-1}$. The direct reparameterized update, ΔW_2 in this case, is given by

$$\begin{aligned} \Delta W_2 &= -\left[\left(A - \eta_A (\nabla_{\boldsymbol{r}} L)(\boldsymbol{x})^T \right)^{-1} - A^{-1} \right] \\ &= -\left[\left(\left[G^{-1} - W \right]^{-1} - \eta_A (\nabla_{\boldsymbol{r}} L)(\boldsymbol{r})^T \left[G^{-1} - W^T \right] \right)^{-1} - G^{-1} - W \right]. \end{aligned}$$

Proof. Since $A = \left[G^{-1} - W \right]^{-1}$, we have $W = G^{-1} - A^{-1}$. Let W^0 and A^0 represent previous step update before W and A then

$$\begin{aligned} \Delta W &= W - W^0 \\ &= G^{-1} - A^{-1} - \left(\left[G^0 \right]^{-1} - [A^0]^{-1} \right) \\ &= (G^{-1} - \left[G^0 \right]^{-1}) - A^{-1} + [A^0]^{-1} \\ &= -\left(A^0 + \Delta A \right)^{-1} + [A^0]^{-1} \\ &= -\left(A^0 - \eta_A \left(\nabla_{\boldsymbol{r}} L \right) \left(\boldsymbol{x} \right)^T \right)^{-1} + [A^0]^{-1} \\ &= -\left(A^0 - \eta_A \left(\nabla_{\boldsymbol{r}} L \right) \left(\boldsymbol{r} \right)^T [A^0]^{-T} \right)^{-1} + [A^0]^{-1}. \end{aligned}$$

To get the expression that has only G and W, we can substitute $A = [G^{-1} - W]^{-1}$ and $A^{-1} = G^{-1} - W$ and use $G = G^T$ and $G^{-T} = G^{-1}$ since G is a diagonal matrix. This gives

$$\begin{aligned} \Delta W_2 &= -\left(A - \eta_A \left(\nabla_{\boldsymbol{r}} L \right) \left(\boldsymbol{r} \right)^T A^{-T} \right)^{-1} + A^{-1} \\ &= -\left(\left[G^{-1} - W \right]^{-1} - \eta_A \left(\nabla_{\boldsymbol{r}} L \right) \left(\boldsymbol{r} \right)^T \left[G^{-1} - W^T \right] \right)^{-1} + \left[G^{-1} - W \right]. \end{aligned}$$

□

Note that as $G_{jj} \to 0$, $A_{jj}^{-1} = [G_{jj}^i]^{-1} - W_{jj} \to \infty$, so this reparameterization is poorly behaved in situations where $G_{jj} = f'(\mathbf{z}_j)$ becomes small or zero because the second term in the sum diverges while the first term does not.

We also show that linearizing this parameterization around $\eta_A = 0$ still leads to updates that diverge when elements of G become small. Following the linearization from section 2.4, the linearized, reparameterized update is given by

$$\begin{aligned}\Delta W_3 &= -\eta_A A^{-1}(\nabla_{\boldsymbol{r}} L)(\boldsymbol{x})^T A^{-1} \\ &= -\eta_A \left[G^{-1} - W\right](\nabla_{\boldsymbol{r}} L)(\boldsymbol{r})^T \left[G^{-1} - W^T\right]\left[G^{-1} - W\right].\end{aligned}$$

Proof. First note that $\Delta W_2|_{\eta_A=0} = 0$, so we have to linear-order in η_A,

$$\Delta W_2 = \left.\frac{d\Delta W_2}{d\eta_A}\right|_{\eta_A=0} \eta_A + \mathcal{O}(\eta_A^2). \tag{A.3}$$

Now let

$$V = A + \Delta A = A - \eta_A(\nabla_{\boldsymbol{r}} L)\left(A^{-1}\boldsymbol{r}\right)^T;$$

then $\Delta W_2 = A^{-1} - V^{-1}$, so

$$\begin{aligned}\frac{d\Delta W_2}{d\eta_A} &= \frac{dA^{-1}}{d\eta_A} - \frac{dV^{-1}}{d\eta_A} \\ &= V^{-1}\frac{dV}{d\eta_A}V^{-1}\end{aligned}$$

since $dA^{-1}/d\eta_A = 0$. Combining this with equation A.3 and the definition of V gives the linearized update,

$$\begin{aligned}\Delta W_3 &= \left. V^{-1}\frac{dV}{d\eta_A}V^{-1}\right|_{\eta_A=0} \eta_A \\ &= V^{-1}\left(-(\nabla_{\boldsymbol{r}} L)\left(A^{-1}\boldsymbol{r}\right)^T\right)V^{-1}\Big|_{\eta_A=0}\eta_A \\ &= -A^{-1}(\nabla_{\boldsymbol{r}} L)(\boldsymbol{r})^T A^{-T}A^{-1}\eta_A \\ &= -\left[G^{-1} - W\right]^{-1}(\nabla_{\boldsymbol{r}} L)(\boldsymbol{r})^T\left[G^{-1} - W^T\right]\left[G^{-1} - W\right]\eta_A.\end{aligned}$$

Again, substitute $A^{-1} = G^{-1} - W$, to get the final expression. Notice that $\Delta W_3 = A^{-1}A^{-T}\Delta W_1 A^{-T}A^{-1}$, so one can let $B = A^{-1}A^{-T}$ and $C = A^{-T}A^{-1}$, which are symmetric, and $\Delta W_3 = B\Delta W_1 C$. □

Note again that ΔW_3 diverges if elements of G go to zero. Therefore, the natural reparameterization $A = [G^{-1} - W]^{-1}$ is not well suited for learning.

A.3 Linearization of the Corrected Reparameterization. Here, we derive the linearized update, ΔW_3, given in equation 2.19. This update rule

is derived by expanding ΔW_2 from equation 2.17 to linear-order. Recall that ΔW_2 was derived from the reparameterization $A = [G - GWG]^{-1}$. Let $U = [G^{-1} - W]^{-1}$; then we can rewrite equation 2.17 as

$$\Delta W_2 = -\left[U - \eta_A G^2 \left(\nabla_{\boldsymbol{r}} L\right)(\boldsymbol{r})^T [I - GW]^T G\right]^{-1} + U^{-1}.$$

Now, denote everything inside of the inverse as V so

$$V = U - \eta_A G^2 \left(\nabla_{\boldsymbol{r}} L\right)(\boldsymbol{r})^T [I - GW]^T G.$$

Then equation 2.17 can be further rewritten as

$$\Delta W_2 = U^{-1} - V^{-1}.$$

Now, following the same approach as in section A.2, note that $\Delta W_2|_{\eta_A=0} = 0$, so the linearization of ΔW_2 around $\eta_A = 0$ is given by

$$\begin{aligned}
\Delta W_3 &= \left.\frac{d\Delta W_2}{d\eta_A}\right|_{\eta_A=0} \eta_A \\
&= \left[\frac{dU^{-1}}{d\eta_A} - \left(-V^{-1}\frac{dV}{d\eta_A}V^{-1}\right)\right]_{\eta_A=0} \eta_A \\
&= V^{-1}\left(-G^2 \left(\nabla_{\boldsymbol{r}} L\right)(\boldsymbol{r})^T [I - GW]^T G\right) V^{-1}\Big|_{\eta_A=0} \eta_A \\
&= U^{-1}\left(-G^2 \left(\nabla_{\boldsymbol{r}} L\right)(\boldsymbol{r})^T [I - GW]^T G\right) U^{-1}\eta_A \\
&= -\eta_A [G^{-1} - W] G^2 \left(\nabla_{\boldsymbol{r}} L\right)(\boldsymbol{r})^T [I - GW]^T G [G^{-1} - W] \\
&= -\eta_A [I - WG]\, G \left(\nabla_{\boldsymbol{r}} L\right)(\boldsymbol{r})^T [I - GW]^T [I - GW].
\end{aligned}$$

The radius of convergence of the Taylor expansion that we used to derive ΔW_3^i from ΔW_2^i is defined by

$$\begin{aligned}
&\left\|\eta_A \left[\left[A^i\right]^{-1}\right]\left[G^i\right]^2 \left(\Delta A^i\right)\left[G^i\right]^2\right\| \\
&= \|\eta_A \left[G^i - W\right]\left[G^i\right]^2 \left(\nabla_{\boldsymbol{r}^i} L\right)\left(\boldsymbol{r}^i\right)^T \left[G^i - G^i W^T G^i\right]\left[G^i\right]^2\| < 1. \qquad \text{(A.4)}
\end{aligned}$$

Therefore, if

$$\eta_A \ll \frac{1}{\|\left[G^i - G^i W G^i\right]\left[G^i\right]^2 \left(\nabla_{\boldsymbol{r}^i} L\right)\left(\boldsymbol{r}^i\right)^T \left[G^i - G^i W^T G^i\right]\left[G^i\right]^2\|},$$

the ΔW_3^i from equation 2.19 approximates ΔW_2^i from equation 2.17.

Acknowledgments

This material is based on work supported by the Air Force Office of Scientific Research under award FA9550-21-1-0223 and National Science Foundation awards DMS-1654268 and DBI-1707400.

References

Almeida, L. B. (1990). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In J. Diederich (Ed.), *Artificial neural networks: Concept learning* (pp. 102–111).

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*(2), 251–276. 10.1162/089976698300017746

Amari, S.-I., & Douglas, S. C. (1998). Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing* (vol. 2, pp. 1213–1216). 10.1109/ICASSP.1998.675489

Bahri, Y., Dyer, E., Kaplan, J., Lee, J., & Sharma, U. (2021). *Explaining neural scaling laws*. arXiv:2102.06701.

Bai, S., Kolter, J. Z., & Koltun, V. (2019). Deep equilibrium models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*, *32*. Curran.

Bai, S., Koltun, V., & Kolter, J. Z. (2020). Multiscale deep equilibrium models. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 5238–5250). Curran.

Baker, C., Zhu, V., & Rosenbaum, R. (2020). Nonlinear stimulus representations in neural circuits with approximate excitatory-inhibitory balance. *PLOS Computational Biology*, *16*(9), e1008192. 10.1371/journal.pcbi.1008192

Brenner, M., Hess, F., Mikhaeil, J. M., Bereska, L. F., Monfared, Z., Kuo, P.-C., & Durstewitz, D. (2022). Tractable dendritic RNNs for reconstructing nonlinear dynamical systems. In *Proceedings of the International Conference on Machine Learning* (pp. 2292–2320).

Cramer, B., Stöckel, D., Kreft, M., Wibral, M., Schemmel, J., Meier, K., & Priesemann, V. (2020). Control of criticality and computation in spiking neuromorphic networks with plasticity. *Nature Communications*, *11*(1), 2853.

Curto, C., Geneson, J., & Morrison, K. (2019). Fixed points of competitive threshold-linear networks. *Neural Computation*, *31*(1), 94–155. 10.1162/neco_a_01151

Dayan, P., & Abbott, L. F. (2001). *Theoretical neuroscience*. MIT Press.

DePasquale, B., Cueva, C. J., Rajan, K., Escola, G. S., & Abbott, L. (2018). Full-force: A target-based method for training recurrent networks. *PLOS One*, *13*(2), e0191527. 10.1371/journal.pone.0191527

Dubreuil, A., Valente, A., Beiran, M., Mastrogiuseppe, F., & Ostojic, S. (2022). The role of population structure in computations through neural dynamics. *Nature Neuroscience*, *25*(6), 783–794. 10.1038/s41593-022-01088-4

Durstewitz, D., Koppe, G., & Thurm, M. I. (2023). Reconstructing computational system dynamics from neural data with recurrent neural networks. *Nature Reviews Neuroscience*, *24*(11), 693–710. 10.1038/s41583-023-00740-7

Durstewitz, D., Seamans, J. K., & Sejnowski, T. J. (2000). Neurocomputational models of working memory. *Nature Neuroscience*, *3*(11), 1184–1191. 10.1038/81460

Ebsch, C., & Rosenbaum, R. (2018). Imbalanced amplification: A mechanism of amplification and suppression from local imbalance of excitation and inhibition in cortical circuits. *PLOS Computational Biology*, *14*(3), e1006048. 10.1371/journal.pcbi.1006048

Ferster, D., & Miller, K. D. (2000). Neural mechanisms of orientation selectivity in the visual cortex. *Annual Review of Neuroscience*, *23*(1), 441–471. 10.1146/annurev.neuro.23.1.441

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press.

Girko, V. L. (1985). Circular law. *Theory of Probability and Its Applications*, *29*(4), 694–706. 10.1137/1129095

Gunasekar, S., Lee, J., Soudry, D., & Srebro, N. (2018). Characterizing implicit bias in terms of optimization geometry. In *Proceedings of the International Conference on Machine Learning* (pp. 1832–1841).

Gunasekar, S., Woodworth, B. E., Bhojanapalli, S., Neyshabur, B., & Srebro, N. (2017). Implicit regularization in matrix factorization. In I. Guyon, Y. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*, *30*. Curran.

Kreutzer, E., Senn, W., & Petrovici, M. A. (2022). Natural-gradient learning for spiking neurons. *eLife*, *11*. 10.7554/eLife.66526

Levina, A., Herrmann, J. M., & Geisel, T. (2009). Phase transitions towards criticality in a neural system with adaptive interactions. *Physical Review Letters*, *102*(11), 118110. 10.1103/PhysRevLett.102.118110

Liang, J., & Zhou, C. (2022). Criticality enhances the multilevel reliability of stimulus responses in cortical neural networks. *PLOS Computational Biology*, *18*(1), e1009848.

Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., . . . Zemel, R. (2018). Reviving and improving recurrent back-propagation. In *Proceedings of the International Conference on Machine Learning* (pp. 3082–3091).

Lillicrap, T. P., & Santoro, A. (2019). Backpropagation through time and the brain. *Current Opinion in Neurobiology*, *55*, 82–89. 10.1016/j.conb.2019.01.011

Martens, J. (2020). New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, *21*(1), 5776–5851.

Márton, C. D., Zhou, S., & Rajan, K. (2022). Linking task structure and neural network dynamics. *Nature Neuroscience*, *25*(6), 679–681.

Morales, G. B., & Muñoz, M. A. (2021). Optimal input representation in neural systems at the edge of chaos. *Biology*, *10*(8), 702. 10.3390/biology10080702

Murray, J. D., Bernacchia, A., Freedman, D. J., Romo, R., Wallis, J. D., Cai, X., . . . Wang, X. J. (2014). A hierarchy of intrinsic timescales across primate cortex. *Nature Neuroscience*, *17*(12), 1661–1663. 10.1038/nn.3862

Ollivier, Y., Tallec, C., & Charpiat, G. (2015). *Training recurrent networks online without backtracking*. arXiv:1507.07680.

Ozeki, H., Finn, I. M., Schaffer, E. S., Miller, K. D., & Ferster, D. (2009). Inhibitory stabilization of the cortical network underlies visual surround suppression. *Neuron*, *62*(4), 578–592. 10.1016/j.neuron.2009.03.028

Pineda, F. (1987). Generalization of back propagation to recurrent and higher order neural networks. In D. Anderson (Ed.), *Neural information processing systems.* American Institute of Physics.

Pogodin, R., Cornford, J., Ghosh, A., Gidel, G., Lajoie, G., & Richards, B. (2023). *Synaptic weight distributions depend on the geometry of plasticity*. arXiv:2305.19394.

Pyle, R., & Rosenbaum, R. (2019). A reservoir computing model of reward- modulated motor learning and automaticity. *Neural Computation*, *31*(7), 1430–1461. 10.1162/neco_a_01198

Rajan, K., Harvey, C. D., & Tank, D. W. (2016). Recurrent network models of sequence generation and memory. *Neuron*, *90*(1), 128–142. 10.1016/j.neuron.2016.02.009

Richards, B. A., & Kording, K. P. (2023). The study of plasticity has always been about gradients. *Journal of Physiology*, *601*(15), 3141–3149. 10.1113/JP282747

Rubin, D. B., Van Hooser, S. D., & Miller, K. D. (2015). The stabilized supralinear network: A unifying circuit motif underlying multi-input integration in sensory cortex. *Neuron*, *85*(2), 402–417. 10.1016/j.neuron.2014.12.026

Safavi, S., Chalk, M., Logothetis, N., & Levina, A. (2023). *Signatures of criticality in efficient coding networks.* bioRxiv, 2023–02.

Song, H. F., Yang, G. R., & Wang, X.-J. (2017). Reward-based training of recurrent neural networks for cognitive and value-based tasks. *eLife*, *6*, e21492.

Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S., & Srebro, N. (2018). The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, *19*(1), 2822–2878.

Spivak, M. (2018). *Calculus on manifolds: A modern approach to classical theorems of advanced calculus*. CRC Press.

Surace, S. C., Pfister, J.-P., Gerstner, W., & Brea, J. (2020). On the choice of metric in gradient-based theories of brain function. *PLOS Computational Biology*, *16*(4), e1007640. 10.1371/journal.pcbi.1007640

Sussillo, D. (2014). Neural circuits as computational dynamical systems. *Current Opinion in Neurobiology*, *25*, 156–163. 10.1016/j.conb.2014.01.008

Sussillo, D., & Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, *63*(4), 544–557. 10.1016/j.neuron.2009.07.018

Valente, A., Pillow, J. W., & Ostojic, S. (2022). Extracting computational mechanisms from neural data using low-rank RNNs. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems*, *35* (pp. 24072–24086). Curran.

Velikanov, M., & Yarotsky, D. (2021). *Universal scaling laws in the gradient descent training of neural networks.* arXiv:2105.00507.

Vogels, T. P., Rajan, K., & Abbott, L. F. (2005). Neural network dynamics. *Annual Review of Neuroscience*, *28*, 357–376. 10.1146/annurev.neuro.28.061604.135637

Williams, R. J., & Peng, J. (1990). An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural Computation*, *2*(4), 490–501. 10.1162/neco.1990.2.4.490

Winston, E., & Kolter, J. Z. (2020). Monotone operator equilibrium networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 10718–10728). Curran.

Xie, R., & Marsili, M. (2022). A random energy approach to deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, *2022*(7), 073404.

Yang, G. R., Joglekar, M. R., Song, H. F., Newsome, W. T., & Wang, X.-J. (2019). Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, *22*(2), 297–306. 10.1038/s41593-018-0310-2

Yang, G. R., & Wang, X.-J. (2020). Artificial neural networks for neuroscientists: A primer. *Neuron*, *107*(6), 1048–1070. 10.1016/j.neuron.2020.09.005

Zeraati, R., Priesemann, V., & Levina, A. (2021). Self-organization toward criticality by synaptic plasticity. *Frontiers in Physics*, *9*, 619661. 10.3389/fphy.2021.619661

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, *64*(3), 107–115. 10.1145/3446776

Received August 30, 2023; accepted March 18, 2024.