CORRELATIONS ARE RUINING YOUR GRADIENT DESCENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Natural gradient descent, and the preconditioned optimizers which have followed from it, correct gradient descent by estimating curvature and applying its inverse to the gradient. For any single layer of a deep neural network, this correction can be shown to approximately separate into two factors: one which accounts for correlations in the gradient signal, and one which accounts for correlations in the data arriving at that layer. Herein we show that this second factor can be expressed as a change of basis rather than of curvature and, considered thus, it can be removed from the optimizer entirely and placed within the network itself. Specifically, we show that, once forward weights are corrected, right-sided natural gradient descent can be implemented in a form which has three exactly equivalent forms: a gradient preconditioner, a decorrelating linear transform in the forward pass, or a set of recurrent dynamics in which all updates are Hebbian and rely only upon locally available signals. The (recurrent) decorrelation method we discover aligns closely with proposed methods for inhibitory learning and homeostatic plasticity in the brain. Notably, our transform requires no matrix inverse, no matrix decomposition, and no information beyond that which is available at an individual synapse. It can therefore be applied in distributed and physically constrained systems, including neuromorphic hardware and nervous systems, where existing preconditioners cannot. We demonstrate its effectiveness empirically in convolutional and residual network models while significantly improving a range of approximate biologically plausible methods for credit assignment.

1 INTRODUCTION

The method of gradient descent is as popular as it is intuitive. Nonetheless, a gradient is only 'steepest direction' with respect to some measure of distance in parameter space, and gradient descent makes a particular, and rarely examined, choice of that measure. Much of the recent progress in optimization has come from revisiting this choice, and methods which precondition the gradient (Martens & Grosse, 2015; Gupta et al., 2018; Vyas et al., 2024; Jordan et al., 2024) have thereby moved from the edges of the field to its centre.

Natural gradient descent (Amari, 1998) is the oldest of these perspectives, measuring distance by a Riemannian metric rather than a Euclidean one. Applied to a single layer of a deep neural network, its correction separates, approximately, into two factors: one (left) built from correlations in the gradient signal arriving at that layer, and one (right) built from correlations in the data entering it (Heskes, 2000; Martens & Grosse, 2015). The left factor, which undoes the skew of the loss landscape, has received almost all of the attention. The right factor is the subject of this work, and our starting observation is that it is not really a statement about curvature at all. Correlations in the data arriving at a layer mean only that the layer's parameters live in a non-orthonormal basis, and the right factor is precisely the transformation which returns them to an orthonormal one. This problem of basis, unlike a problem of curvature, doesn't need to be solved by an optimizer but can be tracked and solved within a network outside of training.

1.1 GRADIENT DESCENT, NATURAL GRADIENTS, AND THE CHOICE OF METRIC

For a dataset $\mathcal{D}$ of input and output pairs $(\boldsymbol{x}, \boldsymbol{y})$, we wish to find parameters $\boldsymbol{\theta}$ of a function $\boldsymbol{f_\theta}$ which minimise a loss $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \ell(\boldsymbol{f_\theta}(\boldsymbol{x}), \boldsymbol{y})$, with a sample-wise loss ℓ. Gradient descent

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

proposes the update $\delta\boldsymbol{\theta}^{\text{GD}} = -\eta\nabla\mathcal{L}(\boldsymbol{\theta})$, and though this is usually described as a 'small step' in the direction of steepest descent, it is equivalently the minimum of a linearised loss which is penalised by the size of the parameter change,

$$\delta\boldsymbol{\theta}^{\text{GD}} = \underset{\delta\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^\top\delta\boldsymbol{\theta} + \frac{1}{2\eta}\delta\boldsymbol{\theta}^\top\delta\boldsymbol{\theta}.$$

The final term is the Euclidean norm of the parameter change, and it is here that gradient descent makes its choice of metric.

Amari (1998) proposed that this choice be replaced by a general Riemannian metric, $\delta\boldsymbol{\theta}^\top\boldsymbol{G}(\boldsymbol{\theta})\delta\boldsymbol{\theta}$, whereupon minimisation yields the natural gradient descent update

$$\delta\boldsymbol{\theta}^{\text{NGD}} = -\eta\boldsymbol{G}(\boldsymbol{\theta})^{-1}\nabla\mathcal{L}(\boldsymbol{\theta}). \tag{1}$$

Standard natural gradient descent takes $\boldsymbol{G}$ to be the Fisher information matrix, which assumes the problem to be probabilistic. We instead follow Heskes (2000), who use a deterministic perspective to derive the Fisher, though we instead consider the change induced in the loss of network, and thereby obtains, $\boldsymbol{G} = \left\langle \nabla_{\boldsymbol{\theta}}\ell\, \nabla_{\boldsymbol{\theta}}\ell^\top \right\rangle_{\boldsymbol{x},\boldsymbol{y}}$, where $\langle\cdot\rangle_a$ denotes the expectation over the variable a and the derivation is provided in Appendix B.

Note that this metric, being an outer product of gradients, is a well established quantity commonly referred to as the *empirical* Fisher information matrix. As will be seen below, under a Kronecker factorization the factor built from a layer's input data is identical whether one begins from the empirical Fisher, the true Fisher, or the generalized Gauss-Newton matrix; these differ only in the factor built from the gradient signal. Since the method we develop acts upon the input factor alone, and discards the other entirely, the known deficiencies of the empirical Fisher can be ignored.

For a feed-forward network $\boldsymbol{x}_i = \phi(\boldsymbol{h}_i) = \phi(\boldsymbol{W}_i\boldsymbol{x}_{i-1})$ over layers $i \in [1 \dots L]$, with ϕ a fixed continuous non-linearity, $\boldsymbol{x}_0$ the input data, $\boldsymbol{x}_L$ the model output and biases omitted throughout, it is standard to assume that the natural gradient update may be computed for each layer independently, an approximation which works well in practice (Desjardins et al., 2015) and is fully justified for linear networks (Bernacchia et al., 2018). Under this assumption, and a further assumption that the gradient signal and the input data may be separately averaged, the metric for a single layer factorises as a Kronecker product of two terms and the update becomes

$$\delta\boldsymbol{\theta}^{\text{NG}}_{\boldsymbol{W}_i} = -\eta\text{Vec}\left(\left\langle \frac{\partial\ell}{\partial\boldsymbol{h}_i}\frac{\partial\ell}{\partial\boldsymbol{h}_i}^\top \right\rangle_{\boldsymbol{y}}^{-1} \left\langle \frac{\partial\ell}{\partial\boldsymbol{h}_i}\boldsymbol{x}_{i-1}^\top \right\rangle_{\boldsymbol{y},\boldsymbol{x}_{i-1}} \left\langle \boldsymbol{x}_{i-1}\boldsymbol{x}_{i-1}^\top \right\rangle_{\boldsymbol{x}_{i-1}}^{-1}\right), \tag{2}$$

where $\text{Vec}(\cdot)$ flattens a matrix into a vector. This result has been arrived at independently by Heskes (2000), Martens & Grosse (2015), Desjardins et al. (2015) and Bernacchia et al. (2018); the full derivation is reproduced in Appendix C.

The structure of Equation 2 is the starting point for everything which follows. In the middle sits the ordinary gradient, as computed by backpropagation. From the left it is multiplied by the inverse correlation of the gradient vectors, which undoes skew and scale in the loss landscape and is the term with which natural gradients are usually associated. From the right it is multiplied by the inverse correlation of the data which acts as input to that layer, and it is this right factor alone which this work targets. Note that we refer to these outer product terms as correlations throughout, despite the fact that a true correlation would require a centering and a normalization of the data, and we do so purely for ease of discussion.

1.2 Preconditioned optimizers and what they require

Equation 2 is rarely applied directly, as both factors must be estimated and inverted. K-FAC (Martens & Grosse, 2015) estimates the two Kronecker factors and inverts them, refreshing these inverses only every several hundred steps in order that the cost be amortised. Shampoo (Gupta et al., 2018) instead maintains matrix roots of preconditioners built from the gradients themselves, and SOAP (Vyas et al., 2024) runs an adaptive method within the eigenbasis which such a preconditioner provides. Muon (Jordan et al., 2024) dispenses with curvature estimation altogether and orthogonalizes the gradient matrix directly, which amounts to steepest descent under a spectral rather than a Euclidean norm.

Recently proposed one-sided variants of these methods (An et al., 2026) precondition on a single side only, and thereby approach the term which we study here from the direction of the optimizer.

Two properties are common to all of these methods. First, each depends upon statistics of the gradient, which are available only once a backward pass has been computed. Second, each requires a matrix inverse, decomposition, or iterative orthogonalization, which gathers the statistics of an entire layer into one place and processes them by expensive machinery which sits outside of the network. The right factor of Equation 2 is unusual in requiring neither: it is built from a layer's inputs, and so is available during the forward pass alone.

1.3 Decorrelation and whitening of layer inputs

That the right factor may be addressed by acting upon a network's activations rather than upon its gradients is not a new idea, and two families of method exist for doing so. The first measures the correlation matrix of the data at a layer and inverts it by matrix decomposition, whether at pre-defined checkpoints (Desjardins et al., 2015; Luo, 2017), at the level of individual mini-batches (Bernacchia et al., 2018), or as an extension of batch normalization applied to networks of considerable depth (Huang et al., 2018; 2019). Of these, Desjardins et al. (2015) alone recognised that re-estimating such a transform alters the function which the network computes, and re-parameterised the downstream weights in order to preserve it. The second family estimates a decorrelating matrix iteratively, with no inversion or decomposition at any point (Silva, 1991; Silva & Almeida, 1991; Ahmad et al., 2022; Dalm et al., 2024; Sussillo & Abbott, 2009). These methods are cheap and they are online, but none of them corrects the downstream weights, and none has been linked to natural gradients. A third and parallel literature, in computational neuroscience, has derived recurrent and local dynamics which decorrelate the activity of a population of neurons; we return to it in Section 4, as our recurrent formulation turns out to resemble it closely.

Approximate methods for credit assignment, which replace backpropagation with signals that a distributed system might plausibly transport, are well known to fail upon tasks of even moderate difficulty (Bartunov et al., 2018), and it has been suggested that their choice of metric is partly to blame (Surace et al., 2020). Notably, such methods, being approximate methods, by definition contain errors in the gradient signal, however their forward pass computations are not approximate. The factorization above therefore predicts that the right factor is exactly that part of the natural gradient which remains repairable within networks and without resorting to gradient computations.

Contributions

In this work we uncover each of these aspects of right-sided natural gradient descent and contribute the following:

1. We show an implementation of right-sided natural gradient descent which, when forward weights are corrected, has three exactly equivalent forms: a gradient preconditioner, a decorrelating linear transform in the forward pass, or a set of recurrent dynamics with purely Hebbian updates.

2. We show that this transform can be learned by a rank-one update, requiring no matrix inverse, no matrix decomposition, and no information beyond that which is presently available at an individual synapse. It therefore could be applied in distributed and physically constrained systems in which existing preconditioners cannot be implemented.

3. We derive the correcting update to the forward weights and show it to be the element missing from existing methods of online decorrelation (Silva, 1991; Ahmad et al., 2022; Dalm et al., 2024; Huang et al., 2018). Without it, every decorrelation step displaces the network upon its loss landscape by an amount which is neither measured nor controlled.

4. We demonstrate empirically that our method matches or outperforms Shampoo and Muon when paired with backpropagation, and that it significantly improves a range of approximate methods for credit assignment.

2 METHODS

2.1 DATA CORRELATIONS PLACE A LAYER'S PARAMETERS IN A NON-ORTHONORMAL BASIS

It is ultimately rather trivial to show why data correlations ruin gradient descent. Consider a set of data with some correlation structure, $\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle = \boldsymbol{\Sigma}$. Supposing this data to be well conditioned, one may equally write it as a whitened dataset multiplied by the matrix square root of its correlation matrix, $\boldsymbol{x} = \boldsymbol{\Sigma}^{1/2}\bar{\boldsymbol{x}}$, where $\langle \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top \rangle = \boldsymbol{I}$. The output of a linear model of these inputs may then be written $\hat{\boldsymbol{z}} = \boldsymbol{W}\boldsymbol{x} = (\boldsymbol{W}\boldsymbol{\Sigma}^{1/2})\bar{\boldsymbol{x}}$, in which the correlation matrix appears as a matrix by which our parameters are brought into a non-orthonormal relationship.[1] Thus, if we compute gradient descent with respect to $\boldsymbol{W}$ without accounting for the correlation inducing matrix $\boldsymbol{\Sigma}^{1/2}$, we update parameters which no longer stand in an orthonormal relationship to one another.

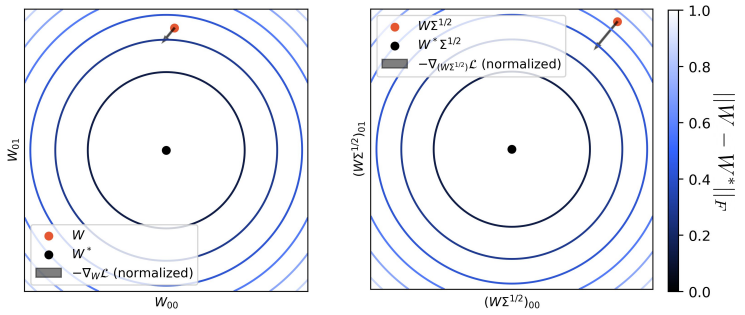

Figure 1: Left, the gradient descent vector is shown for a linear regression problem, though now the input data $\boldsymbol{x}$ has a correlation structure ($\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle = \boldsymbol{\Sigma}$). Right, if the correlation matrix is integrated into the parameters, alignment with the direction of steepest descent in this landscape is returned.

Figure 1 visualises this effect for a linear regression problem, equivalent to a single layer of a network. When the data carries a correlation structure, gradient descent points off-axis and takes a detour during optimization. Viewed in a parameterization which absorbs $\boldsymbol{\Sigma}^{1/2}$ into the parameters, the same vector once again points toward the minimum. The right factor of Equation 2 is precisely the transformation which undoes this, and from this observation arrives our main proposal: removing the input correlations at every layer of a deep neural network would improve gradient descent directions for **any** algorithm, approximate or otherwise.

There are two routes by which this may be achieved. One may measure the correlation structure, invert it, and apply that inverse to the gradient, which is the gradient preconditioning described in Section 1.2. Alternatively, one may modify the network itself, such that the data arriving at every layer is already decorrelated and ordinary gradient descent is thereby already operating in an orthonormal basis. We pursue the second route. In Section 2.4 we show that, provided one additional correction is made, these two routes are not merely similar but exactly equivalent.

2.2 A DECORRELATING TRANSFORM, LEARNED ONLINE

At every layer of a network we construct a linear transformation of the data, $\bar{\boldsymbol{x}}_i = \boldsymbol{M}_i\boldsymbol{x}_i$, where $\bar{\boldsymbol{x}}_i$ is intended to be a decorrelated form of the data $\boldsymbol{x}_i$. A network thereby gains one additional linear transformation per layer and its computation is written

$$\boldsymbol{x}_i = \phi(\boldsymbol{h}_i) = \phi(\boldsymbol{W}_i\bar{\boldsymbol{x}}_{i-1}) = \phi(\boldsymbol{W}_i\boldsymbol{M}_{i-1}\boldsymbol{x}_{i-1}),$$

with all elements as described in Section 1.1, and with the addition of a square decorrelating matrix at every layer.

[1] Note that we specifically use the term *orthonormal* rather than *orthogonal*. Because our derivation is based on a loss-distance metric, both the cross-terms (correlations) and the diagonal terms (variances) of the data define the parameter impact. Thus, properly preconditioning the gradients requires both orthogonalizing the features and normalizing their variances to exactly restore an orthonormal basis.

Suppose that we wish to whiten the data, such that $\left\langle \bar{\boldsymbol{x}}_{i-1}\bar{\boldsymbol{x}}_{i-1}^\top \right\rangle = \boldsymbol{I}$.[2] Silva (1991) propose to minimise a whitening loss equivalent to $\mathcal{L}_{\text{Whiten}}(\boldsymbol{M}_i) = \frac{1}{4}\| \left\langle \boldsymbol{M}_i \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{M}_i^\top \right\rangle_{\boldsymbol{x}_i} - \boldsymbol{I}\|_F^2$. Notably, this loss is itself minimised at a linear transformation, and so is subject to precisely the effect described above. Applying right-sided natural gradient descent to it, rather than ordinary gradient descent, we derive the update equation

$$\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i - \eta_M \left(\bar{\boldsymbol{x}}_i \bar{\boldsymbol{x}}_i^\top - \boldsymbol{I}\right) \boldsymbol{M}_i, \tag{3}$$

which is similar in form to those of Silva (1991) and Ahmad et al. (2022), and whose derivation is given in Appendix D. This update requires only the decorrelated activity of a layer and the matrix itself, and so is available wherever that activity is.

2.3 Correcting the forward weights

One issue which arises is that every update to the decorrelation matrix via Equation 3 alters the function which the network computes as it alters the forward pass. The consequence is a displacement of the network upon its task's loss landscape as we show in Appendix F. In other words, the minimisation of the whitening loss competes against the task loss. Such an effect also disturbs any momentum terms and gradient traces upon which modern optimizers depend. Existing methods for online decorrelation (Silva, 1991; Ahmad et al., 2022; Dalm et al., 2024; Huang et al., 2018) ignore this as a factor, and control try to limit its effect by keeping η_M small, even though one would prefer it to be as large as possible in order to track a changing correlation structure.

We propose instead to remove this conflict, by deriving the exact change in the parameters $\boldsymbol{W}_i$ that would ensure that any learning done for decorrelation does not affect the network's output at all, but only the basis in which that layer represents and uses its inputs. Formally, we require that before vs after a decorrelation update, the effective map of a layer remains unchanged:

$$\boldsymbol{W}_i^{(u)} \boldsymbol{M}_{i-1}^{(u)} \boldsymbol{x}_{i-1} = \boldsymbol{W}_i \boldsymbol{M}_{i-1} \boldsymbol{x}_{i-1},$$

where the superscript (u) denotes the updated matrices. Because the update of Equation 3 is a rank-one modification when applied to a single sample, this condition may be solved by the Sherman-Morrison inverse without any matrix decomposition, giving

$$\boldsymbol{W}_i^{(u)} = \frac{1}{\gamma}\boldsymbol{W}_i + \frac{\eta_M \boldsymbol{h}_i \bar{\boldsymbol{x}}_{i-1}^\top}{\gamma^2 - \eta_M \gamma \bar{\boldsymbol{x}}_{i-1}^\top \bar{\boldsymbol{x}}_{i-1}}, \tag{4}$$

with $\gamma = 1 + \eta_M$, and with the full derivation given in Appendix E. Note that this update is itself Hebbian, being an outer product of a layer's input $\bar{\boldsymbol{x}}_{i-1}$ with its own output $\boldsymbol{h}_i$ and is easily computed.

With this correction in place, the network's function is preserved exactly at every step, and the decorrelating transform is free to learn as quickly as it likes. It is worth noting what this buys beyond stability. Because the function computed by the network is unchanged, the transform cannot remove information which the network might otherwise have used, and the concern raised by Wadia et al. (2021), that whitening a dataset restricts the space of generalization unless carefully regularised, is mitigated. Our networks are whitened at every layer and yet express exactly the family of functions they would have expressed without whitening.

2.4 Three equivalent forms

We may now state the central result of this work. Equations 3 and 4 together describe a single mechanism which admits three implementations, and these implementations are exactly equivalent in the updates which they produce. It is convenient to state this in terms of the *effective* map of a layer, $\boldsymbol{V}_i = \boldsymbol{W}_i \boldsymbol{M}_{i-1}$, being the linear transformation which the layer actually applies to its input, and to write $\boldsymbol{\delta}_i = \partial \ell / \partial \boldsymbol{h}_i$ for the gradient arriving at that layer.

Theorem 1 (Three equivalent forms)**.** Let $\boldsymbol{M}_{i-1}$ be updated by Equation 3 upon a single sample. Then the three procedures below induce identical updates to the effective map $\boldsymbol{V}_i = \boldsymbol{W}_i \boldsymbol{M}_{i-1}$:

[2] In contrast, the objective of decorrelation is $\left\langle \bar{\boldsymbol{x}}_{i-1}\bar{\boldsymbol{x}}_{i-1}^\top \right\rangle = \boldsymbol{D}$, for an arbitrary diagonal matrix $\boldsymbol{D}$. Therefore, whitening is a special case of decorrelation.

	Layer input	Weight update
(i)	$\boldsymbol{h}_i = \boldsymbol{W}_i \boldsymbol{x}_{i-1}$	$\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \eta\, \boldsymbol{\delta}_i \boldsymbol{x}_{i-1}^\top \boldsymbol{M}_{i-1}^\top \boldsymbol{M}_{i-1}$
(ii)	$\boldsymbol{h}_i = \boldsymbol{W}_i \bar{\boldsymbol{x}}_{i-1}$, $\bar{\boldsymbol{x}}_{i-1} = \boldsymbol{M}_{i-1} \boldsymbol{x}_{i-1}$	$\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \eta\, \boldsymbol{\delta}_i \bar{\boldsymbol{x}}_{i-1}^\top$, then Equation 4
(iii)	$\boldsymbol{h}_i = \boldsymbol{W}_i \bar{\boldsymbol{x}}_{i-1}$, $\tau \dot{\bar{\boldsymbol{x}}}_{i-1} = \boldsymbol{x}_{i-1} - \boldsymbol{R}_{i-1} \bar{\boldsymbol{x}}_{i-1}$, $\boldsymbol{R}_{i-1} = \boldsymbol{M}_{i-1}^{-1}$	$\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \eta\, \boldsymbol{\delta}_i \bar{\boldsymbol{x}}_{i-1}^\top$, then Equations 4 and 7

Whenever $\boldsymbol{M}_{i-1}$ additionally sits at the whitening fixed point, this common update is the right-sided natural gradient update of Equation 2.

Under (ii), gradient descent upon $\boldsymbol{W}_i$ gives $\delta \boldsymbol{W}_i = -\eta \boldsymbol{\delta}_i \bar{\boldsymbol{x}}_{i-1}^\top = -\eta \boldsymbol{\delta}_i \boldsymbol{x}_{i-1}^\top \boldsymbol{M}_{i-1}^\top$, so the induced change in the effective map is $\delta \boldsymbol{V}_i = \delta \boldsymbol{W}_i \boldsymbol{M}_{i-1} = -\eta \boldsymbol{\delta}_i \boldsymbol{x}_{i-1}^\top \boldsymbol{M}_{i-1}^\top \boldsymbol{M}_{i-1}$, which is exactly the update of (i). Equation 4 guarantees that the remaining change to $\boldsymbol{V}_i$, that arising from the update of $\boldsymbol{M}_{i-1}$ itself, is zero. Equivalence of (ii) and (iii) follows from the fixed point of the dynamics being $\bar{\boldsymbol{x}}_{i-1} = \boldsymbol{R}_{i-1}^{-1} \boldsymbol{x}_{i-1} = \boldsymbol{M}_{i-1} \boldsymbol{x}_{i-1}$, with $\boldsymbol{R}_{i-1}$ maintaining that inverse relationship by induction. The final statement follows from $\boldsymbol{M}_{i-1}^\top \boldsymbol{M}_{i-1} = \langle \boldsymbol{x}_{i-1} \boldsymbol{x}_{i-1}^\top \rangle^{-1}$ at the whitening fixed point. Since the equivalence holds for the effective map of every layer, it holds for the network as a whole, and the backward pass is unaffected: propagating through $\boldsymbol{V}_i^\top = \boldsymbol{M}_{i-1}^\top \boldsymbol{W}_i^\top$ in (ii) and (iii) is identical to propagating through $\boldsymbol{V}_i^\top$ in (i).

Note, the equivalences we have outlined above strictly hold only for single-sample updates. For a mini-batch of B samples the corresponding update is of rank $\min(B, n_i)$, and its exact inverse requires the Woodbury identity together with a $B \times B$ inversion, which is no longer computable from quantities available at a single connection. However note that the regime in which our method is exact (single-sample online updating) is also the regime in which its locality matters, as neither a nervous system nor an analogue device processes a mini-batch.

As a gradient preconditioner At the whitening fixed point, $\boldsymbol{M}_i \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle \boldsymbol{M}_i^\top = \boldsymbol{I}$, and therefore $\boldsymbol{M}_i^\top \boldsymbol{M}_i = \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle^{-1}$. The transform thus carries an estimate of the inverse input correlation matrix, and one may dispense with the forward transformation altogether and write the update as

$$\boldsymbol{W}_i \leftarrow \boldsymbol{W}_i - \eta_W \frac{\partial \ell}{\partial \boldsymbol{h}_i} \boldsymbol{x}_{i-1}^\top \boldsymbol{M}_{i-1}^\top \boldsymbol{M}_{i-1}, \quad (5)$$

which is right-sided natural gradient descent as it appears in Equation 2, with the inverse correlation supplied by $\boldsymbol{M}$ rather than by a matrix inversion. Note that far from the fixed point, $\boldsymbol{M}_i^\top \boldsymbol{M}_i$ is not the inverse correlation matrix, but instead akin to a (typical) preconditioner which is learning its parameters and continuously tuning to the correlation structure of the data.

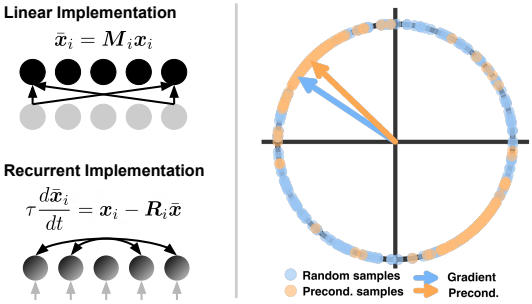

Figure 2: Left, our proposed method can be equivalently represented as either a single linear transformation or the fixed point of a recurrent set of dynamics (see the main text). Right, randomly sampled directions on a skewed quadratic loss function landscape that fall within 90 degrees of the natural/preconditioned gradient direction can be improved by our method.

As a linear transform Alternatively, one may propagate the decorrelated activities $\bar{\boldsymbol{x}}_i$ through the network as in Section 2.2, apply ordinary gradient descent to $\boldsymbol{W}_i$, and additionally apply the correction of Equation 4 after every update of $\boldsymbol{M}_i$. The network then performs ordinary gradient descent in a basis which is continuously tending toward being orthonormal and comes at the cost of increased computational complexity for forward passes.

As a system of recurrent dynamics Finally, the same transform may be realised as the fixed point of a set of linear recurrent dynamics,

$$\tau \frac{d\bar{\boldsymbol{x}}_i}{dt} = \boldsymbol{x}_i - \boldsymbol{R}_i \bar{\boldsymbol{x}}_i, \tag{6}$$

which is exactly equivalent to the linear transform above when $\boldsymbol{R}_i = {\boldsymbol{M}_i}^{-1}$. The matrix $\boldsymbol{R}_i$ need never be inverted, but may be updated directly by the same Sherman-Morrison argument used for Equation 4, giving

$$\boldsymbol{R}_i^{(u)} = \frac{1}{\gamma}\boldsymbol{R}_i + \frac{\eta_M \boldsymbol{x}_i \bar{\boldsymbol{x}}_i^\top}{\gamma^2 - \eta_M \gamma \bar{\boldsymbol{x}}_i^\top \bar{\boldsymbol{x}}_i}. \tag{7}$$

Equations 4 and 7 have the same form, and both are Hebbian. See Figure 2, left, for a depiction of the linear and recurrent implementations.

The equivalence of these three forms is the point of this section, and it has a consequence which we wish to state plainly. In the recurrent form, every quantity appearing in Equations 3, 4 and 7 is available at the connection being updated: the activity entering it, and the activity leaving it. There is no matrix inverse, no decomposition, and no gathering of a layer's statistics by machinery outside of the network. Nor is any of it computed from the gradient, so the whole mechanism runs during the forward pass, before a backward pass exists. Right-sided natural gradient descent is thereby available in settings where a preconditioner is not: in distributed systems in which no coordinator exists, in analogue and neuromorphic hardware in which matrix inversion is not a primitive, and in nervous systems, in which a synapse knows only what its two neurons are doing.

Computational cost All three forms add in memory a square matrix of order $\mathcal{O}(n_i^2)$ per layer during training, where n_i is the number of nodes in layer i, along with the cost of propagating inputs through it and of updating it. For convolutional layers, note that this matrix acts upon the patch dimension rather than the spatial extent of a feature map, and so is of size $(Ck^2)^2$ for C channels and a $k \times k$ kernel, which is considerably smaller than a naive reading of $\mathcal{O}(n_i^2)$ would suggest. To give an indication of wall-clock costs, we report these in Appendix I. The recurrent form's efficiency is highly dependent upon the software and hardware upon which it is implemented, and one should not consider it efficient unless applied in custom neuromorphic, analogue, or other exotic hardware, nervous systems included.

3 Results

We test our proposed approach by training a LeNet-like network and a five hidden-layer convolutional network upon CIFAR-10 and CIFAR-100 with backpropagation (BP), by feedback alignment (FA) (Lillicrap et al., 2016), and by uniform sign-concordance (SC) (Liao et al., 2016). We also test in VGG-7, VGG-9, and ResNet-18 architectures with error predictive coding (ePC) (Goemaere et al., 2025), a more recent local learning method which has been shown to scale to deeper networks. Network architectures and training hyperparameters are given in Appendix G, and pseudocode for both the preconditioning and the linear-transform implementations in Appendix J.

Figure 3 shows test-set performances for all three credit assignment methods, with and without our method. Every one of these algorithms computes its update in a basis where data correlations have made non-orthonormal, and so every one of them gains from a return to an orthonormal basis, backpropagation included. Consistent with our proposal that the effect of right-sided gradient correction is separate to the gradient quality, the least accurate credit assignment method, feedback alignment, gains the most from our method, and the most accurate, backpropagation, gains the least.

Section 1.2 observed that existing preconditioners require statistics of the gradient and a centralised matrix operation, neither of which our method needs, but whose impact could be nonetheless important. Figure 4 compares our method against Shampoo (Gupta et al., 2018) and Muon (Jordan et al., 2024) in the convolutional network upon CIFAR-100, the same comparison across every network and dataset provided in Appendix H.

With backpropagation and with sign-concordance, our method matches or outperforms both modern optimizers. We take this to be the central claim of this work: a correction which never leaves the forward pass, never inverts a matrix, and never gathers the statistics of a layer into one place is

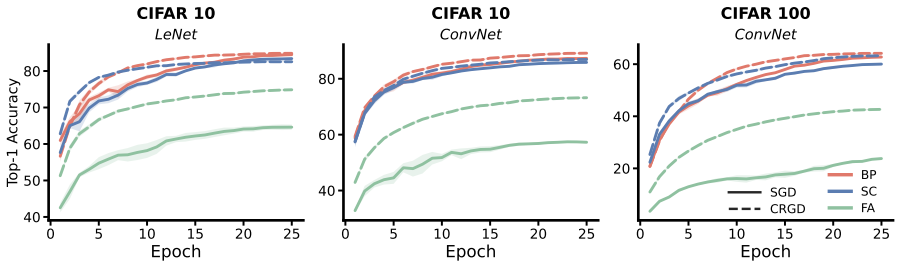

Figure 3: Computing right-sided natural gradient updates massively helps in generalisation performance of approximate backpropagation algorithms. Results are shown for backpropagation (BP), feedback alignment (FA), and sign-concordant-fa (SC) trained upon the CIFAR10/100 in a LeNet-like and a five hidden-layer convolutional networks. Network architectures and training hyperparameters are described in detail in Appendix G. Envelopes show the standard deviation in accuracy levels across five randomly seeded networks.

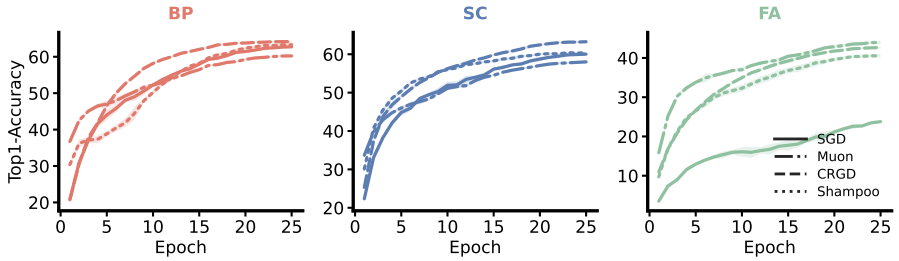

Figure 4: Our method compared against Shampoo (Gupta et al., 2018) and Muon (Jordan et al., 2024), for each credit assignment method, in the five hidden-layer convolutional network trained upon CIFAR-100. Envelopes show the standard deviation across five randomly seeded networks. The same comparison for every network and dataset is given in Appendix H. Note that our correction is computed during the forward pass and from locally available quantities alone, whereas both baselines require statistics of the gradient together with a matrix decomposition or orthogonalization carried out outside of the network.

nonetheless competitive with preconditioners which do all three. Feedback alignment is the exception, and Muon outperforms our method here. We attribute this to the fact that feedback alignment suffers from two distinct gradient estimation difficulties: its updates suffer from a low-rank tendency in it's credit assignment subspace (Boeshertz et al., 2026), and updates are additionally expressed in a non-orthonormal basis we have identified. Our method acts strictly upon the second of these factors, while Muon can also address the gradient rank issue. That a method targeting the gradient factor might win where the gradient factor is the dominant problem, while a method targeting the input factor wins elsewhere, precisely aligns with the separation which we create between backward-pass based and forward-pass based corrections.

Table 1: Top-1 test accuracy for error-based predictive coding (ePC), for ePC combined with our method, and for backpropagation, across deeper architectures. Mean and standard deviation across seeds. Bold marks the better of the two ePC variants. Backpropagation is given in grey as a reference rather than as a competing method, and is italicised where it exceeds both.

Model	Dataset	ePC	ePC + CRGD	BP
VGG-7	CIFAR-10	88.84 ± 0.31	$\mathbf{90.59 \pm 0.22}$	89.60 ± 0.16
	CIFAR-100	58.62 ± 0.20	$\mathbf{64.24 \pm 0.30}$	$\mathit{65.14 \pm 0.29}$
VGG-9	CIFAR-10	86.81 ± 0.09	$\mathbf{88.85 \pm 0.27}$	$\mathit{89.76 \pm 0.20}$
	CIFAR-100	60.65 ± 0.25	$\mathbf{63.46 \pm 0.45}$	61.11 ± 0.45
ResNet-18	CIFAR-10	91.73 ± 0.21	$\mathbf{92.41 \pm 0.08}$	91.85 ± 0.24
	CIFAR-100	69.47 ± 0.32	$\mathbf{71.26 \pm 0.23}$	$\mathit{71.46 \pm 0.32}$

To establish that these effects are not confined to the small networks or simple credit assignment algorithms above, we additionally apply our method to error-based predictive coding (ePC) (Goemaere et al., 2025) in deeper architectures. Table 1 shows that our method improves ePC in every architecture and upon both datasets, and that the combination closes much of the remaining distance to backpropagation, exceeding it in several settings. The margins here are of a few percent rather than the far larger ones seen above, which is consistent with predictive coding supplying a considerably better estimate of the gradient than feedback alignment does.

4 Discussion

Herein we have shown that the input-side factor of the natural gradient need not be computed by an optimizer at all. It can be removed by the network itself, provided the forward weights are simultaneously corrected. The three forms of Theorem 1 are the consequence of this finding, and the recurrent form describes one half of natural gradients as being computable during the forward pass, from quantities present at a single connection, with no inverse and no decomposition required. Existing preconditioners apply their correction to the gradient because that is where an optimizer has access, however one half can in fact happen, locally, in parallel, and without any gradient computation.

Our decorrelating transform is close in form to rules proposed repeatedly over three decades (Silva, 1991; Silva & Almeida, 1991; Ahmad et al., 2022; Dalm et al., 2024), and the practice of whitening layer activations is well established (Huang et al., 2018; 2019; Luo, 2017). We add theoretical backing, an efficient learning rule, and a correction to the forward weights, Equation 4, which holds the network's function fixed while decorrelation is carried out and without which our recurrent method would have no preconditioner equivalent.

In computational neuroscience, lateral connections updated in proportion to the correlation of the activities they connect, so-called anti-Hebbian learning, have been proposed as a mechanism for competition and decorrelation in populations of neurons for a similar length of time (Oja, 1989; Földiák, 1990; Pehlevan et al., 2015; Lipshutz & Simoncelli, 2024) with experimental support in the centre-surround processing of early vision (Pitkow & Meister, 2012) and in the spatial decorrelation produced by inhibitory plasticity (He & Cline, 2019). Our results and derivation of Equation 7 suggest another interpretation of this body of work: a circuit which decorrelates its own inputs is, for whatever else it may be doing, placing the synapses downstream of it into an orthonormal basis, making ordinary Hebbian-style learning at those synapses behave as natural gradient descent. Furthermore, the update of Equation 4 then adds the suggestion that such systems should also adjust a neuron's forward weights so as to hold its output fixed while the representation arriving at it is reorganised, a form of homeostatic plasticity. That these two properties emerge from an optimization level derivation are remarkable and hint at a relationship between the optimization of credit assignment and the homeostatic and inhibitory properties of biological neural networks.

Having said all of the above, practically speaking the equivalences we derive are exact only for single-sample updates and, as discussed in Section 2.4, a mini-batch update scheme would require the computation of at least one inverse matrix. Our method also requires the addition of a square matrix at every network layer, which for very wide and deep networks is a real cost. Finally our evaluation is confined to convolutional and residual networks at the CIFAR task scale. Exploring efficient scaling options for very large networks and for larger datasets, and exploring the method's applicability to other credit assignment methods, are natural next steps.

Finally, our method repairs the input factor of Equation 2 and leaves the gradient factor alone, whereas methods such as Muon do the reverse, and the results of Section 3 suggest the two failures to be genuinely distinct. A method which repaired both, one within the network and one within the optimizer, seems to us the natural thing to try next though questions remain as to how it could be implemented in a biological system or neuromorphic platform.

AI use statement

We have not used generative AI tools for any of the tasks listed as 'required disclosure' for the ICLR 2027 conference.

We have **not** used generative AI tools to: help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, implement methods, or to interpret results.

The following were not applicable to this work: generate synthetic data sets, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, and to support qualitative and thematic data analysis.

Additionally (recommended disclosure), we **used** generative AI tools to: create or edit software code, summarize or analyse existing literature, edit a research paper to improve readability, and to identify relevant literature.

We have reviewed all AI-assisted work. We never directly allowed a generative AI tool to modify our manuscript and instead always iterated for clarity on text or sections and then placed and edited them to fit. All code was checked and developed by the authors and AI usage was limited to refactoring or reorganising code and was checked multiple times post-hoc. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

N/A

Reproducibility statement

In order to ensure reproducibility, we provide pseudocode for both the preconditioning and the linear-transform implementations of our method in Algorithms 1 and 2 of Appendix J. Appendix G describes the network architectures, the datasets and the augmentations applied to them, and the training hyperparameters, the latter including the ranges searched for every optimizer (Table 3), the configuration selected for each combination of network, credit assignment method and optimizer which we report (Table 4), and those used for the predictive coding experiments (Table 5). For our theoretical results, the assumptions upon which Theorem 1 rests are stated in Section 2.4, including the restriction to single-sample updates, and the derivations are given in full in Appendices B, C, D and E. Lastly, we make available all code used to generate the experiments at <SEE ATTACHED ZIP>.

References

Nasir Ahmad, Ellen Schrader, and M Gerven. Constrained parameter inference as a principle for learning. *Trans. Mach. Learn. Res.*, 2023, March 2022.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, February 1998.

Kang An, Yuxing Liu, Rui Pan, Yi Ren, Shiqian Ma, Donald Goldfarb, and Tong Zhang. Asgo: Adaptive structured gradient optimization. *Advances in Neural Information Processing Systems*, 38:126775–126814, 2026.

Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in neural information processing systems*, 31, 2018.

Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. *Advances in neural information processing systems*, 31, 2018.

Gauthier Boeshertz, Razvan Pascanu, and Claudia Clopath. Overcoming rank collapse in feedback alignment. *arXiv preprint arXiv:2606.11123*, 2026.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 4(5):11, 2015.

Sander Dalm, Joshua Offergeld, Nasir Ahmad, and Marcel van Gerven. Efficient deep learning with decorrelated backpropagation. May 2024. arXiv: 2405.02385.

Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. Natural neural networks. *Advances in neural information processing systems*, 28, 2015.

P Földiák. Forming sparse representations by local anti-hebbian learning. *Biol. Cybern.*, 64(2): 165–170, 1990.

Cédric Goemaere, Gaspard Oliviers, Rafal Bogacz, and Thomas Demeester. epc: Fast and deep predictive coding in digital simulation. *arXiv preprint arXiv:2505.20137*, 2025.

Vineet Gupta, Tomer Koren, and Y Singer. Shampoo: Preconditioned stochastic tensor optimization. *ICML*, pp. 1837–1845, February 2018.

Hai-Yan He and Hollis T Cline. What is Excitation/Inhibition and how is it regulated? a case of the elephant and the wisemen. *J. Exp. Neurosci.*, 13:1179069519859371, June 2019.

T Heskes. On 'natural' learning and pruning in multi-layered perceptrons. *Neural Comput.*, 12(4): 881–901, April 2000.

Lei Huang, Dawei Yang, B Lang, and Jia Deng. Decorrelated batch normalization. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 791–800, April 2018.

Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4869–4878, April 2019.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Qianli Liao, Joel Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

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, November 2016.

David Lipshutz and Eero P Simoncelli. Shaping the distribution of neural responses with interneurons in a recurrent circuit model. May 2024.

Ping Luo. Learning deep architectures via generalized whitened neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2238–2246. PMLR, 2017.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. March 2015.

Erkki Oja. NEURAL NETWORKS, PRINCIPAL COMPONENTS, AND SUBSPACES. *Int. J. Neural Syst.*, 01(01):61–68, January 1989.

Cengiz Pehlevan, Tao Hu, and Dmitri B Chklovskii. A Hebbian/Anti-Hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data. *Neural Comput.*, 27(7):1461–1495, July 2015.

Xaq Pitkow and Markus Meister. Decorrelation and efficient coding by retinal ganglion cells. *Nat. Neurosci.*, 15(4):628–635, March 2012.

Fernando M Silva. A distributed solution for data orthonormalization. In *Proc. 1991 Int. Conf. Artificial Neural Networks*, pp. 943–948, 1991.

Fernando M Silva and Luís B Almeida. Speeding-up backpropagation by data orthonormalization. In *Artificial Neural Networks*, pp. 1503–1506. Elsevier, 1991.

Simone Carlo Surace, Jean-Pascal Pfister, Wulfram Gerstner, and Johanni Brea. On the choice of metric in gradient-based theories of brain function. *PLoS computational biology*, 16(4):e1007640, 2020.

David Sussillo and L F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, August 2009.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing shampoo using adam. *arXiv [cs.LG]*, September 2024.

Neha Wadia, Daniel Duckworth, Samuel S Schoenholz, Ethan Dyer, and Jascha Sohl-Dickstein. Whitening and second order optimization both make information in the dataset unusable during training, and can reduce or prevent generalization. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10617–10629. PMLR, 2021.

Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*, 2018.

A Background on Gradient Descent and Natural Gradients

A.1 Gradient Descent

Consider the case in which we have a dataset which provides input and output pairs $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$, and we wish for some parameterised function, $\boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$, with parameters $\boldsymbol{\theta}$ to produce a mapping relating these. This mapping would be optimal if it minimised a loss function $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \ell(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})$, with a sample-wise loss function ℓ.

The problem which we wish to solve in general is to minimise our loss function and find the optimal set of parameters, effectively to find $\text{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$. However, finding this minimum directly is challenging for most interesting problems. Gradient descent proposes a first-order optimization process by which we identify an update direction (not directly the optimized value) for our parameters based upon a linearization of our loss function. This is often formulated as taking a 'small step' in the steepest descent direction of a function, in its gradient direction, such that

$$\delta\boldsymbol{\theta}^{\text{GD}} = -\eta\nabla\mathcal{L}(\boldsymbol{\theta}),$$

where η is the step size.

However, this supposition of taking a 'small step' in the gradient direction hides a specific assumption. In fact, it is equivalent to minimising a linear approximation (1st order Taylor expansion) of our loss function with an added penalization based upon the change in our parameters. Specifically, this is the optimum solution of

$$\delta\boldsymbol{\theta}^{\text{GD}} = \underset{\delta\boldsymbol{\theta}}{\text{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^\top\delta\boldsymbol{\theta} + \frac{1}{2\eta}\delta\boldsymbol{\theta}^\top\delta\boldsymbol{\theta}.$$

where, $\delta\boldsymbol{\theta}^\top\delta\boldsymbol{\theta} = ||\delta\boldsymbol{\theta}||_2^2 = \sum_i \delta\boldsymbol{\theta}_i^2$. Note, one can derive gradient descent by simply finding the minimum of this optimization function. Thus, gradient descent assumes that it is sensible to measure and limit the distance of our parameter update in terms of the squared (Euclidean) norm of the parameter change. Natural gradients supposes that this is not the best choice.

A.2 Natural Gradient Descent

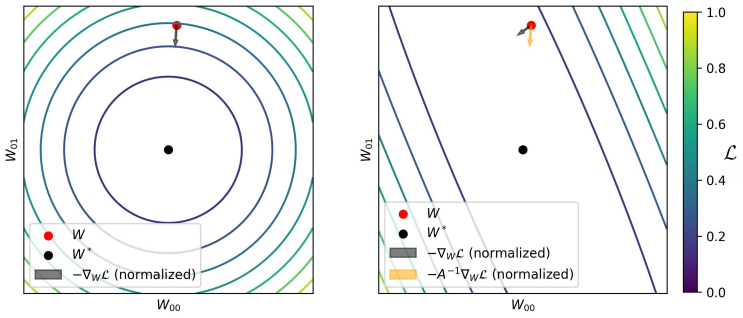

Figure 5: Left is shown a loss function landscape (in parameter space) which is a well-conditioned convex loss. In red is shown a model being optimised and the gradient descent direction (grey arrow) points toward the minimum of the function. Right, the loss function is modified to include a positive semi-definite scaling matrix ($\mathcal{L} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{X}}(\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y})^\top\boldsymbol{A}(\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y})$), which causes a skew and stretch of the loss landscape. Gradient descent no longer points toward the minimum but instead in the direction of steepest descent (a detour), whereas natural gradient descent (orange arrow) points directly once more at the loss functions minimum.

Amari (1998), proposed the concept of natural gradients, and put forth the Riemannian metric as the sensible alternative to a regular Euclidean measurement of distance. Specifically, rather than define squared distance with the Euclidean metric ($|\delta\boldsymbol{\theta}|^{\text{Euclidean}^2} = \sum_i \delta\boldsymbol{\theta}_i^2$), you can instead use a general metric instead,

$$|\delta\boldsymbol{\theta}|^{\text{Riemannian}} = \sum_i\sum_j g_{ij}\delta\boldsymbol{\theta}_i\delta\boldsymbol{\theta}_j = \delta\boldsymbol{\theta}^\top\boldsymbol{G}(\boldsymbol{\theta})\delta\boldsymbol{\theta}$$

where $\boldsymbol{G}$ is the Riemannian metric matrix. Note, that the values of the matrix, $\boldsymbol{G}$, are a function of $\boldsymbol{\theta}$ and are thus not static. We can now make use of our Riemannian metric in place of the previous Euclidean metric, and by finding the minimum of this optimization, we can arrive at a neat formulation of natural gradient descent,

$$\delta\boldsymbol{\theta}^{\text{NGD}} = -\eta\boldsymbol{G}^{-1}\nabla\mathcal{L}(\boldsymbol{\theta}).$$

Natural gradient descent aims to ensure that it is not the steepest direction of descent which is taken, but instead the direction which undoes any skew in the loss function to arrive at a more direct descent toward (local) minima. This difference is illustrated in Figure 5.

B LOSS DISTANCE: THE APPROPRIATE RIEMANN METRIC FOR NATURAL GRADIENTS

In the main text, we propose that one may find the appropriate form of the Riemann metric for gradient descent by supposing that the metric measured should tend to the total summed sample-wise loss distance

$$\delta\theta^\top G \delta\theta \to \frac{1}{|\mathcal{D}|} \sum_{(x_0,y)\in\mathcal{D}} (\ell(f(\theta+\delta\theta, x_0), y) - \ell(f(\theta, x_0), y))^2.$$

By expanding the loss function with it's Taylor series,

$$\begin{aligned}
(\ell(f(\theta+\delta\theta, x_0), y) - \ell(f(\theta, x_0), y))^2 &= (\ell + \nabla_\theta \ell \cdot \delta\theta + \delta\theta^\top \nabla^2_\theta \ell \delta\theta + ... - \ \ell)^2 && (8)\\
&= (\delta\theta^\top \nabla_\theta \ell + \delta\theta^\top \nabla^2_\theta \ell \delta\theta + ...)^2 && (9)\\
&= \delta\theta^\top \nabla_\theta \ell \nabla_\theta \ell^\top \delta\theta + 2(\nabla_\theta \ell \delta\theta \times \delta\theta^\top \nabla^2_\theta \ell \delta\theta) + ... && (10)\\
& && (11)
\end{aligned}$$

As the change in parameters tend to small values (or equivalently as $\lim_{\eta\to 0}$), this term can be approximated by its first term. This first term, when used as the measure, is thus a simple equivalence to the Riemann metric

$$\delta\theta^\top G \delta\theta = \delta\theta^\top \left\langle \nabla_\theta \ell \nabla_\theta \ell^\top \right\rangle_{x_0\in\mathcal{D}} \delta\theta.$$

C Derivation for the Riemann metric for a deep neural network

Defining our model as

$$x_i = \phi(h_i) = \phi(W_i x_{i-1})$$

for $i \in [1...L]$, where L is the number of layers in our network, ϕ is a non-linear activation function (which we shall assume is a fixed continuous transfer function for all layers), W_i is a matrix of parameters for layer i, x_0 is our input data, and x_L is our model output. We ignore biases for now as a simplification of our derivation.

First let us begin by collecting all parameter matrices into a single vector of parameters, such that $\theta = \text{Vec}(W_1, W_2, ...W_L)^\top$ where $\text{Vec}(\cdot)$ indicates the flattening of a tensor into a vector. Next we can define, the derivative of our loss with respect to the parameters,

$$\nabla_\theta \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \text{Vec}(W_1, W_2, ..W_L)} \right) = \left\langle \text{Vec}\left(\frac{\partial \ell}{\partial h_1} x_0^\top, \frac{\partial \ell}{\partial h_2} x_1^\top, ... \frac{\partial \ell}{\partial h_L} x_{L-1}^\top \right) \right\rangle_{x_0 \in \mathcal{D}}$$

where we have replaced the derivative terms with the layer computation as can be calculated by backpropagation. We can now compute more straightforwardly the outer product of this gradient vector with itself, such that

$$\begin{aligned} G(\theta) &= \left\langle \nabla_\theta \ell \nabla_\theta \ell^\top \right\rangle_{x_0} \\ &= \left\langle \text{Vec}\left(\frac{\partial \ell}{\partial h_1} x_0^\top, \frac{\partial \ell}{\partial h_2} x_1^\top, ... \frac{\partial \ell}{\partial h_L} x_{L-1}^\top \right) \text{Vec}\left(\frac{\partial \ell}{\partial h_1} x_0^\top, \frac{\partial \ell}{\partial h_2} x_1^\top, ... \frac{\partial \ell}{\partial h_L} x_{L-1}^\top \right)^\top \right\rangle_{x_0} \end{aligned}$$

This is now a matrix of shape $G(\theta) \in \mathbb{R}^{M \times M}$ assuming $\theta \in \mathbb{R}^M$.

Rather than making explicit how to calculate the entire matrix, is is more fruitful to break down this computation into the separate blocks of this term, such that the (i, j)-block of our matrix is computed

$$G_{ij} = \left\langle \text{Vec}\left(\frac{\partial \ell}{\partial h_j} x_{j-1}^\top \right) \text{Vec}\left(\frac{\partial \ell}{\partial h_i} x_{i-1}^\top \right)^\top \right\rangle_{x_0} = \left\langle \left(x_{j-1} \otimes \frac{\partial \ell}{\partial h_j} \right) \left(x_{i-1} \otimes \frac{\partial \ell}{\partial h_i} \right)^\top \right\rangle_{x_0}$$

where $\otimes$ represents the Kronecker (Zehfuss) product. The mixed-product property of the Kronecker product allows us to also re-formulate this as

$$G_{ij} = \left\langle \left(x_{j-1} x_{i-1}^\top \right) \otimes \left(\frac{\partial \ell}{\partial h_j} \frac{\partial \ell}{\partial h_i}^\top \right) \right\rangle_{x_0} = \left\langle x_{j-1} x_{i-1}^\top \right\rangle_{x_0} \otimes \left\langle \frac{\partial \ell}{\partial h_j} \frac{\partial \ell}{\partial h_i}^\top \right\rangle_y .$$

Note that we here made a separation of the expectation values based upon the fact that the loss value depends entirely on label or target output, y, and the network activities depend entirely on the value of the inputs, x_0. We assume, as Bernacchia et al. (2018) do, that these can therefore be separately averaged.

There are now two possible ways to proceed, each requiring an alternative assumption. First, one may suppose that we decide to optimise only a single weight matrix of the network at a time, which would allow us to ignore all but the diagonal blocks of our matrix, G. Bernacchia et al. (2018) alternatively arrived at a sole consideration of the (inverse) diagonal blocks of G by assuming a linear network (i.e. that $\phi(x) = x$), and showing that in such a case, the matrix G is singular and that it's pseudo-inverse can be taken using only the diagonal blocks.

Regardless, if we take a single block at a time, and suppose that we are only updating single weight matrices, we can now formulate the natural gradients learning rule as

$$
\begin{aligned}
\delta\theta_{W_i} &= -\eta \langle \nabla_{\theta_{W_i}} \ell \, \nabla_{\theta_{W_i}} \ell^\top \rangle_{x_0}^{-1} \nabla_{\theta_{W_i}} \mathcal{L} \\
&= -\eta \left(\left\langle x_{i-1} x_{i-1}^\top \right\rangle_{x_0} \otimes \left\langle \frac{\partial \ell}{\partial h_i} \frac{\partial \ell}{\partial h_i}^\top \right\rangle_y \right)^{-1} \nabla_{\theta_{W_i}} \mathcal{L} \\
&= -\eta \left(\left\langle x_{i-1} x_{i-1}^\top \right\rangle_{x_0}^{-1} \otimes \left\langle \frac{\partial \ell}{\partial h_i} \frac{\partial \ell}{\partial h_i}^\top \right\rangle_y^{-1} \right) \text{Vec} \left\langle \frac{\partial \ell}{\partial h_i} x_{i-1}^\top \right\rangle_{x_0} \\
&= -\eta \text{Vec} \left(\left\langle \frac{\partial \ell}{\partial h_i} \frac{\partial \ell}{\partial h_i}^\top \right\rangle_y^{-1} \left\langle \frac{\partial \ell}{\partial h_i} x_{i-1}^\top \right\rangle_{x_0} \left\langle x_{i-1} x_{i-1}^\top \right\rangle_{x_0}^{-1} \right)
\end{aligned}
$$

Thus, we arrive at an expression which allows us to better interpret the impact of computing the natural gradient. We can appreciate that the natural gradient formulation thus achieves two things: - The left matrix multiplication removes any skew and mis-scaling of the loss function with respect to the hidden activities, dealing with the problem that we classically associate with a skewed loss landscape - The right matrix multiplication removes the impact of any *correlation structure in our input data*. Correlation structure in our input-data is equivalent to our parameters living in a non-orthonormal basis set and thus affects the speed and accuracy of training!

D MINIMIZING THE WHITENING LOSS VIA RIGHT-SIDED GRADIENT DESCENT

We wish to minimize the loss

$$\mathcal{L}_{\text{Whiten}}(\boldsymbol{M}_i) = \frac{1}{4}\| \left\langle \boldsymbol{M}_i \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{M}_i^\top \right\rangle_{\boldsymbol{x}_i} - \boldsymbol{I}\|_F^2,$$

without being ruined by data correlations. Therefore, our update equation will be of the form

$$\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i - \eta_M \nabla \mathcal{L}_{\text{Whiten}}(\boldsymbol{M}_i) \langle \boldsymbol{x}_i \boldsymbol{x}_i \rangle_{\boldsymbol{x}_i}^{-1}$$

First, we take the derivative of the loss with respect to $\boldsymbol{M}_i$:

$$\begin{aligned}
\nabla \mathcal{L}_{\text{Whiten}}(\boldsymbol{M}_i) &= \left(\langle \boldsymbol{M}_i \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{M}_i^\top \rangle_{\boldsymbol{x}_i} - \boldsymbol{I} \right) \langle \boldsymbol{M}_i \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i} \\
&= \left(\boldsymbol{M}_i \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i} \boldsymbol{M}_i^\top - \boldsymbol{I} \right) \boldsymbol{M}_i \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i}
\end{aligned}$$

Filling the gradient back in gives us the update equation

$$\begin{aligned}
\boldsymbol{M}_i \leftarrow & \boldsymbol{M}_i - \eta_M \nabla \mathcal{L}_{\text{Whiten}}(\boldsymbol{M}_i) \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i}^{-1} \\
= & \boldsymbol{M}_i - \eta_M \left(\boldsymbol{M}_i \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i} \boldsymbol{M}_i^\top - \boldsymbol{I} \right) \boldsymbol{M}_i \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i} \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i}^{-1} \\
= & \boldsymbol{M}_i - \eta_M \left(\boldsymbol{M}_i \langle \boldsymbol{x}_i \boldsymbol{x}_i^\top \rangle_{\boldsymbol{x}_i} \boldsymbol{M}_i^\top - \boldsymbol{I} \right) \boldsymbol{M}_i
\end{aligned}$$

This update equation requires us to aggregate the data correlations before updating. In practice, we use stochastic gradient descent to relax the equation to arrive at our stochastic update

$$\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i - \eta_M \left(\boldsymbol{M}_i \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{M}_i^\top - \boldsymbol{I} \right) \boldsymbol{M}_i.$$

E OUR DECORRELATION MECHANISM AS A RECURRENT SYSTEM OF DYNAMICS

We desire $\boldsymbol{W}_i^{(u)} \boldsymbol{M}_{i-1}^{(u)} \boldsymbol{x}_{i-1} = \boldsymbol{W}_i \boldsymbol{M}_{i-1} \boldsymbol{x}_{i-1}$, where the superscript (u) denotes the updated matrices. In the following, we will drop the subscripts as it should be obvious from the context. We begin the derivation using Equation 3

$$\begin{aligned}
\boldsymbol{W}^{(u)} \boldsymbol{M}^{(u)} \boldsymbol{x} &= \boldsymbol{W} \boldsymbol{M} \boldsymbol{x} \\
\boldsymbol{W}^{(u)} \boldsymbol{M}^{(u)} &= \boldsymbol{W} \boldsymbol{M} \\
\boldsymbol{W}^{(u)} (\boldsymbol{M} - \eta_M (\bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top - \boldsymbol{I}) \boldsymbol{M}) &= \boldsymbol{W} \boldsymbol{M} \\
\boldsymbol{W}^{(u)} &= \boldsymbol{W} \boldsymbol{M} \left(\boldsymbol{M} - \eta_M (\bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top - \boldsymbol{I}) \boldsymbol{M} \right)^{-1} \\
\boldsymbol{W}^{(u)} &= \boldsymbol{W} \boldsymbol{M} \left((1 + \eta_M) \boldsymbol{I} - \eta_M \bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top) \boldsymbol{M} \right)^{-1} \\
\boldsymbol{W}^{(u)} &= \boldsymbol{W} \left((1 + \eta_M) \boldsymbol{I} - \eta_M \bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top \right)^{-1}
\end{aligned}$$

In this form, we can apply the Sherman-Morrison inverse.

$$\begin{aligned}
\left((1 + \eta_M) \boldsymbol{I} - \eta_M \bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top \right)^{-1} &= \frac{1}{(1 + \eta_M)} \boldsymbol{I} + \eta_M \frac{\frac{1}{(1+\eta_M)^2} \bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top}{1 - \frac{\eta_M}{1+\eta_M} \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}} \\
&= \frac{1}{(1 + \eta_M)} \boldsymbol{I} + \eta_M \frac{\bar{\boldsymbol{x}} \bar{\boldsymbol{x}}^\top}{(1 + \eta_M)^2 - \eta_M (1 + \eta_M) \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}}
\end{aligned}$$

We can now define $\gamma = 1 + \eta_M$ to slightly simplify this equation and plug it back in.

$$\begin{aligned}
\boldsymbol{W}^{(u)} &= \boldsymbol{W}\left((1+\eta_M)\boldsymbol{I} - \eta_M \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top\right)^{-1} \\
&= \boldsymbol{W}\left(\frac{1}{\gamma}\boldsymbol{I} + \eta_M \frac{\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top}{\gamma^2 - \eta_M \gamma \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}}\right) \\
&= \frac{1}{\gamma}\boldsymbol{W} + \eta_M \frac{\boldsymbol{h}\bar{\boldsymbol{x}}^\top}{\gamma^2 - \eta_M \gamma \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}}
\end{aligned}$$

Note that in the last step, we have used that $\boldsymbol{h}_i = \boldsymbol{W}_i \boldsymbol{x}_{i-1}$. This finishes the derivation for Equation 4.

To derive Equation 7, we can use a similar approach. We iteratively want to update $\boldsymbol{R}_i$ to be equivalent to $\boldsymbol{M}_i$ at every point during the training process. Therefore, we initialize the two matrices as the identity matrix, so that the equivalence holds at the first time-step. Then, we make an inductive argument to derive the update equation to ensure that this equivalence holds after updating $\boldsymbol{M}_i$.

$$\begin{aligned}
\boldsymbol{R}_i^{(u)} &= (M_i^{(u)})^{-1} \\
&= \left(\boldsymbol{M} - \eta_M(\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top - \boldsymbol{I})\boldsymbol{M}\right)^{-1} \\
&= \left((1+\eta_M)\boldsymbol{I} - \eta_M \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top)\boldsymbol{M}\right)^{-1} \\
&= \boldsymbol{R}\left((1+\eta_M)\boldsymbol{I} - \eta_M \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top\right)^{-1}
\end{aligned}$$

We can reuse the derivation of the inverse using Sherman-Morrison to derive the update equation.

$$\begin{aligned}
\boldsymbol{R}_i^{(u)} &= \boldsymbol{R}\left((1+\eta_M)\boldsymbol{I} - \eta_M \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top\right)^{-1} \\
&= \boldsymbol{R}\left(\frac{1}{\gamma}\boldsymbol{I} + \eta_M \frac{\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top}{\gamma^2 - \eta_M \gamma \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}}\right) \\
&= \frac{1}{\gamma}\boldsymbol{R} + \eta_M \frac{\boldsymbol{x}\bar{\boldsymbol{x}}^\top}{\gamma^2 - \eta_M \gamma \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}}
\end{aligned}$$

F PROBLEMS OF ITERATIVE DECORRELATION

Here, we want to demonstrate that updating a decorrelation matrix, without accounting for the change in the network leads to instabilities during convergence.

We consider a linear neural network parameterized by $\boldsymbol{W}_1$ that maps from multiple inputs to a single output, and decorrelation matrix $\boldsymbol{M}_0$. Further, we consider a simple mean-squared error loss function $\mathcal{L}(\boldsymbol{W}_1, \boldsymbol{M}_0, \mathcal{D}) = \frac{1}{2}\left\langle (\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y})^2 \right\rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in D}$. We are interested in the loss difference between this network, and the network upon updating the decorrelation matrix denoted by $\boldsymbol{M}_0^{(u)} = \boldsymbol{M}_0 + \delta\boldsymbol{M}_0$.

$$
\begin{aligned}
\epsilon &= L(\boldsymbol{W}_1, \boldsymbol{M}_0, \mathcal{D}) - L(\boldsymbol{W}_1, \boldsymbol{M}_0^{(u)}, \mathcal{D}) \\
&= \frac{1}{2}\left\langle (\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y})^2 - (\boldsymbol{W}_1(\boldsymbol{M}_0 + \delta\boldsymbol{M}_0)\boldsymbol{x}_0 - \boldsymbol{y})^2 \right\rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in D}. \\
&= \frac{1}{2}\left\langle (\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y})^2 - ((\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y}) + (\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0))^2 \right\rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in D}. \\
&= \frac{1}{2}\left\langle 2(\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y})(\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0) - (\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0)^2 \right\rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in D} \\
&= \left\langle (\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y})(\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0) - \frac{1}{2}(\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0)^2 \right\rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in D} \\
&= \left\langle (\boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{x}_0 - \boldsymbol{y})(\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0) - \frac{1}{2}(\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{x}_0)^2 \right\rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in D} \\
&= \boldsymbol{W}_1\boldsymbol{M}_0\boldsymbol{\Sigma}\delta\boldsymbol{M}_0^\top\boldsymbol{W}_1^\top - \boldsymbol{c}\delta\boldsymbol{M}_0^\top\boldsymbol{W}_1^\top - \frac{1}{2}\boldsymbol{W}_1\delta\boldsymbol{M}_0\boldsymbol{\Sigma}\delta\boldsymbol{M}_0^\top\boldsymbol{W}_1^\top,
\end{aligned}
$$

where $\boldsymbol{\Sigma} = \langle \boldsymbol{x}_0\boldsymbol{x}_0^\top \rangle_{\boldsymbol{x}_0\in\mathcal{D}}$, and $\boldsymbol{c} = \langle \boldsymbol{x}_0\boldsymbol{y} \rangle_{\boldsymbol{x}_0, \boldsymbol{y}\in\mathcal{D}}$.

What we can see is that as long as our decorrelation updates are not the zero matrix, a loss is induced in our network propagated by the downstream matrices and our matrix change. Both the sign and magnitude of the loss difference are under-determined, but depend on the weights, the decorrelation matrix, and the inputs and outputs of the network. In other words, we have no control over where the model ends up after a decorrelation update. We can only tone down the influence through the decorrelation learning rate, although we, at the same time, wish for it to be as high as possible to drive down the decorrelation loss.

As a consequence, this form of decorrelation without adjusting the weights leads to a strong dependence on careful tuning of the learning rate. Furthermore, we expect that very deep networks may be very unstable to train. Not only due to the highly fluctuating inputs to every intermediate layer, but also due to the momentum terms commonly used in optimization that are becoming inaccurate after every decorrelation update.

G ARCHITECTURES AND HYPERPARAMETERS

The networks trained for the experiments of Figure 3 are of two types: a LeNet-like convolutional architecture and a deeper five hidden-layer convolutional architecture. Both are given in Table 2. Training and testing use the CIFAR-10 and CIFAR-100 datasets (Krizhevsky, 2009), each composed of a training set of 50,000 images, split into either 10 or 100 classes, and a test set of 10,000 images. We apply the standard data augmentations for the training set: random crops to 32×32 pixels with padding $= 4$, random horizontal flips, and normalization.

Table 2: The neural network architectures for the experiments in Figure 3. For convolutional and pooling layers, shapes are organised as 'Kernel Width $\times$ Kernel Height $\times$ Output Channels (Stride, Padding)'. Each parameterized layer is followed by an exponential linear unit (ELU) activation function (Clevert et al., 2015).

NETWORK	LAYER TYPES	LAYER SIZE
LENET ARCHITECTURE	INPUT	$32 \times 32 \times 3$
	CONV	$5 \times 5 \times 32, (1, 2)$
	MAXPOOL	$3 \times 3, (2, 1)$
	CONV	$5 \times 5 \times 32, (1, 2)$
	MAXPOOL	$3 \times 3, (2, 1)$
	LINEAR	512
	LINEAR	10
CONVNET ARCHITECTURE (CIFAR10/100)	INPUT	$32 \times 32 \times 3$
	CONV	$3 \times 3 \times 128, (1, 1)$
	MAXPOOL	$2 \times 2(2, 0)$
	CONV	$3 \times 3 \times 128, (1, 1)$
	MAXPOOL	$2 \times 2(2, 0)$
	CONV	$3 \times 3 \times 256, (1, 1)$
	MAXPOOL	$2 \times 2(2, 0)$
	CONV	$3 \times 3 \times 256, (1, 1)$
	MAXPOOL	$2 \times 2(2, 0)$
	CONV	$3 \times 3 \times 512, (1, 1)$
	MAXPOOL	$2 \times 2(2, 0)$
	FC	10 (OR 100 FOR CIFAR 100)

Table 3: Ranges searched for each optimizer. Every hyperparameter was drawn from a log-uniform distribution over the interval given, with 100 trials for each combination of network, credit assignment method and optimizer. A dash marks a hyperparameter which does not apply.

OPTIMIZER	LEARNING RATE	WEIGHT DECAY	ADDITIONAL
SGD	$[10^{-5}, 5 \times 10^{-1}]$	$[10^{-5}, 10^{-3}]$	–
CRGD (OURS)	$[10^{-5}, 5 \times 10^{-1}]$	$[10^{-5}, 10^{-3}]$	$\eta_M \in [10^{-7}, 10^{-4}]$
SHAMPOO	$[10^{-5}, 5 \times 10^{-1}]$	$[10^{-5}, 10^{-3}]$	–
MUON	$[10^{-4}, 5 \times 10^{-1}]$	$[10^{-6}, 10^{-2}]$	ADAMW LR $\in [10^{-4}, 10^{-1}]$

The training hyper-parameters which were not searched over were held fixed across all simulations. We used a mini-batch size of 128 and trained for 25 epochs, scheduling the learning rate with a cosine scheduler which decays it from its selected value to one hundredth of that value over the course of training. All model weights were initialized using the Kaiming-Uniform scheme, and decorrelation weights as the identity. The forward weights were optimized by SGD with a momentum of 0.9, except where an alternative optimizer is named in Table 4, and the cross-entropy loss was used throughout except where stated otherwise.

For every combination of network, credit assignment method and optimizer we ran a separate random search of 100 trials, drawing each hyperparameter independently from a log-uniform distribution over the ranges of Table 3 and selecting the best configuration upon a held-out validation set. All

Table 4: Hyperparameters selected for the simulations of Figures 3 and 4, for every combination of network, credit assignment method and optimizer. A dash marks a hyperparameter which does not apply to that optimizer: the decorrelation learning rate η_M exists only for our method, and the AdamW learning rate only for Muon, which applies AdamW to those parameters it does not orthogonalize. Values are given to three significant figures.

Dataset, Network	Method	Optimizer	Learning rate	Weight decay	Decor. LR	AdamW LR
CIFAR-10, LeNet	BP	SGD	2.12×10^{-2}	4.90×10^{-4}	–	–
		CRGD (ours)	3.86×10^{-2}	2.14×10^{-4}	5.51×10^{-6}	–
		Shampoo	3.89×10^{-2}	7.34×10^{-4}	–	–
		Muon	1.14×10^{-2}	1.97×10^{-3}	–	2.12×10^{-3}
	SC	SGD	2.77×10^{-2}	5.91×10^{-4}	–	–
		CRGD (ours)	1.80×10^{-2}	5.74×10^{-5}	4.05×10^{-5}	–
		Shampoo	3.51×10^{-2}	8.07×10^{-4}	–	–
		Muon	1.24×10^{-2}	2.92×10^{-5}	–	2.53×10^{-3}
	FA	SGD	5.80×10^{-3}	5.11×10^{-4}	–	–
		CRGD (ours)	4.13×10^{-2}	2.88×10^{-4}	9.73×10^{-5}	–
		Shampoo	7.65×10^{-3}	6.34×10^{-4}	–	–
		Muon	3.07×10^{-3}	6.26×10^{-6}	–	2.76×10^{-4}
CIFAR-10, ConvNet	BP	SGD	3.16×10^{-2}	1.92×10^{-5}	–	–
		CRGD (ours)	4.52×10^{-2}	5.40×10^{-4}	1.69×10^{-5}	–
		Shampoo	1.96×10^{-2}	6.43×10^{-4}	–	–
		Muon	7.65×10^{-3}	6.25×10^{-4}	–	2.39×10^{-3}
	SC	SGD	1.98×10^{-2}	8.37×10^{-4}	–	–
		CRGD (ours)	3.82×10^{-2}	4.10×10^{-4}	9.29×10^{-7}	–
		Shampoo	4.22×10^{-2}	3.84×10^{-4}	–	–
		Muon	5.89×10^{-3}	4.05×10^{-4}	–	8.73×10^{-4}
	FA	SGD	1.05×10^{-2}	8.17×10^{-4}	–	–
		CRGD (ours)	3.09×10^{-2}	1.49×10^{-5}	8.49×10^{-5}	–
		Shampoo	1.49×10^{-2}	6.85×10^{-5}	–	–
		Muon	1.91×10^{-3}	1.31×10^{-3}	–	1.43×10^{-4}
CIFAR-100, ConvNet	BP	SGD	3.94×10^{-2}	8.95×10^{-4}	–	–
		CRGD (ours)	4.13×10^{-2}	9.54×10^{-4}	1.61×10^{-6}	–
		Shampoo	5.24×10^{-2}	7.77×10^{-4}	–	–
		Muon	6.60×10^{-3}	1.85×10^{-4}	–	7.42×10^{-3}
	SC	SGD	4.32×10^{-2}	9.58×10^{-4}	–	–
		CRGD (ours)	5.63×10^{-2}	9.97×10^{-4}	7.71×10^{-5}	–
		Shampoo	5.14×10^{-2}	4.63×10^{-4}	–	–
		Muon	5.25×10^{-3}	8.79×10^{-3}	–	5.13×10^{-3}
	FA	SGD	7.55×10^{-3}	1.20×10^{-5}	–	–
		CRGD (ours)	3.11×10^{-2}	1.87×10^{-5}	7.52×10^{-5}	–
		Shampoo	2.00×10^{-2}	1.64×10^{-5}	–	–
		Muon	2.70×10^{-3}	6.69×10^{-3}	–	4.16×10^{-4}

searches were run from a single seed. The selected values are given in Table 4, and those for the error-based predictive coding experiments in Table 5.

Those hyperparameters which were not searched over were held fixed as follows. Forward parameters carried a momentum of 0.9, whereas the decorrelation matrices were updated with neither momentum nor weight decay. Muon was applied to the hidden weight matrices alone, and not to biases nor to the input and output layers of the network, with the remaining parameters updated by AdamW at the searched learning rate, with $\beta = (0.9, 0.95)$ and a weight decay of 0.01. Shampoo was run with $\beta = (0, 0.999)$, $\epsilon = 10^{-12}$, a maximum preconditioner dimension of 8192, SGD grafting, and classical momentum of 0.9 without dampening or Nesterov acceleration. Its preconditioner was recomputed once every 100 steps, which is the amortisation discussed in the Introduction: between those refreshes the transform which Shampoo applies is that of a layer's gradient statistics as they stood up to 100 updates previously. Final results were thereafter produced upon the full training and test sets, with five randomly seeded network models for each curve. For BP, SC and FA we adapted the implementation of Xiao et al. (2018). The predictive coding models were taken directly from the implementation of Goemaere et al. (2025), whose choices we follow throughout. For these

Table 5: Hyperparameters for the error-based predictive coding experiments of Table 1, for both the cross-entropy and mean-squared-error objectives. The learning rate and weight decay are those reported by Goemaere et al. (2025) and are shared by both the ePC baseline and ePC + CRGD; the decorrelation learning rate η_M applies to ePC + CRGD alone. Values are given to three significant figures.

Model	Dataset	Loss	Learning rate	Weight decay	Decor. LR η_M
VGG-7	CIFAR-10	Cross-entropy	1.56×10^{-3}	5.46×10^{-4}	2.50×10^{-4}
	CIFAR-10	MSE	4.26×10^{-4}	2.16×10^{-6}	2.50×10^{-4}
	CIFAR-100	Cross-entropy	3.13×10^{-4}	7.99×10^{-4}	2.50×10^{-4}
	CIFAR-100	MSE	4.02×10^{-4}	1.47×10^{-5}	1.00×10^{-4}
VGG-9	CIFAR-10	Cross-entropy	5.36×10^{-4}	6.88×10^{-4}	7.50×10^{-5}
	CIFAR-10	MSE	6.61×10^{-4}	4.01×10^{-5}	2.50×10^{-4}
	CIFAR-100	Cross-entropy	3.23×10^{-4}	4.03×10^{-4}	1.00×10^{-4}
	CIFAR-100	MSE	2.01×10^{-4}	2.62×10^{-6}	2.50×10^{-4}
ResNet-18	CIFAR-10	Cross-entropy	3.39×10^{-3}	1.51×10^{-6}	1.00×10^{-4}
	CIFAR-10	MSE	7.65×10^{-4}	1.82×10^{-4}	2.50×10^{-4}
	CIFAR-100	Cross-entropy	3.03×10^{-3}	1.20×10^{-5}	7.50×10^{-5}
	CIFAR-100	MSE	3.67×10^{-4}	7.30×10^{-4}	7.50×10^{-5}

experiments we adopted the learning rates and weight decays reported by Goemaere et al. (2025) rather than searching over them ourselves, and applied them unchanged to both the ePC baseline and to ePC + CRGD. The two therefore differ in the presence of our decorrelating transform and in nothing else, and the only quantity we selected was the decorrelation learning rate η_M, by a grid search over $[2.5\times10^{-4}, 10^{-4}, 7.5\times10^{-5}, 5\times10^{-5}, 2.5\times10^{-5}, 10^{-5}]$. We would note that these are custom architectures rather than the VGG and ResNet models as they are conventionally constructed, and we retain their names only in order that our results be comparable with those reported alongside that implementation. All training was conducted upon a single A100 GPU, with per-epoch costs for every optimizer reported in Appendix I. We note that our code is not optimized for runtime efficiency, and we would expect that the runtime of our decorrelation mechanism can be improved substantially.

H COMPARISON WITH PRECONDITIONED OPTIMIZERS, ACROSS ALL NETWORKS AND DATASETS

Figure 4 of Section 3 shows the comparison against Shampoo and Muon for the convolutional network upon CIFAR-100 alone. Figure 6 gives the same comparison for every network and dataset which we train. The pattern described in the main text holds throughout: our method matches or outperforms both baselines under backpropagation and under sign-concordance, whereas Muon outperforms it under feedback alignment, where the low-rank pathology of the fixed feedback matrices, rather than the input basis, is the dominant problem.

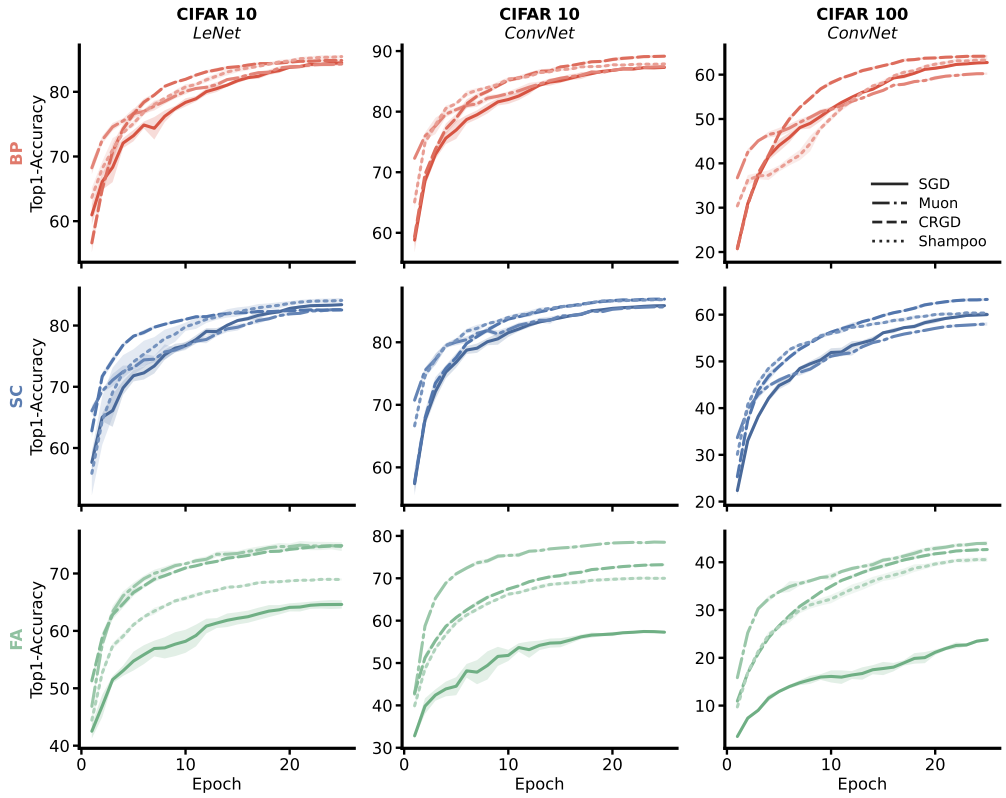

Figure 6: Our method compared against Shampoo (Gupta et al., 2018) and Muon (Jordan et al., 2024), for each credit assignment method (rows) and each network and dataset (columns). Envelopes show the standard deviation across five randomly seeded networks. Note that our correction is computed during the forward pass and from locally available quantities alone, whereas both baselines require statistics of the gradient together with a matrix decomposition or orthogonalization carried out outside of the network.

I WALL-CLOCK COST

Table 6 reports the measured time per training epoch for each optimizer, upon CIFAR-10. Our method costs approximately four times ordinary gradient descent per epoch, which is appreciably less than Shampoo and appreciably more than Muon.

Two remarks are due upon these numbers. The first is that our implementation applies the decorrelating transform as a dense matrix multiplication upon a GPU, which is the least favourable setting for it: none of the distributability which the locality of the update affords is expressed in such a measurement, and the comparison shown here is therefore one of a local method implemented centrally against methods which are centralised by construction. The second is that the cost is borne during training alone. The transform of a trained network may be absorbed into the forward weights of the layer which follows it, leaving only the effective map $\boldsymbol{V}_i = \boldsymbol{W}_i \boldsymbol{M}_{i-1}$ of Section 2.4, and inference therefore proceeds at exactly the cost of the original architecture.

Table 6: Wall-clock time per training epoch, in seconds, upon CIFAR-10 (Conv). Mean and standard deviation across five randomly seeded networks. Ordinary gradient descent is given in grey as a reference point rather than as a competing method.

Credit assignment	SGD	CRGD (ours)	Shampoo	Muon
Backpropagation	4.04 ± 0.17	17.60 ± 0.13	23.21 ± 0.10	5.79 ± 0.16
Sign-concordance	4.16 ± 0.24	17.96 ± 0.13	23.42 ± 0.08	5.72 ± 0.43

J TRAINING PSEUDOCODE

Algorithm 1 Right-sided natural gradient descent through preconditioning

Input: Input data $\boldsymbol{x}_0$, Target Output y, Forward Weights $\boldsymbol{W}_l$, Decorrelation Weights $\boldsymbol{M}_l$, Total Number of Layers L, Forward Learning Rate η_W, Decorrelation Learning Rate η_M

`Compute forward pass`
for $i \in [0, 1, ..L-1]$ **do**
  $\bar{\boldsymbol{x}}_i = \boldsymbol{M}_i \boldsymbol{x}_i$ ▷ Decorrelate data at layer i
  $\boldsymbol{x}_{i+1} = \phi(\boldsymbol{W}_{i+1}\boldsymbol{x}_i)$ ▷ Pass forward to the next layer
end for

`Compute credit assignment method (Example BP)`
for $i \in [L, L-1, ...1]$ **do**
  if $i == L$ **then**
    $\boldsymbol{\delta}_L = \phi'(\boldsymbol{x}_L)(\boldsymbol{x}_L - \boldsymbol{y})$ ▷ Compute gradient at output layer (regression or CCE)
  else if then
    $\boldsymbol{\delta}_i = \phi'(\boldsymbol{x}_i)\boldsymbol{W}_{i+1}^\top \boldsymbol{\delta}_{i+1}$ ▷ Backpropagate (or replace $\boldsymbol{W}_{i+1}$ for FA)
  end if
end for

`Update Forward Parameters`
for $i \in [0, 1, ..L-1]$ **do**
  $\boldsymbol{W}_{i+1} \leftarrow \boldsymbol{W}_{i+1} - \eta_W \boldsymbol{\delta}_{i+1} \boldsymbol{x}_i^\top \boldsymbol{M}_i^\top \boldsymbol{M}_i$ ▷ Precondition, then update by an optimizer
end for

`Update Decorrelation Parameters`
for $i \in [0, 1, ..L-1]$ **do**
  $\boldsymbol{C} = \bar{\boldsymbol{x}}_i \bar{\boldsymbol{x}}_i^\top$ ▷ Compute remaining correlation
  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i - \eta_M(\boldsymbol{C} - \boldsymbol{I})\boldsymbol{M}_i$ ▷ Update decorrelation matrix
end for

Algorithm 1 describes pseudocode for the preconditioning variant for updating a network which is carried out in this work, whereas Algorithm 2 shows the pseudocode for the linear transformation variant with equivalent dynamics. This is shown assuming the presentation of a single sample with its corresponding network update, though is in-practice used with a mini-batch of size 128 by default. The algorithm is shown for the case of backpropagation, though it is compatible with alternative credit assignment methods such as feedback alignment or difference target propagation.

Algorithm 2 Right-sided natural gradient descent through a Linear Decorrelation Transformation

Input: Input data $\boldsymbol{x}_0$, Target Output y, Forward Weights $\boldsymbol{W}_l$, Decorrelation Weights $\boldsymbol{M}_l$, Total Number of Layers L, Forward Learning Rate η_W, Decorrelation Learning Rate η_M

`Compute forward pass`
for $i \in [0, 1, ..L-1]$ **do**
  $\bar{\boldsymbol{x}}_i = \boldsymbol{M}_i \boldsymbol{x}_i$ ▷ Decorrelate data at layer i
  $\boldsymbol{h}_{i+1} = \boldsymbol{W}_{i+1} \bar{\boldsymbol{x}}_i$
  $\boldsymbol{x}_{i+1} = \phi(\boldsymbol{h}_{i+1})$ ▷ Pass forward to the next layer
end for

`Compute credit assignment method (Example BP)`
for $i \in [L, L-1, ...1]$ **do**
  if $i == L$ **then**
    $\boldsymbol{\delta}_L = \phi'(\boldsymbol{x}_L)(\boldsymbol{x}_L - \boldsymbol{y})$ ▷ Compute gradient at output layer (regression or CCE)
  else if then
    $\boldsymbol{\delta}_i = \phi'(\boldsymbol{x}_i)\boldsymbol{M}_i^\top \boldsymbol{W}_{i+1}^\top \boldsymbol{\delta}_{i+1}$ ▷ Backpropagate (or replace $\boldsymbol{W}_{i+1}$ for FA)
  end if
end for

`Update Decorrelation Parameters`
for $i \in [0, 1, ..L-1]$ **do**
  $\boldsymbol{C} = \bar{\boldsymbol{x}}_i \bar{\boldsymbol{x}}_i^\top$ ▷ Compute remaining correlation
  $\gamma = 1 + \eta_M$
  $\boldsymbol{g} = \gamma^2 - \eta_M \gamma (\bar{\boldsymbol{x}}_i^\top \bar{\boldsymbol{x}}_i)$ ▷ Compute rescaling
  $\boldsymbol{M}_i \leftarrow \boldsymbol{M}_i - \eta_M (\boldsymbol{C} - \boldsymbol{I}) \boldsymbol{M}_i$
  $\boldsymbol{W}_{i+1} \leftarrow \frac{1}{\gamma} \boldsymbol{W}_{i+1} + \eta_M \boldsymbol{h}_{i+1} \bar{\boldsymbol{x}}_i^\top / \boldsymbol{g}$ ▷ Update decorrelation matrix
end for

`Update Forward Parameters`
for $i \in [0, 1, ..L-1]$ **do**
  $\boldsymbol{W}_{i+1} \leftarrow \boldsymbol{W}_{i+1} - \eta_W \boldsymbol{\delta}_{i+1} \boldsymbol{x}_i^\top$ ▷ Precondition, then update by an optimizer
end for