Understanding Sample Efficiency in Predictive Coding

Gaspard Oliviers* **Elene Lominadze*** **Rafal Bogacz**
Nuffield Department of Clinical Neurosciences, University of Oxford, United Kingdom
MRC Centre of Research Excellence in Restorative Neural Dynamics, United Kingdom
Correspondence to: rafal.bogacz@ndcn.ox.ac.uk

Abstract

Predictive Coding (PC) is an influential account of cortical learning. Much of recent work has focused on comparing PC to Backpropagation (BP) to find whether PC offers any advantages. Small scale experiments show that PC enables learning that is more sample efficient and effective in many contexts, though a thorough theoretical understanding of the phenomena remains elusive. To address this, we quantify the efficiency of learning in BP and PC through a metric called "target alignment", which measures how closely the change in the output of the network is aligned to the output prediction error. We then derive and empirically validate analytical expressions for target alignment in Deep Linear Networks. We show that learning in PC is more efficient than BP, which is especially pronounced in deep, narrow and pre-trained networks. We also derive exact conditions for guaranteed optimal target alignment in PC and validate our findings through experiments. We study full training trajectories of linear and non-linear models, and find the predicted benefits of PC persist in practice even when some assumptions are violated. Overall, this work provides a mechanistic understanding of the higher learning efficiency observed for PC over BP in previous works, and can guide how PC should be parametrised to learn most effectively.

1 Introduction

Predictive coding (PC) offers an influential account of cortical learning based on minimising prediction errors through local weight updates [1, 2, 3]. PC differs markedly from backpropagation (BP), the dominant training paradigm in modern machine learning [4, 5, 6]. Recent work [7] has suggested that PC is more sample efficient and outperforms BP on learning regimes that are both biologically relevant and particularly prone to catastrophic interference, such as learning from limited data, online parameter updates, and continual learning. It has been argued [7] that these advantages arise because PC parameter updates induce changes in the model's predictions that are better aligned with the output error than those induced by BP (Figure 1A-B). Changes in the model's predictions that are misaligned with the output error are referred to in this paper as *interference* because they introduce errors in directions the model already predicts accurately, so that incorporating new information interferes with preserving prior knowledge. The lack of interference can be quantified by *target alignment*, which measures the similarity between the output error during training and subsequent changes in network's prediction (Figure 1C). The more effective learning performance of PC over BP could therefore be explained by PC's higher target alignment.

An additional perspective arises from the observation that optimal target alignment is equivalent to performing learning along natural gradients [8]. In this regime, the change in the model's predictions is exactly aligned with the output error, yielding the steepest descent in output error [8]. Natural gradient methods are known to accelerate learning [9, 10, 11], reduce sensitivity to parameterisation [12], and improve learning stability [13, 14]. In deep linear networks (DLNs), they also render online and batch learning equivalent [8]. Interpreted through this lens, target alignment provides a measure

*These authors contributed equally to this work.

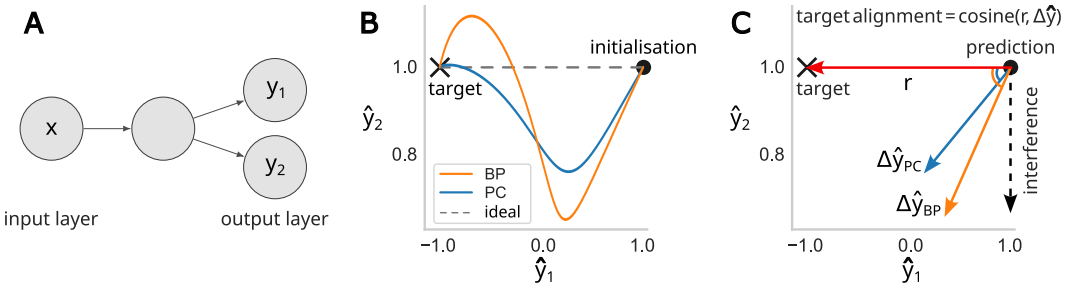


Figure 1: **Target alignment. (A)** Deep Linear Network (DLN) used in the toy model has one input neuron, one hidden neuron and two output neurons y_1 and y_2. All weights are initialised to be one. **(B)** Evolution of a DLN's predictions $\hat{y}_i$ during training with BP and PC. The network receives an input of $x = 1$ with a target output of $y = [-1, 1]$. Training is carried out separately using PC and BP until the model output matches the target. The learning trajectory shown with the grey dashed line is ideal, as the model's prediction changes only for incorrectly predicted output $\hat{y}_1$, remaining unchanged for $\hat{y}_2$ that is already well predicted. PC changes its output more directly towards the target than BP. **(C)** Definition of target alignment, where $r = y - \hat{y}$, while blue and orange vectors show changes in output $\Delta\hat{y}$ after one weight update. Additionally a dashed arrow shows direction perpendicular to the output error r labelled 'interference' because change in model prediction, $\Delta\hat{y}$, that is partially aligned with it creates errors in directions the model already predicts correctly.

of how closely learning dynamics approximate natural gradient descent, thereby indicating potential benefits of PC over BP by operating closer to the natural gradient regime.

However, a principled analytical explanation of when and why PC should outperform BP, and on what classes of problems, has remained elusive. Moreover, existing evidence of potential benefits of PC has largely been confined to small-scale models [7].

This paper provides an analytical account of the interference-reducing properties of PC. We show that, in DLNs, PC reduces the weight-dependent interference that arises in BP. As a consequence, PC exhibits improved target alignment, especially in regimes where model weights are poorly conditioned, including architectures with information bottlenecks and pre-trained models. We derive closed-form expressions for the change in prediction induced by PC and BP updates in DLNs and compare their target alignment. We also characterise how PC's update can be modified to guarantee interference-free learning for learning with individual samples and for a batch of inputs. We show the theoretical benefits offered by higher target alignment can lead to improved sample efficiency throughout longer training trajectories.

Summary of contributions

- We show theoretically that learning in PC has less interference than BP, which is especially pronounced in deep, narrow and pre-trained networks.
- We demonstrate that PC with layer-specific learning rates guarantees interference-free single-sample learning, while PC with weight update rescaling factors enables interference-free batch learning.
- We show that the benefits predicted from the theory translate to longer training trajectories, making PC more sample efficient. We give preliminary evidence that these findings can be extended to different architectures and nonlinear networks.

2 Background and Related Work

2.1 Deep linear networks

Deep linear networks (DLNs) are fully connected linear networks with one or more hidden layers. DLNs have been extensively used to analytically describe the learning dynamics of BP [15, 16, 9, 17] and PC [18, 19]. Despite learning linear models, DLNs exhibit non-linear learning dynamics. These learning dynamics have proved useful for understanding non-linear networks[15]. Similarly, DLNs have been used to analyse PC. For instance, it has been shown that PC's equilibrium activity

interpolates between the feedforward dynamics of BP and backwards-propagated targets, such as those used in target propagation [18].

For a hierarchical model with L layers, a DLNs learns with BP by minimising a standard squared error energy function (which we also rewrite as a dot product for easy comparison with PC below):

$$\mathcal{L}_{BP}(x, y, W) = \frac{1}{2}||y - W_{L:1}x||_2^2 = \frac{1}{2}(y - W_{L:1}x)^\top (y - W_{L:1}x) \tag{1}$$

where x is an input activity, y is a target output activity, the parameter W includes the weight matrices W_l mapping layer $l-1$ to layer l, and $W_{L:1}$ denotes the product of weights from the input to the output layer, $W_{L:1} = W_L ... W_1$.

2.2 Learning with predictive coding

DLNs can also learn using PC. In a PC model with L layers, the activity in each layer is denoted by x_l, with x_0 the input layer and x_L the output layer. During training the input layer is clamped to $x_0 = x$, and the output layer x_L is fixed to the target $x_L = y$. PC trains a linear DLN by minimising the energy function:

$$E(x_0, ..., x_L, W) = \sum_{l=1}^{L} \frac{1}{2}||x_l - W_l x_{l-1}||_2^2. \tag{2}$$

This energy function is minimised through two steps. First, the neural activity of the network is updated to minimise the energy until convergence, and we refer to this process as *inference*. Then, the weights are updated using one gradient step on the energy function, evaluated at the activity minimising energy found during inference.

For a DLN, PC's inference and learning dynamics are analytically tractable. The neural activity in equilibrium of inference x_l^* can be calculated analytically [18]. Substituting the equilibrium activity in hidden layers ($x_l = x_l^*$ for $1 < l < L$) into the energy function of equation 2, allows expressing the energy of PC as a function of just input, target and weights [19]:

$$E(x, y, W) = \frac{1}{2}(y - W_{L:1}x)^\top S^{-1}(y - W_{L:1}x). \tag{3}$$

This energy has analogous forms as for BP (equation 1) differing only in additional term S given by:

$$S = I + \sum_{l=1}^{L-1} W_{L:l+1} W_{L:l+1}^\top. \tag{4}$$

Taking a gradient of equation 3 will allow us to find analytically the change of weights in PC (which in the model is normally computed through inference and subsequent local weight updates).

2.3 Target alignment of learning

Learning updates that follow the shortest path between the current prediction and the target reduce interference [7, 20] and lead to faster convergence [8, 21]. Neither BP nor PC follows this shortest path for a simple DLN with architecture shown in Figure 1A. Figure 1B illustrates how the output prediction of this DLN changes during training. It shows that PC learning trajectories are more closely aligned with the ideal path than those of BP. This behaviour has been observed in both small linear models and deeper non-linear architectures [7].

The extent to which a learning rule follows the ideal update direction can be quantified using target alignment. Target alignment measures how strongly the change in a model's prediction induced by a weight update is aligned with its prediction error, using cosine similarity [7]:

$$\text{Target alignment} = cos(\angle(r, \Delta\hat{y})) = \frac{r^\top \Delta\hat{y}}{\|r\| \, \|\Delta\hat{y}\|} \tag{5}$$

Here, r denotes the output prediction error $r = y - W_{L:1}x$, $\hat{y} = W_{L:1}x$ is the feedforward prediction, and $\Delta\hat{y}$ represents the change in prediction resulting from a weight update. For DLNs, $\Delta\hat{y} = (W_{L:1}^{\text{post}} - W_{L:1}^{\text{pre}})x$, with pre and post indicating the weights' state before and after the update respectively. A target alignment of one indicates perfect alignment, zero indicates orthogonality, and minus one indicates opposite directions. Definition of target alignment for PC and BP is illustrated in Figure 1C.

The reasons why PC exhibits higher target alignment than BP remain unclear. It has been shown that PC inference combines BP-like feedforward activity with signals resembling target propagation [18], and that models trained with target propagation achieve optimal target alignment [22, 7]. From this perspective, PC may attain higher target alignment than BP by effectively integrating BP with target-propagation-like updates [7]. However, this explanation remains empirical and lacks analytical proof.

Alternative learning algorithms with higher target alignment than BP have been studied extensively in machine learning. In particular, natural gradient methods [8] guarantee maximally aligned learning updates by rescaling BP parameter updates with the inverse Fisher information matrix. By accounting for interactions between parameters in their effect on the model output, the Fisher matrix reconditions the gradient to produce updates that achieve optimal target alignment and maximise the reduction of the objective function per update. Natural gradient methods have also been investigated in the context of DLNs [9], but the relevance of natural gradients to large-scale deep learning and to neuroscience remains limited.

3 Results

3.1 PC is more sample efficient for training deep, narrow, and pre-trained networks than BP

To understand when PC yields higher target alignment than BP, we first derive the change in output prediction of a DLN trained with these algorithms. Specifically, we analyse the alignment between changes in the network's prediction and the training output residual, $r = y - \hat{y}$. We show that, in BP, the change in prediction is given by a sum of terms in which r is pre-multiplied by a weight matrix. This matrix distorts the direction of r, thereby introducing interference. In contrast, PC reduces this interference. As a consequence, PC exhibits improved target alignment, especially in deep models that are either narrow in width or with poorly conditioned weights as could be observed in a pre-trained model. We also show that PC's higher target alignment leads to more sample efficient learning than BP. These results are derived for models with an MSE output loss. However, Appendix N generalises the result to arbitrary loss functions.

3.1.1 Interference in BP

In BP, the weight updates follow the steepest descent of the model's objective function with respect to the weights. Although the weight updates occur in discrete steps, we analyse the corresponding continuous-time limit, which holds as long as the learning rate is sufficiently small. This approach aligns with prior studies on the weight dynamics of DLNs under BP [15, 9] and PC [19]. The weight dynamics for an input x and output y are given by:

$$\frac{\partial W_l}{\partial t} = -\nabla_{W_l}\mathcal{L}_{BP} = W_{L:l+1}^\top (y - W_{L:1}x)(W_{l-1:1}x)^\top = W_{L:l+1}^\top r (W_{l-1:1}x)^\top.$$

The change in prediction resulting from the weight dynamics can then be calculated by differentiating using the product rule, as follows:

$$\frac{\partial \hat{y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1}\frac{\partial W_l}{\partial t} W_{l-1:1}x = \sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^\top r \hat{x}_{l-1}^\top \hat{x}_{l-1} \tag{6}$$

where the feed forward activity at layer l equal to $\hat{x}_l = W_{l:1}x$, and $W_{L:L+1}$ and $W_{0:1}$ are equal to the identity matrix. This result is equivalent to previous derivations for DLNs [9].

The derived expression for the change in prediction under BP reveals a single source of interference. In the ideal case, the change in prediction would equal the target residual r. However, the actual update is given by a sum in which each term consists of r pre-multiplied by $W_{L:l-1}W_{L:l-1}^\top$ and post-multiplied by $\hat{x}_{l-1}^\top \hat{x}_{l-1}$. The matrix $W_{L:l-1}W_{L:l-1}^\top$ acts directly on r, thereby distorting its direction and introducing interference. In contrast, the factor $\hat{x}_{l-1}^\top \hat{x}_{l-1}$ is a scalar rather than a matrix. Therefore, it only rescales the update magnitude and does not contribute to interference.

Interference is therefore expected to be large when $W_{L:l-1}W_{L:l-1}^\top$ deviates substantially from the identity matrix. This occurs when $W_{L:l-1}$ is far from orthogonal, which can arise from two sources: (i) architecture: deeper networks introduce more interacting terms, while narrower layers are prone to

rank deficiency, increasing interference. (ii) training: weights may develop correlated features and amplify certain feature directions while suppressing others, leading to poor conditioning.

3.1.2 Interference in PC

Interference in a network trained with PC differs from that in BP due to an additional scaling matrix S in equation 3. This additional term counteracts distortions induced by the weights, attenuating interference. We derive the change in prediction $\frac{\partial \hat{y}}{\partial t}$ of a DLN trained with PC's objective function E given in equation (3). We show in Appendix E that weight dynamics of PC are given by:

$$\frac{\partial W_l}{\partial t} = -\nabla_{W_l} E = W_{L:l+1}^\top \, S^{-1} r \, x_{l-1}^{*\top} \tag{7}$$

The change in prediction resulting from PC's weight dynamics can be calculated by differentiating using the product rule, as follows:

$$\frac{\partial \hat{y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1} \frac{\partial W_l}{\partial t} W_{l-1:1} x = \sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^\top \, S^{-1} r \, x_{l-1}^{*\top} \hat{x}_{l-1} \tag{8}$$

Crucially, this expression differs from BP (equation 6) through the scaling matrix S^{-1}. S was defined in equation (4) and can be written more compactly using notation $W_{L:L+1} = I$ introduced earlier as:

$$S = \sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^\top.$$

The structure of S^{-1} enables cancellation with the weight-dependent terms $W_{L:l+1} W_{L:l+1}^\top$ in the prediction change (equation 8), thereby attenuating interference. When the scalar factor $x_{l-1}^{*\top} \hat{x}_{l-1}$ is constant across layers, interference matrices cancel completely. Under this assumption $x_{l-1}^{*\top} \hat{x}_{l-1} = c$ for all l, the PC target alignment simplifies to:

$$\frac{\partial \hat{y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^\top S^{-1} r \, x_{l-1}^{*\top} \hat{x}_{l-1} = \underbrace{\sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^\top}_{S} S^{-1} r \, c = r \, c, \tag{9}$$

This reveals PC's intrinsic interference-reduction mechanism via S^{-1}, independent of network architecture or training stage. While $x_{l-1}^{*\top} \hat{x}_{l-1}$ is not strictly constant across layers in practice, the scaling matrix S^{-1} substantially reduces interference in PC over BP whenever these quantities remain reasonably consistent.

Theoretical Takeaway: When will PC have significantly less interference than BP?

PC has higher target alignment than BP, assuming pre- and post-inference activity norms in PC are approximately constant across layers. The difference will be significantly different when BP's target alignment is low due to:

- **Architecture.** BP interference increases in **deep** and **narrow** networks, whereas PC interference is less affected by architecture.
- **Training.** As weight conditioning deteriorates during training, BP interference increases, while PC interference is much less affected.

3.1.3 Higher target alignment in PC leads to more sample-efficient learning than BP

We experimentally validate our theoretical predictions that PC has higher target alignment than BP for single weight updates in randomly initialised networks. We also show that PC's higher target alignment leads to more sample efficient learning.

We first evaluate target alignment in randomly initialised DLNs after a single weight update using either BP or PC. We vary network depth and width. We compare two weight initialisation schemes:

Kaiming uniform initialisation, used as Pytorch's default initialisation for linear layers, and Norm-preserving initialisation, which aims to preserve the norm of activity ($||x_l||_2^2$) across all layers in expectation. Kaiming initialisation causes the activity norm of a network to decay rapidly with depth, whereas Norm-preserving initialisations preserve stable activation norms (see Appendix C.1), and thus brings the model closer to satisfying the assumption $x_{l-1}^{*\top}\hat{x}_{l-1} = c$, enabling perfect target alignment (equation 9). Unless stated otherwise, we consider a DLN with input, hidden, and output widths all set to 512, a single hidden layer. Inputs and targets are sampled independently from standard normal distributions, the batch size is one, and results are averaged over ten random seeds.

Figure 2A-D shows that PC achieves better target alignment than BP under both Kaiming and Norm-preserving initialisations. These results indicate that PC's interference-reducing mechanism remains effective even when the activity norm is not constant across layers. Norm-preserving initialisation attains near-perfect alignment across settings. Although alignment decreases slightly with increasing depth, it remains largely insensitive to changes in network width. In contrast, Kaiming initialisation results in poorer overall alignment. Under Kaiming initialisation, PC alignment deteriorates markedly as depth increases and exhibits greater variability across model configurations.

We further evaluate how PC and BP perform over whole training trajectories rather than one training step. To do this, we set up a simple synthetic regression task, whereby the goal of the network is to learn a mapping $W_{data} \sim \mathcal{N}(0, 1/d_{in})$ where d_{in} is the input dimension and $W_{data} \in \mathbb{R}^{d_{out} \times d_{in}}$ ($\mathcal{N}(0, \Sigma)$ denotes a normal distribution with mean 0 and variance Σ, see Appendix A.1 for further details). During training, we measure the quantity $\frac{1}{N}||W_{data} - W_{L:1}||_F^2$ where $N = d_{out} \times d_{in}$. For the experiments labelled "Whole training" in Figure 2, we choose a square neural network with 20 input, hidden, and output units similar to [9]. We vary the depth of the network between 1 and 8 hidden layers, as well as varying the initialisation. By default, we set the batch size to 64 unless stated otherwise. We perform learning rate sweeps for both BP and PC, identifying optimal learning rates for each by sampling 100 points within the region $10^{-3.5} - 10^{0.3}$. We average our results over ten random seeds.

Figure 2E-H shows that for both initialisations, Norm-preserving and Kaiming, PC is more sample efficient while learning over longer training trajectories. Panels E and G are consistent with the results from panels A and C, showing that for a square network with one hidden layer, PC outperforms BP. For eight hidden layer networks depicted in panels F and H, PC with Norm-preserving initialisation clearly outperforms all alternatives considered for the given architecture. The expected benefits also translate when varying widths as demonstrated in Appendix K. Moreover, the results are confirmed on CIFAR-10, CIFAR-100 and STL-10 in Appendix Q.

3.2 Rescaling learning rate for online learning

Although our previous results show that PC exhibits better alignment than BP, its standard formulation does not achieve optimal target alignment. Introducing layer-specific learning rates, ensures that PC achieves optimal alignment for linear models trained on a single sample, irrespective of neural activities, model weights or model architecture.

Theorem 1 (PC with adaptive learning rate)**.** *For layer-specific adaptive learning rates with α_l set to the scalar $(x_{l-1}^{*\top}\hat{x}_{l-1})^{-1}$, PC achieves optimal target alignment when learning on an individual input-output pair. For PC's weight update*

$$\frac{\partial W_l}{\partial t} = -\alpha_l \nabla_{W_l} E, \quad \textit{for } l = 1, \ldots, L-1$$

the change in model prediction is given by:

$$\frac{\partial \hat{y}}{\partial t} = r$$

This theorem is proven in Appendix F.

To assess whether layer-specific learning rates allow PC to achieve perfect alignment, we measure target alignment in a DLN trained with and without scaling under both BP and PC. For BP, we scale the BP weight updates by the inverse of corresponding scaling factors, $\alpha_l = (\hat{x}_{l-1}^{\top}\hat{x}_{l-1})^{-1}$, ensuring a fair comparison. We vary the conditioning of the model weights to illustrate the effect of training

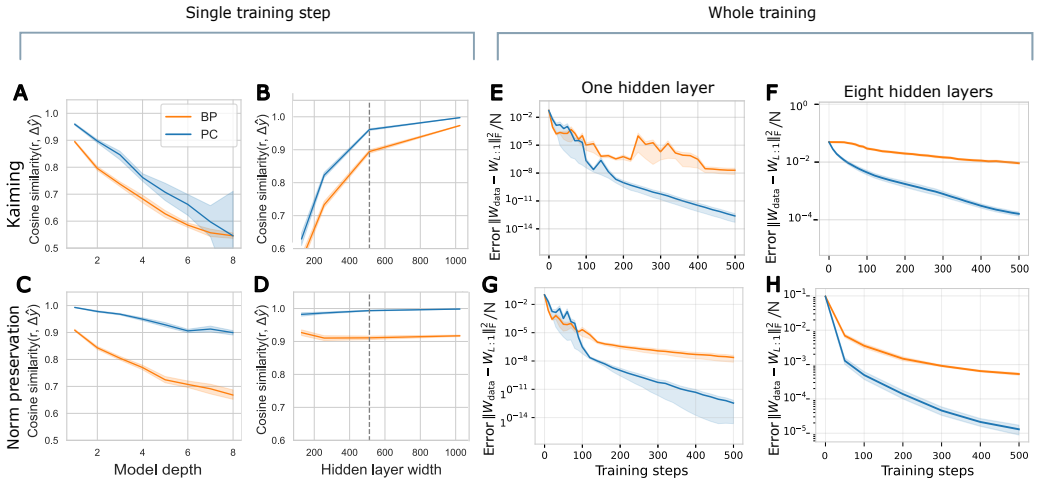


Figure 2: **Target Alignment in Predictive Coding (PC) and Backpropagation (BP). (A-D)** Target alignment of BP (orange) and PC (blue) after one training step for a deep linear network with different architectures and initialisations. The dashed grey lines in panels **B** and **D** denote square networks. **(E, G)** Training dynamics of linear BP and PC under different initialisations and a batch size of 64. The models are square with 20 units per layer and one hidden layer. They are initialised identically and are exposed to identical batches at each training step. **(F, H)** Identical setup to experiments in panels **E** and **G** but for networks with eight hidden layers. The high variability of PC's target alignment for Kaiming initialisation is investigated in Appendices O and P.

discussed in section 3.1.2. The conditioning is quantified using the condition number (see details in Appendix D.2), which is the ratio of the largest to the smallest singular value. Larger condition numbers indicate poorer conditioning. All other experimental settings are set to the default values of the previous experiments, and models were initialised using Kaiming uniform initialisation.

Figure 3A confirms that introducing layer-specific learning rates ensures that PC always learns with the maximal target alignment of one. BP benefits from layer-specific scaling factors as well, but does not achieve maximal alignment or independence from architecture. Increasing the condition number of the weight matrices reduces target alignment for both PC and BP, except for PC with activity-dependent learning rates. This behaviour is expected, as a higher condition number corresponds to stronger interference from the weight product $W_{L:l+1}W_{L:l+1}^\top$.

In panels B and C of Figure 3, we again initialised DLNs with Norm-preservation and a square architecture with 1 hidden layer in panel B and 8 hidden layers in panel C. We tested all four models on the online synthetic regression task and carried out extensive learning rate sweeps over the range $10^{-3.5} - 10^{-0.04}$ (see Appendix K.1). Figure 3B shows the training dynamics of the square network. In the online setting, while PC and scaled PC still outperform BP and scaled BP, the scaling provides little benefit for sample efficiency at the beginning of training. This also holds for deeper networks displayed in Figure 3C. To understand why this happens, we draw an analogy to natural gradients, which is what the weight updates for rescaled PC reduce to for larger batch sizes. Estimating the Fisher Information Matrix with a single data point, as we do for online learning, diminishes its geometric advantages. However, some benefits can still be observed as the number of training steps becomes large. Appendix K shows experimental results as the width is varied between $15, 20$ and 40 hidden units for a DLN with 1 hidden layer.

Figure 3C shows a small nonlinear autoencoder with 3 hidden layers trained on MNIST [23] (see Appendix A.3 for architecture and training details). Again, rescaling provides limited benefit for batch size 1. We expect the performance of PC and rescaled PC to greatly benefit from batch training due to the emerging similarity between rescaled PC and natural gradients as the batch size grows.

3.3 Weight update rescaling for batch training

Training on a mini-batch introduces additional sources of interference in both BP and PC. This interference depends on the norm of layer activities within and across samples. This result is derived

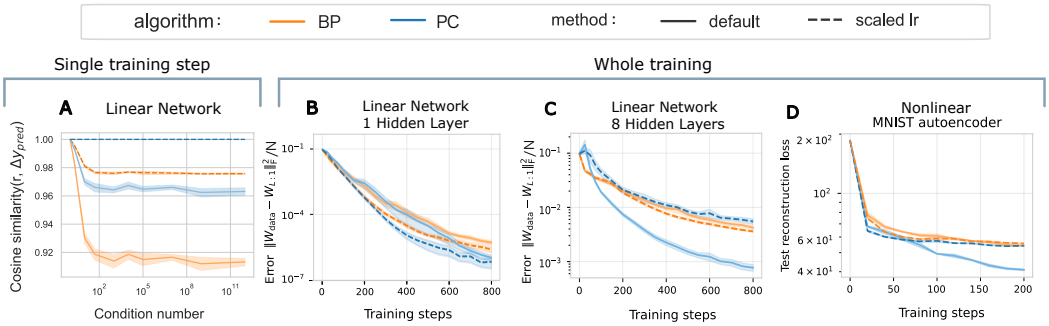


Figure 3: **Rescaling learning rate for online learning. (A)** Target Alignment as a function of the condition number of the weight matrix for four models, BP, PC and their rescaled counterparts. **(B)** Training dynamics of a square linear network with 20 units per layer. **(C)** Training of the same square network with 8 hidden layers. Curves are averaged over 10 runs. **(D)** Nonlinear autoencoder with 3 hidden layers and trained on online MNIST reconstruction task. Results averaged over 3 seeds.

in Appendix G by calculating the change in predictions of a DLN when training on a mini-batch of data for BP and PC.

Standard PC does not have a mechanism to reduce inter-sample interference. However, PC can maximise target alignment during multi-sample learning by multiplying its weight updates by a matrix that removes correlations between sample-specific updates.

Theorem 2 (PC with weight update rescaling)**.** *For the layer-specific weight update rescaling factors A_l equal to $(\frac{1}{B}\sum_{b=1}^{B}[\hat{x}_{l-1,b}\, x_{l-1,b}^{*\top}])^{+}$, for the pseudo inverse operation $.^{+}$ when the batch size B is smaller or equal than the smallest width of activity layers 0 to $L-1$, PC achieves optimal target alignment for batch learning. For PC's weight update*

$$\frac{\partial W_l}{\partial t} = -\frac{1}{B}\sum_{b=1}^{B} \nabla_{W_l} E\; A_l, \quad \textit{for } l = 1, \dots, L$$

the change in model prediction is given by:

$$\frac{\partial \hat{y}_b}{\partial t} = r_b, \quad \textit{for } b = 1, \dots, B$$

This theorem is proven in Appendix H. While the leaning rule in theorem 2 is not local, factor A_l is closely linked to the precision of neural activities, which can be computed by PC's neural circuits [3].

We repeat the target alignment experiments with a single weight update while varying the batch size. For BP, we employ a scaling matrix $A_l = (\frac{1}{B}\sum_{b=1}^{B}[\hat{x}_{l-1,b}\hat{x}_{l-1,b}^{\top}])^{-1}$. To ensure numerical stability, we employ Moore–Penrose pseudoinverses.

Introducing layer-wise scaling matrices enables PC to achieve optimal target alignment for batch training (Figure 4A). The target alignment of both PC and BP in their standard form decreases with increasing batch size. PC with weight update scaling achieves optimal target alignment independently of batch size. BP also exhibits improved alignment with scaling matrices, but still fails to reach optimal alignment and consistently underperforms PC.

For whole training experiments, we keep the protocol outlined in section 3.2 but change the batch size from 1 to 64 and increase the range used to find the optimal learning rate to $10^{-3.5} - 10^{0.4}$. In order to ensure numerical stability when inverting covariance matrices, we apply a small diagonal regularisation of the form $\tilde{A}_l = A_l + \varepsilon I$, $\varepsilon > 0$ is a scalar regularisation parameter. We select ε based on the spectrum of Σ such that $\varepsilon = \max\big(0, \alpha\lambda_{\max} - \lambda_{\min}\big)$. α is set to 10^{-5} for networks with 1 hidden layer and to 10^{-4} for networks with 8 hidden layers. In panels B and C of Figure 4, the default PC and BP implementations made fast initial progress but slowed down soon, scaled PC reached the minimum achievable error with much less training steps. Figure 4D suggests that the advantages seen in linear networks extend to nonlinear Autoencoders. Rescaled PC reached the minimum test error fastest, followed by default PC, rescaled BP and BP.

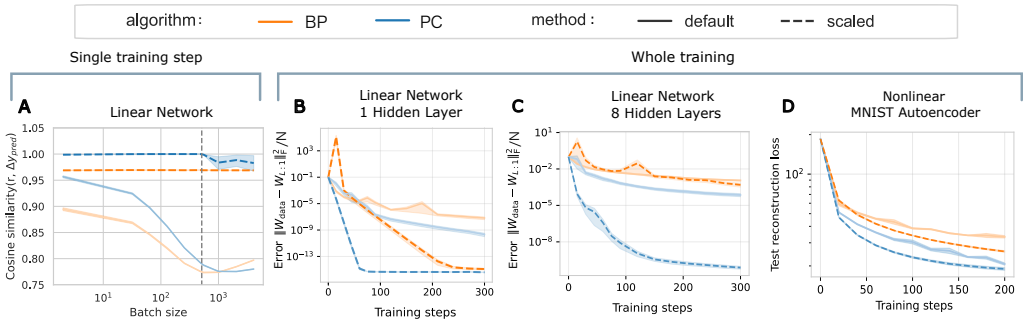


Figure 4: **Weight update rescaling for batch training (A)** Target Alignment as a function of the batch size for four models, BP (solid orange), PC (solid blue) and their rescaled counterparts (dashed). **(B)** Training dynamics of the models trained with a batch size of 64 for a 1 hidden layer square linear network with 20 units per layer, as well as its deeper counterpart with 8 hidden layers **(C)**. **(D)** Nonlinear autoencoder with 3 hidden layers and ReLU activations trained on MNIST reconstruction task with batch size 64. Runs averaged over 10 seeds for MLPs and 3 for Autoencoders.

3.4 Target alignment with a finite number of inference steps

The analysis so far has assumed equilibrated energy, though in practice equilibrium is often not reached due to a finite number of inference steps taken. Previous theorems can be extended to show that, if parameter updates are appropriately rescaled, perfect target alignment can be obtained regardless of the number of inference steps for online and batch learning.

Theorem 3 (PC rescaling with finite number of inference steps)**.** *Let T denote the number of inference steps and let $x^{(T)}$ be the activity of neurons after T inference steps. For the layer-specific weight update rescaling factors $A_l^{(T)}$ equal to $(\frac{1}{B}\sum_{b=1}^{B}[\hat{x}_{l-1,b}\hat{x}_{l-1,b}^{(T)\top}])^{+}$, for the pseudo inverse operation $.^{+}$ when the batch size B is smaller than or equal to the smallest width of layers 1 to $L-1$, PC achieves optimal target alignment for batch learning. For PC's weight update*

$$\frac{\partial W_l}{\partial t} = -\frac{1}{B}\sum_{b=1}^{B} \nabla_{W_l} E_b^{(T)}\, A_l^T, \quad \textit{for } l = 1, \dots, L-1$$

the change in model prediction is given by:

$$\frac{\partial \hat{y}_b}{\partial t} = r_b, \quad \textit{for } b = 1, \dots, B.$$

For online learning ($B = 1$), a scalar rescaling of $\alpha_l^{(T)} = \frac{1}{x_{l-1}^{(T)\top}\hat{x}_{l-1}}$ is sufficient.

Theorem 3 is proven in Appendix I. The form of the rescaling is very similar to those in theorems 1 and 2, but equilibrated activities x^* are replaced by $x^{(T)}$ (activities after T inference steps).

To further investigate the implications of theorem 3, we numerically simulate target alignment after one weight update (with and without rescaling) as a function of the number of inference steps in a toy model from Figure 1, a larger linear MLP, and a non-linear MLP. The training task for the toy model is identical to the setup in Figure 1, while the larger MLPs in Figures 5C-D are trained on the same synthetic dataset as before (batch size of 128) and have 512 neurons per layer with 1 hidden layer. We use a rate of 10^{-2} for parameter and activity updates. Figure 5A shows that target alignment does not necessarily improve with increasing number of inference steps in standard PC without rescaling. Because of the setup of the toy problem (see Figure 1A), small T means the shared parameters are minimally altered leading to less interference and therefore higher target alignment, however, this corresponds to a lazy learning regime where learning largely occurs in the final layer. Regardless of T, however, Figure 5B confirms that introducing layer-specific rescaling ensures optimal target alignment in PC as long as the assumptions of theorem 3 are satisfied. This result holds in larger linear MLPs (Figure 5C), and the benefits of rescaling may also translate to non-linear networks as

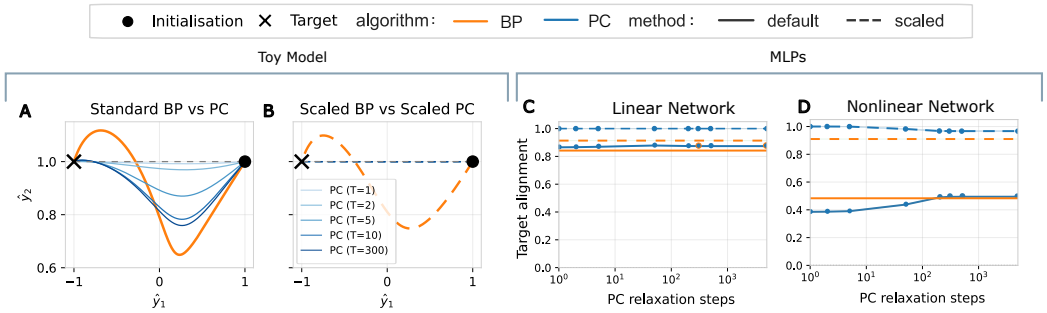


Figure 5: **Target alignment with and without rescaling as a function of inference steps (A)** Evolution of DLN's predictions $\hat{y}$ during training with BP and PC as a function of the number of inference steps T (legend displayed in panel B) for the toy example discussed in figure 1. Grey horizontal dashed line corresponds to the trajectory with perfect target alignment. **(B)** Evolution of predictions $\hat{y}$ with rescaled weight updates following theorem 3. All models with finite T still follow the optimal trajectory with appropriate rescaling. **(C)** Target alignment in a deeper linear network after one weight update with and without rescaling as a function of the number of inference steps. **(D)** MLP with ReLU nonlinearities with and without rescaling as a function of inference steps.

demonstrated in Figure 5D, where ReLU nonlinearities are introduced.

Takeaway: How can you maximise target alignment of PC?

Maximisation of target alignment in predictive coding can be achieved through:

- **Initialisation.** Use initialisation that preserves the activity norm across layers.
- **Weight update rescaling.** Normalise the weight updates by:
  - the dot product of pre and post inference activity for online learning, or
  - the covariance of pre and post inference activity for batch training.

4 Conclusion and Future directions

Inspired by early empirical reports of reduced destructive interference in PC relative to BP, we set out to provide a principled analytical account of when and why such advantages should arise. By deriving closed-form expressions for the changes in model predictions induced by BP and PC in DLNs, we show that the two learning rules differ fundamentally in how interference emerges during learning. In particular, BP updates are distorted by parameter-dependent interference arising from correlated and poorly conditioned weights, whereas PC attenuates interference, providing improved target alignment. We further show that, by introducing layer-dependent learning rates one can guarantee interference-free learning for single samples, independently of the number of inference steps, architecture or weight conditioning. Finally, extending the analysis to multi-sample learning reveals an additional source of cross-sample interference. In PC, this can be mitigated through activity-dependent decorrelation of weight updates, recovering interference-free learning trajectories.

4.1 Limitations and future directions

While our analysis provides useful theoretical and empirical insights, it has several limitations. First, the theory assumes linear networks, enabling tractability but not fully capturing the complexity of nonlinear models. Although preliminary experiments on nonlinear autoencoders suggest similar behavior, a rigorous extension to nonlinear settings remains open. Second, the analysis focuses on Multi-Layer Perceptrons. While the theory generalises to other linear models such as residual (Appendix J) and convolutional networks, we do not study these extensively. Extending target alignment analysis to broader architectures is left for future work. Finally, while weight or learning rate rescaling can accelerate PC, they introduce biologically implausible computations and are best viewed as optimal performance baselines. Biological systems may employ other strategies to maximise target alignment, such as activity normalisation [24]. Hence the presented theory can guide future investigation on how biological neural networks achieve their remarkable sample efficiency.

Acknowledgements

This work has been supported by the Medical Research Council UK grants MC_UU_00003/1, UKRI/MR/B000936/1, the Clarendon Fund, and by the Wellcome Trust grant 313955/Z/24/Z.

Code availability

Code for all simulations described in the paper are freely available at: `https://github.com/EleneLomi/TAFigures.git`.

References

[1] R. P. N. Rao and D. H. Ballard. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999.

[2] K. Friston. Learning and inference in the brain. *Neural Networks*, 16(9):1325–1352, 2003.

[3] R. Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76(Part B):198–211, 2017.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, Cambridge, MA, 2016.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] Y. Song, B. Millidge, T. Salvatori, T. Lukasiewicz, Z. Xu, and R. Bogacz. Inferring neural activity before plasticity as a foundation for learning beyond backpropagation. *Nature Neuroscience*, 27(2):348–358, 2024.

[8] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[9] A. Bernacchia, M. Lengyel, and G. Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[10] D. Huh. Curvature-corrected learning dynamics in deep neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[11] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

[12] J. Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

[13] A. Jnini and F. Vella. Dual natural gradient descent for scalable training of physics-informed neural networks, 2025.

[14] J. Müller and M. Zeinhofer. Achieving high accuracy with pinns via energy natural gradients, 2023.

[15] A. Saxe, J. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014.

[16] K. Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[17] Clementine Domine, Nicolas Anguita, Alexandra M Proca, Lukas Braun, Daniel Kunin, Pedro Mediano, and Andrew Saxe. From lazy to rich: Exact learning dynamics in deep linear networks. In *International Conference on Learning Representations*, volume 2025, pages 102485–102536, 2025.

[18] Beren Millidge, Yuhang Song, Tommaso Salvatori, Thomas Lukasiewicz, and Rafal Bogacz. A theoretical framework for inference and learning in predictive coding networks. In *The Eleventh International Conference on Learning Representations*, 2023.

[19] F. Innocenti, E. M. Achour, R. Singh, and C. L. Buckley. Only strict saddles in the energy landscape of predictive coding networks? In *Advances in Neural Information Processing Systems*, volume 37, pages 53649–53683, 2024.

[20] Ta-Chu Kao, Kristopher Jensen, Gido Van De Ven, Alberto Bernacchia, and Guillaume Hennequin. Natural continual learning: success is a journey, not (just) a destination. *Advances in neural information processing systems*, 34:28067–28079, 2021.

[21] G. Zhang, J. Martens, and R. B. Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[22] Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F. Grewe. A theoretical framework for target propagation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20024–20036. Curran Associates, Inc., 2020.

[23] Y. LeCun, C. Cortes, and C. J. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[24] Matteo Carandini and David J Heeger. Normalization as a canonical neural computation. *Nature reviews neuroscience*, 13(1):51–62, 2012.

[25] C. Goemaere, G. Oliviers, R. Bogacz, and T. Demeester. Error optimization: Overcoming exponential signal decay in deep predictive coding networks, 2025. arXiv preprint arXiv:2505.20137.

[26] Satoki Ishikawa, Rio Yokota, and Ryo Karakida. Local loss optimization in the infinite width: Stable parameterization of predictive coding networks and target propagation. In *International Conference on Learning Representations*, volume 2025, pages 49143–49182, 2025.

[27] C. Qi, M. Forasassi, T. Lukasiewicz, and T. Salvatori. Towards the training of deeper predictive coding neural networks, 2025. arXiv preprint arXiv:2506.23800.

Appendix

Table of Contents

A Experimental Configurations **14**
- A.1 Training data . . . 14
- A.2 One training step experiments . . . 14
- A.3 Training for multiple steps . . . 14

B Training with BP and PC **15**
- B.1 Training BP models . . . 15
- B.2 Training PC models . . . 15

C Mechanisms for improving alignment **16**
- C.1 Different initialisation schemes . . . 16
- C.2 Activity-dependent learning rates . . . 17
- C.3 Decorrelation matrix factors . . . 18

D Metrics **18**
- D.1 Target alignment . . . 18
- D.2 Condition number . . . 19

E Derivation of PC's weight updates **19**

F Proof of Theorem 1: target alignment for adaptive learning rates **20**

G Changes in prediction for batch training **21**

H Proof of Theorem 2: mitigating batch interference in PC **21**

I Proof of Theorem 3: target alignment without equilibrated energy **22**

J Extension to linear residual networks **24**

K Additional Experimental Data **26**
- K.1 1 Hidden Layer Network with Varying Width . . . 26
- K.2 Training a nonlinear Autoencoder . . . 27
- K.3 Square Networks with 8 Hidden Layers . . . 28

L Computational resources **28**

M Rescaled PC reproduces the natural-gradient parameter update **28**

N Extending the benefits of PC beyond an MSE loss **31**

O High variability of PC in deep models initialised with Kaiming **32**

P Isolating S^{-1}: a decorrelation ablation **32**

Q Nonlinear autoencoders on image data **33**

A Experimental Configurations

A.1 Training data

All experiments use synthetically generated data unless stated otherwise. Input vectors are sampled independently from a standard normal distribution, $x \sim \mathcal{N}(0, I)$, unless stated otherwise. Target outputs are generated via a fixed linear mapping,

$$y = W_{\text{data}} x,$$

where W_{data} is sampled once per experiment seed with entries drawn independently from a zero-mean normal distribution with variance $1/n_{\text{in}}$, where n_{in} refers to the number of input neurons. This scaling ensures that output magnitudes remain comparable across different input dimensionalities. To train nonlinear models, we use the standard MNIST dataset.

A.2 One training step experiments

In this section we focus solely on the experiments carried out for a single training step. All experiments use fully connected DLNs. The default architecture consists of a linear model with 512 input neurons, a single hidden layer of 512 neurons, and 512 output neurons. This model is initialised using Kaiming uniform initialisation. The models are trained either on a single sample (Figure 3) or on batches of 128 samples (Figures 2 and 4), depending on the experiment.

We consider two initialisations in our experiments, Kaiming Uniform and Norm preservation (see section C.1 for more details). Under Kaiming uniform, each weight matrix $W_l \in \mathbb{R}^{m \times n}$ is sampled as $W_l \sim U\Big(-\sqrt{\frac{1}{n}}, \sqrt{\frac{1}{n}}\Big)$. Under Norm-preserving normal initialisation, weights are sampled according to $W_l \sim \mathcal{N}\big(0, \frac{1}{m}\big)$. Orthogonal initialisation is performed by setting the condition number of one as described below.

Architectural variations are introduced by independently varying network depth and width. Network depth is defined as the number of hidden layers and is varied over values $1, \ldots, 8$. When varying depth, all hidden layers have equal width. Network width is varied over $128, 256, 512, 1024$ neurons, with input and output dimensions set to 512 neurons. The settings with 128 and 256 hidden neurons cause an information bottleneck. When width is varied, the model contains a single hidden layer.

To examine the effect of weight conditioning, we explicitly control the condition number of the weight matrices. Conditioning values are varied over $1, 2, 10, 50, 10^3, 10^4, 10^5, 10^7, 10^9, 10^{12}$. Conditioning is imposed by rescaling the singular values of initially Kaiming-initialised weight matrices while preserving their singular vectors, as described in section D.2.

Batch size is varied to study inter-sample interference effects. Batch sizes are varied across $1, 32, 64, 128, 256, 480, 550, 1000, 2048$. For experiments involving decorrelation matrices, batch sizes exceeding the smallest layer width are included to probe the limits of exact interference cancellation.

Across all experiments, parameters not explicitly varied are fixed to their default values. Results are averaged over multiple random seeds as specified in the corresponding figure captions.

A.3 Training for multiple steps

We conduct controlled experiments comparing backpropagation (BP) and predictive coding (PC), as well as their rescaled counterparts, on a synthetic supervised learning task where the target mapping is a known low-rank linear transformation as detailed in section A.1. For each run, we fix random seeds ranging from 0 to 10, pre-sample identical minibatches (set to either 1 for online learning or 64 for batch learning), and initialise all models with the same weights to ensure fair comparisons. We train multilayer perceptrons of varying depths and widths using stochastic gradient descent across a sweep of learning rates, and apply two different weight initialisation schemes, Kaiming Uniform and Norm Preservation. For each model, we choose the learning rate which results in the lowest error at the end of training to produce the plots of the training curves. The default architecture consists of a linear MLP with 20 input neurons, 20 output neurons, and a single hidden layer which we vary over $15, 20$ and 40 hidden neurons. We also consider a network with 8 hidden layers to study the

effect of depth on training. Performance is assessed by computing the mean squared error between the model's effective linear mapping - obtained by a product of its layer weights $W_{L:1}$ - and the ground-truth matrix W_{data}. This setup isolates differences in learning dynamics between BP and PC. All results are averaged over the 10 random seeds before plotting.

A nonlinear autoencoder is also trained on MNIST reconstruction tasks using BP, PC, and their rescaled variants. The default architecture is a fully connected MLP with layer widths $(784, 128, 32, 128, 784)$. We use standard LeCun initialisation, and nonlinearities are imposed by ReLU activations after each hidden layer, while the output layer applies a sigmoid function. The architecture compresses the input into a 32-dimensional bottleneck representation before reconstructing it through a mirrored decoder. For a fast implementation on digital hardware, we use ePC [25], a method developed to speed up the convergence of the inference phase in PC.

B Training with BP and PC

B.1 Training BP models

Linear models. DLNs trained with BP are optimised by minimising a mean squared error objective using stochastic gradient descent. For a mini-batch of B input-output pairs $\{(x_b, y_b)\}_{b=1}^{B}$, The continuous-time gradient descent dynamics for the weights of layer l are given by

$$\frac{\partial W_l}{\partial t} = -\frac{1}{B}\sum_{b=1}^{B} \nabla_{W_l} \mathcal{L}_{\text{BP}} = \frac{1}{B}\sum_{b=1}^{B} W_{L:l+1}^{\top} \left(y_b - W_{L:1} x_b\right) \left(W_{l-1:1} x_b\right)^{\top}, \tag{10}$$

where $W_{L:l+1}$ and $W_{l-1:1}$ denote the products of weight matrices above and below layer l, respectively, with the convention $W_{L:L+1} = W_{0:1} = I$.

In practice, these dynamics are discretised using an explicit Euler step, yielding the weight update

$$\Delta W_l = \alpha \frac{1}{B}\sum_{b=1}^{B} W_{L:l+1}^{\top} \left(y_b - W_{L:1} x_b\right) \left(W_{l-1:1} x_b\right)^{\top}, \tag{11}$$

where α denotes the learning rate. We use a fixed learning rate of $\alpha = 10^{-4}$ in all experiments for a single training step. This learning rule is commonly employed for training deep neural networks. Varying the learning rate does not affect the results.

For experiments illustrating learning trajectories in Figures 2, 3, and 4, learning rate sweeps are performed over intervals shown in section K. Learning rates which achieve the lowest error by the end of training are picked for each model separately.

Non-linear models. Training in deep non-linear networks follows the same procedure, but with a non-linear forward mapping. The sample-wise loss function is given by

$$\mathcal{L}_{\text{BP}}(x, y, W) = \frac{1}{2} \left\| y - f_W(x) \right\|_2^2, \tag{12}$$

where the network output is defined as

$$f_W(x) = W_L f(W_{L-1} f(\cdots f(W_1 x))), \tag{13}$$

and $f(\cdot)$ denotes the ReLU activation function applied elementwise. Gradients are computed via standard BP through the non-linear activations, and the resulting weight updates are applied using the same discrete-time update rule and learning rate as in the linear case.

B.2 Training PC models

Linear models. DLNs trained with PC minimise local prediction errors by first iteratively updating the neural activity until convergence, followed by a single weight update with stochastic gradient descent. In the experiments, the steady-state activity for each DLN layer x_l^* is computed using the expression given in equation 34. This is equivalent to iterative inference but is less computationally

intensive. For a mini-batch of B input-output pairs, the continuous-time gradient descent dynamics for the weight of layer l are given by

$$\frac{\partial W_l}{\partial t} = -\nabla_{W_l} E = \frac{1}{B} \sum_{b=1}^{B} W_{L:l+1}^{\top} \, S^{-1}(y_b - W_{L:1} x_b) \, x_{l-1,b}^{*\top}.$$

In the experiments, these dynamics are discretised using an explicit Euler step, yielding

$$\Delta W_l = \alpha \frac{1}{B} \sum_{b=1}^{B} W_{L:l+1}^{\top} \, S^{-1}(y_b - W_{L:1} x_b) \, x_{l-1,b}^{*\top}. \tag{14}$$

where α denotes the learning rate. We use a fixed learning rate of $\alpha = 10^{-4}$ in all experiments for one training step, unless stated otherwise. Varying the learning rate does not affect the results.

As for BP, for experiments illustrating learning trajectories with PC in Figures 2, 3, and 4, learning rate sweeps are performed over intervals shown in section K. Learning rates which achieve the lowest error by the end of training are picked for each model separately.

Non-linear models. For deep non-linear networks, PC's energy function is given by

$$E(x_0, ..., x_L, W) = \frac{1}{B} \sum_{b=1}^{B} \sum_{l=1}^{L} \frac{1}{2} ||x_{l,b} - f(W_l x_{l-1,b})||_2^2.$$

where $f(\cdot)$ is a ReLU activation function. The feedforward pass of this model is identical to that of the corresponding non-linear model trained with BP. The training procedure otherwise follows that used for PC in DLNs. However, inference is no longer analytically tractable in the non-linear setting. Consequently, the converged neural activities x_l^* are obtained via iterative inference. Inference is performed by minimising PC's energy function using gradient descent for $10,000$ steps, with an inference step of 0.05. We use a large number of inference steps to ensure that the neural activity reaches steady-state.

C Mechanisms for improving alignment

In order to maintain high target alignment, the norm of layer activities should remain approximately constant throughout training as mentioned in section 3.1.3. A loose constraint to encourage this is initialising networks with a Norm-preserving initialisation. We discuss this initialisation in section C.1 and show why the default Kaiming Uniform initialisation for linear networks violates activity norm preservation. In several experiments, we introduce tighter constraints where we modify the standard PC and BP weight updates via additional multiplicative factors. These factors are designed to reduce interference, thereby improve target alignment. Two classes of factors are considered: scalar, activity-dependent learning factors for single-sample training, and matrix-valued decorrelation factors for batch training. In sections C.2 and C.3, we give a detailed account of these.

C.1 Different initialisation schemes

In this section, we show the requirements to preserve the norm of the layers at initialisation and show why the default Kaiming initialisation in pytorch violates this.

C.1.1 Norm-preserving initialisation

The goal is to preserve the expected squared norm

$$\mathbb{E}\|Wx\|^2 = \mathbb{E}\|x\|^2, \tag{15}$$

where $x \in \mathbb{R}^n$, with $x_i \sim \mathcal{N}(0, 1)$ and $W \in \mathbb{R}^{m \times n}$, with $W_{ij} \sim \mathcal{N}(0, \sigma^2)$.

Each output component is $(Wx)_i = \sum_{j=1}^{n} W_{ij} x_j$. Since everything is zero-mean and independent, we obtain

$$\mathrm{Var}((Wx)_i) = \sum_{j=1}^{n} \mathrm{Var}(W_{ij}x_j) = \sum_{j=1}^{n} \sigma^2 = n\sigma^2. \tag{16}$$

This leads us to the following expressions:

$$\mathbb{E}\|Wx\|^2 = \sum_{i=1}^{m} \mathbb{E}[(Wx)_i^2] = m \cdot n\sigma^2, \tag{17}$$

$$\mathbb{E}\|x\|^2 = n. \tag{18}$$

Equating the expectations of both expressions, we obtain the final result:

$$\sigma = \frac{1}{\sqrt{m}}. \tag{19}$$

For a square network, since $m = n$ the variance of individual neurons is also preserved since, letting $z_j = \sum_{i=1}^{n} W_{ji}x_i$, we have that $\mathrm{Var}(z_j) = \mathrm{Var}(x_i)$ as long as $\sigma = \frac{1}{\sqrt{n}}$. Norm-preserving initialisation does not generally preserve variance at neuron level for arbitrary architectures, which may lead to instabilities during training for extreme bottlenecks or very wide hidden layers.

C.1.2 Kaiming Uniform initialisation and activity-norm decay

For default Kaiming initialisation, we again have $x \in \mathbb{R}^n$, with $x_i \sim \mathcal{N}(0, 1)$ and $W \in \mathbb{R}^{m\times n}$, with $W_{ij} \sim \mathcal{U}(-\sqrt{\frac{1}{n}}, \sqrt{\frac{1}{n}})$.

Then

$$\mathbb{E}[W_{ij}] = 0, \qquad \mathrm{Var}(W_{ij}) = \mathbb{E}[W_{ij}^2] = \frac{1}{3n}. \tag{20}$$

This leads us to the following:

$$\mathbb{E}\|Wx\|^2 = \mathbb{E}[x^\top W^\top Wx] = \mathbb{E}\big[x^\top \mathbb{E}[W^\top W]x\big] \tag{21}$$

$$= \frac{m}{3n}\,\mathbb{E}\|x\|^2 \tag{22}$$

because $\mathbb{E}[W^\top W] = \sum_{i=1}^{m} \mathbb{E}[w_i w_i^\top] = m\,\mathbb{E}[W_{ij}^2]\,I = m\frac{1}{3n}I$. w_i denotes the i-th row of W.

Therefore, overall we have that

$$\mathbb{E}\|Wx\|^2 = \frac{m}{3n}\,\mathbb{E}\|x\|^2. \tag{23}$$

Considering square networks where $m = n$, we can see that the layer norm of the activities decays in expectation throughout the forward pass by a factor of 3.

C.2 Activity-dependent learning rates

We first introduce layer-specific scalar factors that rescale the weight updates at each layer. For PC, the continuous-time weight update at layer l is modified as

$$\frac{\partial W_l}{\partial t} = -\alpha_l \nabla_{W_l} E. \tag{24}$$

The scaling factor α_l is chosen to cancel the activity-dependent scalar term appearing in PC's change in prediction and is defined as

$$\alpha_l = \frac{1}{x_{l-1}^{*\top}\hat{x}_{l-1}}, \tag{25}$$

where $\hat{x}_{l-1}$ denotes the feedforward activity at layer $l-1$ and x_{l-1}^* denotes the converged activity following PC inference.

For comparison, we also apply analogous activity-dependent scaling to BP. In this case, the scaling factor at layer l is given by

$$\alpha_l = \frac{1}{\hat{x}_{l-1}^\top \hat{x}_{l-1}}, \tag{26}$$

which removes the corresponding activity-dependent scalar factor from BP's prediction update.

Introducing these activity-dependent learning factors is equivalent to using layer-dependent learning rates. For instance, for networks trained with PC, the discretised weight updates given in equation 14 become

$$\Delta W_l = \alpha_l \alpha W_{L:l+1}^\top S^{-1}(y - W_{L:1}x)\, x_{l-1}^{*\top}.$$

Here, α denotes the global learning rate and α_l is a layer-specific scaling factor. Their product can therefore be interpreted as an effective learning rate for layer l.

In all experiments employing these scaling factors, the layer-specific coefficients α_l are recomputed for each sample and applied only for the duration of a single weight update. Apart from the inclusion of these factors, the training procedure is identical to that used without scaling. In particular, all experiments use the same global learning rate and the same number of inference steps.

C.3 Decorrelation matrix factors

We next introduce layer-specific matrix-valued factors that rescale the weight updates to remove inter-sample interference. These factors are designed to cancel the activity-dependent matrix terms that appear in the change in prediction when learning from multiple samples, thereby improving target alignment.

For PC, the continuous-time weight update at layer l is equal to

$$\frac{\partial W_l}{\partial t} = -\frac{1}{B}\sum_{b=1}^{B} \nabla_{W_l} E\, A_l, \tag{27}$$

where A_l is a decorrelation matrix applied to the right of the gradient.

The decorrelation matrix is chosen to cancel the batch-averaged activity correlation term appearing in PC's change in prediction. Specifically, we define

$$A_l = \left(E_B\left[\hat{x}_{l-1,b}\, x_{l-1,b}^{*\top}\right] \right)^+, \tag{28}$$

where $\hat{x}_{l-1,b}$ denotes the feedforward activity at layer $l-1$ for sample b, $x_{l-1,b}^*$ denotes the corresponding converged activity following PC inference. $E_B[\cdot]$ denotes the expectation over batch elements calculated with the empirical mean across the batch, and $.^+$ denotes the pseudo inverse.

For comparison, we also apply an analogous decorrelation factor to BP. In this case, the matrix factor is defined using feedforward activities only:

$$A_l = \left(E_B\left[\hat{x}_{l-1,b}\, \hat{x}_{l-1,b}^\top\right] \right)^+. \tag{29}$$

These matrix-valued factors are recomputed separately for each layer at every weight update. Matrix inversion is performed using the Moore–Penrose pseudoinverse to maintain numerical stability. The training procedure remains unchanged with identical training parameters to those used in the standard settings.

D Metrics

D.1 Target alignment

Target alignment quantifies how closely a learning update follows the direction of the output prediction error. Target alignment is defined as the cosine similarity between the output residual $r = y - \hat{y}$ and the change in the model's prediction induced by a weight update, $\Delta\hat{y}$ (see equation 5). A value of one indicates perfect alignment with the residual, zero indicates orthogonality, and negative values indicate updates that move the prediction away from the target. In experiments involving mini-batches, target alignment is computed separately for each sample and reported as the mean across samples. All reported results correspond to the alignment induced by a single parameter update.

D.2 Condition number

The conditioning of a model's weights is quantified using the condition number of each weight matrix. For a weight matrix W_l, the condition number is defined as

$$\kappa(W_l) = \frac{\sigma_{\max}(W_l)}{\sigma_{\min}(W_l)}, \tag{30}$$

where $\sigma_{\max}$ and $\sigma_{\min}$ denote the largest and smallest singular values, respectively. A condition number of one corresponds to perfectly conditioned (orthogonal) weights, while larger values indicate poorer conditioning and increased susceptibility to interference. When conditioning is varied experimentally, all layers are assigned the same condition number.

We set the conditioning by modifying the singular value spectrum of each weight matrix while preserving its singular vectors. For a given layer l, we compute the singular value decomposition

$$W_l = U_l \Sigma_l W_l^\top, \tag{31}$$

where $\Sigma_l = \mathrm{diag}(\sigma_1, \ldots, \sigma_k)$ contains the singular values in descending order. We then replace the original singular values by a linearly spaced spectrum

$$\sigma_i' = \sigma_{\max} \left(1 - \frac{i-1}{k-1}\left(1 - \frac{1}{\kappa}\right)\right), \tag{32}$$

where $\sigma_{\max}$ is the largest original singular value, k is the rank of W_l, and κ is the desired condition number. This construction ensures $\sigma'_{\max}/\sigma'_{\min} = \kappa$.

To isolate the effect of conditioning from changes in the overall weight scale, the modified singular values, Σ_l', are rescaled to preserve the Frobenius norm of the original weight matrix,

$$\Sigma_l' \leftarrow \Sigma_l' \frac{\|\Sigma_l\|_F}{\|\Sigma_l'\|_F}. \tag{33}$$

The conditioned weight matrix is then reconstructed as $W_l \leftarrow U_l \Sigma_l' W_l^\top$. This procedure alters only the conditioning of the weights while keeping their total energy and singular directions fixed.

E Derivation of PC's weight updates

PC updates its weights by following the negative gradient of its energy function with respect to the synaptic parameters. For the PC energy function

$$E(x_0, ..., x_L, W) = \sum_{l=1}^{L} \frac{1}{2} \|x_l - W_l x_{l-1}\|_2^2,$$

the weight dynamics are given by

$$\frac{\partial W_l}{\partial t} = -\nabla_{W_l} E = (x_l^* - W_l x_{l-1}^*) x_{l-1}^{*\top} = \epsilon_l^* x_{l-1}^{*\top},$$

where x_l^* denotes the equilibrium activity of layer l, and ϵ_l^* is the corresponding prediction error at equilibrium. The full expression for x_l^* is given below and derived in [26]:

$$x_l^* = \underbrace{W_{l:1} x}_{\text{BP feed forward activity}} + \underbrace{\left(W_{L:l+1}^\top + \sum_{i=2}^{l} W_{l:i} W_{L:i}^\top\right)\left(I + \sum_{i=2}^{L} W_{L:i} W_{L:i}^\top\right)^{-1} r}_{\text{propagated errors}}, \tag{34}$$

x_l^* can be seen as a sum of BP's feedforward pass activity values and the errors propagated backwards [18, 26]. At the activity equilibrium reached during inference, the gradient of the energy with respect to the neural activities vanishes. This yields, for $l = 2, \ldots, L-1$,

$$\nabla_{x_l} E = (x_l^* - W_l x_{l-1}^*) - W_{l+1}^\top (x_{l+1}^* - W_{l+1} x_l^*) = 0.$$

Rearranging this expression gives

$$(x_l^* - W_l x_{l-1}^*) = W_{l+1}^\top (x_{l+1}^* - W_{l+1} x_l^*),$$
$$\epsilon_l^* = W_{l+1}^\top \epsilon_{l+1}^*.$$

By recursively applying this relation, the prediction error at layer l can be expressed in terms of the output-layer error. Substituting into the weight update yields

$$\frac{\partial W_l}{\partial t} = \epsilon_l^* x_{l-1}^{*\top} = W_{L:l+1}^\top \epsilon_L^* x_{l-1}^{*\top}, \tag{35}$$

where $W_{L:l+1} = W_L W_{L-1} \cdots W_{l+1}$.

The equilibrium activity can be written equivalently as

$$x_l^* = W_l x_{l-1}^* + \epsilon_l^*.$$

Using this recursive form, the output activity can be expanded as

$$\begin{aligned} x_L^* &= W_L x_{L-1}^* + \epsilon_L^* \\ &= W_L (W_{L-1} x_{L-2}^* + \epsilon_{L-1}^*) + \epsilon_L^* \\ &= W_{L:1} x_0^* + W_{L:2} \epsilon_1^* + W_{L:3} \epsilon_2^* + \cdots + \epsilon_L^* \\ &= W_{L:1} x_0^* + \sum_{l=2}^{L} W_{L:l} \epsilon_{l-1}^* + \epsilon_L^*. \end{aligned}$$

Since the input and output layers are clamped to the input x and target y, respectively, and using the recursive expression for the prediction errors, the above expression becomes

$$x_L^* = W_{L:1} x_0^* + \sum_{l=2}^{L} W_{L:l} \epsilon_{l-1}^* + \epsilon_L^*,$$
$$y = W_{L:1} x + \sum_{l=2}^{L} W_{L:l} W_{L:l}^\top \epsilon_L^* + \epsilon_L^*,$$
$$y - W_{L:1} x = \Big(I + \sum_{l=2}^{L} W_{L:l} W_{L:l}^\top \Big) \epsilon_L^*.$$

Using the definitions of S and r introduced in the main text, this simplifies to

$$r = S\, \epsilon_L^*, \tag{36}$$
$$\epsilon_L^* = S^{-1} r. \tag{37}$$

Finally, combining equations 35 and 37, we obtain the expression used in the manuscript for the PC weight updates (equation 7):

$$\boxed{\frac{\partial W_l}{\partial t} = W_{L:l+1}^\top \epsilon_L^* x_{l-1}^{*\top} = W_{L:l+1}^\top S^{-1} r\, x_{l-1}^{*\top}.}$$

F Proof of Theorem 1: target alignment for adaptive learning rates

Let us remind that theorem 1 assumes that we modify PC's weight update to include a layer-specific scaling factor α_l:

$$\frac{\partial W_l}{\partial t} = -\alpha_l \nabla_{W_l} E \tag{38}$$

In practice, this is equivalent to assigning different learning rates for each layer. Choosing α_l equal to the inverse of PC's scalar factor $x_{l-1}^{*\top} \hat{x}_{l-1}$ removes the dependency of PC's change in prediction

on this scalar factor. This allows the expression for PC's change in prediction to be aligned with the residual r, leading to maximal target alignment as shown below:

$$\frac{\partial \hat{y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1} \frac{\partial W_l}{\partial t} W_{l-1:1} x = -\sum_{l=1}^{L} W_{L:l+1}\, \alpha_l \nabla_{W_l} E\, \hat{x}_{l-1} \tag{39}$$

$$= \sum_{l=1}^{L} W_{L:l+1} (1/x_{l-1}^{*\top} \hat{x}_{l-1}) W_{L:l+1}^{\top}\, S^{-1} r\, x_{l-1}^{*\top} \hat{x}_{l-1} \tag{40}$$

$$= \underbrace{\sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^{\top}\, S^{-1}}_{=I} \quad r \tag{41}$$

$$= r \tag{42}$$

G Changes in prediction for batch training

We re-derive the change in predictions of a DLN when training on a mini-batch of data. For batch training, weight updates are the average of the per-sample updates across the batch.

Backpropagation. For BP with a mini-batch of size B, the weight update for layer l is

$$\frac{\partial W_l}{\partial t} = -\frac{1}{B} \sum_{b=1}^{B} \nabla_{W_l} \mathcal{L}_{\mathrm{BP}} = \frac{1}{B} \sum_{b=1}^{B} W_{L:l+1}^{\top} r_b\, \hat{x}_{l-1,b}^{\top} = \frac{1}{B} W_{L:l+1}^{\top} R\, \hat{X}_{l-1}^{\top}. \tag{43}$$

where x_b and y_b denote the input and target output for the b-th data point in the batch, and r_b is the corresponding output prediction residual. For compactness, we define $R = [r_1, \ldots, r_B]$ as the matrix of residuals and $\hat{X}_{l-1} = [\hat{x}_{l-1,1}, \ldots, \hat{x}_{l-1,B}]$ as the matrix of feedforward activities at layer $l-1$.

The resulting change in the predictions across the batch is

$$\frac{\partial \hat{Y}}{\partial t} = \frac{1}{B} \sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^{\top} R\, \hat{X}_{l-1}^{\top} \hat{X}_{l-1}, \tag{44}$$

where $\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_B]$ collects the predictions for all batch elements.

Predictive coding. For PC trained on a mini-batch, the weight update for layer l is given by

$$\frac{\partial W_l}{\partial t} = -\frac{1}{B} \sum_{b=1}^{B} \nabla_{W_l} E = W_{L:l+1}^{\top} S^{-1} R\, X_{l-1}^{*\top}. \tag{45}$$

with $X_{l-1}^{*} = [x_{l-1,1}^{*}, \ldots, x_{l-1,B}^{*}]$.

The corresponding change in PC's predictions is

$$\frac{\partial \hat{Y}}{\partial t} = \frac{1}{B} \sum_{l=1}^{L} W_{L:l+1} W_{L:l+1}^{\top} S^{-1} R\, X_{l-1}^{*\top} \hat{X}_{l-1}. \tag{46}$$

H Proof of Theorem 2: mitigating batch interference in PC

Let us first remind the theorem. Define the layer-wise rescaling factor

$$A_l \;=\; \Big(\tfrac{1}{B} \textstyle\sum_{b=1}^{B} \hat{x}_{l-1,b}\, x_{l-1,b}^{*\top}\Big)^{+} \;=\; \big(\tfrac{1}{B}\, \hat{X}_{l-1} X_{l-1}^{*\top}\big)^{+}. \tag{47}$$

If the batch size satisfies $B \le d_l$ for every rescaled layer $l = 0, \ldots, L-1$ and the batch activities are in general position (i.e., $\hat{X}_{l-1}$ and X_{l-1}^{*} have full column rank B), then the weight update $\partial W_l / \partial t = -\frac{1}{B} \sum_b \nabla_{W_l} E\, A_l$ yields, for every sample in the batch, $\partial \hat{y}_b / \partial t \propto r_b$, i.e. optimal target alignment holds *exactly and simultaneously for all* b.

Proof. The rescaled change in prediction is

$$\frac{\partial \hat{Y}}{\partial t} = \frac{1}{B} \sum_l W_{L:l+1} W_{L:l+1}^\top S^{-1} R \left[X_{l-1}^{*\top} A_l \hat{X}_{l-1} \right]. \tag{48}$$

Exact per-sample alignment requires the inter-sample interference matrix to be a (scalar multiple of the) identity, i.e. the term in square brackets in the equation above needs to satisfy $X_{l-1}^{*\top} A_l \hat{X}_{l-1} = c\, I_B$, where I_B denotes an identity matrix of size B. Under $B \leq min_{l=0,\ldots,L-1} d_l$ and general position, $\hat{X}_{l-1}$ has full column rank and $X_{l-1}^{*\top}$ has full row rank, so the reverse-order pseudo-inverse law applies exact ly, $(\hat{X}_{l-1} X_{l-1}^{*\top})^+ = (X_{l-1}^{*\top})^+ (\hat{X}_{l-1})^+$, and the one-sided identities $X_{l-1}^{*\top} (X_{l-1}^{*\top})^+ = I_B$, $(\hat{X}_{l-1})^+ \hat{X}_{l-1} = I_B$ give

$$X_{l-1}^{*\top} A_l \hat{X}_{l-1} = B \big[X_{l-1}^{*\top} (X_{l-1}^{*\top})^+ \big] \big[(\hat{X}_{l-1})^+ \hat{X}_{l-1} \big] = B I_B. \tag{49}$$

Substituting equation 49 into equation 48 and using $\sum_l W_{L:l+1} W_{L:l+1}^\top = S$, we obtain:

$$\frac{\partial \hat{Y}}{\partial t} = \frac{1}{B} \left[S S^{-1} \right] R\,(B) = R. \tag{50}$$

This implies that for every training sample in the batch: $\partial \hat{y}_b / \partial t = r_b$. The global scalar $c = B$ cancels the $1/B$; in general it is absorbed into the learning rate, and target alignment (a cosine) is scale-invariant. □

Necessity and the $B >$ width regime. The condition $B \leq \min_l d_{l-1}$ is also necessary: $X_{l-1}^{*\top} A_l \hat{X}_{l-1}$ factors through $\mathbb{R}^{d_{l-1}}$, so $\mathrm{rank}(X_{l-1}^{*\top} A_l \hat{X}_{l-1}) \leq d_{l-1}$ for *any* A_l. If $B > d_{l-1}$ then $\mathrm{rank}(cI_B) = B > d_{l-1}$ is unattainable. This is the departure from alignment past the dashed line (network width) in Figure 4A.

Table 1 experimentally validates the use of pseudo-inverses and their conditions on the batch size.

Table 1: Deviation of $X^{*\top} A \hat{X}$ from a scalar multiple of the identity ($0 \Rightarrow$ optimal per-sample target alignment).

d	B	regime	code order $(\hat{X} X^{*\top})^+$	reversed order $(X^* \hat{X}^\top)^+$
64	16	$B \leq d$	3.2×10^{-15}	5.0×10^{-1}
128	32	$B \leq d$	2.5×10^{-15}	4.6×10^{-1}
64	63	$B \leq d$	8.3×10^{-14}	1.9×10^{1}
64	64	$B = d$	7.0×10^{-14}	3.8×10^{2}
64	80	$B > d$ (rank-lim.)	2.4	3.9×10^{1}
16	64	$B > d$ (rank-lim.)	2.1	2.4

I Proof of Theorem 3: target alignment without equilibrated energy

Proof. We first prove the theorem for batch size B, followed by a special case of online learning $B = 1$.

Batch learning. Let $\hat{x}_{0,b} = x_b$, and let $\hat{x}_{l,b} = W_l \hat{x}_{l-1,b}$ be the feedforward activities. Then the prediction is $\hat{y}_b = \hat{x}_{L,b}$.

After T inference steps, the prediction errors are

$$\epsilon_{l,b}^{(T)} = x_{l,b}^{(T)} - W_l x_{l-1,b}^{(T)}. \tag{51}$$

Since the input and target activities are clamped, $x_{0,b}^{(T)} = x_b$ and $x_{L,b}^{(T)} = y_b$.

Collect the feedforward and inferred activities and prediction errors into matrices

$$\hat{X}_{l-1} = [\hat{x}_{l-1,1}, \ldots, \hat{x}_{l-1,B}], \tag{52}$$

$$X_{l-1}^{(T)} = [x_{l-1,1}^{(T)}, \ldots, x_{l-1,B}^{(T)}], \tag{53}$$

$$\mathcal{E}_l^{(T)} = [\epsilon_{l,1}^{(T)}, \ldots, \epsilon_{l,B}^{(T)}]. \tag{54}$$

The partial gradient with respect to W_l is

$$\nabla_{W_l} E_b^{(T)} = -\epsilon_{l,b}^{(T)} x_{l-1,b}^{(T)\top}. \tag{55}$$

In the matrix notation introduced above, the rescaled weight update assumed in Theorem 3 can be written as:

$$\frac{\partial W_l}{\partial t} = \frac{1}{B} \mathcal{E}_l^{(T)} X_{l-1}^{(T)\top} A_l^{(T)}. \tag{56}$$

To simplify expressions, we write $W_L W_{L-1} \cdots W_{l+1} = W_{L:l+1}$ (note $W_{L:L+1}$ is set to identity).

For a linear network, differentiating the feedforward prediction with respect to learning time gives

$$\frac{\partial \hat{Y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1} \frac{\partial W_l}{\partial t} \hat{X}_{l-1}, \tag{57}$$

where $\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_B]$.

Substituting Eq. (56) into Eq. (57), we obtain

$$\frac{\partial \hat{Y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1} \mathcal{E}_l^{(T)} \left[\frac{1}{B} X_{l-1}^{(T)\top} A_l^{(T)} \hat{X}_{l-1} \right]. \tag{58}$$

We can make the expression in square brackets equal to the identity matrix I_B by choosing

$$A_l^{(T)} = B \left(X_{l-1}^{(T)\top} \right)^{+} \hat{X}_{l-1}^{+}, \tag{59}$$

where $(\cdot)^+$ denotes the Moore–Penrose pseudoinverse.

By assumption, both $X_{l-1}^{(T)}$ and $\hat{X}_{l-1}$ have full column rank B (we assume the batch size is less than or equal to the minimum layer width in the network $B \leq \min_l d_l$). Therefore,

$$X_{l-1}^{(T)\top} \left(X_{l-1}^{(T)\top} \right)^{+} = I_B \tag{60}$$

and

$$\hat{X}_{l-1}^{+} \hat{X}_{l-1} = I_B. \tag{61}$$

Also note that the prediction errors satisfy the telescoping identity

$$\sum_{l=1}^{L} W_{L:l+1} \epsilon_{l,b}^{(T)} = \sum_{l=1}^{L} W_{L:l+1} \left(x_{l,b}^{(T)} - W_l x_{l-1,b}^{(T)} \right),$$

expanding the brackets, we observe that all terms except initial and final cancel each other out, and we obtain:

$$\sum_{l=1}^{L} W_{L:l+1} \epsilon_{l,b}^{(T)} = y_b - W_L \cdots W_1 x_b = y_b - \hat{y}_b = r_b. \tag{62}$$

This identity holds independently of the number of inference steps T.

Substituting Eq. (59), we find

$$\frac{1}{B} X_{l-1}^{(T)\top} A_l^{(T)} \hat{X}_{l-1} = X_{l-1}^{(T)\top} \left(X_{l-1}^{(T)\top} \right)^{+} \hat{X}_{l-1}^{+} \hat{X}_{l-1} = I_B I_B = I_B. \tag{63}$$

Using Eq. (63) in Eq. (58) gives

$$\frac{\partial \hat{Y}}{\partial t} = \sum_{l=1}^{L} W_{L:l+1} \mathcal{E}_l^{(T)} = Y - \hat{Y}, \tag{64}$$

where the last equality follows from the telescoping identity in Eq. (62), applied to all B samples.

Thus, for every sample $b = 1, \ldots, B$,

$$\boxed{\frac{\partial \hat{y}_b}{\partial t} = y_b - \hat{y}_b = r_b.} \tag{65}$$

Hence, the instantaneous change in the feedforward prediction is perfectly aligned with its target residual. Since the telescoping identity holds for all $T \geq 0$, the result does not require inference to converge to equilibrium.

Online learning. For online learning, $B = 1$. In this case, consider the scalar rescaling

$$\alpha_l^{(T)} = \frac{1}{x_{l-1}^{(T)\top} \hat{x}_{l-1}}, \tag{66}$$

assuming $x_{l-1}^{(T)\top} \hat{x}_{l-1} \neq 0$.

The rescaled local weight update becomes

$$\frac{\partial W_l}{\partial t} = \alpha_l^{(T)} \epsilon_l^{(T)} x_{l-1}^{(T)\top}. \tag{67}$$

The induced change in the feedforward prediction is

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial t} &= \sum_{l=1}^{L} W_{L:l+1} \frac{\partial W_l}{\partial t} \hat{x}_{l-1} \\
&= \sum_{l=1}^{L} W_{L:l+1} \epsilon_l^{(T)} \frac{x_{l-1}^{(T)\top} \hat{x}_{l-1}}{x_{l-1}^{(T)\top} \hat{x}_{l-1}} \\
&= \sum_{l=1}^{L} W_{L:l+1} \epsilon_l^{(T)} \\
&= y - \hat{y} = r.
\end{aligned} \tag{68}$$

Therefore, online learning also satisfies

$$\boxed{\frac{\partial \hat{y}}{\partial t} = r} \tag{69}$$

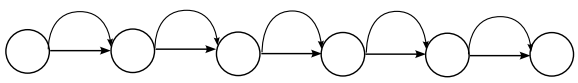
for any finite number of inference steps T, provided the rescaling denominators are nonzero.

Therefore, under our assumptions rescaled predictive coding achieves perfect instantaneous target alignment independently of inference convergence for batch and online learning. □

J Extension to linear residual networks

The analytical framework developed for DLNs extends directly to linear residual networks (ResNets). We briefly summarize these results, which demonstrate that the fundamental mechanisms of PC interference reduction remain unchanged under skip connections.

In linear ResNets, each layer composes as $x_l = x_{l-1} + W_l x_{l-1} = (I + W_l) x_{l-1}$, where weights accumulate as $\widetilde{W} = \prod_{k=1}^{l} (I + W_k)$ rather than as pure products $W_{l-1:1}$.



For clarity, we derive the ResNet expressions in the same order used above for DLNs: first the weight dynamics, then the induced change in prediction.

Backpropagation

For linear ResNets with $\hat{y} = \widetilde{W}_{L-1:1}\, x$ and residual $r = y - \hat{y}$, the continuous-time BP weight dynamics are

$$\frac{\partial W_l}{\partial t} = \widetilde{W}_{L-1:l+1}^{\top}\, r\, \hat{x}_{l-1}^{\top}, \tag{70}$$

where $\widetilde{W}_{L-1:l+1} = \prod_{k=l+1}^{L-1} (I + W_k)$ and $\hat{x}_{l-1} = \widetilde{W}_{l-1:1}\, x$.

The change in prediction then follows from product-rule differentiation of $\widetilde{W}_{L-1:1}$:

$$\frac{\partial \hat{y}}{\partial t} = \frac{\partial\, \widetilde{W}_{L-1:1}}{\partial t} x \tag{71}$$

$$= \sum_{l=1}^{L-1} \widetilde{W}_{L-1:l+1}\, \frac{\partial W_l}{\partial t}\, \widetilde{W}_{l-1:1}\, x \tag{72}$$

$$= \sum_{l=1}^{L-1} \widetilde{W}_{L-1:l+1} \widetilde{W}^{\top}_{L-1:l+1}\, r\, \hat{x}^{\top}_{l-1} \hat{x}_{l-1}. \tag{73}$$

Hence, as in DLNs, interference is introduced by the matrix factor $\widetilde{W}_{L-1:l+1}\widetilde{W}^{\top}_{L-1:l+1}$ acting on r.

Predictive coding

For PC in ResNets, we start from the ResNet prediction-error objective

$$E^{ResNet}(x_1, ..., x_L, W) = \sum_{l=1}^{L-1} \frac{1}{2} \left\| x_{l+1} - (I + W_l) x_l \right\|_2^2 . \tag{74}$$

Define layer-wise prediction errors

$$\epsilon_l := x_{l+1} - (I + W_l) x_l, \qquad l = 1, \ldots, L-1, \tag{75}$$

so that $E^{ResNet} = \frac{1}{2} \sum_{l=1}^{L-1} \|\epsilon_l\|_2^2$. At inference equilibrium (with x_1 and $x_L = y$ clamped), stationarity with respect to hidden activities gives the recursion

$$\epsilon_l = (I + W_{l+1})^{\top} \epsilon_{l+1}, \qquad l = 1, \ldots, L-2, \tag{76}$$

which implies

$$\epsilon_l = \widetilde{W}^{\top}_{L-1:l+1}\, \epsilon_{L-1}. \tag{77}$$

The output residual can be written as

$$r := y - \widetilde{W}_{L-1:1}\, x_1 = \epsilon_{L-1} + \sum_{l=1}^{L-2} \widetilde{W}_{L-1:l+1}\, \epsilon_l = \Big(I + \sum_{l=1}^{L-2} \widetilde{W}_{L-1:l+1} \widetilde{W}^{\top}_{L-1:l+1} \Big) \epsilon_{L-1} = \tilde{S}\, \epsilon_{L-1}, \tag{78}$$

with

$$\tilde{S}^{ResNet} = I + \sum_{l=1}^{L-2} \widetilde{W}_{L-1:l+1} \widetilde{W}^{\top}_{L-1:l+1}\, . \tag{79}$$

Hence $\epsilon_{L-1} = \tilde{S}^{-1} r$, and substituting back into

$$E^{ResNet} = \frac{1}{2} \sum_{l=1}^{L-1} \|\epsilon_l\|_2^2 = \frac{1}{2} \epsilon^{\top}_{L-1} \tilde{S}\, \epsilon_{L-1} \tag{80}$$

yields the equivalent output-space form

$$E^{ResNet}(x, y, W) = \frac{1}{2}\, r^{\top} \tilde{S}^{-1} r, \qquad r = y - \widetilde{W}_{L-1:1}\, x. \tag{81}$$

Using the chain rule,

$$\nabla_{W_l} E^{ResNet} = -\, \widetilde{W}^{\top}_{L-1:l+1}\, \tilde{S}^{-1} r\, x^{*\top}_{l-1}, \tag{82}$$

so the continuous-time gradient-flow dynamics are

$$\frac{\partial W_l}{\partial t} = \widetilde{W}^{\top}_{L-1:l+1}\, \tilde{S}^{-1} r\, x^{*\top}_{l-1}, \tag{83}$$

where $\tilde{S}^{ResNet}$ is defined above.

Applying the same product-rule step gives

$$\frac{\partial \hat{y}}{\partial t} = \sum_{l=1}^{L-1} \widetilde{W}_{L-1:l+1} \, \frac{\partial W_l}{\partial t} \, \widetilde{W}_{l-1:1} \, x \tag{84}$$

$$= \sum_{l=1}^{L-1} \widetilde{W}_{L-1:l+1} \widetilde{W}^{\top}_{L-1:l+1} \, \tilde{S}^{-1} r \, x^{*\top}_{l-1} \hat{x}_{l-1}. \tag{85}$$

This has the identical structure to the DLN expression in Eq. (8), with $W_{L-1:l+1}$ replaced by $\widetilde{W}_{L-1:l+1}$ and the corresponding preconditioner defined above (analogous to Eq. (4)). Consequently, the same cancellation argument applies: if $x^{*\top}_{l-1} \hat{x}_{l-1}$ is approximately constant across layers,

$$\sum_{l=1}^{L-1} \widetilde{W}_{L-1:l+1} \widetilde{W}^{\top}_{L-1:l+1} \, \tilde{S}^{-1} = I, \tag{86}$$

and the resulting change in prediction is aligned with r.

All key theoretical results therefore transfer directly to ResNets:

- BP interference in ResNets is characterized by accumulated $(I + W_k)$ products, with potentially better conditioning due to identity padding.
- PC attains improved target alignment compared to BP through the preconditioning mechanism $\tilde{S}^{-1}$.
- For PC, layer-specific learning rates $\alpha_l = 1/(x^{*\top}_{l-1} \hat{x}_{l-1})$ for online learning guarantee perfect alignment (TA $= 1.0$) in ResNets, independent of depth, width, or weight conditioning.
- Batch decorrelation matrices achieve perfect alignment for multi-sample learning in ResNets.

A key distinction emerges from ResNets' structure: the identity components in $(I + W_l)$ factors improve the conditioning of accumulated weights compared to pure weight products. Specifically, the singular values of $\widetilde{W} = \prod_k (I + W_k)$ remain larger than those of equivalent DLN products W_l, reducing distortion of the residual r by interference matrices. This suggests that ResNets enjoy a natural advantage over DLNs in baseline target alignment due to improved weight conditioning.

Figure 6 compares target alignment between BP and PC in ResNets versus DLNs across network depths. The comparison validates two key observations: (1) ResNets achieve slightly better baseline alignment than DLNs, particularly for BP, and (2) PC consistently outperforms BP in both architectures, with the advantage amplified as depth increases. These results indicate that PC's advantages extend to modern architectures with skip connections, and provide empirical support for the theoretical framework.

Figure 7 demonstrates that layer-specific learning rate scaling enables perfect alignment in ResNets. With rescaling α_l, PC achieves target alignment of 1.0 independently of network depth, confirming that the theoretical guarantee extends to skip-connected architectures.

K Additional Experimental Data

K.1 1 Hidden Layer Network with Varying Width

Figure 8 shows training curves and learning rate sweeps for varying widths $(15, 20, 40)$ and initialisations (Kaiming Uniform and Norm-Preservation) in a network with 20 input and output units and one hidden layer. Both PC and BP are trained for 500 steps with a batch size of 64, and the results are averaged over 10 seeds. The two middle figures on the left (within the "Training Curves for Varying Widths") panel are also used in Figure 2. The learning rate sweeps are extensive to ensure that the optimal point is reached for each model individually. PC can be seen to maintain a better performance across all widths considered.

In figure 9, we show the performance of all four models (BP, PC and rescaled versions of BP and PC), on online learning and batch learning synthetic regression tasks, as well as the corresponding

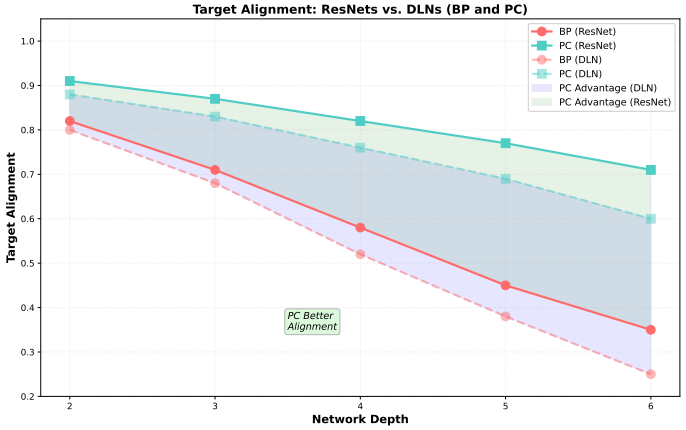


Figure 6: **Target alignment comparison: ResNets vs. DLNs.** Target alignment as a function of network depth for BP and PC in linear residual networks and deep linear networks. ResNets achieve slightly higher target alignment than equivalent DLNs due to improved weight conditioning from skip connections. PC outperforms BP in both architectures, with the advantage growing with network depth as BP interference accumulates.

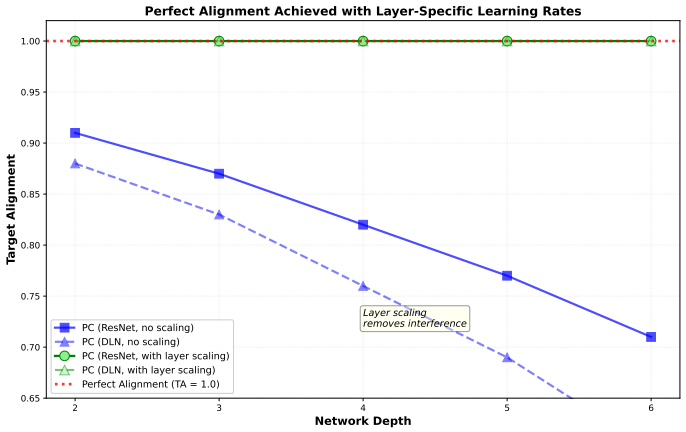


Figure 7: **Perfect alignment with layer-specific learning rates.** With layer-dependent scaling of weight updates according to activity norms, PC achieves near-perfect target alignment (TA ≈ 1.0) in both ResNets and DLNs. This demonstrates that interference-free learning is a general property that can be induced through appropriate learning rate scheduling, independent of network architecture or depth. The scaling factors depend only on local activities, making this approach computationally practical.

learning rate sweeps. Note that for batch learning experiments presented in this figure, we used spectral regularisation for rescaled models as described in section 3.3 to ensure numerical stability when inverting covariance matrices $(\frac{1}{B}\sum_{b=1}^{B}[x^*_{l-1,b}\hat{x}^\top_{l-1,b}])$ present in theorem 2. The middle row showing the plots for training trajectories of square networks are identical to the plots shown in Figures 3 and 4.

For online learning panel in figure 9, we can see the benefits of online natural gradients more clearly for a wider network (width$= 40$). In this scenario, PC and rescaled PC clearly outperform BP and rescaled BP.

K.2 Training a nonlinear Autoencoder

Training curves as well as the learning rate sweeps for a nonlinear Autoencoder described in section A.3 are shown in Figure 10. We use LeCun initialisation and the learning rates for all four models are chosen by sampling eleven points in the interval $10^{-4} - 10$. The behaviour observed in DLNs carries over in the nonlinear setting, with rescaling offering minimal benefits in the case of online learning at

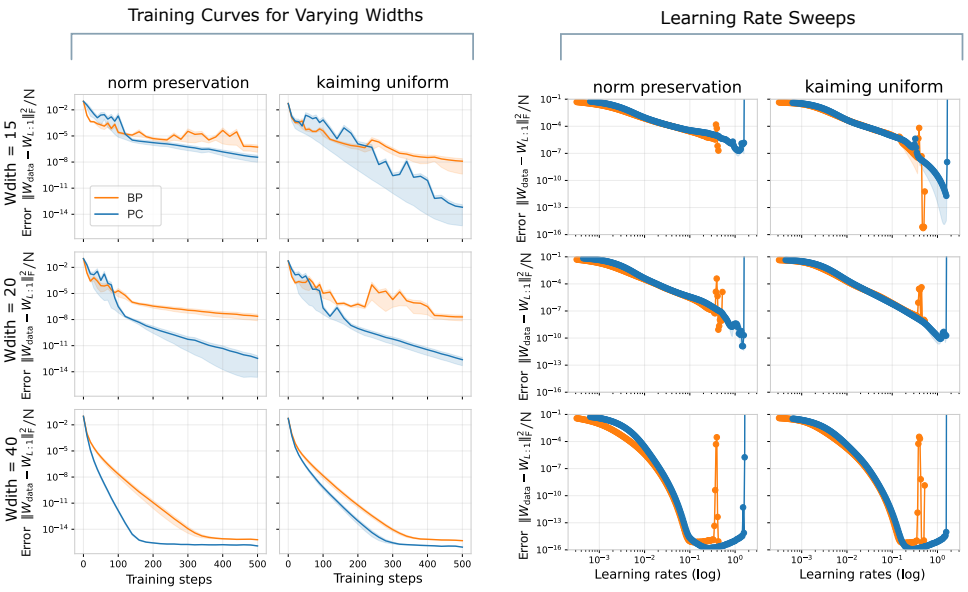


Figure 8: **Training curves and learning rate sweeps for** 1 **hidden layer DLNs of varying widths.** Training networks with a batch size of 64 shows that default PC consistently outperforms BP for DLNs with 1 hidden layer of varying widths (15, 20 and 40). Note that our results do not capture the behaviour for extremely narrow/wide networks. The learning rate sweeps are performed over the interval $10^{-3.5} - 10^{0.5}$ and we sample 100 points within it to ensure that sharp decreases in error for certain learning rates are not overlooked.

the beginning of training. The final error achieved by PC trained networks also remains lower than BP in the nonlinear case after 200 training steps. We average the results for our Autoencoders over 3 seeds.

K.3 Square Networks with 8 Hidden Layers

Figure 11 shows training curves as well as learning rate sweeps for Figures 3C and 4C. The networks in the figure again have 20 units per layer, though we now increase the number of hidden layers to 8 as opposed to 1 hidden layer networks discussed in section K.1. All simulations were repeated and averaged over 10 runs. Figure 11 demonstrates how the advantages of rescaling are diminished during online learning, but return as we increase the batch size to 64. All observations from studying learning trajectories in 1 hidden layer network also apply to 8 hidden layer networks.

L Computational resources

Simulations were conducted on a computing cluster consisting of P100, V100, A100, RTX, H100, L40s GPUs. The longest experiments (non-linear autoencoder tasks) ran in about 1 hour and 30 minutes.

M Rescaled PC reproduces the natural-gradient parameter update

Here we show that rescaled PC reproduces the natural gradient as a *parameter update*, in the deep-linear batch limit.

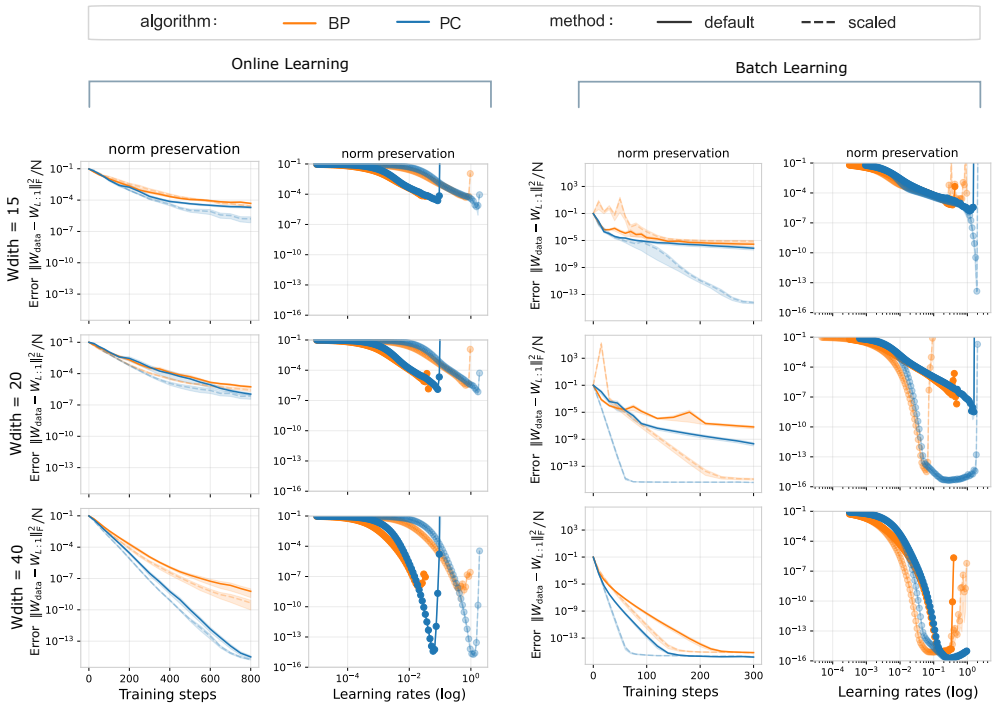


Figure 9: **Training curves and learning rate sweeps for scaled and default versions of PC and BP during online as well as batch learning.** For online learning, learning rate rescaling provides effectively no benefits compared to the default algorithms. For batch training, however, rescaled PC consistently outperforms all other models. Comparison between default PC and BP again shows that PC consistently performs better across varying widths.

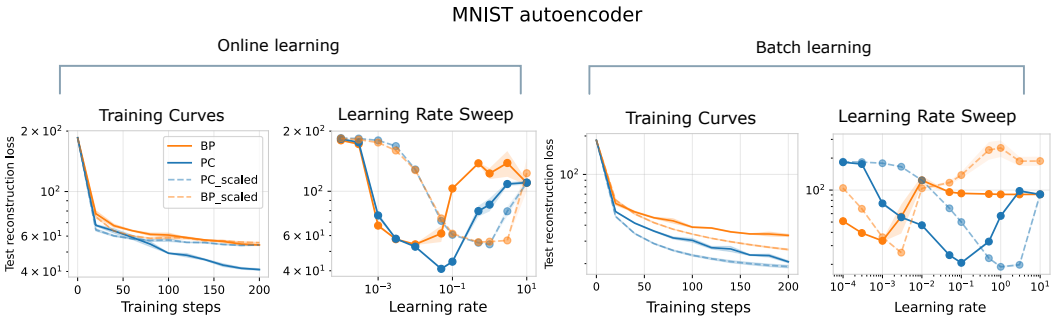


Figure 10: **Training trajectories and learning rate sweeps for a Nonlinear Autoencoder.** Comparison of training trajectories for BP, PC and their rescaled versions for online learning and batch learning. PC outperforms BP in both cases, though the rescaled algorithms only benefit from larger batch sizes. For larger batch sizes, rescaled PC weight updates approach natural gradients.

Setup. Take a linear teacher $y = \bar{V}x$ with input covariance $\Sigma = \mathbb{E}[xx^\top]$. Its residual is $r = \bar{V}x - W_{L:1}x = (\bar{V} - W_{L:1})x$. The rescaled-PC weight update for layer l (with A_l in its population form $A_l = \mathbb{E}[\hat{x}_{l-1}x_{l-1}^{*\top}]^{-1}$) is

$$\frac{\partial W_l}{\partial t} = \alpha_l \, W_{L:l+1}^\top S^{-1} r \, x_{l-1}^{*\top} \, A_l. \tag{87}$$

For the setup involving the linear teacher, the PC inference equilibrium (equation 34) can be written as a linear transformation of the input:

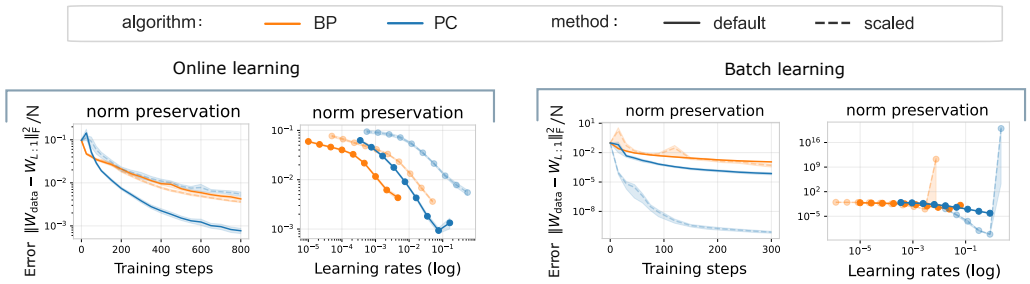


Figure 11: **Training trajectories and learning rate sweeps for networks with 8 hidden layers.** Comparison between default/scaled BP and PC for deep networks with 8 hidden layers. For these networks, the same trend emerges as for 1 hidden layer counterparts - benefits offered by rescaling can be observed mostly when we increase the batch size to 64. Learning rate sweeps are performed over the interval $10^{-5} - 10^{0}$, sampling 10 points within the region for each model.

$$x_l^* = W_{l:1}x + \Big(W_{L:l+1}^\top + \sum_{i=2}^{l} W_{l:i}W_{L:i}^\top\Big)\Big(I + \sum_{i=2}^{L} W_{L:i}W_{L:i}^\top\Big)^{-1}(y - W_{L:1}x) \tag{88}$$

$$= W_{l:1}x + \Big(W_{L:l+1}^\top + \sum_{i=2}^{l} W_{l:i}W_{L:i}^\top\Big)S^{-1}(\bar{V}x - W_{L:1}x) \tag{89}$$

$$= \Big(W_{l:1} + \Big(W_{L:l+1}^\top + \sum_{i=2}^{l} W_{l:i}W_{L:i}^\top\Big)S^{-1}(\bar{V} - W_{L:1})\Big)x \tag{90}$$

$$= M_l\, x \tag{91}$$

Cancellation. The two batch statistics are $\mathbb{E}[r\, x_{l-1}^{*\top}] = (\bar{V} - W_{L:1})\Sigma M_{l-1}^\top$ and $\mathbb{E}[\hat{x}_{l-1}x_{l-1}^{*\top}] = W_{l-1:1}\Sigma M_{l-1}^\top$, so the decorrelation collapses the layer-local geometry exactly:

$$\mathbb{E}[r\, x_{l-1}^{*\top}]\, A_l = (\bar{V} - W_{L:1})\Sigma M_{l-1}^\top\big(W_{l-1:1}\Sigma M_{l-1}^\top\big)^{-1} = (\bar{V} - W_{L:1})\, W_{l-1:1}^{-1}. \tag{92}$$

Telescoping the induced update of the product map, $\partial W_{L:1}/\partial t = \sum_l W_{L:l+1}(\partial W_l/\partial t)W_{l-1:1}$, with a common rate $\alpha_l = \alpha$ and $\sum_l W_{L:l+1}W_{L:l+1}^\top = S$:

$$\frac{\partial W_{L:1}}{\partial t} = \alpha\Big(\textstyle\sum_l W_{L:l+1}W_{L:l+1}^\top\Big)S^{-1}(\bar{V} - W_{L:1}) = \alpha\, SS^{-1}(\bar{V} - W_{L:1}) = \alpha\,(\bar{V} - W_{L:1}). \tag{93}$$

Since $\bar{V} = \mathbb{E}[yx^\top]\Sigma^{-1}$ is the least-squares map, $W_{L:1}$ obeys a first-order linear ODE and converges exponentially to the natural-gradient/least-squares solution (for MSE the loss decays exponentially to zero), provided the network is not initialised at a saddle.

Numerical confirmation. Reproducing the exact released update rule and comparing the induced total update on $W_{L:1}$ to the natural gradient $\bar{V} - W_{L:1}$ (square DLN, batch = width so X is invertible and Theorem 2 is exact; mean over 5 seeds):

Table 2: Scaled PC's total parameter update equals the natural gradient (NG) to machine precision; BP is essentially orthogonal to it.

depth	cos(scaled-PC, NG)	cos(BP, NG)	$\|dW_{\text{pc}} - \alpha\,\text{NG}\|/\|dW_{\text{pc}}\|$
2	1.00000	0.081	1.7×10^{-12}
4	1.00000	0.075	5.6×10^{-10}
8	1.00000	0.080	1.6×10^{-6}

N Extending the benefits of PC beyond an MSE loss

The analysis so far assumes a squared output error (SE). Under that loss the output residual $r = y - \hat{y}$ is exactly the negative gradient of the loss with respect to the prediction. Here we ask which of our results extend to other loss functions using softmax cross-entropy (CE) as the running example. In brief: the results for backpropagation carry over unchanged. The predictive coding (PC) results carry over exactly if the output error is held fixed, but not if inference is run on the loss itself.

Notation. For a general loss L we write the negative output gradient and the output curvature as

$$g := -\nabla_{\hat{y}} L, \qquad \text{SE: } g = y - \hat{y} = r, \qquad \text{CE: } g = y - p, \quad p = \mathrm{softmax}(\hat{y}),$$
$$H := \nabla^2_{\hat{y}} L, \qquad \text{SE: } H = I, \qquad \text{CE: } H = \mathrm{diag}(p) - pp^\top.$$

SE is the special case in which $g = r$ and $H = I$. Every difference below comes from $H \neq I$.

Backpropagation is unaffected. BP uses the output error only as the signal that it propagates backwards. A general loss therefore just replaces r by g:

$$\frac{\partial W_l}{\partial t} = W_{L:l+1}^\top \, g \, \hat{x}_{l-1}^\top,$$

which has the same structure as before.

PC with inference on the loss picks up the output curvature. Now suppose L is placed inside the PC energy and inference is run to equilibrium. The output logits move to $\hat{z} = \hat{y} + (S - I)g^*$. The top-layer error is no longer fixed. It is re-evaluated at the new logits, $g^* = -\nabla_{\hat{y}} L(\hat{z})$. Linearising the loss gradient about $\hat{y}$ and writing $M := S - I$ gives a self-consistency condition for g^*:

$$g^* = g - H(S - I)\, g^* \quad \Longrightarrow \quad g^* = S_H^{-1} g, \qquad S_H := I + H(S - I) = I + HM. \tag{94}$$

For SE we have $H = I$, so $S_H = S$ and $g^* = S^{-1} r$, which recovers the main result. The Theorem 1 rescaling then multiplies by S and cancels the interference exactly. For CE, however, $S_H \neq S$. The rescaled change in prediction becomes

$$\partial \hat{y} \;\propto\; S\, S_H^{-1} g = (I + M)(I + HM)^{-1} g \;\neq\; g.$$

Exact interference cancellation, *when derived through inference on the loss*, therefore depends on the loss being quadratic.

Fixing the output error removes the obstruction. The curvature H enters only because inference re-linearises the softmax at an equilibrium that keeps moving. Suppose instead that the CE error $g = y - p$ is computed once, at the feedforward pass, and then held fixed as a logit-space error. Equivalently, the logits are soft-clamped towards the target logit $\hat{y} + g$ through the squared top-layer term

$$\tfrac{1}{2} \big\| (\hat{y} + g) - W_L x_{L-1} \big\|^2.$$

This top term is quadratic with a fixed target, so the fixed point is linear exactly as in the SE case, with r replaced by g. Hence $g^* = S^{-1} g$, and

$$\partial \hat{y} \;\propto\; S\, S^{-1} g = g \qquad \text{(exact target alignment).} \tag{95}$$

The fixed error $g = y - p$ is the same output error that BP uses, and BP never re-evaluates its curvature either. Some PC classifiers are already configured this way [27].

Toy verification. We test these predictions in a deep linear network trained with CE over 8 seeds. We measure target alignment $\cos(\partial \hat{y}, g)$. No rescaling is applied except in the "scaled" rows, which use the Theorem 1 factor. As predicted, only the combination of a frozen error and rescaling is exact (Table 3). Rescaling under full CE inference applies $S\, S_H^{-1}$ instead of the identity and ends up below BP.

MNIST demonstration without rescaling. We next train deep linear networks with softmax CE on MNIST. The weights use a norm-maintaining Gaussian initialisation, $W \sim \mathcal{N}(0, 1/\text{fan_out})$. Training is online, one sample at a time, and no weight-update rescaling is applied. The three learners (BP, full-inference PC and frozen-error PC) start from identical weights. We report target alignment $\cos(\partial \hat{y}, g)$ with $g = y - \mathrm{softmax}(\hat{y})$, over 8 seeds (Table 4). In this setting freezing the output error alone gives near-perfect alignment. Full-inference PC falls below BP, as the softmax curvature predicts.

Table 3: Cross-entropy target alignment in a deep linear network (8 seeds). Only *frozen + scaled* is exact. Scaling without freezing applies the mismatched factor $S\,S_H^{-1}$ and falls below BP.

variant	$\cos(\partial\hat{y}, g)$
BP	0.830 ± 0.084
PC unscaled, full CE inference	0.819 ± 0.086
PC scaled, full CE inference (S_H)	0.741 ± 0.110
PC unscaled, frozen error	0.641 ± 0.177
PC scaled, frozen error	$\mathbf{1.000 \pm 0.000}$

Table 4: CE target alignment on MNIST without rescaling (8 seeds). Freezing the output error alone gives near-perfect alignment. Full-inference PC falls below BP because of the softmax curvature.

network	BP	PC full-inference	PC frozen
784→256→256→10	0.980 ± 0.008	0.952 ± 0.029	$\mathbf{1.000 \pm 0.001}$
784→64→64→64→10	0.915 ± 0.031	0.895 ± 0.028	0.90 (higher var.)

O High variability of PC in deep models initialised with Kaiming

The results of Figure 2A can be explained as follows. The main text compares PC and BP through the *pre*-multiplication terms (S^{-1} vs. $W_{L:l+1}W_{L:l+1}^\top$). The post-inference activity enters PC's change-in-prediction only through the scalar multiplier $s_l := x_{l-1}^{*\top}\hat{x}_{l-1}$. It is coupled to S^{-1}: the cancellation $\sum_l W_{L:l+1}W_{L:l+1}^\top S^{-1} = I$ factors out of the layer sum only when the s_l are approximately constant across layers.

From the equilibrium-activity expression (Appendix E), the scaling constant can be decomposed to $s_l = \|\hat{x}_{l-1}\|^2 + r^\top C_{l-1}^\top \hat{x}_{l-1}$.

Under Kaiming initialisation, the activity norm decays geometrically with depth, so in deep layers the magnitude of s_l varies significantly, and some s_l might flip sign. This breaks the factorisation assumptions presented in the paper for PC and PC's alignment loses mean and gains variance. Under a norm-preserving initialisation, the activity norm is stable, and the alignment stays high. Table 5 (all widths 512, 10 seeds) reports PC and BP target alignment together.

Table 5: Target alignment vs. depth and initialisation. The widening error bars in Fig. 2A are the Kaiming activity-decay regime: as $\|\hat{x}\|^2$ collapses, some post-inference scalars flip sign and PC's alignment variance blows up. Under norm-preserving init PC stays near 1 to depth 12.

init	depth	PC TA	BP TA	frac. layers $s_l < 0$	$\min\|\hat{x}\|^2$
norm-preserving	1	1.000 ± 0.001	0.912 ± 0.006	0.00	5.2×10^2
norm-preserving	8	0.990 ± 0.012	0.711 ± 0.036	0.00	4.5×10^2
norm-preserving	12	0.980 ± 0.022	0.675 ± 0.052	0.01	4.4×10^2
Kaiming	1	0.972 ± 0.006	0.896 ± 0.009	0.00	5.2×10^2
Kaiming	4	0.806 ± 0.027	0.672 ± 0.017	0.00	1.9×10^1
Kaiming	8	0.573 ± 0.120	0.528 ± 0.023	0.03	2.3×10^{-1}
Kaiming	12	0.161 ± 0.517	0.463 ± 0.029	0.05	2.9×10^{-3}

The results in Table 5 summarise the target alignment of PC and BP for Kaiming and norm-preserving initialisation for model depths 1 to 12. At Kaiming depth 12 the activity norm collapses to $\sim 3 \times 10^{-3}$, 5% of scalars flip sign, PC's mean alignment drops to 0.161 with standard deviation 0.517, and BP's mean (0.463) exceeds it. This is exactly the

P Isolating S^{-1}: a decorrelation ablation

To isolate the contribution of the pre-multiplication reconditioning S^{-1}, we replace PC's post-multiplication term (post-inference activity $x_{l-1}^{*\top}\hat{x}_{l-1}$) with BP's (feedforward activity $\hat{x}_{l-1}^\top\hat{x}_{l-1}$), giving three change-in-prediction variants that share the same post term:

$$\begin{aligned}
\text{BP:} \quad & \partial\hat{y} = \textstyle\sum_l W_{L:l+1}W_{L:l+1}^\top\, r\,(\hat{x}_{l-1}^\top\hat{x}_{l-1}),\\
\text{S-only:} \quad & \partial\hat{y} = \textstyle\sum_l W_{L:l+1}W_{L:l+1}^\top S^{-1} r\,(\hat{x}_{l-1}^\top\hat{x}_{l-1}),\\
\text{PC (full):} \quad & \partial\hat{y} = \textstyle\sum_l W_{L:l+1}W_{L:l+1}^\top S^{-1} r\,(x_{l-1}^{*\top}\hat{x}_{l-1}).
\end{aligned}$$

This is equivalent to using the feedforward activations for the weight update. This modification makes predictive coding non-local and forms an ablation.

Table 6 reports the target alignment in networks initialised with Kaiming and norm-preserving for BP, PC and the modified version of PC that uses feedforward activities for weight updates. This table shows that removing the post-inference activity from the weight update reduces PC performance deterioration under Kaiming initialisation. This shows the sustained benefit of iterative inference in introducing S^{-1} to the weight updates, but it also shows that post-inference activities cause additional variability in target alignment.

Table 6: Isolating S^{-1}. Under norm-preserving init, S^{-1} alone recovers essentially all of PC's advantage (S-only $\approx$ PC $\gg$ BP). Under Kaiming at depth, S-only is *more* stable than full PC, i.e. the post-inference activity scalar—not S^{-1}—is the source of the depth variance.

	norm-preserving			Kaiming		
depth	BP	S-only	PC	BP	S-only	PC
1	0.912	1.000	1.000	0.896	0.972	0.972
2	0.846	1.000	1.000	0.797	0.917	0.918
4	0.777	1.000	0.999	0.672	0.797	0.806
8	0.711	0.997	0.990	0.528	0.603	0.573
12	0.675	0.996	0.980	0.463	0.506	0.161

Q Nonlinear autoencoders on image data

To test whether the linear-theory predictions transfer to nonlinear networks on more complex data, we train bias-free MLP autoencoders with PC and BP on CIFAR-10, CIFAR-100, and STL-10. We use two depths ("shallow" $= 4$ weight layers, "deep" $= 8$) and three bottleneck widths $\{256, 64, 16\}$, orthogonal weight init, PC with Adam inference, and a per-configuration weight-learning-rate sweep (grid-searched on one seed, then re-run over 3 seeds; mean $\pm$ std). PC uses 100 inference steps (equilibrium); BP is standard backprop. We evaluate the reconstruction MSE of the models after 300 weight updates for CIFAR images and 250 updates for STL10.

Table 7 shows that PC beats BP across settings, and the advantage is consistently largest at depth: going deeper hurts BP far more than PC. On STL-10 the deep BP flat-lines at ~ 0.234 across all bottlenecks while PC keeps improving. These confirm that the linear-theory predictions, interference reduction, and depth as the sharp axis, transfer to nonlinear networks on real images.

Table 7: Final reconstruction MSE (mean $\pm$ std over 3 seeds; lower is better). Across all 18 configurations PC beats BP, the advantage is largest at depth, and error bars are tight.

depth	bottleneck	BP (MSE)	PC (MSE)	gain
CIFAR-10				
shallow	256	0.0886 ± 0.0036	0.0439 ± 0.0001	$2.02\times$
shallow	64	0.0944 ± 0.0018	0.0518 ± 0.0004	$1.82\times$
shallow	16	0.1114 ± 0.0090	0.0795 ± 0.0004	$1.40\times$
deep	256	0.1531 ± 0.0019	0.0591 ± 0.0001	$2.59\times$
deep	64	0.1655 ± 0.0064	0.0666 ± 0.0003	$2.48\times$
deep	16	0.1905 ± 0.0203	0.0939 ± 0.0020	$2.03\times$
CIFAR-100				
shallow	256	0.1014 ± 0.0039	0.0465 ± 0.0008	$2.18\times$
shallow	64	0.1107 ± 0.0088	0.0540 ± 0.0005	$2.05\times$
shallow	16	0.1195 ± 0.0048	0.0816 ± 0.0006	$1.47\times$
deep	256	0.1556 ± 0.0036	0.0619 ± 0.0007	$2.51\times$
deep	64	0.1707 ± 0.0028	0.0692 ± 0.0006	$2.47\times$
deep	16	0.2160 ± 0.0298	0.0965 ± 0.0019	$2.24\times$
STL-10 (96×96)				
shallow	256	0.1595 ± 0.0083	0.0855 ± 0.0004	$1.87\times$
shallow	64	0.1648 ± 0.0013	0.0918 ± 0.0006	$1.79\times$
shallow	16	0.1850 ± 0.0156	0.1154 ± 0.0010	$1.60\times$
deep	256	0.2342 ± 0.0001	0.0970 ± 0.0006	$2.41\times$
deep	64	0.2344 ± 0.0000	0.1042 ± 0.0013	$2.25\times$
deep	16	0.2371 ± 0.0039	0.1260 ± 0.0017	$1.88\times$

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Our claims are explicitly stated in both the abstract and the introduction. We provide a summary of our contributions at the end of the introduction. Each contribution is investigated in the results section.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We clearly state the limitations of our work throughout the discussion of our experiments. We also include a "Limitations" section at the end of the discussion.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: Each assumption is clearly stated and discussed.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We provide the full details to reproduce the experiments in the first sections of the supplementary materials. We also fully describe the equations governing the proposed model in the paper.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: The code to train the models and reproduce the experiments is provided in an anonymised zip file. We will also open-source the code upon publication.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: We provide the details of all experimental settings in the paper's experiments section and in the first sections of the supplementary materials.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: All the results include error bars to compare the significance of the results. We systematically report the standard error of the mean/standard deviation obtained over several seeds as indicated in the supplementary materials.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We discussed the compute resources used for our experiment in supplementary section K.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Our work followed the NeurIPS code of Ethics. Our work did not involve human subjects and the data used were from publicly available datasets.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: This work has no societal impact because it investigates models and datasets that are not appropriate for modern applications.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: This paper does not include data or models that have a high risk of misuse.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We credit the creators of all datasets and main code libraries used. We abide by the licensing terms of the datasets and code libraries we used.

    Guidelines:

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

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

13. **New assets**

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

    Answer: [Yes]

    Justification: The codebase of this paper includes documentation.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: This paper did not include crowdsourcing or research with human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: We did not conduct research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: Usage of LLMs was not an important, original, or non-standard component of the core methods in this research.

    Guidelines:

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