ACCELERATED PREDICTIVE CODING NETWORKS VIA DIRECT KOLEN–POLLACK FEEDBACK ALIGNMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Backpropagation (BP) is the cornerstone algorithm for training artificial neural networks, yet its reliance on update-locked global error propagation limits biological plausibility and hardware efficiency. Predictive coding (PC), originally proposed as a model of the visual cortex, relies on local updates that allow parallel learning across layers. However, practical implementations face two key limitations: error signals must still propagate from the output to early layers through multiple inference-phase steps, and feedback decays exponentially during this process, leading to vanishing updates in early layers. These issues restrict the efficiency and scalability of PC, undermining its theoretical advantage in parallelization over BP. We propose direct Kolen–Pollack predictive coding (DKP-PC), which simultaneously addresses both feedback delay and exponential decay, yielding a more efficient and scalable variant of PC while preserving update locality. Leveraging the direct feedback alignment and direct Kolen–Pollack algorithms, DKP-PC introduces learnable feedback connections from the output layer to all hidden layers, establishing a direct pathway for error transmission. This yields an algorithm that reduces the theoretical error propagation time complexity from $\mathcal{O}(L)$, with L being the network depth, to $\mathcal{O}(1)$, enabling parallel updates of the parameters. Moreover, empirical results demonstrate that DKP-PC achieves performance at least comparable to, and often exceeding, that of standard PC, while offering improved latency and computational performance. By enhancing both scalability and efficiency of PC, DKP-PC narrows the gap between biologically-plausible learning algorithms and BP, and unlocks the potential of local learning rules for hardware-efficient implementations.

1 INTRODUCTION

Major advances in artificial intelligence, from image recognition (LeCun et al., 2002; Krizhevsky et al., 2017; Alom et al., 2018) to image generation (Kingma & Welling, 2013; Parmar et al., 2018; Goodfellow et al., 2020) and natural language processing (Hochreiter & Schmidhuber, 1997; Vaswani et al., 2017; Beck et al., 2024), have all been enabled by backpropagation of error (BP), the fundamental algorithm underlying the training of artificial neural networks (ANNs) (Linnainmaa, 1970; Rumelhart et al., 1986; Werbos, 1988). However, several studies have put into question the plausibility of its direct implementation in biological neural systems (Grossberg, 1987; Lillicrap et al., 2016; Lillicrap & Santoro, 2019; Whittington & Bogacz, 2019; Ellenberger et al., 2024). Two primary concerns come from (i) the reliance on a global error signal that must be propagated backward and sequentially through the network hierarchy, thereby blocking parameter updates, and (ii) early layers depending directly on error signals generated by distant nodes. These biological plausibility issues of BP are commonly referred to as *update locking* and *non-locality* (Nøkland, 2016; Frenkel et al., 2021; Ororbia, 2023). They also lead to inefficiencies in hardware implementations, imposing memory and latency overheads (Mostafa et al., 2018; Frenkel et al., 2023).

Predictive coding (PC), originally proposed as a model of the visual cortex in the human brain (Rao & Ballard, 1999; Huang & Rao, 2011), is emerging as an alternative algorithm that may alleviate BP's limitations in update locking and non-locality (Millidge et al., 2022a; Salvatori et al., 2023). Its framework is grounded in Bayesian inference under the Free Energy Principle (Friston, 2005; Friston et al., 2006; Friston & Kiebel, 2009), providing a rigorous mathematical foundation with connections to information theory (Elias, 1955; 2003) and energy-based models (Millidge et al.,

2022b;c). Rather than minimizing a global error signal, PC minimizes the network's *variational free energy* (FE), defined as the sum of layer-wise squared errors between each layer's activity and its incoming prediction. Unlike BP, where weights are directly updated, PC learning has two phases. In the *inference phase*, neural activity is updated to minimize the FE, and in the *learning phase*, weights are updated based on the optimized neural activity. While this framework yields local and layer-wise update rules, the error in PC is still generated at the output and must propagate backward during inference. This error-delay limitation causes PC to be significantly slower than BP, and limits its efficiency and suitability for custom hardware implementations (Zahid et al., 2023). Moreover, the delayed error decays exponentially with depth, yielding vanishing updates in early layers (Pinchetti et al., 2024; Goemaere et al., 2025).

To address these limitations, we propose to propagate error information from the output layer to all hidden layers, yielding an instantaneous error term across the hierarchy. We thus build on feedback alignment methods (Lillicrap et al., 2014). Direct feedback alignment (DFA) (Nøkland, 2016) uses random direct feedback connections to deliver error signals from the output to all hidden layers, avoiding both error delay and decay. However, DFA scales poorly, especially in deep convolutional networks. Direct Kolen-Pollack (DKP) (Webster et al., 2020) improves DFA by learning the feedback matrices, incorporating learning rules inspired by the Kolen-Pollack (KP) algorithm (Kolen & Pollack, 1994; Akrout et al., 2019), thereby enhancing performance while preserving locality. Fig. 1 illustrates these frameworks and shows how our proposed direct KP predictive coding (DKP-PC) integrates primitives of both PC and DKP.

Our contributions are summarized as follows:

1. We extend the empirical analysis of Webster et al. (2020) by providing a mathematical motivation for why DKP achieves closer alignment with BP than standard DFA. This novel view further supports the integration of DKP within the PC framework as an efficient preliminary weight update to generate an instantaneous error term at every layer.
2. We introduce the DKP-PC algorithm, which simultaneously mitigates the feedback error delay and exponential decay limitations of BP while preserving locality. This, for the first time, enables full parallelization in PC networks regardless of batch size. We further discuss how our proposed PC variant has a time complexity of $\mathcal{O}(1)$, compared to $\mathcal{O}(L)$ for BP, with L being the network depth, to $\mathcal{O}(1)$.
3. We empirically demonstrate that DKP-PC performs on par with, or outperforms, both DKP and PC, benchmarking them across fully connected and convolutional networks of up to VGG-9 on CIFAR-100. We further assess DKP-PC's computational efficiency, showing that it consistently achieves lower training times compared to PC, accelerating training by over 200% for VGG-9.

2 Background

In this section, we review the core concepts of BP, DFA/DKP, and PC, which form the basis of our DKP-PC algorithm, from a mathematical perspective.

2.1 Backpropagation

BP enables recursive and efficient computation of parameter gradients by applying the chain rule of calculus to propagate error derivatives from the output layer back through the network (Linnainmaa, 1970; Rumelhart et al., 1986). Let us consider a neural network as shown in Fig. 1(A), where each layer $\ell \in \{0, \dots, L\}$ is associated with an activity vector $x_\ell \in \mathbb{R}^{d_\ell}$, where x_0 denotes the input and x_L the output, and d_ℓ is the number of neurons in layer ℓ. The forward pass is defined recursively as

$$z_\ell = \Theta_{\ell-1} x_{\ell-1}, \quad x_\ell = f(z_\ell), \quad 1 \leq \ell \leq L, \tag{1}$$

where $\Theta_{\ell-1} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the synaptic weight matrix mapping activity at layer $\ell - 1$ to layer ℓ, and $f : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ is typically an element-wise nonlinear activation function. The output error is then expressed in terms of the least-squared error (LSE)

$$\mathcal{L} = \frac{1}{2} \|x_L - y\|_2^2, \tag{2}$$

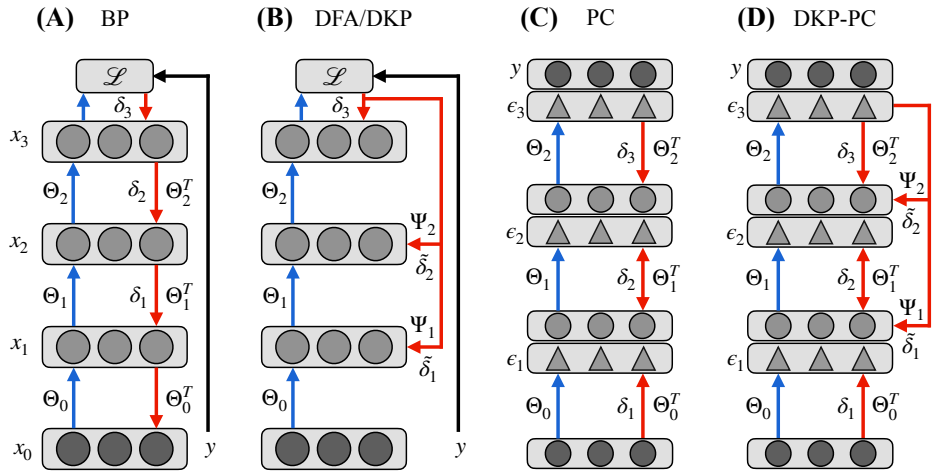

Figure 1: DKP-PC embeds DKP within the PC framework to address the error feedback delay and exponential decay issues of PC. Blue arrows represent forward connections, red arrows represent feedback connections. Neural activities are shown as gray circles, with clamped values in darker gray; x_0 denotes the input, y the target. $\mathcal{L}$ is the loss function, with δ_ℓ the BP error, $\tilde{\delta}_\ell$ its approximations, and ϵ_ℓ the PC error neurons, represented as triangles. (A) BP propagates the global error sequentially. (B) DFA and DKP propagate the error directly from the output to each layer. (C) PC minimizes local errors through an inference phase, followed by a learning phase that updates weights. (D) DKP-PC uses DKP's direct feedback to provide instantaneous error signals at all layers, accelerating the PC inference phase while preserving local weight updates.

where $y \in \mathbb{R}^{d_L}$ is the target vector. Applying the chain rule, the recursively backpropagated errors $\frac{\partial \mathcal{L}}{\partial z_\ell} = \delta_\ell \in \mathbb{R}^{d_\ell}$ are thus

$$\delta_\ell = \begin{cases} x_L - y & \text{if } \ell = L, \\ f'(z_\ell) \odot (\Theta_\ell^\top \delta_{\ell+1}) & \text{otherwise,} \end{cases} \tag{3}$$

where $\odot$ denotes the Hadamard product between the activation derivative $f'(z_\ell) \in \mathbb{R}^{d_\ell}$ and the error term[1]. The weights are then updated according to

$$\Delta\Theta_\ell = -\alpha \frac{\partial \mathcal{L}}{\partial \Theta_\ell} = -\alpha\big(\delta_{\ell+1} x_\ell^\top\big), \tag{4}$$

where $\alpha \in (0, 1)$ is the weight learning rate.

2.2 Direct Kolen-Pollack Feedback Alignment

DFA explicitly addresses the challenge of iteratively backpropagating error information in BP, yielding a local and more biologically plausible algorithm (Nøkland, 2016). To achieve this, DFA introduces random matrices $\Psi_\ell \in \mathbb{R}^{d_\ell \times d_L}$ that connect the output layer directly to each preceding layer in the network, as illustrated in Fig. 1(B). These matrices enable the direct propagation of the error signal $\delta_L \in \mathbb{R}^{d_L}$ generated at the output layer, avoiding the iterative backward propagation in (3), where

$$\tilde{\delta}_\ell = f'(z_\ell) \odot (\Psi_\ell \delta_L), \tag{5}$$

thereby leading to the following local weight update rule:

$$\Delta\tilde{\Theta}_\ell = -\alpha\big(\tilde{\delta}_{\ell+1} x_\ell^\top\big). \tag{6}$$

[1]Formally, $f'(z_\ell) = \frac{\partial x_\ell}{\partial z_\ell} \in \mathbb{R}^{d_\ell \times d_\ell}$ is the Jacobian matrix of the activation function. However, for element-wise activations, this Jacobian is diagonal with entries $f'(z_\ell)$, so the matrix-vector multiplication simplifies to a Hadamard product.

DKP (Webster et al., 2020) extends DFA by introducing a local learning rule for the feedback matrices Ψ_ℓ, inspired by the KP algorithm (Kolen & Pollack, 1994; Akrout et al., 2019). In contrast to DFA where the random matrices Ψ_ℓ are kept fixed, DKP updates them with

$$\Delta\Psi_\ell = -\alpha\big(x_{\ell+1}\delta_L^\top\big), \tag{7}$$

where the synaptic plasticity of Ψ_ℓ depends only on the connected hidden layer's activity and the output error signal. Note that this update depends only on local information on top of global error information that is shared across the network, and thus can be parallelized without update locking. In Appendix A.1, we extend the empirical work of Webster et al. (2020) by mathematically demonstrating that the feedback matrices converge to values approximating the recursive chain in (4), despite the dimensionality discrepancy between Ψ_ℓ and Θ_ℓ. This result offers a novel theoretical explanation on why DKP aligns more closely with BP than DFA.

2.3 Predictive Coding

PC models the brain as a Bayesian hierarchical generative model, in which latent variables represent the causes of sensory stimuli and are assumed to follow a Gaussian distribution (Rao & Ballard, 1999; Friston, 2005; Friston & Kiebel, 2009). Neural activities $x_\ell \in \mathbb{R}^{d_\ell} \sim \mathcal{N}(\mu_\ell, \Sigma_\ell)$ represent latent variables at layer ℓ, with mean $\mu_\ell \in \mathbb{R}^{d_\ell}$ and covariance matrix $\Sigma_\ell \in \mathbb{R}^{d_\ell \times d_\ell}$. As done by several works in literature, we assume the generative model's covariance to be the identity matrix $\Sigma_\ell = I$ (Pinchetti et al., 2022; Millidge et al., 2022a; Salvatori et al., 2024). The distribution's mean μ_ℓ is parametrized by the previous layer's state through the synaptic weights $\Theta_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ connecting them, according to the relation $\mu_\ell = f(\Theta_{\ell-1}x_{\ell-1})$, where $f : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ is a nonlinear mapping. The joint generative model over all $L+1$ latent-variables layers is

$$p(x_0, \dots, x_L; \Theta_0, \dots, \Theta_{L-1}) = \mathcal{N}\big(x_0; \mu_0, \Sigma_0\big) \prod_{\ell=1}^{L} \mathcal{N}\big(x_\ell; f(\Theta_{\ell-1}x_{\ell-1}), \Sigma_\ell\big), \tag{8}$$

where x_0 and x_L are clamped respectively to the input and target vectors, in classification settings. The exact posterior distribution inference $p(x_0, \dots, x_{L-1} \mid x_L)$ is generally intractable (Friston, 2005), so PC employs variational inference to approximate the latter with a tractable distribution $q(x_0, \dots, x_L)$, defined as

$$q(x_0, \dots, x_{L-1}) = \prod_{\ell=0}^{L-1} q(x_\ell), \tag{9}$$

where $q(x_\ell) \in \mathbb{R}^{d_\ell}$ is the variational distribution over the layer ℓ. Following PC literature for classification tasks, we model the variational posterior as a Dirac delta centered on parameter ϕ_ℓ:

$$q(x_\ell; \phi_\ell) = \delta(x_\ell - \phi_\ell), \tag{10}$$

where $\phi_\ell \in \mathbb{R}^{d_\ell}$ approximates x_ℓ, which corresponds to the mode of the true posterior (Bogacz, 2017; Millidge et al., 2021; Pinchetti et al., 2022; Salvatori et al., 2024). This formulation provides a deterministic approximation of the latent variables' mode. Variational inference reduces the problem of maximizing (8) to minimizing the Kullback–Leibler divergence between the variational and true posterior distributions $D_{KL}\left(q||p\right)$. Equivalently, this corresponds to minimizing the variational FE, which constitutes an upper bound on $D_{KL}(q \,\|\, p)$ (Friston & Kiebel, 2009). Under the assumptions of identity covariance matrices for the generative model and a Dirac delta variational posterior, the FE can be expressed as

$$F = \frac{1}{2} \sum_{\ell=1}^{L} \|\epsilon_\ell\|_2^2, \tag{11}$$

where the prediction errors $\epsilon_\ell \in \mathbb{R}^{d_\ell}$ at layer ℓ are defined as

$$\epsilon_\ell = \phi_\ell - f(\Theta_{\ell-1}\phi_{\ell-1}), \tag{12}$$

and are considered as dedicated units, represented by triangles in Fig. 1(C). In prediction tasks, ϕ_0 is not inferred, as the input layer is clamped to the input vector, and the network is initialized via a forward pass as in (1). After this initialization, the output layer ϕ_L is clamped to the target vector, so it is also not inferred. Minimizing (11) results in local updates for both neurons and weights, enabling layer-wise learning.

PC learning is divided into two phases: the inference and the learning phases. During the inference phase, (11) is optimized with respect to the variational parameters ϕ_ℓ updating the neural activities iteratively, whereas during the learning phase, the synaptic parameters Θ_ℓ are updated using the resulting neural configuration to minimize the same objective. This yields the local activity update

$$\Delta\phi_\ell = -\gamma\frac{\partial F}{\partial \phi_\ell} = \gamma\Big((f'(\Theta_\ell\phi_\ell)\odot\Theta_\ell)^\top\,\epsilon_{\ell+1} - \epsilon_\ell\Big), \tag{13}$$

where $\gamma \in (0,1)$ is the neural activity learning rate. The rule is local, as the activity of ϕ_ℓ is influenced only by the adjacent error nodes ϵ_ℓ and $\epsilon_{\ell+1}$. After the inference phase is completed, the synaptic connections are updated using the final neural activity configuration ϕ_ℓ^*, following

$$\Delta\Theta_\ell = -\alpha\frac{\partial F}{\partial \Theta_\ell} = \alpha\left(f'(\Theta_\ell\phi_\ell^*)\odot\,\epsilon_{\ell+1}\,\phi_\ell^{*\top}\right), \tag{14}$$

where $\alpha \in (0,1)$ is the weight learning rate. The weight update is local, as it depends only on the error neurons of the next layer $\epsilon_{\ell+1}$, and on the optimized neural activity of the current layer ϕ_ℓ^*.

3 METHODOLOGY

In this section, we first introduce the issues of error delay and error decay in PC, and subsequently demonstrate how the proposed DKP-PC algorithm provides a unified solution to both issues.

3.1 FEEDBACK ERROR DELAY AND DECAY

Let us consider a forward-initialized PC network with $L+1$ latent variables layers, whose neural dynamics evolve in discrete time steps $t \in \mathbb{N}_0$ according to (13), explicitly denoting the time dependence of neural activities $\phi_\ell(t)$ and prediction errors $\epsilon_\ell(t)$ during the inference phase.

Error propagation delay – At the initial time $t=0$, the neural activity of each layer corresponds to the prediction from the previous one, resulting in null error values at every layer in the network:

$$\phi_\ell(0) = f(\Theta_{\ell-1}\phi_{\ell-1}(0)) \quad\Longrightarrow\quad \epsilon_\ell(0) = \phi_\ell(0) - f(\Theta_{\ell-1}\phi_{\ell-1}(0)) = 0. \tag{15}$$

The network is thus at equilibrium, since the FE in (11) is minimized (Whittington & Bogacz, 2017). Assuming an incorrect prediction, clamping the target vector y to the output layer at $t=0$ induces a nonzero error at the final layer $\epsilon_L(0) \neq 0$, as shown in Fig. 2(A). Since each layer updates its activity based on its own and the subsequent layer's error (see (13)), the error propagates backward at most one layer per time step. Thus, the error takes $\hat{t} = L - \ell$ inference-phase steps to reach layer ℓ (Zahid et al., 2023) (see theorem and proof in Appendix A.2).

Error exponential decay – The error at optimization time $\hat{t} = L - \ell$, denoted $\epsilon_\ell(\hat{t})$, decays exponentially as it propagates backwards through the network, as shown in Fig. 2(A). The decay rate is determined by both the learning rate and the distance from the output layer (Goemaere et al., 2025), and the squared ℓ_2-norm $\|\epsilon_\ell(\hat{t})\|_2^2$ is upper-bounded by a quantity $\propto \gamma^{2(L-\ell)}$ (see theorem and proof in Appendix A.3).

Both issues originate from the fact that the error is generated only at the output layer, with the delay arising from its layer-by-layer propagation through the network and the exponential decay resulting from its progressive reduction by the learning rate at each iteration.

3.2 DIRECT KOLEN-POLLACK PREDICTIVE CODING

We propose to introduce learnable feedback connections from the output layer to each hidden layer $\Psi_\ell \in \mathbb{R}^{d_\ell \times d_L}$, $\forall\,\ell \in \{1,\cdots,L-1\}$, so as to enable a preliminary update of the forward weights (Fig. 1(D)). This approach, inspired by the DKP algorithm, not only induces approximate alignment with BP toward minimizing the output error (Appendix A.1), but also ensures that a nonzero error term is generated at every layer at the beginning of the inference phase, since the condition in (15) no longer holds (Fig. 2(B)).

The resulting algorithm, denoted as DKP-PC, not only solves the error propagation delay and exponential decay issues of PC, but also allows speeding up the inference phase of PC. Indeed, after the

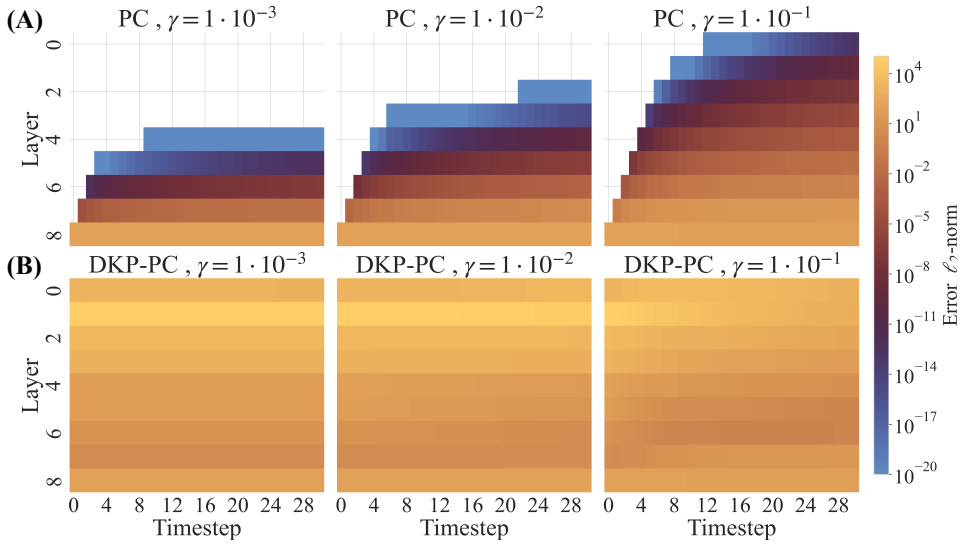

Figure 2: Error propagation in PC (A) and DKP-PC (B) during the inference phase of VGG-9 network trained on a CIFAR-10 dataset batch, at different neural activity learning rate γ. In (A), PC exhibits both an error decay problem, where the error magnitude decreases exponentially with network depth, and a delay problem, as the error signal flows through the network sequentially, undermining the theoretical parallelism. White colour represents values under numerical precision. In (B), DKP-PC mitigates both issues, generating a more uniform error signal to all layers at the start of neural activity optimization.

error is directly propagated, we empirically show that a single inference-phase step is sufficient to match or even surpass the performance of standard PC, which typically requires a number of steps at least equal to, and often exceeding, the network depth (Pinchetti et al., 2024). Therefore, we suggest that the inference phase acts as a correction on the preliminary update from DKP. We provide the pseudocode of DKP-PC in Algorithm 1, and formally outline the main steps below.

Direct feedback alignment update – After the forward initialization of the network, assuming a non-zero error at the output layer ϵ_L, we perturb the equilibrium by taking a first weight update according to (6), using PC's last layer error neurons:

$$\begin{aligned} \tilde{\Theta}_\ell &= \Theta_\ell + \Delta\tilde{\Theta}_\ell \\ &= \Theta_\ell - \alpha\big(f'(\Theta_\ell\phi_\ell) \odot (\Psi_{\ell+1}\epsilon_L)\phi_\ell^\top\big), \end{aligned} \tag{16}$$

which can be performed in parallel for each layer, as there is no recursive dependency.

Inference phase – After this update, an error term is generated at every layer since the first timestep:

$$\|\epsilon_\ell(0)\|_2^2 = \|\phi_\ell(0) - f(\tilde{\Theta}_{\ell-1}\phi_{\ell-1}(0))\|_2^2 > 0, \tag{17}$$

where $\phi_\ell(0) = f(\Theta_{\ell-1}\phi_{\ell-1}(0))$ after forward initialization. This provides a non-zero error term instantaneously in every layer-wise component of the FE (11). Consequently, every layer can independently update its neural activity according to (13), without performing null updates while waiting for the propagation of the error from the last layer. Thus, the single-step neural activity optimization performed takes the following form:

$$\Delta\phi_\ell = \gamma\Big((J_f(\tilde{\Theta}_\ell\phi_\ell)\,\tilde{\Theta}_\ell)^\top\,\epsilon_{\ell+1} - \epsilon_\ell\Big). \tag{18}$$

Learning phase and DKP update – After this single local update, both feedforward and feedback weight matrices are updated, which can be fully parallelized. Starting from the feedforward weight matrices Θ_ℓ, the update rule is unchanged from the standard PC update in (14), with the only difference of using the neural activity resulting from (18). Feedback weight matrices Ψ_ℓ now incorporate PC's optimized neural activity:

$$\Delta\Psi_\ell = -\alpha\big(\phi^*_{\ell+1}\epsilon_L^\top\big). \tag{19}$$

Algorithm 1 Direct Kolen-Pollack Predictive Coding (DKP-PC)

for each $(x, y) \in \mathcal{D}$ **do**
  0) *Forward initialization*
  $\phi_0 \leftarrow x$
  for $\ell = 1$ to $L-1$ **do** ▷ **Sequential**
    $\phi_\ell \leftarrow f(\Theta_{\ell-1}\phi_{\ell-1})$
  end for
  $\phi_L \leftarrow y$
  $\epsilon_L \leftarrow y - f(\Theta_{L-1}\phi_{L-1})$
  1) *Direct Feedback Alignment update*
  for $\ell = 0$ to $L-1$ **do** ▷ **Parallel**
    $\Theta_\ell \leftarrow \Theta_\ell - \alpha\big(f'(\Theta_\ell\phi_\ell) \odot (\Psi_{\ell+1}\epsilon_L)\phi_\ell^\top\big)$
  end for
  2) *Inference phase*
  for $t = 0$ to T **do** ▷ **T=1 for DKP-PC**
    for $\ell = 1$ to $L-1$ **do** ▷ **Parallel**
      $\epsilon_\ell \leftarrow \phi_\ell - f(\Theta_{\ell-1}\phi_{\ell-1})$
      $\phi_\ell \leftarrow \phi_\ell - \gamma\frac{\partial F}{\partial \phi_\ell}$
    end for
  end for
  3) *Learning phase*
  for $\ell = 0$ to $L-1$ **do** ▷ **Parallel**
    $\Theta_\ell \leftarrow \Theta_\ell - \alpha\frac{\partial F}{\partial \Theta_\ell}$
  end for
  4) *Direct Kolen-Pollack update*
  for $\ell = 1$ to $L-1$ **do** ▷ **Parallel**
    $\Psi_\ell \leftarrow \Psi_\ell - \alpha\,\phi_{\ell+1}\epsilon_L^\top$
  end for
end for

With DKP-PC, we introduce the first PC variant that is fully parallelizable. Indeed, each step of DKP-PC involves only local updates, and can thus be fully parallelized across layers. This reduces the backward time complexity of the network from $\mathcal{O}(L)$ to $\mathcal{O}(1)$, as it no longer depends on the network depth L. We note that DKP-PC goes beyond the incremental predictive coding (iPC) algorithm (Salvatori et al., 2024), which delivers more stable training by alternating between neural activity updates (13) and feedforward weight updates (14). While iPC partially unlocks PC's parallelization potential, it requires full-batch training to do so, whereas DKP-PC achieves this independently of the training batch size.

4 Results

In this section, we assess DKP-PC against BP, DKP, PC, and iPC, in terms of classification performance and training speed.

Setup – We evaluate the scalability of DKP-PC from multi-layer perceptrons (MLPs) to VGG-like convolutional neural networks (CNNs) (Simonyan & Zisserman, 2014). For the MLP experiments, a 4-layer architecture is evaluated on MNIST and Fashion-MNIST (Yann, 2010; Xiao et al., 2017). For the CNN experiments, we assess the performance of VGG-7 and VGG-9 on both CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009) datasets. For comparability with prior PC works and to facilitate future benchmarking, we employ the architectures reported by Pinchetti et al. (2024) in their discriminative mode experiments, and report their performance for PC, iPC and BP. Additional implementation details are reported in Appendix A.4. All implementations are based on the PyTorch framework and are available on GitHub.[2]

[2]Link omitted to preserve the double-blind review process. The GitHub repository will also contain all hyperparameters for reproducibility.

Table 1: Test accuracy in % (mean ± standard deviation) averaged over 5 random seeds. Results for PC, iPC and BP are taken from Pinchetti et al. (2024). The best results among local algorithms are highlighted in bold.

% Accuracy	DKP	PC	iPC	**DKP-PC**	BP
MLP					
MNIST	$97.78^{\pm 0.12}$	$98.26^{\pm 0.04}$	$\mathbf{98.45^{\pm 0.09}}$	$98.11^{\pm 0.08}$	$98.29^{\pm 0.08}$
FashionMNIST	$87.28^{\pm 0.14}$	$89.58^{\pm 0.13}$	$\mathbf{89.90^{\pm 0.06}}$	$89.40^{\pm 0.16}$	$89.48^{\pm 0.07}$
VGG-7					
CIFAR-10	$76.51^{\pm 0.33}$	$\mathbf{81.91^{\pm 0.30}}$	$80.15^{\pm 0.18}$	$81.64^{\pm 0.40}$	$89.91^{\pm 0.10}$
CIFAR-100 (Top-1)	$37.09^{\pm 0.89}$	$37.52^{\pm 2.60}$	$43.99^{\pm 0.30}$	$\mathbf{50.61^{\pm 0.26}}$	$65.36^{\pm 0.15}$
CIFAR-100 (Top-5)	$65.75^{\pm 0.68}$	$66.73^{\pm 2.37}$	$73.23^{\pm 0.30}$	$\mathbf{78.58^{\pm 0.23}}$	$84.41^{\pm 0.26}$
VGG-9					
CIFAR-10	$75.09^{\pm 0.43}$	$75.33^{\pm 0.25}$	$79.02^{\pm 0.21}$	$\mathbf{80.35^{\pm 0.65}}$	$90.02^{\pm 0.18}$
CIFAR-100 (Top-1)	$39.42^{\pm 0.44}$	$39.57^{\pm 0.18}$	$44.76^{\pm 0.40}$	$\mathbf{49.17^{\pm 0.79}}$	$65.51^{\pm 0.23}$
CIFAR-100 (Top-5)	$67.95^{\pm 0.42}$	$66.90^{\pm 0.26}$	$72.88^{\pm 0.29}$	$\mathbf{75.98^{\pm 0.31}}$	$84.70^{\pm 0.28}$

Classification performance – The classification results are summarized in Table 1. For the MLP architecture, all algorithms achieve comparable performance on both MNIST and FMNIST, with local algorithms even surpassing BP performance. For VGG-7 and VGG-9, DKP-PC outperforms all local-learning variants, achieving up to 9.6% higher top-1 accuracy for VGG-9 on CIFAR-100 compared to standard PC, and 4.4% higher top-1 accuracy than its more stable variant, iPC. Furthermore, DKP-PC outperforms vanilla DKP in every setup evaluated, marking a gap of 9.75% top-1 accuracy for VGG-9 on CIFAR-100. Interestingly, by combining the advantages of DKP and PC, DKP-PC delivers superior performance compared to DKP and PC alone, and reduces the gap with BP, especially in deep architectures.

Training speed – Table 2 presents the training times, in minutes, for BP, DKP, PC, iPC, and DKP-PC, which are necessary to produce the results reported in Table 1, with MLPs trained for 25 epochs and VGGs for 50 epochs. Importantly, DKP-PC requires only a single PC inference-phase step to achieve an accuracy equal or superior to iPC and PC, which require a number of steps of at least the network depth (Pinchetti et al., 2024), e.g., 4, 7, and 9 for the MLP, VGG-7, and VGG-9 models, respectively. All measurements were performed using PyTorch on an NVIDIA RTX A6000 GPU where the parallelization opportunities offered by DKP and DKP-PC have not been leveraged as they would require the use of custom CUDA kernels[3]. These models were thus executed sequentially, a setting in which BP naturally exploits highly-optimized hardware mapping and execution. Therefore, despite only highlighting speedup through reduced inference-phase steps and not through parallelization, the speedup achieved by DKP-PC is notable and these results already show substantial computational efficiency improvements: on VGG-9, DKP-PC respectively achieves $> 200\%$ and $> 400\%$ improvements in training speed compared to PC and iPC, respectively. We elaborate further on the computational trade-offs of DKP-PC in Appendix A.5.

5 Conclusion and Future Work

We introduced DKP-PC, the first training algorithm that releases PC's feedback error delay and exponential decay toward enabling fully parallelized, local learning. We evaluated its classification performance, training speed, and computational efficiency against BP, DKP, PC, and iPC. Our results show that, by accelerating PC with DKP, DKP-PC scales better than the evaluated local-learning algorithms, while exhibiting a substantial improvement in computational efficiency and training time compared to PC and its newer variant iPC. These results indicate that local learning rules can approach BP's efficiency and scalability, which is particularly relevant for neuromorphic computing

[3] A standard parallelization of DKP-PC in PyTorch introduces significant thread management and synchronization overhead, which cancels out the potential speedup.

Table 2: Training time in minutes (mean ± standard deviation), averaged over 5 runs. MLPs are trained for 25 epochs, while VGGs for 50 epochs.

Minutes	DKP	PC	iPC	**DKP-PC**	BP
MLP					
MNIST	$2.44^{\pm 0.05}$	$3.61^{\pm 0.07}$	$5.38^{\pm 0.11}$	$2.78^{\pm 0.09}$	$2.42^{\pm 0.06}$
FashionMNIST	$2.41^{\pm 0.03}$	$3.58^{\pm 0.10}$	$5.46^{\pm 0.08}$	$2.74^{\pm 0.05}$	$2.38^{\pm 0.07}$
VGG-7					
CIFAR-10	$7.37^{\pm 0.09}$	$28.52^{\pm 0.07}$	$49.51^{\pm 0.51}$	$10.94^{\pm 0.16}$	$7.38^{\pm 0.05}$
CIFAR-100	$7.17^{\pm 0.08}$	$28.61^{\pm 0.08}$	$49.47^{\pm 0.11}$	$11.30^{\pm 0.13}$	$7.22^{\pm 0.11}$
VGG-9					
CIFAR-10	$7.26^{\pm 0.10}$	$45.68^{\pm 0.05}$	$82.47^{\pm 0.24}$	$14.18^{\pm 0.08}$	$7.31^{\pm 0.22}$
CIFAR-100	$7.21^{\pm 0.18}$	$45.73^{\pm 0.09}$	$82.54^{\pm 0.04}$	$14.58^{\pm 0.08}$	$7.15^{\pm 0.08}$

and on-chip learning (Millidge et al., 2022a; Frenkel et al., 2023) as these algorithms also better align with biological plausibility. Future work should focus on custom CUDA kernels to address the thread management and synchronization overheads of the current PyTorch implementation. Indeed, despite already a significant speedup compared to PC and iPC, the training time of DKP-PC will still lag behind that of BP as long as parallelization opportunities are not fully exploited. Lastly, as feedback matrices introduce memory overhead, sparsity and quantization of feedback weights should be explored, as incentivized by prior work (Crafton et al., 2019; Han & Yoo, 2019).

6 Ethics Statement

We affirm that this work fully adheres to the ICLR Code of Ethics. This study does not involve human subjects and relies exclusively on well-known open-source datasets. The research presents no conflicts of interest and does not raise any privacy, security, or safety concerns. All experiments, results, and comparisons have been conducted and presented with scientific integrity, ensuring accuracy, transparency, and reproducibility. Care has been taken to ensure fairness in the evaluation and comparison of methods, avoiding bias in reporting or interpretation.

7 Reproducibility Statement

We are fully committed to ensuring the reproducibility of our results and research. We provide detailed mathematical derivations and pseudocode to support a thorough theoretical understanding of our method. Additionally, the appendix contains a comprehensive description of the experimental settings and technical details to enable fair and complete reproducibility. Upon acceptance, we will include a GitHub repository containing all code, experiments, and parameters necessary to reproduce all results reported in this manuscript. These measures ensure transparency, integrity, and reproducibility in accordance with the ICLR Code of Ethics.

References

Kingma DP Ba J Adam et al. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 1412(6), 2014.

Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. *Advances in neural information processing systems*, 32, 2019.

Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024.

Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of mathematical psychology*, 76:198–211, 2017.

Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience*, 13:525, 2019.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Peter Elias. Predictive coding–ii. *IRE Transactions on Information Theory*, 1(1):24–33, 1955.

Peter Elias. Predictive coding–i. *IRE Transactions on Information Theory*, 1(1):16–24, 2003.

Benjamin Ellenberger, Paul Haider, Jakob Jordan, Kevin Max, Ismael Jaras, Laura Kriener, Federico Benitez, and Mihai A Petrovici. Backpropagation through space, time, and the brain. *arXiv preprint arXiv:2403.16933*, 2024.

Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in neuroscience*, 15:629892, 2021.

Charlotte Frenkel, David Bol, and Giacomo Indiveri. Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence. *Proceedings of the IEEE*, 111(6):623–652, 2023.

Karl Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

Karl Friston and Stefan Kiebel. Predictive coding under the free-energy principle. *Philosophical transactions of the Royal Society B: Biological sciences*, 364(1521):1211–1221, 2009.

Karl Friston, James Kilner, and Lee Harrison. A free energy principle for the brain. *Journal of physiology-Paris*, 100(1-3):70–87, 2006.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Cédric Goemaere, Gaspard Oliviers, Rafal Bogacz, and Thomas Demeester. Error optimization: Overcoming exponential signal decay in deep predictive coding networks. *arXiv preprint arXiv:2505.20137*, 2025.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

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

Donghyeon Han and Hoi-jun Yoo. Efficient convolutional neural network training with direct feedback alignment. *arXiv preprint arXiv:1901.01986*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Yanping Huang and Rajesh PN Rao. Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593, 2011.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

John F Kolen and Jordan B Pollack. Backpropagation without weight transport. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pp. 1375–1380. IEEE, 1994.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1): 13276, 2016.

S Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (Doctoral dissertation, Master's Thesis*. PhD thesis, MA thesis. University of Helsinki, 1970.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Beren Millidge, Anil Seth, and Christopher L Buckley. Predictive coding: a theoretical and experimental review. *arXiv preprint arXiv:2107.12979*, 2021.

Beren Millidge, Tommaso Salvatori, Yuhang Song, Rafal Bogacz, and Thomas Lukasiewicz. Predictive coding: Towards a future of deep learning beyond backpropagation? *arXiv preprint arXiv:2202.09467*, 2022a.

Beren Millidge, Yuhang Song, Tommaso Salvatori, Thomas Lukasiewicz, and Rafal Bogacz. Backpropagation at the infinitesimal inference limit of energy-based models: Unifying predictive coding, equilibrium propagation, and contrastive hebbian learning. *arXiv preprint arXiv:2206.02629*, 2022b.

Beren Millidge, Yuhang Song, Tommaso Salvatori, Thomas Lukasiewicz, and Rafal Bogacz. A theoretical framework for inference and learning in predictive coding networks. *arXiv preprint arXiv:2207.12316*, 2022c.

Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608, 2018.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Alexander G Ororbia. Brain-inspired machine intelligence: A survey of neurobiologically-plausible credit assignment. *arXiv preprint arXiv:2312.09257*, 2023.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pp. 4055–4064. PMLR, 2018.

Luca Pinchetti, Tommaso Salvatori, Yordan Yordanov, Beren Millidge, Yuhang Song, and Thomas Lukasiewicz. Predictive coding beyond gaussian distributions. *arXiv preprint arXiv:2211.03481*, 2022.

Luca Pinchetti, Chang Qi, Oleh Lokshyn, Gaspard Olivers, Cornelius Emde, Mufeng Tang, Amine M'Charrak, Simon Frieder, Bayar Menzat, Rafal Bogacz, et al. Benchmarking predictive coding networks–made simple. *arXiv preprint arXiv:2407.01163*, 2024.

Rajesh PN Rao and Dana 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.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Tommaso Salvatori, Ankur Mali, Christopher L Buckley, Thomas Lukasiewicz, Rajesh PN Rao, Karl Friston, and Alexander Ororbia. Brain-inspired computational intelligence via predictive coding. *arXiv preprint arXiv:2308.07870*, 13, 2023.

Tommaso Salvatori, Yuhang Song, Yordan Yordanov, Beren Millidge, Zhenghua Xu, Lei Sha, Cornelius Emde, Rafal Bogacz, and Thomas Lukasiewicz. A stable, fast, and fully automatic learning algorithm for predictive coding networks. *arXiv preprint arXiv:2212.00720*, 2024.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

Matthew Bailey Webster, Jonghyun Choi, et al. Learning the connections in direct feedback alignment. 2020.

Werbos. Backpropagation: past and future. In *IEEE 1988 International Conference on Neural Networks*, pp. 343–353 vol.1, 1988. doi: 10.1109/ICNN.1988.23866.

James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5): 1229–1262, 2017.

James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 23(3):235–250, 2019.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

LeCun Yann. Mnist handwritten digit database. *ATT Labs.*, 2010.

Umais Zahid, Qinghai Guo, and Zafeirios Fountas. Predictive coding as a neuromorphic alternative to backpropagation: a critical evaluation. *Neural Computation*, 35(12):1881–1909, 2023.

A Appendix

A.1 Convergence of Feedback Matrices under the Direct Kolen–Pollack Algorithm

In this appendix, we extend the empirical observations of Webster et al. (2020) by providing a mathematical argument for why DKP achieves a better alignment with BP than DFA. While their work demonstrates this empirically, it does not provide a formal theoretical justification, instead attributing the behavior to analogies with KP. However, while it has been proven for KP that the feedback matrices Ψ_ℓ converge to $\Theta_\ell^\top$ for all layers (Kolen & Pollack, 1994; Akrout et al., 2019), in DKP this result strictly holds only for the last layer, as all other hidden layers have a dimensionality mismatch between Ψ_ℓ and Θ_ℓ. Here, we offer a novel theoretical perspective on the work of Webster

et al. (2020), demonstrating that DKP drives the feedback matrices toward values approximating the chain of transposed forward weights, similar to BP, up to the Moore-Penrose pseudoinverse.

Let us consider a feedforward neural network with L layers, where d_ℓ denotes the number of neurons at layer ℓ. The weight matrix $\Theta_\ell \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ maps activations from layer $x_\ell \in \mathbb{R}^{d_\ell}$ to the next layer $x_{\ell+1} \in \mathbb{R}^{d_{\ell+1}}$, according to

$$x_{\ell+1} = f(\Theta_\ell x_\ell)\,, \tag{20}$$

where $f(\cdot)$ denotes an arbitrary nonlinear activation function. In the following derivations, we assume an identity activation function, so that the derivative of the activation function can be omitted.

In DFA, error feedback is provided by fixed random matrices $\Psi_\ell \in \mathbb{R}^{d_\ell \times d_L}$ that project the output error term δ_L directly to each hidden layer ℓ. These matrices replace the layer-specific error term δ_ℓ used in BP, computed by transporting the error backward from the next layer through the transposed weight matrix $\Theta_{\ell-1}^\top$, as described by

$$\delta_\ell = \Theta_{\ell+1}^\top \delta_{\ell+1}. \tag{21}$$

In DFA, as illustrated in Fig. 1, this layer-specific error term δ_ℓ is instead approximated by

$$\tilde{\delta_\ell} = \Psi_\ell \delta_L. \tag{22}$$

In contrast, DKP allows the feedback matrices to be updated following the local update rule

$$\Delta\Psi_\ell = x_{\ell+1}\, \delta_L^\top. \tag{23}$$

With both forward and feedback weights subject to a decay term, this learning has been empirically shown to enable Ψ_ℓ to provide a more BP-aligned update for Θ_ℓ compared to standard DFA (Webster et al., 2020). We extend their work by demonstrating that the feedback matrices gradually align with the pseudoinverses of the forward weight matrices in a recursive dependency, thereby yielding a closer approximation of BP error propagation compared to DFA.

Starting from the last layer, the update rule for the weight matrix preceding it, denoted as $\Theta_{L-1} \in \mathbb{R}^{d_L \times d_{L-1}}$, is the same for BP, DFA, and DKP, and is given by

$$\Delta\Theta_{L-1} = -\alpha \left(\delta_L x_{L-1}^\top + \Theta_{L-1}\right), \tag{24}$$

where $\alpha \in (0, 1)$ is the learning rate, and the second term of the update is the weight decay term. According to (23) and assuming the same learning rate for the feedback weights, the update rule for the feedback matrix connecting the last layer to the penultimate one is given by

$$\Delta\Psi_{L-1} = -\alpha \left(x_{L-1}\delta_L^\top + \Psi_{L-1}\right). \tag{25}$$

In this specific case, Θ_{L-1} has the shape of $\Psi_{L-1}^\top$ and their updates are transposes of each other. As training progresses, both matrices converge to the same value, since the contribution of the initial condition vanishes under the effect of the learning rate α (Kolen & Pollack, 1994; Akrout et al., 2019). Indeed, following KP, by defining

$$\Omega_{L-1}(t+1) = \Theta_{L-1}(t+1) - \Psi_{L-1}^\top(t+1), \tag{26}$$

and using the update rules (24) and (25), we obtain

$$\begin{aligned}
\Omega_{L-1}(t+1) &= \left(\Theta_{L-1}(t) + \Delta\Theta_{L-1}(t)\right) - \left(\Psi_{L-1}^\top(t) + \Delta\Psi_{L-1}^\top(t)\right) \\
&= \Theta_{L-1}(t) - \Psi_{L-1}^\top(t) - \alpha\left(\delta_L(t)x_{L-1}^\top(t) + \Theta_{L-1}(t) - \delta_L(t)x_{L-1}^\top(t) - \Psi_{L-1}^\top(t)\right) \\
&= \Theta_{L-1}(t) - \Psi_{L-1}^\top(t) - \alpha\left(\Theta_{L-1}(t) - \Psi_{L-1}^\top(t)\right) \\
&= \left(1-\alpha\right)\left(\Theta_{L-1}(t) - \Psi_{L-1}^\top(t)\right) \\
&= \left(1-\alpha\right)\Omega_{L-1}(t),
\end{aligned} \tag{27}$$

We can now further unroll (27) in time, resulting in

$$\begin{aligned}
\Omega_{L-1}(t+1) &= \left(1-\alpha\right)^t \Omega_{L-1}(0) \\
&= \left(1-\alpha\right)^t \left(\Theta_{L-1}(0) - \Psi_{L-1}^\top(0)\right).
\end{aligned} \tag{28}$$

Therefore, $\Omega_{L-1}(t)$ converges to zero as training progresses, with the initial difference between the forward and feedback matrices decaying exponentially due to the learning rate α. In other words, we have that

$$\lim_{t\to\infty} \Omega_{L-1}(t) = 0 \quad \Longrightarrow \quad \lim_{t\to\infty} \Psi_{L-1}^\top(t) = \Theta_{L-1}(t). \tag{29}$$

The DKP update rule for Θ_{L-2} is given by

$$\begin{aligned} \Delta\Theta_{L-2} &= -\alpha \left(\tilde{\delta}_{L-1} x_{L-2}^\top + \Theta_{L-2}\right) \\ &= -\alpha \left(\Psi_{L-1}\delta_L x_{L-2}^\top + \Theta_{L-2}\right), \end{aligned} \tag{30}$$

which effectively approximates the BP update for Θ_{L-2} and ultimately matches it as $t \to \infty$, as using (29) yields

$$\lim_{t\to\infty} \Delta\Theta_{L-2} = -\alpha \left(\Theta_{L-1}^\top \delta_L x_{L-2}^\top + \Theta_{L-2}\right). \tag{31}$$

Here and throughout the rest of this appendix, we omit the time index since we consider the limit $t \to \infty$. Therefore, we approximate Ψ_{L-1} by $\Theta_{L-1}^\top$, noting that this approximation introduces a small error, since exact equality holds only at the limit.

Unfortunately, the convergence obtained for Ψ_{L-1} in (29) does not directly apply to $\Psi_{L-2} \in \mathbb{R}^{d_{L-2}\times d_L}$, as its dimensions do not match those of $\Theta_{L-2} \in \mathbb{R}^{d_{L-1}\times d_{L-2}}$. The update rule for Ψ_{L-2}, given by

$$\Delta\Psi_{L-2} = -\alpha\big(x_{L-2}\delta_L^\top + \Psi_{L_2}\big), \tag{32}$$

can be substituted into (31), leading to

$$\begin{aligned} \Delta\Theta_{L-2} &= -\alpha \left(\Theta_{L-1}^\top \delta_L x_{L-2}^\top + \Theta_{L-2}\right) \\ &= -\alpha \left(\Theta_{L-1}^\top \left(-\alpha^{-1}\Delta\Psi_{L-2}^\top - \Psi_{L-2}^\top\right) + \Theta_{L-2}\right) \\ &= \Theta_{L-1}^\top \Delta\Psi_{L-2}^\top - \alpha \left(\Theta_{L-2} - \Theta_{L-1}^\top \Psi_{L-2}^\top\right). \end{aligned} \tag{33}$$

Here, we neglect the decay terms, since they vanish asymptotically during training, as previously discussed for Θ_{L-1} and Ψ_{L-1}. We can now transpose both sides, and multiply them by $\Theta_{L-1}^\top$, resulting in

$$\Delta\Psi_{L-2}\Theta_{L-1}\Theta_{L-1}^\top = \Delta\Theta_{L-2}^\top\Theta_{L-1}^\top. \tag{34}$$

This allows us to link the update of Ψ_{L-2} to that of Θ_{L-2} through

$$\begin{aligned} \Delta\Psi_{L-2} &= \Delta\Theta_{L-2}^\top\Theta_{L-1}^\top \left(\Theta_{L-1}\Theta_{L-1}^\top\right)^{-1} \\ &= \Delta\Theta_{L-2}^\top\Theta_{L-1}^+, \end{aligned} \tag{35}$$

where $\Theta_{L-1}^+ \in \mathbb{R}^{d_{L-1}\times d_L}$ is the Moore-Penrose pseudoinverse of Θ_{L-1}, assuming the latter has full row rank. On the one hand, the BP update of Θ_{L-3} is given by

$$\begin{aligned} \Delta\Theta_{L-3} &= -\alpha\big(\delta_{L-2} x_{L-3}^\top + \Theta_{L-3}\big) \\ &= -\alpha\big(\Theta_{L-2}^\top \delta_{L-1} x_{L-3}^\top + \Theta_{L-3}\big) \\ &= -\alpha\big(\Theta_{L-2}^\top\Theta_{L-1}^\top \delta_L x_{L-3}^\top + \Theta_{L-3}\big). \end{aligned} \tag{36}$$

On the other hand, by making use of (35), which is valid under the previously mentioned assumptions and approximations, the DKP update can be expressed as

$$\begin{aligned} \Delta\Theta_{L-3} &= -\alpha\big(\tilde{\delta}_{L-2} x_{L-3}^\top + \Theta_{L-3}\big) \\ &= -\alpha\big(\Psi_{L-2}^\top \delta_L x_{L-3}^\top + \Theta_{L-3}\big) \\ &= -\alpha\big(\Theta_{L-2}^\top\Theta_{L-1}^+ \delta_L x_{L-3}^\top + \Theta_{L-3}\big). \end{aligned} \tag{37}$$

Hence, the DKP update of Θ_{L-3} approximates the BP one, and is exact if $\Theta_{L-1}^+ = \Theta_{L-1}^\top$, meaning that Θ_{L-1} is orthogonal. We now move on to the DKP update of Ψ_{L-3}, given by

$$\Delta\Psi_{L-3} = -\alpha\big(x_{L-3}\delta_L^\top + \Psi_{L-3}\big). \tag{38}$$

By repeating the same procedure as for Ψ_{L-2}, we substitute (38) into (37), resulting in

$$\begin{aligned} \Delta\Theta_{L-3} &= -\alpha\big(\Theta_{L-2}^\top\Theta_{L-1}^+\big(- \alpha^{-1}\Delta\Psi_{L-3} - \Psi_{L-3}\big)^\top + \Theta_{L-3}\big) \\ &= \Theta_{L-2}^\top\Theta_{L-1}^+\Delta\Psi_{L-3}^\top - \alpha\big(\Theta_{L-3} - \Theta_{L-2}^\top\Theta_{L-1}^+\Psi_{L-3}^\top\big). \end{aligned} \tag{39}$$

We now again do not consider the terms decaying with time, as they tend to zero as the training goes on, and focus only on the term the update converges to. As done previously, after dropping the decay term, we transpose both sides, yielding

$$\Delta\Psi_{L-3}\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)=\Delta\Theta_{L-3}^{\top}, \tag{40}$$

and multiply them by $\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)^{\top}$, leading to

$$\Delta\Psi_{L-3}\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)^{\top}=\Delta\Theta_{L-3}^{\top}\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)^{\top}. \tag{41}$$

Lastly, by multiplying both sides by the inverse of the product of matrices on the right of $\Delta\Psi_{L-3}$, we obtain

$$\begin{aligned}\Delta\Psi_{L-3}&=\Delta\Theta_{L-3}^{\top}\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)^{\top}\left[\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)^{\top}\right]^{-1}\\&=\Delta\Theta_{L-3}^{\top}\left(\left(\Theta_{L-1}^{+}\right)^{\top}\Theta_{L-2}\right)^{+},\\&=\Delta\Theta_{L-3}^{\top}\left(\left(\Theta_{L-1}^{\top}\right)^{+}\Theta_{L-2}\right)^{+},\end{aligned} \tag{42}$$

where again, the feedback matrix includes a chain of forward matrix pseudoinverses. More generally, under the assumption of $t\to\infty$ and that Θ_ℓ is a full row rank rectangular matrix, we have

$$\Psi_{L-\ell}=\begin{cases}\Theta_{L-\ell}^{\top} & \text{if } \ell=1,\\ \Theta_{L-\ell}^{\top}\left(\Psi_{L-\ell+1}^{\top}\right)^{+} & \text{if } 1<\ell<L,\end{cases} \tag{43}$$

It should be noted that in practice, the assumptions we made are never perfectly met, since neural network training does not proceed for an infinite number of iterations, meaning that the decay terms we neglected do not completely vanish, and also involves the derivatives of the nonlinear activation functions. Nonetheless, our derivation demonstrates how DKP extends DFA by updating the feedback random matrices with terms that also appear in BP's error propagation, providing a clearer understanding of why DKP achieves better alignment and consequently improves performance compared to standard DFA Webster et al. (2020).

A.2 Error propagation delay

In this section, we introduce and provide a formal proof of Theorem 1, which quantifies the delay in error propagation in forward-initialized PC networks. Specifically, we show that the feedback error signal reaches a given layer with a delay equal to its distance from the output (Zahid et al., 2023).

Theorem 1 (Error propagation delay)**.** *Consider a forward-initialized PC network with discrete-time updates. Assuming an incorrect prediction, the neural activity ϕ_ℓ at layer ℓ requires at least $\hat{t}=L-\ell$ inference-phase steps before it deviates from equilibrium and begins to evolve according to (13) (proof provided in Appendix A.2).*

Proof. Let us consider a forward-initialized PC network with $L+1$ layers, where the neural activity evolves in discrete time steps $t\in\mathbb{N}_0$, and each layer is initialized as $\phi_\ell(0)=f(\Theta_{\ell-1}\phi_{\ell-1}(0))$. The error neurons at layer ℓ are defined as $\epsilon_\ell(0)=\phi_\ell(0)-f(\Theta_{\ell-1}\phi_{\ell-1}(0))$. By construction, after forward initialization, all error neurons vanish and the network's dynamics is at equilibrium:

$$\begin{aligned}\|\epsilon_\ell(0)\|_2^2&=\|\phi_\ell(0)-f(\Theta_{\ell-1}\phi_{\ell-1}(0))\|_2^2\\&=\|\phi_\ell(0)-\phi_\ell(0)\|_2^2\\&=0\quad\text{for}\quad 0<\ell\leq L.\end{aligned} \tag{44}$$

At the beginning of the inference phase, where the network's neural activity evolves to minimize (11), we clamp the target vector y to the output layer ϕ_L. Assuming an incorrect prediction, i.e., $y-f(\Theta_{L-1}\phi_{L-1}(0))\neq 0$, the prediction error satisfies

$$\|\epsilon_L(0)\|_2^2=\|y-f(\Theta_{L-1}\phi_{L-1}(0))\|_2^2\geq 0. \tag{45}$$

Consequently, according to (13), the activity at layer $L-1$ has a nonzero update at $t=1$:

$$\begin{aligned}\phi_{L-1}(1) &= \phi_{L-1}(0) + \Delta\phi_\ell(0) \\ &= \phi_{L-1}(0) - \gamma\frac{\partial F}{\partial \phi_{L-1}}(0) \\ &= \phi_{L-1}(0) - \gamma\Big(J_{\phi_{L-1}}(0)^\top \epsilon_L - \epsilon_{L-1}\Big) \\ &= \phi_{L-1}(0) - \gamma\Big(J_{\phi_{L-1}}(0)^\top \epsilon_L\Big),\end{aligned} \tag{46}$$

where $J_{\phi_\ell}(0) = \frac{\partial f(\Theta_\ell \phi_\ell)}{\partial \phi_\ell}(0) \in \mathbb{R}^{d_{\ell+1}\times d_\ell}$ is the Jacobian matrix of prediction by layer ℓ with respect to its neural activity. For all preceding layers, we have that $\Delta\phi_\ell(0) = 0$, since both ϵ_ℓ and $\epsilon_{\ell+1}$ are null.

After ϕ_{L-1} has been updated at time $t=1$, the corresponding error becomes nonzero:

$$\begin{aligned}\|\epsilon_{L-1}(1)\|_2^2 &= \|\phi_{L-1}(1) - f(\Theta_{L-2}, \phi_{L-2}(1))\|_2^2 \\ &= \|\phi_{L-1}(1) - f(\Theta_{L-2}, \phi_{L-2}(0))\|_2^2 \\ &= \|\phi_{L-1}(1) - \phi_{L-1}(0)\|_2^2 \\ &= \|\Delta\phi_{L-1}(1)\|_2^2 \\ &\geq 0.\end{aligned} \tag{47}$$

At the subsequent timestamp $t=2$, layer $L-2$ also receives a nonzero error and updates accordingly:

$$\begin{aligned}\phi_{L-2}(2) &= \phi_{L-2}(1) + \Delta\phi_{L-2}(1) \\ &= \phi_{L-2}(1) - \gamma\frac{\partial F}{\partial \phi_{L-2}}(1) \\ &= \phi_{L-2}(0) - \gamma\frac{\partial F}{\partial \phi_{L-2}}(0) \\ &= \phi_{L-2}(0) - \gamma\Big(J_{\phi_{L-2}}(0)^\top \epsilon_{L-1} - \epsilon_{L-2}\Big) \\ &= \phi_{L-2}(0) - \gamma\Big(J_{\phi_{L-2}}(0)^\top \epsilon_{L-1}\Big),\end{aligned} \tag{48}$$

where $\phi_{L-2}(1) = \phi_{L-2}(0)$ and $\frac{\partial F}{\partial \phi_{L-2}}(1) = \frac{\partial F}{\partial \phi_{L-2}}(0)$, since ϕ_{L-2} has remained unchanged at $t=1$, due to $\epsilon_{L-2}(0) = \epsilon_{L-1}(0) = 0$. Again, at time $t=2$ all previous layers $\phi_\ell(2)$, $0 < \ell < L-2$ remain unchanged, as $\epsilon_{L-\ell}(1) = \epsilon_{L-\ell+1}(1) = 0$. By induction, it follows that under these assumptions, any layer ℓ requires at least $\hat{t} = L-\ell$ timestamps to update its neural activity and corresponding error neurons.

For the general case, at a specific time t, the error neurons at layer $L-t$ can be expressed as

$$\begin{aligned}\|\epsilon_{L-t}(t)\|_2^2 &= \|\phi_{L-t}(t) - f(\Theta_{L-t-1}, \phi_{L-t-1}(t-1))\|_2^2 \\ &= \|\phi_{L-t}(t) - f(\Theta_{L-t-1}, \phi_{L-t-1}(0))\|_2^2 \\ &= \|\phi_{L-t}(t) - \phi_{L-t}(0)\|_2^2,\end{aligned} \tag{49}$$

where $\phi_{L-t}(t) - \phi_{L-t}(0) \neq 0$ if and only if $\phi_{L-t}(t) \neq \phi_{L-t}(0)$. Crucially, this condition can occur only after $\hat{t} = L-\ell$ timestamps. According to (13), the update of layer $L-t$ depends on the errors from the current and subsequent layers, ϵ_{L-t} and ϵ_{L-t+1}, respectively. Since ϵ_{L-t} is itself blocked until ϕ_{L-t} changes, the driving term is provided by ϵ_{L-t+1}. However, the latter becomes nonzero only after the previous layer has been updated. Therefore, the propagation of activity changes and error signals strictly follows the network's hierarchy, advancing at most one layer per timestamp, starting from ϵ_L at $t=0$ and reaching layer ℓ only after $\hat{t} = L-\ell$ steps. □

A.3 Error exponential decay

In this section, we introduce and provide a formal proof of Theorem 2, showing that the squared-ℓ_2-norm of the feedback error signal for a layer ℓ generated at time $\hat{t} = L-\ell$ is bounded by a quantity that decays exponentially proportional to t (Goemaere et al., 2025).

Theorem 2 (Error exponential decay)**.** *Consider a forward-initialized PC network with discrete-time updates. Assuming an incorrect prediction, the squared ℓ_2-norm of the feedback error signal at layer ℓ, $\|\epsilon_\ell(\hat{t})\|_2^2$, at time $\hat{t} = L - \ell$, is upper-bounded by a quantity that decays $\propto \gamma^{2(L-\ell)}$ (proof provided in Appendix A.3).*

Proof. Let us consider a forward-initialized PC network with $L+1$ layers, where the neural activity evolves in discrete time steps $t \in \mathbb{N}_0$, and each hidden layer $\phi_\ell(t)$ is initialized as $\phi_\ell(0) = f(\Theta_{\ell-1}\phi_{\ell-1}(0))$. The error neurons are defined as $\epsilon_\ell(t) = \phi_\ell(t) - f(\Theta_{\ell-1}\phi_{\ell-1}(t))$. Here, we make the time dependence of neural activities explicit, as they evolve with time during the inference phase of PC. Considering the update of an arbitrary neuron $\phi_{\ell-1}$, $1 < \ell \leq L$ at time $\hat{t}+1$, with $\hat{t} = L - \ell$:

$$\phi_{\ell-1}(\hat{t}+1) = \phi_{\ell-1}(\hat{t}) - \gamma \frac{\partial F}{\partial \phi_{\ell-1}}(\hat{t}), \tag{50}$$

where we can move the on the left side the current neural activity value, obtaining:

$$\begin{aligned}
\phi_{\ell-1}(\hat{t}+1) - \phi_{\ell-1}(\hat{t}) &= -\gamma \frac{\partial F}{\partial \phi_{\ell-1}}(\hat{t}) \\
\Delta\phi_{\ell-1}(\hat{t}+1) &= -\gamma \frac{\partial F}{\partial \phi_{\ell-1}}(\hat{t}) \\
&= -\gamma \left(\epsilon_{\ell-1}(\hat{t}) - \frac{\partial f\left(\Theta_{\ell-1}\phi_{\ell-1}(\hat{t})\right)^\top}{\partial \phi_{\ell-1}(\hat{t})} \epsilon_\ell(\hat{t}) \right) \\
&= \gamma \left(\frac{\partial f\left(\Theta_{\ell-1}\phi_{\ell-1}(\hat{t})\right)^\top}{\partial \phi_{\ell-1}(\hat{t})} \epsilon_\ell(\hat{t}) \right),
\end{aligned} \tag{51}$$

where $\epsilon_{\ell-1}(\hat{t}) = 0$ is implied by Theorem 1, since $L - \ell - 1 < \hat{t}$. We defined $\frac{\partial f\left(\Theta_{\ell-1}\phi_{\ell-1}(\hat{t})\right)}{\partial \phi_{\ell-1}(\hat{t})} = J_{\phi_{\ell-1}} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ the Jacobian matrix of f at layer $\ell - 1$. Here, we have omitted time in $\phi_{\ell-1}$ for brevity, as $\phi_{\ell-1}(\hat{t}) = \phi_{\ell-1}(0)$, following again from Theorem 1. Continuing from (51), we expand the error neurons according to their definition and apply Theorem 1, resulting in:

$$\begin{aligned}
\Delta\phi_{\ell-1}(\hat{t}+1) &= \gamma J_{\phi_{\ell-1}}^\top \left[\phi_\ell(\hat{t}) - f\left(\Theta_{\ell-1}\phi_{\ell-1}(\hat{t})\right) \right] \\
&= \gamma J_{\phi_{\ell-1}}^\top \left[\phi_\ell(\hat{t}-1) - \gamma \frac{\partial F}{\partial \phi_\ell}(\hat{t}-1) - f\left(\Theta_{\ell-1}\phi_{\ell-1}(0)\right) \right] \\
&= \gamma J_{\phi_{\ell-1}}^\top \left[\phi_\ell(0) - \gamma \frac{\partial F}{\partial \phi_\ell}(\hat{t}-1) - f(\Theta_{\ell-1}\phi_{\ell-1}(0)) \right] \\
&= \gamma J_{\phi_{\ell-1}}^\top \left[f\left(\Theta_{\ell-1}\phi_{\ell-1}(0)\right) - \gamma \frac{\partial F}{\partial \phi_\ell}(\hat{t}-1) - f\left(\Theta_{\ell-1}\phi_{\ell-1}(0)\right) \right],
\end{aligned} \tag{52}$$

where on the first line we have substituted $\phi_\ell(\hat{t})$ with its definition in (50). According to Theorem 1, $\phi_\ell(\hat{t}-1) = \phi_\ell(0)$ as $L - \ell < \hat{t} - 1$, thus this allows to re-write it using the forward initialization definition, as done on the fourth line. By repeating the same steps as in (51), we obtain

$$\begin{aligned}
\Delta\phi_{\ell-1}(\hat{t}+1) &= \gamma J_{\phi_{\ell-1}}^\top \left[-\gamma \frac{\partial F}{\partial \phi_\ell}(\hat{t}-1) \right] \\
&= \gamma J_{\phi_{\ell-1}}^\top \left[-\gamma \left(\epsilon_\ell(\hat{t}-1) - J_{\phi_\ell}^\top \epsilon_{\ell+1}(\hat{t}-1) \right) \right] \\
&= \gamma^2 J_{\phi_{\ell-1}}^\top J_{\phi_\ell}^\top \epsilon_{\ell+1}(\hat{t}-1).
\end{aligned} \tag{53}$$

By unrolling the formulation backward until the last layer, we get:

$$\Delta\phi_{\ell-1}(\hat{t}+1) = \gamma^{L-\ell+1} \left(\prod_{i=0}^{L-\ell} J_{\phi_{\ell-1+i}}^\top \right) \epsilon_L(0). \tag{54}$$

By substituting (51) in (54), we can continue as follows:

$$\gamma \left(J_{\phi_{\ell-1}}^{\top} \epsilon_\ell(\hat{t}) \right) = \gamma^{L-\ell+1} \left(\prod_{i=0}^{L-\ell} J_{\phi_{\ell-1+i}}^{\top} \right) \epsilon_L(0), \tag{55}$$

which can be further simplified by eliding common terms, resulting in:

$$\epsilon_\ell(\hat{t}) = \gamma^{L-\ell} \left(\prod_{i=1}^{L-\ell} J_{\phi_{\ell-1+i}}^{\top} \right) \epsilon_L(0). \tag{56}$$

By writing the squared-ℓ_2 norm of (56), we finally derive the upper-bound for the error term:

$$\begin{aligned} \|\epsilon_\ell(\hat{t})\|_2^2 &= \|\gamma^{L-\ell} \left(\prod_{i=1}^{L-\ell} J_{\phi_{\ell-1+i}}^{\top} \right) \epsilon_L(0)\|_2^2 \\ &\leq \gamma^{2(L-\ell)} \left(\prod_{i=1}^{L-\ell} \|J_{\phi_{\ell-1+i}}^{\top}\|_2^2 \right) \|\epsilon_L(0)\|_2^2. \end{aligned} \tag{57}$$

□

A.4 Details of Classification Experiments

Models and datasets – For MLP architectures, we use MNIST and Fashion-MNIST (Yann, 2010; Xiao et al., 2017), consisting of 28×28 grayscale images with 60k training and 10k test samples, across 10 classes. For CNNs, we use CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009), consisting of 32×32 RGB images with 50k training and 10k test samples, with 10 and 100 classes, respectively. Data normalization is applied, and for CIFAR datasets, data augmentation includes random horizontal flipping (50% probability) and random cropping to 32×32 with 4-pixel padding. MLPs consist of three hidden layers with 128 units each. VGG-like CNNs of 7 and 9 layers (Simonyan & Zisserman, 2014) are tested on CIFAR-10 and CIFAR-100. For standardized comparison with prior PC work and future benchmarking, we adopt the architectures employed in Pinchetti et al. (2024). As Pinchetti et al. (2024) do not specify VGG-9 details, we assume it to be consistent with other recent works in the PC literature (Goemaere et al., 2025).

Training setup – MLPs are trained for 25 epochs and CNNs for 50 epochs, using a batch size of 128, consistent with Pinchetti et al. (2024). Forward weights are optimized with a warmup-cosine-annealing scheduler without restart. Optimizers considered include Adam and AdamW. Feedback connections are learned with a separate optimizer, using an exponentially-decaying learning rate updated per batch (decay factor $\eta \in [0.999, 1]$, with $\eta = 1$ corresponding to no decay). Various feedback initializations are tested during the hyperparameters search: Xavier-uniform/normal (Glorot & Bengio, 2010), Kaiming-uniform/normal (He et al., 2015), and orthogonal (Saxe et al., 2013). Feedback optimizers include Adam, AdamW, and Nadam (Adam et al., 2014; Dozat, 2016; Loshchilov & Hutter, 2017). All optimal hyperparameters and precise network specifications are available in our GitHub repository.

A.5 Computational Trade-offs

Resource consumption experiments focus on latency and floating-point operations (FLOPs), evaluated on both an MLP and a CNN. For the MLP, experiments are conducted on a network with 256 units per layer using a single sample from MNIST, whereas for the CNN, a VGG-like model with 64-channel 3×3 convolutions is evaluated on a single sample from CIFAR-10. Latency is defined as the time required for a complete parameter update, including feedforward initialization and the update of forward and feedback matrices (when applicable). Computational cost is estimated by counting FLOPs in forward and backward passes, restricted to core MAC operations, with 1 MAC = 2 FLOPs[4].

[4]Variable contributions, such as activation function FLOPs, are excluded as they depend on the specific nonlinearity. Accordingly, the reported FLOPs represent a lower-bound estimate of the actual computational cost

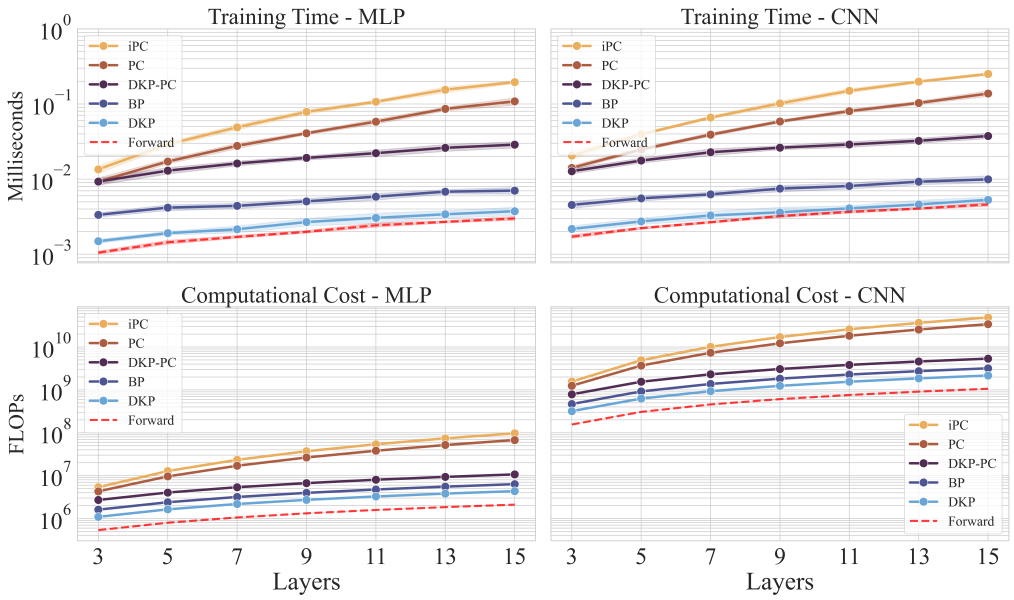

Figure 3: The first row compares the training time on a logarithmic scale, measured as the sum of the forward pass and the complete parameter updates, with the contribution of the forward pass illustrated as a red dashed line. Results are averaged over 20 samples. The second row compares the minimum FLOPs requirements, estimated from the core MAC operations. In all plots, for both PC and iPC, the number of inference-phase steps is assumed to equal the network depth. In contrast, for DKP-PC, only a single step is considered, as it has been empirically demonstrated to be enough to achieve comparable performance.

The first row of Fig. 3 illustrates the differences in training time, expressed in milliseconds, across the various models. Training time includes the forward pass and a complete backward pass, encompassing both forward and feedback weight updates, if applicable. The forward pass contribution is indicated by a red dashed line. The fastest algorithm is DKP, owing to the reduced dimensionality introduced in its update in (23). BP is the second fastest algorithm, followed by DKP-PC. The latter, as well as the other local algorithms, have been evaluated in sequential mode, and therefore their parallelization potential is not considered in this section. Nevertheless, this already highlights the speedup of DKP-PC compared to standard PC or iPC, as it consistently requires less time to fully update its parameters for both the MLP and CNN.

The second row of Fig. 3 compares the minimum FLOPs requirements across algorithms. DKP emerges as the most efficient method in architectures where hidden layers exceed the output layer in size. In this scenario, computing the intermediate error term δ_ℓ only requires multiplying the output error $\delta_L \in \mathbb{R}^{d_L}$ by the random matrix $\Psi_\ell \in \mathbb{R}^{d_\ell \times d_L}$, which entails fewer MAC operations than BP. In contrast, BP requires multiplying $\Theta_{\ell+1}^\top \in \mathbb{R}^{d_{\ell+1} \times d_{\ell+2}}$ with the higher-layer error $\delta_{\ell+2} \in \mathbb{R}^{d_{\ell+2}}$, typically with $d_L < d_\ell$ for all $\ell \in \{0, \dots, L-1\}$. BP is the second most efficient algorithm overall, followed in order by DKP-PC, PC, and iPC. The logarithmic scale of the plot highlights the growth in computational complexity for PC and iPC as depth increases, since both the minimal number of inference-phase steps and the number of multiple matrix–vector multiplications per inference-phase step increase with depth. DKP-PC scales better than PC and iPC as it requires only one inference-phase step to match or surpass their accuracy, achieving nearly an order of magnitude fewer FLOPs, thereby underscoring its efficiency advantage.

A.6 Analysis of Parallel Execution

While a fully-parallel implementation of PC is theoretically possible, it has so far been practically limited by the signal error delay and exponential decay problems (Zahid et al., 2023; Pinchetti et al.,

2024; Goemaere et al., 2025) detailed in Section 3.1. Here, we discuss how DKP-PC overcomes these limitations, yielding an algorithm capable of achieving lower training latency than BP.

Referring to Algorithm 1 and excluding the forward initialization, which incurs the same computational cost as BP, we first consider the *Direct Feedback Alignment update* (1). Since each forward weight update depends only on the local neural activity and the error signal propagated from the output layer via the corresponding feedback matrix, all updates can be executed in parallel, reducing the time complexity of this phase from $\mathcal{O}(L)$ to $\mathcal{O}(1)$. In the subsequent *Inference phase* (2), which is typically executed over multiple steps ($T \geq L$) in PC, we empirically demonstrate in Table 1 that a single step suffices to achieve accuracy comparable to or exceeding that of standard PC, reducing the time complexity from $\mathcal{O}(T)$ to $\mathcal{O}(1)$. Within this phase, updates of error neurons and neural activities are also parallelizable, as each relies solely on locally available information, resulting in $\mathcal{O}(1)$ time complexity. For completeness, we highlight that synchronization is still required, as error terms must be computed before updating neural activities. Successively, the *Learning phase* (3) and *Direct Kolen-Pollack update* (4) are executed sequentially, though they are independent and can be performed simultaneously. While these phases respectively have to iterate over all $L - 1$ forward and backward parameter matrices, their computations are entirely local and hence can be executed in parallel across layers, reducing the time complexity from $\mathcal{O}(L)$ to $\mathcal{O}(1)$.

In summary, DKP-PC consists of four phases, each theoretically fully parallelizable with time complexity $\mathcal{O}(1)$. Each phase blocks the next, except for the final two, which may run concurrently. Consequently, DKP-PC's time complexity is independent of network depth, in contrast to BP, which requires $\mathcal{O}(L)$. For sufficiently deep networks, we claim that the overall training time of DKP-PC will be significantly lower than that of BP. In practice, however, this advantage is challenged by existing hardware, which is heavily optimized for BP, and by the synchronization overhead inherent in software-based parallelization. These limitations could nevertheless be overcome by custom hardware accelerators designed to fully exploit DKP-PC's parallelizability.

A.7 Large Language Models usage disclosure

Large Language Models (LLMs) were employed only for language polishing, such as grammar and phrasing refinement. They were not used for content generation, results analysis, or methodological development.