ePC: Fast and Deep Predictive Coding for Digital Hardware

Anonymous authors
Paper under double-blind review

Abstract

Predictive Coding (PC) offers a bio-inspired alternative to backpropagation for neural network training, described as a physical system minimizing its internal energy. However, in practice, PC is predominantly *digitally simulated*, requiring excessive amounts of compute while struggling to scale to deeper architectures. This paper reformulates PC to overcome this hardware-algorithm mismatch. First, we uncover how the canonical state-based formulation of PC (sPC) is, by design, deeply inefficient in digital simulation, inevitably resulting in exponential signal decay that stalls the entire minimization process. Then, to overcome this fundamental limitation, we introduce error-based PC (ePC), a novel reparameterization of PC which does not suffer from signal decay. Though no longer biologically plausible, ePC numerically computes exact PC weights gradients and runs orders of magnitude faster than sPC. Experiments across multiple architectures and datasets demonstrate that ePC matches backpropagation's performance even for deeper models where sPC struggles. Besides practical improvements, our work provides theoretical insight into PC dynamics and establishes a foundation for scaling PC-based learning to deeper architectures on digital hardware and beyond.

1 Introduction

Originally a neuroscience theory of cortical function (Friston and Kiebel, 2009), Predictive Coding (PC) has recently been reformulated as a general machine learning algorithm, offering a bio-inspired alternative to backpropagation with distinct learning dynamics (Bogacz, 2017; Whittington and Bogacz, 2017, 2019; Millidge et al., 2022b). Unlike backprop, PC produces weight gradients through a two-step process: first, infer the optimal state of neuron activations that should result from learning, and only then update the weights. This approach of "inferring activity before plasticity" (Song et al., 2024) has been found advantageous for learning: it improves the geometry of the loss landscape (Innocenti et al., 2024b) and reduces interference between competing training signals, leading to improved learning capabilities in online and continual learning settings (Song et al., 2024).

Our work focuses on PC's critical first step, inferring the optimal activations, specified as an energy minimization. Concretely, each layer tries to predict the state of the next layer, continually adjusting its own state to reduce the local prediction loss (known as 'energy'). Reminiscent of a physical process, PC would, in theory, be ideally suited for neuromorphic implementation, though no such hardware exists yet. Instead, current PC research relies on digital simulation with numerical solvers, requiring numerous iterations to reach state convergence. Due to this hardware-algorithm mismatch, PC incurs substantial overhead compared to backprop, which maps naturally to digital hardware.

Another scaling issue, observed by Pinchetti et al. (2025), is that, even in simple supervised settings, deeper PC-trained models often perform worse than shallower ones, in contrast to backpropagation. Recent efforts have explored this depth scaling failure from different angles. Several works observed a highly uneven energy distribution across the network (Ha et al., 2025; Pinchetti et al., 2025; Qi et al., 2025), leading to weaker weight gradients for deeper layers; however, the underlying mechanism remained unclear. Proposed solutions either modify PC's weight gradient formulas (Qi et al., 2025)

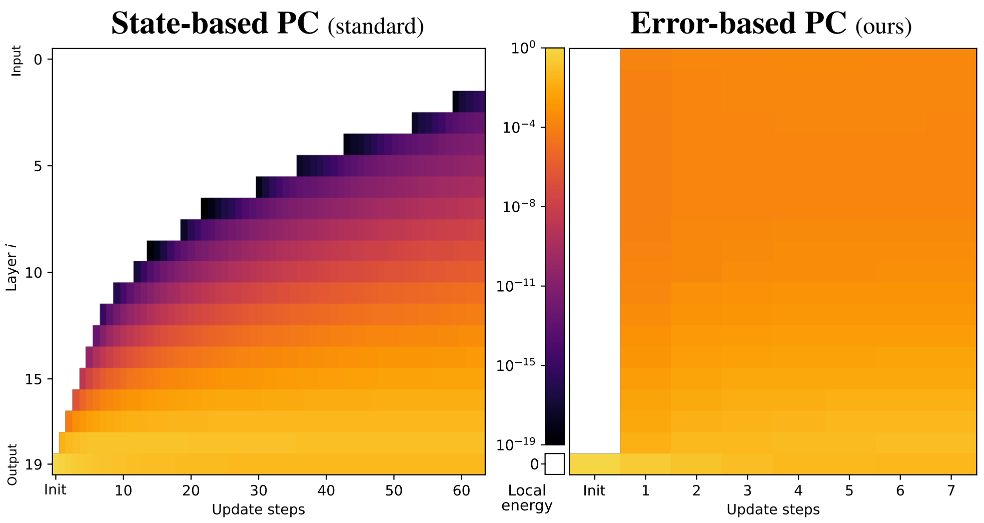


Figure 1: **Dynamics of layerwise energies during energy minimization.** Standard state-based PC struggles to propagate signals through the network, with progressively longer delays at deeper layers. By contrast, our error-based PC converges to equilibrium within just a few update steps, thanks to its global signal propagation. Results for an untrained 20-layer MLP on a random MNIST input.

or stay limited to densely-connected or residual architectures (Ha et al., 2025; Innocenti et al., 2025). A general solution for standard feedforward models using exact PC remains an open problem.

In this paper, we address the fundamental limitations of PC's digital simulation—which currently represents nearly all practical PC research. Our work connects the seemingly disparate problems of depth scaling and computational efficiency in PC networks, uncovering a common underlying cause and providing a simple solution.

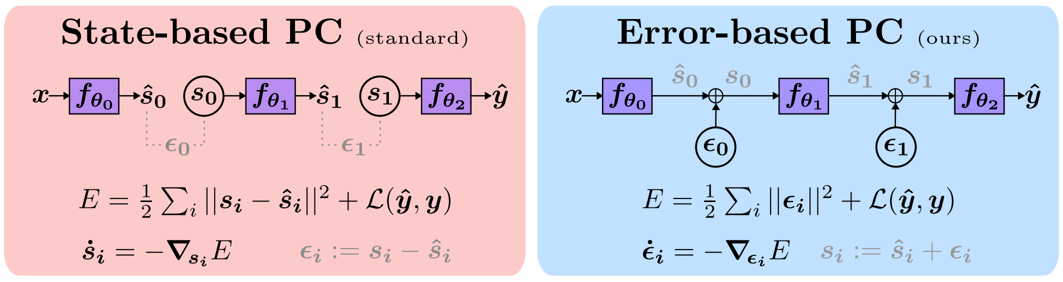
Our contributions

- We identify a fundamental signal decay mechanism in traditional state-based PC (sPC), whereby signals attenuate exponentially as they propagate through the network (see Fig. 1), explaining both slow convergence and poor performance in deeper PC networks.
- We introduce error-based PC (ePC), a novel reparameterization of PC that eliminates signal decay by directly optimizing over prediction errors rather than faithfully simulating the physical process. Crucially, ePC provably computes the exact same state equilibrium as sPC.
- We empirically demonstrate that ePC converges up to three orders of magnitude faster than sPC on deep networks, resolving a major computational bottleneck in PC research.
- Through comprehensive experiments across architectures of varying depths, following the setup of Pinchetti et al. (2025), we demonstrate that ePC consistently achieves performance comparable to backpropagation, providing an effective solution to PC's depth scaling issues.

Algorithmic focus of this work While PC originated as a neuroscience theory, our work focuses solely on its application as a machine learning algorithm. Strictly local interactions, though essential for biological systems, lead to slow convergence in digital simulations. By relaxing this constraint, ePC achieves dramatic speedups while computing the same weight gradients as sPC at equilibrium.

2 Predictive Coding as an Energy Minimization over States

To establish the groundwork for our contributions, we first present the canonical formulation of PC as an energy minimization over neural states. We will refer to this as ***state-based PC*** **(sPC)**.


State-based PC (standard)
x
f_{θ_0}
$\hat{s}_0$
s_0
f_{θ_1}
$\hat{s}_1$
s_1
f_{θ_2}
$\hat{y}$
ϵ_0
ϵ_1
$E = \frac{1}{2}\sum_i ||s_i - \hat{s}_i||^2 + \mathcal{L}(\hat{y}, y)$
$\dot{s}_i = -\nabla_{s_i} E$
$\epsilon_i := s_i - \hat{s}_i$
Error-based PC (ours)
x
f_{θ_0}
$\hat{s}_0$
s_0
f_{θ_1}
$\hat{s}_1$
s_1
f_{θ_2}
$\hat{y}$
ϵ_0
ϵ_1
$E = \frac{1}{2}\sum_i ||\epsilon_i||^2 + \mathcal{L}(\hat{y}, y)$
$\dot{\epsilon}_i = -\nabla_{\epsilon_i} E$
$s_i := \hat{s}_i + \epsilon_i$

(a) State-based PC, the standard formulation of PC with a locally connected computational graph

(b) Error-based PC, our reparametrization of PC with a fully-connected (global) graph structure

Figure 2: Structural comparison of sPC (left) and ePC (right), highlighting functional equivalence

In sPC (and PC in general), each layer attempts to predict the state of the next layer. The main goal is to minimize E, the sum of all prediction errors, typically expressed as an energy function:

$$E(\boldsymbol{s}, \boldsymbol{\theta}) = \frac{1}{2} \sum_{i=0}^{L-1} ||\boldsymbol{s_i} - \boldsymbol{\hat{s}_i}||^2 + \mathcal{L}(\boldsymbol{\hat{y}}, \boldsymbol{y}), \tag{1}$$

where $\boldsymbol{s_i}$ denotes the neural state at layer i and $\boldsymbol{\hat{s}_i} := \boldsymbol{f_{\theta_i}}(\boldsymbol{s_{i-1}})$ the parametrized prediction of $\boldsymbol{s_i}$ based on the preceding layer's state $\boldsymbol{s_{i-1}}$, as illustrated in Fig. 2a. For ease of notation, we define $\boldsymbol{s_{-1}} := \boldsymbol{x}$ (the input data) and $\boldsymbol{\hat{y}} := \boldsymbol{\hat{s}_L}$ (output prediction of the target $\boldsymbol{y}$). The output loss $\mathcal{L}$ may be chosen freely (Pinchetti et al., 2022), with squared error being the common choice in PC literature.

As a learning algorithm, PC's primary purpose is to produce informative gradients for training the parameters $\boldsymbol{\theta}$. Contrary to backpropagation, sPC achieves this through purely local weight updates, relying on intermediaries (like the states) to spread the relevant learning signals across the network.

A single weight update step in sPC consists of a two-phased energy minimization of $E(\boldsymbol{s}, \boldsymbol{\theta})$:

1. **State updates:** With the parameters $\boldsymbol{\theta}$ kept fixed, the states $\boldsymbol{s}$ evolve continuously to minimize E, until equilibrium is reached. The state dynamics for layer i follow:

$$\dot{s}_i := \frac{\partial \boldsymbol{s_i}}{\partial t} = -\boldsymbol{\nabla}_{\boldsymbol{s_i}} E(\boldsymbol{s}, \boldsymbol{\theta}) = -\boldsymbol{\epsilon_i} + \boldsymbol{\epsilon_{i+1}} \frac{\partial \boldsymbol{f_{\theta_{i+1}}}}{\partial \boldsymbol{s_i}}(\boldsymbol{s_i}), \tag{2}$$

   where $\boldsymbol{\epsilon_i} := \boldsymbol{s_i} - \boldsymbol{\hat{s}_i}$ represents the layerwise prediction error.

2. **Weight update:** With $\boldsymbol{s}$ kept fixed, the parameters $\boldsymbol{\theta}$ are updated once, further minimizing E:

$$\Delta \boldsymbol{\theta_i} \propto -\boldsymbol{\nabla}_{\boldsymbol{\theta_i}} E(\boldsymbol{s}, \boldsymbol{\theta}) = \boldsymbol{\epsilon_i} \frac{\partial \boldsymbol{f_{\theta_i}}}{\partial \boldsymbol{\theta_i}}(\boldsymbol{s_{i-1}}) \tag{3}$$

Full training involves repeating this procedure over numerous data batches, as in standard Deep Learning. The distinctive feature of sPC, however, is that both phases can be implemented efficiently in biological neural circuits with strictly local computation (Whittington and Bogacz, 2017, 2019).

Finding the state equilibrium Notice how Eq. (3) requires only the final equilibrium states, discarding the trajectory taken to reach them. The specific method used to find these states is irrelevant to PC's weight updates, allowing researchers to freely choose their preferred approach.

By far the most popular choice is to discretize Eq. (2) in time, reducing it to an SGD optimization

$$\boldsymbol{s_i} \overset{\text{SGD}}{\leftarrow} \boldsymbol{s_i} - \lambda \boldsymbol{\nabla}_{\boldsymbol{s_i}} E, \tag*{(State update step)}$$

with λ the state learning rate, commonly on the order of 0.01-0.1. Typically, in practice, the number of update steps T is kept constant (a hyperparameter), and convergence is simply assumed.

The PC community has also experimented with more advanced options. Some have looked into ODE solvers (Innocenti et al., 2024a) or momentum-based optimizers (Pinchetti et al., 2025). Others have explored approximate one-step regimes (Salvatori et al., 2024), sequential update orders (Alonso et al., 2024), or auxiliary networks for direct equilibrium prediction (Tschantz et al., 2023). Yet, despite these advances, substantial gaps remain in computational efficiency and overall understanding of PC.

Feedforward state initialization A common practice is to set the initial states $\boldsymbol{s}^{t=0}$ via a feedforward pass of the input $\boldsymbol{x}$ through the network. At each layer, the prediction $\hat{\boldsymbol{s}}_{\boldsymbol{i}}$ is copied onto $\boldsymbol{s}_{\boldsymbol{i}}$, initializing the local prediction error $\boldsymbol{\epsilon}_{\boldsymbol{i}}$ to exactly zero. Despite lacking theoretical justification, the technique is widely used due to its empirical success in accelerating the state optimization process.

3 The Problem of Exponential Signal Decay in State-based PC

In this section, we uncover a previously unidentified mechanism in state-based PC networks: the exponential decay of training signal during energy minimization. This discovery represents a fundamental limitation that affects the scalability of deep PC networks and helps explain the performance gap with backpropagation, which was observed to worsen for deeper models (Pinchetti et al., 2025).

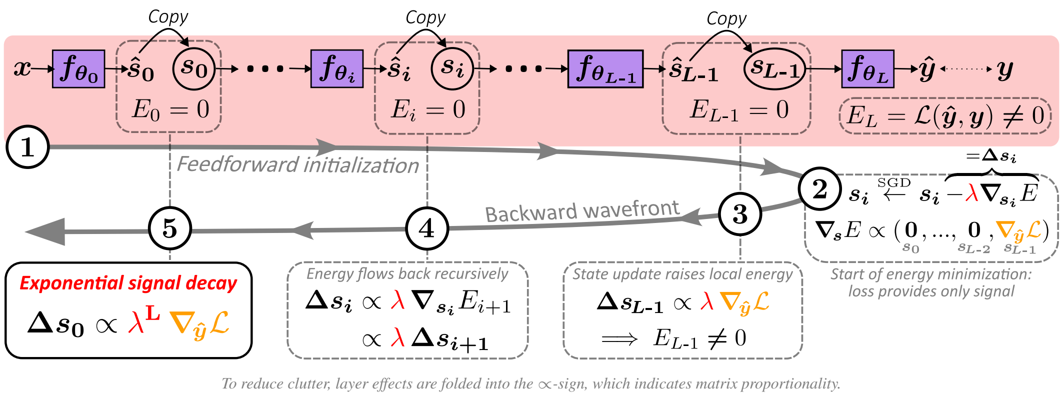
3.1 Signal Propagation in sPC: Smooth in Theory, Suppressed in Practice

In sPC, after feedforward state initialization, all energies are set to zero except for the output loss $\mathcal{L}$. Next, during state updates, we expect a backward signal to travel continuously from output to input, advancing one layer per update step. The theory suggests a clear chain reaction: non-zero energy at any layer should induce changes in neighboring states, thereby continuously propagating the signal further down the network.

However, our empirical observations contradict this expectation. Fig. 1 illustrates how, in practice, the signal travels discontinuously through the network, halting at deeper layers with progressively longer delays. Paradoxically, we observe that a non-zero energy at one layer fails to immediately propagate to adjacent layers, remaining dormant for multiple update steps before inducing detectable changes. Moreover, this behavior seems to scale logarithmically with time, requiring exponentially many update steps for signals to reach the bottom layers—an impractical computational requirement.

3.2 Uncovering a Mechanism of Exponential Signal Decay

To gain some insight into the cause of this suppressed signal propagation, we can track the state dynamics at the start of sPC. Below, in Fig. 3, we present a step-by-step description of the initial wavefront travelling backwards through the network. Our analysis uncovers a signal decay mechanism: when an energy gradient propagates from one layer to the next, it is attenuated by the state learning rate λ (necessarily < 1 for stability). With each subsequent layer traversal, this attenuation compounds multiplicatively, resulting in exponential decay with respect to network depth.


Copy
Copy
Copy
x
f_{θ_0}
$\hat{s}_0$
s_0
$E_0 = 0$
f_{θ_i}
$\hat{s}_i$
s_i
$E_i = 0$
$f_{\theta_{L\text{-}1}}$
$\hat{s}_{L\text{-}1}$
$s_{L\text{-}1}$
$E_{L\text{-}1} = 0$
f_{θ_L}
$\hat{y}$
y
$E_L = \mathcal{L}(\hat{y}, y) \neq 0$
1
Feedforward initialization
2
$s_i \overset{\text{SGD}}{\leftarrow} s_i \overbrace{-\lambda \nabla_{s_i} E}^{=\Delta s_i}$
$\nabla_s E \propto (\underset{s_0}{0}, ..., \underset{s_{L-2}}{0}, \underset{s_{L-1}}{\nabla_{\hat{y}} \mathcal{L}})$
Start of energy minimization: loss provides only signal
3
State update raises local energy
$\Delta s_{L\text{-}1} \propto \lambda \nabla_{\hat{y}} \mathcal{L}$
$\implies E_{L\text{-}1} \neq 0$
Backward wavefront
4
Energy flows back recursively
$\Delta s_i \propto \lambda \nabla_{s_i} E_{i+1}$
$\propto \lambda \Delta s_{i+1}$
5
Exponential signal decay
$\Delta s_0 \propto \lambda^L \nabla_{\hat{y}} \mathcal{L}$
To reduce clutter, layer effects are folded into the $\propto$-sign, which indicates matrix proportionality.

Step-by-step dynamics of state-based Predictive Coding for input x and target y

① **Feedforward initialization:** Due to state copying, all internal energies $E_0, \ldots, E_{L-1}$ start at zero, with in general a non-zero output energy E_L.

② **Start of energy minimization:** As gradient descent (with state learning rate λ) begins, the output loss introduces the only non-zero gradient $\nabla_{\hat{y}} \mathcal{L}$.

③ **Top-layer state update:** The top layer is updated first, raising its energy above zero.

④ **Recursive propagation:** A backward wavefront emerges: each non-zero energy induces a state change in the preceding layer, recursively propagating a diminishing signal.

⑤ **Exponential decay:** By the input layer, the signal has faded exponentially with depth L.

Figure 3: Step-by-step dynamics of sPC reveals an exponential signal decay in the backward path.

Specifically, for a network of depth L, the first non-zero signal to reach state $\boldsymbol{s_i}$ can be modelled as

$$\boldsymbol{\Delta s_i} \propto \lambda^{L-i}\, \boldsymbol{\nabla_{\hat{y}}}\mathcal{L}, \tag{4}$$

with $\boldsymbol{\nabla_{\hat{y}}}\mathcal{L}$ the initial gradient provided by the loss, and the $\propto$-sign used to ignore layer effects, such as regular vanishing gradients, which would only exacerbate the issue. Crucially, architectural tweaks, like skip connections and normalization layers, cannot resolve this sPC-specific decay.

Moreover, for typical values of λ (0.01-0.1), signals attenuate below numerical precision bounds within just 4 to 8 update steps.[1] This explains why theoretically continuous propagation manifests as discrete, delayed jumps in practice, with increasingly pronounced effects at greater network depths.

Crucially, the signal decay mechanism persists beyond the initial wavefront, plaguing the entire energy minimization process. Appendix B provides an approximate global analysis, hinting at a hardware-algorithm mismatch where *digital simulations* of sPC are problematic, not sPC itself. While physical realizations (like the brain) would handle local interactions efficiently, enforcing these constraints digitally leads to signal decay and potentially misleading conclusions on (s)PC.

3.3 Implications for Deep PC Networks

This exponential signal decay has profound implications for the digital simulation of sPC networks. Critically, deeper layers may remain entirely untrained if optimization is terminated before any signal has arrived, with these layers now effectively representing purely random input transformations. Such incomplete training would be hard to detect from state convergence metrics alone, which may incorrectly suggest equilibrium has been reached based on a lack of state changes, when in reality, these layers have yet to begin meaningful optimization.

A more nuanced implication emerges when considering which signals do successfully penetrate the network. Only hard-to-classify or mislabeled inputs could produce output gradients $\boldsymbol{\nabla_{\hat{y}}}\mathcal{L}$ that are large enough to overcome the exponential attenuation, potentially creating a systemic bias where different layers of the feedforward network train on different subsets of the data distribution.

Even when signals eventually reach deeper layers, the ensuing state modification will struggle to propagate back to upper layers. This creates a persistent misalignment where top layers, despite receiving strong output signals, cannot efficiently adapt to changes in deeper representations. While feedforward state initialization partially mitigates this issue, it cannot eliminate the intrinsic interdependencies that exist between states throughout optimization.

This signal propagation challenge represents a significant theoretical and practical limitation to scaling sPC networks to greater depths. Common remedies, like increased learning rates, higher numerical precision, or alternative optimizers, address symptoms without resolving the core issue (Pinchetti et al., 2025), underscoring the need to identify a more suitable standard formulation of PC.

4 Shifting from States to Errors: PC without Signal Decay

We introduce ***error-based PC*** **(ePC)**, a novel reparameterization of Predictive Coding that directly addresses the exponential signal decay problem identified in the previous section.

The key insight of ePC is to reformulate PC dynamics in terms of errors rather than states. By restructuring the computational graph from locally to globally connected, ePC enables signals to reach all layers simultaneously without attenuation. Though no longer biologically plausible, ePC provably computes the same state equilibrium as sPC, resulting in exact-PC weight gradients.

4.1 Mathematical Formulation of Error-Based PC

ePC reparameterizes PC by making the local prediction errors $\boldsymbol{\epsilon}$ the primary variables to optimize, rather than the states $\boldsymbol{s}$. The energy function remains the same, now formulated as:

$$E(\boldsymbol{\epsilon}, \boldsymbol{\theta}) = \frac{1}{2}\sum_{i=0}^{L-1} \|\boldsymbol{\epsilon_i}\|^2 + \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \qquad \text{with} \quad \hat{\boldsymbol{y}} = \boldsymbol{f_\theta}(\boldsymbol{x}, \boldsymbol{\epsilon}) \tag{5}$$

[1] In float32, addition only works up to 8 orders of magnitude (e.g., $1{+}10^{-8} = 1$), a.k.a. "machine epsilon".

Algorithm 1: State-based PC *(standard)*

```
    State updates
 1: Initialize states $\{s_i\} \leftarrow$ ff_init($x$)
 2: for $t = 1$ to $T$ do
 3:     $s_{-1} \leftarrow x$
 4:     for $i = 0$ to $L-1$ do                     ▷ Parallel
 5:         $\hat{s}_i \leftarrow f_{\theta_i}(s_{i-1})$
 6:         $\epsilon_i \leftarrow s_i - \hat{s}_i$
 7:     $\hat{y} \leftarrow f_{\theta_L}(s_{L-1})$
 8:     $E \leftarrow \frac{1}{2}\sum_{i=0}^{L-1} \|s_i - \hat{s}_i\|^2 + \mathcal{L}(\hat{y}, y)$
 9:     $\nabla_{s_j} E \leftarrow \epsilon_j - \frac{\partial \hat{s}_{j+1}}{\partial s_j}^T \epsilon_{j+1}$     ▷ Local
10:     $s_j \leftarrow s_j - \lambda \nabla_{s_j} E$ for all $j$
    Weight update
11: $\nabla_{\theta_j} E \leftarrow -\frac{\partial \hat{s}_j}{\partial \theta_j}^T \epsilon_j$     ▷ Local
12: $\theta_j \leftarrow \theta_j - \eta \nabla_{\theta_j} E$ for all $j$
```

Algorithm 2: Error-based PC *(ours)*

```
    Error updates
 1: Initialize errors $\{\epsilon_i\} \leftarrow$ zero_init
 2: for $t = 1$ to $T$ do
 3:     $s_{-1} \leftarrow x$
 4:     for $i = 0$ to $L-1$ do                     ▷ Sequential
 5:         $\hat{s}_i \leftarrow f_{\theta_i}(s_{i-1})$
 6:         $s_i \leftarrow \hat{s}_i + \epsilon_i$
 7:     $\hat{y} \leftarrow f_{\theta_L}(s_{L-1})$
 8:     $E \leftarrow \frac{1}{2}\sum_{i=0}^{L-1} \|\epsilon_i\|^2 + \mathcal{L}(\hat{y}, y)$
 9:     $\nabla_{\epsilon_j} E \leftarrow \epsilon_j + \frac{\partial \hat{y}}{\partial \epsilon_j}^T \nabla_{\hat{y}} \mathcal{L}$     ▷ Backprop
10:     $\epsilon_j \leftarrow \epsilon_j - \lambda \nabla_{\epsilon_j} E$ for all $j$
    Weight update
11: $\nabla_{\theta_j} E \leftarrow -\frac{\partial \hat{s}_j}{\partial \theta_j}^T \epsilon_j$     ▷ Local
12: $\theta_j \leftarrow \theta_j - \eta \nabla_{\theta_j} E$ for all $j$
```

Figure 4: Algorithmic comparison of sPC vs. ePC, with structural differences highlighted in color. Loops over j are omitted for brevity. An extended version is provided in Appendix A.

The core dynamics remain unchanged: during training, errors ϵ are iteratively updated to minimize E, followed by a gradient step to further minimize E with respect to θ (exactly Eq. (3) again). Crucially, ePC remains a valid PC algorithm (as technically verified in Appendix C.1).

When needed, states can be derived from errors through the recursive relationship $s_i := \hat{s}_i + \epsilon_i$, where still $\hat{s}_i := f_{\theta_i}(s_{i-1})$. Conceptually, this amounts to a feedforward pass starting from the input x with perturbations ϵ_i applied at each layer, as graphically shown in Fig. 2b.

Fig. 4 demonstrates the close algorithmic parallels between sPC and ePC, with a more extensive comparison given in Fig. A.1. Such strong similarities should not be surprising, as both methods are valid parametrizations of PC; in fact, they are equivalent (see proof in Appendix C.2).

4.2 Computational Advantages: Resolving the Signal Decay Problem

The key difference between sPC and ePC lies in the structure of their computational graph, as shown in Fig. 2. Striving for biological plausibility, sPC intentionally breaks the graph to enforce local update information, inadvertently resulting in exponential signal decay when simulated on digital hardware, as explained in Section 3. To avoid this issue, ePC reconnects the entire network graph, thereby creating a direct relationship between all input variables and the predicted output:

$$\begin{aligned} \textbf{(ePC)} \quad & \hat{y} = \text{func}(x, \epsilon_0, \epsilon_1, \ldots, \epsilon_{L-1}) \\ \text{vs. } \textbf{(sPC)} \quad & \hat{y} = \text{func}(s_{L-1}) \end{aligned}$$

This restructuring enables the main advantage of ePC: the use of backpropagation to transmit signals from the output loss $\mathcal{L}(\hat{y}, y)$ directly to all errors ϵ_i via $\hat{y}$, without intermediate attenuation.

A brief step-by-step analysis reveals how ePC successfully decouples stability from propagation speed, which were problematically intertwined in sPC. First, backpropagation computes gradients throughout the entire network, ensuring signals reach all layers unattenuated. Only thereafter, during the actual error update step, is the learning rate applied, affecting stability but not propagation reach. This separation allows signals to influence all network layers simultaneously, regardless of depth, thereby eliminating the exponential decay problem seen in sPC.

While ePC might appear to be a hybrid of PC and backprop, this characterization is misleading: ePC remains fundamentally a PC algorithm. Backpropagation serves only as a computational tool to efficiently reach state equilibrium on digital hardware, without influencing the weight updates, which stay temporally local following PC principles. Appendices C.3 and C.4 explore the nuanced relationship between ePC and backpropagation in greater detail.

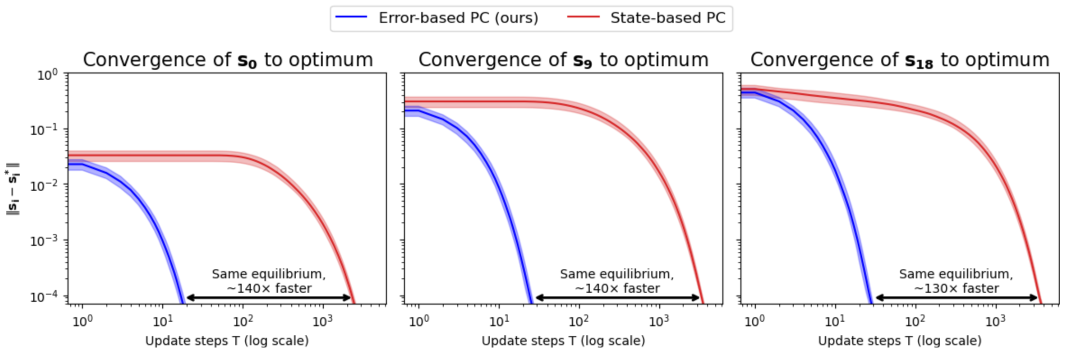


Figure 5: State convergence dynamics of bottom, middle, and top hidden layers in a 20-layer linear PC network trained on MNIST. Curves show batch medians (n=64) of L2 distance to the analytical optimum, with interquartile shading. ePC converges over $100\times$ faster than sPC for the same model.

4.3 Proof-of-Concept on MNIST

To evaluate the practical advantages of ePC over sPC, we compared the two methods for a 20-layer linear PC network trained on MNIST. This architecture provides an ideal testbed as it offers a unique and analytically tractable equilibrium state. For an unbiased comparison, we used identical network weights for both approaches (obtained through backpropagation as neutral method), with hyperparameters optimized for convergence speed. Complete details are provided in Appendix D.1.

Fig. 5 illustrates how both sPC and ePC converge to the analytical optimum, reconfirming their theoretical equivalence. However, for the exact same model, ePC converges over $100\times$ faster than sPC, a huge difference in speed that highlights ePC's practical advantage for training deep PC networks.

The figure also provides further evidence for the discontinuous signal propagation issue identified in Section 3. In sPC, the signal takes roughly 30 steps to advance 9 layers and reach $\boldsymbol{s_9}$, and nearly 100 steps to traverse the 20-layer network and reach $\boldsymbol{s_0}$. In contrast, ePC has long converged by then, with its global connectivity enables all layers to optimize immediately and simultaneously.

Additional experiments with deep non-linear MLPs (see Appendix D.2) yielded similar results, with ePC consistently outperforming sPC in terms of convergence speed. Notably, sPC required an impractical number of update steps (>100,000) to actually reach proper convergence, reinforcing the necessity of our ePC reformulation for scaling PC to deeper networks.

4.4 Implications for Deep PC Networks

The benefits of ePC's improved signal propagation extend beyond faster convergence, addressing several fundamental limitations of sPC. Most importantly, by providing signal to all layers from the first step, ePC completely resolves the issue of untrained deep layers. It enables all layers to begin optimization simultaneously, regardless of network depth.

Furthermore, ePC eliminates the potential systemic bias in sPC, where only inputs generating large output gradients could successfully influence deeper layers. With ePC, all inputs contribute equally to training at all depths, promoting uniform learning across the network.

Moreover, any change in deeper layers is efficiently communicated to the upper layers through the feedforward pass required for $\hat{\boldsymbol{y}}$. This bidirectional efficiency explains why the widely-used feedforward state initialization heuristic in PC works so well: it essentially implements the first step of ePC. Our formulation thus provides theoretical backing for this empirical practice while extending its benefits throughout the optimization process.

By resolving these fundamental limitations, ePC establishes a solid foundation for scaling PC to deeper architectures. Our MNIST proof-of-concept demonstrates significant convergence improvements, validating this theoretical advancement and motivating large-scale empirical evaluation.

5 Experiments

To evaluate ePC's effectiveness in training deep networks and compare it against sPC, we conducted extensive experiments using backpropagation as gold standard. Our experimental design follows the benchmark established by Pinchetti et al. (2025), allowing direct comparison with their findings.

Table 1: Test accuracies (in %) of ePC, sPC and backprop for various models, losses, and datasets. Bold indicates best results within confidence intervals (mean ± 1 std. dev.; taken over 5 seeds).

Loss $\mathcal{L}$	*Mean Squared Error*			*Cross-Entropy*		
Training algorithm	ePC	sPC	Backprop	ePC	sPC	Backprop
MLP (4 layers)						
MNIST	$\mathbf{98.28}^{\pm \mathbf{0.09}}$	$\mathbf{98.42}^{\pm \mathbf{0.08}}$	$\mathbf{98.30}^{\pm \mathbf{0.15}}$	$\mathbf{98.11}^{\pm \mathbf{0.08}}$	$\mathbf{98.01}^{\pm \mathbf{0.15}}$	$\mathbf{98.13}^{\pm \mathbf{0.08}}$
FashionMNIST	$87.02^{\pm 0.24}$	$88.01^{\pm 0.09}$	$\mathbf{88.79}^{\pm \mathbf{0.21}}$	$87.58^{\pm 0.13}$	$88.00^{\pm 0.24}$	$\mathbf{88.87}^{\pm \mathbf{0.27}}$
VGG-5						
CIFAR-10	$\mathbf{88.70}^{\pm \mathbf{0.12}}$	$86.67^{\pm 0.20}$	$\mathbf{88.58}^{\pm \mathbf{0.12}}$	$\mathbf{88.27}^{\pm \mathbf{0.18}}$	$84.66^{\pm 0.33}$	$\mathbf{87.95}^{\pm \mathbf{0.29}}$
CIFAR-100 (Top-1)	$64.37^{\pm 0.17}$	$50.41^{\pm 1.45}$	$\mathbf{64.80}^{\pm \mathbf{0.24}}$	$63.39^{\pm 0.25}$	$56.85^{\pm 0.69}$	$\mathbf{63.83}^{\pm \mathbf{0.15}}$
CIFAR-100 (Top-5)	$85.28^{\pm 0.38}$	$77.41^{\pm 1.21}$	$\mathbf{85.80}^{\pm \mathbf{0.13}}$	$\mathbf{87.34}^{\pm \mathbf{0.14}}$	$83.11^{\pm 0.19}$	$\mathbf{87.43}^{\pm \mathbf{0.06}}$
VGG-7						
CIFAR-10	$\mathbf{88.98}^{\pm \mathbf{0.19}}$	$77.79^{\pm 0.34}$	$\mathbf{88.94}^{\pm \mathbf{0.32}}$	$88.84^{\pm 0.31}$	$77.98^{\pm 0.40}$	$\mathbf{89.60}^{\pm \mathbf{0.16}}$
CIFAR-100 (Top-1)	$\mathbf{66.55}^{\pm \mathbf{0.45}}$	$42.90^{\pm 0.43}$	$\mathbf{66.23}^{\pm \mathbf{0.42}}$	$58.62^{\pm 0.20}$	$53.45^{\pm 0.38}$	$\mathbf{65.14}^{\pm \mathbf{0.29}}$
CIFAR-100 (Top-5)	$\mathbf{85.65}^{\pm \mathbf{0.12}}$	$70.01^{\pm 0.52}$	$84.10^{\pm 0.39}$	$85.09^{\pm 0.14}$	$80.48^{\pm 0.38}$	$\mathbf{88.60}^{\pm \mathbf{0.24}}$
VGG-9						
CIFAR-10	$88.80^{\pm 0.71}$	$76.40^{\pm 0.20}$	$\mathbf{90.04}^{\pm \mathbf{0.50}}$	$86.81^{\pm 0.09}$	$78.60^{\pm 0.30}$	$\mathbf{89.76}^{\pm \mathbf{0.20}}$
CIFAR-100 (Top-1)	$61.35^{\pm 0.76}$	$45.70^{\pm 0.14}$	$\mathbf{66.28}^{\pm \mathbf{0.29}}$	$\mathbf{60.65}^{\pm \mathbf{0.25}}$	$54.19^{\pm 0.41}$	$\mathbf{61.11}^{\pm \mathbf{0.45}}$
CIFAR-100 (Top-5)	$\mathbf{84.74}^{\pm \mathbf{0.40}}$	$73.04^{\pm 0.46}$	$\mathbf{84.96}^{\pm \mathbf{0.29}}$	$\mathbf{85.84}^{\pm \mathbf{0.15}}$	$80.65^{\pm 0.41}$	$85.14^{\pm 0.32}$
ResNet-18						
CIFAR-10	$\mathbf{92.17}^{\pm \mathbf{0.26}}$	“$53.74^{\pm 0.43}$”	$\mathbf{92.36}^{\pm \mathbf{0.12}}$	$\mathbf{91.73}^{\pm \mathbf{0.21}}$	“$43.19^{\pm 0.61}$”	$\mathbf{91.85}^{\pm \mathbf{0.24}}$
CIFAR-100 (Top-1)	$68.52^{\pm 0.34}$	“$22.83^{\pm 0.38}$”	$\mathbf{69.94}^{\pm \mathbf{0.54}}$	$69.47^{\pm 0.32}$	“$16.01^{\pm 0.42}$”	$\mathbf{71.46}^{\pm \mathbf{0.32}}$
CIFAR-100 (Top-5)	$86.86^{\pm 0.44}$	“$50.18^{\pm 0.52}$”	$\mathbf{87.76}^{\pm \mathbf{0.41}}$	$90.47^{\pm 0.12}$	“$40.67^{\pm 0.70}$”	$\mathbf{91.91}^{\pm \mathbf{0.23}}$

“...”: *ResNet-18 was unstable in our sPC experiments, so we copied the results from Pinchetti et al. (2025)*

5.1 Experimental Setup

We evaluated performance across four standard computer vision datasets: MNIST (LeCun, 1998; Cohen et al., 2017), FashionMNIST (Xiao et al., 2017), and CIFAR-10/100 (Krizhevsky, 2009). The architecture selection spanned an MLP, VGG-style convolutional networks of various depths (Simonyan and Zisserman, 2014), and a deep residual network (He et al., 2016). The output loss $\mathcal{L}$ is either Mean Squared Error (MSE) or Cross-Entropy (CE), again mirroring Pinchetti et al. (2025).

Complete implementation details, including hyperparameter settings, are provided in Appendix E.

An anonymized version of our codebase is available in the supplementary materials.

5.2 Results and Analysis

Our results in Table 1 confirm the significant performance gap between sPC and backprop previously reported by Pinchetti et al. (2025), while demonstrating that ePC substantially narrows this gap.

Several key findings emerge from our experiments:

- **Depth scaling:** ePC exhibits the expected performance improvement with increasing network depth, similar to backpropagation, whereas sPC performance degraded in deeper networks. This is most noticeable for ResNet-18, where ePC achieved competitive performance while sPC suffered from instability issues in our implementation.
- **Performance parity:** ePC nearly matches backpropagation's performance across most datasets and architectures, with results falling within statistical confidence intervals in many cases.
- **Loss function effects:** Both Mean Squared Error (MSE) and Cross-Entropy (CE) loss functions resulted in comparable performance across experimental settings, despite CE's typically superior gradient properties compared to MSE. However, we did observe greater sensitivity to hyperparameter selection with CE loss in both ePC and sPC algorithms.

Overall, our experimental results validate ePC's theoretical advantages. By resolving sPC's signal decay problem, ePC successfully scales PC to deeper architectures, unlocking its ability to handle substantially more complex machine learning challenges than previously possible.

6 Conclusion and Future Directions

This paper identifies and addresses a fundamental limitation in Predictive Coding networks: the exponential decay of signal propagation during state-based energy minimization. Our proposed error-based formulation overcomes this limitation by restructuring PC's computational graph while preserving theoretical equivalence, achieving dramatic performance improvements that finally establish PC as a competitive alternative to backpropagation for training deep neural networks.

6.1 Reinterpreting Predictive Coding as Minimal-Norm Perturbations

ePC provides a fresh perspective on PC's energy minimization process. Essentially, (e)PC searches for minimal-norm layerwise perturbations that collectively produce optimal outputs. At each layer, these corrections are added to the feedforward pass, incrementally refining the final output prediction. From these targeted state modifications, local weight learning rules can then be derived.

This reframing connects naturally with the Least-Control Principle (Meulemans et al., 2022), in which an external controller tries to minimally steer network activities to produce the target output. In their Appendix S4, they briefly explore PC through the lens of control theory, identifying the errors as an optimal control. With their framework allowing arbitrary controller circuits, it may be possible to find a biologically plausible implementation of ePC that does not explicitly require backpropagation, thereby addressing what some may consider essential for a PC algorithm.

6.2 Predictive Coding Beyond The Hardware Lottery

Algorithmic success is often dictated not by theoretical merit but by compatibility with prevailing hardware (Hooker, 2021). Serving as a prime example, PC has struggled to prove its worth despite theoretical soundness. To unlock its full potential, ePC reformulates PC in a way that aligns naturally with digital processors, relying on backpropagation to efficiently spread signals across deep networks. Meanwhile, sPC remains highly relevant for neuromorphic implementations, where physical energy minimization would occur naturally and near-instantaneously, regardless of network depth.

Despite their structural differences, both approaches still minimize the same energy function to reach identical equilibria. This functional equivalence creates a pragmatic research methodology: rather than being limited by sPC's digital inefficiency, researchers can turn to ePC for rapid prototyping, generating insights that remain valid for understanding bio-plausible PC learning mechanisms.

6.3 The Road Ahead for Predictive Coding

With PC's viability as a learning algorithm now firmly established, research must shift from proof-of-concept to practical impact. We highlight two research directions with great potential:

1. **Unblocking neuromorphic hardware development:** Despite its theoretical suitability for ultra-energy-efficient neuromorphic implementation, hardware development for PC has been scarce. A key obstacle is our limited understanding of PC's behavior at exact equilibrium—the regime to which any physical implementation would naturally settle. While a recent analysis of this setting identified improved learning capabilities (Innocenti et al., 2024b), our experiments consistently preferred hyperparameter configurations of approximate backpropagation, leaving little appeal to hardware developers. With ePC as an efficient tool to further study equilibrium dynamics, research can finally begin to address this critical barrier to neuromorphic advancement.
2. **Identifying PC's distinctive advantages:** Rather than competing with backpropagation in its domains of strength, research should focus on areas where PC uniquely excels. As Song et al. (2024) demonstrated with online and continual learning, such domains exist but remain under-explored. Although ePC's reliance on backpropagation puts an upper limit on PC's computational efficiency (as noted before in Zahid et al., 2023), few-step ePC could offer a compromise that maintains PC's unique properties while keeping training times practical.

With ePC effectively addressing PC's computational limitations on digital hardware, the field must now face its true test: demonstrating that Predictive Coding offers substantive advantages in specific domains, sufficient to justify its adoption over established approaches.

A limitations section is provided at the start of the appendix.

Reproducibility Statement

We took great care to ensure reproducibility, listing architectural details, hyperparameter sweep intervals, final values and even pseudorandom seeds, which can all be found in Appendix E. On the algorithmic level, Appendix A provides an extensive description of both sPC and ePC. Finally, we attached an anonymous version of our codebase to the supplementary materials.

References

Alonso, Nicholas, Jeffrey Krichmar, and Emre Neftci (2024). “Understanding and Improving Optimization in Predictive Coding Networks.” In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.10, pp. 10812–10820.

Bogacz, Rafal (2017). “A tutorial on the free-energy framework for modelling perception and learning.” In: *Journal of Mathematical Psychology* 76, pp. 198–211.

Cohen, Gregory et al. (2017). “EMNIST: Extending MNIST to handwritten letters.” In: *2017 international joint conference on neural networks (IJCNN)*. IEEE, pp. 2921–2926. DOI: `10.1109/IJCNN.2017.7966217`.

Friston, Karl and Stefan Kiebel (2009). “Predictive coding under the free-energy principle.” In: *Philosophical transactions of the Royal Society B: Biological sciences* 364.1521, pp. 1211–1221.

Ha, Myoung Hoon et al. (2025). *Towards Stable Learning in Predictive Coding Networks*. URL: `https://openreview.net/forum?id=FwdN0KovFp`.

He, Kaiming et al. (2016). “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Hendrycks, Dan and Kevin Gimpel (2016). “Gaussian Error Linear Units (GELUs).” In: *arXiv preprint arXiv:1606.08415*.

Hooker, Sara (2021). “The Hardware Lottery.” In: *Communications of the ACM* 64.12, pp. 58–65.

Hu, Wei, Lechao Xiao, and Jeffrey Pennington (2020). “Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks.” In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=rkgqN1SYvr`.

Innocenti, Francesco, El Mehdi Achour, and Christopher Buckley (2025). “μPC: Scaling Predictive Coding to 100+ Layer Networks.” In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. URL: `https://openreview.net/forum?id=lSLSzYuyfX`.

Innocenti, Francesco et al. (2024a). “JPC: Flexible Inference for Predictive Coding Networks in JAX.” In: *arXiv preprint arXiv:2412.03676*.

Innocenti, Francesco et al. (2024b). “Only Strict Saddles in the Energy Landscape of Predictive Coding Networks?” In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. URL: `https://openreview.net/forum?id=eTu6kvrkSq`.

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization.” In: *arXiv preprint arXiv:1412.6980*.

Kingma, Diederik P. and Max Welling (2013). “Auto-encoding variational Bayes.” In: *arXiv preprint arXiv:1312.6114*.

Krizhevsky, Alex (2009). “Learning multiple layers of features from tiny images.” In.

LeCun, Yann (1998). “The MNIST database of handwritten digits.” In: *http://yann. lecun. com/exdb/mnist/*.

Lee, Kwangjun, Cyriel M. A. Pennartz, and Jorge F. Mejias (Sept. 2025). “Cortical networks with multiple interneuron types generate oscillatory patterns during predictive coding.” In: *PLOS Computational Biology* 21.9, pp. 1–25.

Loshchilov, Ilya and Frank Hutter (2019). “Decoupled Weight Decay Regularization.” In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=Bkg6RiCqY7`.

Meulemans, Alexander et al. (2022). “The least-control principle for local learning at equilibrium.” In: *Advances in Neural Information Processing Systems* 35, pp. 33603–33617.

Millidge, Beren, Anil Seth, and Christopher L Buckley (2021). “Predictive coding: A theoretical and experimental review.” In: *arXiv:2107.12979*.

Millidge, Beren, Alexander Tschantz, and Christopher L Buckley (2022a). “Predictive coding approximates backprop along arbitrary computation graphs.” In: *Neural Computation* 34.6, pp. 1329–1368.

Millidge, Beren et al. (2022b). “Predictive Coding: Towards a Future of Deep Learning beyond Backpropagation?” In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. International Joint Conferences on Artificial Intelligence Organization, pp. 5538–5545.

Millidge, Beren et al. (Apr. 2024). “Predictive coding networks for temporal prediction.” In: *PLOS Computational Biology* 20.4, pp. 1–31.

Oliviers, Gaspard, Rafal Bogacz, and Alexander Meulemans (2024). “Learning probability distributions of sensory inputs with Monte Carlo Predictive Coding.” In: *bioRxiv*, pp. 2024–02.

Pinchetti, Luca et al. (2022). “Predictive Coding Beyond Gaussian Distributions.” In: *36th Conference on Neural Information Processing Systems*.

Pinchetti, Luca et al. (2025). “Benchmarking Predictive Coding Networks – Made Simple.” In: *The Thirteenth International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=sahQq2sH5x`.

Qi, Chang, Thomas Lukasiewicz, and Tommaso Salvatori (2025). “Training Deep Predictive Coding Networks.” In: *New Frontiers in Associative Memories*. URL: `https://openreview.net/forum?id=s3E08R4AMK`.

Salvatori, Tommaso et al. (2024). “Incremental Predictive Coding: A parallel and fully automatic learning algorithm.” In: *International Conference on Learning Representations*.

Salvatori, Tommaso et al. (2026). “A survey on neuro-mimetic deep learning via predictive coding.” In: *Neural Networks* 195, p. 108161. ISSN: 0893-6080.

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

Song, Yuhang et al. (2020). “Can the Brain Do Backpropagation? — Exact Implementation of Backpropagation in Predictive Coding Networks.” In: *Advances in Neural Information Processing Systems*. Vol. 33.

Song, Yuhang et al. (2024). “Inferring neural activity before plasticity as a foundation for learning beyond backpropagation.” In: *Nature Neuroscience*, pp. 1–11.

Tschantz, Alexander et al. (2023). “Hybrid predictive coding: Inferring, fast and slow.” In: *PLOS Computational Biology* 19.8, pp. 1–31.

Whittington, James C. R. and Rafal Bogacz (2017). “An approximation of the error backpropagation algorithm in a predictive coding network with local Hebbian synaptic plasticity.” In: *Neural Computation* 29.5.

Whittington, James C. R. and Rafal Bogacz (2019). “Theories of error back-propagation in the brain.” In: *Trends in Cognitive Sciences*.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). “Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms.” In: *arXiv preprint arXiv:1708.07747*.

Zahid, Umais, Qinghai Guo, and Zafeirios Fountas (2023). “Predictive coding as a neuromorphic alternative to backpropagation: a critical evaluation.” In: *Neural Computation* 35.12, pp. 1881–1909.

Zahid, Umais, Qinghai Guo, and Zafeirios Fountas (2024). “Sample as you Infer: Predictive Coding with Langevin Dynamics.” In: *Forty-first International Conference on Machine Learning*. URL: `https://openreview.net/forum?id=6VQXLUy4sQ`.

Appendix

Limitations

Our work demonstrates significant improvements in Predictive Coding efficiency and scalability. Nonetheless, certain limitations remain, which we discuss below.

Different optimization trajectories: Although ePC and sPC are mathematically equivalent at equilibrium, they follow distinct optimization trajectories. Therefore, ePC cannot be used for research on the intermediate state dynamics of sPC (e.g., Millidge et al., 2024; Lee et al., 2025). Furthermore, considering the abundance of local minima present in deep neural networks, it is, in theory, possible that ePC and sPC may converge to different equilibria, though we did not observe any evidence of this during our experiments, not even for very deep MLPs (see Appendix D.2).

Experimental scope: Following the established PC benchmark by Pinchetti et al. (2025), we tested exclusively on standard supervised learning tasks (MNIST, FashionMNIST, CIFAR) where backprop is known to perform exceptionally well. The goal of our experiments was solely to demonstrate ePC's superiority over sPC, not to prove PC superiority over backpropagation. It would be valuable to explore ePC in domains where PC might have advantages, such as online and continual learning (Song et al., 2024), to determine whether these benefits extend to deeper architectures (now possible with ePC) or were simply artifacts of sPC's poor signal propagation.

Clarifying Remarks

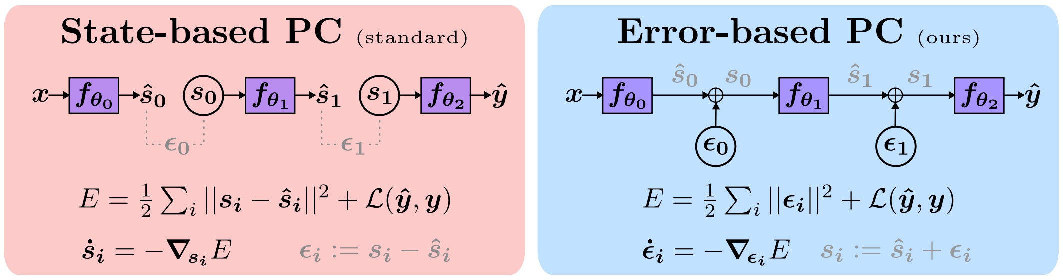
Certain aspects of our methodology may be misunderstood as limitations, but they instead reflect deliberate design choices and intrinsic advantages. For clarity, we highlight them here.

Choice of baseline methods: Our experiments compare ePC only against sPC and backpropagation (as neutral gold standard), rather than including a broader range of PC variants. This focused scope is intentional: Pinchetti et al. (2025) already extensively tested the most popular PC variants and found that none successfully scaled to deep networks (see their Table 1). Notably, our ResNet18 results (where ePC matches BP while all other PC methods failed entirely) demonstrate substantially stronger performance than any existing alternative. This improvement is expected, since all previous PC variants rely on the state-based formulation that we identified as fundamentally flawed (see Section 3). Therefore, comparing against the base sPC implementation, which is the most established and widely used variant, was the most appropriate choice.

Biological plausibility: While ePC preserves PC's core theoretical foundations and resulting weight updates, its use of backpropagation for the energy minimization process still makes it unsuitable for direct biological implementation. However, we'd like to emphasize that ePC was never intended as a biologically plausible algorithm. Instead, it serves as a computationally efficient tool for studying PC dynamics on digital hardware. Despite satisfying biological constraints, sPC proves impractical in digital simulations, as shown in Section 3. Biological and digital systems operate under fundamentally different constraints and mechanisms. By making PC practically viable on digital hardware, ePC actually enables more extensive research into PC dynamics, potentially offering greater value to computational neuroscience than the biologically constrained but computationally intractable sPC.

Computational efficiency: In a persistent misconception, sPC is often touted for its parallelization abilities (Millidge et al., 2022b; Pinchetti et al., 2022; Salvatori et al., 2024; Pinchetti et al., 2025). However, this alleged advantage is fundamentally flawed. Even with perfect parallelization, PC networks with depth L require at least L sequential update steps because signals can only advance one layer per step due to local-only interactions (Zahid et al., 2023). In fact, our experiments demonstrate this to be a very loose lower bound: sPC requires *exponentially* many update steps to reach equilibrium, fully undoing any potential (linear) speed-up from parallelization. Moreover, it may prove difficult to actually parallelize layers with different dimensions, forcing sequential processing in practice (Pinchetti et al., 2025, Section 6.1). As a result, our PyTorch implementation of ePC takes only 5-20% longer per step compared to sPC, despite its strictly sequential nature. This minor cost is easily offset by ePC's exponential reduction in required steps, representing a massive net gain in computational efficiency. Of course, these comparisons of digital implementations may ultimately be less relevant, as PC's true advantage lies in its suitability for ultra-fast neuromorphic hardware.

A Comparison of State-based vs. Error-based Predictive Coding


State-based PC (standard)
$x \to f_{\theta_0} \to \hat{s}_0 \quad s_0 \to f_{\theta_1} \to \hat{s}_1 \quad s_1 \to f_{\theta_2} \to \hat{y}$
ϵ_0
ϵ_1
$E = \frac{1}{2}\sum_i \|s_i - \hat{s}_i\|^2 + \mathcal{L}(\hat{y}, y)$
$\dot{s}_i = -\nabla_{s_i} E \qquad \epsilon_i := s_i - \hat{s}_i$
Error-based PC (ours)
$x \to f_{\theta_0} \to f_{\theta_1} \to f_{\theta_2} \to \hat{y}$
$\hat{s}_0 \quad s_0 \quad \hat{s}_1 \quad s_1$
ϵ_0
ϵ_1
$E = \frac{1}{2}\sum_i \|\epsilon_i\|^2 + \mathcal{L}(\hat{y}, y)$
$\dot{\epsilon}_i = -\nabla_{\epsilon_i} E \qquad s_i := \hat{s}_i + \epsilon_i$

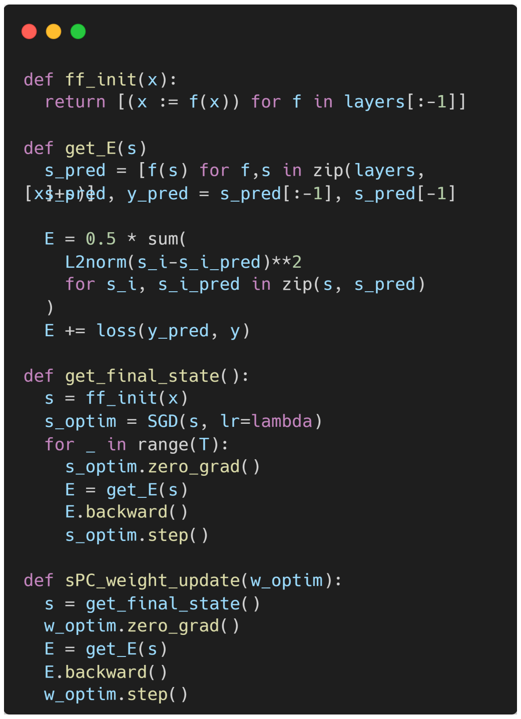
Algorithm 3: State-based PC *(standard)*

```
Require: Input $x$, target $y$, layers $\{f_{\theta_i}\}_{i=0}^{L}$,
    optimization steps $T$, state learning rate $\lambda$,
    weight learning rate $\eta$, output loss $\mathcal{L}$

    Feedforward state initialization (ff_init)
1:  $s_{-1} \leftarrow x$
2:  for $i = 0$ to $L-1$ do                              ▷ Sequential
3:      $\hat{s}_i \leftarrow f_{\theta_i}(s_{i-1})$
4:      $s_i \leftarrow \hat{s}_i$

    State updates
5:  for $t = 1$ to $T$ do
6:      for $i = 0$ to $L-1$ do                          ▷ Parallel
7:          $\hat{s}_i \leftarrow f_{\theta_i}(s_{i-1})$
8:          $\epsilon_i \leftarrow s_i - \hat{s}_i$
9:      $\hat{y} \leftarrow f_{\theta_L}(s_{L-1})$
10:     $E \leftarrow \frac{1}{2}\sum_{i=0}^{L-1} \|s_i - \hat{s}_i\|^2 + \mathcal{L}(\hat{y}, y)$
        Local energy gradients w.r.t. states
11:     $\epsilon_L \leftarrow \nabla_{\hat{y}} \mathcal{L}$
12:     for $j = 0$ to $L-1$ do                          ▷ Parallel
13:         $\nabla_{s_j} E \leftarrow \epsilon_j - \frac{\partial \hat{s}_{j+1}}{\partial s_j}^T \epsilon_{j+1}$
14:         $s_j \leftarrow s_j - \lambda \nabla_{s_j} E$

    Local weight update
15: for $j = 0$ to $L-1$ do                              ▷ Parallel
16:     $\nabla_{\theta_j} E \leftarrow -\frac{\partial \hat{s}_j}{\partial \theta_j}^T \epsilon_j$
17:     $\theta_j \leftarrow \theta_j - \eta \nabla_{\theta_j} E$
```

Algorithm 4: Error-based PC *(ours)*

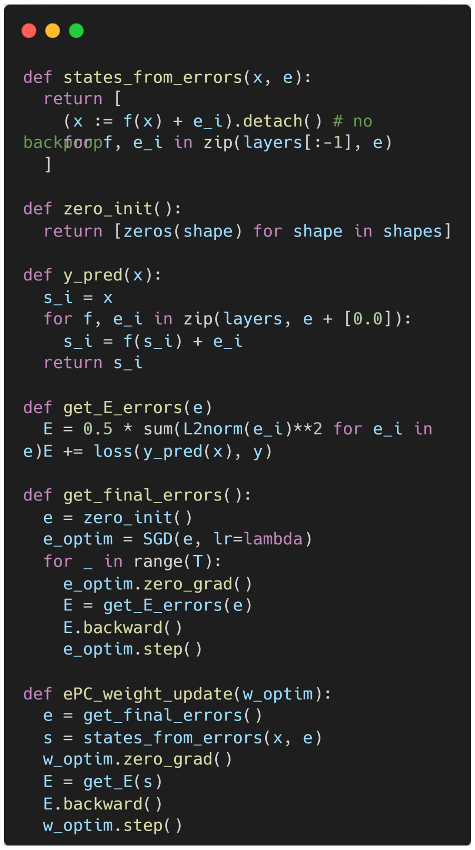
```
Require: Input $x$, target $y$, layers $\{f_{\theta_i}\}_{i=0}^{L}$,
    optimization steps $T$, error learning rate $\lambda$,
    weight learning rate $\eta$, output loss $\mathcal{L}$

    Zero error initialization (zero_init)
1:  for $i = 0$ to $L-1$ do                              ▷ Parallel
2:      $\epsilon_i \leftarrow \mathbf{0}$

    Error updates
3:  for $t = 1$ to $T$ do
4:      $s_{-1} \leftarrow x$
5:      for $i = 0$ to $L-1$ do                          ▷ Sequential
6:          $\hat{s}_i \leftarrow f_{\theta_i}(s_{i-1})$
7:          $s_i \leftarrow \hat{s}_i + \epsilon_i$
8:      $\hat{y} \leftarrow f_{\theta_L}(s_{L-1})$
9:      $E \leftarrow \frac{1}{2}\sum_{i=0}^{L-1} \|\epsilon_i\|^2 + \mathcal{L}(\hat{y}, y)$
        Global backprop of energy w.r.t. errors
10:     $\epsilon_L \leftarrow \hat{y}$
11:     for $j = L-1$ to $0$ do                          ▷ Sequential
12:         $\nabla_{\epsilon_j} \mathcal{L} \leftarrow \frac{\partial \epsilon_{j+1}}{\partial \epsilon_j}^T \nabla_{\epsilon_{j+1}} \mathcal{L}$
13:     for $j = 0$ to $L-1$ do                          ▷ Parallel
14:         $\nabla_{\epsilon_j} E \leftarrow \epsilon_j + \nabla_{\epsilon_j} \mathcal{L}$
15:         $\epsilon_j \leftarrow \epsilon_j - \lambda \nabla_{\epsilon_j} E$

    Local weight update
16: for $j = 0$ to $L-1$ do                              ▷ Parallel
17:     $\nabla_{\theta_j} E \leftarrow -\frac{\partial \hat{s}_j}{\partial \theta_j}^T \epsilon_j$
18:     $\theta_j \leftarrow \theta_j - \eta \nabla_{\theta_j} E$
```

Figure A.1: Full side-by-side comparison of state-based PC (left) and error-based PC (right)



```
def ff_init(x):
  return [(x := f(x)) for f in layers[:-1]]

def get_E(s)
  s_pred = [f(s) for f,s in zip(layers,
[x]+s)]
  s_pred, y_pred = s_pred[:-1], s_pred[-1]

  E = 0.5 * sum(
    L2norm(s_i-s_i_pred)**2
    for s_i, s_i_pred in zip(s, s_pred)
  )
  E += loss(y_pred, y)

def get_final_state():
  s = ff_init(x)
  s_optim = SGD(s, lr=lambda)
  for _ in range(T):
    s_optim.zero_grad()
    E = get_E(s)
    E.backward()
    s_optim.step()

def sPC_weight_update(w_optim):
  s = get_final_state()
  w_optim.zero_grad()
  E = get_E(s)
  E.backward()
  w_optim.step()
```

(a) State-based Predictive Coding



```
def states_from_errors(x, e):
  return [
    (x := f(x) + e_i).detach() # no
backprop
    for f, e_i in zip(layers[:-1], e)
  ]

def zero_init():
  return [zeros(shape) for shape in shapes]

def y_pred(x):
  s_i = x
  for f, e_i in zip(layers, e + [0.0]):
    s_i = f(s_i) + e_i
  return s_i

def get_E_errors(e)
  E = 0.5 * sum(L2norm(e_i)**2 for e_i in
e)E += loss(y_pred(x), y)

def get_final_errors():
  e = zero_init()
  e_optim = SGD(e, lr=lambda)
  for _ in range(T):
    e_optim.zero_grad()
    E = get_E_errors(e)
    E.backward()
    e_optim.step()

def ePC_weight_update(w_optim):
  e = get_final_errors()
  s = states_from_errors(x, e)
  w_optim.zero_grad()
  E = get_E(s)
  E.backward()
  w_optim.step()
```

(b) Error-based Predictive Coding

Figure A.2: PyTorch-style pseudocode comparison of sPC vs. ePC

B TEMPORAL EVOLUTION OF STATES IN STATE-BASED PC

This appendix extends our analysis of the exponential signal decay phenomenon identified in Section 3. While the main paper demonstrated how signals attenuate during the initial backward wavefront, we here derive a complete characterization of network dynamics that reveals the underlying mathematical structure governing signal propagation throughout energy minimization.

Our analysis uncovers a striking similarity to a simple binomial model, providing both theoretical insights into the discrete-time nature of the problem and practical understanding of why physical continuous-time implementations of sPC (like the brain) would not suffer from the same limitations.

B.1 SIMPLIFIED MODEL FOR ANALYTICAL TRACTABILITY

To enable rigorous mathematical analysis, we introduce a simplified model that captures the essential dynamics while remaining analytically tractable. Note that this setting provides only a coarse approximation to the true state dynamics of sPC, in contrast to our exact analysis of the initial backward wavefront in Section 3.

Key Assumption for Appendix B *After feedforward state initialization, all state predictions $\hat{\mathbf{s}}_i$ remain constant throughout energy minimization. This assumption implies that signal propagation occurs exclusively in the top-down direction, from output toward input layers.*

This simplification provides a reasonable approximation during early-stage optimization, where state dynamics are primarily driven by the output loss $\mathcal{L}$ before significant bottom-up signals emerge. However, it breaks down as the system evolves and predictions begin to change.

Simplified Backward Dynamics As described in Section 2, the temporal dynamics of states in sPC follow gradient descent on the energy function E with state learning rate λ:

$$\begin{aligned} \boldsymbol{s}_i^{t+1} &= \boldsymbol{s}_i^t - \lambda \boldsymbol{\nabla}_{\boldsymbol{s}_i} E^t \\ &= \boldsymbol{s}_i^t - \lambda \boldsymbol{\epsilon}_i^t + \lambda \boldsymbol{\epsilon}_{i+1}^t \, \frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}_{i+1}}}{\partial \boldsymbol{s}_i}(\boldsymbol{s}_i^t), \end{aligned}$$

where $\boldsymbol{\epsilon}_i := \boldsymbol{s}_i - \hat{\boldsymbol{s}}_i$ represents the layerwise prediction error. Given our key assumption above, this is equivalent to the deviation of each layer's state from its fixed prediction.

To further simplify our analysis, we set $\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}_{i+1}}}{\partial \boldsymbol{s}_i}(\boldsymbol{s}_i^t) \equiv \boldsymbol{I}$, reducing the dynamics to:

$$\boldsymbol{s}_i^{t+1} = \boldsymbol{s}_i^t - \lambda \boldsymbol{\epsilon}_i^t + \lambda \boldsymbol{\epsilon}_{i+1}^t$$

B.2 Recursive State Dynamics Beyond the Wavefront

Since state predictions $\hat{\boldsymbol{s}}_i$ remain fixed by assumption, the prediction errors $\boldsymbol{\epsilon}_i$ follow the same temporal dynamics as the states themselves:

$$\begin{aligned} \boldsymbol{s}_i^{t+1} &= \boldsymbol{s}_i^t - \lambda \boldsymbol{\epsilon}_i^t + \lambda \boldsymbol{\epsilon}_{i+1}^t \\ \iff (s_i^{t+1} - \hat{s}_i) &= (s_i^t - \hat{s}_i) - \lambda \boldsymbol{\epsilon}_i^t + \lambda \boldsymbol{\epsilon}_{i+1}^t \\ \iff \boldsymbol{\epsilon}_i^{t+1} &= \boldsymbol{\epsilon}_i^t - \lambda \boldsymbol{\epsilon}_i^t + \lambda \boldsymbol{\epsilon}_{i+1}^t \\ \iff \boldsymbol{\epsilon}_i^{t+1} &= (1-\lambda) \boldsymbol{\epsilon}_i^t + \lambda \boldsymbol{\epsilon}_{i+1}^t \end{aligned}$$

For small errors and/or learning rates, we can approximate the magnitude of the right-hand side as the sum of magnitudes, giving rise to recursive dynamics:

$$||\boldsymbol{\epsilon}_i^{t+1}|| \approx (1-\lambda)||\boldsymbol{\epsilon}_i^t|| + \lambda ||\boldsymbol{\epsilon}_{i+1}^t||$$

This recursive formula, when traced through the first few time steps, generates a striking pattern. Writing the magnitudes relative to the driving output gradient $\boldsymbol{\nabla}_{\hat{\boldsymbol{y}}} \mathcal{L}$:

Time	$t=0$	$t=1$	$t=2$	$t=3$		$t=0$	$t=1$	$t=2$	$t=3$
$\lvert\lvert\boldsymbol{\nabla}_{\hat{\boldsymbol{y}}}\mathcal{L}\rvert\rvert \propto$	1	$1-\lambda$	$1-2\lambda+\lambda^2$	$1-3\lambda+3\lambda^2-\lambda^3$		1	$(1-\lambda)$	$(1-\lambda)^2$	$(1-\lambda)^3$
$\lvert\lvert\boldsymbol{\epsilon}_{L-1}\rvert\rvert \propto$	0	λ	$2\lambda-2\lambda^2$	$3\lambda-6\lambda^2+3\lambda^3$	$=$	0	λ	$2\lambda(1-\lambda)$	$3\lambda(1-\lambda)^2$
$\lvert\lvert\boldsymbol{\epsilon}_{L-2}\rvert\rvert \propto$	0	0	λ^2	$3\lambda^2-3\lambda^3$		0	0	λ^2	$3\lambda^2(1-\lambda)$
$\lvert\lvert\boldsymbol{\epsilon}_{L-3}\rvert\rvert \propto$	0	0	0	λ^3		0	0	0	λ^3

The state at time $t=0$ follows from feedforward state initialization, where all internal errors begin at zero. By construction, every entry in the table equals the sum of $(1-\lambda)$ times its left neighbor (its previous value) and λ times its upper-left neighbor (influence from the layer above).

B.3 The Binomial Formula for Signal Propagation

Examining the coefficient patterns reveals a fundamental mathematical structure: Pascal's triangle. We can formalize this behavior with the following binomial formula:

$$||\boldsymbol{\epsilon}_{L-i}^t|| \propto \binom{t}{i} \lambda^i (1-\lambda)^{t-i}, \tag{6}$$

where L represents the total number of layers, i is the distance from the output layer, and t denotes the update step. The binomial coefficient $\binom{t}{i}$ encapsulates the number of possible paths through which a signal from the output layer can reach layer $L-i$ within exactly t update steps, given our top-down propagation assumption.

Aside from the initial signal $\boldsymbol{\nabla}_{\hat{\boldsymbol{y}}}\mathcal{L}$ at the output, the formula reveals three additional factors that influence signal magnitude throughout the network:

1. **Exponential depth decay λ^i:** confirms the exponential attenuation with network depth identified in Section 3. This explains PC's exponential energy decay across layers, as first observed by Pinchetti et al. (2025) and later reproduced by Qi et al. (2025).

2. **Temporal decay** $(1-\lambda)^{t-i}$**:** represents the gradual weakening of the original output signal over time, an artifact of our assumption that energy flows exclusively toward lower layers.

3. **Different propagation routes** $\binom{t}{i}$**:** accounts for the spatio-temporal variety of signal propagation pathways from output to the current layer.

B.4 State-based PC with High-Precision Simulation

The binomial formula of Eq. (6) serves as a powerful analytical tool to study sPC dynamics without the confounding effects of numerical precision limitations. By implementing this formula directly in logarithmic space using `scipy.special.gammaln`, we can achieve near-infinite precision and observe the theoretical behavior of signals in sPC unhindered by computational constraints.

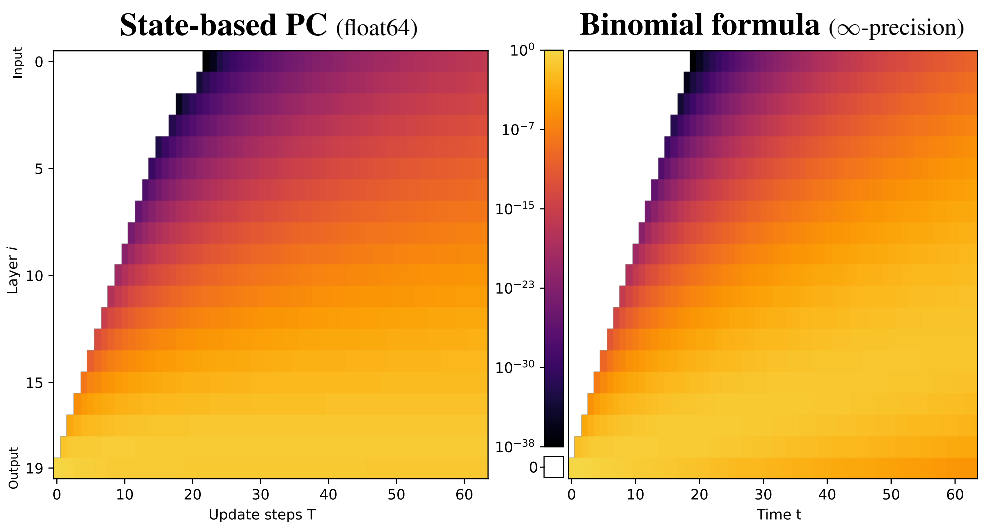


Figure B.1: Evolution of layerwise energies for sPC with float64 vs. near-infinite precision (simulated via Eq. (6)). Same setup as in Fig. 1, described in Appendix E.1.2.

Fig. B.1 presents a direct comparison between our high-precision binomial model and a float64 implementation of sPC for $\lambda = 0.1$. The striking similarity between these plots confirms that our mathematical characterization accurately captures the fundamental early-stage dynamics, despite simplifying assumptions. The orthogonal weight initialization used in our experiments certainly helps here, as it aligns well with our simplified backward dynamics assumption.

Comparing the float64 implementation in Fig. B.1 with the float32 version from Fig. 1 highlights both the importance and limitations of numerical precision in sPC. Even with enhanced double-precision floating-point arithmetic, the discontinuous signal propagation persists, though manifesting later and less pronounced.

B.5 Continuous vs. Discrete Time: Limitations of Digital Simulations of sPC

While it is clear that numerical precision is the cause of propagation issues in practice, the fundamental source of the exponential signal decay still remains unknown. We hypothesize:

The exponential signal decay identified in this paper is primarily an artifact of time discretization in digital implementations of sPC.

To analyze this claim rigorously, we examine our binomial formula (Eq. (6)) in the continuous-time limit, where $\lambda \to 0$ (infinitesimal state learning rate) and $t \to \infty$ (continuous updates), with a constant total time λt. Setting $t = \tau/\lambda$, we find:

$$\begin{aligned}
\lim_{\lambda\to 0} ||\boldsymbol{\Delta s}_{\boldsymbol{L-i}}^{t=\tau/\lambda}|| &\propto \lim_{\lambda\to 0} \binom{\tau/\lambda}{i} \lambda^i (1-\lambda)^{\tau/\lambda - i} \\
&\approx \lim_{\lambda\to 0} \underbrace{\frac{(\tau/\lambda)^i}{i!}}_{\textit{(Stirling's approximation)}} \lambda^i \underbrace{e^{-\tau}}_{\textit{(limit definition of e)}} \\
&= \frac{\tau^i}{i!} e^{-\tau}
\end{aligned}$$

In the continuous-time limit, our binomial formula transforms into a Poisson distribution, representing the spatial profile of a diffusion process. In this regime, signals diffuse smoothly over time, rather than being subject to the stepwise attenuation seen in discrete updates.

Digital implementations can approach continuous-time behavior with smaller time steps and more advanced ODE solvers, but this dramatically increases computational cost and reintroduces the original scalability problem. In contrast, physical substrates can effortlessly model a diffusion process, naturally operating at infinitesimal time constants, with countless update steps passing every second.

This observation leads to a reassuring conclusion: neuromorphic implementations of sPC, like the brain, would not suffer from the exponential signal decay problem identified in our research. The issue is specific to the time discretization required for digital implementations, subject to both stability constraints and limited numerical precision. In digital systems, we cannot practically approach the continuous-time limit without incurring prohibitive computational costs.

Therefore, the exponential decay with depth is not an inherent limitation of sPC as a theoretical framework but rather a consequence of its discretized formulation for digital hardware. For instance, biological neural systems, operating in continuous time with analog computation, would not struggle with the same fundamental barriers to depth scaling, although they may face other challenges, such as noise and non-idealities.

B.6 Temporal Scope and Limitations

Our binomial model primarily captures early-stage dynamics but becomes progressively less accurate for extended optimization periods. For longer time horizons, especially with larger learning rates, our assumption of fixed predictions becomes increasingly unrealistic. The model dictates permanent downward energy transmission, moving from output to input, due to a lack of bottom-up signals. In practical implementations, however, layerwise energies will settle across the network in an effort to minimize prediction errors globally. In particular, the output loss $\mathcal{L}$ will generally remain relatively large, even at equilibrium.

C Error-based PC and Connections to Other Algorithms

This section examines how ePC related to other established learning algorithms. More specifically, we show that:

- Despite its lack of locality, ePC is still a valid PC algorithm, according to the definition by Salvatori et al. (2026). (Appendix C.1)
- ePC and sPC are essentially the same PC algorithm, but follow different optimization paths. (Appendix C.2)
- Under specific conditions, ePC can implement exact backpropagation, closely resembling *Zero-divergent Inference Learning* (Z-IL; Song et al., 2020). (Appendix C.3)
- In general, however, ePC generates weight gradients that are different from those in standard backpropagation, matching those produced by full-equilibrium sPC. (Appendix C.4)
- When considering PC as a hierarchical probabilistic model, ePC simply implements the VAE reparametrization trick from Kingma and Welling (2013). (Appendix C.5)

C.1 ePC Broadens the Definition of Predictive Coding

This appendix section analyzes the technical definition of PC by Salvatori et al. (2026) and demonstrates that ePC meets all criteria, thereby definitively establishing it as an exact PC algorithm. Moreover, we show that ePC broadens our understanding of what PC is, eliminating the condition of local-only interactions, commonly believed to be a core component of PC.

Informal definition of PC by Salvatori et al. (2026), adapted to our notation. *Let us assume that we have a hierarchical generative model $g(\boldsymbol{x}, \boldsymbol{s})$, inverted using an algorithm $\mathcal{A}$. Then, $\mathcal{A}$ is a Predictive Coding algorithm if and only if:*

1. *it maximizes the model evidence $\log p(\boldsymbol{s})$ by minimizing a variational free energy,*
2. *the posterior distributions of the nodes of the hierarchical structure are factorized via a mean-field approximation, and*
3. *each posterior distribution is approximated under the Laplace approximation (i.e., random effects are Gaussian).*

Note that the above definition does not say anything explicitly about prediction error or properties such as locality, which, as mentioned earlier, are commonly used to describe PC. [...] the above definition is quite general and does not impose any constraint on the exact computation of the posteriors as well as the optimization technique(s) used to minimize the variational free energy.

Let us go over each of the requirements of the definition:

0. ePC employs the exact same hierarchical generative model $g(\boldsymbol{x}, \boldsymbol{s})$ as sPC, as reflected by its identical energy function (see Appendix C.2). A reparametrization does not affect this property.
1. ePC minimizes sPC's variational free energy E and reaches the same state minima (see Appendix C.2). As in sPC, this energy minimum corresponds to a maximum-likelihood estimation of the model evidence over the states.
2. ePC imposes a mean-field approximation of the posterior distribution. Concretely, this means that every state component $\boldsymbol{s}_{\boldsymbol{i}_j}$ can be set independently of any other state component. Although ePC builds a global computational graph, thereby imposing a dependence of $\boldsymbol{s_i}$ on all previous states $\boldsymbol{s_{<i}}$, we can still set $\boldsymbol{s_i}$ to any arbitrary value by modifying $\boldsymbol{\epsilon_i}$.
3. ePC's 'random effects' are the errors $\boldsymbol{\epsilon}$, which are implicitly modelled as Gaussians. More specifically, the energy E (representing $-\log g(\boldsymbol{x}, \boldsymbol{s})$) contains a $||\boldsymbol{\epsilon_i}||^2$ term that corresponds to the negative log-likelihood of a standard Gaussian $\mathcal{N}(\boldsymbol{0}, \boldsymbol{1})$. This is entirely analogous to the reparametrization trick in VAEs (Kingma and Welling, 2013), which formulates any Gaussian as a transformation of a standard normal. We explore this connection further in Appendix C.5.

Remarkably, requirement 2 does *not* imply the need for locality, despite Salvatori et al. (2026) noting that *"the mean field approximation enforces independence, and hence, results in locality in the update rules"*. ePC proves that a non-local mean-field approximation exists, without violating the assumptions of the hierarchical model.

Technically speaking, one might argue that the final node in our model, $\hat{\boldsymbol{y}}$, is not independent of the other nodes. However, the loss $\mathcal{L}$ still provides the necessary freedom to deviate from $\boldsymbol{y}$, effectively acting as an additional random effect. Along with the definition's assumption of Gaussian posterior distributions, the loss $\mathcal{L}$ becomes an MSE loss and may be equivalently modelled as an additional error term $\boldsymbol{\epsilon_L}$.

C.2 Theoretical Equivalence between sPC and ePC

Here, we provide a formal proof of the theoretical equivalence between the state- and error-based formulations of PC. We demonstrate that despite their different parameterizations, both approaches converge to identical equilibrium points and represent the same underlying optimization problem.

Bijective Mapping We first establish a bijective mapping between the optimization variables of sPC (the states $\boldsymbol{s}$) and ePC (the errors $\boldsymbol{\epsilon}$).

Theorem C.1 (Bijective Mapping)**.** *For any fixed set of parameters $\boldsymbol{\theta}$ and input $\boldsymbol{x}$, there exists a bijective mapping between any state configuration $\boldsymbol{s} = \{\boldsymbol{s_0}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_{L-1}}\}$ in sPC and error configuration $\boldsymbol{\epsilon} = \{\boldsymbol{\epsilon_0}, \boldsymbol{\epsilon_1}, \ldots, \boldsymbol{\epsilon_{L-1}}\}$ in ePC.*

Proof. Given states $\boldsymbol{s} = \{\boldsymbol{s_0}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_{L-1}}\}$ in sPC, we can directly compute the corresponding errors:

$$\boldsymbol{\epsilon_i} = \boldsymbol{s_i} - \boldsymbol{\hat{s}_i} = \boldsymbol{s_i} - \boldsymbol{f_{\theta_i}}(\boldsymbol{s_{i-1}}) \quad \text{for} \quad i \in \{0, 1, \ldots, L-1\}$$

where for $i = 0$, we define $\boldsymbol{s_{-1}} := \boldsymbol{x}$ (the input data).

Conversely, given errors $\boldsymbol{\epsilon} = \{\boldsymbol{\epsilon_0}, \boldsymbol{\epsilon_1}, \ldots, \boldsymbol{\epsilon_{L-1}}\}$ in ePC and input $\boldsymbol{x}$, we can recursively compute the corresponding states:

$$\begin{aligned} \boldsymbol{s_0} &= \boldsymbol{\hat{s}_0} + \boldsymbol{\epsilon_0} = \boldsymbol{f_{\theta_0}}(\boldsymbol{x}) + \boldsymbol{\epsilon_0} \\ \boldsymbol{s_i} &= \boldsymbol{\hat{s}_i} + \boldsymbol{\epsilon_i} = \boldsymbol{f_{\theta_i}}(\boldsymbol{s_{i-1}}) + \boldsymbol{\epsilon_i} \quad \text{for} \quad i \in \{1, 2, \ldots, L-1\} \end{aligned}$$

For a fixed set of parameters $\boldsymbol{\theta}$ and input $\boldsymbol{x}$, this mapping is one-to-one and onto (i.e., bijective): for any given $\boldsymbol{s}$, there is exactly one corresponding $\boldsymbol{\epsilon}$, and for any given $\boldsymbol{\epsilon}$, there is exactly one corresponding $\boldsymbol{s}$. □

Energy Function Equivalence Next, we prove that under this mapping, the energy functions of both formulations are equivalent.

Theorem C.2 (Energy Equivalence)**.** *Under the bijective mapping between $\boldsymbol{s}$ and $\boldsymbol{\epsilon}$, for any fixed parameter set $\boldsymbol{\theta}$ and input-output pair $(\boldsymbol{x}, \boldsymbol{y})$, the energy functions $E_{sPC}(\boldsymbol{s}, \boldsymbol{\theta})$ and $E_{ePC}(\boldsymbol{\epsilon}, \boldsymbol{\theta})$ are identical when evaluated on corresponding configurations.*

Proof. Let us first recall the energy functions for both formulations:

$$E_{\text{sPC}}(\boldsymbol{s}, \boldsymbol{\theta}) = \frac{1}{2} \sum_{i=0}^{L-1} \|\boldsymbol{s_i} - \boldsymbol{\hat{s}_i}\|^2 + \mathcal{L}(\boldsymbol{\hat{y}}, \boldsymbol{y})$$

$$E_{\text{ePC}}(\boldsymbol{\epsilon}, \boldsymbol{\theta}) = \frac{1}{2} \sum_{i=0}^{L-1} \|\boldsymbol{\epsilon_i}\|^2 + \mathcal{L}(\boldsymbol{\hat{y}}, \boldsymbol{y})$$

Starting with E_{ePC} and substituting the definition $\boldsymbol{\epsilon_i} = \boldsymbol{s_i} - \boldsymbol{\hat{s}_i}$:

$$\begin{aligned} E_{\text{ePC}}(\boldsymbol{\epsilon}, \boldsymbol{\theta}) &= \frac{1}{2} \sum_{i=0}^{L} \|\boldsymbol{\epsilon_i}\|^2 + \mathcal{L}(\boldsymbol{\hat{y}}, \boldsymbol{y}) \\ &= \frac{1}{2} \sum_{i=0}^{L} \|\boldsymbol{s_i} - \boldsymbol{\hat{s}_i}\|^2 + \mathcal{L}(\boldsymbol{\hat{y}}, \boldsymbol{y}) \\ &= E_{\text{sPC}}(\boldsymbol{s}, \boldsymbol{\theta}) \end{aligned}$$

Therefore, the energy functions evaluate to the same value for corresponding configurations of states and errors. □

Jacobian of the Transformation To analyze how gradients and critical points relate between the two formulations, we need the Jacobian matrix of the transformation from errors to states.

Lemma C.3 (Jacobian Structure)**.** *The Jacobian matrix $\boldsymbol{J} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{\epsilon}}$ representing how states change with respect to errors has a lower triangular structure with identity matrices on the diagonal.*

Proof. From the recursive definition of states in terms of errors:

$$\boldsymbol{s_i} = \boldsymbol{f_{\theta_i}}(\boldsymbol{s_{i-1}}) + \boldsymbol{\epsilon_i}$$

Taking partial derivatives with respect to $\boldsymbol{\epsilon_j}$:

1. If $j > i$: $\frac{\partial \boldsymbol{s_i}}{\partial \boldsymbol{\epsilon_j}} = \mathbf{0}$, since $\boldsymbol{s_i}$ doesn't depend on future errors.
2. If $j = i$: $\frac{\partial \boldsymbol{s_i}}{\partial \boldsymbol{\epsilon_i}} = \mathbf{I}$, the identity matrix.
3. If $j < i$: $\frac{\partial \boldsymbol{s_i}}{\partial \boldsymbol{\epsilon_j}} = \frac{\partial \boldsymbol{f_{\theta_i}}(\boldsymbol{s_{i-1}})}{\partial \boldsymbol{s_{i-1}}} \cdot \frac{\partial \boldsymbol{s_{i-1}}}{\partial \boldsymbol{\epsilon_j}}$

Let's denote $\mathbf{J}_i = \frac{\partial \boldsymbol{f_{\theta_{i+1}}}(\boldsymbol{s_i})}{\partial \boldsymbol{s_i}}$ as the Jacobian of layer $i+1$ with respect to its input (state i).

Then we can write:

$$\frac{\partial \boldsymbol{s_i}}{\partial \boldsymbol{\epsilon_j}} = \begin{cases} \mathbf{0} & \text{if } j > i \\ \mathbf{I} & \text{if } j = i \\ \mathbf{J}_{i-1} \cdot \frac{\partial \boldsymbol{s_{i-1}}}{\partial \boldsymbol{\epsilon_j}} & \text{if } j < i \end{cases}$$

This recursive structure leads to a lower triangular Jacobian matrix with identity matrices on the diagonal:

$$\mathbf{J} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{J}_0 & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{J}_1\mathbf{J}_0 & \mathbf{J}_1 & \mathbf{I} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \prod_{k=0}^{L-1} \mathbf{J}_k & \prod_{k=1}^{L-1} \mathbf{J}_k & \cdots & \mathbf{J}_{L-1} & \mathbf{I} \end{bmatrix}$$

This structure has important implications: $\mathbf{J}$ is invertible with determinant 1, since the determinant of a triangular matrix is the product of its diagonal entries, all of which are 1. □

Gradient Equivalence and Critical Points We now establish the relationship between gradients in both formulations and use it to prove that they share the same critical points.

Theorem C.4 (Gradient Relationship)**.** *The gradients of the energy functions in the sPC and ePC formulations are related by:*

$$\boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{ePC} = \mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{sPC}$$

where $\mathbf{J} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{\epsilon}}$ is the Jacobian matrix derived in Lemma C.3.

Proof. By the chain rule of calculus:

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{ePC}} &= \boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{sPC}}(\boldsymbol{s}(\boldsymbol{\epsilon}), \boldsymbol{\theta}) \\ &= \left(\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{\epsilon}}\right)^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} \\ &= \mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} \end{aligned}$$

□

Theorem C.5 (Critical Point Correspondence)**.** *A configuration $\boldsymbol{s}^*$ is a critical point of E_{sPC} if and only if the corresponding configuration $\boldsymbol{\epsilon}^*$ is a critical point of E_{ePC}.*

Proof. From Theorem C.4, we have:

$$\boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{ePC}} = \mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}$$

Since $\mathbf{J}$ is invertible (as shown in Lemma C.3), its transpose $\mathbf{J}^T$ is also invertible. Therefore:

$$\boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{ePC}} = \mathbf{0} \iff \mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} = \mathbf{0} \iff \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} = \mathbf{0}$$

This establishes that $\boldsymbol{s}^*$ is a critical point of E_{sPC} if and only if the corresponding $\boldsymbol{\epsilon}^*$ is a critical point of E_{ePC}. □

Local Structure of Critical Points To complete our proof of optimization equivalence, we need to show that the local structure of critical points (minima, maxima, or saddle points) is preserved between formulations.

Theorem C.6 (Preservation of Local Structure). *A critical point $\boldsymbol{s}^*$ is a local minimum / maximum / saddle point of E_{sPC} if and only if the corresponding critical point $\boldsymbol{\epsilon}^*$ is a local minimum / maximum / saddle point of E_{ePC}.*

Proof. The local structure of critical points is determined by the eigenvalues of the Hessian matrices:

$$\mathbf{H}_{\boldsymbol{s}} = \boldsymbol{\nabla}^2_{\boldsymbol{s}} E_{\text{sPC}}$$
$$\mathbf{H}_{\boldsymbol{\epsilon}} = \boldsymbol{\nabla}^2_{\boldsymbol{\epsilon}} E_{\text{ePC}}$$

To relate these Hessians, we differentiate the relationship in Theorem C.4:

$$\boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{ePC}} = \mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}$$

Taking another derivative with respect to $\boldsymbol{\epsilon}$:

$$\begin{aligned}
\boldsymbol{\nabla}^2_{\boldsymbol{\epsilon}} E_{\text{ePC}} &= \frac{\partial}{\partial \boldsymbol{\epsilon}} \left(\mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} \right) \\
&= \frac{\partial \mathbf{J}^T}{\partial \boldsymbol{\epsilon}} \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} + \mathbf{J}^T \frac{\partial \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}}{\partial \boldsymbol{\epsilon}} \\
&= \frac{\partial \mathbf{J}^T}{\partial \boldsymbol{\epsilon}} \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} + \mathbf{J}^T \boldsymbol{\nabla}^2_{\boldsymbol{s}} E_{\text{sPC}} \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{\epsilon}} \\
&= \frac{\partial \mathbf{J}^T}{\partial \boldsymbol{\epsilon}} \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} + \mathbf{J}^T \mathbf{H}_{\boldsymbol{s}} \mathbf{J}
\end{aligned}$$

At a critical point where $\boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} = \mathbf{0}$, the first term vanishes, giving:

$$\mathbf{H}_{\boldsymbol{\epsilon}} = \mathbf{J}^T \mathbf{H}_{\boldsymbol{s}} \mathbf{J}$$

This establishes that $\mathbf{H}_{\boldsymbol{\epsilon}}$ and $\mathbf{H}_{\boldsymbol{s}}$ are congruent matrices, considering $\mathbf{J}$ is invertible.

By Sylvester's law of inertia, congruent matrices have the same number of positive, negative, and zero eigenvalues. Therefore:

- $\mathbf{H}_{\boldsymbol{s}}$ is positive definite (all eigenvalues positive) if and only if $\mathbf{H}_{\boldsymbol{\epsilon}}$ is positive definite
- $\mathbf{H}_{\boldsymbol{s}}$ is negative definite (all eigenvalues negative) if and only if $\mathbf{H}_{\boldsymbol{\epsilon}}$ is negative definite
- $\mathbf{H}_{\boldsymbol{s}}$ has mixed positive/negative eigenvalues (saddle point) if and only if $\mathbf{H}_{\boldsymbol{\epsilon}}$ has the same eigenvalue signature

This preserves the classification of critical points as local minima, maxima, or saddle points between the two formulations. □

Dynamical Systems Analysis While the energy functions and their critical points are identical, the optimization dynamics differ significantly due to the reparameterization.

Theorem C.7 (Dynamical Equivalence). *The continuous-time dynamics in sPC and ePC both converge to the same equilibrium points, but follow different trajectories in their respective spaces.*

Proof. Under the notation $\dot{\boldsymbol{x}} := \frac{d\boldsymbol{x}}{dt}$, the continuous-time dynamics for both formulations are:

$$\begin{aligned}
\text{sPC:} \quad \dot{\boldsymbol{s}} &= -\boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} \\
\text{ePC:} \quad \dot{\boldsymbol{\epsilon}} &= -\boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{ePC}}
\end{aligned}$$

Using the relation $\boldsymbol{\nabla}_{\boldsymbol{\epsilon}} E_{\text{ePC}} = \mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}$, the ePC dynamics can be rewritten as:

$$\dot{\boldsymbol{\epsilon}} = -\mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}$$

To compare these dynamics in the same space, we need to transform $\dot{\boldsymbol{\epsilon}}$ to $\dot{\boldsymbol{s}}$. Using the chain rule:

$$\begin{aligned}\dot{\boldsymbol{s}} &= \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{\epsilon}}\dot{\boldsymbol{\epsilon}} \\ &= \mathbf{J}\dot{\boldsymbol{\epsilon}} \\ &= -\mathbf{J}\mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}\end{aligned}$$

Comparing with the sPC dynamics:

$$\begin{aligned}\dot{\boldsymbol{s}}_{sPC} &= -\boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} \\ \dot{\boldsymbol{s}}_{ePC} &= -\mathbf{J}\mathbf{J}^T \boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}}\end{aligned}$$

The difference is the matrix $\mathbf{J}\mathbf{J}^T$, which acts as a preconditioner for the gradient descent. This matrix is positive definite (since $\mathbf{J}$ has full rank), meaning that the ePC dynamics will always move in a descent direction for E_{sPC}, but with a different step size and direction than sPC.

Both dynamical systems will converge to the same equilibrium points where $\boldsymbol{\nabla}_{\boldsymbol{s}} E_{\text{sPC}} = \mathbf{0}$, but will follow different trajectories to get there. Whereas sPC suffers from an ill-conditioned optimization landscape (Innocenti et al., 2025), explaining its slow convergence, ePC seems to solve this problem through a cleverly constructed preconditioner. □

Global Connectivity and Signal Propagation The key computational advantage of ePC over sPC lies in its global connectivity structure. This difference affects how signals propagate through the network.

Theorem C.8 (Signal Propagation)**.** *In the sPC formulation, signals propagate sequentially through the network layers, resulting in exponential decay with network depth. In contrast, ePC allows direct signal propagation to all layers simultaneously, eliminating the signal decay problem.*

Proof. In sPC, the state update equations are:

$$\begin{aligned}\dot{\boldsymbol{s}_i} &= -\boldsymbol{\nabla}_{\boldsymbol{s}_i} E_{\text{sPC}} \\ &= -\boldsymbol{\epsilon_i} + \boldsymbol{\epsilon_{i+1}}\,\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}_{i+1}}}{\partial \boldsymbol{s_i}}(\boldsymbol{s_i})\end{aligned}$$

The crucial observation is that $\dot{\boldsymbol{s}_i}$ depends only on errors from adjacent layers ($\boldsymbol{\epsilon_i}$ and $\boldsymbol{\epsilon_{i+1}}$). This local connectivity means that a signal from the output layer ($\boldsymbol{\nabla}_{\hat{\boldsymbol{y}}}\mathcal{L}$) must propagate through all intermediate layers to reach the input layer, attenuating at each step.

In ePC, by contrast, the gradient is computed through the entire computational graph:

$$\begin{aligned}\dot{\boldsymbol{\epsilon}_i} &= -\boldsymbol{\nabla}_{\boldsymbol{\epsilon}_i} E_{\text{ePC}} \\ &= -\boldsymbol{\epsilon_i} - \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{\epsilon_i}}^T \boldsymbol{\nabla}_{\hat{\boldsymbol{y}}}\mathcal{L}\end{aligned}$$

$$\text{with} \qquad \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{\epsilon_i}} = \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{\epsilon_{L-1}}} \cdot \frac{\partial \boldsymbol{\epsilon_{L-1}}}{\partial \boldsymbol{\epsilon_{L-2}}} \cdot \ldots \cdot \frac{\partial \boldsymbol{\epsilon_{i+1}}}{\partial \boldsymbol{\epsilon_i}}$$

Hence, $\dot{\boldsymbol{\epsilon}_i}$ directly depends on all errors from layer i to L. This global connectivity allows signals from the output layer to immediately affect all earlier layers, eliminating the signal decay problem.

The mathematical consequence of this difference is that in sPC, the influence of the output error on layer i decreases exponentially with the distance from the output (as explained extensively in Section 3), while in ePC, this influence is direct and unattenuated. □

Limitations and Caveats While the two formulations are theoretically equivalent in terms of equilibrium points, several practical considerations affect their performance:

1. **Optimization Landscape**: The different parameterizations create different state trajectories that may encounter different local minima under stochastic optimization.

2. **Numerical Stability**: The formulations may exhibit different numerical properties, particularly with respect to hyperparameter sensitivity and discretization effects. For instance, in Section 3, the numerical issues of sPC are highlighted.
3. **Implementation Efficiency**: The global connectivity of ePC imposes different computational demands than the local connectivity of sPC, affecting implementation efficiency on different hardware architectures. On GPU, the backpropagation algorithm behind ePC is highly efficient, despite being sequential. However, the local and parallel nature of sPC enables a far more efficient neuromorphic implementation.

Despite these practical differences, our theoretical equivalence analysis confirms that ePC is a valid reparameterization of PC that preserves its fundamental principles while offering significant computational advantages for deep networks.

C.3 Exact Backpropagation using Error-based Predictive Coding

Below, we explore an important theoretical property of ePC: under specific conditions, ePC can become mathematically equivalent to standard backpropagation. This relationship deserves careful examination, as it affects how researchers should implement and interpret ePC results.

Note that, in general and under more reasonable circumstances, ePC does *not* equal backpropagation, and produces notably different weight gradients, as demonstrated in Appendix C.4.

C.3.1 When ePC Reduces to Backpropagation

The use of backpropagation within ePC's computational structure naturally raises the question of when the two methods become mathematically equivalent. We demonstrate that ePC reduces to standard backpropagation under specific conditions. Importantly, these conditions do *not* make backprop a PC algorithm, which would require, at least in theory, for the errors to be at equilibrium.

Theorem C.9. *ePC becomes mathematically equivalent to backpropagation when either:*

- *The number of update steps T is exactly 1.*
- *The error learning rate λ is sufficiently small relative to $1/T$.*

Proof. We consider each case separately, proving their equivalence to backpropagation.

Case 1: Single Update Step ($T = 1$)
With a single update step, the error variables are updated to:

$$\boldsymbol{\epsilon_i} = -\lambda \boldsymbol{\nabla}_{\boldsymbol{\epsilon_i}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

The subsequent parameter update becomes:

$$\begin{aligned}
\Delta \boldsymbol{\theta_i} &\propto -\frac{\partial \hat{\boldsymbol{s}}_{\boldsymbol{i}}}{\partial \boldsymbol{\theta_i}}^T \boldsymbol{\epsilon_i} \\
&= \lambda \frac{\partial \hat{\boldsymbol{s}}_{\boldsymbol{i}}}{\partial \boldsymbol{\theta_i}}^T \underbrace{\frac{\partial \boldsymbol{s_i}}{\partial \hat{\boldsymbol{s}}_{\boldsymbol{i}}}^T}_{=\boldsymbol{I}} \underbrace{\frac{\partial \boldsymbol{\epsilon_i}}{\partial \boldsymbol{s_i}}^T}_{=\boldsymbol{I}} \boldsymbol{\nabla}_{\boldsymbol{\epsilon_i}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\
&= \lambda \boldsymbol{\nabla}_{\boldsymbol{\theta_i}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})
\end{aligned}$$

This is precisely the gradient from standard backpropagation, but scaled by the error learning rate λ. Note that the weight update itself would involve an additional scaling by the *weight* learning rate.

When $\lambda = 1$, we find that this setup exactly matches that of *Zero-divergent Inference Learning* (Z-IL; Song et al., 2020). Z-IL adds a "fixed prediction assumption" to sPC and only models a backward signal wavefront, similar to that of Section 3, but with $\lambda = 1$. With these two constraints added to sPC, Z-IL effectively implements a 1-step version of ePC, which, as we outlined above, indeed corresponds to exact backpropagation.

Case 2: Small Learning Rate ($\lambda \ll 1/T$)
For a small λ, after T update steps, the error can be approximated as a linear accumulation of T identical updates:

$$\boldsymbol{\epsilon_i} \approx -\lambda T \boldsymbol{\nabla}_{\boldsymbol{\epsilon_i}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

Following the same reasoning as for case 1, the resulting parameter update now becomes:

$$\Delta\boldsymbol{\theta_i} \propto \lambda T \boldsymbol{\nabla}_{\boldsymbol{\theta_i}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

This approximation holds when λT remains sufficiently small, such that the error-perturbed output prediction $\hat{\boldsymbol{y}}$ still closely approximates the unperturbed feedforward prediction $\hat{\boldsymbol{y}}$ of backprop. Note that, when using larger learning rates and/or sufficient update steps, this is no longer the case, and the output prediction $\hat{\boldsymbol{y}}$ differs sufficiently between ePC and backprop, with the natural consequence being notably distinct gradients. $\square$

C.3.2 Experimental Considerations

In our experiments, we found that smaller values of λT generally performed best. However, we deliberately maintained this value above the threshold that would cause ePC to reduce to regular backpropagation. Complete experimental details are provided in Appendix E.

C.3.3 Contrast with sPC

This situation differs notably from sPC, which can also become equivalent to backpropagation under certain conditions (Song et al., 2020; Millidge et al., 2022a). However, these conditions typically involve specific algorithmic tweaks that rarely occur in practice, thereby protecting sPC implementations from accidentally reducing to backpropagation.

ePC, by contrast, presents a more subtle boundary. During hyperparameter tuning, one might inadvertently select learning rates and iteration counts that effectively transform ePC into standard backpropagation. Researchers working with ePC should therefore carefully monitor these parameters to ensure they are truly studying PC dynamics rather than rediscovering backprop in disguise.

C.4 Weight Gradients from (Error-Based) PC are Distinct from Backprop

In Section 4.3, we analyzed the evolution of states to equilibrium for a 20-layer linear PC network trained on MNIST. Here, we examine the evolution of the weight gradients themselves, which are ultimately the quantities of interest for learning.

While weight gradients in PC have no inherent dynamics (they are computed only after energy minimization), we can track how they would evolve if optimization were stopped at intermediate steps using their local formulas (Eq. (3)). The results are shown in Fig. C.1.

The analysis reveals several key insights about the gradient behavior of the two PC variants. Both ePC and sPC eventually converge to identical analytical PC weight gradients, reconfirming their theoretical equivalence from Appendix C.2. Notably, these PC gradients are distinct from backpropagation gradients across all layers, differing in both direction and magnitude.

Nonetheless, the gradient dynamics differ dramatically between the two methods. At $T = 1$, ePC starts at the backprop gradients (as shown in Appendix C.3) and rapidly transitions toward the PC solution. By contrast, sPC appears to transition from exact-zero gradients directly to PC.[2]

As expected, sPC suffers from severe signal propagation issues. Deeper layers stay at zero weight gradients for a long time, while layer 18 (the only initially non-zero gradient) actually becomes *more* aligned with backprop before slowly moving toward PC. This creates a significant risk, where early (pre-equilibrium) termination of sPC surely implements something other than true PC, despite achieving reasonable learning performance with extremely small yet informative gradients.

C.5 ePC implements the VAE reparametrization trick in PC

Although PC has long been described as a variational Bayes algorithm (Friston and Kiebel, 2009; Bogacz, 2017; Millidge et al., 2021), it is only recently that its probabilistic generative properties

[2]The directional wandering is likely just an artifact from the cosine similarity with a near-zero vector.

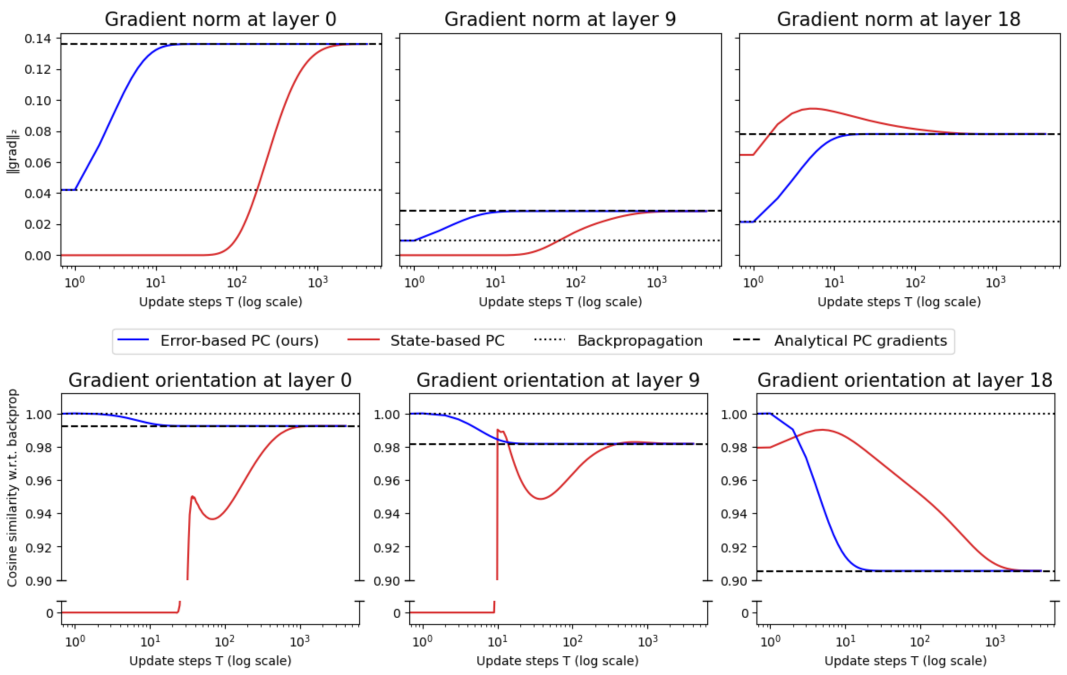


Figure C.1: Evolution of batch-averaged weight gradients of bottom, middle, and top hidden layers in a 20-layer linear PC network trained on MNIST (same setup as Fig. 5). ePC starts at backprop, while sPC starts mostly at zero. Both eventually reach the analytical PC gradients (notably distinct from backprop), with ePC converging roughly $100\times$ faster than sPC.

have been explored (Oliviers et al., 2024; Zahid et al., 2024). In this context, PC is seen as a hierarchical Gaussian graphical model, where:

- Layer predictions $\hat{s}_i$ represent the predicted means (μ) of Gaussian distributions (where negative log-likelihood corresponds to a squared error loss)
- Variance is typically fixed at $\sigma = 1$ (or precision weights are introduced)
- The states $\boldsymbol{s}$ are sampled from the predicted Gaussians: $s_i \sim \mathcal{N}(\hat{s}_i, 1)$
- Standard PC's "energy minimization" (as described in Section 2) corresponds to finding the maximum-likelihood states s instead of sampling from the full distribution

When ported to this setting, ePC becomes: $\quad s_i = \hat{s}_i + 1 \odot \epsilon_i \quad$ (where 1 represents unit variance)

A common problem in probabilistic graphical models is that direct sampling breaks the computational graph, inhibiting the gradient flow needed for backpropagation. One ingenious and highly successful solution is the VAE reparameterization trick (Kingma and Welling, 2013), which transforms a standard normal into the predicted distribution:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

Notice the strong similarity with the earlier formulation of ePC when $z \to s, \quad \mu \to \hat{s}_i, \quad \sigma \to 1$

Our description of ePC in Section 4.1 uses variational inference to find maximum-likelihood values for ϵ_i rather than drawing samples. However, when ported to the probabilistic interpretation of PC, one could sample the error variables ϵ_i from a standard normal distribution, making the connection to the VAE reparameterization trick more direct. The mathematical structure is entirely analogous: both methods reparameterize in terms of error/noise variables to enable efficient gradient flow.

D Proof-of-Concept: Deep PC Networks Trained on MNIST

This appendix describes the experimental setup from Section 4.3 and contains additional experiments on deep non-linear networks. The results demonstrate that ePC's convergence advantages hold across both linear and non-linear architectures, at least in our proof-of-concept MNIST setting.

D.1 Details of Deep Linear Network Trained on MNIST

Below, we briefly summarize the technical details needed to reproduce Fig. 5. Specifically, we used the following architecture:

- **Number of layers**: 20
  - Specifically: $\boldsymbol{x} - \boldsymbol{s_0} - \boldsymbol{s_1} - \cdots - \boldsymbol{s_8} - \boldsymbol{s_9} - \boldsymbol{s_{10}} - \cdots - \boldsymbol{s_{17}} - \boldsymbol{s_{18}} - \boldsymbol{y}$
    where '$-$' represents a layer (20 layers in total, leading to 19 hidden states $\boldsymbol{s_i}$)
- **Hidden state dim**: 128
- **Activation function**: None (not even at the output)
- **Weight init**: orthogonal (linear gain) (Hu et al., 2020)
- **Bias init**: zero
- **State/error optimizer**: SGD
- **Pretraining**
  - **Weight optimizer**: Adam (Kingma and Ba, 2014)
  - **Weight learning rate**: 0.001 (not tuned for this proof-of-concept)
  - **Gradient algorithm**: Backpropagation (fast, stable, and neutral w.r.t. sPC & ePC)
  - **Dataset**: EMNIST-MNIST (Cohen et al., 2017)
  - **Batch size**: 64
  - **Epochs**: 2
  - **Final test accuracy**: 84.5%

For a fair comparison between sPC and ePC, we tuned the internal learning rate for both, with the objective of maximum convergence to the analytical optimum:

- **ePC**
  - **e_lr sweep**: {0.001, 0.005, 0.01, 0.05, 0.1}
  - **Optimal e_lr**: 0.05
  - **#iters**: 256
- **sPC**
  - **s_lr sweep**: {0.05, 0.1, 0.3, 0.5}
  - **Optimal s_lr**: 0.3
  - **#iters**: 4096

The analytical solution was obtained via sparse matrix inversion using `scipy.sparse.linalg.spsolve`.

Note: Fig. 5 shows *state* dynamics for both sPC and ePC. To get states for the latter, we project from errors to states at every time step.

D.2 State Dynamics in Deep Non-Linear Models Trained on MNIST

Building on our analysis of linear models in Section 4.3, we extend our investigation to the more practical scenario of non-linear networks. This extension allows us to evaluate whether the signal propagation advantages of ePC generalize beyond the analytically tractable linear case.

D.2.1 Experimental Setup

We employed the 20-layer "Deep MLP" architecture detailed in Appendix E, pretrained on MNIST using one of two different loss functions: squared error and cross-entropy. Unlike the linear models, these non-linear networks achieve higher test accuracy (95% vs. 85%), representing a more realistic training scenario. However, this improved performance introduces an important methodological consideration: since analytical solutions are unavailable for non-linear models, we must turn to ePC's convergence state as our reference equilibrium point. This choice inherently favors ePC and should be considered when interpreting results.

D.2.2 Impact of Loss Function and Input Difficulty

As pointed out in Section 3, the signal decay problem of sPC depends critically on the output gradient $\nabla_{\hat{y}}\mathcal{L}$. In well-trained non-linear models, the loss—and hence its gradient—can become extremely small for easily classified examples, leading to even worse signal propagation issues. To investigate this effect systematically, we varied both the loss function (squared error vs. cross-entropy) and input difficulty (easy vs. hard-to-classify images). All experiments were implemented with float64 precision to ensure numerical stability and avoid precision-related confounders in the convergence analysis.

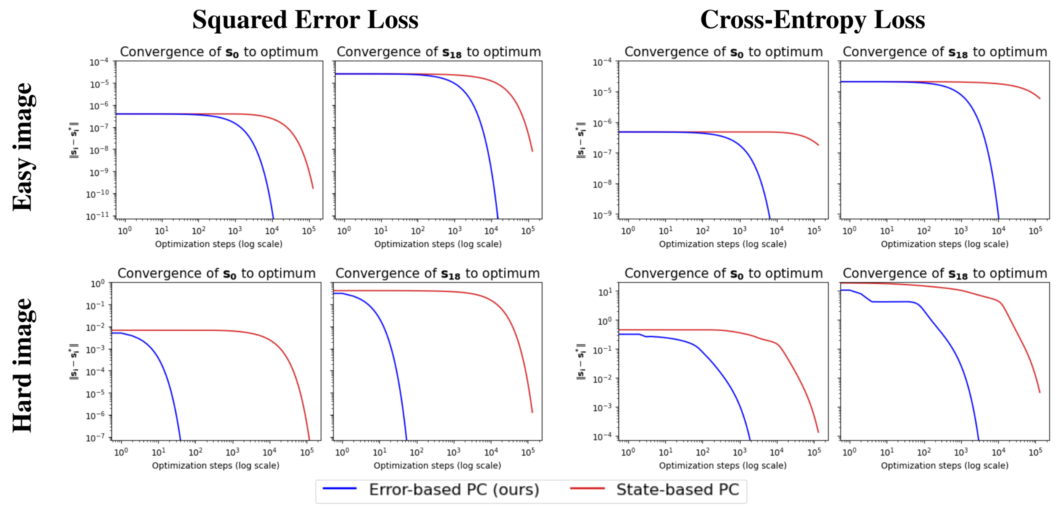


Figure D.1: Convergence dynamics in 20-layer PC-MLPs trained on MNIST with float64 precision. For easily-classified inputs (marked by near-zero loss), gradient signals become prohibitively small, hindering convergence. ePC consistently outperforms sPC by orders of magnitude across all conditions.

Fig. D.1 presents the convergence dynamics for both sPC and ePC across these conditions. Several important observations emerge from these experiments:

1. **Performance across loss functions:** Cross-entropy loss appears to create a more challenging optimization landscape for both sPC and ePC, despite its generally more favorable gradient properties compared to squared error. However, the smaller gradient signals from squared error do lead to long propagation delays in sPC, requiring almost 1000 update steps to progress just a single layer and reach s_{18}.
2. **Input difficulty effects:** For highly accurate models, most inputs will be "easy" (classified with high confidence), thereby generating minimal loss gradients. This greatly hinders overall signal propagation, even for ePC in our float64 simulations. By contrast, "hard" inputs (resulting in larger gradients) greatly accelerate ePC, while modestly improving sPC's convergence speed.
3. **Overall convergence speed:** ePC consistently converges orders of magnitude faster than sPC across all experimental conditions. This advantage is most pronounced with squared

error loss on hard images (bottom left panel), where ePC converges approximately 10,000 times faster than sPC for the exact same model.

4. **Identical equilibria:** Even in the non-linear case, sPC and ePC seem to head towards the same equilibrium points, highlighting their equivalence as established in Appendix C.2. Note that this is not necessarily true for all model architectures and datasets, as spurious local minima may affect sPC and ePC in different ways.
5. **Practical implications:** Perhaps most critically, sPC requires an impractical number of update steps (>100,000) to reach convergence in these non-linear networks, underscoring the practical importance of our ePC reformulation for deep PC architectures.

These findings extend and reinforce the convergence analysis presented in Section 4.3. They confirm that the advantages of ePC's global connectivity structure generalize from the analytically tractable linear case to practical non-linear networks with different loss functions.

E Overview of Experimental Implementation Details

In this appendix, we provide all details necessary to reproduce our experimental results from Table 1. Furthermore, we perform additional experiments on a 20-layer deep MLP architecture, which was used for Figs. 1 and D.1.

An anonymized version of our codebase is available in the supplementary materials.

E.1 MLPs on MNIST & FashionMNIST

Compute resources

- **CPU**: Intel Xeon E5-2620 v4
- **RAM**: 32 GiB
- **GPU**: NVIDIA GeForce GTX 1080 Ti
- **Compute time per experiment**: *(without early stopping or failure)*
  - **MNIST - MLP**:
    - **ePC**: 10min (#iters=4, 25 epochs) – 1h (#iters=256, 5 epochs)
    - **sPC**: 10min (#iters=4, 25 epochs) – 45min (#iters=256, 5 epochs)
    - **Backprop**: 2-7min
  - **MNIST - Deep MLP**:
    - **ePC**: 12min (#iters=4, 25 epochs) – 3h (#iters=256, 5 epochs)
    - **sPC**: 25min (#iters=4, 25 epochs) – 3h (#iters=256, 5 epochs)
    - **Backprop**: 2-8min
  - **FashionMNIST - MLP**:
    - **ePC**: 6min (#iters=4, 14 epochs) – 14min (#iters=64, 5 epochs)
    - **sPC**: 13min (#iters=16, 14 epochs) – 45min (#iters=256, 5 epochs)
    - **Backprop**: 3-7min
  - **FashionMNIST - Deep MLP**:
    - **ePC**: 20min (#iters=4, 17 epochs) – 45min (#iters=64, 5 epochs)
    - **sPC**: 1h (#iters=64, 7 epochs) – 5h30 (#iters=256, 13 epochs)
    - **Backprop**: 4-7min
- **Total compute time estimate**:
  - **MNIST**: $\pm$150h
  - **FashionMNIST**: $\pm$150h

Architecture

- **Number of layers**: 4 (MLP), 20 (Deep MLP; see below)
- **Hidden state dim**: 128

- **Activation function**: GELU (Hendrycks and Gimpel, 2016) (+ Sigmoid for MSE loss)
- **Weight init**: orthogonal (with ReLU gain) (Hu et al., 2020)
- **Bias init**: zero
- **Pseudorandom seed**: 42 for hyperparameter sweep, {0, 1, 2, 3, 4} for final test accuracy over 5 seeds. We set the seed using `lightning.seed_everything(workers=True)` before any data or weight initialization.
- **State/error optimizer**: SGD
- **Weight optimizer**: Adam (Kingma and Ba, 2014)

E.1.1 Additional Experiments on Deep MLPs

Table E.1: Additional test accuracies (in %) of ePC, sPC and backprop for a deep MLP architecture. Bold indicates best results within confidence intervals (mean ± 1 std. dev.; taken over 5 seeds).

Loss $\mathcal{L}$	*Mean Squared Error*			*Cross-Entropy*		
Training algorithm	ePC	sPC	Backprop	ePC	sPC	Backprop
Deep MLP (20 layers)						
MNIST	$97.11^{\pm 0.39}$	$96.89^{\pm 0.33}$	$\mathbf{97.89^{\pm 0.15}}$	$94.84^{\pm 0.54}$	$95.23^{\pm 1.24}$	$\mathbf{97.20^{\pm 0.07}}$
FashionMNIST	$\mathbf{85.04^{\pm 2.94}}$	$84.92^{\pm 0.40}$	$\mathbf{87.91^{\pm 0.45}}$	$81.37^{\pm 0.40}$	$79.95^{\pm 1.62}$	$\mathbf{87.78^{\pm 0.18}}$

As an addition to the benchmark of Pinchetti et al. (2025), we evaluated the performance for a 20-layer deep version of the MLP. As this architecture presents training challenges even with backpropagation, we turned to orthogonal weight initialization for enhanced stability (Hu et al., 2020). The results are stated in Table E.1.

Despite our expectation of a large performance gap, both ePC and sPC performed similarly. We hypothesize that orthogonal initialization may ease the signal decay problem by maintaining eigenvalues close to 1, thereby creating an unexpectedly strong baseline. This suggests that careful weight initialization can mitigate some of sPC's inherent optimization challenges, in line with findings from Innocenti et al. (2025). However, this strategy is less applicable to non-residual convolutional architectures, where the performance difference between ePC and sPC remains substantial.

To avoid confusion, we decided to move these results to appendix, rather than state them in Section 5.

E.1.2 Details of Figure 1

Below, we provide all details necessary to reproduce Fig. 1. Above all, our goal was to illustrate the problem of signal decay under realistic conditions.

The model is an untrained Deep MLP + Cross-Entropy, as detailed above. We track the layerwise energies throughout energy minimization for a single MNIST data pair $(\boldsymbol{x}, \boldsymbol{y})$. These are calculated using the energy functions corresponding to sPC and ePC (shown side-by-side in Fig. 2). We perform 64 update steps for sPC and 8 for ePC, both with a learning rate $\lambda = 0.1$.

E.1.3 MNIST

Data We used EMNIST-MNIST (Cohen et al., 2017), which is a well-documented reproduction of the original MNIST dataset (LeCun, 1998). The images are first rescaled to the range [0, 1], then they are normalized using the fixed values mean=0.5 and std=0.5 (same as Pinchetti et al., 2025). We set the batch size constant at 64. The validation set was 10% of the training data, split randomly but with a fixed seed. For final test performance, we don't split a separate validation set, but simply train on the whole training set.

ePC First, we did a hyperparameter sweep over the inner optimization hyperparameters (error learning rate (e_lr) and number of update steps (#iters)), with the weight learning rate (w_lr) constant at 3e-4 (or 3e-5 for Deep MLP+CE). During these sweeps, we train for 5 epochs. Then, we fixed the best inner optimization hyperparameters for each setting, and tuned w_lr and the number of epochs by means of early stopping, with a maximum of 25 epochs. See Table E.2 for more details of the sweep.

Table E.2: Hyperparameter sweep intervals and optimal values for ePC-MLPs on MNIST

Hyperparams	Sweep values	MLP+MSE	MLP+CE	Deep MLP+MSE	Deep MLP+CE
e_lr	{0.001, 0.005, 0.01, 0.05, 0.1, 0.5}	0.05	0.001	0.001	0.001
#iters	{4, 16, 64, 256}	4	4	4	4
w_lr	{1e-5, 3e-5, 5e-5, 1e-4, 3e-4, 5e-4, 1e-3}	1e-4	1e-4	1e-4	1e-5
#epochs	Early Stopping(patience=3), up to 25	25	20	14	25

sPC First, we did a hyperparameter sweep over the inner optimization hyperparameters (state learning rate (still denoted as e_lr for implementation purposes) and number of update steps (#iters)), with the weight learning rate (w_lr) constant at 1e-4 (the optimal rate for ePC). During these sweeps, we train for 5 epochs. Then, we fixed the best inner optimization hyperparameters for each setting, and tuned w_lr and the number of epochs by means of early stopping, with a maximum of 25 epochs. See Table E.3 for more details of the sweep.

Table E.3: Hyperparameter sweep intervals and optimal values for sPC-MLPs on MNIST

Hyperparams	Sweep values	MLP+MSE	MLP+CE	Deep MLP+MSE	Deep MLP+CE
e_lr	{0.01, 0.03, 0.1, 0.3}	0.01	0.03	0.3	0.3
#iters	{4, 16, 64, 256}	16	4	64	64
w_lr	{3e-5, 5e-5, 1e-4, 3e-4, 5e-4, 1e-3}	1e-4	1e-4	1e-4	5e-5
#epochs	Early Stopping(patience=3), up to 25	25	21	12	15

Backprop Since there is no inner optimization in backprop, we simply tuned w_lr and the number of epochs by means of early stopping, with a maximum of 25 epochs. See Table E.4 for all details.

Table E.4: Hyperparameter sweep intervals and optimal values for backprop-MLPs on MNIST

Hyperparams	Sweep values	MLP+MSE	MLP+CE	Deep MLP+MSE	Deep MLP+CE
w_lr	{3e-5, 5e-5, 1e-4, 3e-4, 5e-4, 1e-3}	3e-4	1e-4	3e-4	5e-5
#epochs	Early Stopping(patience=3), up to 25	16	20	22	18

E.1.4 FASHIONMNIST

Data We used the FashionMNIST dataset (Xiao et al., 2017). The images are first rescaled to the range [0, 1], then they are normalized using the fixed values mean=0.5 and std=0.5 (same as Pinchetti et al., 2025). We set the batch size constant at 64. The validation set was 10% of the training data, split randomly but with a fixed seed. For final test performance, we don't split a separate validation set, but simply train on the whole training set.

ePC First, we did a hyperparameter sweep over the inner optimization hyperparameters (error learning rate (e_lr) and number of update steps (#iters)), with the weight learning rate (w_lr) constant at 1e-4 (3e-5 for Deep MLP+CE). During these sweeps, we train for 5 epochs. Then, we fixed the best inner optimization hyperparameters for each setting, and tuned w_lr and the number of epochs by means of early stopping, with a maximum of 25 epochs. See Table E.5 for more details of the sweep.

Table E.5: Hyperparameter sweep intervals and optimal values for ePC-MLPs on FashionMNIST

Hyperparams	Sweep values	MLP+MSE	MLP+CE	Deep MLP+MSE	Deep MLP+CE
e_lr	{0.001, 0.003, 0.01, 0.03, 0.1}	0.01	0.003	0.001	0.001
#iters	{4, 16, 64}	16	4	4	4
w_lr	{3e-5, 5e-5, 1e-4, 3e-4}	5e-5	5e-5	3e-4	3e-5
#epochs	Early Stopping(patience=3), up to 25	12	14	17	2

sPC First, we did a hyperparameter sweep over the inner optimization hyperparameters (state learning rate (still denoted as e_lr for implementation purposes) and number of update steps (#iters)), with the weight learning rate (w_lr) constant at 1e-4. During these sweeps, we train for 5 epochs. Then, we fixed the best inner optimization hyperparameters for each setting, and tuned w_lr and the number of epochs by means of early stopping, with a maximum of 25 epochs. See Table E.6 for more details of the sweep.

Table E.6: Hyperparameter sweep intervals and optimal values for sPC-MLPs on FashionMNIST

Hyperparams	Sweep values	MLP+MSE	MLP+CE	Deep MLP+MSE	Deep MLP+CE
e_lr	{0.01, 0.03, 0.1, 0.3}	0.03	0.01	0.3	0.1
#iters	{4, 16, 64, 256}	64	16	64	256
w_lr	{3e-5, 5e-5, 1e-4, 3e-4}	1e-4	1e-4	5e-5	3e-5
#epochs	Early Stopping(patience=3), up to 25	14	14	7	/

Backprop Since there is no inner optimization in backprop, we simply tuned w_lr and the number of epochs by means of early stopping, with a maximum of 25 epochs. See Table E.7 for all details.

Table E.7: Hyperparameter sweep intervals and optimal values for BP-MLPs on FashionMNIST

Hyperparams	Sweep values	MLP+MSE	MLP+CE	Deep MLP+MSE	Deep MLP+CE
w_lr	{3e-5, 5e-5, 1e-4, 3e-4}	3e-4	1e-4	1e-4	3e-4
#epochs	Early Stopping(patience=3), up to 25	15	25	17	10

E.2 VGG-MODELS AND RESNET-18

Compute resources We report the resources used for training and each model's training time. The training times required for a model with MSE or CE loss are comparable. For hyperparameter tuning, we evaluate 200 distinct parameter configurations, making the total computational cost approximately 200 times greater than that of a single training run.

- **CPU**: Intel Xeon w5-3423
- **RAM**: 197 GiB
- **GPU**: NVIDIA RTX A6000
- **Compute time per experiment**: *(without early stopping or failure)*

Dataset	Model	ePC	sPC	Backprop
CIFAR-10	VGG-5	6min	9min	2min
	VGG-7	7min	11min	2min
	VGG-9	9min	17min	3min
	ResNet-18	29min	–	6min
CIFAR-100	VGG-5	6min	9min	2min
	VGG-7	7min	12min	2min
	VGG-9	9min	19min	3min
	ResNet-18	29min	–	6min

- **Total compute time estimate for tuning across model architecture and loss function**:
  - **ePC**: ±680h
  - **sPC**: ±510h
  - **Backprop**: ±170h

Data We used the CIFAR-10/100 datasets (Krizhevsky, 2009). The images are first rescaled to the range [0, 1], then they are normalized with the mean and standard deviation given in Table E.8 (same as Pinchetti et al., 2025). We set the batch size constant at 256. The validation set was 5% of the training data, split randomly but with a fixed seed. For final test performance, we don't split a separate validation set, but simply train on the whole training set.

VGG architecture VGG models are deep convolutional neural networks (Simonyan and Zisserman, 2014). Table E.9 provides a detailed summary of the model architectures used for the VGG-5, VGG-7 and VGG-9 models. After the convolutional layers, a single linear layer produces a class prediction. The activation function of the models was selected from among ReLU, Tanh, Leaky ReLU, and GELU (Hendrycks and Gimpel, 2016) during model tuning.

Table E.8: Data normalization

	Mean (μ)	**Std** (σ)
CIFAR-10	[0.4914, 0.4822, 0.4465]	[0.2023, 0.1994, 0.2010]
CIFAR-100	[0.5071, 0.4867, 0.4408]	[0.2675, 0.2565, 0.2761]

Table E.9: Detailed architectures of VGG models. The locations of the pooling layers correspond to the indices of the convolutional layers after which the max-pooling operations are applied.

	VGG-5	**VGG-7**	**VGG-9**
Channel Sizes	[128, 256, 512, 512]	[128, 128, 256, 256, 512, 512]	[128, 128, 256, 256, 512, 512, 512, 512]
Kernel Sizes	[3, 3, 3, 3]	[3, 3, 3, 3, 3, 3]	[3, 3, 3, 3, 3, 3, 3, 3]
Strides	[1, 1, 1, 1]	[1, 1, 1, 1, 1, 1]	[1, 1, 1, 1, 1, 1, 1, 1]
Paddings	[1, 1, 1, 1]	[1, 1, 1, 0, 1, 0]	[1, 1, 1, 1, 1, 1, 1, 1]
Pool location	[0, 1, 2, 3]	[0, 2, 4]	[0, 2, 4, 6]
Pool window	2 × 2	2 × 2	2 × 2
Pool stride	2	2	2

ResNet-18 architecture The ResNet-18 model is a convolutional neural network with skip connections (He et al., 2016). Our implementation follows the standard ResNet-18 architecture with modifications tailored for CIFAR-10/100. It is composed of an initial convolutional stem followed by four residual stages, each consisting of two residual blocks. Each residual block comprises two 3×3 convolutional layers with batch normalization and ReLU activation, followed by an identity shortcut connection. Spatial downsampling is performed via stride-2 convolutions at the beginning of each stage beyond the first. Table E.10 details the layer configuration.

Table E.10: ResNet-18 architecture adapted for CIFAR-10/100 image classification. The feature shape describes the image height and width after each stage. The residual configuration gives the dimension of the convolution mask, the number of channels and the stride used for the residual stream. All the convolutional layers used a padding of one, and each convolution was followed by a batch normalisation layer. Stages one to four include skip connections for every residual.

Stage	**Feature shape**	**Residual configuration**
Conv Stem	32×32	Conv3x3, 64, stride $= 1$
Stage 1	32×32	$\begin{bmatrix}\text{Conv3x3}, 64, \text{stride} = 1\\ \text{Conv3x3}, 64, \text{stride} = 1\end{bmatrix} \times 2$
Stage 2	16×16	$\begin{bmatrix}\text{Conv3x3}, 128, \text{stride} = 2\\ \text{Conv3x3}, 128, \text{stride} = 1\end{bmatrix} \times 2$
Stage 3	8×8	$\begin{bmatrix}\text{Conv3x3}, 256, \text{stride} = 2\\ \text{Conv3x3}, 256, \text{stride} = 1\end{bmatrix} \times 2$
Stage 4	4×4	$\begin{bmatrix}\text{Conv3x3}, 512, \text{stride} = 2\\ \text{Conv3x3}, 512, \text{stride} = 1\end{bmatrix} \times 2$
Head	1×1	Global AvgPool + Linear classifier

Learning rate schedule The following learning rate schedule was used to help stabilize training:

1. For the first 10% of training, the learning rate increases linearly from w_lr up to 1.1×w_lr.
2. After the warmup phase, a cosine decay is applied. The learning rate smoothly decreases to 0.1×w_lr, following a cosine curve, for the remaining training steps.

Weight initialization We used the default PyTorch weight initialization, which amounts to a random uniform weight and bias initialization. For pseudorandom seeds, we use 42 for the hyperparameter sweeps, and {0, 1, 2, 3, 42} for the final test accuracy over 5 seeds. We set the seed using `lightning.seed_everything(workers=True)` before any data or weight initialization.

Table E.11: Summary of hyperparameter tuning and training settings for convolutional models

Method	Tuned hyperparameter range	Optimizer	Optim steps (T)	Epochs (sweep/final)
ePC	e_lr: fixed at 0.001 e_momentum: fixed at 0.0 w_lr: log-uniform [1e-5, 1e-2] w_decay: log-uniform [1e-6, 1e-3]	SGD (error) Adam (weights) (Kingma and Ba, 2014)	5 (all models)	25/25 (VGG) 25/50 (ResNet-18)
sPC	s_lr: log-uniform [1e-3, 5e-1] s_momentum: {0, 0.25, 0.5, 0.75, 0.9} w_lr: log-uniform [1e-5, 1e-2] w_decay: log-uniform [1e-6, 1e-3]	SGD (state) AdamW (weights) (Loshchilov and Hutter, 2019)	8 (VGG-5) 10 (VGG-7) 12 (VGG-9)	25/25 (VGG)
Backprop	w_lr: log-uniform [1e-5, 1e-2] w_decay: log-uniform [1e-6, 1e-3]	Adam (weights)	—	25/25 (VGG) 25/50 (ResNet-18)

Glossary: w_lr: base weight learning rate (see learning rate schedule below), w_decay: weight decay, {e,s}_lr: error / state learning rate, {e,s}_momentum: error / state momentum, T: nr. of update steps

Hyperparameter tuning We performed hyperparameter tuning using Hyperband Bayesian optimization provided by Weights and Biases. The search was conducted over the hyperparameter spaces specified in Table E.11 across different model architectures, datasets, and loss functions. All tuning was guided by top-1 validation accuracy as the primary objective. Final top-5 accuracy metrics reported in Table 1 are for the models that achieved the highest top-1 accuracy. The best hyperparameters for each model identified through the sweep are provided in Table E.12, as well as in the "configs_results/" folder of the codebase. The "configs_sweeps/" folder contains all the sweep configs.

Table E.12: Overview of optimal hyperparameter configurations, used in our experiments

Data	Loss	Algo	Architecture	s/e_lr	s/e_momentum	w_lr	w_decay	act_fn
CIFAR-10	Squared Error	sPC	VGG-5	2.66e-2	0	4.21e-4	2.68e-6	gelu
			VGG-7	2.28e-3	0.05	2.07e-3	3.10e-6	gelu
			VGG-9	1.73e-2	0.5	5.77e-5	6.49e-4	tanh
		ePC	VGG-5	0.001	0	4.71e-4	1.48e-5	gelu
			VGG-7	0.001	0	4.26e-4	2.16e-6	gelu
			VGG-9	0.001	0	6.61e-4	4.01e-5	gelu
			ResNet-18	0.001	0	7.65e-4	1.82e-4	—
		BP	VGG-5	—	—	6.30e-4	1.09e-6	gelu
			VGG-7	—	—	5.45e-4	1.37e-6	gelu
			VGG-9	—	—	5.24e-4	1.27e-6	gelu
			ResNet-18	—	—	3.00e-4	9.04e-4	—
	Cross-Entropy	sPC	VGG-5	1.47e-2	0.05	2.64e-4	1.21e-5	gelu
			VGG-7	1.59e-3	0	1.76e-3	1.03e-5	gelu
			VGG-9	5.80e-2	0	8.09e-5	4.18e-5	tanh
		ePC	VGG-5	0.001	0	7.79e-4	1.72e-4	gelu
			VGG-7	0.001	0	1.56e-3	5.46e-4	gelu
			VGG-9	0.001	0	5.36e-4	6.88e-4	tanh
			ResNet-18	0.001	0	3.39e-3	1.51e-6	—
		BP	VGG-5	—	—	1.66e-3	4.55e-4	gelu
			VGG-7	—	—	1.10e-3	4.51e-5	gelu
			VGG-9	—	—	6.21e-4	3.58e-5	gelu
			ResNet-18	—	—	1.67e-3	1.49e-4	—
CIFAR-100	Squared Error	sPC	VGG-5	3.73e-3	0.75	9.80e-4	2.14e-6	gelu
			VGG-7	1.44e-2	0	1.88e-4	9.38e-5	tanh
			VGG-9	4.78e-2	0.25	7.07e-5	7.79e-5	tanh
		ePC	VGG-5	0.001	0	8.05e-4	2.33e-6	gelu
			VGG-7	0.001	0	4.02e-4	1.47e-5	gelu
			VGG-9	0.001	0	2.01e-4	2.62e-6	gelu
			ResNet-18	0.001	0	3.67e-4	7.30e-4	—
		BP	VGG-5	—	—	4.57e-4	1.27e-5	gelu
			VGG-7	—	—	4.47e-4	6.71e-6	gelu
			VGG-9	—	—	4.70e-4	4.34e-6	gelu
			ResNet-18	—	—	3.95e-4	5.45e-4	—
	Cross-Entropy	sPC	VGG-5	2.13e-2	0	8.61e-4	1.48e-6	tanh
			VGG-7	1.04e-1	0.5	3.00e-4	6.69e-5	tanh
			VGG-9	1.25e-2	0.75	4.69e-4	3.45e-4	tanh
		ePC	VGG-5	0.001	0	8.27e-4	8.22e-4	tanh
			VGG-7	0.001	0	3.13e-4	7.99e-4	tanh
			VGG-9	0.001	0	3.23e-4	4.03e-4	tanh
			ResNet-18	0.001	0	3.03e-3	1.20e-5	—
		BP	VGG-5	—	—	1.04e-3	7.69e-4	gelu
			VGG-7	—	—	1.38e-3	4.13e-4	gelu
			VGG-9	—	—	8.24e-4	1.62e-6	gelu
			ResNet-18	—	—	1.33e-3	1.96e-4	—

F Disclosure of LLM Usage

Large language models (LLMs) were used in this research for the following purposes:

- **Writing assistance**: Improving text clarity, flow, and conciseness throughout the paper
- **Theoretical contribution**: Proposing the preconditioner idea in Appendix C.2 and contributing to the theoretical analysis in that section

All other research ideas, methodological contributions, experimental design, and conclusions presented in this paper were conceived and developed by the authors. The creation of figures and implementation of code involved minimal LLM assistance.

Naturally, this section itself also benefited from LLM editing.