# TRAINING DEEP PREDICTIVE CODING NETWORKS

**Chang Qi**[1], **Thomas Lukasiewicz**[1], **Tommaso Salvatori**[1,2]
[1]Institute of Logic and Computation, Vienna University of Technology, Vienna, Austria
[2]VERSES AI Research Lab, Los Angeles, US

## ABSTRACT

Predictive coding models trained with equilibrium propagation are neural networks that perform inference through an iterative energy minimization process. Previous studies have demonstrated the effectiveness of this class of models in shallow architectures, but their performance degrades significantly as the depth increases to more than five/seven layers. In this study, we show that the reason behind this degradation is due to (1) errors between layers during weight updating, and (2) predictions from the previous layer not being effective in guiding updates in deeper layers. We address this by introducing both a novel weight update mechanism that reduces error accumulation in deeper layers, and a method to optimize the distribution of energy among layers during the 'relaxation phase'. Empirically, we show that our methods largely improve both training and test accuracy across networks with more than seven layers. These initial findings suggest that a better understanding of the relaxation phase is important to train models using equilibrium propagation at scale, and open new possibilities for their application in complex tasks.

## 1 INTRODUCTION

Training and inference in large-scale models is extremely expensive in terms of energy consumption, due to the large computational costs, making this technology not accessible to small businesses and academics, and prevent its use in low-power edge devices. As this is caused by the use of GPUs, general-purpose machines that are not specialized to perform machine learning tasks, a recent direction of research is studying the use of alternative accelerators, such as analog hardware that performs in-memory computations (Tsai et al., 2018; Haensch et al., 2018). One way of doing this is by training the model using equilibrium propagation, an algorithm that allows the learning of the parameters of a neural network by simulating a physical system brought to an equilibrium (Scellier & Bengio, 2017). This physical system is defined via an energy function that describes the state of a neural network in terms of its weights and neurons.

In the last years, researchers have put a large amount of effort into trying to make energy-based models work at scale. Two recent works have also carefully benchmarked multiple variations of the usual learning algorithms by using the Hopfield energy function (Scellier et al., 2024), and the predictive coding energy (Pinchetti et al., 2024), reaching similar results: this class of models is able to perform as well as standard deep learning models trained with backpropagation (BP) when it comes to train shallow models, with a maximum of five or seven layers. However, for predictive coding networks, the performance gap increases when considering deeper models: the test accuracy of backprop-based models increases, and that of energy-based models decreases. Understanding and addressing the causes of this mismatch would allow the training of large-scale energy-based models. Our contributions are briefly as follows:

- We empirically study the propagation of the energy on different layers of the model, and show that such energy is orders of magnitude larger in the layers that are closer to the label. We then conjecture that this creates two problems: the first is that the current training techniques fail to make full use of the depth of the model, as the update of the parameters is largely concentrated in a small number of layers; the second, is that the large magnitude of the energy in the last layers makes the latent states of the model diverge too much from the forward pass, leading to a sub-optimal weight update.

- To verify and address the aforementioned conjectures, we develop two variations of the learning algorithm. The first one regularizes the propagation of the total energy through the network by weighting the energy of different layers in different ways through the relaxation phase. The second one changes the way synaptic weights are updates, by using the activities of the neurons at initializations, and not at convergence. The results show that both of our method largely improve the performance of predictive coding networks in models with more than ten layers.

## 2 Related Works

**Equilibrium Propagation (EP).** EP is a learning algorithm for supervised learning that is largely inspired by contrastive learning on continuous Hopfield networks (Movellan, 1991). Here, neural activities are updated in two phases: In the first, to minimize an energy function defined on the parameters of the neural network; in the second, to minimize the same energy with the addition of a loss function defined on the labels (Scellier & Bengio, 2017). Interestingly, these two phases allow us to approximate the gradient of the loss function up to arbitrary levels of accuracy using finite difference coefficients (Zucchet & Sacramento, 2022). The consequence is that EP can be seen as a technique that allows minimizing loss functions using arbitrary physical systems that can be brought to an equilibrium, and it has hence been studied in a large number of domains (Scellier, 2024; Kendall et al., 2020). In terms of simulations that aim to scale up machine learning experiments, most of the works performed experiments using Hopfield energies (Hopfield, 1982), mostly on image classification tasks using convolutional networks (Laborieux et al., 2021; Laborieux & Zenke, 2022). The state-of-the-art is that EP models are able to match the performance of BPTT (BP-Through-Time) on models with $5$ hidden layers (Scellier et al., 2024), with the exception of hybrid models, that manage to reach good performances on models with $15$ layers by alternating blocks of layers trained with BP and blocks trained with EP Nest & Ernoult (2024).

**Predictive Coding (PC).** The formulation of PC that we use here was initially developed to model hierarchical information processing in the brain (Rao & Ballard, 1999; Friston, 2005). Intuitively, this theory states that neurons and synapses at one level of the hierarchy are updated to better predict the activities of the neurons of the layers below, and hence minimize the *prediction error*. Interestingly, the same algorithm can be used as a training algorithm for deep neural networks (Whittington & Bogacz, 2017), where several similarities with backpropagation were observed (Song et al., 2020; Salvatori et al., 2021). To this end, it has been used in a large number of machine learning tasks, from image generation and classification to natural language processing (Sennesh et al., 2024; Salvatori et al., 2023; Pinchetti et al., 2022; Ororbia & Kifer, 2020). Again, the state of the art has been reached by training convolutional models with $5$ hidden layers, with performance starting to get worse as soon as we use models that are 7 layers deep (Pinchetti et al., 2024).

## 3 Background

Let us consider a neural network with $L$ layers, and let us denote $\mathbf{W}^l$ and $\mathbf{x}_t^l$ the weight parameters and the neural activities of layer $l$, respectively. Note that, differently from standard models trained with backpropagation, the neural activities are random variables of the model, optimized over multiple time steps $t$. This optimization is performed with the goal of allowing the activities of every layer to predict those of the layer below. Together with the neural activities, the two other quantities related to single layers are the *prediction* $\mu_{\mathbf{t}}^{\mathbf{l}} = \mathbf{W^l f}\left(\mathbf{x_t^{l-1}}\right)$, given by the layer-wise operation through an activation function, and the *prediction error*, defined as the deviation of the actual activity from the prediction, that is, $\boldsymbol{\epsilon}_t^l = \mathbf{x}_t^l - \mu_t^l$. The predictive coding energy then is $E = \frac{1}{2}\sum_{l=1}^{L} \|\boldsymbol{\epsilon}^l\|^2$, that is the sum of the squared norms of the prediction errors of every layer.

Given a labelled data point $(\mathbf{o}, \mathbf{y})$, where $\mathbf{o}$ denotes sensory inputs and $\mathbf{y} \in \mathbb{R}^o$ is the label. Training is then performed via a form of bi-level optimization (Zucchet & Sacramento, 2022), divided into three phases. In the first phase, the neural activities of every neuron are initialized via a forward pass, that is, we set $\mathbf{x_0^l} = \mu_{\mathbf{0}}^{\mathbf{l}}$ for every layer, with $x_0^0 = o$. In the second phase, that we call the *inference* phase, we fix the neural activities of the output layer to the label, that is, we do $\mathbf{x}_L = \mathbf{y}$, and we update the neural activities via gradient descent, to minimize the total energy of the model.

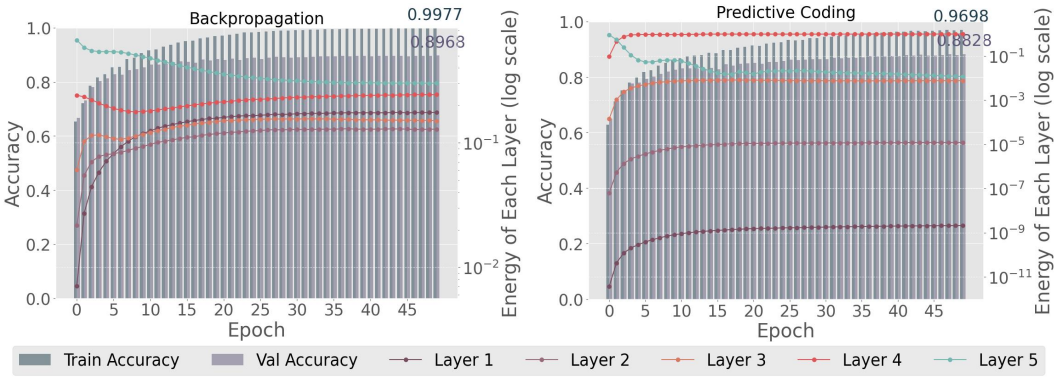



Figure 1: Layer-wise Energy Distribution and Accuracy Comparison between BP and PC in VGG5 on the CIFAR10 dataset. The colored lines represent the total energy of the individual layers of the model (or, the squared error of every layer for BP). The vertical lines represent the train and test accuracies of the model.

The update rule is then the following:

$$\Delta \mathbf{x}^l \propto -\frac{\partial E}{\partial \mathbf{x}^l} = -\boldsymbol{\epsilon}^l + \mathbf{W}^{(l+1)\top} \boldsymbol{\epsilon}^{l+1} \odot f'(\mathbf{x}^l). \tag{1}$$

This phase will continue until it reaches the fixed number of iterations $\mathbf{T}$ or achieves convergence. The third phase is the *learning* phase, where the neural activities $\mathbf{x}_T^l$ are fixed, and the weight parameters are updated to decrease the energy via the following equation:

$$\Delta \mathbf{W}^l \propto \frac{\partial E}{\partial \mathbf{W}^l} = -\boldsymbol{\epsilon}^l f(\mathbf{x}^{l-1}). \tag{2}$$

**Nudging.** Instead of providing the original label $y$ to the model, it is common in the literature to slightly translate the output neurons of the system $x_0^L$ in the direction of $y$. More precisely, it fixes $x_t^L = \mu_0^L + \beta(y - \mu_0^L)$ for every time step $t$, where $\beta$ controls the supervision strength. The sign of $\beta$ determines supervision polarity: positive for standard nudging and negative for inverse supervision. In the case of inverse supervision, the weight update is then performed to *increase* the loss function, allowing the model to be updated in the direction of the label. These two forms of weak supervision are called positive and negative nudging, respectively. Alternating $\beta$ values across training epochs enables center nudging. Empirically, the best test accuracies have been reached by performing a stochastic sampling from $\{\beta, -\beta\}$ across training epochs and batches, implementing a technique called *center nudging* (Scellier et al., 2024). Due to its success in practical tasks, we will consider center nudging as a baseline.

## 4 METHODS

In this section we first discuss the energy imbalance across different levels of the network, and then present two novel optimization methods: the first allows us to better distribute the energy across the model; the second proposes the use of the neural activities at initialization to perform a better update of the parameters.

To study the energy imbalance across different levels of the network, we have tracked the total energy of each layer during training, along with the test and training loss, and compared it against that of BP. As BP does not have a proper definition of energy, we have used the squared error of every neuron computed during the backward pass. We have tested this on models of multiple depths trained on the CIFAR10 dataset. Our analysis revealed a significant energy imbalance in the energy distribution across layers: As shown in Figure 1, the BP-trained model exhibits a more uniform energy distribution across layers, with even the layer of lowest energy above $10^{-1}$. In contrast, the PC-trained model shows that the significant energy disparities among the different layers, final layer's energy drops dramatically to as low as $10^{-8}$. The bar chart displays the training and test accuracy for each epoch, while the line plot shows the energy distribution across layers, showing that despite this energy imbalance, shallow PC models are able to reach test accuracies as good as those of BP. This, however, can be problematic when training deeper models.

**Energy Decay.** To address the issue of imbalanced energy distribution, which prevents the model from fully using its depth, we introduce an energy weighting strategy. Here, the energy of different layers decays exponentially over time, as defined as follows:

$$E_t = \frac{1}{2}\sum_{l=1}^{L} A_t^l \left(\epsilon_t^l\right)^2, \qquad \text{with } \mathbf{A}_t^l = \begin{cases} \frac{e^{-k\cdot(l-L+t)}}{\sum_{j=0}^{T-L-1+l} e^{-k\cdot j}}, & \text{when } t \geq L-l, \\ 0, & \text{when } t < L-l. \end{cases} \tag{3}$$

Here, $A_t^l$ represents the relative weight of energy at layer $l$ during iteration $t$, and $k$ is a hyperparameter that controls the decay rate ($k = 1$ in our experiments). When $t < L - l$, We set $A_t^l = 0$ as backward signals have not propagated to layer $l$ (corresponding $\epsilon_t^l = 0$). The $\sum_{j=0}^{T-L-1+l} e^{-k\cdot j}$ is a normalization term, it ensures the $\sum_{t=0}^{T-1} A_t^l = 1$. The numerator $e^{-k\cdot(l-L+t)}$ implements an intrinsic bias that lower layers (smaller $l$) automatically receive larger weights when activated ($t \geq L-l$), thereby helping to achieve a more balanced energy distribution during the inference phase.

**Forward Updates.** Due to large prediction errors that we find in the last layers, the neural activities observed at the end of the inference process tend to significantly deviate from their initial feedforward values. But the feedforward values are the ones that are then used for predictions. We then conjecture that synaptic weight updates based on $\mathbf{x}_T^l$ could potentially introduce errors that accumulate with network depth, leading to performance degradation in deeper architectures. To asses whether the proposed conjecture is correct, we introduce a new method for updating the weight parameters, that uses both the starting and final states of neurons, according to the energy function defined as

$$E_{T_0} = \frac{1}{2}\sum_{i,\ell} \left(\epsilon_{i,T_0}^{\ell}\right)^2, \qquad \text{where} \quad \epsilon_{i,T_0}^{\ell} = x_{i,T}^{\ell} - \mu_{i,0}^{\ell}. \tag{4}$$

Our method makes sure weight adjustments stay connected to the initial feed-forward predictions while incorporating the refined representations obtained through iterative inference. This approach helps maintain stability during learning and prevents the accumulation of errors in deeper layers, which is crucial for scaling PC networks. The pseudocode is provided in Algorithm 1.

**Algorithm 1** Learning a dataset $\mathcal{D} = \{x_i, y_i\}$ with proposed Deep PCNs

**Require:** $\mathbf{x}^0$ is fixed to $x_i$ and $\mathbf{x}^L$ is fixed to $y_i$ for every mini-batch

```
for t = 0 to T do
    for every neuron i and level ℓ do
        Update x_i^ℓ at t to minimize E_t.
    end for
    if t = T then
        Update W_i^ℓ to minimize E_{T_0}.
    end if
end for
```

## 5 EXPERIMENTS

In this section, we test our proposed method combined on models with more than 7 layers, and show that we are able to reach performance comparable to those of BP when trained on the same models. To provide a comprehensive evaluation, we test them on three popular computer vision benchmarks, that are CIFAR-10/100 (Krizhevsky et al., 2009), and Tiny ImageNet (Le & Yang, 2015). As models, we use VGG-like models (Simonyan & Zisserman, 2014) — convolutional models with one linear layer at the end — of increasing depths. All the experiments have been performed using mean squared error for consistent comparisons. We consider the following three baselines: standard PC (described in Section 3), PC with center nudging (PC-CN), the best performing algorithm according to previous studies (Pinchetti et al., 2024; Scellier et al., 2024), and standard BP[1]. We call our proposed method, developed to train deep models DPC, and also test a variation trained with with center nudging, named DPC-CN. The details needed to reproduce the results can be found in the supplementary material.

[1]Here we follow the standard PC literature, where performance are compared against BP, and differ from standard works in the Eqprop literature, that use BPTT as a baseline.

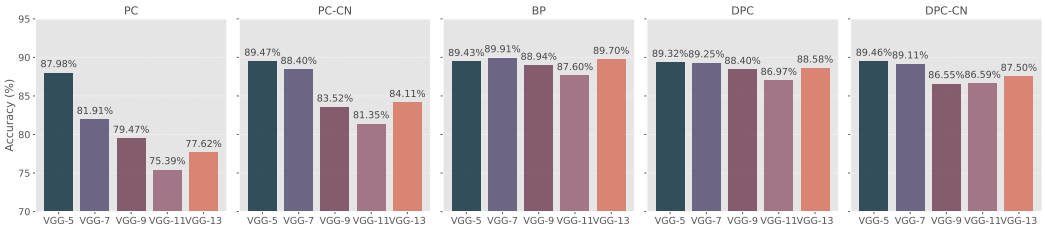



Figure 2: Test accuracies of different algorithms on the CIFAR10 dataset, tested on models of different depths.

Table 1: Test accuracies of the different algorithms on different datasets.

| **Dataset** | **Algorithm** | **VGG5** | **VGG7** | **VGG9** | **VGG11** | **VGG13** |
|---|---|---|---|---|---|---|
| CIFAR10 | BP | $89.43^{\pm 0.12}$ | $\mathbf{89.91^{\pm 0.1}}$ | $\mathbf{88.94^{\pm 0.08}}$ | $\mathbf{87.6^{\pm 0.12}}$ | $\mathbf{89.70^{\pm 0.11}}$ |
| | DPC | $89.32^{\pm 0.14}$ | $89.25^{\pm 0.09}$ | $88.40^{\pm 0.18}$ | $86.97^{\pm 0.10}$ | $88.58^{\pm 0.07}$ |
| | DPC-CN | $\mathbf{89.46^{\pm 0.14}}$ | $89.11^{\pm 0.26}$ | $86.55^{\pm 0.81}$ | $86.59^{\pm 0.23}$ | $87.50^{\pm 0.61}$ |
| CIFAR100 (Top-1) | BP | $66.28^{\pm 0.23}$ | $\mathbf{65.36^{\pm 0.15}}$ | $\mathbf{65.07^{\pm 0.23}}$ | $\mathbf{62.29^{\pm 0.16}}$ | $\mathbf{63.08^{\pm 0.18}}$ |
| | DPC | $66.10^{\pm 0.09}$ | $64.86^{\pm 0.10}$ | $63.23^{\pm 0.16}$ | $59.23^{\pm 0.37}$ | $60.19^{\pm 0.28}$ |
| | DPC-CN | $\mathbf{66.85^{\pm 0.07}}$ | $63.89^{\pm 0.22}$ | $60.59^{\pm 0.47}$ | $60.56^{\pm 0.22}$ | $61.27^{\pm 0.13}$ |
| CIFAR100 (Top-5) | BP | $85.85^{\pm 0.27}$ | $84.41^{\pm 0.26}$ | $\mathbf{82.69^{\pm 0.23}}$ | $\mathbf{83.22^{\pm 0.18}}$ | $\mathbf{84.13^{\pm 0.18}}$ |
| | DPC | $\mathbf{85.85^{\pm 0.10}}$ | $\mathbf{84.55^{\pm 0.20}}$ | $81.24^{\pm 0.33}$ | $81.08^{\pm 0.24}$ | $81.47^{\pm 0.30}$ |
| | DPC-CN | $85.85^{\pm 0.13}$ | $83.80^{\pm 0.20}$ | $80.95^{\pm 0.55}$ | $82.98^{\pm 0.07}$ | $83.30^{\pm 0.19}$ |
| TinyImageNet (Top-1) | BP | $44.9^{\pm 0.19}$ | $\mathbf{46.08^{\pm 0.15}}$ | $\mathbf{52.05^{\pm 0.18}}$ | $\mathbf{50.38^{\pm 0.12}}$ | $\mathbf{48.3^{\pm 0.42}}$ |
| | DPC | $\mathbf{45.98^{\pm 0.18}}$ | $44.07^{\pm 0.39}$ | $42.01^{\pm 0.11}$ | $48.80^{\pm 0.14}$ | $41.79^{\pm 0.50}$ |
| | DPC-CN | $43.99^{\pm 0.23}$ | $43.87^{\pm 0.45}$ | $42.03^{\pm 0.24}$ | $48.64^{\pm 0.31}$ | $41.13^{\pm 1.20}$ |
| TinyImageNet (Top-5) | BP | $65.26^{\pm 0.37}$ | $\mathbf{66.65^{\pm 0.2}}$ | $\mathbf{73.02^{\pm 0.14}}$ | $70.21^{\pm 0.12}$ | $\mathbf{71.46^{\pm 0.25}}$ |
| | DPC | $\mathbf{66.81^{\pm 0.31}}$ | $65.52^{\pm 0.29}$ | $64.12^{\pm 0.41}$ | $\mathbf{71.93^{\pm 0.15}}$ | $63.89^{\pm 0.49}$ |
| | DPC-CN | $64.39^{\pm 0.29}$ | $65.40^{\pm 0.31}$ | $64.27^{\pm 0.17}$ | $71.85^{\pm 0.18}$ | $64.20^{\pm 1.18}$ |

**Results.** Firstly, we train models of different depths on the CIFAR10 dataset, and report the best accuracies in the barplots in Figure 2. The plots show that both the original formulation of PC and the one that uses center nudging significantly drop in test accuracy when the depth of the model is increased. On the other hand, the newly proposed DPC, its centered version, and BP, tend to perform similarly. We have also performed a more comprehensive comparison against BP, reported in Table 1. The results show that for shallow networks (VGG5), DPC and DPC-CN can match or exceed BP performance. The performance gap becomes more noticeable as network depth and difficulty of task increases, suggesting room for further refinement in our approach. While our method doesn't consistently outperform BP yet, it represents an important step toward biologically plausible learning algorithms that can scale to deeper architectures. Furthermore, the performance gain from center nudging in DPC-CN is significantly smaller than its contribution in baseline PC-CN. This reduced suggests that DPC's modified weight update mechanism inherently mitigates latent node divergence during error propagation, thereby decreasing the need of nudging the output.

## 6 Conclusion

In this work, we have tackled the problem of scaling up predictive coding and equilibrium propagation for image classification tasks. Specifically, we investigated the following research question: Why do deep models trained with the predictive coding energy fail to match the accuracy of their counterparts trained with backpropagation? We have addressed this by proposing two novel regularization techniques and showed that the combination of our approaches allows the training of deep predictive coding models that come close to the performance of backprop-based models on the same task. We believe that these initial results will inspire future work directed towards making this class of models work at scale, that will tackle more complex datasets and modalities that we have considered, such as the training of equilibrium propagation-based ResNets on ImageNet (He et al., 2016), or small transformer models.

## REFERENCES

Karl Friston. A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456), 2005.

Wilfried Haensch, Tayfun Gokmen, and Ruchir Puri. The next generation of deep learning hardware: Analog computing. *Proceedings of the IEEE*, 107(1):108–122, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79, 1982.

Jack Kendall, Ross Pantone, Kalpana Manickavasagam, Yoshua Bengio, and Benjamin Scellier. Training end-to-end analog neural networks with equilibrium propagation. *arXiv preprint arXiv:2006.01981*, 2020.

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

Axel Laborieux and Friedemann Zenke. Holomorphic equilibrium propagation computes exact gradients through finite size oscillations. *Advances in Neural Information Processing Systems*, 35: 12950–12963, 2022.

Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *Frontiers in Neuroscience*, 15:129, 2021.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Javier R Movellan. Contrastive hebbian learning in the continuous hopfield model. In *Connectionist models*, pp. 10–17. Elsevier, 1991.

Timothy Nest and Maxence Ernoult. Towards training digitally-tied analog blocks via hybrid gradient computation. *Advances in Neural Information Processing Systems*, 37:83877–83914, 2024.

Alex Ororbia and Daniel Kifer. The neural coding framework for learning generative models. *arXiv:2012.03405*, 2020.

Luca Pinchetti, Tommaso Salvatori, Beren Millidge, Yuhang Song, Yordan Yordanov, and Thomas Lukasiewicz. Predictive coding beyond Gaussian distributions. *36th Conference on Neural Information Processing Systems*, 2022.

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

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

Tommaso Salvatori, Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Predictive coding can do exact backpropagation on convolutional and recurrent neural networks. *arXiv:2103.03725*, 2021.

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

Benjamin Scellier. Quantum equilibrium propagation: Gradient-descent training of quantum systems. *arXiv preprint arXiv:2406.00879*, 2024.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017.

Benjamin Scellier, Maxence Ernoult, Jack Kendall, and Suhas Kumar. Energy-based learning algorithms for analog computing: a comparative study. *Advances in Neural Information Processing Systems*, 36, 2024.

Eli Sennesh, Hao Wu, and Tommaso Salvatori. Divide-and-conquer predictive coding: a structured bayesian inference algorithm. *arXiv preprint arXiv:2408.05834*, 2024.

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

Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? — Exact implementation of backpropagation in predictive coding networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Hsinyu Tsai, Stefano Ambrogio, Pritish Narayanan, Robert M Shelby, and Geoffrey W Burr. Recent progress in analog memory-based accelerators for deep learning. *Journal of Physics D: Applied Physics*, 51(28):283001, 2018.

James C. R. Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local Hebbian synaptic plasticity. *Neural Computation*, 29(5), 2017.

Nicolas Zucchet and João Sacramento. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation*, 34(12):2309–2346, 2022.

# APPENDIX

Here we explain how experiments were performed and what results were found. Full details for repeating the experiments and results for ablation study are included in later sections.

## A EXPERIMENTS SETTING

**Model.** We conducted experiments on five models: VGG5, VGG7, VGG9, VGG11 and VGG13. The detailed architectures of these models are presented in Table 2.

Table 2: Detailed architectures of base models

| | **VGG5** | **VGG7** | **VGG9** |
|---|---|---|---|
| Channel Sizes | [128, 256, 512, 512] | [128, 128, 256, 256, 512, 512] | [64, 128, 256, 256, 512, 512] |
| Kernel Sizes | [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] |
| Paddings | [1, 1, 1, 0] | [1, 1, 1, 0, 1, 0] | [1, 1, 1, 1, 1, 1] |
| Pool window | $2 \times 2$ | $2 \times 2$ | $2 \times 2$ |
| Pool stride | 2 | 2 | 2 |
| Linear Layers | 1 | 1 | 3 |

| | **VGG11** | **VGG13** |
|---|---|---|
| Channel Sizes | [64, 128, 256, 256, 512, 512, 512, 512] | [128, 128, 128, 128, 256, 256, 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, 3, 3] |
| Strides | [1, 1, 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, 1] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] |
| Pool window | $2 \times 2$ | $2 \times 2$ |
| Pool stride | 2 | 2 |
| Linear Layers | 3 | 1 |

**Experiments.** The benchmark results of above models are obtained with CIFAR10, CIFAR100 and Tiny ImageNet, The datasets are normalized as in Table 3.

Table 3: Data normalization

| | **Mean** ($\mu$) | **Std** ($\sigma$) |
|---|---|---|
| CIFAR10 | [0.4914, 0.4822, 0.4465] | [0.2023, 0.1994, 0.2010] |
| CIFAR100 | [0.5071, 0.4867, 0.4408] | [0.2675, 0.2565, 0.2761] |
| Tiny ImageNet | [0.485, 0.456, 0.406] | [0.229, 0.224, 0.225] |

For data augmentation on CIFAR10, CIFAR100, and Tiny ImageNet training sets, we use 50% random horizontal flipping. We also apply random cropping with different setups. For CIFAR10 and CIFAR100, images are randomly cropped to 32×32 resolution with 4-pixel padding. For Tiny ImageNet trained on VGG5 and VGG7, images are randomly cropped to 56×56 resolution without padding. For testing on Tiny ImageNet with VGG5 and VGG7, we use center cropping to get 56×56 resolution images without padding. For Tiny ImageNet trained on VGG9, VGG11, and VGG13, random cropping produces 64×64 resolution images with 8-pixel padding. For testing on these models, no augmentation is used because Tiny ImageNet's original resolution is 64×64.

For the optimizer and scheduler, we use mini-batch gradient descent (SGD) with momentum for $x$ in the relaxing phase. We use AdamW with weight decay for $W$ in the learning phase. We also use a warmup-cosine-annealing scheduler without restart for $W$'s learning rates. This scheduler starts with a low learning rate (warmup), then smoothly transitions to a cosine-shaped decay, avoiding sudden drops in performance. The peak learning rate is 1.1 times the initial rate. The end learning rate is 0.1 times the initial rate. The warmup steps are 10% of all iteration steps.

We conduct hyperparameter selection based on the search space specified in Table 4. The results presented in Table 1 and Figure 2 are obtained using 5 different random seeds with the optimal

Table 4: Hyperparameters search configuration

| Parameter | PC/DPC | BP |
|---|---|---|
| Epoch | 25 | |
| Batch Size | 128 | |
| Activation | [leaky relu, gelu, hard tanh, relu] | |
| $k$ | 1 | - |
| $\beta$ | [0.0, 1.0], 0.15[1] | - |
| $lr_x$ | (5e-3, 9e-1)[2] | - |
| $lr_W$ | (1e-5, 3e-2)[2] | (1e-5, 3e-4)[2] |
| $momentum_x$ | [0.0, 1.0], 0.1[1] | - |
| $weightdecay_w$ | (1e-5, 1e-2)[2] | |
| T (VGG-5) | [5,6,7,8] | - |
| T (VGG-7) | [7,9,11,13] | - |
| T (VGG-9) | [9,11,13,15] | - |
| T (VGG-11) | [11,13,15,17] | - |
| T (VGG-13) | [13,15,18,21] | - |

[1]: "[a, b], c" denotes a sequence of values from a to b with a step size of c.

[2]: "(a, b)" represents a log-uniform distribution between a and b.

hyperparameter configuration. The training process is capped at 100 epochs, with an early stopping mechanism that terminates training if no accuracy improvement is observed for 10 consecutive epochs. To maintain consistency with the hyperparameter search settings, we employ a two-phase learning rate schedule: during the first 25 epochs, the weight learning rate follows a warmup-cosine-annealing schedule as previously described, after which it remains fixed at the final learning rate of the scheduler. For the results shown in Figure 1 and 3, we utilize a single random seed with the optimal hyperparameters, setting the maximum training epochs to 50 without implementing early stopping. The weight learning rate schedule remains identical to the aforementioned approach.

## B Energy Propagation in DPC models

In this section, we deliberately selected the shallowest (VGG5) and deepest (VGG13) architectures from our experimental framework for comparative energy distribution visualization. Figure 3 presents layer-wise energy distribution and classification accuracy among Backpropagation (BP), Predictive Coding (PC), and our proposed Deep PCNs (DPC), evaluated on VGG5 (a) and VGG13 (b) architectures using the CIFAR-10 dataset.

In shallow networks (VGG5), while all three methods demonstrate comparable accuracy, DPC achieves superior energy equilibrium across network layers compare to PC, thereby demonstrating that our proposed method effectively enhances the model's layer-wise energy balance. Interestingly, PC/DPC achieves good results despite having a less uniform energy distribution than BP. Figure 3(b) demonstrates that the proposed DPC maintains superior energy balancing compared to standard PC in deep networks, but not as uniform as that of BP.

## C Ablation Study

In this section, we conducted ablation studies to evaluate the effectiveness of each proposed component. We separately removed the energy decay term (denoted as "w/o EnergyDecay") and the forward update term (denoted as "w/o ForwardUpdates") from the DPC model. We performed experiments on both CIFAR-10 and CIFAR-100 datasets, resulting in six experimental configurations for each architecture and dataset combination. The results are presented in Table 5.

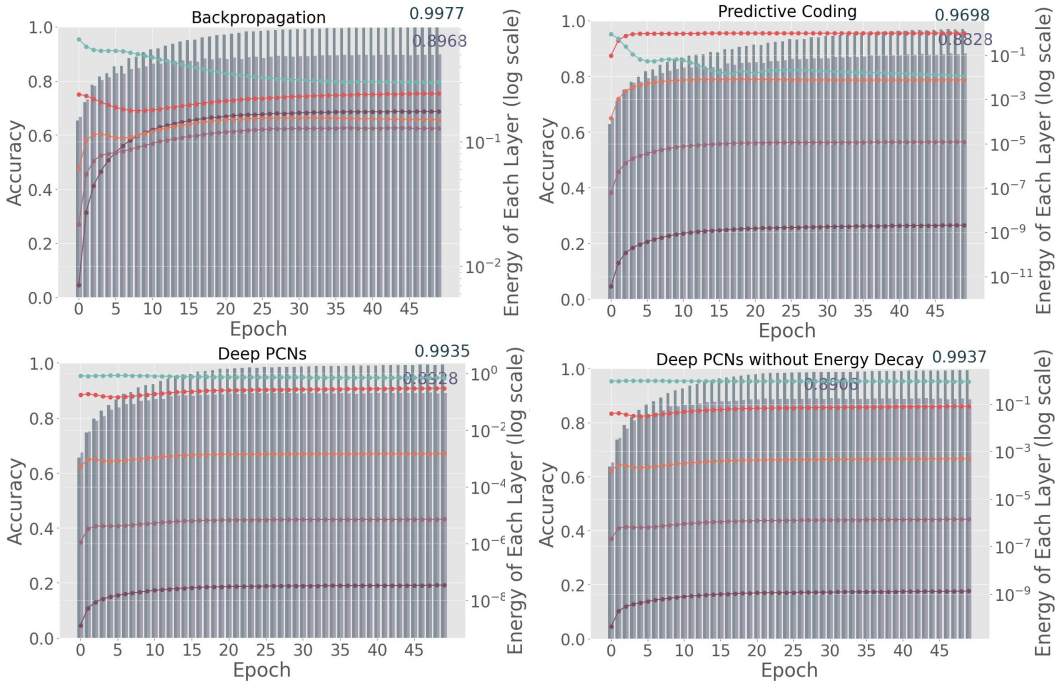
Firstly, we observe that removing Forward Updates (using the standard PC model with Energy Decay) causes significant accuracy degradation, which becomes more pronounced with increasing network depth. Simultaneously, the effectiveness of Center Nudging reappears when Forward Updates

are removed, with its impact also strengthening in deeper networks. This phenomenon confirms our earlier hypothesis: synaptic weight updates based on $x_t^l$ may introduce errors that accumulate across layers, leading to performance degradation in deep architectures.

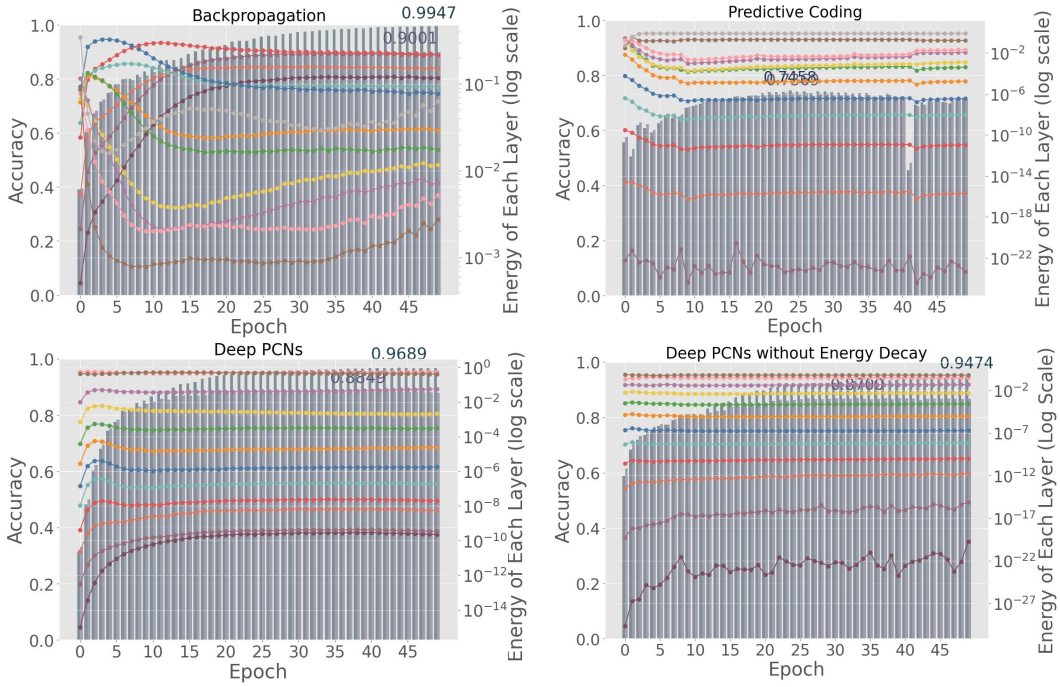
Secondly, we observed that removing Energy Decay leads to performance degradation in most cases. This effect is particularly evident in deeper models such as VGG13, where the energy distribution across layers becomes significantly imbalanced without the Energy Decay term. As shown in Figure 3(b), in the standard DPC model with Energy Decay, the first layer's energy proportion is approximately $10^{-10}$, whereas in DPC without Energy Decay, this proportion drops dramatically to $10^{-22}$. Visualization of layer-wised energy distributions between standard DPC and DPC without Energy Decay (as shown in Figure 3) confirms that our proposed Energy Decay method successfully balances energy distribution across layers, which contributes to improved model performance.

Table 5: Test accuracies of ablation study

| Dataset | Algorithm | VGG5 | VGG7 | VGG9 | VGG11 | VGG13 |
|---|---|---|---|---|---|---|
| CIFAR10 | DPC | $\mathbf{89.32^{\pm 0.14}}$ | $\mathbf{89.25^{\pm 0.09}}$ | $\mathbf{88.40^{\pm 0.18}}$ | $\mathbf{86.97^{\pm 0.10}}$ | $\mathbf{88.58^{\pm 0.07}}$ |
| | DPC w/o EnergyDecay | $89.00^{\pm 0.10}$ | $88.60^{\pm 0.11}$ | $85.38^{\pm 0.82}$ | $84.25^{\pm 2.74}$ | $87.03^{\pm 0.13}$ |
| | DPC w/o ForwardUpdates | $87.91^{\pm 0.22}$ | $81.04^{\pm 0.19}$ | $81.25^{\pm 0.36}$ | $73.01^{\pm 0.86}$ | $72.59^{\pm 1.95}$ |
| | DPC-CN | $\mathbf{89.46^{\pm 0.14}}$ | $\mathbf{89.11^{\pm 0.26}}$ | $\mathbf{86.55^{\pm 0.81}}$ | $\mathbf{86.59^{\pm 0.23}}$ | $\mathbf{87.50^{\pm 0.61}}$ |
| | DPC-CN w/o EnergyDecay | $87.82^{\pm 0.55}$ | $87.42^{\pm 0.76}$ | $85.27^{\pm 0.12}$ | $85.42^{\pm 0.13}$ | $86.90^{\pm 0.16}$ |
| | DPC-CN w/o ForwardUpdates | $88.24^{\pm 0.06}$ | $86.48^{\pm 1.22}$ | $79.10^{\pm 3.62}$ | $76.64^{\pm 1.37}$ | $80.61^{\pm 0.29}$ |
| CIFAR100 (Top-1) | DPC | $\mathbf{66.10^{\pm 0.09}}$ | $\mathbf{64.86^{\pm 0.10}}$ | $\mathbf{63.23^{\pm 0.16}}$ | $59.23^{\pm 0.37}$ | $\mathbf{60.19^{\pm 0.28}}$ |
| | DPC w/o EnergyDecay | $63.92^{\pm 0.62}$ | $63.87^{\pm 0.42}$ | $62.89^{\pm 0.25}$ | $\mathbf{60.31^{\pm 3.35}}$ | $59.76^{\pm 0.15}$ |
| | DPC w/o ForwardUpdates | $57.76^{\pm 0.33}$ | $45.05^{\pm 0.37}$ | $46.39^{\pm 0.39}$ | $45.97^{\pm 0.13}$ | $35.99^{\pm 0.64}$ |
| | DPC-CN | $\mathbf{66.85^{\pm 0.07}}$ | $\mathbf{63.89^{\pm 0.22}}$ | $\mathbf{60.59^{\pm 0.47}}$ | $\mathbf{60.56^{\pm 0.22}}$ | $\mathbf{61.27^{\pm 0.13}}$ |
| | DPC-CN w/o EnergyDecay | $64.48^{\pm 0.21}$ | $63.07^{\pm 0.83}$ | $59.61^{\pm 0.25}$ | $58.37^{\pm 0.40}$ | $60.81^{\pm 0.29}$ |
| | DPC-CN w/o ForwardUpdates | $55.89^{\pm 0.26}$ | $62.76^{\pm 0.56}$ | $54.77^{\pm 0.43}$ | $51.45^{\pm 0.19}$ | $50.29^{\pm 0.86}$ |
| CIFAR100 (Top-5) | DPC | $\mathbf{85.85^{\pm 0.10}}$ | $\mathbf{84.55^{\pm 0.20}}$ | $81.24^{\pm 0.33}$ | $\mathbf{81.08^{\pm 0.24}}$ | $\mathbf{81.47^{\pm 0.30}}$ |
| | DPC w/o EnergyDecay | $83.89^{\pm 0.44}$ | $81.68^{\pm 1.51}$ | $\mathbf{84.12^{\pm 0.11}}$ | $81.94^{\pm 1.63}$ | $79.21^{\pm 0.30}$ |
| | DPC w/o ForwardUpdates | $81.59^{\pm 0.13}$ | $74.00^{\pm 0.30}$ | $73.72^{\pm 0.06}$ | $72.30^{\pm 0.26}$ | $62.46^{\pm 0.47}$ |
| | DPC-CN | $\mathbf{85.85^{\pm 0.13}}$ | $\mathbf{83.80^{\pm 0.20}}$ | $80.95^{\pm 0.55}$ | $\mathbf{82.98^{\pm 0.07}}$ | $\mathbf{83.30^{\pm 0.19}}$ |
| | DPC-CN w/o EnergyDecay | $81.86^{\pm 0.28}$ | $81.02^{\pm 0.83}$ | $\mathbf{82.89^{\pm 0.09}}$ | $80.49^{\pm 0.15}$ | $80.19^{\pm 0.17}$ |
| | DPC-CN w/o ForwardUpdates | $80.06^{\pm 0.2}$ | $83.46^{\pm 0.38}$ | $80.54^{\pm 0.22}$ | $77.84^{\pm 0.3}$ | $76.22^{\pm 0.58}$ |




(a)




(b)

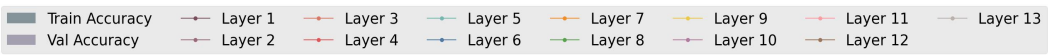



Figure 3: Layer-wise Energy Distribution and Accuracy Comparison between BP, PC, proposed DPC and DPC without Energy Decay in VGG5(a) and VGG13(b) on the CIFAR10 dataset. The colored lines represent the total energy of the individual layers of the model (or, the squared error of every layer for BP). The vertical lines represent the train and test accuracies of the model.