University of Amsterdam

MSc Artificial Intelligence Thesis

Predictive Coding Graphs for bidirectional learning with local updates

Tutor:
Parva Alavian (PhD)

mentor:
Dr. J.F. Jorge Mejias

Student:
Erencan Tatar
12681598

Group:
Cognitive and Systems neuroscience

Lecturer:
Dr. Pascal Mettes

Abstract

We study predictive coding (PC) on arbitrary directed graphs, unifying classification, generation, and occlusion within one model. We evaluate hierarchical, fully connected, and stochastic block models (SBMs) on MNIST-scale tasks. We show that shallow PC models are competitive for classification, but deeper models amplify training instability and update divergence compared with traditional backpropagation (BP). Generative PC models querying via clamped labels yield plausible yet weakly controllable MNIST image samples and are sensitive to hyperparameters. Fully connected graphs favor shortcut credit paths that collapse into near direct (sensory–label) mappings, bypassing rich latent structure. In contrast, clustered sparse topologies discourage direct sensory-to-label bypasses and scale to larger node counts without degrading accuracy. Overall, PC is promising for parallel, local learning on sparse graphs, but remains limited by sensitivity to initialization, step sizes, and topology.

Keywords: predictive coding, inference-learning, graph topology

1 Introduction

The backpropagation algorithm (BP) has been essential to the success of deep learning; however, its reliance on sequential backward passes, non-local computations, and acyclic architectures makes it biologically implausible. Neuroscience-inspired alternatives, such as predictive coding (PC), leverage strictly local updates and bidirectional message passing, and have the potential to overcome these limitations. PC networks (PCNs) can operate as classifiers, generators, and associative memory systems within a single framework and have been adapted in recent years to work with arbitrary graph topologies, enabling them to address a broader range of problems beyond traditional neural network structures. Building on the fully connected and arbitrary graph formulations of [1] and the hierarchical PC implementation of [2], we implement PCNs on arbitrary graphs to unify classification, generation, and associative memory tasks in one model. Therefore, we compare different topologies, such as hierarchical, fully connected, and stochastic-block graphs, to formally characterize PCNs from multiple perspectives and assess their viability as an alternative to backpropagation in modern ML. To improve generalization and scalability, we test amplifying slow-moving gradient updates and stochastic learning blocks to scale PC models.

2 Predictive coding background

Any computational system with learnable parameters must determine how changes to an individual connection affect the system's output, a challenge known as the *credit assignment problem*. Most modern deep learning frameworks are heavily dependent on backpropagation (BP), which computes how a global loss function changes with respect to individual weight parameters. The brain faces a credit assignment problem analogous to that in ANNs. Despite BP's success in machine learning, evidence suggests that the brain uses a different approach. BP training is divided into a forward pass (to compute activations) and a backward pass (to compute gradients and propagate errors). This two-phase update is not how the brain appears to learn, as there is no evidence of a distinct reverse signaling phase in biological circuits. Moreover, biological learning appears local in space and time, meaning neurons update based on nearby spiking activation and plasticity rules rather than a single global objective. The needed memory is carried by short-lived synaptic traces, not by explicit neuron-level caches stored after the forward pass and reused for backward (gradient) updates. Predictive coding, on the contrary, performs inference and learning in a unified process without a dedicated backward pass, meaning that it continually updates neuron activities to minimize prediction errors and that weight updates occur based on these local errors. In fact, the standard PC learning algorithm, often called Inference Learning (IL), does not store or reuse the original feedforward activations to compute weight updates, unlike BP, which must store activations from the forward pass. Therefore, PC's learning rule operates *online* on current activity and error signals, whereas BP explicitly caches neuron outputs from the forward traversal to calculate gradients during the backward traversal. A detailed comparison between BP and PC can be found in the Appendix section A and in [3, 4].

Backpropagation benefits from established techniques such as BatchNorm, dropout, and robust optimizers, making it highly effective, especially on larger datasets like CIFAR-100, where it consistently outperforms inference-based learning. While predictive coding can also leverage some of these techniques, it faces challenges such as instability due to errors, vanishing or exploding dynamics, and sensitivity in non-sequential or recurrent graphs. Loops and dense connectivity demand careful weight initialization, damping, and edge-specific tuning. While the Predictive coding framework can approximate backpropagation under certain conditions, it demonstrates unique capabilities and flexibility that traditional deep learning methods do not possess [5].

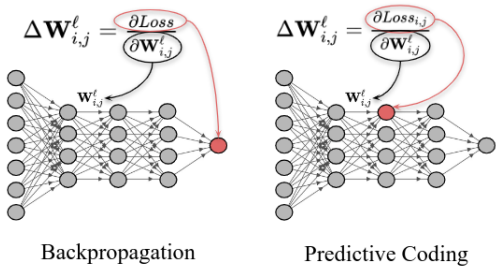


Figure 1: Locality contrast: BP propagates a global error through the network; PC updates each synapse using local activity and prediction error. Backpropagation updates its synaptic weights $W_{i,j}$ with respect to a global local, even if the *loss* node is not directly connected to it. PC models on the other hand, perform their updates to correct the error of their directly connected postsynaptic neuron. Figure taken from [6]

From a machine learning perspective, predictive coding has promising properties, achieving excellent results in classification [7] and memorization [8]. PC has been shown by [9, 10] to be competitive with BP algorithms (on small datasets) when using feedforward PC models. A hierarchical PC implementation from [11] even achieves 99.9% accuracy on CIFAR-10.

Predictive coding networks were first introduced for unsupervised feature learning by Rao & Ballard [12] and were later extended to perform supervised learning [7]. This extension allows predictive coding to be used in a range of machine learning tasks, from classification to generative modeling. The versatility of PCNs stems from their ability to integrate information in a manner similar to how the brain processes sensory inputs and updates its internal models. Although the brain learns unsupervised, there are tricks to train predictive coding models without explicit labels as shown in [13, 14].

The concept of (inference) learning is extensively described in Hohwy's account of the *self-evidencing brain* [15] in which the brain's primary function is to ensure its own *model* of reality is well supported by the sensory stimuli. In other words, the brain sets out to confirm and refine the internal picture it holds about the world by constantly seeking to reduce the gap between what it expects (its predictions) and what it observes. *Active inference* is the process of performing generative actions guided by beliefs that reshape the world to align more closely with one's internal model. *Perception inference*, used in our predictive coding model, creates and parameterize an internal model and tries to match that to external stimuli after which its updates its internal beliefs and therefore increases its evidence.

The core idea of predictive coding is that the brain maintains an internal model that continuously generates predictions about incoming sensory data. These predictions are compared with the actual sensory inputs, and the discrepancies, known as prediction errors, are used to update the internal model. Under predictive coding, the brain is modeled as keeping an internal generative model that predicts its sensory inputs; perception then amounts to (approximate) Bayesian inference that minimizes prediction errors or free energy [16–19].

In the Predictive Coding framework, higher-level (cortical columns in the brain) predict activity in lower-level neurons, all the way down to predicting direct sensory input. Given a stimulus at a sensory neuron, we compare it with the prediction from higher layers, yielding a prediction error. Given the clamped sensory stimulus x_0, the representations $x_1 - x_L$ are updated (using one or multiple bottom-up prediction errors) and passed as a message to higher layers, which in turn are used to compute the difference between this new representation and its predictions.

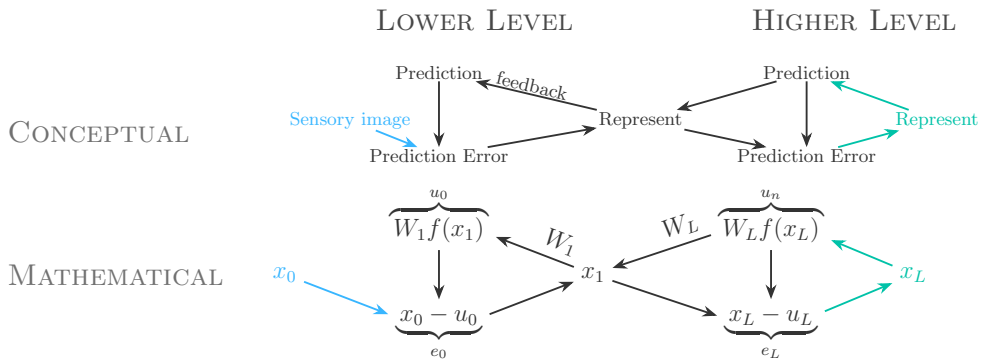


Figure 2: Hierarchical predictive coding framework showing conceptual and mathematical processes across lower and higher levels. In supervised learning, the representation x_L is clamped to the label corresponding to the sensory image x_0.

Each layer attempts to minimize its own prediction error by comparing top-down predictions with bottom-up inputs. The lowest level receives sensory input (e.g., an image), while higher levels generate predictions to explain activity in the layer below. Errors are propagated upward and used to adjust internal representations, with feedback refining predictions. This iterative process continues until a stable representation is reached throughout the hierarchy. From a mathematical perspective, this mechanism effectively implements hierarchical variational Bayesian inference, assuming a Gaussian variational posterior and a compatible generative model. This bidirectional flow of information, where both bottom-up sensory inputs and top-down predictions interact, enables efficient and adaptive learning. Furthermore, this bidirectional processing is fundamental for the functioning of brain areas, such as the hippocampus [8, 20].

3 Learning

The Rao and Ballard 1999 model was initially formulated as a hierarchical generative model with both feedforward and feedback connections, rather than a purely feedforward model. Later work cast predictive coding algorithm as approximating a Bayesian inference process based upon Gaussian generative models [21, 22]. Recent works also formalized PC in terms of energy-based models, linking PC dynamics to the principles of variational inference. We formalize this with Bayesian inference: given (sensory) observations o (images or labels), we wish to infer the latent states x that caused them. We assume a generative model of the data-generating process: the joint probability $p(o, x; \theta) = p(o|x; \theta)p(x; \theta)$. Using Bayes' rule, we can compute the posterior distribution over latent states given the observation. The posterior $p(x|o)$ captures our updated belief about the latent state after observing the data (omitting θ).

$$p(x|o) = \frac{p(o|x)p(x)}{p(o)} = \frac{p(o|x)p(x)}{\int_x p(o|x)p(x)\,dx}$$

Furthermore, once the latent state x has been inferred, it can be used to predict the expected sensory input. This prediction is computed as the expected value of the observation conditioned on the latent state:

$$\text{prediction} = \mathbb{E}[o \mid x] = \int_o p(o \mid x)\,do$$

This prediction allows us to compare the expected and actual observations, thereby generating a prediction error. This prediction error is central to predictive coding, where it updates neural representations of latent states, effectively refining the inference process [12]. However, the denominator $p(o|\theta)$, called the marginal likelihood (or model evidence), is often intractable, as it requires integration over all possible latent states x. To do this efficiently, we optimize a tractable upper bound on the divergence, known as the variational free energy, or equivalently, in machine learning, the negative evidence lower bound (ELBO). To generate this bound, we apply Bayes' rule to the true posterior, rewriting it as a generative model and the evidence. We approximate it with a variational posterior $q(x|o; \phi)$, parameterized by a learnable ϕ to minimize the divergence between this approximate distribution and the true posterior.

This approach allows us to iteratively update our belief about the latent state, using prediction errors to drive inference and learning. In this way, variational Bayesian inference and predictive coding are unified under a common framework for solving the inverse problem. Given observations o, infer the most likely latent causes x that generated them. In particular, PC can be seen as a process of inverting a generative model by minimizing an energy function corresponding to variational free energy. This formulation helps us understand how PC can perform both inference and learning using only local signals. *Free Energy* from the equation below can be arranged as Complexity − Accuracy, showing a trade-off between the first term, which forces the minimization of complexity and thus generalizes well, and the second term, which forces memorization or fitting of the data. The objective becomes minimizing the variational free energy (negative ELBO):

$$\mathcal{F}(x, o, \theta) = \underbrace{\mathrm{KL}[q(x \mid o, \theta) \parallel p(x \mid \theta)]}_{\text{Complexity}} - \underbrace{\mathbb{E}_{q(x|o,\theta)}[\log p(o \mid x, \theta)]}_{\text{Accuracy}}$$

To obtain the standard PC energy from [23] and used in practice, assume a hierarchical Gaussian generative model with layers $x = \big(x^{(0)}, \ldots, x^{(L)}\big)$ and parameters $\theta = \big(\theta^{(0)}, \ldots, \theta^{(L-1)}\big)$, where layer 0 corresponds to the generated data o and layer L is the highest in the hierarchy. In supervised training, it is standard to clamp the endpoints during inference (e.g., $x^{(0)} = o_{\text{image}}$ and $x^{(L)} = o_{\text{label}}$), similar to a standard MLP or, similar to Figure 2. Here $\mu(\text{l})$ is the prediction of layer l according to the layer above, with $f(\cdot)$ being a non-linear function and $\mu_l = x_l$. The joint or marginal probability of all layers of the causes is written as a product of Gaussian conditionals, which implicitly encodes dependencies between layers:

$$p\big(x^{(0)}, \ldots, x^{(L)}\big) \;=\; \prod_{l=0}^{L} \mathcal{N}\big(\mu^{(l)}, \Sigma^{(l)}\big), \qquad \mu^{(l)} = \theta^{(l)}\, f\big(x^{(l+1)}\big), \;\; \mu^{(L)} = x^{(L)}$$

Here, we assume Gaussian distributed prediction errors at each layer. With a Laplace mean-field approximation, the true posterior is simplified: the Laplace part replaces the generally non-Gaussian posterior with a Gaussian around its mode (MAP), and the mean-field part factorizes it across layers so that updates are independent and tractable. With unit covariance $\Sigma^{(l)} = I$, the upper-bound of the negative log-likelihood terms collapses to squared prediction errors up to constants, resulting in the free energy $\mathcal{F}$:

$$\boxed{\mathcal{F}(x, y, \theta) \;=\; \sum_{l=0}^{L} \big\|x^{(l)} - \mu^{(l)}\big\|^2 \;=\; \sum_{l=0}^{L} \big\|\varepsilon^{(l)}\big\|^2, \quad \varepsilon^{(l)} = x^{(l)} - \mu^{(l)}}$$

This formulation is equivalent to the original PC objective of Rao & Ballard. It is jointly minimized over activities and parameters and establishes PC as an energy-based inference arising from a Gaussian hierarchical model. Maximizing the evidence (marginal likelihood) for a model, or equivalently minimizing the discrepancy between the internal model and the observed world, can be seen as reducing prediction error. This formulation shows how the *Free energy* and *prediction errors* are derived and will serve as a basis for a predictive coding model applied to arbitrary topologies. The complete mathematical derivation can be seen in [24].

4 Inference learning

While predictive coding provides a principled account of inference as free-energy minimization, different computational schemes can be used to implement this process in practice. These approaches vary in their assumptions, computational cost, and accuracy, but they all aim to reduce the divergence between the approximate and true posteriors. There are multiple approaches to reduce the divergence between the approximate and true posteriors such as; (Variational) Laplace approximations [25], Monte-Carlo–based schemes (e.g., Monte Carlo Predictive Coding) [26], amortized inference via Variational Inference (VI) [27], and the Expectation–Maximization (EM) algorithm [28–30]. These methods trade off accuracy, assumptions, and efficiency. In this context, the learning dynamics of many predictive coding models can be interpreted as an EM-like alternation that jointly minimizes a shared variational free energy F, described by the inference learning (IL) framework [21, 24, 31, 32].

The first step in the PC EM-algorithm is called *inference* where we try to infer the best causes $x_1 - x_{L-1}$ (node activities) given the fixed parameters θ and clamped observation o data, consisting of sensory nodes x_0 (image) and supervised nodes x_L (label) or in traditional ML referred to as y. This step is done until convergence (energy equilibrium) or a fixed number of steps T. The second M-step is *learning*, in which we update the model parameters θ based on the inferred causes to maximize the expected output likelihood.

$$\begin{cases} 1. \text{ Inference (E-step):} \\ \quad x = \text{minimize } \mathcal{F}(x, o, \theta) \\ \\ 2. \text{ Learning (M-step):} \\ \quad \theta = \text{minimize } \mathcal{F}(x, o, \theta) \end{cases}$$

The model, see Figure 3, is initialized by clamping the sensory input x_0 and the supervision nodes x_L. The model infers latent states (gray) given the fixed parameters (W) by minimizing prediction error (red). Notice that we perform multiple inference steps before a single weight update or learning step. This is the main goal of predictive coding: inferring latent states and learning parameters by minimizing prediction error. Minimizing these errors, E will be the same as *maximizing* the posterior, which is here the same as minimizing the negative log likelihood.

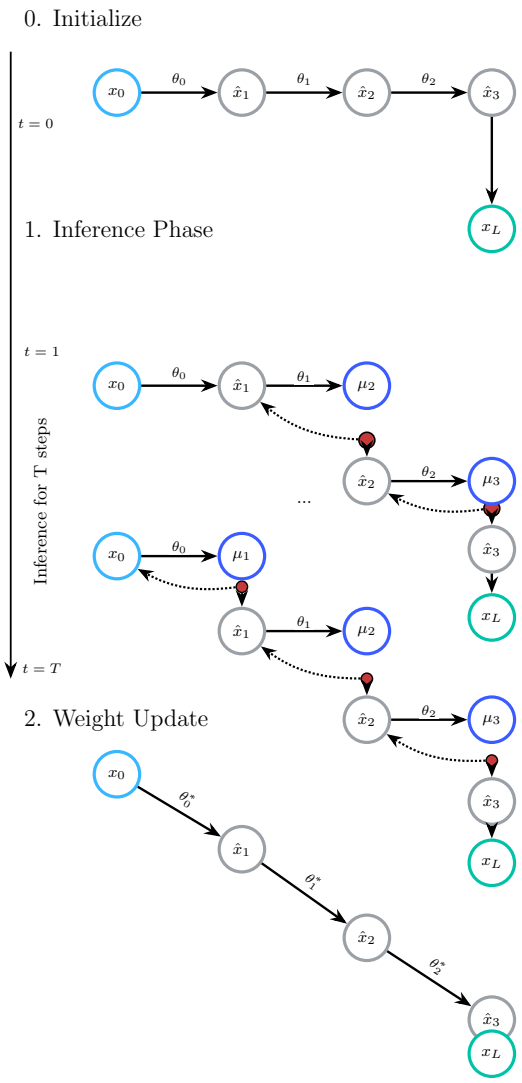


Figure 3: Inference Learning. Figure altered from [33]

4.1 Incremental inference learning

Incremental variants of predictive coding have also been proposed by [23] to bridge the gap between standard PC and BP. In traditional PC models, we first infer node values for T iterations (E-steps) before performing a single weight update (M-step). In Incremental PC (IPC), the network updates its weights at every time step of the inference process, rather than waiting for full convergence to reach a stable equilibrium of neural activities. Intuitively, this provides a continuum between the simultaneous convergence of standard PC and the strictly phased update of BP, effectively a continuous shift between PC and BP's behaviors. Incremental PC (IPC) updates the activity and weight at every step (an incremental-EM view), preserving a relaxed version of energy $\mathcal{F}$ while improving stability and efficiency. IPC tends to improve convergence speed and learning stability (since weights are nudged gradually), but when applied to general, non-layered networks, it still does not precisely follow backpropagation updates. This continuous updating process allows the algorithm to avoid the need for external control signals that typically switch between the inference and learning phases, making it more autonomous and biologically plausible.

5 PC graph

We consider an (arbitrary) directed graph $G = (V, E)$ where V is a set of N vertices representing nodes, and E is a set of directed edges encoding relationships between these nodes. We define our direction to be similar to the standard ML convention, see section 5.1 *Direction*. This means that our adjacency matrix A is indexed with rows corresponding to targets (receivers), and columns with sources (senders), meaning W_{ij} is the weight on the directed edge $j \to i$ such that j predicts i. Each node $i \in V$ has an activity value (or value node) denoted x_i (with a time index when dynamics are considered, e.g. $x_{i,t}$). A subset of these nodes is designated as sensory nodes (e.g., the first s nodes correspond to external data or sensory inputs), while the remainder are internal nodes that represent latent or higher-level variables. If trained under supervision, an additional subset l is selected for the label. In this paper, we only mention PC graphs trained in a supervised manner.

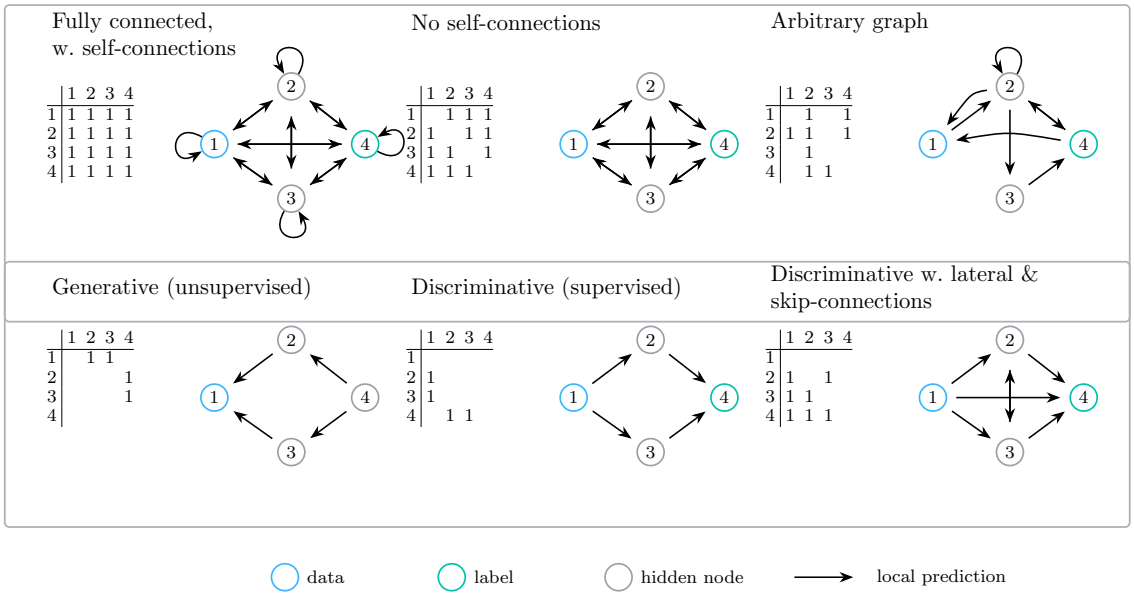


Figure 4: Different adjacency matrices, determining the topology of the graph and therefore also the learning dynamics. Figure taken from [2] and altered.

At each time step, node i computes a prediction $\mu_{i,t}$, derived from incoming signals transmitted through its incoming edges, based on an aggregation function, typically the weighted sum of signals. The function $f(\cdot)$ serves as a nonlinear transformation of the activity $x_{j,t}$ from the sending nodes. Additionally, each node computes a prediction error $\epsilon_{i,t}$, quantifying the difference between the node's actual state $x_{i,t}$ and its prediction $\mu_{i,t}$ at time t. This error reflects the amount of surprise or deviation from the expected activity and is propagated backward along the graph to inform updates at preceding nodes. Thus, each node computes a prediction μ_i of its own activity x_i and compares the two, resulting in a prediction error ϵ_i, computed as follows:

$$\begin{aligned} \mu_{i,t} &= \sum_{j}^{N} W_{ij} f(x_{j,t}) = W f(x) \\ \epsilon_{i,t} &= x_{i,t} - \mu_{i,t} \end{aligned} \tag{1}$$

The binary variable $A_{i,j}$ from the adjacency matrix, which indicates the presence of an edge from node j to node i, is omitted in the equation above since it is only used to construct the weight matrix W accordingly.

PC graph models are trained by minimizing the model's global energy and incorporating the adjacency matrix A. This total energy E_t is the sum of the prediction errors of the network at time t. The total error of the predictive coding network is given by:

$$E_t = \frac{1}{2}\sum_{i}^{N}(\epsilon_{i,t})^2 = \frac{1}{2}\sum_{i}^{N}(x_{i,t} - \mu_{i,t})^2 \tag{2}$$

Notice how this Energy function E is similar to our energy function $\mathcal{F}$, rewritten over the sum of each node instead of the sum over each hierarchical layer.

Training: The learning objective is to minimize the energy function E_t. In our case, the parameters of the generative model W need to be learned, where we first infer the latent variables of a data point given our model (**E-step**) using Gradient descent, and then use those latent representations to update the model weights w **(M-step)**. During both the inference and weight phase, we set the input layer to equal the datapoint, while the output layer nodes are clamped to the desired label (in supervised learning).

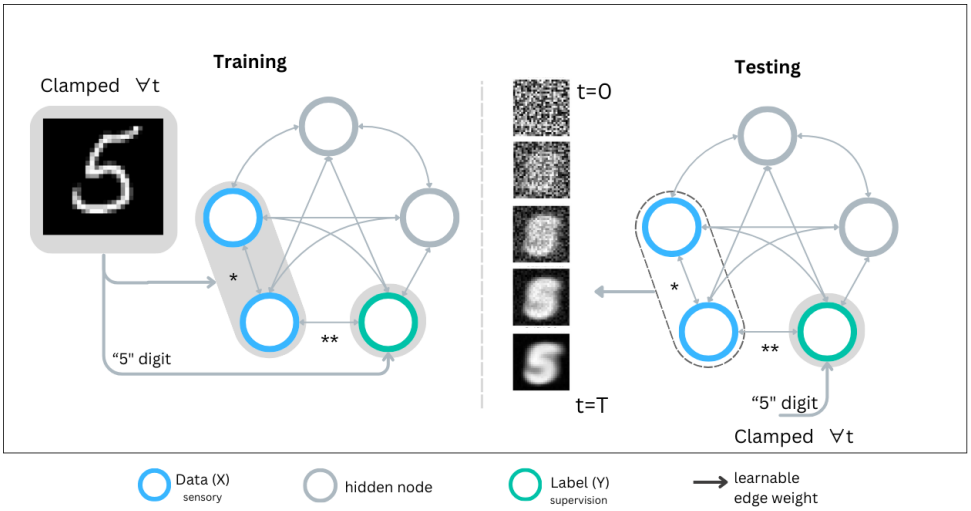


Figure 5: This figure shows the training and testing of a fully connected graph with sensory (gray), internal (white), and label (green) nodes. During supervised training, both sensory and label nodes are clamped (grey background). At test time, clamp either the sensory or label nodes, perform inference until the model converges to an equilibrium, and recover the unclamped node of interest for either classification or generation.

1. Inference: During the inference phase, the weights are fixed and the values are updated for T iterations or until convergence, to minimize the error. The update equations with learning rate γ for the value x_i is given by:

$$\begin{aligned}\Delta x_i = -\gamma\frac{\partial E_{i,t}}{\partial x_i} &= \gamma\left(-\epsilon_{i,t} + f'(x_{i,t})\sum_j W_{ij}\cdot\epsilon_{j,t}\right) \\ &= \gamma\left(-\epsilon_{i,t} + f'(x_{i,t}), (W^\top\epsilon_t)_i\right)\end{aligned} \tag{3}$$

The neuron's activity is adjusted to find a compromise between its own error e_i, which drives it to align with its top-down prediction. The second term, which encourages it to better predict the activity in the layer below. This equation embodies the view of the passing of messages: node i adjusts upward or downward in response to its own local error e_i and the weighted errors of the nodes it projects (sending feedback indicating whether the prediction of i was too high or low). This iterative update of the value nodes distributes the output error throughout the entire PC graph.

2. Weight Update:
The weight update for W_{ij} using learning rate α :

$$\Delta W_{ij} = \alpha \frac{\partial E_T}{\partial W_{ij}} = \alpha \left(-\epsilon_{i,T} f(x_{j,T}) \right) \tag{4}$$

The synaptic weights W_{ij} are adapted to minimize prediction errors, via a Hebbian-like update involving presynaptic activity $f(x_j)$ at step T and postsynaptic error e_i. This rule intuitively increases W_{ij} if node j consistently under-predicts (positive e_j) when i is active, and decreases W_{ij} if node j over-predicts (negative e_j) given i's activity. The result is that over time the weights learn to encode the dataset, meaning W_{ij} will adjust to better predict j from i, corresponding to the original Rao-Ballard error-driven learning rule generalized to graphs.

PC and IPC are 1-hop in time, meaning each update depends only on the present presynaptic activity $f(x_t)$ and postsynaptic error e_t. Temporal credit assignment (like BPTT) would require explicitly unrolling the inference dynamics and summing derivatives through all earlier steps. Unlike PC and IPC which by default they perform local relaxation (to spread information spatially over neighbors) and local Hebbian-like weight updates using the current error only.

Importantly, these derivations of the error E with respect to either node value x_i or weight W involve only information from the node's direct neighbors, making these updates local in both implementation and time. For full mathematical derivations, see [5, 8].

Testing Although the model is not directly trained on a single task, it can be used for tasks such as classification, generation, and associative memory. During training, the model parameters are initialized, the data and labels are clamped, and the internal nodes converge to an equilibrium before updating the weights. Essentially, given training data, the model is trained to *construct* an energy landscape where low-energy minima correspond to the distribution of training samples; see Figure 6 below.

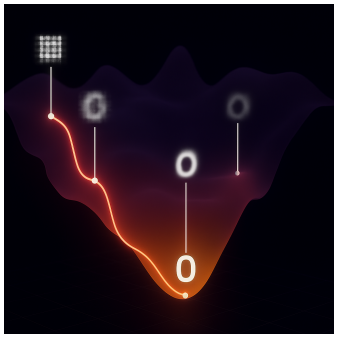


Figure 6: Inference moves the node values towards regions of low energy, while altering the weights creates an energy landscape such that lower energy states correspond to our preferred states.

Testing is performed either by *query by initialization*, where all nodes are randomly initialized and only a subset is set at $t = 0$ without clamping, allowing inference to naturally denoise or fill in missing information or by *query by conditioning*, where a subset of nodes (e.g., pixels or labels) is clamped throughout inference so that the remaining nodes converge to the corresponding conditional expectation. Under these schemes, classification is achieved by conditioning on image pixels to infer the one-hot label, while generation starts from a zero-valued image and conditions on a target label to infer pixel intensities. Reconstruction can be performed by conditioning on half of the image to infer the missing half, or by additionally conditioning on the label to obtain more accurate and disambiguated completions.

5.1 Direction in PCN

The literature on predictive coding in neuroscience and the literature in machine learning employ opposite conventions for the local prediction, and this opposition is visible when one compares the bottom-up formulation from Van Zwol [2] with the graph-oriented top-down formulation of Salvatori [1]. Both papers implement the same free-energy objective, yet they adopt different notations in order to serve other goals. Throughout this section, we use a fixed row i as receivers/targets and columns j as senders/sources, so (W_{ij}) is the weight on edge $j \to i$. Rule of thumb: predictions follow W; errors flow via $W^\top$.

The bottom-up formulation is mainly used to demonstrate that a discriminative predictive coding network reduces to an ordinary feed-forward neural network at test time once inference has converged. To make this link transparent, Van Zwol et al. [2] define the local prediction in layer ℓ as:

$$\mu_\ell = f(W_{\ell-1}x_{\ell-1}),$$

so that predictions travel away from the input data toward progressively higher latent layers, exactly as activations do in a multilayer perceptron. Errors then propagate via $W^\top$ during inference. With this choice, the forward sweep executed at test time is numerically identical to that of a standard classifier, and all proofs that the weight update approximates backpropagation can be carried out[1]. Within this convention, the matrix element W_{ij} is stored in row i and column j. The index j labels the pre-synaptic source of activity, and the index i labels the postsynaptic target that is being predicted. One therefore reads W_{ij} in Van Zwol as *unit j influences unit i's prediction*.

The top-down formulation instead aims for biological plausibility and a generative latent-variable interpretation that allows for both hierarchical and arbitrary graph topologies. This choice keeps the historical convention introduced by Rao & Ballard in which higher or parent nodes synthesize predictions of lower or child nodes. Their local rule for a fully connected graph reads:

$$\mu_i = f(\sum_j W_{ji}x_j),$$

And if the placement of the activation function $f(\cdot)$ is ignored, this collapses to $\mu_\ell = f(W_{\ell+1}x_{\ell+1})$ when the graph is an ordinary chain of layers or $\mu = f(W^T x)$ in matrix form. In this perspective, predictions flow toward the sensory data, and errors are transmitted upward, preserving the intuitive idea that the cortex explains away incoming signals by sending hypotheses downward. Notably, these rules are Hebbian-like: they rely exclusively on the local product of source and target (or pre- and postsynaptic) activities on each edge, with no need for global or non-local signals. In this notation, the activity dynamics of each node depend on both its own prediction error and those of its child nodes. These child-node errors are conveyed upward to the parent node (i.e., by multiplying errors with the appropriate weight matrix in the opposite direction). As a result, prediction errors naturally propagate upwards along parent-child edges, following the established PCN conventions for hierarchical inference [7]. For classification vs. generation, the local equations are unchanged; only the boundary conditions differ (which layers are clamped). A full comparison can be seen in Appendix Section E.

To translate between the notations, first transpose the weight matrix like: $W^{VanZwol} = (W^{Salvatori})^T$. Next, during supervised training, data and labels are clamped; at test time we clamp $x^{(0)}$ for classification and $x^{(L)}$ for generation. Local equations are unchanged, only boundary conditions differ. Finally, in figures interpret arrow direction as prediction direction (see Figure 7).

[1] Furthermore, PCNs can be mathematically considered a superset of traditional feedforward neural networks, as shown in [34].

Differences in placement of the activation function f comes from neuroscience literature where standard point-neuron rate models apply a postsynaptic nonlinearity after linear summation, $\mu_i = f\left(\sum_j W_{ij}x_j + b_i\right)$, whereas dendritic subunit models use local branch nonlinearities before summation $\mu_i = \sum_j W_{ij}f(x_j)$ seen in [35, 36]. An optional bias term b_i can be added such that each neuron can shift its activation independently; without biases, networks lose flexibility, especially with non–zero-centered inputs, because units cannot compensate for mean offsets in their pre-activations [37–39]. Biologically, a bias corresponds to a neuron's effective threshold or resting potential offset rather than a dedicated bias weight [35, 40]. Updated equation for μ_{it} with bias and learning parameter α is defined as:

$$\mu_i = \sum_j W_{ij}f(x_j) + b_i, \quad \Delta b_i = \alpha\epsilon_{i,T}$$

In normalized datasets, this choice of activation placement and bias often does not affect model performance, but it is worth noting that it changes the form of all update equations.

The two definitions of local prediction are complementary views of the same predictive-coding principle. Van Zwol chose a matrix orientation that foregrounds compatibility with feed-forward neural networks, whereas Salvatori preserved the classical cortical metaphor in which higher representations generate hypotheses about lower ones. In either case, a single elementary transpose reconciles the mathematics, and the element W_{ij} should always be read as meaning that the activity in node j contributes to the prediction in node i. This is a pure index conversion, where only the directional interpretation flips from top-down prediction (Salvatori) to bottom-up prediction (Van Zwol). However, for a fully connected or arbitrary graph, there is no concrete definition of bottom or up, only a conceptual notion of closer to the sensory input data.

In summary, we have revised the directionality assumptions to the Van Zwol convention: $\mu = Wf(x)$ (forward) and inference with $\Delta x = \gamma(-\epsilon + f'(x) \odot W^\top\epsilon)$ (backward); sensory data remains clamped at the input layer ($\ell = 0$ or sensory nodes s), while optionally supervision data is clamped at the output layer ($\ell = L$ or supervision nodes l). Predictions travel upward (from lower-index to higher-index nodes), and errors travel downward (from higher-index to lower-index nodes).

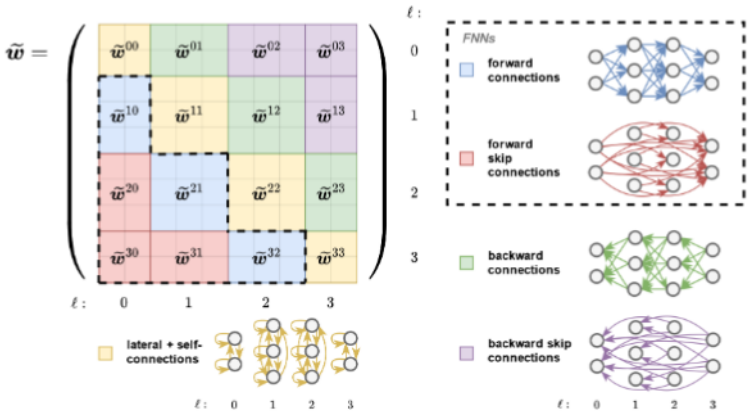


Figure 7: Similar to the image from [41] Fig. 2, we fix sensory at l_0 and label at l_N. Meaning sensory are at W^0 (images) and weights corresponding to the labels are fixed to W^N. Here, weights from block W^{lk} represent weights from layer k sending predictions to layer l.

6 Results

In the method above, we define a directed graph G and pose no further restriction on topology (sequential) layer or connection information, apart from the sensory and supervision nodes present. The model inference phase can be seen as a sequential chain of message passing operations on the same graph as a whole.
Setup: For the given MNIST dataset, we have trained different hierarchical discriminative, generative models, and a fully connected model trained with Inference Learning.

6.1 Discriminative PC

The graph consists of 784 (28x28 for MNIST) sensory nodes, internal nodes in L layers, and 10 supervision nodes for the one-hot encoding of the digit class, enabling the model to be queried in a discriminative manner. Here, the model learns a direct mapping from our data X to a label Y. For this task, we query the model to return the correct class given the sensory stimulus by taking the highest node values of the supervision nodes. The stimulus and, thus, the sensory node values are clamped throughout the evaluation, so only the internal nodes are updated during inference.

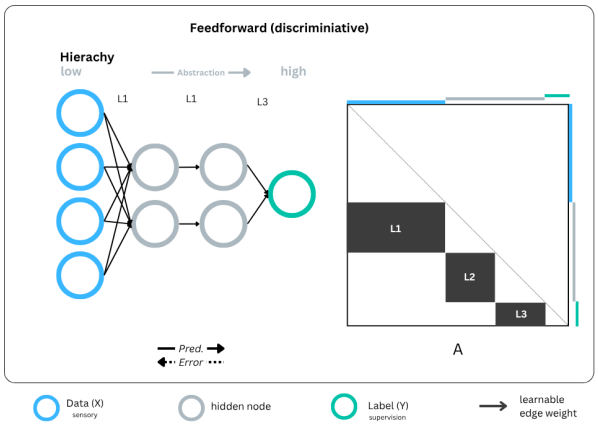


Figure 8: Simplified MLP architecture which can be trained with either BP or PC. Prediction and errors are stored in the nodes themselves; however, for analogy purposes, we assign prediction and error flow direction.

Table 1: MLP architecture trained for 15 epochs with PC, IPC, and BP and classification accuracy evaluated on the MNIST unseen test dataset. Tested with the same weight init and activation function for BP and (I)PC. $T_{train} = T_{test} = 35$ for (I)PC. Δw is the normalized average absolute difference between IPC and BP.

Layer size	**PC**	**IPC**	**BP**	$\Delta w[\%]$
L_{32}	0.72	0.93	0.93	126.2
$L_{64,32}$	0.82	0.93	0.94	136.6
$L_{128,64,32}$	0.92	0.91	0.98	141.8
$L_{256,128,64}$	0.87	0.92	0.97	144.6
$L_{512,256,128}$	0.90	0.92	0.99	171.5

As shown in Table 1, increasing the number of layers does not consistently lead to higher accuracy, particularly in the PC model, suggesting diminishing returns or instability in deeper architectures. Both the depth and the width of the networks appear to influence performance. Notably, in the PC framework, as the number of layers increases, the Δw metric also rises, suggesting that deeper models may introduce more noise in signal propagation and thus greater divergence in training dynamics than BP.

6.2 Generative PC

The discriminative model, using high-dimensional input data (images) and outputs a low-dimensional class label, is generally a more straightforward mapping than the reverse for generative models. Using MLPs in a generative manner is more challenging and less common, as MLPs process spatial information less efficiently compared to CNNs or VAEs. To query the generative capabilities, we clamp the supervision label (one-hot) to match the desired digit label. We provide the sensory nodes with random noise, fix the labels, and use gradient descent to update the values using the fixed weights. For the generated output images, we take the values of the unconstrained sensory nodes, which were originally fixed to the image pixels during training. Prediction flows from clamped labels to the sensory nodes. We test different topologies with direct mappings, one and multiple hidden layers.

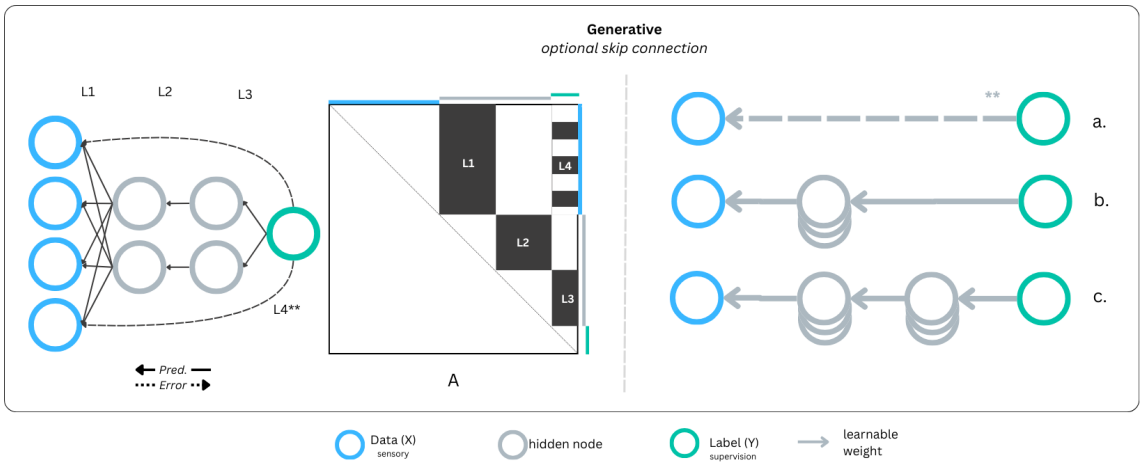
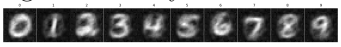


Figure 9: Hierarchical generative PC model, similar to a standard MLP with inverted edge directions. Optional skip connection (L4) directly from label nodes to the sensory nodes.

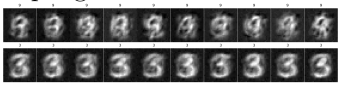
a. Direct L2S, no hidden layers

b. Single hidden layer



c. Multiple hidden layers

c. Sample generative distribution



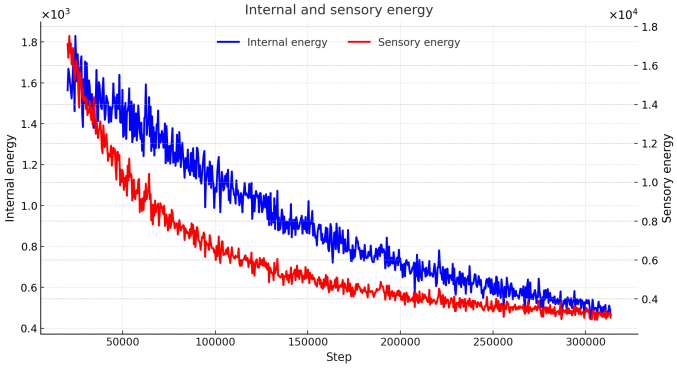


Figure 10: MLP-like hierarchical architecture trained using Incremental predictive coding. A-D corresponding to generation of MNIST digits 0-9, corresponding with Figure 9 A-C. The energy plot of a hierarchical generative PC with hidden nodes $L = \{150, 100, 100\}$.

Generation does not have direct control over any output attributes, such as capsule networks or Predictive coding with topographic variational autoencoders [42]. To sample different digits from the same class, the hidden values from the layer closest to the label are initialized before inference. Results in the Figure 9 C. shows no control or chaotic images using this approach.

6.3 Fully connected

In the fully connected model (or any other topology without direct apparent hierarchy), there is no notion of *top-down* (predictions) and *bottom-up* (errors). For a single model that simultaneously learns to classify and generate, we turn to the Boltzmann and Hopfield families of architectures, with restricted and unrestricted fully connected PC models. Alongside fully connected, we tested the influence of different parts of the graph on learning processes by removing certain edge types, such as direct sensory-to-and-from-label-node connections.

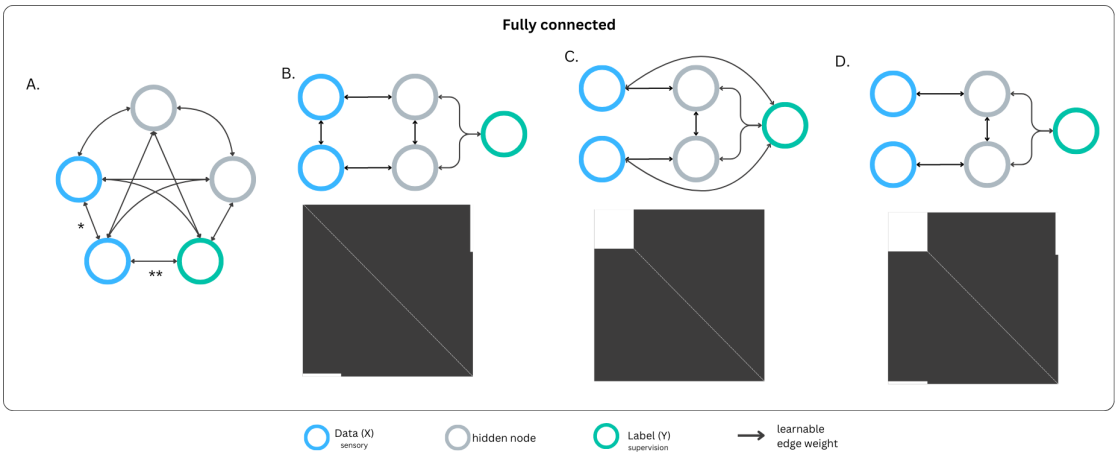
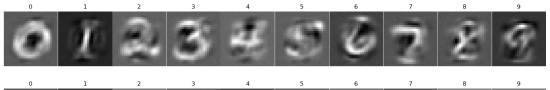
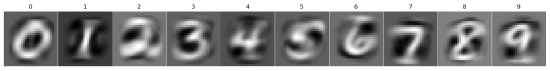
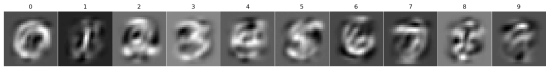
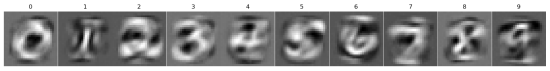
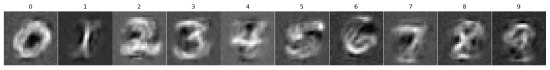
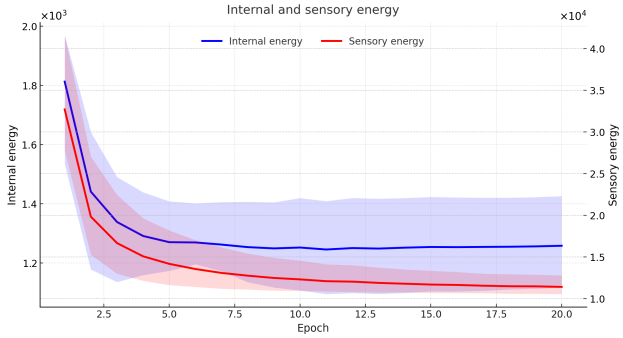


Figure 11: Fully connected model (A. results in $Adj_{matrix} = I$) topology similar to a Hopfield network. Optionally, connections are cut, such as B. Sensory-to-Label (S2L) removed, C. Sensory-to-Sensory (S2S) removed, D. Both S2S and S2L removed. Arrows are bidirectional, meaning connections go both ways and do not enforce any hierarchy. Topologies on top of the adjacency matrix, where some connections are cut in white.

Table 2: Best (stable) IPC model, with $N = 1000$, for each topology evaluation on both classification accuracy on 1000 MNIST test images and digit generation. Some topologies are highly dependent on the learning rate parameters lr_x and lr_w and on weight initialization. All results trained on 10 epochs, but (generation) results were taken from epoch 1 of training, because of instability (on the generation task).

Topology	Class.	Generation
$A_{N=1000}$	0.88	
$A_{N=1}$	0.85	
B.	0.87	
C.	0.84	
D.	0.89	



(a) Averaged internal and sensory energy at the last inference step $t = T$, over a training period of 20 epochs for the models A-D from Table 2

(b) Top two images are reconstruction without providing the label, where the model's only information is the top part of the sensory information. The bottom image is the reconstruction of the occluded image provided with the label.

Figure 12: Reconstruction of a provided half of the sensory image while occluding the other part. Reconstruction can be done with and without a label, similar to associative memory tasks. Given these sensory inputs parts; what is the most likely other sensory part (and thus indirectly the most likely class label). Notice the top image where the *1* does not collapse to a single average image of the digit *1*. The model maps both variants to the same class label.

Results: The fully connected PC graph was used (as a proof-of-concept in [1]) due to its generality, and not to obtain the best performance. The fully connected model falls short when compared to hierarchical networks trained using either backpropagation (BP) or inference learning. Model depth, i.e., the number of stacking layers, is often considered essential for achieving strong performance in standard neural networks. Unlike the traditional feedforward MLP, this model does not contain any implicit hierarchy, which empirically appears crucial to obtaining good results. However, fully connected PC model training is more similar to how associative memory models work than a sequential model, where all nodes are used for the same mapping task. In the fully connected PC model case, the same node value can be used for both the generative and discriminative tasks. Nevertheless, these results show that this framework can learn an internal representation of a dataset and that a single model can be queried to solve multiple classification and generation tasks reasonably.

6.3.1 Value and weight Initialization

Before model inference, the values and predictions are initialized with either zeros or small normally distributed values to promote diversity in the learned model weights. Some implementations like in [43], [44], and [2] describe a feedforward initialization pass of latent states values before the inference step. This reduces the discrepancy between the initial latent states and their predictive targets and thus provides a good starting point for the inference phase, helping to avoid the unstable dynamics caused by poor initial conditions. It aligns the latent states more closely with expected values, smoothing the inference trajectory, minimizing significant updates, and yielding better results. The above feedforward initialization is only applicable to sequential predictive coding models that already perform reasonably well on the tested datasets.

Classical ML literature describes variance-preserving weight initializations (matched to the activation function) to avoid vanishing or exploding signals in deep feedforward networks like Xavier/Glorot for tanh/sigmoid, He (Kaiming) for ReLU-family activations [45–47]. To prevent the gradients of the network's activations from vanishing or exploding, traditional models adhere to the following rules: the mean of the activations should be zero, and the variance of the activations should stay in the same range across every layer.

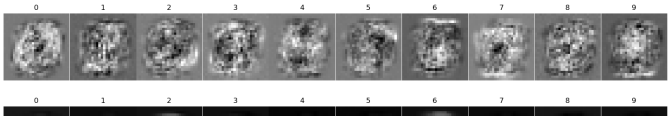
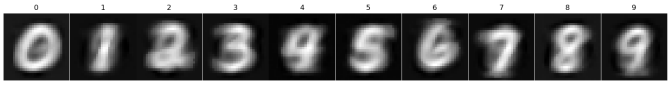
By maintaining these two conditions, the back-propagated gradient signal will not be multiplied by values that are too small or too large in any layer, allowing it to propagate to the input layer without exploding or vanishing. Authors of [48] showed with a convergence analysis that sequential PCNs only converge under certain conditions, such as small initializations, small weights, and appropriate step sizes. They also provide counterexamples where PCNs fail to converge due to nonlinear dynamics and bifurcations, causing for example, oscillations that produce values that never stabilize. For discrimination, we can use a standard MLP with weight initialization based on the layer size. For a generative predictive coding model, usually not modeled with an MLP-like architecture, we initialize the weights using a normal $\mathcal{N}(0, 0.05)$.

Similar fully connected energy-based models, such as Ising models and Boltzmann machines, typically sample weights from a zero-mean Gaussian distribution and enforce symmetry ($W_{ij} = W_{ji}$), which, under appropriate assumptions, provides theoretical convergence guarantees. Our fully connected PC model does not impose symmetry. We initialize weights by sampling from a zero-mean Gaussian with a small standard deviation or by setting a small constant value for all weights (usually 0.01 or 0.001) and adding a small stochastic deviation. Both choices keep initial predictions across internal nodes similar and help prevent large early error spikes that could destabilize inference.

6.4 Convergence instability

Predictive coding networks can converge to correct solutions that minimize prediction error, but may also settle in incorrect local minima. When convergence fails, the system may collapse; too large learning rates cause unstable inference and explosive values, while too small rates halt learning altogether. In some cases, the model learns a degenerate inverse representation (e.g., inverting black and white), leading to a collapse in which it minimizes the training loss via a trivial representation, yielding predictions that are locally optimal under the objective but lack semantic alignment with the data. Although [23] shows that Incremental PC (IPC) is less efficient due to the extra weight update at each step T, it is stable and yields better performance for hierarchical models. For fully connected models, Table 3 shows that FC can achieve similar accuracy results in hierarchical IPC, at the expense of its generative capabilities. In this case, changing the learning rate and ht decay parameters, we can change the behavior of the model.

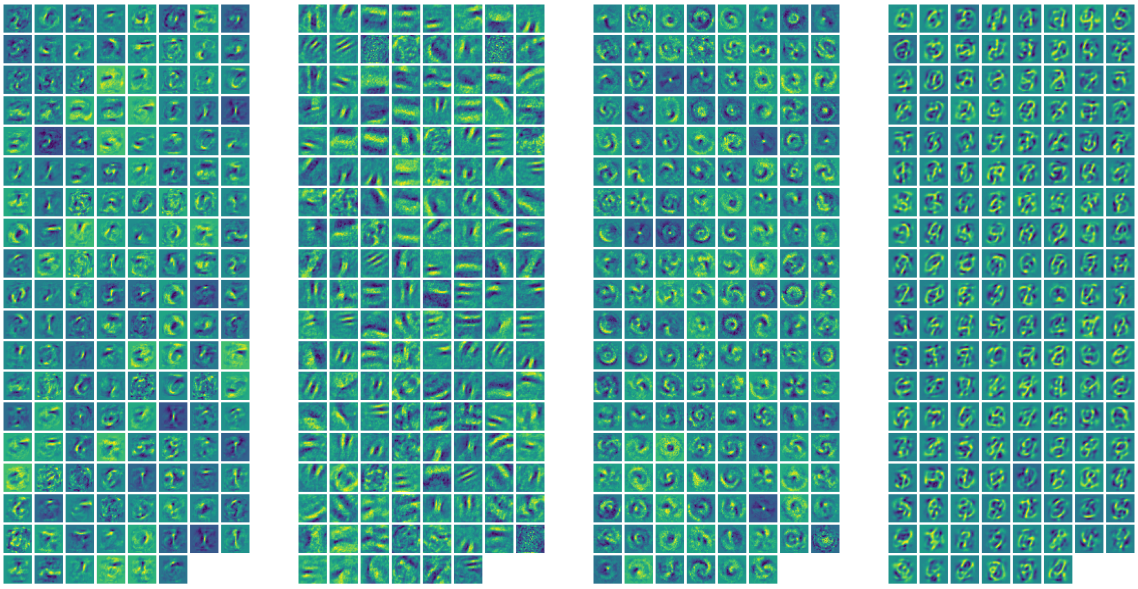
Table 3: Best unstable or partially collapsed IPC with 1000 internal nodes with bot Sensory-2-sensory and Sensory-2-Label connections removed. In the appendix below, we show the differences in PCA, std, and mean trajectory of the weight during training, highlighting the implications of the learning parameters on the system.

Classification	Generation
0.94	
0.84	

For this fully connected model, it seems challenging to perform well on both generation and classification tasks simultaneously. Research on the energy landscape of discriminative, generative, or both discriminative and generative hierarchical predictive coding models has been conducted by [49],which showsg cases of misalignment and the inability to perform both tasks. Nevertheless, the authors also showed using an alternative predictive coding approach that a Bidirectional predictive coding (bPC) model is capable of both top-down (generative) and bottom-up (discriminative) mappings. The model is biologically plausible (local errors and Hebbian weight updates), however uses two error neurons per value neuron and two error streams.

6.5 Learned weights

Predictive coding networks can develop features comparable to those learned by backpropagation [5, 7], including classical edge detectors and Gabor-like filters [12]. In our experiments, the learned weights adapted to the specific transformations present in the data, demonstrating PC's ability to self-organize around underlying visual symmetries without explicit architectural constraints.



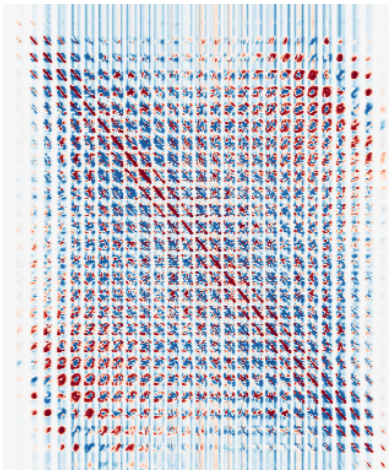
(a) Discrim. PC MNIST (b) Discrim. PC MNIST translation (c) Discrim. PC MNIST 90° rotation (d) Generative PC MNIST

Figure 13: Left: Discriminative PC with Rotation and zoom transforms on MNIST, mapping the 784 sensory nodes via $L = 150, 50$ hidden nodes to the 10 supervision label nodes. Plotting the 150 weights that connect the 784 sensory neurons to our first layer. Plotting the most abstract, but best interpretable, weights closest to the data. Accuracy drops with translation and rotation for PC and BP-trained discriminator models. But learning general features. Right: Generative PC weights after training with standard MNIST.

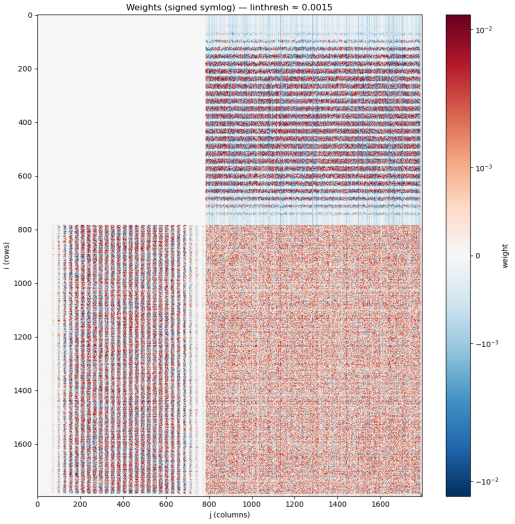
Discriminative PC trained on standard MNIST learned that CNNs like filters that average digits, similar to BP, as expected. For a dataset containing translated digits, the model independently learned the Fourier transform. This is an optimal solution because the Fourier transform diagonalizes shifts, allowing for easy translation by simply adjusting the phase of the Fourier components [50]. In a dataset with rotated and scaled digits, the model learned a Gabor transform. This transform, which can be conceptualized as a Fourier transform in a log-polar coordinate system, is effective at handling rotations and scaling. The model discovered these optimal representations without being explicitly programmed to do so, highlighting its ability to adapt to the data's underlying symmetries even when using PC.

Fully connected

The learned weights in a fully connected model are more difficult to interpret. Comparing the sensory-to-sensory weights, there is a resemblance to the weight components learned via an (un)restricted Boltzmann machine. Boltzmann machine is a member of the same family of models with similar (Hebbian) learning rules and architecture, given that it is fully connected with sensory-to-sensory connections. To display the fully connected weights, we use a symmetric logarithmic scale (symlog). This scale makes both small values near zeros and large values visible in the same image, allowing us to see fine details of weak weights while still capturing strong ones.



(a) A trained fully connected network focused on the sensory-2-sensory part, showing small patches are weighing and learning from the full sensory grid, similar to a correlation matrix



(b) A fully connected model with only hidden nodes, meaning no Sensory-2-Sensory and no Sensory-2-Label. Inhibitory and excitatory edge weights at the top and left of the plot show mappings for generative and discriminative tasks, while mappings between internal nodes does not capture any relationships.

Figure 14: In this model, the lack of incentives for internal-to-internal communication yields no visible structure.

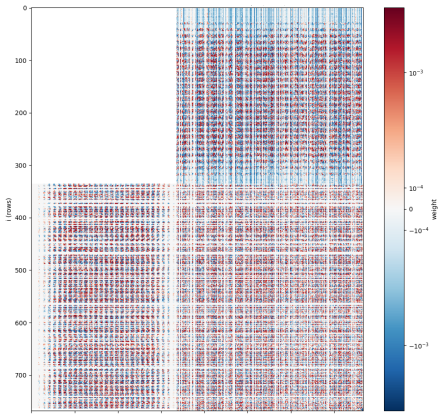
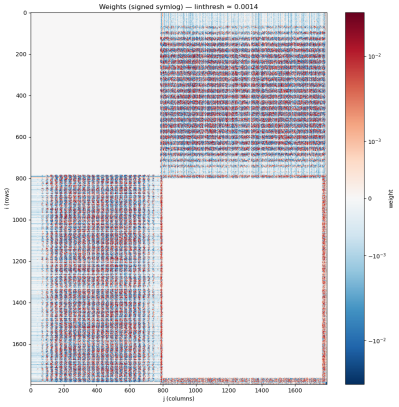


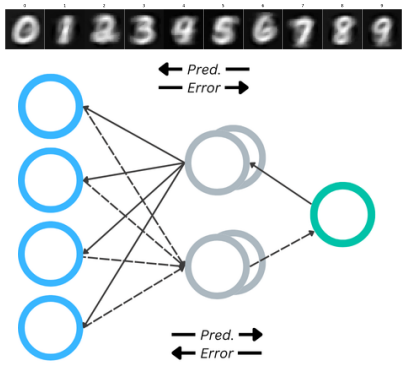
Figure 15: Trained Weight matrix using weight decay values $\lambda_v = 0$ (nodes values decay), $\lambda_w = 0.0001$ (weight decay), and value initialization of $\mathcal{N}(0, 1e{-}5)$. Initializing hidden node values during training and increasing weight decay λ, lr_x, and lr_w can impose some internal structure, without an increase in classification or generation results, suggesting these effects are random artifacts rather than dataset-driven structure.

Direct mapping

Starting from a fully connected predictive coding graph that includes direct sensory-to-label connections, energy minimisation naturally concentrates credit on the shortest error–reducing routes. When weights between sensory and supervision nodes are available, the gradient on these blocks is large. It rapidly drives a shallow one or two–hop solution that bypasses most internal nodes. This is not a consequence of predictive coding (or BP) but of the objective together with the topology that allows for shortcuts. Without topological or objective constraints, the fully connected predictive–coding model collapses to two effective branches: a near–direct mapping from images to labels and a near–direct mapping from labels to images. Internally, the network treats the clamped subsets as sources of prediction error and assigns credit to the couplings that most rapidly cancel these errors. In the presence of powerful sensory–label blocks, there is little incentive to form compositional features in deeper hidden assemblies. The outcome resembles a shallow associative link rather than a hierarchical representation.



(a) Inhibitory and excitatory edge weights at the top and left of the plot show mapping for generative and for discriminative tasks, while mapping between internal nodes is largely reduced.



(b) The resulting trained model looks like the figure above, where two distinct (discriminative and generative tasks) mapping are created. Final classification accuracy of 0.87 on 1000 test images.

Figure 16: starting with a fully connected graph and removing all but 10 internal nodes, forcing communication through these nodes

This architecture minimizes error in an efficient and less complex manner. As sensory nodes can directly relate to the labels, strong sensory-to-label connections allow the model to rapidly converge on expected outputs without needing deeper internal processing. This structure does not inherently promote feature learning in internal nodes, as internal connections may not significantly reduce error when direct connections are optimized. This is a problem because, within the hidden layer of an ANN, we learn intrinsic features of the data that generalize well beyond the training dataset, bypassing the internal hidden nodes with a single trivial connection, like in Figure 16b, where the model learns a map from x $\rightarrow$ y with a single layer of hidden nodes (similarly from $y \rightarrow x$).

A possible solution explored by [51], on causal inference using a different predictive coding framework involves presupposing the graph and inferring it directly from data. In this framework the sequential MLP-like model begins with a fully connected mapping x $\rightarrow$ y, discovers that this flat architecture underfits the classification tasks, and spontaneously inserts latent nodes and removes direct input-output mappings. Those intermediates separate low-level features from high-level labels, creating a hierarchical graph that both minimizes free energy and halves the test error. When the flat mapping $x \rightarrow y$ underfits (e.g. MNIST), it creates a pressure by introducing a sparsity and acyclicity penalty in the loss, causing the PC graph to spawn hidden nodes and forces itself to organize a two-layer hierarchy.

7 Stabilization and scalability PC models

One of its major strengths of PC is that each layer computation (both in the forward and backward pass) is local and, therefore, can be executed in parallel, removing one of the main bottlenecks of BP when training deep networks (where the computations induced by each layer have to be executed sequentially). Thus, we expect PC to scale well on large architectures or topologies, especially sparse graphs, when paired with a message-passing scheme implementation (see Appendix section E.1). However, in practice, it is hard to scale PC models due to model collapse or weak error propagation of local signals, which degrade performance as model depth increases. BP has the upper hand when scaling on models or datasets (for large dimension datasets (CIFAR-100 and bigger) as is shown by [52].First, we use generalization techniques at the optimizer level, tested on traditional ANNs and applied to PC models. Second, we explore different graph structures for efficient communication while simultaneously promoting feature learning.

7.1 Fast and slow updates

1. **memorization vs generalization**

Memorization or associative memory is generally simpler than true learning. A common way to distinguish the two is by observing discrepancies between training and test performance. However, this raises the question of whether the used testing procedure actually assesses generalization or rather memorization. For instance, in smaller datasets like MNIST, memorization can sometimes yield performance that appears "good enough." This leads to a foundational bias-variance trade-off question in machine learning: *"Do models memorize their training data (i.e., directly map inputs to outputs), or do they genuinely generalize to new, unseen examples?"*. In reality, both processes are essential, meaning that without memorization, a model (or brain) would need to relearn everything from scratch. Without generalization, it could not adapt to novel environments or tasks. The goal is therefore to balance retaining specific details through memorization with abstracting invariant features that support generalization. Humans likely follow a similar pathway: we start by memorizing patterns, then develop more abstract representations through exposure and learning.

2. Fast and slow learning

We know that inference and learning occur at different rates and time scales, as seen in the different learning rates lr_x and lr_w, reflecting the fact that we perform T inference steps and a single weight update during training in the traditional PC case. The difference between PC and IPC directly relates to the frequency of latent value updates with respect to weight updates, and thus requires different learning rate tuning. Given the latent state x_t/x_T and errors ϵ_t/ϵ_T (depending on PC or IPC) we update the weights that contain high-frequency and low-frequency gradients. Moving beyond the view of purely top-down or bottom-up driven predictions, the paper [53] uses a different predictive coding framework in which the model's neurons self-organize and divide themselves into error and prediction neurons over time. The authors demonstrate, via virtual lesioning experiments, that networks perform predictions on two timescales: fast lateral predictions among sensory units and slower prediction cycles that integrate evidence over time.

Authors of the paper [54] addresses how neural circuits can both control behavior and learn by combining fast and slow synaptic plasticity. Fast plasticity enables real-time behavioral adaptation, acting as a dynamic controller, while slow plasticity consolidates useful changes for long-term learning. The authors propose a two-speed learning rule and validate it in single-neuron and recurrent network models. Fast changes that immediately suppress downstream errors and slow changes that consolidate optimal learning. This setup reflects biological processes such as dopamine modulation, offering a concrete, stable solution for simultaneous learning and control.

The goal is to have an architecture that naturally supports fast-learning outer loops and slow-learning inner modules. This mimics the distinction between short-term adaptation and long-term consolidation in biological systems, as discussed similarly in the Forward-Forward Algorithm in [55]. In certain (survival) cases, it might be beneficial to have fast-working initial inference guesses or reflexes that can be adjusted over time.

Experiments from [56] using neural pixels in mice, as well as observations in macaques and humans, demonstrate a *time scale hierarchy*, where the response time of a cortical area increases with its hierarchical position. This is a crucial feature of the cortex, as it enables sensory areas to respond quickly, while higher-order areas can slow down for complex functions like decision-making.

In predictive coding, (ascending) prediction errors are transmitted via high-frequency gamma oscillations from superficial layers, while descending predictions use low-frequency alpha and beta oscillations from deep layers. This spectral asymmetry shown by [57] reflects functional roles: nonlinear generation of prediction errors amplifies high frequencies, while linear accumulation into predictions filters them out, favoring slower rhythms.

3. Other works

Hybrid predictive coding learning, introduced by [58], combines the strengths of both approaches: fast and slow learning. Their approach uses (standard) iterative predictive coding with amortised inference, so that both bottom-up and top-down signals convey predictions (and where prediction errors also flow in both directions).

The paper [44] introduces improvements to Inference Learning where the authors address its high computational cost and convergence issues by (1) proposing *sequential inference* to speed up error propagation and reduce computational load, (2) introducing a novel low-memory optimizer called Matrix Update Equalization (*MQ-optimizer*) that balances weight updates without the need for Adam, and (3) offering theoretical insights connecting IL to implicit stochastic gradient descent and second-order optimization. These enhancements enable IL to match or exceed backpropagation's performance on image classification and autoencoding tasks, while being more memory-efficient and biologically plausible.

When performing message passing for probabilistic inference, tricks from (loopy) Belief Propagation, such as damping (also called relaxation), can be used to stabilize the algorithm. In graphs with loops, the same messages can circle back and reinforce each other, leading to oscillations or non-convergence. Instead of replacing each message with the new update, they damp it with the previous one, turning volatile updates into gradual, steady changes, making convergence far more likely on loopy graphs.

4. Grokfast

The grokking phenomenon, or delayed generalization, is described as a lazy training regime of a (language) model in which memorization is the dominant force of learning before abruptly transitioning from overfitting to generalization. The authors of [59] accelerate the grokking phenomenon by amplifying low frequencies of the parameter gradients with an augmented optimizer. They make a key assumption about machine learning models in which parameter updates occur on two different timescales: *Fast-varying parameters* that are chaotic and specific to a given batch, adjusting their values rapidly, resulting in overfitting of the training data. *Slow-varying parameters,* that do not vary much with each batch, learn the consistent patterns in the data slowly, and contribute massively to the actual generalization of the data. The authors assume that the reason behind grokking is the delayed generalization of the slow-varying parameters in the data. This technique keeps track of the average gradient direction over time (using exponential smoothing) and gently pushes the model in that long-term direction at each update. The idea is to amplify the functional, slow-changing parts of the learning signal, which are believed to guide better generalization. The authors claim that omitting faster-moving gradients completely leads to slower, more unstable training. So the fast-moving gradients do serve a purpose. In our case, direct mapping from data to label nodes is fast, variable, trivial, and non-generalizable, and thus requires no distinct feature learning. These weight updates are noisy but faster, while internal weight connections are learned more slowly but generalize better.

Discriminative IPC model: Recreating a training setup to induce showing delayed generalization (i.e., grokking), inspired by prior work [60], Using AdamW on a discirminiaive IPC with hidden layers $L = \{200, 200\}$ with a training dataset of 1000 images[2]. We recreated a similar training scheme that overfitted on the training set, yet showed no delayed generalization in test accuracy, leading to stagnation (even after 150 epochs).

Grokfast fails to induce generalization of the validation data set in our IPC model setup. However, both training and validation accuracy increase in parallel when hidden values are randomly initialized before inference, suggesting that our optimizer and inference dynamics may inherently stabilize learning without requiring gradient amplification.

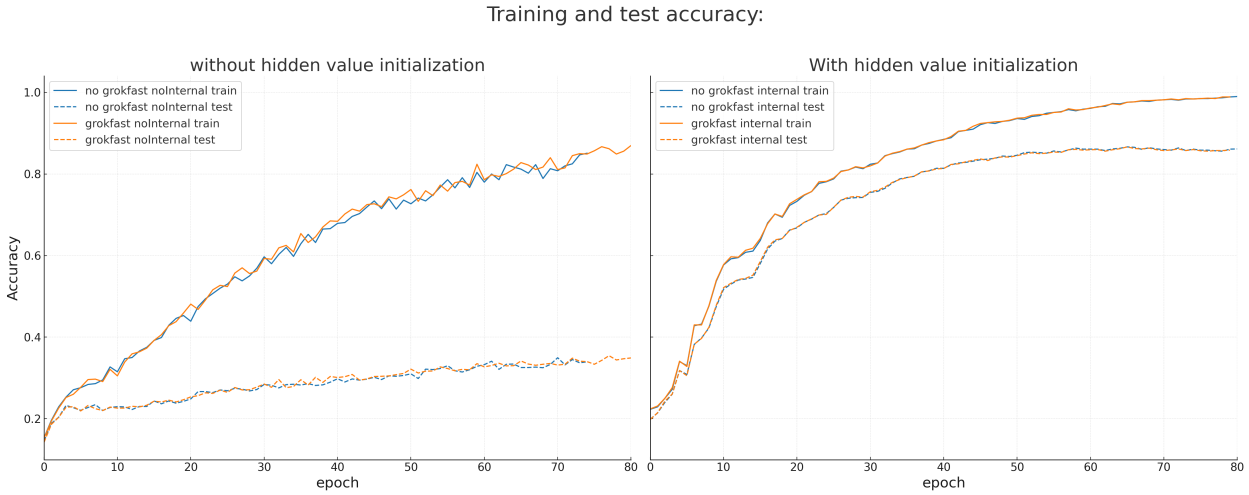


Figure 17: Discriminative IPC training on 1000 samples showing Grokfast alone does not improve test accuracy.

Fully connected For fully connected predictive coding models, the use of Grokfast was negligible on model performance. Accuracy and generative capability did not improve due to the fully connected starting topology being the main bottleneck.

[2]In the overfit training setup described by [60], $8x$ kaiming weight initialization is omitted due to model instability

7.2 Graph topology

The expressiveness of the PC model is limited by the expressiveness of the type of message passing and the graph architecture. Above, we showed the limitation of starting with a fully connected graph and trying to learn a meaningful internal hidden-node structure that captures the dataset. Another approach, used by [51], creates a training system to build the hierarchy layer-wise. However, this is limited to directed acyclic graphs, relies on priors and assumptions, and is primarily validated in MNIST-scale setups.

In our credit assignment problem, the sensory observation node x and supervision node y, and thus the shortest path length between these nodes, must fall within T, the number of (hop) steps between nodes. According to our model definition, the signal propagates through the errors and is limited by T. Forcing T to be small constrains the graph to learn only short-range connections, blocking learning.

DisGen PC

Combining the topology of the discriminative and generative PC, both using two dense connected layers with each $L_0 = L_1 = \{150, 50\}$ nodes. Any lateral connections between the two separate branches are omitted, effectively training two smaller, distinct models simultaneously. The initial parameters used for the discriminative and generative PC models differ, leading the DisGen model to perform worse than the individual models.

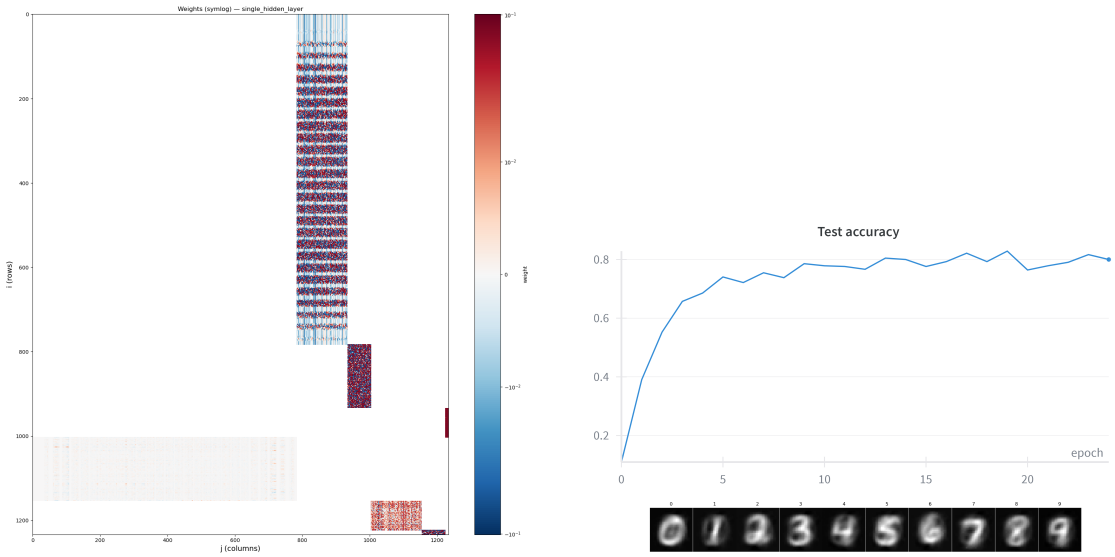


Figure 18: Left: The DisGen PC model with both branches having $L = \{150, 50\}$ internal nodes, and using $T_{train} = T_{test} = 100$ making sure the latent node values have converged. After training, most model weights are focused on the generative branch on top Right: Classification on test MNIST and generative task

Random SBM

The first step is combining the generative and discriminative PC models into a single network with shared sensory and supervision nodes, but distinct non-interactive layers for the generative and discriminative parts. Taking inspiration from the human connectome implemented with weighted stochastic block models used as wiring/weight priors for the graph shown by [61]. We experiment with a random stochastic block model in which clusters can act as small building blocks that store parts of a memory or stimulus. The graph is treated as a classical directed SBM with one sensory block, K (equal-sized) internal communities, and a supervision block where edges are sampled with intra-community probability p_{intra} and inter-community probability p_{inter}. Collectively, these prior architecture choices reflect a trade-off between density inside communities and clustered sparsity between communities.

The tested model was initialized with $p_{intra} = 0.25$, $p_{inter} = 0.1$, and 5 clusters of 100 internals each, yielding a classification accuracy of 0.81. Increasing the total number of internal nodes to 6000 only improves classification accuracy, while the generative capability degrades, similar to the fully connected case.

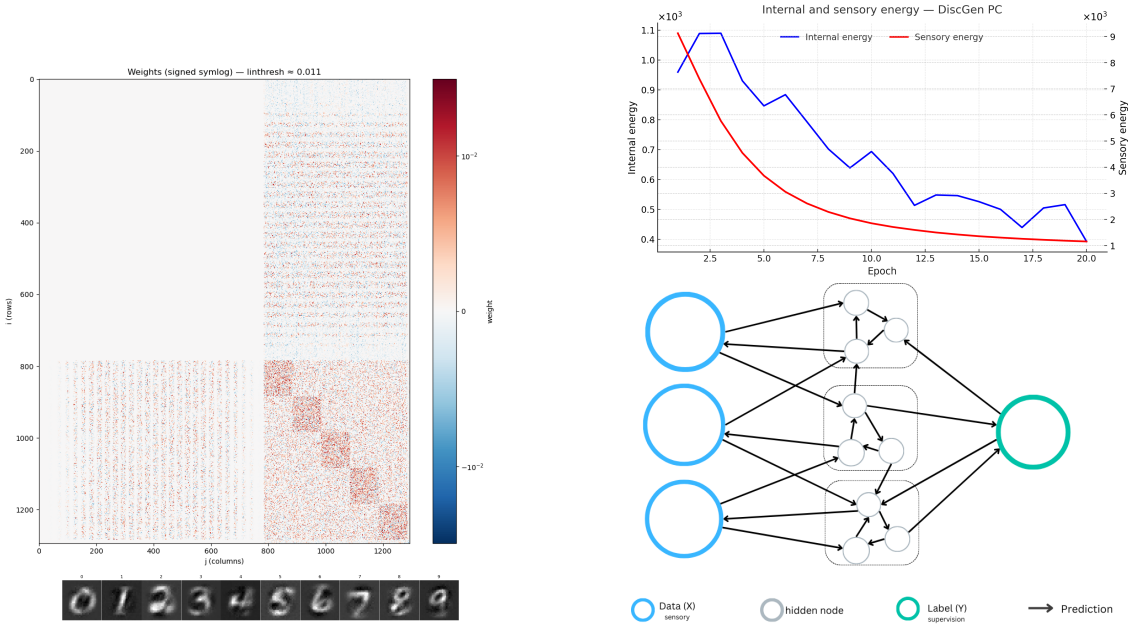


Figure 19: Left: The Random SBM PC model with five internal clusters of 100 nodes each. Below is the evaluation of the generative task with this model. Right: The internal and sensory energy over the training duration. Below is an illustration of the connections in this random SBM model.

Two way-Layered SBM

To scale, we initialize the branches layer by layer and impose a hierarchy that prevents direct mapping, as in the fully connected model. This topology can be characterized by the following: First, the *Two-way branched*, referring to the use of a branch for the discriminative tasks ($x \rightarrow H_{discrim_1} \rightarrow \cdots \rightarrow H_{discrim_N} \rightarrow y$) and a branch for the generative tasks ($x \leftarrow H_{gen_1} \leftarrow \cdots \leftarrow H_{gen_N} \leftarrow y$). To initialize more like standard sequential model a hierarchical Layering is used to force signal propagation through layers and promote feature learning at different scales. Instead of using dense layers, we use multiple stochastic blocks, similar to the SBM topology above, as a single layer of neurons. Wiring from one layer to the next is determined by p_{inter}, allowing for tunable modularity by increasing the number of layers (hop length) or clusters while inducing sparse connections. The pixel values of the MNIST $(28, 28)$ image are used as input to the sensory nodes, where neighboring nodes are grouped into overlapping or non-overlapping $(4, 4)$ patches. The patches from each layer are pooled into similarly constructed, sequential layers within the same branch.

Using this architecture, we can increase the model's depth (number of layers), width (layer size), and clustering inside each layer, while tuning edge sparsity and signal efficiency. We experimented with this architecture design with 14.000 nodes with over 2 million total edges.

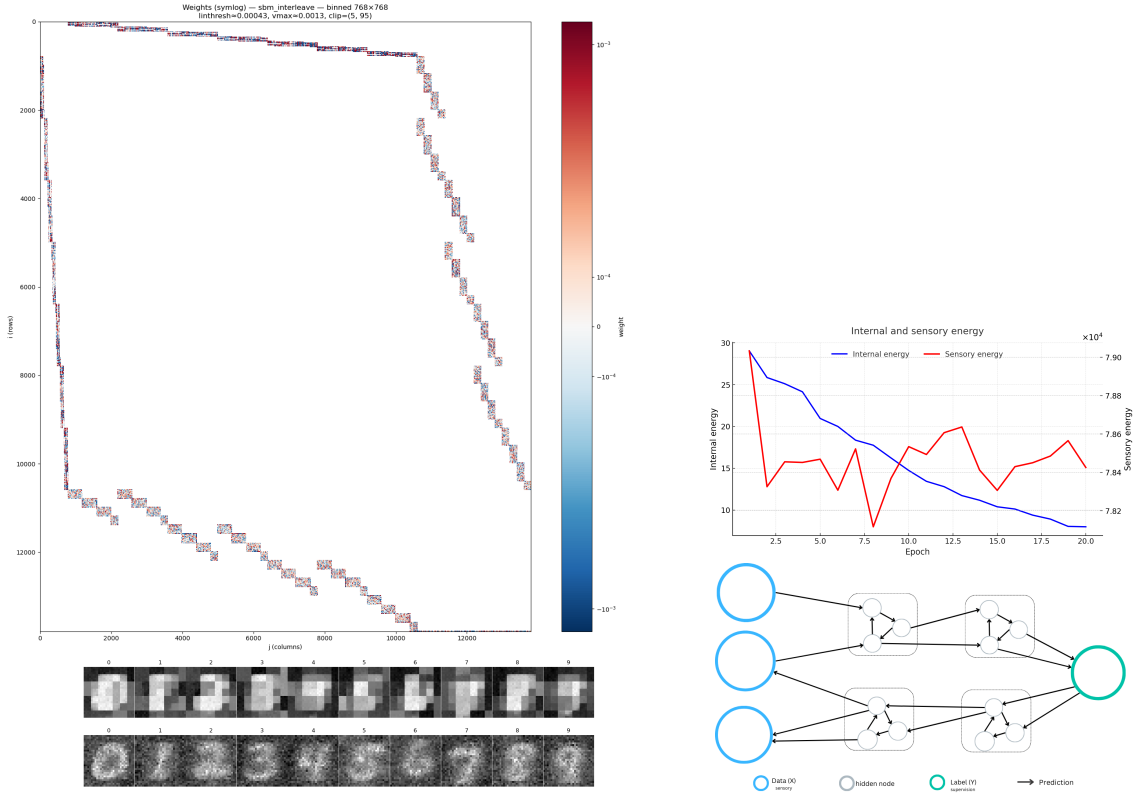


Figure 20: Left: Graph topology of the SBM model with 14k nodes and over 2 million edges: The generative output granularity is highly dependent on the number of clusters in the first layer, allowing it to capture information of all 4x4 patches. The classification test accuracy of 0.78 indicates that scaling up the topology (while keeping sparse connections) does not affect the classification metrics. Right: The energy of the internal nodes during training. The graph topology is shown below, with the patching mechanism omitted. Both branches have two layers, each with a single cluster.

8 Conclusion

We formalized PC on arbitrary directed graphs, clarified notation across the top-down and bottom-up conventions, and derived local inference and weight-update rules that require only neighborhood information, unifying inference learning (IL) and its incremental variant (IPC). This framing treats PC as a single energy-minimization process that can be instantiated across many topologies and queried for multiple tasks within the same model.

Although our findings are restricted to the MNIST dataset, PC/IPC training appears sensitive to hyperparameters such as initialization, step sizes, and the number of steps T. Shallow hierarchical PC/IPC models reached competitive classification accuracy, but simply adding depth did not reliably improve performance. Instead, deeper stacks increased the divergence between PC/IPC and BP updates (larger Δw), indicating noisier error propagation, convergence problems, and less stable training as graphs increase layers. Beyond discrimination, we used the same PC graphs in a generative manner by clamping labels and inferring pixel values. While the approach produced plausible digits, controllability was limited, and the dynamics were fragile to hyperparameters. Despite these limitations, the learned representations for sequential PC models were often meaningful. Under translation and rotation variants of MNIST, discriminative PC discovered filters that diagonalize shifts and demonstrate that PC can self-organize around task symmetries without architectural hand-crafting, similar to BP. In fully connected graphs, a single PC model could both classify and generate. Still, credit assignment tended to flow through the shortest paths, encouraging near-direct mappings that bypass richer internal structure.

In network design, backpropagation is limited to predominantly feedforward (and recurrent) structures for stable operation. In contrast, predictive coding can train arbitrary graphs and even use one network for multiple tasks or modalities simultaneously. This indicates a potential path forward for AI: by leveraging Inference learning, we might incorporate more brain-like connectivity (recurrent loops, multi-sensory integration within a single network, etc.) that backpropagation would struggle with. First, as a learning paradigm, PC is not merely a quirky approximation to BP: it trades exact global gradients for strict biological plausibility, parallel activity and weight updates, and the ability to operate on cyclic, multi-functional graphs. However, it should be noted that PC remains sensitive to initialization parameters and step sizes.

9 Future work

The strengths of PC lie primarily in locality, graph flexibility, multi-modal inputs, and bidirectional use cases (classification, generation, completion) within a single network. The path forward is therefore not to force PC into matching BP gradients exactly, but to pair it with topology and objectives that make use of those strengths: e.g., connectome-inspired priors or regularizers that push the graph toward sparse, layered substructures; penalties that disfavor *shortcut* edges; and skip-connected designs that preserve short effective credit paths.

1). Model with prior (connectome) topology: We propose to move beyond generic fully connected graphs by constraining the model's topology and, where appropriate, its initial weights using connectome data. Recent work shows that a continuous-state Hopfield network whose weights are set directly from fMRI-derived functional connectivity exhibits a low-dimensional energy landscape with reproducible attractor states and realistic dynamics, providing a concrete *form follows function* recipe for turning brain measurements into model structure. This suggests an immediate path for predictive-coding (PC) graphs and Hopfield-like models where the graph is initialized (or regularized) with a connectome before studying how inference and learning reshape the energy landscape, similar to other works [62, 63].

2). Alter lost function: Ways to alter the error function of our predictive coding model to penalize structural characteristics or promote hierarchy. Weight Regularization Mechanism, such as an $\ell 2$-penalty on weights (implemented via the weight decay parameter), is used to restrain excessive weight growth, encouraging smoother and more generalizable representations, indirectly altering the energy landscape. Work by [64] provides a unified framework linking predictive coding and sparse coding, where it discusses how the sparse coding objective (typically an ℓ_1 penalty) maps to predictive coding dynamics and tests model predictions for neural nonlinearities. A method to directly alter the energy function is *Power-Law (Scale-Free) Degree Regularization*, which encourages a heavy-tailed, power-law degree distribution by penalizing nodes with low degree more, allowing a few hub nodes to emerge with many connections [3].

$$E = \tfrac{1}{2} \sum_i \epsilon_i^2 + \gamma \sum_j k_j^{-\alpha}$$

To make the graph both compact and cycle-free, the author of [51] augments the free-energy loss with two regularizers: pushing the network toward a minimal, feed-forward hierarchy whose layers emerge solely from the twin pressures of *keep it sparse* and *keep it acyclic*, ending in a two-layer Directed Acyclic Graph

3). Dynamical graph: For structure learning, we can take advantage of locally stored errors in the PC model. Methods like NEAT, GradMAX, or Firefly, not exclusive to PC, can use pockets of high error or strong gradients to grow the graph where it matters, adding connections within clusters or layers and linking high- and low-error regions to dissipate mismatch. This can encourage (power-law–like) growth with paired pruning, allowing nodes and connections to expand or shrink as needed. After initial training, the model can adapt to novel stimuli by creating new supervision nodes and internal clusters (akin to liquid neural networks) and, at a later stage, pruning unused parts, effectively *forgetting* unneeded information.

To find a good network topology, one can use search methods such as neural architecture search (for example, NASWOT, which does not require training), simulated annealing, or evolutionary methods like NEAT. Another option is to learn the graph directly from data. For directed acyclic graphs, NOTEARS treats structure learning as a continuous optimization problem [65]. In causal inference, predictive coding has also been used to help align model topology with the data [51].

[3] Degree regularization with $k_j = \sum_i \mathbf{1}(W_{ij} \neq 0)$ denoting the degree of node j; α controls the steepness of the power-law decay (how strongly hubs dominate), and γ sets the overall regularization strength.

References

[1] Tommaso Salvatori, Luca Pinchetti, Beren Millidge, Yuhang Song, Tianyi Bao, Rafal Bogacz, and Thomas Lukasiewicz. Learning on arbitrary graph topologies via predictive coding. *Advances in neural information processing systems*, 35:38232–38244, 2022.

[2] Björn van Zwol, Ro Jefferson, and Egon L Broek. Predictive coding networks and inference learning: Tutorial and survey. *arXiv preprint arXiv:2407.04117*, 2024.

[3] Beren Millidge, Anil K. Seth, and Christopher L. Buckley. Predictive coding: A theoretical and experimental review. *arXiv preprint arXiv:2107.12979*, 2021. doi: 10.48550/arXiv.2107.12979. URL `https://arxiv.org/abs/2107.12979`.

[4] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications*, 11(1):3625, 2020. doi: 10.1038/s41467-020-17236-y.

[5] Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182*, 2020.

[6] 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.

[7] 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):1229–1262, 2017. doi: 10.1162/NECO_a_00949.

[8] Tommaso Salvatori, Yuhang Song, Yujian Hong, Simon Frieder, Lei Sha, Zhenghua Xu, Rafal Bogacz, and Thomas Lukasiewicz. Associative memories via predictive coding. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/1fb36c4ccf88f7e67ead155496f02338-Abstract.html`.

[9] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[10] 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.

[11] Mikko Stenlund. Introduction to predictive coding networks for machine learning. *arXiv preprint arXiv:2506.06332*, 2025.

[12] 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. doi: 10.1038/4580.

[13] Matthias Brucklacher, Sander M Bohté, Jorge F Mejias, and Cyriel MA Pennartz. Local minimization of prediction errors drives learning of invariant object representations in a generative network model of visual perception. *Frontiers in Computational Neuroscience*, 17:1207361, 2023.

[14] Gaspard Oliviers, Mufeng Tang, and Rafal Bogacz. Bidirectional predictive coding, 2025. URL `https://arxiv.org/abs/2505.23415`.

[15] Jakob Hohwy. The self-evidencing brain. *Noûs*, 50(2):259–285, 2016. doi: 10.1111/nous.12062.

[16] Karl Friston, James Kilner, and Lee Harrison. A free energy principle for the brain. *Journal of Physiology-Paris*, 100(1-3):70–87, 2006. doi: 10.1016/j.jphysparis.2006.10.001.

[17] Karl Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010. doi: 10.1038/nrn2787.

[18] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, John O'Doherty, and Giovanni Pezzulo. Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68:862–879, 2016. doi: 10.1016/j.neubiorev.2016.06.022.

[19] David C. Knill and Alexandre Pouget. The bayesian brain: The role of uncertainty in neural coding and computation. *Trends in Neurosciences*, 27(12):712–719, 2004. doi: 10.1016/j.tins.2004.10.007.

[20] Helen C. Barron, Ryszard Auksztulewicz, and Karl Friston. Prediction and memory: A predictive coding account. *Progress in Neurobiology*, 192:101821, 2020. ISSN 0301-0082. doi: 10.1016/j.pneurobio.2020.101821.

[21] Karl Friston. A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456):815–836, 2005. doi: 10.1098/rstb.2005.1622.

[22] David C Knill and Alexandre Pouget. The bayesian brain: the role of uncertainty in neural coding and computation. *TRENDS in Neurosciences*, 27(12):712–719, 2004.

[23] Tommaso Salvatori, Yuhang Song, Yordan Yordanov, Beren Millidge, Zhenghua Xu, Lei Sha, Cornelius Emde, Rafal Bogacz, and Thomas Lukasiewicz. A stable, fast, and fully automatic learning algorithm for predictive coding networks. *arXiv preprint arXiv:2212.00720*, 2022.

[24] Beren Millidge, Anil K. Seth, and Christopher L. Buckley. Predictive coding: A theoretical and experimental review. *arXiv preprint arXiv:2107.12979*, 2021. doi: 10.48550/arXiv.2107.12979.

[25] Karl Friston, Jérémie Mattout, Nelson Trujillo-Barreto, John Ashburner, and Will Penny. Variational free energy and the laplace approximation. *NeuroImage*, 34(1):220–234, 2007. doi: 10.1016/j.neuroimage.2006.08.035.

[26] Gaspard Oliviers, Rafal Bogacz, and Alexander Meulemans. Learning probability distributions of sensory inputs with monte carlo predictive coding. *bioRxiv*, 2024. doi: 10.1101/2024.02.29.581455.

[27] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014. doi: 10.48550/arXiv.1312.6114. arXiv:1312.6114.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39 (1):1–38, 1977. doi: 10.1111/j.2517-6161.1977.tb01600.x.

[29] Radford M. Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In Michael I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, Dordrecht, 1998. doi: 10.1007/978-94-011-5014-9_12.

[30] Sebastian Gottwald and Daniel A. Braun. The two kinds of free energy and the bayesian revolution. *PLOS Computational Biology*, 16(12):e1008420, 2020. doi: 10.1371/journal.pcbi.1008420.

[31] Christopher L. Buckley, Chang Sub Kim, Simon McGregor, and Anil K. Seth. The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*, 81:55–79, 2017. doi: 10.1016/j.jmp.2017.09.004.

[32] Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017. doi: 10.1016/j.jmp.2015.11.003.

[33] Nick Alonso. Predictive coding: A brief introduction and review for machine learning researchers. URL https://neuralnetnick.com/2022/12/28/predictive-coding-a-brief-introduction-and-review-for-machine-learning-researchers/.

[34] Björn van Zwol. Predictive coding graphs are a superset of feedforward neural networks. In *The First Workshop on NeuroAI@ NeurIPS2024*.

[35] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Cambridge, MA, 2001.

[36] Panayiota Poirazi, Terrence Brannon, and Bartlett W. Mel. Pyramidal neuron as two-layer neural network. *Neuron*, 37(6):989–999, 2003. doi: 10.1016/S0896-6273(03)00149-1.

[37] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006. ISBN 978-0-387-31073-2.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. ISBN 978-0262035613. URL https://www.deeplearningbook.org/.

[39] Yann A. LeCun, Léon Bottou, Geneviève B. Orr, and Klaus-Robert Müller. Efficient backprop. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012. doi: 10.1007/978-3-642-35289-8_3.

[40] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Cambridge, UK, 2002. doi: 10.1017/CBO9780511815706.

[41] Björn van Zwol. Predictive coding graphs are a superset of feedforward neural networks. In *The First Workshop on NeuroAI @ NeurIPS2024*, 2024. URL https://openreview.net/forum?id=J36z3R0sNq.

[42] T Anderson Keller and Max Welling. Predictive coding with topographic variational autoencoders. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1086–1091, 2021.

[43] Myoung Hoon Ha, Yoondo Sung, Youngha Jo, Hyunjun Kim, and Sang Wan Lee. Towards stable learning in predictive coding networks, 2025. URL https://openreview.net/forum?id=FwdN0KovFp.

[44] Nicholas Alonso, Jeffrey Krichmar, and Emre Neftci. Understanding and improving optimization in predictive coding networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10812–10820, 2024.

[45] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. URL https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf.

[47] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147. PMLR, 2013. URL https://proceedings.mlr.press/v28/sutskever13.html.

[48] Simon Frieder and Thomas Lukasiewicz. (non-)convergence results for predictive coding networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6793–6810. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/frieder22a.html.

[49] Gaspard Oliviers, Mufeng Tang, and Rafal Bogacz. Bidirectional predictive coding. *arXiv preprint arXiv:2505.23415*, 2025. doi: 10.48550/arXiv.2505.23415. URL https://arxiv.org/abs/2505.23415.

[50] Ho Yin Chau, Frank Qiu, Yubei Chen, and Bruno Olshausen. Disentangling images with lie group transformations and sparse coding. *arXiv preprint arXiv:2012.12071*, 2020.

[51] Tommaso Salvatori, Luca Pinchetti, Amine M'Charrak, Beren Millidge, and Thomas Lukasiewicz. Predictive coding beyond correlations. *arXiv preprint arXiv:2306.15479*, 2023.

[52] Luca Pinchetti, Chang Qi, Oleh Lokshyn, Gaspard Oliviers, Cornelius Emde, Mufeng Tang, Amine M'Charrak, Simon Frieder, Bayar Menzat, Rafal Bogacz, Thomas Lukasiewicz, and Tommaso Salvatori. Benchmarking predictive coding networks – made simple. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2407.01163. Published as a conference paper at ICLR 2025.

[53] Abdullahi Ali, Nasir Ahmad, Elgar de Groot, Marcel Antonius Johannes van Gerven, and Tim Christian Kietzmann. Predictive coding is a consequence of energy efficiency in recurrent neural networks. *Patterns*, 3(12), 2022.

[54] Brendan A Bicknell and Peter E Latham. Fast and slow synaptic plasticity enables concurrent control and learning. March 2025. doi: 10.7554/elife.105043.1. URL http://dx.doi.org/10.7554/eLife.105043.1.

[55] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022. URL https://arxiv.org/abs/2212.13345.

[56] Guanchun Li, Songting Li, and Xiao-Jing Wang. A hierarchy of time constants and reliable signal propagation in the marmoset cortex. *bioRxiv*, pages 2025–05, 2025.

[57] Ryszard Auksztulewicz and Karl Friston. Repetition suppression and its contextual determinants in predictive coding. *Cortex*, 80:125–140, 2016. ISSN 0010-9452. doi: https://doi.org/10.1016/j.cortex.2015.11.024. URL https://www.sciencedirect.com/science/article/pii/S0010945216000101. Special Issue:Repetition suppression-an integrative view.

[58] Alexander Tscshantz, Beren Millidge, Anil K Seth, and Christopher L Buckley. Hybrid predictive coding: Inferring, fast and slow. *PLoS computational biology*, 19(8):e1011280, 2023.

[59] Jaerin Lee, Bong Gyun Kang, Kihoon Kim, and Kyoung Mu Lee. Grokfast: Accelerated grokking by amplifying slow gradients. *arXiv preprint arXiv:2405.20233*, 2024.

[60] Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. *arXiv preprint arXiv:2210.01117*, 2022. doi: 10.48550/arXiv.2210.01117. URL https://arxiv.org/abs/2210.01117.

[61] Joshua Faskowitz, Xiaoran Yan, Xi-Nian Zuo, and Olaf Sporns. Weighted stochastic block models of the human connectome across the life span. *Scientific Reports*, 8(1):12997, 2018. doi: 10.1038/s41598-018-31202-1.

[62] Robert Englert, Balint Kincses, Raviteja Kotikalapudi, Giuseppe Gallitto, Jialin Li, Kevin Hoffschlag, Choong-Wan Woo, Tor D. Wager, Dagmar Timmann, Ulrike Bingel, and Tamas Spisak. Connectome-based attractor dynamics guide brain activity in rest, task, and disease. *bioRxiv*, 2023. doi: 10.1101/2023.11.03.565516. URL https://www.biorxiv.org/content/10.1101/2023.11.03.565516v2.

[63] Stefan Frässle, Ekaterina I. Lomakina, Lars Kasper, Zina M. Manjaly, Alex Leff, Klaas P. Pruessmann, Joachim M. Buhmann, and Klaas E. Stephan. A generative model of whole-brain effective connectivity. *NeuroImage*, 179:505–529, 2018. doi: 10.1016/j.neuroimage.2018.05.058.

[64] Yanbo Lian and Anthony N Burkitt. Relating sparse and predictive coding to divisive normalization. *PLOS Computational Biology*, 21(5):e1013059, 2025.

[65] Xun Zheng, Bryon Aragam, Pradeep K. Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, volume 31, pages 9492–9503, 2018. URL https://proceedings.neurips.cc/paper/2018/file/e347c51419ffb23ca3fd5050202f9c3d-Paper.pdf.

[66] James C. R. Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, 2019. doi: 10.1016/j.tics.2018.12.005. URL https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(19)30012-9.

[67] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7:13276, 2016. doi: 10.1038/ncomms13276. URL https://www.nature.com/articles/ncomms13276.

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

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

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

[71] Beren Millidge, Alexander Tschantz, Anil K. Seth, and Christopher L. Buckley. Relaxing the constraints on predictive coding models. *arXiv preprint arXiv:2010.01047*, 2020. URL https://arxiv.org/abs/2010.01047.

[72] Alexander Ororbia, Ankur Mali, Daniel Kifer, and C Lee Giles. Large-scale gradient-free deep learning with recursive local representation alignment. *arXiv preprint arXiv:2002.03911*, 2020.

[73] Alexander G Ororbia and Ankur Mali. Biologically motivated algorithms for propagating local target representations. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4651–4658, 2019.

[74] Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. *arXiv preprint arXiv:2201.11665*, 2022. doi: 10.48550/arXiv.2201.11665. URL https://arxiv.org/abs/2201.11665.

[75] Yang Shen, Julia Wang, and Saket Navlakha. A correspondence between normalization strategies in artificial and biological neural networks. *Neural Computation*, 33(12):3179–3203, 2021. doi: 10.1162/neco_a_01439.

[76] Mohamed Akrout, Collin Wilson, Peter C. Humphreys, Timothy Lillicrap, and Douglas Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://papers.neurips.cc/paper/8383-deep-learning-without-weight-transport.pdf.

[77] Bernd Illing, Jean Ventura, Guillaume Bellec, and Wulfram Gerstner. Local plasticity rules can learn deep representations using self-supervised contrastive predictions. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/file/feade1d2047977cd0cefdafc40175a99-Paper.pdf.

[78] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fec87a37cdeec1c6ecf8181c0aa2d3bf-Abstract.html.

[79] Beren Millidge, Alexander Tschantz, and Christopher L Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *Neural Computation*, 34(6):1329–1368, 2022.

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

[81] Jeff Orchard, Wei Sun, and Neil Liu. Why aren't all predictive coding networks generative? In *Neural Information Processing Systems*, 2019.

[82] Matthias Brucklacher, Sander M. Bohté, Jorge F. Mejias, and Cyriel M. A. Pennartz. Local minimization of prediction errors drives learning of invariant object representations in a generative network model of visual perception. *Frontiers in Computational Neuroscience*, 17:1207361, 2023. doi: 10.3389/fncom.2023.1207361.

[83] T. Anderson Keller and Max Welling. Predictive coding with topographic variational autoencoders. In *ICCV 2021 Workshops (VIPriors)*, pages 1086–1091, 2021. URL https://openaccess.thecvf.com/content/ICCV2021W/VIPriors/html/Keller_Predictive_Coding_With_Topographic_Variational_Autoencoders_ICCVW_2021_paper.html.

[84] Billy Byiringiro, Tommaso Salvatori, and Thomas Lukasiewicz. Robust graph representation learning via predictive coding. *arXiv preprint arXiv:2212.04656*, 2022.

Appendix

The following appendix provides supplementary material that supports the main text of this document. It contains additional explanations, extended derivations, and detailed data that were omitted from the main chapters to maintain readability. In particular, it includes a direct comparison between BP and PC, along with details of the neural inference experiment. Furthermore, the appendix provides an overview of the hyperparameters used in Section E.1 and a description of the pseudocode in Section F.

A BP and PC

Predictive coding (PC) differs fundamentally from standard backpropagation (BP) in both algorithmic procedure and biological plausibility. PC is often described both as an algorithm and as a neural architecture. In this context, we treat PC as a (recurrent) network architecture with local learning rules, in contrast to BP, which is a specific algorithm (error backpropagation) for training layered networks. Below, we outline key differences and connections between BP and PC:

A.1 BP vs PC

Backpropagation (BP) and predictive coding (PC) differ fundamentally in how they compute and propagate error signals for learning. BP is a two-phase procedure: a *forward pass* first maps inputs to outputs through the network, and a *backward pass* then propagates the output error gradient through each layer to adjust weights. This backward phase uses the chain rule to compute the gradient of each weight, requiring a global error signal that is sequentially propagated from the output layer to earlier layers. Every weight update in BP implicitly depends on the computations of downstream layers, tying the whole network together in the credit assignment process. In contrast, PC uses an iterative, unified inference and learning process rather than distinct forward and backward passes. The PC update dynamics typically involve running model inference until the network's prediction errors converge to a minimum, then performing a local weight update to minimize a global free energy objective in two alternating steps. PC, on the other hand, uses strictly local learning rules. Each synapse's update in a PC network depends only on quantities from *neighboring layers*: the pre-synaptic neuron's activity and the post-synaptic neuron's prediction error. BP's weight changes are *non-local*, involving information about neurons not directly connected to the synapse in question. In contrast, PC's learning rule is local in space and time (each connection is adjusted based on locally available error and activity). Practically, this locality means a PC network can compute updates in a distributed fashion across layers without needing to route a single global error backward, making it naturally amenable to parallel or even asynchronous implementation.

A.2 Biological Plausibility Comparison

In neuroscience and AI, opinions vary on what constitutes a biologically plausible algorithm. Generally, an algorithm is considered biologically plausible if it operates according to principles similar to those found in the brain. In this context, the we outline these minimal criteria: all computations should be local (i.e., each synapse updates using information available at that connection) and there should be no single global control signal orchestrating the updates. These conditions mimic how real neural circuits function autonomously, without external supervision or control. Traditional predictive coding (PC) networks include dedicated *error neurons* that explicitly compute prediction errors, however the case can be made that having separate error-calculating neurons is biologically implausible. This limitation can be overcome by a different neural design where instead of special error nodes, the dendrites of regular neurons can encode the error signals. In other words, the same neuron's dendritic inputs carry the error information, as illustrated by [66]. This *dendritic implementation* avoids separate error neurons, thereby improving biological realism.

Additionally, PC does not require storing intermediate activations from the forward pass for later gradient computation. In BP, the activations of each layer must be retained to compute the weight gradients later during backpropagation. By contrast, PC continually updates neuron states during the inference phase, and weight updates use the current activity and error values at convergence (or time T), rather than needing the original feedforward activations to be cached. Therefore, PC's learning algorithm inherently updates the necessary signals through ongoing inference, potentially reducing memory overhead and more closely mimicking how a biological system processes information (with no distinct recall of a forward pass). Moreover, PC naturally supports asynchronous and continuous updates, whereas BP is a synchronous, clocked procedure. BP's forward and backward passes must be carefully coordinated, layer by layer, often implemented in a lock-step fashion across the network. In contrast, predictive coding updates can in principle occur without a global clock; neurons and synapses can be updated in parallel or even in a slightly staggered/asynchronous manner, since each update uses local error feedback that does not require waiting for a full forward–backward sweep of the entire network. Furthermore, the incremental predictive coding algorithm is more autonomous than the standard version since it can adjust its synaptic weights *without* any external trigger or timing signal. In the original PC formulations, an external control signal was required before any weight update could happen. By eliminating this requirement, the IPC algorithm behaves more like actual neural tissue, where learning is an ongoing, self-driven process.

The Weight Transport

One well-known implausibility in many deep learning and PC models is the weight transport problem. This problem arises from the unrealistic assumption that the same synaptic weight is used for both the forward signal and the backward error signal. In other words, the connection strength from neuron A to neuron B is assumed to be exactly reused for carrying error information from B back to A. Biological neurons lack an obvious mechanism to ensure that forward and backward signals use identical weights, so this symmetry in algorithms (notably backpropagation) is suspect from a neuroscience perspective. Works such as [67, 68] have been exploring solutions to the weight transport problem. In the broader deep learning field, several approaches can perform as well as backpropagation without requiring identical forward and backward weights. For example, [69] demonstrated that alternative mechanisms can achieve backpropagation-level performance on large image classification tasks without perfect weight symmetry.

Similarly, within the predictive coding framework, the weight transport problem is avoided by using local error signals; however, this creates a *weight symmetry problem*, as forward and reciprocal connections between neurons must still be identical in strength for learning to work correctly. This paper [70] argues that this is more biologically plausible than weight transport because reciprocal connections are common in the brain, and local Hebbian plasticity could naturally synchronize the forward and backward weights over time. However, it is even possible within the predictive coding network to decouple the forward and backward pathways in terms of weights while still achieving accurate classification, as shown by [71]. Other works have designed predictive coding-like algorithms without any weight symmetry requirement at all. These approaches [72, 73] modify the learning process so that each connection can be adjusted using local information alone, without weight mirroring, thereby further aligning the algorithms with biological plausibility.

At last, completely forward-only credit assignment schemes have been proposed as alternatives to BP, for instance, the Forward-Forward algorithm described in [55] and the *PEPITA* rule by [74], which eliminate the backward pass by using a second forward phase with perturbations or negative samples. These approaches, like PC, aim to avoid the weight transport, non-locality, and sequential locking problems of BP. The authors of [75] builds a conceptual bridge between deep learning (batch/layer/weight) normalization tricks and the brain's homeostatic plasticity, claiming they serve the same control goal across scales, from single neurons to whole networks. By mapping these correspondences, it reframes normalization as a general principle for stabilizing activity and improving coding. As a proof of concept, it introduces synaptic scaling (normalization), illustrating a biologically plausible normalization technique inspired by standard ML.

One of the central arguments against predictive coding, as described in [76, 77], is the biological implausibility of its core mechanisms. Predictive coding models often assume continuous activation levels and the ability of neurons to handle both positive and negative error signals. However, biological neurons operate using discrete firing rates and cannot send a negative error signal. The model of summing weighted inputs also fails to account for the crucial role of signal timing and phase, which can, however, be implemented with Spiking predictive coding models.

A.3 PC approximates BP

Predictive coding (PC) approximates backpropagation (BP) under specific conditions, primarily when inference fully converges and the model architecture adheres to a hierarchical or directed acyclic graph (DAG) structure. In these cases, PC's local updates on node activities and weights align closely with the gradients computed by BP, especially when inference is initialized using feedforward activations and weight updates are carefully timed. This approximation holds theoretically and has been validated in layered neural architectures. If one carefully orchestrates when weight updates occur during PC inference, PC *can be made equivalent to BP*. The authors of [78] demonstrated a variant called Zero-Divergence Inference Learning (Z-IL) in which each layer's weights are updated at the precise moment (when the global error signal reaches that layer), yielding exactly the same parameter changes as backpropagation. This differs from the standard PC formulation, which updates the parameters only when the total error has converged. Unlike PC, IPC updates the parameters at every time step t. Intuitively, Z-IL can be seen as a *continuous shift* between BP and PC. Traditional PC can be slower per iteration than BP because it may require multiple inference iterations to stabilize the network's activity before a weight update, and often requires a separate *switch* to trigger the weight update phase. The IPC algorithm, introduced by [23], updates both neuron states and synaptic weights in parallel (on different timescales) without needing a distinct pause to switch from inference to learning, eliminating the need for an external control signal to toggle between phases, making the learning process *fully automatic*. Notably, these BP-equivalent variants of PC still manage to use local learning rules, which is intriguing from a biological perspective: they show it's theoretically possible to get the same learning outcome as BP *without* global non-local computations, by relying on PC's local error unit.

As shown in the work of [5], PCNs can, under specific conditions, approximate standard backpropagation across general architectures (CNNs, RNNs, LSTMs) using only Hebbian updates. PC is seen to approximate BP if it converges to the same critical points of the BP loss function, despite relying on local update rules. The authors of [79] show that when the output error is small, the parameter update of PC is an approximation of that of BP. Similar correspondence has been observed in the following key scenarios:

1. Influence of Precision Parameters Theory and experiments in [7] demonstrated that when feedback precision is low (i.e., feedback signals are given low weight), the network dynamics are dominated by the feedforward pathway. In this regime, the equilibrium activities in a PC network closely match the activations from a standard forward pass. Consequently, the prediction errors at equilibrium approximate the gradients used in BP. Thus, under low feedback precision, PC behaves similarly to BP.

2. Locally Linear Activations under Decoupled Optimization Predictive Coding can approximate Backpropagation when each layer is optimized independently, a condition known as *decoupled optimization*, as shown in [5]. This means the inference in one layer assumes the neighboring layers are fixed. If the activation functions are approximately linear in the operating range, then the gradient-like signals propagated through the PC resemble those computed in BP.

3. Early-Step Error Similarity Authors of [78] found that if the network is initialized using a standard feedforward pass, the prediction errors in the early stages of PC inference already align with BP gradients. This is particularly true if the lower-layer errors begin near zero. In some special initializations, this alignment can occur immediately in the first inference step.

Together, these results demonstrate that PC can serve as a viable approximation to BP, offering a biologically plausible alternative with equivalent learning dynamics under the right conditions.

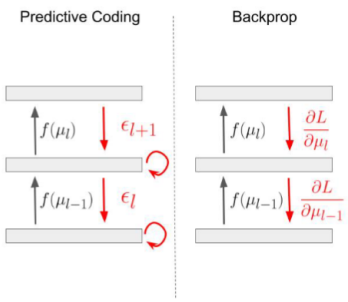


Figure 21: Predictive coding first proceeds with an iterative error minimization step before weight update based upon prediction errors between layers. Backpropagation propagates gradients backwards in its backwards phase and then uses the gradients to update weights directly. Image taken from [80]

A.3.1 Implicit vs. explicit gradients

Backpropagation (BP) performs explicit gradient descent on a task loss: one forward pass to compute activations, one backward pass to transport gradients by the chain rule, then an update. Predictive coding (PC) separates inference and learning: an inner loop infers latent activities by minimizing an energy, and a weight update uses neighboring prediction errors. In explicit PC, both state and weight updates use the current gradient, which is fast but can oscillate in deep, feedback-rich networks. Implicit PC uses proximal (future-state) updates, adding damping that allows larger, more stable steps on stiff dynamics. Some recent PC papers [44, 80]) explore implicit inference updates (e.g., using linear solvers to stabilize updates), especially when running very deep or stiff networks.

A.4 Memory efficiency

Computation: Backpropagation performs a forward sweep to produce activations and then a strictly sequential backward sweep to propagate errors; because each layer's backward computation depends on the previous one, these matrix multiplications cannot be parallelized across layers. Classical predictive coding (PC), in contrast, runs short inference updates of values and errors before updating the weights; crucially, within each inference step, the work per layer is independent, so the expensive matrix multiplications can be executed (for all layers) in parallel. **Space:** BP relies on a global backward pass and additional places to store the backward errors, on top of caching forward activations. PC and IPC instead maintain local per-layer states and local prediction and error variables and do not require a separate global gradient tape; In other words, the scaling is comparable (per-layer state), but PC avoids BP's extra backward error storage. For a detailed comparison between BP and PC, and between computation and storage, see [23].

B Layer wise updating

Synchronous Sequential Updates vs. Asynchronous Parallel Updates:
BP's error propagation is a sequential procedure, meaning lower layers must wait for gradient signals from higher layers, which enforces a strict ordering. The brain, however, performs credit assignment in a far more parallel and asynchronous manner, with different regions adjusting simultaneously without waiting for a global signal (neurons fire in parallel, with slight delays or oscillatory waves). Predictive coding aligns with this *parallelism*: all layers update their activities in parallel based on local prediction errors. There is no strict layer-by-layer waiting as in BP's backward pass. This greater parallelizability is a noted advantage of PC networks, potentially allowing more efficient training on neuromorphic hardware by eliminating the waiting time induced by BP's sequential backward pass. In essence, BP locks the network into two alternate phases (forward then backward), whereas PC networks continuously and concurrently refine their neuron states until errors are minimized (often described as a relaxation or inference phase), making the update process more akin to real-time asynchronous computation.

B.1 Signal propagation

The authors of this paper [43] considered stabilization techniques for sequential PC models, such as length regularization, which prevents latent state explosion by penalizing deviations from a target norm, and sequential training, which balances error propagation by gradually training layers with skip connections. This strategy improves gradient flow, distributes prediction errors more evenly across layers, and avoids overfitting early on. Predictive Coding Networks (PCNs) exhibit instability when applied to deep architectures. During inference, the norms of latent states and prediction errors can grow exponentially, leading to exploding and vanishing gradients that make training unstable and ineffective. To analyze and understand these problems, the study applies dynamical mean-field theory to track the latent-state "lengths" (normalized squared norms), prediction errors, and weights, which serve as a statistically robust proxy for measuring instability.

Additionally, prediction errors are not evenly spread throughout the network; instead, they tend to concentrate near the input and output layers. This uneven distribution prevents intermediate layers from learning properly and contributes to performance degradation as network depth increases. While PCNs are theoretically capable of mimicking backpropagation if they reach an equilibrium state (where $\Delta z = 0$), achieving this condition in practice is difficult. Even when the network is close to equilibrium, residual prediction errors persist and continue to interfere with learning. To counter these issues, the authors propose a two-fold solution: length regularization to control the growth of latent norms, and a sequential training framework with skip connections that helps distribute prediction errors more evenly across all layers. Sequential Inference alone in Inference Learning has already been shown by [44] to increase stability. This strategy improves gradient flow, distributes prediction errors more evenly across layers, and avoids overfitting early on. Length regularization prevents latent state explosion by penalizing deviations from a target norm, while sequential training balances error propagation by gradually training layers with skip connections.

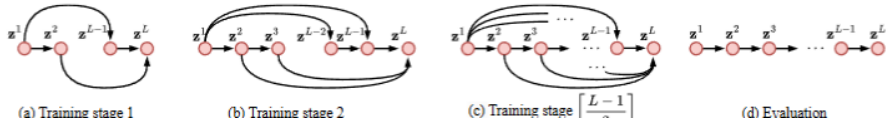


Figure 22: Stage-wise training in hierarchical PCN. Image taken from [43].

C Neural inference dynamics and simulation

To understand the behavior of predictive coding networks under simplified and controlled conditions, we simulate inference dynamics on synthetic graph topologies using ordinary differential equations (ODEs) in continuous time. Unlike task-driven setups (e.g., MNIST classification), these simulations focus purely on the temporal evolution of node activities and prediction errors under fixed sensory and supervision inputs, without relying on real data. We examine two architectural regimes. First feedforward (MLP-like) a sequential structure in which information flows unidirectionally from a clamped sensory node ($x_0 = 1$) through three hidden nodes (x_1, x_2, x_3), culminating in a supervision node clamped at ($x_4 = -1$). We simulate the node values for supervision and label, clamped to 1 and -1, whereas in the real models, we use the full MNIST dataset with images and corresponding one-hot labels. Each layer receives only top-down predictions from the preceding layer. Second a Fully Connected (FC) a densely interconnected system where all five nodes (excluding self-loops) are connected with asymmetric weights. Only the sensory and supervision nodes (x_0, x_4) are clamped; all intermediate nodes update their activity in response to incoming prediction errors from all others. The direction of prediction is indicated by arrows (e.g., $x_j \to x_i$ means that node j predicts node i). In both architectures, the sensory and supervision nodes are clamped (i.e., fixed throughout the inference process). Only the hidden nodes are free to update their activities to minimize their local prediction errors. In this framework, each node in the network represents a dynamic variable whose value evolves over time based on prediction errors and local updates. In both models, the goal is for the internal (free) nodes to minimize their local prediction errors, given fixed boundary conditions. The values evolve under nonlinear update dynamics using the activation function $\mathbf{f}(\mathbf{x}) = \tanh(\mathbf{x})$ and its derivative $\mathbf{f}'(\mathbf{x}) = 1 - \tanh^2(\mathbf{x})$. The system's behavior is governed by the following core vectorized update equations:

Component	**Equation**	**Description**
Prediction	$\boldsymbol{\mu} = \mathbf{W}\,\mathbf{f}(\mathbf{x})$	Top-down prediction received at each node
Prediction Error	$\boldsymbol{\varepsilon} = \mathbf{x} - \boldsymbol{\mu}$	Deviation of value from predicted input
Update Rule (Free Nodes)	$\Delta\mathbf{x} = -\boldsymbol{\varepsilon} + \mathbf{f}'(\mathbf{x}) \odot (\mathbf{W}^\top \boldsymbol{\varepsilon})$	Local error-driven update for hidden nodes
Inference Dynamics	$\mathbf{x}_{t+1} = \mathbf{x}_t + \gamma\,\Delta\mathbf{x}_t$	Step-based update rule
Energy (node i)	$E_i = \varepsilon_i^2 = [\mathbf{x} - \mathbf{W}\mathbf{f}(\mathbf{x})]_i^2$	Squared local prediction error

Table 4: Key vectorized equations for Predictive Coding Dynamics for inference only.

$$\mathbf{W}_{\text{MLP}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1.00 & 0 & 0 & 0 & 0 \\ 0 & 0.90 & 0 & 0 & 0 \\ 0 & 0 & 0.80 & 0 & 0 \\ 0 & 0 & 0 & 1.10 & 0 \end{bmatrix}, \quad \mathbf{W}_{\text{FC}} = \begin{bmatrix} 0 & 0.11 & -0.25 & -0.22 & -0.23 \\ 0.11 & 0 & 0.84 & -0.09 & -0.15 \\ -0.25 & 0.84 & 0 & -0.26 & -0.53 \\ -0.22 & -0.09 & -0.26 & 0 & -0.34 \\ -0.23 & -0.15 & -0.53 & -0.34 & 0 \end{bmatrix}$$

The stream plots below visualize how pairs of node values evolve over time under predictive coding updates. Each subplot shows the local dynamics between two connected nodes, where arrows indicate the joint evolution of their activities. In the MLP model, the flow is directional and layered, often converging to a single attractor. In contrast, the fully connected model has symmetric weights, no self-loops, and mixed signs, resulting in more complex and potentially unstable dynamics. For both models, the (blue) *sensory* node x_0 and (green) *label* node x_4 have clamped values.

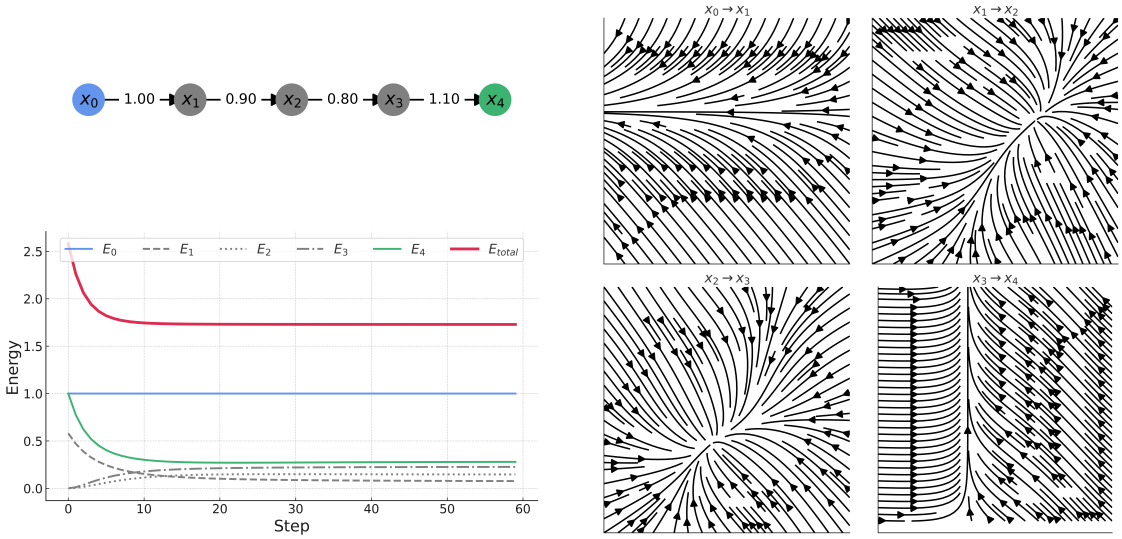


Figure 23: Left: graph topology of an MLP-like model and energy during inference. Right: vector fields (streamplot) for each pairwise active edge

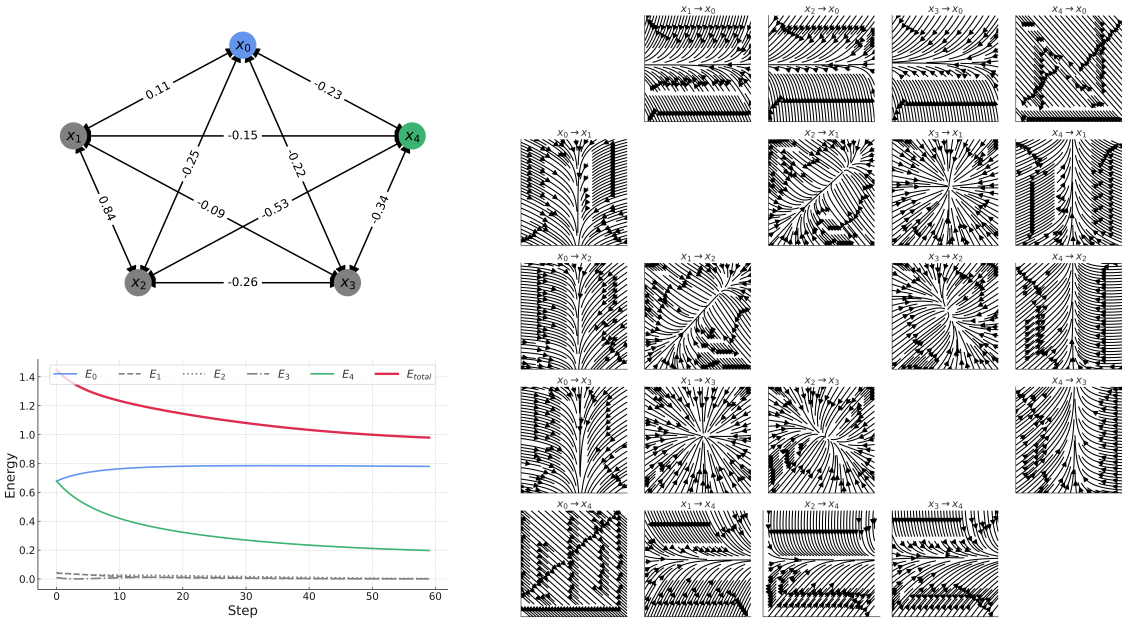


Figure 24: Dynamics of a fully connected predictive coding network with symmetric weights and clamped endpoints (x_0, x_4). Both the sensory (blue) and label (green) nodes have their node values clamped. Left: all-to-all graph topology (excluding self-loops), color-coded by function. Bottom left: individual and total node prediction error energies over time, reflecting transient instability followed by convergence. Right: rich vector fields (streamplots) for each pairwise interaction,

Both models during inference show local attractor regions. Although the fully connected model has feedback connections that further complicate inference dynamics, both experiments show a decrease in total inference error.

D Parameter effect

This section examines the influence of key hyperparameters on the performance and behavior of our Incremental Predictive Coding (IPC) models. We systematically investigate how parameter variations affect both the discriminative (classification accuracy) and the generative (image quality) capabilities of the models. The findings presented below highlight the sensitivity of PC networks to these configurations, which is crucial for optimization and understanding the model's functional characteristics.

D.1 Discriminative IPC

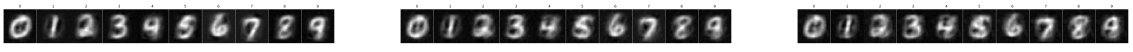
The most influential parameters for this discriminative IPC model are the learning rates and the number of steps T. For stable training, the ratio between the two learning rates is one of the most important parameters. The baseline model, in bold, uses IPC, a discriminative model with hidden layers $L = \{300, 100\}$, the *swish* activation function, and zero weight decay for both optimizers.

Table 5: Training configuration and accuracy (per experiment). The baseline experiment shown in bold or with (−). Other rows are experiments where a single hyperparameter differs from the baseline.

Run	Hyperparameter	Setting	Accuracy
1.	Learning rate	$\boldsymbol{\eta_x = 0.5, \eta_w = 1e^{-5}}$	**0.94**
2.		$\eta_x = 0.01, \eta_w = 1e^{-6}$	0.48
3.		$\eta_x = 1, \eta_w = 1e^{-4}$	0.37
4.	$T_{\text{train \& test}}$	**15**	-
5.		50	0.97
6.		100	0.96
7.	Init $x_{t=0}/\mu_{t=0}$	**0**	-
8.		$\mathcal{N}(0, 1e^{-7})$	0.94
9.	Use Grokfast	**False**	-
10.		True	0.93
11.	Weight init	$\boldsymbol{\mathcal{N}(0, 005)}$	**-**
12.		$\mathcal{N}(0, 0.05)$	0.96
13.		MLP-like	0.96

D.2 Generative IPC

The findings of Orchard et al. [81] claim that predictive generative coding networks often fail to generate meaningful inputs when run in reverse (when a class label is clamped and the corresponding input is inferred). They showed that this generative task is *ill-posed* because the inverse problem in a sequential model has infinitely many solutions, particularly when the input space is larger than the output space. The authors claim that clamping the desired class vector and running the network to equilibrium does not yield a sensible state in the input layer (i.e., generating images). To address this, the paper introduced explicit *minimum-norm constraints* on both the weights and the value nodes. They showed that adding a decay term to the dynamics of both the latent values and the learnable weights yields significantly better generative behavior. However, experiments with our sequential generative predictive coding models show sensible output sensory images even with weight decay terms $\lambda_w = \lambda_w = 0$.



(a) $\lambda_w = \lambda_w = 0$ (b) $\lambda_w = 1e^{-4}, \lambda_x = 1e^{-5}$ (c) $\lambda_w = 1e^{-2}, \lambda_x = 0$

Figure 25: weight decay values λ_w used for the weight optimizer Adam and value $\lambda_x = 0$ used for the SGD optimizer.

D.3 Model weights during training:

Using different hyperparameter settings with two models, we track the Fully Connected model weights from Table 3 at each epoch during training. Given all the weights, we use PCA and metrics to visualize the difference in training dynamics. We also plot the mean weights against the standard deviation of the weights, since these statistics can help determine phase shifts and criticality (as in similar models, such as Ising models).

Both models in Table 3 have 1000 nodes and use $\mathcal{N}(0, 005)$ for weight initialization, so they start at the same point. However, the (light blue) model uses activation function *swish*, $T = 40$, no GrokFast amplification, and zero weight decay. The other models uses activation function *relu*, $T = 15$, with grokfast amplification and weight decay $\lambda_x = \lambda_w = 1e^{-6}$. Our first model performs well on classification at the expense of its generative capabilities, while the other models yield good generation but lower classification accuracy.

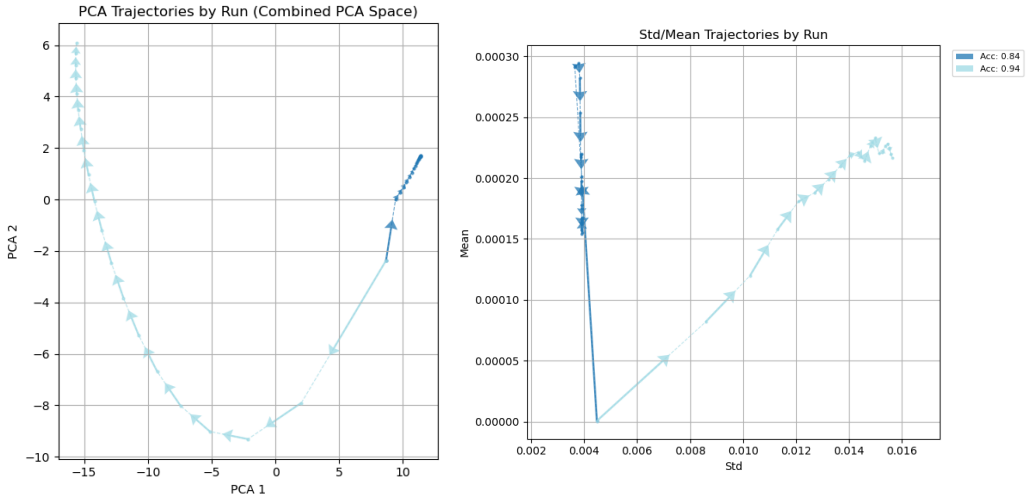


Figure 26: PCA trajectories and Standard deviation/Mean trajectories of the models in Table 3.

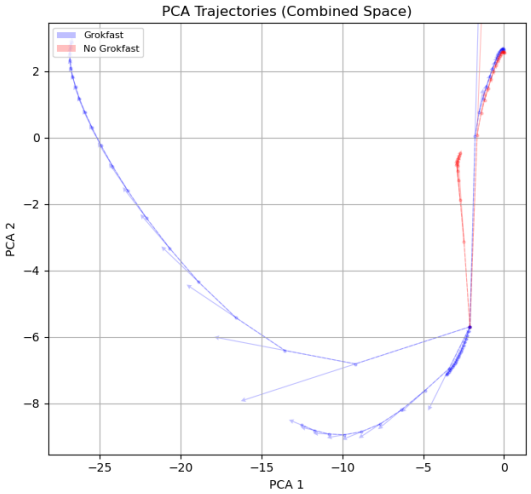


Figure 27: The weight dynamics of multiple fully connected IPC models during training, visualized via PCA (a single arrow indicates an epoch), showing the influence of Grokfast training. Grokfast appears as damping in the PCA plot, while generalization, and thus classification accuracy, does not improve as shown in Table 5.

D.4 Invariant features in predictive coding models

All the experiments above focus on a simple mapping using the MNIST dataset; however, predictive coding has also been used to learn invariant features. Experiments from [82] show that in generative PC models trained on videos, higher layers become selective for identity and support top-down completion under occlusion. Their PC model shows that, when presented with short sequences of smooth transformations such as translation, rotation, or scale, higher layers come to represent stable causes like object identity, while lower layers track pose, position, and even lighting. This follows a slowness prior in which identity changes slowly and nuisances change quickly. Topographic PC VAE makes this more explicit, where it arranges latent capsules on a two-dimensional grid so that a change in the input maps to a roll or shift in capsule coordinates. Invariance is stored in capsule amplitudes, and equivariance is stored in positions or phases. The predictive pathway uses these structured latents to forecast the next frame, which encourages separation of identity from pose, as shown in [83].

D.5 Optimization

Predictive coding can be seen as performing gradient descent on an energy (loss) function with respect to both neural activities and weights. In contrast, backpropagation does so only with respect to weights. BP has separate forward and backward passes, which require sequential updates. IL performs inference in parallel across layers and iteratively refines activities. Below are the different phases of the learning algorithm, Backpropagation and Inference Learning on a sequential model with l layers side by side. See the Appendix section F for pseudo code and implementation details.

Table 6: A comparison of backpropagation and inference learning

	Backpropagation		**Inference learning**
Forward	$x^\ell = f\big(w^{\ell-1}x^{\ell-1}\big)$ $\quad (\ell = 1..L)$	Inference	$\mu^\ell = f\big(w^{\ell-1}x^{\ell-1}\big)$
Backward	$\delta^\ell = \begin{cases} \dfrac{\partial \mathcal{L}}{\partial z^\ell}, & \ell = L, \\ (w^\ell)^\top \delta^{\ell+1}\, f'\big(w^\ell x^\ell\big), & \ell = (L-1)..1 \end{cases}$		$\epsilon^\ell = x^\ell - \mu^\ell$
			$\Delta a^\ell = -\big(\epsilon^\ell + (w^\ell)^\top \epsilon^{\ell+1}\big)\, f'\big(w^\ell x^\ell\big)$
Learning	$\Delta w^\ell = \delta^{\ell+1}(x^\ell)^\top$	Learning	$\Delta w^\ell = \epsilon^{\ell+1}(x^\ell)^\top$

As optimizers for value updating, we use SGD, and Adam for weight update optimizer. To generalize to topology and improve speed, vector-matrix multiplication can be used, as well as a message-passing implementation well suited for sparse graphs.

Inference learning as active recall analogy
Memory strengthens through *active recall*, not passive exposure. Similarly, inference learning starts with an initial guess (a proto-memory), recalls it to make a prediction, compares it to the truth, and updates it. This cycle, recall, compare, and update repeats until the representation aligns with reality. Incremental Predictive Coding (IPC) mirrors this process, continuously adjusting activities and weights, much like the E-step (inference) and M-step (update) in EM. Just as understanding grows through repeated recall, IPC refines its memory by iteratively predicting and correcting.

E Implementation & reproducibility

We implemented discriminative, generative, fully connected, stochastic block model (SBM), and other topologies using both Predictive Coding (PC) and Incremental Predictive Coding (IPC). All models were developed in PyTorch, Inference Learning dynamics, both with Torch matrix multiplication and Message Passing dynamics for graph variants built with `torch_geometric`. Experiments were run on a single GPU (RTX 4090), with fixed random seeds to ensure reproducibility. Source code and logs are openly available at GitHub and Weights & Biases.

Notation & Direction

Shared notation. x = activities, μ = predictions, $\epsilon = x - \mu$, $f(\cdot)$ and $f'(x)$ act element-wise; $\odot$ is the Hadamard product; γ is the inference step size.

Matrix meanings (native).
Van Zwol: rows = receivers/targets, columns = senders/sources;
W_{ij} is the weight on edge $j \to i$.

Salvatori: rows = senders/sources, columns = receivers/targets;
$W_{i,k}$ is the weight on edge $i \to k$.

Original index notation and matrix (vectorized) notation are used from both papers.

Van Zwol

$$\mu_i = \sum_j W_{ij}\, f(x_j)$$
$$\boldsymbol{\mu} = W\, f(x)$$
$$\Delta x_i = \gamma\left(-\epsilon_i + f'(x_i) \sum_j W_{ji}\, \epsilon_j\right)$$
$$\Delta \boldsymbol{x} = \gamma\left(-\epsilon + f'(x) \odot W^\top \epsilon\right)$$

Salvatori

$$\mu_i = \sum_k W_{ki}\, f(x_k)$$
$$\boldsymbol{\mu} = W^\top f(x)$$
$$\Delta x_i = \gamma\left(-\epsilon_i + f'(x_i) \sum_k W_{ik}\, \epsilon_k\right)$$
$$\Delta \boldsymbol{x} = \gamma\left(-\epsilon + f'(x) \odot W\, \epsilon\right)$$

Rule of thumb. Predictions follow the stored weight matrix (left: W; right: $W^\top$). Error messages flow *against* that direction in the notation (left: $W^\top \epsilon$; right: $W\,\epsilon$).

Dense vs. Message Passing

These dynamics can be implemented with either dense matrix multiplication or message passing. Dense formulations compute over the full weight matrix W, including non-existing edges, which is inefficient for sparse graphs. Message passing, in contrast, restricts the computation to edges stored in an *edge index*, scaling with $|E|$, the number of edges, instead of with the number of nodes scaling with N^2. This better reflects the locality of PC: updates depend only on pre- and post-synaptic quantities, making the algorithm naturally suited for sparse or non-layered topologies.

Relation to Graph Neural Networks

Although PC can be implemented via message passing, its update rules differ from those of standard Graph Neural Networks (GNNs). In GNNs, node embeddings are updated via neighborhood aggregation, and after T layers, they represent k-hop structural information. In PC graphs, node values x are updated by minimizing the energy function via local error feedback, rather than by aggregating embeddings. Thus, while both frameworks exploit graph structure, PC nodes encode predictions and errors rather than similarity-based embeddings. A more detailed comparison is given by Byiringiro [84].

E.1 Hyperparameter

Table 7: General training configuration

Item	**Setting**
Dataset	MNIST (train 60k / test 10k), grayscale 28×28
Tasks	Classification, Generation, Occlusion
Topologies	Hierarchical; Fully connected; SBM (clustered); Two-branch topology
Activation $f(\cdot)$	$\tanh, \text{hardtanh}, \text{relu}, \text{swish}$: no significant differences
Batch size	100 (hierarchical and fully connected), 50 or 10 (bigger graphs)
Hidden nodes initialization	Normal $\mathcal{N}(0, 1e{-}4)$ - $\mathcal{N}(0, 1e{-}7)$
Weight initialization	Normal $\mathcal{N}(0, 0.1)$ - $\mathcal{N}(0, 0.01)$, optional bias $b{=}0$
dataset normalization	Zero mean and unit standard deviation
Update mode	PC vs IPC
Seed	$\{2\}$
Inference steps $T_{\text{train \& test}}$	$T \in \{3, 5, 10, 50, 100\}$
Lr node value η_x (γ)	{1, 0.5, 0.1, 0.01}
Lr edge weights η_w (α)	$\{1e{-}4, 1e{-}6\}$
Weight decay node values λ_v	0
Weight decay edge weights λ_w	{0, 1e−4, 5e−4}
Hardware	Single GPU (e.g., RTX 4090)

As for the optimizers, we have used the SDG optimizer for value updating and the Adam optimizer for weight update. Batch sizes were 100 for hierarchical/fully connected models, reduced to 50 or 10 for larger graphs. The data sets were normalized to have a mean of 0 and unit variance. Activation functions tested included tanh, hardtanh, ReLU, and swish, with comparable performance across tasks. We have not used multiple seeds since we are not comparing our results with those of SOTA BP or PC.

Reproducibility. In all experiments, we have maintained a fixed T during training and testing, rather than a dynamic T that stops at convergence. A dynamic stop at convergence would be particularly useful for adjusting at test time and can thus be tuned to the level of granularity needed for testing different tasks or combining datasets. All baseline training configurations used Incremental Predictive Coding (IPC).

F Pseudo code

Graph G consists of sensory nodes s (which are fixed to the data), internal or hidden (i) nodes, and supervision or label nodes (l), which is fixed to the corresponding label. Each sensory node is connected to a single pixel, and the label nodes are connected to the one-hot encoding of the image label. Our implementation uses simultaneous updating across all nodes, rather than layer-wise updating. Thus all nodes are updated at the same time step t, and thus do not wait for other signals. For sequential predictive coding models, such as [41], we set the error at the target or supervision nodes to zero at test time. The already-clamped supervision values cannot be updated, but their errors can still affect other nodes. We omit feedforward initialization as in [2], where in hierarchical graphs we require to initialize layer by layer.

INFERENCE LEARNING

Algorithm 1 Supervised learning on dataset $D = \{X_i, Y_i\}$ with PC. on graph G

Require: $(x_{0:s})$ is fixed to (X) and $(x_{i:l})$ is fixed to (Y)
Require: Initialize edge weight and/or values hidden nodes $(x_{s:l})$
for $t = 0$ to T **do**
  for each vertex i (E-step) **do**
    Update $x_{i,t}$ to minimize E_t using gradient descent via Eq. (3)
  end for
  if $t = T$ (M-step) **then**
    Update selected weights Wi, j by edge type to minimize E_t via Eq. (4)
  end if
end for

Algorithm 2 Supervised learning on dataset $D = \{X_i, Y_i\}$ with IPC. on graph G

Require: $(x_{0:s})$ is fixed to (X) and $(x_{i:l})$ is fixed to (Y)
Require: Initialize edge weight and/or values hidden nodes $(x_{s:l})$
for $t = 0$ to T or until convergence **do**
  for each vertex i (EM-step) **do**
    Update $x_{i,t}$ to minimize E_t using gradient descent via Eq. (3)
    Update selected weights $W_{i,j}$ by edge type to minimize E_t via Eq. (4)
  end for
end for

GROKFAST

Algorithm 3 Grokfast-EMA Gradient Amplification

Require: Model parameters W, loss function $\mathcal{L}$, learning rate η
Require: EMA decay $\alpha \in [0, 1]$, amplification factor $\lambda > 0$
Initialize EMA of gradients: $\bar{g}_0 \leftarrow 0$
for each training step t **do**
  Compute gradient: $g_t \leftarrow \nabla_W \mathcal{L}(W_t)$
  Update EMA: $\bar{g}_t \leftarrow \alpha \cdot \bar{g}_{t-1} + (1 - \alpha) \cdot g_t$
  Amplify: $g'_t \leftarrow g_t + \lambda \cdot \bar{g}_t$
  Update parameters: $W_{t+1} \leftarrow W_t - \eta \cdot g'_t$
end for

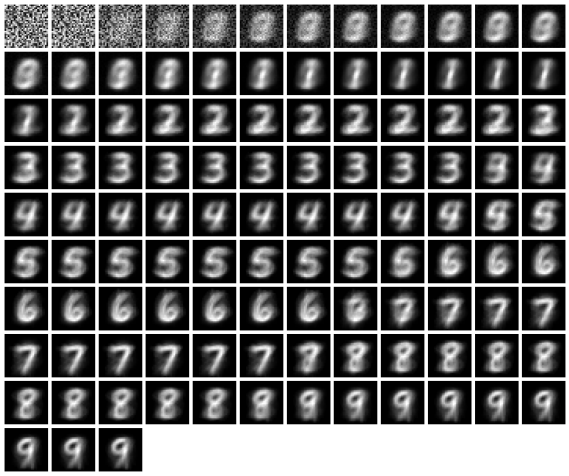


Figure 28: Transitioning from noise to increasing digits by altering the label

Figure 29: Image generation where the supervision labels are clamped to the one-hot encoding of the label *5*.

Figure 30: Reconstruction experiments where the dataset has been rotated or scaled.

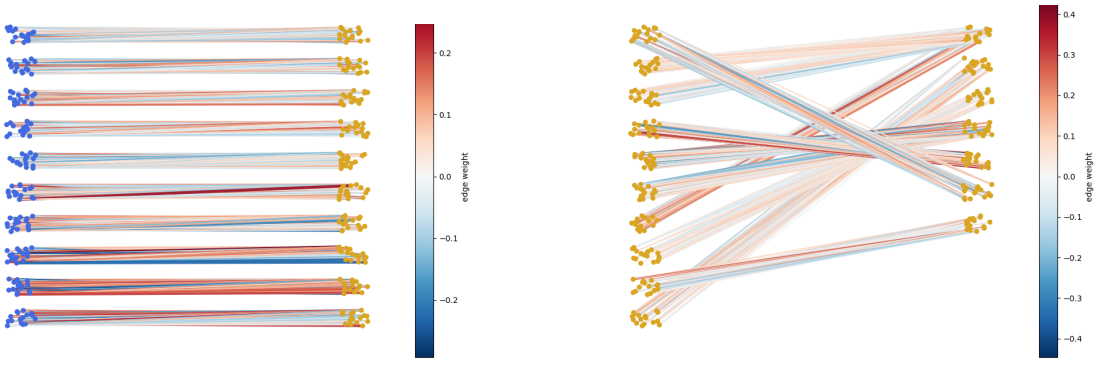


Figure 31: Illustration of the SBM models from section 7.2 where the sensory nodes (blue) correspond to multiple clusters, each mapping a 4x4 square of pixel values to a cluster. The hidden nodes are also clustered. Depending on the SBM topology type, connections inside each hidden cluster (yellow) are fully connected or follow a common direction.