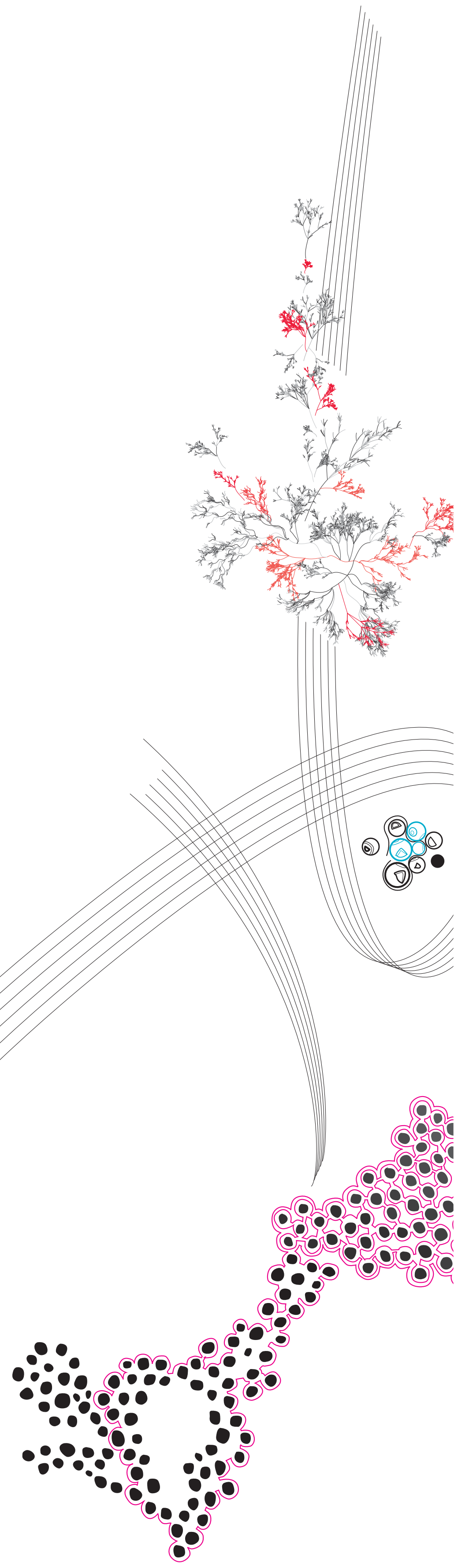
MSc Thesis



Predictive coding with generalized losses and its mathematical relationship with backpropagation

Wei-Ting Sun

Graduation committee:	Role:
prof.dr. A. J. Schmidt-Hieber	Chair and Daily supervisor
dr.ir. Werner Scheinhardt	Independent examiner
dr.ir. Bettina Schwab	Additional member
dr. Juntong Chen	Additional member

August 14, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente

UNIVERSITY OF TWENTE.

Acknowledgments

This master's thesis is a culmination of 6.5 months of work. I would like to thank my supervisor, Johannes Schmidt-Hieber, for the research topic and for providing helpful suggestions and feedback along the way. I would also like to thank my graduation assessment committee members, Werner Scheinhardt, Bettina Schwab, and Juntong Chen: thanks for making it possible for me to graduate. Special thanks to my friends Jitse Kuilman and Serge Johanns, who helped look over my thesis.

I wouldn't have been able to finish my thesis on time without the Bibliotheek Twente Hengelo Stad. I have a lot of trouble focusing and getting things done at home, so I've been going to the library on weekdays to work on my thesis since June 4, and that has helped me tremendously. The restaurant/café in the library, STOET, is also great. The atmosphere is quite nice. I've eaten there a few times to avoid having to cook, and the food is decent — I recommend the “Pulled Chicken Wrap” (€7.50 as of the time of writing). Another establishment that has saved me from cooking is the The Döner Company branch in Hengelo station: I often get the “Broodje falafel groot” (€5.99 as of the time of writing) (though I feel like recently they haven't been putting as much vegetables in the broodje, perhaps due to shrinkflation).

One thing I've noticed is that, unlike my bachelor's thesis where I spent the last three weeks rushing to get it done and almost failed to turn it in on time, it hasn't been as much of a rush for me in the final days of my master's thesis (though I am still writing this on the day it is due). Perhaps I have gotten better at working on bigger projects. Good job, me!

Another thing I've noticed is that you can really write just about anything in the Acknowledgments section. In that spirit, I will share some media I have enjoyed recently: the anime/manga series 甘々と稲妻 (*Sweetness and Lightning*), the manga series RTA走者はゲーム世界から帰れない (*The Speedrunner Can't Return From the Game World*), the game *Stardew Valley*, and the mathematics YouTube channel “Another Roof”.

Of course, I would like to thank my parents for their financial support. I wouldn't be where I am now without them.

Finally, a big shoutout to my friends and acquaintances throughout my master's program, especially Ties and Mei. The Applied Mathematics master's program was tough, but I did it — we did it (I hope).

Wei-Ting Sun
August 12, 2024

Predictive coding with generalized losses and its mathematical relationship with backpropagation

Wei-Ting Sun

August 14, 2024

Abstract

Backpropagation is the supervised learning algorithm that has underpinned the continued success of neural network models in a wide range of applications. However, it has not found success as a model for learning in the biological neural network of the brain, as the algorithm is generally considered to be biologically implausible. Predictive coding algorithms, on the other hand, are a promising class of learning algorithms that have a biological basis and have been shown to approximate backpropagation. One problem is that so far, they have mostly been formulated only for squared loss functions. In this paper, we present a framework for predictive coding algorithms with general loss functions, and we prove that for certain classes of loss functions, predictive coding approximates — and in a certain limit is equal to — backpropagation. Our results pave the way for predictive coding to be applied on more complex neural network architectures and machine learning tasks, and further closes the gap between biologically realistic models of learning and artificial neural networks.

Keywords: predictive coding, backpropagation, loss, biological plausibility

Contents

1 Introduction

Neural network models and artificial intelligence has seen a boom in recent years, with new architectures and technologies such as transformers [36], large language models [7, 35], and diffusion models [33, 16, 28]. These technologies have all been made possible with the error backpropagation algorithm, or "backprop" for short [19, 37]. The success of backprop in training neural networks has led to it being the by far most widely used (supervised) learning algorithm for the task today [29].

While artificial neural networks and backprop were initially inspired by the biological neural network of the human brain, it is generally considered implausible that backprop is the algorithm used by the brain to learn. The major criticism is that backprop requires an error or a gradient to be transported in the reverse direction, from the end to the start of the network, which is unrealistic for the brain [9, 18].

However, we would like to have a successful, biologically plausible learning algorithm for neural network models. A major goal of neuroscience is to better understand how the brain learns and thinks — *cognition*. Such biologically plausible models would lead to a better mechanistic understanding of human cognitive abilities, such as memory [31] and language [26], as well as diseases of the brain [2]. Additionally, they could also lead to improved neuromorphic hardware — hardware that is inspired by and/or aims to mimic how the brain functions. The hope is that computers that function more similarly to the brain would be more efficient [3].

In recent years, predictive coding has been developed as a biologically plausible algorithm for learning in the brain. Predictive coding, in addition to a learning algorithm, is also a data compression strategy and a neurological theory of perception. This theory posits that perception in the brain is a combination of a top-down generative model of the expected sensory information and the actual received sensory information, and the errors between the two are used to tweak the generative model, akin to Bayesian inference [8, 13]. From this, various predictive coding algorithms have been developed for learning on neural networks, with the aim of being both biologically plausible and useful [12, 10, 11, 23, 24, 34].

A commonly used method in the literature of showing that predictive coding algorithms learn effectively is to show that they approximate or are equal to backprop. [38] showed that predictive coding with squared loss on a multilayer perceptron approximates, and in some cases approaches, backprop. [22] extended that result to neural networks that are directed acyclic graphs. [34] devised variant predictive coding algorithms that are able to produce exactly the same parameter updates as backprop, with squared loss and on multilayer perceptrons. [30] then generalized those algorithms to arbitrary directed acyclic graph networks.

However, there are few results on extending predictive coding algorithms to loss functions other than squared losses. While [25] has formulated and experimentally compared predictive coding with general distributions (equivalently, general losses) to backprop, to the best of our knowledge, there do not yet exist mathematical proofs of this correspondence in the literature.

The contribution of this paper is in *mathematically proving* that predictive coding with *general loss functions* and on neural networks that are arbitrary directed acyclic graphs, under certain conditions of the loss functions, also approximates and approaches backprop in a certain limit.

We first describe the notation used in this paper in Section 2. In Section 3 we describe the backprop algorithm (BP), and derive and present the predictive coding algorithms

PC, PC-SQ, and PC-SQ-e, with PC being able to accommodate general loss functions. Section 4 is dedicated to discussing the biological plausibility of the presented algorithms. This culminates in Section 5, where we prove various theorems relating PC to BP. Section 6 concludes the paper.

2 Notation

Scalars are written in lowercase italics (x), vectors and arrays in bold lowercase italics ($\boldsymbol{x}$), and random variables in their uppercase counterparts (X, $\boldsymbol{X}$); sets are written in uppercase italics (X).

2.1 Data

In the context of supervised learning, let $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [n]}$ be a set of paired data — there are n datapoints, indexed by i from 1 to n — with $\boldsymbol{x}_i \in \mathbb{R}^{d_{\boldsymbol{x}}}$ being the input and $\boldsymbol{y}_i \in \mathbb{R}^{d_{\boldsymbol{y}}}$ being the output of each datapoint, and $d_{\boldsymbol{x}}, d_{\boldsymbol{y}} \in \mathbb{N}$ being their respective dimensions. We assume that they are i.i.d. samples from some random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$. The goal of the general supervised learning problem is to learn a function that predicts $\boldsymbol{Y}$ given $\boldsymbol{X}$.

In the context of unsupervised learning, let $D = \{\boldsymbol{x}_i\}_{i \in [n]}$ be a set of unpaired data, with $\boldsymbol{x}_i \in \mathbb{R}^{d_{\boldsymbol{x}}}$ for all datapoints $\boldsymbol{x}_i$. We assume that they are i.i.d. samples from some random variable $\boldsymbol{X}$. The goal of the general unsupervised learning problem is to learn the distribution of $\boldsymbol{X}$, i.e. to estimate the probability density at each value of $\boldsymbol{x}$.

2.2 Computational graphs

Computational graphs are a class of functions often used as a general-purpose model for relationships between observed variables, and are the models used by both backprop and predictive coding. In a general sense, a computational graph G is a tuple

$$G = (V, A, \{(j, f_j)\}_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}}). \tag{1}$$

In this notation, V is the set of nodes, which can be split into $V_{\boldsymbol{x}}$, $V_{\boldsymbol{y}}$, and V_{hid}, which are the set of input nodes (no parents), output nodes (no children), and hidden nodes (has both parents and children, i.e. all other nodes) respectively. Additionally, A is the set of all arcs (directed edges). We also assume that the underlying graph (V, A) is a directed simple acyclic graph — there cannot be multiple arcs between the same pair of nodes, and there are no cycles. There are varieties of predictive coding algorithms that work on arbitrary (including cyclic) computational graph [30]. However, since backpropagation only works on acyclic graphs, and acyclicity is a useful property, we limit ourselves only to acyclic computational graphs. Lastly, f_j is the node function associated with non-input node $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$.

Given a node $j \in V$, the nodes that have an arc going to j are its parents, and the nodes that have an arc coming from j are its children. These are denoted by the sets $\text{pa}(j)$ and $\text{ch}(j)$ respectively.

We assume that the number of input and output nodes $|V_{\boldsymbol{x}}|$ and $\left|V_{\boldsymbol{y}}\right|$ are equal to the input and output dimensions of the data, $d_{\boldsymbol{x}}$ and $d_{\boldsymbol{y}}$, respectively. We also assume that each entry of an input datapoint $\boldsymbol{x} \in \mathbb{R}^{d_{\boldsymbol{x}}}$ corresponds to a unique input node $j \in V_{\boldsymbol{x}}$ — the entry corresponding to input node $j \in V_{\boldsymbol{x}}$ is denoted by $(\boldsymbol{x})_j$. Similarly, each entry of an output datapoint $\boldsymbol{y} \in \mathbb{R}^{d_{\boldsymbol{y}}}$ corresponds to a unique output node $j \in V_{\boldsymbol{y}}$, and this entry is denoted by $(\boldsymbol{y})_j$.

We often want to have a "value" stored in each node of the computational graph — we call these **node values**. While different node values (feedforward values, predictive coding node values, etc.) will be denoted with different letters, we write a_j for a general value at node j. We assume that node values are scalar real numbers. We use the notation $\boldsymbol{a}_{\text{pa}(j)}$ to denote an array of the node values of the parent nodes of node j.

As it is a computational graph, each non-input node $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ is associated with a **node function** $f_j : \mathbb{R}^{\deg^-(j)} \times \Theta_j \to \mathbb{R}$. The function takes as input an array of dimension equal to the indegree $\deg^-(j)$ of node j — the number of arcs pointing into j — and function parameters $\boldsymbol{\theta}_j \in \Theta_j$, which are the parameters to be learned, with Θ_j being the set of all possible parameters at node j (usually $\mathbb{R}^k$ for some k). The output is a scalar. We assume that all node functions are differentiable. However, we also note that some node functions used in practice are not differentiable everywhere, e.g. the ReLU function. In such cases, one should define functions that act effectively as gradients, e.g. $\mathbb{1}(x > 0)$ as the derivative of the ReLU function.

In practice, these node functions are almost always chosen to be of the form

$$f_j(\boldsymbol{a}_{\text{pa}(j)}; \boldsymbol{\theta}_j) = \psi_j\left(\sum_{i \in \text{pa}(j)} \phi_j(a_i; \boldsymbol{\theta}_{j,i}); \boldsymbol{\theta}_{j,j}\right), \tag{2}$$

for some functions $\phi_j : \mathbb{R} \times \Theta_{j,i} \to \mathbb{R}$ and $\psi_j : \mathbb{R} \times \Theta_{j,j} \to \mathbb{R}$. Node functions of this form are preferred because they are symmetric with respect to permutations of the parent nodes, can accommodate different numbers of parent nodes, and have a limited complexity in the interactions between the parent node values. We call node functions of this form **generalized sum node functions**. In this class of functions, the parameters $\boldsymbol{\theta}_{j,i}$ for $i \in \text{pa}(j)$ model the strength of the influence of node i on node j — we call these the **weight parameters** of node j. The weight parameters $\boldsymbol{\theta}_{j,i}$ and the remaining parameters $\boldsymbol{\theta}_{j,j}$ together make up the node parameters $\boldsymbol{\theta}_j$.

For example, a commonly used node function in machine learning is

$$f_j(\boldsymbol{a}_{\text{pa}(j)}; \boldsymbol{\theta}_j) = \sigma\left(\theta_{j,j} + \sum_{i \in \text{pa}(j)} \theta_{j,i} a_i\right), \tag{3}$$

where σ is a non-linear function, often the ReLU function. This is an example of a generalized sum node function, with $\phi_j(a_i; \theta_{j,i}) = a_i \theta_{j,i}$ and $\psi_j(x; \theta_{j,j}) = \sigma(x + \theta_{j,j})$.

The algorithms in this paper also make use of so called **node loss functions**, functions $\ell_j : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ that quantify how much its two arguments differ from each other. Like for node functions, we also assume that all node loss functions are differentiable.

Since the graph underlying the computational graph G is directed and acyclic, there exists a (usually non-unique) **topological ordering** τ_G of the nodes V: an ordering such that for each node, its parents come before it in the ordering. In our notation, $\tau_G(k)$ gives the kth node in the topological ordering of the nodes of G.

We sometimes want to refer to all the function parameters of a computational graph together. For this, we use the notation $\boldsymbol{\theta}_G \in \Theta_G$ and we call these the **graph parameters**: $\boldsymbol{\theta}_G$ is a concatenation of the parameters $\boldsymbol{\theta}_j$ of all non-input nodes $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$, and Θ_G is the set of all possible graph parameters. When $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_j$ both occur in an equation, the node parameters $\boldsymbol{\theta}_j$ are assumed to be equal to the entries of the graph parameters $\boldsymbol{\theta}_G$ corresponding to node j.

We can now define the **feedforward values** z_j for nodes $j \in V$. Given an input $\boldsymbol{x}$ and graph parameters $\boldsymbol{\theta}_G \in \Theta_G$, they are defined by the recursive relation

$$z_j(\boldsymbol{x}; \boldsymbol{\theta}_G) := \begin{cases} (\boldsymbol{x})_j & j \in V_{\boldsymbol{x}} \\ f_j(\boldsymbol{z}_{\text{pa}(j)}(\boldsymbol{x}; \boldsymbol{\theta}_G); \boldsymbol{\theta}_j) & j \in V_{\text{hid}} \cup V_{\boldsymbol{y}} \end{cases}. \tag{4}$$

Note that the feedforward values of all nodes only depend on the feedforward values of the input nodes, which is simply the input $\boldsymbol{x}$, and the node parameters of all nodes $\boldsymbol{\theta}_G$, hence the notation $z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$. We use the notation $\boldsymbol{z}_{\mathrm{pa}(j)}$ to denote an array whose entries are the feedforward values z_i of the parent nodes $i \in \mathrm{pa}(j)$ of j. The entries are assumed to be in the correct order to serve as input in f_j. Similarly, the notation $\boldsymbol{z}_{\boldsymbol{x}}$ and $\boldsymbol{z}_{\boldsymbol{y}}$ denote an array whose entries are the feedforward values of the input and output nodes respectively, with the order of the entries matching that of the input and output data $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively.

In practice, the feedforward values can be calculated by visiting the nodes in an order given by a topological ordering, and for every node, evaluating its feedforward value using (4). In a topological ordering, each node's parents always come before it, so the right hand side in (4) is always well-defined, and since computational graphs always have a topological ordering, their feedforward values are always well-defined.

A computational graph G as a whole can be viewed as a single function $\boldsymbol{f}_G$ that takes as input an array with dimension $d_{\boldsymbol{x}}$, corresponding to the input nodes, and parameters $\boldsymbol{\theta}_G \in \Theta_G$, and that returns an output with dimension $d_{\boldsymbol{y}}$. We define this function to output the feedforward values of the output nodes given the input node feedforward values and graph parameters:

$$\boldsymbol{f}_G(\boldsymbol{x}; \boldsymbol{\theta}_G) := \boldsymbol{z}_{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{\theta}_G). \tag{5}$$

We note that in the literature, a difference in terminology is often made between computational graphs learned with backpropagation, termed ANNs (artificial neural networks), and computational graphs learned with predictive coding, termed PCNs (predictive coding networks) [21, 34]. In this paper, we call the underlying functions to be learned "computational graphs" for all algorithms, while the names BP, PC, etc. only refer to the algorithm and not the underlying function.

The notation is summarized in Table 1.

Object	Type	Notation
Probability density function of distribution $\mathcal{D}$ with parameters $\boldsymbol{\theta}$	pdf	$\mathbb{p}_{\mathcal{D}}(\cdot;\boldsymbol{\theta})$
Input data	array $\mathbb{R}^{d_{\boldsymbol{x}}}$	$\boldsymbol{x}$
Output data	array $\mathbb{R}^{d_{\boldsymbol{y}}}$	$\boldsymbol{y}$
Paired datapoint	pair of arrays	$(\boldsymbol{x},\boldsymbol{y})$
Dimensionality of input, output data	positive integer	$d_{\boldsymbol{x}}$, $d_{\boldsymbol{y}}$
Paired dataset	set of paired arrays	$D=\{(\boldsymbol{x}_i,\boldsymbol{y}_i)\}_{i\in[n]}$
Unpaired dataset	set of arrays	$D=\{\boldsymbol{x}_i\}_{i\in[n]}$
Number of datapoints	positive integer	n
Computational graph	computational graph	$G=(V,A,\{(j,f_j)\}_{j\in V_{\text{hid}}\cup V_{\boldsymbol{y}}})$
Set of all nodes	set of vertices	$V=V_{\boldsymbol{x}}\cup V_{\text{hid}}\cup V_{\boldsymbol{y}}$
Set of input, hidden, output nodes	set of vertices	$V_{\boldsymbol{x}},V_{\text{hid}},V_{\boldsymbol{y}}$
Node of computational graph	vertex	$j\in V$
Set of parent and child nodes of node j	set of vertices	$\text{pa}(j)$, $\text{ch}(j)$
Indegree of node j	natural number	$\deg^-(j):=\lvert\text{pa}(j)\rvert$
Outdegree of node j	natural number	$\deg^+(j):=\lvert\text{ch}(j)\rvert$
Generic node value of node j	variable in $\mathbb{R}$	a_j
Feedforward value of node j given input $\boldsymbol{x}$ and graph parameters $\boldsymbol{\theta}_G$	function $\mathbb{R}^{d_{\boldsymbol{x}}}\times\Theta_G\to\mathbb{R}$	$z_j(\boldsymbol{x};\boldsymbol{\theta}_G)$
Node value of node j	variable in $\mathbb{R}$	v_j
Node error of node j	variable in $\mathbb{R}$	ϵ_j
Input node values	array in $\mathbb{R}^{d_{\boldsymbol{x}}}$	$\boldsymbol{a}_{\boldsymbol{x}}$
Hidden node values	array in $\mathbb{R}^{\lvert V_{\text{hid}}\rvert}$	$\boldsymbol{a}_{\text{hid}}$
Output node value	array in $\mathbb{R}^{d_{\boldsymbol{y}}}$	$\boldsymbol{a}_{\boldsymbol{y}}$
Node values of parents of node j	array in $\mathbb{R}^{\deg^-(j)}$	$\boldsymbol{a}_{\text{pa}(j)}$
Node parameters of node j	array	$\boldsymbol{\theta}_j\in\Theta_j$
Node function of node j	function $\mathbb{R}^{\deg^-(j)}\times\Theta_j\to\mathbb{R}$	$f_j(\boldsymbol{a}_{\text{pa}(j)};\boldsymbol{\theta}_j)$
Graph parameters	array	$\boldsymbol{\theta}_G\in\Theta_G$
Graph function	function $\mathbb{R}^{d_{\boldsymbol{x}}}\times\Theta_G\to\mathbb{R}^{d_{\boldsymbol{y}}}$	$\boldsymbol{f}_G(\boldsymbol{x};\boldsymbol{\theta}_G)$
Node loss function of node j with true value a and predicted value b	function $\mathbb{R}\times\mathbb{R}\to\mathbb{R}$	$\ell_j(a,b)$
Squared node loss	function $\mathbb{R}\times\mathbb{R}\to\mathbb{R}$	$\ell^{\text{SQ}}(a,b):=\frac{1}{2}(a-b)^2$
Cross-entropy node loss	function $[0,1]\times[0,1]\to\mathbb{R}$	$\ell^{\text{CE}}(a,b):=-a\log b$
Likelihood node loss	function $\mathbb{R}\times\mathbb{R}_+\to\mathbb{R}$	$\ell^{\text{LH}}(a,b):=-\log b$
Datapoint loss function for comp. graph G	function $\Theta_G\times(\mathbb{R}^{d_{\boldsymbol{x}}}\times\mathbb{R}^{d_{\boldsymbol{y}}})\to\mathbb{R}$	$\mathcal{L}_G(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y}))$
Dataset loss function for comp. graph G	function $\Theta_G\times(\mathbb{R}^{d_{\boldsymbol{x}}}\times\mathbb{R}^{d_{\boldsymbol{y}}})^+\to\mathbb{R}$	$\mathcal{F}_G(\boldsymbol{\theta}_G;D)$
Learning rate for v, ϵ, $\boldsymbol{\theta}$	scalar in $\mathbb{R}_+$	$\eta_v,\eta_\epsilon,\eta_{\boldsymbol{\theta}}$
Gradient descent step for parameters $\boldsymbol{\theta}_j$	array	$\Delta\boldsymbol{\theta}_j$
Max number of epochs	scalar in $\mathbb{Z}_+$	c
Backpropagation algorithm	algorithm	BP
Pred. coding with general loss	algorithm	PC
Pred. coding with squared loss and errors	algorithm	PC-SQ-e

TABLE 1: Notation used in this paper

3 From backprop to predictive coding

In this section, we first discuss the backpropagation algorithm, which will lead us to motivating and deriving our predictive coding algorithms.

3.1 Backpropagation with stochastic gradient descent (BP)

Backpropagation with stochastic gradient descent (**BP**) is one of the most widely-used algorithms for performing supervised learning of paired data on computational graphs. We assume that we have a computational graph G, a paired dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [n]}$, and initial graph parameters $\boldsymbol{\theta}_G \in \Theta_G$. The idea behind BP is to have a function $\mathcal{L}_G^{\mathrm{BP}}$ that, for each paired datapoint $(\boldsymbol{x}, \boldsymbol{y})$, quantifies how much the graph function value $\boldsymbol{f}_G(\boldsymbol{x})$ given input data $\boldsymbol{x}$ "deviates" from the output data $\boldsymbol{y}$, and the graph parameters $\boldsymbol{\theta}_G$ are adjusted to lower this "deviation". We call the function $\mathcal{L}_G^{\mathrm{BP}}$ the **datapoint loss function** of BP.

How $\mathcal{L}_G^{\mathrm{BP}}$ is defined is by first defining a **node loss function** $\ell_j^{\mathrm{BP}} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ for each output node $j \in V_{\boldsymbol{y}}$, which quantifies how much the output data (the first argument) and the graph function value (the second argument) differ at the node j. The datapoint loss function is then the sum of the node loss functions of the output nodes:

$$\mathcal{L}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) := \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y})_j, z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)), \tag{6}$$

where $z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ is the feedforward value at node j given input $\boldsymbol{x}$ and graph parameters $\boldsymbol{\theta}_G$. We note that because of the definition of the feedforward values z, they need to be computed for all nodes in order to obtain their values at the output nodes. As a result, they serve as the node values of BP.

We can now derive the node parameter update formulas. BP uses stochastic gradient descent to learn from a dataset: it iterates through all the datapoints, and for each datapoint $(\boldsymbol{x}, \boldsymbol{y})$, it adjusts the parameters to decrease the datapoint loss function for the datapoint $(\boldsymbol{x}, \boldsymbol{y})$. The node parameter update $\Delta\boldsymbol{\theta}_j^{\mathrm{BP}}$ of the parameters $\boldsymbol{\theta}_j^{\mathrm{BP}}$ at non-input node $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$ is proportional to the negative of the gradient of $\mathcal{L}_G^{\mathrm{BP}}$ with respect to $\boldsymbol{\theta}_j^{\mathrm{BP}}$,

$$\Delta\boldsymbol{\theta}_j^{\mathrm{BP}} = -\eta_{\boldsymbol{\theta}} \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial \boldsymbol{\theta}_j}(\boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})), \tag{7}$$

where $\eta_{\boldsymbol{\theta}} > 0$ is the learning rate. The gradient of $\mathcal{L}_G^{\mathrm{BP}}$ with respect to the node parameter $\boldsymbol{\theta}_j$ is then

$$\begin{aligned}\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial \boldsymbol{\theta}_j}(\boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) &= \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) \frac{\partial z_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{x}; \boldsymbol{\theta}_G) \\ &= \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{z}_{\mathrm{pa}(j)}(\boldsymbol{x}; \boldsymbol{\theta}_G); \boldsymbol{\theta}_j).\end{aligned}$$

This holds because $\boldsymbol{\theta}_j$ can only affect $\mathcal{L}_G^{\mathrm{BP}}$ through the node value z_j, and because the feedforward values of non-input nodes are defined using the relation $z_j(\boldsymbol{x}; \boldsymbol{\theta}_G) := f_j(\boldsymbol{z}_{\mathrm{pa}(j)}(\boldsymbol{x}; \boldsymbol{\theta}_G); \boldsymbol{\theta}_j)$. The term $\frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{z}_{\mathrm{pa}(j)}(\boldsymbol{x}; \boldsymbol{\theta}_G); \boldsymbol{\theta}_j)$ is readily obtainable, while the term $\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y}))$ can be expressed as a recursive relation using the chain rule. If $j \in V_{\boldsymbol{y}}$

— an output node — then since output nodes have no children, we get

$$\begin{aligned}\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y})) &= \sum_{k\in V_{\boldsymbol{y}}} \frac{\partial \ell_k^{\mathrm{BP}}}{\partial z_j}((\boldsymbol{y})_k, z_k(\boldsymbol{x};\boldsymbol{\theta}_G)) \\ &= \frac{\partial \ell_j^{\mathrm{BP}}}{\partial z_j}((\boldsymbol{y})_j, z_j(\boldsymbol{x};\boldsymbol{\theta}_G)),\end{aligned}$$

and if $j \in V_{\text{hid}}$ — a hidden node — then by the chain rule, we have

$$\begin{aligned}\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y})) &= \sum_{k\in \mathrm{ch}(j)} \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_k}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y}))\frac{\partial z_k}{\partial z_j}(\boldsymbol{x};\boldsymbol{\theta}_G) \\ &= \sum_{k\in \mathrm{ch}(j)} \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_k}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y}))\frac{\partial f_k}{\partial z_j}(\boldsymbol{z}_{\mathrm{pa}(k)}(\boldsymbol{x};\boldsymbol{\theta}_G);\boldsymbol{\theta}_k).\end{aligned}$$

Putting the equations together, we get that the gradient descent parameter steps $\Delta\boldsymbol{\theta}_j^{\mathrm{BP}}$ for node $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ is given by

$$\Delta\boldsymbol{\theta}_j^{\mathrm{BP}} = -\eta_{\boldsymbol{\theta}} \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y}))\frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{z}_{\mathrm{pa}(j)};\boldsymbol{\theta}_j), \quad \text{where} \tag{8}$$

$$\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y})) = \begin{cases} \frac{\partial \ell_j^{\mathrm{BP}}}{\partial z_j}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i;\boldsymbol{\theta}_G)) & j \in V_{\boldsymbol{y}} \\ \sum_{k\in \mathrm{ch}(j)} \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_k}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y}))\frac{\partial f_k}{\partial z_j}(\boldsymbol{z}_{\mathrm{pa}(k)};\boldsymbol{\theta}_k) & j \in V_{\text{hid}} \end{cases} \tag{9}$$

In practice, the terms $\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y}))$ for non-input nodes $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ can be calculated efficiently by calculating them in a reverse topological order, hence the name "backpropagation". This works because by following a reverse topological order, each node's children are visited before the node itself, so Eq. (9) is always evaluated after all the terms on the right hand side have been calculated. Hence, in BP, for each observed datapoint $(\boldsymbol{x},\boldsymbol{y})$, the node values z_j are calculated in topological order, which we call the **forward pass**, and then the parameter updates $\Delta\boldsymbol{\theta}_j^{\mathrm{BP}}$ are calculated in reverse topological order, which we call the **backward pass**.

The BP algorithm is described in Algorithm 1. As the focus of this paper is on the individual update steps, the stopping criteria for $\boldsymbol{\theta}_G$ are of less relevance and are not specified. Additionally, the datapoints $i \in [n]$ can be traversed in any order.

3.1.1 Backpropagation with squared loss (BP-SQ)

The **squared error node loss**

$$\ell^{\mathrm{SQ}}(a,b) := \frac{1}{2}(a-b)^2 \tag{10}$$

is a node loss function often used for regression. If all of the node loss functions of BP are the squared error node loss, i.e. $\forall j \in V_{\boldsymbol{y}} : \ell_j^{\mathrm{BP}} = \ell^{\mathrm{SQ}}$, then simplifying (9), we get for $j \in V_{\boldsymbol{y}}$

$$\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G;(\boldsymbol{x},\boldsymbol{y})) = z_j(\boldsymbol{x};\boldsymbol{\theta}_G) - (\boldsymbol{y})_j \tag{11}$$

Algorithm 1: Backpropagation with stochastic gradient descent (BP)

```
Input: Computational graph $G$, step size $\eta_{\boldsymbol{\theta}}$, max number of epochs $c$
Data: paired data $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [n]}$
Result: $\boldsymbol{\theta}_G$
Initialize $\boldsymbol{\theta}_G$ randomly;
while $\boldsymbol{\theta}_G$ not converged and number of epochs $< c$ do
    for $i \in [n]$ do
        /* Forward pass */
        for $j \in V$ traversed through $\tau_G$ do
            Calculate feedforward values $z_j(\boldsymbol{x}_i; \boldsymbol{\theta}_G)$ using Eq. (4)
        end
        /* Backward pass */
        for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ traversed through reverse $\tau_G$ do
            Calculate $\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_j}(\boldsymbol{\theta}_G; (\boldsymbol{x}_i, \boldsymbol{y}_i))$ using Eq. (9);
            Calculate $\Delta \boldsymbol{\theta}_j^{\mathrm{BP}}$ using Eq. (8);
            $\boldsymbol{\theta}_j \leftarrow \boldsymbol{\theta}_j + \Delta \boldsymbol{\theta}_j^{\mathrm{BP}}$;
        end
    end
end
return $\boldsymbol{\theta}_G$
```

with no changes to the rest of the algorithm. We denote this flavor of backpropagation with stochastic gradient descent as **BP-SQ**.

3.2 Predictive coding algorithm with general loss (PC)

Just as with BP, **predictive coding with general loss** (**PC**) is a supervised learning algorithm for computational graphs: given a computational graph G that models the relationship between input and output variables $\boldsymbol{X}$ and $\boldsymbol{Y}$, it learns its graph parameters $\boldsymbol{\theta}_G$ from a paired dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i\in[n]}$. The predictive coding algorithm is derived from the predictive coding theory of cognition, and unlike BP, is meant to be a more biologically plausible algorithm for learning in the brain [12, 10, 11].

3.2.1 Derivation

In this section, we will derive PC as an algorithm to minimize the sum of local node losses.

Let us first recall the main idea behind predictive coding theory. The cortex of the brain is made up of several layers, with sensory information entering the network at the lowest, least abstract layer, and this information needs to make its way to the highest, most abstract layer. Each layer of the cortex predicts the values of the layer below, and the deviation between the predicted and observed values is used to modify the connections between the layers so that the deviation is decreased [8].

PC differs from BP in that there is a local loss at each layer that the layer tries to minimize, whereas in BP, there is a global loss that all layers try to minimize. Thus instead of having one large computational graph, the idea is to treat each non-input node, together with its parents, like a small computational graph with its own node loss, and applying stochastic gradient descent to minimize the node loss of each node separately.

To have a node loss be defined for each non-input node, each non-input node needs an "output value" to compare with the node function value $f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$. For this, we introduce **predictive coding node values** v_j for all nodes $j \in V$. Each v_j is a scalar real value stored at node j, which can be interpreted as the guess of the "true value" of that node given the input and output datapoint $\boldsymbol{x}$ and $\boldsymbol{y}$ for the entire computational graph G.

In terms of the theory of predictive coding, the node values can be seen as modeling the guesses of the true values at each cortex layer. However, for each observed datapoint, we need to find the "best guess" values first before we modify the node parameters. This is so that the parameters can learn from node values that accurately reflect the observed datapoint. For input and output nodes, the predictive coding node values $\boldsymbol{v}_{\boldsymbol{x}}$ and $\boldsymbol{v}_{\boldsymbol{y}}$ are simply the observed input and output datapoints $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. For the hidden nodes, the node values are guesses of the "true value" of that node, and the node loss functions serve as our measure of how good that guess is. Hence we aim to find values $\boldsymbol{v}_j$ for $j \in V_{\mathrm{hid}}$ that minimize the sum of all node loss functions — this sum of node loss functions is then our datapoint loss function for PC $\mathcal{L}_G^{\mathrm{PC}}$. With the best guess values, a stochastic gradient descent step can then be performed on the datapoint loss function with respect to the parameters $\boldsymbol{\theta}_G$.

For each node $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$, the node loss function ℓ_j^{PC} is applied on the predictive coding node value v_j and the node function value of the node values of its parents, $f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$:

$$\ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)). \tag{12}$$

The datapoint loss function for PC, $\mathcal{L}_G^{\mathrm{PC}}$, is then the sum of the node loss functions ℓ_j^{PC} for all non-input nodes j, with the input and output node values assigned to the input and output datapoints $(\boldsymbol{x}, \boldsymbol{y})$:

$$\mathcal{L}_G^{\mathrm{PC}}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) := \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} . \tag{13}$$

Notice that unlike the datapoint loss function for BP, the datapoint loss function for PC also depends on the hidden node values $\boldsymbol{v}_{\mathrm{hid}}$.

PC thus works in two phases for each datapoint: first to optimize the node values v_j for $j \in V_{\mathrm{hid}}$ (the **inference phase**) and second to adjust the parameters $\boldsymbol{\theta}_G$ (the **learning phase**) [21], and in both phases $\mathcal{L}_G^{\mathrm{PC}}$ is the function used for calculating the gradient descent step.

We can now derive the gradient descent update steps for both the inference and learning phases. In the inference phase, the gradient of $\mathcal{L}_G^{\mathrm{PC}}$ with respect to the node value v_j for $j \in V_{\mathrm{hid}}$ is

$$\begin{aligned}
&\frac{\partial \mathcal{L}_G^{\mathrm{PC}}}{\partial v_j}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) \\
&= \left[\sum_{k \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \frac{\partial \ell_k^{\mathrm{PC}}}{\partial v_j}(v_k, f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \left[\frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) + \sum_{k \in \mathrm{ch}(j)} \frac{\partial \ell_k^{\mathrm{PC}}}{\partial v_j}(v_k, f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \left[\frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) + \sum_{k \in \mathrm{ch}(j)} \frac{\partial \ell_k^{\mathrm{PC}}}{\partial f_k}(v_k, f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} ,
\end{aligned}$$

since v_j only affects the node losses of the node j and the child nodes of j. Note that only hidden nodes are optimized. The gradient descent step is then

$$\Delta v_j^{\mathrm{PC}} = -\eta_v \frac{\partial \mathcal{L}_G^{\mathrm{PC}}}{\partial v_j}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) \tag{14}$$

for $j \in V_{\mathrm{hid}}$, where $\eta_v > 0$ is the learning rate.

We use v_j^* for $j \in V$ to denote the **post-inference node values**, the node values obtained at the end of the inference phase and used in the learning phase. Unless otherwise specified, we assume that the node values have reached equilibrium by the end of the inference phase, i.e. $\Delta v_j^{\mathrm{PC}} = 0$ for all $j \in V_{\mathrm{hid}}$ at the post-inference node values.

In the learning phase, the gradient of $\mathcal{L}_G^{\mathrm{PC}}$ with respect to the node parameters v_j for $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$ is

$$\begin{aligned}
\frac{\partial \mathcal{L}_G^{\mathrm{PC}}}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) &= \left[\sum_{k \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \frac{\partial \ell_k^{\mathrm{PC}}}{\partial \boldsymbol{\theta}_j}(v_k, f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \left[\frac{\partial \ell_j^{\mathrm{PC}}}{\partial \boldsymbol{\theta}_j}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \left[\frac{\partial \ell_j^{\mathrm{PC}}}{\partial f_j}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} .
\end{aligned}$$

The gradient descent step for node parameters $\boldsymbol{\theta}_j$ is then for the gradient evaluated at the post-inference node values v_j^* for $j \in V$,

$$\Delta\boldsymbol{\theta}_j^{\mathrm{PC}} = -\eta_{\boldsymbol{\theta}} \frac{\partial \mathcal{L}_G^{\mathrm{PC}}}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\mathrm{hid}}^*, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) \tag{15}$$

for $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$, where $\eta_{\boldsymbol{\theta}} > 0$ is the learning rate.

In summary, the PC gradient descent steps are given by:

Inference phase:

$$\forall j \in V_{\mathrm{hid}}: \quad \Delta v_j^{\mathrm{PC}} = -\eta_v \left[\frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) + \sum_{k \in \mathrm{ch}(j)} \frac{\partial \ell_k^{\mathrm{PC}}}{\partial f_k}(v_k, f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \tag{16}$$

Learning phase:

$$\forall j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}: \quad \Delta\boldsymbol{\theta}_j^{\mathrm{PC}} = -\eta_{\boldsymbol{\theta}} \frac{\partial \ell_j^{\mathrm{PC}}}{\partial f_j}(v_j^*, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}^*; \boldsymbol{\theta}_j)) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\mathrm{pa}(j)}^*; \boldsymbol{\theta}_j) \tag{17}$$

The algorithm is described in Algorithm 2. A few comments: While we assume for our analysis that the node values converge in the inference phase, in practice there is no guarantee how many steps it would take for the node values to converge using gradient descent, so a stopping criterion is employed to stop the inference phase when the values are "close enough" to an equilibrium. As the focus of this paper is on the individual update steps, the stopping criteria for $\boldsymbol{\theta}_G$ and $\boldsymbol{v}_{\mathrm{hid}}$ are of less relevance and are not specified. The datapoints $i \in [n]$ can be traversed in any order. The calculations for Δv_j^{PC} for all vertices can be done concurrently; the same goes for v_j. For-loops involving nodes j can be done for all nodes concurrently.

3.2.2 Predictive coding algorithm with squared loss (PC-SQ)

For **predictive coding with squared loss** (**PC-SQ**), we choose in particular the squared error loss $\ell^{\mathrm{SQ}}(a, b) := \frac{1}{2}(a-b)^2$ for all node loss functions $\ell_j^{\text{PC-SQ}}$, $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$, akin to BP-SQ. We thus get

$$\mathcal{L}_G^{\text{PC-SQ}}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) = \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \frac{1}{2}(v_j - f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))^2 \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \tag{18}$$

for the datapoint loss function of PC-SQ, $\mathcal{L}_G^{\text{PC-SQ}}$.

In the inference phase, the gradient of $\mathcal{L}_G^{\text{PC-SQ}}$ with respect to v_j for $j \in V_{\mathrm{hid}}$ then

Algorithm 2: Predictive coding algorithm

```
Input: Computational graph $G$, learning rates $\eta_v, \eta_{\boldsymbol{\theta}} > 0$ for node values and
       parameters respectively, max number of epochs $c$
Data: paired data $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i\in[n]}$
Result: $\boldsymbol{\theta}_G$
Initialize $\boldsymbol{\theta}_G$ randomly;
Initialize $\forall j \in V_{\text{hid}} : v_j \leftarrow 0$;
while $\boldsymbol{\theta}_G$ not converged and number of epochs $< c$ do
    for $i \in [n]$ do
        $\boldsymbol{v}_{\boldsymbol{x}} \leftarrow \boldsymbol{x}_i$;
        $\boldsymbol{v}_{\boldsymbol{y}} \leftarrow \boldsymbol{y}_i$;
        /* Inference phase */
        while $\boldsymbol{v}_{hid}$ not converged do
            for $j \in V_{hid}$ do
                Calculate $\Delta v_j^{\text{PC}}$ using Eq. (16)
            end
            for $j \in V_{hid}$ do
                $v_j \leftarrow v_j + \Delta v_j^{\text{PC}}$;
            end
        end
        /* Learning phase */
        for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ do
            Calculate $\Delta\boldsymbol{\theta}_j^{\text{PC}}$ using Eq. (17);
        end
        for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ do
            $\boldsymbol{\theta}_j \leftarrow \boldsymbol{\theta}_j + \Delta\boldsymbol{\theta}_j^{\text{PC}}$;
        end
    end
end
return $\boldsymbol{\theta}_G$
```

becomes

$$\begin{aligned}
&\frac{\partial \mathcal{L}_G^{\text{PC-SQ}}}{\partial v_j}(\boldsymbol{v}_{\text{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) \\
&= \left[\sum_{k \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \frac{\partial}{\partial v_j} \frac{1}{2} (v_k - f_k(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k))^2 \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} \\
&= \left[(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)) + \sum_{k \in \text{ch}(j)} (v_k - f_k(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k)) \cdot -\frac{\partial f_k}{\partial v_j}(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} .
\end{aligned}$$

In the learning phase, we get for the gradient of $\mathcal{L}_G^{\text{PC-SQ}}$ with respect to $\boldsymbol{\theta}_j$, for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$,

$$\begin{aligned}
\frac{\partial \mathcal{L}_G^{\text{PC-SQ}}}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\text{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) &= \left[\sum_{k \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \frac{\partial}{\partial \boldsymbol{\theta}_j} \frac{1}{2} (v_k - f_k(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k))^2 \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} \\
&= \left[-(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} .
\end{aligned}$$

The gradient descent steps of PC-SQ are then

Inference phase:

$$\begin{aligned}
\forall j \in V_{\text{hid}} : \quad \Delta v_j^{\text{PC-SQ}} = \eta_v \Bigg[&- (v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)) + \\
&\sum_{k \in \text{ch}(j)} (v_k - f_k(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k)) \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k) \Bigg]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}}
\end{aligned} \tag{19}$$

Learning phase:

$$\forall j \in V_{\text{hid}} \cup V_{\boldsymbol{y}} : \quad \Delta \boldsymbol{\theta}_j^{\text{PC-SQ}} = \eta_{\boldsymbol{\theta}} (v_j^* - f_j(\boldsymbol{v}_{\text{pa}(j)}^*; \boldsymbol{\theta}_j)) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\text{pa}(j)}^*; \boldsymbol{\theta}_j) \tag{20}$$

3.2.3 Predictive coding with squared loss and errors (PC-SQ-e)

Predictive coding with squared loss and errors (**PC-SQ-e**), known in the literature as **inference learning** (**IL**), is the mainstream variety of predictive coding algorithm, and is described in [4] and [22]. It is a modification to the PC-SQ algorithm with the introduction of error values.

Notice that when v_j appears in the gradient descent step equations of PC-SQ, (19) and (20), it always occurs in the form $v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)$. The idea is to simplify the equations by introducing **error values** ϵ_j for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ that are stored at their respective nodes and that should be equal to $v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)$. However, we do not set ϵ_j directly to that value in PC-SQ-e. In order to match the update rule of ϵ_j with that of the node values v_j and node parameters $\boldsymbol{\theta}_j$, an iterative update scheme is also used for ϵ_j. We would like that at convergence in the inference phase, for all $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$, the relation $\epsilon_j^* = v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ be satisfied. This can be done by updating the error with steps

$$\Delta\epsilon_j^{\text{PC-SQ-e}} = \eta_\epsilon \left(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) - \epsilon_j\right),$$

where its learning rate satisfies $0 < \eta_\epsilon < 2$. With this, and by replacing $v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ with ϵ_j in the gradient descent step equations (19) and (20), we get the PC-SQ-e update formulas:

Inference phase:

$$\forall j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}: \quad \Delta\epsilon_j^{\text{PC-SQ-e}} = \eta_\epsilon \left[v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) - \epsilon_j\right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x}\\ \boldsymbol{v_y}=\boldsymbol{y}}} \tag{21}$$

$$\forall j \in V_{\text{hid}}: \quad \Delta v_j^{\text{PC-SQ-e}} = \eta_v \left[-\epsilon_j + \sum_{k\in\text{ch}(j)} \epsilon_k \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}_{\text{pa}(k)}; \boldsymbol{\theta}_k)\right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x}\\ \boldsymbol{v_y}=\boldsymbol{y}}} \tag{22}$$

Learning phase:

$$\forall j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}: \quad \Delta\boldsymbol{\theta}_j^{\text{PC-SQ-e}} = \eta_{\boldsymbol{\theta}} \epsilon_j^* \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j) \tag{23}$$

The PC-SQ-e and PC-SQ algorithms do not have exactly the same inference phase because of the introduction of the error values ϵ_j. However, the two algorithms have matching equilibria in the inference phase.

Theorem 1 (Matching inference equilibria for PC-SQ-e and PC-SQ)**.** *On the same computational graph G, graph parameters $\boldsymbol{\theta}_G$, and paired datapoint $(\boldsymbol{x}, \boldsymbol{y})$, $v_j = v_j^*$ for $j \in V_{hid}$ is an equilibrium for the inference phase of PC-SQ if and only if $v_j = v_j^*$ for $j \in V_{hid}$ and $\epsilon_j = \epsilon_j^* = v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ is an equilibrium for the inference phase of PC-SQ-e.*

Proof. ($\implies$) Let $v_j = v_j^*$ for $j \in V_{\text{hid}}$ be an equilibrium in the inference phase of PC-SQ, and let us consider the point $v_j = v_j^*$ for $j \in V_{\text{hid}}$ and $\epsilon_j = v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ in the inference phase of PC-SQ-e. First, the gradient descent steps for the errors, from (21), are zero. Then for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$, we get

$$\begin{aligned}\Delta\epsilon_j^{\text{PC-SQ-e}} &= \eta_\epsilon \left[v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j) - \epsilon_j\right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x}\\ \boldsymbol{v_y}=\boldsymbol{y}}} \\ &= \eta_\epsilon \left[v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j) - (v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j))\right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x}\\ \boldsymbol{v_y}=\boldsymbol{y}}} \\ &= 0.\end{aligned}$$

The gradient descent steps for the node values, from (22), would be, for $j \in V_{\text{hid}}$,

$$\begin{aligned}&\Delta v_j^{\text{PC-SQ-e}} \\ &= \eta_v \left[-\epsilon_j^* + \sum_{k\in\text{ch}(j)} \epsilon_k^* \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}^*_{\text{pa}(k)}; \boldsymbol{\theta}_k)\right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x}\\ \boldsymbol{v_y}=\boldsymbol{y}}} \\ &= \eta_v \left[-(v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)) + \sum_{k\in\text{ch}(j)} (v_k^* - f_k(\boldsymbol{v}^*_{\text{pa}(k)}; \boldsymbol{\theta}_k)) \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}^*_{\text{pa}(k)}; \boldsymbol{\theta}_k)\right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x}\\ \boldsymbol{v_y}=\boldsymbol{y}}} \\ &= \Delta v_j^{\text{PC-SQ}} = 0.\end{aligned}$$

We thus conclude that $v_j = v_j^*$ for $j \in V_{\text{hid}}$ and $\epsilon_j = v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ is an equilibrium for PC-SQ-e.

($\Longleftarrow$) Let $v_j = v_j^*$ for $j \in V_{\text{hid}}$ and $\epsilon_j = v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ be an equilibrium in the inference phase of PC-SQ-e. Then the gradient descent step for the node value v_j in PC-SQ would be, from (19),

$$\begin{aligned}
&\Delta v_j^{\text{PC-SQ}} \\
&= \eta_v \left[-(v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)) + \sum_{k \in \text{ch}(j)} (v_k^* - f_k(\boldsymbol{v}^*_{\text{pa}(k)}; \boldsymbol{\theta}_k)) \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}^*_{\text{pa}(k)}; \boldsymbol{\theta}_k) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \eta_v \left[-\epsilon_j^* + \sum_{k \in \text{ch}(j)} \epsilon_k^* \frac{\partial f_k}{\partial v_j}(\boldsymbol{v}^*_{\text{pa}(k)}; \boldsymbol{\theta}_k) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \Delta v_j^{\text{PC-SQ-e}} = 0.
\end{aligned}$$

Hence at $v_j = v_j^*$ for $j \in V_{\text{hid}}$, the inference phase of PC-SQ is also at equilibrium. □

The following corollary about the resulting parameter update step follows directly from the above theorem:

Corollary 1. *If PC-SQ and PC-SQ-e reach the same equilibrium in the inference phase, then their parameter update steps in the learning phases are identical.*

Proof. The parameter update step for PC-SQ and PC-SQ-e are respectively, for all $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$,

$$\begin{aligned}
\Delta \boldsymbol{\theta}_j^{\text{PC-SQ}} &= \eta_{\boldsymbol{\theta}} (v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j) \quad \text{and} \\
\Delta \boldsymbol{\theta}_j^{\text{PC-SQ-e}} &= \eta_{\boldsymbol{\theta}} \epsilon_j^* \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j),
\end{aligned}$$

from (20) and (23). Since from Theorem 1 we know that at the same equilibrium, it holds that the equilibrium node values v_j^* are identical and $\epsilon_j^* = v_j^* - f_j(\boldsymbol{v}^*_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ for all $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$, we conclude that the parameter update steps of the two algorithms are identical if the same inference equilibrium is reached. □

The PC-SQ-e algorithm is given in Algorithm 3. A few comments: As the focus of this paper is on the individual update steps, the stopping criteria for $\boldsymbol{\theta}_G$, $\boldsymbol{\epsilon}_G$, and $\boldsymbol{v}_{\text{hid}}$ are of less relevance and are not specified. Furthermore, the exact implementation may vary; for example, [38] chose to always stop the inference phase after 20 steps, while [22] chose to stop at "convergence", which took around 100 to 200 inference steps. The datapoints $i \in [n]$ can be traversed in any order. The calculations for $\Delta\epsilon_j^{\text{PC-SQ-e}}$ and $\Delta v_j^{\text{PC-SQ-e}}$ for all vertices can be done concurrently; the same goes for ϵ_j and v_j. For-loops involving nodes j can be done for all nodes concurrently.

3.2.4 Other node losses

The squared loss is not the only loss able to be used in predictive coding; PC allows for any differentiable loss function to be used. Here we illustrate two machine learning tasks — in addition to regression for PC-SQ-e — that require the use of two different loss functions in PC: classification and probability distribution fitting.

Algorithm 3: Predictive coding with squared loss & errors (PC-SQ-e)

```
Input: Computational graph $G$, learning rates for errors, values, and parameters
       $\eta_\epsilon, \eta_v, \eta_{\boldsymbol{\theta}}$, max number of epochs $c$
Data: paired data $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [n]}$
Result: $\boldsymbol{\theta}_G$
Initialize $\boldsymbol{\theta}_G$ randomly;
Initialize $\forall j \in V_{\text{hid}} \cup V_{\boldsymbol{y}} : \epsilon_j \leftarrow 0$;
Initialize $\forall j \in V_{\text{hid}} : v_j \leftarrow 0$;
while $\boldsymbol{\theta}_G$ not converged and number of epochs $< c$ do
    for $i \in [n]$ do
        $\boldsymbol{v}_{\boldsymbol{x}} \leftarrow \boldsymbol{x}_i$;
        $\boldsymbol{v}_{\boldsymbol{y}} \leftarrow \boldsymbol{y}_i$;
        /* Inference phase */
        while $\boldsymbol{\epsilon}_G$, $\boldsymbol{v}_G$ not converged do
            for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ do
                Calculate $\Delta\epsilon_j^{\text{PC-SQ-e}}$ using Eq. (21);
            end
            for $j \in V_{hid}$ do
                Calculate $\Delta v_j^{\text{PC-SQ-e}}$ using Eq. (22);
            end
            for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ do
                $\epsilon_j \leftarrow \epsilon_j + \Delta\epsilon_j^{\text{PC-SQ-e}}$;
            end
            for $j \in V_{hid}$ do
                $v_j \leftarrow v_j + \Delta v_j^{\text{PC-SQ-e}}$;
            end
        end
        /* Learning phase */
        for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ do
            Calculate $\Delta\boldsymbol{\theta}_j^{\text{PC-SQ-e}}$ using Eq. (23);
        end
        for $j \in V_{hid} \cup V_{\boldsymbol{y}}$ do
            $\boldsymbol{\theta}_j \leftarrow \boldsymbol{\theta}_j + \Delta\boldsymbol{\theta}_j^{\text{PC-SQ-e}}$;
        end
    end
end
return $\boldsymbol{\theta}_G$
```

Multiclass classification using PC

Supervised learning of a multiclass classification task can be done using PC through the use of the cross-entropy node loss. Let $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i\in[n]}$ be a paired dataset where the output data $\boldsymbol{y}_i \in \{0,1\}^{d_{\boldsymbol{y}}}$ is in a one-hot encoded representation of categorical data with $d_{\boldsymbol{y}}$ categories — to encode category k, we have the output data $\boldsymbol{y}$ be the $\mathbf{0}$-vector, with the kth entry replaced by a 1. Next, let G be a computational graph with each of its output nodes representing the probability that a given input datapoint corresponds to each of the $d_{\boldsymbol{y}}$ categories — this can be done by having the output nodes being the output of the softmax function applied onto their parent layer of nodes. Then multiclass classification can be done by having the output nodes $j \in V_{\boldsymbol{y}}$ take the cross-entropy loss as the node loss ℓ_j^{PC}. The **cross-entropy loss** ℓ^{CE} is

$$\ell^{\text{CE}}(a,b) := -a\log b, \tag{24}$$

where a is the node value (the one-hot encoded categorical output data), log is the natural logarithm, and b is the function of its parent nodes (the probabilities of the input data being of each of the categories). This node loss function leads to multiclass classification because the sum of the node losses of the output nodes,

$$\begin{aligned}\left[\sum_{j\in V_{\boldsymbol{y}}} \ell_j^{\text{PC}}(v_j, f_j(\boldsymbol{v}_{\text{pa}(j)};\boldsymbol{\theta}_j))\right]_{\boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}} &= \sum_{j\in V_{\boldsymbol{y}}} -(\boldsymbol{y})_j \log(f_j(\boldsymbol{v}_{\text{pa}(j)};\boldsymbol{\theta}_j)) \\ &= -\log\left(\prod_{j\in V_{\boldsymbol{y}}} f_j(\boldsymbol{v}_{\text{pa}(j)};\boldsymbol{\theta}_j)^{(\boldsymbol{y})_j}\right)\end{aligned}$$

is the negative log-likelihood of observing the output data $\boldsymbol{y}$ given that the output node functions $f_j(\boldsymbol{v}_{\text{pa}(j)};\boldsymbol{\theta}_j)$ give the likelihood of each of the categories.

It is important to note that only the *output* node losses need to be the cross-entropy node loss; there are no additional limitations on what the hidden node losses could be. This means that the hidden nodes can take the squared loss, and thus multiclass classification can also be done on PC-SQ-e — the output nodes then take the cross-entropy loss and do not have error values. This also means that a mixed regression–classification setup is also possible: some output nodes can take the cross-entropy loss, while others can take other node losses, e.g. squared loss.

Probability distribution fitting using PC

PC can also be used for the unsupervised learning problem of probability distribution fitting. Given a dataset $D = \{\boldsymbol{x}_i\}_{i\in[n]}$ containing i.i.d. samples of a random variable $\boldsymbol{X}$, and given a class of continuous probability distributions whose probability density function is represented by a computational graph G and parameterized by parameters $\boldsymbol{\theta}_G \in \Theta_G$, the probability distribution fitting task is to find the parameter values such that the corresponding probability density given by the output node function best matches the probability density of the underlying random variable. Since the probability density at a given point is a scalar, the computational graph G only has one output node, which we denote by $k \in V_{\boldsymbol{y}}$.

The loss function for the output nodes that achieves this can be derived through considering maximizing the log-likelihood of observing the dataset assuming that the function value at the output node, $f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)$, gives the probability density:

$$\begin{aligned}
&\underset{\boldsymbol{\theta}_G \in \Theta_G}{\operatorname{argmax}} \log \left(\prod_{i \in [n]} \left[f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) \right]_{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i} \right) \\
&= \underset{\boldsymbol{\theta}_G \in \Theta_G}{\operatorname{argmax}} \sum_{i \in [n]} \left[\log \left(f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) \right) \right]_{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i} \\
&= \underset{\boldsymbol{\theta}_G \in \Theta_G}{\operatorname{argmin}} \frac{1}{n} \sum_{i \in [n]} \left[-\log \left(f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) \right) \right]_{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i} .
\end{aligned}$$

This optimization problem can thus be seen as doing *batch* gradient descent — gradient descent on the average

$$\frac{1}{n} \sum_{i \in [n]} \mathcal{L}_G(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; \boldsymbol{x}_i) \tag{25}$$

of the datapoint loss functions $\mathcal{L}_G$ evaluated for each datapoint of the whole dataset — with the datapoint loss function

$$\mathcal{L}_G(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; \boldsymbol{x}) = \left[-\log \left(f_k(\boldsymbol{v}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) \right) \right]_{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}} . \tag{26}$$

To fit this within the framework of PC, we approximate batch gradient descent with stochastic gradient descent with the same datapoint loss function $\mathcal{L}_G$, and this datapoint loss function is then equivalent to having the node loss function

$$\ell^{\mathrm{LH}}(a, b) = -\log b \tag{27}$$

on the output node $l \in V_{\boldsymbol{y}}$. We call ℓ^{LH} the **likelihood node loss function**. Notice that as this is an *unsupervised* learning problem, this node loss function does not depend on its first argument, the node value, but only on its second argument, the node function, and in the datapoint loss function $\mathcal{L}_G$, only the input node values are assigned (to the input data).

3.2.5 Statistical interpretation of PC and PC-SQ-e

Predictive coding also has a statistical interpretation and can be derived from applying variational inference on a statistical model [4]. In this statistical model, which we denote by $\mathcal{M}$, the computational graph G is interpreted as a *Bayesian network*: each non-input node $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$ has a random variable A_j that is modeled as being a sample from a distribution $\mathcal{A}_j$ — we call this the **node value distribution**. This distribution is determined completely by the node loss function ℓ_j and the node function value $f_j(\boldsymbol{A}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$ of the random variables of its parent nodes. With the input nodes assigned to input data $\boldsymbol{x}$, a joint distribution across all non-input nodes is defined. This joint distribution is thus dependent on the input data $\boldsymbol{x}$ and the graph parameters $\boldsymbol{\theta}_G$.

We can then perform Bayesian inference on this Bayesian network: given the input and output data $(\boldsymbol{x}, \boldsymbol{y})$, we want to infer the probability distribution of the value at each node. The prior distribution is the unconditioned joint probability distribution of the node variables A_j for $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$. The posterior distribution is the distribution of the hidden node random variables $\boldsymbol{A}_{\mathrm{hid}}$ conditioned on observing the output data, $\boldsymbol{A}_{\boldsymbol{y}} = \boldsymbol{y}$. Thus by Bayes's Theorem, we have, under this statistical model $\mathcal{M}$,

$$\mathbb{P}_{\mathcal{M}}\big(\boldsymbol{A}_{\mathrm{hid}} = \boldsymbol{a}_{\mathrm{hid}} \mid \boldsymbol{A}_{\boldsymbol{y}} = \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G\big) = \frac{\mathbb{P}_{\mathcal{M}}\big(\boldsymbol{A}_{\mathrm{hid}} = \boldsymbol{a}_{\mathrm{hid}}, \boldsymbol{A}_{\boldsymbol{y}} = \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G\big)}{\mathbb{P}_{\mathcal{M}}\big(\boldsymbol{A}_{\boldsymbol{y}} = \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G\big)} .$$

With arbitrary node functions f_j, the posterior distribution cannot in general be expressed as a simple, known distribution. Thus we would like to use variational inference to approximate the posterior. The PC algorithm can be derived as applying variational inference to fit a point mass distribution (Dirac delta distribution) to the posterior [20]. We derive this now.

We first consider the special case of PC-SQ. This corresponds to the network where the node value distribution $\mathcal{A}_j$ is a normal distribution centered around the node function value with variance 1, i.e. $A_j \sim \mathcal{A}_j = \mathcal{N}(f_j(\boldsymbol{A}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j), 1)$ [4]. The joint likelihood is then

$$\begin{aligned}
&\mathbb{P}_{\mathcal{M}}\big(\boldsymbol{A}_{\mathrm{hid}} = \boldsymbol{a}_{\mathrm{hid}}, \boldsymbol{A}_{\boldsymbol{y}} = \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G\big) \\
&= \left[\prod_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \mathbb{P}_{\mathcal{N}}\Big(a_j; f_j(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j), 1\Big)\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\
&= \left[\prod_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(a_j - f_j(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))^2\right)\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} .
\end{aligned}$$

The distribution that we approximate the posterior with is the point mass distribution, which is centered at hidden node values $\boldsymbol{v}_{\mathrm{hid}}$ that are to be optimized in variational inference. We denote this distribution by $\mathcal{D}(\boldsymbol{v}_{\mathrm{hid}})$.

We apply variational inference — we want to find the values of $\boldsymbol{v}_{\mathrm{hid}}$ that minimize the KL divergence between the posterior and the approximating distributions,

$$\begin{aligned}
&D_{\mathrm{KL}}\Big[\mathbb{P}_{\mathcal{D}}(\cdot; \boldsymbol{v}_{\mathrm{hid}}) \;\Big\|\; \mathbb{P}_{\mathcal{M}}(\cdot \mid \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G)\Big] \\
&= \mathbb{E}_{\boldsymbol{A}_{\mathrm{hid}} \sim \mathcal{D}(\boldsymbol{v}_{\mathrm{hid}})}\left[\log \frac{\mathbb{P}_{\mathcal{D}}(\boldsymbol{A}_{\mathrm{hid}}; \boldsymbol{v}_{\mathrm{hid}})}{\mathbb{P}_{\mathcal{M}}(\boldsymbol{A}_{\mathrm{hid}} \mid \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G)}\right] \\
&= \mathbb{E}_{\boldsymbol{A}_{\mathrm{hid}} \sim \mathcal{D}(\boldsymbol{v}_{\mathrm{hid}})}\Big[\log \mathbb{P}_{\mathcal{D}}(\boldsymbol{A}_{\mathrm{hid}}; \boldsymbol{v}_{\mathrm{hid}})\Big] - \mathbb{E}_{\boldsymbol{A}_{\mathrm{hid}} \sim \mathcal{D}(\boldsymbol{v}_{\mathrm{hid}})}\Big[\log \mathbb{P}_{\mathcal{M}}(\boldsymbol{A}_{\mathrm{hid}} \mid \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G)\Big].
\end{aligned}$$

However, the entropy term $\mathbb{E}_{\boldsymbol{A}_{\mathrm{hid}} \sim \mathcal{D}(\boldsymbol{v}_{\mathrm{hid}})}[\log \mathbb{P}_{\mathcal{D}}(\boldsymbol{A}_{\mathrm{hid}}; \boldsymbol{v}_{\mathrm{hid}})] = \log \mathbb{P}_{\mathcal{D}}(\boldsymbol{v}_{\mathrm{hid}}; \boldsymbol{v}_{\mathrm{hid}})$ is infinite, since the point mass distribution has infinite probability density at its center. To obtain a meaningful function to minimize, we simply drop this term. We justify this as follows: The point mass distribution $\mathcal{D}$ has the property that $\boldsymbol{D} \sim \mathcal{D}(\boldsymbol{d}) \iff \boldsymbol{D}' := \boldsymbol{D} - \boldsymbol{d} \sim \mathcal{D}(\boldsymbol{0})$ for all $\boldsymbol{d}$ — that is, its parameter simply shifts the distribution. Additionally, all continuous distributions $\mathcal{C}$ with this property have an entropy that is a constant independent of its parameter $\boldsymbol{c}$:

$$\forall \boldsymbol{c}: \quad \mathbb{E}_{\boldsymbol{C} \sim \mathcal{C}(c)}\Big[\log \mathbb{P}_{\mathcal{C}}(\boldsymbol{C}; \boldsymbol{c})\Big] = \mathbb{E}_{\boldsymbol{C}' \sim \mathcal{C}(\boldsymbol{0})}\Big[\log \mathbb{P}_{\mathcal{C}}(\boldsymbol{C}'; \boldsymbol{0})\Big],$$

where $\boldsymbol{C}' = \boldsymbol{C} - \boldsymbol{c}$. Furthermore, the point mass distribution can be constructed as a limit of continuous distributions with this property. Altogether, this suggests that we can treat the entropy term of the point mass distribution simply as a constant, meaning that dropping the term does not change the shape of the KL divergence function. We instead minimize the term

$$-\mathbb{E}_{\boldsymbol{A}_{\mathrm{hid}} \sim \mathcal{D}(\boldsymbol{v}_{\mathrm{hid}})}\Big[\log \mathbb{P}_{\mathcal{M}}(\boldsymbol{A}_{\mathrm{hid}} \mid \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G)\Big] = -\log \mathbb{P}_{\mathcal{M}}(\boldsymbol{v}_{\mathrm{hid}} \mid \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G).$$

Doing variational inference with a point mass distribution is thus equivalent to finding values that maximize the log likelihood of the posterior distribution, with the maximizer being the optimal parameter of the point mass distribution. Substituting in the model probabilities for the PC-SQ case, we get

$$\begin{aligned}
&-\log \mathbb{p}_{\mathcal{M}}(\boldsymbol{v}_{\text{hid}} \mid \boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta}_G) \\
&= -\log \left[\prod_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))^2\right) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} \\
&= -\left[\sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))^2\right) \right) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} \\
&= -\left[\sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} -\frac{1}{2}(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))^2 + \text{const.} \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} \\
&= \left[\sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \frac{1}{2}(v_j - f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))^2 \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}} + \text{const.} \\
&= \mathcal{L}_G^{\text{PC-SQ}}(\boldsymbol{v}_{\text{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) + \text{const.}
\end{aligned}$$

Therefore, minimizing the KL divergence is equivalent to minimizing the datapoint loss function, done in the inference phase of PC-SQ. Furthermore, the post-inference node values $\boldsymbol{v}_{\text{hid}}^*$ — if they converge to the global minimum — are the maximum a posteriori estimates of the hidden node values in the posterior distribution, as well as parameters of the point mass distribution that best approximates the posterior.

We can further decrease the KL divergence by adjusting the graph parameters $\boldsymbol{\theta}_G$ to lower the datapoint loss function. This then corresponds to the learning phase of PC-SQ.

In addition to framing it as a variational inference problem, it has been shown that predictive coding algorithms are also related to the expectation–maximization (EM) algorithm [20].

We can reverse the above steps to determine the node value distribution that corresponds to a general node loss function of the PC algorithm. Recall that the datapoint loss function of PC is

$$\mathcal{L}_G^{\text{PC}}(\boldsymbol{v}_{\text{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) = \left[\sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\text{PC}}(v_j, f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}}.$$

We posit that this is the negative log-likelihood of some joint probability distribution (up to an additive constant). Then minimizing this datapoint loss function is then equivalent to maximizing the joint probability density (up to a positive multiplicative constant)

$$\left[\prod_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \exp\left(-\ell_j^{\text{PC}}(v_j, f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))\right) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}}}. \tag{28}$$

To make this a proper probability density on graph G, we require that the integral of the term at each node corresponding to its probability distribution, when integrated over all possible node values, equals 1. Hence we normalize it to get the probability density of the node distribution at node j,

$$\mathbb{p}_{\mathcal{A}_j}\left(v_j; f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)\right) = \frac{\exp\left(-\ell_j^{\text{PC}}(v_j, f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))\right)}{\int_{\mathbb{R}} \exp\left(-\ell_j^{\text{PC}}(a, f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j))\right) \mathrm{d}a}. \tag{29}$$

It is only when the normalizing integral is finite for all node function values $f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$ that the probability density function is well-defined. Hence if this condition is satisfied for all node losses ℓ_j, $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$, in a PC algorithm, then its loss-based interpretation has an equivalent statistical interpretation with node probability density as in (29).

3.3 Example

We now look at a toy example of a computational graph and compare the results of executing BP-SQ, PC-SQ, and PC-SQ-e on it. First, let G be the following computational graph:

$$\begin{tikzcd} 0 \arrow[r, "f_1 = \theta_1 a_0"] & 1 \arrow[r, "f_2 = \theta_2 a_1"] & 2 \end{tikzcd}$$

G has nodes $V = \{0, 1, 2\}$, with input node $V_{\boldsymbol{x}} = \{0\}$, hidden node $V_{\text{hid}} = \{1\}$, and output node $V_{\boldsymbol{y}} = \{2\}$. The node functions are $f_1(a_0; \theta_1) = \theta_1 a_0$ and $f_2(a_1; \theta_2) = \theta_2 a_1$. The graph parameters are then $\boldsymbol{\theta}_G = (\theta_1, \theta_2)$, and are initialized as $\theta_1 = \theta_2 = 1$. We perform one iteration of each algorithm on the datapoint $(x, y) = (2, 4)$, with learning rates $\eta_{\boldsymbol{\theta}} = \eta_v = \eta_\epsilon = 0.1$.

3.3.1 BP-SQ

The BP-SQ datapoint loss function for this example is

$$\mathcal{L}_G^{\text{BP}}(\boldsymbol{\theta}_G; (x, y)) = \frac{1}{2}\Big(y - z_2(x; \boldsymbol{\theta}_G)\Big)^2 = \frac{1}{2}\Big(y - \theta_2 \theta_1 x\Big)^2.$$

Forward pass

Given the datapoint $(x, y) = (2, 4)$, at the end of the forward pass, we get for the feedforward values z_0, z_1, z_2:

$$\begin{tikzcd} z_0 = 2 \arrow[r, "f_1 = \theta_1 z_0 = 2"] & z_1 = 2 \arrow[r, "f_2 = \theta_2 z_1 = 2"] & z_2 = 2 \end{tikzcd}$$

Backward pass

For the backward pass, substituting the known terms into the backprop equations (8) and (9), we get

$$\Delta\theta_j^{\text{BP}} = -\eta_{\boldsymbol{\theta}} \frac{\partial \mathcal{L}_G^{\text{BP}}}{\partial z_j}(\boldsymbol{\theta}_G; (x, y)) \cdot z_{\text{pa}(j)}, \quad \text{where}$$

$$\frac{\partial \mathcal{L}_G^{\text{BP}}}{\partial z_j}(\boldsymbol{\theta}_G; (x, y)) = \begin{cases} z_j(x; \boldsymbol{\theta}_G) - y & j \in V_{\boldsymbol{y}} \\ \displaystyle\sum_{k \in \text{ch}(j)} \frac{\partial \mathcal{L}_G^{\text{BP}}}{\partial z_k}(\boldsymbol{\theta}_G; (x, y)) \cdot \theta_k & j \in V_{\text{hid}} \end{cases}.$$

Performing the backward pass, we thus get

$$\begin{aligned}
\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_2}(\boldsymbol{\theta}_G;(x,y)) &= z_2(x;\boldsymbol{\theta}_G) - y = 2-4 = -2\\
\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_1}(\boldsymbol{\theta}_G;(x,y)) &= \frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_2}(\boldsymbol{\theta}_G;(x,y))\cdot\theta_1 = -2\cdot 1 = -2\\
\Delta\theta_2^{\mathrm{BP}} &= -\eta_{\boldsymbol{\theta}}\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_2}(\boldsymbol{\theta}_G;(x,y))\cdot z_1 = -0.1\cdot -2\cdot 2 = 0.4\\
\Delta\theta_1^{\mathrm{BP}} &= -\eta_{\boldsymbol{\theta}}\frac{\partial \mathcal{L}_G^{\mathrm{BP}}}{\partial z_1}(\boldsymbol{\theta}_G;(x,y))\cdot z_0 = -0.1\cdot -2\cdot 2 = 0.4.
\end{aligned}$$

3.3.2 PC-SQ

In PC-SQ, we use the squared node loss function for all (non-input) nodes, so we get for the datapoint loss function

$$\begin{aligned}
\mathcal{L}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G;(x,y)) &= \left[\frac{1}{2}\Big(v_1 - f_1(v_0;\theta_1)\Big)^2 + \frac{1}{2}\Big(v_2 - f_2(v_1;\theta_2)\Big)^2\right]_{\substack{v_0=x\\ v_2=y}}\\
&= \frac{1}{2}\Big(v_1-\theta_1 x\Big)^2 + \frac{1}{2}\Big(y-\theta_2 v_1\Big)^2.
\end{aligned}$$

Inference phase

At the start of the inference phase, the node values are initialized such that the input nodes take the input data values, the output nodes take the output data values, and the hidden nodes are 0:

$$\begin{tikzcd}
\boxed{v_0 = 2} \arrow[r, "f_1 = \theta_1 v_0 = 2"] & \boxed{v_1 = 0} \arrow[r, "f_2 = \theta_2 v_1 = 0"] & \boxed{v_2 = 4}
\end{tikzcd}$$

The node value step equation, (19), simplifies to

$$\Delta v_j^{\text{PC-SQ}} = \eta_v\left[-(v_j - \theta_j v_{\mathrm{pa}(j)}) + \sum_{k\in\mathrm{ch}(j)}(v_k - \theta_k v_{\mathrm{pa}(k)})\cdot\theta_k\right]_{\substack{v_0=x\\ v_2=y}}$$

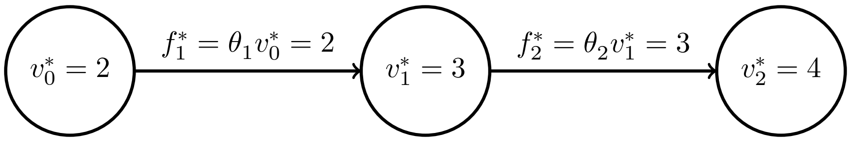
for all hidden nodes $j \in V_{\mathrm{hid}}$. The only hidden node is $j=1$, and plugging in the initial values, we get

$$\begin{aligned}
\Delta v_1^{\text{PC-SQ}} &= \eta_v\Big[-(v_1-\theta_1 v_0) + (v_2 - \theta_2 v_1)\cdot\theta_2\Big]_{\substack{v_0=x\\ v_2=y}}\\
&= 0.1\cdot(-(0-2)+(4-0)\cdot 1) = 0.6.
\end{aligned}$$

After this inference step, the graph looks as follows:

$$\begin{tikzcd}
\boxed{v_0 = 2} \arrow[r, "f_1 = \theta_1 v_0 = 2"] & \boxed{v_1 = 0.6} \arrow[r, "f_2 = \theta_2 v_1 = 0.6"] & \boxed{v_2 = 4}
\end{tikzcd}$$

Node value steps of the inference phase are taken until the node values converge to an equilibrium; in this case, there is only one equilibrium, $v_1^* = 3$. However, in practice, the node value v_1 only approaches but never exactly reaches the equilibrium. For this example, we continue onto the learning phase using the equilibrium node value $v_1^* = 3$.



Learning phase

In the learning phase, the parameter update step equation, (20), simplifies to

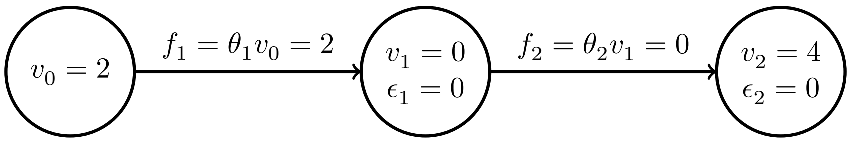
$$\Delta\theta_j^{\text{PC-SQ}} = \eta_{\boldsymbol{\theta}}(v_j^* - \theta_j v_{\text{pa}(j)}^*) \cdot \boldsymbol{v}_{\text{pa}(j)}^*$$

for all non-input nodes $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$. This evaluates to

$$\begin{aligned}\Delta\theta_1^{\text{PC-SQ}} &= \eta_{\boldsymbol{\theta}}(v_1^* - \theta_1 v_0^*) \cdot v_0^* = 0.1 \cdot (3-2) \cdot 2 = 0.2\\ \Delta\theta_2^{\text{PC-SQ}} &= \eta_{\boldsymbol{\theta}}(v_2^* - \theta_2 v_1^*) \cdot v_1^* = 0.1 \cdot (4-3) \cdot 3 = 0.3.\end{aligned}$$

3.3.3 PC-SQ-e

Applying PC-SQ-e to this example, we initialize the node and error values as follows:



Inference phase

In the inference phase, the error (21) and node value step equations (22) simplify to

$$\Delta\epsilon_j^{\text{PC-SQ-e}} = \eta_\epsilon \left[v_j - \theta_j v_{\text{pa}(j)} - \epsilon_j\right]_{\substack{v_0=x\\v_2=y}}$$

for all $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$, and

$$\Delta v_j^{\text{PC-SQ-e}} = \eta_v \left[-\epsilon_j + \sum_{k \in \text{ch}(j)} \epsilon_k \theta_k\right]_{\substack{v_0=x\\v_2=y}}$$

for all $j \in V_{\text{hid}}$. The first inference step is then

$$\begin{aligned}\Delta\epsilon_1^{\text{PC-SQ-e}} &= \eta_\epsilon \left[v_1 - \theta_1 v_0 - \epsilon_1\right]_{\substack{v_0=x\\v_2=y}} = 0.1 \cdot (0-2-0) = -0.2\\ \Delta\epsilon_2^{\text{PC-SQ-e}} &= \eta_\epsilon \left[v_2 - \theta_2 v_1 - \epsilon_2\right]_{\substack{v_0=x\\v_2=y}} = 0.1 \cdot (4-0-0) = 0.4\\ \Delta v_1^{\text{PC-SQ-e}} &= \eta_v \left[-\epsilon_1 + \epsilon_2\theta_2\right]_{\substack{v_0=x\\v_2=y}} = 0.1 \cdot (-0 + 0 \cdot 1) = 0,\end{aligned}$$

which results in the following graph at the end of the first inference step:

$$\begin{tikzcd}[column sep=huge] v_0 = 2 \arrow[r, "f_1 = \theta_1 v_0 = 2"] & \begin{matrix} v_1 = 0 \\ \epsilon_1 = -0.2 \end{matrix} \arrow[r, "f_2 = \theta_2 v_1 = 0"] & \begin{matrix} v_2 = 4 \\ \epsilon_2 = 0.4 \end{matrix} \end{tikzcd}$$

Notice that the node value v_1 has not changed. This is because the errors were all initialized as 0. We perform another inference step to show a change in node values: the second inference step is

$$\begin{aligned}
\Delta\epsilon_1^{\text{PC-SQ-e}} &= \eta_\epsilon \left[v_1 - \theta_1 v_0 - \epsilon_1\right]_{\substack{v_0=x\\v_2=y}} = 0.1 \cdot (0 - 2 + 0.2) = -0.18\\
\Delta\epsilon_2^{\text{PC-SQ-e}} &= \eta_\epsilon \left[v_2 - \theta_2 v_1 - \epsilon_2\right]_{\substack{v_0=x\\v_2=y}} = 0.1 \cdot (4 - 0 - 0.4) = 0.36\\
\Delta v_1^{\text{PC-SQ-e}} &= \eta_v \left[-\epsilon_1 + \epsilon_2 \theta_2\right]_{\substack{v_0=x\\v_2=y}} = 0.1 \cdot (0.2 + 0.4 \cdot 1) = 0.06.
\end{aligned}$$

$$\begin{tikzcd}[column sep=huge] v_0 = 2 \arrow[r, "f_1 = \theta_1 v_0 = 2"] & \begin{matrix} v_1 = 0.06 \\ \epsilon_1 = -0.38 \end{matrix} \arrow[r, "f_2 = \theta_2 v_1 = 0.06"] & \begin{matrix} v_2 = 4 \\ \epsilon_2 = 0.76 \end{matrix} \end{tikzcd}$$

Inference steps are taken until convergence. Since there is only one equilibrium in the PC-SQ inference phase, by Theorem 1, there is also only one equilibrium in the PC-SQ-e inference phase, namely $v_1^* = 3$, $\epsilon_1^* = 1$, $\epsilon_2^* = 1$, which we take as the inference node values:

$$\begin{tikzcd}[column sep=huge] v_0^* = 2 \arrow[r, "f_1^* = \theta_1 v_0^* = 2"] & \begin{matrix} v_1^* = 3 \\ \epsilon_1^* = 1 \end{matrix} \arrow[r, "f_2^* = \theta_2 v_1^* = 3"] & \begin{matrix} v_2^* = 4 \\ \epsilon_2^* = 1 \end{matrix} \end{tikzcd}$$

Learning phase

For the learning phase, by Corollary 1, the parameter update steps of PC-SQ-e are the same as those of PC-SQ, so we get $\Delta\theta_1^{\text{PC-SQ-e}} = 0.2$ and $\Delta\theta_2^{\text{PC-SQ-e}} = 0.3$.

3.3.4 Comparison

To summarize, the parameter update steps of the three algorithms for this example are as follows:

Algorithm	$\Delta\theta_1$	$\Delta\theta_2$
BP-SQ	0.4	0.4
PC-SQ	0.2	0.3
PC-SQ-e	0.2	0.3

First, BP and the predictive coding algorithms do not produce the same parameter update steps, and the deviation between the update steps is in general different for different parameters. This also means that after learning on the same dataset, BP and PC in general produce different parameter values. We also see that with the same parameter learning rate $\eta_{\boldsymbol{\theta}}$, the parameter update steps of PC are smaller in absolute value than those of BP. The reason for this is that the loss in BP is accumulated at the output nodes, while the loss in PC is spread out throughout the graph, which causes the loss at each output node to be in general less in PC than in BP (assuming identical output node loss functions). For the squared loss function (and many other losses), a smaller loss implies a smaller (in absolute value) gradient. Since the parameter update steps are obtained by gradient descent, this explains the smaller (in absolute value) parameter update steps of PC.

We also note that the inference phase of PC-SQ converges faster than in PC-SQ-e — v_1 is already 0.6 at the end of the first inference step in PC-SQ, while it is only 0.06 at the end of the second inference step in PC-SQ-e. This is mostly a consequence of the initialization of the error values ϵ_1, ϵ_2 — they were initialized as 0, which does not accurately reflect the initial value of the difference $v_j - f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$. But even if they are initialized as this difference, the convergence of PC-SQ-e would still be slower than PC-SQ, as the error values ϵ_j lag behind the true difference value. This raises the question: why even consider PC-SQ-e, when PC-SQ is a more computationally efficient algorithm? The answer is that PC-SQ-e is, in addition to a learning algorithm for computational graphs, a candidate model for how the brain learns [20], which we explore in the next section.

4 Biological plausibility

Neuroscientists search for realistic models of cognition. For a model to be realistic, it is necessary for the model to reflect our understanding of the brain at all levels, from the activity potentials of neurons to the connectivity between different regions of the brain [27, 26]. As the focus of the paper is on learning algorithms for neural networks, we focus on the level of *synaptic plasticity* — how the connections between neurons are modified during learning.

As the development of predictive coding algorithms is motivated by a desire for realistic algorithms for synaptic plasticity, in this section, we evaluate how realistic the BP and PC algorithms are. "Realism" for synaptic plasticity is not strictly defined; instead, various criteria for biological plausibility that the learning process in the brain is believed to satisfy have been proposed [1, 5]. We list some commonly mentioned biological plausibility criteria below:

- **Local plasticity**

  Neurons can only be influenced by other neurons that they are connected with. In particular, changes to a synapse should only be able to be induced by the two neurons that the synapse joins. This idea is known as *local plasticity* or *locality* [4, 38, 34, 5]. However, a precise criterion is still elusive, as our understanding of the processes that affect a synapse is still incomplete [5]. We refer to [6] for an extended discussion of these issues and an attempt at mathematically formalizing the idea of local plasticity.

  Nevertheless, we can still attempt to formalize local plasticity within our framework of computational graphs. Recall from Section 2 that a *generalized sum node function* is a node function f_j that is of the form

  $$f_j(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j) = \psi_j\left(\sum_{i \in \mathrm{pa}(j)} \phi_j(a_i; \boldsymbol{\theta}_{j,i}); \boldsymbol{\theta}_{j,j}\right). \tag{30}$$

  This class of node functions models neuronal activity: the terms $\phi_j(a_i; \boldsymbol{\theta}_{j,i})$ model the activities of the parent nodes, and the parameters $\boldsymbol{\theta}_{j,i}$ model the corresponding synaptic weights. If a node j has a generalized sum node function, it may be considered to satisfy local plasticity if the updates of each of its parameters $\boldsymbol{\theta}_{j,i}$ for $i \in \mathrm{pa}(j)$ only depend on the current parameter $\boldsymbol{\theta}_{j,i}$ and the node values a_j and a_i:

  $$\Delta\boldsymbol{\theta}_{j,i} \hookleftarrow \boldsymbol{\theta}_{j,i}, a_j, a_i. \tag{31}$$

- **Hebbian plasticity rules**

  Hebbian plasticity is a general rule for synaptic learning proposed by Hebb in his book *The Organization of Behavior*, published in 1949, which states

  > When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. [15]

  In other words, if (neuronal) cell A is a parent of (neuronal) cell B, and cell A firing causes cell B to fire, then the synaptic connection between cell A and B is strengthened. This idea of plasticity is generally regarded as being important for biological realism [27].

Hebbian plasticity has been quantified as the change in the weight of a synapse being proportional to the product of the neuron activities of its two adjacent neurons [39]. However, it has also been quantified as the change in synaptic weight being *dependent* on the product [4], or that the product is of *monotonic increasing functions of* the two neuron activities [38]. Note that in all of these formulations, Hebbian plasticity implies local plasticity.

Translated into our framework of computational graphs, if the function at node j is a generalized sum node function, then the node has Hebbian plasticity if the parameter step $\Delta\boldsymbol{\theta}_{j,i}$ for all $i \in \text{pa}(j)$ is proportional to (or dependent on) the product of (monotonic increasing functions of) the node values a_i and a_j:

$$\Delta\boldsymbol{\theta}_{j,i} \propto a_j a_i \quad \text{or} \quad \Delta\boldsymbol{\theta}_{j,i} = g_1(a_j)g_2(a_i), \tag{32}$$

where g_1 and g_2 are monotone increasing functions.

- **Uncoupled forward and backward weights**

  Synapses are unidirectional: synapses are used to send signals from the parent neuron to the child neuron, and the child neuron cannot use the same synapse to send signals to the parent. To send feedback to the parent, the child neuron must create a separate connection to the parent with a separate synapse. Since the synapses of the two connections are different, their synaptic weights should be independent and uncoupled [39, 5]. This criterion is unfortunately not met by many learning algorithms — this is known as the *weight transport problem* [23, 18].

  Within our framework of computational graphs, if the node function of j is a generalized sum node function, then this means that the parameters $\boldsymbol{\theta}_{j,i}$, which are used to adjust the value a_j of the child node j, should not be used to adjust the value a_i of the parent node i.

It is generally accepted that BP does not satisfy local plasticity, and this is its main criticism [5, 34]. Even though the equations used in BP only refer to adjacent nodes, the parameter steps calculated in the backwards pass are propagated from the end of the computational graph throughout the graph to each node, and depend on the values and parameters of all the nodes downstream. Because BP does not have local plasticity, it also does not have Hebbian plasticity. Additionally, BP does not have uncoupled forward and backward weights. There have been attempts to resolve this problem. It has been shown that BP with *random* synaptic weights in the backwards direction is able to learn from data, though not as well as unmodified BP [17]. This idea has since been improved upon to reach a level of performance comparable to BP [32].

PC-SQ-e, on the other hand, was motivated primarily as a model for synaptic learning and perception, extending from the predictive coding theory of the brain, and so is made with the goal of being biologically plausible. Unlike PC and PC-SQ, PC-SQ-e contains error values, which can be interpreted as the activity of "error neurons" in the brain as a possible neural interpretation [20, 4]. PC-SQ-e satisfies local plasticity [5] and has Hebbian plasticity rules [4, 39]. However, like BP, it does not have uncoupled forward and backward weights [5]. Results have been published to address this; for example, a modified version of PC-SQ-e, named BioCCPC, has been shown to resolve the weight transport problem for some node functions [14]. There have, however, been other biological plausibility concerns for PC-SQ-e, such as the unrealistically large number of error neurons needed in the brain — one error neuron per node — in some neural interpretations of the algorithm [20].

We conclude that while neither algorithm is fully realistic as a model for synaptic plasticity in the brain, PC-SQ-e is indeed more biologically plausible than BP, and modifications to both have been proposed that improve their realism. We refer the reader to [5] for a further discussion on biological plausible criteria and a comparison of biological plausibility between several learning algorithms.

5 Results

Over the past few years, it has been proven that various predictive coding algorithms and backprop produce approximately or exactly equal parameter update steps. [38] showed that PC-SQ-e approximates BP-SQ for multilayer perceptrons, and later [34] showed that Z-IL, a modified algorithm based on PC-SQ-e, produces exactly the same parameter updates as BP-SQ. This latter result was generalized to arbitrary computational graphs in [30].

Our contribution in this paper is in showing that predictive coding *for general node losses/distributions* has approximately the same parameter updates as BP with the corresponding loss function on arbitrary computational graphs. In Subsection 5.1 we first show that the datapoint loss function always has a minimum in the inference phase of PC. Then in Subsection 5.2 we show that the parameters that minimize BP and PC are the same under some conditions. Finally, in Subsection 5.3, we show that the parameter update steps of BP and PC are approximately equal, and this culminates in a theorem stating that they are equal in a certain limit.

5.1 Minimization of the inference phase of PC

Many results regarding PC require that the datapoint loss function has a minimum in the inference phase, and/or assume that a minimum is attained. Furthermore, we would like to have guarantees about convergence in the inference phase. In this subsection, we prove that such a minimum is guaranteed to exist, as long as the node loss functions satisfy certain conditions. Since the proof for PC (for general node loss functions) is quite technical, we first start with proving the special case of the theorem for squared node loss — i.e. for PC-SQ-e — which uses the same overall proof idea but is more transparent.

Theorem 2 (PC-SQ-e inference phase minimum existence theorem)**.** *Given any computational graph G with parameters $\boldsymbol{\theta}_G$ and learning from any datapoint $(\boldsymbol{x}, \boldsymbol{y})$ using PC-SQ-e, the minimization problem of the datapoint loss function $\mathcal{L}_G^{PC\text{-}SQ}$ w.r.t. the hidden node values $\boldsymbol{v}_{hid}$ (corresponding to the inference phase),*

$$\inf_{\boldsymbol{v}_{hid} \in \mathbb{R}^{|V_{hid}|}} \mathcal{L}_G^{PC\text{-}SQ}(\boldsymbol{v}_{hid}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) = \inf_{\boldsymbol{v}_{hid} \in \mathbb{R}^{|V_{hid}|}} \left[\sum_{j \in V_{hid} \cup V_{\boldsymbol{y}}} \frac{1}{2} (v_j - f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))^2 \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}},$$

is guaranteed to have a global minimum.

Proof. We prove this as follows. First, to simplify the problem, we remove the $\frac{1}{2}$ in the datapoint loss function and denote the resulting equivalent function to be minimized by h:

$$h(\boldsymbol{v}_{\mathrm{hid}}) := 2\mathcal{L}_G^{\mathrm{PC\text{-}SQ}}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) = \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} (v_j - f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))^2 \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}}.$$

Since each f_j is continuous, the function h is also continuous. The minimization is thus on the value of the hidden nodes $\boldsymbol{v}_{\mathrm{hid}}$ over the domain $\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}$. We show that there is a compact subset $U \subset \mathbb{R}^{|V_{\mathrm{hid}}|}$ such that if a minimum exists, it must be in U. Then since U is compact and h is continuous, a minimum exists in U, and thus also in $\mathbb{R}^{|V_{\mathrm{hid}}|}$.

Now the proof. First, let us denote a function value attainable by the function h by h_0. Since h is a sum of squares and so is non-negative, we have that $h_0 \geq 0$. We now construct the U such that the function h has a value greater than h_0 for all points outside of U. Then according to a topological ordering of the computational graph, we visit each non-output node $j \in V_{\boldsymbol{x}} \cup V_{\text{hid}}$ in order and associate with node j a closed interval I_j, whose interpretation is that of a bound for the optimal value of v_j, and is defined as

$$I_j := \begin{cases} [(\boldsymbol{x})_j, (\boldsymbol{x})_j] & j \in V_{\boldsymbol{x}} \\ \left[\min\limits_{\forall i \in \text{pa}(j): a_i \in I_i} f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) - \sqrt{h_0}, \max\limits_{\forall i \in \text{pa}(j): a_i \in I_i} f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) + \sqrt{h_0}\right] & j \in V_{\text{hid}} \end{cases}.$$

Since we defined the I_j following a topological ordering of the computational graph, the right hand side, which only references the parent nodes of j, is always defined when defining I_j. Also, the min and max exist because they are extrema of continuous functions on a compact domains. Hence the I_j are well-defined. We claim that for all $\boldsymbol{v}_{\text{hid}} \in \mathbb{R}^{|V_{\text{hid}}|}$, if the node value of some node is not in its corresponding interval, then $h(\boldsymbol{v}_{\text{hid}}) > h_0$. Let j' denote the earliest node along the topological ordering such that $v_{j'} \notin I_{j'}$. This implies that all of its parent nodes $i \in \text{pa}(j')$ satisfy $v_i \in I_i$. To not be in the interval, the node value $v_{j'}$ must be less than the lower endpoint or greater than the upper endpoint of $I_{j'}$. In the case that it is less than the lower endpoint, we get a lower bound for the term in h corresponding to node j':

$$\begin{aligned} v_{j'} &< \min_{\forall i \in \text{pa}(j'): v_i \in I_i} f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}) - \sqrt{h_0} \\ 0 \le \sqrt{h_0} &< \min_{\forall i \in \text{pa}(j'): v_i \in I_i} f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}) - v_{j'} \\ h_0 &< \left(\min_{\forall i \in \text{pa}(j'): v_i \in I_i} f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}) - v_{j'}\right)^2. \end{aligned}$$

Since all parent nodes $i \in \text{pa}(j')$ of j' satisfy $v_i \in I_i$, we get that $f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}) \ge \min_{\forall i \in \text{pa}(j'): v_i \in I_i} f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'})$, and as the latter minus $v_{j'}$ is non-negative, we get

$$\begin{aligned} &\le (f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}) - v_{j'})^2 \\ &= (v_{j'} - f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}))^2. \end{aligned}$$

Similarly, if $v_{j'}$ is greater than the upper endpoint, then we get

$$\begin{aligned} v_{j'} &> \max_{\forall i \in \text{pa}(j): v_i \in I_i} f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) + \sqrt{h_0} \\ 0 \le \sqrt{h_0} &< v_{j'} - \max_{\forall i \in \text{pa}(j): v_i \in I_i} f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j) \\ h_0 &< \left(v_{j'} - \max_{\forall i \in \text{pa}(j): v_i \in I_i} f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)\right)^2 \\ &\le (v_{j'} - f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}))^2. \end{aligned}$$

In either case, we get that $h_0 < (v_{j'} - f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}))^2$. Since h is a sum of squares, one of them being $(v_{j'} - f_{j'}(\boldsymbol{v}_{\text{pa}(j')}; \boldsymbol{\theta}_{j'}))^2$, this implies that $h_0 < h(\boldsymbol{v}_{\text{hid}})$ if there is a node j' such that $\boldsymbol{v}_{j'} \notin I_{j'}$. Hence if we let $U = \bigtimes_{j \in V_{\text{hid}}} I_j$, then we get $\boldsymbol{v}_{\text{hid}} \notin U \implies h(\boldsymbol{v}_{\text{hid}}) > h_0$, as desired.

Since all the intervals I_j are closed and finite, U is a compact subset of $\mathbb{R}^{|V_{\text{hid}}|}$. Thus h attains a minimum value on the domain U. And since $h_0 < h(\boldsymbol{v}_{\text{hid}})$ for all $\boldsymbol{v}_{\text{hid}} \notin U$, the attained minimum on U is the minimum of h on all of $\mathbb{R}^{|V_{\text{hid}}|}$. □

For the corresponding theorem for general loss functions in PC, we will need the following definition:

Definition 1 (Proper node loss function)**.** *A function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a **proper node loss function** if it satisfies the following properties:*

(i) $\ell(a, b)$ is differentiable everywhere

(ii) ℓ has a global lower bound

(iii) If $a' \leq a \leq b \leq b'$ or $a' \geq a \geq b \geq b'$, then $\ell(a, b) \leq \ell(a', b')$

(iv) $\lim_{b\to-\infty} \ell(a, b) = \lim_{b\to\infty} \ell(a, b) = \infty$ for all real numbers a

(v) $\lim_{a\to-\infty} \ell(a, b) = \lim_{a\to\infty} \ell(a, b) = \infty$ for all real numbers b

These properties imply an additional property:

Lemma 1. *Let ℓ be a proper node loss function. Then $\ell(a, a)$ has the same function value for all $a \in \mathbb{R}$.*

Proof. We claim that each (a, a), where $a \in \mathbb{R}$, must be a critical point of the function ℓ, i.e. its gradient must be $\mathbf{0}$. Assume, to be contradicted, that the gradient $\frac{\partial \ell}{\partial(a,b)}$ is not $\mathbf{0}$ at some $\boldsymbol{a'} = (a', a')$ where $a' \in \mathbb{R}$. We then consider the directional derivatives at $\boldsymbol{a'}$ in the directions $H = \{(1, 0), (-1, 0), (0, 1), (0, -1)\}$. At least one of the directional derivatives must be strictly negative — we denote this direction by $\boldsymbol{h} \in H$. Then there must exist an $\alpha > 0$ such that $\ell(\boldsymbol{a'}) > \ell(\boldsymbol{a'} + \alpha\boldsymbol{h})$. We also note that either

$$a' + \alpha\boldsymbol{h}_1 \leq a' \leq a' + \alpha\boldsymbol{h}_2 \quad \text{or} \quad a' + \alpha\boldsymbol{h}_1 \geq a' \geq a' + \alpha\boldsymbol{h}_2$$

must hold. Thus by property (iii) of the definition of a proper node loss function, the inequality $\ell(\boldsymbol{a'}) \leq \ell(\boldsymbol{a'} + \alpha\boldsymbol{h})$. That is a contradition, so the gradient $\frac{\partial \ell}{\partial(a,b)}(a', a')$ must be $\mathbf{0}$ for all $a' \in \mathbb{R}$.

We now assume, again to be contradicted, that there exist two points (a', a') and (a'', a''), with $a' < a''$, with different values of ℓ. Then by the mean value theorem, there must exist some $\tilde{a} \in (a', a'')$ such that the directional derivative of ℓ at $(\tilde{a}, \tilde{a})$ in the direction $(1, 1)$ is equal to $\frac{\ell(a'',a'')-\ell(a',a')}{a''-a'} \neq 0$. However, the gradient must be $\mathbf{0}$ for all $\tilde{a} \in \mathbb{R}$, which is a contradiction. Hence we conclude that $\ell(a, a)$ has the same value for all $a \in \mathbb{R}$. □

The squared node loss function $\ell^{\mathrm{SQ}}(a, b) = \frac{1}{2}(a-b)^2$ satisfies the definition of a proper node loss function: It is differentiable everywhere and has the lower bound of 0. If $a' \leq a \leq b \leq b'$ or $a' \geq a \geq b \geq b'$, then $|a - b| \leq |a' - b'|$, and so $\ell^{\mathrm{SQ}}(a, b) \leq \ell^{\mathrm{SQ}}(a', b')$. Finally, it is easy to see that taking the limit of either argument to $\pm\infty$ of ℓ^{SQ} gives ∞. One can show that $(a - b)^2(a^2 + b^2)$ and $(a - b)^4$ are also examples of proper node loss functions

On the other hand, neither the cross entropy node loss $\ell^{\mathrm{CE}}(a, b) = -a \log b$ nor the likelihood node loss $\ell^{\mathrm{LH}}(a, b) = -\log b$ is a proper node loss function. For both functions, taking the limit as $a \to \pm\infty$ does not both give ∞. A more fundamental problem, however, is that neither function is defined for all $b \in \mathbb{R}$ — they are only defined for positive b.

We now state and prove the generalized theorem.

Theorem 3 (PC inference phase minimum existence theorem)**.** *Given any computational graph G with parameters $\boldsymbol{\theta}_G$ and learning from any datapoint $(\boldsymbol{x}, \boldsymbol{y})$ using PC, the minimization problem of the datapoint loss function $\mathcal{L}_G^{PC}$ w.r.t. the hidden node values $\boldsymbol{v}_{hid}$ (corresponding to the inference phase) is guaranteed to have a minimum if its node loss functions satisfy the following properties:*

(i) ℓ_j is continuous and has a global lower bound for all output nodes $j \in V_{\boldsymbol{y}}$

(ii) ℓ_j is a proper node loss function for all hidden nodes $j \in V_{hid}$.

The proof follows the same strategy as for the squared node loss case: to show that a minimum of the function exists, we construct a compact set U such that input values outside the set cannot minimize the function, and then since U is compact and the function is continuous, the function must have a minimum on U, which must also be a minimum on the whole input space.

Proof. We denote the datapoint loss function of PC at the given datapoint $(\boldsymbol{x}, \boldsymbol{y})$ and graph parameters $\boldsymbol{\theta}_G$ by h:

$$h(\boldsymbol{v}_{\text{hid}}) := \mathcal{L}_G^{\text{PC}}(\boldsymbol{v}_{\text{hid}}, \boldsymbol{\theta}_G; D_i) = \left[\sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \ell_j(v_j, f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} .$$

We assume that all loss functions ℓ_j for $j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}$ are lower-bounded by 0. We can assume this, as one can consider instead new loss functions

$$\hat{\ell}_j(a, b) := \ell_j(a, b) - c_j,$$

where $c_j \in \mathbb{R}$ is a lower bound for ℓ_j, and the new function to be minimized,

$$\hat{h}(\boldsymbol{v}_{\text{hid}}) := \left[\sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} \hat{\ell}_j(v_j, f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} = h(\boldsymbol{v}_{\text{hid}}) - \sum_{j \in V_{\text{hid}} \cup V_{\boldsymbol{y}}} c_j.$$

These new node loss functions are then lower-bounded by 0. If the new function $\hat{h}$ has a minimum, then so does the original function h, as the two functions only differ by a constant.

With this assumption, let $h_0 \in \mathbb{R}$ be a function value attainable by h. We then construct a compact interval $I_j \subset \mathbb{R}$ for each non-output node $j \in V_{\boldsymbol{x}} \cup V_{\text{hid}}$. We would like these intervals to have the property that for all $j \in V_{\text{hid}}$, if $v_j \notin I_j$ and $v_i \in I_i$ for all parent nodes $i \in \text{pa}(j)$, then the function value $h(\boldsymbol{v}_{\text{hid}})$ would exceed h_0.

We construct the intervals sequentially in a topological order of the computational graph. If $j \in V_{\boldsymbol{x}}$, we define $I_j = \{(\boldsymbol{x}_i)_j\}$, the component of the input datapoint $\boldsymbol{x}_i$ corresponding to input node j. The case $j \in V_{\text{hid}}$ is more involved. Assume that all parent node values v_i of node j are in their respective compact intervals I_i (which have already been defined since they are defined in a topological order). Equivalently, $\boldsymbol{v}_{\text{pa}(j)}$ lies in a connected and compact set $I_{\text{pa}(j)} := \bigtimes_{i \in \text{pa}(j)} I_i$. Since $I_{\text{pa}(j)}$ is compact, the function $f_j(\boldsymbol{v}_{\text{pa}(j)}; \boldsymbol{\theta}_j)$ attains a minimum and a maximum on $I_{\text{pa}(j)}$, and we denote values that attain them by $\boldsymbol{v}_{\text{pa}(j)}^{\text{min}}$ and $\boldsymbol{v}_{\text{pa}(j)}^{\text{max}}$ respectively. Since $I_{\text{pa}(j)}$ is connected, we can move along a continuous path in the set from $\boldsymbol{v}_{\text{pa}(j)}^{\text{min}}$ to $\boldsymbol{v}_{\text{pa}(j)}^{\text{max}}$, and since f_j is continuous, the function value along this path attains all values between its minimum and maximum on $I_{\text{pa}(j)}$. This

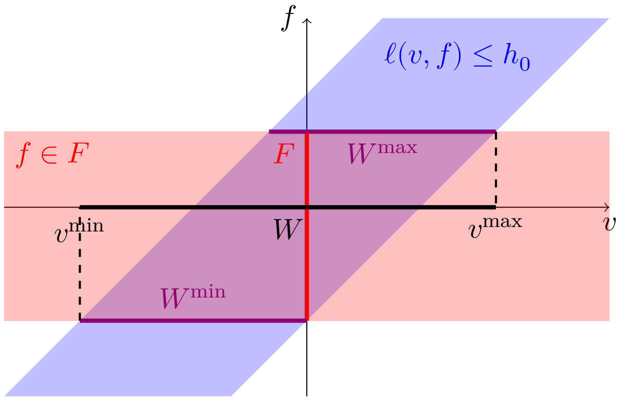


FIGURE 1: Illustration of F, W, $W^{\min}$, $W^{\max}$, $v^{\min}$, and $v^{\max}$ used in this proof.

implies that the set of all values of f_j on $I_{\mathrm{pa}(j)}$ is a compact interval on $\mathbb{R}$ — we denote this set by

$$F_j := \left\{ f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j) \;\middle|\; \boldsymbol{v}_{\mathrm{pa}(j)} \in I_{\mathrm{pa}(j)} \right\}.$$

We claim that if we define the set

$$W_j := \left\{ v_j \in \mathbb{R} \;\middle|\; \exists f_j \in F_j : \ell_j(v_j, f_j) \leq h_0 \right\}$$

and construct I_j to be a compact interval that is a superset of W_j, then it would have our desired property. We show this now. Any $\boldsymbol{v}_{\mathrm{hid}}$ with $v_j \notin I_j$ and $v_i \in I_i$, $\forall i \in \mathrm{pa}(j)$ for some $j \in V_{\mathrm{hid}}$ would imply $v_j \notin W_j$, meaning that $\forall f_j \in F_j : \ell_j(v_j, f_j) > h_0$. Thus we get that

$$\begin{aligned} h(\boldsymbol{v}_{\mathrm{hid}}) &= \left[\sum_{j' \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_{j'}(v_{j'}, f_{j'}(\boldsymbol{v}_{\mathrm{pa}(j')}; \boldsymbol{\theta}_{j'})) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} \\ &\geq \left[\ell_j(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x} \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}}} && \text{(since all } \ell_{j'} \geq 0\text{)} \\ &> h_0, && \text{(since } f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j) \in F_j\text{)} \end{aligned}$$

which is our desired property of I_j. Hence it remains to show that the sets W_j are always bounded.

We proceed by showing that any set W satisfying

$$W = \{ v \in \mathbb{R} \mid \exists f \in F : \ell(v, f) \leq h_0 \}$$

where $F \subset \mathbb{R}$ is a compact set and ℓ is a proper node loss function is bounded from below. This is illustrated in Fig. 1. We consider the sets

$$\begin{aligned} W^{\min} &:= \{ v \in \mathbb{R} \mid \ell(v, \min F) \leq h_0 \}, \\ W^{\max} &:= \{ v \in \mathbb{R} \mid \ell(v, \max F) \leq h_0 \}. \end{aligned}$$

Since ℓ is a proper loss function, the limits of $\ell(v, f)$ as $v \to \pm\infty$ are both ∞ for each fixed value of f. This implies that $W^{\min}$ and $W^{\max}$ are bounded. We can then define

$$v^{\min} := \begin{cases} \inf W^{\min} & \text{if } W^{\min} \text{ is non-empty} \\ \min F & \text{if } W^{\min} \text{ is empty} \end{cases},$$

$$v^{\max} := \begin{cases} \sup W^{\max} & \text{if } W^{\max} \text{ is non-empty} \\ \max F & \text{if } W^{\max} \text{ is empty} \end{cases},$$

which (as the notation suggests) we claim to be a lower bound and upper bound of W respectively.

If $W^{\min}$ is non-empty, i.e. $\exists w \in W^{\min}$, then since $w \leq \min F$ or $w \geq \min F$, we have that (since ℓ is a proper node loss function) $\ell(\min F, \min F) \leq \ell(w, \min F) \leq h_0$, and so $\min F \in W^{\min}$, and therefore $\inf W^{\min} \leq \min F$, must hold. This implies that $v^{\min} \leq \min F$.

Similarly, since $v = \max F$ minimizes the function $v \mapsto \ell(v, \max F)$, if $W^{\max}$ is non-empty, then $\max F \in W^{\max}$, and this implies that $v^{\max} \geq \max F$.

Additionally, we claim that $v < v^{\min}$ implies $\ell(v, \min F) > h_0$. If $W^{\min}$ is empty, then $\ell(v, \min F) > h_0$ for all $v \in \mathbb{R}$. Otherwise, if $W^{\min}$ is non-empty, then $v^{\min} = \inf W^{\min}$, so $v < v^{\min}$ would imply that $v \notin W^{\min}$, i.e. $v \notin \{v \in \mathbb{R} \mid \ell(v, \min F) \leq h_0\}$. Hence we conclude that $\ell(v, \min F) > h_0$ if $v < v^{\min}$.

Similarly, we show that if $v > v^{\max}$, then $\ell(v, \max F) > h_0$. If $W^{\max}$ is empty, then $\ell(v, \max F) > h_0$ for all $v \in \mathbb{R}$. Otherwise, if $W^{\max}$ is non-empty, then $v^{\max} = \sup W^{\max}$, so $v > v^{\max}$ would imply that $v \notin W^{\max}$, i.e. $v \notin \{v \in \mathbb{R} \mid \ell(v, \max F) \leq h_0\}$, and so we get that $\ell(v, \max F) > h_0$.

We prove that the set $W = \{v \in \mathbb{R} \mid \exists f \in F : \ell(v, f) \leq h_0\}$ is lower-bounded by $v^{\min}$ and upper-bounded by $v^{\max}$. For the lower bound, we consider the value of $\ell(v, f)$ for $v < v^{\min}$ and $f \in F$. Since ℓ is a proper loss function and $v < v^{\min} \leq \min F \leq f$, we have that

$$\ell(v, \min F) \leq \ell(v, f).$$

Since $v < v^{\min}$, we know from two paragraphs earlier that $\ell(v, \min F) > h_0$, so we furthermore have

$$\ell(v, \min F) > h_0.$$

We thus conclude that $\ell(v, f) > h_0$, if $v < v^{\min}$.

Similarly, for $v > v^{\max}$, since it holds that $v > v^{\max} \geq \max F \geq f$, we have that

$$\ell(v, \max F) \leq \ell(v, f).$$

Since $v > v^{\max}$, we know from two paragraphs earlier that $\ell(v, \max F) > h_0$, so we furthermore have

$$\ell(v, \max F) > h_0.$$

Hence in this case we also get that $\ell(v, f) > h_0$, if $v > v^{\max}$.

Putting the two cases together, this means that $v < v^{\min}$ or $v > v^{\max}$ implies $v \notin W$, thus proving that $v^{\min}$ is a lower bound and $v^{\max}$ is an upper bound of W. Thus W is a bounded set, and $I := [v^{\min}, v^{\max}]$ is a compact superset of W.

For this result, we have only assumed that F is a compact set and ℓ is a proper loss function. In particular, since F_j is compact and ℓ_j is a proper node loss function, we get that W_j is bounded, and we can construct a compact interval I_j that is a superset of W_j, and as we have shown, all the I_j then have the desired properties.

To summarize, we have been able to construct, for each non-output node $j \in V_{\boldsymbol{x}} \cup V_{\text{hid}}$, a compact interval I_j with the property that if $v_i \in I_i$ for all parent nodes $i \in \text{pa}(j)$ and $v_j \notin I_j$, then $h(\boldsymbol{v}_{\text{hid}}) > h_0$. We now claim that

$$U := \bigtimes_{j \in V_{\text{hid}}} I_j$$

has the property that $\boldsymbol{v}_{\text{hid}} \notin U$ implies $h(\boldsymbol{v}_{\text{hid}}) > h_0$. Let $\boldsymbol{v}_{\text{hid}} \notin U$. Note that $v_j = (\boldsymbol{x})_j \in I_j$ for all input nodes $j \in V_{\boldsymbol{x}}$, since in predictive coding with general loss, the input node values are always set to the input datapoint values. Then some hidden node value is not in its corresponding interval; let node $j' \in V_{\text{hid}}$ denote the first node in the topological ordering such that $v_{j'} \notin I_{j'}$. Then $v_i \in I_i$ for all of its parent nodes $i \in \text{pa}(j')$. Since all intervals I_j have the above property, we can conclude that $h(\boldsymbol{v}_{\text{hid}}) > h_0$, as required.

Finally, U is compact, and so the function h has a minimum value on U. And since $h_0 < h(\boldsymbol{v}_{\text{hid}})$ for all $\boldsymbol{v}_{\text{hid}} \notin U$, this minimum on U is the minimum of h on all of $\mathbb{R}^{|V_{\text{hid}}|}$. $\square$

5.2 Approximate dataset loss functions

We investigate the similarity of PC and BP by comparing the topology of their losses on the entire dataset.

While learning from a dataset with stochastic gradient descent, every datapoint is iterated through an equal number of times, so effectively, BP searches for graph parameters $\boldsymbol{\theta}_G \in \Theta_G$ that approximately minimize the *average* of its datapoint loss functions,

$$\mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D) := \frac{1}{n} \sum_{i \in [n]} \mathcal{L}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; (\boldsymbol{x}_i, \boldsymbol{y}_i)) \tag{33}$$

$$= \frac{1}{n} \sum_{i \in [n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}_G)). \tag{34}$$

This is also the loss function of the corresponding batch gradient descent algorithm. We call this the **dataset loss function** of BP. Similarly, we can define a similar function for PC. Recall that for each datapoint, we find the values v_j for $j \in V_{\mathrm{hid}}$ that minimize the datapoint loss function (the inference phase), then we adjust the parameters $\boldsymbol{\theta}_G$ via a gradient descent step towards minimizing the same datapoint loss function (the learning phase). Notice that we are only learning after the the inference phase is done — the datapoint loss function $\mathcal{L}_G^{\mathrm{PC}}$ has been minimized w.r.t. the hidden values $\boldsymbol{v}_{\mathrm{hid}}$. We thus define the dataset loss function as the average of the datapoint loss functions at this minimum:

$$\mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G; D) := \frac{1}{n} \sum_{i \in [n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \mathcal{L}_G^{\mathrm{PC}}(\boldsymbol{v}_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}_i, \boldsymbol{y}_i)) \tag{35}$$

$$= \frac{1}{n} \sum_{i \in [n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}}. \tag{36}$$

We write "min" instead of "inf" because we know that a minimum exists by Theorem 3 as long as the node loss functions satisfy the conditions in the theorem (and Theorem 2 for the special case of PC-SQ-e).

Comparing the dataset loss functions of BP and PC, we can see that they are not identical. However, they are approximately equal; we claim that they have the same set of minimizers if there exist parameters such that the computational graph fits the data perfectly. Such parameters are called *interpolating*:

Definition 2 (Interpolating parameters)**.** *Parameters $\boldsymbol{\theta}_G^*$* ***interpolate*** *a computational graph G and a paired dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [n]}$ if the predicted and expected outputs are equal for each datapoint in the dataset, i.e. $\boldsymbol{y}_i = \boldsymbol{f}_G(\boldsymbol{x}_i; \boldsymbol{\theta}_G^*)$ for all $i = 1, \dots, n$.*

Now the theorem and proof.

Theorem 4 (General interpolating minimizer theorem)**.** *Let G be a computational graph and $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [n]}$ a paired dataset. Consider BP for training on G and D with node loss functions ℓ_j^{BP} for $j \in V_{\boldsymbol{y}}$, and PC with node loss functions*

$$\ell_j^{PC}(a, b) = \begin{cases} \ell_j^{BP}(a, b) & j \in V_{\boldsymbol{y}} \\ \ell_j(a, b) & j \in V_{hid} \end{cases},$$

where

(i) for all output nodes $j \in V_{\boldsymbol{y}}$: ℓ_j^{BP} is continuous and $\ell_j^{BP}(a,a)$ for all $a \in \mathbb{R}$ are its only global minima, and

(ii) for all hidden nodes $j \in V_{hid}$: ℓ_j is a proper node loss function and $\ell_j(a,a)$ for all $a \in \mathbb{R}$ are its only global minima.

If interpolating parameters exist, then

$$\begin{aligned}\{\boldsymbol{\theta}_G^* \in \Theta_G \mid \boldsymbol{\theta}_G^* \textit{ interpolates } G \textit{ and } D\} &= \operatorname*{argmin}_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{BP}(\boldsymbol{\theta}_G; D) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{PC}(\boldsymbol{\theta}_G; D),\end{aligned}$$

where $\mathcal{F}_G^{BP}$ and $\mathcal{F}_G^{PC}$ are the dataset loss functions of BP and PC respectively for the computational graph G.

Proof. We prove this separately for BP and for PC. Note that the conditions for the node loss functions imply those in Theorem 3, so a minimum exists for the minimization found in the PC dataset loss function $\mathcal{F}_G^{\mathrm{PC}}$.

BP: The dataset loss function to minimize for BP is

$$\mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D) = \frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}_G)).$$

If interpolating parameters exist, then at any interpolating parameters $\boldsymbol{\theta}_G^*$, the function evaluates to

$$\begin{aligned}\mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G^*; D) &= \frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}_G^*)) \\ &= \frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, (\boldsymbol{y}_i)_j).\end{aligned}$$

Since we require that $\ell_j^{\mathrm{BP}}(a,a)$ be a global minimum for all $a \in \mathbb{R}$, we get

$$\begin{aligned}&\le \frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \min_{\boldsymbol{\theta}_G \in \Theta_G} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}_G)) \\ &\le \min_{\boldsymbol{\theta}_G \in \Theta_G} \frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}_G)) \\ &= \min_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D).\end{aligned}$$

At the same time, it must also be true that $\mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G^*; D) \ge \min_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D)$, so we conclude that equality is attained for all the above steps, and hence

$$\boldsymbol{\theta}_G^* \in \operatorname*{argmin}_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D).$$

Now the converse. Assuming that interpolating parameters exist, and some $\boldsymbol{\theta}'_G$ minimizes the dataset loss function, then

$$\begin{aligned}\frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}'_G)) &=: \mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}'_G; D) \\ &= \min_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D) \\ &= \frac{1}{n} \sum_{i\in[n]} \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{BP}}((\boldsymbol{y}_i)_j, (\boldsymbol{y}_i)_j),\end{aligned}$$

from the equality shown above. Since each $\ell_j^{\mathrm{BP}}(a,b)$ has for all a the unique minimizer w.r.t. b that is $b=a$, we conclude that for all datapoints $i \in [n]$ and $j \in V_{\boldsymbol{y}}$,

$$(\boldsymbol{y}_i)_j = z_j(\boldsymbol{x}_i; \boldsymbol{\theta}'_G),$$

i.e. $\boldsymbol{\theta}'_G$ interpolate G and D. Combining the results, we get that if interpolating parameters exist for G and D, then

$$\{\boldsymbol{\theta}^*_G \in \Theta_G \mid \boldsymbol{\theta}^*_G \text{ interpolates } G \text{ and } D\} = \operatorname*{argmin}_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{BP}}(\boldsymbol{\theta}_G; D).$$

PC: The dataset loss function to minimize for PC is

$$\mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G; D) = \frac{1}{n}\sum_{i\in[n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}}.$$

If interpolating parameters $\boldsymbol{\theta}^*_G$ exist, then at the interpolating parameters $\boldsymbol{\theta}^*_G$,

$$\begin{aligned}
&\mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}^*_G; D) \\
&= \frac{1}{n}\sum_{i\in[n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}^*_j))\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}} \\
&= \frac{1}{n}\sum_{i\in[n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \left[\sum_{j \in V_{\mathrm{hid}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}^*_j)) + \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}^*_j))\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}}.
\end{aligned}$$

If we in particular choose $\boldsymbol{v}_{\mathrm{hid}} = \boldsymbol{z}_{\mathrm{hid}}$, we get

$$\begin{aligned}
&\le \frac{1}{n}\sum_{i\in[n]} \left(\sum_{j \in V_{\mathrm{hid}}} \ell_j^{\mathrm{PC}}(z_j(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G), f_j(\boldsymbol{z}_{\mathrm{pa}(j)}(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G); \boldsymbol{\theta}^*_j)) + \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}((\boldsymbol{y}_i)_j, f_j(\boldsymbol{z}_{\mathrm{pa}(j)}(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G); \boldsymbol{\theta}^*_j))\right) \\
&= \frac{1}{n}\sum_{i\in[n]} \left(\sum_{j \in V_{\mathrm{hid}}} \ell_j^{\mathrm{PC}}(z_j(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G), z_j(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G)) + \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}((\boldsymbol{y}_i)_j, z_j(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G))\right) \\
&= \frac{1}{n}\sum_{i\in[n]} \left(\sum_{j \in V_{\mathrm{hid}}} \ell_j^{\mathrm{PC}}(z_j(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G), z_j(\boldsymbol{x}_i; \boldsymbol{\theta}^*_G)) + \sum_{j \in V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}((\boldsymbol{y}_i)_j, (\boldsymbol{y}_i)_j)\right).
\end{aligned}$$

Since we assumed that $\ell_j^{\mathrm{PC}}(a,a)$ is a global minimum for all $a \in \mathbb{R}$,

$$\begin{aligned}
&= \frac{1}{n}\sum_{i\in[n]} \sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \min_{a,b\in\mathbb{R}} \ell_j^{\mathrm{PC}}(a,b) \\
&\le \frac{1}{n}\sum_{i\in[n]} \min_{a,b\in\mathbb{R}} \sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(a,b) \\
&\le \frac{1}{n}\sum_{i\in[n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}, \boldsymbol{\theta}_G \in \Theta_G} \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}} \\
&\le \min_{\boldsymbol{\theta}_G \in \Theta_G} \frac{1}{n}\sum_{i\in[n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j))\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}} \\
&= \min_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G; D).
\end{aligned}$$

Since it must also hold that $\mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G^*; D) \geq \min_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G; D)$, this implies that equality is attained at every step, and so we conclude that

$$\boldsymbol{\theta}_G^* \in \operatorname*{argmin}_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G; D).$$

Conversely, assuming that interpolating parameters $\boldsymbol{\theta}_G^*$ exist, we let $\boldsymbol{\theta}'_G$ be a minimizer of $\mathcal{F}_G^{\mathrm{PC}}(\cdot; D)$. Since equality was attained above, we get that

$$\begin{aligned}
&\frac{1}{n} \sum_{i \in [n]} \min_{\boldsymbol{v}_{\mathrm{hid}} \in \mathbb{R}^{|V_{\mathrm{hid}}|}} \left[\sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}'_j)) \right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}} = \boldsymbol{x}_i \\ \boldsymbol{v}_{\boldsymbol{y}} = \boldsymbol{y}_i}} \\
&=: \mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}'_G; D) \\
&= \mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}^*_G; D) \\
&= \frac{1}{n} \sum_{i \in [n]} \sum_{j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \min_{a, b \in \mathbb{R}} \ell_j^{\mathrm{PC}}(a, b).
\end{aligned}$$

Since $\mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}'_G; D)$ is a sum of the node losses, and is equal to the sum of the minima of the same node loss functions, this implies that each individual node loss $\ell_j^{\mathrm{PC}}(v_j, f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}'_j))$ must attain its respective minimum value. By our requirement that the node loss functions ℓ_j must have $\ell_j(a, a)$ for all $a \in \mathbb{R}$ as the only global minima, this means that the two arguments must be equal,

$$v_j = f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}'_j),$$

for all $i \in [n]$ and $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$.

Since $\boldsymbol{v}_{\boldsymbol{x}}$ is assigned to $\boldsymbol{x}_i$, the values for $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$ must be the feedforward values, $v_j = z_j(\boldsymbol{x}_i; \boldsymbol{\theta}'_G)$, by definition. Then since $\boldsymbol{v}_{\boldsymbol{y}}$ is assigned to $\boldsymbol{y}_i$, we conclude that $z_j(\boldsymbol{x}_i; \boldsymbol{\theta}'_G) = v_j = (\boldsymbol{y}_i)_j$ for all $j \in V_{\boldsymbol{y}}$ and $i \in [n]$, i.e. $\boldsymbol{\theta}'_G$ interpolate G and D. Therefore we have proven that

$$\{\boldsymbol{\theta}_G^* \in \Theta_G \mid \boldsymbol{\theta}_G^* \text{ interpolates } G \text{ and } D\} = \operatorname*{argmin}_{\boldsymbol{\theta}_G \in \Theta_G} \mathcal{F}_G^{\mathrm{PC}}(\boldsymbol{\theta}_G; D).$$

Combining the results from BP and PC gives the theorem. $\square$

5.3 Approximate parameter update steps

We now show that the PC and BP algorithms produce similar — but not identical — update steps $\Delta\boldsymbol{\theta}_G$ for the parameters $\boldsymbol{\theta}_G$.

In order to characterize more precisely how an algorithm "approximates" BP, we introduce the following definition.

Definition 3 (BP-like algorithm)**.** *A supervised learning algorithm for computational graphs is* ***BP-like with loss*** ℓ^{BP} *if there exists a function g for node values such that for all computational graphs G, graph parameters $\boldsymbol{\theta}_G \in \Theta_G$, and datapoints $(\boldsymbol{x}, \boldsymbol{y})$, the corresponding parameter update is equal to the parameter update function of BP with loss ℓ^{BP}_j for $j \in V_{\boldsymbol{y}}$, namely for all $j \in V_{hid} \cup V_{\boldsymbol{y}}$*

$$\Delta\boldsymbol{\theta}_j = -c\lambda_j \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$$

where $c > 0$ is a constant and λ_j satisfies the recursive relation

$$\lambda_j = \begin{cases} \dfrac{\partial \ell^{BP}_j}{\partial f_j}((\boldsymbol{y})_j, f_j(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) & j \in V_{\boldsymbol{y}} \\ \displaystyle\sum_{k \in \mathrm{ch}(j)} \lambda_k \frac{\partial f_k}{\partial a_j}(\boldsymbol{a}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) & j \in V_{hid} \end{cases}$$

and the node values are given by $a_j = g(G, j, \boldsymbol{\theta}_G, (\boldsymbol{x}, \boldsymbol{y}))$ for all $j \in V_{\boldsymbol{x}} \cup V_{hid}$.

The recursive relation in the definition is exactly the one found in the formula for the parameter update step in BP. The idea behind this definition is that algorithms with the same update formulas as BP, with perhaps different values used as the node values, are approximately the same as BP.

We also need to define a new class of node loss functions:

Definition 4 (Difference node loss function)**.** *A function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a* ***difference node loss function*** *if it has the representation $\ell(a, b) = \tilde{\ell}(a - b)$ for some function $\tilde{\ell} : \mathbb{R} \to \mathbb{R}$ and for all real numbers a, b. The function $\tilde{\ell}$ is called its* ***difference function****.*

Definition 5 (Proper difference node loss function)**.** *A* ***proper difference node loss function*** *is a function that is both a proper node loss function (Def. 1) and a difference node loss function (Def. 4).*

Lemma 2 (Properties of proper difference node loss functions)**.** *If ℓ is a proper difference node loss function, then its difference function $\tilde{\ell}$ satisfies the following properties:*

(i) $\tilde{\ell}$ *is differentiable everywhere,*

(ii) If $c' \le c \le 0$ or $0 \le c \le c'$, then $\tilde{\ell}(c) \le \tilde{\ell}(c')$,

(iii) $\lim_{c \to -\infty} \tilde{\ell}(c) = \lim_{c \to \infty} \tilde{\ell}(c) = \infty$

Proof. Recall that proper node loss functions are defined in Def. 1.

(i) is trivial.

For (ii), if $c' \le c \le 0$, then we have $c' \le c \le 0 \le 0$; if $0 \le c \le c'$, then we have $c' \ge c \ge 0 \ge 0$. These fit the first and second conditions respectively of (iii) of a proper node loss function, and hence $\tilde{\ell}(c) = \ell(c, 0) \le \ell(c', 0) = \tilde{\ell}(c')$.

For (iii), since proper node loss functions have the property $\lim_{a \to -\infty} \ell(a, b) = \lim_{a \to \infty} \ell(a, b) = \infty$ for all real numbers b, if we consider in particular $b = 0$, then we get $\lim_{c \to -\infty} \tilde{\ell}(c) = \lim_{a \to -\infty} \ell(a, 0) = \infty$ and $\lim_{c \to \infty} \tilde{\ell}(c) = \lim_{a \to \infty} \ell(a, 0) = \infty$. □

The squared node loss function $\ell^{\mathrm{SQ}}(a,b) = \frac{1}{2}(a-b)^2$ is not only a proper node loss function, but also a difference node loss function: its difference function is $\widetilde{\ell}^{\mathrm{SQ}}(c) = \frac{1}{2}c^2$. Similarly, the proper node loss function $(a-b)^4$ is also a difference node loss function. However, the function $\ell(a,b) = (a-b)^2(a^2+b^2)$, which is a proper node loss function, is not a difference node loss function, as $\ell(1,0) = 1$ but $\ell(2,1) = 5$.

On the other hand, the function $(a-b)^3$ is a difference node loss function, but it has no lower bound, so is not a proper node loss function. Similarly, neither the cross entropy node loss $\ell^{\mathrm{CE}}(a,b) = -a \log b$ nor the likelihood node loss $\ell^{\mathrm{LH}}(a,b) = -\log b$ is a difference node loss function.

We now have all the ingredients needed to prove that PC is BP-like.

Theorem 5. *Consider BP with a continuous and finitely lower-bounded loss function ℓ_j^{BP} and PC with loss function*

$$\ell_j^{PC}(a,b) = \begin{cases} \ell_j^{BP}(a,b) & j \in V_{\boldsymbol{y}} \\ \ell_j(a,b) & j \in V_{hid} \end{cases}$$

where ℓ_j for $j \in V_{hid}$ are proper difference node loss functions. Then the PC algorithm is BP-like with loss ℓ^{BP}.

Proof. Given arbitrary node loss functions for BP, we would like to find node loss functions for PC such that it is BP-like.

We investigate the gradient descent step for the parameter $\boldsymbol{\theta}_j$ for datapoint $(\boldsymbol{x}, \boldsymbol{y})$ for PC with arbitrary loss functions ℓ_j^{PC} for $j \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}$. We assume that the node values have converged to an equilibrium point of the datapoint loss function $\mathcal{L}_G^{\mathrm{PC}}$ by the end of the inference phase. We recall that the post-inference node values are denoted with a *, and introduce the shorthand $f_j^* := f_j(\boldsymbol{v}^*_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)$. For all hidden nodes $j \in V_{\mathrm{hid}}$, we get

$$\begin{aligned} 0 = \frac{\partial \mathcal{L}_G^{\mathrm{PC}}}{\partial v_j}(\boldsymbol{v}^*_{\mathrm{hid}}, \boldsymbol{\theta}_G; (\boldsymbol{x}, \boldsymbol{y})) &= \frac{\partial}{\partial v_j}\left(\sum_{j' \in V_{\mathrm{hid}} \cup V_{\boldsymbol{y}}} \ell_{j'}^{\mathrm{PC}}(v^*_{j'}, f^*_{j'})\right) \\ &= \frac{\partial}{\partial v_j}\left(\ell_j^{\mathrm{PC}}(v_j^*, f_j^*) + \sum_{k \in \mathrm{ch}(j)} \ell_k^{\mathrm{PC}}(v_k^*, f_k(\boldsymbol{v}^*_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k))\right) \\ &= \frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j^*, f_j^*) + \sum_{k \in \mathrm{ch}(j)} \frac{\partial \ell_k^{\mathrm{PC}}}{\partial v_j}(v_k^*, f_k(\boldsymbol{v}^*_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \\ \implies \frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j^*, f_j^*) &= \sum_{k \in \mathrm{ch}(j)} -\frac{\partial \ell_k^{\mathrm{PC}}}{\partial v_j}(v_k^*, f_k(\boldsymbol{v}^*_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k)) \\ &= \sum_{k \in \mathrm{ch}(j)} -\frac{\partial \ell_k^{\mathrm{PC}}}{\partial f_k}(v_k^*, f_k^*)\frac{\partial f_k}{\partial v_j}(\boldsymbol{v}^*_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k). \end{aligned}$$

This means that

$$\frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j^*, f_j^*) = \begin{cases} \frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}((\boldsymbol{y})_j, f_j^*) & j \in V_{\boldsymbol{y}} \\ \sum_{k \in \mathrm{ch}(j)} -\frac{\partial \ell_k^{\mathrm{PC}}}{\partial f_k}(v_k^*, f_k^*)\frac{\partial f_k}{\partial v_j}(\boldsymbol{v}^*_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) & j \in V_{\mathrm{hid}} \end{cases}. \quad (37)$$

Additionally, the gradient step of the parameters $\boldsymbol{\theta}_j$ is, from (17),

$$\Delta\boldsymbol{\theta}_j^{\mathrm{PC}} = -\eta_{\boldsymbol{\theta}} \left[\frac{\partial \ell_j^{\mathrm{PC}}}{\partial f_j}(v_j^*, f_j^*) \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{v}_{\mathrm{pa}(j)}^*; \boldsymbol{\theta}_j) \right]_{\substack{\boldsymbol{v_x}=\boldsymbol{x} \\ \boldsymbol{v_y}=\boldsymbol{y}}} . \tag{38}$$

We compare this with the BP-like parameter gradient descent step with learning rate $c > 0$ as found in Def. 3,

$$\Delta\boldsymbol{\theta}_j = -c\lambda_j \frac{\partial f_j}{\partial \boldsymbol{\theta}_j}(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j),$$

$$\lambda_j = \begin{cases} \dfrac{\partial \ell_j^{\mathrm{BP}}}{\partial f_j}((\boldsymbol{y})_j, f_j(\boldsymbol{a}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j)) & j \in V_{\boldsymbol{y}} \\ \displaystyle\sum_{k \in \mathrm{ch}(j)} \lambda_k \frac{\partial f_k}{\partial a_j}(\boldsymbol{a}_{\mathrm{pa}(k)}; \boldsymbol{\theta}_k) & j \in V_{\mathrm{hid}} \end{cases} .$$

From these equations, we see that if we have the node loss functions satisfy the conditions

$$\forall j \in V_{\boldsymbol{y}} : \quad \ell_j^{\mathrm{PC}} = \ell_j^{\mathrm{BP}} \tag{39}$$

and

$$\forall j \in V_{\mathrm{hid}} : \quad \frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j^*, f_j^*) = -\frac{\partial \ell_j^{\mathrm{PC}}}{\partial f_j}(v_j^*, f_j^*), \tag{40}$$

and let $\eta_{\boldsymbol{\theta}} = c$, $\frac{\partial \ell_j^{\mathrm{PC}}}{\partial f_j}(v_j^*, f_j^*) = \lambda_j$, and $v_j^* = a_j$, and if we replace all instances of the terms on the left with the equivalent terms on the right in (38) and (37), then we get exactly the BP-like equations.

The condition in (40) is satisfied when the node loss functions of the hidden nodes are difference node loss functions, i.e.

$$\forall j \in V_{\mathrm{hid}} : \quad \exists \tilde{\ell}_j : \mathbb{R} \to \mathbb{R} : \quad \ell_j^{\mathrm{PC}}(v, f) = \tilde{\ell}_j(v - f),$$

since

$$\frac{\partial \ell_j^{\mathrm{PC}}}{\partial f_j}(v_j^*, f_j^*) = \frac{\partial \tilde{\ell}_j}{\partial f_j}(v_j^* - f_j^*) \cdot -1 = -\frac{\partial \tilde{\ell}_j}{\partial f_j}(v_j^* - f_j^*)$$

and

$$-\frac{\partial \ell_j^{\mathrm{PC}}}{\partial v_j}(v_j^*, f_j^*) = -\frac{\partial \tilde{\ell}_j}{\partial f_j}(v_j^* - f_j^*) \cdot 1 = -\frac{\partial \tilde{\ell}_j}{\partial f_j}(v_j^* - f_j^*)$$

are equal.

Furthermore, since we have assumed that the node values $\boldsymbol{v}_{\mathrm{hid}}$ have converged, we want our choice of loss function to guarantee that a minimizer $\boldsymbol{v}^*$ of $\mathcal{L}^{\mathrm{PC}}$ exists. By Theorem 3, a value estimation phase minimum is guaranteed to exist if for $j \in V_{\mathrm{hid}}$, ℓ_j is proper, and for $j \in V_{\boldsymbol{y}}$, ℓ_j is continuous and lower-bounded. Hence PC is BP-like if the hidden node loss functions are proper difference node loss functions. $\square$

We now claim that BP can be seen as a limit of a sequence of PC algorithms with different node loss functions.

Theorem 6. *Consider BP with continuous and finitely lower-bounded loss functions ℓ_j^{BP}, $j \in V_{\boldsymbol{y}}$, and the class of PC algorithms with the loss function*

$$\ell_j^{PC}(a,b) = \begin{cases} \ell_j^{BP}(a,b) & j \in V_{\boldsymbol{y}} \\ w\ell_j(a,b) & j \in V_{hid} \end{cases} \tag{41}$$

where $w > 0$, and ℓ_j are proper difference loss functions with difference functions $\tilde{\ell}_j(c)$ having the unique minimizer $c = 0$. If right before seeing a new datapoint i they have the same graph parameters $\boldsymbol{\theta}_G$ and assuming that a minimum is attained in the inference phase, then at the end of processing datapoint $(\boldsymbol{x}, \boldsymbol{y})$, we have that as $w \to \infty$, for all $j \in V_{hid}$, $v_j^* \to z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ and for all $j \in V_{hid} \cup V_{\boldsymbol{y}}$, $\Delta\boldsymbol{\theta}_j^{PC} \to \Delta\boldsymbol{\theta}_j^{BP}$.

Proof. We simplify the statement by assuming that the unique minimum of the hidden loss functions is 0. This can be done because adding any constant to any loss function does not change the resulting PC and BP algorithms, and this is true because the only terms involving the loss function in the formulas for the value estimation phase as well as the parameter update phase is its partial derivative.

From Theorem 5, we know that this class of PC algorithms is BP-like with loss ℓ_j^{BP} for output node $j \in V_{\boldsymbol{y}}$. This, and the fact that all functions here are continuous, means that $v_j^* \to z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ implies $\Delta\boldsymbol{\theta}_j^{\mathrm{PC}} \to \Delta\boldsymbol{\theta}_j^{\mathrm{BP}}$.

First, trivially, $v_j^* = (\boldsymbol{x})_j = z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ for all input nodes $j \in V_{\boldsymbol{x}}$. Then, to prove that $v_j^* \to z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ for all hidden nodes $j \in V_{\mathrm{hid}}$, we extend from the proof of Theorem 3. In that proof, we showed that each minimum node value v_j^* for $j \in V_{\boldsymbol{x}} \cup V_{\mathrm{hid}}$ must be contained in a compact interval $I_j = [v_j^{\min}, v_j^{\max}]$ that we constructed, so we just need to show that $v_j^{\min}, v_j^{\max} \to z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ for all hidden nodes.

We show this sequentially with the nodes following a topological order, and hence when proving this for node j, we can assume that we have already proven it for its parent nodes $\mathrm{pa}(j)$. We first tackle the lower bound. Recall that $v_j^{\min}$ is defined as

$$v_j^{\min} := \begin{cases} \min W_j^{\min} & \text{if } W_j^{\min} \text{ is non-empty} \\ \min F_j & \text{if } W_j^{\min} \text{ is empty} \end{cases},$$

where

$$F_j := \left\{ f_j(\boldsymbol{v}_{\mathrm{pa}(j)}; \boldsymbol{\theta}_j) \;\middle|\; \boldsymbol{v}_{\mathrm{pa}(j)} \in I_{\mathrm{pa}(j)} \right\},$$

and

$$W_j^{\min} := \left\{ v_j \in \mathbb{R} \;\middle|\; w\ell_j(v_j, \min F_j) \leq h_0 \right\},$$

where $h_0 \geq 0$ is any value attainable by the datapoint loss loss function $\mathcal{L}_G^{\mathrm{PC}}$. For this

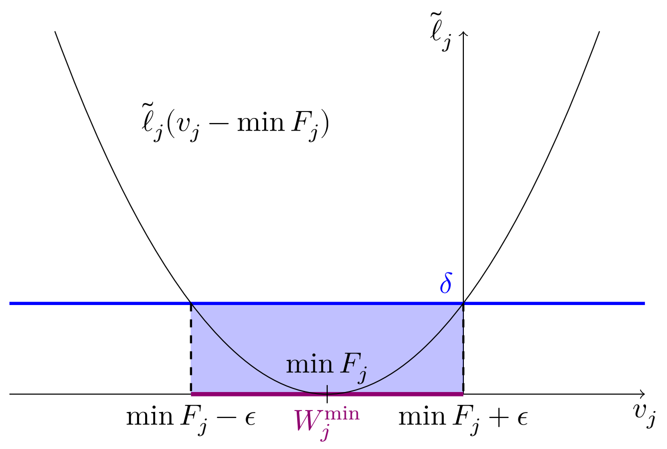


FIGURE 2: Illustration of the relationship between v_j, $\min F_j$, $\tilde{\ell}_j$, $W_j^{\min}$, δ, and ϵ used in this proof.

proof, we in particular choose

$$\begin{aligned}
h_0 &:= \mathcal{L}_G^{\text{PC}}(\boldsymbol{z}_{V_{\text{hid}}}(\boldsymbol{x};\boldsymbol{\theta}_G),\boldsymbol{\theta}_G;D)\\
&= \left[\sum_{j\in V_{\text{hid}}\cup V_{\boldsymbol{y}}} \ell_j^{\text{PC-SQ}}(v_j(\boldsymbol{x};\boldsymbol{\theta}_G), f_j(v_{\text{pa}(j)}(\boldsymbol{x};\boldsymbol{\theta}_G);\boldsymbol{\theta}_j))\right]_{\substack{\boldsymbol{v}_{\boldsymbol{x}}=\boldsymbol{x}\\ \boldsymbol{v}_{\boldsymbol{y}}=\boldsymbol{y}\\ \boldsymbol{v}_{\text{hid}}=\boldsymbol{z}_{V_{\text{hid}}}}}\\
&= \sum_{j\in V_{\text{hid}}} w\ell_j(z_j(\boldsymbol{x};\boldsymbol{\theta}_G), f_j(z_{\text{pa}(j)}(\boldsymbol{x};\boldsymbol{\theta}_G);\boldsymbol{\theta}_j)) + \sum_{j\in V_{\boldsymbol{y}}} \ell_j^{\text{BP}}((\boldsymbol{y})_j, f_j(z_{\text{pa}(j)}(\boldsymbol{x};\boldsymbol{\theta}_G);\boldsymbol{\theta}_j))\\
&= \sum_{j\in V_{\text{hid}}} w\ell_j(z_j(\boldsymbol{x};\boldsymbol{\theta}_G), z_j(\boldsymbol{x};\boldsymbol{\theta}_G)) + \sum_{j\in V_{\boldsymbol{y}}} \ell_j^{\text{BP}}((\boldsymbol{y})_j, z_j(\boldsymbol{x};\boldsymbol{\theta}_G))\\
&= \sum_{j\in V_{\boldsymbol{y}}} \ell_j^{\text{BP}}((\boldsymbol{y})_j, z_j(\boldsymbol{x};\boldsymbol{\theta}_G))\\
&\geq 0.
\end{aligned}$$

Notice that we have used that the hidden loss functions evaluate to 0 when the difference between the arguments is 0. Notice also that with this choice of h_0, it is independent of w.

Since we assume that we have proved $v_{j'}^{\min}, v_{j'}^{\max} \to z_{j'}(\boldsymbol{x};\boldsymbol{\theta}_G)$ for all parent nodes $j' \in \text{pa}(j)$, we get that $F_j \to \{f_j(\boldsymbol{z}_{\text{pa}(j)}(\boldsymbol{x};\boldsymbol{\theta}_G);\boldsymbol{\theta}_j)\} = \{z_j(\boldsymbol{x};\boldsymbol{\theta}_G)\}$ as $w \to \infty$. Hence $\min F_j \to z_j(\boldsymbol{x};\boldsymbol{\theta}_G)$ as $w \to \infty$ as desired.

For the first case, we then show that if $W_j^{\min}$ is non-empty then $\min W_j^{\min} \to \min F_j$, which combined with the above result, gives $\min W_j^{\min} \to z_j(\boldsymbol{x};\boldsymbol{\theta}_G)$. We assumed that $w > 0$ all hidden loss functions are proper difference loss functions, so we can rewrite $W_j^{\min}$ as

$$W_j^{\min} = \left\{v_j \in \mathbb{R} \;\middle|\; \tilde{\ell}_j(v_j - \min F_j) \leq \delta\right\},$$

where $\tilde{\ell}_j$ is the difference function of ℓ_j and $\delta := \frac{h_0}{w} \to 0^+$. This is illustrated in Fig. 2. As ℓ_j is a proper difference node loss function, by Lemma 2, we get that $\tilde{\ell}_j$ has a unique minimum at $\tilde{\ell}_j(0) = 0$ by assumption and is monotone decreasing on $(-\infty, 0]$ and monotone increasing on $[0, \infty)$. This implies that for all $\epsilon > 0$ there exists a $\delta > 0$ such that $\tilde{\ell}_j(v_j - \min F_j) \leq \delta$ implies $|v_j - \min F_j| < \epsilon$. An explicit example is namely

$$\delta = \frac{1}{2} \min \left\{ \tilde{\ell}_j(-\epsilon), \tilde{\ell}_j(\epsilon) \right\},$$

as $\tilde{\ell}_j(v_j - \min F_j) \leq \delta < \tilde{\ell}_j(-\epsilon)$ and $\tilde{\ell}_j(v_j - \min F_j) \leq \delta < \tilde{\ell}_j(\epsilon)$, together with the monotonicity properties, imply $-\epsilon < v_j - \min F_j < \epsilon$. Thus as $\delta \to 0^+$, the set $W_j^{\min}$ approaches the set $\left\{ v_j \in \mathbb{R} \mid \forall \delta > 0 : |v_j - \min F_j| \leq \delta \right\} = \{\min F_j\}$, as desired. We can thus conclude that indeed $v_j^{\min} \to z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ as $w \to \infty$.

By the same line of reasoning, we can show that the maximum $v_j^{\max}$ of the interval I_j also approaches $z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ as $w \to \infty$. Retracing the chain of implications, we can thus conclude that $\forall j \in V_{\boldsymbol{x}} \cup V_{\text{hid}} : v_j^* \to z_j(\boldsymbol{x}; \boldsymbol{\theta}_G)$ and $\Delta \boldsymbol{\theta}_j^{\text{PC}} \to \Delta \boldsymbol{\theta}_j^{\text{BP}}$ as $w \to \infty$, as desired. $\square$

6 Discussion and conclusion

In this paper, we have formulated a general predictive coding algorithm with arbitrary node loss functions, which we call the PC algorithm. We then showed that PC and backprop, with corresponding loss functions, have identical minimizers under certain conditions (Theorem 4). Furthermore, for certain classes of node loss functions, the parameter update steps of PC are approximately equal to those of backprop (Theorem 5), and this approaches equality in a certain limit (Theorem 6).

Through generalizing to arbitrary losses, our formulation allows predictive coding algorithms to be applied on more complex neural network architectures and a wider range of machine learning tasks. This further closes the gap between biologically realistic models of learning and learning *in silico*. Additionally, our results add mathematical rigor to the existing literature on the correspondence between predictive coding algorithms and backprop.

Future work would be to test and evaluate the performance of predictive coding with general losses used on complex deep learning architectures. Additionally, one can explore the possibility of extending our predicting coding algorithm framework and results to graphs that contain cycles, which may be useful for training architectures such as recurrent neural networks.

References

[1] N. Alonso and E. Neftci. Tightening the biological constraints on gradient-based predictive coding. In *International Conference on Neuromorphic Systems 2021*, ICONS 2021. ACM, July 2021. DOI: `10.1145/3477145.3477148`.

[2] K. Amunts, M. Axer, S. Banerjee, L. Bitsch, J. G. Bjaalie, P. Brauner, A. Brovelli, N. Calarco, M. Carrere, S. Caspers, C. J. Charvet, S. Cichon, R. Cools, I. Costantini, E. U. D'Angelo, G. De Bonis, G. Deco, J. DeFelipe, A. Destexhe, T. Dickscheid, M. Diesmann, E. Düzel, S. B. Eickhoff, G. Einevoll, D. Eke, A. K. Engel, A. C. Evans, K. Evers, N. Fedorchenko, S. J. Forkel, J. Fousek, A. D. Friederici, K. Friston, S. Furber, L. Geris, R. Goebel, O. Güntürkün, A. I. A. Hamid, C. Herold, C. C. Hilgetag, S. M. Hölter, Y. Ioannidis, V. Jirsa, S. Kashyap, B. S. Kasper, A. d. K. d'Exaerde, R. Kooijmans, I. Koren, J. H. Kotaleski, G. Kiar, W. Klijn, L. Klüver, A. C. Knoll, Z. Krsnik, J. Kämpfer, M. E. Larkum, M.-L. Linne, T. Lippert, J. M. Abdullah, P. D. Maio, N. Magielse, P. Maquet, A. L. A. Mascaro, D. Marinazzo, J. Mejias, A. Meyer-Lindenberg, M. Migliore, J. Michael, Y. Morel, F. O. Morin, L. Muckli, G. Nagels, L. Oden, N. Palomero-Gallagher, F. Panagiotaropoulos, P. S. Paolucci, C. Pennartz, L. M. Peeters, S. Petkoski, N. Petkov, L. S. Petro, M. A. Petrovici, G. Pezzulo, P. Roelfsema, L. Ris, P. Ritter, K. Rockland, S. Rotter, A. Rowald, S. Ruland, P. Ryvlin, A. Salles, M. V. Sanchez-Vives, J. Schemmel, W. Senn, A. A. de Sousa, F. Ströckens, B. Thirion, K. Uludağ, S. Vanni, S. J. van Albada, W. Vanduffel, J. Vezoli, L. Vincenz-Donnelly, F. Walter, and L. Zaborszky. The coming decade of digital brain research: A vision for neuroscience at the intersection of technology and computing. *Imaging Neuroscience*, 2:1–35, Apr. 2024. ISSN: 2837-6056. DOI: `10.1162/imag_a_00137`.

[3] Big data needs a hardware revolution. *Nature*, 554(7691):145–146, Feb. 2018. ISSN: 1476-4687. DOI: `10.1038/d41586-018-01683-1`.

[4] R. Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017. ISSN: 0022-2496. DOI: `10.1016/j.jmp.2015.11.003`. Model-based Cognitive Neuroscience.

[5] C. Bredenberg and C. Savin. Desiderata for Normative Models of Synaptic Plasticity. *Neural Computation*, 36(7):1245–1285, June 2024. ISSN: 0899-7667. DOI: `10.1162/neco_a_01671`.

[6] C. Bredenberg, E. Williams, C. Savin, B. Richards, and G. Lajoie. Formalizing locality for normative synaptic plasticity models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 5653–5684. Curran Associates, Inc., 2023.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. arXiv: `2005.14165`.

[8] A. Clark. Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3):181–204, 2013. DOI: `10.1017/S0140525X12000477`.

[9] F. Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, Jan. 1989. ISSN: 1476-4687. DOI: 10.1038/337129a0.

[10] K. 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.

[11] K. Friston. Hierarchical models in the brain. *PLOS Computational Biology*, 4(11):1–24, Nov. 2008. DOI: 10.1371/journal.pcbi.1000211.

[12] K. Friston. Learning and inference in the brain. *Neural Networks*, 16(9):1325–1352, 2003. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2003.06.005. Neuroinformatics.

[13] K. Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, Jan. 2010. ISSN: 1471-0048. DOI: 10.1038/nrn2787.

[14] S. Golkar, T. Tesileanu, Y. Bahroun, A. M. Sengupta, and D. B. Chklovskii. Constrained predictive coding as a biologically plausible model of the cortical hierarchy, 2023. arXiv: 2210.15752.

[15] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Wiley, 1949. ISBN: 9780471367277.

[16] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020. arXiv: 2006.11239.

[17] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1), Nov. 2016. ISSN: 2041-1723. DOI: 10.1038/ncomms13276.

[18] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, Apr. 2020. ISSN: 1471-0048. DOI: 10.1038/s41583-020-0277-3.

[19] S. Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT*, 16(2):146–160, June 1976. ISSN: 1572-9125. DOI: 10.1007/bf01931367.

[20] B. Millidge, A. Seth, and C. L. Buckley. Predictive coding: a theoretical and experimental review, 2022. arXiv: 2107.12979.

[21] B. Millidge, Y. Song, T. Salvatori, T. Lukasiewicz, and R. Bogacz. A theoretical framework for inference and learning in predictive coding networks, 2022. arXiv: 2207.12316.

[22] B. Millidge, A. Tschantz, and C. L. Buckley. Predictive Coding Approximates Backprop Along Arbitrary Computation Graphs. *Neural Computation*, 34(6):1329–1368, May 2022. ISSN: 0899-7667. DOI: 10.1162/neco_a_01497.

[23] A. Ororbia. Spiking neural predictive coding for continual learning from data streams, 2022. arXiv: 1908.08655.

[24] A. Ororbia, A. Mali, D. Kifer, and C. L. Giles. Lifelong neural predictive coding: learning cumulatively online without forgetting, 2022. arXiv: 1905.10696.

[25] L. Pinchetti, T. Salvatori, Y. Yordanov, B. Millidge, Y. Song, and T. Lukasiewicz. Predictive coding beyond gaussian distributions, 2022. arXiv: 2211.03481.

[26] F. Pulvermüller. Neurobiological mechanisms for language, symbols and concepts: clues from brain-constrained deep neural networks. *Progress in Neurobiology*, 230:102511, 2023. ISSN: 0301-0082. DOI: 10.1016/j.pneurobio.2023.102511.

[27] F. Pulvermüller, R. Tomasello, M. R. Henningsen-Schomers, and T. Wennekers. Biological constraints on neural network models of cognitive function. *Nature Reviews Neuroscience*, 22(8):488–502, June 2021. ISSN: 1471-0048. DOI: `10.1038/s41583-021-00473-5`.

[28] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models, 2021. arXiv: `2112.10752`.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986. ISSN: 1476-4687. DOI: `10.1038/323533a0`.

[30] T. Salvatori, Y. Song, T. Lukasiewicz, R. Bogacz, and Z. Xu. Reverse differentiation via predictive coding, 2023. arXiv: `2103.04689`.

[31] C. Savin, P. Dayan, and M. Lengyel. Optimal recall from bounded metaplastic synapses: predicting functional adaptations in hippocampal area ca3. *PLOS Computational Biology*, 10(2):1–22, Feb. 2014. DOI: `10.1371/journal.pcbi.1003489`.

[32] N. Shervani-Tabar and R. Rosenbaum. Meta-learning biologically plausible plasticity rules with random feedback pathways. *Nature Communications*, 14(1), Mar. 2023. ISSN: 2041-1723. DOI: `10.1038/s41467-023-37562-1`.

[33] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. arXiv: `1503.03585`.

[34] Y. Song, T. Lukasiewicz, Z. Xu, and R. Bogacz. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579. Curran Associates, Inc., 2020.

[35] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: open and efficient foundation language models, 2023. arXiv: `2302.13971`.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. arXiv: `1706.03762`.

[37] P. Werbos. *Applications of advances in nonlinear sensitivity analysis.* In *System Modeling and Optimization.* Volume 38. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, pages 762–770. ISBN: 3-540-11691-5. DOI: `10.1007/BFb0006203`.

[38] J. C. R. Whittington and R. Bogacz. An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural Computation*, 29(5):1229–1262, May 2017. ISSN: 0899-7667. DOI: `10.1162/NECO_a_00949`.

[39] J. C. Whittington and R. Bogacz. Theories of Error Back-Propagation in the Brain. en. *Trends in Cognitive Sciences*, 23(3):235–250, Mar. 2019. ISSN: 13646613. DOI: `10.1016/j.tics.2018.12.005`.