Reverse Differentiation via Predictive Coding

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep learning has redefined the field of artificial intelligence (AI) thanks to the rise of artificial neural networks, which are architectures inspired by their neurological counterpart in the brain. Through the years, this dualism between AI and neuroscience has brought immense benefits to both fields, allowing neural networks to be used in dozens of applications. These networks use an efficient implementation of reverse differentiation, called backpropagation (BP). This algorithm, however, is often criticized for its biological implausibility (e.g., lack of local update rules for the parameters). Therefore, biologically plausible learning methods that rely on predictive coding (PC), a framework for describing information processing in the brain, are increasingly studied. Recent works prove that these methods can approximate BP up to a certain margin on multilayer perceptrons (MLPs), and asymptotically on any other complex model, and that zero-divergence inference learning (Z-IL), a variant of PC, is able to exactly implement BP on MLPs. However, the recent literature shows also that there is no biologically plausible method yet that can exactly replicate the weight update of BP on complex models. To fill this gap, in this paper, we generalize (PC and) Z-IL by directly defining them on computational graphs, and show that it can perform exact reverse differentiation. What results is the first biologically plausible algorithm that is equivalent to BP in the way of updating parameters on any neural network, and it is thus an important bridge for the interdisciplinary research of neuroscience and deep learning.

1 Introduction

In recent years, neural networks have achieved amazing results in multiple fields, such as image recognition [18, 27], natural language processing [53, 13], and game playing [50, 49]. All the models designed to solve these problems share a common ancestor, multilayer perceptrons (MLPs), which are fully connected neural networks with a feedforward multilayer structure and a mapping function $\mathbb{R}^n \to \mathbb{R}^m$. Although MLPs are able to approximate any continuous function [21] and theoretically can be used for any task, the empirical successes listed above show that more complex and task-oriented architectures perform significantly better than their fully connected ones. Hence, the last decades have seen the use of different layer structures, such as recurrent neural networks (RNNs) [19], transformers [53], convolutional neural networks (CNNs), and residual neural networks [18]. Albeit diverse architectures may look completely different, their parameters are all trained using gradient-based methods, creating a need for a general framework to efficiently compute gradients. Computational graphs, which are decompositions of complex functions into elementary ones, represent the ideal solution for this task, as they generalize the concept of neural network. In fact, they allow the use of reverse differentiation to efficiently compute derivatives and hence update the parameters of the network. In deep learning, this technique is used to quickly propagate the output error through the network, and it is hence famous under the name of *error backpropagation* (*BP*) [43]. While being a milestone of the field, this algorithm has often been considered biologically

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Table 1: Divergence between one-weight updates of backpropagation (BP) and zero-divergence inference learning (Z-IL) on different models, starting from the same initialization.

	MLP	CNN	RNN	ResNet18	Transformer
Divergence:	0	0	0	4.53×10^7	7.29×10^4

implausible, as it does not follow the rules of biological networks in the brain to update the parameters and propagate information.

A classic model of information processing in the brain is *predictive coding* (*PC*), which is used by computational neuroscientists to describe learning in the brain, and has promising theoretical interpretations, such as the minimization of free energy [10, 15, 16, 59] and probabilistic models [58]. Originally proposed to solve unsupervised learning tasks, PC is successful also in supervised models [58], and its variant, *inference learning* (*IL*) [58], is also able to approximate BP up to a certain margin on MLPs, and asymptotically on any other complex model [34]. Furthermore, a recent work proved that PC can do exact BP on MLPs, CNNs, and many-to-one RNNs using a learning algorithm called *zero-divergence inference learning* (*Z-IL*) [51, 45]. Z-IL is a biologically plausible method with local connections and local plasticity, and both its prediction and learning phases minimize the same energy function, which is an important biological property lacking in classical models. While this exactness result is thrilling and promising, Z-IL has limited generality, as it has only been shown to hold for MLPs, CNNs, and many-to-one RNNs. Actually, a recent study shows that there is no work yet to train high-performing deep neural networks on difficult tasks (e.g., ImageNet classification) using any algorithm other than BP [29]. This shows the existence of a gap in our understanding of the biological plausibility of BP, which can be summarized as follows: there is an approximation result (IL), which has been shown to hold for any complex model [58, 34], and an exactness result (Z-IL), only proven for MLPs, CNNs, and many-to-one RNNs. If the exactness result of Z-IL is extended to any complex model, then this would allow Z-IL to reach the performance of BP on complicated tasks such as ImageNet, bridging the interdisciplinary research of neuroscience and deep learning.

In this work, we close this gap by analyzing the Z-IL algorithm, and generalize the exactness result to every complex neural network. Particularly, we start from analyzing the Z-IL algorithm on different architectures by performing one iteration of BP and one iteration of Z-IL on two identically initialized networks, and compared the two weight updates by computing the Euclidean distance. The numbers reported in Table 1, show that the exactness result holds for CNNs and many-to-one RNNs, but fails for more complex architectures, such as residual and transformer neural networks. An analysis of the dynamics of the error propagation of Z-IL shows that the root of the problem is in the structure of the computational graph: in ResNet, for example, the skip connections design a pattern that does not allow Z-IL to exactly replicate the weight update of BP.

Consequently, this paper resolves the above problems by the following contributions: First, we generalize IL and Z-IL to work for every computational graph, and hence every possible neural network. Particularly, we propose a variant of Z-IL directly defined on computational graphs, which we prove to be equivalent to BP in the way of updating parameters on any possible model. This is, to our knowledge, the first biologically plausible algorithm that can exactly replicate the weight updates of BP on mapping functions of complex models. This sets an unprecedented state-of-the-art accuracy for the interdisciplinary research of neuroscience and deep learning, and Z-IL can now be considered as an alternative to BP, instead of just a theoretical tool. Second, we experimentally analyze the running time of Z-IL, IL, and BP on different popular architectures. We show that Z-IL is not only equivalent to BP in terms of accuracy, but also comparable in terms of computational efficiency. Furthermore, it is several orders of magnitude faster than IL.

2 Preliminaries

A computational graph $G = (V, E)$, where V is a finite nonempty set of vertices, and E is a finite set of edges, is a *directed acyclic graph* (*DAG*), which represents a complex function as a composition of elementary functions. Every vertex represents a computational step expressed by one of these elementary functions, and every edge pointing to this vertex represents an input of this function. We now briefly recall BP on computational graphs, and introduce how to perform PC on them. An example of a computational graph for the function $\mathcal{G}(z_1, z_2) = (\sqrt{z_1} + z_2)^2$ is shown in Fig. 1,

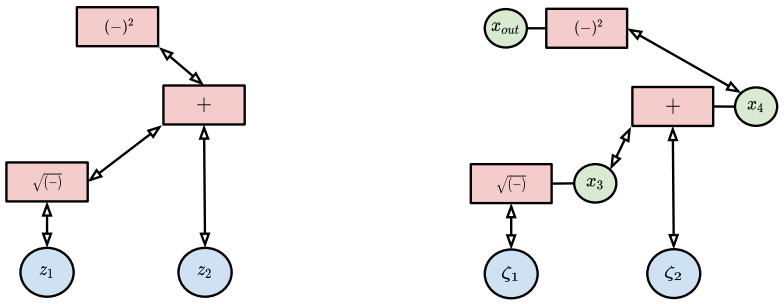


Figure 1: Left: Computational graph of the function $\mathcal{G}(z_1, z_2) = (\sqrt{z_1} + z_2)^2$. Right: Its predictive coding counterpart. Pointed upwards, the arrows related to the feedforward pass. Every internal vertex (red box) pictures the function g_i associated with it. The value nodes x_i of the input neurons are set equal to the input of the function (ζ_1 and ζ_2 in the above figure) . Hence, we have omitted them from the plots to make the notation lighter. The same notation is adopted in later figures.

where the arrows pointing upwards denote the forward pass, and the ones pointing downwards the reverse pass. We call $C(i)$ and $P(i)$ the indices of the children and parents of v_i, respectively. For ease of presentation, the direction considered when using this notation will always be the reverse pass (downwards arrows in Fig. 1). Hence, leaf nodes (nodes at the bottom) have no children vertices, and output node (nodes at the top) have no parent vertices. Furthermore, we call v_i the vertices of the graph G, and $e_{i,j}$ the directed edge that starts at v_i and ends at v_j. The first n vertices $v_1, \dots, v_n$ are the leafs of the graph and represent the n parameters of $\mathcal{G}$, while the last vertex, v^{out}, represents the output of the function. We call d_i the minimum distance from the output node v^{out} to v_i (i.e., the minimum number of edges separating v_i and v_{out}).

2.1 BP on Computational Graphs

Let $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, and $\{g_i\}$ be a factorization of $\mathcal{G}$ in elementary functions, which have to be computed according to a computational graph. Particularly, a computational graph $G = (V, E)$ associated with $\mathcal{G}$ is formed by a set of vertices V with cardinality $|V|$, and a set of directed edges E, where an edge $e_{i,j}$ is the arrow that points to v_j starting from v_i. With every vertex $v_i \in V$, we associate an elementary function $g_i : \mathbb{R}^{k_i} \to \mathbb{R}$, where k_i is the number of edges pointing to v_i. The choice of these functions is not unique, as there exist infinitely many ways of factoring $\mathcal{G}$. It hence defines the structure of a particular computational graph. Given a parameter vector $\bar{z} \in \mathbb{R}^n$, we denote by μ_i the value of the vertex v_i during the forward pass, which is set to z_i if v_i is an input vertex. Particularly, it is computed iteratively as follows:

$$\mu_i = \begin{cases} z_i & \text{for } i \leq n\,; \\ g_i(\{\mu_j\}_{j \in C(i)}) & \text{for } i > n\,. \end{cases} \tag{1}$$

Note that if the considered function $\mathcal{G}$ is a neural network, the parameter vector $\bar{z}$ would correspond to the weight parameters of a neural network. We then have $\mathcal{G}(\bar{z}) = \mu_{|V|} = \mu_{out}$. The computational flow just described is represented by the red arrows in Fig. 1. We now introduce the classical problem of reverse differentiation, and show how it is used to compute the derivative relative to the output. Let $\bar{z} = (z_1, \dots, z_n)$ be a vector of parameters, and $\mathcal{G}(\bar{z}) = \mu_{out}$ be the output. Reverse differentiation is a key technique in machine learning and artificial intelligence (AI), as it allows to compute $\frac{\partial \mathcal{G}}{\partial z_i}$ for every $i < n$ efficiently. This is necessary to implement BP at a reasonable computational cost, especially considering the extremely overparametrized architectures used today. This is done iteratively, according to the following equation:

$$\partial \mathcal{G}/\partial \mu_i = \sum\nolimits_{j \in P(i)} \partial \mathcal{G}/\partial \mu_j \cdot \partial \mu_j/\partial \mu_i = \sum\nolimits_{j \in P(i)} \partial \mathcal{G}/\partial \mu_j \cdot \partial \mu_j/\partial \mu_i. \tag{2}$$

To obtain the desired formula for the parameters, it suffices to recall that $\mu_i = z_i$ for every $i \leq n$.

Update of the leaf nodes: Given a vector of parameters $\bar{z}$, we consider a desired output y for the function $\mathcal{G}$. The goal of a learning algorithm is to update the parameters $(z_1, \dots, z_n)$ of a

computational graph to minimize the quadratic loss $E = \frac{1}{2}(\mu_{out} - y)^2$. Hence, the parameters are updated as follows:

$$\Delta z_i = -\alpha \cdot \partial E/\partial z_i = \alpha \cdot \sum\nolimits_{j \in P(i)} \delta_j \, \partial \mu_j / \partial z_i, \tag{3}$$

where α is the learning rate, and $\partial E/\partial z_i$ is computed using reverse differentiation. We use the parameter δ_j to represent the error signal, i.e., the propagation of the output error among the vertices of the graph. It can be computed according to the following recursive formula:

$$\delta_i = \begin{cases} \mu_{out} - y & \text{if } i = |V|\,; \\ \sum_{j \in P(i)} \delta_j \, \partial \mu_j / \partial z_i & \text{if } n < i < |V|. \end{cases} \tag{4}$$

2.2 IL on Computational Graphs

We now show how the just introduced forward and backward passes change when considering a PC computational graph $G = (V, E)$ of the same function $\mathcal{G}$. A similar framework to the one that we are about to show, has been developed in [34]. We associate with every vertex v_i, with $i > n$, a new time-dependent random variable $x_{i,t}$, called value node, and a prediction error $\varepsilon_{i,t}$. Given a parameter vector $(\zeta_1, \dots, \zeta_n)$, the values μ_i are computed as follows: for the leaf vertices, we have $\mu_{i,t} = \zeta_i$ and $\varepsilon_{i,t} = 0$ for $i \leq n$, while for the other values, we have

$$\mu_{i,t} = g_i(\{x_{j,t}\}_{j \in C(i)}) \quad \text{and} \quad \varepsilon_{i,t} = \mu_{i,t} - x_{i,t}. \tag{5}$$

This allows to compute the value $\mu_{i,t}$ of a vertex by only using information coming from vertices connected to v_i. As in the case of PCNs, every computation is strictly local. The value nodes of the network are updated continuously in order to minimize the following loss function, defined on all the vertices of G:

$$F_t = \frac{1}{2} \sum\nolimits_{i=1}^{|V|} (\mu_{i,t} - x_{i,t})^2 = \frac{1}{2} \sum\nolimits_{i=1}^{|V|} (\varepsilon_{i,t})^2. \tag{6}$$

The output x_{out} of $\mathcal{G}(\bar{\zeta})$ is then computed by minimizing this energy function through an inference process. The update rule is $\Delta x_{i,t} = -\gamma \, \partial F_t / \partial x_{i,t}$, where γ is a small positive constant called *integration step*. Expanding this equation gives:

$$\Delta x_{i,t} = -\gamma \, \partial F_t / \partial x_{i,t} = \gamma(\varepsilon_{i,t} + \sum\nolimits_{j \in P(i)} \varepsilon_{j,t} \, \partial \mu_j / \partial x_{i,t}) \, . \tag{7}$$

Note that during the forward pass, all the value nodes $x_{i,t}$ converge to μ_i, as t grows to infinity. This makes the final output of the forward passes of inference learning on the new computational graph equivalent to that of the normal computational graph.

Update of the leaf nodes: Let $\bar{\zeta}$ be a parameter vector, and y be a fixed target. To update the parameter vector and minimize the error on the output, we fix $x_{out} = y$. Thus, we have $\varepsilon_{out,t} = \mu_{out} - y$. By fixing the value node $x_{out,t}$, most of the error nodes can no longer decay to zero. Hence, the error $\varepsilon_{out,t}$ gets spread among the other error nodes on each vertex of the computational graph by running the inference process. When the inference process has either converged, or it has run for a fixed number of iterations T, the parameter vector gets updated by minimizing the same loss function F_t. Thus, we have:

$$\Delta \zeta_i = -\alpha \, \partial F_t / \partial \zeta_i = \alpha \sum\nolimits_{j \in P(i)} \varepsilon_{j,t} \, \partial \mu_j / \partial \zeta_i. \tag{8}$$

All computations are local (with local plasticity) in IL, and the model can autonomously switch between prediction and learning via running inference. The main difference between BP and IL on computational graphs is that the update of the parameters of BP is invariant of the structure of the computational graph: the way of decomposing the original function $\mathcal{G}$ in elementary functions, does not affect the update of the parameters. This is not the case for IL, as different decompositions lead to different updates of the value nodes, and hence of the parameters. However, it has been shown that, while following different dynamics, these updates are asymptotically equivalent [34].

2.3 Z-IL on MLPs

Recently, a new learning algorithm, called zero-divergence inference learning (Z-IL), was shown to perform *exact* backpropagation on fully connected predictive coding networks (PCNs), a biologically

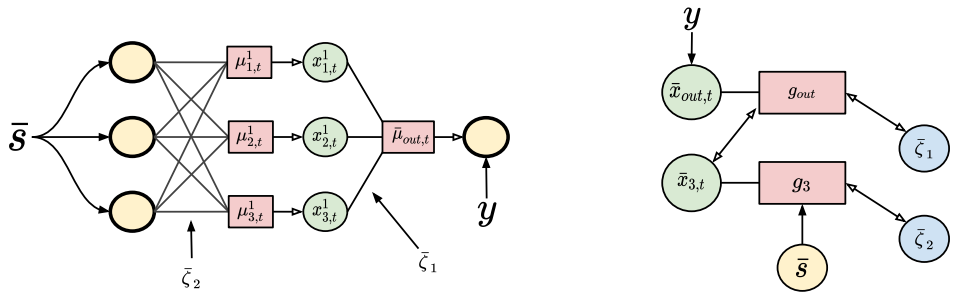


Figure 2: Left: Example of a 2-layer PCN. In these networks, it is possible to realize every computation locally using error nodes and value nodes in a biologically plausible way. For a more detailed discussion, we refer to [58]. Right: The corresponding computational graph.

Algorithm 1 Learning one training pair $(\bar{s}, y)$ with Z-IL

Require: x_{out} is fixed to y; $\gamma = 1$
Initialize $x_{l,0} = \zeta_l$ for every leaf node; $x_{i,0} = \mu_{i,0}$ for every internal node
for $t = 0$ to L **do**
  for each vertex v_i **do**
    Update $x_{i,t}$ to minimize F_t via Eq. (7)
  end for
  if $t = l$ **then**
    Update $\bar{\zeta}_l$ to minimize F_t via Eq. (8)
  end if
end for

plausible alternative of MLPs. In detail, starting from a network with the same parameters, the update of the weights after one iteration of BP is identical to the one given by one iteration of Z-IL. To be as close as possible to the original formulation of Z-IL, we adopt the same notation of that work, and index the layers starting from the output layer (layer 0), and finishing at the input layer (layer L).

Let $\mathcal{G}(\bar{z})$ be the function expressed by an artificial neural network (ANN). The leaf vertices of its computational graph are the weights, and every weight of a layer l has the distance l from the output vertex. The figure for the multidimensional case is equivalent, as it suffices to consider $x_i \in \mathbb{R}^n$ and $z_i \in \mathbb{R}^{n \times n}$.

This new algorithm differs from IL, as it sets the initial error $\varepsilon_{i,0}$ of every vertex v_i to zero. This is done by performing a forward pass and setting $\mu_{i,0} = x_{i,0}$ for every vertex v_i. Furthermore, its inference phase only lasts for L iterations.

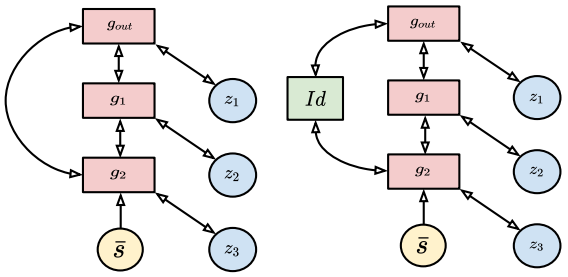
Update of the leaf nodes: Z-IL introduces a new rule to update the weights of a fully connected PCN: the weight parameters ζ_l of layer l get only updated at time step $t = l$. Using the notation adopted for computational graphs, every leaf node ζ_i in Fig. 2 gets updated at $t = i$. Note that this condition needs to be rephrased, as computational graphs of general functions are usually not divided into layers. Hence, a new formulation of Z-IL for computational graphs is given in Section 5. Alg. 1 shows how Z-IL performs a single update the parameters when trained on a labelled point $(\bar{s}, y)$. For a detailed derivation of all the equations, we refer to the original paper [51]. The main theoretical result is as follows.

Theorem 1. *Let M be a full connected PCN trained with Z-IL, and let M' be its corresponding MLP, initialized as M, and trained with BP. Then, given the same datapoint s to both networks, we have*

$$\Delta \bar{z}_l = \Delta \bar{\zeta}_l \tag{9}$$

for every layer $l \geq 0$.

In the rest of this work, we generalize this result to the case where $\mathcal{G}(\bar{z})$ is a general function, and not only an MLP. In the next section, we use residual connections to highlight the problem of applying Z-IL to general computational graphs. We will then solve it by defining Z-IL for computational graphs, and prove a generalization of Theorem 1.



Algorithm 2 Generating a levelled DAG G'

Require: G is a DAG.
Require: $(v_0, \dots, v_n)$ is a topological sort.
for every j in $(0, n)$ included **do**
  for each vertex v_i in $P(j)$ **do**
    Add $(d_j - D_i)$ identity vertices to $e_{i,j}$
  end for
end for

Figure 3: Left: Computational graphs of an ANN with a residual connection. Centre: A version with an identity vertex. Right: Pseudocode of the algorithm used to generate levelled graphs.

3 The Problem of Skip Connections

In this section, we provide a toy example that shows how Z-IL and BP behave on the computational graph of an ANN with a skip connection. Particularly, we show that it is impossible for Z-IL to replicate the same update of BP on all the parameters, unless the structure of the computational graph is altered. Let us consider the network expressed in Fig. 3, left side, which corresponds to the function $\mathcal{G}(x, z) = z_3(z_1 z_2 + 1)x$.

BP: Given an input value $\bar{s}$ and a desired target y, BP computes the gradient of every leaf node using reverse differentiation, and updates the parameters of z_3 as follows:

$$\Delta z_3 = -\alpha \cdot \partial E / \partial z_3 = \alpha \cdot \delta(z_1 z_2 + 1)x, \tag{10}$$

where $\delta = (\mu_{out} - y)$, and E is the quadratic loss defined on the output node.

Z-IL: Given an input value $\bar{s}$ and a desired target y, the inference phase propagates the output error through the graph via Eq. (8). Z-IL updates ζ_3 at $t = 3$, as it belongs to the third hidden layer. This leads to the following:

$$\Delta\zeta_3 = -\alpha \cdot \partial F_3 / \partial \zeta_3 = \alpha \cdot \delta \zeta_1 \zeta_2 x, \tag{11}$$

where $\delta = \varepsilon_{out,0} = (\mu_{out,0} - y)$, and F_2 is computed via Eq. (6). Note that this update is different from the one obtained by BP. We now analyze the reason of this mismatch and provide a solution.

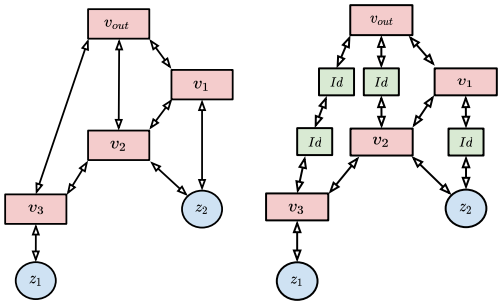
3.1 Identity Vertices

The error signal propagated by the inference process reaches ζ_3 in two different moments: $t = 2$ from the output vertex, and $t = 3$ from g_2. Dealing with vertices that receive error signals in different moments is problematic for our formulation of the Z-IL algorithm, as every leaf node only gets updated once. Furthermore, changing the update rule of Z-IL does not solve the problem, as no other combination of updates produces the same weight update defined in Eq. (10). To solve this problem, we then have to assure that every node of the graph is reached by the error signal in a single time step. This result is trivially obtained on computational graphs that are levelled DAGs, i.e., graphs where every directed path connecting two vertices has the same length.

For every vertex v_i, it is possible to write the associated elementary function g_i as a composition with the identity function, i.e., $g_i \circ Id$. Given two vertices v_i, v_j connected by the edge $e_{i,j}$, it is then possible to add a new vertex v_k by splitting the edge $e_{i,j}$ into $e_{i,k}, e_{k,j}$, whose associated function g_k is the identity. This leaves the function expressed by the computational graph unvaried, as well as the computation of the derivatives, the forward pass, and the backward pass of BP. Adding these identity vertices in the right places, makes the computational graph levelled, allowing every vertex to receive the error signals at the same time step. Consider now the levelled graph of Fig. 3, where an identity node has been added in the skip connection. The error signal of both g_1 and g_{out} reaches g_2 simultaneously at $t = 2$. Hence, at $t = 3$, Z-IL updates ζ_3 as follows:

$$\Delta\zeta_3 = -\alpha \cdot \partial F_3 / \partial \zeta_3 = \alpha \cdot \delta(\zeta_1 \zeta_2 + 1)x. \tag{12}$$

If we have $\zeta_i = z_i$, this weight update is equivalent to the one performed by BP and expressed in Eq. (10). Hence, we have shown that Z-IL is able to produce the same weight update of BP in a



Algorithm 3 Z-IL for computational graphs.

Require: x_{out} is fixed to a label y,
Require: $\{S_k\}_{k=0,\dots,K}$ is a level structure of $G(V, E)$;
Require: $x_{i,0} = \mu_{i,0}$ for every internal node.
for $t = 0$ to K **do**
  for each internal vertex v_i **do**
    Update $x_{i,t}$ to minimize F_t via Eq. (7)
    if $t = k$ **then**
      Update each leaf node $\zeta_{i,t} \in S_k$ to minimize F_t via Eq. (8)
    end if
  end for
end for

Figure 4: Computational graphs of the same function $\mathcal{G}$. Left: Original graph G. Centre: Transformed graph, with the identity vertices in green. Right: Pseudocode of Z-IL on computational graphs.

simple neural network with one skip connection, thanks to adding an identity vertex. In the next section, we generalize this result.

4 Levelled Computational Graphs

In this section, we show that, given any computational graph, it is always possible to generate a levelled equivalent. Particularly, we provide an algorithm that performs this task by adding identity nodes. This leads to the first result needed to prove our main theorem: Every function admits a computational graph with a level structure, where a level structure of a directed graph is a partition of the vertices into subsets that have the same distance from the top vertex.

Let $G = (V, E)$ be a computational graph, and let $S_1, \dots, S_K$ be the family of subsets of V defined as follows: a vertex v_i is contained in S_k if there exists a directed path of length k connecting v_i to v_{out}, i.e.,

$$S_k = \{v_i \in V | \ \exists \text{ a path } (e_{out,j_1}, \dots, e_{j_{k-1},i})\}. \tag{13}$$

Hence, we have that v_{out} is contained in S_0, its children vertices in S_1, and so on. In a levelled graph, every vertex is contained in one and only one of the subsets. Denote by D_i the maximum distance between v_{out} and the parent nodes of v_i, i.e., $D_i = \max_{v_j \in P(i)} d_j$. We now show for every DAG G how to make every vertex v_i to be contained in only one subset S_k, without altering the dynamics of the computational graph. This is done by adding identity nodes in the graph.

Let G be a DAG with root v_0, and $(v_0, v_1, \dots, v_n)$ be a topological sort of the vertices of G. Starting from the root, for every vertex v_j, we replace every existing edge $e_{i,j}$ with the following path:

$$v_i \to Id \to \cdots \to Id \to v_j, \tag{14}$$

which connects v_i to v_j via $d_j - D_i$ identity nodes. When this process has been repeated on all the vertices, we obtain a levelled DAG. To use the introduced notation, this is equivalent to having every $v_i \in G$ that belongs to one and only one subset S_k. This follows because every pair of disconnected paths between two vertices is forced to have the same length, due to the addition of identity vertices. Hence, we obtain the following theorem.

Theorem 2. *Given a function $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$ and any factorization of it expressed by elementary functions $\{g_i\}$, there exist a levelled computational graph $G = (V, E)$ that represents this factorization.*

Under the machine learning perspective, this theorem shows that every neural network can be expressed as a levelled computational graph when needed, thanks to the addition of identity nodes. Hence, every result shown for levelled computational graphs can be naturally extended to every possible neural network structure. This is the case for the result of the next section, which states that Z-IL allows PCNs to do exact BP on every possible neural network.

Table 2: Average running time of each weights update (in ms) of BP, IL [34], and Z-IL for computational graphs.

Training Method	MLP	AlexNet [27]	Many-to-one RNN	ResNet18 [18]	Transformer Net [53]
BP	3.72	8.61	5.64	12.43	20.43
IL	594.25	661.53	420.01	1452.34	1842.64
Z-IL	3.81	8.86	5.67	12.53	20.53

5 Z-IL for Levelled Computational Graphs

Let $G = (V, E)$ be the levelled computational graph of a function $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$, and consider the partition of V via its level structure $S_1, \dots, S_K$. As in the original formulation of Z-IL, we present two variants of IL that allow predictive coding to exactly replicate the parameter update of BP.

Variant 1: Instead of continuously running the inference on all the leaf nodes of G, we only run it on the internal vertices. Then, at every time step t, we update all the leaf nodes $v_i \in S_t$. Particularly, for every internal vertex v_i, training continues as usual via Eq. (7). On the other hand, leaf nodes are updated according to the folllowing equation:

$$\Delta\zeta_{i,t} = \begin{cases} \alpha \cdot \sum_{j \in P(i)} \varepsilon_{j,t}\, \partial\mu_j / \partial\zeta_i. & \text{if } v_i \in S_t \\ 0 & \text{if } v_i \notin S_t. \end{cases} \tag{15}$$

This shows that one full update of the parameters requires $t = K$ steps. Note that, in the case of multilayer networks, K is equal to the number of layers L.

Variant 2: Differently from IL, where the input parameters and the output are presented simultaneously, Z-IL first presents the input vector to the function, and computes a forward pass. Then, once the values μ_i of all the internal vertices have been computed, their value nodes are initialized to have zero error, i.e., $x_{i,0} = \mu_i$, and the output node is set equal to the label y. This is done to emulate the behavior of BP, which first computes the output vector, and then compares it to the label.

Overall, the functioning of Z-IL for computational graphs is summarized in the algorithm in Fig 4. We now show that this new formulation of Z-IL is able to replicate the same weight update of BP on any function $\mathcal{G}$.

Theorem 3. *Let $(\bar{z}, y)$ and $(\bar{\zeta}, y)$ be two points with the same label y, and $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$ be a function. Assume that the update $\Delta\bar{z}$ is computed using BP, and the update $\Delta\bar{\zeta}$ using Z-IL with $\gamma = 1$. Then, if $\bar{z} = \bar{\zeta}$, and we consider a levelled computational graph of $\mathcal{G}$, we have*

$$\Delta z_i = \Delta\zeta_i \tag{16}$$

for every $i \leq n$.

This proves the main claims made about Z-IL: (i) exact BP and exact reverse differentiation can be made biologically plausible on the computational graph of *any* function, and (ii) Z-IL is a learning algorithm that allows PCNs to perfectly replicate the dynamics of BP on any function.

6 Experiments

In the above sections, we have theoretically proved that the proposed generalized version of Z-IL is exactly equivalent to BP on every related neural networks. So, there is no further need for an experimental evaluation of the equivalence. Counter-checking experiments have further confirmed this: the divergences of weight updating between BP and Z-IL are always zero on all tested neural networks. In this section, we complete the picture of this work with experimental studies to evaluate the computational efficiency of Z-IL, and quantitatively compare it with those of BP and IL. Table 2 shows the average running time of each weights update of BP, IL, and Z-IL, on the following five neural networks. All dependencies and their versions are specified in the supplementary material.

MLP: We trained three MLPs with different depth (2, 3, and 4 layers, respectively) on FashionMNIST, and the dimension of each layer is 128 neurons. The batch size and learning rate are 20 and 0.01, respectively. The results of BP and Z-IL are averaged numbers over the three architectures.

AlexNet and ResNet: AlexNet [27] and ResNet18 [18] are trained on ImageNet. The batch size and learning rate are the same as for the MLPs.

RNNs: We trained a reinforcement learning agent on a single-layer many-to-one RNN on eight different Atari games. Similarly, the size of hidden and output layers is 128, batch size and learning rate are 20 and 0.01, respectively, and the reported results are averaged over eight games.

Transformer: We trained a 1-layer transformer architecture using torchtext to generate a Wikitext-2 dataset. Batch size and learning rate are 4 and 0.01, respectively.

6.1 Results and Evaluations

As shown in Table 2, the computational time costs of Z-IL is very close to that of BP, and is several orders of magnitude lower than that of IL. Consequently, this proves that Z-IL is an alternative to BP in practice, instead of just being a theoretical tool.

The high computational time costs of IL is due to the fact that it has to wait for the error to be converged before updating weights once. Since the convergence of the error may take a huge number of iterations, in practice, a fixed number T is usually used. For example, for small MLPs, T is set to 20 in [58], and as larger models require higher numbers of iterations for the error to converge, T is set between 100 and 200 for mid-size architectures, such as RNNs and CNNs in [34]. For a fair comparison, the values of T in our experiments follow the settings in [34]. Furthermore, although Z-IL also requires L inference steps to complete one update of weights in all layers, L in Z-IL is set to be the number of layers of the corresponding network, whose value is usually much smaller than that of T in IL, i.e., $T >> L$. Consequently, IL requires much more steps of inference than Z-IL for each weight update, resulting in a significantly higher running time than Z-IL.

Furthermore, an interesting aspect is that both [58] and [34] never verified that the inference indeed has converged in their IL models, though it is the most important theoretical requirement for IL to approximate BP. However, their models still perform comparably to BP in terms of testing accuracy. Moreover, the reasonable approximation in [58] is achieved with a quite small $T = 20$. Actually, the proposed Z-IL explains the above findings, as we show that strict equivalence can be achieved with a small number of inference steps; one just needs to satisfy the proposed conditions properly.

7 Related Work

PC is an influential theory of cortical function in theoretical and computational neuroscience. It has appealing theoretical interpretations, such as free-energy minimization [10, 15, 16, 59] and variational inference of probabilistic models [58]. It offers a single mechanism that accounts for diverse perceptual phenomena observed in the brain, such as end-stopping [41], repetition-suppression [5], illusory motions [33, 55], bistable perception [20, 56], and even attentional modulation of neural activity [14, 23]. There are also variants of PC developed into different biologically plausible process theories specifying cortical microcircuits that potentially implement such theories [8, 23, 48, 52]. Due to this solid biological grounding, PC is also attracting interest in the machine learning community recently, especially focusing on finding the links between PC and BP [58, 34].

Biologically plausible approximations to BP have been intensively studied since the flourishing of BP, because on the one hand, the underlying principles of BP are unrealistic for an implementation in the brain [17, 12, 1, 30, 42, 59], but on the other hand, BP outperforms all alternative discovered frameworks [6] and closely reproduces activity patterns observed in the cortex [64, 31, 11, 63, 25, 62, 24, 7, 57]. However, earlier biologically plausible approximations to BP were not scaling to larger and more complicated problems [30, 38, 26, 9, 28, 35, 46, 47, 32, 22]. More recent works show the capacity of scaling up biologically plausible approximations to the level of BP [60, 37, 36, 4, 3, 2, 54]. However, to date, none of the earlier or recent models has bridged the gaps at a degree of demonstrating an equivalence to BP, though some of them [28, 58, 36, 40, 39, 34] demonstrate that they approximate BP, or are equivalent to BP under unrealistic restrictions, e.g., the feedback is sufficiently weak [61, 58, 44].

8 Conclusion

In this paper, we have extended the use of the Z-IL algorithm to all possible neural networks. While IL approximates BP in single-step weight updates under unrealistic and non-trivial requirements, the proposed generalized version of Z-IL is proved to be always equivalent to BP, with no extra restriction on the mapping function and the type of neural networks. Furthermore, experimental studies have been conducted to show that the computational efficiency of Z-IL is comparable to that of BP, and is several orders of magnitude better than IL. This demonstrates that Z-IL is very suitable for practical applications, as it can be implemented for any large and state-of-the-art neural network, with a time efficiency that is similar to BP. This significantly strengthens the link between PC and BP, which is an important finding for both the deep learning and the neuroscience community. Specifically, it suggests that (i) PC and IL are interesting directions to explore in the research of deep learning, and (ii) that some form of BP may indeed be performed in the brain. Our work could also indicate that BP happens in the brain as just one part of the learning process, since Z-IL can be seen as a specific moment during the training of PC with IL; thus, there may be something missing apart from BP. Hence, BP may be more important in neuroscience than commonly thought.

References

[1] M. Abdelghani, T. P. Lillicrap, and D. B. Tweed. Sensitivity derivatives for flexible sensorimotor learning. *Neural Computation*, 20(8):2085–2111, 2008.

[2] M. Akrout, C. Wilson, P. C. Humphreys, T. Lillicrap, and D. Tweed. Using weight mirrors to improve feedback alignment. *arXiv:1904.05391*, 2019.

[3] J. Aljadeff, J. D'amour, R. E. Field, R. C. Froemke, and C. Clopath. Cortical credit assignment by Hebbian, neuromodulatory and inhibitory plasticity. *arXiv:1911.00307*, 2019.

[4] Y. Amit. Deep learning with asymmetric connections and Hebbian updates. *Frontiers in Computational Neuroscience*, 13:18, 2019.

[5] R. Auksztulewicz and K. Friston. Repetition suppression and its contextual determinants in predictive coding. *Cortex*, 80, 2016.

[6] P. Baldi and P. Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83, 2016.

[7] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557, 2018.

[8] A. M. Bastos, W. M. Usrey, R. A. Adams, G. R. Mangun, P. Fries, and K. J. Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

[9] Y. Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv:1407.7906*, 2014.

[10] R. Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017.

[11] C. F. Cadieu, H. Hong, D. L. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, and J. J. DiCarlo. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS Computational Biology*, 10(12), 2014.

[12] F. Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics.* Association for Computational Linguistics, 2019.

[14] H. Feldman and K. Friston. Attention, uncertainty, and free-energy. *Frontiers in Human Neuroscience*, 4, 2010.

[15] K. Friston. Learning and inference in the brain. *Neural Networks*, 16(9):1325–1352, 2003.

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

[17] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63, 1987.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.

[20] J. Hohwy, A. Roepstorff, and K. Friston. Predictive coding explains binocular rivalry: An epistemological review. *Cognition*, 108(3), 2008.

[21] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 1989.

[22] B. Illing, W. Gerstner, and J. Brea. Biologically plausible deep learning—but how far can we go with shallow networks? *Neural Networks*, 118, 2019.

[23] R. Kanai, Y. Komura, S. Shipp, and K. Friston. Cerebral hierarchies: Predictive processing, precision and the pulvinar. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370, 2015.

[24] A. J. Kell, D. L. Yamins, E. N. Shook, S. V. Norman-Haignere, and J. H. McDermott. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98, 2018.

[25] S.-M. Khaligh-Razavi and N. Kriegeskorte. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS Computational Biology*, 10(11), 2014.

[26] K. P. Körding and P. König. Supervised and unsupervised learning with two sites of synaptic integration. *Journal of Computational Neuroscience*, 11(3):207–215, 2001.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *26th Annual Conference on Neural Information Processing Systems (NIPS) 2012*, 2012.

[28] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. Difference target propagation. In *Proc. ECMLPKDD*, 2015.

[29] T. Lillicrap, A. Santoro, L. Marris, C. Akerman, and G. Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21, 04 2020.

[30] 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):1–10, 2016.

[31] T. P. Lillicrap and S. H. Scott. Preference distributions of primary motor cortex neurons reflect control solutions optimized for limb biomechanics. *Neuron*, 77(1), 2013.

[32] T.-H. Lin and P. T. P. Tang. Dictionary learning by dynamical neural networks. *arXiv:1805.08952*, 2018.

[33] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv:1605.08104*, 2016.

[34] B. Millidge, A. Tschantz, and C. L. Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv:2006.04182*, 2020.

[35] A. Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, 2016.

[36] A. Nøkland and L. H. Eidnes. Training neural networks with local error signals. *arXiv:1901.06656*, 2019.

[37] D. Obeid, H. Ramambason, and C. Pehlevan. Structured and deep similarity matching via structured and deep Hebbian networks. In *Advances in Neural Information Processing Systems*, 2019.

[38] R. C. O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, 8(5):895–938, 1996.

[39] A. G. Ororbia and A. Mali. Biologically motivated algorithms for propagating local target representations. In *Proc. AAAI*, volume 33, pages 4651–4658, 2019.

[40] I. Ororbia, G. Alexander, P. Haffner, D. Reitter, and C. L. Giles. Learning to adapt by minimizing discrepancy. *arXiv:1711.11542*, 2017.

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

[42] P. R. Roelfsema and A. Holtmaat. Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience*, 19(3):166, 2018.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[44] J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, pages 8721–8732, 2018.

[45] T. Salvatori, Y. Song, T. Lukasiewicz, Z. Xu, and R. Bogacz. Predictive coding can do exact backpropagation on convolutional and recurrent neural networks. *arXiv:2103.03725*, 2021.

[46] B. Scellier and Y. Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017.

[47] B. Scellier, A. Goyal, J. Binas, T. Mesnard, and Y. Bengio. Generalization of equilibrium propagation to vector field dynamics. *arXiv:1808.04873*, 2018.

[48] S. Shipp. Neural elements for predictive coding. *Frontiers in Psychology*, 7:1792, 2016.

[49] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 2016.

[50] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550, 2017.

[51] Y. Song, T. Lukasiewicz, Z. Xu, and R. Bogacz. Can the brain do backpropagation? — Exact implementation of backpropagation in predictive coding networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*, 2020.

[52] M. W. Spratling. Reconciling predictive coding and biased competition models of cortical function. *Frontiers in Computational Neuroscience*, 2:4, 2008.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017.

[54] X. Wang, X. Lin, and X. Dang. Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Networks*, 2020.

[55] E. Watanabe, A. Kitaoka, K. Sakamoto, M. Yasugi, and K. Tanaka. Illusory motion reproduced by deep neural networks trained for prediction. *Frontiers in Psychology*, 9:345, 2018.

[56] V. Weilnhammer, H. Stuke, G. Hesselmann, P. Sterzer, and K. Schmack. A predictive coding account of bistable perception-a model-based fmri study. *PLoS Computational Biology*, 13(5), 2017.

[57] J. Whittington, T. Muller, S. Mark, C. Barry, and T. Behrens. Generalisation of structural knowledge in the hippocampal-entorhinal system. In *Advances in Neural Information Processing Systems*, 2018.

[58] J. C. 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), 2017.

[59] J. C. Whittington and R. Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 2019.

[60] W. Xiao, H. Chen, Q. Liao, and T. Poggio. Biologically-plausible learning algorithms can scale to large datasets. *arXiv:1811.03567*, 2018.

[61] X. Xie and H. S. Seung. Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation*, 15(2), 2003.

[62] D. L. Yamins and J. J. DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, 19(3), 2016.

[63] D. L. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23), 2014.

[64] D. Zipser and R. A. Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158), 1988.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] This is a theoretical work: the conditions for our claims to hold are defined throughout the paper and in the theorems.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] This paper does not raise any negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] This paper do not raise any ethical issues.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Yes, in the supplementary material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] No, because the code is proprietary, but we give all the information needed to reproduce the results.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] The few experiments of this paper do not require error bars.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] We use public data.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use public data.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We use public data, which do not contain any of these.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We do not use crowdsourcing resource.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We do not use crowdsourcing resource.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We do not use crowdsourcing resource.

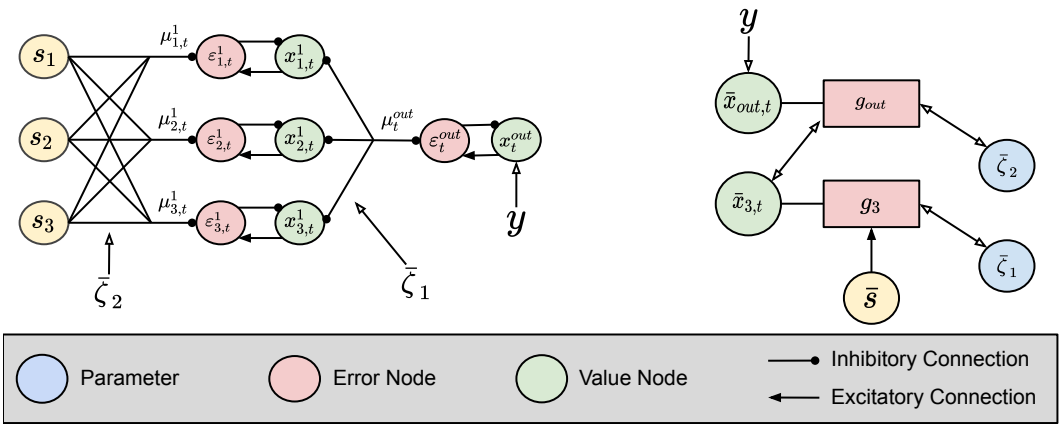


Figure 5: Left: Example of a 2-layer PCN with inhibitory and excitatory connections. In these networks, it is possible to realize every computation locally using error nodes and value nodes in a biologically plausible way. For a more detailed discussion, we refer to [58]. Right: The corresponding computational graph.

A Biological Plausibility of PCNs

To improve the clarity of the presentation, we have decided to describe PCNs using value nodes $\bar{x}^l_{i,t}$ and their predictions $\bar{\mu}^l_{i,t}$. This presentation, however, does not fully highlights the reasons that make PCNs trained with IL and ZIL biologically plausible. We now address this problem. It is in fact possible to represent PCNs using only local information, which gets propagate through the network via inhibitory and excitatory connections. Particularly, the value nodes of a layer l are connected to the error nodes of layer $l-1$ via inhibitory connections. The same holds for computational graphs. Graphical representations of a 2-layer PCN and its computational graph are found in Fig. 5.

B Proof of Theorem 2

In this section, we prove the main theorem of our work, which has already been stated in Section 5 of the main body.

Theorem 4. *Let $(\bar{z}, y)$ and $(\bar{\zeta}, y)$ be two points with the same label y, and $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$ be a function. Assume that the update $\Delta\bar{z}$ is computed using BP, and the update $\Delta\bar{\zeta}$ uses Z-IL with $\gamma = 1$. Then, if $\bar{z} = \bar{\zeta}$, and we consider a levelled computational graph of $\mathcal{G}$, we have*

$$\Delta z_i = \Delta \zeta_i, \tag{17}$$

for every $i \leq n$.

Proof. As Z-IL acts on the levelled version of G, in this proof we consider levelled computational graphs, i.e., graphs where the distance from the top generates a partition of the vertices. We denote d_i the distance of a vertex v_i to the root vertex v_{out}, i.e., $d_i = k$ if $v_i \in S_k$. Furthermore, we denote by d_{max} the maximum distance between the root and any vertex v_i, i.e., $d_{max} = \max_i d_i$.

We now divide the proof in two parts, which we call Claim 1 and Claim 2. The first part of the proof (i.e., Claim 1) consists in showing that the errors δ_i and $\varepsilon_{i,t}$ are equal when $v_i \in S_k$ and $t = k$, which is the time at which the parameters get updated. Particularly:

Claim 1: At any fixed time t, we have $\varepsilon_{i,t} = \delta_i$ for every $v_i \in S_t$.

We prove this claim by induction on d_{max}. Let us start with the *basic step* $d_{max} = 1$:

We have the output vertex v_{out} and leaf vertices. The value $\mu_{out,t}$ of the output node is given by the elementary function g_{out} defined on all the parameters. Hence, we have

$$\delta_i = \varepsilon_{i,0} = \mu_{out,t} - y. \tag{18}$$

This proves the basic case. Now we move to the *induction step*: let us assume that Claim 1 holds for every computation graph with where $d_{max} = m$.

Let $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$ be a function whose computation graph $G(V, E)$ has $d_{max} = m + 1$. For every non-leaf node v_i such that $d_i < m$ and $v_i \in S_t$, we have that $\delta_i = \varepsilon_{i,t}$. Furthermore, note that $\varepsilon_{i,t} = \varepsilon_{i,d_i}$.

$$\varepsilon_{i,d_i} = \sum_{j \in P(i)} \varepsilon_{j,d_i-1} \frac{\partial \mu_{j,d_i}}{\partial x_{i,0}} \qquad \text{by Lemma A.1,}$$

$$\delta_i = \sum_{j \in P(i)} \delta_j \frac{\partial \mu_j}{\partial \mu_i} \qquad \text{by Eq. (4).}$$

The two quantities above are equal. This follows from the induction step, which gives $\varepsilon_{j,d_i-1} = \delta_i$ and from the condition that states that $\mu_{i,t} = x_{i,0}$ for $d < d_i$. This concludes the proof of Claim 1.

Claim 2**:** We have $\Delta z_i = \Delta \zeta_i$ for every $i \leq n$.

Eqs. (3) and (8) state the following:

$$\Delta z_i = \alpha \cdot \sum_{j \in P(i)} \delta_j \frac{\partial \mu_j}{\partial z_i},$$

$$\Delta \zeta_i = \alpha \sum_{j \in P(i)} \varepsilon_{j,t} \frac{\partial \mu_{j,t}}{\partial \zeta_i}.$$

The update of every parameter ζ_i in Z-IL happens at $t = d_i$. Claim 1 shows that, at that specific time, we have $\delta_i = \varepsilon_{i,t}$, while Lemma A.2 states that $\mu_{i,t} = \mu_{i,0}$ for every $t \leq d_i$. The proof of the claim, and hence, the whole theorem, follows from $\zeta_i = z_i$ for every $i \leq n$. □

Lemma A.1. *Let $\bar{\zeta}$ be an input of a continuous and differentiable function $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$ with computational graph $G(V, E)$, and also assume that the update $\Delta\bar{\zeta}$ using Z-IL with the partition of V described by Eq.* (13)*, we then have $\mu_{i,t} = \mu_{i,0}$ and $\varepsilon_{i,t} = 0$ for every $t \leq d_i$.*

Proof. This directly follows from the fact that we are applying Z-IL on a levelled graph. In fact, the value μ_{i,d_i} of every vertex v_i differs from its initial state $\mu_{i,0}$ only if the node values $\{x_{j,t}\}_{j \in C(i)}$ of the children vertices have changed in the time interval $[0, d_i]$. This may only happen if we have $d_j < d_i$ for one of the vertices $\{v_j\}_{j \in C(i)}$. But this is impossible, as the distance from the top d_i of a parent node is always strictly smaller than the one of any of its children nodes in a levelled graph. □

Lemma A.2. *The prediction error in Z-IL at $t = d_i$, i.e., $\varepsilon_{i,t}$, can be derived from itself at previous inference moments. Formally,*

$$\varepsilon_{i,d_i} = \gamma \sum_{j \in P(i)} \varepsilon_{j,d_i-1} \frac{\partial \mu_{j,d_i}}{\partial x_{i,0}}. \tag{19}$$

Proof. Let us write $\varepsilon_{i,t}$ as a function of $\varepsilon_{i,t-1}$:

$$\varepsilon_{i,t} = \varepsilon_{i,t-1} + (\Delta x_{i,t-1} - \Delta \mu_{i,t-1}), \tag{20}$$

where $\Delta \mu_{i,t-1} = \mu_{i,t} - \mu_{i,t-1}$. Then, we expand ε_{i,d_i} with the above equation and simplify it with Lemma A.1, i.e., $\varepsilon_{i,d_i-1} = 0$ and $\Delta \mu_{i,t<d_i-1} = 0$:

$$\varepsilon_{i,d_i} = \varepsilon_{i,d_i-1} + (\Delta x_{i,d_i-1} - \Delta \mu_{i,d_i-1}) = \Delta x_{i,d_i-1}. \tag{21}$$

We further investigate $\Delta x_{i,d_i-1}$ expanded with the inference dynamic Eq. (7) and simplify it with Lemma A.1, i.e., $\varepsilon_{i,t<d_i} = 0$,

$$\Delta x_{i,d_i-1} = \gamma(\varepsilon_{i,d_i-1} + \textstyle\sum_{j \in P(i)} \varepsilon_{j,d_i-1} \dfrac{\partial \mu_j}{\partial x_{i,d_i-1}}) \tag{22}$$

$$= \gamma \textstyle\sum_{j \in P(i)} \varepsilon_{j,d_i-1} \dfrac{\partial \mu_j}{\partial x_{i,d_i-1}}. \tag{23}$$

Putting Eq. (23) into Eq. (21), we obtain:

$$\varepsilon_{i,d_i} = \gamma \sum_{j \in P(i)} \varepsilon_{j,d_i-1} \frac{\partial \mu_{j,d_i}}{\partial x_{i,d_i-1}} \tag{24}$$

$$= \gamma \sum_{j \in P(i)} \varepsilon_{j,d_i-1} \frac{\partial \mu_{j,d_i}}{\partial x_{i,0}}. \tag{25}$$

With Lemma A.1, x_{i,d_i-1} can be replaced with $x_{i,0}$. □

C Empirical Validation of the Theorems

Table 3: Euclidean distance of the weights after one training step of Z-IL (and variations), and BP.

Model	Z-IL	Z-IL without Layer-dependent Update	Z-IL with $\varepsilon^l_{i,0} \neq 0$	Z-IL with $\gamma = 0.5$
ANN	0	1.42×10^2	7.22	8.67×10^4
RNN	0	6.05×10^3	9.60	6.91×10^5
CNN	0	5.93×10^5	7.93×10^2	9.87×10^8
ResNet	0	9.43×10^7	4.53×10^5	6.44×10^9
Transformer	0	1.12×10^{11}	3.41×10^6	8.63×10^{16}

To empirically validate the results of our theorems, we show that all the conditions of Z-IL are needed to obtain exact backpropagation. Particularly, by starting from the same weight initialization, we have conducted one training step of the following five different learning algorithms: (i) BP, (ii) Z-IL, (iii) Z-IL without level-dependent update, (iv) Z-IL with $\varepsilon^l_{i,0} \neq 0$, and (v) Z-IL with $\gamma = 0.5$. Note that the last three algorithms are the variations of Z-IL obtained ablating each one of the initial conditions.

After conducting one training step of each algorithm, we have computed the Euclidean distance between the weights obtained by one of the algorithms (ii)-(v), and the ones obtained by BP. The results of these experiments, reported in Table 3, show that all the three conditions of Z-IL are necessary in order to achieve zero divergence with BP. To provide full evidence of the validation of our theoretical results, we have conducted this experiment using ANNs, CNNs, RNNs, ResNets, and Transformer networks. Further details about the experiments can be found in the section below, although they are similar to the ones shown in the main body.

D Reproducibility of the Experiments

In this section, we provide the details of all the experiments shown in Section 5 and C.

ANNs: To perform our experiments with fully connected networks, we have trained three architectures with different depth on FashionMNIST. Particularly, these networks have a hidden dimension of 128 neurons, and $2, 3$, and 4 layers, respectively. We have used a batch of 20 training points, and a learning rate of 0.01. The numbers reported for the experiments are the averages over the three architectures.

CNNs: For our experiments on CNNs, we have used AlexNet trained on both FashionMNIST and ImageNet. As above, we have used a batch of 20 training points, a learning rate of 0.01, and reported the average of the experiments over the two datasets.

RNNs: We have trained a reinforcement learning agent on a single-layer many-to-one RNN, with $n = n^{out} = 128$, on eight different Atari games. Batch size and learning rate are 32 and 0.001, respectively. Again, the reported results are the average of all the experiments performed on this architecture.

ResNets: We have used a 5-layers fully connected network with 256 hidden neurons per layer. The residual connections are defined at every layer. Particularly, we have defined it in a way that allows its computational graph to be levelled.

Transformer: We have used a single-layer transformed architecture, trained on randomly generated data.

All experiments are conducted on 2 Nvidia GeForce GTX 1080Ti GPUs and 8 Intel Core i7 CPUs, with 32 GB RAM. Furthermore, to avoid rounding errors, we have initialized the weights in *float32*, and then transformed them in *float64*, and all later computations are in *float64*.