Reverse Differentiation via Predictive Coding

Tommaso Salvatori [1], **Yuhang Song** [1,2 *], **Zhenghua Xu** [3], **Thomas Lukasiewicz** [1], **Rafal Bogacz** [2]

[1] Department of Computer Science, University of Oxford, UK
{tommaso.salvatori, yuhang.song, thomas.lukasiewicz}@cs.ox.ac.uk
[2] MRC Brain Network Dynamics Unit, University of Oxford, UK; rafal.bogacz@ndcn.ox.ac.uk
[3] State Key Laboratory of Reliability and Intelligence of Electrical Equipment,
Hebei University of Technology, Tianjin, China; zhenghua.xu@hebut.edu.cn

Abstract

Deep learning has redefined AI thanks to the rise of artificial neural networks, which are inspired by neurological networks 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 a plethora of applications. Neural 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 it on computational graphs, and show that it can perform exact reverse differentiation. What results is the first PC (and so biologically plausible) algorithm that is equivalent to BP in the way of updating parameters on any neural network, providing a bridge between the interdisciplinary research of neuroscience and deep learning. Furthermore, the above results in particular also immediately provide a novel local and parallel implementation of BP.

Introduction

In recent years, neural networks have achieved amazing results in multiple fields, such as image recognition (He et al. 2016; Krizhevsky, Sutskever, and Hinton 2012), natural language processing (Vaswani et al. 2017; Devlin et al. 2019), and game playing (Silver et al. 2017, 2016). 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 (Hornik, Stinchcombe, and White 1989) 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) (Hochreiter and Schmidhuber 1997), transformers (Vaswani et al. 2017), convolutional neural networks (CNNs), and residual neural networks (He et al. 2016). 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 in 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*) (Rumelhart, Hinton, and Williams 1986). While being a milestone of the field, this algorithm has often been considered biologically implausible, as it does not follow the rules of biological networks in the brain to update the parameters and propagate information (Crick 1989). Here, we use the term "biologically plausible" to refer to models that satisfy a list of minimal properties required by a possible neural implementation, namely, local computations and local plasticity (change in a connection weight depending only on the activity of the connected neurons) (Whittington and Bogacz 2017).

An influential model of information processing in the brain, called *predictive coding* (*PC*) (Rao and Ballard 1999), is used to describe learning in the brain, and has promising theoretical interpretations, such as the minimization of free energy (Bogacz 2017; Friston 2003, 2005; Whittington and Bogacz 2019) and probabilistic models (Whittington and Bogacz 2017). Originally proposed to solve unsupervised learning tasks, PC has been found to be successful also in supervised models (Whittington and Bogacz 2017), and its standard implementation, called *inference learning* (*IL*) (Whittington and Bogacz 2017), has also been shown to be able to approximate asymptotically BP on MLPs, and on any other complex model (Millidge, Tschantz, and Buckley 2020). Furthermore, a recent work has proved that PC can do exact BP on MLPs using a learning algorithm

*Corresponding author

called *zero-divergence inference learning* (*Z-IL*) (Song et al. 2020). Z-IL is a biologically plausible method with local connections and local plasticity. While this exactness result is thrilling and promising, Z-IL has limited generality, as it has only been shown to hold for MLPs. Actually, a recent study shows that there are no published successful methods to train high-performing deep neural networks on difficult tasks (e.g., ImageNet classification) using any algorithm other than BP (Lillicrap et al. 2020). 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 (Whittington and Bogacz 2017; Millidge, Tschantz, and Buckley 2020), and an exactness result (Z-IL), only proven for MLPs.

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 compare the two weight updates by computing the Euclidean distance. The results, reported in Table 1 below, show two interesting things: first, they suggest that the exactness result holds for CNNs and many-to-one RNNs; second, they show that it does not hold 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. In CNNs and RNNs, this does not happen. The main contributions of this paper are briefly summarized as follows.

- We show that Z-IL is also able to exactly implement BP on CNNs and RNNs. Particularly, we give a direct derivation of the equations, and extend the proof of the original formulation of Z-IL on MLPs to CNNs and RNNs.
- We then generalize IL (and Z-IL) to work for every computational graph, and so any neural network. We also propose a variant of Z-IL that is directly defined on computational graphs, which we prove to be equivalent to BP in the way of updating parameters on any neural network.
- This results into a novel local and parallel implementation of BP. We experimentally analyze the running time of Z-IL, IL, and BP on different architectures. The experiments show that Z-IL is comparable to BP in terms of efficiency, and several orders of magnitude faster than IL.
- There are other impacts on machine learning beyond the above. In particular, the above novel formulation of BP in terms of IL may inspire other neuroscience-based alternatives to BP. Furthermore, deep-learning-based approaches may actually be closer related to information processing in the brain than commonly thought.
- At the same time, the first biologically plausible algorithm that exactly replicates the weight updates of BP on mapping functions of complex models may have a similarly big impact in neuroscience, as it shows that deep learning is actually highly relevant in neuroscience.

Computational Graphs

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) that represents a complex function $\mathcal{G}$ as a composition of elementary functions. Every internal vertex v_i is associated with one elementary function g_i, and represents the computational step expressed by g_i. Every edge pointing to this vertex represents an input of g_i. For ease of presentation, the direction considered when using this notation is the reverse pass (downwards arrows in Fig. 1). Furthermore, we call $e_{i,j} \in E$ the directed edge that starts at v_i and ends at v_j. The first n vertices $v_1, \dots, v_n$ are the leaves of the graph and represent the n inputs of the function $\mathcal{G}$, 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}). 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, 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 child and parent vertices of v_i, respectively. Hence, input nodes (nodes at the bottom) have no child vertices, and output nodes (nodes at the top) have no parent vertices. We now briefly recall reverse differentiation, and so BP, on computational graphs, and we then newly define how to perform PC on computational graphs.

BP on Computational Graphs

Let $\mathcal{G}\colon \mathbb{R}^n \to \mathbb{R}$ 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 $g_i\colon \mathbb{R}^{k_i} \to \mathbb{R}$, 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 an input vector $\bar{z} \in \mathbb{R}^n$, we denote by μ_i the value of the vertex v_i during the forward pass. This value is computed iteratively as follows:

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

We then have $\mathcal{G}(\bar{z}) = \mu_{|V|} = \mu_{out}$. The computational flow just described is represented by the upward 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 an input (which in the case of MLPs will correspond to the weight parameters on the basis of which the output of the network is computed, as we will explain in the next section), and $\mathcal{G}(\bar{z}) = \mu_{out}$ be the output. Reverse differentiation is a key technique in machine learning and AI, as it allows to compute $\partial\mathcal{G}/\partial z_i$ for every $i < n$ efficiently. This is necessary to implement BP at a reasonable computational cost, espe-

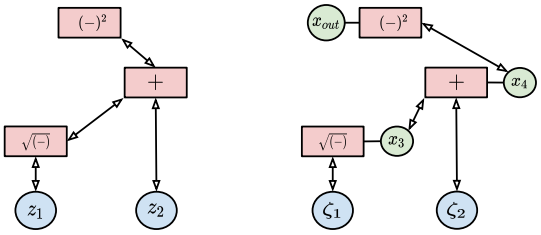

Figure 1: Left: computational graph of the function $\mathcal{G}(z_1, z_2) = (\sqrt{z_1} + z_2)^2$. Every internal vertex (red box) pictures its associated function g_i. Right: its predictive coding counterpart. Pointed upwards, the arrows related to the feedforward pass. The value nodes x_i of the input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

cially considering the extremely overparametrized architectures used today. This is done iteratively as follows:

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

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

Update of the leaf nodes: Given an input $\bar{z}$, we consider a desired output y for the function $\mathcal{G}$. The goal of a learning algorithm is to update the input parameters $(z_1, \ldots, z_n)$ of a computational graph to minimize the quadratic loss $E = \frac{1}{2}(\mu_{out} - y)^2$. Hence, the input parameters are updated by:

$$\Delta z_i = -\alpha \cdot \frac{\partial E}{\partial z_i} = \alpha \cdot \sum\nolimits_{j \in P(i)} \delta_j \cdot \frac{\partial g_j}{\partial z_i}, \quad (3)$$

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:

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

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 (Millidge, Tschantz, and Buckley 2020). 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 $\varepsilon_{i,t}$. We denote a parameter vector (which for MLPs corresponds to weights) by $(\zeta_1, \ldots, \zeta_n)$, so ζ_i in IL corresponds to z_i in BP, but we use different symbols, as they may not be necessarily equal to each other. 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

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

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 PC networks, every computation is strictly local. The value nodes of the network are updated continuously to minimize the following loss function, defined on all the vertices of G:

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

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 gives:

$$\Delta x_{i,t} = -\gamma \cdot \frac{\partial F_t}{\partial x_{i,t}} = \gamma \cdot (\varepsilon_{i,t} + \sum\nolimits_{j \in P(i)} \varepsilon_{j,t} \cdot \frac{\partial \mu_{j,t}}{\partial x_{i,t}}). \quad (7)$$

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:

$$\Delta \zeta_i = -\alpha \cdot \frac{\partial F_t}{\partial \zeta_i} = \alpha \cdot \sum\nolimits_{j \in P(i)} \varepsilon_{j,t} \cdot \frac{\partial \mu_{j,t}}{\partial \zeta_i}. \quad (8)$$

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 $\mathcal{G}$ into 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 so of the parameters. However, it has been shown that, while following different dynamics, these updates are asymptotically equivalent (Millidge, Tschantz, and Buckley 2020).

Z-IL for 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), the PC equivalent of MLPs. Particularly, this result states that starting from a PCN and a MLP 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. We now provide a brief description of the original Z-IL algorithm. To be as close as possible to the original formulation of Z-IL, we adopt the same notation of that work, and

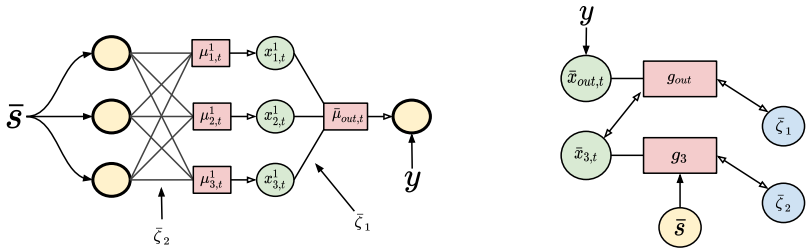

Figure 2: Left: example of a 2-layer PCN. In these networks, it is possible to realize every computation locally using error nodes and value nodes in a biologically plausible way. For a more detailed discussion, we refer to (Whittington and Bogacz 2017). Right: the corresponding computational graph.

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

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

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), represented in Fig. 2. The leaf vertices of its computational graph are the weight matrices, represented by the blue nodes in Fig. 2. Every weight matrix $\bar{\zeta}_l$ has the distance l from the output vertex.

This new algorithm differs from standard inference learning for the following reasons:

1. The initial error $\varepsilon_{i,0}$ of every vertex v_i is set to zero. This is done by performing a forward pass from an input vector $\bar{s}$ and setting $\mu_{i,0} = x_{i,0}$ for every vertex v_i.
2. The weight parameters ζ_l of layer l get only updated at time step $t = l$, making the inference phase only last for L iterations.

Update of the leaf nodes: As stated, Z-IL introduces a new rule to update the weights of a fully connected PCN. Using the notation adopted for computational graphs, every leaf node $\bar{\zeta}_l$ in Fig. 2 gets updated at $t = l$. Alg. 1 shows how Z-IL performs a single update of 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 (Song et al. 2020). The main theoretical result is as follows, formally stating that the update rules of BP and Z-IL are equivalent in MLPs.

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

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

for every layer $l \geq 0$.

Table 1: Divergence between one update of weights of BP and Z-IL on different models, initialized in the same way.

	MLP	CNNs	RNNs	ResNet18	Transformer
Divergence:	0	0	0	4.53×10^7	7.29×10^4

Z-IL for CNNs and RNNs

CNNs are a neural architecture that is highly used in computer vision, with a connectivity pattern that resembles the structure of animals' visual cortex. The parameters of a convolutional layer are contained in different *kernels*, vectors that act on the input pattern via an operation called *convolution*. Many-to-one RNNs, on the other hand, deal with sequential inputs, and consist of three different weight matrices: two are used recursively for the inputs and hidden layers, and the last one is the output layer.

While Theorem 1 has only been proven for MLPs, the experimental results presented in Table 1 suggest that the original formulation of Z-IL is also able to exactly replicate the weight update of BP on CNNs and RNNs. Inspired by our empirical findings, we prove that the update rules of BP and Z-IL are equivalent in convolutional and recurrent networks, generalizing the result of Theorem 1 to CNNs and RNNs:

Theorem 2. *Let M be a convolutional or a recurrent PCN trained with Z-IL, and let M' be its corresponding model, initialized as M, and trained with BP. Then, given the same data point to both networks, the update of all parameters performed by Z-IL on M is equivalent to that of BP on M'.*

The experimental results presented in Table 1, however, show that the original definition of Z-IL does *not* generalize to more complex architectures. In what follows, we solve this problem by defining Z-IL directly on computational graphs, and prove a generalization of Theorems 1 and 2.

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

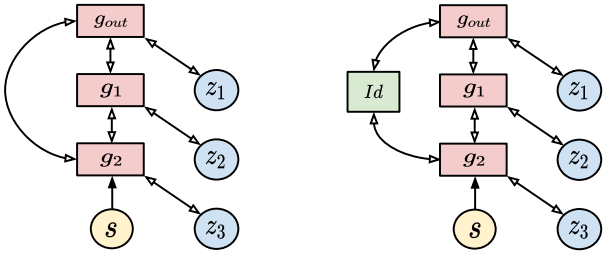

Figure 3: Left: computational graph of a 3-layer MLP with a residual connection, corresponding to the function $\mathcal{G}(s, \bar{z}) = sz_3 + sz_3z_2z_1$. Right: an equivalent computational graph, with the addiction of an identity node.

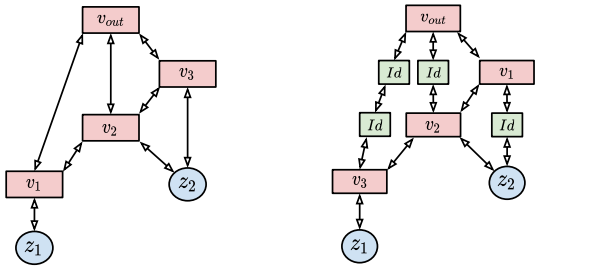

Figure 4: Computational graphs of the same function $\mathcal{G}$. Left: the original graph G. Right: the transformed graph, with the identity vertices in green.

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. Consider the following function, corresponding to a simple MLP with a skip connection, represented in Fig. 3, left side:

$$\mathcal{G}(s, \bar{z}) = sz_3 + sz_3z_2z_1. \tag{10}$$

BP: Given an input value 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:

$$\Delta z_3 = -\alpha \cdot \frac{\partial E}{\partial z_3} = \alpha \cdot \delta(z_1z_2 + 1)s, \tag{11}$$

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

Z-IL: Given an input value 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:

$$\Delta\zeta_3 = -\alpha \cdot \frac{\partial F_3}{\partial \zeta_3} = \alpha \cdot \delta\zeta_1\zeta_2 s, \tag{12}$$

where $\delta = \varepsilon_{out,0} = (\mu_{out,0} - y)$, and F_2 is computed according to 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.

Identity Vertices

The error signal propagated by the inference 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 the original 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. (11). 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. Here, the error reaches every vertex at a single, specific time step, no matter how complex the graph structure is. We now show how to make every computational graph levelled, without affecting the underlying function and the computations of the derivatives.

Every elementary function g_i can be written as a composition with the identity function, i.e., $g_i \circ Id$. Given two vertices v_i and v_j connected via the edge $e_{i,j}$, it is then possible to add a new vertex v_k by splitting the edge $e_{i,j}$ into $e_{i,k}$ and $e_{k,j}$, 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. However, placing the identity vertices in the correct 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, right side, 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:

$$\Delta\zeta_3 = -\alpha \cdot \frac{\partial F_3}{\partial \zeta_3} = \alpha \cdot \delta(\zeta_1\zeta_2 + 1)s. \tag{13}$$

If we have $\zeta_i = z_i$, this weight update is equivalent to the one performed by BP and expressed in Eq. (11). Hence, Z-IL is able to produce the same weight update of BP in a simple neural network with one skip connection, thanks to a single identity vertex. In the next section, we generalize this result.

Levelled Computational Graphs

In this section, we show that, given any computational graph, it is always possible to generate an equivalent, levelled version of it. 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: given any function $\mathcal{G}$, it is always possible to consider an equivalent, levelled, computational graph. This allows to partition the nodes of G in 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 be a computational graph, and $S_1, \ldots, S_K$ 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.,

$$S_k = \{v_i \in V \,|\, \exists \text{ a path } (e_{out,j_1}, \ldots, e_{j_{k-1},i})\}. \tag{14}$$

Algorithm 2: Generating a levelled DAG G' from G

Require: G is a DAG, and $(v_0, \dots, v_n)$ 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

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, and this partition defines its level structure. Let D_i be 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 via the addition of identity nodes.

Let G be a DAG with root v_0, and let $(v_0, v_1, \dots, v_n)$ 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:

$$v_i \to Id \to \cdots \to Id \to v_j, \tag{15}$$

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. This is equivalent to having every $v_i \in G$ that belongs to one and only one subset S_k, as every pair of disconnected paths between two vertices has the same length, thanks to the addition of identity vertices. Hence:

Theorem 3. *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.*

The above theorem shows that every neural network can be expressed as a levelled computational graph, and hence that every result shown for levelled computational graphs can be naturally extended to every possible neural network.

Z-IL for Levelled Computational Graphs

In this section, we show that a generalized version of Z-IL allows PCNs to do exact BP on any computational graph.

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 $S_1, \dots, S_K$. We now present a variation of IL for computational graphs that allows predictive coding to exactly replicate the parameter update of BP, called Z-IL for computational graphs. This algorithm is similar to IL, but the following two differences are introduced:

Forward pass: Differently from IL, where input and output are presented simultaneously, Z-IL first presents the input vector to the function, and performs a forward pass. Then, once the values μ_i of all the internal vertices have been computed, the 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 behaviour of BP, which first computes the output vector, and then compares it to the label.

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)
    Update each leaf node $\zeta_{i,t} \in S_t$ to minimize F_t via Eq. (8)
  end for
end for

Update of the leaf nodes: Instead of continuously running 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 $v_i \in S_t$, if any. More formally, for every internal vertex v_i, training continues as usual via Eq. (7), while leaf nodes are updated according to the following equation:

$$\Delta\zeta_{i,t} = \begin{cases} \gamma \cdot \sum_{j \in P(i)} \varepsilon_{j,t} \cdot \frac{\partial \mu_j}{\partial \zeta_i} & \text{if } v_i \in S_t \\ 0 & \text{if } v_i \notin S_t. \end{cases} \tag{16}$$

This shows that one full update of the parameters requires $t = K$ steps. Note that for multilayer networks, K is equal to the number of layers L. Overall, the functioning of Z-IL for computational graphs is summarized in Algorithm 3. 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 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 $\Delta\bar{\zeta}$ using Z-IL with $\gamma = 1$. Then, if $\bar{z} = \bar{\zeta}$, and we consider a levelled computational graph of $\mathcal{G}$, we have*

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

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. Particularly, adding identity nodes to the computational graphs to produce equivalence to BP has non-trivial implications: it shows that the key difference between the PC model of learning in the brain and BP lies in the synchronization of error propagation. This offers a novel perspective to investigate the gap between BP and neural models.

Experiments

In the above sections, we have theoretically proved that the proposed generalized version of Z-IL is equivalent to BP on every possible neural model. Multiple experiments, reported in the supplementary material, further confirmed this: the divergences of weight updating between BP and Z-IL are always zero on all tested neural networks. So, there is no need for detailed experimental evaluation for the equivalence. In this section, we will complete the picture of this

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

Method	MLP	AlexNet (Krizhevsky, Sutskever, and Hinton 2012)	RNN	ResNet18 (He et al. 2016)	Transformer (Vaswani et al. 2017)
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

work with experimental studies to evaluate the computational efficiency of Z-IL, and quantitatively compare it with those of BP and IL. Particularly, we perform extensive experiments on different architectures, testing multiple models per architecture. The results of BP, IL, and Z-IL, averaged over all the experiments per model, are reported in Table 2, and a detailed description of the experiments, as well as all the parameters needed to reproduce the results, are provided in the supplementary material.

Results and Evaluations

As shown in Table 2, the computational time of Z-IL is very close to that of BP, and orders of magnitude lower than that of IL. This proves that Z-IL is an efficient alternative to BP in practice, instead of just being a theoretical tool. The high computational time of IL is due to the large number of iterations T. For example, for small MLPs, T is set to 20 in (Whittington and Bogacz 2017), and as larger models require higher numbers of iterations to converge, T is set between 100 and 200 for mid-size architectures, such as RNNs and CNNs in (Millidge, Tschantz, and Buckley 2020). Note that the approximation results of these works are achieved with fixed values of T, and not at convergence. 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.

Related Work

PC is an influential theory of cortical function in theoretical and computational neuroscience, as it provides a computational framework, able to describe information processing in multiple brain areas (Friston 2005). It has appealing theoretical interpretations, such as free-energy minimization (Bogacz 2017; Friston 2003, 2005) and variational inference of probabilistic models (Whittington and Bogacz 2017). There are also variants of PC developed into different biologically plausible process theories specifying cortical microcircuits that potentially implement such theories (Bastos et al. 2012; Kanai et al. 2015; Shipp 2016). Moreover, the central role of top-down predictions is consistent with the ubiquity and importance of top-down diffuse connections between cortical areas. PC is then consistent with many known aspects of neurophysiology, and has been translated into biologically plausible process theories which specify potential cortical microcircuits which could implement the algorithm. Due to this solid biological grounding, PC is also attracting interest in machine learning recently, especially focusing on finding the links between PC and BP (Whittington and Bogacz 2017).

Biologically plausible approximations to BP have been intensively studied, because on the one hand, the underlying principles of BP are unrealistic for an implementation in the brain (Crick 1989; Lillicrap et al. 2016, 2020), but on the other hand, BP outperforms all alternative discovered frameworks (Baldi and Sadowski 2016). However, earlier biologically plausible approximations to BP have not been shown to scale to complex problems, such as learning colored images (Lillicrap et al. 2016; O'Reilly 1996; Körding and König 2001; Bengio 2014; Lee et al. 2015; Nøkland 2016; Scellier and Bengio 2017; Scellier et al. 2018; Lin and Tang 2018; Illing, Gerstner, and Brea 2019). More recent works show the capacity of scaling up biologically plausible approximations to the level of BP (Xiao et al. 2018; Obeid, Ramambason, and Pehlevan 2019; Nøkland and Eidnes 2019; Amit 2019; Aljadeff et al. 2019; Akrout et al. 2019; Wang, Lin, and Dang 2020). 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 (Lee et al. 2015; Whittington and Bogacz 2017; Nøkland and Eidnes 2019; Ororbia et al. 2017; Millidge, Tschantz, and Buckley 2020) demonstrate that they approximate BP, or are equivalent to BP under unrealistic restrictions (Xie and Seung 2003; Sacramento et al. 2018).

Summary and Outlook

The gap between machine learning and neuroscience is currently opening up: on the one hand, recent neural architectures trained by BP are invented with impressive performance in machine learning; on the other hand, models in neuroscience can only match the performance of BP in small-scale problems. There is thus a crucial open question of whether the advanced architectures in machine learning are actually relevant for neuroscientists. In this paper, we show that all these advanced architectures can be trained with one of their neural models: the proposed generalization of Z-IL is always equivalent to BP, with no extra restriction on the mapping function and the type of neural networks. (Previous works only showed that IL approximates BP in single-step weight updates under unrealistic and non-trivial requirements.) Also, the computational efficiency of Z-IL is comparable to that of BP, and is several orders of magnitude better than IL. Hence, we obtain a novel local and parallel implementation of BP. Moreover, the novel formulation of BP in terms of IL may inspire other neuroscience-based alternatives to BP. The exploration of such alternatives to BP are a topic of our ongoing research. Furthermore, our results show that deep-learning-based models may actually be more closely related to information processing in the brain than commonly thought, which may have a big impact on both the machine learning and the neuroscience community.

Acknowledgments

This work was supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1, by the AXA Research Fund, by the EPSRC grant EP/R013667/1, and by the EU TAILOR grant. We also acknowledge the use of the EPSRC-funded Tier 2 facility JADE (EP/P020275/1) and GPU computing support by Scan Computers International Ltd. This work was also supported by the China Scholarship Council under the State Scholarship Fund, by the UK Medical Research Council under the grant MC_UU_00003/1, by the National Natural Science Foundation of China under the grant 61906063, by the Natural Science Foundation of Hebei Province, China, under the grant F2021202064, by the Natural Science Foundation of Tianjin City, China, under the grant 19JCQNJC00400, and by the "100 Talents Plan" of Hebei Province, China, under the grant E2019050017.

References

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

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

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

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

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

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

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

Crick, F. 1989. The recent excitement about neural networks. *Nature*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. 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.*

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

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

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

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

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

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

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

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

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

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

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

Lillicrap, T. P.; Cownden, D.; Tweed, D. B.; and Akerman, C. J. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1): 1–10.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Song, Y.; Lukasiewicz, T.; Xu, Z.; and Bogacz, R. 2020. 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*.

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

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

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

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

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

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