Towards Scaling Difference Target Propagation by Learning Backprop Targets

Maxence Ernoult [1] **Fabrice Normandin** [*2] **Abhinav Moudgil** [*2 3] **Sean Spinney** [2 4] **Eugene Belilovsky** [2 3]
Irina Rish [2 4] **Blake Richards** [2 5 6] **Yoshua Bengio** [2 4]

Abstract

The development of biologically-plausible learning algorithms is important for understanding learning in the brain, but most of them fail to scale-up to real-world tasks, limiting their potential as explanations for learning by real brains. As such, it is important to explore learning algorithms that come with strong theoretical guarantees and can match the performance of backpropagation (BP) on complex tasks. One such algorithm is Difference Target Propagation (DTP), a biologically-plausible learning algorithm whose close relation with Gauss-Newton (GN) optimization has been recently established. However, the conditions under which this connection rigorously holds preclude layer-wise training of the feedback pathway synaptic weights (which is more biologically plausible). Moreover, good alignment between DTP weight updates and loss gradients is only loosely guaranteed and under very specific conditions for the architecture being trained. In this paper, we propose a novel feedback weight training scheme that ensures both that DTP approximates BP and that layer-wise feedback weight training can be restored without sacrificing any theoretical guarantees. Our theory is corroborated by experimental results and we report the best performance ever achieved by DTP on CIFAR-10 and ImageNet 32×32.

1. Introduction

Although artificial neural networks were originally inspired by the brain, the strict implementation of the backpropagation algorithm (BP) violates biological constraints, and no known biologically plausible candidate algorithm can match its performance on challenging tasks. Conversely, bridging this gap could bring a better understanding of biological learning (Richards et al., 2019). Recent efforts towards this goal suggest that it could be achieved by developing learning algorithms that relax the requirements of BP while preserving strong theoretical guarantees.

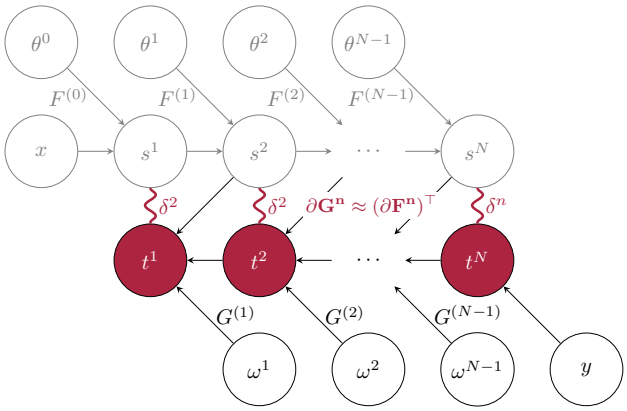

Figure 1. Computational graph of the feedforward pathway $\mathcal{F}$ (on the left, shaded) with input x and associated DTP feedback pathway with ground-truth label y (right). The *targets* t^n (purple nodes) are forward-propagated through the G^{n+1} operator whose Jacobian has been made to approximately match that of the transpose of F^n. This way, the resulting local activation differences $\delta^n \propto t^n - s^n$ encode backprop error signals. We thus learn to estimate layer-wise *backprop* targets.

Target Propagation (TP) (LeCun, 1987) and its *Difference Target Propagation* (DTP) variants (Bengio, 2014; Lee et al., 2015; Bartunov et al., 2018; Ororbia & Mali, 2019; Bengio, 2020; Meulemans et al., 2020) constitute a family of such algorithms which, from the biological prospective, sidesteps two issues of BP. Most importantly, TP computes error signals in feedforward architectures by propagating *target values* for the neurons rather than error gradients, thereby aligning better with our understanding of what feedback pathways in the brain communicate (Lillicrap et al., 2020). A major consequence of handling neural activation targets across all layers is that feedforward weights can be updated in a fully local fashion to push neural activations closer to

[*]Equal contribution [1]IBM Research, Paris. Work done during a remote internship at Mila. [2]Mila. [3]Concordia University. [4]UdeM. [5]McGill University. [6]Montreal Neurological Institute. Correspondence to: Maxence Ernoult <maxence.ernoult@ibm.com>, Yoshua Bengio <yoshua.bengio@mila.quebec>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

their target values. Second, TP routes those targets through a distinct set of feedback weights rather than transporting the weights from the feedforward pathway (Lillicrap et al., 2016). Rather than being fixed throughout learning, though, these feedback weights learn to *invert* the feedforward pathway. But what would it take to learn to *backprop* the feedforward pathway? This is the central question addressed by the present work.

Nevertheless, TP algorithms have yet to scale to complex tasks, and as such, they do not yet stand as a compelling biological learning model. Recent work highlighting the connection between TP and Gauss-Newton (GN) optimization (Bengio, 2020; Meulemans et al., 2020) has incentivized to revisit the scalability of TP algorithms (Bartunov et al., 2018). More precisely, Meulemans et al. (2020) demonstrate that while TP neural activations updates emulates GN optimization in invertible neural networks, this connection can be maintained with DTP on non-invertible networks if the feedback weights training scheme is changed accordingly. Indeed, to emulate GN optimization, as the pseudo-inverse of the whole feedforward pathway does not factorize as the product of each feedforward module's pseudo-inverse, each feedback module should capture the pseudo-inverse of the *whole* downstream feedforward pathway: when computing the resulting Difference Reconstruction Loss (DRL), noisy perturbations subsequently need to be propagated all the way up to the output layer. However, their approach still has limitations from a biological learning perspective. First, enforcing GN optimization in DTP like this precludes *layer-wise* feedback weights training and instead calls for the use of direct connections in the feedback pathway: this topological restriction seriously compromises biological plausibility. Second, the resulting optimization algorithm used to update the feedforward weights is a hybrid between between gradient descent and GN optimization. Therefore, only loose alignment between backprop and DTP updates can be accounted for by their theory and with restrictive assumptions on the architecture being trained. Finally, although that theory offers a principled way to design the architectures trained by these variants of DTP, the CIFAR-10 training experiments they report are limited to relatively shallow architectures with poor performance.

In this paper, we propose to revisit the GN interpretation of DTP by having the feedback pathway synaptic weights compute layer-wise *BP targets* rather than GN targets. To this end, we propose a novel feedback weights training scheme which, by construction, pushes the Jacobian of the feedback operator towards the transpose of its feedforward counterpart, in a layer-wise fashion and without having to use direct feedback connections in the feedback pathway. Therefore, assuming this condition holds for all the layers and keeping everything else unchanged in the DTP algorithm, the DTP feedforward weight updates closely approach those of BP. This leads us to a scalable biologically plausible approximation of BP.

More specifically, our contributions are as follows:

- We propose a novel *Local Difference Reconstruction Loss* (L-DRL) along with an algorithm to train the feedback weights which ensures that the Jacobian of the feedback pathway matches the transpose of the Jacobian of the associated feedforward pathway (Section 4.1, Theorem 4.2, Alg. 3). We call this condition the *Jacobian Matching Condition* (JMC, Definition 4.1).
- Assuming the JMC holds for a given architecture and using the standard DTP equations to propagate targets, we demonstrate that DTP feedforward weight updates approximate BP gradients (Section 4.2, Theorem 4.3). We say that such an architecture satisfies the *Gradient Matching Property* (GMP).
- We numerically demonstrate the GMP and JMC, showing that L-DRL is more efficient than DRL to align feedforward and feedback weights and that the GMP is subsequently significantly better satisfied (Section 5.1-5.2).
- Finally, we validate our novel implementation of DTP on training experiments on MNIST, Fashion MNIST, CIFAR-10 and ImageNet 32×32 (van den Oord et al., 2016) (Section 5.3). In particular, we achieve a 89.38 % accuracy on CIFAR-10 and 60.6 % top-5 accuracy on ImageNet 32×32, which are the best performances ever reported in the DTP literature on these datasets and nearly match the performance of BP on the same architectures.

2. Related Work

DTP borrows several key concepts from the biologically plausible deep learning literature. First, resorting to a distinct set of weights to route error signals in the feedback pathway as done in DTP solves a problem known as *weight transport* (Lillicrap et al., 2016). While having randomly initialized *fixed* feedback weights is sufficient to carry useful error signals on MNIST (Lillicrap et al., 2016; Nøkland, 2016), subsequent studies demonstrated it was insufficient to scale to harder tasks (Moskovitz et al., 2018; Bartunov et al., 2018; Launay et al., 2019; Crafton et al., 2019). The main approach undertaken to overcome this issue is to add extra mechanisms to promote alignment between feedforward and feedback weights (Xiao et al., 2018; Lansdell et al., 2019; Guerguiev et al., 2019; Akrout et al., 2019; Kunin et al., 2020). More specifically, many of these mechanisms are based on the idea of perturbing the feedforward activations with noise, and communicating the resulting

noisy activations in the feedback pathway to coordinate feedback and feedforward weight updates consistently (Akrout et al., 2019; Lansdell et al., 2019; Kunin et al., 2020), which constitutes a second important feature of DTP. However, a limitation of many of these algorithms is that they require gradient computation of the operations carried out in the feedforward pathway. One solution to mitigate this issue, which is the third important ingredient of DTP, is to propagate neural activation differences as implicit error signals rather than error gradients (Lillicrap et al., 2020). These error signals may typically arise from a mismatch between feedforward (bottom-up) predictions and (top-down) actual feedback (Whittington & Bogacz, 2017; Sacramento et al., 2018; Choromanska et al., 2019), as it is the case for DTP, or from a perturbation from equilibrium (Scellier & Bengio, 2017). Recent works have explored the application of DTP to recurrent neural networks (Manchev & Spratling, 2020; Roulet & Harchaoui, 2021), albeit with the implausible requirement of processing inputs backward in time during target computation, a challenge that we do not aim to address in the present paper. As emphasized in the introduction, the closest work to ours is that of (Meulemans et al., 2020), and we show the theoretical and experimental advantages of our approach.

3. Background

We first introduce the key notations and assumptions used throughout this paper.

Definition 3.1. We define a *feedforward* architecture as:

$$\mathcal{F}(x) = F^{N-1} \circ F^{N-1} \circ \cdots \circ F^0(x), \tag{1}$$

where each feedforward module $F^n(\cdot;\theta^n)$ is parametrized by its feedforward weights θ^n. Each F^n is paired with a feedback module $G^n(\cdot;\omega^n)$ with distinct weights ω^n.

Definition 3.2. We recursively define the *layers* $s^1, \dots, s^N$ of an architecture $\mathcal{F}$ defined by Definition 3.1 as:

$$\begin{cases} s^0 &= x \\ s^{n+1} &= F^n(s^n;\theta^n) \quad \forall n = 0 \cdots N-1 \end{cases} \tag{2}$$

G^n can either take as input the feedforward path activations $s^{n+1} = F^n(s^n)$ (with G^n then forming the decoder part of a kind of auto-encoder with F^n as encoder) or the backward path targets t^{n+1} produced by G^{n+1} from t^{n+2} and representing targets for s^{n+1}.

Learning setting. We study the supervised context where, given a target y, the goal is to find the forward weights θ^n which minimize a predictive loss $\mathcal{L}_{\text{pred}}(s^N, y)$.

Notations. We denote the Frobenius dot product between two matrices A and B as $\langle A, B\rangle_F \triangleq \text{Tr}\left(A \cdot B^\top\right)$. Also, we denote $\partial_x F(x_\star) = \frac{\partial F}{\partial x}(x_\star)$ the Jacobian of F with respect to x evaluated at $x_\star$. For notational simplicity, we may omit to write $x_\star$, in which case the Jacobians are implicitly evaluated on the feedforward activations.

3.1. Difference Target Propagation (DTP)

Instead of transporting the transpose Jacobian of the feedforward operators $\partial_{s^n} F^{n\top}$ to the feedback pathway, TP and variants use a separate set of parameters through the feedback operator G^n to carry targets across layers. The G^n operators are subsequently trained, layer-wise, to approximately invert their associated feedforward counterpart: $G^n \approx (F^n)^{-1}$. TP learning thus entangles feedforward *and* feedback weights training.

Forward weights training. Target values for the neurons should be such that they decrease the predictive loss $\mathcal{L}_{\text{pred}}$. For most TP algorithms, the first target is computed as:

$$t^N_\beta = s^N - \beta \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N}, \tag{3}$$

where β is a small nudging parameter. In TP, the subsequent upstream targets are *propagated* the feedback operators as $t^n_\beta = G^n(t^{n+1}_\beta;\omega^n)$. However in non-invertible feedforward networks, this results in a significant reconstruction error $s^n - G^n(s^n;\omega^n)$, which was shown to compromise learning. *Difference* Target Propagation (Lee et al., 2015) aims to solve this issue by removing this reconstruction term from the target computation:

$$t^n_\beta = G^n(t^{n+1}_\beta;\omega^n) + s^n - G^n(s^{n+1};\omega^n). \tag{4}$$

For later convenience, we denote

$$\tilde{G}(t^{n+1}, s^{n+1};\omega^n) \triangleq G^n(t^{n+1}_\beta;\omega^n) + s^n - G^n(s^{n+1};\omega^n) \tag{5}$$

the feedback operation used to propagate the targets in Eq. (4). Finally, the parameters θ^n are updated by the local loss $\mathcal{L}^n_\theta$, defined as:

$$\mathcal{L}^n_\theta = \frac{1}{2\beta}\|t^n_\beta - s^n\|^2, \tag{6}$$

where t^n_β is treated as a constant and the gradients blocked at s^{n-1}. For example, if F^n was linear, we would have the weight update $\Delta\theta^n \propto s^{n-1} \cdot (t^n_\beta - s^n)^\top$.

Feedback weights training. Both TP and DTP employ the same mechanism to train the G^n operators. First, the feedforward activations s^n get a noisy perturbation ϵ. The

resulting noisy activations s_ϵ^n perturb the next layer s_ϵ^{n+1} through F^n, which in return yields a noisy reconstruction r_ϵ^n through G^n. The feedback weights are then updated to minimize the local loss $\mathcal{L}_\omega^n$ defined as:

$$\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2, \tag{7}$$

where s^n is treated as a constant and the gradient are blocked at s^{n+1}. Assuming again a linear G^n, the resulting feedback weight update also reads in a local fashion: $\Delta\omega^n \propto s^{n+1} \cdot (r_\epsilon^n - s_\epsilon^n)^\top$.

Algorithm 1 Standard DTP feedback weight training (Lee et al., 2015)

$\epsilon \sim \mathcal{N}(0, \sigma^2)$, $s_\epsilon^n = s^n + \epsilon$
$s_\epsilon^{n+1} = F^n(s_\epsilon^n; \theta^n)$
$r_\epsilon^n = G^n(s_\epsilon^{n+1}; \omega^n)$
Update ω^n with $\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2$.

3.2. Connection between DTP and Gauss-Newton Optimization

Using Eq. (3)-(4) and sending $\beta \to 0$, note that the DTP activation updates can be conveniently defined as:

$$\delta_{\text{DTP}}^n \triangleq \lim_{\beta\to 0} \frac{t_\beta^n - s^n}{\beta} = -\left[\prod_{k=n}^{N-1} \partial_{s^{k+1}} G^k\right] \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N} \tag{8}$$

It was suggested that under some conditions, δ_{DTP}^n encoded Gauss-Newton updates (Gauss, 1877) of the layer activations with respect to the output loss function (Bengio, 2020; Meulemans et al., 2020). In *invertible* networks, i.e. assuming $(F^n)^{-1}$ exists, the Gauss-Newton update of layer s^n with respect to $\mathcal{L}_{\text{pred}}$ is:

$$\delta_{\text{GN}}^n = -\left[\partial_{s^n} \overline{F}^n\right]^{-1} \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N}, \tag{9}$$

where $\overline{F}^n = F^N \circ \cdots \circ F^n$ denotes the forward mapping from s^n to s^N. Furthermore, assuming $G^n = F^{n-1^{-1}}$ for all $n = 1 \cdots N-1$, from Eq. (8), Eq. (9) and the inverse function theorem, it can be seen that $\delta_{\text{GN}}^n = \delta_{\text{DTP}}^n$.

In *non-invertible* networks, Meulemans et al. (2020) show that with a block-diagonal approximation of the Gauss-Newton curvature matrix, the Gauss-Newton update of s^n with respect to $\mathcal{L}_{\text{pred}}$ reads:

$$\delta_{\text{GN}}^n = -\left[\partial_{s^n} \overline{F}^n\right]^\dagger \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N}, \tag{10}$$

where $A^\dagger = \lim_{\lambda\to 0} A^\top \cdot \left(A \cdot A^\top - \lambda\right)^{-1}$ denotes the *Moore-Penrose* pseudo-inverse. However, there are are two reasons why in this case we may not have $\delta_{\text{GN}}^n = \delta_{\text{DTP}}^n$. First, using Eq. (7) as a reconstruction loss, it may not hold in general that $\partial_{s^{n+1}} G^n = (\partial_{s^n} F^n)^\dagger$. Second, even assuming this condition holds, $\left[\partial_{s^n} \overline{F}^n\right]^\dagger$ generally does not factorize as $\prod_{k=n}^N \left[\partial_{s^k} F^k\right]^\dagger$. A direct consequence of this is that DTP standard layer-wise feedback weights training leads to mostly inefficient feedforward weight updates that fail to move the output layer towards its target.

Meulemans et al. (2020) show that by adapting DTP standard feedback training scheme, $\delta_{\text{GN}}^n = \delta_{\text{DTP}}^n$ can be recovered for non-invertible networks. Instead of propagating perturbated activations s_ϵ^n back and forth through F^n and G^n into the reconstructed activation r_ϵ^n to train ω^n, they prescribe sending s_ϵ^n up to $\hat{y}$ through $\overline{F}^n$, back into r_ϵ^n through $\tilde{G}^n \circ \cdots \circ \tilde{G}^N$ where $\tilde{G}^n$ (Eq. (5)) stands for the operator used for the target computation in Eq. (4). Finally, an extra noisy perturbation in the output layer $s^N + \eta$ needs to be propagated back into r_η^n. The resulting *Difference Reconstruction Loss* (DRL) to be optimized is defined as:

$$\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2 + \lambda\|r_\eta^n - s^n\|^2. \tag{11}$$

In practice though, they replace the second term by weight decay. Taking expectation of Eq. (11) and sending the noise amplitude to 0, it can be shown that minimizing $\hat{\mathcal{L}}_\omega^n$ yields the desired property: $\prod_{k=n}^{N-1} \partial_{s^{k+1}} G^k = \left(\partial_{s^n} \overline{F}^n\right)^\dagger$.

Algorithm 2 Difference Reconstruction Loss (DRL) feedback weight training (Meulemans et al., 2020)

$\epsilon \sim \mathcal{N}(0, \sigma^2)$, $s_\epsilon^n = s^n + \epsilon$
for $k = n \cdots N-1$ **do**
  $s_\epsilon^{k+1} = F^k(s_\epsilon^k; \theta^k)$
end for
$r_\epsilon^N = s_\epsilon^N$
for $k = N-1 \cdots n$ **do**
  $r_\epsilon^k = G^k(r_\epsilon^N, r_\epsilon^{k+1}; \omega^k) + s^k - G^k(s^N, s^{k+1}; \omega^k)$
end for
Update ω^n with $\hat{\mathcal{L}}_\omega^n = \frac{1}{2}\|r_\epsilon^n - s_\epsilon^n\|^2 + \lambda\omega^n$.

4. Learning Backprop Targets rather than Gauss-Newton Targets

In the spirit of Meulemans et al. (2020), we propose to adapt the feedback weight training and the reconstruction loss, but we make it so that G^n learns the *transpose Jacobian* of its associated feedforward module F^n rather than its pseudo-inverse. This way, by construction, the DTP weight updates are made to match *BP* weight updates rather than a hybrid between BP and Gauss-Newton updates. We can also avoid

the requirement of direct connections and restore layer-wise feedback weights training while preserving theoretical guarantees with respect to BP.

4.1. Feedback weights training

Definition 4.1. For a given architecture $\mathcal{F}$ defined by Definition 3.1, we say that a feedforward module F^n and associated feedback module G^n satisfy the *Jacobian-Matching Condition* (JMC) if:

$$\left(\partial_{s^n} F^n(s^n)\right)^\top = \partial_{s^{n+1}} G^n(s^{n+1}) \tag{12}$$

We say that an architecture $\mathcal{F}$ satisfies the JMC if for $n = 1 \cdots N$, (F^n, G^n) satisfy the JMC.

To illustrate our proposed algorithm to train the feedback weights, let us consider the feedforward module F^n and associated feedback module G^n. Let $\epsilon \sim \mathcal{N}(0, \sigma^2)$ be a perturbation to input feature s^n so that the resulting noisy activations s^n_ϵ triggers a noisy perturbation in the next layer s^{n+1}_ϵ through F^n. Then, we assume s^{n+1}_ϵ yields in turn a noisy reconstruction r^n_ϵ through $\tilde{G}^n$ from Eq. (5) (rather than G^n). Furthermore, we let $\eta \sim \mathcal{N}(0, \sigma^2)$ be a second source of noise in layer s^{n+1}. The resulting noisy activations s^{n+1}_η create the noisy reconstructions r^n_η again through $\tilde{G}^n$. We then prescribe updating the feedback weights with the *Local Difference Reconstruction Loss* (L-DRL) which we define as:

$$\hat{\mathcal{L}}^n_\omega = -\epsilon^\top \cdot (r^n_\epsilon - s^n) + \frac{1}{2}\|r^n_\eta - s^n\|^2. \tag{13}$$

Algorithm 3 Local Difference Reconstruction Loss (L-DRL)

for i = 1 to K **do**
  $s^{n+1} = F^n(s^n; \theta^n)$
  $\epsilon \sim \mathcal{N}(0, \sigma^2)$, $s^n_\epsilon = s^n + \epsilon$
  $s^{n+1}_\epsilon = F^n(s^n_\epsilon; \theta^n)$
  $r^n_\epsilon = G^n(s^{n+1}_\epsilon; \omega^n) - G^n(s^{n+1}; \omega^n) + s^n$
  $\eta \sim \mathcal{N}(0, \sigma^2)$, $r^{n+1}_\eta = s^{n+1} + \eta$
  $r^n_\eta = G^n(r^{n+1}_\eta; \omega^n) - G^n(s^{n+1}; \omega^n) + s^n$
  Update ω_n to descend layer-wise loss $\mathcal{L}^n_\omega$:
    $\hat{\mathcal{L}}^n_\omega = -\epsilon^\top \cdot (r^n_\epsilon - s^n) + \frac{1}{2}\|r^n_\eta - s^n\|^2$
end for

Contrary to existing DTP approaches, the above procedure is repeated K times per training batch, so that feedback weights can quickly and locally (per-layer) adapt on the fly to the feedforward activations and recent feedforward weight updates. This avoids interleaving phases of *pure* feedback weight training with frozen feedforward weights and instead makes it possible to train feedback and feedforward weights together from the beginning.

We now state Theorem 4.2 which guarantees that minimizing $\mathcal{L}^n_\omega$ as defined in Eq. (13) yields the JMC for layer n.

Theorem 4.2. *Let:*

$$\widehat{\mathcal{L}}^n_\omega = -\frac{1}{\sigma^2}\epsilon^\top \cdot (r^n_\epsilon - s^n) + \frac{1}{2\sigma^2}\left\|r^n_\eta - s^n\right\|^2, \tag{14}$$

$$\mathcal{L}^n_\omega = \frac{1}{2}\left\|\partial_{s^n} F^{n^\top} - \partial_{s^{n+1}} G^{n+1}\right\|^2_F. \tag{15}$$

Then:

$$\lim_{\sigma\to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}^n_\omega\right] = -\left\langle \partial_{s^n} F^{n^\top}, \partial_{s^{n+1}} G^n \right\rangle_F + \frac{1}{2}\left\|\partial_{s^{n+1}} G^n\right\|^2_F \tag{16}$$

$$\frac{\partial}{\partial\omega}\lim_{\sigma\to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}^n_\omega\right] = \frac{\partial \mathcal{L}^n_\omega}{\partial \omega}, \tag{17}$$

This means that training the feedback weights with respect to the local layer loss of Eq. (13) makes the feedback path compute the Jacobian of the feedforward path in the limit of small noise and in expectation over the noisy samples.

4.2. Feedforward weight training

Although our new implementation of DTP uses the exact same equations as standard DTP to propagate the targets (Eq. (3)-Eq. (4)) and update the forward weights (Eq. (6)), they acquire a very different meaning with our novel feedback weights training scheme. If we assume that an architecture $\mathcal{F}$ satisfies the JMC upon applying Alg. (3) with fixed feedforward weights, then combining Eq. (8) and Eq. (12) yields:

$$\delta^n_{\text{DTP}} = -\left[\prod_{k=n}^{N-1} \partial_{s^k} F^{k^\top}\right] \cdot \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s^N} = \delta^n_{\text{BP}}, \tag{18}$$

where δ^n_{BP} denotes the activation updates computed by BP. Subsequently, given that the feedforward loss $\mathcal{L}^n_\theta$ defined in Eq. (6) is updated by gradient descent, the whole DTP gradient computing scheme exactly implements BP rather than a hybrid between gradient descent and Gauss-Newton optimization. We now formally state our result.

Theorem 4.3 (Gradient Matching Property)**.** *Let a feedforward architecture $\mathcal{F}$ defined per Definition 3.1 which satisfies the JMC. Then the following holds:*

$$\forall n \in [1, N], \qquad \frac{\partial \mathcal{L}_{\text{pred}}}{\partial \theta^n} = \lim_{\beta\to 0}\frac{1}{2\beta}\frac{\partial}{\partial \theta^n}\|t^n_\beta - s^n\|^2, \tag{19}$$

where the targets $(t_\beta^n)_{n\geq 1}$ *obey the following recursive equations,* $\forall n = N-1 \dots 1$*:*

$$\begin{cases} t_\beta^N = s^N - \beta \frac{\partial \mathcal{L}_{\text{pred}}}{\partial s} \\ t_\beta^n = s^n + G^n(t_\beta^{n+1}; \omega^n) - G^n(s^{n+1}; \omega^n) \end{cases} \tag{20}$$

5. Experiments

In this section, we present several experimental results supporting the above theory. We first numerically demonstrate the claims stated by Theorem 4.2 and Theorem 4.3, thereby showing the efficiency of the proposed approach to align feedforward and feedback weights (JMC) and subsequently compute DTP feedforward weight updates well aligned with BP gradients (GMP). Next, we present training simulation results on MNIST, F-MNIST and CIFAR-10, where our approach significantly outperforms Meulemans et al. (2020)'s DTP. Finally, we report the best results ever obtained on ImageNet 32×32 by a DTP algorithm.

5.1. Demonstrating the JMC

Experimental set-up. The goal of the following experiment is to compare Meulemans et al. (2020)'s DRL algorithm with our L-DRL approach in terms of their ability to align the (transposed) feedforward weights and their associated feedback weights for the last fully connected layer, and thereby realize the JMC in the output layer. We perform this test with randomly initialized and *fixed* feedforward weights and on a *single* randomly selected input batch x (for a given seed). The choice of focusing only on the output layer is justified below.

Architecture. We consider a random batch of CIFAR-10 data along with a LeNet (LeCun et al., 1989) architecture consisting of two convolutional layers and two fully connected (FC) layers. For both algorithms, we use the same feedforward pathway for the model. However since regular DRL prescribes by construction direct connections in the feedback pathway, the form of the G^n functions used depends on the feedback algorithm used. *For DRL*, we use the DDTP-linear architecture as per Meulemans et al. (2020), where the output layer are directly connected to each upstream layer via linear connections. Therefore, the parameters of the resulting G^n functions have dimension $\dim(\omega^n) = s^n \times s^N$ for $n = N-1, \cdots, 1$. However, since the associated feedforward parameters θ^n have dimension $\dim(\theta^n) = s^{n+1} \times s^n$, we can only readily compare $\theta^{N^\top}$ and ω^N in the last FC layer. *For L-DRL*, we use layer-wise G^n functions such that $\dim(\omega^n) = s^n \times s^{n+1}$ for $n = N-1, \cdots, 1$. Full architecture details are included in the Appendix.

Results. We illustrate in Fig. 2 the results obtained. We show the angle (in degrees) and the relative distance between the last layer feedforward ($\theta^{N^\top}$) and feedback weights (ω^N) throughout pure feedback training on a single input batch. Therefore, each feedback training iteration here corresponds to a feedback weight update on the *same* input batch (for a given seed). However, we do use different input batches across different seeds. For each algorithm, the amount of noise and learning rates have been carefully tuned to achieve the minimal angles and distances after 5000 iterations, which we empirically found to be large enough to reach convergence for both algorithms. We observe that L-DRL achieves an angle of $\approx 3°$ and a relative distance of ≈ 0, while DRL can only reduce these quantities to $18.7°$ and ≈ 1.8 respectively. These results confirm that our L-DRL is more suited than DRL to achieve the JMC in the output layer.

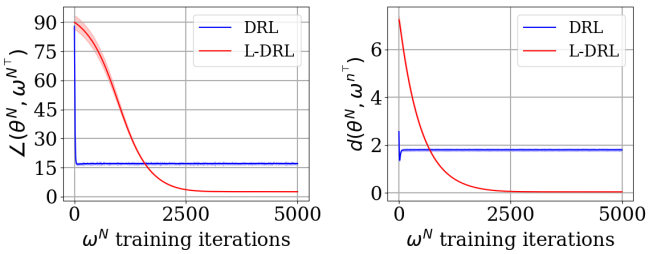

Figure 2. Angle in degrees ($\angle(\theta^N, \omega^{N^\top})$) and relative distance ($d(\theta^N, \omega^{N^\top})$) between θ^N and $\omega^{N^\top}$ throughout feedback weight learning with L-DRL (ours) and DRL (Meulemans et al., 2020) with *fixed* feedforward weights.

5.2. Demonstrating the GMP

Experimental set-up. In this experiment, we want to demonstrate the ability of our proposed DTP to compute *feedforward* weight updates closely matching those prescribed by BP (therefore achieving the GMP), assuming that the JMC is initially satisfied, as hypothesized by Theorem 4.3. Again here, we assume a *single* randomly selected input batch x (also with different input across different seeds). In contrast with the previous experiment though, we carry out this analysis across *all* the layers. Indeed, regardless of the form of the G^n functions (whether we use direct connections or not), the DTP feedforward weight updates can always be compared against those of BP. Given randomly sampled feedforward parameters θ^n, we study five different feedback weight initialization schemes and associated targets computation: (a) ω^n are random and targets are computed through the DDTP-linear feedback pathway ($\text{DRL}_{\text{random}}$); (b) same as (a) with targets computed through the layer-wise feedback pathway ($\text{L-DRL}_{\text{random}}$); (c) ω^n are trained with DRL (DRL); (d) ω^n are trained with L-DRL (L-DRL); (e) Finally, $\omega^n = \theta^{n^\top}$ with targets propagated through the layer-wise feedback pathway ($\text{L-DRL}_{\text{sym}}$). For each of these situations, the feedforward

DTP weight updates are thereafter obtained with Eq. (6) on the one hand. On the other hand, we compute BP gradients via standard BP through the feedforward pathway.

Architecture. The architecture used for this experiment is the same LeNet architecture than the one used for the previous experiment, with two convolutional layers and two fully connected layers.

Results. We show on Fig. 3 the results obtained. The blue, red, green and purple bars correspond to the angle between DTP feedforward weight updates and those of BP ($\angle(\Delta\theta^n_{\text{DTP}}, \Delta\theta^n_{\text{BP}})$) for the first Conv, second Conv, first FC and second FC layers respectively: the lower $\angle(\Delta\theta^n_{\text{DTP}}, \Delta\theta^n_{\text{BP}})$, the more the GMP is satisfied. We show these quantities for each of the five feedback weight initialization mentioned above. We observe that upon training the feedback weights with L-DRL (compared to a random configuration), the GMP is significantly better satisfied ($\angle(\Delta\theta^n_{\text{DTP}}, \Delta\theta^n_{\text{BP}})$ going from $\approx 90°$ to $\lessapprox 35°$) than when trained with DRL ($\lessapprox 79°$)), and almost as well as in the ideal situation with symmetrically initialized weights. Overall, these results confirm the prediction of Theorem 4.3.

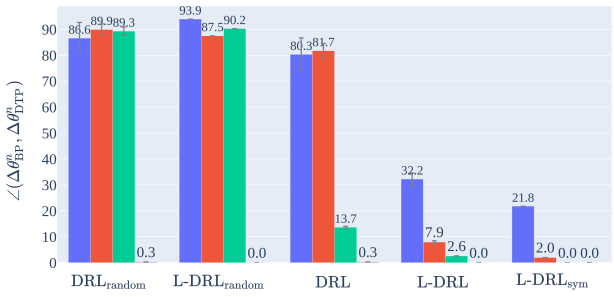

Figure 3. Angle between the forward weight updates obtained through L-DRL (ours) or DRL (Meulemans et al., 2020) and those obtained through BP, for each layer, under various initial conditions.

5.3. DTP learning dynamics

Experimental set-up. We present here our training experiments obtained on MNIST, F-MNIST and CIFAR-10 with our implementation of DTP (refered to as "DTP" or "Ours" below) and that of Meulemans et al. (2020) which we will refer to as "DDTP". While the previous DRL/L-DRL terminology concerns feedback weights training specifically, the term "DDTP" is used here to refer to the resulting feedforward weights training algorithm when GN targets are being computed, rather than the architecture itself. Also, we want to emphasize that two features of DDTP fundamentally differs from our DTP. First, our DTP is made to emulate BP while DDTP is a hybrid between GN optimization and BP as highlighted previously. Second, while DDTP employs feedback weights pre-training and subsequent interleaved epochs of pure feedback weights training, our DTP trains together *at all times* feedforward weights and feedback weights and allowing for *multiple* feedback weight updates per mini-batch. To disentangle these two aspects and ensure a fair comparison between our DTP and DDTP, we propose two different implementations of DDTP.

Simple DDTP ("s-DDTP") is the standard DDTP implementation of Meulemans et al. (2020) that yields their best training results. For s-DDTP, training starts with $N_{\omega,\text{i}}$ epochs of pure feedback weights training, then at each subsequent epoch feedback weights and feedforward weights are both updated once per batch, and each of these epoch is followed by $N_{\omega,\text{e}}$ epoch of pure feedback training. Therefore, denoting N_θ the number of epochs where the feedforward weights are trained, there are $N_\omega = N_{\omega,\text{i}} + N_\theta \times (1 + N_{\omega,\text{e}})$ epochs where the feedback weights are trained, therefore $\mathcal{O}(N_{\omega,\text{i}} + N_\theta \times (1 + N_{\omega,\text{e}}))$ feedback weight updates.

We define *Parallel* DDTP ("p-DDTP") as a variant of DDTP where there is no initial feedback pre-training ($N_{\omega,\text{i}} = 0$), nor interleaved epochs of pure feedback training ($N_{\omega,\text{e}} = 0$), but where feedback weights and feedforward weights are always trained altogether, with K feedback weight updates per batch, yielding $\mathcal{O}(N_\theta \times K)$ feedback weight updates. Therefore, p-DDTP has the same complexity cost for feedback weights training as in our DTP. We use the same architecture in this study as in Section 5.1-5.2.

Results. We display in Table 1 the accuracies obtained with our DTP, s-DDTP and p-DDTP on MNIST, Fashion MNIST ("F-MNIST") and CIFAR-10. Our DTP outperforms s-DDTP and p-DDTP on all tasks, by $\approx 0.3\%$ on MNIST and F-MNIST, by at least $\approx 9\%$ on CIFAR-10 and is within $\approx 1\%$ of the BP baseline performance. While p-DDTP slightly outperforms s-DDTP on MNIST and F-MNIST, it performs worse than s-DDTP on CIFAR-10, suggesting that DDTP does not benefit much from multiple feedback weight updates per batch. An important conclusion to be drawn here is that the gap in performance between our DTP and DDTP is not due to updating the feedback weights multiple times per training batch, but more fundamentally to the feedback training scheme at use (L-DRL for our DTP, or DRL for DDTP), yielding better feedforward error signals with our DTP. This conclusion is also confirmed by Fig. 4 where we plot the angle between the DTP feedforward weight updates and those of BP ($\angle(\Delta\theta^n_{\text{BP}}, \Delta\theta^n_{\text{DTP}})$) throughout learning CIFAR-10, for each layer and each algorithm. While the angles obtained by DTP, s-DDTP and p-DDTP are comparable for the last two (FC) layers ($\approx 0°$), they are at least twice as smaller for DTP compared to s-DDTP and p-DDTP in the first two (Conv) layers. Finally, these angles are $\approx 15°$ smaller for s-DDTP compared to

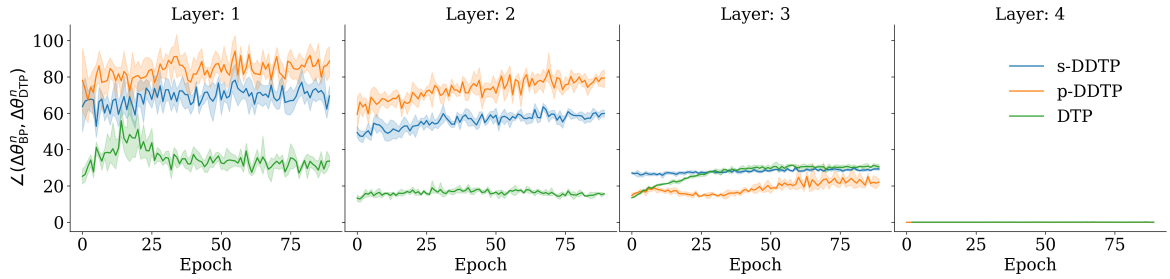

Figure 4. Angle between forward weight updates obtained through DTP (ours), s-DDTP, p-DDTP and those obtained through backpropogation for each layer throughout training on CIFAR-10 with LeNet architecture. Each epoch represents a feedforward training epoch, pure feedback training epochs for s-DDTP are not displayed.

p-DDTP. Consequently, these curves directly account for the discrepancies in results on CIFAR-10 reported in Table 1.

Table 1. Accuracies (%) obtained with BP, DDTP and our DTP on the LeNet architecture for MNIST, F-MNIST and CIFAR-10 test set. Each result is in terms of the mean and standard deviation obtained over five different seeds.

	MNIST	F-MNIST	CIFAR-10
s-DDTP	$98.59_{\pm 0.16}$	$88.86_{\pm 0.44}$	$76.33_{\pm 0.27}$
p-DDTP	$98.58_{\pm 0.13}$	$89.42_{\pm 0.69}$	$72.15_{\pm 0.29}$
Ours	$\mathbf{98.93_{\pm 0.04}}$	$\mathbf{90.35_{\pm 0.11}}$	$\mathbf{85.33_{\pm 0.32}}$
BP	$98.92_{\pm 0.04}$	$91.94_{\pm 0.33}$	$86.34_{\pm 0.35}$

5.4. Towards scaling up DTP

Since we have demonstrated that our DTP learns better error signals to update the feedforward weights than DDTP with consistently better performance, we focus in this section on learning a slightly deeper and wider architecture on CIFAR-10 and ImageNet 32×32 (van den Oord et al., 2016), a downsampled version of the full ImageNet data. For these experiments, we employ a 6-layers VGG-like architecture, consisting of 5 Conv layers and 1 FC layer (see Appendix for architecture details).

Results. We report our results in Tables 2–3. With this choice of architecture, our DTP achieves 89.38% accuracy on CIFAR-10 and 60.55% top-5 accuracy on ImageNet 32 × 32, which is both cases within $< 1\%$ of the BP baseline.

6. Discussion

Training feedforward weights with Gauss-Newton targets results in optimal updates to move the feedforward activations towards their associated target, yet they appear suboptimal to decrease the prediction loss (Meulemans et al., 2020), which calls for the design of a principled way to build backprop-like targets. In this work, we have demonstrated the benefits of such an approach, with mathematically and experimentally grounded arguments. We showed the efficiency of our L-DRL algorithm to align feedforward and (transposed) feedback weights and therefore achieve the Jacobian matching condition (JMC). We also showed that the resulting feedforward weight updates prescribed by Difference Target Propagation (DTP) closely match those of BP, a property we called the gradient-matching property (GMP). Our DTP implementation subsequently outperforms DDTP (Meulemans et al., 2020) on all training tasks and approaches the BP baseline performance. We also consistently showed that the more the GMP is satisfied throughout learning, the better the resulting performance. The best CIFAR-10 performance obtained by our DTP is $\approx 13\%$ higher than the existing DTP performances reported in the literature (Bartunov et al., 2018; Meulemans et al., 2020) and to our knowledge this is the first report of a DTP performance closely matching that of BP on such a complex task

Table 2. Accuracies (%) obtained on CIFAR-10 with BP and our DTP on a VGG-like architecture. Each result is in terms of the mean and standard deviation obtained over five different seeds. We also report below the current best CIFAR-10 accuracies obtained by DTP in the literature on any architecture.

	Accuracy
BP	$89.07_{\pm 0.22}$
Ours	$\mathbf{89.38_{\pm 0.20}}$
Meulemans et al. (2020)	76.01
Bartunov et al. (2018)	60.53

Table 3. Top-1 and Top-5 Accuracies for ImageNet 32×32 validation set obtained with BP and our DTP on a VGG-like architecture across five seeds.

	Top-1	Top-5
BP	37.29 ± 0.14	61.28 ± 0.11
Ours	36.79 ± 0.05	60.55 ± 0.06

as ImageNet 32×32.

Limitations and Future Work. Our prescription to run several feedback weight updates per training batch entails longer simulation times but may be biologically plausible since local recurrent paths will have shorter axons that should have much shorter delays than long-range paths with complex, long axons (Debanne, 2004; Debanne et al., 2011). As it can be seen from Appendix E, our DTP implementation can be up to 30 times slower than BP. Future work could be done to leverage the parallelism allowed by our layer-wise feedback weight training strategy to accelerate training and subsequently scale up our DTP implementation to ImageNet on deeper architectures. However we emphasize again that DTP is not meant as a new practical optimization method but rather an abstract but plausible biological implementation of a BP-like update mechanism.

Our code is available at `https://github.com/ernoult/scalingDTP`.

Acknowledgements. BR was supported by NSERC (Discovery Grant: RGPIN-2020-05105; Discovery Accelerator Supplement: RGPAS-2020-00031) and CIFAR (Canada CIFAR AI Chair and Learning in Machines and Brains Program). EB and AM are supported by NSERC Discovery Grant RGPIN-2021-04104. We acknowledge resources provided by Compute Canada. YB was funded by NSERC and CIFAR. Many thanks to Anirudh Goyal and Guillaume Bellec for their useful feedback.

References

Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T., and Tweed, D. B. Deep learning without weight transport. *Advances in Neural Information Processing Systems*, 32: 976–984, 2019.

Bartunov, S., Santoro, A., Richards, B. A., Marris, L., Hinton, G. E., and Lillicrap, T. P. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 9390–9400, 2018.

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

Bengio, Y. Deriving differential target propagation from iterating approximate inverses. *arXiv preprint arXiv:2007.15139*, 2020.

Choromanska, A., Cowen, B., Kumaravel, S., Luss, R., Rigotti, M., Rish, I., Diachille, P., Gurev, V., Kingsbury, B., Tejwani, R., et al. Beyond backprop: Online alternating minimization with auxiliary variables. In *International Conference on Machine Learning*, pp. 1193–1202. PMLR, 2019.

Crafton, B., Parihar, A., Gebhardt, E., and Raychowdhury, A. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience*, 13:525, 2019.

Debanne, D. Information processing in the axon. *Nature Reviews Neuroscience*, 5(4):304–316, 2004.

Debanne, D., Campanac, E., Bialowas, A., Carlier, E., and Alcaraz, G. Axon physiology. *Physiological reviews*, 91 (2):555–602, 2011.

Gauss, C. F. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, volume 7. FA Perthes, 1877.

Guerguiev, J., Kording, K., and Richards, B. Spike-based causal inference for weight alignment. In *International Conference on Learning Representations*, 2019.

Kunin, D., Nayebi, A., Sagastuy-Brena, J., Ganguli, S., Bloom, J., and Yamins, D. Two routes to scalable credit assignment without weight symmetry. In *International Conference on Machine Learning*, pp. 5511–5521. PMLR, 2020.

Lansdell, B. J., Prakash, P. R., and Kording, K. P. Learning to solve the credit assignment problem. In *International Conference on Learning Representations*, 2019.

Launay, J., Poli, I., and Krzakala, F. Principled training of neural networks with direct feedback alignment. *arXiv preprint arXiv:1906.04554*, 2019.

LeCun, Y. *Modeles connexionnistes de lapprentissage*. PhD thesis, These de Doctorat, Universite Paris, 1987.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pp. 498–515. Springer, 2015.

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

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

Manchev, N. and Spratling, M. W. Target propagation in recurrent neural networks. *J. Mach. Learn. Res.*, 21:7–1, 2020.

Meulemans, A., Carzaniga, F. S., Suykens, J. A. K., Sacramento, J., and Grewe, B. F. A theoretical framework for target propagation. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Moskovitz, T. H., Litwin-Kumar, A., and Abbott, L. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

Nøkland, A. Direct feedback alignment provides learning in deep neural networks. *Advances in Neural Information Processing Systems*, 29:1037–1045, 2016.

Ororbia, A. G. and Mali, A. Biologically motivated algorithms for propagating local target representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4651–4658, 2019.

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., Ganguli, S., et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22(11): 1761–1770, 2019.

Roulet, V. and Harchaoui, Z. Target propagation via regularized inversion. *arXiv preprint arXiv:2112.01453*, 2021.

Sacramento, J. a., Ponte Costa, R., Bengio, Y., and Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

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

van den Oord, A., Kalchbrenner, N., Espeholt, L., kavukcuoglu, k., Vinyals, O., and Graves, A. Conditional image generation with pixelcnn decoders. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Whittington, J. C. and Bogacz, R. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5):1229–1262, 2017.

Xiao, W., Chen, H., Liao, Q., and Poggio, T. Biologically-plausible learning algorithms can scale to large datasets. In *International Conference on Learning Representations*, 2018.

Appendix

The appendix is structured as follows:

- Section A provides the demonstrations of Theorem 4.2 and Theorem 4.3
- Section B shows the explicit equations used for inference and weight updates on a simple example.
- Section C precisely describes the architectures used for the experiments.
- Section D lists all the hyperparameters used across all the experiments.
- Finally, Section E gives the simulation run times of all the training experiments.

A. Theoretical results

A.1. Feedback weights training

We re-state our main theorem for feedback weights training (Theorem (4.2) in the main).

Theorem A.1. *Let:*

$$\widehat{\mathcal{L}}_{\omega}^{n} = -\frac{1}{\sigma^2}\epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) + \frac{1}{2\sigma^2}\left\|r_{\eta}^{n} - s^{n}\right\|^2, \tag{21}$$

$$\mathcal{L}_{\omega}^{n} = \frac{1}{2}\left\|\partial_{s^n} F^{n^{\top}} - \partial_{s^{n+1}} G^{n+1}\right\|_F^2. \tag{22}$$

Then:

$$\lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = -\left\langle \partial_{s^n} F^{n^{\top}}, \partial_{s^{n+1}} G^{n} \right\rangle_F + \frac{1}{2}\left\|\partial_{s^{n+1}} G^{n}\right\|_F^2 \tag{23}$$

$$\frac{\partial}{\partial \omega^n} \lim_{\sigma \to 0} \mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = \frac{\partial \mathcal{L}_{\omega}^{n}}{\partial \omega^n}, \tag{24}$$

Proof. We have:

$$\begin{aligned}
\epsilon^{\top} \cdot (r_{\epsilon}^{n} - s^{n}) &= \epsilon^{\top} \cdot (G^{n} \circ F^{n}(s^{n} + \epsilon) - G^{n} \circ F^{n}(s^{n})) \\
&= \epsilon^{\top} \cdot (\partial_{s^{n+1}} G^{n} \cdot \partial_{s^n} F^{n} \cdot \epsilon + o(\|\epsilon\|)) \\
&= \mathrm{Tr}(\epsilon \cdot \epsilon^{\top} \cdot \partial_{s^{n+1}} G^{n} \cdot \partial_{s^n} F^{n}) + o(\|\epsilon\|^2).
\end{aligned} \tag{25}$$

Likewise:

$$\left\|r_{\eta}^{n} - s^{n}\right\|^2 = \mathrm{Tr}\left(\eta \cdot \eta^{\top} \cdot \partial_{s^{n+1}} G^{n} \cdot \partial_{s^{n+1}} G^{n^{\top}}\right) + o(\|\eta\|^2). \tag{26}$$

Moreover, since $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$ and $\eta \sim \mathcal{N}\left(0, \sigma^2\right)$, $\mathbb{E}_{\epsilon}\left[o(\|\epsilon\|^2)\right] = \mathbb{E}_{\eta}\left[o(\|\eta\|^2)\right] = o(\sigma^2)$. Altogether with Eq. (25) and Eq. (26), we finally obtain:

$$\mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = -\frac{1}{\sigma^2}\mathrm{Tr}(\underbrace{\mathbb{E}_{\epsilon}\left[\epsilon\cdot\epsilon^{\top}\right]}_{=\sigma^2\times\mathbf{1}}\cdot\partial_{s^{n+1}}G^{n}\cdot\partial_{s^{n}}F^{n}) + \frac{1}{2\sigma^2} + \mathrm{Tr}\left(\underbrace{\mathbb{E}_{\eta}\left[\eta\cdot\eta^{\top}\right]}_{=\sigma^2\times\mathbf{1}}\cdot\partial_{s^{n+1}}G^{n}\cdot\partial_{s^{n+1}}G^{n}{}^{\top}\right) + o(1)$$
$$= -\left\langle\partial_{s^{n}}F^{n}{}^{\top},\partial_{s^{n+1}}G^{n}\right\rangle_{F} + \frac{1}{2}\left\|\partial_{s^{n+1}}G^{n}\right\|_{F}^{2} + o(1). \tag{27}$$

Therefore, sending $\sigma \to 0$ yields the desired result Eq. (23). Finally, noticing that:

$$\lim_{\sigma\to 0}\mathbb{E}_{\epsilon,\eta}\left[\hat{\mathcal{L}}_{\omega}^{n}\right] = \mathcal{L}_{\omega}^{n} - \frac{1}{2}\left\|\partial_{s^{n}}F^{n}\right\|_{F}^{2}, \tag{28}$$

with the second term of Eq. (28) not depending on the feedback weights ω^n, Eq. (24) is straightforward. □

A.2. Feedforward weights training

We re-state here our main theorem for feedforward weights training (Theorem 4.3) in the main).

Theorem A.2 (Gradient Matching Property)**.** *Let a feedforward architecture $\mathcal{F}$ defined per Definition 3.1 which satisfies the JMC. Then the following holds:*

$$\forall n\in[1,N],\qquad \frac{\partial\mathcal{L}_{\mathrm{pred}}}{\partial\theta^{n}} = \lim_{\beta\to 0}\frac{1}{2\beta}\frac{\partial}{\partial\theta^{n}}\|t_{\beta}^{n}-s^{n}\|^{2}, \tag{29}$$

where the targets $(t_\beta^n)_{n\geq 1}$ obey the following recursive equations, $\forall n = N\dots 2$:

$$\begin{cases} t_{\beta}^{N} = s^{n} - \beta\frac{\partial\mathcal{L}_{\mathrm{pred}}}{\partial s} \\ t_{\beta}^{n} = s^{n} + G^{n}(t_{\beta}^{n+1};\omega^{n}) - G^{n}(s^{n+1};\omega^{n}) \end{cases} \tag{30}$$

Proof. First, note we have:

$$\begin{cases} t_{\beta}^{N} - s^{n} = -\beta\frac{\partial\mathcal{L}_{\mathrm{pred}}}{\partial s}, \\ t_{\beta}^{n} - s^{n} = \partial_{s^{n+1}}G^{n}\cdot\left(t_{\beta}^{n+1} - s_{\beta}^{n+1}\right) + o\left(\left\|t_{\beta}^{n+1} - s_{\beta}^{n+1}\right\|\right). \end{cases} \tag{31}$$

Since $t_\beta^N - s^N = o(\beta)$, by immediate induction $t_\beta^n - s^n = o(\beta) \quad \forall n = N-1\cdots 1$. Denoting $\hat{\delta}_{\mathrm{DTP}}^{n}(\beta) \triangleq \frac{t^n - s^n}{\beta}$, we therefore obtain:

$$\begin{cases} \hat{\delta}_{\mathrm{DTP}}^{N}(\beta) = -\frac{\partial\mathcal{L}_{\mathrm{pred}}}{\partial s} + o(1), \\ \hat{\delta}_{\mathrm{DTP}}^{n}(\beta) = \partial_{s^{n+1}}G^{n}\cdot\hat{\delta}_{\mathrm{DTP}}^{n+1}(\beta) + o\left(1\right). \end{cases} \tag{32}$$

Furthermore note that, treating t_β^n as a constant, we also have:

$$\frac{1}{2\beta}\frac{\partial}{\partial\theta^{n}}\|t_{\beta}^{n}-s^{n}\|^{2} = -\frac{1}{\beta}\partial_{\theta^{n}}F^{n}{}^{\top}\cdot(t_{\beta}^{n}-s^{n})$$
$$= -\partial_{\theta^{n}}F^{n}{}^{\top}\cdot\hat{\delta}_{\mathrm{DTP}}^{n}(\beta). \tag{33}$$

Finally, sending $\beta \to 0$, defining:

$$\delta^n_{\mathrm{DTP}} \triangleq \lim_{\beta \to 0} \hat{\delta}^n(\beta) \tag{34}$$

$$\Delta^n_{\theta,\mathrm{DTP}} \triangleq \lim_{\beta \to 0} \frac{1}{2\beta} \frac{\partial}{\partial \theta^n} \|t^n_\beta - s^n\|^2 \tag{35}$$

along with the JMC property $\partial_{s^{n+1}} G^n = \partial_{s^n} {F^n}^\top$, we obtain:

$$\begin{cases} \delta^N_{\mathrm{DTP}} = -\frac{\partial \mathcal{L}_{\mathrm{pred}}}{\partial s}, \\ \delta^n_{\mathrm{DTP}} = \partial_{s^{n+1}} {F^n}^\top \cdot \hat{\delta}^{n+1}_{\mathrm{DTP}} \\ \Delta^n_{\theta,\mathrm{DTP}} = -\partial_{\theta^n} {F^n}^\top \cdot \hat{\delta}^n_{\mathrm{DTP}} \end{cases} \tag{36}$$

Note that Eq. (36) is equivalent to computing $\frac{\partial \mathcal{L}_{\mathrm{pred}}}{\partial \theta^n}$ by backprop, yielding the desired result Eq. (29). □

B. A concrete example with explicit equations

We detail for completeness and clarity all the equations for the neural dynamics and the learning rules of the forward and of the backward weights for a LeNet-like architecture with two Conv layers and one fully connected layer for the sake of simplicity.

Forward operations.

$$\begin{aligned} s^1 &= F^0(x; \theta^0) = \sigma(\theta^0 \star x), \\ s^2 &= F^1(s^1; \theta^1) = \sigma(\theta^1 \star s_1), \\ s^3 &= F^2(s_2; \theta^2) = \theta^2 \cdot s^2, \\ \hat{y} &= \mathrm{softmax}(s_3), \end{aligned}$$

where we implicitly assume the flattening operation between s^2 and s^3 for notational convenience.

Backward operations. We assume here the following feedback operators G^1 and G^2 associated with F^1 and F^2 respectively:

$$\begin{aligned} G^2(s^3; \omega^2) &= \omega^2 \cdot s^3 \\ G^1(s^2; \omega^1) &= \omega^1 \star \sigma(s^2). \end{aligned}$$

Again, note that there is no G^0 feedback operator paired to F^0 since we do not need to propagate error signals down to the input layer.

Feedback weights training. Given input noises ϵ^2 and η^3 in layers s^2 and s^3 respectively, we update ω^2 so as to minimize the loss $\mathcal{L}_{\omega^2}$. More precisely, ϵ^2, η^3 and $\mathcal{L}_{\omega^2}$ are defined as:

$$\begin{aligned} \epsilon^2 &\sim \mathcal{N}(0, \sigma^2), \\ \eta^3 &\sim \mathcal{N}(0, \sigma^2), \\ \mathcal{L}_{\omega^2} &= -\left(\epsilon^2\right)^\top \cdot \left(G^2(F^2(s^2 + \epsilon^2)) - G^2(s^3)\right) + \frac{1}{2} \left\| G^2(s^3 + \eta^3) - G^2(s^3) \right\|^2, \end{aligned}$$

which results in the weight update for ω^2:

$$\Delta\omega^2 = \epsilon^2 \cdot (\theta^2 \cdot \Delta x^2)^\top - (\omega^2 \cdot \eta^3) \cdot \Delta {y^3}^\top. \tag{37}$$

Similarly, we train the feedback convolutional filters ω^1 by injecting the input noise ϵ^1 and η^2 in s^1 and s^2 respectively and minimizing the loss $\mathcal{L}_{\omega^1}$ defined as:

$$\begin{aligned}
\epsilon^1 &\sim \mathcal{N}(0, \sigma^2), \\
\eta^2 &\sim \mathcal{N}(0, \sigma^2), \\
\mathcal{L}_{\omega^1} &= -\left(\epsilon^1\right)^\top \cdot (G^1(F^1(s^1 + \epsilon^1)) - G^1(s^2)) + \frac{1}{2}\left\|G^1(s^2 + \eta^2) - G^1(s^2)\right\|^2,
\end{aligned}$$

which results in the weight update for the convolutional feedback filters ω^1:

$$\Delta\omega^1 = \epsilon^1 \star (\sigma(F^1(s^1 + \epsilon^1)) - \sigma(s^2)) - (G^1(s^2 + \eta^2) - G^1(s^2)) \star (\sigma(s^2 + \eta^2) - \sigma(s^2)). \tag{38}$$

Forward weights training. We compute the first target t^3_β and associated weight update $\Delta\theta^2$ as:

$$t^3_\beta = s^3 + \beta(\hat{y} - y), \tag{39}$$

$$\Delta\theta^2 = \frac{1}{\beta}(t^3_\beta - s^3) \cdot {s^2}^\top. \tag{40}$$

Then, the target t^3_β passes through G^2, yielding the target t^2_β and associated weight update $\Delta\theta^1$:

$$t^2_\beta = s^2 + G^2(t^3; \omega^2) - G^2(s^3; \omega^2), \tag{41}$$

$$\Delta\theta^1 = \frac{1}{\beta}((t^2_\beta - s^2) \odot \sigma'(s^2)) \star s^1. \tag{42}$$

Similarly, we compute t^1_β and $\Delta\theta^0$ as:

$$t^1_\beta = s^1 + G^1(t^2; \omega_2) - G^1(s^2; \omega_2) \tag{43}$$

$$\Delta\theta^0 = \frac{1}{\beta}((t^1_\beta - s^1) \odot \sigma'(s^1)) \star x \tag{44}$$

C. Architecture Details

In Table 4, we give the details of the two representative architectures studied in our work.

D. Hyperparameters

In Tables 5,6,8,9, we report the hyperparameters for each method and dataset studied. In both CIFAR-10 and Imagenet 32×32 experiments we use the same data augmentation consisting of random horizontal flipping with 0.5 probability and random cropping with padding 4.

LeNet	VGGNet
Conv 5x5x32 (stride=1, pad=2)	Conv 3x3x128 (stride=1, pad=1)
Maxpool 3x3 (stride=2, pad=1)	Maxpool 2x2 (stride=2, pad=0)
Conv 5x5x64 (stride=1, pad=2)	Conv 3x3x128 (stride=1, pad=1)
Maxpool 3x3 (stride=2, pad=1)	Maxpool 2x2 (stride=2, pad=0)
FC 512	Conv 3x3x256 (stride=1, pad=1)
FC+Softmax 10	Maxpool 2x2 (stride=2, pad=0)
-	Conv 3x3x256 (stride=1, pad=1)
-	Maxpool 2x2 (stride=2, pad=0)
-	Conv 3x3x512 (stride=1, pad=1)
-	Maxpool 2x2 (stride=2, pad=0)
-	FC+Softmax 10

Table 4. Architectures described by layer

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
activation	ELU	ELU	ELU
lr_f	0.007938	0.01374	0.03
forward optimizer	SGD	SGD	SGD
forward momentum	0.9	0.9	0.9
wd_f	0.0001	0.0001	0.0001
scheduler	cosine	cosine	cosine
scheduler eta min	0.00001	0.00001	0.00001
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	kaiming uniform	kaiming uniform	kaiming uniform
batch size	166	140	193
epochs	40	40	90

Table 5. Tuned BP LeNet hyperparameters

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
β	0.47	0.47	0.46
σ	[0.4, 0.4, 0.2]	[0.4, 0.4, 0.2]	[0.41, 0.28, 0.19]
activation	ELU	ELU	ELU
lr_f	0.02	0.005	0.0.01
forward optimizer	SGD	SGD	SGD
forward momentum	0.9	0.9	0.9
wd_f	0.0001	0.0001	0.0001
lr_fb	[0.06, 0.006, 0.018]	[0.068, 0.006, 0.018]	[0.001,0.005,0.045]
feedback training iterations	[18, 23, 12]	[18, 23, 12]	[41, 51, 24]
backward optimizer	SGD	SGD	SGD
backward momentum	0.9	0.9	0.9
wd_fb	None	None	None
scheduler	cosine	cosine	cosine
scheduler eta min	0.00001	0.00001	0.00001
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	kaiming uniform	kaiming uniform	kaiming uniform
batch size	107	107	100
epochs	40	40	90

Table 6. Tuned DTP LeNet hyperparameters

Hyperparameter	Dataset	
	CIFAR-10	ImageNet 32×32
channels	[128, 128, 256, 256, 512]	[128, 128, 256, 256, 512]
β	0.7	0.7
σ	[0.4, 0.4, 0.2, 0.2, 0.08]	[0.4, 0.4, 0.2, 0.2, 0.08]
activation	ELU	ELU
lr_f	0.05	0.01
forward optimizer	SGD	SGD
forward momentum	0.9	0.9
wd_f	0.0001	0.0001
lr_fb	[1e-4, 3.5e-4, 8e-3, 8e-3, 0.18]	[1e-4, 3.5e-4, 8e-3, 8e-3, 0.18]
feedback training iterations	[20, 30, 35, 55, 20]	[25, 35, 40, 60, 25]
backward optimizer	SGD	SGD
backward momentum	0.9	0.9
wd_fb	None	None
scheduler	cosine	cosine
scheduler eta min	0.00001	0.00001
scheduler Tmax	85	85
scheduler interval/frequency	epoch/1	epoch/1
initialization	kaiming uniform	kaiming uniform
batch size	128	256
epochs	90	90

Table 7. Tuned DTP VGGNet hyperparameters

Table 8. Tuned s-DDTP LeNet hyperparameters

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
target stepsize η	0.044	0.044	0.016
β_1	0.9	0.9	0.9
β_2	0.999	0.999	0.999
ϵ	[6.5e-05, 1.1e-05, 6.8e-05, 2.5e-05]	[6.5e-05, 1.1e-05, 6.8e-05, 2.5e-05]	[2.7e-08, 1.9e-08, 4.5e-06, 4.0e-05]
ϵ_{fb}	9.5e-08	9.5e-08	7.5e-07
$\beta_{1,\text{fb}}$	0.999	0.999	0.999
$\beta_{2,\text{fb}}$	0.999	0.999	0.999
σ	[4.7e-05, 8.7e-4, 2.4e-4, 3.9e-05]	[4.7e-05, 8.7e-4, 2.4e-4, 3.9e-05]	[9.2e-3, 9.2e-3, 9.2e-3, 9.2e-3]
activation	tanh	tanh	tanh
lr_f	[1.7e-3, 0.09, 0.02, 0.002]	[0.0016, 0.097, 0.025, 1.9e-3]	[2.6e-4, 8.9e-4, 1.4e-4, 3.4e-06]
forward optimizer	Adam	Adam	Adam
wd_f	0	0	0
lr_fb	1.6e-4	1.6e-4	4.5e-3
feedback training iterations	[1, 1, 1, 1]	[1, 1, 1, 1]	[1, 1, 1, 1]
feedback activation	linear	linear	linear
backward optimizer	Adam	Adam	Adam
wd_fb	3.993e-05	3.993e-05	6.169295107849636e-05
feedback pre-training epochs	10	10	10
feedback extra training epochs	1	1	1
scheduler	cosine	cosine	cosine
scheduler eta min	1e-5	1e-5	1e-5
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	xavier normal	xavier normal	xavier normal
batch size	143	143	128
epochs	40	40	90

Table 9. Tuned p-DDTP LeNet hyperparameters

Hyperparameter	Dataset		
	MNIST	F-MNIST	CIFAR-10
channels	[32, 64]	[32, 64]	[32, 64]
target stepsize η	0.0428	0.01308	0.09983
β_1	0.9	0.9	0.9
β_2	0.999	0.999	0.999
ϵ	[4.4e-06, 9e-07, 2.2e-05, 1.4e-05]	[2.4e-08, 4.8e-06, 4.7e-06, 1.6e-07]	[3.8e-05, 1e-07, 4.7e-07, 2.4e-06]
ϵ_{fb}	1.2e-08	2.5e-07	2e-07
$\beta_{1,\text{fb}}$	0.999	0.999	0.999
$\beta_{2,\text{fb}}$	0.999	0.999	0.999
σ	[1.3e-05, 2.4e-4, 2.9e-05, 3.3e-4]	[3.8e-3, 2.2e-3, 5.6e-4, 4.2e-3]	[3e-4, 3e-05, 7.9e-05, 6.6e-4]
activation	tanh	tanh	tanh
lr_f	[0.004, 1.4e-3, 6.1e-4, 1.3e-4]	[3.9e-4, 1.8e-3, 1.5e-3, 1.5e-4]	[2.2e-4, 6.2e-3, 1.5e-4, 5.5e-05]
forward optimizer	Adam	Adam	Adam
wd_f	0	0	0
lr_fb	6.4e-4	1.8e-4	1.1e-3
feedback training iterations	[49, 32, 54, 11]	[48, 42, 52, 22]	[24, 35, 36, 19]
feedback activation	linear	linear	linear
backward optimizer	Adam	Adam	Adam
wd_fb	2.236e-05	0.000128	3.564e-06
feedback pre-training epochs	0	0	0
feedback extra training epochs	0	0	0
scheduler	cosine	cosine	cosine
scheduler eta min	0.00001	0.00001	0.00001
scheduler Tmax	85	85	85
scheduler interval/frequency	epoch/1	epoch/1	epoch/1
initialization	xavier normal	xavier normal	xavier normal
batch size	138	141	190
epochs	40	40	90

E. Simulation run times

Table 10. Run time and accuracy of the training experiments (*hours:minutes*) for ImageNet 32×32 obtained with BP and our DTP on a VGG-like architecture across 5 seeds.

	TOP-1	TOP-5	RUN TIME
BP	37.29 ± 0.14	61.28 ± 0.11	$2{:}22 \pm 0{:}2$
OURS	36.79 ± 0.05	60.55 ± 0.06	$61{:}41 \pm 0{:}54$

Table 11. Run time of the training experiments (*hours:minutes*) obtained with BP, DDTP and our DTP on the LeNet architecture for MNIST, F-MNIST and CIFAR-10 across 5 seeds.

	MNIST	F-MNIST	CIFAR-10
S-DDTP	$0{:}36 \pm 0{:}06$	$0{:}32 \pm 0{:}05$	$1{:}11 \pm 0{:}04$
P-DDTP	$3{:}03 \pm 0{:}04$	$3{:}01 \pm 0{:}03$	$3{:}42 \pm 0{:}05$
OURS	$1{:}17 \pm 0{:}10$	$1{:}05 \pm 0{:}07$	$4{:}13 \pm 0{:}33$
BP	$0{:}9 \pm 0{:}02$	$0{:}10 \pm 0{:}02$	$0{:}22 \pm 0{:}03$

Table 12. Run time of the training experiments (*hours:minutes*) obtained on CIFAR-10 with BP and our DTP on a VGG-like architecture across 5 seeds.

	RUN TIME
BP	$0{:}17 \pm 0{:}03$
OURS	$6{:}03 \pm 0{:}39$