Neural Networks 169 (2024) 572–583

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

Low-variance Forward Gradients using Direct Feedback Alignment and momentum

Florian Bacho *, Dominique Chu *

CEMS, School of Computing, University of Kent, Canterbury, United Kingdom

ARTICLE INFO

Keywords:
Backpropagation
Low variance
Forward Gradient
Direct Feedback Alignment
Gradient estimates

ABSTRACT

Supervised learning in deep neural networks is commonly performed using error backpropagation. However, the sequential propagation of errors during the backward pass limits its scalability and applicability to low-powered neuromorphic hardware. Therefore, there is growing interest in finding local alternatives to backpropagation. Recently proposed methods based on forward-mode automatic differentiation suffer from high variance in large deep neural networks, which affects convergence. In this paper, we propose the Forward Direct Feedback Alignment algorithm that combines Activity-Perturbed Forward Gradients with Direct Feedback Alignment and momentum. We provide both theoretical proofs and empirical evidence that our proposed method achieves lower variance than forward gradient techniques. In this way, our approach enables faster convergence and better performance when compared to other local alternatives to backpropagation and opens a new perspective for the development of online learning algorithms compatible with neuromorphic systems.

1. Introduction

Over the past decades, the Backpropagation (BP) algorithm (Rumelhart et al., 1986) has emerged as a crucial technique for training Deep Neural Networks (DNNs). However, despite its success, BP has limitations that restrict its efficiency and scalability.

By sequentially propagating errors through multiple layers, BP limits the ability to parallelize the backward pass; this is often referred to as *Backward Locking* (Huo et al., 2018; Launay et al., 2020; Nøkland, 2016) and leads to time-consuming gradient computations. Secondly, BP relies on the transport of symmetric weights during the backward pass. Known as the *weight transport problem* (Akrout et al., 2019; Lillicrap et al., 2016; Nøkland, 2016), it is a source of significant power consumption on dedicated neural processors (Crafton et al., 2019; Han et al., 2019; Han & Yoo, 2019; Launay et al., 2020) and represents a major obstacle to the implementation of BP on low-powered continuous-time neuromorphic hardware (Neftci et al., 2017). Therefore, there is a growing need for alternatives to BP that can parallelize gradient computations without requiring global knowledge of the entire network.

In this context, several alternatives to BP have been proposed (Amato et al., 2019; Baydin et al., 2022; Belilovsky et al., 2019; Crafton et al., 2019; Han & jun Yoo, 2019; Hinton, 2022; Hjelm et al., 2019; Jabri & Flower, 1992; Jaderberg et al., 2017; Le Cun et al., 1988; Lillicrap et al., 2016; Löwe et al., 2019; Mostafa et al., 2018; Nøkland, 2016; Nøkland & Eidnes, 2019; Ren et al., 2023; Silver et al., 2021; Webster et al., 2021; Wen et al., 2018). For example, greedy learning can be used to sequentially learn complex representations of inputs (Belilovsky et al., 2019; Hinton, 2022; Löwe et al., 2019) or decoupled neural interfaces can be constructed to predict future gradients in an asynchronous manner (Jaderberg et al., 2017). Some approaches also introduce biologically inspired phenomenas such as Self-Backpropagation (SBP) that alternates between local unsupervised *sleep* phases and non-local supervised *wake* phases which reduces the computational cost of training (Zhang et al., 2022). Another example of BP alternative is the Random Feedback Alignment (FA) algorithm (Lillicrap et al., 2016). FA overcomes the weight transport problem by using random feedback weights for error backpropagation. Alternatively, the Random Direct Feedback Alignment (DFA) algorithm (Nøkland, 2016) directly projects output errors onto hidden neurons using fixed linear random feedback connections, removing the need for sequential propagation of errors and allowing parallel gradient computation. Other approaches such as Direct Random Target Propagation (DRTP) directly projects targets instead of output errors onto hidden layers to decrease the computational cost of DFA (Frenkel et al., 2021; Zhang et al., 2022). However, while FA, DFA and DRTP keep the feedback connections fixed, feedback learning mechanisms can be introduced to improve the

* Corresponding author.
E-mail addresses: f.bacho@kent.ac.uk (F. Bacho), d.f.chu@kent.ac.uk (D. Chu).

https://doi.org/10.1016/j.neunet.2023.10.051
Received 2 June 2023; Received in revised form 24 October 2023; Accepted 30 October 2023
Available online 4 November 2023

performance, as in the recently proposed Direct Kolen–Pollack (DKP) algorithm (Webster et al., 2021).

Methods based on Forward-Mode Automatic Differentiation have recently gained attention (Margossian, 2019). Referred to as *Forward Gradients* (Baydin et al., 2022; Ren et al., 2023; Silver et al., 2021), these techniques evaluate directional derivatives in random directions during the forward pass to compute unbiased gradient estimates without backpropagation. However, in large DNNs, forward gradients suffer from high variance which has a detrimental effect on convergence (Ren et al., 2023; Silver et al., 2021). While stochastic gradient descent (SGD) is guaranteed to converge for unbiased gradients estimators with sufficiently small learning rates (Robbins & Monro, 1951), theoretical analyses have shown that the convergence rate of SGD depends on the variance of the estimates (Bottou et al., 2018; Bubeck, 2015; Chee & Toulis, 2018; Faghri et al., 2020; Gower et al., 2019; Moulines & Bach, 2011; Murata, 1999; Needell et al., 2016). Estimators with low variance have less variability and are more consistent, leading to better convergence than those with high variance.

One approach for reducing the variance of Forward Gradients is to perturb neuron activations instead of weights (Ren et al., 2023). Because DNNs typically have fewer neurons than weights, perturbing neurons results in estimating fewer derivatives through forward gradients, leading to lower variance. Additionally, local greedy loss functions can be used to further reduce the variance, where each loss function trains only a small portion of the network. As demonstrated by Ren et al. (2023), this approach improves the convergence of Local MLP Mixers over the original FG algorithm. However, the method circumvents large gradient variance by ensuring that each local loss function only trains a small number of neurons. In the case where layers contain large numbers of neurons, the method would still suffer from high variance. Therefore, alternative solutions should be found to better reduce the variance of forward gradients in DNNs.

In this work, we propose the Forward Direct Feedback Alignment (FDFA) algorithm, a method that combines Activity-Perturbed Forward Gradients (Ren et al., 2023) with Direct Feedback Alignment (Nøkland, 2016) and momentum to computing low-variance gradient estimates. Our method addresses the limitations of BP by avoiding sequential error propagation and the transport of weights. We present theoretical and empirical results that demonstrate the effectiveness of FDFA in reducing the variance of gradient estimates, enabling fast convergence with DNNs. Compared to other Forward Gradient and Direct Feedback Alignment methods, FDFA achieves better performance with both fully-connected and convolutional neural networks, making it a promising alternative for scalable and energy-efficient training of DNNs.

2. Background

We start by reviewing the Forward Gradient algorithm (Baydin et al., 2022) and its DNN applications (Ren et al., 2023; Silver et al., 2021) as well as Direct Feedback Alignment (Nøkland, 2016), which form the technical foundation of the FDFA algorithm.

2.1. Forward Gradient (FG)

The Forward Gradient (FG) algorithm (Baydin et al., 2022; Silver et al., 2021) is a recent weight perturbation technique that uses Forward-Mode Automatic Differentiation (AD) (Margossian, 2019) to estimate gradients without backpropagation. Consider a differentiable function $\boldsymbol{f} : \mathbb{R}^m \mapsto \mathbb{R}^n$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$, Forward-Mode AD evaluates the directional gradient $\boldsymbol{d} = \boldsymbol{J} \cdot \boldsymbol{v}$ of $\boldsymbol{f}$ in the direction $\boldsymbol{v}$. Here, $\boldsymbol{J} \in \mathbb{R}^{n \times m}$ is the Jacobian matrix of f, and $\boldsymbol{d}$ is obtained by computing the matrix–vector product between $\boldsymbol{J}$ and $\boldsymbol{v}$ during the function evaluation.

By sampling each element of the direction vector $v_i \sim \mathcal{N}(0, 1)$ from a standard normal distribution and multiplying back each computed directional derivatives by $\boldsymbol{v}$, an unbiased estimate of the Jacobian is computed:

$$\mathbb{E}\left[\boldsymbol{d} \otimes \boldsymbol{v}\right] = \mathbb{E}\left[(\boldsymbol{J} \cdot \boldsymbol{v}) \otimes \boldsymbol{v}\right] = \boldsymbol{J} \tag{1}$$

Here, $\otimes$ is the outer product. See Baydin et al. (2022) or Theorem 3 in Appendix for the proof of unbiasedness and Fig. 1 for a visual representation of forward gradients.

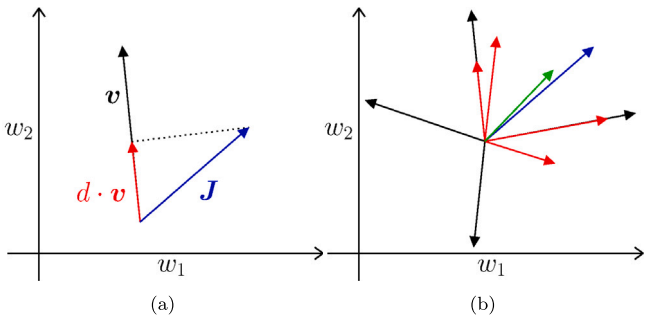


Fig. 1. Fig. 1(a): Projection of the Jacobian J at $\boldsymbol{w}$ onto a given direction $\boldsymbol{v}$. The vector $d \cdot \boldsymbol{v}$ is obtained by scaling the direction $\boldsymbol{v}$ by the directional derivative d evaluated at $\boldsymbol{w}$ in the direction of $\boldsymbol{v}$. Fig. 1(b): The expected directional derivative (green arrow), computed by averaging directional gradients (red arrows) over many random directions (black arrows), is an unbiased estimate of the true gradient (blue arrow).

2.2. Weight-Perturbed Forward Gradient (FG-w)

When the FG algorithm is applied to DNNs, a random perturbation matrix $\boldsymbol{V}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is drawn from a standard normal distribution for each weight matrix $\boldsymbol{W} \in \mathbb{R}^{n_l \times n_{l-1}}$ of each layer $l \leq L$. Forward-Mode AD thus computes the directional derivative $d^{\mathrm{FG-W}}$ of the loss $\mathcal{L}(\boldsymbol{x})$ along the drawn perturbation, such as:

$$d^{\mathrm{FG-W}} = \sum_{l=1}^{L} \sum_{i=1}^{n_l} \sum_{j=1}^{n_{l-1}} \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(l)}} v_{i,j}^{(l)} \tag{2}$$

where $v_{i,j}^{(l)}$ is the element of the perturbation matrix $V^{(l)}$ in row i and column j, associated with the weight $w_{i,j}^{(l)}$.

Thus referred to as Weight-Perturbed Forward Gradient (FG-W) (Ren et al., 2023), the gradient estimate $g^{\mathrm{FG-W}}(\boldsymbol{W}^{(l)})$ for the weights of the layer l is obtained by scaling its perturbation $\boldsymbol{V}^{(l)}$ matrix with the directional derivative $d^{\mathrm{FG-W}}$:

$$g^{\mathrm{FG-W}}\left(\boldsymbol{W}^{(l)}\right) := d^{\mathrm{FG-W}} \boldsymbol{V}^{(l)} \tag{3}$$

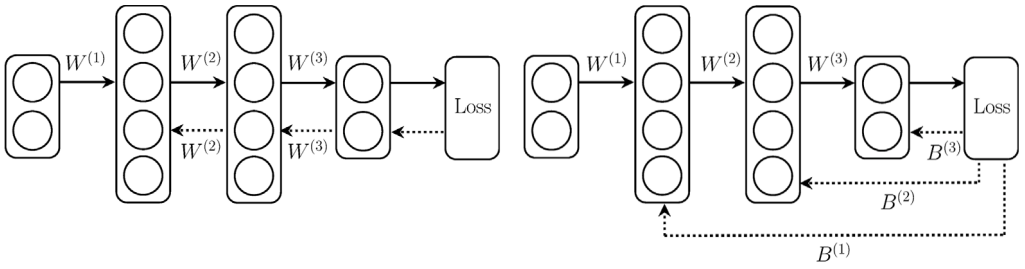
See Algorithm 2 in Appendix for the full algorithm applied to a fully-connected DNN.

It is essential to understand that the directional derivative $d^{\mathrm{FG-W}}$, takes the form of a *scalar*. This scalar is defined as the result of the vector product between the true gradient and the given direction that is implicitly evaluated with Forward-Mode AD. Unlike Reverse-Mode AD, where the individual explicit derivatives constituting the gradient are accessible, Forward-Mode AD does not provide them directly. Therefore, multiplying the directional derivative by the given direction is an essential step for computing an unbiased approximation of the gradient. For more details, refer to Baydin et al. (2022).

However, it has been previously shown that the variance of FG-W scales poorly with the number of parameters in DNNs which impacts the convergence of SGD (Ren et al., 2023).

2.3. Activity-Perturbed Forward Gradient (FG-a)

To address the variance issues of FG-W, Ren et al. proposed the *Activity-Perturbed FG* (FG-A) algorithm that perturbs neurons activations instead of weights. A perturbation vector $\boldsymbol{u}^{(l)} \in \mathbb{R}^{n_l}$ is drawn from



(a) Error Backpropagation (b) Direct Feedback Alignment

Fig. 2. Illustrations of the error backpropagation (Fig. 2(a)) and Direct Feedback Alignment (Fig. 2(b)). Solid arrows represent forward paths and dotted arrows represent backpropagation paths.

a multivariate standard normal distribution for each layer $l \leq L$. Forward-Mode AD thus computes the following directional derivative:

$$d^{\mathrm{FG-A}} = \sum_{l=1}^{L} \sum_{i=1}^{n_l} \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial y_i^{(l)}} u_i^{(l)} \tag{4}$$

where $\boldsymbol{y}^{(l)}$ are the activations of the lth layer. Note that directional derivatives computed in the FG-A algorithm are defined as the sum over all neurons rather than all weights. The activity-perturbed forward gradient $g^{\mathrm{FG-A}}\left(\boldsymbol{W}^{(l)}\right)$ is then:

$$g^{\mathrm{FG-A}}\left(\boldsymbol{W}^{(l)}\right) := \left(d^{\mathrm{FG-A}}\boldsymbol{u}^{(l)}\right) \frac{\partial \boldsymbol{y}^{(l)}}{\partial \boldsymbol{W}^{(l)}} \tag{5}$$

where $\partial \boldsymbol{y}^{(l)} / \partial \boldsymbol{W}^{(l)}$ are local gradients computed by neurons with locally-available information. See Algorithm 3 in Appendix for the full algorithm applied to a fully-connected DNN.

This method reduces the number of derivatives to estimate since the number of neurons is considerably lower than the number of weights (see Table B.5 in Appendix for some examples). Consequently, the method leads to lower variance than FG-W (Ren et al., 2023).

Activity-Perturbed FG demonstrated improvements over Weight-Perturbed FG on several benchmark datasets (Ren et al., 2023) but still suffers from high variance as the number of neurons in DNNs remains large. To avoid this issue, Ren et al. proposed the Local Greedy Activity-Perturbed Forward Gradient (LG-FG-A) method, which uses local loss functions to partition the gradient computation and decrease the number of derivatives to estimate. By adopting this local greedy strategy, LG-FG-A greatly improved the performance of Local MLP Mixers, a specific architecture that uses shallow multi-layer perceptrons to perform vision without having to use convolution. However, no results were reported for conventional fully-connected or convolutional neural networks, where the number of neurons per layer is larger than in MLP Mixers.

2.4. Direct Feedback Alignment

While BP relies on symmetric weights to propagate errors to hidden layers, it has been shown that weight symmetry is not mandatory to achieve learning (Lillicrap et al., 2016). For example, FA has proven that random fixed weights can also be used for backpropagating errors and still achieve learning (Lillicrap et al., 2016).

DFA takes the idea of FA one step further by directly projecting output errors to hidden layers using fixed linear feedback connections (Nøkland, 2016) (see Fig. 2). Feedback matrices $\boldsymbol{B}^{(l)} \in \mathbb{R}^{n_L \times n_l}$ replace the derivatives $\partial \boldsymbol{y}^{(L)} / \partial \boldsymbol{y}^{(l)}$ of output neurons with respect to hidden neurons. The approximate gradient $g^{\mathrm{DFA}}\left(\boldsymbol{W}^{(l)}\right)$ for the weights of the hidden layer l is then computed as follows:

$$g^{\mathrm{DFA}}\left(\boldsymbol{W}^{(l)}\right) := \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \boldsymbol{y}^{(L)}} \boldsymbol{B}^{(l)} \frac{\partial \boldsymbol{y}^{(l)}}{\partial \boldsymbol{W}^{(l)}} \tag{6}$$

In DFA, feedback matrices for hidden layers are chosen to be random and kept fixed during training. For the output layer, the identity matrix is used as no propagation of errors is required.

The success of DFA depends on the alignment between the forward and feedback weights, which results in the alignment between the approximate and true gradient (Lillicrap et al., 2016; Nøkland, 2016; Refinetti et al., 2021). When the angle between these gradients is within 90 degrees, the direction of the update is descending (Lillicrap et al., 2016; Nøkland, 2016).

DFA can scale to modern deep learning architectures such as Transformers (Launay et al., 2020) but is unable to train deep convolution layers (Launay et al., 2019) and fails to learn challenging datasets such as CIFAR-100 or ImageNet without the use of transfer learning (Bartunov et al., 2018; Crafton et al., 2019). However, recent methods to learn symmetric feedbacks such as the Direct Kolen–Pollack (DKP) algorithm (Webster et al., 2021) showed promising results with convolutional neural networks due to improved gradient alignments.

3. Method

Algorithm 1 Forward Direct Feedback Alignment algorithm with a fully-connected DNN.

Input: Training data $\mathcal{D}$
Randomly initialize $w_{ij}^{(l)}$ for all l, i and j.
Initialize $\boldsymbol{B}^{(l)} = \boldsymbol{0}$ for all l.
repeat
  {Inference (sequential)}
  for all $\boldsymbol{x}$ **in** $\mathcal{D}$ **do**
    $\boldsymbol{y}^{(0)} \leftarrow \boldsymbol{x}_s$
    $\boldsymbol{d}^{(0)} \leftarrow \boldsymbol{0}$
    for $l = 1$ **to** L **do**
      Sample $\boldsymbol{v}^{(l)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
      $\boldsymbol{a}^{(l)} \leftarrow \boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)}$
      $\boldsymbol{y}^{(l)} \leftarrow \sigma\left(\boldsymbol{a}^{(l)}\right)$
      $\boldsymbol{d}^{(l)} \leftarrow \left(\boldsymbol{W}^{(l)} \boldsymbol{d}^{(l-1)}\right) \odot \sigma'\left(\boldsymbol{a}^{(l)}\right)$
      if $l < L$ **then**
        $\boldsymbol{d}^{(l)} \leftarrow \boldsymbol{d}^{(l)} + \boldsymbol{v}^{(l)}$
      end if
    end for
    $\boldsymbol{e} \leftarrow \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \boldsymbol{y}^{(L)}}$
    {Weights updates (parallel)}
    $\boldsymbol{W}^{(L)} \leftarrow \boldsymbol{W}^{(L)} - \lambda\, \boldsymbol{e} \otimes \boldsymbol{y}^{(L-1)}$
    for $l = 1$ **to** $L - 1$ **do**
      $\boldsymbol{B}^{(l)} \leftarrow (1 - \alpha)\, \boldsymbol{B}^{(l)} - \alpha \left(\boldsymbol{d}^{(L)} \otimes \boldsymbol{v}^{(l)}\right)$
      $\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \lambda \left(\boldsymbol{e}\boldsymbol{B}^{(l)} \odot \sigma'\left(\boldsymbol{a}^{(l)}\right)\right) \otimes \boldsymbol{y}^{(l-1)}$
    end for
  end for
until $\mathbb{E}[\mathcal{L}(\boldsymbol{x})] < \epsilon$

In this section, we describe our proposed Forward Direct Feedback Alignment (FDFA) algorithm which uses forward gradients to estimate derivatives between output and hidden neurons as direct feedback connections.

Similarly to FG-A, we sample perturbation vectors $\boldsymbol{u}^{(l)} \in \mathbb{R}^{n_l}$ for each layer $l \leq L$ from a multivariate standard normal distribution. However, in contrast to FG-A, we use the directional derivatives that are computed at the output layer rather than at the loss function, such as:

$$\boldsymbol{d}^{\text{FDFA}} = \sum_{l=1}^{L-1} \sum_{i=1}^{n_l} \frac{\partial \boldsymbol{y}^{(L)}}{\partial y_i^{(l)}} u_i^{(l)} \tag{7}$$

which produces a vector of directional derivatives. Note that only partial derivatives of the network outputs with respect to the activations of hidden neurons are considered, as represented by feedback connections in DFA. Output neurons are updated using a fixed identity feedback matrix as no error backpropagation is required.

Rather than relying solely on the most recent forward gradient, as in FG-W and FG-A, it is possible to obtain a more accurate estimate of $\frac{\partial \boldsymbol{y}^{(L)}}{\partial \boldsymbol{y}^{(l)}}$ by averaging the forward gradient over the past training steps.

Formally, we define an update rule for the direct feedback connections of DFA that performs an exponential average of the estimates, such as:

$$\boldsymbol{B}^{(l)} \leftarrow (1-\alpha)\, \boldsymbol{B}^{(l)} + \alpha\, \boldsymbol{d}^{\text{FDFA}} \otimes \boldsymbol{u}^{(l)} \tag{8}$$

where $0 < \alpha < 1$ is the feedback learning rate. Eq. (8) can also be re-written in a form compatible with the stochastic gradient descent algorithm:

$$\boldsymbol{B}^{(l)} \leftarrow \boldsymbol{B}^{(l)} - \alpha \nabla \boldsymbol{B}^{(l)} \tag{9}$$

where

$$\nabla \boldsymbol{B}^{(l)} = \boldsymbol{B}^{(l)} - \boldsymbol{d}^{\text{FDFA}} \otimes \boldsymbol{u}^{(l)} \tag{10}$$

In essence, Eq. (9) minimizes the following Mean Squared Error (MSE) function:

$$\mathcal{L}_B(\boldsymbol{x}) = \mathbb{E}\left[\sum_{l=1}^{L-1} \sum_{o=1}^{n_L} \sum_{i=1}^{n_l} \left(B_{o,j}^{(l)} - d_o^{\text{FDFA}} u_i^{(l)}\right)^2\right] \tag{11}$$

where $d_o^{\text{FDFA}} u_i^{(l)}$ is the target value for the optimized feedback connection $B_{o,j}^{(l)}$. The global minimum of this loss function occurs at the point where all feedback connections are equal to the expected derivative between output and hidden neurons, such as:

$$\boldsymbol{B}_{o,i}^{(l)} = \mathbb{E}\left[d_o^{\text{FDFA}} u_i^{(l)}\right] = \mathbb{E}\left[\frac{\partial y_o^{(L)}}{\partial y_i^{(l)}}\right] \tag{12}$$

for all output o and all hidden neuron i of all hidden layers $l < L$. Therefore, the FDFA algorithm is a dual optimization procedure where both loss functions $\mathcal{L}(\boldsymbol{x})$ and $\mathcal{L}_B(\boldsymbol{x})$ are minimized concurrently: weights are updated with direct feedbacks to minimize prediction errors and feedbacks connections converge towards the derivatives between output and hidden neurons. In the case where output neurons are linear, the penultimate feedback matrix becomes symmetric with the output weight matrix:

$$\mathbb{E}\left[\boldsymbol{B}^{(L-1)}\right] = \mathbb{E}\left[\frac{\partial \boldsymbol{y}^{(L)}}{\partial \boldsymbol{y}^{(L-1)}}\right] = \boldsymbol{W}^{(L)} \tag{13}$$

In this particular case, gradient estimates become mathematically equivalent to BP. For lower hidden layers, feedback matrices linearly approximate the non-linear derivatives between output and hidden neurons, which introduce a bias in the gradient estimates. This is because the feedback learning rule introduced in FDFA acts as momentum for these derivatives (see Eq. (8)), which is known to mitigate the effect of gradient variance on convergence at the cost of increased bias (Defazio, 2020).

Finally, the FDFA gradient estimate $g^{\text{FDFA}}\left(\boldsymbol{W}^{(l)}\right)$ for the weights $\boldsymbol{W}^{(l)}$ of the hidden layer l is computed as in DFA. Formally, output errors are linearly projected onto hidden neurons using the feedback matrix $B^{(l)}$, such as:

$$g^{\text{FDFA}}\left(\boldsymbol{W}^{(l)}\right) := \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \boldsymbol{y}^{(L)}} \boldsymbol{B}^{(l)} \frac{\partial \boldsymbol{y}^{(l)}}{\partial \boldsymbol{W}^{(l)}} \tag{14}$$

Table 1
Performance of 4-layers fully connected DNNs trained on the MNIST, Fashion MNIST and CIFAR10 dataset.

Method	Local	MNIST	Fashion MNIST	CIFAR10
BP	✗	98.46 ± 0.05%	89.99 ± 0.15%	56.92 ± 0.19%
FG-W	✓	80.03 ± 0.62%	71.72 ± 0.63%	25.17 ± 0.32%
FG-A		83.40 ± 1.33%	77.62 ± 0.25%	35.53 ± 0.21%
LG-FG-A		86.89 ± 0.90%	77.98 ± 0.21%	34.56 ± 0.34%
DFA		98.15 ± 0.05%	89.15 ± 0.09%	53.61 ± 0.16%
DKP		98.27 ± 0.05%	89.48 ± 0.09%	54.35 ± 0.21%
FDFA		**98.32 ± 0.05%**	**89.56 ± 0.11%**	**54.97 ± 0.19%**

Table 2
Performance of a shallow CNN (15C5-P2-40C5-P2-128-10 where 15C5 represents 15 5×5 convolutional layers and P2 represents a 2×2 max pooling layer) on the MNIST, Fashion MNIST and CIFAR10 datasets.

Method	Local	MNIST	Fashion MNIST	CIFAR10
BP	✗	99.32 ± 0.03%	92.10 ± 0.16%	68.11 ± 0.57%
FG-W	✓	87.63 ± 1.20%	74.93 ± 0.55%	33.10 ± 0.37%
FG-A		96.63 ± 0.31%	82.71 ± 0.89%	42.51 ± 0.32%
LG-FG-A		95.42 ± 0.72%	79.96 ± 0.94%	37.74 ± 2.41%
DFA		98.80 ± 0.08%	89.69 ± 0.22%	58.14 ± 0.94%
DKP		99.08 ± 0.03%	90.39 ± 0.25%	60.30 ± 0.34%
FDFA		**99.16 ± 0.03%**	**91.54 ± 0.11%**	**66.96 ± 0.70%**

The FDFA gradient estimate is then used to update the weights of the network with stochastic gradient descent, such as:

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \lambda\, g^{\text{FDFA}}\left(\boldsymbol{W}^{(l)}\right) \tag{15}$$

where $\lambda > 0$ is a learning rate. Note that other gradient-based optimization techniques can also be applied for both feedback and forward weight updates in place of SGD, such as Adam (Kingma & Ba, 2017).

The full algorithm applied to a fully connected DNN is given in Algorithm 1.

4. Results

We now present detailed theoretical and empirical results with our proposed FDFA algorithm and other local alternatives to BP. Details about our experimental settings are given in Appendix A.

4.1. Performance

We compared the performance of the proposed FDFA method with various local alternatives to BP related to our algorithm, including Weight-Perturbed Forward Gradient (FG-W), activity-perturbed Forward Gradient (FG-A), Local-Greedy activity-perturbed Forward Gradient (LG-FG-A), Random Direct Feedback Alignment (DFA) and the Direct Kolen–Pollack (DKP) algorithm. We evaluated each approach using fully-connected DNNs and Convolutional Neural Networks, including the AlexNet architecture (Krizhevsky et al., 2012). We used several datasets widely adopted as standard benchmarks for training algorithm comparisons and presenting increasing levels of difficulty, namely MNIST, Fashion MNIST, CIFAR10, CIFAR100, and Tiny ImageNet 200. For each method, average test performance over 10 training runs is reported in Tables 1, 2, and 3. Additional results with fully-connected DNNs of different depths are also given in Table B.6 in Appendix B.

Overall, the FG-W algorithm achieves poor generalization compared to BP. More importantly, its performance significantly decreases with the size of the network and the complexity of the task. For example, the method is unable to converge with AlexNet on the CIFAR100 or Tiny ImageNet 200 datasets with 3.43% and 0.70% test accuracy respectively. The FG-A method slightly improves the generalization of forward gradients but still fails to match the performance of BP. The greedy approach used in LG-FG-A further improves performance when

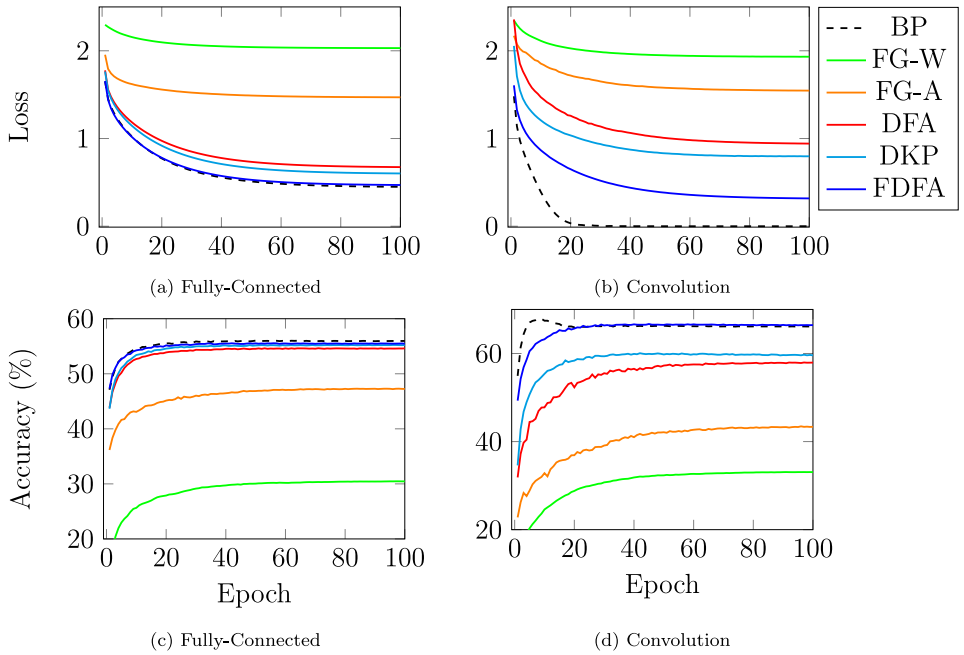


Fig. 3. Training loss and test accuracy of a 2-layer fully-connected network (Figs. 3(a) and 3(c)) and a CNN (Figs. 3(b) and 3(d)) trained on the CIFAR10 dataset. FDFA has a similar convergence rate as BP on fully-connected networks. With CNNs, FDFA is not able to overfit the training data. However, our method has the highest convergence rate compared to FG-W, FG-A, and DFA.

Table 3
Performance of AlexNet (Krizhevsky et al., 2012) trained on the CIFAR100 and Tiny ImageNet 200 datasets.

Method	Local	CIFAR100	Tiny ImageNet
BP	✗	60.43 ± 0.35%	40.55 ± 0.31%
FG-W		3.43 ± 0.43%	0.70 ± 0.09%
FG-A		15.84 ± 0.32%	3.52 ± 0.32%
LG-FG-A	✓	15.27 ± 0.29%	2.90 ± 0.14%
DFA		35.75 ± 0.58%	17.47 ± 0.34%
DKP		49.15 ± 0.34%	25.36 ± 0.82%
FDFA		**57.27 ± 0.11%**	**36.47 ± 0.40%**

the number of neurons per layer is relatively low, as in fully-connected networks for MNIST and Fashion MNIST. However, the method does not perform as well as FG-A on networks that contain large layers such as convolution. This suggests, that LG-FG-A is most suitable for specific architectures where each local loss function sends error signals to a small number of neurons.

DFA achieves performance closer to BP than forward gradient methods. However, it fails to scale to AlexNet on CIFAR100 and Tiny ImageNet 200, as observed in previous work (Bartunov et al., 2018; Crafton et al., 2019; Launay et al., 2019). The performance of DFA is increased by the DKP algorithm but a gap still exists with BP, especially on difficult tasks such as CIFAR100 ad Tiny ImageNet 200. In contrast, our proposed FDFA method performs closer to BP than DFA and DKP on all benchmarked networks and datasets. For example, our method doubles the test accuracy of DFA and improves by at least 10% the performance of DKP on Tiny ImageNet 200.

4.2. Convergence

To evaluate the convergence improvements of our method, we measured the evolution of the training loss and test accuracy during the training process. As shown in Fig. 3, the FG-W algorithm converges slowly compared to both DFA and BP. Although FG-A slightly improves the convergence of FG-W, it was still unable to converge as quickly as BP and DFA. Both the DKP and FDFA algorithms showed better convergence rates than FG-W, FG-A, and DFA. However, DKP seems to be unable to reduce the loss as low as BP with convolutional layers. Finally, our proposed FDFA algorithm achieved a similar convergence rate as BP with fully connected networks and substantially improved the convergence rate of DFA and DKP with CNNs.

To also assess the rate of convergence in relation to the computational cost of each method, we measured the evolution of the training loss and test accuracy as a function of each algorithms' runtime. Fig. 4 shows the evolution of these metrics over time during the training a CNN on the CIFAR10 dataset, using an Intel Core i7 6700 CPU and NVIDIA GeForce RTX 3060 GPU. The proposed FDFA algorithm appears to converge more slowly on CPUs than on GPUs when compared the other alternatives to BP. Nevertheless, it is noteworthy that, on both CPUs and GPUs, FDFA outperforms all other methods in terms of convergence rate.

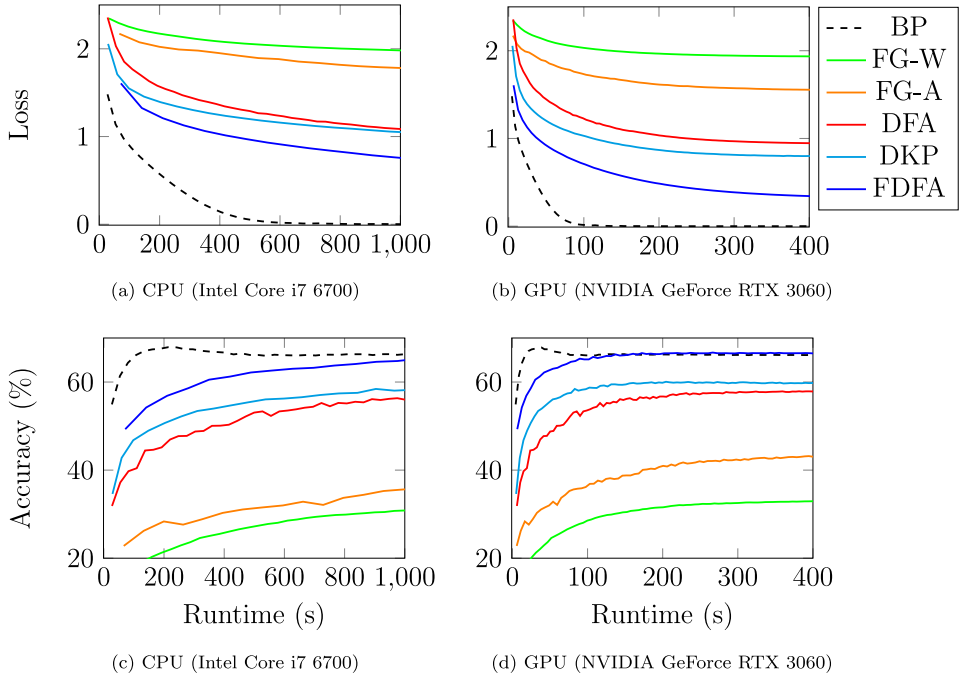
4.3. Variance of gradient estimates

To gain a deeper understanding of the reasons for the improvements observed in FDFA compared to FG-W and FG-A, we compare the theoretical variances of each method and empirically demonstrate the impact of gradient variance in the convergence of SDG.

Following Wen et al. (2018) and Ren et al. (2023), it can be shown that the variance of unbiased gradient estimates can be decomposed into two terms.

Proposition 1 (*Ren et al., 2023; Wen et al., 2018*). *Let $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n_1,n_0}$ be the hidden weights of a two-layers fully-connected neural network. We denote by $\boldsymbol{x}$ the independent input samples and by $\boldsymbol{v}^{(1)}$ the independent random perturbations used to estimate the gradients of the hidden layer. The variance of a gradient estimate $g\left(w_{i,j}^{(1)}\right)$ for the hidden weight $w_{i,j}^{(1)}$ can be decomposed into two parts:*

$$\operatorname{Var}\left[g\left(w_{i,j}^{(1)}\right)\right] = z_1 + z_2 \tag{16}$$



(a) CPU (Intel Core i7 6700) (b) GPU (NVIDIA GeForce RTX 3060)

(c) CPU (Intel Core i7 6700) (d) GPU (NVIDIA GeForce RTX 3060)

Fig. 4. Training loss and test accuracy of a CNN trained on CIFAR-10 as a function of runtime on both an Intel Core i7 6700 CPU (Figs. 4(a) and 4(c)) and a NVIDIA GeForce RTX 3060 GPU (Figs. 4(b) and 4(d)). Figs. 4(a) and 4(c) show the impact of the higher computational cost of FDFA when gradient estimates are sequentially computed on a CPU. In contrast, Figs. 4(b) and 4(d) demonstrate that the parallel aspect of FDFA on GPUs mitigates the impact of this computational cost on convergence.

where

$$z_1 := \operatorname*{Var}_{\boldsymbol{x}}\left[\operatorname*{\mathbb{E}}_{\boldsymbol{v}^{(1)}}\left[g\left(w_{i,j}^{(1)}\right) \mid \boldsymbol{x}\right]\right] \tag{17}$$

and

$$z_2 := \operatorname*{\mathbb{E}}_{\boldsymbol{x}}\left[\operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[g\left(w_{i,j}^{(1)}\right) \mid \boldsymbol{x}\right]\right] \tag{18}$$

The first term z_1 defined in Proposition 1 corresponds to the gradient variance from data sampling. This term vanishes with the size of the batch in the case of mini-batch learning (Ren et al., 2023; Wen et al., 2018). The second term z_2 corresponds to the expected additional variance produced by the stochastic estimation of the gradient, which scales differently for each algorithm. In the case of BP and DFA, this term equals zero as both algorithms are deterministic. In Ren et al. (2023), a third term z_3 was considered, which corresponds to the correlation between gradient estimates. However, this term is zero in the case where perturbations are independent (Ren et al., 2023).

Next, we prove that the gradient estimation variance z_2 for the proposed FDFA algorithm vanishes as α tends towards zero.

Proposition 2. *Let $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n_1,n_0}$ be the hidden weights of a two-layers fully-connected neural network evaluated with an input sample $\boldsymbol{x} \in \mathbb{R}^{n_0}$. We denote by $g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right)$ the FDFA gradient estimate for the weight $w_{i,j}^{(l)}$ and assume that all the elements of the perturbation vector $\boldsymbol{u}^{(2)}$ for the activations of the output layer are 0. We also assume that all derivatives $\left(\partial\mathcal{L}(\boldsymbol{x})/\partial w_{i,j}^{(l)}\right)^2 \leq \beta$ are bounded and that the feedback matrix $\boldsymbol{B}^{(1)}$ converged to $\partial y^{(2)}/\partial y^{(1)} = \boldsymbol{W}^{(1)}$, making the gradient estimates unbiased. If each element $u_i^{(1)} \sim \mathcal{N}(0,1)$ of $\boldsymbol{u}^{(1)}$ follows a standard normal distribution, then:*

$$\operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right) \mid \boldsymbol{x}\right] = \alpha^2 \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[\sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)}\right]\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2 \tag{19}$$

and

$$\lim_{\alpha\to 0} \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right) \mid \boldsymbol{x}\right] = 0 \tag{20}$$

Proof. Starting from the gradient estimate $g^{\mathrm{FG-A}}\left(w_{i,j}^{(1)}\right)$, we have:

$$\begin{aligned} g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right) &= \sum_{o=1}^{n_2} \frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_o^{(2)}} b_{o,i}^{(1)} \frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}} \\ &= \sum_{o=1}^{n_2} \frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_o^{(2)}} \left(\alpha \sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)} + (1-\alpha)\frac{\partial y_o^{(2)}}{\partial y_i^{(1)}}\right) \frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}} \\ &= (1-\alpha)\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}} + \alpha \sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)} \frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}} \end{aligned} \tag{21}$$

Because both $\partial\mathcal{L}(\boldsymbol{x})/\partial w_{i,j}^{(1)}$ and $\partial y_i^{(1)}/\partial w_{i,j}^{(1)}$ are constants, the variance of $g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right)$ given an input sample $\boldsymbol{x}$ reduces to:

$$\begin{aligned} \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right) \mid \boldsymbol{x}\right] =& \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[(1-\alpha)\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}} + \alpha \sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)} \frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}} \mid \boldsymbol{x}\right] \\ =& \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[(1-\alpha)\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}} \mid \boldsymbol{x}\right] \\ &+ \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[\alpha \sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)} \frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}} \mid \boldsymbol{x}\right] \\ =& \alpha^2 \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[\sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)} \mid \boldsymbol{x}\right]\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2 \end{aligned} \tag{22}$$

Therefore:

$$\begin{aligned} \lim_{\alpha\to 0} \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[g^{\mathrm{FDFA}}\left(w_{i,j}^{(1)}\right) \mid \boldsymbol{x}\right] &= \lim_{\alpha\to 0} \alpha^2 \operatorname*{Var}_{\boldsymbol{v}^{(1)}}\left[\sum_{k=1}^{n_1} \frac{\partial y_o^{(1)}}{\partial y_k^{(1)}} u_k^{(1)} u_i^{(1)} \mid \boldsymbol{x}\right]\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2 \\ &= 0 \end{aligned} \tag{23}$$

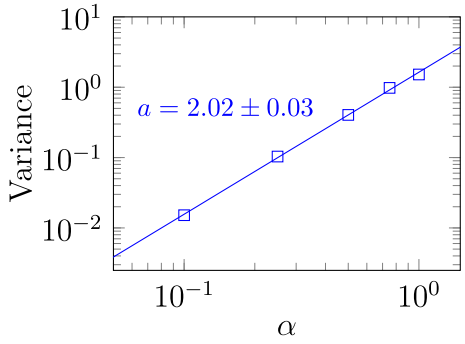


Fig. 5. The variance of FDFA gradient estimates as a function of the feedback learning rate α in a two-layer fully connected network. Each point was produced by computing the variance of gradient estimates over 10 iterations of the MNIST dataset. The blue line was fitted using linear regression. The slope a and the asymptotic standard error is given with the same color. This figure shows that the variance of FDFA scales quadratically with α. Note that both axes have a logarithmic scale. Therefore, the slope of the straight line on a logarithmic scale gives the power of the relationship on a linear scale.

Table 4
Theoretical gradient estimation variance produced by the FG-W, FG-A and FDFA algorithm given a single input sample.

Algorithm	Local	Estimation variance
BP	✗	0
FG-W	✓	$O(n_1 n_0)$
FG-A		$O(n_1)$
DFA		0
FDFA ($\alpha \to 0$)		0

which concludes the proof.

We also derived the gradient estimation variance for the FG-W (Proposition 5) and FG-A (Proposition 6) algorithms in Appendices G and H respectively. Numerical verifications are provided in Figs. 5 and 6. Table 4 shows the comparison of these theoretical results.

Proposition 2 analytically shows that the gradient estimation variance of FDFA estimates quadratically vanishes as α tends towards zero (see Fig. 5 for numerical verifications). Therefore, by choosing values of α close to zero, the variance of FDFA estimates approaches the variance of the true gradient.

In contrast, Propositions 5 and 6 show that the variance of FG-W and FG-A have different scaling with respect to the number of neurons and parameters in the network. FG-W scales linearly with the number of parameters and FG-A scales linearly with the number of neurons. This indicates that, in large DNNs, gradient estimates provided by the FG-W algorithm have a larger variance than FG-A estimates. However, as DNNs also contain large numbers of neurons, the variance of FG-A remains high.

To understand the impact of gradient variance on the convergence of SGD, we conducted an experiment where we measured the training loss of DNNs after one epoch as a function of variance. We simulated the variance of forward gradients by injecting Gaussian noise into the gradient computed with BP. By increasing the standard deviation of the injected noise, we artificially generated gradients with varying levels of variance. We then trained a two-layer DNN with these noisy gradients and measured the loss after one epoch of training on the MNIST dataset.

Fig. 7 illustrates that the variance strongly influences the training loss achieved after one epoch, as low-variance gradients tend to converge towards lower loss values compared to high-variance gradients. Hence, in this context, variance is the determining factor for convergence. Additionally, Fig. 7 shows the variance and loss values associated with the FG-W, FG-A, and FDFA algorithms. Notably, all data points align with the line formed by the noisy BP. This shows that the differences in gradient variance are solely responsible for the differences in convergence among the FG-W, FG-A, and FDFA algorithms.

4.4. Gradient alignment in feedback methods

In our experiments, we observed that the proposed FDFA algorithm exhibits better convergence compared to DFA, despite having similar variance. This suggests that their gradient estimation variance alone does not fully explain the difference in convergence between these two algorithms.

Both FDFA and DFA exhibit different degrees of biasedness. On the one hand, the FDFA algorithm introduces bias through its feedback learning rule, which acts as momentum for the derivatives between output and hidden neurons. On the other hand, DFA uses random feedback connections, resulting in strongly biased gradient estimates. Therefore, we conjecture that the disparity in convergence between DFA and the proposed FDFA algorithm can be attributed to their respective bias.

To test this, we measured the bias of each method during training by recording the angle between the true gradient and the averaged gradient estimates produced by DFA, FDFA, and the DKP algorithm. This measure, often referred to as *gradient alignment* (Refinetti et al., 2021; Webster et al., 2021), indicates the extent to which the expected gradient estimate deviates from the true gradient.

Fig. 8 shows the evolution of the layer-wise gradients alignment of each algorithm in a 5-layer fully connected DNN trained on the MNIST dataset during 100 epochs. This figure shows that the proposed FDFA method achieves faster alignment with the true gradient compared to DFA and DKP, which suggests earlier descending updates. Remarkably, the alignment of the gradient in FDFA is globally enhanced by 30 degrees in comparison to DFA and DKP. This significant improvement suggests that our method provides less biased gradient estimates. Thus, these findings strongly support the fact that the differences in convergence between DFA, DKP, and FDFA can be attributed to their respective levels of biasedness.

5. Discussion

The increasing size of DNNs, coupled with the emergence of low-powered neuromorphic hardware, has highlighted the need to explore alternative training methods that can overcome the limitations associated with BP. As a result, the exploration of local and parallel alternatives to BP has gained significant importance.

In this work, we proposed the FDFA algorithm that combines the FG-A algorithm with DFA and momentum to train DNNs without relying on BP. The algorithm involves a dual optimization process where weights are updated with direct feedback connections to minimize classification errors, and derivatives are estimated as feedback using activity-perturbed forward gradients. Our experiments showed that the FDFA gradient estimate closely aligns with the true gradient, particularly in the last layers. However, the sole difference between the FDFA estimate and the true gradient are the feedback connections replacing the derivatives between outputs and hidden neurons. Hence, the convergence of feedback connections toward these derivatives is the only factor responsible for this gradients alignment. We can thus conclude that the proposed feedback learning rule is capable of learning the relevant derivatives to approximate BP. Moreover, the averaging process introduced in our feedback learning rule acts as momentum for the derivatives estimates between output and hidden neurons. Our results demonstrated that the introduce momentum significantly reduces the gradient estimation variance compared to other forward gradient methods. Consequently, our method provides more accurate gradient estimates, leading to improved convergence and better performances on several benchmark datasets and architectures.

Using our proposed method, feedback connections in the penultimate layer become equal to the output weights. This results in gradient estimates that are equivalent to BP. However, in deeper layers, feedback connections linearly approximate the non-linear derivatives between output and hidden neurons, introducing a bias. To address this bias, the FDFA algorithm could be adapted to Feedback Alignment, where feedback matrices replace the weights during the backward

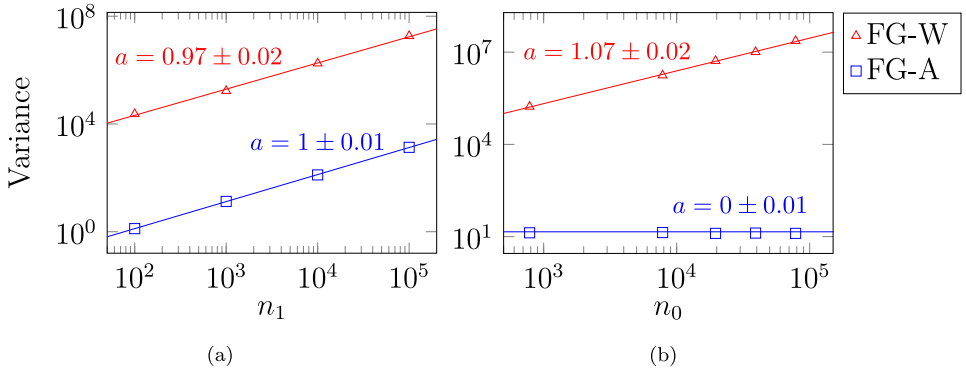


Fig. 6. Variance of the FG-W (red triangles) and FG-A (blue squares) gradient estimates as a function of the number of neurons n_1 (Fig. 6(b)) and number of inputs n_0 (Fig. 6(a)) in a two-layer fully connected network. Each point was produced by computing the variance of gradient estimates over 10 iterations of the MNIST dataset. The pixels of input images were duplicated, to increase the number of inputs n_0. The red and blue lines were fitted using linear regression. The slope a and the asymptotic standard error of each line are given with the same color. These figures show that the variance of FG-W scales linearly with the number of neurons and inputs while the variance of FG-A only scales linearly with the number of neurons. Note that every axis has a logarithmic scale. Therefore, the slopes of straight lines relate to their exponents on linear scales.

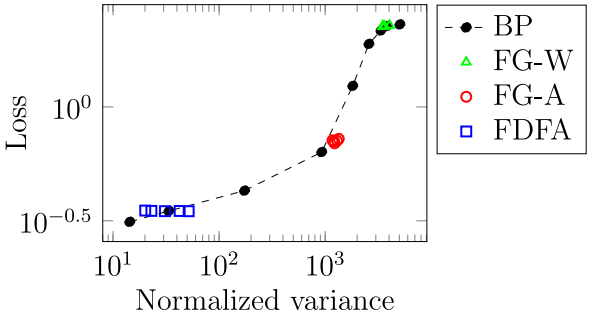


Fig. 7. Correlation between the normalized variance of gradient estimates and the loss of a two-layer network with 1000 hidden neurons, following a single training epoch on the MNIST dataset. The variance of BP was artificially increased by adding Gaussian noise to the gradients to simulate the stochasticity of forward gradients. All gradient variances were normalized with the expected squared norm of the gradient estimates to ensure invariance with regard to the norm. Pairs of variance-loss for the FG-W, FG-A, and FDFA algorithms are represented in green, red, and blue, respectively. This figure shows that the differences in convergence are solely due to the variance of the gradient estimates.

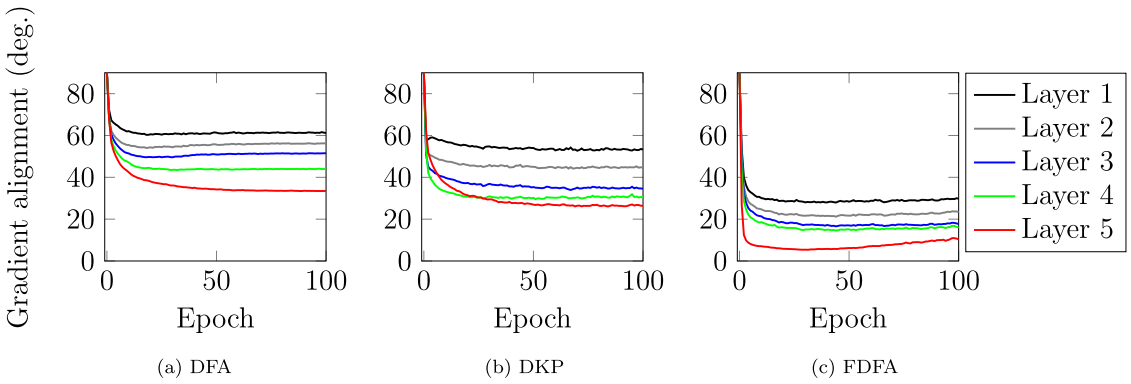
pass of BP, rather than direct connections between output and hidden layers. Although this modification would lower biasedness and better approximate BP without requiring weight transport, it would come at the cost of increased computation time due to backward locking. This highlights a particular tradeoff between computation cost and biasedness in FDFA. Future work could explore novel ways to leverage this biasedness without hindering the parallel aspect of FDFA.

It is also important to note that, while both DFA and our proposed FDFA algorithm compute pseudo-gradients in a similar way, they differ fundamentally in their respective definitions of the feedback connections. In DFA, feedback connections are initialized randomly and remain fixed throughout training, thereby introducing a substantial bias into its pseudo-gradient. In contrast, our FDFA algorithms learns to represent the derivatives between output and hidden neurons as feedback connections. This distinction results in gradient estimates that align more closely with the steepest descent direction compared to DFA. This difference in biasedness was empirically verified in Fig. 8 by measuring the angle between the gradient estimate of each algorithm and the true gradient computed with BP. As shown in Table 4, DFA and FDFA have the same variance in the limit of small feedback learning rates (i.e. 0). Hence, their divergence lies solely in their respective biasness. We observed in our experiments that this difference in biasness consistently influences both the rate of convergence and the performance of DNNs on various benchmark datasets. Therefore, by providing an update direction closer to the direction of steepest descent, our proposed FDFA algorithm demonstrates the ability to converge faster and achieve greater performance than DFA.

The method proposed in this work shares similarities with the DKP algorithm in the way feedback connections are adapted through learning, in contrast to DFA, where feedback matrices are kept fixed. However, the FDFA and DKP algorithms fundamentally differ in their feedback update rules. The convergence of Kolen–Pollack learning primarily relies on the principle of weight decay, which allows the forward and feedback weights to align (Akrout et al., 2019; Webster et al., 2021). In contrast, the proposed FDFA algorithm directly estimates the derivatives between output and hidden neurons using FG-A, without relying on any weight decay. While both algorithms aim to learn the feedback connections, our experiments demonstrated that the proposed FDFA algorithm benefits from greater weight and gradient alignment than DKP. This improved alignment, in turn, appears to lead to better convergence rates and greater performance on benchmark datasets.

Another important aspect to consider is the computational cost of our method. Due to the computation of directional derivatives, the application of our feedback learning rule, and the computation of errors through feedback connections, the proposed FDFA has ultimately a greater computational cost than DFA, FG-A, and DKP. However, our experiments demonstrated that, despite this higher computation cost, the improved convergence rate of FDFA allows for fast convergence relative to the execution time of the algorithm, even with sequential computations on CPUs. Furthermore, when operations are parallelized on GPUs, the computational cost of FDFA has minimal impact on convergence relative to execution time, making it a suitable choice for fast convergence of DNN without resorting to BP.

Finally, while methods relying on feedback connections showed better performance than forward gradients methods, the use of feedback matrices increases the number of weights to store in memory and ultimately poses a significant memory limitation, especially as networks grow larger. This highlights another important tradeoff between memory usage and performance. Future work could explore mechanisms such as sparse feedback matrices (Crafton et al., 2019) or reduced weight precision (Han & jun Yoo, 2019) to mitigate the memory impact of the proposed FDFA algorithm, further enhancing its practicality and scalability. Moreover, in the context of DNNs, local gradients of neurons are locally computed by differentiating the activations of neurons with respect to their weights. Similarly to DFA, the computation local change of weights is not restricted to the exact differentiation of neurons but can also be adapted to other local learning mechanisms. For example, the spikes fired by biologically plausible spiking neurons are known to be non-differentiable, making the computation of local gradients more challenging in Spiking Neural Networks. However, the FDFA could be adapted to SNNs by combining our feedback learning rule with locally-computed surrogate gradients or biologically plausible learning rules such as Spike Timing-Dependant Plasticity (STDP). Future work could thus explore the applicability of the FDFA algorithm to other local learning rules compatible with neuromorphic hardware.



(a) DFA (b) DKP (c) FDFA

Fig. 8. Layerwise alignment between gradient estimates and the true gradient computed using BP. These figures show that the proposed FDFA method (Fig. 8(c)) produces gradient estimates that better align with the true gradients than DFA (Fig. 8(a)) which suggests improved descending directions.

6. Conclusion

The FDFA algorithm represents a promising local alternative to error backpropagation, effectively resolving backward locking and the weight transport problem. Its ability to approximate backpropagation with low variance not only opens new possibilities for the creation of efficient and scalable training algorithms but also holds significant importance in the domain of neuromorphic computing. By solely propagating information in a forward manner, the FDFA algorithm aligns with the online constraints of neuromorphic systems, presenting new prospects for developing algorithms specifically tailored to meet the requirements of these hardware. Therefore, the implications of our findings highlight the potential of FDFA as a promising direction of research for online learning on neuromorphic systems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data used in this research is publicly available.

Acknowledgments

Dominique Chu acknowledges funding from EPSRC grant EP/T008296/1.

Appendix A. Experimental settings

Each architecture uses the ReLU activation function in the hidden layers and linear activations in the output layer. For LG-FG-A, additional local linear outputs with their loss function were added after each layer $l < L-1$ to perform local greedy learning (Ren et al., 2023) In BP, FG-W, FG-A, LG-FG-A, DKP, and FDFA, forward weights were initialized with the uniform Kaiming initialization (He et al., 2015). For FDFA, feedback connections were initialized to 0. For Random DFA, forward weights were initialized to 0 and feedback weights were drawn from a uniform Kaiming distribution and kept fixed during training. We removed the dropout layer in AlexNet as we found that it negatively affects feedback learning. Finally, we also added batch normalization (Ioffe & Szegedy, 2015) after each convolutional layer of AlexNet to help training.

We train each network over 100 epochs with Adam (Kingma & Ba, 2017) and softmax cross-entropy loss functions. However, note that any differentiable loss functions can be used instead of the softmax cross entropy, such as the MSE. We also used Adam for the updates of feedback matrices in the FDFA and DKP algorithms. Adam was used because it is invariant to rescaling of the gradient (Kingma & Ba, 2017), making it a good optimization method to benchmark convergence with gradient estimators that exhibit different scales. We used a learning rate of $\lambda = \alpha = 10^{-4}$ and the default values of the parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ in Adam. Finally, no regularization or data augmentation has been applied. Finally, we used learning rate decay with a decay rate of 0.95 after every epoch.

In this work, all hyperparameters, including α and λ were manually tuned. We found that, in many cases, better convergence behaviors were achieved if $\alpha \leq \lambda$. This motivated our choices for the learning rates of FDFA. However, hyperparameter optimization and learning rate scheduling could be used to further improve the convergence and performance of the method.

Appendix B. Performance of fully-connected networks with different depths

See Tables B.5 and B.6.

Table B.5
Number of neurons and number of parameters in fully-connected DNNs with different depths.

	Depth	MNIST	Fashion MNIST	CIFAR10
N. Neurons	2 Layers	810	810	1010
	3 Layers	1610	1610	2010
	4 Layers	2410	2410	3010
N. Params.	2 Layers	636 K	636 K	3.08 M
	3 Layers	1.2 M	1.2 M	4.08 M
	4 Layers	1.9 M	1.9 M	5.08 M

Table B.6
Performance of fully-connected DNNs with different depths. LG-FG-A was not evaluated with two-layers DNNs as these networks are not deep enough to require greedy learning.

Depth	Method	Local	MNIST	Fashion MNIST	CIFAR10
2 Layers	BP	✗	98.25 ± 0.04%	89.37 ± 0.09%	56.20 ± 0.12%
	FG-W		85.87 ± 0.25%	77.72 ± 0.18%	30.47 ± 0.32%
	FG-A		93.39 ± 0.06%	84.73 ± 0.14%	47.39 ± 0.23%
	DFA	✓	98.04 ± 0.07%	88.67 ± 0.12%	54.80 ± 0.22%
	DKP		98.12 ± 0.05%	89.09 ± 0.11%	55.41 ± 0.18%
	FDFA		**98.21 ± 0.02%**	**89.27 ± 0.07%**	**55.74 ± 0.14%**
3 Layers	BP	✗	98.39 ± 0.06%	90.01 ± 0.08%	56.66 ± 0.18%
	FG-W		82.36 ± 0.48%	73.94 ± 0.57%	27.52 ± 0.34%
	FG-A		90.86 ± 0.17%	82.21 ± 0.13%	43.39 ± 0.14%
	LG-FG-A	✓	91.64 ± 0.07%	82.41 ± 0.13%	43.08 ± 0.18%
	DFA		98.18 ± 0.06%	89.19 ± 0.13%	54.20 ± 0.11%
	DKP		98.28 ± 0.06%	89.63 ± 0.12%	55.15 ± 0.24%
	FDFA		**98.30 ± 0.04%**	**89.82 ± 0.13%**	**55.70 ± 0.15%**

(*continued on next page*)

Table B.6 (*continued*).

Depth	Method	Local	MNIST	Fashion MNIST	CIFAR10
4 Layers	BP	✗	98.46 ± 0.05%	89.99 ± 0.15%	56.92 ± 0.19%
	FG-W	✓	80.03 ± 0.62%	71.72 ± 0.63%	25.17 ± 0.32%
	FG-A		83.40 ± 1.33%	77.62 ± 0.25%	35.53 ± 0.21%
	LG-FG-A		86.89 ± 0.90%	77.98 ± 0.21%	34.56 ± 0.34%
	DFA		98.15 ± 0.05%	89.15 ± 0.09%	53.61 ± 0.16%
	DKP		98.27 ± 0.05%	89.48 ± 0.09%	54.35 ± 0.21%
	FDFA		**98.32 ± 0.05%**	**89.56 ± 0.11%**	**54.97 ± 0.19%**

Appendix C. Weight-Perturbed Forward Gradient Algorithm

Algorithm 2 Weight-Perturbed Forward Gradient algorithm (Baydin et al., 2022) with a fully-connected DNN.

Input: Training data $\mathcal{D}$
Randomly initialize $w_{ij}^{(l)}$ for all l, i and j.
repeat
  {Inference (sequential)}
  for $\boldsymbol{x}$ **in** $\mathcal{D}$ **do**
    $\boldsymbol{y}^{(0)} \leftarrow \boldsymbol{x}_s$
    $\boldsymbol{d}^{(0)} \leftarrow \boldsymbol{0}$
    for $l = 1$ **to** L **do**
      Sample $\boldsymbol{V}^{(l)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
      $\boldsymbol{a}^{(l)} \leftarrow \boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)}$
      $\boldsymbol{y}^{(l)} \leftarrow f\left(\boldsymbol{a}^{(l)}\right)$
      $\boldsymbol{d}^{(l)} \leftarrow \left(\boldsymbol{W}^{(l)}\boldsymbol{d}^{(l-1)} + \boldsymbol{V}^{(l)}\boldsymbol{y}^{(l-1)}\right) \odot \sigma'\left(\boldsymbol{a}^{(l)}\right)$
    end for
    $d \leftarrow \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \boldsymbol{y}^{(L)}} \boldsymbol{d}^{(L)}$
    {Weights updates (parallel)}
    for $l = 1$ **to** L **do**
      $\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \lambda \boldsymbol{V}^{(l)} d$
    end for
  end for
until $\mathbb{E}[\mathcal{L}(\boldsymbol{x})] < \epsilon$

Appendix D. Activity-Perturbed Forward Gradient Algorithm

Algorithm 3 Activity-Perturbed Forward Gradient algorithm (Ren et al., 2023) with a fully-connected DNN.

Input: Training data $\mathcal{D}$
Randomly initialize $w_{ij}^{(l)}$ for all l, i and j.
repeat
  {Inference (sequential)}
  for all $\boldsymbol{x}$ **in** $\mathcal{D}$ **do**
    $\boldsymbol{y}^{(0)} \leftarrow \boldsymbol{x}_s$
    $\boldsymbol{d}^{(0)} \leftarrow \boldsymbol{0}$
    for $l = 1$ **to** L **do**
      Sample $\boldsymbol{v}^{(l)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
      $\boldsymbol{a}^{(l)} \leftarrow \boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)}$
      $\boldsymbol{y}^{(l)} \leftarrow \sigma\left(\boldsymbol{a}^{(l)}\right)$
      $\boldsymbol{d}^{(l)} \leftarrow \left(\boldsymbol{W}^{(l)}\boldsymbol{d}^{(l-1)}\right) \odot \sigma'\left(\boldsymbol{a}^{(l)}\right)$
      if $l < L$ **then**
        $\boldsymbol{d}^{(l)} \leftarrow \boldsymbol{d}^{(l)} + \boldsymbol{v}^{(l)}$
      end if
    end for
    $d \leftarrow \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \boldsymbol{y}^{(L)}} \boldsymbol{d}^{(L)}$
    {Weights updates (parallel)}
    for $l = 1$ **to** L **do**
      $\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \lambda \left(d^{(L)}\boldsymbol{v}^{(l)} \odot \sigma'\left(\boldsymbol{a}^{(l)}\right)\right) \otimes \boldsymbol{y}^{(l-1)}$
    end for
  end for
until $\mathbb{E}[\mathcal{L}(\boldsymbol{x})] < \epsilon$

Appendix E. Unbiasedness of forward gradients

Theorem 3 (*Unbiasedness of Forward Gradients (Baydin et al., 2022)*). *Let $\boldsymbol{x} \in \mathbb{R}^n$ be a vector of size n and $\boldsymbol{v} \in \mathbb{R}^n$ be a random vector of n independent variables. If $\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ follows a multivariate standard normal distribution, then:*

$$\mathbb{E}[(\boldsymbol{x} \cdot \boldsymbol{v})\,\boldsymbol{x}] = \boldsymbol{x} \tag{E.1}$$

Proof. Focusing on the ith element, we have:

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{x} \cdot \boldsymbol{v})\, v_i\right] &= \mathbb{E}\left[\sum_{j=1}^{n} x_j v_j v_i\right] \\ &= \mathbb{E}\left[x_i v_i^2\right] + \sum_{\substack{j=1 \\ j\neq i}}^{n} \mathbb{E}\left[x_j v_j v_i\right] \\ &= x_i \mathbb{E}\left[v_i^2\right] + \sum_{\substack{j=1 \\ j\neq i}}^{n} x_j \mathbb{E}\left[v_j\right] \mathbb{E}\left[v_i\right]\end{aligned} \tag{E.2}$$

However, we know that $v_i \sim \mathcal{N}(0, 1)$. Therefore, $\mathbb{E}\left[v_i\right] = 0$ and $\mathbb{E}\left[v_i^2\right] = \mathbb{E}\left[v_i\right]^2 + \mathrm{Var}\left[v_i\right] = 1$. Using these properties, Eq. (E.2) reduces to:

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{v}}\left[(\boldsymbol{x} \cdot \boldsymbol{v})\, v_i\right] = x_i \tag{E.3}$$

and

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{v}}\left[(\boldsymbol{x} \cdot \boldsymbol{v})\,\boldsymbol{v}\right] = \boldsymbol{x} \tag{E.4}$$

which concludes the proof.

Appendix F. Variance of forward gradients

Lemma 4. *Let $\boldsymbol{x} \in \mathbb{R}^n$ be vector of size n and $\boldsymbol{v} \in \mathbb{R}^n$ be a random vector of n independent variables. If $\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ follows a multivariate standard normal distribution, then:*

$$\mathrm{Var}\left[(\boldsymbol{x} \cdot \boldsymbol{v})\, v_i\right] = x_i^2 + \|\boldsymbol{x}\|_2^2 \tag{F.1}$$

Proof. Because the elements of $\boldsymbol{x}$ are considered as constants and all the elements of $\boldsymbol{v}$ are independent from each other, the variance of $(\boldsymbol{x} \cdot \boldsymbol{v})\, v_i$ decomposes as follows:

$$\begin{aligned}\mathrm{Var}\left[(\boldsymbol{x} \cdot \boldsymbol{v})\, v_i\right] &= \mathrm{Var}\left[\sum_{j=1}^{n} x_j v_j v_i\right] \\ &= \sum_{j=1}^{n} x_j^2\, \mathrm{Var}\left[v_j v_i\right]\end{aligned} \tag{F.2}$$

We can show that, if $j \neq i$:

$$\begin{aligned}\mathrm{Var}\left[v_j v_i\right] &= \mathbb{E}\left[\left(v_j v_i\right)^2\right] - \mathbb{E}\left[v_j v_i\right]^2 \\ &= \mathbb{E}\left[v_i^2\right] \mathbb{E}\left[v_j^2\right] - \mathbb{E}\left[v_i\right]^2 \mathbb{E}\left[v_j\right]^2 \\ &= 1\end{aligned} \tag{F.3}$$

and if $j = i$:

$$\begin{aligned}\mathrm{Var}\left[v_i^2\right] &= \mathbb{E}\left[\left(v_i^2\right)^2\right] - \mathbb{E}\left[v_i^2\right]^2 \\ &= \mathbb{E}\left[v_i^4\right] - \mathbb{E}\left[v_i^2\right]^2 \\ &= 2\end{aligned} \tag{F.4}$$

as $\mathbb{E}\left[v_i^2\right]^2 = \mathrm{Var}\left[v_i\right]^2 = 1$ and $\mathbb{E}\left[v_i^4\right] = 3\mathrm{Var}\left[v_i\right] = 3$.

Therefore, by plugging Eqs. (F.3) and (F.4) into Eq. (F.2), we find:

$$\begin{aligned}\mathrm{Var}\left[(\boldsymbol{x}\cdot\boldsymbol{v})\,v_i\right] &= \sum_{j=1}^{n} x_j^2\,\mathrm{Var}\left[v_j v_i\right]\\ &= x_i^2\mathrm{Var}\left[v_i^2\right] + \sum_{\substack{j=1\\ j\neq i}}^{n} x_j^2\mathrm{Var}\left[v_j v_i\right]\\ &= 2x_i^2 + \sum_{\substack{j=1\\ j\neq i}}^{n} x_j^2\\ &= x_i^2 + \sum_{j=1}^{n} x_j^2\\ &= x_i^2 + \|\boldsymbol{x}\|_2^2\end{aligned} \tag{F.5}$$

which concludes the proof.

Appendix G. Variance of Weight-Perturbed Forward Gradients

Proposition 5 (*Variance of Weight-Perturbed Forward Gradients*). *Let $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n_1,n_0}$ be the hidden weights of a two-layers fully-connected neural network evaluated with an input sample $\boldsymbol{x} \in \mathbb{R}^{n_0}$. We denote by $g^{\mathrm{FG-W}}\left(w_{i,j}^{(1)}\right)$ the weight-perturbed forward gradient for the weight $w_{i,j}^{(l)}$. We also assume that all derivatives $\left(\partial\mathcal{L}(\boldsymbol{x})/\partial w_{i,j}^{(l)}\right)^2 \leq \beta$ are bounded and that all the elements of the perturbation matrix $\boldsymbol{V}^{(2)}$ for the weights of the output layer are 0. If each element $v_{i,j}^{(1)} \sim \mathcal{N}(0,1)$ of $\boldsymbol{V}^{(1)}$ follows a standard normal distribution, then:*

$$\begin{aligned}\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-W}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] &= \left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}}\right)^2 + \left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{W}^{(1)}}\right\|_2^2\\ &= O\left(n_1 n_0\right)\end{aligned} \tag{G.1}$$

in the limit of large n_1 and large n_0.

Proof. By application of Lemma 4, we know that:

$$\begin{aligned}\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-W}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] &= \underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[\sum_{k=1}^{n_1}\sum_{l=1}^{n_0}\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{k,l}^{(1)}}v_{k,l}\,v_{i,j}\right]\\ &= \left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}}\right)^2 + \left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{W}^{(1)}}\right\|_2^2\end{aligned} \tag{G.2}$$

However, in the limit of large n_0 and large n_1:

$$\left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{W}^{(1)}}\right\|_2^2 = \sum_{i=1}^{n_1}\sum_{j=1}^{n_0}\left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}}\right)^2 = O\left(n_1 n_0\right) \tag{G.3}$$

Therefore, we conclude that:

$$\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-W}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] = O\left(n_1 n_0\right) \tag{G.4}$$

Appendix H. Variance of Activity-Perturbed Forward Gradients

Proposition 6 (*Variance of Activity-Perturbed Forward Gradients*). *Let $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n_1,n_0}$ be the hidden weights of a two-layers fully-connected neural network evaluated with an input sample $\boldsymbol{x} \in \mathbb{R}^{n_0}$. We denote by $g^{\mathrm{FG-A}}\left(w_{i,j}^{(1)}\right)$ the activity-perturbed forward gradient for the weight $w_{i,j}^{(l)}$. We also assume that all derivatives $\left(\partial\mathcal{L}(\boldsymbol{x})/\partial w_{i,j}^{(l)}\right)^2 \leq \beta$ are bounded and that all the elements of the perturbation vector $\boldsymbol{u}^{(2)}$ for the activations of the output layer are 0. If each element $u_i^{(1)} \sim \mathcal{N}(0,1)$ of $\boldsymbol{u}^{(1)}$ follows a standard normal distribution, then:*

$$\begin{aligned}\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-A}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] &= \left[\left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_i^{(1)}}\right)^2 + \left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{y}^{(1)}}\right\|_2^2\right]\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2\\ &= O\left(n_1\right)\end{aligned} \tag{H.1}$$

in the limit of large n_1 and large n_0.

Proof. Starting from the variance of $g^{\mathrm{FG-A}}\left(\boldsymbol{W}^{(1)}\right)$, we have:

$$\begin{aligned}\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-A}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] &= \underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[\left(\sum_{k=1}^{n_1}\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_k^{(1)}}u_k^{(1)}\right)u_i^{(1)}\,\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right]\\ &= \underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[\sum_{k=1}^{n_1}\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_k^{(1)}}u_k^{(1)}u_i^{(1)}\mid\boldsymbol{x}\right]\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2\end{aligned} \tag{H.2}$$

as the partial derivative $\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}$ is considered as constant.

By applying Lemma 4, we obtain:

$$\begin{aligned}\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-A}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] &= \left[\left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_i^{(1)}}\right)^2 + \left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{y}^{(1)}}\right\|_2^2\right]\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2\\ &= \left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial w_{i,j}^{(1)}}\right)^2 + \left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{y}^{(1)}}\right\|_2^2\left(\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}}\right)^2\end{aligned} \tag{H.3}$$

Finally, we can show that

$$\left\|\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial\boldsymbol{y}^{(1)}}\right\|_2^2 = \sum_{i=1}^{n_1}\left(\frac{\partial\mathcal{L}(\boldsymbol{x})}{\partial y_i^{(1)}}\right)^2 = O\left(n_1\right) \tag{H.4}$$

Therefore:

$$\underset{\boldsymbol{v}^{(1)}}{\mathrm{Var}}\left[g^{\mathrm{FG-A}}\left(w_{i,j}^{(1)}\right)\mid\boldsymbol{x}\right] = O\left(n_1\right) \tag{H.5}$$

References

Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T., & Tweed, D. B. (2019). Deep learning without weight transport. *Advances in Neural Information Processing Systems*. Curran Associates, Inc,

Amato, G., Carrara, F., Falchi, F., Gennaro, C., & Lagani, G. (2019). Hebbian learning meets deep convolutional neural networks. In *Image Analysis and Processing – ICIAP 2019: 20th International Conference, Trento, Italy, September (2019) 9–13, Proceedings, Part I* (pp. 324–334). Berlin, Heidelberg: Springer-Verlag.

Bartunov, S., Santoro, A., Richards, B. A., Marris, L., Hinton, G. E., & Lillicrap, T. P. (2018). 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). Red Hook, NY, USA: Curran Associates Inc.

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., & Torr, P. (2022). Gradients without backpropagation. arXiv:2202.08587.

Belilovsky, E., Eickenberg, M., & Oyallon, E. (2019). Greedy layerwise learning can scale to imagenet. In *Proceedings of the 36th International Conference on Machine Learning, Proceedings of Machine Learning Research* (pp. 583–593).

Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, *60*, 223–311. http://dx.doi.org/10.1137/16M1080173.

Bubeck, S. (2015). Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, *8*, 231–357. http://dx.doi.org/10.1561/2200000050.

Chee, J., & Toulis, P. (2018). Convergence diagnostics for stochastic gradient descent with constant learning rate. In A. Storkey, & F. Perez-Cruz (Eds.), *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research* (pp. 1476–1485).

Crafton, B., Parihar, A., Gebhardt, E., & Raychowdhury, A. (2019). Direct feedback alignment with sparse connections for local learning. *Frontiers in Neuroscience*, *13*, http://dx.doi.org/10.3389/fnins.2019.00525.

Defazio, A. (2020). Momentum via primal averaging: Theoretical insights and learning rate schedules for non-convex optimization. arXiv:2010.00406.

Faghri, F., Duvenaud, D., Fleet, D. J., & Ba, J. (2020). A study of gradient variance in deep learning. arXiv:2007.04532.

Frenkel, C., Lefebvre, M., & Bol, D. (2021). Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in Neuroscience*, *15*, http://dx.doi.org/10.3389/fnins.2021.629892.

Gower, R. M., Loizou, N., Qian, X., Sailanbayev, A., Shulgin, E., & Richtarik, P. (2019). SGD: General analysis and improved rates. arXiv:1901.09401.

Han, D., Lee, J., Lee, J., & Yoo, H. J. (2019). A 1.32 TOPS/w energy efficient deep neural network learning processor with direct feedback alignment based heterogeneous core architecture. In *2019 Symposium on VLSI Circuits*. IEEE, http://dx.doi.org/10.23919/vlsic.2019.8778006.

Han, D., & Yoo, H. j. (2019). Direct feedback alignment based convolutional neural network training for low-power online learning processor. In *2019 IEEE/CVF International Conference on Computer Vision Workshop* ICCVW, (pp. 2445–2452). http://dx.doi.org/10.1109/iccvw.2019.00299.

Han, D., & jun Yoo, H. (2019). Efficient convolutional neural network training with direct feedback alignment. arXiv:1901.01986.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision* ICCV, (pp. 1026–1034). Los Alamitos, CA, USA: IEEE Computer Society, http://dx.doi.org/10.1109/ICCV.2015.123.

Hinton, G. (2022). The forward-forward algorithm: Some preliminary investigations. arXiv:2212.13345.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., & Bengio, Y. (2019). Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*.

Huo, Z., Gu, B., & Huang, H. (2018). Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning, Proceedings of Machine Learning Research* (pp. 2098–2106).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning - Volume 37, Journal of Machine Learning Research* (pp. 448–456).

Jabri, M., & Flower, B. (1992). Weight perturbation: an optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks. *IEEE Transactions on Neural Networks*, *3*, 154–157. http://dx.doi.org/10.1109/72.105429.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., & Kavukcuoglu, K. (2017). Decoupled neural interfaces using synthetic gradients. In D. Precup, & Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research* (pp. 1627–1635).

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization. arXiv:1412.6980.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Vol. 1* (pp. 1097–1105). Red Hook, NY, USA: Curran Associates Inc.

Launay, J., Poli, I., Boniface, F., & Krzakala, F. (2020). Direct feedback alignment scales to modern deep learning tasks and architectures. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (pp. 9346–9360). Curran Associates, Inc.

Launay, J., Poli, I., & Krzakala, F. (2019). Principled training of neural networks with direct feedback alignment. arXiv:1906.04554.

Le Cun, Y., Galland, C., & Hinton, G. E. (1988). Gemini: Gradient estimation through matrix inversion after noise injection. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems*. Morgan-Kaufmann.

Lillicrap, T. P., Cownden, D., Tweed, D. B., & Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, *7*(13276), http://dx.doi.org/10.1038/ncomms13276.

Löwe, S., O' Connor, P., & Veeling, B. (2019). Putting an end to end-to-end: Gradient-isolated learning of representations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Margossian, C. C. (2019). A review of automatic differentiation and its efficient implementation. *WIREs Data Mining and Knowledge Discovery*, *9*, http://dx.doi.org/10.1002/widm.1305.

Mostafa, H., Ramesh, V., & Cauwenberghs, G. (2018). Deep supervised learning using local errors.

Moulines, E., & Bach, F. (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, & K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Murata, N. (1999). *A Statistical Study of On-Line Learning* (pp. 63–92). USA: Cambridge University Press.

Needell, D., Srebro, N., & Ward, R. (2016). Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Mathematical Programming*, *155*, 549–573. http://dx.doi.org/10.1007/s10107-015-0864-7.

Neftci, E. O., Augustine, C., Paul, S., & Detorakis, G. (2017). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *11*, http://dx.doi.org/10.3389/fnins.2017.00324,

Nøkland, A. (2016). Direct feedback alignment provides learning in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (pp. 1045–1053). Red Hook, NY, USA: Curran Associates Inc.

Nøkland, A., & Eidnes, L. H. (2019). Training neural networks with local error signals. In K. Chaudhuri, & R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning, Proceedings of Machine Learning Research* (pp. 4839–4850).

Refinetti, M., D'Ascoli, S., Ohana, R., & Goldt, S. (2021). Align, then memorise: the dynamics of learning with feedback alignment. In *International Conference on Machine Learning* (pp. 8925–8935).

Ren, M., Kornblith, S., Liao, R., & Hinton, G. (2023). Scaling forward gradient with local losses. In *The Eleventh International Conference on Learning Representations*.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *22*, 400–407. http://dx.doi.org/10.1214/aoms/1177729586.

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

Silver, D., Goyal, A., Danihelka, I., Hessel, M., & van Hasselt, H. (2021). Learning by directional gradient descent. In *International Conference on Learning Representations*.

Webster, M. B., Choi, J., & Ahn, Changwook (2021). Learning the connections in direct feedback alignment. URL: https://openreview.net/forum?id=zgGmAx9ZcY.

Wen, Y., Vicol, P., Ba, J., Tran, D., & Grosse, R. (2018). Flipout: Efficient pseudo-independent weight perturbations on mini-batches. arXiv:1803.04386.

Zhang, T., Jia, S., Cheng, X., & Xu, B. (2022). Tuning convolutional spiking neural network with biologically plausible reward propagation. *IEEE Transactions on Neural Networks and Learning Systems*, *33*, 7621–7631. http://dx.doi.org/10.1109/TNNLS.2021.3085966.