MACHINE LEARNING
Science and Technology

PAPER

# Random feedback alignment algorithms to train neural networks: why do they align?

**Dominique Chu*** and **Florian Bacho**

School of Computing, CEMS, University of Kent, CT2 7NF Canterbury, United Kingdom
* Author to whom any correspondence should be addressed.

**E-mail:** D.f.chu@kent.ac.uk and F.Bacho@kent.ac.uk

**Keywords:** neural networks, feedback alignment, random walk

OPEN ACCESS

RECEIVED
25 October 2023

REVISED
23 February 2024

ACCEPTED FOR PUBLICATION
15 April 2024

PUBLISHED
2 May 2024



## Abstract

Feedback alignment algorithms are an alternative to backpropagation to train neural networks, whereby some of the partial derivatives that are required to compute the gradient are replaced by random terms. This essentially transforms the update rule into a random walk in weight space. Surprisingly, learning still works with those algorithms, including training of deep neural networks. The performance of FA is generally attributed to an alignment of the update of the random walker with the true gradient—the eponymous gradient alignment—which drives an approximate gradient descent. The mechanism that leads to this alignment remains unclear, however. In this paper, we use mathematical reasoning and simulations to investigate gradient alignment. We observe that the feedback alignment update rule has fixed points, which correspond to extrema of the loss function. We show that gradient alignment is a stability criterion for those fixed points. It is only a necessary criterion for algorithm performance. Experimentally, we demonstrate that high levels of gradient alignment can lead to poor algorithm performance and that the alignment is not always driving the gradient descent.

## 1. Introduction

The backpropagation algorithm (BP) [1] underpins a good part of modern neural network (NN) based AI. BP-based training algorithms continue to be the state of the art in many areas of machine learning ranging from benchmark problems such as the MNIST dataset [2] to the most recent transformer-based architectures [3]. While its success is undeniable, BP has some disadvantages. The main one is that BP is computationally expensive and relies on sequential processing of layers during both the forward and backward pass, limiting its scope for parallelisation. This is sometimes called *backward locking* [4].

Secondly, BP is biologically implausible. One aspect of this is the need for feedforward and feedback weights to be symmetric. From a purely machine learning point of view, the lack of biological plausibility may not be too concerning, since the aim of applied AI is more often performance, rather than neuroscientific realism. However, there is a sense in which biological plausibility becomes a real concern after all: The update of any particular connection weight in a neural network requires global information about the entire network. This entails intense data processing needs [5], which in turn leads to high energy consumption [5, 6], which is practically and environmentally undesirable [7, 8]. BP is also not compatible with neuromorphic hardware platforms [9], such as Loihi [10] or SpiNNaker [11].

In the light of this, there has been some recent interest in alternatives to BP that alleviate these issues [12]. One particularly intriguing example are *random feedback alignment* (FA) algorithms [13]. The basic FA algorithm is just BP with the symmetric feedback weights replaced by a randomly chosen, but fixed, feedback matrix, thus addressing some concerns regarding the biological plausibility of BP. Note the error computation in the output layer does not depend on feedback weights, and is identical in for both FA and BP. In practice particularly relevant is a variant of FA, called *direct feedback alignment* (DFA) [14]. In DFA the errors are transmitted directly to each layer via random feedback matrices. This enables layer-wise parallel

updates of weights. Furthermore, training no longer requires global knowledge of the entire network, which makes DFA amenable to implementation on neuromorphic hardware [15].

FA algorithms replace partial derivatives in the gradient computation by random or partially random matrices. Mathematically, the resulting update will no longer be a gradient of the loss function, but must be expected to be orthogonal to the gradient. One would therefore assume that DFA and FA will not work. Surprisingly they have been found to perform well on a number of benchmark problems [16, 17]. Recently, it has been reported that they even work on large-scale architectures such as transformers [18], often reaching performances that are comparable, albeit not exceeding, those of BP-based algorithms.

While there is to date no detailed model explaining the performance of FA, the general idea was formulated in the original paper by Lillicrap *et al* [13]. According to them the update direction of FA is not orthogonal to the gradient after all, but there is *weight alignment*, i.e. the weights of the network align with (i.e. point into approximately the same direction as) the feedback matrices and *gradient alignment* where the updates of the FA algorithm align with the gradient as computed by BP. They conjectured that this alignment drives the performance of FA. A mechanism that could lead to this alignment was suggested by Refinetti *et al* [19]. They modelled a linear two-layer network using a student-teacher setup based on an approach by Saad and Solla [20]. This showed that, at least in their setup, when starting from initially zero weights, the weight update is in the direction of the feedback matrix, leading to weight alignment and consequently gradient alignment. A corollary of their results is the prediction that alignment is particularly strong when the weights are initially vanishing. Song *et al* [21] prove convergence results for FA and show that over-parametrisation is essential for FA to work. Another important theoretical contribution is by Nøkland and Eidnes [14] who formulated a stability criterion for DFA.

Common to all these theoretical approaches is that they assume that weight/gradient alignment drive the performance of FA. As our first contribution, in section 2.1 we will subject this assumption to experimental scrutiny. Using the MNIST dataset as an example, we will show that at least for this example alignment is not sufficient to explain the performance of FA. Indeed, we will construct a particular example where better alignment leads to worse performance. This single counter-example, will entitle us to reject in general the idea that weight alignment is the key-driver of the performance of FA.

Subsequently, in section 2.2 we will present theoretical arguments to support a different model of how FA works. Acknowledging that FA prescribes an essentially random update direction, there can be no assumption that it will minimise the loss function. In that sense, we will refer to FA as a *random walk* in weight space. As our second contribution, we will show that there are particular points in weight space, that is specific choices of weights, where the jump length vanishes. In a slight abuse of notation, we will refer to those as *fixed points* of the random walk. As will become clear, these correspond to local extrema of the loss function, and as such correspond to valid solutions of the BP algorithm. If the random walker landed exactly on one of those, then it would remain there. However, typically these fixed points are not stable under the FA update rule, that is they are not attractors of the random walker. In this case, a walker initialised in the neighbourhood of the fixed point would move away from the fixed point.

As the main contribution of this paper we will show that the stability criterion for the fixed points of FA is precisely the gradient alignment property (note that this is different from the one derived by Nøkland and Eidnes [14]). This then leads to a novel model for the origin and role of gradient alignment and consequently the performance of FA: Updates of the FA algorithm initially perform a random walk thus exploring the weight space. In the course of this exploration, the walker may enter the basin of attraction of a stable fixed point, which will manifest itself mathematically in that FA updates will be gradient aligned. Given that the fixed point is stable, the network will remain in the neighbourhood of this point for an extended amount of time. Moreover, the fixed point is also a local extremum of the loss function.

## 2. Results

The consensus explanation for FA is based on the observation that gradient alignment leads to FA approximating BP. The idea here is that something in the update rule of FA leads to weight updates that are approximately into the same direction as BP, and hence the algorithm is able to find weights that are near the extrema of the loss function. In that sense, gradient alignment is thought to drive the convergence of weights under FA updates.

We will start by experimentally scrutinising the role of weight and gradient alignment in FA using the particular example of the MNIST training set. We conceived simulation experiments designed to isolate specifically the effect of weight alignment which will enable us to show that FA performs much better than can be explained from weight alignment alone. While we limit ourselves to a particular example only, we can still justify the *general* conclusion that weight alignment is not sufficient to explain FA performance, thus

casting doubt onto the current consensus explanation. Following the description of our experiments, we will then suggest an alternative explanation.

### 2.1. Experiments

For our experiments here, we will focus on a feedforward neural network with 2 hidden layers which we train on the MNIST problem. MNIST was chosen because it is convenient, easy to solve and well known, but the precise choice of example is to some extent irrelevant here. The purpose of the experiments we present is to build up some intuition about how FA works. In particular, we are not trying to optimise the performance of FA on this task, nor are we attempting to give novel approaches to solve MNIST, all of which has been done extensively in the prior literature.

#### *2.1.1. Alignment*

Before discussing our experiments, we will need to explain precisely what we mean by 'alignment.' Throughout this article, we define the *alignment measure* of two vectors or two matrices $\mathbf{a}$ and $\mathbf{b}$ as the cosine of the angle between them. In the case of two matrices, this is computed by flattening the matrices in some way, for example by stacking the columns to obtain $\mathbf{a}'$ and $\mathbf{b}'$. The alignment measure is then computed as the inner product of the vectors divided by their norms,

$$\frac{\mathbf{a}' \cdot \mathbf{b}'}{\|\mathbf{a}'\| \|\mathbf{b}'\|}. \qquad \text{(alignment measure)}$$

The maximal value of the alignment is 1. This can be interpreted as $\mathbf{a}'$ and $\mathbf{b}'$ being completely parallel. The minimal value is $-1$, which indicates that they are anti-parallel. In high dimensional spaces, two randomly chosen matrices/vectors will typically have an alignment of 0, indicating that they are orthogonal to one another [22].

#### *2.1.2. Alignment is not sufficient to explain performance of FA*

An important theoretical contribution to our understanding of FA has been the above cited paper by Refinetti *et al* [19]. For the special case of a two layer linear network they showed that when starting from initially vanishing weights, the update of weights is in the direction of the feedback matrices; this entails weight alignment and consequently gradient alignment, which then drives the performance. To understand whether or not the Refinetti model generalises to arbitrary neural network scenarios, we will now test whether its conclusions hold for a 4 layer, neural network with non-linear activation functions (see Methods for details).

A prediction of the Refinetti model is that initially smaller weights lead to initially larger alignment and faster convergence of FA to a good solution. Our simulations are consistent with this. Figure 1 shows that there is high weight (figure 1(c)) and gradient alignment (figure 1(d)) when initial weights are small. Alignment reduces rapidly as the initial weights increase. Interestingly, weight alignment remains rather modest in comparison to the gradient alignment. Within 10 epochs, it does not even reach a value of 0.1. Still, as expected, FA finds good solutions faster when starting with lower weights (figure 1(a)). This conclusion also holds in the long run. Even after 50 epochs, the initial conditions matter for the achieved accuracy. The higher the initial weights, the lower the accuracy (see figure 1(b)).

These results are consistent with the view that rapid feedback alignment during early updates is important for the eventual performance of the algorithm. A closer examination, however, reveals some additional complexities. The first one is highlighted by figure 2, which shows the norm of the gradient for the hidden and output layers after the first update step, so after the algorithm has been presented with the first example of the training set (figures 2(a)–(c)). The main observation to be made from these figures is that the norm reduces rapidly as the initial weights increase. If we take the norm as an indicator for the step size of the random walker, then this suggests that walkers initialised with high weights suffer from slow speed as a result of small update steps. High weights, therefore mean an effectively reduced learning rate at the beginning of training. Note the dramatic decrease of the gradient norm, and thus effective learning rate, as the weights start to differ from 0.

These results suggest a new explanation for the improved performance of networks initialised with low weights: While Refinetti *et al* [19] argued that initially low weights lead to weight alignment and consequently gradient alignment, here we showed that, at least in this example, there is an additional aspect to this: Low weights improve the initial speed of exploration and hence accuracy can increase over fewer update steps. While we observed this in a particular example only, we can still conclude that gradient alignment is not necessarily the only reason why FA performs.

This begs now the question whether or not gradient alignment drives accuracy, or whether gradient alignment is merely a by-product of the FA dynamics. To explore this we concentrate on the earliest stages of learning, before substantial alignment has formed. We first need to understand the relationship between

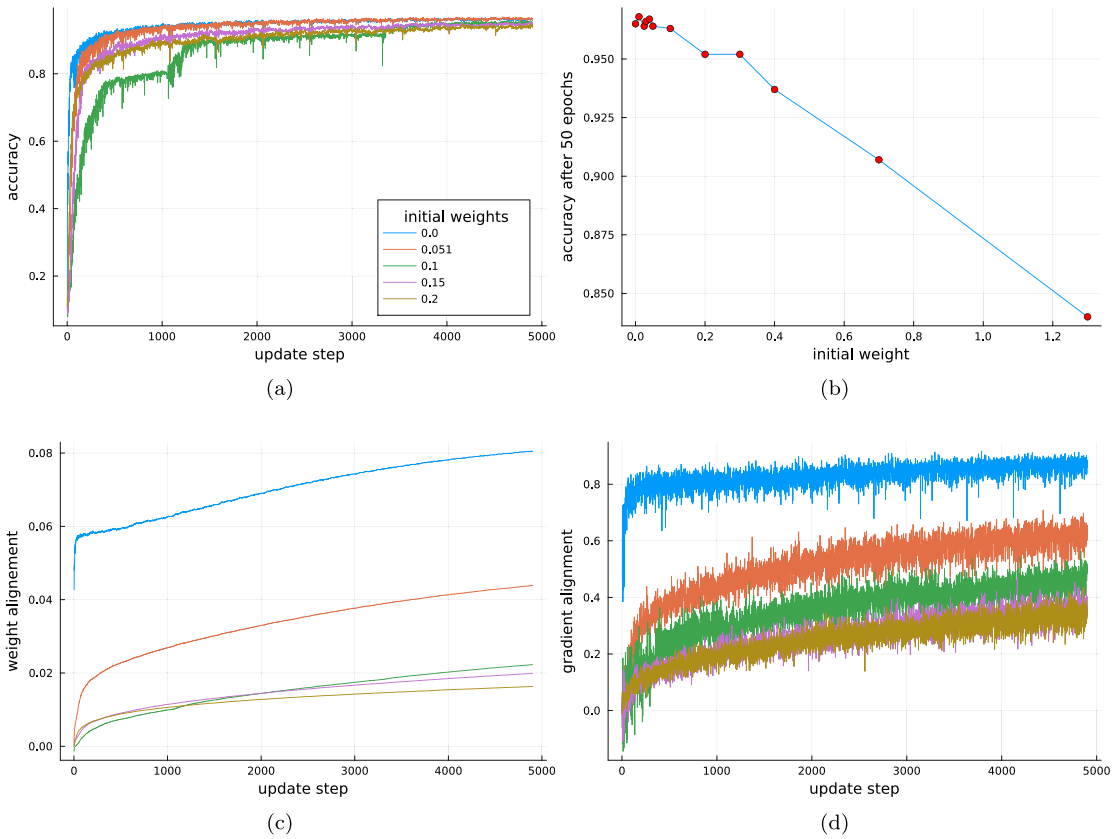



**Figure 1.** The upper panel shows (a) the accuracy as a function of the update step for altogether 10 epochs. Each curve corresponds to different initial weights. Initial weights are drawn from a normal distribution and then multiplied by the scaling factor indicated in the legend. A factor of 0 means that the initial weights are all vanishing. (b) The accuracy after 50 epochs as a function of the initial weights (see Methods for an explanation). An approximate linear dependence is discernible. The lower panel shows the (c) weight alignment and (d) gradient alignment for various initial weights as a function of updates. All results are averaged over 3 independent repetitions and used standard parameters (see Methods).

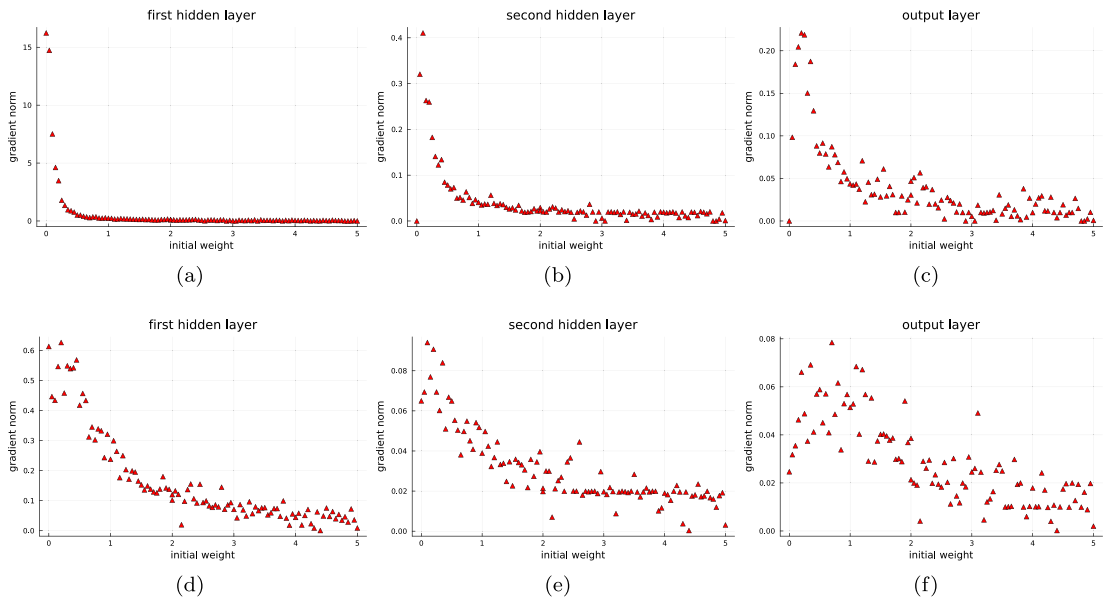



**Figure 2.** (a)–(c) The infinite norm of the gradient at the first update step as a function of the initial weight scaling factor. Initial weights are drawn from a normal distribution and then multiplied by the scaling factor. A factor of 0 means that the initial weights are all vanishing. (d)–(f) Same, but the norm of the gradient after 3 epochs. Note the different scale on the vertical axis, showing how the norm of the gradients has reduced over time. Each point corresponds to a single simulation.

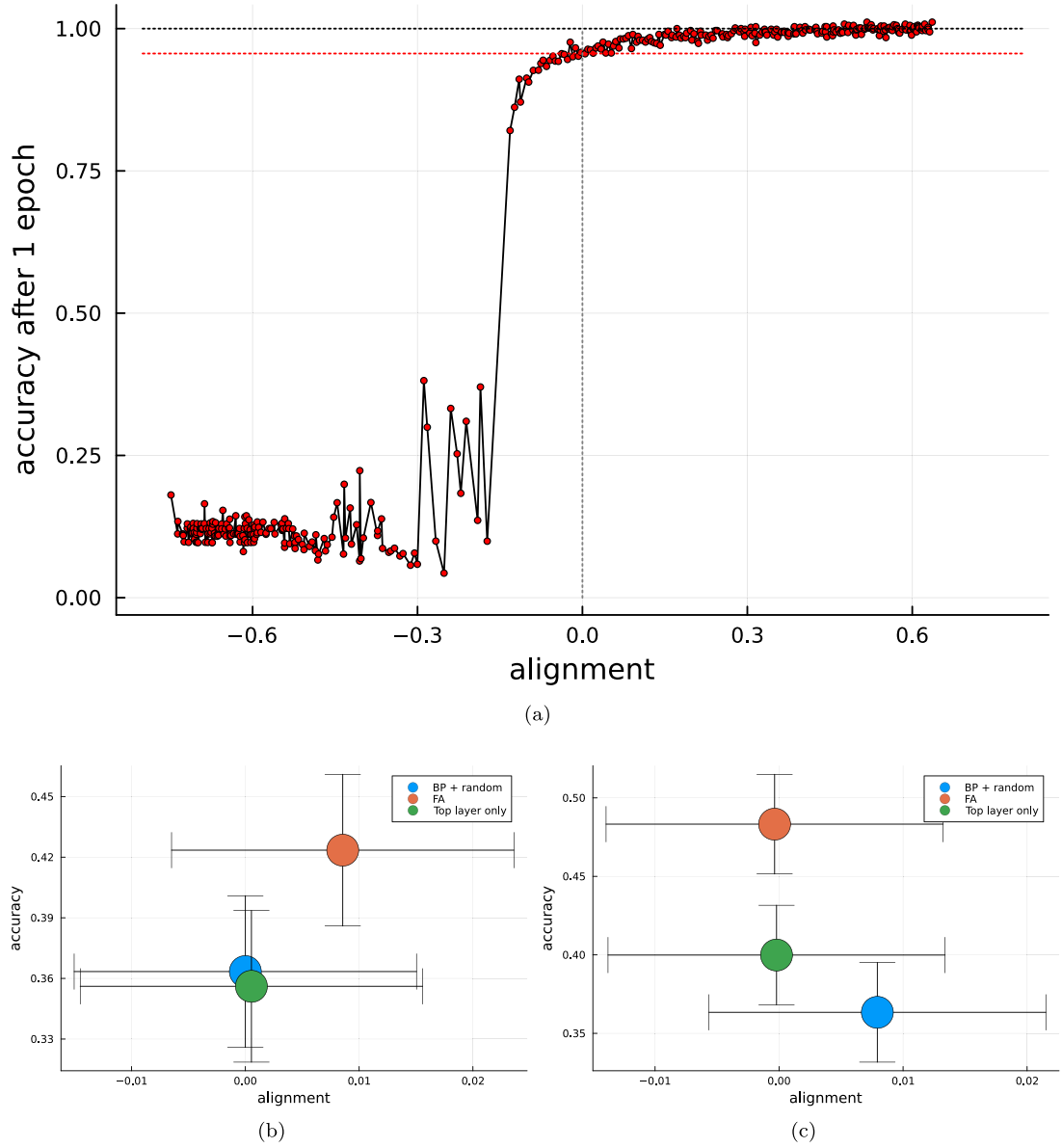
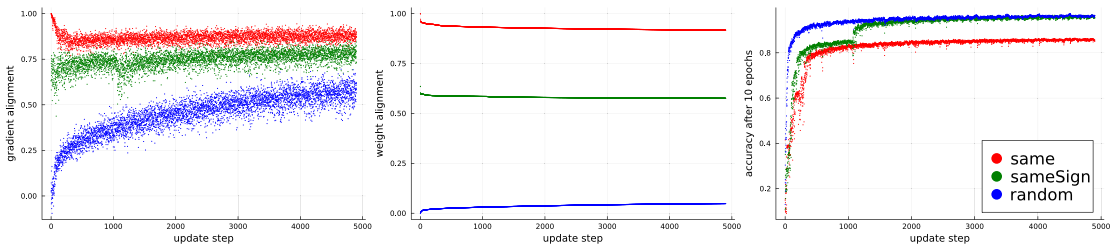



**Figure 3.** (a) We trained a network using BP. For the hidden layers we perturbed the gradient computed via BP. The horizontal axis shows the alignment between the perturbed updates and the exact BP update. The vertical axis shows the accuracy after one epoch relative to BP. A value of 1 indicates that the perturbed network performs equally well as BP. The dotted red line indicates the performance of a network where only the last layer is trained. The accuracy after 5 updates against the mean gradient alignment for the (b) first and (c) second hidden layer. The average is taken over 500 repetitions. The error bars show the standard deviations. For comparison, the perturbed BP is also shown. Clearly, FA does better than the perturbed BP in those examples.

gradient alignment and loss. To this end, we generated a baseline curve as follows: we used the BP algorithm to train the network for one epoch on the MNIST dataset. Each time the gradient was computed in the hidden layers, we randomly perturbed it, such that the actual gradient used for updating the weights was different from the gradient determined by BP. (Note, that we do not perturb the gradient in the output layer, which is computed exactly in both BP and FA.) We then determined the accuracy after one epoch of training as a function of the alignment between the true gradient and the perturbed gradient. We did this systematically for alignments ranging from $-1$ to 1 in figure 3(a). By design, this set of experiments isolates the effect of the gradient alignment, while all other details of the algorithm are left the same. In particular, the figure also shows (in red) the baseline of a multi-layer network where only the last layer is trained, whereas all other layers remain at the initial weights.

A number of observations can be made. Firstly, the accuracy of the network intersects the red line for an alignment of 0. In this case, the perturbed gradients are orthogonal to the actual gradients one would obtain from BP which means that weight updates do not have an overall drift either into the direction of better accuracy or away from it. Consistently, the accuracy of the network then corresponds to simulations where




(a) (b) (c)

**Figure 4.** Feedback alignment, with initial weights chosen at random (blue), randomly but the sign of each element was the same as the feedback matrix (green) and exactly the same as the feedback matrix (red). We show (a) the alignment between the true gradient and the FA pseudo-gradient of the second hidden layer, (a) the alignment between the weights of the second hidden layer and the feedback matrix, (c) the accuracy over time. It is clear that higher alignment does not necessarily translate to better accuracy. Each line shows a single training run over 10 epochs.

only the output layer is trained (indicated by the red line in figure 3(a)). Further increasing the alignment, only improves the performance a little bit, reflecting the well known fact that training only the output layer can be sufficient to achieve high accuracies. On the other hand, for negative alignments, the performance quickly drops to random guessing, which corresponds to an accuracy of 0.1.

The astute reader may at this point be surprised. Updating against the gradient means that the network is systematically driven towards wrong predictions. Instead, we find that the strongest negative alignment only leads to randomisation of the performance, i.e. accuracies 0.1. To understand this, note that in all simulations we use exact, unperturbed gradient descent in the output layer. While the lower layers are driven towards systematically wrong predictions, the output layer pulls the network as a whole away from the loss maxima, altogether resulting in a network that gives randomised predictions.

The data presented in figure 3(a) can now be used as a baseline, which isolates the impact of weight-alignment (or rather misalignment) on the performance of FA. It enables us to understand whether the performance of FA is driven solely by the alignment of the pseudo-gradient or whether there is more to it. If FA performs exactly as well as the perturbed BP with the same alignment, then we know that gradient alignment is the sole factor that drives performance. If, on the other hand, it performs better, then there is some additional driver.

Figure 3(b) shows the performance as a function of the average alignment for FA, gradient perturbed BP and and a network where only the output layer is trained. Figures 3(b) and (c) compare the performance of the three algorithms after 5 updates against the alignment of the hidden layers. This shows that, for similar alignment, FA does much better than the other two algorithms. This suggests two things: (*i*) gradient alignment is not the only driver of performance of FA, at least not during early stages. (*ii*) The performance of the FA during early stages is not entirely driven by the gradient descent in the output layer, but the updates of the hidden layers does add to performance.

### *2.1.3. Alignment can reduce performance*

So far, we have established that alignment between the gradient and pseudo-gradient is not driving performance during early stages of learning. We will now show that alignment is not sufficient for performance, even during later stages, and indeed can be outright detrimental.

Figure 4 shows, as an example, a set of three different simulations of FA with identical hyper-parameters. The only difference between them is the initialisation of the weights. The blue line is just a standard simulation, with weights being initially drawn from a normal distribution before being scaled by 0.05. The green simulation is the same, but the signs of the initial weights were set equal to the entries of the corresponding feedback matrices. Finally, the red points show the results for a simulation where the initial weights were set to be identical to the feedback matrices before being scaled by 0.05. For all three simulations, we drew the elements of the feedback matrices from a normal distribution, rather than from the set $\{-1, 1\}$ (see Methods section). We did this so that the initial weights in the blue simulation are statistically indistinguishable from the other two simulations, while the initial alignment between the weights is different in the three cases. By construction, the blue simulation is weight unaligned initially, the red simulation is perfectly weight aligned and the green simulation is somewhere in between those two cases.

We find from the simulations presented in figure 4, as expected, that good weight alignment translates to high gradient alignment. On the other hand, high gradient alignment does not necessarily translate to high performance of the algorithm. Indeed, the example simulations in figure 4 demonstrate that the initially unaligned simulation performs best, amongst the three runs. While the performance of the red simulation is

still good in the sense that it is apparently learning something, it is possible to construct examples of FA networks that have almost perfect alignment throughout, but learn nothing (data not shown). The simplest way to do this is to use feedback matrices that are initialised by randomly drawing elements from the set $\{-1, 1\}$, and set the initial weights equal to the feedback matrices. These initial conditions are not conducive to algorithm performance, and the network does not train well. Altogether, we find, that gradient alignment is not sufficient to explain the performance of FA algorithms.

While we are not claiming that high gradient alignment is always detrimental, from this single example we can still draw two conclusions: (*i*) Weight alignment is not a necessary condition for performance of the FA algorithm. (*ii*) High gradient alignment is not sufficient for performance of the FA algorithm. A corollary of this is that gradient alignment and even less so weight alignment cannot be used as general explanations for the performance of the FA algorithm. Other explanations are required.

### 2.2. Theory

Having established that weight and gradient alignments are not sufficient to explain the performance of FA, in this subsection we will now sketch how FA manages to find good solutions. The essence is as follows: We start with the observation that the fixed points of FA updates coincide with local extrema of the loss function, but most of these fixed points are unstable under the FA dynamics. After initialisation, the FA update performs a random walk in weight space, until it finds a stable fixed point, at which point it converges to a solution. Mathematically, the stability criterion for the fixed point will turn out to be precisely that alignment is positive.

#### *2.2.1. Notation and basic setup*

We will start by introducing the notation and the basic setup on which the remainder of this paper is based. Throughout, we will consider a feedforward neural network (multi-layer perceptron) parametrised by some weights $\mathbf{w}$. The network takes the vectorised input $\mathbf{x}$ and returns the output vector $\mathbf{m}(\mathbf{x}; \mathbf{w})$. When the input is irrelevant, then we will use the shorthand notation $\mathbf{m}(\mathbf{w})$ to describe the neural network. We consider a network of $L$ layers, where each layer $1 \leqslant l \leqslant L$ comprises $n_l$ artificial neurons, whose output is a scalar non-linear functions $f_i^{(l)}(\cdot)$, where the index $1 \leqslant i \leqslant n_l$ labels the neuron to which this output belongs. The argument to each function is the pre-activation function

$$h_j^{(l)} := \sum_{i=1}^{n_{l-1}} w_{ji}^{(l)} f_i^{(l-1)}$$

with $w_{ji}^{(l)} \in \mathbb{R}$ denoting the parameters (or 'weights') of $h_j^{(l)}$. For convenience, we will write $x_i^{(l)} := f_i^{(l)}$ and in particular $f_i^{(0)} := x_i$ is the input to the network. Throughout this manuscript, we denote the loss function by $\mathcal{L}(\mathbf{m}(\mathbf{x}))$, and assume that it is to be minimised via gradient descent, although all our conclusions will remain valid for gradient ascent problems.

#### *2.2.2. BP and FA algorithms*

Using this notation, we can formulate the BP update rule for layer $l$ of a feedforward multi-layer perceptron as

$$\Delta^{\mathrm{BP}} w_{pq}^{(l)} := \frac{\partial \mathcal{L}}{\partial f_i^{(L)}} \frac{\partial f_i^{(L)}}{\partial h_j^{(L)}} \frac{\partial h_j^{(L)}}{\partial f_l^{(L-1)}} \frac{\partial f_l^{(L-1)}}{\cdots} \cdots \frac{\cdots}{\partial f_k^{(l)}} \frac{\partial f_k^{(l)}}{\partial h_s^{(l)}} \frac{\partial h_s^{(l)}}{\partial w_{pq}^{(l)}}. \tag{1}$$

Here (and in the following), we use the convention that repeated indices are summed over, that is $a_i b_i := \sum_i a_i b_i$; note that this convention does not apply to the superscripts in parenthesis that indicate the layer.

Equation (1) can be evaluated, by noting that the function $f_i^{(l)}$ only depends on $h_i^{(l)}$, and furthermore $\frac{\partial h_i^{(l)}}{\partial w_{jk}^{(l)}} = \delta_{ij} f_k^{(l-1)}$, where $\delta_{ij}$ is 1 if $i = j$ and 0 otherwise. Thus, equation (1) reduces to

$$\Delta^{\mathrm{BP}} w_{pq}^{(l)} = \partial \mathcal{L}_i \widetilde{B}_{ik}^{(l)} \frac{\partial f_k^{(l)}}{\partial h_s^{(l)}} \frac{\partial h_s^{(l)}}{\partial w_{pq}^{(l)}} = \partial \mathcal{L}_i \widetilde{B}_{ik}^{(l)} \partial f_{ks}^{(l)} \delta_{ps} f_q^{(l-1)} = \partial \mathcal{L}_i \widetilde{B}_{ik}^{(l)} \partial f_{kp}^{(l)} f_q^{(l-1)}. \tag{2}$$

Here we abbreviated the partial derivative of the loss by $\partial \mathcal{L}_j := (\partial \mathcal{L} / \partial f_i^{(L)})(\partial f_i^{(L)} / \partial h_j^{(L)})$. We also defined $\partial f_{ks}^{(l)} := \partial f_k^{(l)} / \partial h_s^{(l)}$, and $\widetilde{B}_{ik}^{(l)}$ is a shorthand for the middle terms of the chain rule of equation (1). Note that, $\partial f_{ks}^{(l)}$ is zero for $k \neq s$.

FA is the same as BP, except that the terms $\partial h_i^{(\cdot)}/\partial f_j^{(\cdot)}$ appearing in $\widetilde{B}_{ik}^{(l)}$ are replaced by randomly chosen (but fixed) numbers $R_{ij}$, drawn from some user-determined distribution. This leads to the partially random feedback matrices $\mathbf{B}^{(l)}$ with elements $B_{ik}^{(l)}$. The *FA pseudo-gradient* in layer $l$ is defined by

$$\Delta^{\mathrm{FA}} w_{pq}^{(l)} := \partial \mathcal{L}_i B_{ik}^{(l)} \partial f_{kp}^{(l)} x_q^{(l-1)}. \tag{3}$$

Note, that the rhs of the equation is not a derivative of a a particular function.

DFA is the same as FA, except that in each layer $l$ the matrix $\mathbf{B}^{(l)}$ is replaced by a randomly chosen, but fixed matrix $\mathbf{R}^{(l)}$.

*2.2.3. Fixed points under FA are local extrema of the loss function*

The dynamics induced by the FA-pseudo-gradient (equation (3)) constitutes a random walk in weight space. The randomness is introduced via the choice of the particular training example for the next weight update. Therefore, FA is not a gradient descent/ascent algorithm, except of course in the output layer which follows a gradient to a local extremum of the loss function (in exactly the same way as BP).

We will now show that the FA pseudo-gradient update shares with BP a number of fixed points in weight space. These correspond to local extrema of the loss function. Under certain conditions, FA will converge to those. In order to understand the difference between the FA and BP it is instructive to consider the update in the penultimate layer of the network (which has the simplest form). In the case of BP, this is:

$$\Delta^{\mathrm{BP}} w_{pq}^{(L-1)} = \partial \mathcal{L}_j \frac{\partial h_j^{(L)}}{\partial f_l^{(L-1)}} \partial f_{lk}^{(L-1)} \frac{\partial h_k^{(L-1)}}{\partial w_{pq}^{(L-1)}} = \partial \mathcal{L}_j w_{jl}^{(L)} \partial f_{lk}^{(L-1)} \frac{\partial h_k^{(L-1)}}{\partial w_{pq}^{(L-1)}}. \tag{4}$$

The corresponding expression in the case of FA is then:

$$\Delta^{\mathrm{FA}} w_{pq}^{(L-1)} = \partial \mathcal{L}_j R_{jl}^{(L)} \partial f_{lk}^{(L-1)} \frac{\partial h_k^{(L-1)}}{\partial w_{pq}^{(L-1)}} \tag{5}$$

where $\mathbf{R}$ is a randomly chosen, but fixed matrix. This formulates the update for the weight $(p, q)$ in layer $(L-1)$. We can now partially evaluate this for a standard multi-layer perceptron. In particular, the term to the right of $R_{jl}^{(L)}$ becomes:

$$\partial f_{lk}^{(L-1)} \frac{\partial h_k^{(L-1)}}{\partial w_{pq}^{(L-1)}} = \partial f_{lk}^{(L-1)} \delta_{kp} x_q^{(L-2)} = \partial f_{lk}^{(L-1)} \delta_{kp} x_q^{(L-2)} = \delta_{lp} x_q^{(L-2)} f'_{(p,L-1)}$$

where in the first line, we simply expressed the fact that the pre-activation function $h_i^{\cdot}$ can only depend on its own weights $w_{i\cdot}$, and $f'_{(i,L-1)} := \partial f_{jj}^{(L-1)}|_{j=i}$ is a number rather than an indexed variable, so in the above equation the summation convention does not apply. We can write the pseudogradient of the penultimate layer as:

$$\Delta^{\mathrm{FA}} w_{pq}^{(L-1)} = \delta \mathcal{L}_j R_{jp}^{(L)} x_q^{(L-2)} f'_{(p,L-1)}. \tag{6}$$

Given that $\mathbf{R}^{(L)}$ is a random matrix with iid entries, it will be full rank and therefore $\partial \mathcal{L}_j R_{jp}^{(L)} = 0$ for all $l$ only if $\partial \mathcal{L}_j = 0$ for all $j$. Consequently, the entire expression will only vanish for all $p, q$ if $x_q^{(L-2)} f'_{(p,L-1)} = 0$ for all $p, q$, or if the error $\partial \mathcal{L}_j = 0$ for all $j$. The former case corresponds to vanishing gradients, if $f'_{(p,L-1)} = 0$ for every $p$. Note that these vanishing gradient would affect BP as much as it affects updates under FA. In the more interesting latter case, the local extrema of the pseudogradient are also local extrema of the gradient.

In the case of DFA the same argument can be made for every layer, and we find that a fixed point of DFA is a local extremum under BP as well. In the case of FA we have at every layer $l$ the partially random matrix $\mathbf{B}^{(l)}$ instead of $\mathbf{R}^{(l)}$; see equation (3). The deterministic terms in $\mathbf{B}^{(l)}$ are the local gradients $\partial f_{\cdot\cdot}^{(\cdot)}$, which also appear in the expression for BP. Therefore, for as long as the weight matrices are full rank then the fixed points under DFA will also correspond to local extrema of BP.

The weight matrices appearing in the computation of BP are not necessarily full rank, whence it is conceivable that the gradient $\Delta^{\mathrm{BP}} w_{pq}^{(l)}$ vanishes even if $\partial \mathcal{L}_i \neq 0$ for at least some $i$. There are therefore fixed-points under BP that are not fixed points under the FA pseudo-gradient update. One particular example of this is the trivial case of a vanishing weight matrix. This leads to vanishing weight updates under BP, but not under FA or DFA. We conclude that the fixed points of FA will be local extrema of the loss function, and local attractors under the BP.

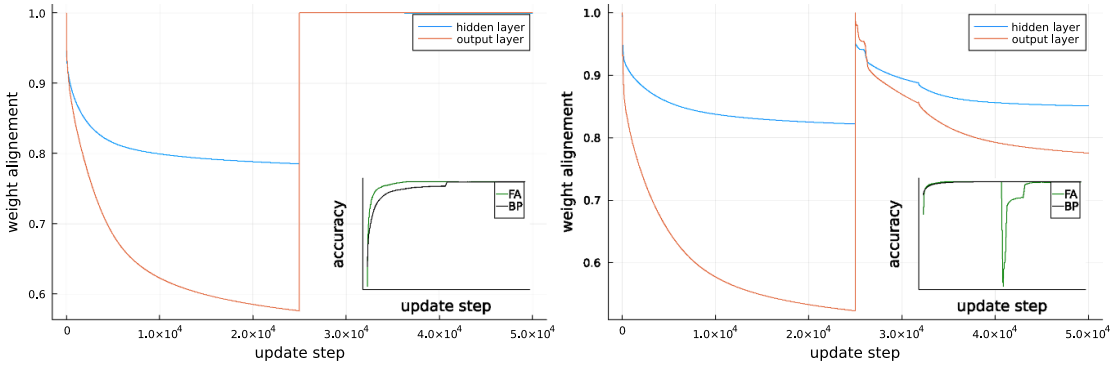



**Figure 5.** FA and BP were trained on the same sequence of examples from MNIST starting from the same initial conditions. The graphs show the weight alignment over time between FA and BP for the second hidden and output layer. A value of 1 means that FA and BP have the same weights up to a constant factor. (a) Initially, the two algorithms diverge. After 25 000 update steps the weights of the FA network were transferred to the network trained using BP. Following this, they remain aligned, demonstrating that the solution found by FA remains stable under BP. The inset shows the accuracy for both for reference.(b) Same, but at 25 000 the FA receives the weights found by BP. This is not stable and the two solutions diverge quickly. Note how the accuracy of the FA drops immediately following the weight change, indicating that it is repelled from the loss extremum.

*2.2.4. Deriving a stability criterion*

Recall that the update of the final layer is an exact gradient descent, which is the same under FA and BP. The weights of the final layer will therefore be driven to values corresponding to an extremum of the loss function. Under BP all other layers are driven to corresponding extrema, such that BP then settles on a particular overall extremum for all layers. This does, however, not necessarily happen under FA because the FA pseudo-gradient performs a random walk in all but the final layer. While the local extrema of the loss function to which BP would evolve are fixed points under FA pseudo-gradient, they will typically not be stable, thus preventing the FA pseudo gradient updates to remain in a particular neighbourhood of a fixed point. See figure 5 for an illustration. Convergence under the FA update will only happen when the weights in all layers are close to stable fixed points.

The question is now under which conditions fixed points are stable under the FA pseudo-gradient update. Stability here means that under the FA update $\Delta^{\mathrm{FA}}\mathbf{w}$ the loss does not increase. Formally, this can be expressed as:

$$\mathcal{L}\left(\mathbf{m}\left(\mathbf{w}\right)\right) - \mathcal{L}\left(\mathbf{m}\left(\mathbf{w} - \Delta^{\mathrm{FA}}\mathbf{w}^{(l)}\right)\right) \geqslant 0. \tag{7}$$

Here, we suppressed the label superscripts for clarity and wrote $\mathbf{w}$ instead of $\mathbf{w}^{(l)}$. Assuming small weight updates, we can expand the argument to the second term to first order by writing

$$\mathbf{m}\left(\mathbf{w} - \Delta^{\mathrm{FA}}\mathbf{w}\right) \approx \mathbf{m}\left(\mathbf{w}\right) - \mathbf{m}'\left(\mathbf{w}\right)\Delta^{\mathrm{FA}}\mathbf{w}, \tag{8}$$

where $\mathbf{m}'$ is a three-dimensional matrix with elements $m'_{ijk} := \partial m_i / \partial w_{jk}$. We can now also expand the loss function to first order to obtain

$$\mathcal{L}\left(\mathbf{m}\left(\mathbf{w}\right)\right) - \mathcal{L}\left(\mathbf{m}\left(\mathbf{w} - \Delta^{\mathrm{FA}}\mathbf{w}\right)\right) \approx \frac{\partial \mathcal{L}}{\partial \mathbf{m}}\mathbf{m}'\left(\mathbf{w}\right)\Delta^{\mathrm{FA}}\mathbf{w}. \tag{9}$$

Thus the stability criterion becomes, in first order approximation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{m}}\mathbf{m}'\left(\mathbf{w}\right)\Delta^{\mathrm{FA}}\mathbf{w} \geqslant 0. \tag{10}$$

We observe that $\frac{\partial \mathcal{L}}{\partial \mathbf{m}}\mathbf{m}'(\mathbf{w})$ is just $\Delta^{\mathrm{BP}}\mathbf{w}$, and $\Delta^{\mathrm{FA}}\mathbf{w}$ is the FA pseudo-gradient. Furthermore, we see that the rhs of equation (10) is, up to normalisation, the alignment measure of the FA update with the BP gradient. Thus, equation (10) formulates a necessary condition for the stability of a local extremum with respect to the update by the FA pseudo-gradient. This stability criterion states that the alignment measure needs to be positive.

$$\frac{\left(\Delta^{\mathrm{BP}}\mathbf{w}\right)\cdot\left(\Delta^{\mathrm{FA}}\mathbf{w}\right)}{\|\Delta^{\mathrm{BP}}\mathbf{w}\|\cdot\|\Delta^{\mathrm{FA}}\mathbf{w}\|} \geqslant 0. \tag{11}$$

Put differently, we can now say that gradient alignment is a stability criterion for FA.

Two remarks are in order to understand this result. Firstly, we derived the stability criterion for an unspecified layer $l$. This means, that the criterion is meaningful for any layer and there are no assumptions in this derivations, for example, that the neural network has to be of a particular depth. Secondly, in order for a particular point in weight space to be stable, the stability criterion must hold for every layer. Naturally, one would assume that the deeper the network the harder it is to find such stable points.

## 3. Discussion

At a first glance, FA and DFA should not work. By replacing a key term in the update equation, the gradient descent of BP is effectively transformed into a random walk in weight-space. In agreement with the literature, we found that updates made during this random walk begin to align with the 'true' BP gradient after a few time-steps. In the literature this alignment is commonly assumed to be the driver for the performance of the FA algorithm. It remains unclear, however, why FA aligns. Existing mathematical models only cover special simplified cases of neural networks. They suggest that the update dynamics of FA leads to weight alignment, which then implies gradient alignment. In that sense, FA approximates the true gradient.

Our theoretical results suggest a somewhat different view: FA is not approximating BP at all, and indeed does not descend or ascend the gradient of a loss function or an approximation thereof. It is not 'learning' in the sense one normally understands this term. Instead, it performs a random walk in weight space. It works based on the following conjectured mechanism:

- Initially, the walker moves randomly through parameter space in the hidden layers.
- In the output layer, the error is minimised, driving the derivative of the loss function $\partial\mathcal{L}_i$ to zero, and hence (as discussed above) towards a fixed point of the FA pseudo-gradient.
- In the other layers, the random walk induced by the FA pseudo-gradient will tend to drive the weights away from extrema of the loss function.
- Convergence to a fixed point will only happen when the random walk finds extrema of the loss function that are also stable under the FA pseudo-gradient. Mathematically, this manifests itself in that the stability criterion is fulfilled.

There is no guarantee that FA finds extrema that are compatible with the alignment criterion. Failure to do so could be either because there are no compatible extrema, or because FA is initialised in a part of parameter space from which it cannot find a route to compatible fixed points. The latter scenario could realise when extrema of the loss function are sparse in parameter space or when the weights are initialised in an area that has small update steps, for example for large initial weights. The often cited failure of FA for convolutional neural networks is likely a consequence of the sparseness of loss extrema for those networks.

Some open questions remain. The most pressing one relates to the distribution of local extrema of the loss function in parameter space. In particular, it may be that for certain types of problems there are conditions that guarantee that FA and DFA find local extrema or alternatively, that they do not find such extrema. We conjecture that the effectiveness of FA algorithms is directly related to this distribution. Presumably, convolutional neural networks, which operate in relatively low-dimensional weight spaces, have only sparse minima, which is the reason why FA does not work well on those networks.

## 4. Methods

Unless otherwise stated, we used a feed forward multi-layer perceptron with 2 hidden layers of 700/1000 neurons. The size of the network was chosen to be large, while still being fast to execute. Our results are not sensitive to variations of the size of the network, although classification performance clearly is. Throughout, we used tanh as the activation function; we also experimented with `relu` which led to qualitatively the same results (data not shown). For the final layer we used the softmax function and as a loss function we used the cross-entropy. The batch size was chosen to be 100 and the learning rate was set to 0.05. Again, our results are not sensitive to those parameters.

Unless stated otherwise, feedback matrices were chosen randomly by drawing matrix elements from the set $\{-1, 1\}$. We found FA not to be overly sensitive to the particular choice of this set. However, we found algorithm performance to depend on it to some extent. The particular choice we made gives good performance, but we made no attempt to choose the optimal one.

For all layers, initial weights were drawn from a normal distribution and then scaled with a weight scaling factor, resulting in both negative and positive weights. If the factor is zero, then all weights were initially zero. The larger the factor, the larger the (absolute value of) initial weights.

All simulations were done using Julia (1.8.5) Flux for gradient computations and network construction, and CUDA for simulations on GPUs. Throughout no optimisers were used. Weight updates were made by adding the gradient/FA-pseudo gradient, scaled by the learning rate, to the weights. The networks were trained on the MNIST dataset as included with the Flux library. Whenever an accuracy is reported it was computed based on the test-set of the MNIST dataset.

## Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

## Appendix. Two basic properties of FA

### A.1. Vanishing weights are unstable for some activation functions

As pointed out by [19], for BP the initialisation $w_{ij} = 0$ results in a deadlock, in the sense that irrespective of the inputs, the weight updates will always vanish. This can be seen clearly from equation (2): For each layer, the matrices $\partial h_i^{(l)}/\partial f_j^{(l)}$ evaluate to $w_{ij}^{(l)}$. Therefore, trivially the weight updates will be vanishing as well. In the case of FA, these expressions are replaced by non-vanishing random terms. With this change, the update of the $l = 1$ layer,

$$\Delta w_{pq} = \partial \mathcal{L}_i B_{ij}^{(1)} f_{jp}^{(1)} x_q$$

is no longer proportional to the weights, and hence not identically zero. Upon the first update, the weights of the first layer will typically be non-vanishing, such that the update of the $l = 2$ layer will be non-vanishing as well. After $L$ updates, all layers may have weights that are different from 0. The initialisation $w_{ij} = 0$ is therefore not a fixed points of the FA pseudo-update rule.

### A.2. Weight increments tend to zero for large weights for some choices of activation function

The behaviour for large weights depends on the activation functions that are chosen. It is therefore not possible to make general statements. However, many popular choices for these functions will lead to vanishing weight updates. We restrict the discussion to the choices made here. The derivative of the softmax function $\sigma(x_1, \ldots, x_n)$ can be seen to vanish at infinite weights. The derivative of the first component of the softmax function with respect to $x_1$ gives

$$\frac{\partial \sigma_1}{\partial x_1} = \frac{\exp(x_1)}{\sum_i \exp(x_i)} - \left(\frac{\exp(x_1)}{\sum_i \exp(x_i)}\right)^2.$$

It is straightforward to confirm that as $\max(x_i) \to \infty$ the first and the second term simultaneously either go to 0 or 1. Similarly, for $k \neq 1$

$$\frac{\partial \sigma_1}{\partial x_k} = -\frac{\exp(x_1)}{\sum_i \exp(x_k)} \frac{\exp(x_i)}{\sum_i \exp(x_i)}.$$

Here at least one of the two terms vanishes for as long as the difference between any two $x_j$ is sufficiently large.

From the shape of the $\tanh(x)$ it is clear that its derivative vanishes with $x \to \pm\infty$. The same is not true, however, for relu which has a constant derivative.

We conclude that for networks that use the softmax function weight updates will vanish in the regime of large weights when the softmax function is used in the last layer.

## ORCID iD

Dominique Chu https://orcid.org/0000-0002-3706-2905

## References

[1] Rumelhart D E, Hinton G E and Williams R J 1986 Learning representations by back-propagating errors *Nature* **323** 533–6
[2] LeCun Y and Cortes C 2010 MNIST handwritten digit database (available at: http://yann.lecun.com/exdb/mnist/)
[3] Launay J, Poli I, Boniface F and Krzakala F 2020 Direct feedback alignment scales to modern deep learning tasks and architectures *Proc. 34th Int. Conf. on Neural Information Processing Systems, NIPS'20* (Curran Associates Inc.)
[4] Huo Z, Gu B, Yang Q and Huang H 2018 Decoupled parallel backpropagation with convergence guarantee (arXiv:1804.10574)
[5] Crafton B, Parihar A, Gebhardt E and Raychowdhury A 2019 Direct feedback alignment with sparse connections for local learning *Front. Neurosci.* **13** 450947

[6] Han D and Yoo H-J 2019 Direct feedback alignment based convolutional neural network training for low-power online learning processor *IEEE/CVF Int. Conf. on Computer Vision Workshop (ICCVW)* pp 2445–52
[7] Sarwar S, Srinivasan G, Han B, Wijesinghe P, Jaiswal A, Panda P, Raghunathan A and Roy K 2018 Energy efficient neural computing: a study of cross-layer approximations *IEEE J. Emerg. Sel. Top. Circuits Syst.* **8** 796–809
[8] Strubell E, Ganesh A and McCallum A 2019 Energy and policy considerations for deep learning in NLP *Proc. 57th Annual Meeting of the Association for Computational Linguistics* (Association for Computational Linguistics) pp 3645–50
[9] Neftci E O, Augustine C, Paul S and Detorakis G 2017 Event-driven random back-propagation: enabling neuromorphic deep learning machines *Front. Neurosci.* **11** 255082
[10] Davies M *et al* 2018 Loihi: a neuromorphic manycore processor with on-chip learning *IEEE Micro* **38** 82–99
[11] Plana L, Clark D, Davidson S, Furber S, Garside J, Painkras E, Pepper J, Temple S and Bainbridge J 2011 Spinnaker: design and implementation of a gals multicore system-on-chip *J. Emerg. Technol. Comput. Syst.* **7** 17:1–17:18
[12] Hinton G 2022 The forward-forward algorithm: some preliminary investigations (arXiv:2212.13345)
[13] Lillicrap T P, Cownden D, Tweed D B and Akerman C J 2016 Random synaptic feedback weights support error backpropagation for deep learning *Nat. Commun.* **7** 13276
[14] Nøkland A and Eidnes L H 2019 Training neural networks with local error signals *Proc. 36th Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 97) ed K Chaudhuri and R Salakhutdinov (PMLR) pp 4839–50
[15] Worden L and Levin S 2007 Evolutionary escape from the prisoner's dilemma *J. Theor. Biol.* **245** 411–22
[16] Sanfiz A and Akrout M 2021 Benchmarking the accuracy and robustness of feedback alignment algorithms (arXiv:2108.13446)
[17] Zhao D, Zeng Y, Zhang T, Shi M and Zhao F 2020 Glsnn: a multi-layer spiking neural network based on global feedback alignment and local stdp plasticity *Front. Comput. Neurosci.* **14** 576841
[18] Launay J, Poli I and Krzakala F 2019 Principled training of neural networks with direct feedback alignment
[19] Refinetti M, D'Ascoli S, Ohana R and Goldt S 2021 Align, then memorise: the dynamics of learning with feedback alignment *Int. Conf. on Machine Learning* pp 8925–35
[20] Saad D and Solla S A 1995 Exact solution for on-line learning in multilayer neural networks *Phys. Rev. Lett.* **74** 4337–40
[21] Song G, Xu R and Lafferty J 2021 Convergence and alignment of gradient descent with random backpropagation weights *Advances in Neural Information Processing Systems* vol 34, ed M Ranzato, A Beygelzimer, Y Dauphin, P Liang and J W Vaughan (Curran Associates, Inc.) pp 19888–98
[22] Gorban A and Tyukin I 2018 Blessing of dimensionality: mathematical foundations of the statistical physics of data *Phil. Trans. R. Soc.* A **376** 20170237