Direct Feedback Alignment Scales to Modern Deep Learning Tasks and Architectures

Julien Launay[1,2] **Iacopo Poli**[1] **François Boniface**[1] **Florent Krzakala**[1,2,3]

[1]LightOn [2]LPENS, École Normale Supérieure [3] IdePhics, EPFL

{firstname}@lighton.ai

lair.lighton.ai/dfa-scales

Abstract

Despite being the workhorse of deep learning, the backpropagation algorithm is no panacea. It enforces sequential layer updates, thus preventing efficient parallelization of the training process. Furthermore, its biological plausibility is being challenged. Alternative schemes have been devised; yet, under the constraint of synaptic asymmetry, none have scaled to modern deep learning tasks and architectures. Here, we challenge this perspective, and study the applicability of Direct Feedback Alignment (DFA) to neural view synthesis, recommender systems, geometric learning, and natural language processing. In contrast with previous studies limited to computer vision tasks, our findings show that it successfully trains a large range of state-of-the-art deep learning architectures, with performance close to fine-tuned backpropagation. When a larger gap between DFA and backpropagation exists, like in Transformers, we attribute this to a need to rethink common practices for large and complex architectures. At variance with common beliefs, our work supports that challenging tasks can be tackled in the absence of weight transport.

1 Introduction

While the backpropagation algorithm (BP) [1, 2] is at the heart of modern deep learning achievements, it is not without pitfalls. For one, its weight updates are non-local and rely on upstream layers. Thus, they cannot be easily parallelized [3], incurring important memory and compute costs. Moreover, its biological implementation is problematic [4, 5]. For instance, BP relies on the transpose of the weights to evaluate updates. Hence, synaptic symmetry is required between the forward and backward path: this is implausible in biological brains, and known as the weight transport problem [6].

Consequently, alternative training algorithms have been developed. Some of these algorithms are explicitly biologically inspired [7–13], while others focus on making better use of available compute resources [3, 14–19]. Despite these enticing characteristics, none has been widely adopted, as they are often demonstrated on a limited set of tasks. Moreover, as assessed in [20], their performance on challenging datasets under the constraint of synaptic asymmetry is disappointing.

We seek to broaden this perspective, and demonstrate the applicability of Direct Feedback Alignment (DFA) [19] in state-of-the-art settings: from applications of fully connected networks such as neural view synthesis and recommender systems, to geometric learning with graph convolutions, and natural language processing with Transformers. Our results define new standards for learning without weight transport and show that challenging tasks can indeed be tackled under synaptic asymmetry.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

1.1 Related work

Training a neural network is a credit assignment problem: an update is derived for each parameter from its contribution to a cost function. To solve this problem, a spectrum of algorithms exists [21].

Biologically motivated methods Finding a training method applicable under the constraints of biological brains remains an open problem. End-to-end propagation of gradients is unlikely to occur [22], implying local learning is required. Furthermore, the weight transport problem enforces synaptic asymmetry [6]. Inspired by auto-encoders, target propagation methods (TP) [10–12] train distinct feedback connections to invert the feedforward ones. Feedback alignment (FA) [13] replaces the transpose of the forward weights used in the backward pass by a random matrix. Throughout training, the forward weights learn to *align* with the arbitrary backward weights, eventually approximating BP.

Beyond biological considerations As deep learning models grow bigger, large-scale distributed training is increasingly desirable. Greedy layer-wise training [14] allows networks to be built layer by layer, limiting the depth of backpropagation. To enable parallelization of the backward pass, updates must only depend on local quantities. Unsupervised learning is naturally suited for this, as it relies on local losses such as Deep InfoMax [17] and Greedy InfoMax [18]. More broadly, synthetic gradient methods, like decoupled neural interfaces [3, 15] and local error signals (LES) [16], approximate gradients using layer-wise trainable feedback networks, or using reinforcement learning [23]. DFA [19] expands on FA and directly projects a global error to each layer. A shared feedback path is still needed, but it only depends on a simple random projection operation.

Performance of alternative methods Local training methods are successful in unsupervised learning [18]. Even in a supervised setting, they scale to challenging datasets like CIFAR-100 or ImageNet [14, 16]. Thus, locality is not too penalizing. However, FA, and DFA are unable to scale to these tasks [20]. In fact, DFA is unable to train convolutional layers [24], and has to rely on transfer learning in image tasks [25]. To enable feedback alignment techniques to perform well on challenging datasets, some form of weight transport is necessary: either by explicitly sharing sign information [26–28], or by introducing dedicated phases of alignment for the forward and backward weights where some information is shared [29, 30]. To the best of our knowledge, no method compatible with the weight transport problem has ever been demonstrated on challenging tasks.

1.2 Motivations and contributions

We focus on DFA, a compromise between biological and computational considerations. Notably, DFA is compatible with synaptic asymmetry: this asymmetry raises important challenges, seemingly preventing learning in demanding settings. Moreover, it allows for asynchronous weight updates, and puts a single operation at the center of the training stage. This enables new classes of training co-processors [31, 32], leveraging dedicated hardware to perform the random projection.

Extensive survey We apply DFA in a large variety of settings matching current trends in machine learning. Previous works have found that DFA is unsuitable for computer vision tasks [20, 24]; but computer vision alone cannot be the litmus test of a training method. Instead, we consider four vastly different domains, across eight tasks, and with eleven different architectures. This constitutes a survey of unprecedented scale for an alternative training method, and makes a strong case for the possibility of learning without weight transport in demanding scenarios.

Challenging settings We demonstrate the ability of DFA to tackle challenging tasks. We successfully learn and render real-world 3D scenes (section 3.1.1); we perform recommendation at scale (section 3.1.2); we explore graph-based citation networks (section 3.2); and we consider language modelling with a Transformer (section 3.3). We study tasks at the state-of-the-art level, that have only been recently successfully tackled with deep learning.

Modern architectures We prove that the previously established failure of DFA to train convolutions does not generalize. By evaluating performance metrics, comparing against a shallow baseline, measuring alignment, and visualizing t-SNE embeddings, we show that learning indeed occurs in layers involving graph convolutions and attention. This significantly broadens the applicability of DFA–previously thought to be limited to simple problems like MNIST and CIFAR-10.

2 Methods

Forward pass In a fully connected network, at layer i out of N, neglecting its biases, with $\mathbf{W}_i$ its weight matrix, f_i its non-linearity, and $\mathbf{h}_i$ its activations, the forward pass is:

$$\forall i \in [1, \dots, N] : \mathbf{a}_i = \mathbf{W}_i \mathbf{h}_{i-1}, \mathbf{h}_i = f_i(\mathbf{a}_i). \tag{1}$$

$\mathbf{h}_0 = X$ is the input data, and $\mathbf{h}_N = f(\mathbf{a}_N) = \hat{\mathbf{y}}$ are the predictions. A task-specific cost function $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ is computed to quantify the quality of the predictions with respect to the targets $\mathbf{y}$.

Backward pass with BP The weight updates are computed by backpropagation of the error vector. Using the chain-rule of derivatives, each neuron is updated based on its contribution to the cost function. Leaving aside the specifics of the optimizer used, the equation for the weight updates is:

$$\delta \mathbf{W}_i = -\frac{\partial \mathcal{L}}{\partial \mathbf{W}_i} = -[(\mathbf{W}_{i+1}^T \delta \mathbf{a}_{i+1}) \odot f_i'(\mathbf{a}_i)] \mathbf{h}_{i-1}^T, \delta \mathbf{a}_i = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} \tag{2}$$

Backward pass with DFA The gradient signal $\mathbf{W}_{i+1}^T \delta \mathbf{a}_{i+1}$ of the (i+1)-th layer violates synaptic asymmetry. DFA replaces it with a random projection of the topmost derivative of the loss, $\delta \mathbf{a}_y$. For common classification and regression losses such as the mean squared error or the negative log likelihood, this corresponds to a random projection of the global error $\mathbf{e} = \hat{\mathbf{y}} - \mathbf{y}$. With B_i, a fixed random matrix of appropriate shape drawn at initialization for each layers:

$$\delta \mathbf{W}_i = -[(\mathbf{B}_i \delta \mathbf{a}_y) \odot f_i'(\mathbf{a}_i)] \mathbf{h}_{i-1}^T, \delta \mathbf{a}_y = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_y} \tag{3}$$

We provide details in appendix C regarding adapting DFA beyond fully-connected layers.

3 Experiments

We study the applicability of DFA to a diverse set of applications requiring state-of-the-art architectures. We start with fully connected networks, where DFA has already been demonstrated, and address new challenging settings. We then investigate geometric learning: we apply DFA to graph neural networks in classification tasks on citation networks, as well as graph autoencoders. These architectures feature graph convolutions and attention layers. Finally, we use DFA to train a transformer-based Natural Language Processing (NLP) model on a dataset of more than 100 million tokens.

3.1 Fully connected architectures

DFA has been successful at training fully connected architectures, with performance on-par with backpropagation [19, 20]. However, only computer vision tasks have been considered, where fully connected networks considerably underperform their convolutional counterpart. Here, we focus on tasks where fully connected architectures are state-of-the-art. Moreover, the architectures considered are deeper and more complex than those necessary to solve a simple task like MNIST.

3.1.1 Neural view synthesis with Neural Radiance Fields

The most recent state-of-the-art *neural view synthesis* methods are based on large fully connected networks: this is an ideal setting for a first evaluation of DFA on a challenging task.

Background There has been growing interest in methods capable of synthesising novel renders of a 3D scene using a dataset of past renders. The network is trained to learn an inner representation of the scene, and a classical rendering system can then query the model to generate novel views. With robust enough methods, real-world scenes can also be learned from a set of pictures.

Until recently, most successful neural view synthesis methods were based on sampled volumetric representations [33–35]. In this context, Convolutional Neural Networks (CNNs) can be used to smooth out the discrete sampling of 3D space [36, 37]. However, these methods scale poorly to higher resolutions, as they still require finer and finer sampling. Conversely, alternative schemes based on a continuous volume representation have succeeded in generating high-quality renders [38], even featuring complex phenomenons such as view-dependant scattering [39]. These schemes make point-wise predictions, and use fully connected neural networks to encode the scene. Beyond 3D scenes, continuous implicit neural representations can be used to encode audio and images [40].

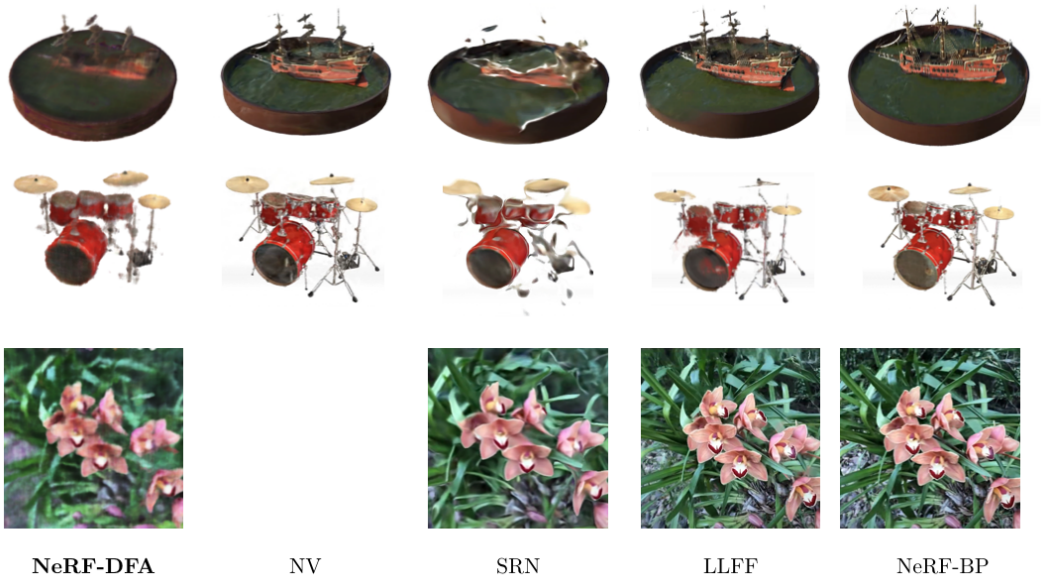


Figure 1: Comparisons of NeRF-DFA with state-of-the-art methods trained with BP on the most challenging synthetic and real-world scenes. While NeRF-DFA generates render of lower quality, they maintain multi-view consistency and exhibit no geometric artefacts. BP results from [39].

Setting We employ Neural Radiance Fields (NeRF) [39], the state-of-the-art for neural view synthesis. NeRF represents scenes as a continuous 5D function of space–three spatial coordinates, two viewing angles–and outputs a point-wise RGB radiance and opacity. A ray-casting renderer can then query the network to generate arbitrary views of the scene. The network modeling the continuous function is 10 layers deep. Two identical networks are trained: the *coarse* network predictions inform the renderer about the spatial coordinates that the *fine* network should preferentially evaluate to avoid empty space and occluded regions.

Results We report quantitative results of training NeRF with DFA in Table 1. Neural view synthesis methods are often better evaluated qualitatively: we showcase some renders in Figure 1.

On a dataset of renders featuring complex scenes with non-Lambertian materials (NeRF-Synthetic [39]), NeRF-DFA outperforms two previous fine-tuned state-of-the-art methods–Scene Representation Networks (SRN) [38] and Local Light Field Fusion (LLFF) [35]–and nearly matches the performance of Neural Volumes (NV) [37]. While DFA underperforms alternative methods trained with BP on the real world view dataset (LLFF-Real [35]), its performance remains significant: real world view synthesis is a challenging tasks, and this level of PSNR indicates that learning is indeed happening.

In particular, we find that NeRF-DFA retains the key characteristics of NeRF-BP: it can render view-dependant effects, and is multi-view consistent. The last point is an especially important achievement, and most visible in the video linked in appendix E, as it is a challenge for most algorithms [33–35, 38]. The main drawback of NeRF-DFA appears to be a seemingly lower render definition. The

Table 1: Peak Signal to Noise Ratio (PSNR, higher is better) of neural view synthesis methods trained with backpropagation against NeRF trained with DFA. Even when trained with DFA, NeRF outperforms two state-of-the-art methods on a synthetic dataset (NeRF-Synthetic), and achieves fair performance on a challenging real world views datasets (LLFF-Real). BP results from [39].

	NV	**SRN**	**LLFF**	**NeRF**	
	BP	BP	BP	BP	DFA
NeRF-Synthetic	26.05	22.26	24.88	31.01	25.41
LLFF-Real	/	22.84	24.13	26.50	20.77

NeRF architecture has not been fine-tuned to achieve these results: DFA works out-of-the-box on this advanced method. Future research focusing on architectural changes to NeRF could improve performance with DFA; some preliminary results are included in appendix E of the supplementary.

3.1.2 Click-through rate prediction with recommender systems

We have demonstrated that DFA can train large fully connected networks on the difficult task of neural view synthesis. We now seek to use DFA in more complex heterogeneous architectures, combining the use of fully connected networks with other machine learning methods. *Recommender systems* are an ideal application for such considerations.

Background Recommender systems are used to model the behavior of users and predict future interactions. In particular, in the context of click-through rate (CTR) prediction, these systems model the probability of a user clicking on a given item. Building recommender systems is hard [41]: their input is high-dimensional and sparse, and the model must learn to extract high-order combinatorial features from the data. Moreover, they need to do so efficiently, as they are used to make millions of predictions and the training data may contain billions of examples.

Factorization Machines (FM) [42] use inner-products of latent vectors between features to extract pairwise feature interactions. They constitute an excellent baseline for shallow recommender systems, but fail to efficiently transcribe higher-level features. To avoid extensive feature engineering, it has been suggested that deep learning can be used in conjunction with wide shallow models to extract these higher-level features [43]. In production, these systems are regularly retrained on massive datasets: the speedup allowed by backward unlocking in DFA is thus of particular interest.

Setting Deep Factorization Machines (DeepFM) [44] combine FM and a deep fully connected neural network, which we train with DFA. The input embedding is still trained directly via gradient descent, as weight transport is not necessary to backpropagate through the FM. Deep & Cross Networks (DCN) [45] replace the FM with a Cross Network, a deep architecture without non-linearities capable of extracting high-degree interactions across features. We train the fully connected network, the deep cross network, and the embeddings with DFA. Finally, Adaptative Factorization Network (AFN) [46] uses Logarithmic Neural Networks [47] to enhance the representational power of its deep component. We evaluate these methods on the Criteo dataset [48], which features nearly 46 million samples of one million sparse features. This is a difficult task, where performance improvements of the AUC on the *0.001-level* can enhance CTR significantly [43].

Results Performance metrics are reported in Table 2. To obtain these results, a simple hyperparameter grid search over optimization and regularization parameters was performed for BP and DFA independently. DFA successfully trains all methods above the FM baseline, and in fact matches BP performance in both DeepFM and AFN. Because of their complexity, recommender systems require intensive tuning and feature engineering to perform at the state-of-the-art level–and reproducing existing results can be challenging [49]. Hence, it is not surprising that a performance gap exists with Deep&Cross–further fine-tuning may be necessary for DFA to reach BP performance.

Alignment measurements corroborate that learning is indeed occurring in the special layers of Deep&Cross and AFN–see appendix A of the supplementary for details. Our results on recommender systems support that DFA can learn in a large variety of settings, and that weight transport is not necessary to solve a difficult recommendation task.

Table 2: AUC (higher is better) and log loss (lower is better) of recommender systems trained on the Criteo dataset [48]. Even in complex heterogeneous architectures, DFA performance is in line with BP. Values in **bold** indicate DFA AUC within 0.001 from the BP AUC or better.

	FM	**DeepFM**		**Deep&Cross**		**AFN**	
		BP	DFA	BP	DFA	BP	DFA
AUC	0.7915	0.7954	**0.7956**	0.8104	0.8009	0.7933	**0.7924**
Loss	0.4687	0.4610	**0.4624**	0.4414	0.4502	0.4630	**0.4621**

3.2 Geometric Learning with Graph Convolutional Networks

The use of sophisticated architectures beyond fully connected layers is necessary for certain tasks, such as *geometric learning* [50], where information lies in a complex structured domain. To address geometric learning tasks, methods capable of handling graph-based data are commonly needed. Graph convolutional neural networks (GCNNs) [51–54] have demonstrated the ability to process large-scale graph data efficiently. We study the applicability of DFA to these methods, including recent architectures based on an attention mechanism. Overall, this is an especially interesting setting, as DFA fails to train more classic 2D image convolutional layers [24].

Background Complex data like social networks or brain connectomes lie on irregular or non-Euclidean domains. They can be represented as graphs, and efficient processing in the spectral domain is possible. Non-spectral techniques to apply neural networks to graphs have also been developed [55–57], but they exhibit unfavorable scaling properties. The success of CNNs in deep learning can be attributed to their ability to efficiently process structured high-dimensional data by sharing local filters. Thus, a generalization of the convolution operator to the graph domain is desirable: [51] first proposed a spectral convolution operation for graphs, and [52] introduced a form of regularization to enforce spatial locality of the filters. We use DFA to train different such GCNNs implementations. We study both spectral and non-spectral convolutions, as well as methods inspired by the attention mechanism. We consider the task of semi-supervised node classification: nodes from a graph are classified using their relationship to other nodes as well as node-wise features.

Setting Fast Localized Convolutions (ChebConv) [53] approximate the graph convolution kernel with Chebyshev polynomials, and are one of the first scalable convolution methods on graph. Graph Convolutions (GraphConv) [54] remove the need for an explicit parametrization of the kernel by enforcing linearity of the convolution operation on the graph Laplacian spectrum. It is often considered as the canonical graph convolution. More recent methods do not operate in the spectral domain. Spline Convolutions (SplineConv) [58] use a spline-based kernel, enabling the inclusion of information about the relative positioning of nodes, enhancing their representational power–for instance in the context of 3D meshes. Graph Attention Networks (GATConv) [59] use self-attention [60] layers to enable predictions at a given node to *attend* more specifically to certain parts of its neighborhood. Finally, building upon Jumping Knowledge Network [61], Just Jump (DNAConv) [62] uses multi-head attention [63] to enhance the aggregation process in graph convolutions and enable deeper architectures. Note our implementation of DFA allows for limited weight transport within attention – see appendix D. We use PyTorch Geometric [64] for implementation of all of these methods. We evaluate performance on three citation network datasets: Cora, CiteSeer, and PubMed [65].

Results We report classification accuracy in Table 3. BP and DFA regularization and optimization hyperparameters are fine-tuned separately on the Cora dataset. In general, we find that less regularization and lower learning rates are needed with DFA. DFA successfully trains all graph methods, independent of whether they use the spectral domain or not, and even if they use attention. Furthermore, for GraphConv, SplineConv, and GATConv DFA performance nearly matches BP.

As GCNNs struggle with learning meaningful representations when stacking many layers [66], all architectures but DNAConv are quite shallow (two layers). However, DFA performance is still significantly higher than that of a shallow training method–see appendix B for details. The lower performance on DNAConv is not a failure to learn: alignment measurements in appendix A show that

Table 3: Classification accuracy (%, higher is better) of graph convolution methods trained with BP and DFA, on citation networks [65]. But for ChebConv and DNAConv, DFA performance nearly matches BP performance. Values in **bold** when DFA is within 2.5% of BP.

	ChebConv		**GraphConv**		**SplineConv**		**GATConv**		**DNAConv**	
	BP	DFA	BP	DFA	BP	DFA	BP	DFA	BP	DFA
Cora	79.2	75.4	80.1	**79.9**	81.0	77.7	82.6	**80.6**	84.6	**82.9**
CiteSeer	69.5	**67.6**	71.6	**69.4**	70.0	**69.8**	72.0	**71.2**	73.4	70.8
PubMed	79.5	75.7	78.8	**77.8**	77.5	**77.2**	77.7	**77.1**	87.2	79.9

		GAE	
		BP	DFA
Cora	AUC	0.918	0.900
	AP	0.918	0.900
CiteSeer	AUC	0.886	0.879
	AP	0.895	0.889
PubMed	AUC	0.967	0.945
	AP	0.966	0.945

Table 4: AUC and Average Precision (AP, higher is better) for a GraphConv GAE trained with BP or DFA on citation networks. DFA reproduces BP performance.

Figure 2: t-SNE visualization of the hidden layer activations of a two-layer GraphConv trained on Cora with DFA. Classes forms clear clusters, indicating that a useful intermediary representation is learned. Colors represent different classes.

learning is indeed occurring. It may be explained instead by a need for more in-depth fine-tuning, as this is a deep architecture with 5 successive attention layers.

We further demonstrate that DFA helps graph convolutions learn meaningful representations by aplying t-SNE [67, 68] to the hidden layer activations in GraphConv (Figure 2). Cluster of classes are well-separated, indicating that a useful intermediary representation is derived by the first layer.

Graph autoencoders We consider one last application of graph convolutions, in the context of graph autoencoders (GAE). We train a non-probabilistic GAE [69] based on GraphConv with DFA, and report results in Table 4. DFA performance is always in line with BP.

3.3 Natural Language Processing with Transformers

We complete our study by training a Transformer [63] on a language modelling task. Transformers have proved successful in text, image, music generation, machine translation, and many supervised NLP tasks [63, 70–73]. Here, we demonstrate that DFA can train them, and we show the influence of tuning the optimizer hyperparameters in narrowing the gap with BP.

Background NLP has largely benefited from advances in deep learning. Recurrent Neural Networks were responsible for early breakthroughs, but their sequential nature prevented efficient parallelization of data processing. Transformers are attention-based models that do not rely on recurrence or convolution. Their ability to scale massively has allowed the training of models with several billion parameters [74, 75], obtaining state-of-the-art results on all NLP tasks: Transformers now top the prominent SQuAD 2.0 [76, 77] and SuperGLUE [78] benchmarks. In parallel, transfer learning in NLP has leaped forward thanks to language modelling, the unsupervised task of predicting the next word. It can leverage virtually unlimited data from web scraping [79]. This enabled the training of *universal language models* [80] on extremely large and diversified text corpora. These models are useful across a wide range of domains, and can solve most NLP tasks after fine-tuning.

Setting The prominence of both language modelling and Transformers gives us the ideal candidate for our NLP experiments: we train a Transformer to predict the next word on the WikiText-103 dataset [81], a large collection of *good* and *featured* Wikipedia articles. We use byte-pair-encoding [82] with 32,000 tokens. We adopt a Generative Pre-Training (GPT) setup [70]: we adapt the Transformer, originally an encoder-decoder model designed for machine translation, to language modelling. We keep only the encoder and mask the tokens to predict. Our architecture consists in 6 layers, 8 attention heads, a model dimension of 512, and a hidden size of 2048 in the feed-forward blocks. The text is sliced in chunks of 128 tokens and batches of 64 such chunks, resulting in 8192 tokens per batch. Our baseline is trained with BP using the optimization setup of [63]. We found perplexity after 20 epochs to be an excellent indicator of perplexity at convergence; to maximize the number of experiments we could perform, we report the best validation perplexity after 20 epochs. We study two ways of implementing DFA: applying the feedback after every encoder block (*macro*) or after every layer in

Table 5: Best validation perplexity after 20 epochs of a Transformer trained on WikiText-103 (lower is better). The BP and DFA baselines share all hyper-parameters. In *Macro* the feedback is applied after every transformer layer, implying weight transport at the layer-scale, while in *Micro* the feedback is applied after every sub-layer. The learning rate of Adam without the learning rate scheduler is 5.10^{-5}. With the scheduler, the initial learning rate is 1.10^{-4} and it is multiplied by 0.2 when performance plateaus, with a patience of 1.
* *score after 22 epochs to let the learning rate scheduler take effect*

	DFA				**BP**	
	Baseline	+ Adam	+ $\beta_2 = 0.999$	+ LR schedule	Baseline	+ $\beta_2 = 0.999$
Macro	95.0	77.1	55.0	52.0	34.4	29.8
Micro	182	166	99.9	93.3*		

those blocks (*micro*). The *macro* setting enables weight transport at the block-scale, and some weight transport remain in the *micro* setting as well: to train the input embeddings layer, by backpropagation through the first encoder block, and for the values matrices in attention – see Appendix D for details.

Results Our results are summarized in Table 5. Hyper-parameters fine-tuned for BP did not fare well with DFA, but changes in the optimizer narrowed the gap between BP and DFA considerably. The learning rate schedule used on top of Adam [83] in [63] proved detrimental. Using Adam alone required reducing the learning rate between BP and DFA. Increasing β_2 from 0.98 [63] to 0.999 improved performance significantly. Finally, a simple scheduler that reduces the learning rate when the validation perplexity plateaus helped reducing it further. Considering that the perplexity of the shallow baseline is over 400, DFA is clearly able to train Transformers. However, our results are not on par with BP, especially in the *micro* setting. A substantial amount of work remains to make DFA competitive with BP, even more so in a minimal weight transport scenario. The large performance improvements brought by small changes in the optimizer indicate that intensive fine-tuning, common in publications introducing state-of-the-art results, could close the gap between BP and DFA.

4 Conclusion and outlooks

We conducted an extensive study demonstrating the ability of DFA to train modern architectures. We considered a broad selection of domains and tasks, with complex models featuring graph convolutions and attention. Our results on large networks like NeRF and Transformers are encouraging, suggesting that with further tuning, such leading architectures can be effectively trained with DFA. Future work on principled training with DFA–in particular regarding the influence of common practices and whether new procedures are required–will help close the gap with BP.

More broadly, we verified for the first time that learning under synaptic asymmetry is possible beyond fully-connected layers, and in tasks significantly more difficult than previously considered. This addresses a notable concern in biologically-plausible architectures. DFA still requires an implausible global feedback pathway; however, local training has already been demonstrated at scale. The next step towards biologically-compatible learning is a local method without weight transport.

While the tasks and architectures we have considered are not biologically inspired, they constitute a good benchmark for *behavioural realism* [20]. Any learning algorithm claiming to approximate the brain should reproduce its ability to solve complex and unseen task. Furthermore, even though the current implementation of mechanisms like attention is devoid of biological considerations, they represent broader concepts applicable to human brains [84]. Understanding how our brain learns is a gradual process, and future research could incorporate further realistic elements, like spiking neurons.

Finally, unlocking the backward pass in large architectures like Transformers is promising. More optimized implementation of DFA–built at a lower-level of existing ML libraries–could unlock significant speed-up. Leveraging the use of a single random projection as the cornerstone of training, dedicated accelerators may employ more exotic hardware architectures. This will open new possibilities in the asynchronous training of massive models.

Broader Impact

Of our survey This study is the first experimental validation of DFA as an effective training method in a wide range of challenging tasks and neural networks architectures. This significantly broadens the applications of DFA, and more generally brings new insight on training techniques alternative to back-propagation. From neural rendering and recommender systems, to natural language processing or geometric learning, each of these applications has its own potential impact. Our task selection process was motivated by current trends in deep learning, as well as by technically appealing mechanisms (graph convolutions, attention). A limit of our survey is that our–arguably biased–selection of tasks cannot be exhaustive. Our experiments required substantial cloud compute resources, with state-of-the-art GPU hardware. Nevertheless, as this study provides new perspectives for hardware accelerator technologies, it may favor the application of neural networks in fields previously inaccessible because of computational limits. Future research on DFA should continue to demonstrate its use in novel contexts of interest as they are discovered.

Of the considered applications Each of the applications considered in our study has a wide potential impact, consider for example the impact of textual bias in pretrained word embeddings [85]. We refer to [86] and references therein for a discussion of ethical concerns of AI applications.

Of DFA as a training method DFA enables parallelization of the backward pass and places a single operation at the center of the training process, opening the prospect of reducing the power consumption of training chips by an order of magnitude [31]. Not only is more efficient training a path to more environmentally responsible machine learning [87], but it may lower the barrier of entry, supporting equality and sustainable development goals. A significant downside of moving from BP to DFA is a far more limited understanding of how to train models and how the trained models behave. There is a clear empirical understanding of the impact of techniques such as batch normalization or skip connections on the performance of BP; new insights need to be obtained for DFA. BP also enjoys decades of works on topics like adversarial attacks, interpretability, and fairness. Much of this work has to be cross-checked for alternative training methods, something we encourage further research to consider as the next step towards safely and responsively scaling up DFA.

Of biologically motivated methods Finally, a key motivation for this study was to demonstrate that learning challenging tasks was possible without weight transport. Biologically motivated methods are a more foundational research direction, and as such the possible long-term impact of our findings is harder to estimate under this light. However, fundamental research of this kind is important to open new pathways for ML and neuroscience.

Acknowledgments and Disclosure of Funding

We thank Igor Carron and Laurent Daudet for the general guidance on the subject of this investigation and the insightful comments, as well as the larger LightOn team for their support. We also thank the anonymous reviewers for their useful comments.

Florent Krzakala acknowledges support by the French Agence Nationale de la Recherche under grants ANR17-CE23-0023-01 PAIL and ANR-19-P3IA-0001 PRAIRIE; additional funding is acknowledged from “Chaire de recherche sur les modèles et sciences des données”, Fondation CFM pour la Recherche.

References

[1] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[3] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1627–1635, 2017.

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

[5] Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, 10:94, 2016.

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

[7] Javier R Movellan. Contrastive hebbian learning in the continuous hopfield model. In *Connectionist models*, pages 10–17. Elsevier, 1991.

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

[9] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455, 2009.

[10] Yann Le Cun. Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer, 1986.

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

[12] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[13] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016.

[14] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International Conference on Machine Learning*, pages 583–593, 2019.

[15] Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, pages 4278–4287, 2017.

[16] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In *International Conference on Machine Learning*, pages 4839–4850, 2019.

[17] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[18] Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pages 3033–3045, 2019.

[19] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[20] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[21] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, pages 1–12, 2020.

[22] Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46, 2008.

[23] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. In *International Conference on Learning Representations*, 2020.

[24] Julien Launay, Iacopo Poli, and Florent Krzakala. Principled training of neural networks with direct feedback alignment. *arXiv preprint arXiv:1906.04554*, 2019.

[25] Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience*, 13:525, 2019.

[26] Qianli Liao, Joel Z Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[27] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[28] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. In *International Conference on Learning Representations*, 2019.

[29] Daniel Kunin, Aran Nayebi, Javier Sagastuy-Brena, Surya Ganguli, Jonathan M. Bloom, and Daniel L. K. Yamins. Two routes to scalable credit assignment without weight symmetry. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[30] Mohamed Akrout, Collin Wilson, Peter C Humphreys, Timothy Lillicrap, and Douglas Tweed. Using weight mirrors to improve feedback alignment. *arXiv preprint arXiv:1904.05391*, 2019.

[31] Julien Launay, Iacopo Poli, Kilian Müller, Igor Carron, Laurent Daudet, Florent Krzakala, and Sylvain Gigan. Light-in-the-loop: using a photonics co-processor for scalable training of neural networks, 2020.

[32] Charlotte Frenkel. *Bottom-Up and Top-Down Neuromorphic Processor Design: Unveiling Roads to Embedded Cognition*. PhD thesis, UCL-Université Catholique de Louvain, 2020.

[33] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017.

[34] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2367–2376, 2019.

[35] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[36] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019.

[37] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)*, 38(4):65, 2019.

[38] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, pages 1119–1130, 2019.

[39] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020.

[40] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[41] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230, 2013.

[42] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[43] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[44] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[45] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, ADKDD'17, New York, NY, USA, 2017. Association for Computing Machinery.

[46] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. Adaptive factorization network: Learning adaptive-order feature interactions. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[47] J Wesley Hines. A logarithmic neural network architecture for unbounded non-linear function approximation. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 2, pages 1245–1250. IEEE, 1996.

[48] Criteo. Kaggle contest dataset is now available for academic use! `http://labs.criteo.com/2014/09/kaggle-contest-dataset-now-available-academic-use/`, 2014. accessed on the 2020-05-20.

[49] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109, 2019.

[50] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[51] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, pages http–openreview, 2014.

[52] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[53] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[54] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[55] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[56] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[57] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

[58] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[59] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[60] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[61] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Machine Learning*, 2018.

[62] Matthias Fey. Just jump: Dynamic neighborhood aggregation in graph neural networks. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[64] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[65] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[66] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[67] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[68] David M Chan, Roshan Rao, Forrest Huang, and John F Canny. Gpu accelerated t-distributed stochastic neighbor embedding. *Journal of Parallel and Distributed Computing*, 131:1–13, 2019.

[69] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[70] Alec Radford, Karthik Narasimhan, Time Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018.

[71] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *ArXiv*, abs/1802.05751, 2018.

[72] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[73] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[74] Mohammad Shoeybi, Mostofa Ali Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *ArXiv*, abs/1909.08053, 2019.

[75] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[76] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[77] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[78] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3261–3275, 2019.

[79] The Common Crawl Team. Common Crawl. `https://commoncrawl.org`, 2020.

[80] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*. Association for Computational Linguistics, 2018.

[81] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *ArXiv*, abs/1609.07843, 2017.

[82] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[83] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[84] Grace W Lindsay. Attention in psychology, neuroscience, and machine learning. *Frontiers in Computational Neuroscience*, 14:29, 2020.

[85] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in neural information processing systems*, pages 4349–4357, 2016.

[86] Alexandra Luccioni and Yoshua Bengio. On the morality of artificial intelligence. *arXiv preprint arXiv:1912.11945*, 2019.

[87] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[88] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*, 2020.

[89] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[90] Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. Fixed encoder self-attention patterns in transformer-based machine translation. *arXiv preprint arXiv:2002.10260*, 2020.

[91] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Appendix

We first provide additional elements to corroborate our findings: alignment measurement (Section A), and shallow baselines (Section B). We then discuss the process of adapting the considered architectures for DFA (Section C), and the issue of weight transport in attention layers (Section D). We provide some supplementary results for NeRF (Section E), including details of performance on each scene of each datatset, and a discussion on possible mitigation of DFA shortcomings. Finally, we outline steps necessary for reproduction of this work (Section F).

A Alignment

Alignment measurement In feedback alignment methods, the forward weights learn to *align* with the random backward weights, making the delivered updates useful. This alignment can be quantified by measuring the cosine similarity between the gradient signal delivered by DFA $\mathbf{B}_i \delta \mathbf{a}_y$ and the gradient signal BP would have delivered $\mathbf{W}_{i+1}^T \delta \mathbf{a}_{i+1}$. For learning to occur and DFA to work as a training method, there must be alignment. This can be measured numerically [24]. Measuring alignments allows to check whether or not the layers are effectively being trained by DFA, regardless of performance metrics. We note that any alignment value superior to 0 signifies that learning is occuring. Values closer to 1 indicate a better match with BP, but small alignment values are sufficient to enable learning. We report values measured at the deepest DFA layer.

Recommender systems We measure alignment on the Criteo dataset, in the two architectures featuring non-conventional fully-connected layers: Deep & Cross and AFN. Alignment is measured after 15 epochs of training, and averaged over a random batch of 512 samples. Results are reported in table A.1. These alignment measurements indicate that learning is indeed occurring in the cross and logarithmic layers. High-variance of alignment in the cross layers is unique: it may be explained by the absence of non-linearity, and account for the difference in performance between BP and DFA on this architecture–which is higher than on the others.

Table A.1: Alignment cosine similarity (higher is better, standard deviation in parenthesis) of recommender systems as measured on the Criteo dataset. Learning occurs in both architectures, and high variance may explain the larger performance gap on Deep & Cross compared to other methods.

	Deep & Cross	**AFN**
Alignment	0.40 (0.91)	0.49 (0.08)

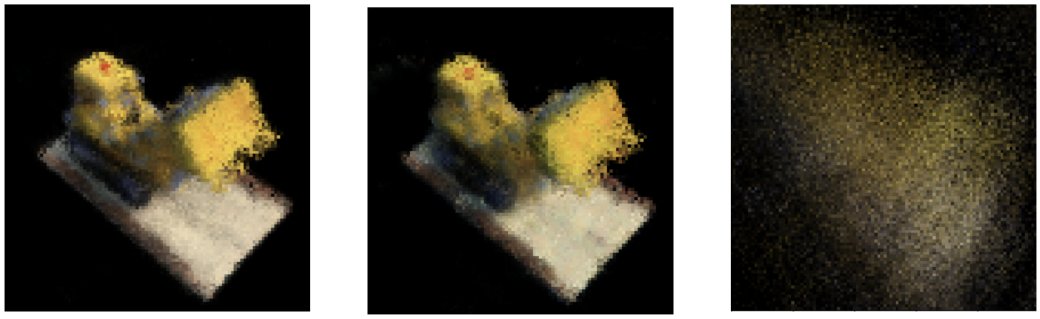
Graph convolutions We measure alignment on the Cora dataset, after 250 epochs of training, averaging values over every sample available–train, validation, and test split included. Results are reported in Table A.2. We observe high alignment values in all architectures, indicative that learning is indeed occuring. Slightly lower values in SplineConv and GATConv may be explained by the use of the Exponential Linear Unit (ELU) instead of the Rectified Linear Unit (ReLU) used as activation in other architectures.

Table A.2: Alignment cosine similarity (standard deviation in parenthesis) of various graph convolutions architectures as measured on the Cora dataset. These values corroborate that DFA successfully trains all architectures considered.

	ChebConv	**GraphConv**	**SplineConv**	**GATConv**	**DNAConv**
Alignment	0.87 (0.12)	0.77 (0.25)	0.56 (0.22)	0.63 (0.18)	0.92 (0.30)

B Shallow baselines

Shallow learning We compare DFA to BP, but also to shallow learning–where only the topmost layer is trained. While DFA may not reach the performance level of BP, it should still vastly



BP DFA SHALLOW

Figure A.1: Comparisons of Tiny-NeRF trained with BP, DFA, and a shallow approach. Shallow training is insufficient to learn scene geometry. Lego scene from the NeRF synthetic dataset.

outperform shallow learning: failure to do so would mean that the weight updates delivered by DFA are useless. On a simple task like MNIST, a shallow baseline may be as high as 90%. However, given the difficulty of the tasks we consider, the shallow baseline is here usually much lower.

NeRF Because NeRF models are expensive to train–up to 15 hours on a V100–we consider a simplified setup for the shallow baseline, NeRF-Tiny. This setup operates at half the full resolution of the training images available, runs for 5000 iterations only, and does away with view-dependant characteristics. Furthermore, the network is cut down to 3 layers of half the width of NeRF, and no coarse network is used to inform the sampling. We train this network on the Lego scene of the NeRF-Synthetic dataset, and compare results.

Figure A.1 presents renders generated by NeRF-Tiny trained with BP, DFA, and a shallow approach. While BP and DFA delivers similar renders, shallow training fails to reproduce even basic scene geometry, instead outputting a diffuse cloud of colors. This highlights that while DFA may not reach a level of performance on-par with BP on NeRF, it nonetheless delivers meaningful updates enabling the learning of complex features.

Recommender systems Because recommender systems require fine-tuning, we perform the same hyperparameter search for shallow learning than for DFA and BP. Results are detailed in Table A.3. Performance of shallow training is always well under BP and DFA–remember that *0.001-level* matter in recommender systems. In particular, in Deep & Cross, where there was the biggest gap between BP and DFA, the performance of the shallow method is extremely poor, well below the FM baseline. Finally, it is expected to see that DeepFM recovers more or less the performance of FM even with a shallow baseline.

Table A.3: Shallow baseline for recommender system models on the Criteo dataset. Performance is always well below BP and DFA, as expected.

	DeepFM	**Deep&Cross**	**AFN**
AUC	0.7920	0.7324	0.7859
Loss	0.4682	0.5010	0.4685

Graph convolutions We use the same hyperparameters as for DFA to produce the shallow baseline on graph datasets. Results are reported in Table A.4. Performance is always much worse than BP and DFA. GATConv recovers the best performance: random attention layers may still deliver useful features [88], as do random convolutions.

We also produce t-SNE visualizations of the hidden layer activations for a BP-trained network and a shallow-trained one (Figure A.2). t-SNE hyperparameters are identical between all three visualizations. In the shallow case, the hidden layer is not trained at all and remained in its random initialized state: in this case, t-SNE is unable to extract any structure.

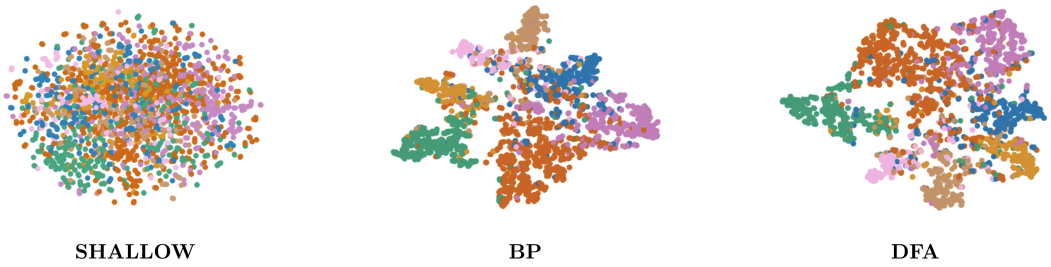


Figure A.2: t-SNE visualization of the hidden layer activations of a two-layer GraphConv trained on Cora with a shallow approach, BP, and DFA. The shallow approach does not train the hidden layer and t-SNE fail to extract any information from the randomly initialized layer. DFA and BP visualizations show identical level of separation between clusters.

Table A.4: Shallow baseline for GCNNs on Cora, CiteSeer, and PubMed [65]. Performance is always well below BP and DFA.

	ChebConv	**GraphConv**	**SplineConv**	**GATConv**	**DNAConv**
Cora	23.3	37.0	39.6	59.4	30.2
CiteSeer	27.4	33.8	30.1	49.8	24.0
PubMed	37.6	44.8	44.2	67.8	42.2

Transformers In the baseline setting (optimizer and hyper-parameters of [63]), a Transformer trained in the shallow regime yields a perplexity of 428 on WikiText-103. We do not consider other settings, as the cost of training a Transformer is high and we do not expect any meaningful improvements–as with NeRF above.

C Adapting architectures to DFA

In general, our implementation of DFA for all architecture follows the spirit of the original paper [19]. We introduce a random feedback $\mathbf{B}\delta\mathbf{a}_y$ after every non-linearity, and do not use any architecture-specific structure or operation to build the feedback. For graphs and transformers, we do share the backward random matrix for all nodes in a graph and for all tokens in a sentence. This is not only more computationally efficient, but also necessary for proper training: if the random matrix was different for each node/token, the graph/attention layers would receive incoherent feedbacks coming from different random matrices and alignment would be impossible. Finally, our global feedback matrix is initialized from $\mathcal{U}(-1, 1)$ and normalized with the square root of the output dimension of every layer.

NeRF We use an architecture identical to the one used in [39], but based on the effective code implementation rather than the description in the paper[1]. During our tests, we have found that lowering the learning rate to 1.10^{-4} rather than 5.10^{-4} works best with DFA.

Recommender systems For all training methods (BP, DFA, and shallow), we have conducted independent hyperparameter searches. We performed a grid search over the learning rate, from 1.10^{-4} to 1.10^{-3} in 1.10^{-4} steps, as well as over the dropout probability, from 0.1 to 0.5 in 0.1 steps (where applicable). On DeepFM, this search leads to reduce the learning rate from 3.10^{-4} with BP to 5.10^{-5} with DFA, but to keep the 0.5 dropout rate. On Deep & Cross, we reduce learning rate from 2.10^{-4} to 5.10^{-5}, with no dropout in both cases. In AFN, we reduce dropout from 4.10^{-4} to 3.10^{-4} and dropout from 0.3 to 0.

Graph convolutions We manually test for a few hyperparameters configuration on the Cora dataset, focusing on learning rate, weight decay, and dropout. We do not consider architectural changes, such

[1] `https://github.com/bmild/nerf/issues/11`

as changing the number of filters or of attention heads. For ChebConv and GraphConv, we reduce weight decay to 1.10^{-4} instead of 5.10^{-4}, and set the dropout rate to 0 and 0.1 respectively, instead of 0.5 with BP. For SplineConv, we find that no change in the hyperparameters are necessary. For GATConv, we reduce weight decay to 1.10^{-4} instead of 5.10^{-4} and reduce dedicated dropout layer to 0.1 instead of 0.6 but keep the 0.6 dropout rate within the GAT layer. Finally, on DNAConv we disable weight decay entirely, instead of an original value of 5.10^{-4}, double the learning rate from 5.10^{-3} to 1.10^{-2}, and disable dropout entirely. In all cases, we share the backward random matrix across all nodes in a graph.

Transformers The model hyper-parameters were fixed across all of our experiments, except for the number of attention heads in one case, that we will precise below, and dropout. We tested several values of dropout probability between 0 and 0.5, but found the original value of 0.1 to perform best. We manually tested a number of optimizers, optimizer parameters and attention mechanisms. We tested four combinations of optimizers and schedulers : Adam with the scheduler used in [63], Adam alone, RAdam [89] alone, and Adam with a scheduler that reduces the learning rate when the validation perplexity plateaus. We found it necessary to reduce the initial learning rate of Adam from 1.10^{-4} to 5.10^{-5}, although it could be set back to 1.10^{-4} with a scheduler. We tried two values of β_2: 0.98 and 0.999. We also tried to change β_1 and observed some small differences that were not significant enough for the main text. Finally, we tried three attention mechanisms in addition to the standard multihead scaled dot-product attention: the dense and random (learnable) Synthesizers of [88], as well as the fixed attention patterns of [90]. The latter needed to be adapted to language modelling to prevent attending to future tokens, which led us to reduced the number of attention heads to 4. The backward random matrix is always shared across all tokens and batches.

D Weight transport and attention

We consider an attention layer operating on input $\mathbf{x}$. The queries, keys, and values are respectively $\mathbf{q} = \mathbf{x}\mathbf{W}_Q; \mathbf{k} = \mathbf{x}\mathbf{W}_K; \mathbf{v} = \mathbf{x}\mathbf{W}_V$, and d_k is the dimension of the queries and keys. The layer performs:

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{k}^T}{\sqrt{d_k}}\right)\mathbf{v} \tag{4}$$

When using DFA on attention, we deliver the random feedback to the top of the layer. Accordingly, to obtain updates to $\mathbf{W}_Q, \mathbf{W}_K,$ and $\mathbf{W}_V$ we still to have to backpropagate through the attention mechanism itself. This involves weight transport on $\mathbf{W}_V$, sacrificing some biological realism for simplicity. Overall weight transport between layers still does not occur, and updating the layers in parallel remains possible.

Beside using FA or DFA within the attention layer, alternative mechanisms like the synthesizer [88]–which uses random attention in place of the query and key system–or fixed attention [90] can remove the need for weight transport. Implementing these mechanisms in DFA-trained Transformers, or other attention-powered architectures, will require further research.

E Supplementary NeRF results

Quantitative results We report per-scene scores for each dataset in Table A.5. BP values are taken from [39]. On three scenes of the synthetic datasets, NeRF-DFA even outperforms past state-of-the-art methods trained with BP. Note that Neural Volumes (NV) is not applicable to forward-facing view synthesis–as is required in LLFF-Real–and thus no results are reported.

Qualitative results We report sample renders from the NeRF-Synthetic dataset (Figure A.3) and the LLFF-Real dataset (Figure A.3), for every scene available. However, we recommend readers to consult the supplementary video[2] to make better sense of characteristics like multi-view consistency and view-dependent effects (most visible on the LLFF-Real Room scene).

[2] `https://www.youtube.com/watch?v=sinch7013LY`

Possible future directions Despite retranscribing scene geometry in a multi-view consistent way, NeRF produces renders of a lower quality when trained with DFA instead of BP. In particular, it struggles to transcribe small-scale details, resulting in "blurry" renders. Moreover, it displays high-frequency artefacts: not in the scene geometry, but in individual pixels taking values very distant from their neighborhood. Interestingly, this noise phenomenon is unique to NeRF-DFA: it is not observed on NeRF-BP with similar PSNR values (achieved during training) or on other methods with similar or lower PSNR. This leads us to hypothesize this is an aspect unique to DFA, possibly due to the alignment process. Indeed, DFA creates a bias on the weights, by encouraging them to be "aligned" with an arbitrary values dependant on the random matrix used. It is possible this could introduce random noise in the final renders–though we leave a more principled experiment to future research.

To attempt to alleviate this issue, we first consider NeRF-Dual. In NeRF-Dual, we average the pixel-wise prediction between the fine and coarse network, to attempt to remove some of the noise. To do so, we first still use the coarse network to create a probability distribution for the hierarchical sampling. Then, we evaluate again both the coarse and fine networks at the locations informed by this probability distribution. Compared to vanilla NeRF, this requires an extra batch of evaluation of the coarse network for all rays–rougly speaking, this increases inference time by 30-50% depending on the coarse network architecture considered. We note that this is not applied during training, so that training times remain identical.

Figure A.3 and Figure A.4 showcase comparisons between NeRF and NeRF-Dual trained with DFA on all scenes. When viewed at high resolution–such as in our supplementary video–the NeRF-Dual renders are more pleasing, especially for the full scenes. They remove most of the high-frequency noise, leading to smoother renders. However, this averaging process further blurs small-scale details in the render. This is especially visible in the NeRF-Synthetic dataset, on scenes like Ficus. Furthermore, NeRF-Dual introduces novel artefacts in the Mic and Ship scenes, with areas improperly colored with a violet tint. The cause for these artefacts is unknown, but they show that NeRF-Dual is far from a silver bullet. The PSNR is also minimally increased, by less than 0.5 per scene. Nevertheless, this shows some promise in possibilities to allievate the shortcomings of NeRF-DFA. It is possible that changes to the overall rendering process, or the use of classic image processing techniques, may help enhance the NeRF-DFA images.

Table A.5: Per-scene PSNR for NeRF DFA and BP against other state-of-the-art methods on the Nerf-Synthetic and LLFF-Real. DFA performance is fairly homogeneous across each dataset and in line with the differences in other methods.

	NV	**SRN**	**LLFF**	**NeRF**	
	BP	BP	BP	BP	DFA
NeRF-Synthetic	**26.05**	**22.26**	**24.88**	**31.01**	**25.41**
Chair	28.33	26.96	28.72	33.00	28.74
Drums	22.58	17.18	21.13	25.01	22.15
Ficus	24.79	20.73	21.79	30.13	25.61
Hotdog	30.71	26.81	31.41	36.18	28.03
Lego	26.08	20.85	24.54	32.54	24.93
Materials	24.22	18.09	20.72	29.62	25.15
Mic	27.78	26.85	27.48	32.91	25.43
Ship	23.93	20.60	23.22	28.65	23.25
LLFF-Real		**22.84**	**24.13**	**26.50**	**20.77**
Room		27.29	28.42	32.70	24.20
Fern		21.37	22.95	25.17	21.82
Leaves		18.24	19.52	20.92	16.50
Fortress		26.63	29.40	31.16	25.16
Orchids		17.37	18.52	20.36	16.73
Flower		26.63	25.46	27.40	21.55
T-Rex		22.87	24.15	26.80	19.43
Horns		24.33	24.70	27.45	20.75

Finally, we also experimented with increasing the capacity of the fine network, by widening its layers to 512 neurons. We call this architecture NeRF-XL. However, we have not succeeded in getting PSNR values higher than with vanilla NeRF on DFA. In particular, the training process becomes much more cumbersome, as multi-GPU parallelism is needed to fit the model. It is possible that higher network capacity may help learning both the task at hand and to align simultaneously, but further work is required.

F Reproducibility

Hardware used All main experiments require at most a single NVIDIA V100 GPU with 16GB of memory to reproduce. Alignment measurement on large architectures (NeRF and Transformers) require a second identical GPU to keep a copy of the network to evaluate BP gradients.

We estimate that a total of around 10,000 GPU-hours on V100s were necessary for this paper. Accordingly, we estimate the cloud-computing carbon impact of this paper to be of 1700 $kgCO_2eq$[3].

However, without hyperparameter searches, our results can be reproduced with less than 500 GPU-hours on V100s, with most of that budget going to NeRF and Transformers.

Implementation We use the shared random matrix trick from [24] to reduce memory use in DFA and enable its scaling to large networks. We use PyTorch [91] for all experiments. For reference implementation of the methods considered, we relied on various sources. Our NeRF implementation is based on the PyTorch implementation by Krishna Murthy[4], with modifications to allow for proper test and validation, as well as DFA and multi-GPU support. For recommender systems, we use the `torchfm` package[5]. Finally, we use PyTorch Geometric [64] for all graph operations. Our Transformer implementation is our own. Our code is available as supplementary material.

NeRF We provide training, testing, and rendering code along with the configurations used to obtain our results. An example to reproduce our results is given in the supplementary code repository. Given the computing cost associated with training a NeRF, we also provide our trained models.

Recommender systems We provide bash scripts to reproduce the results in Table 2 and A.3, with the results of our hyperparameter search. We provide code to reproduce the results in Table A.1.

Graph convolutions We provide the code to reproduce all of our results. Note that the t-SNE results are not exactly reproducible, as the CUDA implementation used is non-deterministic.

Transformers We provide bash scripts to reproduce Table 5 and the shallow results.

[3] `https://mlco2.github.io/impact#compute`

[4] `https://github.com/krrish94/nerf-pytorch`

[5] `https://github.com/rixwew/pytorch-fm`

Figure A.3: Sample renders for every scene of the NeRF-Synthetic dataset, for NeRF and NeRF-Dual trained with DFA.

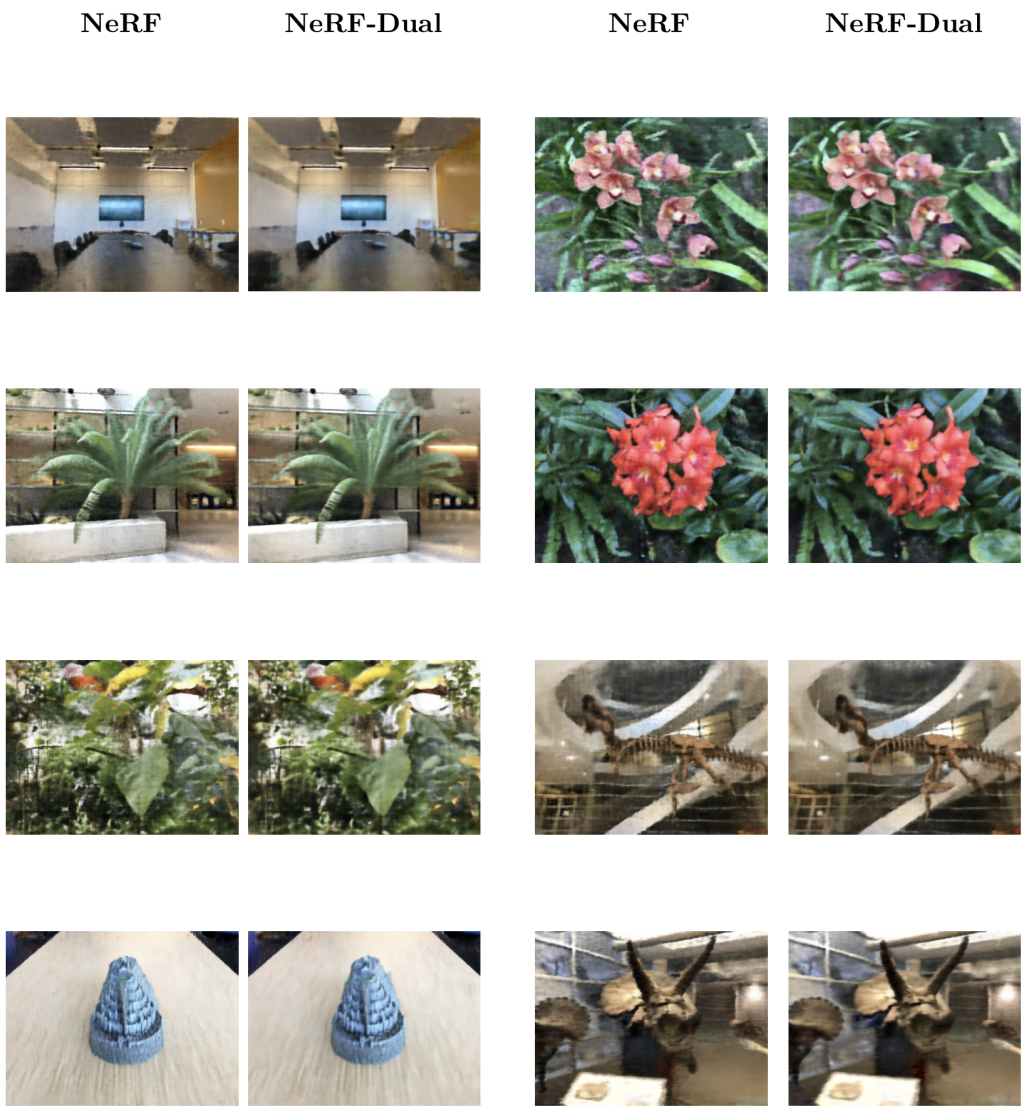


Figure A.4: Sample renders for every scene of the LLFF-Real dataset, for NeRF and NeRF-Dual trained with DFA.