A Survey on Investigating Direct Feedback Alignment as an Alternative to Backpropagation

Can Gozpinar
Department of Computer Science
Koc University
cgozpinar18@ku.edu.tr

Abstract

Backpropagation (BP) has been the standard workhorse for training neural networks. Due to its long history, it is very well understood and many design choices regarding the neural networks are shaped according to it. Despite its success, it has problems such as its sequential nature and biological implausibility. Direct Feedback Alignment (DFA) which is a recently developed algorithm promises benefits over the BP method such as parallelizability and bio-plausibility by (almost) locally approximating the gradients. In spite of its promises, it still fails to compete with the performance of BP. This survey aims to investigate the current state of the issues that hinder the scaling of DFA to the level of BP. These topics are categorized into two main sections, namely the scalability issues section and the architectural issues section. Finally, this survey concludes by sharing the results of a novel experiment which was designed to examine the effects of different types of normalization and activation functions on the DFA training.

1 Introduction

Neural networks are an activate research area which has shown promising results in various tasks such as computer vision and natural language processing. Their design is influenced by the understanding of brain but they are not strictly bounded by it. Essentially their learning mechanisms are as follows. The neural activity is determined by the synaptic weights, which determines the output of the neural network. The generated output then determines the networks error. It is this error that neural network will use to update its parameter to improve its performance. How this update will be performed depends on the learning algorithm that will be used during the training process Lillicrap et al. (2020). Currently, the backpropagation algorithm (BP) is the most popular choice of learning algorithm when neural networks are being trained. As a result neural network architectures heavily rely on the backpropagation algorithm. Backpropagation provides the gradients by back-propagating the error signal starting from the output layer. It relies on the chain rule of calculus to recursively compute these gradients. Due to its recursive implementation, the "error computations start in the final layer and flow backwards, leading to the notion of errors 'backpropagating' through the network" Lillicrap et al. (2020). Then these propagated signals are used to perform weight updates. Stochastic Gradient Descent and ADAM are some of the popular algorithms which utilize these gradients to perform weight updates in the layers of the neural networks. Despite being widely used, the backpropagation algorithm has certain downsides. To begin with, the backpropagation algorithm is sequential. It needs gradients to be computed in order starting from the output layer which makes it limited in parallelizability. Moreover, it is not a biologically plausible algorithm Lillicrap et al. (2020). It relies on the transpose of the forward weights (W_i^T) while performing backwards pass which would require synaptic symmetry to be present between the forward pass and the backward pass. This symmetry requirement is not plausible in the biological brain structures. This requirement for having the same weight on two different connections, namely the forward pass and the backward pass, is

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

referred to as the 'weight transport' problem Lillicrap et al. (2020). Recently, it has been shown that this symmetry is not an absolute necessity. Instead, weights for the backward pass can be fixed random weights (B_i instead of W_i^T) which lead to the development of various bio-plausible learning algorithms including the Direct Feedback Alignment (DFA). Motivated by these pitfalls of backpropagation, this survey paper investigates the Direct Feedback Alignment (DFA) method as a promising alternative to the backpropagation algorithm. DFA is an extension to the Feedback Alignment (FA) method. It furthers the FA method by directly propagating the error signals to the corresponding layers instead of sequentially 'propagating' them (like BP and FA) Nøkland (2016). DFA algorithm offers a systematic way for calculating (almost) locally computed gradients Nøkland (2016). During training, the error signal obtained from the output of the last layer is propagated directly to every trainable layer. At every layer this directly propagated error will be multiplied with a randomly initialized feed backward weight matrix(B_i) to synthetically approximate the true BP gradients to perform weight updates Nøkland (2016). These backward weight matrices (B_i's) will be unique to every trainable layer and they will be kept the same throughout the training process. Unlike backpropagation, these backward weights are not the transpose of the feed forward weights (W_i^T). As the training progresses, the forward weights align themselves with their corresponding backward weights (B_i's), and then converge to a solution by improving the generalization errorRefinetti et al. (2020). One impressive observation in DFA is that the network figures out how to use fixed random backward weights by adapting (aligning forward weights with the pseudoinverse of the backward weights) its forward weights (W_i's) in order to make the error decrease. This process is described as "network learning to learn" by Nøkland (2016). DFA's reliance to the (almost) local gradients promises many benefits over BP. Unlike backpropagation, DFA is a synthetic gradient method. It can perform weight updates using local gradients which makes it a parallelizable method. Moreover, it is a biologically plausible method since the error signal is almost local and no symmetry between the forward and the backward weights are required. Also, DFA training can be improved by utilizing photonic processors which are specialized for DFA. Despite its promises, DFA currently fails to overthrow BP. I believe that the aforementioned prior work suggests that DFA is a strong candidate for becoming an alternative to the backpropagation algorithm. In order for that to actually happen, the problems that withhold DFA from achieving it should be explored from various perspectives such as the state of the current solutions and the underlying dynamics of the issues. To this end, we examine the scalability issues 2.1, architectural issues 2.2, and the issues stemming from the lack of established standards 2.3. Scalability issues section explores the memory and computational problems that DFA faces upon scaling to harder tasks. Next, the architectural issues section goes through the problems that DFA faces regarding the choice of the model architectures. For example, DFA's are notoriously known for their failures when scaling to deep convolutional neural network architectures. This section is essential due to the fact that there are many types of neural network architectures which are specialized for use with different data types. If DFA is to substitute BP, it should be able to scale to these different architectural choices. In addition to these, the lack of best practices section touches upon the current state of the established standards and practices being used in the research of DFA and how it compares to BP. Finally, inspired by the findings of the lack of best practices section, the results of my personal experiments which investigates the effects of different types of normalization and activation functions on DFA is shared 3. These experiments are a result of believing that a deep understanding of how different normalization and activation functions impact the training being the underlying reason for the current unrivalled success of the BP algorithm. As is suggested by the findings of the lack of best practices section of this survey, a similar understanding for DFA is currently missing. These final experiments conducted hopes to aid this problem.

2 Issues that Leave DFA Under the Shadow of BP

Even though DFA promises improvements on the backpropagation algorithm it has some obstacles to overcome before being able to become a viable alternative to the widely used backpropagation algorithm. Most of these hinderings can be grouped into the issues concerning memory and computational requirements2.1, problems scaling up to modern architectures such as convolutional neural networks2.2, and lack of an established understanding of training mechanics2.3.

2.1 Scalability Issues

Recently, the decline of Moores Law combined with the end of Dennard scaling have hinted on the saturation of the capabilities of computing architectures in the near future. Despite this downward trend, we are still observing increased accumulation of data. With this increased abundance of structured and unstructured data comes the interest in processing these data to extract the underlying information. Deep Neural Network architectures is one of the most promising ways to process these data. If Direct Feedback Alignment is to become an alternative to backpropagation, it should be able to work under these foreseen hardware limitations. To this end, we will cover some prominent issues of DFA regarding its memory and computational requirements, and go over some proposed solutions which hint on promising directions. Also, DFA method should be able to scale to training different types of specialized neural network architectures. We will examine how DFA compares to BP regarding its applicability to different models.

2.1.1 Memory and Computation Issues

In Direct Feedback Alignment algorithm, the gradients are locally computed by the product of three scalar values. This helps solving the weight transport problem that backpropagation algorithm was suffering from. Even though, passing global error signals directly from the last layer to the intermediate layers instead of backpropagating them in a sequential way looks like an advantage for DFA at first sight, this is not the case for scaling. To understand this better let us go through an example Crafton et al. (2019). First, consider a simple task of MNIST benchmark. In MNIST there are 10 classes. This would mean that intermediate layers of a Fully Connected Neural Network model would have its random backward weight matrices with of its dimensions as 10 since the error signal will be corresponding to 10 classification scores at the final layers output. Now assume that we are using the same model for a more advanced dataset such as ImageNet. In ImageNet there are 1000 classes. As a result this networks random backward weight matrices will have one of its dimension as 1000. As a result we see that by scaling from MNIST dataset to ImageNet dataset, the size of the backward weight matrices have grown $1000/10 = 100$ per layer. Furthermore, the model architecture would be scaled (deeper and wider networks) to match the difficulty of the task which would mean that increased layers would amplify this 100 times increase in the size of the backward matrices. This results in remarkable increase in the memory and computation requirements of the DFA algorithm. This phenomena hinders Direct Feedback Alignment's chance of competing against the backpropagation algorithm since backpropagation scales better than DFA does. This is due to the fact that in backpropagation the gradients are propagated sequentially using the chain rule from calculus which uses transposes of the forward weight matrices of the corresponding layers. Because mostly the forward weight dimensions get smaller towards the classification layer and for bigger datasets such as ImageNet the classification layer size is the biggest, the backpropagation algorithm uses backwards weights of smaller size while calculating gradients for most of the layers. Now we will look at some ideas proposed to improve the unfavourable memory and computational effects of scaling using DFA.

In the paper Crafton et al. (2019), the authors investigated the effects of using sparse feedback matrices with DFA. They defined feedback weight matrix as sparse if "the values in the feedback matrix that model the connections between the errors and the hidden neurons are mostly zero" Crafton et al. (2019). As a result, as the sparsity of the feedback weight matrix increases, the amount of information passed down to that corresponding layer decreases with which the amount of computation and memory (with specialized hardware architectures) required decreases. Motivated by this, the paper proposes Sparse Direct Feedback Alignment (SDFA) which "compute its error using a fewer number of error signals" due to sparse connections in the backward weight matrices Crafton et al. (2019). They investigated how the sparsity and the rank of the matrices used as backward weights impacted the performance of the models. Their empirical findings show that the matrices can be made sparse as long as the rank of the matrix is near full. Upon structuring matrices under these conditions, the gains in computational requirements can be justified by the negligible drop in the accuracy of the trained model Crafton et al. (2019). They even explored the extreme case of 99.9% sparsity where only one of the many error signals is transported to the layer. They named this Single connection SDFA (SSDFA) Crafton et al. (2019).

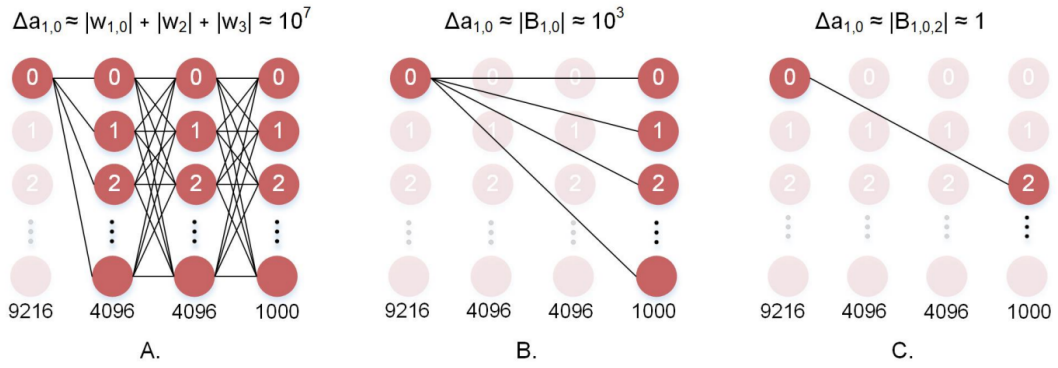

Figure 1: A)Backpropagation: all of the error signals are propagated backwards sequentially. B)DFA: local gradient is computed by $e \cdot B$ where local error is dependent on the all of the global error signal. C)SSDFA: local gradient is computed using $e \cdot B$ where B is an extremely sparse matrix. As a result, local error signal depends on only one of the global error signals. Crafton et al. (2019)

Let a_i be the activation at layer i and δ_l error at hidden layer l, then for DFA

$$\delta a_2 = B_2 \cdot e \odot f'(a_2) \tag{1}$$

Here for regular backward weight matrices we observe that B_2 is a matrix and e is a vector. Whereas for an extremely sparse matrix (B_2) such as the one defined in SSDFA, B_2 becomes a vector. Due to this, regular $B_2 \cdot e$ becomes vector vector dot product from matrix vector multiplication. Note that this change promises improvements in the scalability of DFA by leveraging the neuromorphic hardware Crafton et al. (2019). Furthermore, one of the most important findings of the paper is that the rank of the matrix matters more than the sparsity of the matrix used as the backward weight.This relationship is empirically observed in Figure 2.

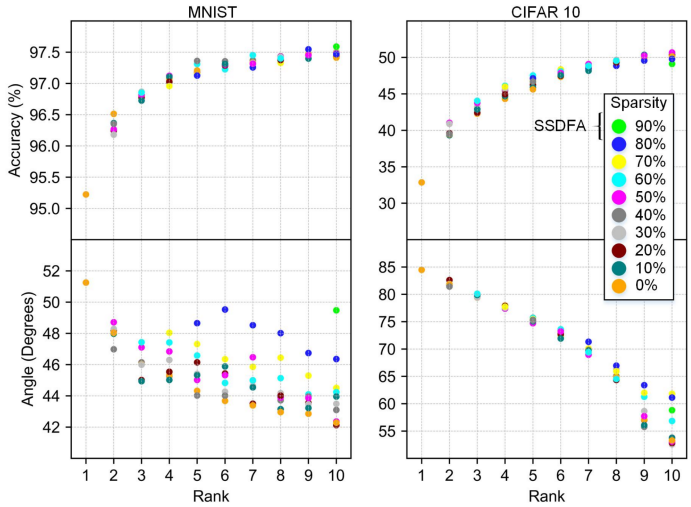

Figure 2: Plots show how the sparsity and the rank of the matrices used for backward weights impacts the accuracy of the model along with the angle between the random backward weights (B) and the the corresponding true backward weights (W^T). Crafton et al. (2019)

Binary Direct Feedback Alignment (BDFA) Han & Yoo (2019) is another paper which tackles the computational scalability issues of DFA algorithm by putting constraints on the feedback weight matrices. In BDFA, the backward weight matrices (B_i) consist of values of $\{+1, -1\}$. These values allow the backward weight matrices to carry only the sign value information (*i.e.* $B_i = sign(B_i)$) of its elements but not its magnitude. Since $\{+1, -1\}$ can be represented as a binary, each element of

B can be stored as a single bit in the memory. Before this binarization, the regular DFA algorithm's feedback weight matrices were holding 32 bit (or 64 bit) floating point representations. Because of this representation difference, implementing BDFA over regular DFA can result in 96.9% reduction in the memory requirements to store the B_i's with the same shape Han & Yoo (2019). By nature, the DFA algorithm can be parallelized but its parallelization is bounded by the limitations of the bandwidth of the hardware it runs on. When the backward weight matrices are large and there are many layers to calculate gradients locally, then there could be a throughput bottleneck issue. This can be caused by many memory inefficient backward weights being transferred to their corresponding layers in a parallelized manner. This is where the memory efficiency gains of binarization introduced in BDFA can help alleviate the bottleneck problem. The Figures 3 and 4 show empirically that BDFA can be used to train models using reduced memory requirements given that learning rate is tuned correctly.

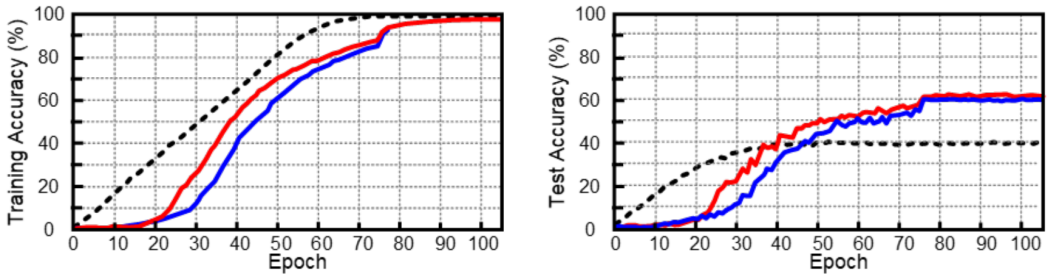

Figure 3: Three lines correspond to same model being trained with different algorithms using small learning rate (black dashed line: backpropagation, red line: BDFA, blue line: DFA). Han & Yoo (2019)

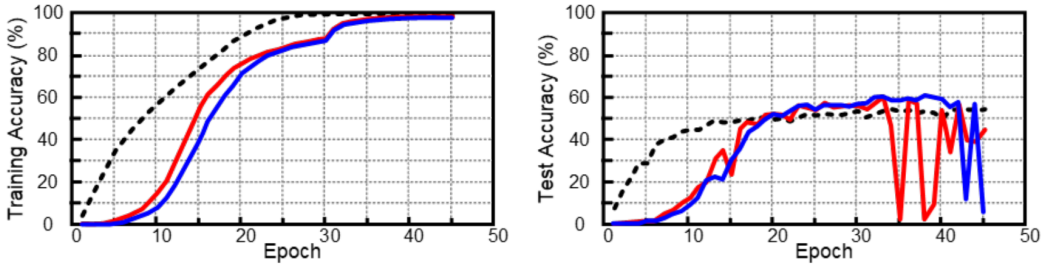

Figure 4: Three lines correspond to same model being trained with different algorithms using large learning rate (black dashed line: backpropagation, red line: BDFA, blue line: DFA). Han & Yoo (2019)

Another improvement on the memory requirement is made regarding the observation that increasing the output size of the network effects memory requirements of the DFA algorithm much more than BP. This is due to the fact that for each backward weight B_i, one of its dimensions has to match the size of the error signal e. Therefore, an increase of k times on the outputs size would result every B_i to scale its size by k times. As was addressed before in our survey this causes problems for scaling DFA to complicated benchmarks such as ImageNet with 1000 classes. The paper Launay et al. (2019) proposes to reduce the negative consequences of this issue by storing one unified random matrix to be used for the backward weights, B_i, instead of generating and storing every backward weight separately. In this approach, we generate and store one random matrix (B) of size $e \times l_{max}$ where e is the size of the error vector and l_{max} is the size of the largest layer. B_i's for layers are obtained by taking fixed slices from this large matrix of size $e \times l_{max}$. Note that it is important to take slices the same at every iteration to keep the backward weights unchanged. B_i's can be obtained from B using $B_{1:l_i,:}$ where l_i is the output of the i^{th} layer. Authors support the legitimacy of slicing from one unified matrix B by the following argument. In the theorem that justify the convergence of DFA, there is no assumption made on the independence of the backward weights (B_i's) Launay et al.

(2019). By the adaptation of this implementation, the memory requirements for DFA can be reduced while scaling to larger problems which could help DFA to scale like BP. Figure 5 gives an example of how much memory is gained by adapting the suggested "unified feedback matrix" implementation Launay et al. (2019).

	Memory complexity	**VGG-16 [GB]**
naive	$\sum_{i=1}^{N-1} l_i e$	55
unified feedback matrix	$l_{\max} e$	13

Figure 5: Example of memory benefits of using unified feedback matrix. Launay et al. (2019)

2.2 Architectural Issues

Neural networks are trained using different types of data. Some of it are structured and some of it are unstructured. As a result, there are different neural network architectures which are tailored by design to exploit the underlying structure of the data it is trained on. For instance, convolutional neural networks are designed to exploit the structure of images, recurrent neural networks are designed to exploit the temporal relations in the data, and graph neural networks are designed to exploit the structures in data which can be better explained by the properties of graphs. In order for Direct Feedback Alignment to become a viable alternative to the backpropagation algorithm, it should be scalable to different neural network architectures.

2.2.1 DFA and Convolutional Neural Network Architectures

Deep Neural Networks are widely adopted to computer vision tasks which is one of the most prominent research areas regarding Neural Networks. They owe most of their success in computer vision tasks to convolutional neural networks (CNNs). Consequently it is important that DFA can train CNN models. Unfortunately, directly applying DFA algorithm to train ConvNets has failed drastically in comparison to using the backpropagation algorithm (BP). This is one of the most important reasons why DFA is not currently regarded as an alternative to BP.

To aid this issue, the paper Crafton et al. (2019) proposed a solution inspired by transfer learning. They proposed training fully connected layers using Direct Feedback Alignment whereas using pretrained convolutional layers which were trained using backpropagation. They keep the weights of the convolutional layers frozen at all times and only train the fully connected layers using DFA. This way they have leveraged the benefits of DFA training such as parallelization on the fully connnected layers. On the other hand, they have bounded themselves to training another architecture using backpropagation to transfer the convolutional layers which was a costly sequential process. Furthermore, it was empirically shown in 6 that this transfer learning approach to DFA can be used to successfully train CNN models such as VGG and AlexNet on even hard benchmarks such as ImageNet Crafton et al. (2019).

Benchmark	BP	DFA
MNIST	99.1	99.1
CIFAR10	77.1	77.8
CIFAR100	48.2	49.0
ImageNet (Alexnet)	49.0	48.8
ImageNet (VGG)	65.8	65.3

Figure 6: Test Accuracy in $\%$. Both *BP* and *DFA* columns used the same pre-trained convolutional backbone but, the final fully connected layers were trained using either BP or DFA. Crafton et al. (2019)

Their work benefits from parallelizing the training of fully connected layers using DFA at the cost of separately training convolutional layers using backpropagation.

Han & Yoo (2019) improves upon the aforementioned work by finding a way to integrate the training of convolutional layers using BP, into the training of fully connected layers using DFA. This approach eliminates the costly need for two separate training processes. This is achieved by training the whole

CNN model from scratch by using DFA on the fully connected layers, and propagating the earliest fully connected layer's locally computed error signal to convolutional layers using backpropagation from there on Han & Yoo (2019). This way the fully connected layers are trained in parallel using DFA, and the earlier convolutional layers are trained sequentially using the backpropagation algorithm. The process is illustrated in Figure 7.

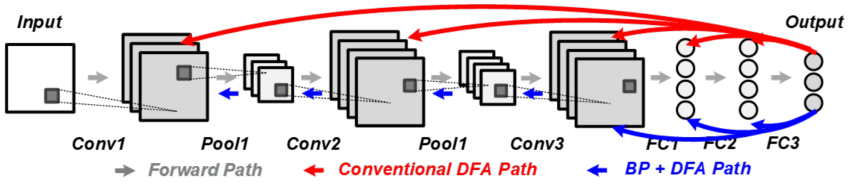

Figure 7: The proposed DFA and BP based CNN training. Han & Yoo (2019)

In addition to extending DFA to CNN training, this approach has theoretical benefits regarding the training of the convolutional layers. To understand this we will think of convolutional layers as the feature extractors and the fully connected layers as the classifiers which decide on a class based on the features extracted by the convolutional layers. Upon intuitively dividing model into separete modules we gain insight into their training dynamics. If the whole model were to be trained using backpropagation, then initially during training the fully connected layers will be bad at classifying. Since the backpropagation algorithm will propagate the errors coming from the fully connected layers to the convolutional layers, at first the convolutional layers will try to adapt to these signals which are in accordance to the initial bad classification layers. As the training continues, the fully connected layers will adapt and become better classifiers. But by the time this happens, the convolutional layers would have adapted to the initially bad classifier layers since the error signals were propagated accordingly. Whereas, the adaptation on the fully connected layers changes the error signal propagated to the convolutional layers which would probably not correspond to the earlier signals. This change in the propagated error signal from the earliest fully connected layer to the convolutional layers is referred to as "constant domain shifting" Han & Yoo (2019). As a result, convolutional layers training can be unstable until the fully connected layers are adapted. It turns out that DFA can improve this constant domain shifting problem that backpropagation faces during the training of CNNs. Because DFA uses random backward weight matrices which are kept unchanged thorough the training, the error signals propagated backwards from the earliest fully connected layer to the convolutional layers would not change given that the earliest fully connected layer is trained to a degree. The training of the earliest fully connected network can be achieved quite easily since DFA allows for local computation of error signals by imposing no data dependency between fully connected layers like backpropagation does. By leveraging this, initially the earliest fully connected layer will be trained directly. Afterwards, the parallelized training of the fully connected layers using DFA, and the sequential training of the convolutional layers using backpropagation will begin. This prevents convolutional layers from getting "confused by the propagated error's domain" shifting Han & Yoo (2019). In the authors experiments, the proposed approach has yielded better performance than solely using backpropagation on the CIFAR-10 and CIFAR-100 benchmarks. On the other hand, the proposed approach was slower to train which can be explained by DFA using randomly initialized backward weights which takes additional time for the forward and the backward weights to align, and also by DFA not being able to leverage higher learning rates like backpropagation algorithm to speed up training Han & Yoo (2019).

Despite its improvements, the aforementioned approach still misses on the change of parallelization in the convolutional layers. A solution to this would call for a way of adapting DFA to the convolutional layers too. Unfortunately, adaptation of DFA to deep convolutional layers have been shown to not work well and suffer great accuracy losses in comparison. On the contrary, for easier benchmarks such as MNIST and CIFAR-10, it was shown that shallow models could be trained successfully using only DFA. This contrast stems from the fact that DFA propagates errors directly. Therefore, as the layers get deeper(i.e. the distance increases) the accuracy of the DFA reduces which makes it infeasible for training deep CNN models. To alleviate this problem, Han et al. (2020) proposes to reduce the distance that DFA is used to propagate errors in the deep convolutional layers. They do this by modularization. The neural network is divided into modules in which the DFA is used to calculate local gradients. Between the modules the error is propagated using backpropagation. The way that

network is divided into modules is a hyperparameter which should be tuned to make distances inside the modules small enough for DFA to work successfully. Also, this hyperparameter can be used to control the trade-off between the degree of parallelization of the training and the accuracy of the computed gradients. For example, if there were a few modules with very long distances, then the Direct Feedback Alignment's accuracy would suffer, whereas there could be more parallel training since backpropagation would be done only between the very few modules. On the contrary, if the hyperparameter was chosen so that there were a lot of modules with very short distances, then the accuracy of the calculated gradients would increase due to using backpropagation most of the time and DFA being used over short distances. But the level of parallelization would decrease due to sequentially using backpropagation for the most of the network. Figure 8 illustrates the way this algorithm works over a network divided to three modules. Despite the good properties of their methods such as control over the parallelization and accuracy trade-off, the algorithm was shown to fail to surpass the accuracy of the backpropagation training Han et al. (2020).

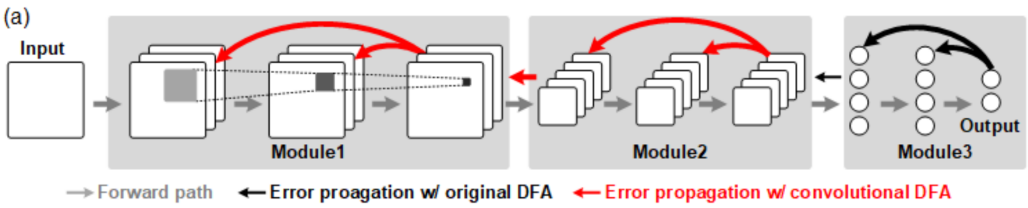

Figure 8: DFA with Network Modularization. Han et al. (2020)

Hybrid Direct Feedback Alignment (HDFA) was proposed to close the performance gap between the aforementioned modularized approach and BP training Han et al. (2020). The proposed approach still combines both DFA and backpropagation but in a different way while still offering control over the degree of parallelization. The prior methods used backpropagation for predetermined parts of the network while the remainder was trained using DFA. In contrast to this HDFA method does not differentiate between the layers of the network and treats it as a whole. In this approach, a random parameter is sampled at every iteration which determines whether backpropagation or DFA is used for the corresponding iteration by comparing the random sample against a hyperparameter threshold value. This threshold controls how frequently BP or DFA is used to compute gradients. As a result it controls the accuracy vs parallelization trade-off of the training. Furthermore, the HDFA optimizer keeps track of momentum values for the BP gradients and the DFA gradients. But they add a novel tweak to regular momentum approaches by using a combination of the BP momentum with the DFA momentum to update the parameters of the model. This combination is only done for the DFA updates and not for the BP updates. The authors observed that "the gradient occasionally generated by BP accelerates the convergence of the DFA algorithm and it finally helps the DFA algorithm to have higher accuracy" Han et al. (2020). This pattern can be understood by remembering that the gradients calculated by BP are the true gradient values, whereas, the gradients generated by DFA are approximations to the BP gradients. Because of this, combining the DFA's approximated gradients with the true gradients of BP can help guide the approximated gradients towards the true gradients. This results in faster convergence in comparsion to earlier DFA approaches and increased accuracy. Figure 9 summarizes the optimization process using HDFA approach.

HDFA is shown to train deep CNN models with accuracies that match of backpropagation's. But unfortunately it still relies on the backpropagation algorithm to close the accuracy gap which also introduces the weaknesses of the backpropagation algorithm accordingly.

We have covered some approaches proposed to deal with the challenges of adapting DFA to training CNN models. But it is important to understand the underlying dynamics of CNN training using DFA. This way, the underlying problems that hinder learning in the convolutional layers can be understood better which could lead to the development of better solutions which are based in theory. Also, the training dynamics of the backpropagation algorithm is very well understood which is not the case for DFA. An advancement in this direction could serve to bring DFA closer to BP regarding this concern too. In their work Lillicrap et al. (2016) show that learning happens in Feedback Alignment since, the forward weight matrices (W_i's) align themselves with their corresponding random feedback weight matrices (B_i's) during the training process. This phenomena is referred to as weight alignment (WA). Eventually WA leads to the gradients approximated by DFA to align with

Algorithm 1 Custom Optimizer of the Hybrid Direct Feedback Alignment

Input: Learning rate η, momentum parameter α, mix parameter γ, BP ratio p
Input: Initial parameter $\boldsymbol{\theta}$, Initial BP velocity $\boldsymbol{v}^{BP}$, Initial DFA velocity $\boldsymbol{v}^{DFA}$
Input: Objective function $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ given input $\boldsymbol{x}$ and parameters $\boldsymbol{\theta}$
  Initialize network parameters $\boldsymbol{\theta}$
  Initialize momentum for BP $\boldsymbol{v}^{BP} \leftarrow 0$ and momentum for DFA $\boldsymbol{v}^{DFA} \leftarrow 0$
  while $\boldsymbol{\theta}$ not converged **do**
    Sample a minibatch of m examples from the training set $\{x^{(1)}, \cdots, x^{(m)}\}$ with corresponding targets $y^{(i)}$
    Set $rand = random(0,\ 1)$
    if $rand \leq p$ **then**
      $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}}^{BP} \sum_i L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$ ▷ Compute gradient through BP
      $\boldsymbol{v}^{BP} \leftarrow \boldsymbol{v}^{BP} + \alpha \boldsymbol{g}$ ▷ Compute momentum update
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \boldsymbol{v}^{BP}$ ▷ Update parameter
    else
      $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}}^{DFA} \sum_i L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$ ▷ Compute gradient through DFA in Section 3.2.2
      $\boldsymbol{v}^{DFA} \leftarrow \boldsymbol{v}^{DFA} + \alpha \boldsymbol{g}$ ▷ Compute momentum update
      $\boldsymbol{u} \leftarrow \gamma \boldsymbol{v}^{DFA} + (1 - \gamma) \boldsymbol{v}^{BP}$ ▷ Mix DFA momentum with BP momentum
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \boldsymbol{u}$ ▷ Update parameter
    end if
  end while
  return $\boldsymbol{\theta}$

Figure 9: HDFA approach to optimization. Han et al. (2020)

the true gradients (i.e. BP gradients). This alignment of gradients is named gradient alignment (GA). GA is the reason that learning occurs. Refinetti et al. (2020) extends their findings to explain the low performance of training CNNs using DFA. They explain this phenomena by investigating the filters of the convolutional layers. Let H_l be the filter for a convolutional layer which can be expressed as a large fully-connected layer with weights expressed as a block Toeplitz matrix ($\phi(H_l)$). Then they show that for WA to be satisfied $\phi(G_l) \propto F_l F_{l-1}^T$ should hold Refinetti et al. (2020) which is not the case. "Therefore, the WA mechanism suggests a simple explanation for why GA doesn't occur in vanilla CNNs" Refinetti et al. (2020). Similarly, Launay et al. (2019) explains this phenomena by referring to the fact that convolutional layers have fewer degrees of freedom in comparison to fully connected layers. This freedom is necessary for allowing the layers to both align themselves with the feedback weights (WA), and learn the task at hand. Due to the sparse nature of convolutional layer filters they lack this freedom. This creates "bottleneck in the network" Launay et al. (2019) and hinders learning.

2.2.2 DFA and Modern State-of-the-art Architectures

Beyond the computer vision tasks, deep neural networks have been scaled to many other different domains. DFA's suitability for these domains should also be considered if DFA is to substitute backpropagation one day. In their work Launay et al. (2020) investigated how well the DFA algorithm scaled to various other state-of-the-art architectures. Neural view synthesis is one of these domains. In this task, the goal is to train a model which is capable of synthesising novel 3D scene renders given some available renders of the scene to the model. Authors pick Neural Radience Fields (NeRF) as the state-of-the-art architecture for that field. To explore the scalability of DFA to this field, the authors trained two NeRF models. One was trained using DFA, and the other was trained using BP. A qualitative comparison of their results show that DFA training can "maintain multi-view consistency and exhibit no artefact" Launay et al. (2020). On the contrary, it yields yields lower render definition. Their results show that DFA is suitable for scaling to neural view synthesis tasks using state-of-the-art architectures even out of the box. Also, it is important to note that the successfully trained architecture heavily relies on the use of fully connected layers which are known to work well with Direct Feedback Alignment. The results obtained for neural view synthesis here implies that for different domains as long as the architecture heavily relies on fully connected layers like NeRF does, it can possibly be trained using DFA.

Moreover, some complex data such as social networks and the interaction of cars on the road are structured using graphs to capture the underlying relationship well. These type of structured data calls for geometric learning with specialized neural network architectures. Graph neural networks

(GNNs) are commonly used for processing these types of data. In their work Launay et al. (2020) experimented with graph convolutional neural network(GCNN) architecture. This is an extension of the GNNs to using convolutional layers. Their findings indicate that DFA can train shallow GCNN models with accuracies comparable to that of BP's. On the other hand, for the deeper models their accuracies drop drastically and cannot compete with backpropagation. These results are in accordance with the findings of Han et al. (2020) for training CNN architectures which also has convolutional layers like GCNNs. This result indicates that DFA's inability to training deep convolutional neural networks also hinders its ability to train GCNNs. It also hints on the need for better understanding the learning dynamics of DFA and establishing best practices for training convolutional layers.

2.3 Lack of Best Practices

Another reason for DFA to lack behind the backpropagation algorithm is the lack of established standards and best practices. The backpropagation algorithm has been around for a very long time and it has been the de facto standard for training neural networks. As a result of this, best practices for using backpropagation is well established in the community. Sadly, this is not the case for DFA since it is a relatively new approach. This is one of the reasons why DFA cannot compete with BP at the moment. This difference in the availability of best practices leads to ill suited practices which hinders the advancement and adaptation of systems that use DFA. Motivated by this, the authors Launay et al. (2019) established some best practices for using DFA. For handling the memory requirements of DFA while scaling it, the paper sets the implementation of "unified feedback matrix"2.1.1 as a standard best practice. Moreover, if done right the normalization of feedback matrices (B_i's) is known to improve the performance (accuracy and convergence speed) of the DFA algorithm. Due to this, it is important to have best practices for normalizing feedback weights. To this end, there are many claims.Han & Yoo (2019)'s is one of them which proposes that initializing feedback weight matrices (B_i's) as a multiplication of the transposed weight in a couple of adjacent layers improves both the convergence and the accuracy. In mathematical notation it can be expressed as $B_i = W_{L,L-1}...W_{i+2,i+1}W_{i+1,i}$, where W's are the feedforward weights. But Launay et al. (2019) rejects this as a best practice due to that approach to normalization depending on the forward weights which conflicts the fixing weight transport problem of DFA. Instead they suggest scaling the feedback weights with respect to their corresponding dimensions. By still adopting the unified feedback matrix (B) their normalization algorithm is as follows.

$$B = \frac{U}{\sqrt{l_{max}e}} \text{ and } B_i = \sqrt{\frac{l_{max}}{l_i}} B_{1:l_i,1:e} \text{ where } U_{ij} \sim \mathcal{N}(0,1) \tag{2}$$

Refer to figure 10 for the empirical results. The authors use alignment similarity score to evaluate the methods performance. They choose to rely on this score instead of metrics such as model accuracies/losses because they believe the alignment similarity reflects the true underlying impact that their tests have on the DFA training process. Also it is important to note that the authors used $U_{ij} \sim \mathcal{N}(0,1)$ and did not explore other initialization methods than $\mathcal{N}(0,1)$. In the paper Sanfiz & Akrout (2021), the impact of different initialization methods were explored and Xaview Uniform initialization of W and B were found to be the best. Additionally, their findings suggest that a mismatch between the initializations of forward weights and backward weights negatively effects the convergence of the training as the depth of the model increases Sanfiz & Akrout (2021). By combining the aforementioned results we suggest using Launay et al. (2019)'s scaling of the feedback weights while also avoiding the mismatch of initializations as suggested by Sanfiz & Akrout (2021).

Batch normalization (BN) is also a popular regularization method commonly utilized in training neural networks with BP. Unlike BP training, Launay et al. (2019) find that it does not benefit the training and result in worse alignment similarity scores. As a result they abandon the use of BN as a best practice for DFA. Dropout is commonly used to prevent overfitting of the model. The alignment similarity scores in figure 10 indicate that its presence worsens the alignment similarity but yet it prevents overfitting. It is interesting that dropouts presence as regularization improves DFA since is not prone to overfitting due to it using approximated gradients instead of the true gradients like BP. Launay et al. (2019) proposes the use of Dropout with a slightly smaller dropout rate ($\sim\ 0.1$) as a best practice. Additionally, activation functions that are commonly used with neural networks turn out to not work very well with DFA. As a best practice, lReLU with slope of -0.5 is found to be working well10. Therefore Launay et al. (2019) sets lReLU with -0.5 slope as the best practice activation function while using DFA. But more importantly, they show that activation

	Test acc.	Train acc.	Alignment cosine similarity		
			1st layer	2nd layer	3rd layer
norm	46.46% (0.35)	86.62% (0.31)	0.79 (0.08)	0.87 (0.10)	0.64 (0.07)
no norm	43.46% (0.27)	96.43% (0.22)	0.64 (0.08)	0.69 (0.08)	0.65 (0.06)
norm+BN	45.73% (0.53)	84.52% (0.30)	0.62 (0.21)	0.43 (0.13)	0.56 (0.14)
norm+$\|x\|$	50.21% (0.39)	62.17% (0.35)	0.79 (0.02)	0.64 (0.04)	0.19 (0.02)
norm+ReLU	40.24% (2.16)	41.68% (0.40)	0.38 (0.33)	0.37 (0.33)	0.06 (0.07)
norm+lReLU(-0.5)	**52.94**% (0.38)	68.67% (1.73)	0.84 (0.02)	0.72 (0.02)	0.22 (0.02)
norm+lReLU(0.01)	37.88% (4.34)	41.46% (0.69)	0.42 (0.35)	0.40 (0.34)	0.07 (0.07)
norm+DO(0.5)	47.90% (0.17)	54.23% (0.15)	0.72 (0.07)	0.73 (0.08)	0.43 (0.06)
norm+DO(0.1)	49.50% (0.34)	69.97% (0.18)	0.79 (0.07)	0.83 (0.05)	0.54 (0.03)

Figure 10: Empirical results of various practices usign DFA. Launay et al. (2019)

functions which are commonly adapted in neural network architectures today does not expand to training with DFA. Therefore, there is "a need to rethink activation functions designed for BP in the context of training with DFA" Launay et al. (2019). Choice of optimization algorithm is another decision which can critically impact the overall performance of the training. These optimizers use the gradients obtained by methods such as DFA and BP to update the model parameters. SGD and ADAM are among the most popular choices for optimization algorithms while training with BP. In their work Sanfiz & Akrout (2021) ADAM was found to outperform SGD while using DFA. This result is explained by referring to ADAM's ability to handle noisy gradients better than SGD. When using DFA, the gradients are approximated. In other words the gradients are noisy. As a result ADAM's performance benefits over SGD is justified. This observation is also empirically supported by performing experiments where norm of the weight ratios of forward and backward weights are tracked (i.e. $||\frac{W_i}{B_i}||$). The results show that earlier layers forward weights (W_i's) were getting much larger than the backward weights (B_i's) when SGD was used. This explosion of the forward weights is connected to error signals being directly propagated to the earlier layers from the last layer and as a result "sidestepping the small norms of the gradients computed using chain rule" Sanfiz & Akrout (2021). Inspired by these findings, we set using ADAM optimizer as a best practice to use with DFA.

3 My Experiments - Investigation of the Effects of Regularization and Activation Functions on DFA Training

Motivated by the findings of the previous section 2.3, in this section we investigate the effects of various regularization and activation functions on the DFA training.

3.1 The Approach

I implemented two fully connected neural networks which are architecture wise identical but one is capable of BP training and the other is capable of DFA training. I equipped these model with various regularization functions and activation functions and observed their impacts by comparing their respective behaviours. For the Direct Feedback Alignment model, I replaced the default fully connected linear layers of torch with my custom direct feedback alignment capable linear layers. I extended *torch.nn.Linear* to create a custom direct feedback alignment linear layer. We have in_features, out_features,output_dim, bias, device, and dtype as input parameters to our custom layer. In_features is the dimension of the input of the layer. Out_features is the dimension of the output of the layer. Bias is a boolean value to indicate if there would be a bias term for the linear layer. Device is the computing device used, cuda or cpu. output_dim is the output dimension of the entire fully connected neural network. All of these parameters except output_dim is inherited from torch.nn.Linear. Output_dim is needed to create the random matrix that will be used as backward weights in direct feedback algorithm. Since these matrices map the loss to the custom linear layer during the backward pass, they need to have access to the shape of the error signal which is the output dimension of the entire neural network. Using this output dimension we initialize random weight matrices with the shape (output_dim, in_features). This matrix is initialized with the custom direct

feedback alignment linear layer and does not get updated throughout the training process. It is vital that these random matrices stay constant for the direct feedback alignment algorithm to work. The second change to a classic linear layer is in the backward pass. I use backward hooks to implement the direct feedback alignment algorithm with the existing torch framework. During back-propagation, our backward hook intercepts the gradients flowing into the custom direct feedback alignment layer. We calculate the new gradient that will be used by multiplying the error signal with the random matrix that we initialized. This maps the error signal to our backward weights. We pass this mapping as the new gradient by returning it via the backward hook. This approach of using backward hooks ensure compatibility with other functions and layers of torch. Furthemore, this also means that we are simulating the direct feedback algorithm. The main advantage of the direct feedback alignment algorithm over the backpropagation algorithm is that, it is parallelizable. Our implementation is sequential since torch waits for the previous backward hook to finish. However, this does not pose a problem for our goal. We aim to analyze the effects of activation functions and normalizations on the direct feedback alignment. We do not need to parallize our implementation to achieve this. Our training time takes a bit longer than the traditional backpropagation algorithm as expected. This is due to our implementation not being parallel and doing additional calculations in the backward hook compared to backpropagation. I implemented several regularization methods. These include dropout, BatchNorm, LayerNorm, Weight decay, L2 regularization, L1 regularization, learning rate scheduling. I added Tensorboard support to log and visualize my experiments. The experiments are easily configured using the *config.yaml* file I have constructed 11. For the experimentation on the MNIST dataset, I have implemented a custom Neural Network. Our model consists of three linear layers with activation functions in between them. By default ReLU activation function is used. In the experimental results you will find certain cases where these activations are swapped with different activation functions but the architecture stays the same. Depending on the regularizations one chooses to use, it allows for dropout layers after the first and the second linear layers. For DFA training, these linear layers are used from our customized implementation that is built on top of the *torch.nn.Linear* implementation. In these layers we leveraged the backward hooks of torch to interfere with the default implementation of autograd into Pytorch and make it perform according to DFA. Using Tensorboard, I gained insight into the training of the model. I tracked many aspects of the model such as layer weights, gradients of the layer weights, loss values and accuracies. An exhaustive list of the parameters I have tracked can be found on the figures 12 and 13. Regularizations I have implemented and experimented with are as follows:

- Dropout
- BatchNorm
- LayerNorm
- Weight Decay
- L2 Regularization
- L1 Regularization
- Learning Rate Scheduling (ConstLR)
- Early Stopping

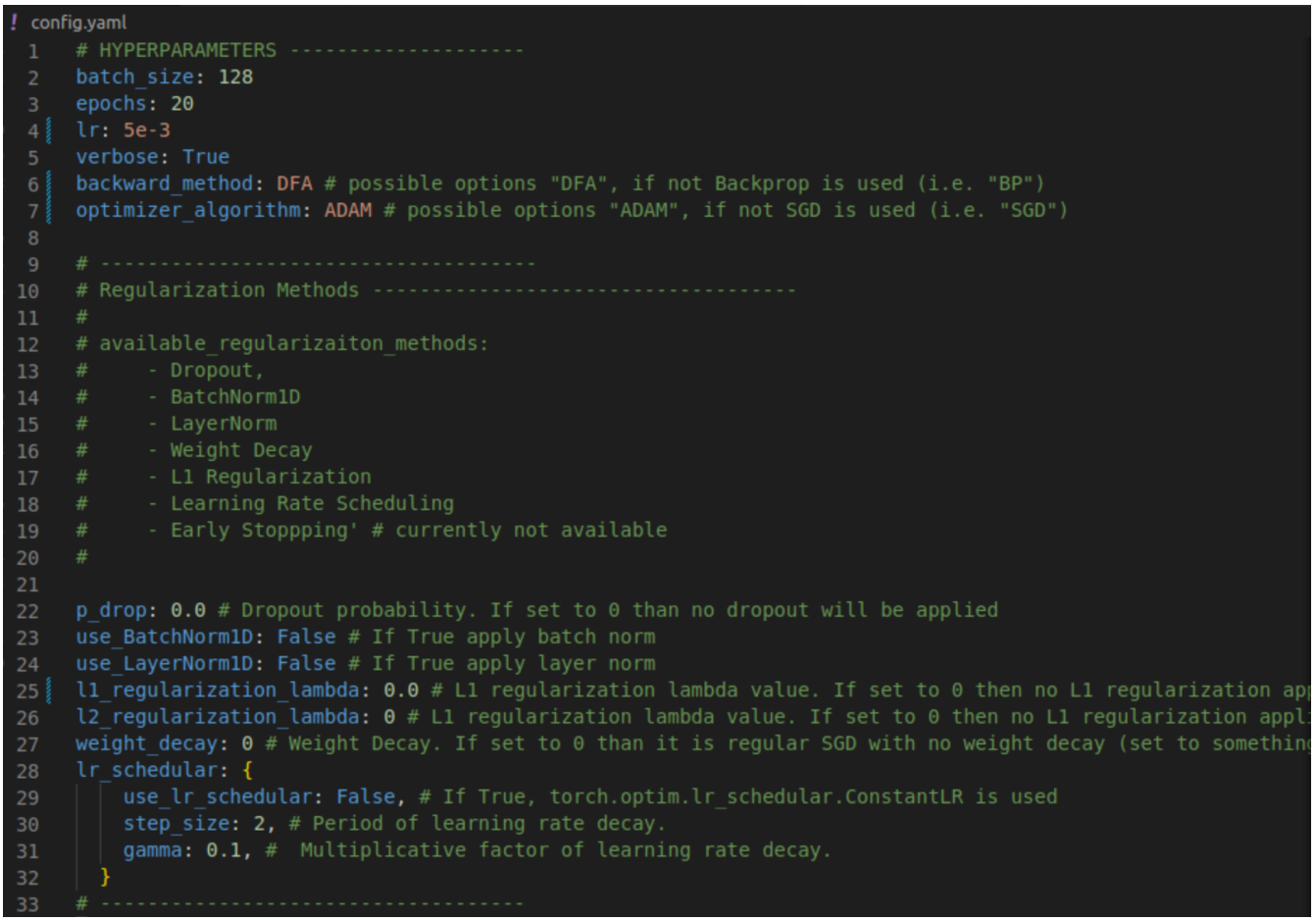

Figure 11: *config.yaml* and the default hyperparameters used as baseline (config.yaml)

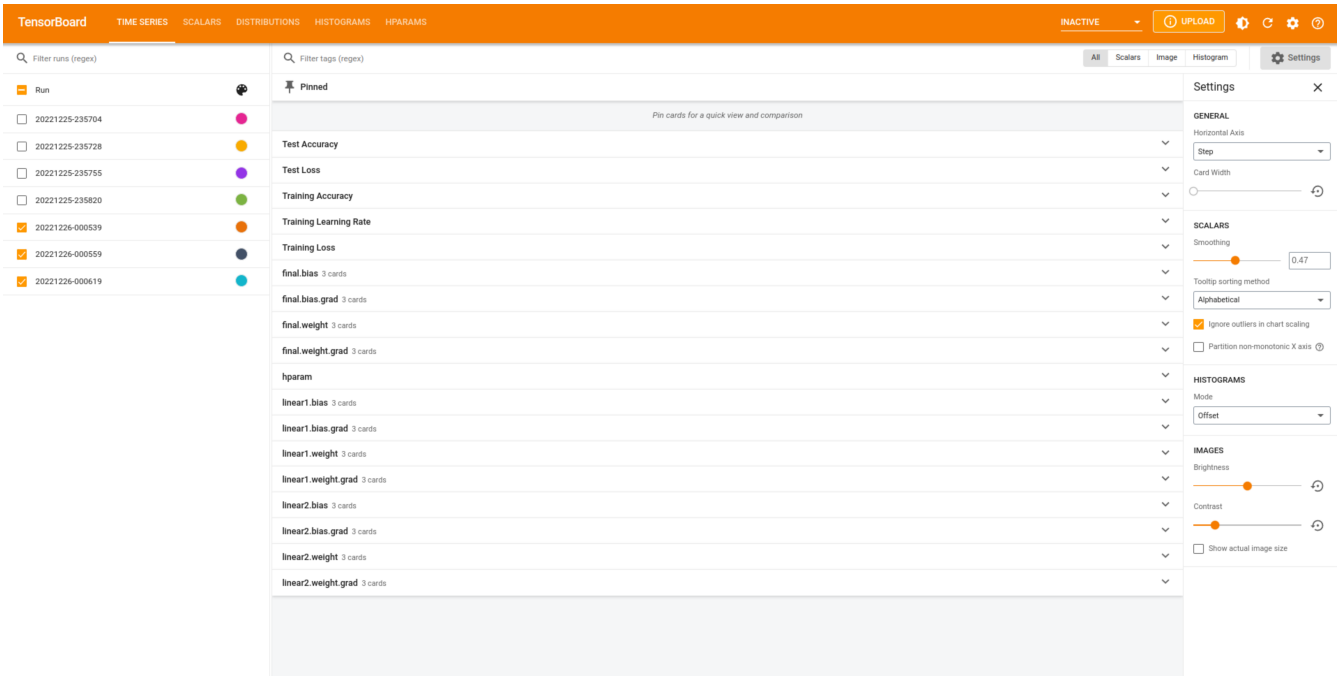

Figure 12: Tensorboard metrics that are kept track of

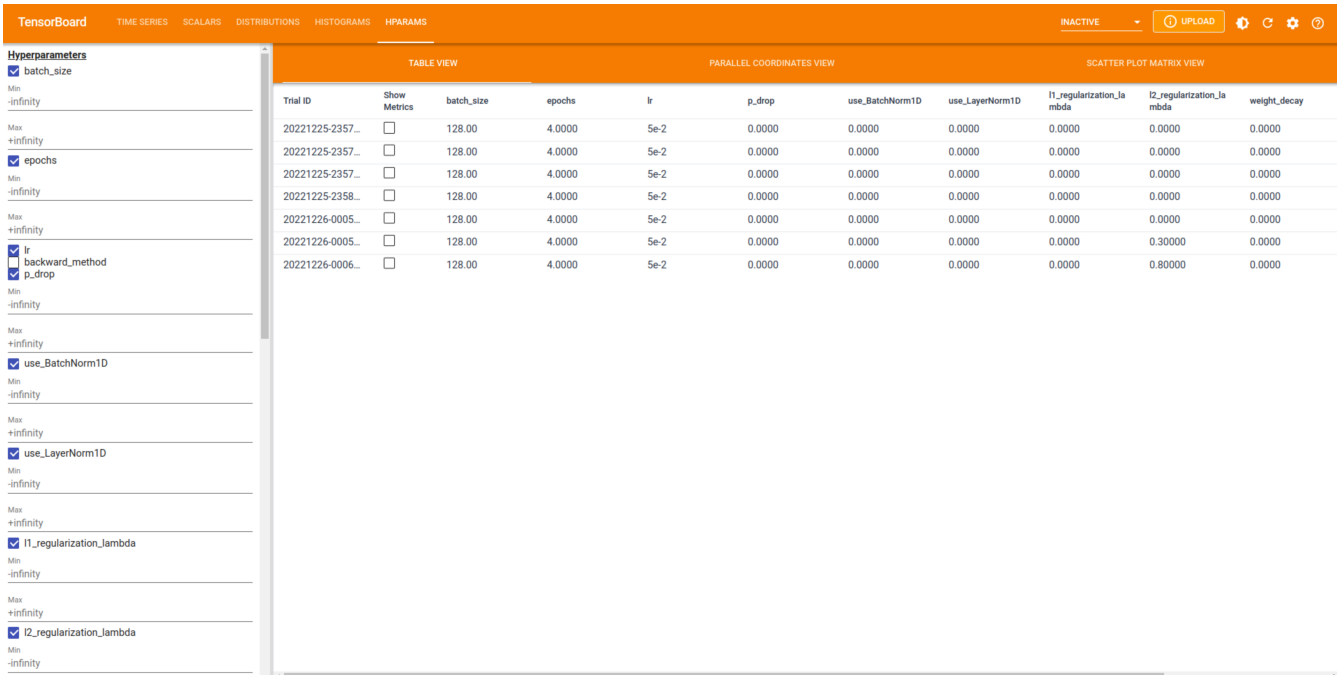

Figure 13: Tensorboard hyperparameter logging

3.2 Experimental Results

3.2.1 Tolerance to High Learning Rate

When the training is proceeded for longer epochs using too high learning rates, the update directions can overshoot the minimas and the loss can increase. This phenomena for backpropagation requires the tuning of the learning rate as a fix. During my experiments this phenomena was observed and one such instance is shown in the figure 15. The blue line shows the BP training and the red line shows the BP training. The results indicate that unlike backpropagation, when trained using DFA, the aforementioned overshooting due to high learning rate might not propose the same problems. In other words, the DFA training appears to be more tolerant to high learning rates, and more resiliant to this overshooting problem. In the figure 15 red line appears to be converging and not improving after certain epochs but still the loss does not start to increase like it did for the BP training. This result might indicate that training dynamics for DFA are more resilient to the choice of the learning rate but it is important to note that this does not mean that no tuning has to be done for the DFA training. Still learning rate will have an impact on the duration of the convergence and which minima the algorithm converges to. In my experiments it resulted with better performance on the test data for the DFA training. This implies that DFA might be better suited for training for long epochs using high learning rates.

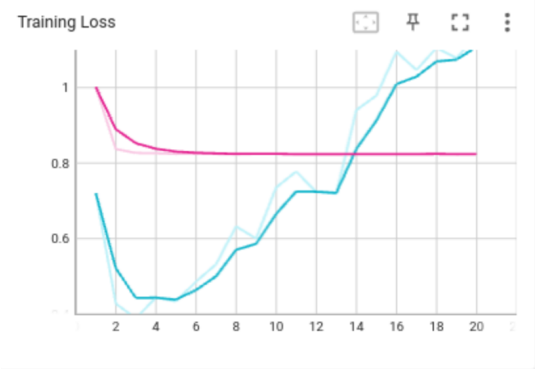

Figure 14: Training Loss curves for, long epoch training with high learning rate. Red line: DFA curve, blue line: BP curve.

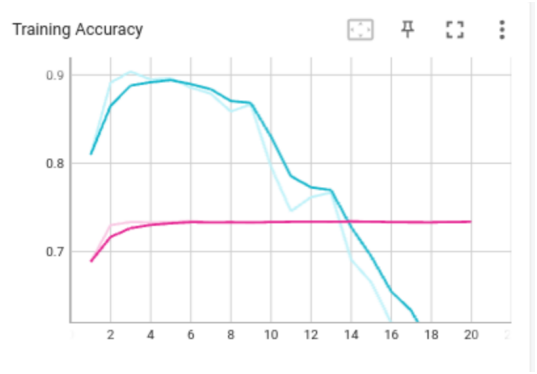

Figure 15: Training Accuracy curves for, long epoch training with high learning rate. Red line: DFA curve, blue line: BP curve.

3.2.2 SGD Training

Stochastic Gradient Descent (SGD) is a very popular learning algorithm used for Neural Network training. My experiments indicate that SGD might not be a very good option for DFA training as it is for BP training. To understand this further let us go over the results 16 I have shared below for one of

my experiments. It shows that even though BP training using SGD converges quickly to very small loss values it is not the case for DFA. There is a considerable difference between the BP training and the DFA training done with the same hyperparameters. Even though the losses appear to decrease lower more quickly with higher learning rates, it is nowhere near the success of the BP training. This hints on the incompetence of SGD when it comes to DFA training. I suspect that this might be due to BP using true gradients during the update step whereas, DFA approximating these gradients almost locally which results in noisy updates. As a result BP training can leverage SGD more efficiently whereas, DFA requires a means of handling this noise that is present in the gradients it computes.

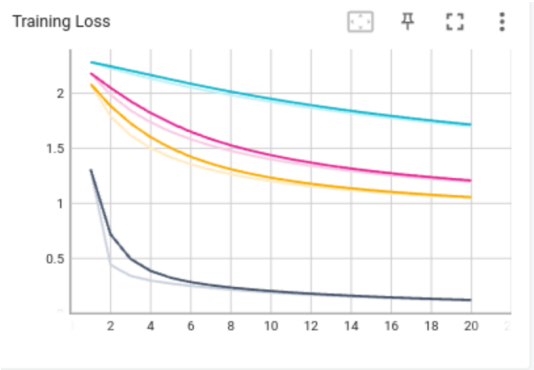

Figure 16: Training Loss curves for Gray curve: BP+SGD+small_lr, blue curve: DFA+SGD+small_lr, red curve: DFA+SGD+mid_lr, orange curve: DFA+SGD+high_lr

3.2.3 ADAM Training

To alleviate the problems that SGD faced in the previous section 3.2.2 we turn to ADAM optimizer. The results of my experiments indicate that ADAM performs better than SGD for DFA training. ADAM results in faster convergence to lower training losses. One such case is shared in the figure 17. I believe that this difference stems from the fact that ADAM optimizer accumulates a "momentum" term which helps to deal with noise. Because, out gradients are approximations due to the nature of DFA, we can imagine these gradients being noisy which the ADAM is more equipped to deal with using the "momentum" term unlike the SGD method. As a result, I believe ADAM optimizer should be the default choice over the popular SGD algorithm.

Figure 17: Training Loss curves for Red curve: DFA+SGD, orange curve: DFA+ADAM

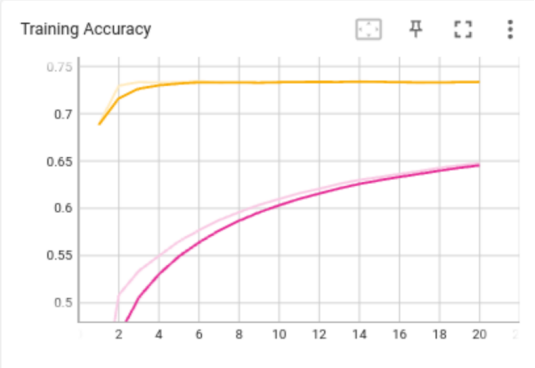

Figure 18: Training accuracy curves for Red curve: DFA+SGD, orange curve: DFA+ADAM

3.2.4 Dropout

In my experiments I have found that dropout is a valid normalization method that can be effectively used with DFA training. Increasing the dropout_p results with less overfitting on the training data and ideally better generalization performance. My experimental results in figure 19 attests to this fact. In the figure one can see as the dropout_p increases the degree of overfitting decreases. Unlike BP, during DFA training the errors are propagated in a parallelized manner. On the contrary, dropout method leverages the sequential nature of the forward pass to increase the resilience of the network to certain layers to increase the generalization performance. Despite this contrast between the parallelization and the sequential computations during the different phases of the training, dropout still proved to be a viable normalization option for the DFA training.

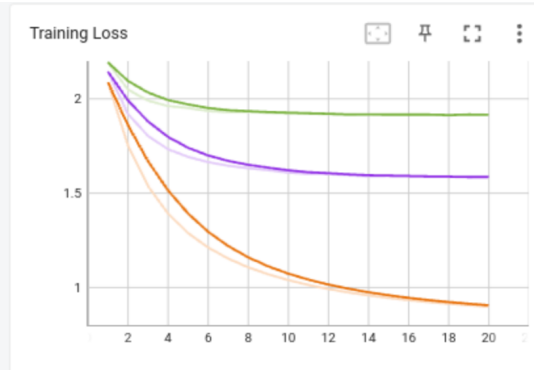

Figure 19: Training loss curves for green curve: high_dropout_p, purple curve: medium_dropout_p, orange curve: small_dropout_p

3.2.5 BatchNorm and LayerNorm

In my experiments I have found BatchNorm to result in higher training loss and, LayerNorm to make almost no substantial difference in the performance of the model. The figure 20 shows one such result from my experiments. Due to additional computational requirements introduced by such methods and the results I have obtained, I advise against the use of such methods with DFA training. It is important to note that this is not the case for the BP training since under the same conditions we got improvements using those methods with BP.

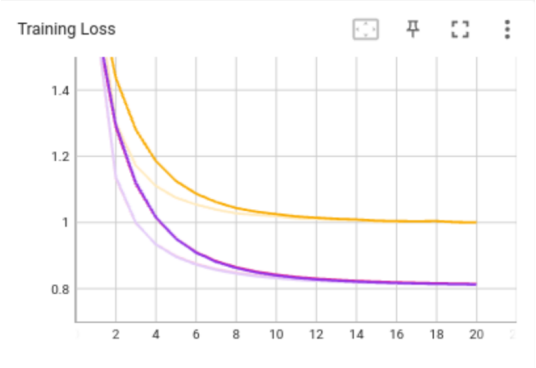

Figure 20: Training loss curves for orange curve: BatchNorm, purple curve: LayerNorm, red curve: no normalization

3.2.6 L1 Regularization, L2 regularization, and Weight Decay

L1 regularization, L2 regularization and Weight Decay all appears to be still viable regularization options for DFA training as they are for BP. I have conducted several experiments with different regularization λ values and compared its effects on BP training and DFA training. I have concluded that they all had similar effects between DFA and BP training which hinted on the fact that it behaves the same for DFA as it did for BP. This result theoretically made sense too since, the regularization terms computed are added to the loss of the model after the final layer which also appears to be the global error of DFA. Since we are using this global error to propagate the error signals backward in the DFA training this method these additional regularization terms are being propagated to every layer of the model and as a result the methods work as they were originally intended. Results of one of my experiments which shows the similarity between how L1 regularization behaves similarly for BP and DFA can be found in figure 21. On the figure one can see that the addition of L1 Regularization translated the loss curves upward for both DFA and BP training with respect to their corresponding baseline loss curves.

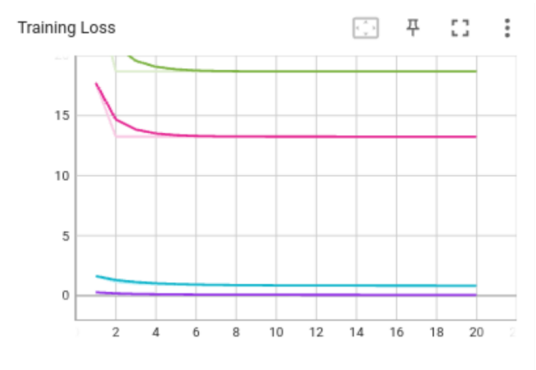

Figure 21: Training loss curves for red curve: DFA+no_L1, green curve: DFA+l1_reg, purple curve: BP+no_L1, blue curve: BP+L1_reg

3.2.7 Activation Functions

The effectiveness of different activation functions are done experimentally. One such experiment I have conducted can be found on the figure 22. Even though, ReLU is among the most popular choice for activation functions for Neural Networks with BP training due to its many benefits such as ease of its computation, it appears to be not the best choice. Activation function such as *SeLU*, *CeLU*, and *tanh* appear to be yielding the best performance. They are then followed by the *PReLU* and *GeLU* activation functions. Among them all ReLU yields the best performance.

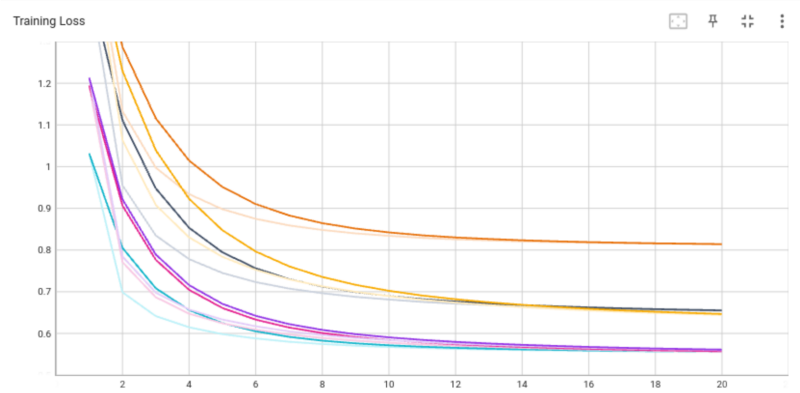

Figure 22: Training loss curves for DFA training with different activation functions (orange curve: ReLU, grey curve: PReLU, blue curve: SeLU, red curve: CeLU, orange curve: GeLU, purple curve: tanh)

3.3 Conclusions

In this work I have successfully implemented Direct Feedback Alignment by extending Pytorch which is one of the most popular Deep Learning Libraries. I have experimented with many regularization functions and activations functions, and investigated how they behaved. I have evaluated those empirical results with respect to how they compared between BP and DFA training and how well they have performed under which circumstances. Some of the findings such as the tolerance to high learning rate, and activation functions hint on the importance of reinvestigating the popular approaches used with BP for their applicabilities to DFA training. As a future direction, DFA implementation can be extended to be parallelized, and harder experiments can be conducted on harder datasets such as the ImageNet dataset.

References

Crafton, B., Parihar, A., Gebhardt, E., and Raychowdhury, A. Direct feedback alignment with sparse connections for local learning, 2019. URL `https://arxiv.org/abs/1903.02083`.

Han, D. and Yoo, H.-j. Efficient convolutional neural network training with direct feedback alignment, 2019. URL `https://arxiv.org/abs/1901.01986`.

Han, D., Park, G., Ryu, J., and Yoo, H.-j. Extension of direct feedback alignment to convolutional and recurrent neural network for bio-plausible deep learning, 2020. URL `https://arxiv.org/abs/2006.12830`.

Launay, J., Poli, I., and Krzakala, F. Principled training of neural networks with direct feedback alignment, 2019. URL `https://arxiv.org/abs/1906.04554`.

Launay, J., Poli, I., Boniface, F., and Krzakala, F. Direct feedback alignment scales to modern deep learning tasks and architectures. 2020. doi: 10.48550/ARXIV.2006.12878. URL `https://arxiv.org/abs/2006.12878`.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1), November 2016. doi: 10.1038/ncomms13276. URL `https://doi.org/10.1038/ncomms13276`.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, April 2020. doi: 10.1038/s41583-020-0277-3. URL `https://doi.org/10.1038/s41583-020-0277-3`.

Nøkland, A. Direct feedback alignment provides learning in deep neural networks, 2016. URL `https://arxiv.org/abs/1609.01596`.

Refinetti, M., d'Ascoli, S., Ohana, R., and Goldt, S. Align, then memorise: the dynamics of learning with feedback alignment. 2020. doi: 10.48550/ARXIV.2011.12428. URL https://arxiv.org/abs/2011.12428.

Sanfiz, A. J. and Akrout, M. Benchmarking the accuracy and robustness of feedback alignment algorithms, 2021. URL https://arxiv.org/abs/2108.13446.