

**SISSA**

Scuola  
Internazionale  
Superiore di  
Studi Avanzati

Data Science Area - PhD course in  
Theoretical and Scientific Data Science

# Direct Feedback Alignment for Continual Learning, Bayesian Neural Networks and Recurrent Neural Networks

Candidate:  
Sara Folchini

Advisor:  
Prof. Sebastian Goldt  
Co-advisors:  
Prof. Alessandro Laio

Academic Year 2023-2024



## Abstract

Despite their huge success in many fields of engineering science, neural networks continue to suffer from a number of shortcomings. For example, they exhibit catastrophic forgetting: they will forget the details of a task when learning a second task. They are also susceptible to adversarial attacks: small input perturbation that make the network change its prediction. These problems do not seem to hamper biological neural networks. One difference between biological and artificial neural networks that could account for this difference is the learning rule: while artificial neural networks are universally trained with (variants of) stochastic gradient descent, this algorithm is not a good model for learning in biological neural networks. In this thesis, we therefore study the potential of a more biologically plausible learning algorithm called Direct Feedback Alignment (DFA) to alleviate these problems. The key idea of DFA is to directly project the error signal via dedicated, random feedback matrices onto the change of the weights. We first explore the idea in the context of continual learning. We find that in fully-connected networks trained on image classification tasks, DFA can alleviate catastrophic forgetting by constraining the network weights to a particular region in weight space when using the same feedback matrix across tasks, or by orthogonalising weight updates by using distinct feedback matrices for each task. We then investigate the ability of DFA to increase robustness to adversarial perturbations, and more generally to mimic the benefits of Bayesian Neural Networks, by adding a dynamic on the feedback weights to sample network parameters. We find that ensembles of networks sampled with dynamical DFA exhibit enhanced robustness to gradient-based adversarial attacks. Furthermore, the test accuracy of the ensemble outperforms a network trained with backpropagation. We finally explore the potential of Direct Feedback Alignment to train recurrent neural networks, which are an important model of recurrent computations which are omnipresent in the brain. We show that DFA can be used to train simple variants of Long Short-term Memory Networks (LSTM), overcoming the bottlenecks of the standard backpropagation-through-time algorithm. In summary, our results highlight the potential of Direct Feedback Alignment in three different domains. Our results raise the possibility that while biologically inspired learning rules for artificial neural networks may not always reach the on-task performance of vanilla backpropagation, their advantages really become clear once they are applied to complex, multi-goal settings like continual learning.

# Contents

<b>Abstract</b>	<b>2</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Neural networks for supervised machine learning	7
1.1.1 The backpropagation algorithm	7
1.2 Drawbacks of Backpropagation and biological plausibility	9
1.3 Direct Feedback Alignment	10
1.4 Continual Learning with DFA	15
1.5 Bayesian neural networks with DFA	16
1.5.1 Main hypothesis	18
1.6 RNNs with DFA	18
1.7 Appendix A	20
1.7.1 Chain rule, the essence of Backpropagation	20
<b>2 Can DFA alleviate Catastrophic Forgetting?</b>	<b>21</b>
2.1 Background	21
2.2 Hypothesis	24
2.3 Methods	25
2.4 Preliminary experiments	30
2.4.1 Degeneracy breaking in different datasets	30
2.4.2 Preliminary experiment: Learning the same dataset with different feedback matrices and different output layers	32
2.5 Empirical results	33
2.5.1 DFA—same between the two ends of the plasticity-stability trade-off	33
2.5.2 DFA—same in the Domain-IL and Task-IL scenario, testing the first hypothesis	34
2.5.3 DFA—diff in Task-IL and Domain-IL scenarios, testing the second hypothesis	34
2.6 Complementary results	35
2.6.1 Comparing DFA with Elastic weight consolidation	35

2.6.2	Ablation experiments	36
2.7	Summary of results and discussion	40
2.8	Appendix B	41
2.8.1	Results evaluated on MNIST	41
2.8.2	Random Seed impact on the accuracy of the firstly learned task	41
2.8.3	DFA performances after convolutional layers	42
2.8.4	DFA with larger architectures	48
<b>3</b>	<b>DFA for Bayesian neural networks averaging</b>	<b>49</b>
3.1	Background	50
3.2	Research hypothesis	53
3.3	Methods	54
3.4	Sampling the posterior with DFA	58
3.4.1	Changing the feedback matrix more rarely	61
3.4.2	The influence of the starting point	61
3.4.3	Averaging over different seeds	63
3.5	Robustness to Adversarial Attacks	65
3.6	Impact of rank and similarity on re-alignment time	65
3.7	Summary and discussion	68
3.8	Appendix C	70
<b>4</b>	<b>Training recurrent neural networks with DFA</b>	<b>81</b>
4.1	Introduction	81
4.1.1	Motivation	81
4.2	Prior work	82
4.3	DFA for recurrent networks	83
4.4	DFA for Gated Recurrent Unit network	85
4.5	Preliminary Experiments	85
4.6	Experiments	85
4.6.1	Methods	87
4.6.2	Results	88
4.7	Parallelization efforts	90
4.8	Conclusion	90
<b>5</b>	<b>Concluding perspectives</b>	<b>93</b>

# Chapter 1

## Introduction

Artificial Intelligence (AI) [Russell and Norvig, 2009] aims to build systems that can understand the world and can learn to solve real-world challenges. An early formulation of the AI concept was given with the Turing Test in the 1950s [Turing, 1950]. After that, AI research has advanced especially with the rise of Machine Learning (ML) [Mitchell, 1997]. ML is based on training algorithms that are used to let parametric models learn from data with very little human input. The availability of large amounts of data and the efficiency of computations have been major factors in the success of ML, which is now at the core of many modern technologies [LeCun et al., 2015]. In particular, artificial neural networks (ANNs) have become indispensable tools for applied machine learning, achieving human-level performance in a variety of domain-specific tasks such as image recognition [Krizhevsky et al., 2012, LeCun et al., 2015, Simonyan and Zisserman, 2015, He et al., 2016, Dosovitskiy et al., 2021] and natural language processing [Devlin et al., 2019, Howard and Ruder, 2018, Radford et al., 2018, Brown et al., 2020, OpenAI, 2024]. The backpropagation (BP) algorithm [Rumelhart and McClelland, 1987] remains the most widely used training method due to its simplicity and demonstrated success. However, "human-level performance" often refers solely to accuracy metrics, ignoring key challenges where ANNs lag behind human abilities. There is still a big gap between "understanding the world and learning to solve real-world tasks" and the learning that Machine Learning (ML) allows. For example, neural networks lose previously learned information when trained on new and different data (problem of catastrophic forgetting). This necessitates frequent retraining on entire datasets, which is both time-consuming and resource-intensive. In practice, in many industries, models need to be retrained daily [Lian et al., 2024] leading to costly downtime and energy usage. In e-commerce and social media, this is the case as user preferences or content trends change. Continual Learning (or Lifelong Learning) has emerged to face this issue, working towards the adaptation of learning systems to adjust to new information without forgetting previous knowledge—an ability that is common in the human brain.

ANNs do not only have limitations dealing with new training data, but also malicious

test data can be an issue [Szegedy et al., 2013, Goodfellow et al., 2014, Biggio and Roli, 2017]. For example, very simple adversarial attacks can cause the network to misinterpret images that appear identical to humans. Techniques such as ensemble averaging have been shown to reduce this risk, but the problem remains a significant hurdle for deploying ANNs in sensitive applications, including self-driving cars [Jung and Ho, 2022] and healthcare [Finlayson et al., 2019].

Finally, the energy consumption required for training large-scale models is a growing concern. For instance, training a state-of-the-art language model like GPT-3 is estimated to consume 1,287 MWh, equivalent to the carbon footprint of 626,000 miles driven by an average car [Brown et al., 2020]. This is far greater than the energy humans use in acquiring and applying the same knowledge. In tasks such as training recurrent neural networks with backpropagation, the computational cost increases further due to the need to unroll the network across many time steps.

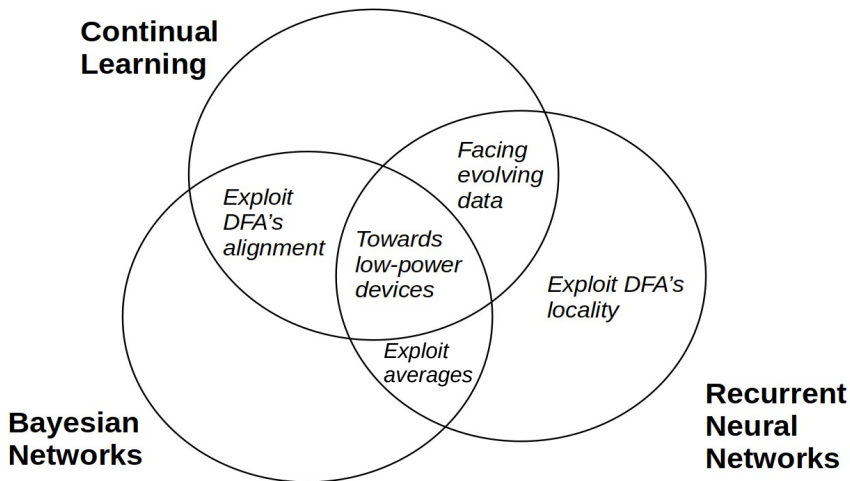


Figure 1.1: Graphic representation of topic overlap.

In this work, we aim to address these challenges in ANNs by building on the approximate-gradient training method, Direct Feedback Alignment (DFA), first introduced by Nøkland in 2016. DFA offers a biologically inspired alternative to backpropagation.

The areas covered in this thesis are interconnected with each other following the scheme in 1.1. These contributions represent a step toward making neural networks more adaptable, secure, and sustainable for future machine learning applications. Before we dive deep into the necessary background on feedback alignment, let us review the standard method to supervised learning with neural networks, from the architecture to the backpropagation

algorithm.

## 1.1 Neural networks for supervised machine learning

Supervised machine learning requires three components: a dataset composed of input-target pairs; a neural network consisting of interconnected computing units with parameters (weights and biases), which is used to process the input to obtain the output; lastly, it requires a loss function to indicate the difference between the output of the network and the ground truth that we have from the dataset. The aim of the training algorithm is to minimize a training loss, which is a function of the network parameters that measures the performance of the network; in a classification task, this could simply be the number of misclassifications. By tuning parameters to minimise the loss, the output of the network will match the target values in the training data. If you adopt the perspective of the output being an answer to the input, then matching the correct labels can be interpreted as learning.

The network, also referred to as *architecture*, is a scheme that breaks down the non-linear function that connects the input  $x \in \mathbb{R}^D$  to the output  $y \in \mathbb{R}^d$ . The building blocks of the networks are *perceptrons*: a mathematical function that processes the input first through a weighted sum, and then through a threshold that is called *activation function*. These two steps of integration and filtering were originally inspired by the behavior of biological neurons: a neuron is a biological cell that receives a chemical stimulation to its dendrites. Some chemical components can polarize (increase the voltage) the membrane of the dendrite, other can de-polarize it. The different polarization of the dendrites travels toward the body of the cell, where a physical summation of the signals happens. Finally, if the body of the cell has a residual polarization over a threshold, a signal is fired from this cell to the other ones connected to its axon. Even though the mathematical representation of this process is not as rich as reality, it is a good approximation of a biological neural network and is entitled to be called Artificial Neural Network. The *Fully-connected* architecture is composed of layers of neurons, where the first layer processes the inputs and then neurons from one layer send the outputs to the following layer, until reaching the last layer that has as many units as the dimension of the desired output (see Figure 1.2).

### 1.1.1 The backpropagation algorithm

Backpropagation states the rule for computing the gradients of the network's parameters by propagating the error backward from the output layer to the input layer. Derivatives (or gradients, if you consider the derivative of the loss by the whole matrix of parameters) are computed by using the chain rule.

The update formula for the weights  $w$  is given by

$$w_{new} = w - \eta \frac{\partial L(w)}{\partial w} \tag{1.1}$$

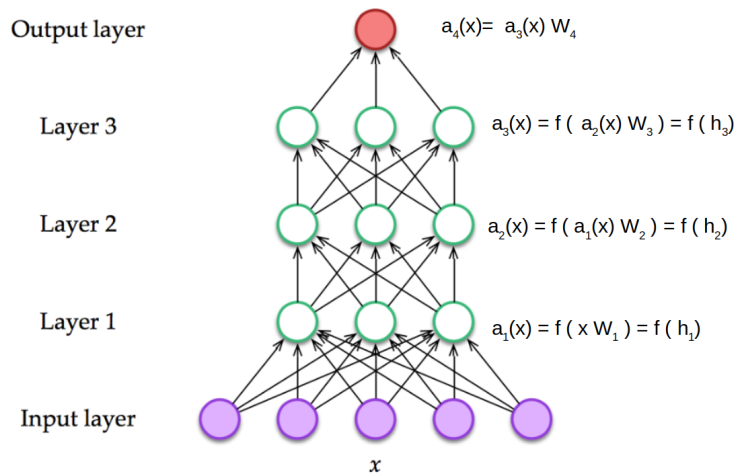


Figure 1.2: Graphic representation of a 4-Layer FC architecture. Every green circle is a perceptron.  $\sigma$  is the softmax function,  $f()$  is an activation function. (Image from [Passerini \[2022-2023\]](#))

where  $\eta > 0$  is the learning rate. The intuition for it is that a derivative is the change in the output given a perturbation of the input, so updating every parameter in the other direction is like adopting all the possible changes that would bring the loss to a lower value of the loss.

Although BP is an old idea, it gained popularity with [Rumelhart and McClelland \[1987\]](#) in 1986 when David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams presented how effective BP can be at training the network to learn the representations. The number of theoretical and experimental papers investigating backpropagation is legion, so we will not attempt to give a review here.

The loss of a multilayer perceptron has several minima, and BP with Gradient Descent (GD) is guaranteed to converge to a local minimum. To help with this, one can follow the procedure of Stochastic Gradient Descent (SGD) instead of GD.

In this case, the training data is divided by the algorithm in mini-batches. In image classification, for example, all the images belonging to one mini-batch are processed at the same time. This procedure is inherent to the Stochastic Gradient Descent protocol, in contrast to online learning where one input is processed at a time, and Gradient Descent where all the data are processed together. The advantage of SGD is both regarding computational costs (for large datasets, processing all the data together can be infeasible for the memory, online learning requires one update step for every entry of the dataset, requiring extremely long training time) and for bringing stochasticity during training, which helps with finding a global minima and in other words improving generalization: finding one



solution that is not specific to the training set, but allows to predict well also the test set.

## 1.2 Drawbacks of Backpropagation and biological plausibility

For supervised learning, the Backpropagation algorithm (BP) [Rumelhart and McClelland, 1987] has few real competitors in training neural networks nowadays due to its simplicity and proven performance. But, although it was originally inspired by biology [Rosenblatt, 1958, Fukushima and Miyake, 1982], and even though many studies showed that ANNs trained with BP could capture similar information as biological neural networks (e.g., specific nodes learn the edges, corners), it is also clear that BP is not compatible with how biological neurons learn [Bengio et al., 2015].

For instance, BP is a **sequential process**, and there are locking mechanisms (forward, backward, and update) that require that none of the processes is executed before its preceding completion. This makes BP infeasible for parallel processing because the computation has to be precisely clocked to alternate between forward and backward passes. Hence deeper and larger networks' training can be not just computationally expensive, but also slow. This is not the case in the mammalian brain, where the synapses are updated asynchronously and the cost of computation is much smaller. In BP, not only each execution has to wait for its preceding process, but the computation of the derivatives themselves needs the sequential computation of the **chain rule** to compute the derivatives of the loss (at the end of the network) over all the parameters, including the ones in the early layers of the network.

Biological networks such as mammalian brains instead operate based on local information because synaptic weights depend solely on the activities of connected neurons [S. Qin, 2021, Tang et al., 2022]. This problem is closely related to the fact that BP assumes and imposes the symmetric weights for the forward and the backward phases. This creates the additional problem of using the **transpose of the weights** of the feedforward connections to calculate the gradients of the loss. In the literature, this issue is known as the weight transport problem, and it is one of the most discussed differences between backpropagation and biological learning. Moreover, the computation of the gradients involves the derivative of the non-linearities at the operating point used in the corresponding feedforward computation. This means that **information** regarding the feedforward pass is needed for the backward pass. This problem is called "transport problem" [Grossberg, 1987, Crick, 1989]. Also, the **ground truth** label is not directly represented in the brain, for rendering it biologically plausible it should be modeled through a reward [Bengio et al., 2015, Lee et al., 2015]. One characteristic of biological systems is robustness to parameter perturbations, most of the time they are resilient in the case of over-expression or under-expression of signaling molecules. In this sense, BP lacks of biological plausibility because it has strict technical requirements, such as the need for interventions against vanishing or **exploding**

**gradients** (gradient clipping) and the need for differentiable activation functions.

Biological plausibility is significant because of several reasons. We know that biological neurons inspired ANNs. Hence, it is interesting to examine the dissimilarity or similarity among them. Moreover, there is a field that is the intersection of neuroscience and deep learning, so it is natural to investigate the biological plausibility feature of the algorithms.

As stated at the beginning, other drawbacks of backpropagation include the fact that most of the time as new tasks are learned the previously acquired ones vanish. This is not the case in biological brains, where knowledge consolidated at the synaptic level is retained and sometimes new tasks can be acquired even leveraging on that. Moreover, biological systems learn from continuous streams of information, allowing for real-time processing of information instead of processing entire batches of data at a time.

Finally, the adversarial attacks can easily trick ANNs trained with BP. These attacks consist in specifically designed inputs [Kurakin et al., 2017] that are misclassified by the ANN, while there is no difference for the human perception.

### 1.3 Direct Feedback Alignment

The scientific community designed some alternative algorithms for addressing the limitations of traditional artificial neural networks (ANNs) that most of the time rely on backpropagation. Some of these methods are the *biologically plausible algorithms*, meaning that there is some consistency between the algorithm’s principles and existing biological, and neuro-scientific knowledge. These biological systems that these algorithms seek to emulate the learning mechanisms are particularly the mammalian brains [S. Qin, 2021, Tang et al., 2022]. Technically, these algorithms can be divided in three categories: self-supervised learning mechanisms, where the update of the weights are based on current neuron activities without extensive supervision [Tang et al., 2022]; Direct Feedback Alignment (DFA), which are methods that use fixed random connections for error propagation, avoiding the need for symmetric feedback connections while still enabling effective learning in deep models [Lillicrap et al., 2016b]; and layer-wise learning, where the layers are updated either sequentially or randomly, facilitating a more flexible and biologically relevant training process [Tang et al., 2022].

DFA was first presented by Arild Nøkland from the University of Trondheim, Norway [Nøkland, 2016] and it belongs to the second branch, of the biologically plausible alternatives to backpropagation that focuses on solving the weight transport problem. It propagates the error through fixed random feedback connections directly to the hidden layers, making the forward and the backward flows different from each other. On top of this solution for the weight transport problem, the update of one layer does not depend on the other layers, so it is possible to update all the layers in parallel. This is crucial to resemble local learning, which is believed to govern synaptic weight updates in the brain [Bengio et al., 2015]. Dellaferrera et al. [2021] introduced a version of DFA with one more biological

constraint, where each neuron can fire only if receives more than one non-zero input. The introduction of this feature regarding our application is left for future work.

Despite relying on random feedback weights for the backward pass, DFA has been proven to yield comparable performance to BP to certain problems such as view-synthesis, recommendation systems, and small scale image problems [Launay et al., 2020].

DFA has been successful in training fully-connected (FC) Neural Networks [Launay et al., 2020, Lillicrap et al., 2016b, Nøklund, 2016] and to fine-tune pre-trained convolutional networks [Crafton et al., 2019], it has also been applied to complex architectures such as transformers [Launay et al., 2020] and GNNs [Zhao et al., 2024].

Its application to Recurrent Neural Networks (RNNs) hasn't been done cleanly until the research described in chapter 4.

Moreover, until now, DFA does not perform as well in convolutional architectures with more complex image datasets [Bartunov et al., 2018]. DFA is unable to train these kinds of layers even though convolutional layer can be represented by a large fully-connected layer whose weights are represented by a block Toeplitz [d'Ascoli et al., 2020]. The hypothesis for this phenomenon is that CNNs don't have enough flexibility to align [Launay et al., 2020]. This lack of alignment makes learning near to impossible [Han et al., 2020], and has led practitioners to design alternatives. The simplest solution one might think of is using BP for training the convolutional layers and DFA for the FC [Han and Yoo, 2023] and another technique would be using sparse Weight Adjustments: by introducing sparse backward weights, the algorithm can better manage the complexity associated with convolutional layers [Han et al., 2020].

Up to today, DFA's original paper has 398 citations<sup>1</sup>. It is gaining popularity in the context of efficient parallelization - recently it has been programmed Physical Neural Networks [Nakajima et al., 2022, Filipovich et al., 2022].

## Theoretical background on DFA

We revise here the algorithmic differences and similarities between FA, DFA and BP. Biases can be modeled by an augmented dataset with an additional unitary pixel for every image.

Let us consider simple neural network model and a binary classification task where the error to minimize is modeled by the binary cross-entropy (BCE) loss. The equation of the BCE loss is the following, indicating the weights of the network  $W$ ; the input  $x$  and the output of the network  $a_4$ :

$$L(W) = -[y \log(a_4) + (1 - y) \log(1 - a_4)] \quad (1.2)$$

BCE measures the difference of predicted probabilities and the actual class <sup>2</sup>, which can

---

<sup>1</sup>Data provided by Semantic Scholar, 18<sup>th</sup> August 2024

<sup>2</sup>We can notice that the logarithmic nature amplifies differences when the model is confidently wrong. If the model is unsure (predicting probabilities close to 0.5), the penalty is moderate, but as it becomes more

be either 0 or 1. In the most common notation, it is generally found as:

$$E(\theta) = - \sum_{(x,y) \in \mathcal{D}} [y \log f_{\theta}(x) + (1 - y) \log(1 - f_{\theta}(x))] \quad (1.3)$$

This example is not chosen arbitrarily. Indeed some of the the classification tasks we will adopt will be binary classification problems. It can be generalized to multi-class classification using a softmax for output layer.

To state the DFA weight updates clearly, and to contrast them with vanilla backpropagation, we consider a FC network of depth (number of layers)  $L$  with weights  $W_l$  in the  $l^{th}$  layer. Given an input  $x \equiv h_0$ , the output  $\hat{y}$  of the network is computed sequentially as  $\hat{y} = f_y(a_L)$ , with  $a_l = W_l h_{l-1}$  and  $h_l = f(a_l)$ , where  $f$  is a pointwise non-linearity function. For classification, the loss function  $L$  is the BCE and  $f_y$  is the Softmax (turning the output of the network into probabilities). Given the error  $e \equiv \frac{\partial L}{\partial a_L} = \hat{y} - y$  of the network on an input  $x$ , the update of the last layer of weights reads  $\delta W_L = -\eta e h_{L-1}$ . The updates of the layers below are given by  $\delta W_l = -\eta \delta a_l h_{l-1}^T$ , with factors  $\delta a_l$  defined sequentially as

$$\delta a_l^{\text{BP}} = \frac{\partial L}{\partial a_l} = W_{l+1}^T \delta a_{l+1} \odot g'(a_l) \quad (1.4)$$

with  $\odot$  denoting the Hadamard product. In the case of FA the transpose of the network weights  $W_l^T$  are replaced by random feedback connections  $F_l$ .

$$\delta a_l^{\text{FA}} = (F_{l+1} \delta a_{l+1}) \odot g'(a_l) \quad (1.5)$$

DFA adds to this idea the fact that the error is propagated directly from the output layer to each hidden layer, replacing the term  $\delta a_{l+1}$

$$\delta a_l^{\text{DFA}} = (F_{l+1} e) \odot g'(a_l) \quad (1.6)$$

To summarize, while the final layer is updated in the same way with both DFA and BP, the weight updates for the other layers are all different. BP implements the exact gradient for each layer by applying the chain rule to compute the derivatives of the loss; instead, DFA keeps only the error term and substitutes the derivative of the following layer by an entry of the feedback matrix.

[Lillicrap et al., 2016a] gave a first theoretical characterization of feedback alignment by analysing two-layer linear networks. In this shallow setup FA and DFA are equivalent, and

---

confident and incorrect, the penalty is stronger - removing the saturation effect of the activation function. In a statistical sense, BCE is closely related to Kullback-Leibler (KL) divergence and it can be interpreted as the expected negative log likelihood of the model's predictions, weighted by the true labels. On another perspective, one can notice that BCE is closely related to mutual information, in fact it can be interpreted as a way of measuring how much information is "lost" when predicting a label distribution from the model. If the model is perfectly accurate, the cross-entropy is low, meaning little information is lost.

only one feedback matrix is involved:  $F_1 \in R^K$ , with  $K$  being the number of hidden nodes in the intermediate layer, which backpropagates the error signal  $e$  to the first layer weights  $W_1$ . The updates of the second layer of weights  $W_2$  are the same as for BP. FA and DFA are able to minimize the loss because the transpose of the second layer of weights  $W_2$  tends to align with the random feedback matrix  $F_1$  during training. This weight alignment (WA) leads the weight updates of FA to align with those of BP, leading to gradient alignment (GA) and thus to successful learning of the previous layer.

DFA learning dynamics in non-linear networks was then studied by [Refinetti et al., 2021]. It is worth going through the main findings of their analyses because there are key concepts for the interpretations of our results. First of all, they study the generalization error of ReLU 2-layers FC networks in the teacher-student setup. The second layer’s weights of the student must retrieve the same weights of the teacher and align with the feedback matrix at the same time, and they find that this is possible only in over-parametrized network because larger networks have more global minima, and DFA is able to reach one of these solutions after aligning to the Feedback matrix. Secondly, they showed that in this setup, DFA proceeds in two steps: an alignment phase, where the forward weights adapt to the feedback weights to improve the approximation of the gradient, and by a convergence phase, where the network sacrifices some alignment to minimise the loss. Out of the same-loss-solutions in the landscape, DFA converges to the one that maximises gradient alignment, an effect we term “degeneracy breaking”<sup>3</sup>.

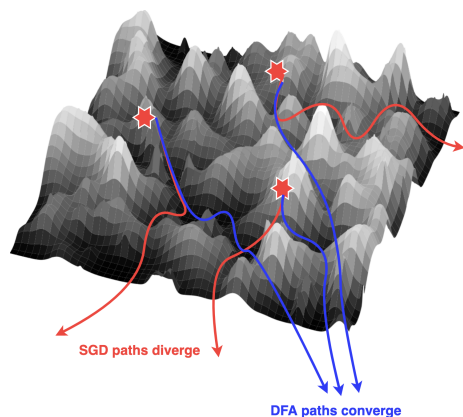


Figure 1.3: Refinetti et al. [2021], cartoon of the alignment that they showed in their experiments on MNIST and CIFAR10.

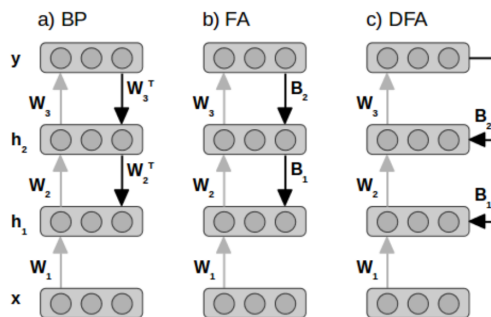


Figure 1.4: Schematic representation of BP, FA and DFA. from Nøkland [2016].

In the case of deeper linear networks, they computed the accumulation of the updates

<sup>3</sup>The term “degeneracy” in physics often refers to multiple solutions to one problem. The inductive bias of the algorithm breaks this degeneracy by selecting one of the solutions.

from time 0 to the training step  $t$ , in a case in which the weights were initialized to zero. This computation is useful to see that the updates of the different layers contain a combination of the feedback matrices of the corresponding layer and the following layers, plus one term called "alignment matrix" that is a Hebbian term. For example, the updates accumulated in the second layer read as follows:

$$W_2^t = -\eta \sum_{t'=0}^{t-1} F_2 e_{t'} (W_1 x_{t'})^T \quad (1.7)$$

$$= F_2 A_2^t F_1 \quad (1.8)$$

The fact that the weight updates are proportional to the Feedback matrix or a combination of feedback matrices is called *weak alignment*. This alignment impacts on the weight more and more as the time steps accumulate. The strict GA arises for layers after the first if the alignment matrices become close to the Identity (*strong alignment*). For this result to hold, the feedback matrices should be left-orthogonal. This can be taken as a structural choice for the feedback matrices, but also if they are sampled from a Gaussian distribution this arises in expectation. In the case of strong alignment though, all the layers after the first are completely sacrificed to be aligned, allowing only the first layer to learn.

This is not harmful for the linear networks as the first layer alone is enough to maintain full expressivity. In nonlinear networks, instead, the alignment is only one phase, followed by the convergence to the closest solution.

Furthermore, Empirical experiments show that also in non-linear networks (4 layers, FC) there are the two phases of alignment and convergence to the closest solution. In particular, the bottom layers (the ones closer to the input) loose gradient alignment first (only the first layer deviates from GA at the end of training on CIFAR10); and WA is lost subsequently for the second layer and then for the third layer.

The **main take-away** from the analysis of [Refinetti et al., 2021] is that weights learnt by DFA carry the fingerprint of the feedback matrices. This can be seen as a byproduct of the way the algorithm is formulated. In this thesis instead, we embrace this fact and ask: can we explicitly use the impact of the feedback matrix to steer the networks in a preferred direction?

### Further related work

Bordelon and Pehlevan [2023] applies the limit of infinitely wide NNs and studies DFA (together with FA, Hebbian learning and Gated Linear Networks) in the rich and lazy regimes (where the feature embeddings at each layer are constant through time). They determine the equations of the dynamics evolution through the Dynamical Mean Field Theory (DMFT) approach. The convergence to this DMFT occurs at large width  $N$  with error  $O(N^{-1/2})$ . Regarding DFA, they confirm that in the rich regime the alignment between the true gradient and the pseudo-gradient (backward pass of DFA) increases during

the training of the layers dedicated to creating representations (all layers except for the output layer).

## 1.4 Continual Learning with DFA

The main challenge in Continual Learning is analyzing and addressing catastrophic forgetting [Goodfellow et al., 2015b, McCloskey and Cohen, 1989], namely the performance degradation of previously-learned tasks. Catastrophic forgetting is a significant drawback of traditional Neural Networks and this limitation impedes their ability to continuously learn and adapt to evolving environments, limiting their practical applicability in real-world scenarios.

---

Initialize $\theta_0$
$\theta_1 \leftarrow$ start from $\theta_0$ and minimize $L_1(\theta)$ with SGD
$\theta_2 \leftarrow$ start from $\theta_1$ and minimize $L_2(\theta)$ with SGD

---

Table 1.1: Simplest scheme: Continual Learning with Stochastic Gradient Descent.

Biology offers an existence proof for successful continual learning in complex environments and also hints at the design principles and trade-offs of successful approaches. For example, the main techniques proposed for continual learning using ANNs have a biological equivalent [Hadsell et al., 2020].

1. Synaptic homeostasis is modeled by regularisation approaches such as Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017] and Synaptic Intelligence [Zenke et al., 2017].
2. The modularity of the brain is reflected by dynamical architectural solutions where the architectural complexity grows with the number of tasks; followed by pruning procedures for keeping the network’s size contained.
3. Memory plays a crucial role in replay approaches [ROBINS, 1995], [Rolnick et al., 2019] that alternate training on new tasks with training on some examples of past tasks;
4. Context-dependent learning of representations is motivated from a neuroscientific perspective given recent evidence that neural population codes orthogonalize with learning [Flesch et al., 2022, Faylor et al., 2021, Zeng et al., 2019]. Context-dependent learning of representations is reflected by Orthogonal Weight Modification algorithms.

With the application of DFA to Continual Learning, we continue this line of thought to explore whether this particular biologically inspired algorithms can mitigate catastrophic forgetting.



In particular, the degeneracy breaking ability of DFA can help in CL because it works as an **implicit regularization** on the weights: as the feedback matrix is kept the same for all tasks, the same-loss solution that the network achieves after training on different datasets will have something in common: to be the closest one to the trajectory of alignment of the weights to the (combination of) feedback matrix. Moreover, the same degeneracy breaking can render DFA a convenient way for adopting **orthogonal gradients** during the training on the different datasets. This is the case if the feedback matrix is set to be orthogonal to the one used during training on another dataset. In this condition, the alignment trajectories will be orthogonal to each other, thus driving learning in the directions that are less relevant for the learning of the previous dataset.

## 1.5 Bayesian neural networks with DFA

Modeling uncertainty is a desirable feature in Continual Learning and beyond, to detect when a new task has started (problem also known as out-of-distribution input detection). Moreover, uncertainty is a key feature that NNs lack for a full reliability. Reliability is essential for the integration of AI in robotics [Loquercio et al., 2020] and in critical applications like healthcare [Kompa et al., 2021, Chua et al., 2022] and autonomous driving [Shao et al., 2024] where understanding the certainty of a prediction can help evaluating the risks and better prevent accidents. With uncertainties, practitioners could focus on gathering more data in areas where the model is less confident and tuning the model to improve robustness on top of the performances. Not only out of distribution data could be found naturally exploring real-world experiences as in robotics, but some malicious attacks could try to manipulate the inputs to change the outcome of the prediction. Adversarial attacks are techniques used to manipulate the inputs of these models in a way that causes them to make incorrect predictions [Bortolussi et al., 2024, Szegedy et al., 2013, Goodfellow et al., 2014, Biggio and Roli, 2017]. These attacks exploit the vulnerabilities inherent in neural networks, which can surprisingly be sensitive to small, often imperceptible changes in input data. The most famous examples regard image classifications where an attacker might alter pixel values in a way that is not noticeable to humans but significantly affects the model’s predictions. This can lead to misclassifications such as identifying a panda as a gibbon [Goodfellow et al., 2015a] or a banana as a toaster [Lee et al., 2024].

Bayesian Neural Networks (BNNs) are a type of neural network that incorporate Bayesian inference to model uncertainty in their predictions. Unlike traditional neural networks, which produce deterministic outputs after training, BNNs aim to estimate a probability distribution over the network’s weights, allowing them to express uncertainty in both the model’s parameters and its predictions.

The core idea behind Bayesian Neural Networks is to apply the principles of Bayesian statistics, where instead of finding a single set of optimal weights, the network learns a posterior distribution over the weights. This posterior is derived from a prior distribution



(representing initial beliefs about the parameters before seeing the data) and the likelihood (the fit of the model to the observed data). This approach enables BNNs to model uncertainty more effectively, especially when data is sparse or noisy.

BNNs are particularly useful in fields where understanding uncertainty is critical, such as medical diagnosis [Abdullah and Hassan, 2022], autonomous driving [Henne et al., 2021], and financial forecasting. In these cases, having a measure of confidence in the predictions is often as important as the predictions themselves. For instance, a BNN could not only predict a possible outcome but also provide a distribution showing how confident the model is in that outcome. Moreover, Bayesian networks have been shown to be more robust to adversarial attacks than plain ANNs [Bortolussi et al., 2024, Cardelli et al., 2019, Li and Gal, 2017, Bekasov and Murray, 2018].

One of the key challenges in Bayesian Neural Networks is the computational complexity of inferring the posterior distribution over the weights. Traditional exact inference methods, such as Markov Chain Monte Carlo (MCMC), are often impractical for large networks. To address this, various approximation techniques like Variational Inference and Monte Carlo Dropout have been introduced, allowing BNNs to scale more efficiently while still capturing useful uncertainty estimates.

One approach to constructing approximately Bayesian Neural Networks (BNNs) is by averaging the outputs of multiple independent networks, a technique known as ensembling. In this method, several neural networks are trained independently on the same task, and their predictions are averaged to approximate a posterior distribution over the model's outputs. This can be seen as an implicit Bayesian model, where the ensemble of networks represents a sample from the posterior distribution.

Each independently trained network in the ensemble corresponds to a different local optimum in the weight space. By averaging their predictions, the ensemble approximates the variability that would otherwise be captured by the posterior distribution in a Bayesian model. This approach captures both epistemic uncertainty (uncertainty due to the model, which can decrease as more data is gathered) and aleatoric uncertainty (inherent noise in the data). As a result, averaging independent networks can improve the model's robustness to adversarial attacks and to noise.

The limitations of Standard Ensembling is the lack of diversity: although ensembling relies on training independent models, the networks can still learn similar patterns if they are initialized similarly or trained on the same dataset using the same architecture. This can limit the diversity of predictions, which reduces the ensemble's ability to generalize and capture uncertainties effectively.

The second limitation to Ensembling is the lack of guarantee that the average of samples obtained by training from different seeds will sample the posterior with the right frequency.

Markov-Chain Monte Carlo (MCMC) is a principled tool to sample the posterior distribution because it meets the ergodicity condition:

**Irreducible** The chain must have the ability to reach any part of the state space from

any starting point, so there are no isolated regions.

**Aperiodic** The chain must not get stuck in periodic loops and should move to new states with some degree of randomness. Technically, the proposal should not propose the same state periodically.

**Positive Recurrent** The chain should return to a given state within a finite expected time.

Overall, ergodicity indicates that a system cannot be reduced into smaller components and that a large collection of random samples can represent the average statistical properties of the entire system. Samples taken from standard network averaging are not guaranteed to satisfy these conditions, but the peculiarity of the degeneracy breaking of DFA can allow to bring the sampling in different minima of the posterior, allowing a better exploration of the space.

### 1.5.1 Main hypothesis

One hypothesis of this thesis, which will be central to the work presented in chapter 3, is that the align, then memorise dynamics of DFA [Refinetti et al., 2021] can overcome this problem: the fact that different feedback matrices drive the training trajectory in different (orthogonal) directions, ensures a more rigorous attempt to the cancellation of the gradients in non-relevant directions. Moreover, DFA is able to reach convergence to different minima among the same-loss minima, so it gives a tool for reaching the different modes of the posterior.

Once one minima is selected, one can still use DFA’s peculiar alignment to collect samples in the region of the space. The practice of taking samples during training was already used to approximate Bayesian Networks [Welling and Teh, 2011a], but we adopt a dynamical sampling method for with the novelty possibility of directing the trajectories in different directions thanks to DFA’s alignment. This makes the exploration of the weights space more efficient and brings sampling during training one step closer to appropriate sampling techniques. We will empirically show that these ideas improve the network’s generalization error and robustness against adversarial attacks. Techniques that improve adversarial robustness, such as adversarial training, often require significant computational resources, this work contributes to the ongoing challenge of energy efficiency in AI.

## 1.6 RNNs with DFA

The adoption of backpropagation-based learning systems, with a few exceptions [Wright et al., 2022], is still mainly limited to digital computers and simulations. It is well known that backpropagation cannot be easily implemented and deployed in physical systems [Momeni et al., 2023, Lillicrap et al., 2020] due to issues like the weight transport where

the synaptic weights of the backward circuit need to be constantly synchronized with the synaptic weights of the forward circuit [Lillicrap et al., 2016b, Akrouf et al., 2019].

Physical deployment of backpropagation is even more challenging in Recurrent Neural Networks (RNNs) [Elman, 1990], where credit assignment must be performed across time. The most used algorithm to date is BackPropagation Through Time (BPTT) [Werbos, 1990], which extends backpropagation to recurrent architectures.

Over time, several backpropagation-free algorithms have been proposed (see Section 4.2 for a non-exhaustive overview), some of them with the explicit objective of being compatible with the implementation in physical systems or on unconventional hardware (e.g., neuromorphic, optical).

DFA has already been implemented in nonconventional hardware, especially photonic [Filipovich et al., 2022]. The photonic co-processor introduced in Launay et al. [2020] scales DFA to trillion-parameter random projections.

Here, we propose an extension of DFA tailored to recurrent neural networks. Our approach is able to compute the update of the recurrent parameters in parallel over all the time steps of the input sequence, thus removing one of the major drawbacks of BPTT. In fact, BPTT sends the error signal computed at the end of the input sequence *back in time* to compute the network parameters update. Instead, the update computed by our version of DFA is local at each time step, as it does not rely on the update computed for other time steps. Due to the weight sharing present in RNNs, the local update is eventually aggregated at the end of the input sequence to compute the final update. The aggregation operation includes information from all the time steps, thus enabling learning of temporal dependencies.

Practical applications of DFA to RNNs have been explored by [Nakajima et al., 2022]. The authors performed physical deep learning with an optoelectronic recurrent neural network. However, in their pioneering work, they do not explore the DFA algorithm in the context of fully trainable RNNs, since they only provide a proof-of-concept using a reservoir computing model with untrained reservoir connections [Lukoševičius and Jaeger, 2009]. In this thesis chapter, we will investigate the potential of DFA on fully-trainable RNNs, which are much more powerful.

[Han et al., 2020] investigated a DFA-inspired algorithm for RNNs. However, their version of DFA is restricted and cannot be applied to any recurrent or gated architecture, like our approach. First, they implement an upper triangular modular structure. Second, they use random projections as powers of the same matrix, which effectively resembles an FA algorithm applied to RNNs rather than a DFA algorithm for RNNs. Overall, our approach stems directly from DFA and closely follows its assumptions without requiring any customization, thus remaining more general and targeting any recurrent model.

In summary, since DFA allows to train all the layers in parallel, this is appealing for RNNs where the layers represent the input stream. The equations for training RNNs with BP need to be unfolded in time because the gradients of the early layers depend on the gradients of the following layers. Instead, with DFA, this problem is solved because the

error is mapped directly to every layer independently.

## 1.7 Appendix A

### 1.7.1 Chain rule, the essence of Backpropagation

Chain rule for derivatives:

$$\frac{\partial L(W)}{\partial w_{lj}} = \frac{\partial L(W)}{\partial a_l} \cdot \frac{\partial a_l}{\partial w_{lj}} \quad (1.9)$$

The term  $\frac{\partial a_l}{\partial w_{lj}}$  can be computed based on the definition of  $a_l$  that we recap here.

$$\begin{aligned} a_4(x) &= a_3(x)W_4 \\ a_3(x) &= f(\underbrace{a_2(x)W_3}_{h_3}) = f(h_3) \\ a_2(x) &= f(a_1(x)W_2) = f(h_2) \\ a_1(x) &= f(xW_1) = f(h_1) \end{aligned} \quad (1.10)$$

The term  $\frac{\partial L(W)}{\partial a_l}$  depends on the layer in consideration:

1. In the output layer, the fourth, this term is computed directly from the expression of the loss (equation 1.3);
2. For the semi-last layer (l=3) it must be expanded with one step of the chain rule

$$\frac{\partial L(W)}{\partial a_3} = \frac{\partial L(W)}{\partial a_4} \frac{\partial a_4}{\partial a_3}$$

where the first term can be computed from the expression of the loss (equation 1.3) and the second term is  $W_4^T$  ;

3. For the layers before, the chain rule iterates: for l=2 it becomes:

$$\frac{\partial L(W)}{\partial a_2} = \frac{\partial L(W)}{\partial a_4} \underbrace{\frac{\partial a_4}{\partial a_3}}_{W_4^T} \underbrace{\frac{\partial a_3}{\partial h_3}}_{f'(h_3)} \underbrace{\frac{\partial h_3}{\partial a_2}}_{W_3^T}$$

where one finds the derivative of the activation function and the transpose of the weight matrices.

After calculating the gradients, one can adjust the network parameters  $p$  using Stochastic Gradient Descent.

## Chapter 2

# Can DFA alleviate Catastrophic Forgetting?

Real-world applications of machine learning require robustness to shifts in the data distribution over time. As we introduced in the previous section, one critical limitation of standard artificial neural networks trained with backpropagation (BP) is their susceptibility to catastrophic forgetting: they “forget” prior knowledge when trained on a new task, while biological neural networks tend to be more robust to catastrophic forgetting. This chapter will start with the revision of various algorithmic ways of mitigating catastrophic forgetting that have been proposed, but developing an algorithm that is capable of learning continuously remains an open problem. As we introduced in the introduction, we are motivated by recent theoretical results on Direct Feedback Alignment (DFA) to explore whether such biologically inspired learning algorithm can mitigate catastrophic forgetting in artificial neural networks. We train fully-connected networks on several continual learning benchmarks using DFA and compare its performance to BP, random features, and other continual learning algorithms. We find that the inherent bias of DFA, called “degeneracy breaking”, leads to low average forgetting on common continual learning benchmarks in the Domain-Incremental learning scenario and in the Task-Incremental learning scenario. We show how to control the trade-off between learning and forgetting with DFA, and relate different modes of using DFA to other methods in the field.

### 2.1 Background

CL also focuses on developing **algorithms** and techniques that enable systems to maintain old knowledge, while also being able to learn new information. This requires resolving the “stability–plasticity dilemma” [Carpenter \[1986\]](#), [Mermillod et al. \[2013\]](#): a model needs plasticity to obtain new knowledge and adapt to new environments, while also requiring stability to prevent forgetting of previous information.

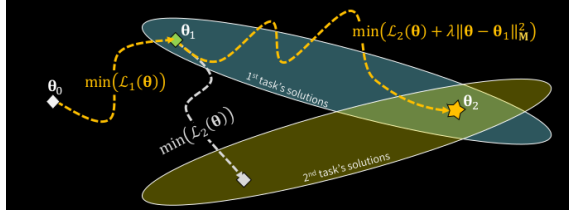


Figure 2.1: Regularization effect represented in the parameter space. In white, the loss is minimized without constraints and the solution is outside the previous task’s solution basin. In the regularized case (yellow), the constraint in the loss allows to reach a common solution.

Typical approaches to CL include the following strategies: regularization techniques, dynamic architectures, progressive learning, episodic memory replay, Orthogonal Gradient Descent (OGD) and functional approaches [De Lange et al. \[2021\]](#).

### Regularization Techniques

An algorithmic approach is regularization, that consists in adding a constraint in the loss. This restricts plasticity by penalizing changes in the parameter space, especially for parameters important to solve past tasks.

These methods need to introduce an estimate of the importance of parameters in solving problems previously encountered. And secondly impose a quadratic penalty on the difference between the parameters for the new and the old task, thus constraining the parameters to stay close to the previous values if this term is minimized.

$$\theta_2 = \operatorname{argmin}(L_2(\theta) + \lambda\|\theta - \theta_1\|_M^2) \quad (2.1)$$

Three highly influential regularization schemes and their regularization weighting are:

1. Elastic Weight Consolidation (EWC) [Kirkpatrick et al. \[2017\]](#)  
 $m_i = (\text{FI matrix of previous tasks})_{ij}$
2. Synaptic Intelligence (SI) [Zenke et al. \[2017\]](#)  
 $m_i \sim \Delta\theta_i$
3. Memory aware synapses (MAS) [Aljundi et al. \[2018\]](#)  
 $m_i = \sum_{X \in S_i} \frac{\partial \|F(x, \theta)\|^2}{\partial \theta_i}$

[Benzing \[2022\]](#) linked all of the three schemes to the Fisher Information of seen tasks.

These methods can also be considered as *prior-focused* because the penalty resembles the prior on the weights in the context of Bayesian Neural Networks. These methods have

been proven effective for over-parametrized models in the multiple epoch setting [Chaudhry et al. \[2019\]](#).

## Dynamic architectures and progressive learning

Dynamic architectures [Razavian et al. \[2014\]](#) limit interference among tasks by using different subsets of modules for each task. The simplest form of architectural regularization in convolutional neural networks is freezing certain weights in the network so that they stay exactly the same [Razavian et al. \[2014\]](#). A slightly more relaxed approach reduces the learning rate for layers shared with the original task while fine-tuning to avoid dramatic changes in the parameters [Donahue et al. \[2014\]](#) [Yosinski et al. \[2014\]](#). This class of methods are alternatively called parameter isolation methods or architectural approaches and, in general, these methods require searching over the space of architectures and causes the architectural complexity to grow with the number of tasks. **Progressive learning** is based on dynamic architecture approaches with the addition of a pruning procedure to counteract the growth in the number of parameters. [Fayek et al. \[2020\]](#),

## Episodic Memory Replay

Methods based on episodic memory replay retain representative data for observed data distributions and pass through the retained data after future tasks. The replay data is either obtained directly with stored samples (GEM [Lopez-Paz and Ranzato \[2017\]](#) , AGEM [Chaudhry et al. \[2019\]](#) , ICARL [Rebuffi et al. \[2017\]](#)) or generated using generative models [Shin et al. \[2017\]](#) [Kamra et al. \[2017\]](#) [Seff et al. \[2017\]](#). Such replay methods require more memory at training time but show better performances in the online setting [Lopez-Paz and Ranzato \[2017\]](#).

## Functional approaches

Functional approaches, focus on minimizing the deviations from previous output patterns. This differs from regularization techniques that focus on deviations in the weights. In functional approaches, the output of one or more intermediate layers are used as features for the new task and for this they are also known as feature extractor methods [Li and Hoiem \[2018\]](#) . These methods are computationally expensive as they require to compute a forward pass through the old task’s network for each new data point.

## Orthogonal Gradient Descent

Making the gradients explicitly orthogonal to each other is known to help against catastrophic forgetting, and the idea is exploited in methods such as Orthogonal Weight Modification (OWM) [[Zeng et al., 2019](#)], conceptor-aided backpropagation [[He and Jaeger, 2018](#)], and orthogonal gradient descent [[Bennani and Sugiyama, 2020](#)], because the features of

different tasks are learned along orthogonal manifolds, and the weights updates do not interfere with the previous ones. For example, in the Conceptor-Aided Backprop [He and Jaeger, 2018], for each layer of a network, CAB computes a conceptor to characterize the linear subspace spanned by the neural activations in that layer that have appeared in already learned tasks. When the network is trained on a new task, CAB uses the conceptor to adjust the gradients given by backpropagation so that the linear transformation restricted to the characterized subspace will be preserved after the gradient descent procedure.

## Theoretical approaches

Despite the empirical research is predominant in the field and many techniques developed for continual learning are based on experimental validation, continual learning is not solely experimental; it is supported by a growing body of theoretical research that seeks to understand its principles and challenges. One of the primary theoretical concerns in CL is catastrophic forgetting, and recent studies have provided theoretical insights into how various factors, such as task similarity and ordering, influence forgetting and generalization performance [Lin et al., 2023, Wang et al., 2024]. Some works have developed explicit mathematical frameworks to analyze the expected forgetting and generalization errors in CL [Saxe et al., 2013, 2018, Dominé et al., 2023, Hiratani, 2024].

Besides, research has proposed probabilistic models as a tool to break down the CL problem into sub-problems, such as within-task prediction and task-id prediction. These models help establish necessary conditions for effective continual learning, indicating that improvements in these areas correlate with better overall performance in class incremental learning (CIL) settings.

In this work, we adopt a dual approach: combining the theory of DFA with empirical validation in Continual Learning.

## 2.2 Hypothesis

We focus on the potential of DFA not just because of its greater biological plausibility compared to vanilla backpropagation. We are encouraged by the recent analysis of Refinetti et al. [2021], who showed that DFA has a *degeneracy breaking* property. To learn with DFA, the neural network has to first align its weights (to some extent) with the feedback matrices to ensure that the error signal can be backpropagated efficiently [Lillicrap et al., 2016a]. This alignment has the effect that neural networks trained by DFA always converge to the same region in the loss landscape, independently of their initialisation, and in contrast to networks trained by vanilla backpropagation. In other words, DFA will drive a neural network to a specific region in the loss landscape. The location of this region depends on the feedback matrices, thereby breaking the degeneracy of the solutions of vanilla SGD. In this chapter, we ask whether we can use the influence of the DFA feedback matrix on the



weights learnt by a neural network to mitigate catastrophic forgetting. We formulate and test two different hypotheses for how DFA can facilitate continual learning.

The **first hypothesis** is that using the *same* feedback matrix for different tasks in a continual learning curriculum prevents catastrophic forgetting by implicitly biasing the weights to a single region of loss landscape, as they always need to align with the same feedback matrix. In this case, DFA would act as an *implicit regulariser*, similar to other algorithms like Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017], a popular implicit regularisation technique. We will call this approach **DFA-same**.

Our **second hypothesis** is that using *different* feedback matrices for each task will effectively update the weight matrices in different directions for each task, thus preventing catastrophic forgetting. This **DFA-diff** approach is inspired by various CL algorithms that explicitly orthogonalise gradients [Zeng et al., 2019, He and Jaeger, 2018, Bennani and Sugiyama, 2020], like Orthogonal Weight Modification (OWM)[Zeng et al., 2019], Conceptor-aided Backprop [He and Jaeger, 2018], and Orthogonal Gradient Descent [Bennani and Sugiyama, 2020]. This idea can also be motivated from a neuroscientific perspective given recent evidence that neural population codes orthogonalize with learning [Flesch et al., 2022, Failor et al., 2021, Zeng et al., 2019].

## 2.3 Methods

### Evaluation metrics

Supervised classifiers require as many samples as possible in order to learn the complex relations of the features that can be used as a model for prediction. The relation between performance and number training examples is usually displayed through the *learning curve*. The learning curve of a machine learning algorithm relates **performance** to **experience**. In this work, we will adopt accuracy as a primary performance measure in all of our experiments with balanced datasets (containing the same amount of examples for every class), but we are aware that the performance of the classifier can be computed using different kind of metrics, for example precision (fraction of positives among examples predicted as positives), recall (fraction of positive samples predicted as positive), F-measure (harmonic mean of precision and recall) or accuracy (fraction of correctly labelled samples among all predictions). Experience is typically measured in number of epochs: every epoch is a full cycle of training in which all the training data has been processed. Different epochs show the data in a different order and multiple iterations are often needed for the algorithms to converge to a solution.

In Continual Learning, performance is difficult to report by one single measure (Performance) because there is the need to monitor performance over more than one dataset at the same time. On one hand a method must generalize to the task on which it is trained, but it is not enough to be considered as a good model in CL, especially if the process of learning the new task erases the ability to perform on previous tasks. On the other hand,

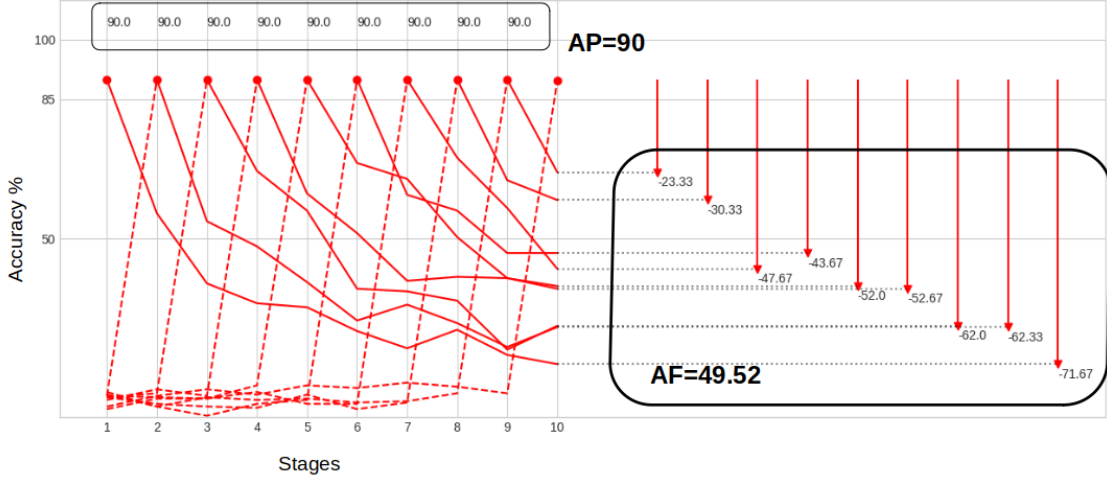


Figure 2.2: Illustration on Average Forgetting (AF) and Average Performance (AP) in the case of back-propagation on the p-FMNIST task in the Task-IL scenario, where the network has a separate head for each task. AP describes the test accuracy of the network, averaged over the tasks, while AF quantifies the average of the difference in accuracy on a given task after training on it and at the end of the whole training trajectory.

if a method like Random Features displays no drop in accuracy on previous datasets, it can still be considered a limited method if it doesn't generalize as well as other methods on single tasks. This trade-off takes the name of plasticity-stability trade-off.

To properly evaluate all methods, we will summarize the performances of the algorithms under study using the following evaluation measures. Consider  $T$  different tasks in succession and  $Acc_{ij}$  the accuracy of task  $j$  at the end of training on task  $i$ .

$$\overline{EA} = \frac{1}{T} \sum_{i=1}^T Acc_{T,i} \qquad \overline{TA} = \frac{1}{T} \sum_{i=1}^T Acc_{i,i}$$

1.  $\overline{TA}$ : Average test accuracy: How accurate is the model on the single tasks?
2.  $\overline{EA}$ : Average end accuracy: How accurate is the model on all the tasks at the end of training?
3.  $100 * (\overline{TA} - \overline{EA}) / \overline{TA}$ : Average loss in accuracy over the average test accuracy. Normalized by the test accuracy and shown as a percentage. This measure can be interpreted as a normalized negative of forgetting.

Other performance measures for CL have been surveyed in [Veniat et al. \[2021\]](#), [Chen et al. \[2023\]](#).

To summarize, some of the plots in this chapter will not display the learning curve of the algorithms, but plots summarizing the final performances in terms of Accuracy and Forgetting.

## Similarity measures

In 2.3, panel B, we analyze the similarity of the datasets arising from the division into the different tasks. We take in the rows all the images of one class in the dataset corresponding to task1 (and their pixels in the columns) and we compare it to the matrix of images of the same class in the next dataset. We average over all the couples of datasets and over the different classes. The similarity measures we apply (described below) measure the similarity from a merely geometrical perspective and we find that in this point of the tasks have classes more similar to each other in the permuted dataset. By construction, the reshuffle of the pixels does not change the mean and the standard deviation of the distributions of the pixels. Nevertheless, the split datasets is easier because it has only two classes and in the Task-IL learning the difference in distribution can be an advantage. In fact, the average forgetting in the Task-IL scenario are overall much smaller than the average forgetting on the permuted dataset. On the other hand, the permuted dataset has only slightly more forgetting in the Domain-IL scenario in the case in which the models are optimized for performance.

we use the following similarity measures:

1. Cosine similarity: Dot product of the matrices reshaped into vectors and subsequently normalized by the L2 norm of the two vectors.
2. Canonical Correlation Analysis (CCA) [Hardoon et al., 2005]: Similarity measure which is invariant to affine transformations (any linear invertible transformation including scaling, rotations, translations).
3. Centered Kernel Alignment (CKA) [Kornblith et al., 2019]: Similarity measure which is invariant to orthogonal transformations, which is a subset of the invertible linear transformations.

CCA is 1 when the two matrices are linearly invariant (they are linked by any affine transformation) and 0 when they are not; CKA is 1 in the case that the two matrices are orthogonally invariant (can be linked by a rotation).

## Data

The goal of each experiment is to measure the evaluation metric of the trained model on the test set. However not all the data available can be used for testing, instead, part of the data must be dedicated to training the network (train set and validation set). In fact, the epochs in the learning curve iterate over the data in the training set; the validation set is

used to test the method during the training procedure alongside with the learning curve to understand how many epochs to use and finally, the test set is used for evaluation of the performance of the experiment.

Classification can be performed on different types of data: audio, images, text, videos, and in general, any digital data that can be gathered. Each element of such data can be treated as a sequence or as a whole. For example, an image can be processed pixel<sup>1</sup> by pixel (sequence) or as a vector containing all pixels. In this chapter, we will use FC networks that takes as input the image as a whole vector.

In our experimental evaluations, we use the classification benchmark datasets of MNIST [Deng, 2012], FashionMNIST (FMNIST) [Xiao et al., 2017] and CIFAR10 [Krizhevsky, 2009].

1. Permuted FMNIST (pFMNIST), where we generate a sequence of learning tasks by permuting the pixels of each image; the idea here is to test a model’s ability to incrementally learn new information with similar statistics [Kemker et al., 2017].
2. Split FMNIST (sFMNIST), where we split the original dataset of 10 classes into five smaller datasets with two disjoint classes for each. The resulting smaller datasets will have different modalities, so a model trained sequentially on them needs to be able to incrementally learn new information with dramatically different statistics.

We show an example of the split and permuted FMNIST in Figure 2.3, panel A. With the plot in panel B of Figure 2.3 we show that the permuted dataset is composed of images with distributional similarity among the tasks. A detailed description of the similarity measures used is in section 2.3 (previous paragraph).

## Architectures

In most of our experiments, we use 3-layer Fully-Connected Networks with 1000 hidden in each hidden layer in our experiments. The choice of using 3 layers is that it is the simplest architecture that allows DFA to express the flexibility of the first layer and the alignment of the second layer at the same time. In Appendix 2.8.3, we show the results of the FC network after pre-trained convolutional layers.

The architecture changes based on the different CL scenarios under consideration. As displayed in Figure 2.3, in the Task-IL scenario, the output layer is not shared for all tasks. In this case, we implement an architecture with separate output layers, each one updated only during the training of the corresponding task and used whenever training/testing on the corresponding task. In the Domain-IL scenario, instead, the last layer is shared among the tasks.

---

<sup>1</sup>Every pixel in an image contains the information of the corresponding picture portion. It is 1-dimensional in grey-scale and 3-dimensional in colored images (Red/Grey/Blue). Every value is between 0 and 1 (white/black in greyscale or indicating the amount of the corresponding color)

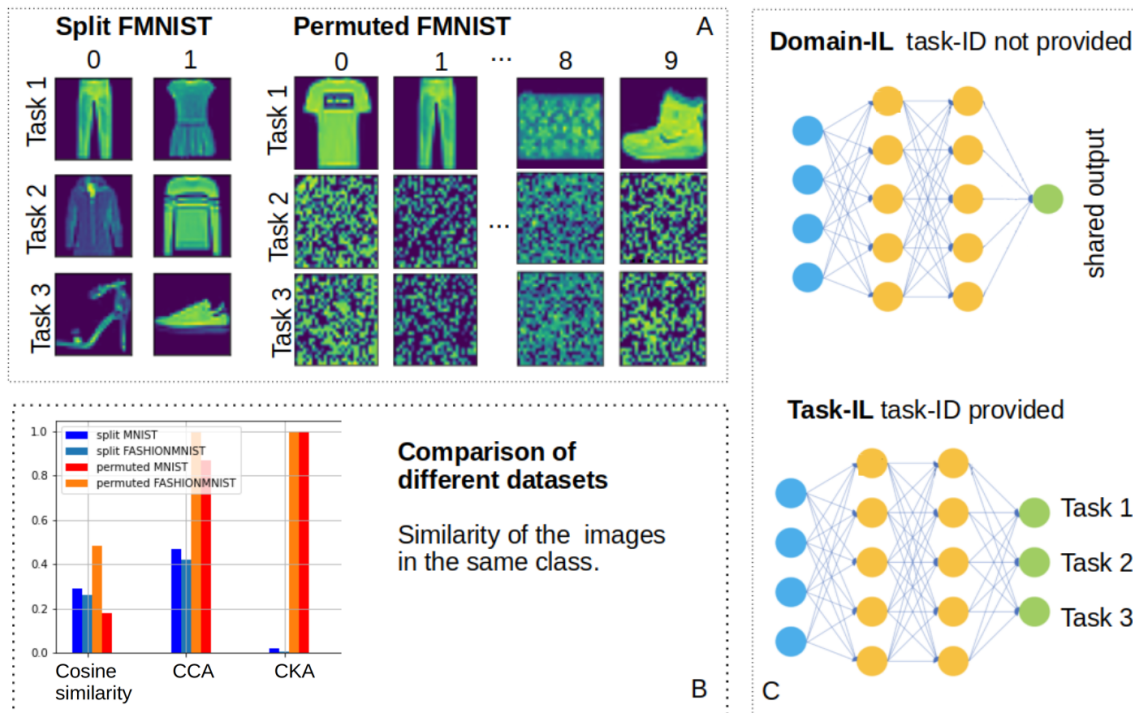


Figure 2.3: **A** One example of the first three tasks in the split (Binary classification) and permuted dataset (classification). **B** Similarity measures of the images in the same class averaged over the different classes of the Split and Permuted datasets. The permuted-FMNIST dataset is the one where the images have the highest similarity among tasks, while split-FMNIST has very different images in the same class. **C** One example of a three-layer architecture used in the Domain-IL and Task-IL scenarios. In the second case, one output node is dedicated to training and evaluating one specific task.

## Algorithms

We used the implementation of DFA contained in the package *tinydfa*<sup>2</sup> [Rebuffi et al., 2017]. We train the networks for 1000 epochs (to ensure DFA has converged), adding an early stopping rule based on the training accuracy (as soon accuracy is larger than 99%, the training is stopped). This choice is taken for convenience and it is related to the qualitative investigation of the success of DFA in CL, so the quantification of the decimals above 99% are superfluous.

All of the layers are initialized using the Xavier uniform initialization [Glorot and Bengio 2010]. When DFA is used, the feedback matrix of each layer (I will refer to the concatenation of all the feedback matrices of the different layers as a single feedback matrix) is

<sup>2</sup>[sdascoli/dfa-dynamics/deep](https://github.com/sdascoli/dfa-dynamics/deep)

initialized with the same distribution as the weights. The variance that the matrix naturally assume with this initialization is of 0.018. In the experiments in which the variance of this matrix will be changed, it will be done simply by dividing the whole matrix by the corresponding factor (0.1, 0.001 for smaller variances and 10, 100 for larger variances). In the case of DFA-same, the feedback matrix is initialized and kept unchanged during the training of all tasks. In DFA-diff, we use a different feedback matrix for each task, by sampling the new ones from the same Uniform distribution used to sampling the first one.

## Baselines and hyper-parameters tuning

To benchmark the performance of DFA, we consider the following baselines:

1. Backpropagation is the most important baseline: we train the same fully-connected networks, adding a Dropout layer [Srivastava et al., 2014] after each layer, which we find helps with generalization, in accordance to the literature [Mirzadeh et al., 2020]. We perform a grid-search optimization for the learning rate in the range between  $1e-2$  and  $1e-4$ .
2. Random Features [Rahimi and Recht, 2008, 2009] are fully-connected networks in which we train only the readout layer with BP. The first and the intermediate layers are kept unchanged during the training phase. This model can be applied only in the Task-IL scenario because it requires a different output layer for each task in the testing phase. The motivation for using this model is to understand the performance of a neural network that does *not* learn data-dependent features, akin to the lazy regime [Chizat et al., 2019]. We use a learning rate of  $1e-2$ .
3. Elastic weight consolidation (EWC) [Kirkpatrick et al., 2017] adds a regularisation term on the loss that penalizes the change of the weights that are more important for previous tasks. It achieves this by pre-multiplying the BP weight updates using the inverse of the diagonal approximation of the Fisher information matrix of the model. In our EWC experiments, we chose a learning rate of  $1e-3$  and an “importance” of 1000, which is another important hyper-parameter for EWC.

## 2.4 Preliminary experiments

### 2.4.1 Degeneracy breaking in different datasets

**Gradient Alignment** and **Weight Alignment** were previously measured on the same dataset, starting from different initialization seeds. Here, we show the overlap of the weights and the gradients among two networks starting from different initialization seed and the same feedback matrix at the end of training on different datasets consecutively. We extend the degeneracy breaking phenomenon to show the overlap of networks initialized at the

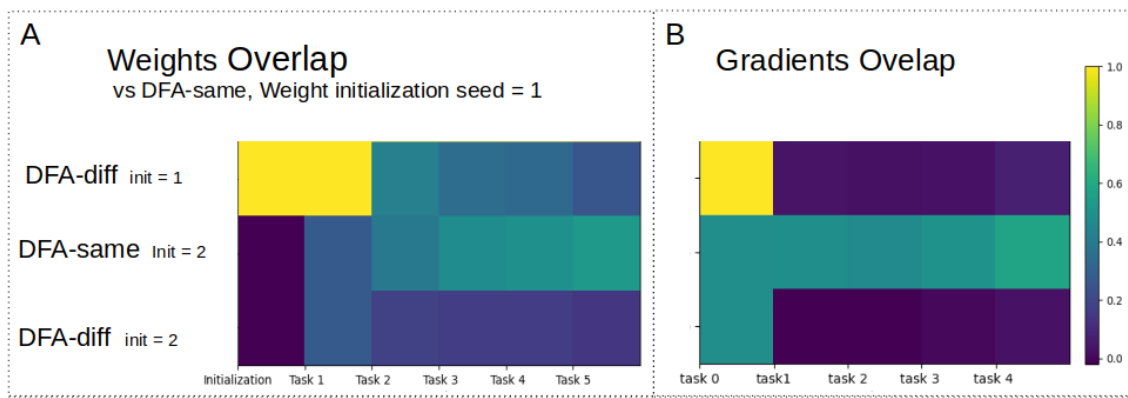


Figure 2.4: **A**: Dot product of the weights at the end of training on the different tasks (averaged over the first two layers, Domain-IL architecture structure) of two networks trained with DFA-same or DFA-diff in contrast with a reference network trained with DFA-same. **Weight Alignment** can be appreciated mostly by looking at the second row, the case in which the networks are initialized from different seeds (seed 1 for the reference network and seed 2 for the network in the second row): the weights at first do not overlap, but thanks to the fact that the networks are trained with the same feedback matrix, the weights overlap more and more. **B**: Overlap of gradients averaged over two different checkpoints during the training on the same dataset (splitFMNIST, Task-IL architecture structure) of two networks trained with DFA-same or DFA-diff in contrast with a reference network trained with DFA-same. **Gradient Alignment** is more visible in the first and the third row: the gradients in panel B become orthogonal (overlap equal to 0) starting from the second task, as soon as the feedback matrix changes, irrespective of the fact that the networks are initialized in the same way (first row) or if the networks are initialized differently (third row).

same initialization point and trained with different feedback matrices after Task 1, where all the networks have the same feedback matrix.

With this experiment, we show that the degeneracy breaking extends to the case in which the dataset is changed, which is essential to our first hypothesis for which DFA-same could alleviate catastrophic forgetting by keeping the weights of the network when training on the second task close to the weights learned on the first task, etc. Let's focus to the second row of Figure 2.4.1 panel A, the case in which the networks are initialized from different seeds: the weights at first do not overlap, but thanks to the fact that the networks are trained with the same feedback matrix, the weights overlap more and more for all of the different tasks.

Our second hypothesis is that the different feedback matrices may bias the dynamics

of the networks in independent directions of the weight landscape for every task. Even though sampling independent and identically distributed (iid) matrices from a uniform distribution does not generally make them orthogonal, we empirically see that the gradients of DFA training procedures with different feedback matrices have gradient overlap which is extremely low (first and third row in panel B).

### 2.4.2 Preliminary experiment: Learning the same dataset with different feedback matrices and different output layers

According to our second hypothesis, we expect that the combination of a dedicated feedback matrix alongside a dedicated output layer can reduce forgetting in the Task-IL scenario.

Let’s focus on the performances of DFA-diff, in which there is a different feedback matrix for every task: in 2.5, we show that in this setting the representations of one dataset are learned in different manifolds.

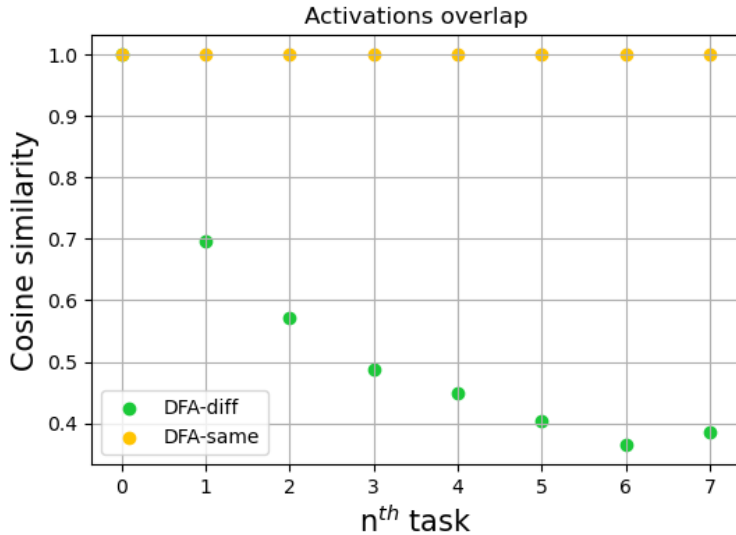


Figure 2.5: Cosine similarity of the change in activations due to the  $n^{\text{th}}$  training vs the change since the first initialization. Values Averaged over the first two layers of the network and the images of the test dataset. The network is trained repeatedly on the same dataset (FMNIST) while the output layer is re-initialized at the beginning of every session. The overall change in activation in DFA-same is always aligned with the change taking place in every single training, while in DFA-diff it is less and less aligned with the changes due to subsequent trainings.



## 2.5 Empirical results

### 2.5.1 DFA—same between the two ends of the plasticity-stability trade-off

Catastrophic forgetting arises in Backpropagation [Rumelhart et al., 1986] due to the fact that the weights of the network are updated in the direction of the minimum of the loss, irrespective of the previous tasks. In this case the network is flexible to fit the task at hand, but forgetting is high. Random features are at the other extreme, since only the weights in the output layer are trained on the task it results in a more rigid method that retains previous performances. BP and random features therefore delineate the two extremes between which we would like to interpolate: we would like DFA to be less susceptible to catastrophic forgetting than BP, while being able to learn data-dependent features that allow it to beat the performance of random features. We report the results of our experiments that we obtain by optimizing the hyper-parameters for maximum AP or for minimum AF in 2.2.

When we train DFA and BP focusing on the performance of the single tasks, we can see that DFA can reach the same performance of 100% accuracy as BP on the split dataset (where the tasks are binary classifications) and almost reach it on the permuted dataset (where the tasks consist of 10-class classifications), where DFA achieves  $88.2 \pm 0.1$ <sup>3</sup> and BP  $89.7 \pm 0.1$ . Thus, DFA proves to be able to adapt the network to learn the new tasks efficiently.

Regarding the comparison of Random Features and DFA with minimized forgetting, we find that DFA can achieve a better Average Performance than a Random Feature model, both in the split and permuted datasets, while maintaining zero forgetting, just as RF. Specifically, DFA shows AP at least 2% higher than RF (see 2.2, column “Minimizing forgetting”). One example is shown in 2.7, where DFA reaches 83% accuracy while Random Features reaches 80.9%. The fact that DFA reaches higher performances than RF means that the weights of the first two layers are updated, and the null forgetting values DFA means that this weights update can happen in such a way that the information of the previous tasks is conserved.

Overall, we find that DFA can embody different behaviors according to the value of its hyper-parameters. It can almost reach the same generalization performances as BP while displaying less forgetting in the majority of the cases; and it can achieve zero forgetting like RF, while generalizing *better* than RF. It thus occupies a sweet spot between the plastic BP and the static RF.

---

<sup>3</sup>We notice that letting DFA train for more epochs, it can achieve the same level as BP, but we decided to keep a maximum of 1000 epochs for all methods.

### 2.5.2 DFA–same in the Domain–IL and Task–IL scenario, testing the first hypothesis

In terms of forgetting, DFA-same achieves 10% less AF than BP in permuted and 2% less in split (binary classification). This trend is conserved when the experiments are performed with the MNIST datasets (see Section 2.8.1). The advantage of DFA in the Domain-IL scenario is in accordance to the hypothesis for which DFA learns close representations for different tasks. In fact, the Domain-IL has a unique output layer shared among all the tasks and this requires similar representations among different tasks in order to correctly classify the previous ones.

In the Task-IL scenario, the average forgetting of BP improves with respect to the Domain-IL scenario by 8% and 22% for the Split and Permuted FMNIST datasets respectively. On the other hand, DFA improves only by 6% on the split dataset and becomes worse, with AF from 45.8% to 50.1%, in the permuted-FMNIST. This trend causes DFA to perform worse than BP on the split dataset and comparably to BP on the permuted dataset<sup>4</sup>. The accuracy trend of DFA can be explained in light of our hypothesis: if the output layer is specialized for every task, keeping the representations similar to each other can bring a disadvantage, as the previous representations are “overwritten”.

### 2.5.3 DFA–diff in Task–IL and Domain-IL scenarios, testing the second hypothesis

We showed that the standard DFA cannot benefit from the introduction of a specialized output layer for every task (Task-IL scenario). We propose an extension of DFA in which there is a different feedback matrix for every task: in 2.5, we show that in this setting the representations of one dataset are learned in different manifolds. According to our hypothesis, this phenomenon can be extended to the learning of different datasets and this would allow to learn new tasks with less interference with previous representations. We expect that the combination of a dedicated feedback matrix alongside a dedicated output layer can reduce forgetting in the Task-IL scenario. In the remaining, we will denote the standard DFA as DFA-same and this extension as DFA-diff.

We find that the main differences with respect to DFA-same are in the Average Forgetting measure and are in accordance with the hypothesis: when optimized for performance and in the Task-IL scenario, DFA-diff has  $9.8\% \pm 2\%$  AF. This value is around 20% lower than the one of DFA-same and 5% lower than BP’s. In the Domain-IL scenario, on the other hand, DFA-diff has at least 8% more forgetting than BP and 10% more than DFA-same. When optimized for minimal forgetting, DFA-diff also achieves null forgetting in the

---

<sup>4</sup>The comparable value of AF between DFA and BP in the Task-IL permuted case hinders the advantage of DFA for the forgetting of the earliest tasks. We show in 2.7 the trend of forgetting the first task and we can notice a clear advantage. In 2.6.2 we report the forgetting trends of all the tasks, where one can see that DFA forgets more tasks seen one or two epochs before, but is more stable for earlier ones.

Table 2.1: Results in terms of Average Performance (AP) and Average Forgetting (AF) in the Domain-IL for all the methods (DFA-diff, DFA-same, BP, BP-ablated, EWC). In the first column, the networks are optimized for maximum performances, and in the second column for minimum forgetting. We present the results visually in Figure 2.6. In the case of minimized forgetting, we report here the minimum AP % values above the RF baseline.

		AP	AF	AP	AF
		Optimized for performance		Minimizing forgetting	
Split FMNIST Domain-IL	DFA-diff	100 ± 0	45.8 ± 2.4	94.0 ± 0	35.56 ± 1.6
	DFA-same	100 ± 0	35.5 ± 0.2	98.2 ± 0	32.1 ± 1.6
	BP	100 ± 0	37.8 ± 0.4	94.5 ± 0.1	35.8 ± 0.6
	BP ablated	100 ± 0	40.5 ± 1.7	98.4 ± 0.1	34.7 ± 0.7
	EWC	100 ± 0	46.9 ± 4.8	98.7 ± 0.4	40.4 ± 3.1
Permuted FMNIST Domain-IL	DFA-diff	88.3 ± 0	71.8 ± 1.2	82.3 ± 0.1	29.1 ± 0.9
	DFA-same	88.2 ± 0.1	45.8 ± 1.9	85.8 ± 0	26.2 ± 1.9
	BP	89.7 ± 0.1	57.6 ± 1.1	85.4 ± 0.1	23.1 ± 1.5
	BP ablated	88.7 ± 0.1	49.4 ± 0.4	84 ± 0.1	26.2 ± 2.4
	EWC	80.7 ± 0.2	55.7 ± 2.5	80.7 ± 0.2	55.7 ± 2.5

Task-IL scenario, but it shows lower performances than DFA-same (up to 4.2% less than DFA-same in the split Domain-IL).

## 2.6 Complementary results

### 2.6.1 Comparing DFA with Elastic weight consolidation

In the Domain-IL scenario, there is an advantage of DFA-same over EWC of about 10% AF. This could be due to the underlying hypothesis that in DFA-same the implicit regularisation given by the feedback matrix is kept constant. On the other hand, the regularisation factor in EWC is different for every new task, as explained in 2.1. On the permuted dataset, EWC has a performance that is intermediate between DFA-same and DFA-diff. In fact, EWC can be considered a method in between DFA-same and DFA-diff because the “pulling forces” in the loss due to the feedback matrices will not be parallel to the ones of the previous tasks as in DFA-same, but not orthogonal as in DFA-diff, either. On the split datasets, on the other hand, EWC tends to be better than DFA-diff. In the Task-IL scenario EWC is the best method both when we evaluate on MNIST (see 2.8.1) and FMNIST. In the Domain-IL it

Table 2.2: Results in Task-IL scenario in terms of Average Performance (AP) and Average Forgetting (AF) in the different datasets for all the methods (DFA-diff, DFA-same, BP, BP-ablated, EWC, RF ). In the first column, the networks are optimized for maximum performances and in the second column for minimum forgetting. We present these results visually in Figure 2.6. In the case of minimized forgetting, we report here the minimum AP % values above the RF baseline.

		AP	AF	AP	AF
		Optimized for performance		Minimizing forgetting	
Split FMNIST Task-IL	DFA-diff	100 $\pm$ 0	9.8 $\pm$ 2	93.9 $\pm$ 0.1	0 $\pm$ 0
	DFA-same	100 $\pm$ 0	29.4 $\pm$ 1.2	95.4 $\pm$ 0	0 $\pm$ 0
	BP	100 $\pm$ 0	15 $\pm$ 2.3	89.5 $\pm$ 0.1	0.3 $\pm$ 0.3
	BP ablated	100 $\pm$ 0	9.7 $\pm$ 1.6	94 $\pm$ 0	0 $\pm$ 0
	EWC	99.2 $\pm$ 0.2	4.5 $\pm$ 1.5	99.1 $\pm$ 0.2	3.4 $\pm$ 1.8
	RF	93.4 $\pm$ 0.1	0 $\pm$ 0	93.4 $\pm$ 0.1	0 $\pm$ 0
Permuted FMNIST Task-IL	DFA-diff	88.5 $\pm$ 0.1	44.5 $\pm$ 2.4	83 $\pm$ 0	0 $\pm$ 0
	DFA-same	88.2 $\pm$ 0.1	50.1 $\pm$ 2.5	83 $\pm$ 0	0 $\pm$ 0
	BP	90.0 $\pm$ 0.1	49.5 $\pm$ 2.5	83 $\pm$ 0	11.8 $\pm$ 0.8
	BP ablated	90 $\pm$ 0.1	47 $\pm$ 2	82.8 $\pm$ 0.1	0 $\pm$ 0
	EWC	88.7 $\pm$ 0.1	46.5 $\pm$ 2.7	83 $\pm$ 0	8.7 $\pm$ 1.2
	RF	80.9 $\pm$ 0.1	0 $\pm$ 0	80.9 $\pm$ 0.1	0 $\pm$ 0

is the best only in the experiments on MNIST. This might be due to a difficulty in finding the correct hyper-parameters that would render EWC performative on FMNIST.

### 2.6.2 Ablation experiments

In order to investigate the reasons behind the success of DFA in some of the CL scenarios, we look for the underlying mechanism that influences DFA’s stability to forgetting the most. For achieving minimum AF, we performed a grid-search over the hyper-parameters and we find that forgetting is reduced with smaller variance of the feedback matrix for both DFA-same and DFA-diff, see 2.6. This impact can be easily explained by observing that the entries of the Feedback matrix are a multiplicative factor in the update rule of the layers 1 and 2 (see equation 1.7); thus the smaller the values in the matrix, the smaller will be the update of the weights of these two layers. The output layer is updated exactly as in BP, so this layer is not affected by the variance of the Feedback matrix. In the limit of a matrix filled with zeroes, DFA approaches a Random Feature behaviour where only

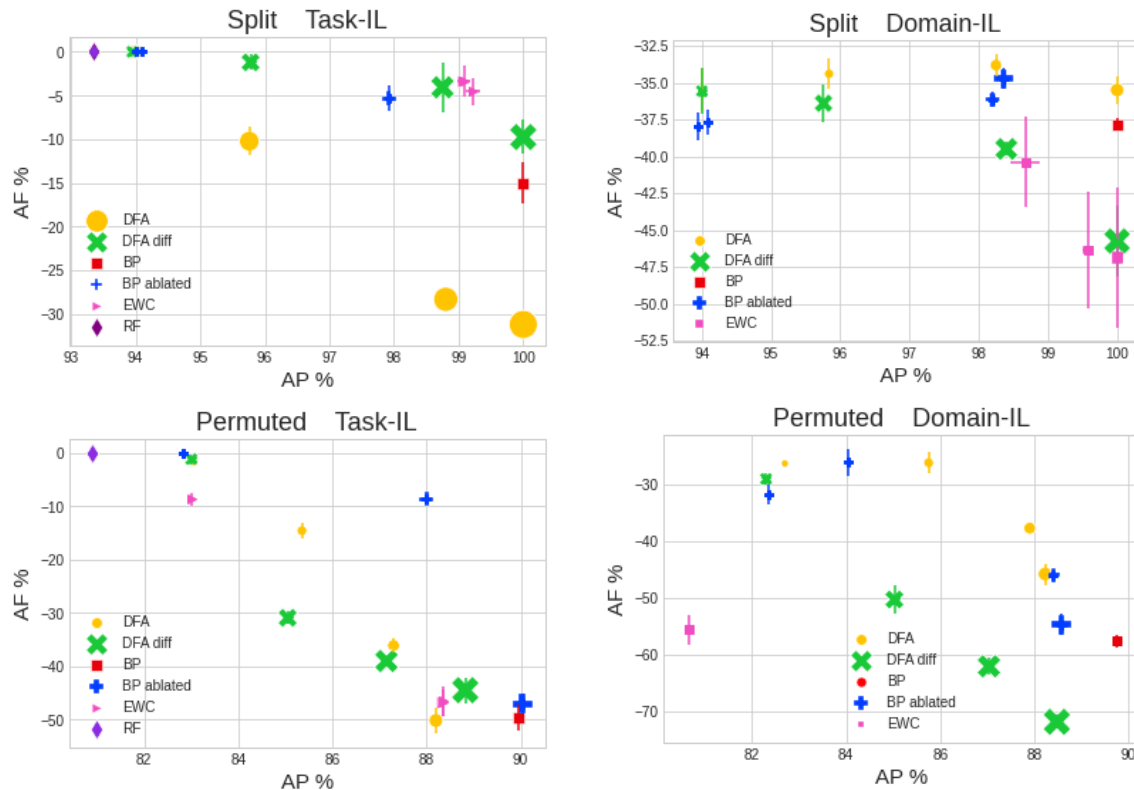


Figure 2.6: AF-AP plots for all datasets (by rows) and scenarios (by columns). In each plot, the increasing size of the marker in DFA and DFA-diff indicates an increase in the variance of the feedback matrix (from  $1.8 \times 10^{-3}$  to 1.8) and for BP an increase of the learning rate of the first two layers (from  $10^{-8}$  to  $10^{-3}$ ); the learning rate of the third layer is kept to 0.01 to match the value used in DFA. We can notice that for larger values of the feedback matrix variance, DFA can reach higher AP at the price of smaller AF (smaller dots moving towards the top-left part of the plots, which culminates in the RF regime). The same trend is followed by BP when the learning rate of the first two layers decreases (BP ablated).

the readout layer is updated.

We therefore performed an ablation experiment with BP where we reduced the learning rate of BP in the first two layers to mimic the effect of feedback matrices with reduced variance, while keeping the learning rate of the readout layer fixed and similar to the one used for DFA. We show the results of the ablated experiment in all the datasets in 2.6. We found that lowering the learning rate of the intermediate layer indeed brings BP towards lower AF and lower AP, similarly to the effect of lowering the variance in DFA’s feedback

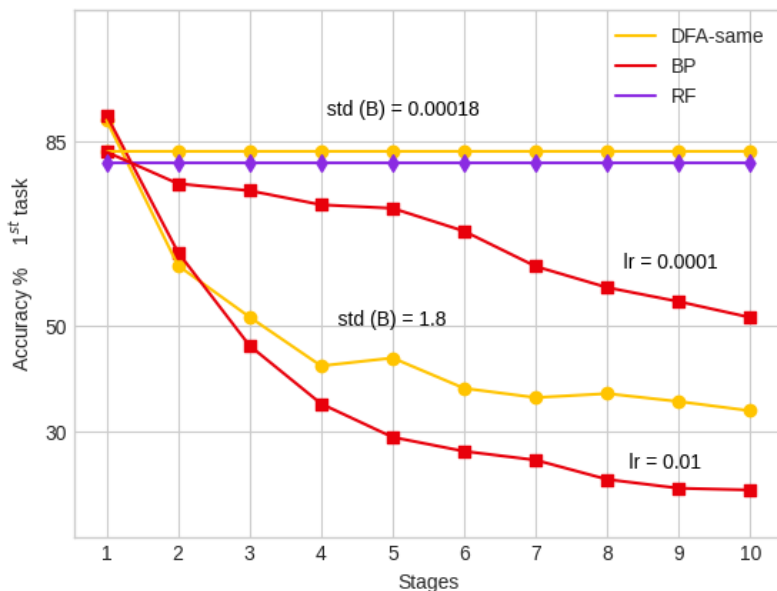


Figure 2.7: Performance of the first task throughout learning all tasks evaluated on the permuted-FMNIST (above) in the Task-IL scenario. DFA-same is more powerful than RF (purple line). Compared to BP, DFA-same has less forgetting in both cases: when we optimize the models for maximum performance and for least forgetting.

matrix. In fact, BP with reduced learning rates can surpass DFA’s stability in some cases, for example in the Task-IL, permuted case. In all the other cases, BP-ablated can reach AF similar to DFA, underlining the importance of variance of the feedback matrix for the stability of DFA. Crucially, DFA still retains higher accuracy for old tasks in the long run: As we show in appendix 2.6.2, figure 2.8, the accuracy drop for the case of Task-IL p-FMNIST in DFA is bounded to a maximum of 59% while BP-ablated reaches 67% forgetting.

### Comparison between DFA and BP ablated at increasing number of tasks

In order to answer the question, ”Is DFA mitigation for catastrophic forgetting solely due to the small learning rate in the first two layers?” we propose a deeper comparison with BP ablated in the context of the permuted dataset. The same results apply to the split dataset. In 2.8 below, we display the accuracy of all the tasks during the CL pipeline (training one task at a time and always monitoring the accuracy of the other tasks; in the case of Task-IL scenario, either for training and for testing one task, the corresponding

output layer is used). We compare the case in which BP ablated has the most similar AF to DFA-same and we find that DFA has the tendency of forgetting more than BP and BP ablated after few stages but is more stable in the long run, for example after 6 tasks in the Task-IL and after 1 or 9 tasks in the Domain-IL scenario; the black and the grey lines in the plot of DFA-same visually show the point where the advantage starts. We conclude that a small learning rate in the first two layers is not always enough to reproduce DFA-same stability and this advantage in the long run might be due to the alignment of the weights.

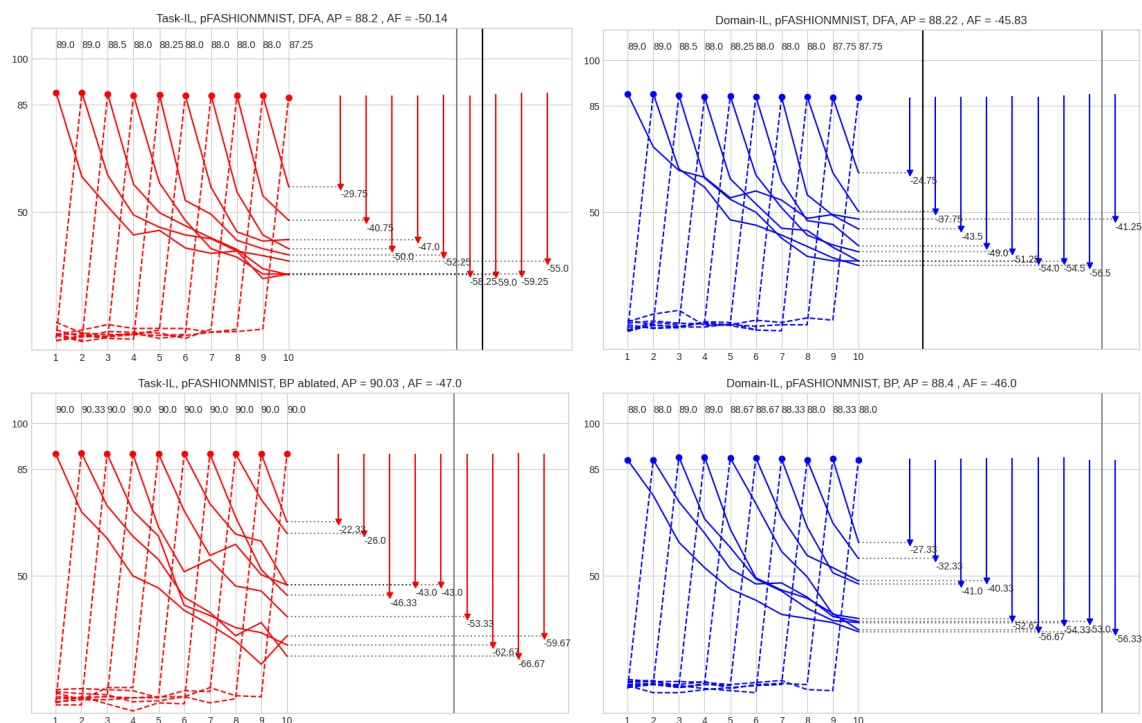


Figure 2.8: Accuracy interpolation lines along the stages. On the left (red lines) we show the results for the Task-IL scenario, we can notice that even though DFA-same has higher average forgetting than BP and BP-ablated, it has lower forgetting for the first three (vs BP, darker line) or four tasks (vs BP ablated): DFA-same retains the accuracy better than BP and BP-ablated in the long run, while it forgets more in the shorter run. This is also true in the Domain-IL scenario (blue lines, on the right) in which DFA has better average forgetting than BP and worse compared to BP-ablated.

## 2.7 Summary of results and discussion

By comparing the performance of DFA and various baseline algorithms on several benchmarks, we found that DFA has the potential for continual learning. For example, we found that **DFA-same** can alleviate catastrophic forgetting better than BP and EWC in the Domain-IL scenario. This result is consistent with the first hypothesis, whereby DFA-same imposes weight alignment in the presence of different datasets. We saw that the networks are indeed able to exploit this to mitigate catastrophic forgetting. According to our hypothesis, DFA-same works as an implicitly regularised method like EWC, but with the advantage of keeping the same constraints on the weights, while EWC adjusts the constraint of the weights after every new dataset encountered.

The empirical evaluation also shows that **DFA with different feedback matrices** for each task in the curriculum has an advantage with respect to EWC and BP in the Task-IL scenario. This is the architectural scenario in which the output layer is specific for every task. This allows learning the features on different manifolds in weight space, while still permitting an efficient encoding of the features for a successful classification of the different tasks. In continual learning, when the features are learned on different manifolds, the learning of the new tasks does not interfere with the separating hyperplane of the previous tasks, so it can be an advantage in the Task-IL scenario. The drastic improvements of DFA-diff in the Task-IL scenario in contrast to the minor improvement of DFA-same in this setting corroborates our second hypothesis, whereby gradient alignment extends to the case of different datasets, allowing DFA-diff to learn in different manifolds. This approach can thus be employed for an effective mitigation to catastrophic forgetting. Finally, we tested gradient alignment in two specific experiments, and we find it visible in the case in which DFA is trained on the same series of datasets starting from different initialisations with the same feedback matrices (2.4.1, panel C) and when DFA is trained on the same dataset repeatedly but with different Feedback Matrices (Figure 2.5).

While backpropagation with layer-specific learning rates in the first two layers (**BP ablated**) has a significantly lower overall forgetting than DFA-same in the case of Task-IL and permuted dataset, DFA-same retained its advantage with respect to BP and BP ablated for the tasks that were processed at the beginning (see 2.6.2 and 2.7). The design of BP ablated itself was inspired by the impact of the scale of DFA’s feedback matrix on forgetting. The advantage of DFA-same over this method is further evidence that weight alignment is taking place and is beneficial against catastrophic forgetting beyond the impact of the Feedback matrix on the learning rate.

In the future, it will be interesting to investigate how the architecture of the networks, and specifically the width and the depth of the networks influence the plasticity-stability trade off curves. A preliminary experiment with 5-layers architectures can be found in the Appendix 2.8.4.

In the case of the split dataset, the forgetting is heavily dependent on the similarity of consecutive tasks (see larger error bars in 2.6). On top of curriculum learning, it might



also be beneficial to use convolutional layers before the Fully-Connected part to replace the highly-variable distribution of pixels with more rationalized features [Crafton et al., 2019]. See appendix 2.8.3 for the preliminary experiments we carried out in this direction, and more detailed results will be subject of our paper review, in the near future. We tested the Continual Learning scenarios where the model is given the task-ID at test time (Task-IL) or not (Domain-IL). The next challenge is the class-incremental learning scenario where the model has to provide the task-ID when solving a task as is the case, for example, in many reinforcement learning scenarios. We made some preliminary tests in this direction and the results are that the performances of DFA in this scenario is poor: the test accuracy drops back to the random accuracy level as soon as the next task is learned, or, if a masked cross entropy is used, the first two learned tasks retain a signal. A detailed report about these results will also be subject of the paper updates.

## 2.8 Appendix B

### 2.8.1 Results evaluated on MNIST

The plots displaying the results of the methods applied to the MNIST datasets in the Task-IL scenario can be found in 2.9. The points in the circle indicates the Split-MNIST, otherwise they refer to the results on the Permuted-MNIST.

We can notice that the results are consistent with the results on F-MNIST: In the Task-IL split-MNIST DFA lays in an intermediate position between RF and BP, slightly behind EWC; In the permuted Task-IL, DFA-diff has less forgetting than all the other methods while achieving the same AP of BP. In Domain-IL (panel on the right), DFA-same is comparable or equal in AF to BP while DFA-diff has the worst performances; EWC is in an intermediate level between DFA-same and DFA-diff on the permuted dataset while is superior to DFA on the split datasets.

### 2.8.2 Random Seed impact on the accuracy of the firstly learned task

We show here the Test Accuracy of the first learned task for all methods (Accuracy % on the y-axis and stage number on the x-axis). The different figures display one method at a time in the following order: DFA-same, DFA-diff, BP-ablated, BP, EWC; permuted FMNIST above and split FMNIST below. The dotted lines are the ones optimized for minimizing forgetting. In EWC permuted FMNIST, the blue dotted line coincides with the blue solid line. These plots reveal that there is no significant difference in the impact of the random initialization among the methods. On top of this, we decided to report them to offer a visual illustration of forgetting of the first task in all methods.

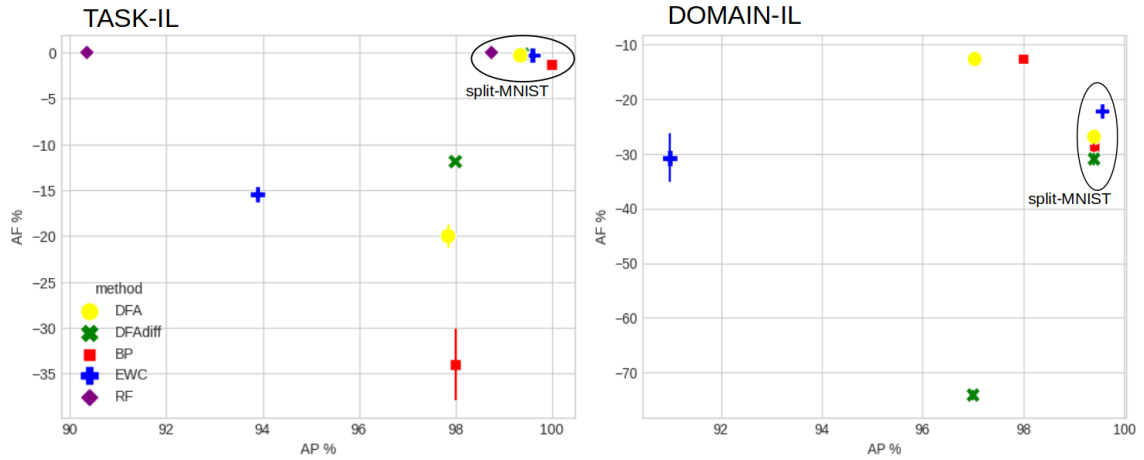


Figure 2.9: AP and AF of the networks trained on MNIST. In the circle, the results of split-MNIST and the other points are permuted-MNIST.

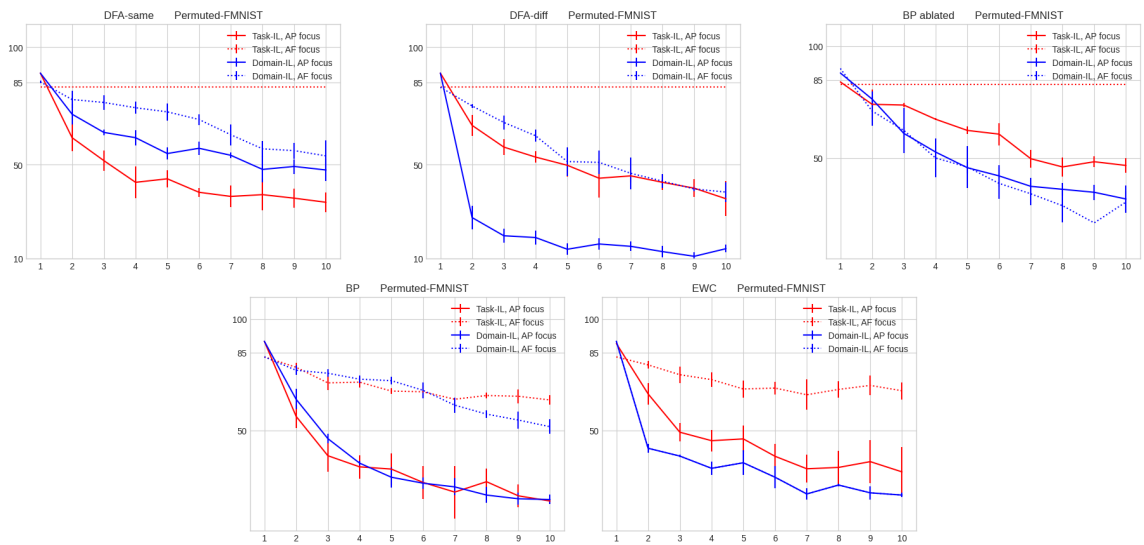


Figure 2.10: Test Accuracy of the first task for all methods through the stages of the Permuted case. The errorbars display the standard deviation over 5 different seeds initialization.

### 2.8.3 DFA performances after convolutional layers

One curiosity that might be left at this point is about how much DFA scales to complex architectures and datasets, for example with convolutional architectures or CIFAR10

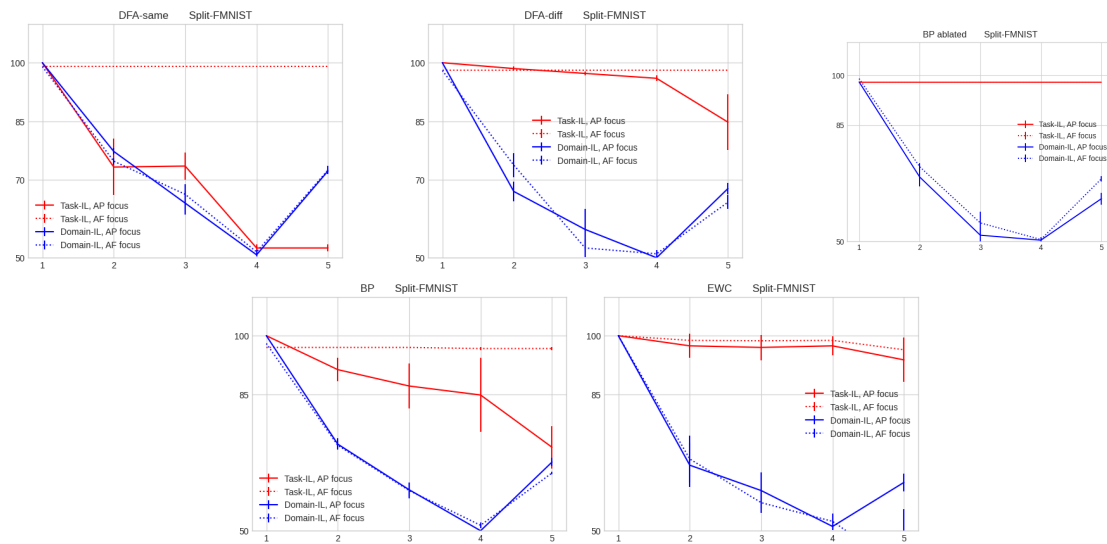


Figure 2.11: Test Accuracy of the first task for all methods through the stages of the Split case. The errorbars display the standard deviation over 5 different seeds initialization.

benchmark dataset (An example of CIFAR10 input images are in Figure 2.12).

**Convolutional networks** take as input the whole image, but the inherent property of convolutional layers is to divide it in sub-sections and process them independently. In this way, the information regarding which pixel is closer to which other is preserved. They are inspired by the ventral pathway of the visual cortex of the brains, that processes information in stages. This hierarchical structure allows each layer of a CNN to extract increasingly complex features from the input data. The initial layers detect simple features like edges, while deeper layers combine these features to recognize more complex patterns and objects [Celeghin et al. \[2023\]](#). Thus, convolutional layers are complex architectures that are specialized for image classification.

As we discussed in the introduction, DFA fails at training convolutional layers, so we will adopt 3 layers of DFA after convolutional layers pre-trained with BP. Since these experiments are setting the ground to further experiments in Continual Learning, we include the case in which the pre-trained convolutional layers are general ones and do not contain information about the specific dataset. We attempt to achieve this using convolutional layers pre-trained on Imagenet and transferred for the classification of CIFAR10. These experiments are preliminary results that aim to reveal the qualitative behaviour of DFA in this context.

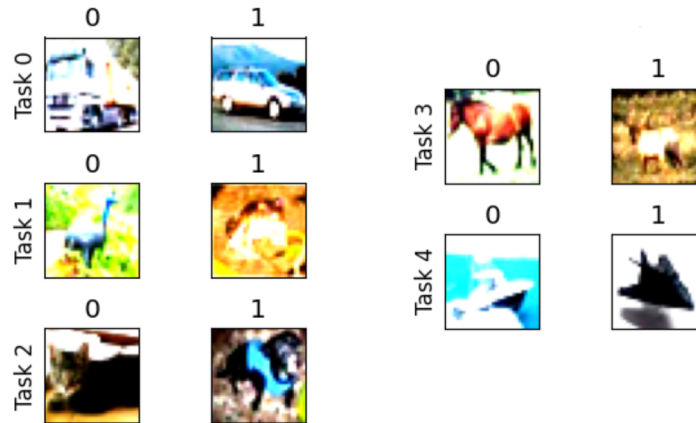


Figure 2.12: Example of the CIFAR10 dataset divided into 5 binary classification tasks.

### Methodological setting

The convolutional architecture used is a VGG11 architecture which is a relatively small convolutional network composed of: 8 convolutional layers ( 3x3 filters with stride 1 and padding 1) followed by Batch Normalization, ReLU activations and Max-pooling ( 2x2 filters and stride 2). The convolutional part is followed by two FC layers (4096 hidden size, dropout 0.5) and one Softmax layer (for a total of 3-layer FC network). The FC network used when DFA is used has a hidden size of 1096 and 4 layers without dropout. These choices are made arbitrarily, but the analysis of what happens with other architectures is outside the scope of this experiments.

### 10-class classification on CIFAR 10

Figure 2.8.3 compares the Test Accuracy of BP (left panel) and DFA (right panel) in the case of a **10-class classification** of CIFAR10 benchmark dataset after fixed pre-trained convolutional layers.

If we only train the fully-connected classifier (blue line) directly on the images, you achieve  $\sim 43\%$  accuracy in BP and  $\sim 55\%$  with DFA. Training the fully-connected classifier on top of **fixed** convolutional filters that were pre-trained on ImageNet, one has up to  $\sim 63\%$  with BP and random guess with DFA (orange). DFA and BP reach the same accuracy if DFA is used on top of convolutional layers pre-trained on CIFAR10 and if BP is trained from scratch  $\sim 85\%$  (green). These results are interesting because they reveal that DFA is not flexible enough to use the features extracted by a general image classification convolutional architecture to fit 10 classes.

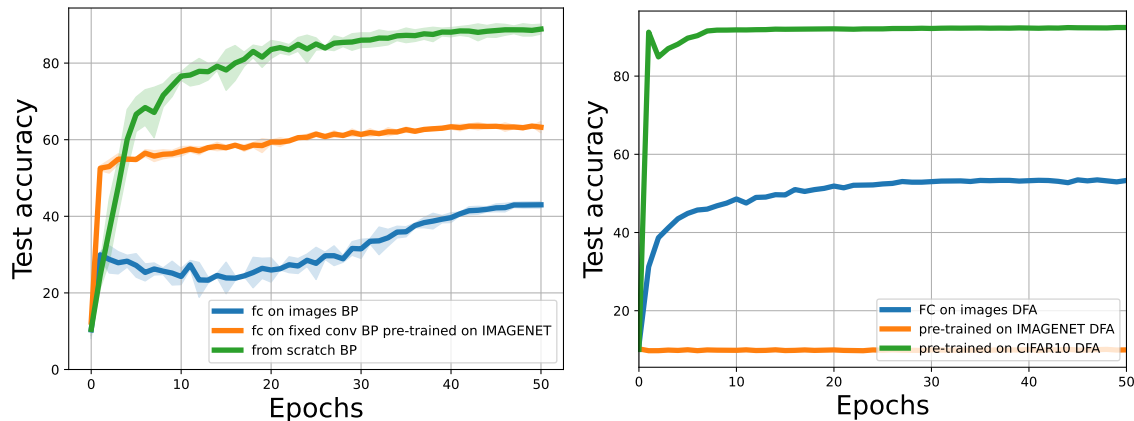


Figure 2.13: **10-class classification of CIFAR10** the shadow on the left panel is one standard deviation over three runs with parameters  $lr=0.01$ ,  $mom=0.9$  and  $wd = 0.0001$ ; on the right there is one seed with  $mom=0.9$ ,  $lr=1e-05$ ,  $wd=0.0001$ . The hidden size is 1096 when is applied directly on the images and 4096 after convolutional pre-trained layers.

### 10-class classification without learning the output layer

This experiment is characterized by the fact that the softmax layer at **the end of the FC network is not trained**. This doesn't allow DFA to align its last-layer weights, and thus the WA and GA don't take place, leading to DFA to be stuck in a training trajectory that focuses on the WA of the updated layers, without any connection with learning (dashed and dotted lines in Figure 2.14).

While BP (solid line) updates the inner layers of the FC network in a meaningful way. The result of this experiment is making visible that the **pre-training on ImageNet** offers DFA with features for which learning starts at 10%, which is the Random Guess Accuracy. When pre-training on a different dataset, the model has learned more general feature representations that aren't specific to the target dataset. Therefore, when transferred, the model starts with some general knowledge, leading to random but non-zero accuracy at the start. Erhan et al. [2010].

Instead, pre-training on CIFAR10 (the same dataset used in the experiment), the Test Accuracy of DFA is stuck at 0%. This can be explained by the opposite argument: the convolutional layers have highly become specialized for that dataset. During fine-tuning, the weights are already adapted to the target data distribution, so the model's initial predictions might be incorrect for the new task (here, there are three layers of random projections mixing the features), causing the initial accuracy to drop to zero. This experiment is useful to prove that using pre-trained convolutional layers on IMAGENET does not carry dataset-specific information about CIFAR10, thus they can be used for the Continual Learning evaluation of DFA on CIFAR10.

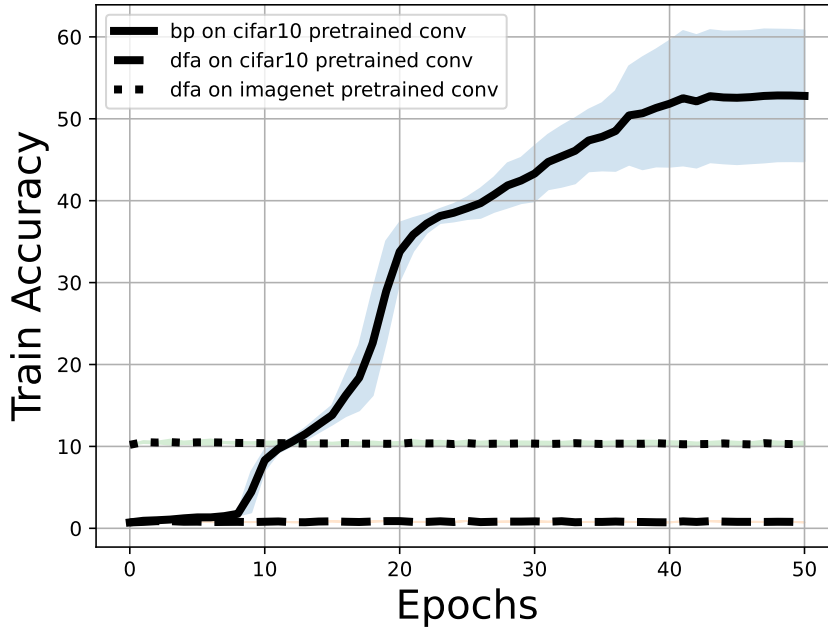


Figure 2.14: Experiment of the pre-trained fixed convolutional layer followed by 3 FC layers and one fixed softmax layer. The shadow is one standard deviation of models trained with hidden size of 4096 and different lr, momentum, batch size computed in grid combinations among the values of  $lr=[0.1,0.01,0.001]$ ;  $wd = [0, 1e-5]$ ,  $mom = [0,0.5]$ ; batch size  $=[30,512]$ .

## 2-class classification

For the interests of Continual Learning experiments, though, when the full dataset is divided into 5 tasks, the classification task simplifies to a **binary classification**.

In this context, as we see in Figure 2.15, if we only train the fully-connected classifier (cyan) directly on the images, DFA achieves  $\sim 90\%$  with DFA. If the data is turned into grey-scale, this goes down to  $80\%$ . Training the fully-connected classifier on top of **fixed** convolutional filters that were pre-trained on ImageNet, one has up to  $\sim 98\%$  with DFA (grey). DFA reaches the same accuracy when is used on top of convolutional layers pre-trained on CIFAR10 (yellow).

This experiment shows that DFA is capable of using the features extracted by the pre-trained convolutional layers on ImageNet for distinguishing only 2 classes among the CIFAR10 classes. The behaviour of the FC net directly on the images (cyan line) also exhibits non trivial performances. With the additional test of evaluating this case on the gray-scale version of the dataset made the accuracy drop slightly because the task becomes harder. This might be useful in Continual Learning benchmarks in which one wants to test

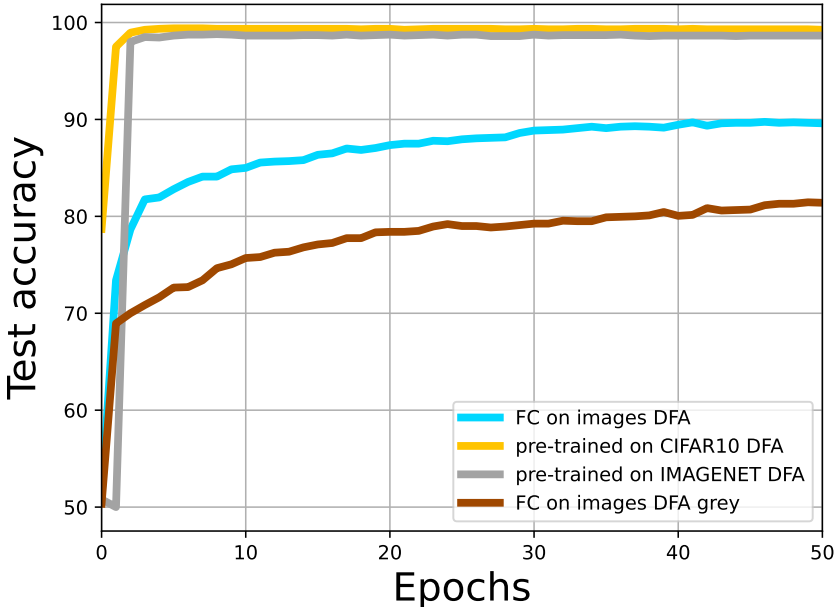


Figure 2.15: **2-class classification of CIFAR10** with hyperparameters [lr=0.1, momentum=0.9, wd=0.0001, hidden size = 1096, batch size=512].

different models on challenging datasets, in order to distinguish clearly the power of the different methods.

### Summary

We conclude that the performances of a 3-layer FC trained with DFA are enough for an evaluation of this network on the Split benchmark dataset of Continual Learning, which is composed by 5 tasks of binary classification. Moreover, using convolutional fixed layers pre-trained on Imagenet as a pre-processing of the dataset is another feasible experiment that do not violate the principle for which the task has to be a novelty. These experiments will be subject of the paper updates, publicly available in OpenReview. Moreover, it would be interesting to understand what are the the performances of DFA **between** the classifications of **2 and 10 classes**, and what precisely influences the transition.

This Appendix was dedicated to describe some preliminary results on the application of DFA to complex architectures (convolutional networks) and datasets (CIFAR10). The aim is to understand if DFA can be used after pre-trained and fixed convolutional layers (one can think of these pre-trained fixed layers as a transformation to the dataset). Ideally, we would like that the features offered by this transformation were not specific about the data distribution of CIFAR10, otherwise the principle of Continual Learning for which the

tasks are novel data distribution would have been violated.

### 2.8.4 DFA with larger architectures

With this experiment we show what is the impact on the Test Accuracy of DFA with a larger architecture like a 5-layer Fully Connected with respect to a 3-layer one that we chose to use throughout the work. We stopped the training of the baselines to 150 epochs and we report the Test Accuracy of all models in the legend of Figure 2.16. For DFA, which is in our focus, we ran for 1500 epochs. The first observation is that DFA with 5 layers has a first plateau at around 50 epochs, and another one at 120. After that, the Test Accuracy increases slowly. This pattern is not present in the 3-Layer network, which follows the Random Feature performances until 100 epochs and afterwards detaches from it and continues to increase.

The behaviour of the 5-layer architecture can be explained in light of the alignment phases necessary for the algorithm to learn. The longer the architecture, the more time it requires for all the layers to achieve gradient alignment.

In our experiments we chose the 3-layer architecture for its computational and performance superiority, but in future perspectives it would be interesting to investigate how the 5-layer network in the different stages of convergence influences Continual learning.

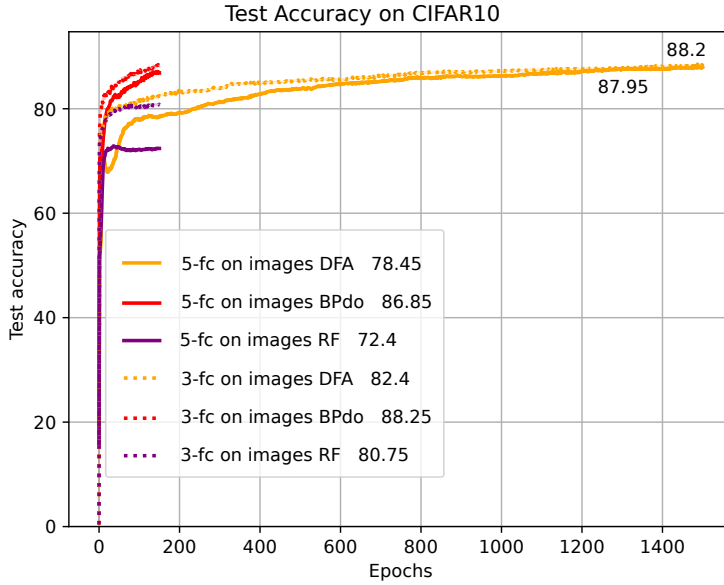


Figure 2.16: **3- and 5- layers FC DFA of CIFAR10** with hyperparameters as follows: lr = 0.01, wd = 0.0001, momentum = 0.3. Dropout of 0.5 is used only for BP.



## Chapter 3

# DFA for Bayesian neural networks averaging

Bayesian *Neural* Networks focus on modeling uncertainty in neural network predictions by placing distributions over the network’s weights. They must not be confused with Bayesian networks that are focused on modeling the relationships between variables in structured domains. Bayesian *Neural* Networks allow to quantify uncertainty in predictions. They are applied in contexts where the goal is to model uncertainty in high dimensional unstructured data, for which Neural Networks are suitable.

In general, parameter estimation can be approached from a Bayesian approach or via Maximum Likelihood. The first approach assumes parameters  $\theta$  to be random variable with some known prior distribution. The Bayes theorem allows to use the observed examples to obtain a posterior distribution. The predictions for new examples are obtained by integrating the model’s predictions over all possible values for the parameters.

In the Maximum Likelihood approach, that is adopted in the other chapters, the parameters  $\theta$  are considered to have fixed, but unknown values and they are computed as the one maximising the probability of the observed examples  $\mathcal{D}$ . The function that is optimized in the classification tasks of our experiments is the Cross-Entropy. Inference is performed using the obtained values of the parameters to compute probabilities for new examples of belonging to the classes. Then the most probable class is elected as final prediction.

Despite the success of neural networks trained in Maximum Likelihood approach with stochastic gradient descent, there has been continued interest in combining Bayesian methods with neural networks. One important motivation is to alleviate the notorious brittleness of neural networks and their susceptibility to adversarial attacks, which is less pronounced in Bayesian Neural Networks (BNNs) [Gal and Smith, 2018, Bekasov and Murray, 2018]. In this chapter, we will explore the potential of DFA to train neural networks in a Bayesian way. Before discussing the DFA-based approach we introduce in this chapter, we give some background information on Bayesian methods for neural networks in Sec. 3.1. We then for-

mulate the research hypothesis for this chapter in Sec. 3.2; outline the methods in Sec.3.3; present the results in Sec.3.4 and summarize the results and discuss them in Sec.3.7.

### 3.1 Background

In this section, I will clarify the difference between **Bayesian Averaging** and **Ensembling**; then describe some of the methods currently used for obtaining such distributions starting from the dataset and a prior distribution, such as MCMC methods and **Langevin SGD**.

With Bayesian Neural Networks, each weight of the network is not a single value but is considered as a random variable with its own distribution. **Performing inference** for a new example  $x$  involves marginalizing over the posterior, a process referred to as **Bayesian averaging**. The predictive distribution is:

$$p(y|x, D) = \int p(y|x, \theta)p(\theta|D)d\theta, \quad (3.1)$$

where:

1.  $p(y|x, \theta)$  is the likelihood, computed with a forward pass of the input  $x$  through the network with parameters  $\theta$ .
2.  $p(\theta|D)$  is the posterior distribution over the weights. More details on this are in the next section.

Since computing this integral exactly is not possible, one typically approximates it using sampling methods. A common way to approximate this marginalization is to use Monte Carlo sampling from the posterior. Instead of directly computing the integral, this procedure samples several weight configurations  $\theta_1, \theta_2, \dots, \theta_M$  from the posterior. Then, inference is performed by averaging over the predictive distributions from these sampled weights.

The approximate predictive distribution becomes:

$$\hat{p}(y|x, D) \approx \frac{1}{M} \sum_{i=1}^M p(y|x, \theta_i) \quad (3.2)$$

where  $M$  is the number of sampled weight sets and  $\theta_i$  are samples from the posterior distribution. This in practice means output of the network feeding the input  $x$  and using  $\theta_i$  as parameters, then average the predicted probabilities across all the samples. The output layer of the network has a final softmax operation, giving  $y$  as vector normalized to 1 containing the probabilities of the label to belong to the different classes. Finally, to get the predicted class, the class corresponding to the maximum average probability is taken:

$$\hat{y} = \underset{y}{arg \max} \hat{p}(y|x, D) \quad (3.3)$$

At this point, **to quantify how certain or uncertain the model is about the prediction**, one can measure the variance of the predictions across different weight samples.

**Ensembling** is a popular technique to approximate even further the Bayesian Averaging process and it is one of the oldest tricks in machine learning literature [Hansen and Salamon, 1990]. With this approach, instead of sampling configurations of the weights as to approximate the posterior, the samples are obtained by naively training networks with different initialization seed and averaging their predictions during testing. These networks are trained as if they were standard neural networks. There’s no direct Bayesian component during training. However, the variability in their learned weights across the ensemble serves as a proxy for the posterior distribution in a Bayesian setting. By combining the outputs of several models, an ensemble can achieve better performance than any of its members. Like true Bayesian methods, Ensembling provides a way to estimate uncertainty.

For what regards true Bayesian Neural Networks, the goal is to **learn the posterior distribution** over the network’s parameter:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \quad (3.4)$$

where:

1.  $\theta$  are the network’s weights (biases can be modeled as weights, if the input is augmented with a feature containing 1).
2.  $p(D|\theta)$  is the likelihood of the dataset.
3.  $p(\theta)$  is the prior distribution of the weights.
4.  $p(D)$  is the evidence, that can be computed as the marginalization over the parameters of the numerator.

However, computing the exact posterior distribution  $p(\theta|D)$  is analytically intractable for most neural network architectures due to the high dimensionality of the parameter space and the marginalization required in the denominator. To address this challenge, several approximate inference methods have been developed.

For example, **Markov Chain Monte Carlo (MCMC)** methods are a class of algorithms used to sample from a probability distribution when direct sampling is difficult [Brooks et al., 2011]. MCMC methods, such as Metropolis-Hastings and Gibbs sampling, iteratively generate samples from the posterior distribution of the model parameters by constructing a Markov chain that has the desired posterior as its equilibrium distribution.

MCMC methods satisfy three conditions that ensure the convergence to the true distribution. First of all, *ergodicity*: Given sufficient time, the trajectory will explore the entire

state space. This allows to "forget" the initial state and, over time, the distribution of states becomes independent of where it started. For the statistical analysis, ergodicity is crucial because it renders time average equivalent to averaging over different systems. The second condition is *aperiodicity*: the proposed move should not be the same after a regular amount of steps and in general there shouldn't be cyclic behaviors. The last condition is *irreducibility*: Any point is reachable from any point, so there are no isolated regions.

Although MCMC provides asymptotically exact samples from the posterior, it is computationally expensive for large-scale neural networks. The need to evaluate the full dataset at each iteration limits its scalability, particularly for high-dimensional weight spaces and large datasets. As a result, various scalable variants of MCMC, such as Stochastic Gradient MCMC (SG-MCMC), have been introduced to address these computational challenges. These methods combine ideas from Markov Chain Monte Carlo (MCMC) methods and stochastic gradient descent (SGD) to approximate the posterior distribution of model parameters in a scalable manner. These methods aim to enable Bayesian inference for large datasets and high-dimensional parameter spaces by incorporating stochasticity from both mini-batch gradients (like in SGD) and noise introduced to simulate a Markov chain (Gaussian noise). There are several variants of SG-MCMC, each with different strategies to improve scalability and accuracy. The most popular ones include:

**Stochastic Gradient Langevin Dynamics (SGLD)** extends standard SGD by adding a Gaussian noise term to the parameter updates, allowing the algorithm to sample from the posterior distribution of the weights [Cheng et al., 2024, Zhang et al., 2022]. The root of this algorithm is Langevin Dynamics, that is governed by a stochastic differential equation incorporating both deterministic forces (The gradient of the potential) and random fluctuations (a Wiener process). While the evolution in the stochastic process is described by a differential equation and time is continuous, the time in Langevin SGD is discretized. In this context, every step of update is a sample.

The weight update rule in Langevin SGD is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} L(\theta_t) + \sqrt{2\eta} \xi_t \tag{3.5}$$

where  $\eta$  is the learning rate,  $\mathcal{L}(\theta_t)$  is the loss function of the network evaluated on a minibatch  $\theta_t$  are the parameters as they are the time step  $t$ , and  $\xi_t$  is Gaussian noise. The noise term ensures that the updates explore the weight space and approximate posterior samples. Over time, as the network trains, Langevin dynamics help the model converge to the posterior distribution [Welling and Teh, 2011b].

One of the main advantages of SGLD is its scalability. By operating on mini-batches of data rather than the entire dataset, it significantly reduces the computational burden compared to full-batch MCMC. However, SGLD introduces additional challenges, such as the need to carefully balance the noise level and learning rate to ensure accurate posterior sampling.

Another issue with SGLD is that it is asymptotically exact only in the case of full-batch gradients (using the entire dataset at every time step) and if the step size is small enough (the variance of the Gaussian noise must be related to the learning rate to ensure proper exploration of the parameter space).

So, in the case of **Stochastic** Gradient Langevin Dynamics, where gradients are estimated using mini-batches instead of the full dataset, the method becomes an approximation [Nemeth and Fearnhead, 2019]. While still powerful and useful for scalable inference, SGLD typically produces biased samples due to the use of noisy, mini-batch gradients. To compensate for this, more sophisticated methods, such as Preconditioned SGLD [Li et al., 2015, Chen et al., 2016], which incorporates preconditioning techniques to adaptively adjust the step sizes based on local geometry, or Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) [Zhang et al., 2021], which utilizes the principles of Hamiltonian dynamics to explore the parameter space. It simulates a physical system where parameters are treated as particles moving in a potential energy landscape defined by the posterior distribution. These methods to reduce this bias while maintaining scalability.

## Adversarial Attacks

The robustness of BNNs to adversarial examples has been already observed by Gal and Smith [2018] and Bekasov and Murray [2018] by empirically showing that, for BNNs trained with HMC, adversarial examples tend to have high uncertainty, and deriving sufficient conditions for idealised BNNs to recognize adversarial examples. Empirical methods to detect adversarial examples for BNNs that utilise pointwise uncertainty have been introduced in Li and Gal [2017], Feinman et al. [2017], Rawat et al. [2017]. Most of these approaches have largely relied on Monte Carlo dropout for posterior inference [Carlini and Wagner, 2017]. Statistical techniques for the quantification of adversarial robustness of BNNs have been introduced by Cardelli et al. [2019] and employed in Michelmoro et al. [2019] to detect erroneous behaviours in the context of autonomous driving. Furthermore, in Ye and Zhu [2018] a Bayesian approach has been considered in the context of adversarial training, where the authors showed improved performances with respect to other, non-Bayesian, adversarial training approaches. One of the last works on this topic is Bortolussi et al. [2024], where it is analyzed the geometry of adversarial attacks in the large-data, overparametrized limit for Bayesian Neural Networks and demonstrate that in the limit BNN posteriors are robust to gradient-based adversarial attacks. The key finding is that the vulnerability to gradient-based attacks arises as a result of degeneracy in the data distribution, i.e., when the data lies on a lower-dimensional submanifold of the ambient space.

## 3.2 Research hypothesis

Given this background, we now discuss how one might use DFA to train Bayesian neural networks. The best achievement that one could aim for is the **sampling** of the posterior

**in an exact way**, thus satisfying the three conditions of ergodicity, irreducibility and aperiodicity described in the previous section. Could we apply the idea of Langevin SGD and use a step of DFA as a proxy of noise? Since DFA is characterized by its alignment to the feedback matrix (as described in Sec.1.3), a step would be driving the trajectory in the direction of the feedback matrix (random direction), but still taking into consideration the minimization of the loss. The second regime we take into consideration is to operate in a further **approximation** for sampling the posterior, thus operating in an **intermediate** level between Ensembling and Langevin SGD. This approach gathers the samples by letting DFA run for some epochs and the networks sampled are used for Ensembling. The difference with classical Ensembling is that instead of averaging over solutions of SGD starting from different weight initialization, we average weights obtained by DFA after re-initializing the feedback matrix. The network is kept in a not fully converged state, but retains memory from the previous iteration. Finally, we will apply DFA in **classical Ensembling**, i.e. starting from different initialization points in the weights. Many researchers demonstrated that a good ensemble is one where the ensemble’s members are both accurate and make independent errors [Perrone and Cooper, 1993, Opitz and Maclin, 1999], which could easily be achieved by DFA thanks to its degeneracy breaking mechanism.

The three regimes will be analyzed separately against the results of the performances of a single network. The tests will consist in measuring the Ensemble Test Accuracy and the robustness to adversarial attacks.

### 3.3 Methods

#### Evaluation metrics

Supervised classifiers require as many samples as possible in order to learn the complex relations of the features that can be used by the model for prediction. As we discussed in the methods section for the Continual Learning experiments, the relation between performance and number training examples can be displayed through the **learning curve**. The learning curve of a machine learning algorithm relates **performance** to **experience**. In this work, we will adopt accuracy as a primary performance measure in all of our experiments. Experience is typically measured in number of epochs: every epoch is a full cycle of training in which all the training data has been processed. Different epochs show the data in a different order and multiple iterations are often needed for the algorithms to converge to a solution. Such epochs/performance relation can be used to understand the amount of data needed for successful training and allow the comparison of different algorithms. In fact, in this chapter, we will extensively compare the different algorithm’s performances by means of their learning curves: using Epochs as a unit of measure for the information processed.

## Data

In our experimental evaluations, we use the classification benchmark dataset of FashionM-NIST (FMNIST) [Xiao et al., 2017]. It consists of 10 classes of images, each one containing a different type of clothing or accessories (See Figure 3.1 [K V and Gripsy, 2020]).










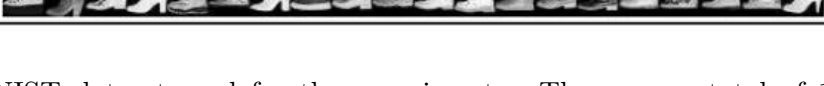
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 3.1: FashionMNIST dataset used for the experiments. There are a total of 10 classes.

## Architecture

We use 3-layer Fully-Connected Neural Networks with ReLU activation function and a Softmax in the last layer. The initialization of the weights and of the Feedback Matrix is drawn from a Xavier uniform initialization [Glorot and Bengio, 2010].



## Algorithms

We optimize the parameters of the Neural Network by applying DFA (see introduction 1.3) without regularization. The loss function minimized is a Cross-Entropy loss.

The prediction can be either given by a single network or be the result of Ensembling multiple networks. We will describe how to obtain the different networks in the specific experiments, but we report the pseudocode of this protocol in Algorithm 1. Note that averaging the network’s prediction doesn’t require to store in memory the configurations of all the networks if the goal is only computing the average of the prediction. One can update the average online, only by keeping track of how many samples it is computed from.

---

**Algorithm 1** Ensembling Protocol given a burn-in number of epochs, the dataset (inputs, targets) and the sampled network’s states. Returns the output (vector with probabilities for each class), standard deviation over the batches and samples, prediction.

---

**for** Batches of the dataset **do**

    For each sampled network, compute the output.

    Average the output of the sampled networks excluding the first n-networks based on the burn-in n.

**end for**

    Use the average output over the batches for prediction.

**return** Average output, Standard Deviation, prediction

---

## Adversarial Attacks

Adversarial attacks are techniques used to construct small manipulations of the inputs of a neural network in such a way that they cause incorrect predictions. These attacks exploit the vulnerabilities inherent in machine learning systems, particularly neural networks, which can be surprisingly sensitive to small changes in input data that are often imperceptible to human observers.

Adversarial attacks can either have complete knowledge of the model architecture, including its weights and parameters, or not. In the first case, more precise manipulation of inputs can be done using gradient information to generate adversarial examples. In the case of Black Box Attacks, the attacker has no access to the internal workings of the model but can observe its outputs. This makes crafting adversarial examples more challenging since the attacker must rely on trial and error or other indirect methods to infer how to manipulate inputs effectively.

Two algorithms that were used by [Bortolussi et al. \[2024\]](#) and that we will replicate belong to the first kind and are gradient-based where every image of the original dataset is perturbed and then saved, obtaining an "attack dataset". The first way to compute the perturbations is **Fast Gradient Sign Method (FGSM)** that computes the gradient of the loss function with respect to the input image and adjusts the image pixels in the



---

**Algorithm 2** FGSM in Pytorch given a dataset, perturbation strength  $\varepsilon$

---

```
for image and label in dataset do
  image.requires_grad=True
  output = Net(image , n_burn-in) #feed the image through the network or the net-
works ensemble
  loss=CrossEntropy(ouptut,label) #compute the loss
  Net.zero_grad()
  loss.backward() #compute the gradients of the loss with respect to the input image
  image_grad = image.grad.data #store the gradients
  perturbed_image = image + $\varepsilon$ * image_grad.sign()
  perturbed_image = torch.clamp(perturbed image , 0 , 1)
end for
return perturbed_image
```

---

---

**Algorithm 3** PGD in Pytorch given a dataset, maximum perturbation  $\varepsilon$ , step size  $\alpha$ , number of iterations  $n\_iter$

---

```
for image and label in dataset do
  original_image = copy.deepcopy(image)
  for i in n_iter do
    image.requires_grad=True
    output = Net(image , n_burn-in) #feed the image through the network or the
networks ensemble
    loss=CrossEntropy(ouptut,label) compute the loss
    Net.zero_grad()
    loss.backward() #compute the gradients of the loss with respect to the input
image
    image_grad = image.grad.data #store the gradients
    perturbed_image = image + $\alpha$ * image_grad.sign()
     $\eta$  = clamp(perturbed_image- original_image, - $\eta$ , + $\eta$ )
    perturbed_image = original_image +  $\eta$ 
  end for
end for
return perturbed_image
```

---

direction that increases loss. See the pseudocode in Algorithm 2 and **Projected Gradient Descent (PGD)** which applies FGSM iteratively multiple times with small perturbations. See the pseudocode in Algorithm 3.

### Initial run details and experiments protocol

Since our goal is to use the change of the feedback matrix to sample the posterior, it is convenient to start from a network that is already at convergence. For computational time economy, we trained an initial network and then loaded the final network’s state for starting the sampling experiments. We save 8 networks trained with different pseudo-random states of the weight initialization and feedback matrix initialization.

In Figure 3.2, we plot the learning curves corresponding to the initial run. We expand on the choice of the rank of the feedback matrix in section 3.6.

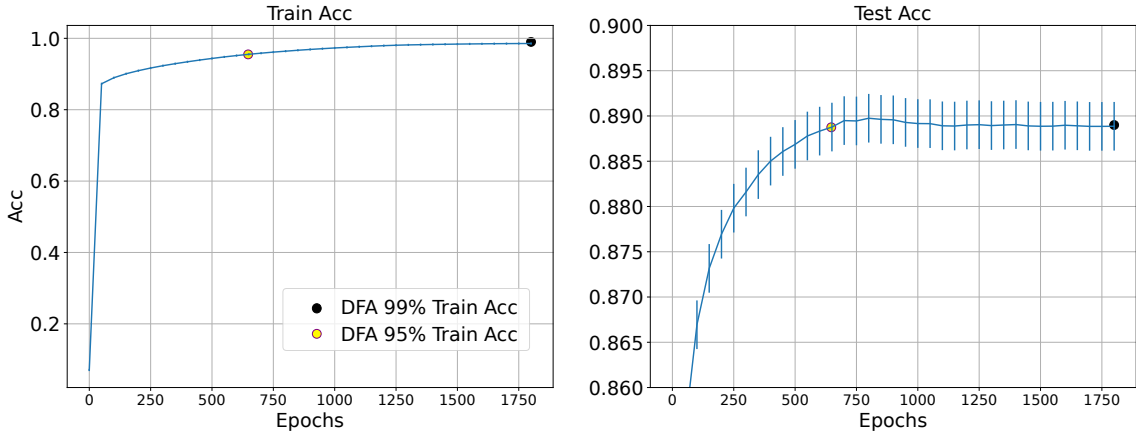


Figure 3.2: Learning curves of DFA used for initializing the Bayesian sampling protocols. Parameters: [lr = 0.1, std(F) = 0.1]

### 3.4 Sampling the posterior with DFA

We attempt to use the peculiar weight updates of DFA as a tool for efficiently gathering samples of the posterior. For this, we build on a first run that is almost in a converged state (Black checkpoint in Figure 3.8). Importantly, in this first test we change the feedback matrix every single epoch. The starting Test Accuracy is  $88.89\% \pm 0.1\%$  and the Training Accuracy is  $98.55\% \pm 0.02$ ). The stochastic nature of training with minibatches and the limited size of the network renders this procedure an approximation of the posterior sampling, but keeping the learning rate very small, in such a way as to make small updates, thus "adiabatic" and "reversible" in a physical perspective. This brings the system

---

**Algorithm 4** Experiment protocol pseudocode

---

```
for seed in seed_list do
  Run a network with DFA until it reaches a threshold in Train Accuracy
  for step in n_steps do
    Re-initialize the Feedback Matrix corresponding to each layer sampling it from
    the distribution  $\mathcal{U}(-a, a)$ ,  $a = 3 * \sqrt{\text{hidden\_size}}$ 
    Update the network with DFA's update rule for n epochs
    Save the network
  end for
end for
return
```

---

to satisfy the detailed balance condition, and satisfying the three conditions of ergodicity, irreducibility and aperiodicity described in the previous paragraph.

We perform an experiment following the protocol described in the Pseudocode 4. The results, averaged over 8 seeds, are shown in Figure 3.3 for two different learning rates. We can notice that, while making small steps in random directions, the system moves toward a lower Training Loss. The maximum Test Accuracy of  $89.02\% \pm 0.19$  is reached around the 30th step. After that point, the Train Accuracy reaches a plateau, and the Test Accuracy decreases. One would normally think that overfitting is taking place, but the Test Loss (lower panel of Figure 3.3) has a plateau, meaning that the system is not going towards a lower generalization error. Probably, the decrease of the Test Accuracy indicates fluctuation due to the sampling.

In order to check if the samples were de-correlating with the initial state, we performed a simple overlap measure of the weights in the layer before classification (See the lower panel of Figure 3.11 in Appendix ). The result is that with a learning rate of  $10^{-5}$ , the overlap is fixed at 1 while with a learning rate of 0.001, the overlap diminishes only by  $10^{-5}$ .

The message we can take from this experiment is that even if we change the feedback matrix every epoch the samples are extremely similar to the starting point, and what we can conclude from the plots of Accuracy and Losses is that the samples do not escape from the minimum in which the dynamics is initialized.

With this protocol, the random step might not be enough to escape the minimum for two reasons: first of all, DFA by construction has small updates when the error is small, which is exactly the starting condition because the Train Accuracy is 98.55%, which is close to interpolation. Secondly, DFA requires on the order of 100 epochs to perform the alignment phase and be able to find a new solution [Refinetti et al., 2021]. For this reason, changing the feedback matrix every epoch doesn't exploit DFA's potential to find different minima as it is instead essential for sampling multi-modal posteriors.

In Appendix 3.8 we show that using the same protocol with a larger learning rate the

minimum can be escaped and the overlap with the initial point can decrease up to 65% in 120 steps, but the system is then driven to a local minimum with a much lower Test Accuracy. Therefore, even increasing the learning rate is not an adequate strategy.

In the next sections, we explore two related questions: how the number of epochs before changing the feedback matrix influences the results and how the training accuracy of the starting point affects the results. As we will see, starting the dynamics from a configuration with *low* training accuracy will be the winning strategy.

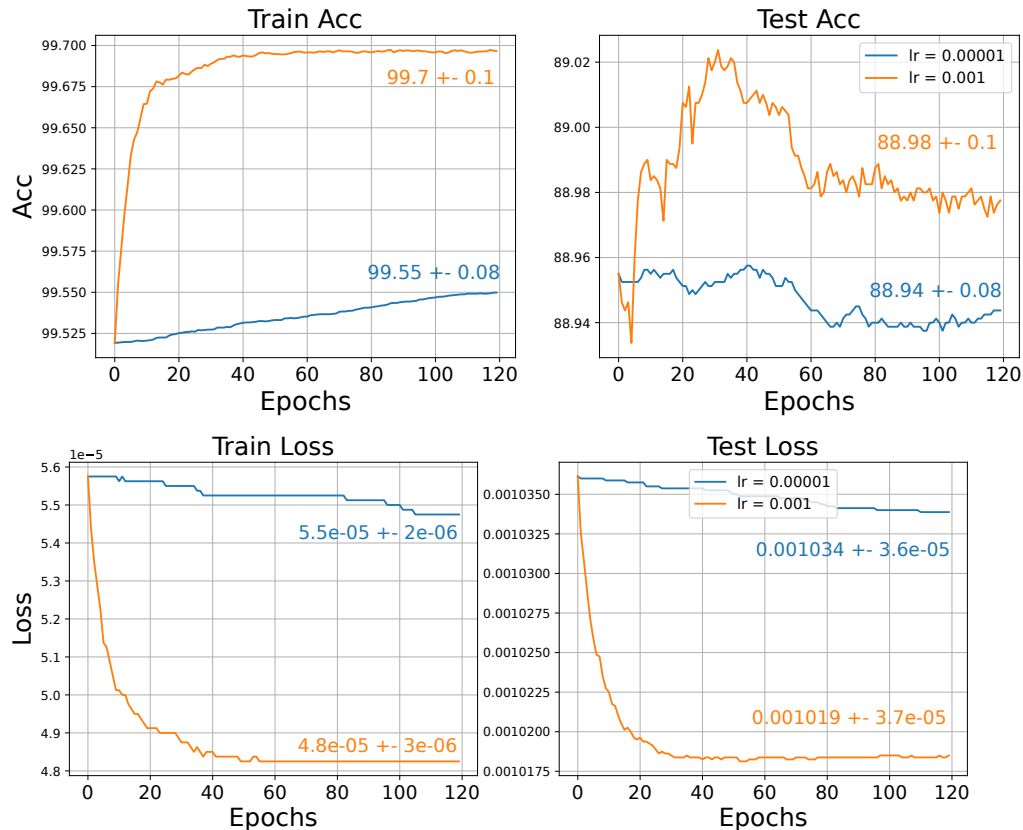


Figure 3.3: Learning curves of the samples (1 sample every epoch) averaged over 8 random seeds. The numbers in the same colour as the curves refer to the last sample and outline the values of the average and the standard deviation. The errorbars are omitted for better readability: in all panels except for the Train Accuracy, the errorbars of the two curves overlap. For a learning rate of 0.001, the average accuracy reaches a maximum of  $89.02\% \pm 0.19\%$  while a lower learning rate reaches  $88.96\% \pm 0.17\%$ . The same plots including the larger learning rate can be found in Appendix 3.8 and precisely in Figure 3.10.

### 3.4.1 Changing the feedback matrix more rarely

In the previous section, we saw that changing the feedback matrix every epoch with a new Feedback Matrix is not an efficient sampling procedure, as it does not allow moving from the minimum determined by the initialization.

In this part, we investigate if we can obtain the uncorrelated samples by letting DFA run for some epochs before changing the feedback matrix, in such a way that it has some time to align to the new feedback matrix. Increasing the number of epochs for each step of sampling.

For these experiments, we use a learning rate of 0.1 and the sampling protocol described in Pseudocode 4, starting from the same point as the previous experiment (black checkpoint in Figure 3.2, corresponding to a training accuracy of 98.55%) but with the difference of using 5, 10, 50 and 200, 500 and 1000 epochs before changing the feedback matrix. As discussed in Appendix (Figure 3.12), we decided to use a stopping rule as soon as the Training Loss increased over a threshold. This results in a row acceptance criterion that stops the movements in directions of high loss.

We compare the accuracy of Ensembles of networks collected in this way and we show the results in Figure 3.4. On the left panel we show the Accuracy for increasing number of steps, and on the right we rescale the values based on the computational time for each step, i.e. the number of epochs.

First of all, we can notice that the average performance of the first samples is lower than the accuracy of the starting point (red line). This is a good indication of the fact that the system moves from the initialization state. Secondly, we notice that the steady state performance of the different experiments depends on the number of epochs one waits before changing the feedback matrix (a "step"). The longer the step, the higher the Test Accuracy of the Ensemble. For a step of 500 epochs and of 1000 epochs the two ensemble averages both converge to the same level of Accuracy of BP, overcoming the historical gap between DFA and BP.

We can conclude that the best length for the step is of at least 500 epochs/step. The initial computational cost to reach 98% Train Accuracy is quite large, as it is visible in Figure 3.2, so this approach is not optimally efficient. In the next section, we test if starting from a shorter run still preserves the results.

### 3.4.2 The influence of the starting point

We now repeat the dynamics starting from a network that reached intermediate convergence, the hollow checkpoint in Figure 3.2 which has  $95.2\% \pm 0.3$  Train Accuracy. In these conditions the algorithm of DFA has by construction larger updates. Indeed, the update of the weights is proportional to the training error.

In this case we consider only the case of 200 and 500 epochs per step. The results are consistent with the previous ones: the higher the number of epochs per step, the higher

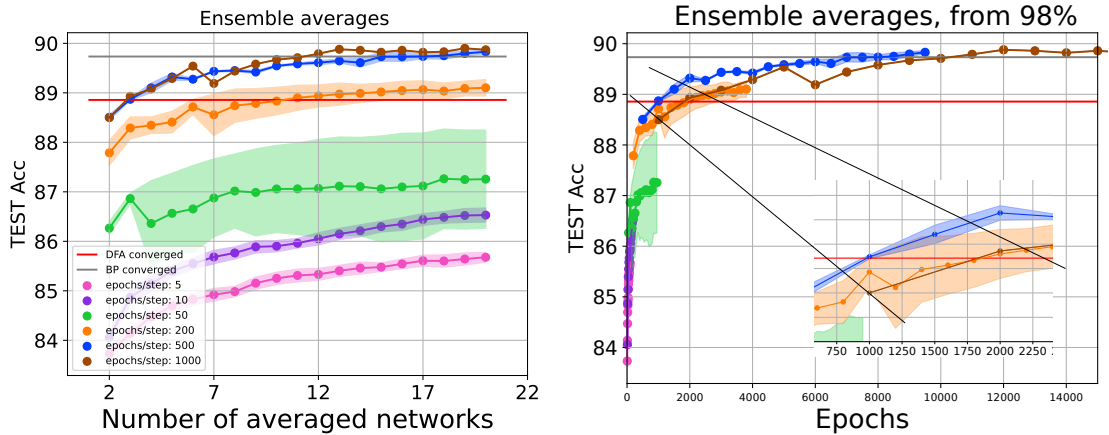


Figure 3.4: Test Accuracy of ensembles for a dynamics started at 98.55% of training accuracy. On the left, the accuracy is reported as a function of the number of samples and on the right as a function of the number of epochs. Recall that the computational time is proportional to the number of epochs. The red horizontal line corresponds to the best accuracy of a single DFA model. The gray horizontal line is the best accuracy of a single model trained by standard backpropagation. The case of 50 epochs/step has a larger errorbar shadow because in this case we didn't use the stopping criterion as discussed in the appendix (section 3.8). The case of 1000 epochs/step has only one seed due to computational time.

the steady-state performances.

The steady state performance of 500epochs/step is consistent with the previous experiment, in which the upper bound of BP is reached.

We can see in Figure 3.5 that the 200 epochs/step overcomes the maximum test accuracy of plain DFA (horizontal line on the right panel, corresponding to the maximum of the red line on the left panel) before 1000 epochs. In the 500 epochs/step this happens before 750 epochs (see the blue line crossing 88.9%).

We observe a marginal benefit in computational efficiency with respect to the results of the experiment in the previous section. The benefit comes both from the epochs saved for computing the initial point: here, 650 epochs while the previous initialization point was obtained with a number of epochs of 1800 (more than double).

The Ensemble Accuracy of the 500 epochs/step after 2 steps (1000 epochs in total) is above the previous experiment: 89.1% vs 88.9%. This is also an effect of a starting point in which the training error is significantly different from zero, an essential prerequisite for making DFA work efficiently.

We can conclude that the protocol we designed is robust with respect to the initialization point. Starting the dynamical sampling at a non converged states is marginally

beneficial for the Ensemble test accuracy and in terms of computational cost.

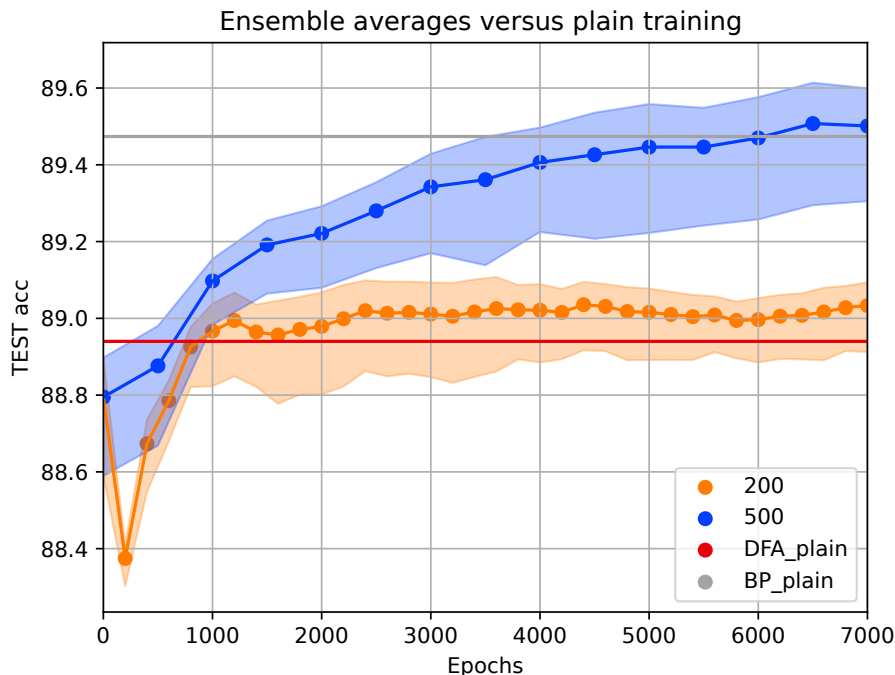


Figure 3.5: Ensemble Test Accuracy of the Ensembled networks in comparison with a plain BP and DFA run starting from the same point.

### 3.4.3 Averaging over different seeds

Classical Ensembling has been previously performed by averaging the output of different neural networks trained on the same data set, starting from different initial weights. In this section, we compare the cost of the dynamic sampling strategy with the naive sampling obtained by training a totally different model for every sample.

We consider  $n$  independent networks of 200, 500 and 1000 epochs each. The results are shown in Figure 3.6.

First of all, looking at the green line in Figure 3.6, we can notice that averaging 250 epochs from the same point yields an Ensemble Test Accuracy which coincides with the convergence accuracy of DFA (even lower with 150 epochs, shown in Appendix Figure 3.16). Instead, with the dynamical sampling, averaging runs of 200 epochs/step can bring the Test Accuracy above this level (See orange lines in Figures 3.4 and 3.5). This comparison shows that our dynamical procedure cannot be "parallelized" by running the 250 epochs at the same time from different starting point.

The Ensemble accuracy using different seeds of DFA and 500 epochs (blue line in Figure 3.6) reaches BP’s performances after averaging on at least 7 networks, so the total ‘computational cost’ of reaching the threshold of BP (89.48%) is 3500 epochs. The same level of performance is achieved by the Ensemble accuracy of the dynamical sampling with larger computational cost, as it is at least of 7150 epochs (650 initial epochs + 13 steps of 500 epochs/step). Therefore, the dynamic sampling procedure described in the previous section is less efficient than the naive sampling procedure.

Finally, we perform Ensembling over DFA networks that run for 1000 Epochs (orange line in Figure 3.6). By averaging over the different seeds, in this case, the Ensemble Test Accuracy is brought to levels as high as 90.17 (or 90.48 with 3000 epochs, shown in Appendix Figure 3.16), which is beyond the accuracy that can be reached with plain BP (or ensembles of BP, see Appendix Figure 3.16). This can be explained thanks to the degeneracy breaking of DFA: the networks fully converged with different feedback matrices end up in different minima, and this brings better performances than BP and dynamical ensembles. This seems to be the best strategy so far. Note that we gathered limited results with steps of 1000 epochs in the dynamical protocol, so a fair comparison cannot be carried out with the current knowledge. This result can be taken as a suggestion for future research on the dynamical sampling protocol.

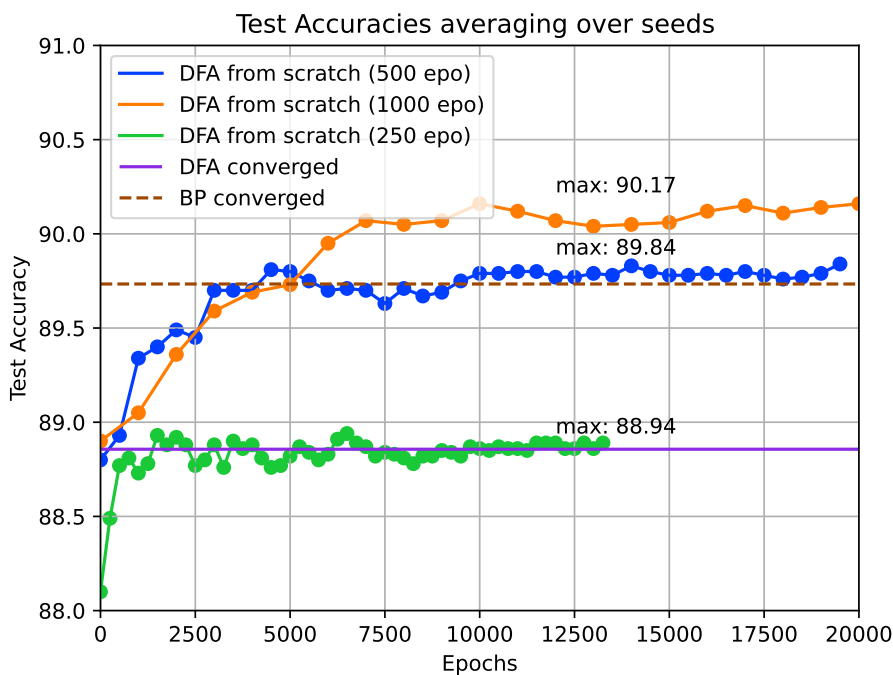


Figure 3.6: Test Accuracy of ensembles of different seeds of DFA compared to the Test Accuracy of one DFA converged network and one BP converged network.



### 3.5 Robustness to Adversarial Attacks

Bayesian Neural Networks have been shown to be more robust to gradient-based adversarial attacks [Bortolussi et al., 2024]. This advantage, in certain theoretical limits, has been explained with the fact that the gradients of the expected loss function of a BNN with respect to input points can vanish [Bortolussi et al., 2024]. This result has been shown to hold for various BNN architectures trained with Hamiltonian Monte Carlo (HMC) and with Variational Inference (VI) on both MNIST and Fashion MNIST. In this section, we test if an Ensemble of networks sampled via the dynamical procedure still has the property of increased robustness to adversarial attacks. In Figure 3.7, we show that the ensembled networks indeed have higher Test Accuracy than the single sampled networks. The Test Accuracy gained on the ensembled networks with respect to the single networks in the manipulated datasets are of 14.12% (FGSM) and 1.6% (PGD, mild attack). So, the gap between the two cases depends on the strength of the attack but for a strong attack as FGSM the gap becomes considerable.

Another feature we can notice is that the single networks exhibit a much larger variability along the steps. This variability reflects the original Test Accuracy for a mild attack and becomes smooth for the larger attack, along with a decreasing trend. The ensembled networks are equally robust along the steps and the strength has a growing trend in the strong attack.

The standard deviation of the single networks has an increasing trend in the stronger attack, reaching up to 17.87% in the last step while the standard deviation of the ensembled networks is not increasing and is 6.11% in the last step.

### 3.6 Impact of rank and similarity on re-alignment time

With this experiment, we show how the re-alignment time is affected by: A) the rank of the new feedback matrix and B) of the similarity of the feedback matrix with the previous one.

In the setup we used in this experiment, we control with the parameter  $\beta$  the similarity with the previous matrix (larger  $\beta$  meaning larger component of the new sampled matrix over the previous matrix). The FM was re-initialized according to the following variance-preserving formula:

$$F_{new} = F_{sampled} * \beta + \sqrt{1 - \beta^2} * F_{old} \tag{3.6}$$

In Algorithm 4 we outline the experiment protocol and in this case, the FM is changed every 400 epochs after a first training period in which DFA reached  $94.79\% \pm 0.12$  Train Accuracy and  $88.80 \pm 0.2$  Test Accuracy corresponding to 400-450 epochs according to the different random seed with a learning rate of 0.1,  $\text{std}(F)$  0.1. as discussed in the paragraph above. We used 8 different initialization seeds, which set the pseudo-random state of the weight initialization and of the feedback matrix initialization.

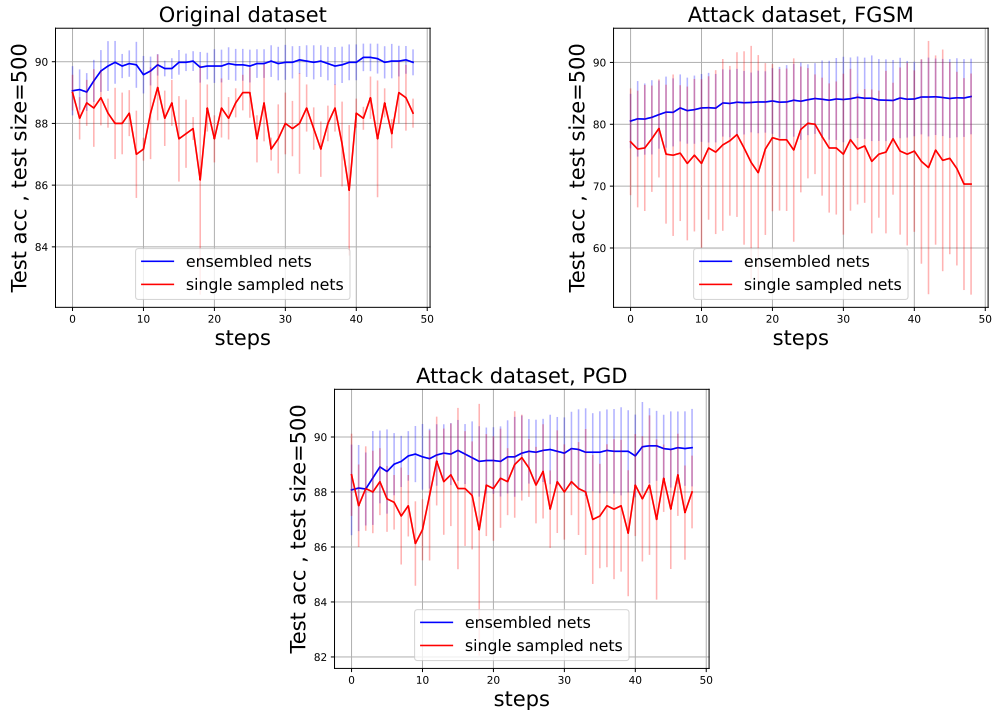


Figure 3.7: Test Accuracy of single sampled nets and their Ensembles. The dataset under test has been attacked by FGSM with  $\epsilon = 0.1$  or PGD ( $\epsilon = 0.05$ ,  $\alpha = 0.01$ , niter=10) the last hyperparameters are for PGD a mild attack. The original dataset is sub-sampled to 500 images in this case.

Consider that the Feedback matrix is a rectangular matrix with dimensions [number of classes, number of hidden neurons] and its rank corresponds to the smallest dimension. In our experiment, the number of classes is 10 while the number of neurons in the hidden layers is 1000, so the maximum rank is 10. In this experiment, we use the Feedback matrices of both layers with a rank of 5 and 10.

**The influence of the Rank of the FM** on the **Train Accuracy** and **Train Loss** is clear: higher the rank (green and orange lines in Figure 3.8), higher is the recovery time necessary to restore alignment in the first switch. What influences the steady-state, of the Train Accuracy, instead, is the similarity between the previous FM with the new ones.

Focusing on the plots for the **Test Loss**, we notice how the full-rank cases (orange and green lines) display overfitting. This is a confirmation of the fact that the re-alignment has been completed, and the training loss is minimized by overfitting the dataset. Despite this overfitting, the separation of the green and orange line in the **Test Accuracy** is still present, indicating that a more similar feedback matrix (orange line) can bring a better generalization.

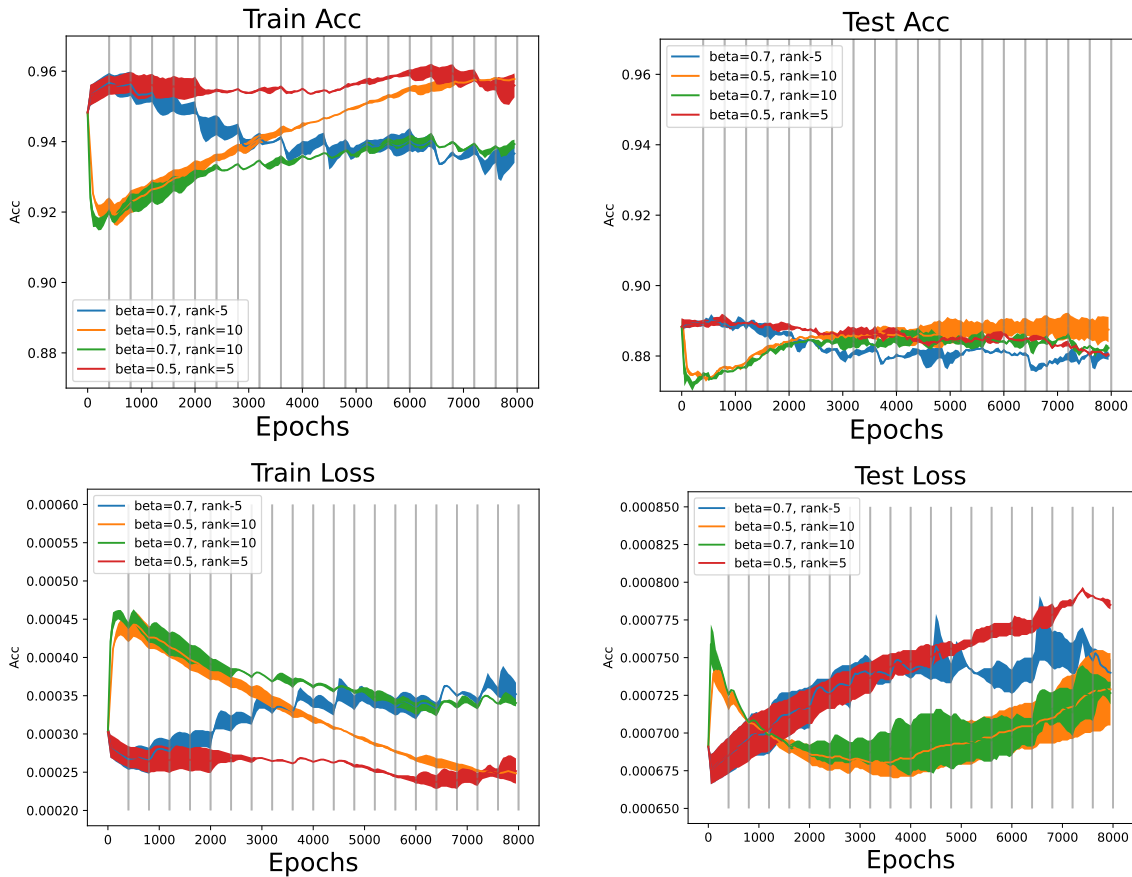


Figure 3.8: Train Accuracy, Test Accuracy, Train Loss and Test Loss of DFA with the Feedback Matrix re-initialized every 300 epochs (vertical gray lines) with different rank and different degrees of similarity with the previous matrix (lower beta, larger similarity). The color bands are the standard deviation over 3 runs.

Regarding the **Test Loss** with a smaller rank (blue and red lines), we can notice that there is a monotonic increase. This could indicate either overfitting or a slow realignment. By the **Test Accuracy**, we can notice that the first four switches (until 1600 epochs) the performances are the same for the blue and the red line. One interpretation of this can be a that this corresponds to a phase of pure re-alignment. After 1600 epochs, the Test accuracies of the blue and red lines becomes different, while keeping a monotonic decrease. This might indicate a transition from pure re-alignment to a blend of re-alignment and overfitting.

The fact that the first switches have a stronger re-alignment than the others is probably due to the fact that the initial point might correspond to a deeper minimum. One

observation from this point of view is that a full-rank matrix directed the training towards a completely different minimum, overcoming the loss barrier between the minima. The matrix with a reduced rank, probably, had enough degrees of freedom to re-align the weights while staying in the same minimum. Further experiments should be required to confirm this scenario, but we can already learn something from this experiment: a reasonable choice seems to be a Feedback Matrices with Rank=5. This choice has Test Accuracy comparable or better than the case of the Full Rank. In this way, the space of parameters to be sampled is restricted to a smaller set, rendering the sampling more computationally efficient.

### 3.7 Summary and discussion

The weights of a network trained by DFA carry the fingerprint of the feedback matrices. This property can be seen as a stronger “bias” in the sense of the classic bias-variance trade off of classification models. This alignment between weights and feedback matrices is the natural result of weights being subject to the gradient alignment constraint in order to fit the dataset. This “degeneracy breaking” [Refinetti and Goldt, 2022] between equivalent solutions among the same-loss solutions of the optimization problem means DFA converges to the solution that is closest to the feedback weights. This solution is not found by following solely the steepest descent direction, as it happens in BP, but the directions spanned for the steepest descent search are only the ones that satisfy the alignment constraints.

The dependence of the trained network on the feedback matrices is usually seen as a weakness, and DFA is usually taken into consideration due to the biological plausibility it was inspired by. In fact, in the brain, it is impossible that the error feedback is communicated to every synapsis precisely rescaled by the value of the derivative of the error of the mathematical function of the forward flow of information from the synapsis to the error. Despite this, neurons still receive a feedback from deeper layers in order to modulate the strengthening of the connections to allow learning [Williams and Holtmaat, 2018]. DFA, as we already underlined, implements this strategy.

In this chapter, we tried to leverage this degeneracy breaking to obtain independent samples for Bayesian Averaging, which is difficult to achieve with vanilla BP.

First of all, we sampled the posterior of Bayesian Neural Networks starting from the same idea applied in Langevin SGD: with small enough steps, one can explore the posterior by injecting a noise in SGD. When the injection of noise is replaced by the iteration through different feedback matrices with DFA, ideally one is driving the trajectory to explore independent directions while maintaining a low loss. We found that in this way we can gather samples of networks with comparable training error. This was indicated by a slowly oscillating Test Accuracy in coexistence of a plateau of Train Accuracy, Train loss and Test Loss. Moreover, the ensemble Test Accuracy of the samples obtained in this way is monotonically increasing with the number of samples, meaning that the samples are

adding information if averaged with each other. The minimum at which the system was initialized at the beginning of sampling was not escaped because the overlap of the initial state and of the samples is decreasing only by a constant factor, and did not decay to zero.

Sampling after letting the system evolve for a fixed amount of epochs, instead, allows to go deeper in one minimum and to catch up with the performance of a single run of BP. In fact, the Ensemble accuracy of the samples collected in this way reach the Test Accuracy of BP.

The oscillations of the performances of the raw samples are in the range of  $\pm 0.2\%$  Accuracy. In order to observe this behaviour, we had to adopt a stopping criterion that stops the moves in a very high loss region. This prevents the system to jump to another minimum, unless there were more minima connected by a low-loss path. This was essential to prevent excessive loss accumulation especially in the cases with a number of epochs per step in the range of 50 and 200 due to the incomplete re-alignment and fitting phases. We noticed that also the 500 epochs per steps case, when these phases are completed, the loss accumulation phenomenon becomes more rare, so more investigations could be dedicated to finding alternative solutions in these cases rather than re-initializing the system. The goal with the dynamical protocol would be to find a way to reach neighbouring minima without going through the high-loss directions. This aim is shared with the ones of continual-learning, in which one would like to find a minimum close to the previous one in such a way that the two solutions are close. Alternatively, any other kind of rejection criterion can be implemented, as re-initializing the system to a configuration obtained by a run from scratch with a different initialization seed and different feedback matrix.

We tested this idea in an extreme case: the naive ensembling of different solutions of DFA and we found that averaging among runs from scratch in this way, the Ensemble average can achieve better performances which are higher than BP and of Ensembles of BP. This suggests to increase the time periods between sampling solutions. The intuition is that the degeneracy breaking characteristic of DFA allows to explore different minima, and this is an essential feature in ensembling because different minima will be a different realization of the "systematic error", or bias, of the network. Averaging the predictions among those samples, the bias cancels out, improving performance. Investigating the trade-offs in terms of accuracy and efficiency of such an approach will be an interesting avenue for future work. One could also incorporate this finding in the dynamical sampling protocol, by allowing much larger steps and letting the network reach interpolation before taking a sample, or using the solution of a different seed after encountering a barrier, combining the two frameworks.

We didn't perform an analysis on the uncertainties of the models, but we expect that the uncertainty would be larger in the case of increasing ensembles, especially if they are composed by samples from different minima. This diversity in predictions obtained as averages is likely related to an increased entropy of the system. Testing this hypothesis is an interesting direction for future research.

Another promising direction for future work is to explore the effectiveness of manipu-

lating the rank of the feedback matrix when encountering a high-loss barrier. The feedback matrix that maps 10-outputs to the hidden-size for every layer has a maximum rank of 10, and only 5 out of these are "independent" feedback directions. In our current implementation, this is achieved by sampling two rectangular matrices and taking their product. What would happen if we reduced the rank by rendering the last 5 dimensions being a linear combination of the first 5, and in the next step constraining the first ones? One could alternatively think of reducing the variance of the feedback matrix, in such a way that the updates are on another scale in expectation. Probing the robustness of the ensembles we obtained with DFA sampling to adversarial attacks, we noticed that they are more robust and stable than the single instances. With the integration of ad-hoc rules, one could improve further. One of them regards a careful selection of the samples: it is possible that not all the samples are likely to bring a benefit to the average, for example if the samples themselves have very low test accuracy. Indeed, there exist practices such as the so-called "super learner" approach [Laan et al., 2007], in which one can assign weights to every sample and give more importance to a subset of samples. The weights are usually learned by trying ensembles of subset of samples and monitoring which subsets yield best test prediction when ensembled together.

## 3.8 Appendix C

### Additional plots for sampling 1-epoch/step

Here we show the details of the runs with three different learning rates, including a larger learning rate of 0.1. In this case, as you can see in the blue lines in Figure 3.10, the Train Accuracy decreases from 99.55% to a range between 85% and 89%. The system is not anymore in the minimum of the initial state, and the decreasing loss indicates that the steps of sampling bring the network towards a new minimum. The Test loss in this case often goes below the test loss of the samples with lower learning rate, indicating that indeed a minimum could be found, but it is not an optimal one due to the lower Test Accuracy.

The Ensemble Test Accuracy of the samples gathered in this way has an average of 85.84%. which is much lower than the value obtained with the smaller learning rates. This is shown in Figure 3.9.

In plot 3.11 we show the correlation of the samples with the initial state. It is clear that only a larger learning rate allows to take samples which are deviating from the initial state. These deviations though are not desirable samples because the Test Accuracy is much worse than the starting point.

One interesting experiment would be to take more samples beyond 120 epochs and see if the Test Accuracy reaches optimal values again. But in this case, one would not need to sample every epoch, rather one should wait for the system to be re-optimized. This is what we show in Section 3.4.1.

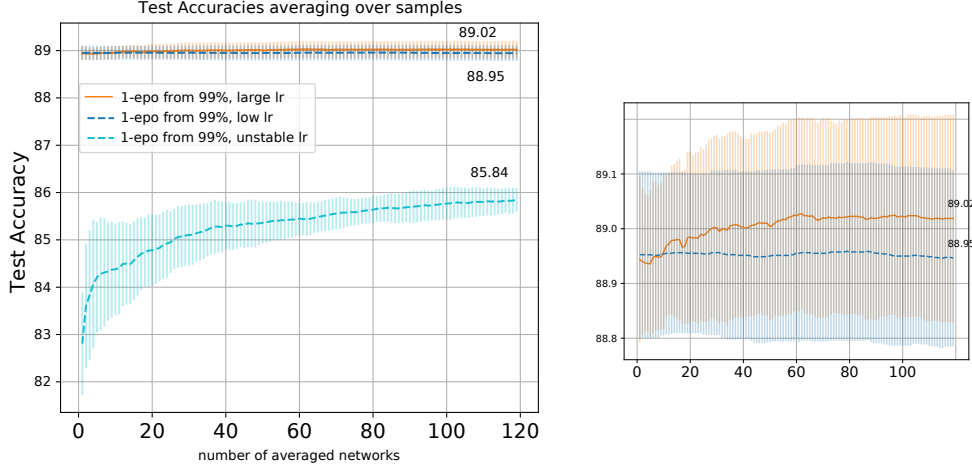


Figure 3.9: Ensemble Test Accuracy in the three regimes with learning rate in  $[0.1, 0.001, 0.00001]$ . The right panel is a zoom on the curves around 89% Test Accuracy.

### Additional plots for sampling with more epochs/step

Looking at Figure 3.12, in the 50 epochs/step case (green line) it is visible that the loss has an increasing trend. This is due to the sequential nature of the alignment and memorization phases of DFA. Alignment brings the weights to a random direction and after it begins the dataset learning. By re-initializing the feedback matrix every 50 epochs, this dynamical system does not have enough time to return to the initial performances, thus the network is every step at a higher and higher loss. Moreover, we can exclude that the test loss increases due to overfitting because the training loss increases in the same way. The same phenomenon is present with steps of 200 epochs even more drastically, so we decided to add a stopping criterion: as soon as the Train Accuracy drops below a threshold we restore the initial state. In this way, we are letting the system move in the new direction unless it is in a high-loss direction. As we discuss in the Appendix, the overlap measures confirm that this procedure allows to obtain samples which are overlapping less with each other even though they maintain an average overlap of 0.6 with the initial point.

We measured the overlap of the samples with respect to the starting point and among themselves with a window size of 2 and 10 and we show the plots in Figures 3.13 and 3.14. From the first measure, we get that the 200 epochs/step case has an oscillating overlap with the starting point around 0.6 while the other cases have a monotonically decreasing overlap with the starting point, which is faster for increasing number of epochs per step. The oscillating result of the 200 epochs/step case is a reflection of the stopping criterion that we introduced. More insights can be gathered from the overlap among the samples distant 2 or 10 steps. From this measure, we get that the 200 epochs/step case has less

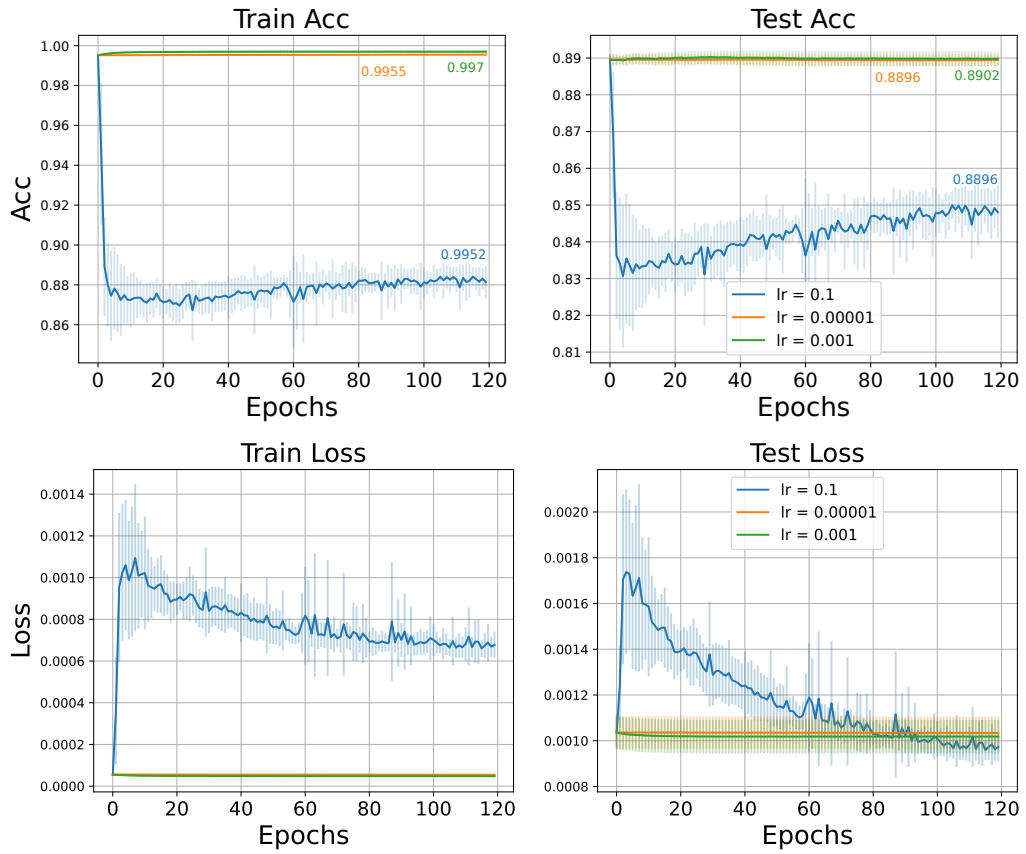


Figure 3.10: Learning curves of the samples (1 sample every epoch) averaged over 8 random seeds. The errorbars correspond to the standard deviation.

overlapping samples.



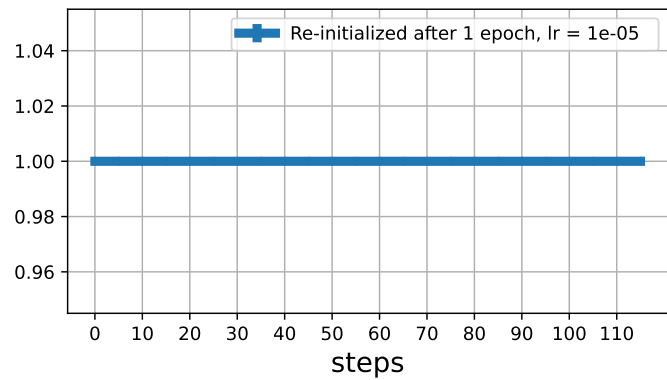
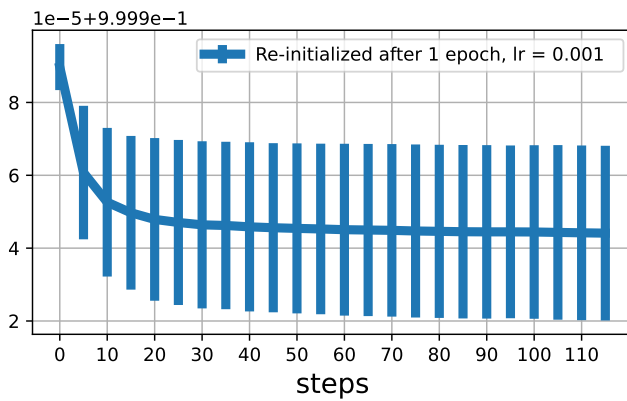
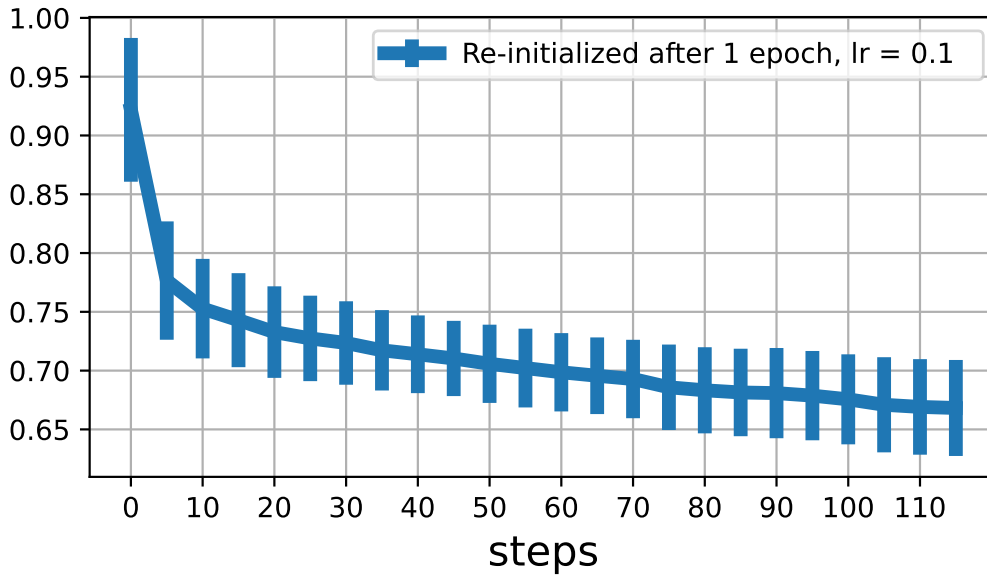


Figure 3.11: Overlap of the layer before classification. Computed as the dot product of the initial converged state with the current state. Note that the range in the changes with  $lr=0.001$  are in the order of  $1e-5$ .

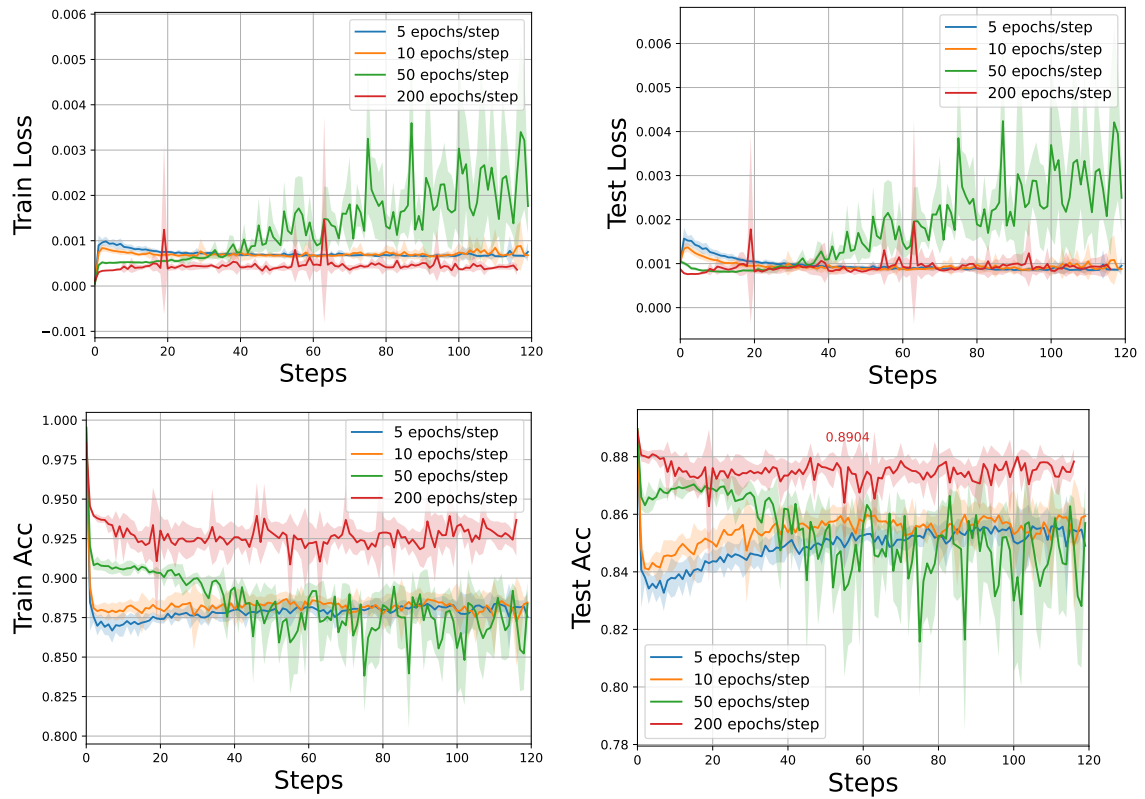


Figure 3.12: Train and Test performances of the samples. Loss above and Accuracy below.

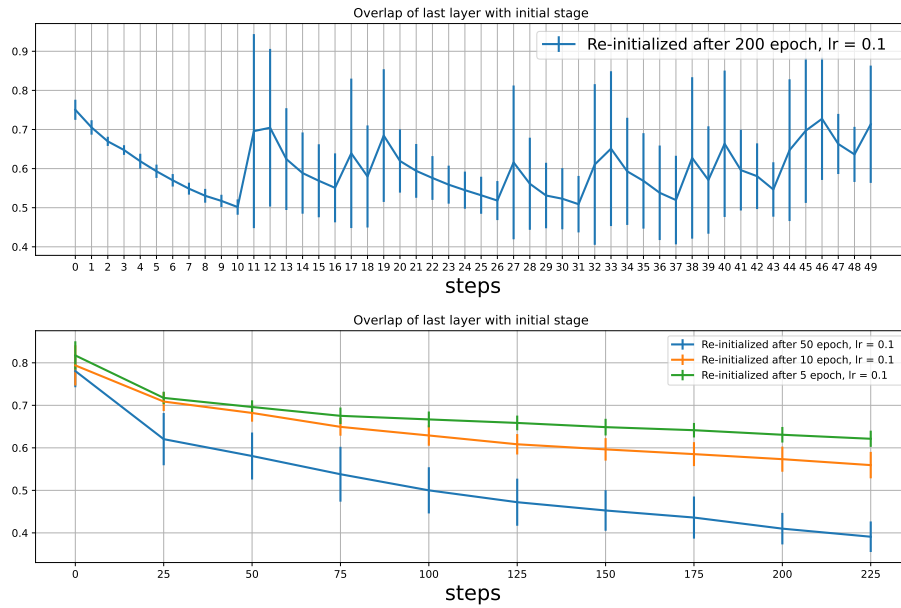


Figure 3.13: Overlaps with the initial configuration. The one of 200 epochs/step (panel above) from step 10 becomes irregular due to the clipping. The lines in panel below show a faster decorrelation for the samples with more epochs per step.

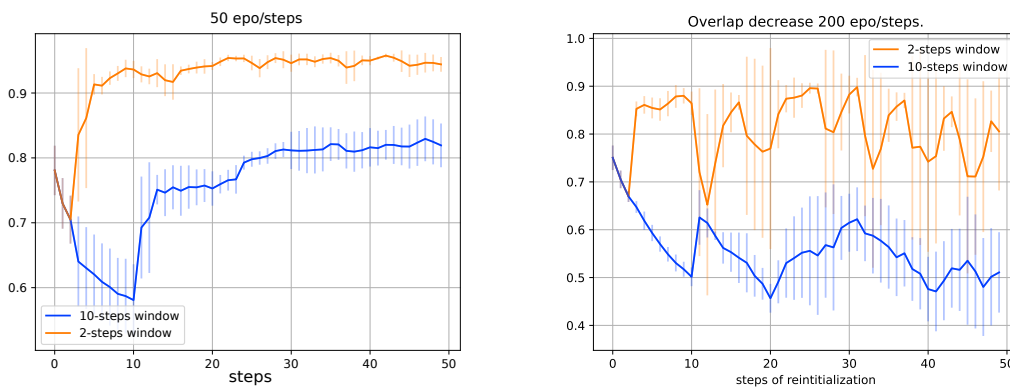


Figure 3.14: Overlaps of samples considering two samples every 2 (orange) and every 10 samples (blue). The first 2 or 10 refer to the overlap with the initialization state.

### Additional plots of Ensembling over seeds

In the main text we show the Ensemble Test Accuracy of DFA runs with different seeds that run for 200, 500 and 1000 epochs. In Figure 3.15, we report the Train Accuracy and the Test Accuracy of these runs (without Ensembling over seeds, but instead showing an errorbar of the standard deviation over the seeds).

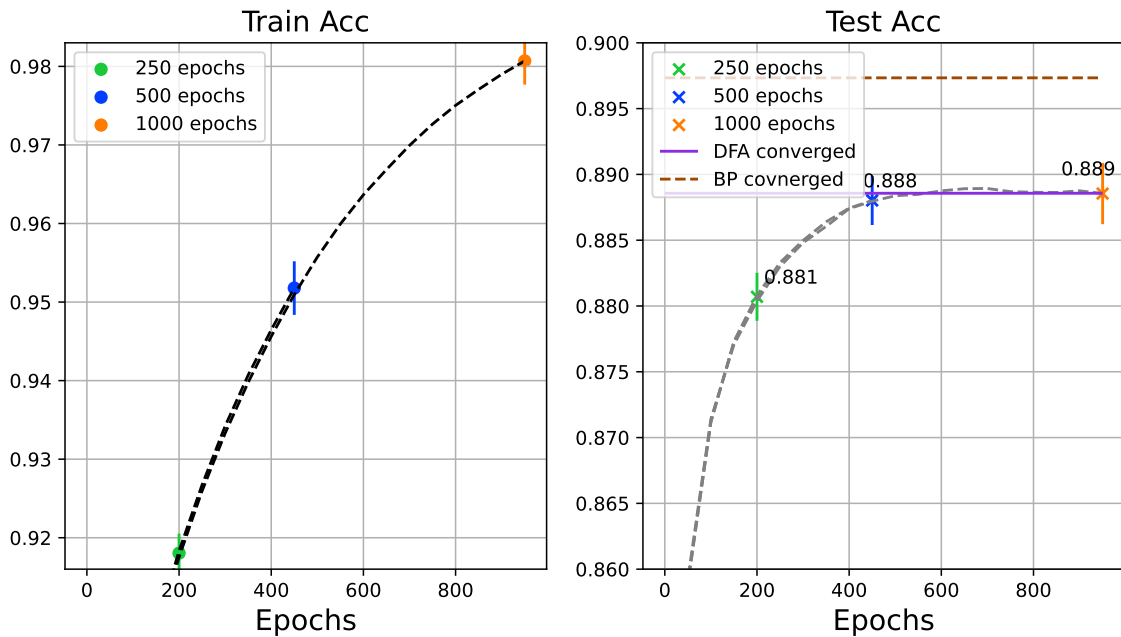


Figure 3.15: Train and Test Accuracy of the runs that we average.

Moreover, here we show in Figure 3.16 additional Ensemble Test Accuracy obtained by the average of DFA-trained networks that generally reached a certain Train Accuracy, regardless of the number of epochs or the parameters used to reach those points. We notice that averaging networks that converged up to 95% Train Accuracy does not yield to Ensemble Test Accuracy that overcomes BP's performance; instead, averaging networks in overfitting (above 95%) did overcome BP's threshold. Moreover, we show that the Ensemble accuracy of BP-trained networks is not as high as the one of the Ensemble of DFA (green line).

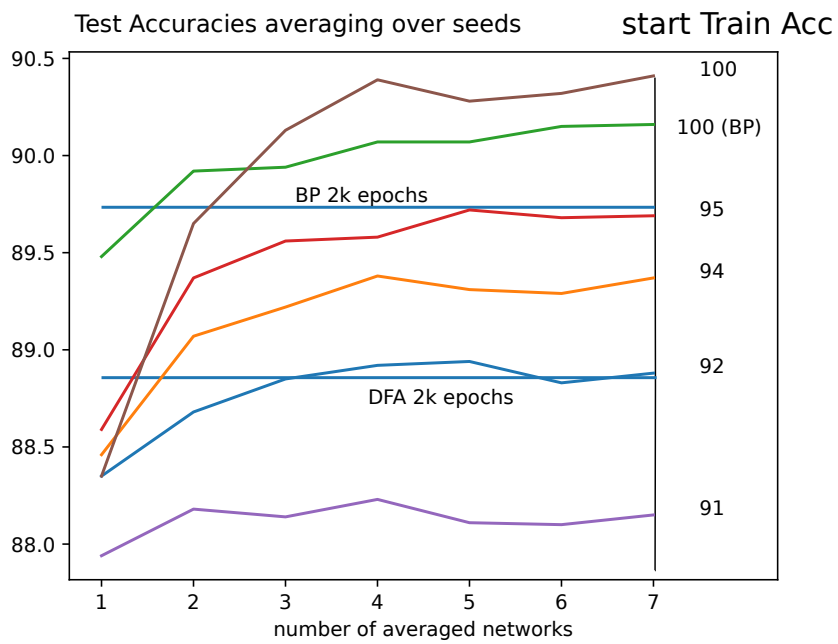


Figure 3.16: Test Accuracy of ensembles of different seeds of DFA compared to the Test Accuracy of one DFA converged network and one BP converged network.

## Additional plots for Adversarial attacks

Guided by curiosity, I wanted to visualize the effect of the stronger attack on the images. Precisely, the aim is to inspect if the attack produced by the gradient of one single sample is different by the attack produced by the gradient of the ensemble. The results look indistinguishable to visual inspection.

## Burn-in effect in ensemble averages

It is known that burn-in is useful in averaging, especially if the Test Accuracy jumps out of the initialization minimum. Nevertheless, in the context of our experiments, each sample can be computationally costly to obtain, for example in the case in which every step consists in 500 epochs of training. We plotted in Figure 3.18 three learning curves of DFA with different discarded samples (0, 5 or 13) in the context of the networks initialized at 95%.

We find that the Ensemble Test accuracy is higher when there are more discarded samples, but after averaging over 20 samples the same Test Accuracy is recovered in all conditions.

For reducing the number of variables in our experiments, we will use no burn-in and future work will be dedicated in clarifying the effect of this on 1-epoch/step simulations.

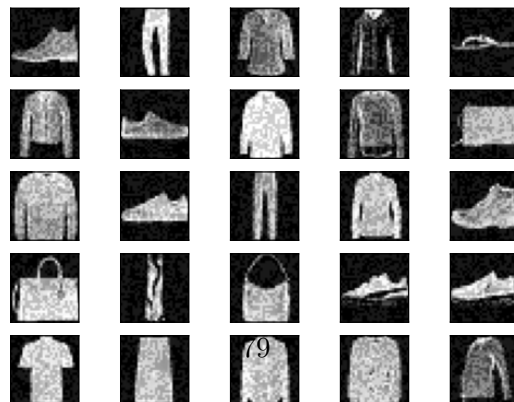
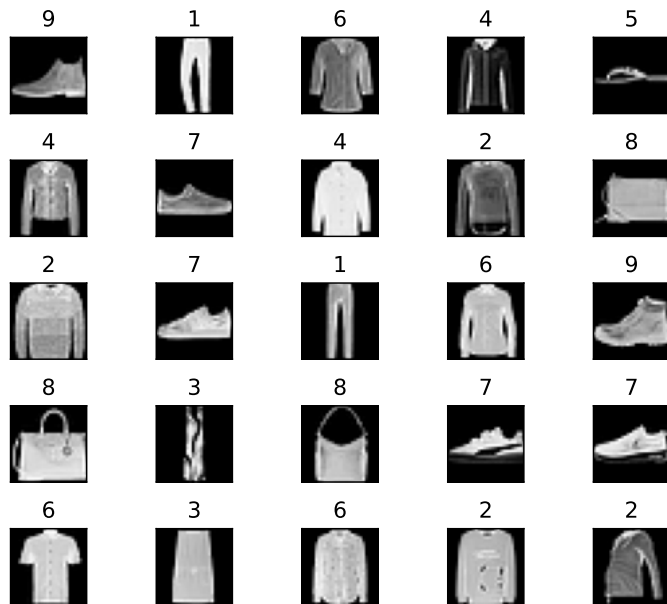


Figure 3.17: Visualization of 25 images sampled randomly from the original (upper) dataset and the corresponding attacked datasets with the FGSM algorithm in the two cases of single (central) network and Ensembled network (bottom).

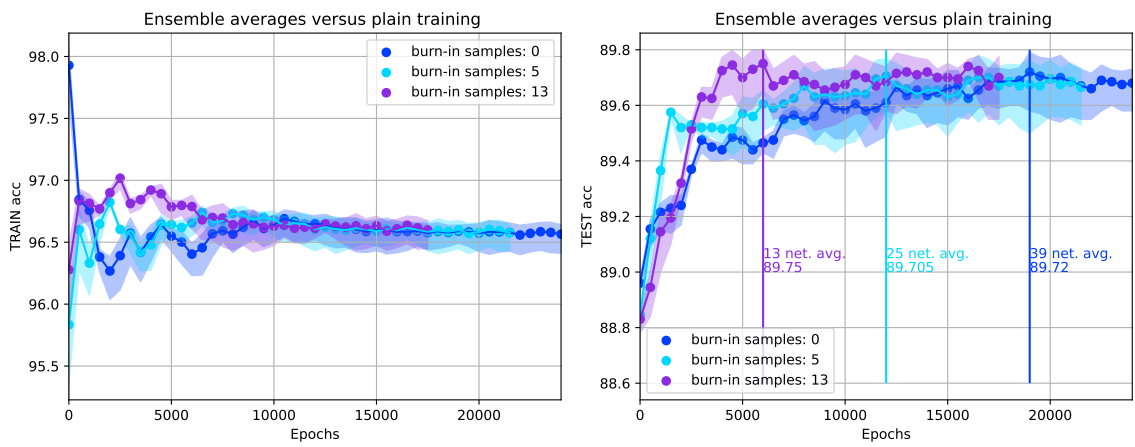


Figure 3.18: Ensemble evaluation of the Train Accuracy and Test Accuracy with different numbers of discarded samples. We indicate the maximum Test Accuracy that can be reached in each case and a vertical line in its correspondence.



## Chapter 4

# Training recurrent neural networks with DFA

### 4.1 Introduction

The publication of the Backpropagation algorithm in [Rosenblatt \[1958\]](#) included a section about the application of BP to recurrent networks. For DFA’s original paper [Nøkland \[2016\]](#) neither the words ”recurrent” and ”time” appear. With the collaboration of Andrea Cossu and Andrea Ceni, upon suggestion of Claudio Gallicchio, we found that DFA can indeed be applied to Recurrent Neural Networks. Moreover, DFA in this setting overcomes some of the traditional limits of RNN which are intrinsically sequential structures. DFA’s update rules can be implemented parallelizing the computations with respect to the time (length of the input sequence). The parallelization opens the path to implement RNNs in neuromorphic hardware (e.g., photonic accelerators). In this chapter, I will show the technical details for extending DFA to Vanilla Recurrent Neural Networks (RNNs) and the Gated Recurrent Units (GRUs) [[Cho et al., 2014](#), [Chung et al., 2014](#)]. Then, I will show the results of applying these equations to several time-series classification datasets. We find that DFA can achieve non-trivial performances in all of the tested datasets but cannot always attain a performance comparable to BPTT. Finally, I will describe my efforts in trying to parallelize the code as enabled by the peculiarity of the equations.

#### 4.1.1 Motivation

Backpropagation [[Rosenblatt, 1958](#)] is the long-standing algorithm for credit assignment in artificial neural networks. Its efficient implementation in digital computers has supported the surge of machine and deep learning techniques as one of the key advancements in the field of artificial intelligence. However, with a few exceptions [[Wright et al., 2022](#)], the adoption of backpropagation-based learning systems is still mainly limited to digital computers and simulations. In fact, it is well known that backpropagation cannot be easily

implemented and deployed in physical systems [Momeni et al., 2023, Lillicrap et al., 2020], for example due to issues like the weight transport, where the synaptic weights of the backward circuit needs to be constantly synchronized with the synaptic weights of the forward circuit [Lillicrap et al., 2016a, Akrouf et al., 2019].

Issues like the weight transport are even more challenging in Recurrent Neural Networks (RNNs) [Elman, 1990], where credit assignment must be performed across time. The most used algorithm to date is BackPropagation Through Time (BPTT) [Werbos, 1990], which extends backpropagation to recurrent architectures.

Over time, a number of backpropagation-free algorithms have been proposed (see Section 4.2 for a non-exhaustive overview), some of them with the explicit objective of being compatible with implementation in physical systems or on unconventional hardware (e.g., neuromorphic, optical).

We focus on Direct Feedback Alignment (DFA) [Nøkland, 2016], a backpropagation-free algorithm for credit assignment that removes the weight transport issue and also allows parallel computation of the weight update. DFA has already been implemented in non conventional hardware, especially photonic Filipovich et al. [2022]. The photonic co-processor introduced in Launay et al. [2020] scales DFA to trillion-parameter random projections.

We review DFA for feedforward networks in Section 1.3. We propose an extension of DFA tailored to recurrent neural networks. Our approach is able to compute the update of the recurrent parameters in parallel over all the time-steps of the input sequence, thus removing one of the major drawbacks of BPTT. In fact, BPTT sends the error signal computed at the end of the input sequence *back in time* to compute the update of the network’s parameters. In our method, instead, the update at each time step is local, as it does not rely on the update computed for other time steps. The local update is computed in parallel for each time step and, due to the weight sharing present in RNNs, ultimately aggregated to compute the final update. The aggregation operation includes information from all the time-steps, thus enabling learning of temporal dependencies.

## 4.2 Prior work

Practical applications of DFA to RNNs have been explored in Nakajima et al. [2022]. The authors performed physical deep learning with an optoelectronic recurrent neural network. However, in their pioneering work, they do not explore the DFA algorithm in the context of fully-trainable RNNs, since only a proof-of-concept using a reservoir computing model [Lukoševičius and Jaeger, 2009] (i.e., untrained recurrent connections) is provided. In this paper, we investigate the potential of DFA on fully-trainable RNNs.

In Han et al. [2020], a DFA-inspired algorithm is derived for RNNs. Nevertheless, they do not implement a genuine version of DFA for RNNs. First, they implement an upper triangular modular structure. Second, they use random projections as powers of the same

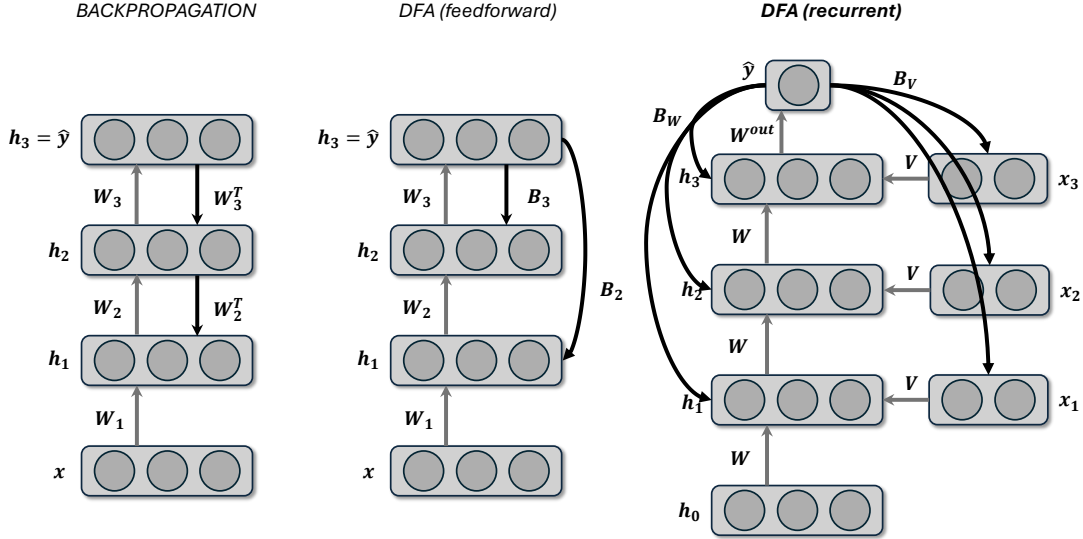


Figure 4.1: We propose DFA applied to recurrent networks (right). The error is projected through random matrices  $B_W$  and  $B_V$ . We also show backpropagation (left) and DFA (middle) applied to feedforward networks. Grey arrows denote the forward phase, black arrows denote the update phase. Note that in the RNN, the matrices  $W$  and  $V$  are shared across time-steps (layers), while in feedforward networks each layer has a different matrix. Also, the RNN receives a different input  $x_t$  at each time step (here, the input sequence has 3 time steps), while the feedforward network only receives one input  $x$ .

matrix, which effectively resembles more an FA algorithm applied to RNNs rather than a DFA algorithm for RNNs. In this paper, we derive a plain version of DFA for RNNs.

### 4.3 DFA for recurrent networks

We develop a version of DFA compatible with RNNs for sequential data processing (Figure 4.1, right). Each example  $x$  is a sequence of  $T$  input vectors:  $x = (x_1, \dots, x_T)$ , where  $x_i \in \mathbb{R}^I$ . We consider the sequence classification task where each sequence  $x$  is associated with a target class  $y$ . The RNN keeps an internal hidden state  $h \in \mathbb{R}^H$  which is updated at each time step. We first focus on the “Vanilla” RNN [Elman, 1990], whose state update of reads:

$$h_{t+1} = \sigma(W h_t + V x_{t+1} + b), \tag{4.1}$$

where  $V \in \mathbb{R}^{H \times I}$  is the input-to-hidden matrix and we call  $a_t$  (pre-activations at time

$t$ ) the terms inside  $\sigma$ . In RNNs, the same layer is applied to all time steps (weight sharing). The output  $\hat{y}$  of the RNN is computed from the hidden state:  $\hat{y} = \sigma(W^{\text{out}}h_t + b^{\text{out}})$ , where  $W^{\text{out}} \in \mathbb{R}^{O \times H}$  and  $b^{\text{out}} \in \mathbb{R}^O$ . The nonlinear function  $\sigma$  can be different from the one used in the hidden layers. For sequence classification tasks the output is computed at the end of the input sequence from  $h_L$ .

Due to the weight sharing, the forward pass of an RNN can be interpreted as the unrolling of the state update function over time. At each time step, the matrix  $W$  and  $V$  (and the bias as well) are used to compute the next hidden state, much like the matrix  $W_l$  is used to compute the layer’s output in a feedforward network. The backpropagation algorithm applied to RNNs, called backpropagation through time (BPTT) updates the hidden-to-hidden weight  $W$  via  $\nabla_W J(\hat{y}, y) = \frac{\partial J}{\partial \hat{y}} \sum_{t=1}^T \frac{\partial \hat{y}}{\partial h_t} \frac{\partial h_t}{\partial W}$ . The term  $\frac{\partial \hat{y}}{\partial h_t}$  hides a dependency between hidden states  $\prod_{j=1}^{t-1} \frac{\partial h_{j+1}}{\partial h_j}$  which is due to the sequential propagation of the error over the time steps.

Our DFA-based algorithm for RNN removes this propagation and updates  $W$  via  $\Delta_W = \frac{\partial J}{\partial \hat{y}} \sum_{t=1}^T \frac{\partial h_t}{\partial W}$ . The error signal  $e$  is projected via a random matrix  $B$ , randomly initialized and kept fixed.

The equations for the update of  $W$  and  $V$  via DFA read:

$$W \leftarrow W - \eta \sum_{t=1}^T ( B e \odot \sigma'(a_t) ) h_{t-1}^T, \quad (4.2)$$

$$V \leftarrow V - \eta \sum_{t=1}^T ( B e \odot \sigma'(a_t) ) x_t^T \quad (4.3)$$

The bias is updated by omitting the outer product.

**DFA for gated recurrent networks.** In addition to the development of DFA for “Vanilla” RNNs (Equation 4.1), we also developed a version of DFA for gated recurrent networks, focusing in particular on the GRU network [Cho et al., 2014, Chung et al., 2014]. The state update for a GRU reads:

$$\begin{aligned} z_{t+1} &= \text{sig}(W_z h_t + V_z x_{t+1} + b_z), \\ r_{t+1} &= \text{sig}(W_r h_t + V_r x_{t+1} + b_r), \\ c_{t+1} &= \tanh(W_c (h_t \odot r_{t+1}) + V_c x_{t+1} + b_c), \\ h_{t+1} &= (1 - z_{t+1}) \odot c_{t+1} + z_{t+1} \odot h_t, \end{aligned}$$

where *tanh* and *sig* are the hyperbolic tangent and sigmoid functions, respectively. Our DFA update for all parameters of the GRU is provided in Appendix 4.4. The output  $\hat{y}$  of the network is computed from the hidden state  $h_t$  as previously discussed.

## More details on DFA GRU equations

### 4.4 DFA for Gated Recurrent Unit network

We provide the update rule of DFA for all the parameters of the GRU.

$$\begin{aligned}
 W_z &\leftarrow W_z - \eta \sum_{t=1}^T (Be \odot h_{t-1} - Be \odot c_t) \odot (r_t \odot (1 - r_t)) h_{t-1}^T, \\
 V_z &\leftarrow V_z - \eta \sum_{t=1}^T (Be \odot h_{t-1} - Be \odot c_t) \odot (r_t \odot (1 - r_t)) x_t^T, \\
 W_r &\leftarrow W_r - \eta \sum_{t=1}^T (W_r (Be \odot (1 - z_t)) * (1 - c_t \odot c_t) h_{t-1}) \odot (r_t \odot (1 - r_t)) h_{t-1}^T, \\
 V_r &\leftarrow V_r - \eta \sum_{t=1}^T (W_r (Be \odot (1 - z_t)) * (1 - c_t \odot c_t) h_{t-1}) \odot (r_t \odot (1 - r_t)) x_t^T, \\
 W_c &\leftarrow W_c - \eta \sum_{t=1}^T (W_r (Be \odot (1 - z_t)) * (1 - c_t \odot c_t) (r_t \odot h_{t-1}))^T, \\
 V_c &\leftarrow V_c - \eta \sum_{t=1}^T (W_r (Be \odot (1 - z_t)) * (1 - c_t \odot c_t)) x_t^T.
 \end{aligned}$$

As in the ‘‘Vanilla’’ RNN, all the bias vectors are updated by omitting the outer product in the corresponding  $W$  or  $V$  update. The matrix  $B$  can also be a different random matrix for each parameter.

### 4.5 Preliminary Experiments

The first test is performed on a synthetic dataset comprising two classes:  $y_1$  is a sinusoid with period 1,  $y_2$  is a sinusoid with period 2. The inputs are presented as a series of 50 points.

The results on the first experiment show that DFA is successful in training the RNN. It does so via its characteristic alignment phase, in which the error increases, followed by the learning phase.

### 4.6 Experiments

We benchmark DFA to four time-series classification datasets:

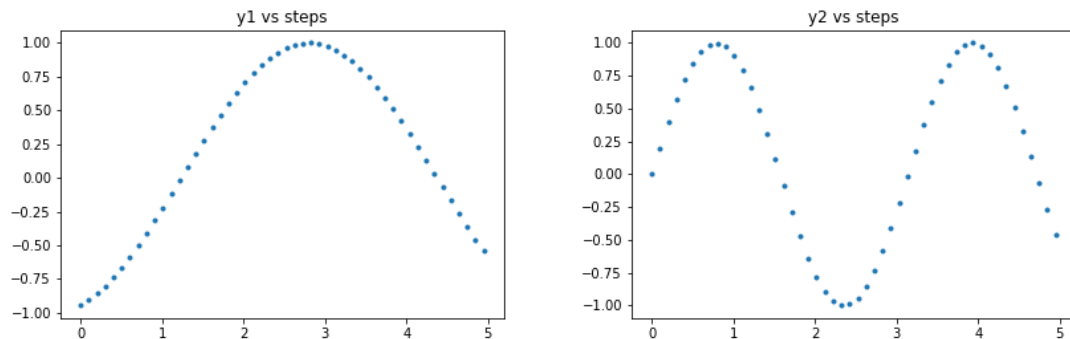


Figure 4.2: Inputs of first experiment

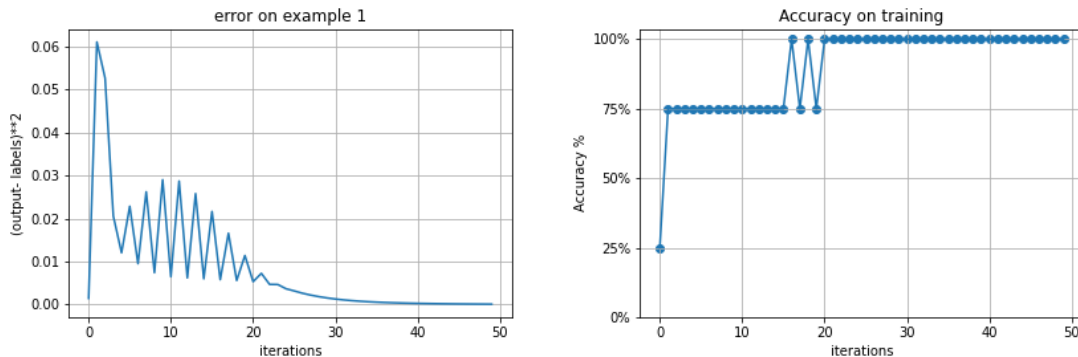


Figure 4.3: Results of first experiment

1. *Libras* [Dias Daniel and Helton \[2009\]](#) which contains 15 classes referencing to a different hand movement type in LIBRAS <sup>1</sup>. The hand movement is represented as a bi-dimensional curve performed by the hand in a period of time;
2. *Row-MNIST* [\[Deng, 2012\]](#): each image of the MNIST dataset is presented to the recurrent model one row at a time;
3. *ECG200* [Olszewski et al. \[2001\]](#) where each series traces the electrical activity recorded during one heartbeat. The two classes are a normal heartbeat and a Myocardial Infarction;
4. *Strawberry* [K. Kemsley](#) consists in the classification of food spectrographs, a task that has obvious applications in food safety and quality assurance. The classes are

<sup>1</sup>LIBRAS is the acronym of the Portuguese name "Lingua BRAsileira de Sinais", is the oficial brazilian sign language.

strawberry (authentic samples) and non-strawberry (adulterated strawberries and other fruits).

### 4.6.1 Methods

The datasets are divided into Train-Validation-Test sets according to the proportions 60%-20%-20%. The hyperparameters have been selected based on a model selection over a restricted range of each parameter, see Appendix 4.6.1 for the details. We run the experiments for a number of epochs sufficient for the learning curves to stabilize and use 5 different Random Seed initialization<sup>2</sup>. Nothing prevents to update the parameters with other optimization algorithms, like Adam. In our PyTorch implementation, we computed the DFA update and used it as the “grad” attribute of the respective weight tensor. In this way, the framework uses the computed update as if it was a gradient, and all the available optimizers can be used out of the box.

Hyperparameters are selected based on the best performances on a validation set among these possible values:  $hsize \in [50, 512]$ ,  $lr \in [0.0005, 0.001, 0.005, 0.01]$ ,  $bs \in [10, 100, 256]$ ,  $clip=2$ . The values selected by the model selection are:

1. Libras: Learning rate = 0.0005 (except for BPTT GRU: learning rate= 0.01), Hidden size = 512, Batch size = 10, Epochs= 900.
2. Strawberry: Learning rate = 0.0005 (except for BPTT GRU: learning rate= 0.005), hidden size = 50 (except for RNN DFA: hidden size= 512), Batch size = 10 (except for RNN DFA: bs=100 and for RNN BPTT: bs= 256), Epochs= 300.
3. ECG200: [ Learning rate = 0.0005 (Except for DFA GRU, lr=0.01), Hidden size = 50, Batch size = 256, Epochs= 500.
4. ROW-MNIST: [Learning rate=0.0005 (Except for RNN DFA and GRU DFA, lr=0.005), Hidden size = 512 ( Except for RNN BPTT, hs= 50), Batch size = 100 (Except for RNN BPTT, bs = 10)].

---

<sup>2</sup>code available upon request

## 4.6.2 Results

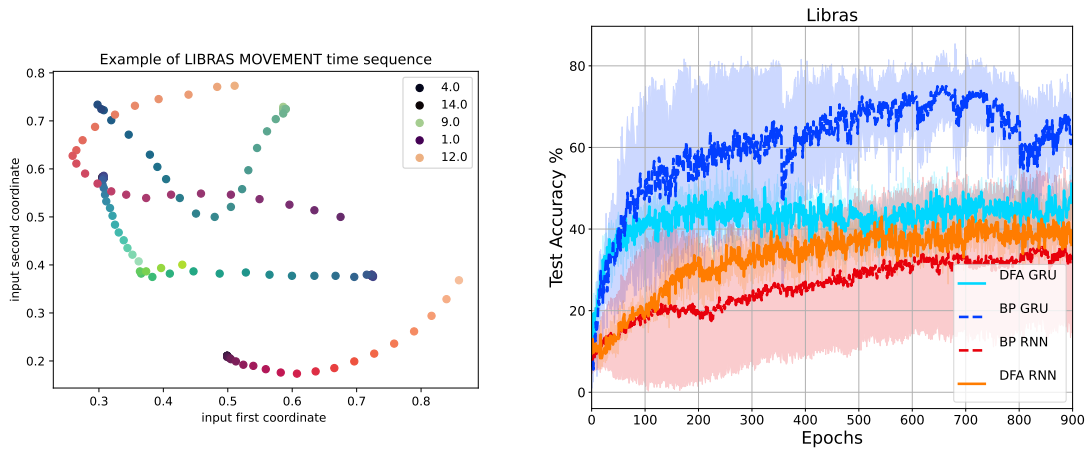


Figure 4.4: Left: Visualization of the first five movements in the test set. Right: Results on the Libras-Movement benchmark dataset obtained on a simple RNN architecture (orange and red) and on a GRU (blue and cyan). We benchmark DFA (darker colors, full line) and BPTT (lighter colors, dashed line). Error-shades: one standard deviation over at least 5 repetitions .

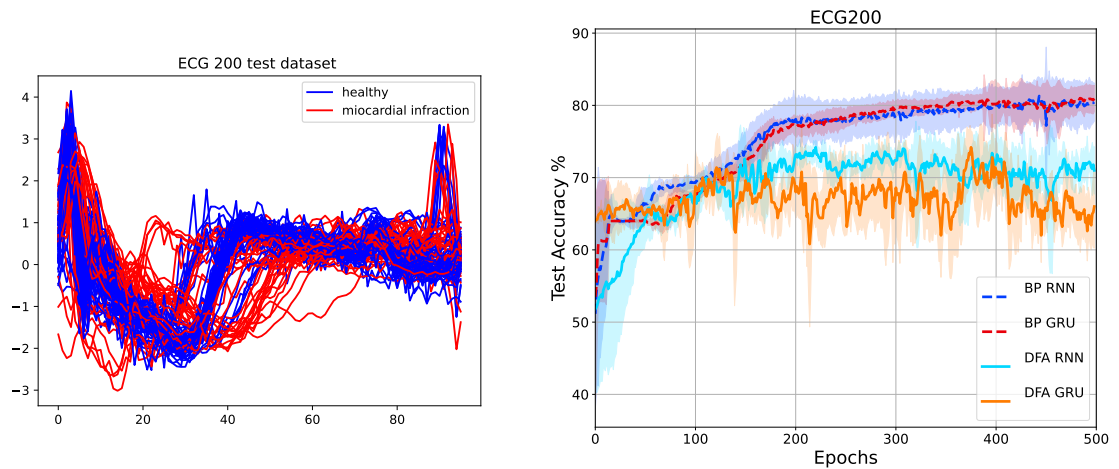


Figure 4.5: Left: Visualization of the test set. Right: Results on the ECG-200 benchmark datasets obtained on a simple RNN architecture (orange and red) and with a GRU (blue and cyan). DFA is indicates by lighter colors and full line; BPTT is in darker colors, dashed line. Error-shades: one standard deviation over at least 5 repetitions.



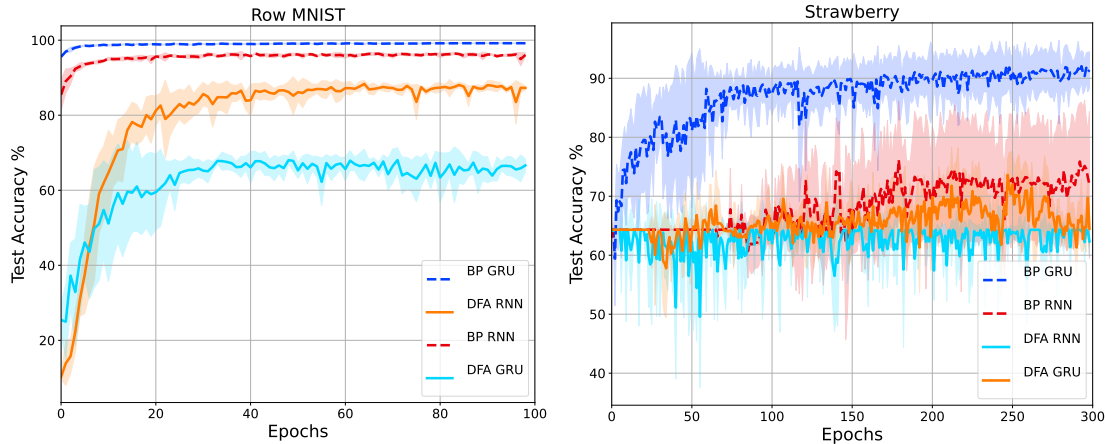


Figure 4.6: Results on the Row-MNIST and ECG-200 benchmark datasets obtained on a simple RNN architecture (orange and red) and on a GRU (blue and cyan). DFA is denoted by light colors and full lines; BPTT is in darker colors and dashed lines. Error-shades: one standard deviation over at least 5 repetitions.

Figure 4.4 shows the learning curves obtained in the datasets Libras movements dataset. Libras is a quite challenging dataset because has a multi-dimensional input (2-d), a sequence length of 90 steps, a limited dataset (360 samples) and 15 classes. It reveals that DFA is capable of training both RNNs and GRUs to non-trivial Test Accuracies. In this plot one can appreciate the fact that RNN DFA (orange line) has higher average accuracy than RNN BPTT (red line) after 150 Epochs. and that DFA has the same learning slope of BPTT either with vanilla RNNs (for the first 150 epochs) and for GRUs (for the first 50 Epochs). The learning curves on the other datasets can be found in Figure 4.6.

We report in Table 4.1 the maximum Test Accuracy that each DFA and BPTT achieved

Table 4.1: Summary of datasets statistics and average test accuracy and standard deviation over 5 repetitions for all datasets and models.

	Strawberry	LIBRAS	ECG200	Row-MNIST
Input size	1	2	1	28
Number of classes	2	15	2	10
Sequence length	235	90	96	28
Dataset size	983	360	200	70000
DFA GRU	$79.73 \pm 1.23$	$67.50 \pm 3.68$	$80.6 \pm 2.25$	$72.49 \pm 1.1$
BPTT GRU	$92.05 \pm 2.54$	$80.83 \pm 9.19$	$82.10 \pm 1.14$	$99.23 \pm 0.03$
DFA RNN	$67.84 \pm 2.66$	$47.92 \pm 3.3$	$78.2 \pm 1.47$	$87.48 \pm 0.74$
BPTT RNN	$79.08 \pm 4.18$	$54.30 \pm 18.32$	$83.30 \pm 2.1$	$96.69 \pm 0.24$

and the test accuracies of the two algorithms overlap if considering a tolerance of 1 standard deviation, except for the cases of RNN ECG200 (Figure 4.6) and of GRU Libras (as it can be seen in Figure 4.4) and both on the Strawberry dataset (see in Appendix 4.6.1, Figure 4.6). In the ECG dataset, which is the one with the smallest amount of data, In the Strawberry dataset, which is the one with the longest sequence, DFA’s Accuracy is above Random performances of at least 12% Test Accuracy. In this setting, BPTT is performing better than DFA up to only +1.53% RNN Test Accuracy.

In the Strawberry dataset, which is the one with the longest sequence, DFA’s Accuracy is above Random performances of at least 6% Test Accuracy. In this setting, BPTT is performing better than DFA up to +8.55% GRU Test Accuracy.

Overall, DFA effectively learns three datasets showing strength in a dataset with 2-dimensional inputs and 15 classes (Libras) and in a dataset with a reduced number of examples (ECG200). DFA shows less accuracy than BPTT in the Strawberry dataset with the longest sequences. The latter weaknesses could be solved by extending the range of the hyperparameters of validation.

## 4.7 Parallelization efforts

Figure 4.7 shows the time required by DFA and BPTT to compute an optimization step in the different settings (CPU or cuda; RNN or GPU).

The sums in the update equations of DFA are computed with a dedicated for loop after the forward pass of the algorithm. Figure 4.8 shows the time dedicated to the computation of this component. In the case of n-jobs=3, we used the python function Parallel of Joblib to parallelize the for loop over time that computes the weights update to sum, but it did not yield to faster code because of the time spent in creating new processes. Nevertheless, Figure 4.8 shows how much time could be saved by computing efficiently the for loop dedicated to the summations in the weights updates.

## 4.8 Conclusion

Unlike backpropagation, the update rule of our DFA can be applied in parallel for each time step, thus removing sequential processing during the update phase. Time series and sequential data are widespread in many real-world environments. However, implementing physical and adaptive dynamical systems remains a challenge.

DFA brings research one step further in this direction, this is why this work has been submitted to the workshop ”ML with New Compute Paradigms (MLNCP) at NeurIPS 2024”.

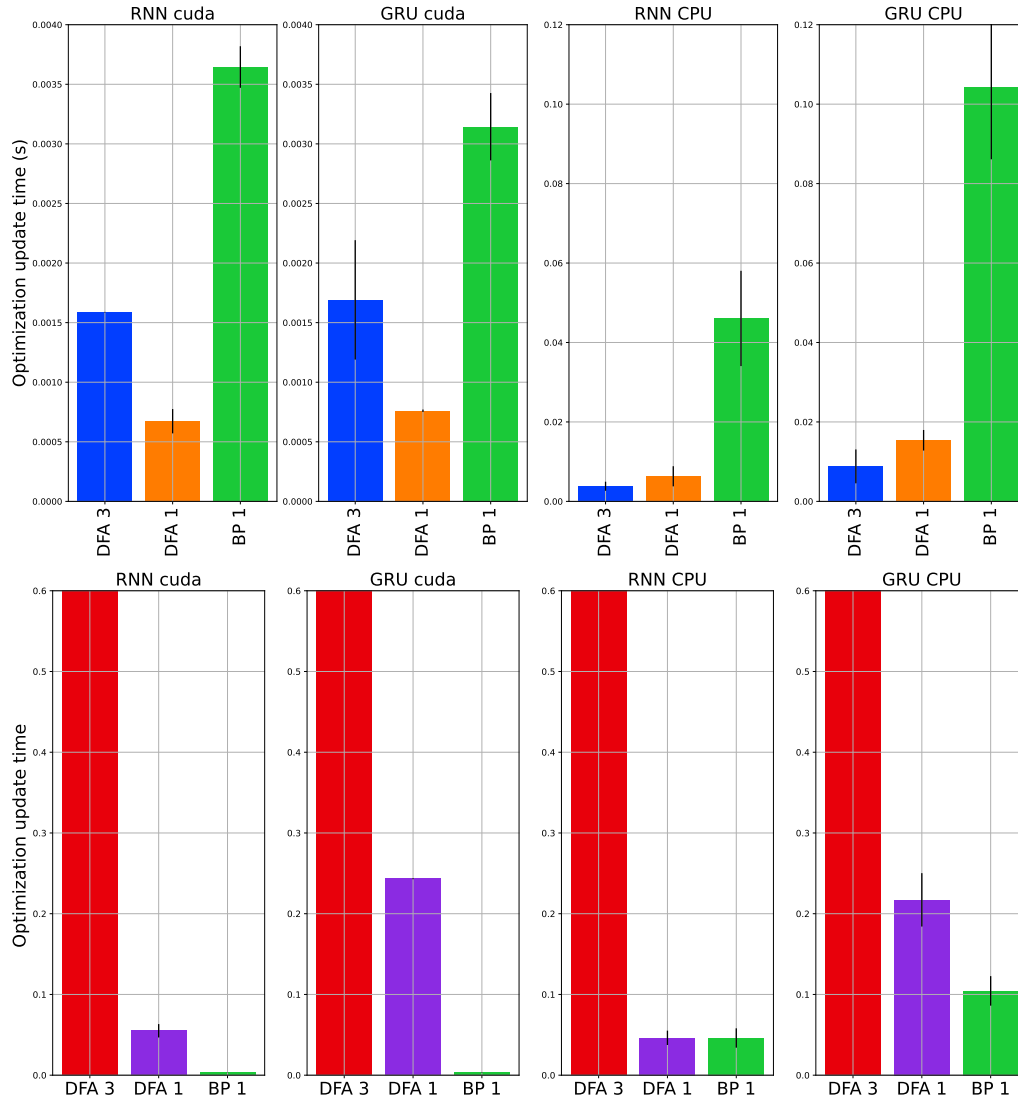


Figure 4.7: Timing of optimization step with (bottom) and without (top) summation loop. Errorbar indicates one standard deviations over 6 Epochs.

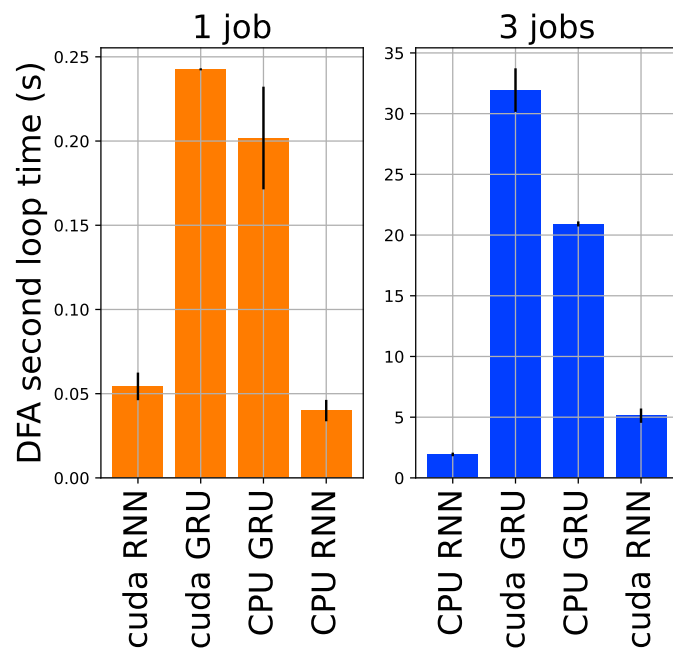


Figure 4.8: Timing of the forward pass of DFA dedicated to accumulate terms of weight update.

## Chapter 5

# Concluding perspectives

In this thesis, we investigated the potential of direct feedback alignment as an example of a biologically plausible learning rule to improve neural networks in the context of supervised learning, and classification tasks in particular. As an alternative to Backpropagation (BP), DFA offers improved biological plausibility. Biological networks such as brains have evolved to optimize the energy available and analyze, memorize, and extrapolate information. DFA reflects the fact that there is no explicit backpropagation of errors in the brain. The brain deals with this constraint and it is able to plastically change synapses to allow learning [Kennedy \[2013\]](#). From an optimization point of view, BP is effective; but optimizing solely for test accuracy does not always bring benefits to other aspects, like catastrophic forgetting, vulnerability to adversarial attacks, waiting time due to sequentially of weight updates, etc. There is a possibility that the mechanisms that DFA employs to learn with approximate gradients, which for now are known to be the phases of alignment and memorization, together with its technical advantages as parallel updates, will offer mitigation to such problems.

In this work, we expand the evaluation of DFA in a more holistic way that goes beyond optimization performances and evaluate neural networks along several dimensions. Our thesis explores three areas to evaluate DFA comprehensively. The starting investigation in this thesis looks at Continual Learning, where alongside the performance of the training tasks, the forgetting rate is in focus. The second investigation direction is ensembling, where robustness to adversarial attacks is enhanced. Finally, we described DFA’s behaviour in the context of Recurrent Neural Networks, where the time for the unfolding of the gradients is an issue. With the combination of these points of view, we get an interesting picture of the comparison between DFA and Backpropagation.

Our results are promising. We found that that in fully-connected networks trained on image classification tasks, DFA can alleviate catastrophic forgetting by constraining the network weights to a particular region in weight space when using the same feedback matrix across tasks, or by orthogonalising weight updates by using distinct feedback matrices for

each task. In the second perspective, we discovered that the characteristic degeneracy breaking mechanism of DFA allows to obtain samples for Ensembling in such a way that robustness to adversarial attacks is increased and the Test Accuracy reaches levels above Backpropagation. In the third chapter, we dealt with sequential data classification and we showed that DFA can be applied to Recurrent Neural Networks. In this context the parallel weight updates of DFA can solve the most challenging bottleneck of training time in RNNs: the computation time of the backpropagation through time algorithm.

Looking forward, combining these directions offers an intriguing plan for future research: for instance, using DFA in Bayesian neural networks to address Continual Learning. Bayesian methods for CL are an emerging research field, as allow for uncertainty estimation and enables the detection a new task [Zeno et al., 2020, Kessler et al., 2023, Li et al., 2020]. Moreover, a recent approach of Uncertainty-guided Continual Learning (UCB) [Ebrahimi et al., 2019] leverages the uncertainty in parameter distributions to regulate learning rates. This method allows BNNs to selectively freeze or update parameters based on their importance, thereby preserving critical information while accommodating new data.

The existing advantages of DFA can be extended, starting from the foundations of the biological realism of DFA. For example, adopting spiking Neural Networks [Lee et al., 2020, Zhang et al., 2024], that operate via discrete "spikes", akin to biological neurons, following the direction of Skatchkovsky et al. [2022].

Even in the non-spiking Neural Networks, the integration rules of DFA can borrow neuro-scientific features, as suggested in Dellaferrera et al. [2021]. In this case, the firing of each neuron is allowed only if it receives at least  $n$  positive inputs, while in the current implementation it is enough that one pre-synaptic neuron delivers a strong signal. This technique has been shown to be beneficial to Continual Learning.

Moreover, the synaptic adaptation is based on environmental feedback instead of a precise computation of errors. Biological brains do not have access to labels (real-world concepts) in the same shape as predictions (neural activation). In this perspective, a reward-based learning approach such as Reinforcement Learning (RL) reflects the biological mechanisms better. In RL, the network is required to take actions based on the input, and the feedback is received in terms of penalties or rewards. The goal is to learn a policy that maximizes the cumulative rewards over time. Solutions to Continual Learning have already been extended to Continual Reinforcement Learning with success, even with the means of small networks as the one considered in our work [Kirkpatrick et al., 2017].

While these applications explore DFA's immediate potential in neural networks, theoretical perspectives are also feasible directions. For example, the combination of the teacher-student setup for DFA [Refinetti and Goldt, 2022] with continual learning [Lee et al., 2021, 2022] may deepen our understanding of its underlying mechanisms, the trade-offs between DFA-same and DFA-diff and enhance their applicability offering new suggestions for practitioners.

Finally, another goal is to apply DFA to fine-tune pre-trained language and vision

models in the context of transfer learning. Low-rank adaptations (LoRa) techniques are popular fine-tuning adaptations for large-scale models where weights are fine-tuned by training a low-rank weight matrix on top of the fixed weights obtained from pre-training. This allows adaptation without modifying the entire network. A natural avenue for further work is studying the relation between DFA and LoRa adaptations: DFA can approximate learning in a way that may resemble lower-rank updates by reducing dependencies on precise weight matrices and gradients, and rank constraints can be applied directly to the feedback matrices.





# Acknowledgments

The decision to start a PhD was fueled by the desire to explore innovative ideas. However, it was not an easy choice—leaving a promising job to return to student life was daunting. For giving me the courage to take this leap, I am deeply grateful to my sister Elisa, my family, Fabio Ravalico, and Alessandro Laio.

The journey of the PhD has been brightened by the companionship of batchmates and friends: Nour El Kazwini, Konstantin Karchev, Romina Wild, Edward Donkor, Alex Zhang, and Gibbs Nwemadji Tiako. It has been an honor to share so many adventures. Together, we witnessed the growth of the Data Science community, and it has been a sincere pleasure to meet all the people in the department. I hope to stay in touch with all of them.

I owe special thanks to Sebastian Goldt and Alessandro Laio for their constant availability and invaluable ideas, as well as to Matteo Santoro, Erica Costantini, and Viplove Arora for their unwavering support throughout the years.

During the thesis phase, the encouragement of Dona Kireta, Santiago Acevedo, Federica Bastiani, Julia El Kazwini, Claudia Merger, and the whole Data Science community was essential. I am incredibly thankful for their belief in me.

A particularly enriching part of my PhD was volunteering with the SISSA Club language courses. I am grateful to Riccardo Ciccone for introducing me to this initiative. Over three years, I organized 48 courses, and it was deeply rewarding to contribute to the international community. Likewise, I am thankful to Ajay for introducing me to volunteering with Sant'Egidio, and I am deeply grateful to all the people I met there for their warmth and kindness.

In Trieste, Andrea Pulisci, Matteo Santoro, and Dmitrii Rachenkov supported me personally and professionally during my PhD journey, including sharing the dance floor with me—an experience that brought joy along the way. I also deeply appreciate the welcoming spirit of the people I met in Pisa and in Lugano, whose academic and personal guidance was invaluable.

Additionally, the Aikido and climbing were invaluable in helping me persevere through tough times. For their consistent company on the climbing walls and in the gym, I thank Antonio Merola and the lively members of the Aikido group.



# Bibliography

- A. Abdullah and M. Hassan. A review on bayesian deep learning in healthcare: Applications and challenges. *IEEE Access*, 10:1–1, 01 2022. doi: 10.1109/ACCESS.2022.3163384.
- M. Akrouf, C. Wilson, P. Humphreys, T. Lillicrap, and D. B. Tweed. Deep Learning without Weight Transport. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 976–984. Curran Associates, Inc., 2019.
- R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, 2018.
- S. Bartunov, A. Santoro, B. A. Richards, L. Marris, G. E. Hinton, and T. Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures, 2018. URL <https://arxiv.org/abs/1807.04587>.
- A. Bekasov and I. Murray. Bayesian adversarial spheres: Bayesian inference and adversarial examples in a noiseless setting. 11 2018. doi: 10.48550/arXiv.1811.12335.
- Y. Bengio, D.-H. Lee, J. Bornschein, and Z. Lin. Towards biologically plausible deep learning. *ArXiv*, abs/1502.04156, 2015.
- M. A. Bennani and M. Sugiyama. Generalisation guarantees for continual learning with orthogonal gradient descent. *ArXiv*, abs/2006.11942, 2020.
- F. Benzing. Unifying importance based regularisation methods for continual learning. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pages 2372–2396. PMLR, 2022. URL <https://proceedings.mlr.press/v151/benzing22a.html>.
- B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84, 12 2017. doi: 10.1016/j.patcog.2018.07.023.
- B. Bordelon and C. Pehlevan. The influence of learning rule on representation dynamics in wide neural networks, 2023. URL <https://arxiv.org/abs/2210.02157>.

- L. Bortolussi, G. Carbone, L. Laurenti, A. Patane, G. Sanguinetti, and M. Wicker. On the robustness of bayesian neural networks to adversarial attacks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2024. doi: 10.1109/TNNLS.2024.3386642.
- S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.
- T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.
- L. Cardelli, M. Kwiatkowska, L. Laurenti, N. Paoletti, A. Patane, and M. Wicker. Statistical guarantees for the robustness of bayesian neural networks. 08 2019. doi: 10.24963/ijcai.2019/789.
- N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. pages 3–14, 11 2017. doi: 10.1145/3128572.3140444.
- G. Carpenter. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 11 1986. doi: 10.1016/S0734-189X(87)80014-2.
- A. Celeghin, A. Borriero, D. Orsenigo, M. Diano, C. Guerrero, A. Perotti, G. Petri, and M. Tamietto. Convolutional neural networks for vision neuroscience: significance, developments, and outstanding issues. *Frontiers in Computational Neuroscience*, 17, 07 2023. doi: 10.3389/fncom.2023.1153572.
- A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-gem. *ArXiv*, abs/1812.00420, 2019.
- C. Chen, D. Carlson, Z. Gan, C. Li, and L. Carin. Bridging the gap between stochastic gradient mcmc and stochastic optimization. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1051–1060, Cadiz, Spain, 09–11 May 2016. PMLR. URL <https://proceedings.mlr.press/v51/chen16c.html>.
- J. Chen, T. Nguyen, D. Gorur, and A. Chaudhry. Is forgetting less a good inductive bias for forward transfer? *ArXiv*, abs/2303.08207, 2023. URL <https://api.semanticscholar.org/CorpusID:257532608>.
- X. Cheng, K. Fu, and F. Farnia. On the mode-seeking properties of langevin dynamics. 06 2024. doi: 10.48550/arXiv.2406.02017.

- L. Chizat, E. Oyallon, and F. Bach. On lazy training in differentiable programming. 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 2933–2943. Curran Associates, Inc., 2019.
- K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012.
- M. Chua, D. Kim, J. Choi, G. Lee, V. Deshpande, J. Schwab, M. Lev, R. Gonzalez, M. Gee, and S. Do. Tackling prediction uncertainty in machine learning for healthcare. *Nature Biomedical Engineering*, 7:1–8, 12 2022. doi: 10.1038/s41551-022-00988-x.
- J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Dec. 2014.
- B. Crafton, A. Parihar, E. Gebhardt, and A. Raychowdhury. Direct Feedback Alignment With Sparse Connections for Local Learning. *Frontiers in Neuroscience*, 13, May 2019. ISSN 1662-453X. doi: 10.3389/fnins.2019.00525.
- F. Crick. The recent excitement about neural networks. *Nature*, 337:129–132, 1989.
- S. d’Ascoli, L. Sagun, J. Bruna, and G. Biroli. Finding the needle in the haystack with convolutions: on the benefits of architectural bias, 2020. URL <https://arxiv.org/abs/1906.06766>.
- M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.
- G. Dellaferrera, S. Wozniak, G. Indiveri, A. Pantazi, and E. Eleftheriou. Learning in deep neural networks using a biologically inspired optimizer, 2021.
- L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- P. S. Dias Daniel and B. Helton. Libras Movement. UCI Machine Learning Repository, 2009. DOI: <https://doi.org/10.24432/C5GC82>.

- C. C. J. Dominé, L. Braun, J. Fitzgerald, and A. M. Saxe. Exact learning dynamics of deep linear networks with prior knowledge. *Journal of Statistical Mechanics (Online)*, 2023, 2023. URL <https://api.semanticscholar.org/CorpusID:258509118>.
- J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. De-caf: A deep convolutional activation feature for generic visual recognition. *ArXiv*, abs/1310.1531, 2014.
- A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
- S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. *ArXiv*, abs/1906.02425, 2019. URL <https://api.semanticscholar.org/CorpusID:174802369>.
- J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 1551-6709. doi: 10.1207/s15516709cog1402\_1.
- D. Erhan, A. C. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *International Conference on Artificial Intelligence and Statistics*, 2010. URL <https://api.semanticscholar.org/CorpusID:15796526>.
- S. W. Failor, M. Carandini, and K. D. Harris. Learning orthogonalizes visual cortical population codes. *bioRxiv*, 2021. doi: 10.1101/2021.05.23.445338.
- H. M. Fayek, L. Cavedon, and H. R. Wu. Progressive learning: A deep learning framework for continual learning. *Neural networks : the official journal of the International Neural Network Society*, 128:345–357, 2020.
- R. Feinman, R. Curtin, S. Shintre, and A. Gardner. Detecting adversarial samples from artifacts. 03 2017. doi: 10.48550/arXiv.1703.00410.
- M. J. Filipovich, Z. Guo, M. Al-Qadasi, B. A. Marquez, H. D. Morison, V. J. Sorger, P. R. Prucnal, S. Shekhar, and B. J. Shastri. Silicon photonic architecture for training deep neural networks with direct feedback alignment. *Optica*, 9(12):1323–1332, Dec. 2022. ISSN 2334-2536. doi: 10.1364/OPTICA.475493.
- S. G. Finlayson, J. Bowers, J. Ito, J. Zittrain, A. Beam, and I. S. Kohane. Adversarial attacks on medical machine learning. *Science*, 363:1287 – 1289, 2019. URL <https://api.semanticscholar.org/CorpusID:85446221>.
- T. Flesch, K. Juechems, T. Dumbalska, A. Saxe, and C. Summerfield. Orthogonal representations for robust context-dependent task performance in brains and neural networks. *Neuron*, 110(7):1258–1270.e11, 2022. ISSN 0896-6273. doi: <https://doi.org/10.1016/j>.

neuron.2022.01.005. URL <https://www.sciencedirect.com/science/article/pii/S0896627322000058>.

- K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. 1982.
- Y. Gal and L. Smith. Idealised bayesian neural networks cannot have adversarial examples: Theoretical and empirical study. 06 2018. doi: 10.48550/arXiv.1806.00667.
- X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL <https://proceedings.mlr.press/v9/glorot10a.html>.
- I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. pages 1–10, 01 2015a.
- I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014. URL <https://api.semanticscholar.org/CorpusID:6706414>.
- I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015b.
- S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cogn. Sci.*, 11:23–63, 1987.
- R. Hadsell, D. Rao, A. Rusu, and R. Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24:1028–1040, 12 2020. doi: 10.1016/j.tics.2020.09.004.
- D. Han and H.-J. Yoo. *New Algorithm 2: Extension of Direct Feedback Alignment to Convolutional Recurrent Neural Network*, pages 71–93. 05 2023. ISBN 978-3-031-34236-3. doi: 10.1007/978-3-031-34237-0\_4.
- D. Han, G. Park, J. Ryu, and H.-j. Yoo. Extension of Direct Feedback Alignment to Convolutional and Recurrent Neural Network for Bio-plausible Deep Learning, June 2020.
- L. Hansen and P. Salamon. Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12:993 – 1001, 11 1990. doi: 10.1109/34.58871.
- D. Hardoon, S. Szedmak, and J. Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural computation*, 16:2639–64, 01 2005. doi: 10.1162/0899766042321814.

- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- X. He and H. Jaeger. Overcoming catastrophic interference using conceptor-aided back-propagation. In *International Conference on Learning Representations*, 2018.
- M. Henne, J. Gansloser, A. Schwaiger, and G. Weiss. Machine learning methods for enhanced reliable perception of autonomous systems. 2021.
- N. Hiratani. Disentangling and mitigating the impact of task similarity for continual learning. *ArXiv*, abs/2405.20236, 2024. URL <https://api.semanticscholar.org/CorpusID:270123146>.
- J. Howard and S. Ruder. Universal language model fine-tuning for text classification, 2018.
- S. I. Jung and S. D. Ho. A study on hacking attacks and vulnerabilities in self-driving car with artificial intelligence. *Proceedings of the International Conference on Industrial Engineering and Operations Management*, 2022. URL <https://api.semanticscholar.org/CorpusID:258904575>.
- A. B. K. Kemsley. Strawberry. <https://timeseriesclassification.com/description.php?Dataset=Strawberry>.
- G. K V and V. Gripsy. Image classification using hog and lbp feature descriptors with svm and cnn. 03 2020.
- N. Kamra, U. Gupta, and Y. Liu. Deep generative dual memory network for continual learning. *ArXiv*, abs/1710.10368, 2017.
- R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan. Measuring catastrophic forgetting in neural networks, 2017.
- M. B. Kennedy. Synaptic signaling in learning and memory. pages 8(2), a016824, 2013. doi: 10.1101/cshperspect.a016824.
- S. Kessler, A. Cobb, T. Rudner, S. Zohren, and S. Roberts. On sequential bayesian inference for continual learning. 01 2023. doi: 10.48550/arXiv.2301.01828.
- J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521 – 3526, 2017.
- B. Kompa, J. Snoek, and A. L. Beam. Second opinion needed: communicating uncertainty in medical machine learning. *NPJ Digital Medicine*, 4, 2021. URL <https://api.semanticscholar.org/CorpusID:230719119>.



- S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3519–3529. PMLR, 09–15 Jun 2019. URL <https://proceedings.mlr.press/v97/kornblith19a.html>.
- A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL <https://api.semanticscholar.org/CorpusID:18268744>.
- A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=BJm4T4Kgx>.
- M. Laan, E. Polley, and A. Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 6:Article25, 02 2007. doi: 10.2202/1544-6115.1309.
- J. Launay, I. Poli, F. Boniface, and F. Krzakala. Direct feedback alignment scales to modern deep learning tasks and architectures. *ArXiv*, abs/2006.12878, 2020.
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539.
- D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. Difference target propagation, 2015. URL <https://arxiv.org/abs/1412.7525>.
- J. Lee, R. Zhang, W. Zhang, Y. Liu, and P. Li. Spike-train level direct feedback alignment: Sidestepping backpropagation for on-chip training of spiking neural nets. *Frontiers in Neuroscience*, 14:143, 03 2020. doi: 10.3389/fnins.2020.00143.
- S. Lee, S. Goldt, and A. M. Saxe. Continual learning in the teacher-student setup: Impact of task similarity. In *International Conference on Machine Learning*, 2021. URL <https://api.semanticscholar.org/CorpusID:235790418>.
- S. Lee, S. S. Mannelli, C. Clopath, S. Goldt, and A. M. Saxe. Maslow’s hammer for catastrophic forgetting: Node re-use vs node activation. *ArXiv*, abs/2205.09029, 2022. URL <https://api.semanticscholar.org/CorpusID:248863415>.
- S. Lee, S. Hong, G. Kim, and J. Ha. Ssim-based autoencoder modeling to defeat adversarial patch attacks. *Sensors*, 24:6461, 10 2024. doi: 10.3390/s24196461.

- C. Li, C. Chen, D. E. Carlson, and L. Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *AAAI Conference on Artificial Intelligence*, 2015. URL <https://api.semanticscholar.org/CorpusID:17043130>.
- H. Li, P. Barnaghi, S. Enshaeifar, and F. Ganz. Continual learning using bayesian neural networks, 2020. URL <https://arxiv.org/abs/1910.04112>.
- Y. Li and Y. Gal. Dropout inference in bayesian neural networks with alpha-divergences. 03 2017. doi: 10.48550/arXiv.1703.02914.
- Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:2935–2947, 2018.
- J. Lian, K. Choi, B. Veeramani, A. Hu, S. Murli, L. Freeman, E. Bowen, and X. Deng. Continual learning and its industrial applications: A selective review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 14, 09 2024. doi: 10.1002/widm.1558.
- T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1): 1–10, Nov. 2016a. ISSN 2041-1723. doi: 10.1038/ncomms13276.
- T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7, 2016b. URL <https://api.semanticscholar.org/CorpusID:10050777>.
- T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, pages 1–12, Apr. 2020. ISSN 1471-0048. doi: 10.1038/s41583-020-0277-3.
- S. Lin, P. Ju, Y. Liang, and N. Shroff. Theory on forgetting and generalization of continual learning. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 21078–21100. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/lin23f.html>.
- D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *NIPS*, 2017.
- A. Loquercio, M. Segu, and D. Scaramuzza. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, PP:1–1, 02 2020. doi: 10.1109/LRA.2020.2974682.
- M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, Aug. 2009. ISSN 1574-0137. doi: 10.1016/j.cosrev.2009.03.005.

- M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989. doi: [https://doi.org/10.1016/S0079-7421\(08\)60536-8](https://doi.org/10.1016/S0079-7421(08)60536-8). URL <https://www.sciencedirect.com/science/article/pii/S0079742108605368>.
- M. Mermillod, A. Bugajska, and P. Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 08 2013. doi: 10.3389/fpsyg.2013.00504.
- R. Michelmore, M. Wicker, L. Laurenti, L. Cardelli, Y. Gal, and M. Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. 09 2019. doi: 10.48550/arXiv.1909.09884.
- S. I. Mirzadeh, M. Farajtabar, R. Pascanu, and H. Ghasemzadeh. Understanding the role of training regimes in continual learning, 2020.
- T. M. Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.
- A. Momeni, B. Rahmani, M. Malléjac, P. del Hougne, and R. Fleury. Backpropagation-free training of deep physical neural networks. *Science*, 382(6676):1297–1303, Dec. 2023. doi: 10.1126/science.adi8474.
- M. Nakajima, K. Inoue, K. Tanaka, Y. Kuniyoshi, T. Hashimoto, and K. Nakajima. Physical deep learning with biologically inspired training method: Gradient-free approach for physical hardware. *Nature Communications*, 13(1):7847, Dec. 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-35216-2.
- C. Nemeth and P. Fearnhead. Stochastic gradient markov chain monte carlo. *Journal of the American Statistical Association*, 116:433 – 450, 2019. URL <https://api.semanticscholar.org/CorpusID:196832046>.
- A. Nøkland. Direct Feedback Alignment Provides Learning in Deep Neural Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1037–1045. Curran Associates, Inc., 2016.
- R. T. Olszewski, R. Maxion, and D. Siewiorek. *Generalized feature extraction for structural pattern recognition in time-series data*. PhD thesis, USA, 2001.
- OpenAI. Gpt-4 technical report, 2024.
- D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. 11, 12 1999. doi: 10.1613/jair.614.

- A. Passerini. Machine learning course handouts in deep learning. Technical report, UniTN, 2022-2023. URL <http://disi.unitn.it/~passerini/teaching/2022-2023/MachineLearning/>.
- M. Perrone and L. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. *Neural networks for speech and image processing*, 08 1993. doi: 10.1142/9789812795885\_0025.
- A. Radford, K. Narasimhan, et al. Improving language understanding by generative pre-training, 2018.
- A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.
- A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, pages 1313–1320, 2009.
- A. Rawat, M. Wistuba, and M.-I. Nicolae. Adversarial phenomenon in the eyes of bayesian deep learning. 11 2017. doi: 10.48550/arXiv.1711.08244.
- A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519, 2014.
- S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5533–5542, 2017.
- M. Refinetti and S. Goldt. The dynamics of representation learning in shallow, non-linear autoencoders. In *International Conference on Machine Learning*, pages 18499–18519. PMLR, 2022.
- M. Refinetti, S. D’Ascoli, R. Ohana, and S. Goldt. Align, then memorise: the dynamics of learning with feedback alignment. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8925–8935. PMLR, 2021. URL <https://proceedings.mlr.press/v139/refinetti21a.html>.
- A. ROBINS. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7 (2):123–146, 1995. doi: 10.1080/09540099550039318. URL <https://doi.org/10.1080/09540099550039318>.
- D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox,

- and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.
- D. E. Rumelhart and J. L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. MIT Press, 1987.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986. ISSN 1476-4687. doi: 10.1038/323533a0.
- S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0136042597.
- C. P. S. Qin, N. Mudur. Contrastive similarity matching for supervised learning. *Neural Computation* 33(5), 1300, 2021.
- A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013. URL <https://api.semanticscholar.org/CorpusID:17272965>.
- A. M. Saxe, J. L. McClelland, and S. Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116: 11537 – 11546, 2018. URL <https://api.semanticscholar.org/CorpusID:53024512>.
- A. Seff, A. Beatson, D. Suo, and H. Liu. Continual learning in generative adversarial nets. *ArXiv*, abs/1705.08395, 2017.
- W. Shao, J. Xu, Z. Cao, H. Wang, and J. Li. Uncertainty-aware prediction and application in planning for autonomous driving: Definitions, methods, and comparison. *ArXiv*, abs/2403.02297, 2024. URL <https://api.semanticscholar.org/CorpusID:268247939>.
- H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *NIPS*, 2017.
- K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.
- N. Skatchkovsky, H. Jang, and O. Simeone. Bayesian continual learning via spiking neural networks. *Frontiers in Computational Neuroscience*, 16, 11 2022. doi: 10.3389/fncom.2022.1037976.

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL <https://api.semanticscholar.org/CorpusID:604334>.
- M. Tang, Y. Yang, and Y. Amit. Biologically plausible training mechanisms for self-supervised learning in deep networks. *Frontiers in Computational Neuroscience*, 16, 2022. ISSN 1662-5188. doi: 10.3389/fncom.2022.789253. URL <https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2022.789253>.
- A. M. Turing. Computing machinery and intelligence. *Mind*, LIX:433–460, 1950. URL <https://api.semanticscholar.org/CorpusID:14636783>.
- T. Veniat, L. Denoyer, and M. Ranzato. Efficient continual learning with modular networks and task-driven priors, 2021. URL <https://arxiv.org/abs/2012.12631>.
- L. Wang, X. Zhang, H. Su, and J. Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(8):5362–5383, 2024. doi: 10.1109/TPAMI.2024.3367329.
- M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning*, 2011a. URL <https://api.semanticscholar.org/CorpusID:2178983>.
- M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML’11, page 681–688, Madison, WI, USA, 2011b. Omnipress. ISBN 9781450306195.
- P. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct. 1990. ISSN 1558-2256. doi: 10.1109/5.58337.
- L. Williams and A. Holtmaat. Higher-order thalamocortical inputs gate synaptic long-term potentiation via disinhibition. *Neuron*, 101, 11 2018. doi: 10.1016/j.neuron.2018.10.049.
- L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601 (7894):549–555, Jan. 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04223-6.
- H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

- N. Ye and Z. Zhu. Bayesian adversarial learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL [https://proceedings.neurips.cc/paper\\_files/paper/2018/file/586f9b4035e5997f77635b13cc04984c-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2018/file/586f9b4035e5997f77635b13cc04984c-Paper.pdf).
- J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.
- G. Zeng, Y. Chen, B. Cui, and S. Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, Aug. 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0080-x.
- F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. *Proceedings of machine learning research*, 70:3987–3995, 2017.
- C. Zeno, I. Golan, E. Hoffer, and D. Soudry. Task agnostic continual learning using online variational bayes with fixed-point updates. 10 2020. doi: 10.48550/arXiv.2010.00373.
- C. Zhang, Z. Li, Z. Shen, J. Xie, and H. Qian. A hybrid stochastic gradient hamiltonian monte carlo method. In *AAAI Conference on Artificial Intelligence*, 2021. URL <https://api.semanticscholar.org/CorpusID:235349259>.
- R. Zhang, X. Liu, and Q. Liu. A langevin-like sampler for discrete distributions. 06 2022. doi: 10.48550/arXiv.2206.09914.
- Y. Zhang, K. Inoue, M. Nakajima, T. Hashimoto, Y. Kuniyoshi, and K. Nakajima. Training spiking neural networks via augmented direct feedback alignment, 2024. URL <https://arxiv.org/abs/2409.07776>.
- G. Zhao, T. Wang, C. Lang, Y. Jin, Y. Li, and H. Ling. Dfa-gnn: Forward learning of graph neural networks by direct feedback alignment, 2024. URL <https://arxiv.org/abs/2406.02040>.