Department of Mathematics and Geosciences
Master Degree in Data Science and Scientific Computing

The Dynamics of Learning Beyond Stochastic Gradient Descent

Supervisor:
Prof. Sebastian Goldt

Candidate:
Dogan Can Demirbilek

Academic Year 2020/2021

To my family

Acknowledgements

Without the support of many people, I would not be able to finish this study. I am too grateful for their contribution and owe big thanks to all of them.
The internet is full of information, but it is not easy to find good quality. I want to start with the creators of the tools I have used for this study. I can not count how many times I used those sources; special thanks to those who created great content and made them accessible.
Sometimes, you can not find the problem about your implementation, but sharing remarkable ideas with other colleagues may open another perspective on it. Hence, I want to thank all the members of the Theory of Neural Networks Group from SISSA and the Data Science and Scientific Computing (DSSC) master program from the University of Trieste that we brainstormed together.
Thanks to the lectures that I had from DSSC, I was comfortable enough to move on with this topic, so I want to present my appreciation to all professors of DSSC.
The entire process was crystal clear and organized, starting from our first meeting. I learned too much from him, and without his invaluable support, I could not finish this study on time. Therefore, special thanks to my esteemed supervisor, Sebastian Goldt.
I must be grateful to all of my friends for helping me maintain social bonds. Nowadays, we realize how important it is. Thanks to their friendship, I could balance my study and social life in this challenging period and study more efficiently on this thesis.
Finally, I want to present the last but enormous thanks to my family. They are always there when I need motivation or struggle with a problem.

Dogan Can Demirbilek
Trieste
17 December 2021

Abstract

Backpropagation is the main algorithm to train neural networks; however, it is not the only alternative. Different algorithms propose more biologically plausible solutions. Here, we perform various experiments on particular problems by using backpropagation and one of the alternative algorithm called direct feedback alignment. The learning problems we study are used in previous studies to show the superiority of backpropagation over linear methods, such as random features, which are sometimes referred to as lazy methods. Our main aim is to compare the predictive power of the direct feedback alignment to backpropagation on the same learning problems and improve its performance with tuned hyperparameters and modern adaptation techniques. Besides, we perform experiments that help understand both methods' learning dynamics. We discover that direct feedback alignment can perform equally well or close to backpropagation on the chosen tasks with sensitive tuning and suitable modern adaptive techniques. We hope that our study would inspire to test direct feedback alignment from various perspectives on more challenging problems.

La retropropagazione dell'errore (in inglese backpropagation) è il più noto algoritmo impiegato per l'addestramento delle reti neurali. Tuttavia, essa non costituisce una scelta obbligata; vari algoritmi alternativi propongono soluzioni più plausibili da un punto di vista biologico. In questo lavoro, conduciamo diversi esperimenti su specifici problemi utilizzando backpropagation e uno degli algoritmi alternativi chiamato direct feedback alignment. I problemi di apprendimento che analizziamo sono stati utilizzati in studi precedenti per dimostrare la superiorità della backpropagation rispetto a metodi lineari, come quelli basati su random features che a volte vengono definiti lazy methods. Il nostro principale obiettivo è di comparare il potere predittivo del direct feedback alignment e della backpropagation sugli stessi problemi di apprendimento e di migliorare le prestazioni con un'accurata selezione degli iperparametri e l'uso di moderne tecniche adattive. Inoltre, produciamo degli esperimenti che aiutano a capire le dinamiche di apprendimento di entrambi i metodi. Le prestazioni ottenute con direct feedback alignment, con un'opportuna configurazione e uso di moderne tecniche adattive, sono confrontabili con quelle della backpropagation sui problemi selezionati. Speriamo che il nostro studio possa ispirare future ricerche sull'applicazione di direct feedback alignment su problemi più sfidanti.

Contents

List of Figures

List of Abbreviations

ANN . . . . . Artificial Neural Network.

BP . . . . . Backpropagation.

DFA . . . . . Direct Feedback Alignment.

BCE . . . . . Binary Cross Entropy.

FA . . . . . Feedback Alignment.

NTK . . . . . Neural Tangent Kernel.

SGD . . . . . Stochastic Gradient Descent.

NAG . . . . . Nesterov Accelerated Gradient.

t-SNE . . . . . t-Distributed Stochastic Neighbor Embedding.

Introduction

Artificial neural networks (ANNs) are a collection of connected computational nodes inspired by biological neural networks. Each connection can transmit a helpful signal to another computational node like synapses in a brain. ANNs demonstrated colossal advancements in the last decades. Thanks to these advancements, it is possible to solve complex problems in computer vision, speech recognition, and natural language processing within a reasonable amount of time and with satisfactory performance. These advancements were actualized through an old but robust algorithm called **backpropagation** (BP). BP is a training algorithm for ANNs based on repeatedly adjusting network weights to minimize the difference (loss) between the output of the network and the ground truth [1]. In the training phase, the information from data is encoded via the minimization of the loss. After this phase, the network is ready to be tested on a validation dataset, and finally, it can be used to infer new information about the new data.

Although nowadays BP is the workhorse algorithm for training ANNs, it has some drawbacks, and it is not the only alternative. Recent studies offered different algorithms to train ANNs by addressing these drawbacks. These algorithms have other properties and principles than BP. Some of them are competitive with BP, or they even outperform it in terms of performance or convergence speed for specific problems.

This thesis investigates the learning structures through BP and one of the alternative algorithm called **direct feedback alignment** (DFA) on the particular learning problems. Unlike BP, the error is propagated through a fixed random matrix parallelly instead of the layers' weights sequentially in DFA. Then network learns how to make this feedback useful [2]. Due to this error propagation mechanism,

DFA is considered more biologically plausible than BP, and it opens the gate of parallelism in the training phase of ANNs.

The fundamental interest of this study is in benchmarking the performance of BP and DFA on a given dataset. In other words, it is intriguing to compare the BP and DFA on a set of challenging learning problems. For this purpose, we consider two learning problems. The first learning problem we use is known as the **parity learning** problem. The parities are well-known to be difficult to learn for ANNs [3], and one of the previous studies showed that these parities are learnable by BP and linear methods in a more simple setting, whereas it is only learnable by BP in a more complex setting [4]. The second learning problem is an example for a task where the inputs have a lower-dimensional structure [5], and the number of dimensions of the data can be adjusted to decide the difficulty of the problem. Both learning problems proved the superiority of the BP compared to the linear methods, such as random features, which are sometimes referred to as lazy methods. That is why it is intriguing to test alternative algorithms on these problems to understand their learning dynamics and capabilities by observing the difference if there is any. The experiment results might lead us to three possible outcomes. First, we might acquire a similar performance as BP. It would be beneficial to test DFA and BP on a more challenging task if it is the case. Second, there might be a gap between BP and DFA then it would be intriguing to understand where the difference is coming from and how we can close this gap. Third, the alternative algorithm might not even learn, and in this case, it is interesting to ask what makes a problem learnable by BP but not DFA. In all cases, results should help understand both methods' learning dynamics.

For applying BP and DFA in a more realistic setting, parity experiments are performed on the MNIST dataset by imitating the learning problem as it is described in [4]. Then, they are also tested on the synthetic data to compare the predictive power of BP and DFA. After putting DFA to these frames, the reason behind the results is interpreted, and possible improvements are motivated and implemented. Apart from the introduction and conclusion, this thesis comprises three chapters.

For the fast readers, all of them have a section highlight. The content of them is explained as the following:

Chapter 1 constructs the theoretical bases of the algorithms that are used for the experiments. These bases are composed of simple definitions, mathematical foundations, and the drawbacks of the algorithms. They help to dig deeper into the learning structures of the training algorithms. It is expected to have more control over their learning behaviors by adjusting components of these foundations. Also, it is beneficial to have these theoretical bases for acquiring a better understanding of the further interventions. Moreover, these theoretical foundations are used to implement the algorithms from scratch to use in experiments.

Chapter 2 introduces the learning problems that we use. First, we start with the parity problem and continue with the synthetic data problem. We explained how their data are generated in detail for both problems and why they are interesting to test BP and DFA. This part is also highly correlated with the training phase of the algorithms. Therefore training phase and simple hyperparameter tuning process are explained in this chapter. Later the experiment results from previous studies [4, 5] are reproduced to have a concrete picture.

Chapter 3 presents the results of the experiments. This chapter is the main contribution of this study. The first experiments test DFA on the same learning problems and compare with BP. As specified before, different further experiments are performed depending on the first experiment outcome. Such as closing the gap between BP and DFA, if any, and trying harder problems if we acquire similar results or explaining why DFA cannot learn the problem if we get a similar behavior as lazy methods. In addition to the main experiment, we perform other experiments because they might help us improve the performance of the algorithms, or they can be helpful to understand the learning dynamics of training algorithms. For instance, using different random matrices for DFA, trying adaptive optimization algorithms with BP and DFA, and for parity problem, observing hidden representation of BP and DFA to understand if the networks learn the digits individually to calculate the parities or capture different information.

Conclusion refers to each chapter briefly and wraps up the outcomes of the experiments by summarizing the key findings. It gives a general idea about the comparison of DFA and BP. It also creates a path for future studies that are not covered in this study.

1 Theoretical Foundations

Contents

The following algorithms, methods are used in the experiments. Therefore, it is essential to embody some of their critical aspects for this study's completeness. However, detailed explanations are out of the scope of this study. On the other hand, some references might help the readers interested in going deeper. Given that, the following algorithms, methods are explained: the training algorithms for neural networks (BP and DFA), linear methods over a fixed representation of the data (lazy methods), optimizers, and dimensionality reduction technique that is used. For this and the following chapters, readers can refer to the list of abbreviations at the beginning.

1.1 Backpropagation

BP is one of the first algorithms that show ANNs could learn hidden representations well. Numerous studies showed that ANNs trained with BP could capture similar information as biological neural networks (e.g., specific nodes learn the edges, corners). We need three components for capturing useful information via BP, a dataset composed of input-output pairs, a network consisting of parameters (weights and biases), and it allows the input to flow through the network to have output. We need a loss function to measure the difference between the output of the network and the ground truth that we have from the dataset.

The main goal of BP is computing the gradients of the loss function (a measure of difference) concerning the parameters of neural networks by using the chain rule. These gradients show how much the parameter needs to change (positively or negatively) to minimize the loss function. After efficiently calculating the gradients, we can adjust the network parameters using gradient descent or its variants.

$$\text{parameter} = \text{parameter} - \text{step size} \times \frac{\partial Loss}{\partial(parameter)}$$

Although BP is an old idea, it gained popularity with [1] where they presented how BP can make a network to learn the representations. After the idea was spread, the community published numerous practical and theoretical papers that investigated the dynamics of BP. It would be infeasible and a repetition to show all the aspects again. However, for completeness and a smoother transition from BP to DFA, it is beneficial to have visual and mathematical explanations showing how the algorithms propagate the errors and the weights. For the mathematical foundations, a binary classification task is demonstrated with binary cross-entropy (BCE) loss as an example in appendix A. BCE measures the difference of predicted probabilities and the actual class, which can be either 0 or 1. It gives a score that shows how close the predicted probabilities and the ground truth are by penalizing the distance. Therefore, it is widely used for binary classification tasks. This example is not chosen arbitrarily. Indeed the parity problem that MNIST imitates and synthetic data

classification problem is binary classification problems. In addition to this, equations from appendix A are used to implement BP from scratch to have more control over the process. Then the exact implementation is modified to obtain DFA. The same steps are valid for different loss and activation functions. Only the calculations will be slightly different. The general idea is the same: obtaining the gradients by calculating the derivative of the loss function concerning the parameters.

1.1.1 Drawbacks of BP

We know that biological neurons inspired ANNs. However, recent studies showed that BP is not precisely how biological neurons learn [6]. That is why the community proposed many alternative algorithms by addressing these limitations of BP. This brings a term called biological plausibility of an algorithm that indicates the algorithm's consistency with existing biological, medical, and neuroscientific knowledge. In the light of this term, we can put in order the drawbacks of BP as the following:

- **Biological implausibility:**
  - The BP computation is purely linear, whereas biological neurons interleave linear and non-linear operations.
  - BP needs precise knowledge of derivatives of the non-linearities at the operating point used in the corresponding feedforward computation on the feedforward path.
  - BP has to use the transpose of the weights of the feedforward connections to calculate the gradients of the loss. Transposing a large matrix is computationally costly. In literature, this issue is known as the weight transport problem, and it is one of the most criticized disadvantages of BP.
  - Real neurons communicate by binary values (spikes), not by clean, continuous values.

  - The computation has to be precisely clocked to alternate between feedforward and BP phases.
  - It is not clear where the output targets would come from. [6, 7]

- **Vanishing or Exploding Gradients**
- **Lack of Parallel Processing** [8]
  - In BP, the backward process is sequential. Meaning that process is executed starting from the last layer to the first layer by visiting all the middle layers.

Simple interventions may handle some of these drawbacks. For instance, implementing gradient clipping or different activation functions might solve exploding gradients and vanishing gradients. However, they may frequently happen in deeper networks, and they must be considered while training ANNs. On the other hand, some of the drawbacks can not be handled with superficial modifications. For instance, BP is a sequential process, and there are locking mechanisms (forward, backward, and update) that ensure none of the processes is executed before its preceding completion. This makes BP infeasible for parallel processing because each execution has to wait for its preceding process. Hence deeper and larger networks' training can be computationally expensive.

Biological plausibility is significant because of a couple of reasons. We know that biological neurons inspired ANNs. Hence, it is interesting to examine the dissimilarity or similarity among them. Besides, 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, especially for this field. Furthermore, even though nowadays ANNs might outperform the human brain in a specific task, we are still distant from fully mimicking it. In other words, most of the time, ANNs are very good on a task that they are trained in, but they are not diverse, and some kinds of attacks like adversarial ones can easily trick them. Adversarial attacks are trying to fool the machine learning models by using malicious inputs [9]. Investigating the

learning dynamics of these algorithms may open the doors of diverse ANNs that are not specialized in a single task or make them more robust to attacks.
Alternative algorithms address some of the drawbacks of BP, and they propose a solution to them, but they also demonstrate a couple of them. However, these algorithms can be considered one or more steps closer to more biologically plausible and robust algorithms.

1.2 Direct Feedback Alignment

So far, we have mentioned that the error is propagated in BP sequentially through a network with the backward pass. Unlike BP, DFA uses a different way to propagate the error. This way uses a random matrix **parallelly** instead of the transpose of the weight matrix sequentially, which brings a solution to the weight transport problem. Before explaining how DFA works, it is better to investigate the **feedback alignment** (FA) algorithm since DFA is the extension of FA. FA is an algorithm for training ANNs where, unlike BP, a random matrix is used **sequentially** to propagate the error instead of the transpose of the weights.
P. Lillicrap et al. [10] proved that precise symmetric weights are not required to obtain learning in ANNs. Without these weights, BP-like learning can be obtained. Any random matrix under some conditions can provide the learning. Implicit dynamics in the standard forward weight updates encourage an alignment between weights and the random matrix. In other words, a random matrix pushes the network in roughly the same direction as BP would. They named this training method feedback alignment. The only difference between BP and FA is the random matrix that is used to propagate the error. They test FA on a linear problem and MNIST classification task. The empirical results demonstrate that FA successfully trains the network and has similar performance results as BP on these tasks.
Even though learning still occurs with random matrix and FA offers the solution to the weight transport problem, it does not provide any computational advantage. Because in FA, the backward process is still sequential. Hence, to extend FA to DFA, we need to change the error propagation mechanism of FA slightly. In

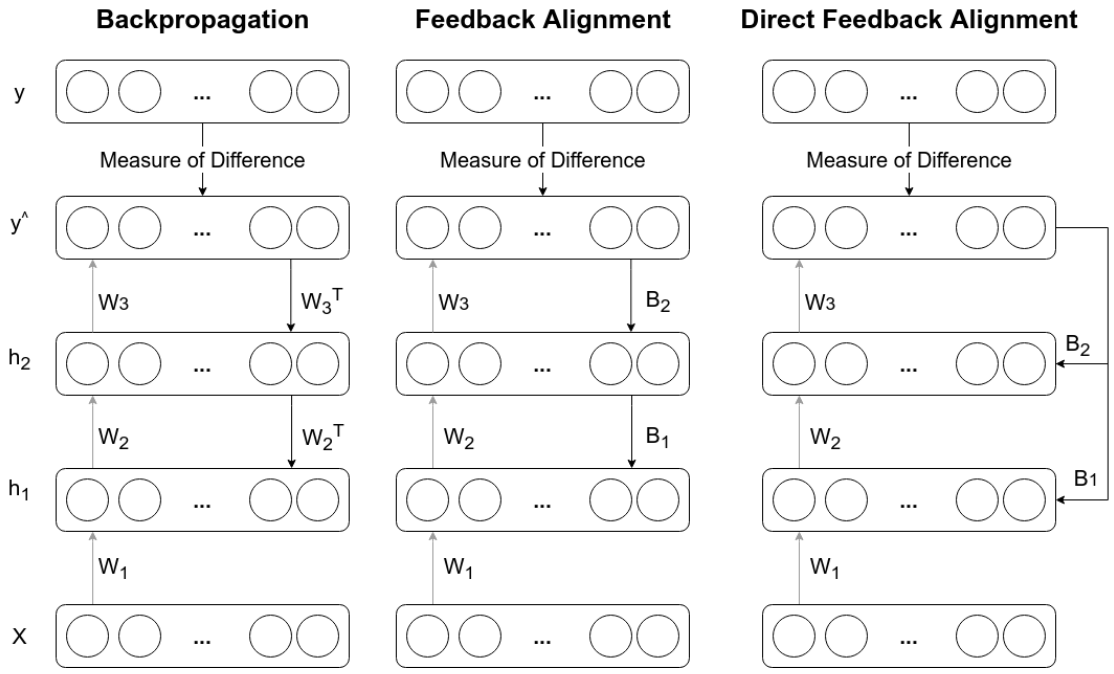

Figure 1.1: Error Transportation in BP, FA, and DFA
Visualization of the error transportation configurations for BP, FA, and DFA for two-layer networks [2]. W_i are the weights, h_i are the output of the hidden layers that is denoted as i, $\hat{y}$ is the output of the network, and y is the ground truth, for the sake of simplicity, biases are not showed in this figure.

DFA, the error is still propagated with the random matrix but parallel to each layer. In other words, DFA takes the loss and distribute it globally to all layers without requiring sequential step. It also creates an opportunity to parallelize the computation that might speed up the training process.

The figure 1.1 shows the difference in the algorithms' error propagation mechanisms that we mentioned before. In our experiments, we have used architecture with only one hidden layer, not the one in figure 1.1. In that case, FA and DFA are identical because, with one layer, there is no room to propagate the random matrix in parallel.

It is crucial to point out, BP and DFA have different learning dynamics. BP calculates the gradients that point to the steepest descent in the loss function space. On the other hand, FA and DFA provide a different update direction but still descending. It is still descending because empirical and theoretical results proved that the networks' weights align with the random matrix that leads to gradients

alignment. Therefore the more alignment we have, the same direction FA and DFA point as BP. However, it does not mean that more alignment is better performance because FA and DFA might find different and possibly better paths that converge to the global minimum. Even though they have different update directions, since they are both descending, the results from [2, 10] showed that FA and DFA are as good as BP in terms of performance for specified tasks in these papers. In addition to this, ANNs trained with DFA show decent separation for labels as in BP's hidden representations of the layers. It means that DFA captures similar information as BP.

Recently, a new study has been published which tests the applicability of DFA on modern deep learning tasks and architectures such as neural view synthesis, recommender systems, geometric learning, and natural language processing [11]. Because even though some of the alternative training algorithms that are competitive with BP in simple tasks like MNIST are not competitive or trainable on more complex tasks. This study showed that DFA successfully trains all these complex architectures with performance close to BP. Moreover, this study supports that complex tasks can be solved without symmetric weight transport, proving that DFA is suitable for more challenging problems.

Let us use the same example as A to present how gradients are calculated in DFA. After having the mathematical foundations of BP, transition to DFA is relatively easy. The forward pass is the same as BP, whereas, in the backward pass, we need to replace the transpose of the weight matrix, which is used to calculate the gradients with the random matrix. Considering the same example, we have a simple binary classification task with BCE loss, and our network has only one hidden layer. In this setting, gradients of the weights can be calculated as the following:

$$\frac{\partial BCE}{\partial w_2} = (\hat{y} - y)\, h_1$$

There is no change in the calculations of gradients of the last layer, whereas, for the hidden layer, we have:

$$\frac{\partial BCE}{\partial w_1} = (X)\,(\hat{y} - y)\,(B) \odot f'(a_1)$$

Where y is the ground truth and $\hat{y}$ is the output of the network, h_k is the output of the layer (which means that $h_0 = X$ and $h_2 = \hat{y}$). $a_k = h_{k-1}w_k + b_k$ where w_k is the weight, b_k is the bias term and f is the non linear function. Please note that w_2 is replaced with the random matrix B. This means we can obtain learning by changing either the random matrix or weight matrix. We know that in DFA, B is fixed, so the feedforward weights of the network will learn to make these signals useful by aligning with the BP's teaching signal.

Update rules are the same as BP, which means gradient descent and its variants can be used. With this tiny modification, DFA brings a solution to some of the drawbacks of BP. Such as using exact symmetric weights of the feedforward connections (weight transport problem), lack of parallel processing (random matrix can be propagated in parallel), and it is less likely to suffer from vanishing or exploding gradients than BP. Eventually, it proposes a more biologically plausible training method. However, it is not the perfect solution either. Because it assumes there is a global feedback path to propagate the error that might be biologically implausible because feedback has to travel a long physical distance. It also suffers some of the drawbacks of BP. For instance, computation is still purely linear. We still need precise knowledge of derivatives of non-linearities. We still communicate by clean, continuous values, and it is unclear where the output targets would come from. Besides, DFA has an extra task to accomplish while training the ANN that aligns with BP's teaching signal, and a layer can not learn before its preceding layers are aligned. This might spawn performance concerns, and DFA might lag behind BP. Furthermore, DFA fails to train convolutional neural networks which dominate the computer vision tasks [12, 13]. Finally, unlike BP, DFA was not investigated on particular subjects like adversarial attacks and interpretability by the community. This leaves some question marks about the robustness of DFA.

1.3 Lazy Methods

Theoretical results present that especially over-parameterized ANNs (not limited to these networks) trained with gradient-based methods might reach zero training loss

with their parameters barely changing (if they converge). The term lazy does not refer to the poor property of methods, whereas they are called lazy methods because their parameters hardly move during training [14]. This study uses the term lazy methods and linear methods interchangeably because the following algorithms show both properties. These properties are: essentially they are linear methods over a fixed representation of the data, and during the training phase, their parameters do not change too much (fixed representation).

Lazy methods are not at the center of our experiments. Hence, detailed explanations of these methods are out of scope in this study, but they have been presented in [4, 5], and they fail to learn the parities in a more complex setting. Also, they demonstrated poorer performance on the synthetic data problem. Given that, we will implement those methods too as a baseline and for completeness. We will also embody at least a simple definition of them and how they are practically implemented.

For this purpose, the following lazy methods will be explained: neural tangent kernel, gaussian features, reLU features, and linear features.

1.3.1 Neural Tangent Kernel

Studies showed that neural networks under some conditions are equivalent to a Gaussian process, and they mathematically approximate the kernel machines if they are trained with gradient descent [15, 16]. Jacot et al. [17] proved that during the training phase, ANNs follow the gradient of the loss converging to a kernel. They named this kernel a Neural Tangent Kernel (NTK). In other words, NTK is a kernel that describes the evolution of an ANN during the learning phase, and it is beneficial to explain the training of ANNs in function space rather than parameters space. It allows us to work with infinite width neural networks using the kernel trick, and it helps us understand the dynamics of learning and inference.

Empirical results demonstrated that the NTK regime performs worse than BP on standard tasks like MNIST. However, NTK is still worth investigating further to understand ANNs' training dynamics since it brings a new perspective on the

training phase.

The practical implementation of NTK is obtained as the following: "we used an architecture that decouples the gating from the linearity of the ReLU, and we kept the gates fixed during training. This means, by using $\text{ReLU}(\langle \mathbf{w}, \mathbf{x} \rangle) = \langle \mathbf{w}, \mathbf{x} \rangle \cdot \mathbf{1}\{\langle \mathbf{w}, \mathbf{x} \rangle\}$ and by decoupling the first and second term during the optimization, the network is forced to stay in NTK regime. [4]" In other words, an extra layer is created with the exact dimensions of the first layer. In the forward pass, concatenation of these two layers' parameters is given as input to the gated linear unit function. Lastly, the extra layer is not considered in the parameters update phase.

1.3.2 Random features

In standard random features, the first layer weights are initialized randomly by a distribution that we will specify in the following. The train is performed only for the second layer weights. In this way, the inputs are mapped into a fixed embedding space before a logistic separation. These mechanisms are particularly good at approximating kernels. They are preferred over kernels because the latter might take too much time to train if the data size is big. In **gaussian features** case, we initialize the first layer weights using a gaussian distribution. In **ReLU features** and **linear features**, we initialize the first layer weights by a uniform distribution. The distinction of the last two regimes is that in the forward pass, non-linear activation functions are replaced by linear ones for linear features.

1.4 Optimizers

Up to this point, we only mentioned how we could use gradient descent and its variants to update the weights of a network superficially. This part is worth further investigation because many variants provide better convergence properties to find the minimum of the loss function. We may take advantage of these methods to have better performance or faster convergence for BP and DFA. These methods may spawn a significant impact on convergence speed and overall performance. As

a reference to the following methods, mostly [18] is used, also the structure of this part and mathematical notations are adapted from the same paper, it is an excellent overview for the optimizers, and it reviews their advantages as well as drawbacks.

1.4.1 Gradient Descent

Gradient descent is a first-order iterative optimization algorithm. It is the most used algorithm to optimize neural networks. It has three variants that depend on how much data we use to compute the gradients. **Batch gradient descent** computes the gradients for the entire dataset and performs only one update. **Stochastic gradient descent** (SGD), in contrast, calculates gradients for each training example and performs parameter update for each of them. Lastly, **mini-batch gradient descent** calculates the gradients of mini-batches and performs updates for each mini-batches. Gradient descent is infeasible for the datasets that do not fit in the memory. In contrast, SGD performs too frequent updates, spawning high variance in parameters that cause fluctuation in the loss function. SGD provides the same convergence properties as batch gradient descent if the learning rate periodically decreases through iterations. We used mini-batch gradient descent for our experiments, which takes the best of two methods. Most of the implementations use SGD term instead of mini-batch gradient descent. The same tradition will be followed in this study too. Update rule of mini-batch gradient descent is the following:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right)$$

where θ is the parameters of the network, η is the learning rate or step size, ∇_θ is the gradients of the parameters and $J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right)$ is the loss function for mini-batch i to $i+n$.

There are a couple of challenges in SGD because it doesn't always guarantee good convergence:

- Choosing a proper learning rate is intricate. Low learning rates may take too long to converge, whereas big learning rates may spawn loss function fluctuations and even diverge.

- SGD does not guarantee the global minimum. It can easily be stuck in the local minimum for highly non-convex loss functions standard for deep learning tasks.
- Same learning rate is applied to all parameters, but we may want to update the parameter by their frequencies.
- Convergence is strongly dependent on where the initial step starts. Unfortunate initializations may never reach the global minimum.

Momentum

SGD has difficulties finding the direction in valleys because the gradients on these areas will be either zero or very close to zero, so it will slow down and make hesitant progress. These areas are prevalent around the local minimum. Momentum is an idea that dampens the oscillations in the relevant direction. It is accomplished by adding a fraction γ of the update vector of the past time step. This fraction is usually set to 0.9. This term usually leads to faster convergence and speeds up the iterations.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta_{t+1} = \theta_t - v_t$$

However, momentum follows the direction of the gradients blindly, **nesterov accelerated gradient** (NAG) is a way of giving our method to intuition by approximating the next position of the parameters with $\theta - \gamma v_{t-1}$, with this we hope to slow down before the hill slopes up. In other words, first, as in the momentum method, we make a big jump in the direction of previous gradients, then we measure the gradients where we end up and make a correction. The new update rule becomes:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J\left(\theta - \gamma v_{t-1}\right)$$
$$\theta_{t+1} = \theta_t - v_t$$

1.4.2 Adaptive Methods

Two main drawbacks of SGD are: tuning the learning rate is complex, and we use the same learning rate for each parameter. Adaptive methods offer solutions to

these problems. They use intelligent ways to modify the learning rate that may differ from parameter to parameter, and some of them even remove the need to set the learning rate. However, they are still gradient-based algorithms with some modifications, and they do not always guarantee convergence.

Adagrad

In vanilla SGD and SGD with momentum, we used the same learning rate for each parameter. On the contrary, adagrad adapts the learning rates for each parameter. It performs larger updates for infrequent features and smaller updates for frequent features. By saying infrequent features, we mean the cases where the component of the feature has a value different than the most common one. Usually, this component is important and informative. These values are called infrequent features. To do this, it updates the learning rate at each time step t for each parameter based on their past gradients concerning the loss function.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

With this update rule, the learning rate is modified at each time step. G_t contains the sum of squares of the past gradients for all parameters. g_t is the gradients of all parameters at time step t, and ϵ is the smoothing constant to avoid zero division, and it is usually set to 10^{-8}. G_t is getting larger with each step since we only add positive terms that make the learning rate very small, and the algorithm cannot learn any more in advancing time steps.

Adadelta

Adadelta is an extension of Adagrad, which tries to solve the decreasing learning rate problem and tries to remove the need for tuning the learning rate manually [19]. Instead of using the squares of all past gradients, Adadelta sets a moving window of gradient updates, and by doing so, it continues learning even after many iterations. It does by storing the exponentially decaying average of the squared gradients.

$$E\left[g^2\right]_t = \rho E\left[g^2\right]_{t-1} + (1-\rho) g_t^2$$

$E\left[g^2\right]_t$ is the running average, ρ is the decay constant which is similar to momentum term (it is usually set to around 0.9 like momentum). The demonitor of the update rule of adadelta is very similar to adagrad, only difference is G_t is replaced with $E\left[g^2\right]_t$. The term $\sqrt{E\left[g^2\right]_t + \epsilon}$ can be rephrased as root mean squares of the previous gradients up to time t.

$$\text{RMS}[g]_t = \sqrt{E\left[g^2\right]_t + \epsilon}$$

Where ϵ is a smoothing constant for avoiding any problem in the denominator, by using this term, we can change the update rule of Adagrad to the following:

$$\theta_{t+1} = \theta_t - \frac{\eta}{RMS[g]_t} \odot g_t$$

For clarity, we can rephrase the update rule as follows:

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

where;

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} \odot g_t$$

D. Zeiler et al. [19] pointed out that parameters updates in SGD, momentum, and Adagrad does not match with the units of the parameters. The units relate to the gradients, not the parameters. They defined an exponentially decaying average of parameters instead of gradients to overcome this issue.

$$E\left[\Delta\theta^2\right]_t = \rho E\left[\Delta\theta^2\right]_{t-1} + (1-\rho)\Delta\theta_t^2$$

The root means squared error of the parameters is:

$$\text{RMS}[\Delta\theta]_t = \sqrt{E\left[\Delta\theta^2\right]_t + \epsilon}$$

Since $\text{RMS}[\Delta\theta]_t$ is unknown at time step t, it is approximated with previous time step.

$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

Normally this final update would remove the need for a learning rate. However, we follow the same approach as in Pytorch implementation [20] so the last term is scaled with learning rate, which finally yields to update rule of Adadelta:

$$\theta_{t+1} = \theta_t - \eta\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

RMSProp

RMSProp is another method offered to solve the decreasing learning rate problem of adagrad. Geoffrey Hinton proposed it in his neural networks for machine learning class[1]. It is identical to the first update rule of Adadelta that is:

$$E\left[g^2\right]_t = \rho E\left[g^2\right]_{t-1} + (1-\rho)g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} \odot g_t$$

Similar to momentum constant, it is suggested to set ρ to 0.9, and ϵ is the smoothing constant similar to previous methods' update rules.

ADAM

Adam is another adaptive method that adjusts the learning rates for each parameter. It also stores an exponentially decaying average of past and squared gradients similar to momentum. It combines the best properties of adagrad and RMSProp algorithms.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2$$

Where m_t is the estimate of the first moment of the gradients and v_t is the estimate of the second moment. However, P.Kingma and Ba [21] noticed that these two terms are biased towards zero with zero initialization. Therefore they proposed bias-corrected forms of these terms to overcome this problem. It is suggested to set default values for β_1 and β_2 as 0.9 and 0.999.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Then the update rule is very similar to Adadelta and RMSProp that is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

[1] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

1.5 t-SNE

t-distributed stochastic neighbor embedding (t-SNE) is a method visualizing high-dimensional data which tries to keep the neighbor property in lower dimensions [22]. It is an unsupervised, non-linear dimensionality reduction technique commonly used for visualization purposes. It is beneficial to understand how data is structured in a high-dimensional space.

The idea of the algorithm is the following: first, it calculates the similarity between pairs in the high dimensional space and the low dimensional space. Then it tries to minimize these two similarity measures. The similarity measures are calculated by converting high-dimensional Euclidean distance to conditional probabilities. Each point centered a Gaussian distribution. This probability will be relatively high for closer points, whereas, for farther points, it will be smaller. For low-dimensional counterparts, a similar process can be followed. However, unlike stochastic neighbor embedding [23], student t-distribution with one degree of freedom (Cauchy distribution) is employed instead of Gaussian distribution. It is a heavy-tailed distribution that helps map the points far apart in low-dimensional space. After having two similarity measures, the objective is minimizing the mismatch among them. The objective function is expressed using Kullback-Leibler divergences and is optimized using a gradient descent method.

Highlights

With t-SNE, we completed the theoretical overview of the methods used for the experiments. We started with the simple definition of BP. We explained how the error is propagated in BP and how gradients are calculated mathematically. We also mentioned the drawbacks of BP. Then, we moved to DFA by slightly changing the error propagation mechanism of BP, and we adapted this change to our mathematical foundations. We specified which drawbacks of BP were addressed by DFA, and we also mentioned the limitations of DFA. After that, lazy methods are summarized since they are implemented and used in experiments of previous studies.

Then, optimizers are described in general. We mentioned why they are essential and how they might increase the convergence property, and we presented their update rules and drawbacks. Lastly, the idea of t-SNE is explained superficially.

2
Learning Problems

Contents

The success of neural networks spawned a great interest in comparing the predictive power of the various models. This involves testing different models on the same learning problem, usually difficult to learn, and observing which model performs better. It is particularly beneficial to understand their learning dynamics, which helps discover their limitations. These studies achieved striking success in finding out the superiority of the neural networks over linear methods. Given that, we are curious about the place of the DFA on these learning problems. For this purpose, we consider two learning problems: **parity learning** and **synthetic data problem**.

2.1 Parity Learning Problem

The parities are notoriously hard to learn by ANNs [3]. By taking advantage of this situation, Daniely and Malach [4] questioned how far neural networks could go beyond the linear models. They did this by focusing on parities that have a complex family of target functions. They demonstrated that this family could

be approximated by a two-layer network trained with Adadelta but not by lazy methods. This study brings an explanation of why neural networks' performance is better than linear methods, and it proves neural networks' learning capacities are beyond them.

Experiments are performed on the MNIST dataset by imitating the parity problem. The task is: given a parameter k (defines the number of digits to be stacked together that is chosen uniformly from the dataset), determine if the sum of the digits is odd or even. When $k = 1$, it is a simplified version of the standard MNIST task to find if a digit is even or odd. Experiment results showed that all models, including the lazy ones, reached a similar performance in the $k = 1$ case where the neural network slightly outperformed others. On the other hand, the problem becomes more difficult for the case $k = 3$ because models need to compute the parity of the digits' sum. In this case, there is a drastic gap between the neural network and lazy methods. Because the predictions of lazy methods did not go beyond the random guess.

Since our goal is comparing DFA and BP on this particular problem, reproducing the results from [4] is unavoidable. The same configurations are used with minor differences and, they are implemented in Pytorch [20]. The network has only a hidden layer with 512 neurons. For the last layer, sigmoid is used as a non-linear activation function. For the hidden layer, reLU is used. BCE is preferred as a loss function, and we have not used weight decay. In addition to previous settings, we have also used SGD to observe how much Adadelta improves. For the case $k = 3$, we performed a simple hyper-parameter tuning process to get a decent learning rate for each method. Same learning rate values are used for the case $k = 1$. The hyperparameter tuning process is the following. First, we define the parameter space, later we run with all different learning rates, and we compare these runs by the average of test accuracy of the last ten epochs, and we choose the highest one. It is also crucial to mention that, at each epoch, train data is recreated to boost the available data for the models. The same is also performed for test data to have an unbiased estimation of test accuracy. It implicitly prevents overfitting and

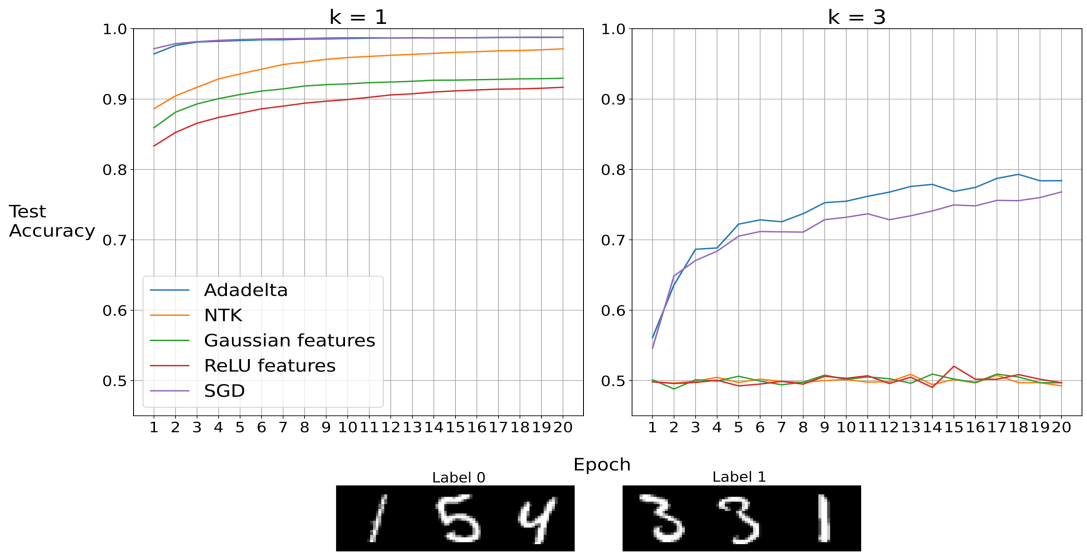

Figure 2.1: Reproduced MNIST-Parity Experiment
Reproduced MNIST-Parity experiment from [4]. The top left is the test accuracy for the parity of a single image. The top right is the test accuracy for the parity of three images. The bottom is the examples from the dataset. The model needs to predict the parity of the sum of the digits.

increases the available data because the data creation is stochastic. The process is the following: random images are sampled uniformly from the available dataset, then according to given parameter k, these random images are horizontally stacked. Hence we have different dataset at each epoch that helps networks to learn and perform better. Finally, these images are normalized before training the networks, which is necessary for deep learning tasks.

The reproduced result can be observed in figure 2.1. Similar to results from [4], all the methods succeed learning for $k = 1$ case. However, adadelta and SGD slightly outperformed lazy methods in this setting. In the case of $k = 3$, adadelta and SGD almost reach 80%, but the performance of lazy methods does not go beyond a random guess. After having the concrete picture from the previous study, it is intriguing to see how DFA would perform with SGD and adaptive methods on this particular problem. We will investigate it in chapter 3 with other experiments.

2.2 Synthetic Data Problem

Chizat and Bach [5] presented implicit bias in two-layer neural networks with cross-entropy loss trained with SGD. Implicit bias is a phenomenon that indicates that SGD is not only successful in finding the global minimum but also is biased towards solutions that generalize well [24]. The study is beneficial to observe this phenomenon of gradient methods and training dynamics of wide neural networks. After demonstrating theoretical results, they performed numerical experiments to validate these results. The numerical experiments have a binary classification problem, and the data are synthetically generated. The problem is an example of a task where the inputs have a lower-dimensional structure. The number of samples and dimensions can be adjusted in the dataset. Similar to parity experiments, the results demonstrated the superiority of the neural network to the lazy method. Considering that all the nice properties of the dataset and the similarity of our previous comparison, it is an excellent problem to compare BP and DFA. Besides, it is nicer to compare BP and DFA in another challenging learning problem which the difficulty of it can be adjusted to avoid limiting our experiments with only the MNIST-Parity task.

Given the parameter c that denotes the number of clusters (in [5], this parameter is denoted as k since we used the same notation in parity problem, to avoid any confusion it is changed). The data is generated as the following: in dimension $d = 2$, the distribution of the input values is a mixture of c^2 uniform distributions on the disk of radius $1/(3c-1)$ on a uniform two-dimensional grid with step $3/(3c-1)$. Larger dimensions follow the uniform distribution on $[-1/2, 1/2]$. Each cluster is assigned randomly to a label [5]. In other words, after having the cluster centers, each input is sampled by following the uniform distribution with the shift angle and magnitude for the first two dimensions. Each input value is sampled from a uniform distribution on $[-1/2, 1/2]$ for other spurious dimensions. Unlike the paper, labels are $0, 1$, not $-1, 1$ because it fits the structure of our previous architecture and training mechanism used for the parity experiment.

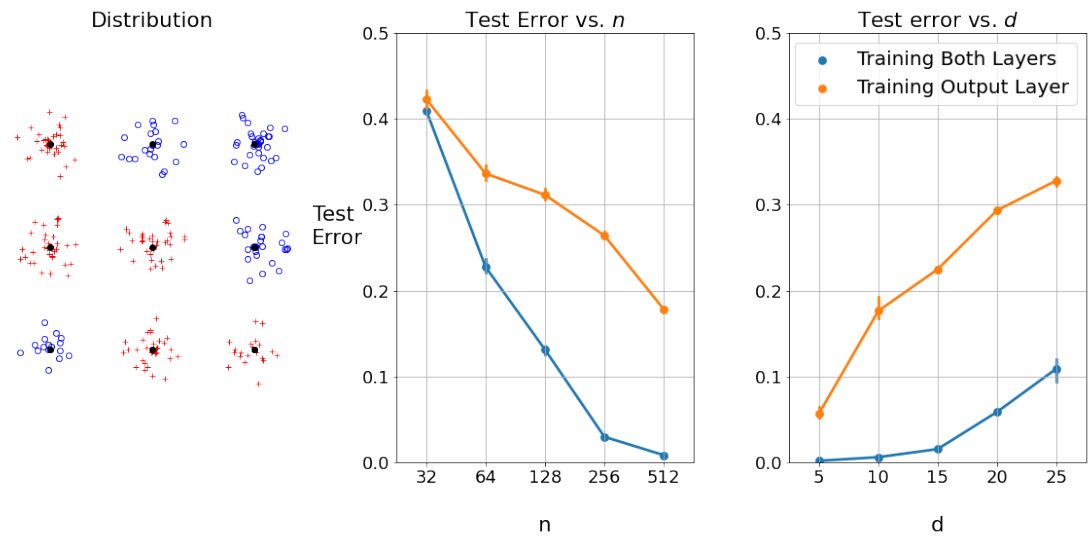

Figure 2.2: Reproduced Synthetic Data Experiment
Reproduced synthetic data experiment from [5]. The networks are trained with backpropagation and SGD. On the left, distribution of the input on two first dimensions. In the middle, test error as a function of the number of inputs (n) with $d = 15$. On the right, test error as a function of the number of dimensions (d) with $n = 256$.

Like the parity problem, before testing DFA on the problem, it is beneficial to reproduce results from the previous study. For this purpose, the scratch implementation is used with the same architecture (only a hidden layer, 512 layer size, with reLU and sigmoid non-linearities). For each n (number of training samples) and d (number of dimensions, ≥ 2), experiments were performed three times, and they are plotted with a 95% confidence interval. Similar to the paper, the learning rate is set to 0.5 (no hyperparameter tuning process is performed for this problem), weight decay is not used, and the epoch number is 500. We give enough time for all methods to converge with this high epoch number.

We can observe the reproduced experiment results in figure 2.2 from [5]. We have the data distribution in two-dimension on the left with $c = 3$ (so we have 9 clusters) used for the experiment. This distribution is chosen because it is difficult to learn (it is not linearly separable). We have the test error and an increasing number of training samples in the middle plot where the dimension is 15. We expect to observe decreasing test error with the increasing number of training samples for each model. On the right, we have the test error and the number of dimensions where the number

of samples is 256. The problem becomes challenging with the increasing number of dimensions because extracting useful information becomes more difficult for models with higher dimensional input since they are non-informative. Therefore we see the increasing test error. Similar to the paper results, we observe that training both layers gives better results than training only the output layer. In other words, increasing the number of training samples helps more to network than the lazy method, and the neural network is more successful in distinguishing useful inputs in high dimensions for this problem. It is important to mention that we have used only one distribution with 3 repetition. The result may vary on distribution but on average, we should observe the superiority of BP as it is presented in [5]. It is interesting to put DFA in this frame to observe if it is closer to the lazy method or BP.

Highlights

After having the reproduced results for the synthetic data problem, we completed the chapter on learning problems. First, we explained the problems and described how their data were generated and the task. Then we explained the details of the architectures of the networks, hyperparameter tuning process, and training phase used in the experiments. We presented the reproduced results from the papers and motivated the testing DFA on these problems.

3
Experiments

After having the previous study results, we can continue to test DFA on the learning problems. Train phase and the hyperparameter tuning process are explained in chapter 2. These processes are the same for the following experiments. For all experiments, scratch implementations are used with minimal Pytorch functionalities. They are performed three times and plotted with their confidence interval.

The same architectures are used for BP and DFA, meaning that we have only a hidden layer with 512 neurons for both problems, reLU is used as a non-linear function for the hidden layer, and sigmoid is preferred for the non-linearity of the last layer. BCE is chosen as a loss function. Weights of the networks are initialized uniformly with $\frac{1}{\sqrt{inputdim}}$ as in default Pytorch weight initialization. Moreover, the random matrix B is initialized with the same way to have similar behaviors as the weight matrix unless other specified. For the parity problem, networks are trained for 20 epochs unless others are specified, and at each epoch, train and test datasets are recreated as it is explained in chapter 2 whereas, for the synthetic data problem, networks are trained for 500 epochs.

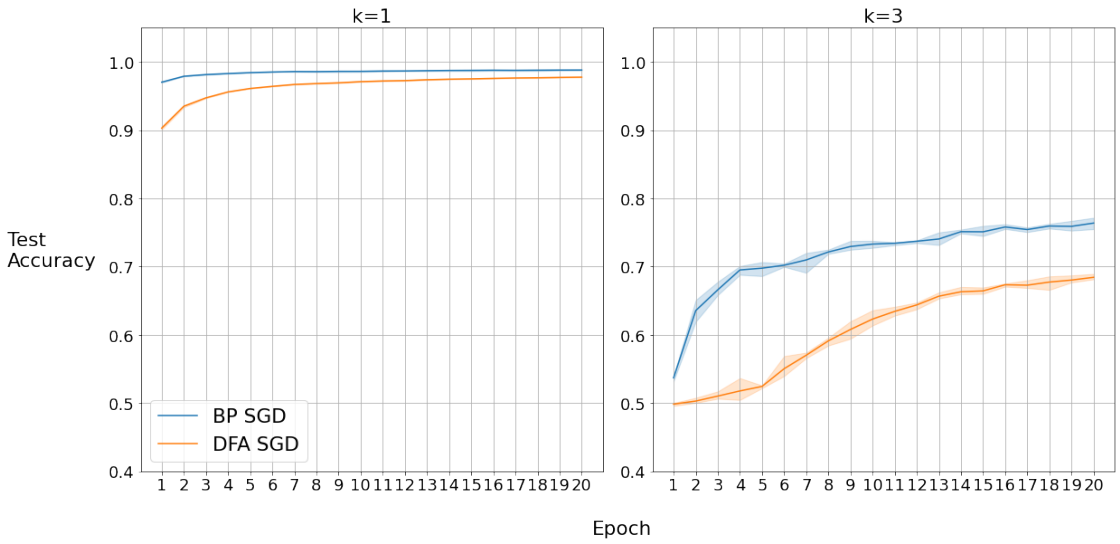

Figure 3.1: BP and DFA on MNIST-Parity Problem with SGD
The predictive power of DFA is presented on the MNIST-Parity experiment among BP. The left is the test accuracy for the parity of a single image. The right is the test accuracy for the parity of three images.

3.1 Parity Learning Experiments

For each experiment, when there is a change in the training method (DFA or BP) or optimization method, hyperparameter tuning is performed to obtain a decent learning rate. Weight decay is not used for the experiments. Since the experiment details are precise, we can test DFA on the parity learning problem with BP using SGD.

We can observe the results in figure 3.1 with 95% confidence interval. In the case $k = 1$ (parity of a single image), DFA outperforms the lazy methods, but it is behind the BP. In the case $k = 3$ (parity of the sum of three digits), it is obvious that DFA performs much better than lazy methods. However, the gap between BP and DFA is a bit higher than the $k = 1$ case. It seems like there is a limit for DFA to reach with SGD that is around %70. The reason is that DFA has an additional task to accomplish, which is aligning with BP's teaching signals. In other words, the network loses time while making teaching signals useful. This delays the convergence and causes performance lag. We can see that during the first iterations, DFA does not converge fast enough to catch up with BP, and it always stays behind

it.

The thrilling question is, is there a performance limit for DFA to reach, and can we get a similar performance as BP by making some changes? For answering the first question, it is better to run DFA for more epochs to see if it can reach a similar performance as BP. Because with longer training, DFA will have time to align and converge. Also, it would be convenient to test DFA with different random matrices to observe any improvement for the second question. Because it is clear that learning in DFA is strongly dependent on random matrix. Besides, it is interesting to test if DFA can learn with different random matrices. While tuning the learning rate for DFA, we noticed that it is susceptible to the learning rate. Therefore we can use adaptive methods to have better convergence properties in BP and DFA. These methods are specifically good at adjusting the learning rate, which is more difficult to tune for DFA than BP.

Given that, we trained DFA for 50 epochs with a tuned learning rate to observe if it can reach a similar performance as BP. At the same time, alignment between the random matrix and the transpose of the weight matrix is plotted. This alignment is measured by using the cosine similarity.

From figure 3.2, we can see that DFA can reach a similar performance as BP trained with SGD. This result approves our comments about the additional task DFA has and why it takes longer to achieve the same performance. On the right side of the plot, we can examine the alignment between the random matrix and the transpose of the weight matrix. At the beginning of training, the similarity is low. However, we can see that alignment increases, similar to the performance with advancing steps. It shows that the network aligns with the BP's teaching signals. In other words, the network learns how to learn by using the random matrix. After having a similar performance from DFA with SGD, it is intriguing to test if we can achieve similar performance within the same epoch number. For this purpose, the first improvement attempt will be related to random matrices. Using different random matrices may influence the performance of DFA. Some of them might align better with BP's teaching signals. On the other hand, it is interesting to observe if we can

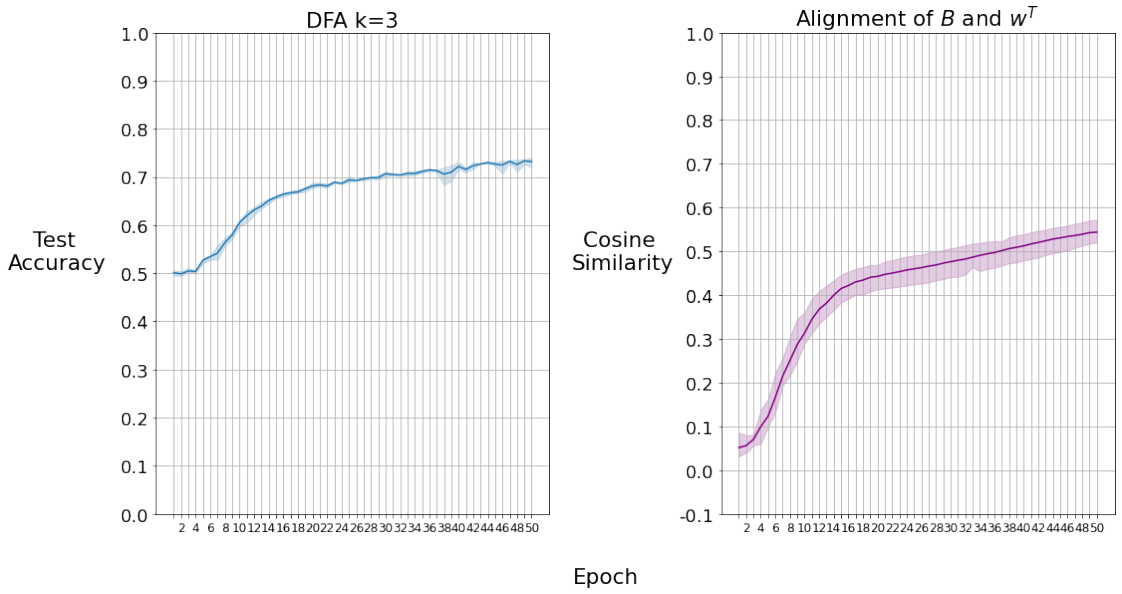

Figure 3.2: DFA on MNIST-Parity Problem with Alignment
DFA on MNIST-Parity task for the case $k = 3$ trained 50 epochs. The left is the test accuracy of DFA, and the right is the cosine similarity of the random matrix and the transpose of the weight matrix.

learn with any random matrix.

Apart from uniform random matrix, three different random matrices are tested. They are initialized as the following: **standard uniform** is default Pytorch initialization that is uniformly distributed from 0 to 1. **Gaussian** is initialized normally with μ and σ are equal to each other that is $\frac{1}{\sqrt{inputdim}}$. Lastly, **standard gaussian** is initialized with $\mu = 0$ and $\sigma = 1$.

From figure 3.3, we can observe that DFA can learn with any random matrices. However, it is essential to specify that learning rates for each random matrix are tuned and drastically different. Apart from standard uniform, the rest of the random matrices achieved similar performances, but they are still behind the BP. However, thanks to these results, we can see that DFA is highly sensitive to the learning rate. Because during the tuning phase, small learning rates did not converge within the specified epoch number. On the other hand, high learning rates demonstrated overfitting for each random matrices. Interested readers may refer to appendix B for more details on this argument. Therefore, since adaptive methods have better convergence properties, they may increase the performance of DFA as they did in

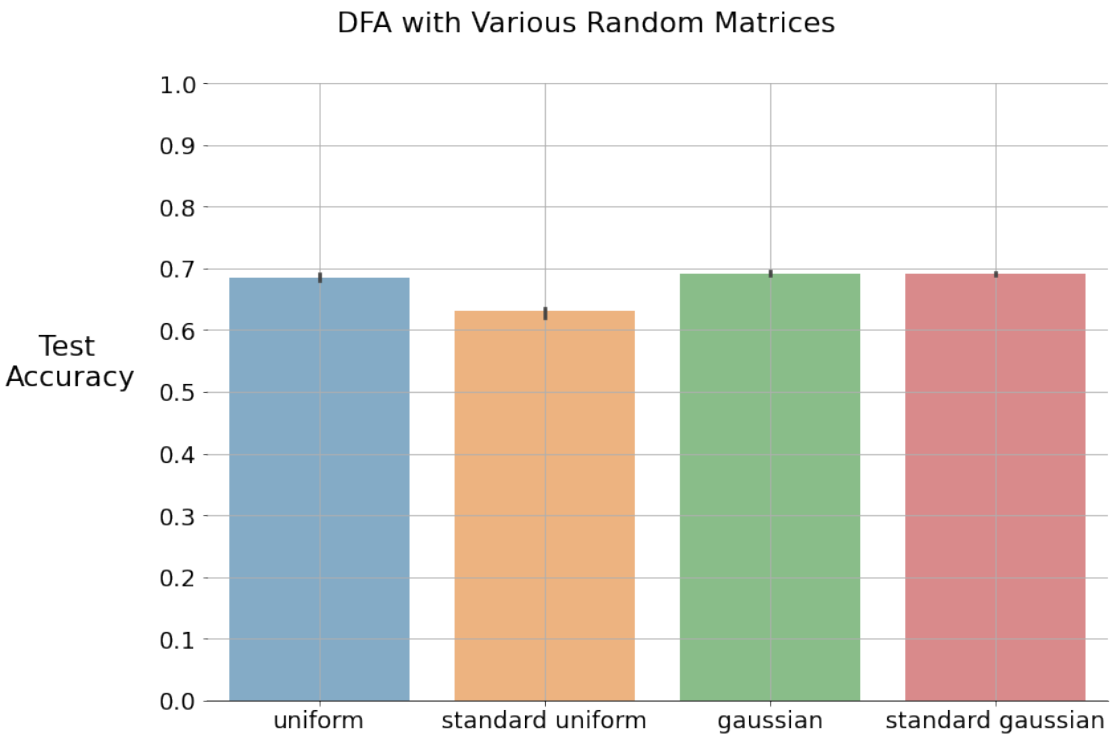

Figure 3.3: DFA on Parity Problem with Various Random Matrices
The final test accuracy of DFA on the MNIST-Parity for the case $k = 3$. The experiments performed three times and plotted with 95% confidence interval. The random matrices are initialized as the following: standard uniform $\sim U(0, 1)$, uniform $\sim U(-a, +a)$, standard gaussian $\sim \mathcal{N}(0, 1)$ and gaussian $\sim \mathcal{N}(a, a)$ where $a = \frac{1}{\sqrt{inputdim}}$.

BP. For the rest of the DFA experiments, the random matrix is uniformly initialized since there is no significant improvement with other initializations.

Following the previous deduction, various adaptive methods are tested on the parity learning problem for BP and DFA. Their learning rates are tuned, as explained in the previous chapter. For the experiments, they are run three times, and their final test accuracies are plotted. The results are presented in figure 3.4.

As expected, adaptive methods improve the final test accuracy significantly for both BP and DFA. On average, DFA is still behind the BP, but with RMSProp and Adadelta, the gap is much smaller than with plain SGD. Sometimes DFA's final test accuracy even exceeds BP. In other words, we can say that some adaptive methods help DFA more than BP on this task. However, we should not ignore that DFA has larger fluctuations for the final test accuracy than BP. Thanks to adaptive methods, the last experiment could close the gap between BP and DFA for the

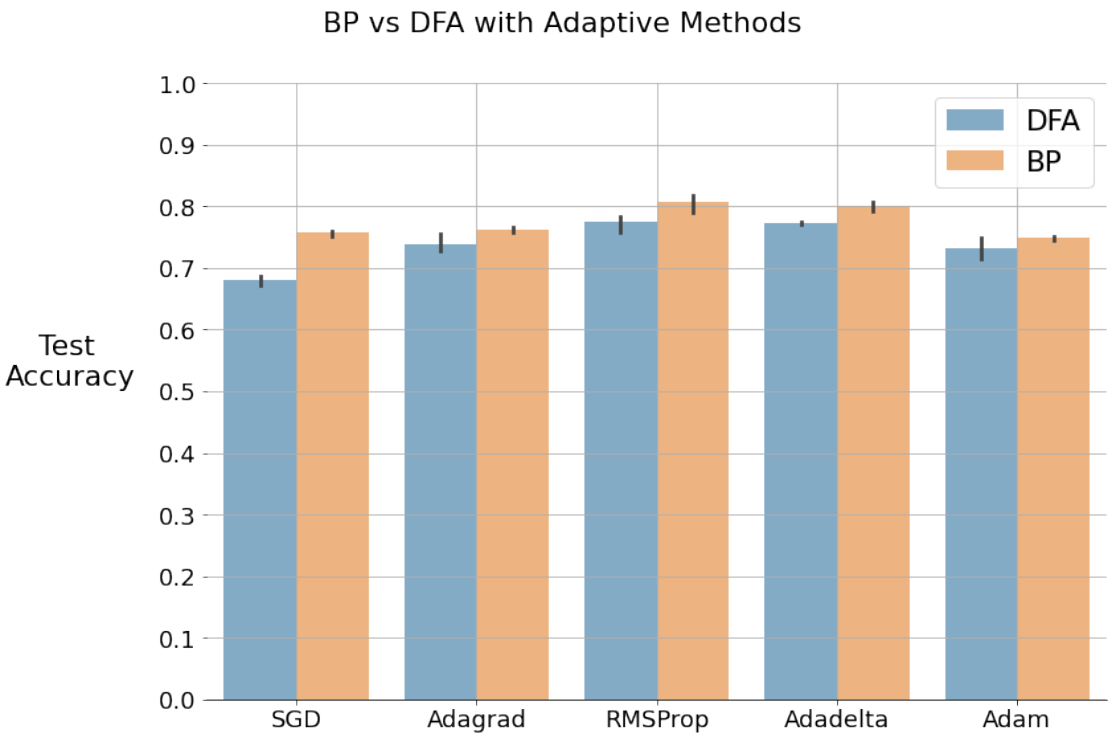

Figure 3.4: DFA and BP on MNIST-Parity Problem with Adaptive Methods The final test accuracies are presented for BP and DFA with adaptive optimization algorithms. The experiments were performed three times and plotted with a 95% confidence interval.

parity learning problem. The reason for the improvement of adaptive methods in DFA is an excellent question to investigate the algorithm's behavior. One possible idea is, adaptive methods may spawn better alignment than SGD. Testing this theory is relatively easy. We can train DFA with SGD and one of the adaptive methods, and then we can observe the alignment of the random matrix and the transpose of the weight matrix with gradient alignments.

In figure 3.5, we can see the alignment measures of SGD and RMSProp. It is interesting to point out that SGD has better alignment during the later training steps than RMSProp. From the results of the previous experiments, we know that RMSProp performed better than SGD on this task. Given that, for having better performance, perfect alignment is not always required. On the other hand, at the beginning of the training, RMSProp aligns faster than SGD. Therefore, we can say that there is a faster alignment with adaptive methods at the beginning that prevents DFA from losing time at first epoch numbers, but later DFA finds

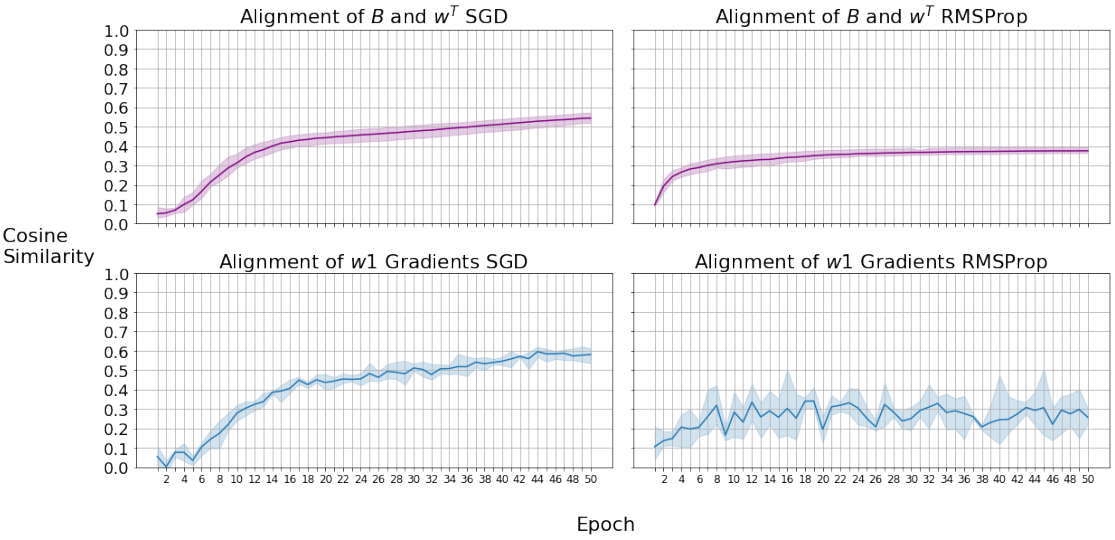

Figure 3.5: Alignment Comparison of SGD and RMSProp
The top is the alignment of the random matrix and the transpose of the weight matrix for SGD and RMSProp. The bottom is the alignment of the gradients of w_1 concerning loss for SGD and RMSProp. Both plots are trained on the MNIST-Parity task for the case $k = 3$ for 50 epochs.

other paths to find the minimum that is different from BP. These outcomes are parallel to results of [12], it was stated that DFA first aligns with BP, and later it sacrifices from this alignment to find better paths to the minimum. Apparently, this phenomenon happens quicker with adaptive methods than plain SGD. This part can be investigated in more detail to understand better the effect of adaptive methods on DFA. It is left for further studies.

One of the exciting questions that are not directly related to the predictive power of the algorithms: do the networks learn the digits individually when we train them for the MNIST-Parity task and make the necessary processes (summation, division as humans), if not what type of information does it capture? We can answer this question by observing the hidden representation of the digits. If the network learns the digits well, we need to observe a good separation like the standard MNIST task (classifying hand digits). For this purpose, hidden representations of the digits from the networks trained with BP and DFA are plotted in two-dimensional space by using t-SNE using the implementation of sklearn [25].

The hidden representation of a single image in the $k = 3$ case is obtained in

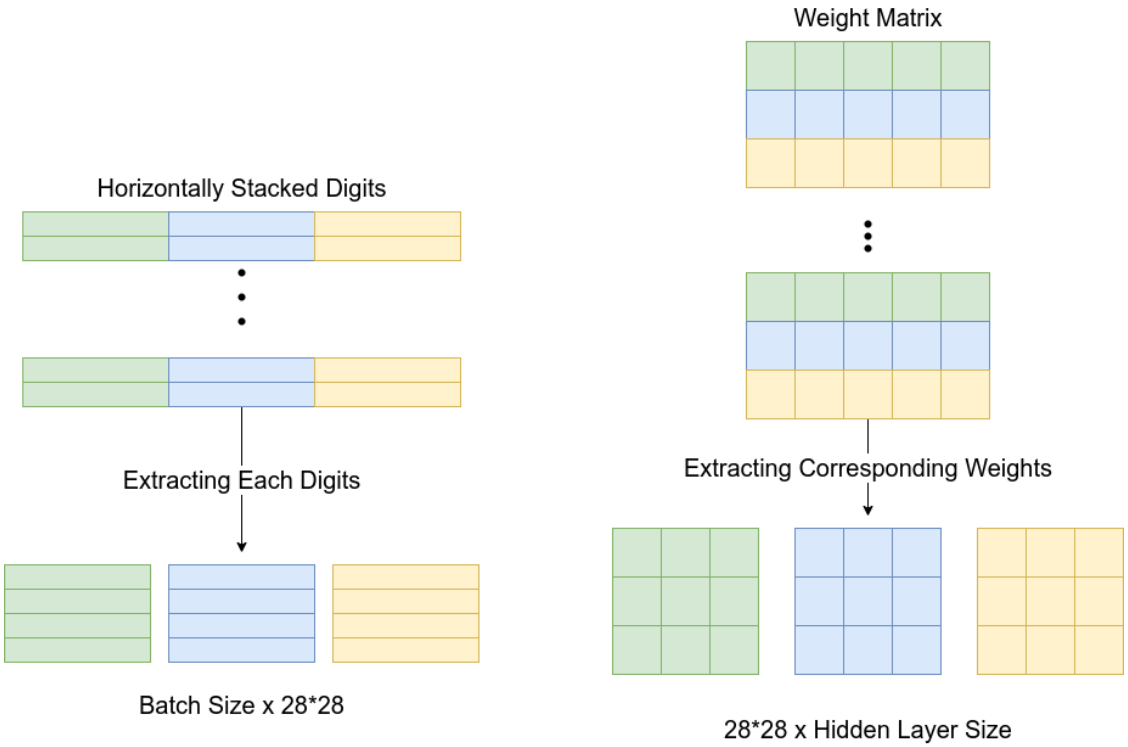

Figure 3.6: Process of Extracting the Hidden Representation of Single Digit After extracting the digits and the corresponding weights, the matrix multiplication (same color) for each digit is performed separately (28 is the pixel size of a single digit); by doing so, we have the hidden representation of a single image from the network trained for parities of the sum of three digits.

the following way; we know that after we flatten the images, particular parts of each image are multiplied by corresponding parts of the weight matrix. Getting these parts and performing multiplication for each digit will reveal the hidden representation of the individual digit trained in the $k = 3$ case. Process is visualized in figure 3.6. After extracting the pixels of each image and the corresponding weights, we end up with the following matrix multiplication: [Batch Size$\times$784][784$\times$ Hidden Layer] where 784 is 28×28 and 28 is the pixel size of a single digit. With this matrix multiplication, we have the hidden representation of each image.

After performing the matrix multiplication, the rest is visualizing these hidden representations by using the t-SNE. Since the t-SNE is computationally expensive, only 7500 random samples are plotted from the dataset. In figure 3.7 we can observe the results in two dimensions and say that the networks do not capture the information about the digits individually when they are trained for parities of

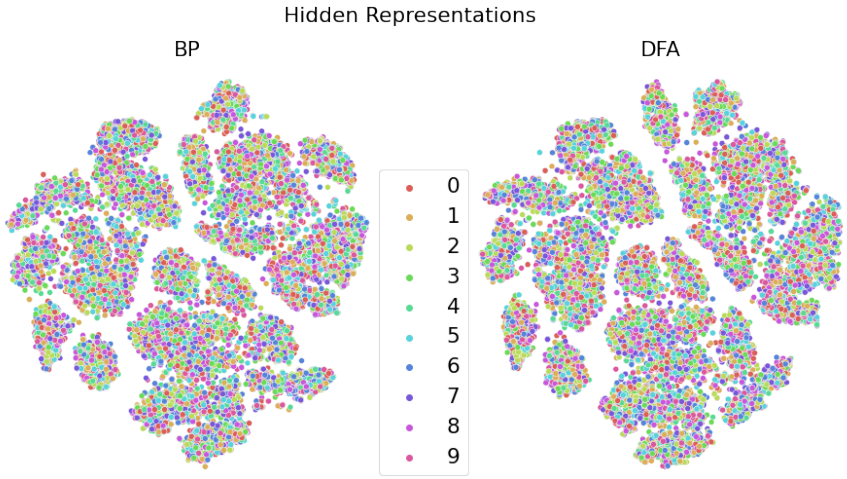

Figure 3.7: Hidden Representation of Digits in BP and DFA
Hidden representations of images are visualized with their labels for BP and DFA. The process of acquiring these representations is explained in figure 3.6.

the sum of three digits. They learn different properties of the data, not the digits individually. However, for the same purpose, transfer learning can be implemented to test if these networks trained for the parity problem can classify the digits accurately. Transfer learning is a method that involves storing the information gained while solving a problem and using this information on a different, but related problem [26]. Also, the same experiment can be performed for deeper networks. Maybe deeper representations of the parities might contain information about the digits individually. These experiments are left for future studies. On the other hand, it is not surprising that BP and DFA capture very similar information from the MNIST-Parity dataset.

3.2 Synthetic Data Experiments

After having the previous experiment result and current experiments details for synthetic data, we can test the DFA on the number of dimensions plots with the train and test error to compare with BP. Most of the parameters are the same as MNIST-Parity experiments. The only difference is the learning rate. The

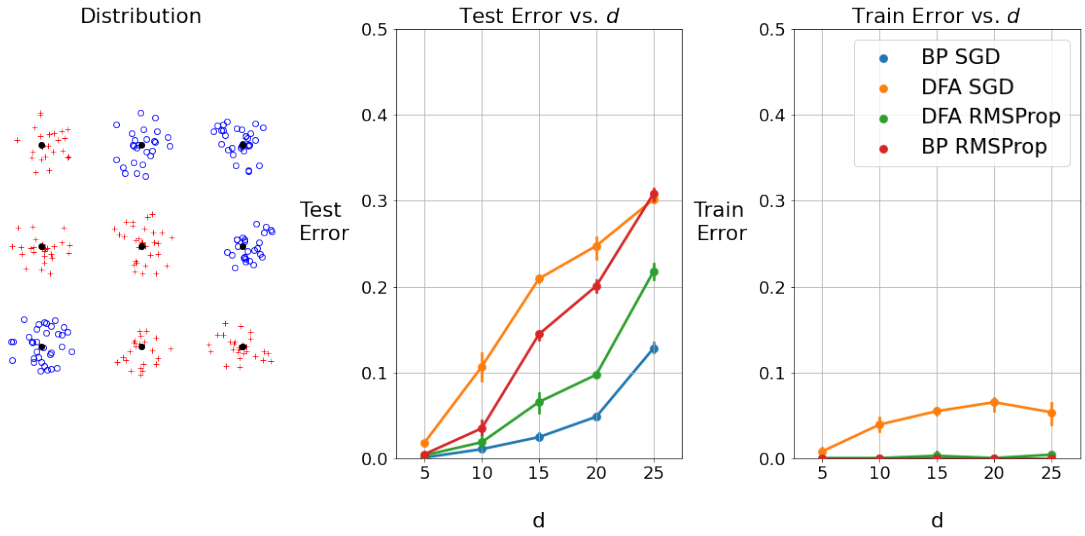

Figure 3.8: BP and DFA on Synthetic Data Problem
On the left, the input distribution of the two first dimensions is used for the experiment. In the middle and on the right, test and train error of BP and DFA with SGD and RMSProp as a function of the number of dimensions (d) where the number of inputs is 256 ($n = 256$).

learning rate is set to 0.5 without hyperparameter tuning for all methods. They are performed with three repetitions and plotted with their 95% confidence interval. Before interpreting the results, it is essential to mention that experiment results will be drastically different if the input distribution has changed. We have chosen a distribution that shows the difference among the algorithms well. It would be more convenient to run for different datasets with more repetition and plot the average. We left this for future considerations. We can observe the results in figure 3.8. Although DFA with SGD cannot fit data well (train error is not 0 like others), it has a better test error than the lazy method. However, BP with SGD outperforms the other methods. Since it is natural to try adaptive methods to hope for an improvement, as we experienced for the parity experiments, we used RMSProp on the same problem both for BP and DFA. It again increases the convergence (train error is 0) and improves the test error of the DFA, but it is still behind the BP with SGD.

It is also interesting that BP with RMSprop did not perform better than plain BP with SGD. It might be due to the implicit bias of adaptive methods (while fitting

the data, the adaptive methods might choose global minima of the training loss with suboptimal generalization error), or sometimes adaptive methods can have unexpected problems in simple settings, such as convex problems (this distribution might be another example of this) [27]. Furthermore, there are some interesting results similar to our findings that show; although adaptive methods fit the train data well, they may generalize poorly, so these results are thus less surprising for BP [28]. Because both our setup and the setup of [28] have in common that the label of the data point depends only on a very low-dimensional subspace of input space, which could explain the similar trends observed for adaptive BP methods. The interesting observation is the opposite effect for DFA. Understanding why adaptive methods affect BP and DFA differently is an intriguing avenue for further works. Moreover, it is promising that DFA improves with one of the adaptive methods for another challenging learning problem. We have to specify that, It would be intriguing to observe well-tuned DFA with different adaptive methods on the same problem; this is again left for future investigations.

Highlights

With the synthetic data experiment, we conclude the experiments. We started with testing the predictive power of DFA on the MNIST-Parity task. Then, we visualized the alignment of the random matrix and the transpose of the weights matrix. After that, we tried to improve the performance of DFA by trying different types of random matrices and adaptive methods. We discovered that with the help of adaptive methods, DFA could perform as well as BP within the same epoch numbers. For investigating why adaptive methods help DFA more than BP, we compared the alignments of SGD and RMSProp for DFA. We concluded that better alignment does not always mean better performance. Detailed investigation of this observation is left for future studies. Then, we visualized the hidden representations of the digits from the network trained for the MNIST-Parity task. We observed that networks do not capture information about the digits individually. Lastly, we performed the synthetic data experiment using a fixed distribution. We plotted

the final train and test error of BP and DFA, both with SGD and RMSProp as the function of the number of dimensions. Once again, we noticed that adaptive methods boost the performance of DFA. We left some further experiments of synthetic data for future studies.

Conclusion

This research aimed to compare DFA with BP on the particular learning problems that proved the superiority of the BP over the linear methods. Moreover, we tried to improve their predictive power on these learning problems. In doing so, we hoped to understand their learning dynamics better.

Thanks to the error propagation mechanism of DFA, it proposes a more biologically plausible algorithm than BP. However, the performance is still an enormous part of the success of the algorithms. Furthermore, based on the theoretical foundations of the algorithms that we presented in chapter 1, it was expected to experience performance lag for DFA behind BP. For validating this empirically, we used a set of learning problems that are explained in chapter 2. In previous studies, these problems proved the dominance of the BP over the lazy methods. After reproducing the previous experiments from these studies, we tested DFA on the same problems. Our expectations were validated with the initial results. These results are demonstrated in chapter 3 among other experiments. Although DFA performed better than the linear methods, there was a performance gap between plain DFA and BP without any intervention. Then, we tried to close this gap by performing a few modifications either on the training algorithm or the optimization process. Even though these modifications did not constantly improve the performance, we acquired a deeper understanding of algorithms' learning behaviors. They opened the gate of correct interventions to close the gaps. Finally, these interventions yield that using adaptive optimizers, DFA could perform equal or close to BP's predictive performance on the particular learning problems that the lazy methods notoriously failed or did not perform well.

Although we performed various experiments that investigate the performance and

learning behaviors of the algorithms, there are still too many aspects that need to be investigated further. For instance, we used architectures with only a hidden layer, and layer size was 512; so the network's depth and the layer's width are not investigated further. Different activation functions are not tested apart from reLU. It would be intriguing to observe the behavior of DFA with these aspects. Also, one of the experiments that analyze the hidden representation of the digits in the parity learning problem can be extended. For instance, transfer learning can be implemented to test if networks trained for the parity problem can classify the digits accurately. In other words, the network trained for the parity learning problem can be tested to classify the digits. Although these hidden representations demonstrated the opposite, one might find a trace of learning of the digits. In addition to these propositions, the learning problems can constantly be enriched, and new challenging problems can be considered with DFA. Also, different alternative algorithms can be tested on these challenging problems. Furthermore, all the theoretical and empirical aspects of the learning problems we use were not analyzed. For instance, we used a fixed distribution for the synthetic data problem. One might want to use more distributions and observe the average of the predictive powers of the algorithms. Moreover, for the synthetic data experiment, we observed that adaptive methods have a different effect on DFA and BP. Understanding the reason behind this is an attractive path to pursue. All these kinds of further experiments are left for future studies.

BP is the touchstone for training neural networks. Although the community demonstrated gigantic advancements for optimization methods or architecture types of the networks, BP is still around for decades, and the algorithm's success is indisputable. It is vital to point out that the advancement of BP did not happen in one day. DFA and other alternative algorithms bring a new perspective on the training phase of the networks. With this, we can remove some of the restrictions enforced by BP, such as biological implausibility, lack of parallelism, et cetera. We believe that testing the different aspects of the alternative algorithms and comparing them with BP would contribute a lot to the success of deep learning. These studies

might spawn more robust training algorithms which obey the biological plausibility, and they might even train faster with a performance close or equal to BP.

Appendices

A

Backpropagation with Binary Cross-Entropy

Following calculations are heavily inspired from these notes[1]. The notation and the general structure might differ, but the idea is the same.

Let us consider a simple binary classification task. It is common to use a network with a single logistic output with the binary cross-entropy loss function and for the sake of simplicity, let us assume that there is only one hidden layer.

$$BCE = -\sum_{i=1}^{nout} \left(y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right) \log\left(1 - \hat{y}_i\right)\right)$$

Where y is the ground truth and $\hat{y}$ is the output of the network. After having the loss function, let us continue with the forward pass.

$$a_k = h_{k-1} w_k + b_k$$

$$h_k = f(a_k)$$

Where, w_k is the weight, b_k is the bias term, h_k is the output of the layer (which means that $h_0 = X$ and $h_2 = \hat{y}$) and f is the non linear function. Please note that for last layer logistic function is used whereas for hidden layer reLU is used as non linear functions.

We can compute the derivative of the weights by using the chain rule.

[1] https://www.ics.uci.edu/~pjsadows/notes.pdf

A. Backpropagation with Binary Cross-Entropy

$$\frac{\partial BCE}{\partial w_2} = \frac{\partial BCE}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

Computing each factor in the term, we have:

$$\begin{aligned} \frac{\partial BCE}{\partial \hat{y}} &= \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \\ &= \frac{\hat{y}-y}{\hat{y}\,(1-\hat{y})} \\ \frac{\partial \hat{y}}{\partial a_2} &= \hat{y}\,(1-\hat{y}) \\ \frac{\partial a_2}{\partial w_2} &= h_1 \end{aligned}$$

This expression gives us:

$$\frac{\partial BCE}{\partial w_2} = (\hat{y}-y)\,h_1$$

We can calculate the derivative of the w_1 concerning loss function as the following:

$$\frac{\partial BCE}{\partial w_1} = \frac{\partial BCE}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

Compute each factor in the term again, we have:

$$\begin{aligned} \frac{\partial BCE}{\partial h_1} &= \frac{\partial BCE}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_2} \frac{\partial a_2}{\partial h_1} \\ &= (\hat{y}-y)\,w_2 \\ \frac{\partial h_1}{\partial a_1} &= f'(a_1) \\ \frac{\partial a_1}{\partial h_1} &= X \end{aligned}$$

This expression gives us:

$$\frac{\partial BCE}{\partial w_1} = (X)\,(\hat{y}-y)\,(w_2) \odot f'(a_1)$$

Where $\odot$ is element-wise multiplication, similarly, bias terms can be calculated by following:

$$\begin{aligned} \frac{\partial BCE}{\partial b_2} &= \frac{\partial BCE}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_2} \frac{\partial a_2}{\partial b_2} \\ &= (\hat{y}-y) \end{aligned}$$

$$\begin{aligned}\frac{\partial BCE}{\partial b_1} &= \frac{\partial BCE}{\partial h_1}\frac{\partial h_1}{\partial a_1}\frac{\partial a_1}{\partial b_1} \\ &= (\hat{y} - y)\,(w_2) \odot f'(a_1)\end{aligned}$$

After having all these results, we can update the parameters (weights and biases) using gradient descent and its variants.

B

Direct Feedback Alignment with Various Learning Rates

We have mentioned that DFA is highly sensitive to the learning rate throughout the experiments. We want to support this argument with some plots. For this purpose, we plotted test accuracy for each learning rate value during the hyperparameter tuning of different random matrices. The networks are trained for the MNIST-Parity task for the case $k = 3$ with SGD up to 20 epochs.

The result can be observed in figure B.1. We can see that some high learning rate values do not even provide performance beyond a random guess as in the standard uniform random matrix. For the gaussian random matrix, if we have a high learning rate, we can observe increasing accuracy and a rapid decrease later. These patterns may also happen with other initializations, even with adaptive methods. We suggest decreasing the learning rate to have more stable patterns in these cases.

We performed the same process for adaptive methods. Again, the networks are trained for the MNIST-Parity task for the case $k = 3$ up to 20 epochs. The results for the adaptive methods can be observed in figure B.2.

We can see that adaptive methods are more robust than plain SGD. However, similar to the previous plot, it is possible to observe comparable patterns. Indeed, Adadelta draws a zig-zag pattern with a high learning rate.

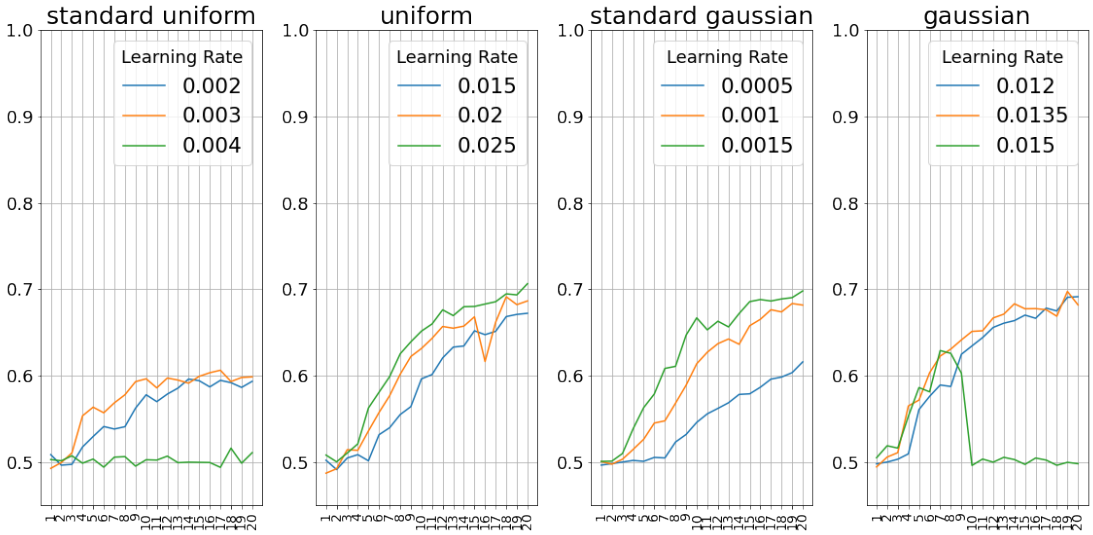

Figure B.1: Various Random Matrix Initialization with Different Learning Rates
The predictive power of DFA with various random matrix initializations on MNIST-Parity task. The model tries to predict the parities of the sum of three digits and it is trained with SGD. The random matrices are initialized as the following: standard uniform $\sim U(0,1)$, uniform $\sim U(-a,+a)$, standard gaussian $\sim \mathcal{N}(0,1)$ and gaussian $\sim \mathcal{N}(a,a)$ where $a = \frac{1}{\sqrt{inputdim}}$.

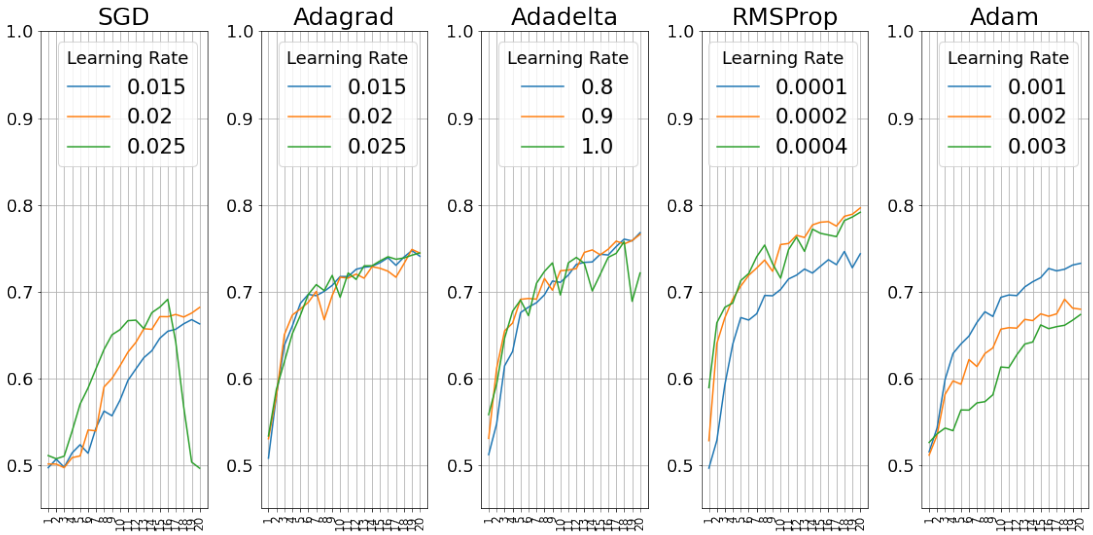

Figure B.2: Various Adaptive Methods with Different Learning Rates
The predictive power of DFA is presented with adaptive methods on the MNIST-Parity task. Random matrices are initialized uniformly. The model tries to predict the parities of the sum of three digits with different learning rates.

C
Reproducibility

For reproducing the experiment results, please refer to this Github repository[1]. It contains all the codes that are used to have the experiment results. It also includes the requirements to run the code. However, due to the stochastic behavior of the neural networks and the data, results will not be precisely the same. Nevertheless, it must be very close to the presented ones on average. Stochastic behaviors can be explained as the following: The neural network weights and the random matrix (for DFA) are sampled from a uniform distribution so that each instance will be slightly different. Given the parameter k, the parity data is sampled uniformly from the MNIST dataset. In addition to this, it is recreated for each epoch, so we had a different dataset at every iteration. DFA is extremely sensitive to the learning rate, so occasionally, overfitting might be observed, meaning that test accuracy might decrease instantly in the middle of training. For overcoming this issue, we suggest decreasing the learning rate. Although we used a fixed distribution, the synthetic data also follows stochastic behavior. The labels of the clusters are assigned randomly, and the cluster samples are distributed from a uniform distribution (it is explained detailly in 2). The distribution of the first two dimensions is plotted with experiments. Other distributions would give different results. Hyperparameters

[1] https://github.com/demirbilek95/Dynamics-of-Learning

used in the experiments can be found in the Github repository, and the process of how they are tuned or which value they are set is explained in the related chapters. Lastly, a local computer is used for all the experiments, and all the experiments are performed on GPU (Nvidia GTX 960M).

References

[1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[2] Arild Nøkland. *Direct Feedback Alignment Provides Learning in Deep Neural Networks*. 2016. arXiv: 1609.01596 [stat.ML].

[3] Maxwell Nye and Andrew Saxe. "Are Efficient Deep Representations Learnable?" In: *CoRR* abs/1807.06399 (2018). arXiv: 1807.06399. URL: http://arxiv.org/abs/1807.06399.

[4] Amit Daniely and Eran Malach. "Learning Parities with Neural Networks". In: *CoRR* abs/2002.07400 (2020). arXiv: 2002.07400. URL: https://arxiv.org/abs/2002.07400.

[5] Lenaic Chizat and Francis Bach. *Implicit Bias of Gradient Descent for Wide Two-layer Neural Networks Trained with the Logistic Loss*. 2020. arXiv: 2002.04486 [math.OC].

[6] Yoshua Bengio et al. *Towards Biologically Plausible Deep Learning*. 2016. arXiv: 1502.04156 [cs.LG].

[7] Dong-Hyun Lee et al. *Difference Target Propagation*. 2015. arXiv: 1412.7525 [cs.LG].

[8] Wan-Duo Kurt Ma, J. P. Lewis, and W. Bastiaan Kleijn. *The HSIC Bottleneck: Deep Learning without Back-Propagation*. 2019. arXiv: 1908.01580 [cs.LG].

[9] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale". In: *CoRR* abs/1611.01236 (2016). arXiv: 1611.01236. URL: http://arxiv.org/abs/1611.01236.

[10] Timothy P. Lillicrap et al. *Random feedback weights support learning in deep neural networks*. 2014. arXiv: 1411.0247 [q-bio.NC].

[11] Julien Launay et al. *Direct Feedback Alignment Scales to Modern Deep Learning Tasks and Architectures*. 2020. arXiv: 2006.12878 [stat.ML].

[12] Maria Refinetti et al. *Align, then memorise: the dynamics of learning with feedback alignment*. 2021. arXiv: 2011.12428 [stat.ML].

[13] Julien Launay, Iacopo Poli, and Florent Krzakala. *Principled Training of Neural Networks with Direct Feedback Alignment*. 2019. arXiv: 1906.04554 [stat.ML].

[14] Lenaic Chizat, Edouard Oyallon, and Francis Bach. *On Lazy Training in Differentiable Programming*. 2020. arXiv: 1812.07956 [math.OC].

[15] Jaehoon Lee et al. *Deep Neural Networks as Gaussian Processes*. 2018. arXiv: 1711.00165 [stat.ML].

[16] Pedro Domingos. *Every Model Learned by Gradient Descent Is Approximately a Kernel Machine*. 2020. arXiv: 2012.00152 [cs.LG].

[17] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks". In: *CoRR* abs/1806.07572 (2018). arXiv: 1806.07572. URL: http://arxiv.org/abs/1806.07572.

[18] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[19] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: 1212.5701 [cs.LG].

[20] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[21] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[22] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[23] Geoffrey Hinton and Sam Roweis. "Stochastic Neighbor Embedding". In: *Advances in neural information processing systems* 15 (2003). Ed. by S Thrun S Becker and KEditors Obermayer, pp. 833–840. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.7959&rep=rep1&type=pdf.

[24] Chulhee Yun, Shankar Krishnan, and Hossein Mobahi. *A Unifying View on Implicit Bias in Training Linear Neural Networks*. 2021. arXiv: 2010.02501 [cs.LG].

[25] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[26] Susmit Jha and Sanjit A. Seshia. "A Theory of Formal Synthesis via Inductive Learning". In: *CoRR* abs/1505.03953 (2015). arXiv: 1505.03953. URL: http://arxiv.org/abs/1505.03953.

[27] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. *On the Convergence of Adam and Beyond*. 2019. arXiv: 1904.09237 [cs.LG].

[28] Ashia C. Wilson et al. *The Marginal Value of Adaptive Gradient Methods in Machine Learning*. 2018. arXiv: 1705.08292 [stat.ML].