Parallelizable training in deep learning through local and distributed approaches

Louis Fournier

Sorbonne Université

Doctoral School **École Doctorale Informatique, Télécommunications et Electronique (ED130)**

University Department **Institut des Systèmes Intelligents et de Robotique**

Thesis defended by **Louis Fournier**

In order to become Doctor from Sorbonne Université

Academic Field **Information and Communication Sciences and Technologies**

Parallelizable Training in Deep Learning Through Local and Distributed Approaches

Thesis supervised by Edouard Oyallon

Colophon

Doctoral dissertation entitled “Parallelizable Training in Deep Learning Through Local and Distributed Approaches”, written by Louis Fournier, completed on August 13, 2024, typeset with the document preparation system LaTeX and the yathesis class dedicated to theses prepared in France.

Sorbonne Université

Doctoral School **École Doctorale Informatique, Télécommunications et Electronique (ED130)**

University Department **Institut des Systèmes Intelligents et de Robotique**

Thesis defended by **Louis Fournier**

In order to become Doctor from Sorbonne Université

Academic Field **Information and Communication Sciences and Technologies**

Parallelizable Training in Deep Learning Through Local and Distributed Approaches

Thesis supervised by Edouard Oyallon

Sorbonne Université

École doctorale **École Doctorale Informatique, Télécommunications et Electronique (ED130)**

Unité de recherche **Institut des Systèmes Intelligents et de Robotique**

Thèse présentée par **Louis Fournier**

En vue de l'obtention du grade de docteur de l'Sorbonne Université

Discipline **Sciences et technologies de l'information et de la communication**

Entraînement Parallélisable en Apprentissage Profond par le biais d'Approches Locales et Distribuées

Thèse dirigée par Edouard Oyallon

Keywords: deep learning, neural network, local learning, distributed learning, backpropagation, data parallelism, model parallelism, forward gradient, ensembling, delayed gradient

Mots clés : apprentissage profond, réseau de neurones, apprentissage local, apprentissage distribué, rétropropagation, parallélisme des données, parallélisme des modèles, gradient avant, ensemblisme, gradient retardé

This thesis has been prepared at

Institut des Systèmes Intelligents et de Robotique
Campus Pierre et Marie Curie
Pyramide, Tour 55
4, place Jussieu
75005 Paris
France

☎ +33(0)144275141
🖷 +33(0)144275145
✉ contact@isir.upmc.fr
Web Site https://www.isir.upmc.fr/

Parallelizable Training in Deep Learning Through Local and Distributed Approaches

Abstract

Recent breakthroughs in deep learning have been driven by the growth of deep neural networks, improving their ability to memorize and generalize. However, this growth requires ever-increasing computational resources to train these networks. In this thesis, we propose to improve the standard deep learning training framework, which consists of parallelized mini-batch SGD with backpropagation. By deviating from it, we can obtain more parallelizable and faster approaches. First, we study the capabilities of local learning approaches, a more parallelizable alternative to the backpropagation gradient estimation method. The model is split into sequential stages connected only by feedforward connections. We find that we can improve self-supervised local learning by removing certain data samples from the local losses computation, preventing information collapse. We also show that forward-mode automatic differentiation, which computes a directional derivative in a single forward pass, can be improved by using local gradients as tangent directions. Second, we study distributed approaches to training deep neural networks, and consider their communication costs in particular. We modify synchronous data parallelism to balance the overall memory and communication overhead by shifting worker execution from simultaneous to sequential. Finally, we propose a novel highly communication-efficient distributed training approach that allows an ensemble of models to be weight-averaged at the end of training, resulting in a single ensemble-level model.

Keywords: deep learning, neural network, local learning, distributed learning, backpropagation, data parallelism, model parallelism, forward gradient, ensembling, delayed gradient

---

Entraînement Parallélisable en Apprentissage Profond par le biais d'Approches Locales et Distribuées

Résumé

Les récentes avancées dans le domaine de l'apprentissage profond ont été poussées par la croissance des réseaux de neurones profonds, améliorant leur capacité de mémorisation et de généralisation. Cependant, cette croissance s'étend aussi aux ressources computationnelles nécessaires à leur entraînement. Dans cette thèse, nous proposons d'améliorer l'algorithme d'apprentissage standard qui consiste en de la rétropropagation parallélisée. En s'en écartant, il est possible d'obtenir des approches plus parallélisables et rapides. Tout d'abord, nous étudions les capacités des approches d'apprentissage local, une alternative plus parallélisable à la méthode standard d'estimation du gradient par rétropropagation. Le modèle est divisé en stages séquentiels reliées uniquement par des connexions de type 'feedforward'. Nous améliorons l'apprentissage local auto-supervisé en supprimant certains échantillons de données des calculs locaux, ce qui permet d'éviter un effondrement de l'information. Nous montrons également que la dérivation automatique en mode direct, qui calcule une dérivée directionnelle en 'feedforward', est améliorée en utilisant les gradients locaux comme directions tangentes. Deuxièmement, nous étudions les approches distribuées pour l'apprentissage profond, en particulier en tenant compte de leurs coûts de communication. Nous modifions le parallélisme de données synchrone pour équilibrer l'utilisation de la mémoire globale et des communications, en passant les calculs de simultanés à séquentiels. Enfin, nous proposons une nouvelle approche d'apprentissage distribué nécessitant peu de communications permettant à un ensemble de réseaux d'être moyenné après entraînement, donnant un modèle très performant au niveau de l'ensemble.

Mots clés : apprentissage profond, réseau de neurones, apprentissage local, apprentissage distribué, rétropropagation, parallélisme des données, parallélisme des modèles, gradient avant, ensemblisme, gradient retardé

Institut des Systèmes Intelligents et de Robotique
Campus Pierre et Marie Curie – Pyramide, Tour 55 – 4, place Jussieu – 75005 Paris – France

Acronyms

A | B | C | D | E | F | G | H | L | M | N | P | R | S | T | Z

A

AD Automatic Differentiation. 3
ADMM Alternating Direction Method of Multipliers. 31
AGREL Attention-Gated Reinforcement Learning. 29
AI Artificial Intelligence. 1

B

BCD Block Coordinate Descent. 31
BTM Branch-Train-Merge. 22

C

CDP Cyclic Data Parallelism. 84
CHL Contrastive Hebbian Learning. 30
CPU Central Processing Unit. 19
CV Computer Vision. 40

D

DART Diversify-Aggregate-Repeat. 22
DFA Direct Feedback Alignment. 32
DL Deep Learning. 1
DNN Deep Neural Network. 1
DP Data Parallelism. 5
DTP Difference Target Propagation. 33

E

E2E End-to-End (training all layers of a network at the same time, with backpropagation for instance). 46
EBM Energy-Based Model. 30
EMA Exponential Moving Average. 22

EP Equilibrium Propagation. 30

F

FA Feedback Alignment. 32
FG Forward Gradient. 36
FLOPs Floating point operations. 1

G

GD Gradient Descent. 3
GPU Graphics Processing Unit. 2

H

HSIC Hilbert-Schmidt independence criterion. 40

L

L-DRL Local Difference Reconstruction Loss. 33
LL Local Learning. 37
LLM Large Language Model. 1

M

MINE Mutual Information Neural Estimation. 40
ML Machine Learning. 46
MLP Multilayer perceptron. 65

N

NGRAD Neural gradient representation by activity differences. 30
NN Neural Network. 2
NTK Neural Tangent Kernel. 65

P

PAPA PopulAtion Parameter Averaging. 22
PCA Principal Component Analysis. 29
PCN Predictive Coding Network. 30
PP Pipeline Parallelism. 24

R

ResNet Residual Network. 35
RNN Recurrent Neural Network. 36

S

SG Synthetic Gradients. 39
SGD Stochastic Gradient Descent. 2
SNN Spiking Neural Network. 29
SWA Stochastic Weight Averaging. 21

T

TP Target Propagation. 33
TPU Tensor Processing Unit. 2

Z

ZeRO-DP Zero Reundancy Optimizer powered DP. 18

Symbols

f Deep Neural Network composed of J stages. ($f = f^J \circ f^{J-1} ... \circ f^1$)	2
f_n^j Stage j (one or several layers) of a DNN, on a worker n.	4
x_i^j Intermediate activation at stage j for the i-th element of the mini-batch.	4
$\theta_n^{j,t}$ Parameters of a DNN stage j for the worker n at training step t.	2
γ_t Learning rate at training step t.	18
$\mathcal{L}$ Training objective.	2
j Index of the stage of a model. ($j \in [1, J]$)	4
t Training step. ($t \in [1, T]$)	18
n Index of the worker. ($n \in [1, N]$)	18
i Index of the mini-batch data example. ($i \in [1, \mathcal{B}]$)	4

Table of contents

Chapter 1

Introduction

The Mechanical Turk was created in 1770 to impress the Empress Maria Theresa of Austria. Able to play chess at the level of a strong human player, the automaton toured Europe, defeating Benjamin Franklin and Napoleon Bonaparte. It fascinated the young Edgar Allan Poe, who nevertheless remained suspicious of the automaton's abilities. Comparing its operation to that of a deterministic calculating machine, he wrote: "It is quite certain that the operations of the Automaton are regulated by mind, and by nothing else". Indeed, he was right, as the Turk was not a machine, but was actually operated by a human hidden inside the automaton [301, 331].

The field of Deep Learning (DL) uses massive amounts of data to train large machines, with the goal of achieving human-like 'artificial intelligence' (AI). This idea has led to impressive breakthroughs in many fields, from board games [45, 301, 335, 320] to even self-driving cars [337] and medicine [32, 164]. Nowadays, modern Deep Neural Networks (DNNs) such as Large Language Models (LLM) [271, 351] can even handle multiple data types and pass the Turing Test, mimicking human interlocutors [352, 314]. These large 'foundation models', generally based on the Transformer architecture [353], are strong networks designed to be applicable to a wide range of tasks [37].

However, both the size of these models and their training datasets are growing rapidly. The number of computations required for training, measured in floating point operations (FLOPs), scales linearly with these sizes [140]. The growth of this number has been empirically exponential, doubling every 100 days with a millionfold increase predicted in the next five years [409, 284]. The reason for this increase is the pursuit of performance dictated by neural scaling laws [140, 44, 5]. These empirical statistical laws show (and predict) that performance improves as a power law of model and dataset sizes. Thus, it is estimated that the training computational requirements must scale at least as a fourth-order polynomial of the performance gains, i.e., $compute = \Omega(performance^4)$. For instance, this means that a 10× increase in performance should require 10 000× more computations [350].

This increase in training computation leads to several problems. DL training generates large carbon [123, 40] and water footprints [125, 209]. For example, training a

BERT [78] model emits about a ton of CO_2 [336]. This also results in longer training times, for instance with the BLOOM LLM taking 3.5 months to train [198]. Finally, training DNNs requires massive computational clusters, composed of many interconnected AI accelerator chips such as GPUs and TPUs [72, 163]. This results in rapidly rising hardware and power costs [68], limiting the ability to train large models to only a handful of companies, which can have detrimental effects [100, 145, 371].

In this thesis, we study the training process of DNNs, to address these increasing computational demands. In particular, we propose to modify the standard training paradigm used for DL, which has remained largely unchanged for decades. Our research aims to find novel and parallelizable training algorithms, that can improve on the training speed, memory overhead, and communication costs of the standard methods. We will focus particularly on distributed learning, where replicas of the DNNs are trained on separate devices, and local learning, where components of the DNN are trained separately without feedback.

In the following section, we first discuss the standard DL training paradigm. We will then show that few methods opt to improve on it, despite its computational limitations. Finally, we will present the background of our thesis and our contributions.

1.1 The standard DL training paradigm

1.1.1 Training DNNs with backpropagation

The main elements of DL training were already introduced decades ago. The idea of Neural Networks (NNs) appeared in the 1950s [234, 303], although limited computational power led to decades known as the AI Winters [306]. Data-driven training of DNNs then became a viable approach mainly due to two things. First, the introduction of now-standard techniques such as the backpropagation gradient estimation algorithm [219, 168, 304] and Stochastic Gradient Descent (SGD) [7, 38]. Second, the popularization of the use of GPU hardware enabled fast parallel computation for DL. This allowed AlexNet [185] in 2012 to be the first DNN to outperform classical image classification methods on the ImageNet dataset, ushering in an era of breakthroughs for DL. Even modern DNNs follow a similar training paradigm as a decade ago, which we describe here.

The goal of DL training is to minimize a training objective $\mathcal{L}$ that takes as input the output of a DNN f. This DNN is parameterized by a data input x sampled from a training dataset $\mathcal{D}$, and its parameters θ, which are modified to minimize the objective. This can be written as the following optimization problem

$$\arg\min_{\theta} \mathbb{E}_{x\sim\mathcal{D}}\mathcal{L}(f(x,\theta)). \qquad \text{(DL optimization)}$$

The standard approach for training DNNs can be described as mini-batch SGD with backpropagation. The DL optimization problem is highly non-convex, but differentiable. As such, an efficient way to find a local minimum is to use a first-order optimization

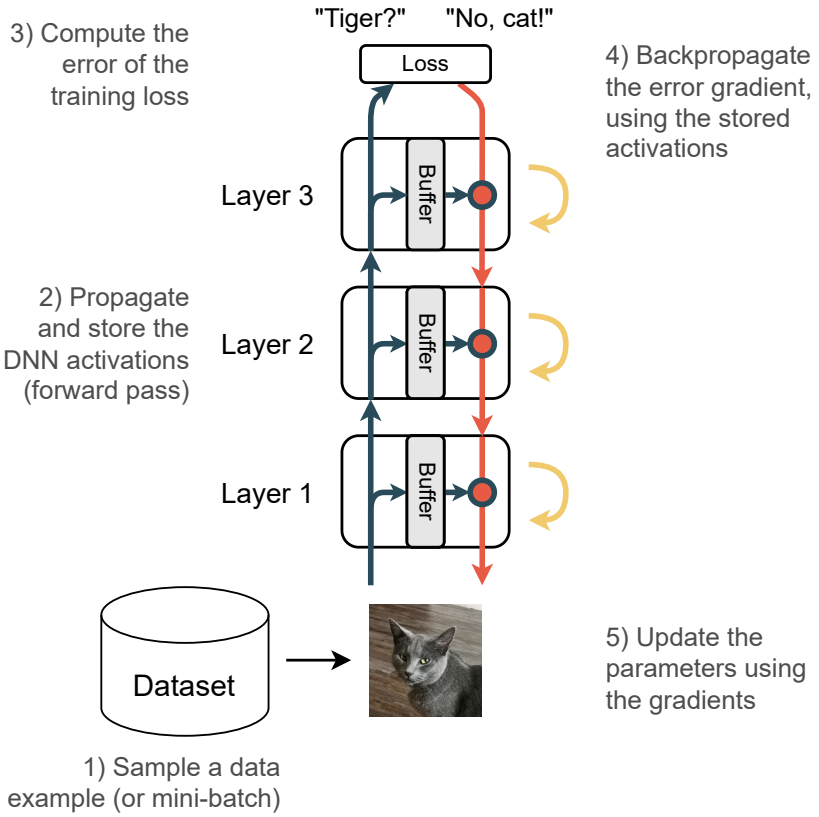


Figure 1.1: **Representation of the training of a DNN with backpropagation.** Data examples for a dataset are sequentially fed to a DNN (here, with 3 layers). It then propagates the data through a forward pass, computing and storing activations (blue). The output of the DNN is used to compute a training loss. This results in an error gradient, which is backpropagated in a backward pass, using the stored activations (red). The DNN parameters are then updated using the computed gradients (yellow).

algorithm such as Gradient Descent (GD). However, modern datasets are too large to compute GD directly, and only subsets of the dataset called mini-batches are randomly sampled at each training step. This approximation of GD is the SGD algorithm. Other more sophisticated stochastic optimization methods, such as adaptive gradient methods like Adam [172, 222], can result in faster training [338].

The mini-batch gradients used to update the parameters via SGD are computed using backpropagation, also known as reverse-mode automatic differentiation (AD). This algorithm requires two sequential steps, called 'forward' and 'backward' passes. First, at each training step, mini-batches are sampled from the training dataset and fed to the DNN, starting the forward pass. Sequentially, each layer of the network computes output values, called activations, using the activations of the previous layer

as input. These activations are then stored in a buffer for the gradient computation later. This process ends when the last layer of the network returns an output, which is used to compute the loss value of the training objective (e.g., a classification error for a given label). The next step is to compute the gradient of the loss with respect to the parameters. Backpropagation efficiently computes these gradients during the backward pass, requiring only the same number of computations as the forward pass [219]. Finally, the parameters are updated according to SGD or similar stochastic optimization methods, and a new training step begins. The backpropagation algorithm is shown schematically in Figure 1.1 and is described in more detail below.

Example 1 End-to-end backpropagation.

We consider the standard training of a DNN following end-to-end backpropagation to compute a mini-batch gradient. The computation is done in two separate stages. First, a forward pass is performed to propagate the activations of the network for a given mini-batch of input data $x^0 = (x_0^0, ...x_{\mathcal{B}}^0)$. These activations are stored in a buffer at each layer, and the training loss is computed using the DNN output $\mathcal{L} = \frac{1}{\mathcal{B}} \sum_i \mathcal{L}(x_i^J)$. Here, a DNN is composed of J sequential layers f_j, each with its own parameters θ_j, and the activations are computed by

$$x_i^{j+1} = f^j(x_i^j, \theta^j), \qquad \text{(Forward pass)}$$

Then a backward pass propagate the loss gradient with respect to the intermediate activations and parameters of each layer. This is done by using the Leibniz chain rule [348], which allows the computation to be performed from the last layer to the first, in the reverse order of the forward pass. We denote δ_i^j the activation gradient of an input i at layer j and Δ^j the parameter gradient at layer j. The training loss gradient with respect to the DNN output is equal to $\delta_i^J = \nabla_{x_i^J} \mathcal{L}$, and we obtain

$$\delta_i^j = \frac{\partial \mathcal{L}}{\partial x_i^j} = \frac{\partial f^j(x_i^j, \theta^j)}{\partial x_i^j} \delta_i^{j+1}, \qquad \text{(Backward pass)}$$

$$\Delta^j = \frac{\partial \mathcal{L}}{\partial \theta^j} = \frac{1}{\mathcal{B}} \sum_i \frac{\partial f^j(x_i^j, \theta^j)}{\partial \theta^j} \delta_i^{j+1}.$$

The parameter gradients are then used to optimize the parameters, following an optimization method like SGD.

This training procedure is generally parallelized on several devices to drastically accelerate computations, which we describe next.

1.1.2 Parallelization of the mini-batch SGD algorithm

Modern AI accelerator chips are tailored for DL training, designed to perform computations such as matrix multiplication in parallel on a massive scale [62, 173, 341]. Following the demand for more computing power, their performance has more than doubled every year, a phenomenon known as Huang's Law [72] that have allowed to tackle the rising amount of training computation. Still, training on a single chip is not enough for modern DNNs. The DL training algorithm is typically parallelized on modern deep learning clusters using thousands of interconnected GPUs or TPUs [74]. There are two main reasons for this. First, this overcomes the memory limitations of individual devices. Indeed, the backpropagation algorithm requires the storage of intermediate activations prior to the backward pass. Since the size of these activations scales with the batch size, this limits the batch size that such devices can handle, and thus the training speed. Outside of activations, even the number of parameters in modern DNNs can exceed the memory capacity of a single GPU. Second, and most importantly, parallelizing the computations leads to a speedup in training by dividing a computational task across multiple devices, allowing such clusters to reach exaflops numbers of computations per second [238].

There are two main ways to parallelize the training procedure of DL. We can categorize these as either parallelizing *how* to estimate the gradient, or parallelizing *how many* gradients are estimated, as we represent in Figure 1.2. We will refer to these methods as either Model Parallelism (MP), which parallelizes the components of the DNN itself, or Distributed Training, which parallelizes the computations on replicas of the model.

The most common distributed approach to DL training is Data Parallelism (DP). The model is replicated on different devices, and each device trains its replica on a separate micro-batch, effectively increasing the batch size [208]. After each training step, each device's micro-batch gradient must be communicated and averaged across all workers. This ensures that the parameter update is the same on all workers, and thus the parameters of all replicas remain the same. In this thesis, we refer to distributed training as all methods that train replicas of the DNN in parallel, separating them from MP approaches. Note that other works may use distributed training as a substitute for parallel training altogether, but we make this choice for a clearer separation.

The other approach to parallelization in DL training is MP, which parallelizes the gradient estimation directly. For example, tensor parallelism [316] splits layers into several sub-layers that can be computed in parallel. But the most interesting idea is to split the DNN in depth. This is a natural approach because the order of computation in a DNN is sequential during the forward and backward passes, so only one layer needs to be computed at a time. The idea is then to divide the network into sequential stages (consisting of one or more layers), that can be trained in parallel on different devices. In particular, pipeline parallelism [146] divides the mini-batch gradient computation into smaller micro-batches that can be processed in parallel by each stage. Note that DP and MP approaches are often integrated together to combine their benefits [316, 323, 325].

In this section, we introduced the standard training paradigm in DL. It consists of the mini-batch SGD algorithm with backpropagation, which is parallelized on multiple

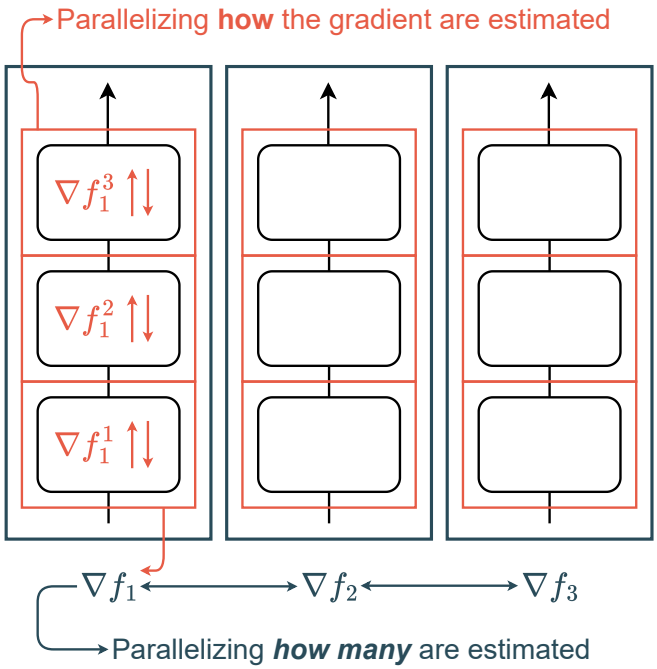


Figure 1.2: **Standard parallel training in DL.** Model Parallel approaches (red) like tensor or pipeline parallelism parallelize the computation of the gradient used in optimization. In distributed approaches like Data Parallelism (blue), the parallelization is done over the mini-batch, by scaling the gradient computation to more data at each time step.

devices using DP and MP approaches. We now present the challenge of improving the speed of this training algorithm, with the goal of mitigating the increasing computational demands of modern DL training. First, we summarize the different ways to accelerate training that are not related to the training algorithm itself. We then show the computational limitations of the current training algorithm, due to backpropagation as well as the standard parallelization approaches.

1.2 Accelerating the DL training process

1.2.1 Improving the other training components

This thesis focuses on improving the training process in DL. The training pipeline can be broadly divided into four major components: the hardware, the datasets, the model architectures, and the training algorithm itself, and each can be improved to accelerate training.

First, as discussed above, the hardware used for DL training continues to improve, allowing for faster parallel computations that can handle the increasing computational

load of modern networks. Existing hardware can also be used more efficiently. For example, mixed precision training [243, 361] halves the memory overhead with minimal loss of precision, and FlashAttention [73] uses parallelization and efficient memory read/writes to improve the implementation of the attention mechanism of Transformers. The datasets used for training can also be a source of improvement. The massive amount of examples scrapped on the internet can be refined to keep only high quality examples, thus reducing the training time for similar performance [283]. In addition, novel architectures can also reduce computational and memory costs. For example, Mixture of Experts models [232, 298, 159] use router layers to assign an 'expert' (i.e., a part of the network) to compute the output for a given data sample, reducing the inference cost of the network to a subset of layers. Alternatives to the standard Transformer architecture, such as state-space models like Mamba [407, 10], aim to avoid the quadratic cost of the attention map layer. Some approaches also suggest learning smaller networks at the beginning of training to reduce computation, before gradually increasing their size [57, 94].

Finally, the training algorithm itself is the remaining component that can be improved. Notably, many approaches opt to improve the stochastic optimization algorithm to accelerate the convergence of the training objective. Adaptive gradient methods [172, 222, 126] use quantities stored in buffers to adjust the gradient parameter-wise. Improved learning rate schedulers [322] or even learning rate-free methods [75] can also better adapt the magnitude of the update. Finally, learned optimizer approaches [239, 240, 129] are a promising way to improve stochastic optimization algorithms by meta-learning them directly.

In this thesis, we also focus on improving the training algorithm. But rather than considering the stochastic optimization algorithm, we choose to modify the training algorithm itself, from the computation of the gradient to its communication across multiple devices. In order to do so, we need to depart from the standard paradigm of parallelized mini-batch SGD. To motivate this choice, we show next the computational limitations of both backpropagation and the synchronous parallel approaches, that hinder the training speed.

1.2.2 Limitations of backpropagation

First, we present the limits of the backpropagation algorithm. There is a clear discrepancy in the energy required by backpropagation [336] compared to the energy required by the brain [16], which calls for more efficient learning algorithms inspired by biological systems. In practice, computing the loss gradient of a DNN can also be unstable, and a common problem is keeping the magnitude of the gradients in the network at an acceptable level. In fact, cascading gradient computations during backpropagation can result in either vanishing gradients with a magnitude close to zero, or conversely, gradients that grow uncontrollably [150]. Furthermore, DL optimization being a non-convex problem, there is no guarantee that descending in the direction of the backpropagation gradient will lead to a global optimum of the objective. Nevertheless, the local optima found by DL are often empirically good enough, although they can be improved with

ensembling methods [99]. Another important issue with backpropagation is its memory overhead. Storing activations during the forward pass is necessary to efficiently compute gradients during the backward pass. However, the size of these activations increases with the size of the mini-batch. A larger mini-batch results in faster training, but the GPU has limited memory, which limits the size of the activation buffers it can store, and thus the size of the mini-batch. It is possible to reduce this memory overhead, but this requires trade-offs, for instance by using more computations [58], or reducing the computational precision [243].

However, the most important computational issues of the backpropagation algorithm are the computational 'locks' defined in [155], that limit the training speed of DL. As described earlier, when training with backpropagation, each layer of the DNN is 'locked' after having propagated its activations in the forward pass. It does not perform any other computations, until it receives its associated gradient backpropagated during the backward pass. Note also that its activations must be kept in a buffer during this time. In other words, the backpropagation algorithm can be referred to as *backward locked*. This locking constrains the training of the layers of the DNN to be sequential and synchronous, forcing for instance the first layer of the network to wait for all subsequent layers to compute their gradients. Other learning algorithms that could relax this locking could allow layers to learn in parallel and even asynchronously. Thus, the computation time could be sped up, as more mini-batches could be processed by the DNN simultaneously. If for such an algorithm, a layer only requires that subsequent layers finish their forward pass, it is only *update locked*. If it can also bypass this requirement and require solely that previous layers finish their forward pass, it is *forward locked*. In the most efficient case, an algorithm could be forward unlocked, and learn even while previous layers are forwarding activations.

1.2.3 Limitations of the standard parallelization approaches

Previously, we introduced how the mini-batch SGD algorithm is typically parallelized. But the two types of approaches we discussed, DP and MP, both suffer from limitations that slow down training.

First, the synchronous communication step of DP can be an obstacle in modern training clusters. This is because communication cannot start until the slowest device has completed its computation, and the volume of communication scales with the number of devices [51]. This can cause the training cluster to slow down as devices wait for all communication steps to be completed. This is not optimal, as each device should ideally be used to its maximum capacity without wasting compute time waiting. In addition, communication can be the largest contributor to energy consumption in a cluster. Second, MP approaches are limited by the compute locks of backpropagation that we have discussed. Even though pipeline parallelism approaches allow model stages (i.e., subsets of the model layers') to compute on micro-batches in parallel, there are still 'bubbles' of idle time because later stages must wait, first at the beginning of the training step to receive microbatches of activations, and at the end while backpropagation finishes in the early stages.

We have seen in this section that accelerating the training in DL is generally done by improving the other components of the training pipeline, rather than the training algorithm itself. Yet, the standard parallelized mini-batch SGD is computationally limited, either by the need for workers to communicate for DP, or by the backward pass in backpropagation for MP. In this thesis, we propose novel training approaches for DL that diverge from this standard paradigm for better parallelization. In the following section, we provide the context of this thesis, namely the distributed training approaches that break synchronization between the replicas for more limited communication, and the MP approaches that use alternatives to backpropagation to estimate the gradients. We present them in Figure 1.3, as well as the positioning of our thesis relative to other training paradigms.

1.3 This thesis: exploring alternatives to synchronous parallel SGD

1.3.1 Thesis context

We first provide the context of this thesis. The first year of this PhD began differently, as a project to better understand the training of DNNs through the lens of simple and explainable alternatives. I focused on improving patch-based approaches to image classification [349, 42] and on learning useful convolution operators for linear node classification networks [55, 343, 177]. Despite promising results, these projects were outperformed by DL approaches, regardless of the high computational requirements of DL training. Inspired by the alternative learning approaches we studied, we focused on improving DL training through novel approaches that allow for faster parallelized training.

More specifically, this thesis was written as part of the ADONIS (Asynchronous Decentralized Optimization of MachiNe LearnIng ModelS [1]) research project of my supervisor Edouard Oyallon; in the MLIA team of the ISIR laboratory at Sorbonne Université. I also spent 3 months in Montréal in 2024 as an academic visitor in the team of Eugene Belilovsky at Concordia University and the Mila Quebec AI Institute.

1.3.2 Communication-efficient distributed training

To break from the limiting synchronous communication step of DP, other distributed training approaches can choose to limit the communication between the devices. In this case, this breaks the synchronicity between the DNN replicas. For instance, communication can be limited to asynchronous gossip between devices [114], thus allowing communications to occur simultaneously to computations [257]. Another approach is to allow multiple optimization steps on each device before communicating and averaging parameters, but the models may diverge between the averaging steps [332, 276]. However, this divergence is not necessarily a disadvantage, as the diversity in the population

[1] https://adonis-research.github.io/

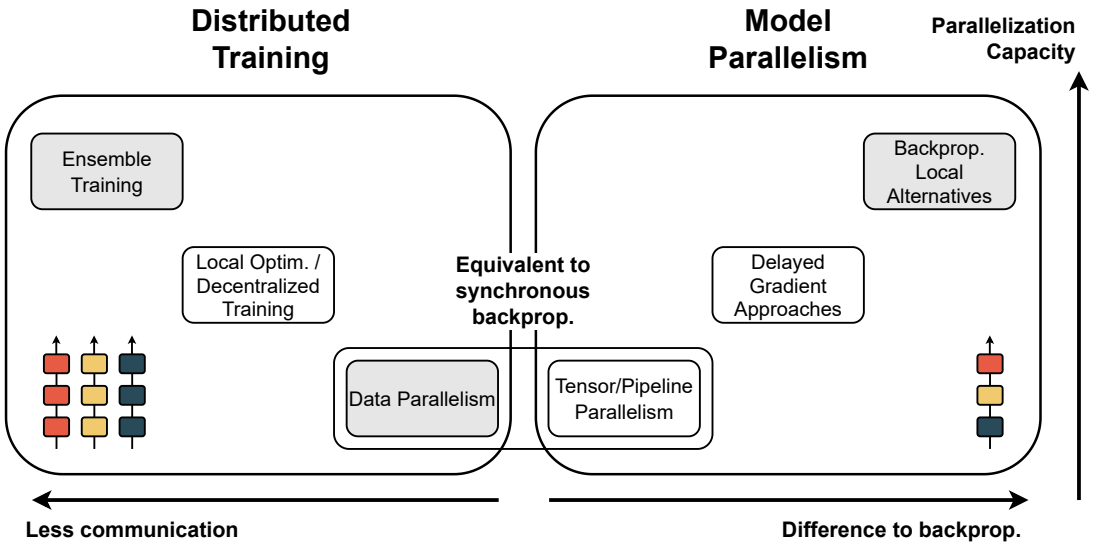


Figure 1.3: **Representation of the parallel training paradigms in DL**, divided into distributed training, which uses replicas of the model, and MP, which divides the DNN components. Standard DL training remain equivalent to backpropagation on one device, by using approaches such as data or tensor parallelism. MP approaches that use alternatives to backpropagation avoid its computational locks and allow for more parallelization. Distributed training approaches can similarly break the synchronicity requirement between replicas of the model to allow for faster parallel training since less communication is required. In this thesis, our proposed approaches can be related to the gray colored paradigms.

of models can be used for ensembling methods, which improve the generalization performance [376]. To maintain this improvement without requiring the inference cost of multiple models [237], the models obtained from this ensemble training can be fused into a single model by averaging their weights, provided that the model weights are close enough [375]. This can be done by keeping the models close during training, for instance by averaging the weights periodically. However, these operations require communication as before, and in this case collapse the models diversity. These examples show that distributed training approaches face trade-offs, between synchronicity of the models and communication.

In Chapters 3 and 4 of this thesis, we will focus on improving these two ends of the spectrum of distributed approaches: either use replicas with equal weights, or use their diversity. At one end is DP, which requires synchronous communication at every step. Because the replicas all compute simultaneously, they reach the end of the forward pass, when memory usage peaks, and the end of the backward pass, when the models must communicate, at the same time. This creates an imbalance in both communication scheduling and overall memory usage, that can be detrimental

to parallel implementations. We will focus on balancing both the communication and memory overhead. At the other end is ensemble training, where the model replicas are encouraged to diverge from each other to improve their ensemble capabilities. Still, the distance between the models must be controlled during training for them to be amenable to averaging, at the expense of both the diversity of the models and a high communication volume during training. In both cases, efficient communication is the key to improve distributed training.

1.3.3 MP approaches using alternatives to backpropagation

As for MP approaches, they aim at finding more efficient ways to estimate the gradient used in DL training. Since backpropagation is at the root of their issues, it seems natural to look for alternative gradient estimation methods that might be easier to parallelize. A first method that remove the idleness of workers in pipeline parallelism is to approximate the backpropagation gradient by using delayed quantities [259, 411], which can affect performance. Several ideas have also been proposed for a different reason, namely to find learning rules for DNNs that are more biologically plausible, i.e., likely to be used by the brain [216]. These approaches are often backward unlocked, requiring only a forward pass in the network, for instance by using forward-mode AD [319, 21, 296], which is the inverse operation to backpropagation. However, they result in a severe drop in performance compared to backpropagation. Conversely, local learning is a promising approach that is motivated by increasing parallelization while maintaining performance [265, 26]. The idea is to keep the backpropagation algorithm, but to divide the network into sequential stages connected only by feedforward connections. In each stage, a local training objective is used to compute backpropagation only in that stage. This approach has been shown to scale well compared to other alternatives to backpropagation, but still results in a performance gap with backpropagation [223, 318]. This is due to the greedy effect of local training losses, which do not optimize early stage activations to be used for later stages, resulting in an information collapse [365]. Like with distributed learning, MP approaches require trade-offs, mainly between parallelization capacity and performance.

In Chapters 5 and 6 of this thesis, we look at MP approaches with a particular focus on local learning methods. They offer a good balance between parallelization and performance, since the stages retain the benefits of backpropagation while computing separately in parallel. Several approaches have been proposed to address the resulting performance gap. For example, regularization of these losses has been shown to have a promising albeit limited effect in the case of supervised local learning [365]. On the other hand, introducing feedback from the last layer's loss could also be used to improve local learning, if this doesn't bring back unwanted computational locks. These two approaches could be a way to bridge the gap with backpropagation, resulting in highly parallelizable learning approaches with similar performance to end-to-end learning.

1.3.4 Research questions

The goal of this thesis is to propose more parallelizable alternatives to standard training paradigm in DL. It is possible to diverge from the synchronous parallelized mini-batch SGD paradigm, but existing approaches require improvements. Distributed training approaches must either keep the synchronicity between the model replicas with an expensive synchronous communication step, or break it for instance for ensemble training, but this requires careful trade-offs between communication and model diversity. MP approaches like local learning are faster by the virtue of being backward unlocked, but their performances are lower than backpropagation, and existing regularization approaches are not enough to close this gap. From these two axes, two main research questions emerge that will drive this thesis and its contributions.

- **How can we improve backward unlocked MP approaches to achieve performance comparable to traditional backpropagation?**
- **How can we improve both synchronous and ensemble training with more communication-efficient approaches?**

We have presented the background of this thesis, i.e. the parallelizable training approaches that diverge from the standard DL paradigm, and the research questions that emerge from it. We now present the contributions and structure of this thesis.

1.4 Contributions of this thesis

In this thesis, we present four contributions that deal with parallelizable approaches to DL training, divided between the two axes discussed earlier into two publications each.

First, we work on alternatives to backpropagation that are more amenable to parallelization, in particular the field of local learning, which aims at dividing the DNN into independent stages, each with its own local training objectives. We will explore ways to reduce information collapse in self-supervised local learning using data subsampling. We will also show how local learning can be used to improve forward-mode AD approaches.

We then contribute to the field of distributed learning by proposing training algorithms that focus on the communication required between devices. We show how changing the execution of computations in standard DP result in synchronous learning with improved balance of communication and memory across workers. Then, we propose a novel ensemble training algorithm that allows training a population of models in parallel with low communication volume, which can be weight averaged at the end of training into a high accuracy model.

1.4.1 Local learning approaches for DL training

Preventing Dimensional Collapse in Contrastive Local Learning with Subsampling
Louis Fournier, *Adeetya Patel, Michael Eickenberg, Edouard Oyallon, Eugene Belilovsky. ICML 2023, Workshop on Localized Learning.*

- We use contrastive self-supervised objectives for local learning. We demonstrate an information collapse between stages, and we show that it can be remedied by subsampling certain examples, either using an oracle or the stagewise feature similarity.
- *Abstract:* This paper presents an investigation of the challenges associated with efficiently training Deep Neural Networks (DNNs) via self-supervised objectives, using local learning as a parallelizable alternative to traditional backpropagation. In our approach, the DNN is segmented into distinct stages, each updated independently via gradients provided by small local auxiliary neural networks. Despite the evident computational benefits, extensive splits often result in performance degradation, a consequence of information loss between stages. Through analysis of a synthetic example, we identify a layer-wise dimensional collapse as a major factor behind such performance losses. To counter this, we propose a novel and straightforward sampling strategy based on stagewise feature-similarity, explicitly designed to evade such dimensional collapse. Extensive experiments on the STL-10 and CIFAR-10 datasets confirm the effectiveness of our proposed approach to prevent this collapse, paving the way for highly parallelized training of self-supervised DNNs on a nearly layer-by-layer basis.

Can Forward Gradient Match Backpropagation?
Louis Fournier*, *Stéphane Rivaud**, *Eugene Belilovsky, Michael Eickenberg and Edouard Oyallon. ICML 2023.*

- We investigate the use of local learning objectives to obtain gradients that can be used for forward-mode AD. We demonstrate that this allows Forward Gradient methods to scale, although a performance gap remains with backpropagation due to the difference in alignment between local and global gradients.
- *Abstract:* Forward Gradients - the idea of using directional derivatives in forward differentiation mode - have recently been shown to be utilizable for neural network training while avoiding problems generally associated with backpropagation gradient computation, such as locking and memorization requirements. The cost is the requirement to guess the step direction, which is hard in high dimensions. While current solutions rely on weighted averages over isotropic guess vector distributions, we propose to strongly bias our gradient guesses in directions that are much more promising, such as feedback obtained from small, local auxiliary networks. For a standard computer vision neural network, we conduct a rigorous study systematically covering a variety of combinations of gradient targets and

gradient guesses, including those previously presented in the literature. We find that using gradients obtained from a local loss as a candidate direction drastically improves on random noise in Forward Gradient methods.

1.4.2 Communication-efficient distributed approaches for DL training

Cyclic Data Parallelism for Efficient Parallelism of Deep Neural Networks
Louis Fournier *and Edouard Oyallon. Preprint.*

- We propose an alternative to the standard data parallelism framework by forcing a cyclic rather than simultaneous execution of workers. By balancing the total memory occupied by activations as well as the stage gradients communication during training, we demonstrate that a variety of parallel implementations using data parallelism can be improved.

- *Abstract:* Training large deep learning models requires parallelization techniques to scale. In existing methods such as Data Parallelism or ZeRO-DP, micro-batches of data are processed in parallel, which creates two drawbacks: the total memory required to store the model's activations peaks at the end of the forward pass, and gradients must be simultaneously averaged at the end of the backpropagation step. We propose Cyclic Data Parallelism, a novel paradigm shifting the execution of the micro-batches from simultaneous to sequential, with a uniform delay. At the cost of a slight gradient delay, the total memory taken by activations is constant, and the gradient communications are balanced during the training step. With Model Parallelism, our technique reduces the number of GPUs needed, by sharing GPUs across micro-batches. Within the ZeRO-DP framework, our technique allows communication of the model states with point-to-point operations rather than a collective broadcast operation. We illustrate the strength of our approach on the CIFAR-10 and ImageNet datasets.

WASH: Train your Ensemble with Communication-Efficient Weight Shuffling, then Average
Louis Fournier*, Adel Nabli, Masih Aminbeidokhti, Marco Pedersoli, Eugene Belilovsky and Edouard Oyallon. Preprint.*

- We propose a novel distributed approach to ensemble training. By randomly shuffling a very small fraction of the weights, the population of models can be averaged into a high performance final network with very low communication overhead.

- *Abstract:* The performance of deep neural networks is enhanced by ensemble methods, which average the output of several models. However, this comes at an increased cost at inference. Weight averaging methods aim at balancing the generalization of ensembling and the inference speed of a single model by averaging the parameters of an ensemble of models. Yet, naive averaging results in poor performance as models converge to different loss basins, and aligning the models

to improve the performance of the average is challenging. Alternatively, inspired by distributed training, methods like DART and PAPA have been proposed to train several models in parallel such that they will end up in the same basin, resulting in good averaging accuracy. However, these methods either compromise ensembling accuracy or demand significant communication between models during training. In this paper, we introduce WASH, a novel distributed method for training model ensembles for weight averaging that achieves state-of-the-art image classification accuracy. WASH maintains models within the same basin by randomly shuffling a small percentage of weights during training, resulting in diverse models and lower communication costs compared to standard parameter averaging methods.

1.5 Structure of this thesis

This thesis is structured as follows.

- We have first introduced our thesis subject and contributions in Chapter 1. We then summarize the literature relevant to our contributions, mainly related to parallel training for DNNs and alternatives to backpropagation, in Chapter 2.
- In the first part of this thesis, we present our two contributions to the field of backpropagation alternatives. We focus our study on approaches related to local learning, a model parallel approach to training DNNs with strong performance relative to backpropagation. In Chapter 3, we study self-supervised local learning and show that we can reduce the performance gap with backpropagation by subsampling examples from the local losses computations. In Chapter 4, we bridge local and global learning by using local losses gradients as tangent directions for forward-mode automatic differentiation.
- Then, in the second part of this thesis, we present our two contributions regarding distributed learning in DL. In Chapter 5, we propose to improve standard DP by shuffling workers execution from simultaneous to cyclic. At the opposite of synchronized distributed learning, we also improve approaches to learn a diverse population of models with limited communications in Chapter 6. By shuffling randomly parameters, we find that these models can be averaged into a high performance model.
- Finally, we present a conclusion of our thesis as well as a perspective on the approaches we tackled. After the bibliography, we also provide the Appendix of our works in Appendixes A to D. A summary of the thesis in French is provided in Appendix E.

Chapter 2

Related work

In this chapter, we propose a more thorough overview of the exisiting works related to our thesis. In Section 2.1, we first summarize the existing techniques that allow a parallel training of DNNs, divided into distributed and MP approaches. Then, in Section 2.2, we provide an overview of the approaches that more specifically propose an alternative to backpropagation, for reasons of biological plausibility or computational efficiency.

2.1 Parallel training in DL

In this section, we present in more detail the current approaches used to parallelize the training of DNNs. These approaches can be broadly classified into two categories: Distributed training, where the DNN is replicated on different workers; and MP, where the components of the DNN itself are divided into multiple stages, each assigned to a dedicated worker.

2.1.1 Distributed training: learning with model replicas

Data parallelism: synchronous distributed learning

Modern DL frameworks frequently operate on large clusters of interconnected nodes of GPUs. The standard DP framework leverages this to parallelize computations for training a DNN [74, 208]. The model is first replicated on several workers. Then at each training step, the mini-batch of data is split into separate micro-batches that are computed on separately by each replica. The locally computed gradients are subsequently averaged across all workers to obtain the gradient with regard to the full mini-batch, using an all-reduce operation, that does not necessitate a centralized parameter server. The resulting averaged gradient is used to locally update the models, typically following SGD. A schematic representation is proposed in Figure 2.1.

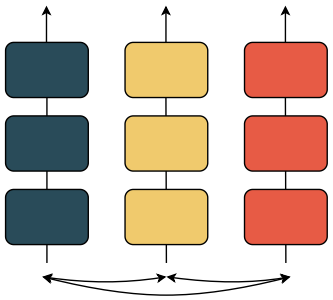


Figure 2.1: **Models training with Data Parallel.** Each color represent a model replica, communicating its gradients with an all-reduce operation after every training step (arrows).

However, this method suffers from several disadvantages. First, the model states, i.e., the DNN parameters, the averaged gradients, and the optimizer states (for instance, the momentum in SGD), are replicated on each model, resulting in the total memory increasing linearly with the number of workers. Secondly, the communication step is synchronous, as all workers must complete their gradient computations before communicating. This results in idle workers waiting for the slowest worker [138]. Furthermore, the communications costs of the all-reduce operation increase either logarithmically or linearly [51]. In Transformers architectures [353], the data is composed of a sequence of tokens. Consequently, it can be divided not only within the batch but also within this sequence. Thus, the approach of sequence parallelism [210, 180, 204] entails the splitting of token computations among the workers, in a manner analogous to that of DP. However, this approach necessitates communications between all workers for every computation of an attention layer.

Example 2 Mini-batch SGD with Data Parallelism.

In a standard DP framework for computing SGD on a mini-batch of data, each worker $n \in [1, N]$ receives mini-batch slices $x_n = (x_{n,1}, ..., x_{n,\mathcal{B}})$, called micro-batches, of size $\mathcal{B}$. The replica of the model f_n is associated with a differentiable loss $\mathcal{L}$, resulting in the training objective $\mathcal{L}_n = \mathcal{L} \circ f_n$, which takes as input x_n and is parameterized by θ_t. At each training step t, using backpropagation, each worker n computes the gradient of its local loss function with respect to the parameters for the entire mini-batch $\frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \nabla_\theta \mathcal{L}_n(x_{n,b}, \theta_t)$. Typically using an all-reduce operation [51], the gradients are averaged over the workers. Finally, each worker locally updates the parameters θ_t using the averaged gradient. With the learning rate γ_t, the standard SGD update rule [299] used in DP can be written as

$$\theta_{t+1} = \theta_t - \frac{\gamma_t}{N\mathcal{B}} \sum_{n=1}^{N} \sum_{b=1}^{\mathcal{B}} \nabla_\theta \mathcal{L}_n(x_{n,b}, \theta_t). \tag{DP}$$

Nevertheless, as datasets and DNNs grow in size [5], the memory required to store a complete model replica, as well as the activations required for backpropagation, often exceeds the memory capacity of a single device. This makes standard DP implementations impractical. To reduce the memory cost of DP, one notable technique is to shard model states across workers, thereby avoiding memory redundancy. This concept was initially proposed as Zero Redundancy Optimizer powered DP (ZeRO-DP) [289] and

subsequently implemented in PyTorch [9] under the name Fully-Sharded DP [404]. We represent the model states sharding of ZeRO-DP in Figure 2.2.

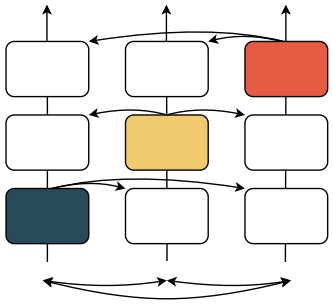


Figure 2.2: **Models training with ZeRO-DP.** The states of the models are divided into stages, with each stage being held by a single worker. This approach reduces the memory overhead but increases the communication volume.

Example 3 Zero Redundancy Optimizer powered DP.

Training with mini-batch SGD with ZeRO-DP results in the following changes during backpropagation. We assume that the DNN can be divided into J sequential stages $j \in [1, J]$ where a stage is a single layer or a set of layers. To train with ZeRO-DP, we have $J = N$ the number of workers. The worker n receives only the parameters and optimizer states of the corresponding stage n.
During the forward pass of the backpropagation training step, when the workers reach stage n, the worker broadcasts the parameters to all other workers. After computing the forward pass on this stage, the parameters are deleted from the memories of all workers except n.
During the backward pass, when reaching the stage n, the parameters are first broadcast from the associated worker as before to compute the gradients. After deleting the stage parameters, the locally computed gradients are then gathered by the worker n, which updates the stage parameters using its shard of the optimizer states. It should be noted that only this worker optimizes the stage parameters, since it holds the only version of the parameters that will be propagated to the others. Fully-Sharded DP is the same algorithm, with the sole distinction being that the model states are sharded equally among the workers for all stages, as opposed to one stage being assigned to one worker.

This method avoids the need for a linearly increasing memory, at the cost of an increase in communication volume of at most $\times 1.5$. The required memory can be further reduced by storing model states on the CPU [295, 288], at the expense of communications

between the CPU and GPU, which are much slower than inter-GPUs ones. This issue is resolved by updating the optimizer on the CPU in parallel of the gradient computations, thus hiding the communication cost. However, this requires that the parameters used for training are one optimizer step behind the latest version, resulting in a 'Delayed Parameter Update'. Delayed gradients are often a way to speed up computations, as we will see in the following section or with MP approaches. Finally, to address the increase in communication volume of ZeRO-DP, improvements have been proposed by either compressing communications [356] or striking a balance with the memory [400].

Communication-efficient distributed training

However, the main issue in DP approaches remains the need to synchronously communicate gradients between workers, which becomes more important as computational hardware improves [281]. We now discuss more general distributed methods that soften the synchronization requirement between the workers, either by allowing models to be potentially different during training, or by updating locally for a number of steps before the synchronous communication. This loss of synchronization has an obvious advantage, which is a reduction in communication volume. The decentralized distributed training methods [391, 93, 114] do not require synchronous global communication between workers. Rather, each worker communicates with its neighbors using a gossip algorithm. By using more complex algorithms leveraging Chebyshev polynomials [311, 182, 328] or the graph resistance [95, 257, 256], this can lead to accelerated communication rates, while reducing the communication overhead. In addition, the communication step can be overlapped with the gradient step, hiding it completely [14, 257, 256]. However, these methods require additional variables, that increase the memory overhead of distributed training.

Another approach, called Local SGD, proposes to maintain synchronicity between the workers but allow several local update steps on the workers before the periodic averaging [332]. This idea predates DNN training [412, 235] and is still being studied theoretically [374, 248]. Specific algorithms that mitigate the effects of the reduced averaging steps have allowed the method to be used for training DNNs, such as EASGD [398], SlowMo [359], or Post-local SGD [218, 276], using momentum buffers at the cost of an additional memory overhead. The communication bottleneck can be completely hidden by overlapping communication and computation, as in Overlap local-SGD [358], COCO-SGD [315] or CO2 [339]; and heterogeneous hardware can be handled with worker-specific computation rates [81, 230]. Federated learning, where models train on their private local data and communicate infrequently, makes particular use of such learning algorithms [236, 179, 212, 293]. This approach has also been linked to Bayesian learning [103] in [366, 86], and to subnetwork training [87, 317]. Note that in this federated case, the averaging step requires a centralized parameter server that all workers will communicate with. This is an opposite approach to the previous decentralized training methods where workers were located on potentially poorly connected graphs.

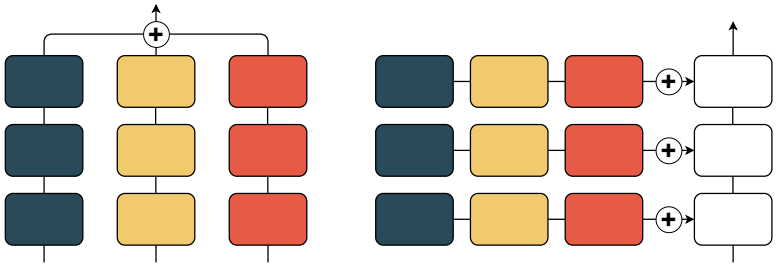


Figure 2.3: **Inference with a population of models.** The predictions of the models can be averaged, resulting in **model ensembling (a)**. Faster and more memory-efficient but with possibly worse results, inferring with the **averaged model (b)** of the population gives in an approximation of the ensemble prediction.

Ensemble training and model averaging

The distributed training approaches discussed so far trade consensus between model replicas for reduced communication overhead. Pushing this idea to its limit, the model replicas could never communicate and simply be trained in parallel without communication. With this approach, the resulting population of DNNs presents a diversity that can be used to improve prediction performance. This approach is referred to as ensembling, where the predictions of multiple models are combined to improve the ability of the predictive system to make accurate generalizations [79, 191] as well as to reduce the variance of the estimator [41]. We represent it in Figure 2.3a. We will refer to distributed training without communication as a form of ensemble training. The variance reduction of ensembling is especially effective when the population of models exhibits diversity: in particular when the models' errors are uncorrelated, i.e., that they do not fail on the same instances simultaneously [113, 99]. However, ensembling requires the memory and inference time of each of the models in the ensemble. These resources are critical for on-device inference [237], rendering the generalization improvement potentially superfluous.

To resolve this issue, the population of models can be merged into a single model to try to combine both the improvements of ensembling and the lower inference cost of a single model [213]. A simple technique is to average the weights of the different models to obtain a fused model [376], as represented in Figure 2.3b. This idea was first explored in simple linear [192] and convex scenarios [286, 39], before being explored in DL in Stochastic Weight Averaging (SWA) [151]. They establish that weight averaging is a first-order approximation of the ensemble when models are close in the weight space. Notably, a simple averaging of multiple points along the SGD trajectory leads to better generalization. SWAD [49] proposed to exclude suboptimal solutions for SWA with a dense and overfit-aware weight sampling strategy. For independently trained models, it was observed that the loss basins to which the models converge to are connectable

[104, 102] (this can referred to as mode connectivity). Thus, [33, 375] proposed learning simplexes in parameter space with a regularization penalty to encourage diversity in weight space. Similarly, [377, 290] train multiple model branches with different last-layer initializations and hyperparameters concurrently. These models are later averaged to enhance generalization and reduce the inference cost. However, for these models to be amenable to weight averaging, they must start with the same pre-trained initialization [261]. This can reduce model diversity at the expense of performance (see Figure 6 of [99]), revealing a trade-off between model diversity and weight averageability. To alleviate this issue, neuron alignment techniques [321, 3, 282, 143] match the units of multiple networks to make them amenable to weight averaging, but they rarely work in practical scenarios [162], often achieving performance below that of the individual models.

Distributed training with limited communication offers an alternative to ensemble training that can allow model averaging. Indeed, approaches have been proposed to train a population of models in parallel on heterogeneous data, while communicating to control model diversity. DART [157] and Branch-Train-Merge (BTM) [207] propose a three-phase training pipeline. The process begins with an initial shared training phase, followed by the parallel training of multiple models, each diversified by different data domains or different data augmentations. Finally, these models are merged into a single model. They find that iterative refinement of the last 2 stages enhances the overall optimization trajectory and improves generalization. To increase the diversity among the models, PAPA [161] proposes to push the model weights more gradually towards the averaged parameters using an Exponential Moving Average (EMA) throughout the training process, thus controlling the model diversity more finely. In particular, they show that training a population this way yields models that generalize better than a model trained alone with the same computational resources as the entire population, demonstrating the potential of these approaches.

These distributed methods are algorithmically very similar to the communication-efficient methods that we discussed earlier, which also traded the consensus requirement for a reduced communication volume. In particular, the training in DART and BTM is similar to the Local SGD training, where models are periodically averaged after several computational steps. PAPA, which uses an EMA of the averaged model to gradually move the models toward consensus, is similar to methods such as EASGD [398] or SlowMo [359]. Only averaging a population at the end of training, as in BTM, has also been proposed for Local SGD [330], and federated learning also uses techniques discussed previously for model merging [357, 392, 56]. Still, these approaches discussed here require an all-reduce operation to compute the averaged model. This communication may be infrequent, but results in a complete loss of the diversity in the population [157]. The EMA of PAPA is applied more frequently, resulting in a high communication cost during the parallel training of the model population, hindering its scalability as the population size increases [276].

Example 4 PopulAtion Parameter Averaging (PAPA).

Compared to previous DP approaches, PAPA does not keep the model replicas at consensus. To maintain diversity between models, the parameters are initialized differently, and the models are trained with different data augmentations and regularizations. The local model parameters are thus denoted as θ_t^n, and the averaged model as $\bar{\theta}_t \triangleq \frac{1}{N}\sum_{n=1}^{N}\theta_t^n$. In each training step, the models are updated locally before being pushed towards the averaged model using an EMA, with parameter α (adapted to the learning rate). This ensures that the models are in the same loss basin. This results in

$$\theta_{t+1/2}^n = \theta_t^n - \gamma_t \frac{1}{\mathcal{B}}\sum_{b=1}^{\mathcal{B}} \nabla_\theta \mathcal{L}_n(x_{n,b}, \theta_t^n), \quad \theta_{t+1}^n = (1-\frac{\alpha\gamma_t}{\gamma_0})\theta_{t+1/2}^n + \frac{\alpha\gamma_t}{\gamma_0}\bar{\theta}_{t+1/2}. \quad \text{(PAPA)}$$

In this section, we considered distributed training methods, that train replicas of a model on different workers, either with regular communication (DP), maintaining a constant consensus between workers, or with less frequent communication, mitigating the consensus between models, which can be used to induce diversity for ensembling. We now consider the other side of parallel approaches to DL training with MP, which directly divides the components of the network during training.

2.1.2 Model parallelism: learning with model shards

Exact backpropagation computation

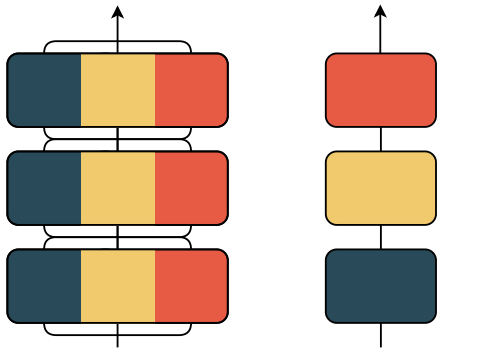


Figure 2.4: **Models training with model parallelism.** The components of a network can be divided in two different ways. In **intra-layer MP (a)**, multiple workers are used to handle the computation of a layer in parallel. On the other hand, workers in **inter-layer MP (b)** each worker computes on a separate slice of the network (called a stage).

Model Parallelism [74] refers to the family of parallel approaches in DL training that partition the network being trained, rather than the micro-batch, to parallelize computations. This family can be further subdivided into approaches that partition the layers of the network itself (i.e., intra-layer MP), or partition the network into different sequential components (i.e., inter-layer MP). These approaches are illustrated in Figure 2.4. Note that these parallelization techniques are often integrated together, with 3D parallelism [316, 323, 325], for example, combining DP and the two forms of MP mentioned above. Intra-layer MP is mostly known as tensor parallelism [158, 316, 360], and partitions the individual layers across workers. For instance, the operation of a linear layer can be subdivided by splitting the weight matrix into several sub-matrices, and concatenating the result of each matrix multiplication to obtain the final output. This method is algorithmically simple, but requires heavy modification of the DNN implementation as well as high communication costs.

Pipeline Parallelism (PP) is the standard example of inter-layer MP, and proposes instead to divide the network into sequential stages (sets of layers, as in ZeRO-DP), each on a different device. A naive training in this framework is suboptimal, since only one of the workers computes at a time since the stages are executed sequentially in the forward-backward pass, as presented in Figure 2.5a. The pipeline framework introduced by GPipe [146] mitigates this problem. It divides mini-batches into micro-batches that are passed sequentially to successive stages. This allows each micro-batch to be propagated to the next stage as soon as it has completed its forward pass, allowing stages to compute micro-batches in parallel. GPipe thus reduces the number of idle workers, but still leaves an unwanted 'bubble', shown in white in Figure 2.5b. Other pipeline schedules have been proposed to strike a balance between the size of the idle bubble and the memory overhead required to store activations and stage parameters [171]. For instance, DAPPLE [97] reduces the activation memory overhead and also combines with DP by using stage replicas to handle micro-batches partitions. GEMS [156] further reduces this overhead, but requires a much larger bubble. Similarly to GEMS, Chimera [211] uses a bidirectional pipeline, but heavily reduces the bubble in exchange for a high peak memory. [193] pushed this idea to its limit, with the same issue. These techniques are effective for synchronous parallel training. Automatic schedulers that optimize both the pipeline schedules and the use of other parallelisms help to further reduce idleness and memory overhead [97, 344, 405]. However, such as the previously mentioned communication-efficient distributed methods, some asynchrony is needed to significantly reduce the resources needed during training.

Asynchronous pipelining

First introduced with AMPNet [105], the key to completely remove the bubble of idle workers in PP is the '1F1B' (One Forward One Backward) pipeline schedule, where each worker alternates between the forward pass of one micro-batch and the backward pass of another. We represent this schedule in Figure 2.5c. To avoid the convergence problems of asynchronous learning, PipeDream [259] popularized this framework by stashing copies of the parameters, such that the backward pass of a micro-batch can be

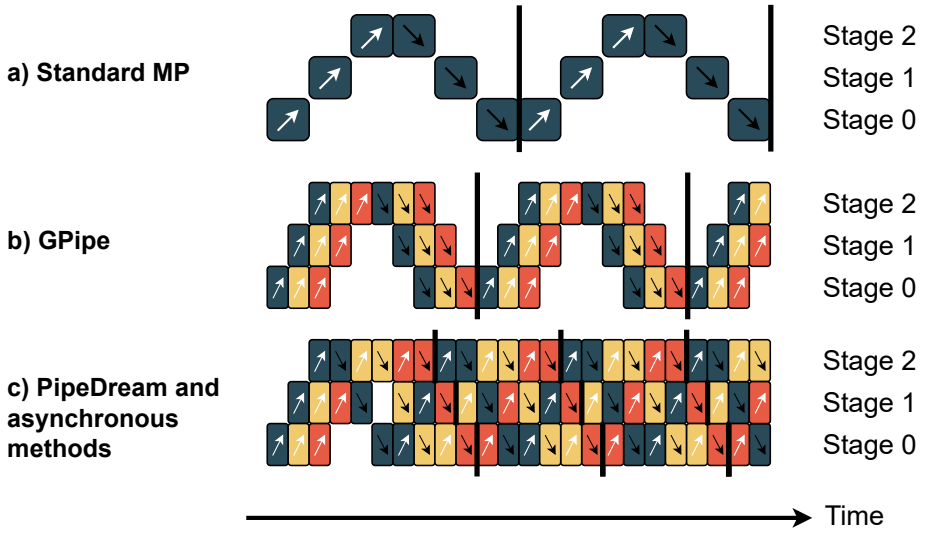


Figure 2.5: **Mini-batch execution in different pipeline frameworks.** The colors correspond to different micro-batches, and a black line represents an optimization step. A naive implementation of a **inter-layer MP framework (a)** results in a single stage (and thus worker) being computed on at a time. By splitting the mini-batch into micro-batches, workers can compute in parallel as proposed in **GPipe (b)**. This still results in idle workers, which can be mitigated by feeding micro-batches are fed as in **PipeDream (c)** at the cost of using delayed gradients. Note that in practice, backward passes take about twice as long as forward passes, compared to our schematic representation.

done with the corresponding version of the parameter used during the forward pass. However, these buffers come at a high memory overhead, which increases for the earlier stages. PipeDream-2BW [258] improves on PipeDream by keeping only two versions of the parameters and accumulating the gradients of the micro-batches to update them. This reduces the staleness to one optimization step, but the memory overhead due to micro-batch activations remains, resulting in a quadratic volume with respect to the number of stages.

Example 5 PipeDream-2BW.

In this learning framework, the network is split into J sequential stages, and each worker receives the parameters of one stage j, of which it keeps two versions at all times: θ_t^j and θ_{t-1}^j. At each time step, a new micro-batch is fed to the first worker for its forward pass, and a new mini-batch is split into micro-batches if necessary. Each worker alternates between a forward step and a backward step. In the former, it executes the forward propagation of the last micro-batch that

the previous stage communicated to it, using the updated parameters θ_t^j, and then forward it to the next stage. In the backward step, it executes the backward propagation of the last micro-batch it received from the upper stage, and similarly propagates the gradient to the previous stage. Compared to the forward pass, it uses the delayed parameters θ_{t-1}^j since the parameters have been updated since the forward pass, and using the updated parameters would not result in a valid gradient. When a worker has computed the gradients for all the B micro-batches of a mini-batch, it updates its parameters with the accumulated gradients. For the model, the resulting update rule (when training with SGD) is the following one-step delayed rule

$$\theta_{t+1} = \theta_t - \frac{\gamma_t}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \nabla_\theta \mathcal{L}(x_b, \theta_{t-1}). \qquad \text{(PipeDream-2BW)}$$

Like AMPNet, more asynchronous methods have been proposed to further decouple the stages and their forward and backward passes. These approaches can be referred to as delayed gradient training methods. In these techniques, delays occur stage-wise: the backward pass may be computed with outdated parameters or activations compared to the forward pass. For instance, [147] proposes a feature replay approach, where a forward pass first stores intermediate activations, which are then 'replayed' to compute the backward pass in parallel. This method still requires heavy synchronization between layers, resulting in a lock on computations. In [410, 411] stale gradients are computed from older parameter versions that differ from the parameters used during the update. These methods, like previous pipelining methods, are limited by their memory overhead because the computational graph is fully stored. A first step to reduce this, as proposed in Diversely Stale Parameters [385], PipeMare [387] and [181], is to keep a single set of parameters and approximate the gradients computed during the backward pass with the updated parameters, that differ from those used in the forward pass. However, the quadratic activation memory overhead still limits the scalability of these methods for many stages.

Learning with delayed gradients

Both the pipelining techniques discussed previously and distributed asynchronous training methods (and some synchronous [295]) require delayed gradients in their updates, or other delayed quantities when approximating the gradients. Due to their ubiquity, their convergence rates have been extensively studied, in the case of distributed SGD with a parameter server, which is centralized compared to our general DP case. Earlier works focused on the maximum delay [2, 333], while more recent ones show that asynchronous SGD is faster than standard mini-batch SGD even with unbounded delays [176, 379, 98, 247]. The impact of momentum on delayed SGD has also been addressed [250, 397]. In practice, several methods have been proposed to mitigate the effect of

the delay on the convergence. When possible, a simple practice is to have a warm-up period where the optimization steps are forced not to be delayed [295]. The idea is that the error due to the delay depends on the difference between the parameters that are delayed or not. As the DNN converges, the influence of the delay becomes negligible. Several techniques propose to rescale the delayed gradient, simply using the delay value [399, 382, 411] or with more elaborate schemes [18]. By approximating the Hessian matrix with the (diagonal of the) outer product matrix of the gradient, also known as the Fisher information matrix, [406] offers to rectify the approximated error due to the delay. In federated learning and delayed gradient training approaches, accumulating delayed values is another way to reduce the effect of their delay [410, 262]. Finally, in PP, another approach has been proposed to mitigate the effect of staleness. By predicting the future weights using the momentum of the optimizer, the gradients used for training are computed on parameters closer to the real parameters. The simplest approach is to directly use the momentum, scaled by the delay, as the direction of the weight change [54]. More elaborate predictions have also been proposed for SGD to further reduce the impact of the delay [181, 387], but the impact of different optimizers is still relatively unexplored [121, 122].

Example 6 Staleness mitigation methods.

Here we consider a DNN learning with SGD with momentum, that receives a mini-batch gradient (written $\nabla f(\theta_{t-\tau})$ for simplicity) computed on a previous version of the parameters τ optimization steps away. A first type of staleness mitigation method is to replace the gradient with a rescaled version, with some examples

$$\begin{aligned}
\nabla\mathcal{L}(\theta_{t-\tau}) &\leftarrow \frac{1}{\tau}\nabla\mathcal{L}(\theta_{t-\tau}) && \text{(Staleness-aware)}\\
&\leftarrow \frac{1}{1+\frac{|\theta_t-\theta_{t-\tau}|}{\gamma_{max}\mathbb{E}_t|\nabla\mathcal{L}(\theta_{t-\tau})|}}\circ\nabla\mathcal{L}(\theta_{t-\tau}) && \text{(Gap-aware)}\\
&\leftarrow \nabla\mathcal{L}(\theta_{t-\tau})+\lambda_t\nabla\mathcal{L}(\theta_{t-\tau})\circ\nabla\mathcal{L}(\theta_{t-\tau})\circ(\theta_t-\theta_{t-\tau}), && \text{(Delay-compensation)}
\end{aligned}$$

where λ_t is a control parameter. The other technique is to predict the weights τ of later optimization steps using the SGD momentum (noted as v_t). These predicted weights are then used for the gradient computation $\nabla\mathcal{L}(\hat{\theta}_t)$, with

$$\hat{\theta}_t \triangleq \theta_{t-\tau}-\tau\gamma_{t-\tau}v_{t-\tau} \qquad \text{(Weight prediction)}$$

As previously observed, parallelizable approaches in DL training, whether following the distributed or MP paradigm, often requires more intricate computational techniques to obtain, communicate, or use the backpropagation gradient, potentially with delays.

As the source of the limitations for efficient learning at scale, it is no surprise that a vast literature aims at finding alternatives to the backpropagation algorithm, which we will discuss in the following section.

2.2 Alternatives to backpropagation

Despite the ubiquity of the backpropagation algorithm, this gradient estimation method is plagued by two main problems: its biological implausibility, which we will discuss next, and its computational cost. For these reasons, other ways of training DNNs have been investigated, with the end goal of finding a biologically plausible, energy-efficient, and faster DL training algorithm. We present in this section different families of alternatives to backpropagation, and we summarize some of the notable ones in Table 2.1. We note how biologically plausible they are (notably, if they require weight transport), whether the training objective followed is global (at the end of the DNN) or local, their computational lock, whether they approximate the backpropagation gradient, and finally a synthetic representation of the performance of DNNs trained according to these frameworks. We will first explain why backpropagation is considered biologically implausible, before presenting bio-inspired learning approaches.

Table 2.1: **Summary of the major alternatives to backpropagation (BP)** We propose to summarize the alternatives discussed in Section 2.2, on their biological plausibility (Bio. plsb.) and if they require weight transport (WT). We also summarize whether their learning objective is global or local (Objective), what are their computational locks (Locking), if they approximate backpropagation (≈ BP) and their approximate performances (Perform.).

Algorithm	**WT**	**Bio. plsb.**	**Objective**	**Locking**	**≈ BP**	**Perform.**
Backprop.	✓	✗	Global	Bwd		+++
Hebbian [310]	✗	✓	Local	Fwd	✗	
Fdbck. Align. [264]	✗	≈	Global	Updt	✗	+
Equil. Propa. [313]	≈	≈	Global	Bwd	✓	+++
Target Propa. [92]	✗	≈	Global	Bwd	✓	+++
Forward Grad. [21]	✗	✗	Global	Updt	✓	+
Synth. Grad. [155]	✓	✗	Local	Bwd	≈	+
Local Learning [26]	≈	≈	Local	Fwd	✗	++

2.2.1 Biologically plausible alternatives

One of the main reason for searching alternatives to backpropagation is to avoid its biological implausibility. The idea is that, although the structure of DNNs is inspired by the network of neurons found in the brain, it seems unlikely that the brain learns according to a mechanism equivalent to the backpropagation used for DNNs [297, 216].

In fact, several problems can be highlighted. First, the backpropagation algorithm suffers from the weight transport problem [47, 69, 119]. Forward and backward propagation require two types of connections between neurons: feedforward and feedback ones. Furthermore, both require the knowledge of the same weights to compute the backward pass associated with a forward pass, an impossibility in the brain known as the weight transport problem [216]. Other similar synchronizations and computations seem difficult to reconcile with our current knowledge of the brain. Neurons must be able to compute and communicate their activations derivative [272]. They must wait for the backward pass before updating with the corresponding activations (a point which we will discuss next). Even the main problem that DNN training aims to solve, which is the minimization of a training objective, is not necessarily the same objective that is followed by synaptic plasticity in the brain [297].

Most importantly, as described in the introduction, the brain is able to learn very efficiently [88] while requiring much less energy than backpropagation [336, 16]. For these reasons, it seems necessary to find more biologically plausible alternatives to backpropagation that could train networks as efficiently as the brain.

Hebbian learning

Computational neuroscience aims to develop biologically plausible neural systems using local learning rules that do not require backpropagation. Following the Hebbian principle [132], the synaptic plasticity of neurons is modeled such that neurons that "fire together, wire together". Oja's rule improve on this idea by adding a weight decay term [269]. The weights of the neurons following this rule align with the principal component of the input data. The Generalized Hebbian algorithm [310] extend on this subspace learning as it computes the Principal Component Analysis (PCA) of the data. The Winner-Takes-All learning rules leverage additional inhibitory interactions [305, 226, 252]. Other approaches called three-factor Hebbian learning models [106], such as REINFORCE [373], use a reward signal to improve the learning rule. Attention-Gated Reinforcement Learning (AGREL) [302] and Q-AGREL [287] add an attention gating mechanism to select the neurons that learn.

Another approach is to consider a different modeling of biological neural networks, for example with Spiking Neural Networks (SNNs) [389, 266, 190]. In this framework, activations are not represented as static values, as in standard DNNs, but as temporal activity spikes, with their values encoded in their timing or frequency. Their training remains a challenge because standard backpropagation is impossible in such DNNs [190]. Several learning rules have been studied for SNNs, such as Spike Time Dependent Plasticity [34], the equivalent of Hebbian learning for SNNs, or the BCM rule [35, 186], which uses a weight decay term as in Oja's rule.

Energy-based approaches

Three different families of backpropagation alternatives can be categorized as energy-based approaches. They opt to view the learning step not as two phases to propagate a

gradient, as with the forward and backward passes of backpropagation, but two phases where the model converges to different states. They are based on Energy-Based Models (EBM) [200] where the training objective, called energy, is a more general function than in standard backpropagation. It depends on both the classical supervised loss but also an additional internal energy defined by the model internal state. The fundamental EBM is the Hopfield network [141], which has binary neurons that serve as a kind of biological associative memory.

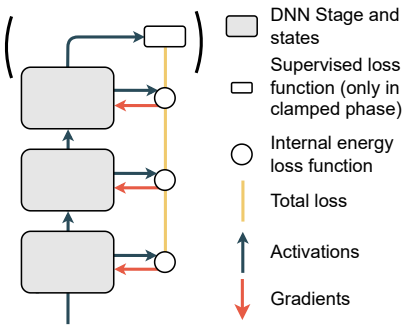


Figure 2.6: **Representation of training with Equilibrium Propagation and similar energy-based methods.** In a first free phase, the model converge according to an internal energy. In the second weakly clamped phase, a supervised loss is added (in parentheses). The scaled difference between the parameter gradients of these two states approximates the backpropagation gradient.

The first family, Contrastive Hebbian Learning (CHL) [255, 267, 383] was introduced to train continuous Hopfield models [142]. In the first free phase, only the input is fixed, the network converges to an equilibrium and then update its weights following a Hebbian rule. In the second clamped phase, the output is also fixed, and after the model converges again, an anti-Hebbian (i.e., negative) weight update is performed. Since it uses neural activation difference rather than gradients, this kind of approach is referred to as 'neural gradient representation by activity differences' (NGRAD) methods by [216]. NGRAD approaches, such as GeneRec [267] recirculation [136] or [369, 308], are seen as more closely following our understanding of feedback in the brain than backpropagation.

Predictive Coding Networks (PCNs) [292, 272] are another EBM approach. They are related to the neuroscience principle of free energy [246], where the network perform both inference and learning, by learning to predict the input. PCNs approximate backpropagation [370, 326] although other approaches try to differentiate from this approximation [6, 327, 242].

Finally, Equilibrium Propagation (EP) [313] is an EBM approach similar to CHL. Rather than clamping the output value during the second phase, the outputs are 'nudged' towards the target with a clamping loss term. Improvements of EP have been proposed to avoid the weight transport problem [175] or to make it more amenable to DNN training [189], notably by updating the parameters during the clamped phase and not only at the end [91]. These three EBM approaches can be unified as approximating backpropagation [245], as all minimize a supervised loss and an additional internal energy. PCNs follow a CHL algorithm using the variation free energy instead of the Hopfield internal energy and EP is an infinitesimal perturbation of the EBM loss function.

Example 7 Equilibrium Propagation.

Training with Equilibrium Propagation is done in two successive stages. In each, the state X of the network (i.e., all its activations) converges by minimizing a total energy E composed of two terms. The first term is a standard supervised loss $\mathcal{L}(X)$, computed with the output of the DNN. The second term is an internal energy – or primitive – function on the state of the network. It is generally a form of Hopfield energy [141, 30], which translates into a symmetric weighting requirement [91]. If the DNN is a multilayer perceptron (MLP), with pre-activation states x_i, an activation function ρ, and layer weights and biases W_{ij}, b_i, the internal energy is defined by $\mathcal{I}(X) = \frac{1}{2}\sum_i x_i^2 - \frac{1}{2}\sum_{i\neq j} W_{ij}\rho(x_i)\rho(x_j) - \sum_i b_i\rho(x_i)$. Then the total energy function is $E(\beta, X) = \mathcal{I}(X) + \beta\mathcal{L}(X)$ where β is an influence parameter. In the first 'free phase' the states of the model converge only according to the internal energy $E(0) = \mathcal{I}$. This results in the state $\bar{X}_0$ such that $\frac{\partial E(0,\bar{X}_0)}{\partial X} = 0$. In the second 'weakly clamped phase', the model converges following the total energy for a small value of β. We obtain $\bar{X}_\beta$ such that $\frac{\partial E(\beta,\bar{X}_\beta)}{\partial X} = 0$. Finally, the parameters θ of the network are updated using the following value Δ

$$\Delta = \frac{1}{\beta}\left(\frac{\partial E(\beta,\bar{X}_\beta)}{\partial\theta} - \frac{\partial E(0,\bar{X}_0)}{\partial\theta}\right). \qquad \text{(Equilibrium Propagation)}$$

When β tends to zero, this value tends exactly to the backpropagation gradient

$$\begin{aligned}\lim_{\beta\to 0}\Delta &= \lim_{\beta\to 0}\frac{1}{\beta}\left(\frac{\partial E(\beta,\bar{X}_\beta)}{\partial\theta} - \frac{\partial E(0,\bar{X}_0)}{\partial\theta}\right) \qquad \text{(EP limit)}\\ &= \frac{\partial}{\partial\beta}\left(\frac{\partial E(0,\bar{X}_0)}{\partial\theta})\right) = \frac{\partial}{\partial\theta}\left(\frac{\partial E(0,\bar{X}_0)}{\partial\beta})\right) = \frac{\partial\mathcal{L}}{\partial\theta}.\end{aligned}$$

Auxiliary variables

An alternative family of DNN training method is to rewrite the DNN learning problem using auxiliary variables. This rewriting of the optimization problem is similar to the previous energy-based approaches, but with a greater focus on the optimization task rather than biological plausibility. Still, this approach requires one forward and backward pass before even tackling the local sub-problems [64]. This idea was proposed in [70], using Lagrange multipliers. Auxiliary variables replace the layer activations (pre or post-activation function) and decompose the deep optimization problem as a combination of coupled simpler problems, that can be solved in parallel. A less restrictive idea proposed using auxiliary variables between stages of the DNN and not all layers [115]. Two different approaches tackle these problems: the Block Coordinate Descent (BCD) methods, and the Alternating Direction Method of Multipliers (ADMM) ones. BCD approaches solve the sub-problems cyclically, keeping the other blocks fixed. Meanwhile, ADMM approaches first use the saddle points of augmented Lagrangian

functions via primal and dual variable updates. The primal variables are updated in an approach similar to BCD.

A BCD approach to DNN training was first proposed in [196]. Lifted Neural Networks [13] is a similar BCD-inspired approach based on representing the activation functions of a DNN as the argmin of a corresponding convex optimization problem. Improvements have been proposed simultaneously in Fenchel Lifted Networks [120] and Lifted Proximal Operator Machines [206]. [395] provides a deeper theoretical understanding of these BCD methods. Regarding the ADMM approaches, the Method of Auxiliary Coordinates [48] uses quadratic penalties to enforce the sub-problems constraints, however, it then requires solvers for these sub-problems. Two concurrent methods have been proposed using ADMM to train the sub-problems more effectively: very deep supervised hashing [401], and [345], which uses Bregman iteration methods. Rather than neuron-level computations, [396] allows for layer-level ones. However, these methods are not adapted to mini-batch training, except for [64].

Learning with no weight transport

Finally, we discuss approaches that aim to avoid the weight transport problem discussed above. Indeed, the credit assignment problem [217] remains a challenge for computational neuroscience. It asks how to assign credit (or blame) to neurons in early layers, for changes in later layers. The weight transport problem makes the answer proposed by backpropagation unsatisfactory, and thus some methods have been proposed to address this specific problem. For example, in [124], segregated dendrites are used to receive feedback and compute local error signals. To train DNNs on large datasets, two main ideas have been proposed.

Feedback Alignment (FA) was introduced by [217] as an approach that avoids the symmetric backward connections of backpropagation by greatly relaxing this requirement. The idea is to propagate the error not with the same weights as in the forward pass, but with random and fixed feedback weights between layers. This method shows promising results in simple cases and is more biologically plausible than backpropagation, and we represent in Figure 2.7a. It has also been coupled with CHL methods in [77]. Sign-symmetry is a slightly relaxed version of FA that allows feedback weights to be equal to the sign of the feedforward weights, and has shown some promising results despite this limited information [215, 381]. Direct Feedback Alignment (DFA) [264] extends the idea of FA even further by not propagating the error through the layers to reach lower stages. Rather, the loss error is given directly to each layer, directly scaling the random fixed layer weights and thus further parallelizing the process. We represent training with DFA in Figure 2.7b. A similar idea is presented in Kickback [17]. [294] studies the learning dynamics of DFA and finds that the DNN weights first learn to align with the feedback weights, before memorizing the data. However, although DFA has been applied to special Optical Processing Units [268], which are photonic coprocessors used for large-scale random projections, this learning approach does not scale well for large-scale DNN training [197].

Instead of using fixed feedback weights, another proposed alternative is to have

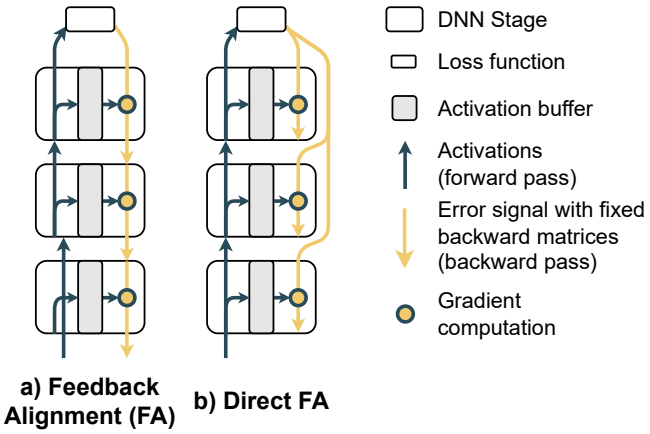


Figure 2.7: **Training with feedback alignment methods.** The feedback connections in a DNN are not symmetric as in backpropagation. **(a)** In Feedback Alignment, the backpropagation algorithm is exactly the same, with the only difference being the use of random and fixed feedback weights. **(b)** To avoid a slow backward pass, Direct Feedback Alignment parallelizes the process by using the error signal directly at each layer instead of propagating it.

distinctive but learnable feedback weights. Target Propagation (TP) was first introduced in [29] with this idea. Targets are propagated from the end of the network with separate feedback operators, and the difference between activations and targets is used to update the feedforward operators. In this approach, the DNN can be seen as a stack of auto-encoders [216], a type of network that learns to reconstruct its input [1]. However, there is a significant reconstruction error in standard TP, which is mitigated in Difference Target Propagation (DTP) by modifying the propagation formula [202, 19]. Other similar approaches have been proposed, such as Local Representation Alignment [275, 274], weight mirror approaches [188, 4], or using auto-encoders [165]. Finally, using the links between DTP and Gaussian-Newton updates [28, 241], [92] introduces Local Difference Reconstruction Loss (L-DRL), which ensures that the feedback-pass Jacobian matches the transpose of the feedforward-pass Jacobian, by using a more sophisticated learning scheme.

Example 8 Local Difference Reconstruction Loss.

This improvement over DTP uses noisy perturbations to train the feedback weights and ensures that the feedback path computes the Jacobian of the feedforward path in expectation and in the limit of small noise. The feedback equivalent of the feedforward layer f_j and parameters θ_j are g_j and ω_j respectively. With $\theta > 0$ and $\epsilon, \mu \sim \mathcal{N}(0, \theta^2)$, the feedback weights at layer j are trained following the following

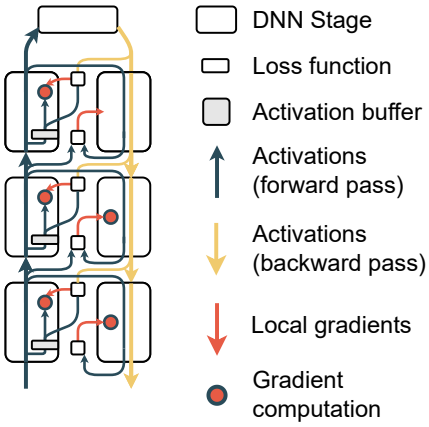


Figure 2.8: **Representation of training with target propagation methods.** As in FA, the feedback weights are different from the feedforward weights. In this case however, they are learned, by using perturbations of activations before or after the forward connections to align the backward ones.

L-LDR loss

$$\mathcal{L}_j = -\epsilon^\top \Big(g_j(f_j(x_j + \epsilon, \theta_j), \omega_j) - g_j(x_{j+1}, \omega_j) \Big) + \frac{1}{2} \| g_j(x_{j+1} + \mu, \omega_j) - g_j(x_{j+1}, \omega_j) \|^2 . \tag{L-LDR}$$

2.2.2 Computationally efficient alternatives

Other alternatives to backpropagation have been proposed, that focus more on improving the efficiency of the backpropagation algorithm than on making it more biologically plausible.

Memory-efficient approaches

Without many modifications to the backpropagation algorithm, two approaches, checkpointing and reversibility, allow to reduce the memory required to store intermediate activations during the forward pass [324]. Tangentially related, compression and quantization schemes are another family of approaches that aim to reduce the memory overhead in DL, but without modifying backpropagation [260, 249]. They can also reduce the communication overhead in distributed learning [46, 128, 386].

Activation (or gradient) checkpointing is a method that trades additional computations for a reduced memory overhead due to the buffered activations [58]. The main idea is not to store all the activations required during the forward pass of the DNN. Rather, the activations chosen to be stored are considered as a kind of 'checkpoint'. During the backward pass, all activations not stored in memory are computed again, starting

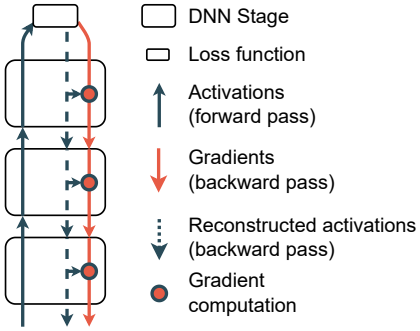


Figure 2.9: **Representation of training with reversible backpropagation in reversible networks.** Unlike standard backpropagation, there are no buffers to store the intermediate activations computed during the forward pass. Instead, the activations needed to compute the gradients are reconstructed during the backward pass, using the output of each stage.

from the last checkpoint. In other words, the memory overhead is reduced, for the approximate computational cost of an additional forward pass for the activations not stored. This idea has been combined, for example, with modern parallelism implementations [170, 221]. Assuming that the J stages of the network are homogeneous, with k the number of stages kept as checkpoints, the maximum number of stages with stored activation memory at any time is k (number of checkpoints) $+\frac{J-k}{k}$ (interval between two checkpoints, fully used during the backward pass). Minimizing this simple case yields the classical number of checkpoints used in modern DL implementations [9], which is every $\sqrt{J}$ stages [133]. However, this hides an important disparity among different DNN architectures. For instance, [180] shows that Transformers [353] can be particularly optimized by checkpointing just before the attention blocks. More generally, the heterogeneity of modern DNN models requires automatic planners to compute the optimal checkpointing schedule [133, 403].

Another approach proposes to reduce the activation memory overhead even further. Inspired by nonlinear independent components estimation [80], which learns a bijective transformation, [110] first proposed RevNets, a reversible DNN based on the ResNet architecture [130]. Reversible DNNs are particular architectures of DNNs that are composed of layers that are invertible. This means that the input of a layer can be computed only from the output of the layer. Such approaches allow to avoid the storage of intermediate activations during the forward pass, since they are reconstructed and propagated with the gradients during the backward pass. However, some layers may not be invertible, requiring the buffering of some activations. Furthermore, reconstruction has a computational cost equivalent to a forward pass, similar to activation checkpointing. Invertible networks is the name given to reversible networks that consist only of invertible layers, removing any activation buffer in the DNN. For example, in convolutional architectures, this requires removing dimensionality reduction steps such as

downsamplings [22, 153]. Reversibility is not limited to one type of architecture or task. Such networks have been used extensively for generative models [12], denoising [220], inverse problems [11], and in other architectures such as Recurrent Neural Networks (RNNs) [52] and Transformers architectures [229, 214]. We represent such methods in Figure 2.9.

Forward pass approaches

The backward pass of the backpropagation algorithm is the reason for its computational locks and activations buffering. Thus, several alternatives propose to train the network with forward passes only. Some methods propagate additional data along with the input to update the layers locally. The forward-forward approach [135, 273] uses two forward passes to propagate positive and negative data. Meanwhile, signal propagation [174] proposes to propagate both the input and a learning signal (e.g., the label in a supervised setting) with the same weights, which are updated using a local loss on the propagated values.

The most common approach to using only forward passes in DNN training is to approximate the backpropagation gradient. Zero-order methods [108] perturb the activations or weights of the model directly, and use finite differences of the losses to approximate the value of the directional derivative of the loss gradient along the perturbation [329]. This idea was first used in early alternatives to backpropagation as weight perturbation [233] and node perturbation [152, 372, 368] methods. However, learning with such methods is very slow due to the variance of the estimation [53]. Nevertheless, this method has shown some success in some fine-tuning tasks, as presented in MeZO [228].

Another similar approach has been proposed that takes advantage of recent deep learning frameworks. The backpropagation gradient is computed using backward-mode AD. Forward-mode AD [367, 117] reverses the order of computation of the gradient. Computations in the chain rule are performed from the input gradients, from the first to last layer. Although it is computable during the forward pass, this method is not applicable to DNN training because the number of computations required to obtain the gradient scales linearly with the input size. This is not competitive with backpropagation, as backward-mode AD is constant with respect to the input size. However, there is a way to make this computation as fast as backpropagation: computing the directional derivative of the loss instead of the entire gradient. In other words, the backpropagation algorithm can be seen as a vector-Jacobian product, where the loss value is a scalar vector and the Jacobian of the DNN is its gradient. Given a tangent vector of the same size as the input, forward-mode AD computes a Jacobian-vector product, where the vector is the tangent. This only returns a scalar value, the directional derivative of the loss with respect to the tangent. Multiplied by the tangent, this is equal to the projection of the gradient on the tangent. Silver et al. [319] showed that this idea could be used to train RNNs, as it provides an ideal candidate to solve the issue with long-term Backpropagation-through-time. [21] popularized using this idea as an alternative to backpropagation under the name Forward Gradient (FG), to train DNNs

in a single forward pass. More specifically, by sampling stochastic Gaussian tangents, they show that the gradient projection computed with forward-mode AD provides an unbiased estimate of the gradient. This training framework is represented in Figure 2.10a. Further theoretical analysis of the FG algorithm was provided by Belouze [27], which showed that sampling the tangent from a Rademacher distribution provides the unbiased estimate with minimum variance. Although the method by which such derivatives could be computed is unclear, this method is more biologically plausible than backpropagation, as it removes the weight transport problem. Following this idea, [15] combines FGs with DFA. To scale FG, [296] proposes three improvements that reduce the variance of the estimate. Instead of perturbing the parameters to compute the FG, they show that, depending on the architecture, perturbing the activations provides a reduction in variance. Instead of trying to approximate the end-to-end loss, they propose to approximate local losses – which we will discuss next – to reduce the size of the estimated Jacobian, as represented in Figure 2.10b. Finally, similar to [279], the output channels are divided into groups, each associated with a local loss, further reducing the variance. Even with these changes, the FG scales poorly and no longer approximates the backpropagation gradient.

Example 9 Forward Gradient.

In the Forward Gradient framework, a tangent direction v with the same size d as the parameters θ is sampled from a distribution with a zero mean value, and a unit covariance. For instance, $v \sim \mathcal{N}(0, I_d)$. Then, during the forward pass, we compute a Jacobian-vector product $\left\langle \frac{1}{\mathcal{B}} \sum_i \nabla_\theta f(x_i, \theta), v \right\rangle$. Finally, the parameters θ of the network are updated using Δ, which is the projection of the mini-batch gradient along the direction v, an unbiased estimate of the backpropagation gradient

$$\Delta = \left\langle \frac{1}{\mathcal{B}} \sum_i \nabla_\theta f(x_i, \theta), v \right\rangle v \,. \qquad \text{(Forward Gradient)}$$

Supervised Local learning

Finally, we present the family of approaches that we refer to as Local Learning (LL) [83]. Also referred to as Modular, Layerwise, or Greedy Learning, we categorize here all the approaches that split a DNN into stages (possibly at the level of a single layer), connected only by feedforward connections, and each with its own local training objective, as represented in Figure 2.11b. We will start by describing LL approaches in the context of supervised learning.

The use of local losses has emerged for reasons other than LL. First, the addition of intermediate local losses to the main training objective [201] can be used to improve the performance of a DNN. By having the network randomly choose a local loss as its output,

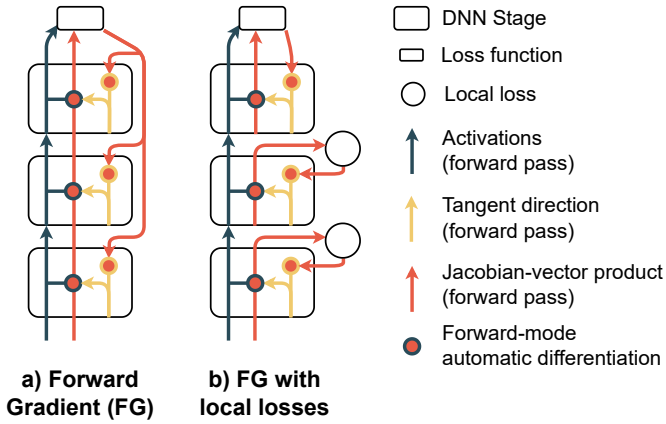


Figure 2.10: **Training with forward gradient methods. (a)** Using a random tangent direction (yellow), a Jacobian-vector product is computed during the backward pass. The value multiplied by the tangent is then used to update the weights. **(b)** To reduce the variance of the gradient estimate, local losses can be used.

it can also speed up training as this amounts to training a smaller network [112]. A related idea has been proposed to not backpropagate through early layers after a certain portion of the training run to train the model faster [43]. Having intermediate local losses can also allow to look at the DNN differently. [144] proposes to consider a ResNet model [130] as a stack of weak classifiers and the whole network as a strong learner. Meanwhile, inspired by the Cascade-Correlation Learning Architecture [96], [231] trains DNNs with Deep Cascade Learning by incrementally increasing the size of the network. Other similar works train a network by increasing its size layer by layer, such as [355], [253] which is inspired by boosting theory, and [167] for generative networks. In particular, [67] proposes to grow a DL model by trying candidate subnetworks to append to the existing network. Finally, the idea of local losses that allow classification at intermediate points of the network can be used to have early exits in the DNN [347, 362, 312]. These are especially important in adaptive inference, which aims to reduce inference time [205, 194, 390].

Our main point of interest in local losses for this thesis is different from the previous ideas. LL can be used to train a network layer by layer (or stage by stage), preserving only the feedforward connections between layers. This results in an update unlocked learning algorithm, that can be parallelized more easily than other MP approaches like PP, which relies on backpropagation. It is also possible to train each layer to convergence before starting to train the next layer. This idea of supervised layerwise learning has been explored in early works but only for simple problems [203, 134], or for a limited class of computer vision problems [227]. In [254], a random classifier matrix is used to produce the stage prediction, and [8] rather uses the difference between pre and post-activation values for their local losses. LL approaches have also been related to

kernel learning. For instance, [187] trains a network layer by layer by solving a kernel problem at each layer. [84, 85] propose to view DNNs as a stack of linear models in feature spaces (looking at the pointwise activations as being applied at the beginning and not at the end of the layers) and link them to kernel machines. Then, a contrastive objective is applied to train these local kernels. However, all previous LL methods remained non-competitive with backpropagation except in simple cases.

As discussed at the beginning of this section, [155] first proposed the different locks of backpropagation. In this work, the authors also proposed an update unlocked training method using Synthetic Gradients (SG), which is slightly different from the classical LL paradigm. In it, instead of a local loss, each stage has an auxiliary network that outputs a synthetic gradient, that is fed back to the previous stage. The error between this synthetic gradient and the real one is used to update the auxiliary network during a later backward pass. This idea is illustrated in Figure 2.11b and has been studied in more detail in [71].

LL has been shown to be competitive with backpropagation even for large-scale image classification tasks in Greedy Learning [25], by demonstrating that using appropriate trainable auxiliary networks before computing the local loss, as in [155], allows to improve the performance of the DNN. Taking this approach further, it is possible to have a forward unlocked learning algorithm called Decoupled Greedy Learning, by storing activations in a buffer at the beginning of each stage and replaying them to train the stages asynchronously [24, 26]. Other improvements to this method have been suggested. [265] uses a combination of two different supervised local loss functions to improve LL. More specifically, the addition of a supervised clustering loss improves on the use of a cross-entropy loss alone. Another exploration proposed by [279] is to add width-wise modularity to the depth-wise modularity of LL by dividing the output of the stages into groups, each with its own local loss. Finally, some approaches have proposed intermediate approaches with more standard backpropagation learning. [109] proposes a method between local and global learning, where the activations of each stage are propagated to a few other stages before using a local loss to compute its gradient. Thus, in this framework, stages are connected with feedback weights, but backpropagation is still limited to a few stages. [263] offers a different setting, related to model distillation. A 'meta-model' is first pretrained with backpropagation. Then each stage is trained separately, keeping the other stages frozen from their pretraining initialization. The concatenation of the trained stages results in the final model.

Example 10 Greedy Learning.

In Greedy Learning and similar LL approaches, the network is split into stages and each stage has at his disposal an auxiliary network a_j (parameterized by ω_j) associated with a local loss. Here we consider that this local loss is the same supervised loss used to train the output of the DNN. Then, for each mini-batch fed to the network, training at a stage j proceeds as follows: first, a forward pass

propagates the activations as in standard backpropagation $x_i^{j+1} = f^j(x_i^j, \theta^j)$. A 'stop gradient' operation is used to avoid any feedback connection from the next stages. Then, the activations are fed into the stage auxiliary network and the local loss to compute the mini-batch loss $\mathcal{L} = \frac{1}{B}\sum_i a_j(x_i^{j+1}, \omega_j)$. Then, a backward pass is computed solely for the auxiliary network and the stage j to compute the gradients

$$\Delta_a^j = \frac{\partial \mathcal{L}}{\partial \omega^j}, \Delta^j = \frac{\partial \mathcal{L}}{\partial \theta^j}, \qquad \text{(Greedy Learning)}$$

which are used to update both the stage and the auxiliary network, following standard optimization algorithms.

Nevertheless, these LL approaches maintain a performance gap with models trained with backpropagation. One explanation for this gap is that local losses greedily optimize the supervised loss, producing suboptimal intermediate representations, as shown by [365]. To address this problem, they propose InfoPro, by adding an expensive reconstruction loss to the local supervised loss to preserve information in the intermediate representations. A similar regularization approach has been proposed by [166], inspired by the minimizing movement scheme of optimal transport. [309] analyzes further local training through the lens of the information bottleneck. They show that backpropagation allows for better propagation of information through the network, and that information compression is approached differently for different stages of the network. Meanwhile, models trained with local training show uniform compression across all stages, or, when trained with InfoPro, a uniform increase. The same idea led to the use of Mutual Information Neural Estimation (MINE) [23] to train DNNs with LL [90, 89]. Another approximation to the information bottleneck, the Hilbert-Schmidt independence criterion (HSIC) was used by [225] to learn a network in which each stage both maximizes the HSIC between its activations and the labels and minimizes it with the input. Further improvements have made this approach more biologically plausible [285].

Unsupervised Local learning

Self-supervised learning has made significant progress in recent years in the fields of LLMs [78] and Computer Vision (CV), where large amounts of unlabeled data can be exploited. In particular, methods based on contrastive learning have enabled major breakthroughs in CV. In this approach, representations are learned by contrasting positive (similar) and negative (dissimilar) examples. Notable algorithms in this area include Momentum Contrast [131], which uses a dynamic dictionary to store and contrast features, and SimCLR [59], which uses a data augmentation strategy to produce positive examples and a projector network before contrastive loss. Other popular methods have been proposed to avoid the need for negative examples, such as BYOL [118], which uses a slow-moving average of the network acting as the target network,

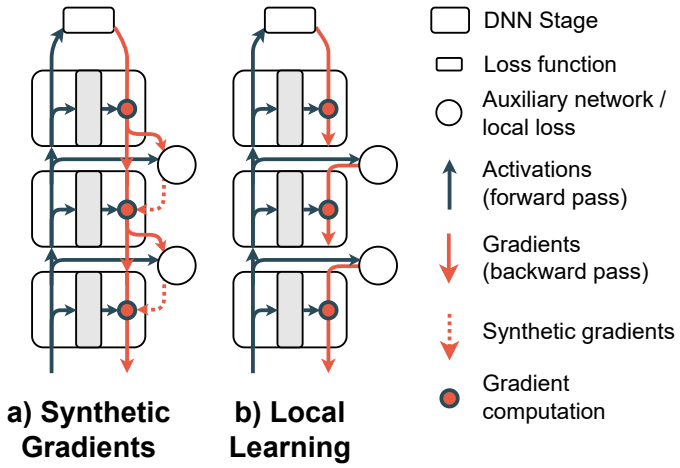


Figure 2.11: **Training with Synthetic Gradients and Local Learning approaches. (a)** Using synthetic gradients means using an auxiliary network that takes into input the activations and outputs a synthetic gradient that is used to backpropagate through the stage. The backward pass will provide the real gradient to update the auxiliary network. **(b)** Local learning approaches are diverse, but can generally be represented as a DNN with only feedforward connections between the stages. Each stage has an auxiliary network (or not) and a local loss, that will provide the gradients to update the stage.

or the Barlow Twins method [393], which trains positive example representations to have a cross-correlation matrix close to the identity. Note, however, that self-supervised contrastive learning can lead to dimensional collapse. In particular, [160] showed that the SimCLR loss leads to low-dimensional embeddings without proper projectors, which is addressed by their proposed architecture. More sophisticated data sampling strategies have been proposed to improve the SimCLR framework [148, 60], such as hard negative sampling, which prioritizes difficult negative examples [300, 342].

LL approaches have also been proposed with unsupervised training objectives. Greedy layerwise unsupervised learning [31] was first used as a way to efficiently pretrain the network, before using backpropagation to fine-tune the entire model. Similar greedy learning was used for fast training of deep belief networks [137]. This approach has been extended to image classification and visual recognition, by learning a sparse dictionary by cascading pooled sparse learned on image patches of a dataset [36]. Unsupervised LL can also be linked back to the Hebbian learning methods shown earlier. In particular, [149] showed that biologically plausible contrastive LL can scale to deep networks. Conversely, standard Hebbian learning rules can be seen as a form of LL, with appropriate losses [244].

Following the improvements of self-supervised approaches in CV and LLM, LL approaches using contrastive methods have shown promising results to improve on the previous unsupervised objectives. [223] proposed a method for CV based on mutual

information criteria, following the InfoNCE bound [270]. Each stage is trained to maximally preserve the information of its inputs (patches from the same image) using the contrastive InfoNCE loss. [384] extended on this idea with an intermediate method between local and global learning, where the stages are synchronized similarly to [111]. Following another self-supervised CV method, [318] trains a DNN on the large-scale ImageNet dataset via LL and the Barlow Twins loss. However, the DNN is split into only four stages, suggesting that achieving greater decoupling is challenging. [195] explored the use of LL for parallel training, showing the difference in gradient directions and features between local and global learning. They also showed that LL can be used to train LLMs. Similarly, [169] suggested using LL to finetune LLMs.

In this chapter, we have summarized the existing parallel approaches for training DNNs. We have also looked at backpropagation alternatives that can lead to faster and more biologically plausible learning. With this background, we now present the contributions of this thesis in the following chapters. First, we introduce our work regarding MP approaches related to local learning.

Part I

Local learning approaches for DL training

Chapter 3

Preventing Dimensional Collapse in Contrastive Local Learning with Subsampling

In this chapter, we present our first contribution to the field of local learning. We complement the knowledge of the informational collapse studied in supervised local learning, to the self-supervised contrastive case. We show that efficient subsampling of the examples can improve the performance of local learning by preventing a dimensional collapse.

This chapter led to a workshop paper at ICML 2023, at the Workshop on Localized Learning (LLW). We present in this chapter a version of this work with additional results. My contributions in this work start from the modification of the original idea, which was doing curriculum learning for local learning by using an oracle. I used local activation similarities in an inverse way, performed all the experiments and showed an dimensional collapse phenomenon and that our my method prevented it.

Contribution

Louis Fournier, Adeetya Patel, Michael Eickenberg, Edouard Oyallon, Eugene Belilovsky. Preventing Dimensional Collapse in Contrastive Local Learning with Subsampling. ICML 2023, Workshop on Localized Learning.

3.1 Introduction

Training a DNN via backpropagation is a computationally expensive process that requires sequential and synchronous processing of layers, with intermediate computations stored in memory [155]. A promising alternative to this approach is local learning,

which separates the objective functions for different subsets of the network and can update in parallel a subset of parameters associated with a particular task or input. In this paradigm, a Neural Network is divided into smaller stages, with each layer updated via gradient estimates that emulate End-to-End (E2E) dynamics. These gradients are typically generated from small auxiliary NNs [265], which allows efficient parallelization of computations and limits memory usage with limited overhead. Nevertheless, the quality of these gradient estimates is highly dependent on the auxiliary NNs, and in supervised settings, larger splits (i.e., smaller stages) often result in a more significant accuracy gap with End-to-End training [26].

This performance discrepancy is often attributed to information loss, where a local auxiliary network may greedily focus only on features relevant to its specific task, inadvertently allowing other potentially useful features for subsequent layers to dissipate and become inaccessible [365]. This phenomenon also seems to manifest in unsupervised settings: very deep NNs are usually divided into a limited number of components (e.g., only 4 in [223, 318]), which limits their parallelization potential. However, this issue has not been investigated until our work. Beyond the computational benefits, the integration of unsupervised and local learning is appealing because it paves the way for the development of more biologically plausible algorithms. However, the comparison is hampered by the length of the stages of the DNNs, given the biological implausibility of backpropagation within large stages.

Unsupervised learning [291], which takes advantage of large unlabeled datasets, has proven effective in generating representations for a variety of Machine Learning (ML) tasks. Recently rebranded as self-supervised learning [107, 82], these techniques train networks on proxy tasks derived from unlabeled data in a supervised manner. If these proxy tasks are chosen wisely, they allow the transfer of intermediate representations to tasks such as object recognition or object detection. The SimCLR framework [59], a leading contrastive learning method, facilitates End-to-End training of representations with competitive performance in object recognition, comparable to supervised learning. Owing to its simplicity and widespread use, we have chosen to adopt this framework for this study, approaching it through the lens of local learning.

Contributions. We consider the challenge of dividing a DNN trained via self-supervision into larger amount of stages, while maintaining competitive final performance. Our contributions are as follows:

1. We identify a dimensional collapse phenomenon caused by local self-supervised learning, which results in an undesirable information loss.

2. Motivated by an oracle subsampling method and a synthetic experiment, we propose a simple feature-similarity-based sampling method that prevents this collapse in local learning settings, allowing to reduce the information degradation from one local block to another.

3. Our experiments, conducted on the large-scale CIFAR-10 and STL-10 datasets, validate the effectiveness of our method.

4. We perform several ablation experiments on the CIFAR-10 dataset to emphasize the improvement achieved by our methods, especially over dimensional collapse.

Chapter organization. This chapter is structured as follows: first, we motivate and then discuss our subsampling strategy in Section 3.2. Next, Section 3.3.1 derives our numerical performance on image classification datasets, while Sections 3.3.2 and 3.3.3 consists of ablation experiments. Finally, we link this work to our current work on self-supervised local learning for LLMs.

Our code is available at: `https://github.com/fournierlouis/subsampled_local_simclr`.

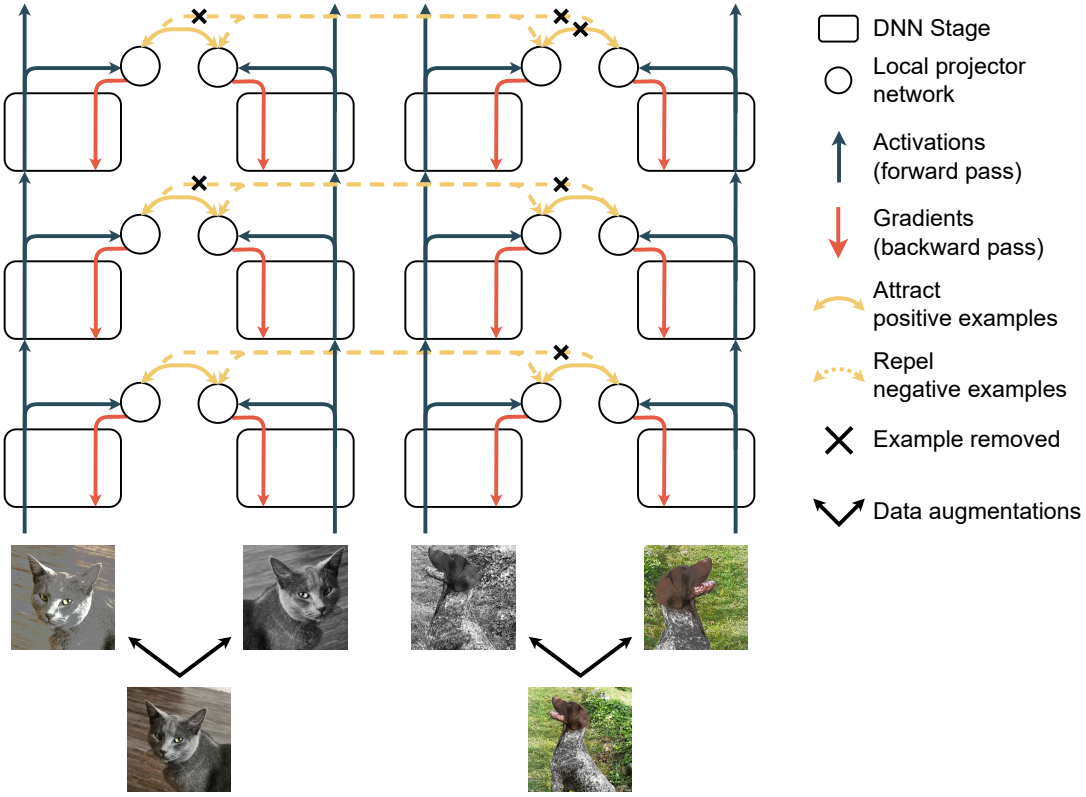


Figure 3.1: **Representation of our local contrastive training framework**. Following the SimCLR framework, two data augmented examples from the same image are fed to the DNN. In contrast to standard end-to-end learning, the DNN is divided into 3 sequential stages with only feed-forward connections. At each stage, a local projector network receives the output of the stage to compute the local contrastive self-supervised loss. The pair of examples generated from the same image is a positive example pair, and all other pairs are negative example pairs. Our subsampling method proposes to remove some examples from the loss computation depending on their alignment.

3.2 Method

3.2.1 Framework: Decoupled SimCLR

End-to-End SimCLR A typical SimCLR pipeline consists of a base encoder DNN f and a small projector head network g. A data augmentation procedure is applied to a mini-batch of size $\mathcal{B}$, generating $2\mathcal{B}$ augmented data examples. Each data augmented pair from the same example is considered a positive example, resulting in $\mathcal{B}$ pairs of positive examples. All other possible pairs are considered negative example pairs, resulting in $2\mathcal{B}(\mathcal{B}-1)$ pairs of negative examples. Each of the $2\mathcal{B}$ samples x_i is passed through the encoder to obtain a representation $h_i = f(x_i)$ and then projected, resulting in $z_i = g(h_i)$. Following [59], a similarity score $\text{sim}(x_i, x_l)$ (here the cosine similarity, which takes values between -1 and 1) is associated with each pair of samples. With τ as a temperature parameter, this results in the SimCLR loss function $\mathcal{L}$ defined by

$$\mathcal{L} = \frac{1}{2\mathcal{B}} \sum_i \mathcal{L}_i, \text{ where, } \mathcal{L}_i = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_{l_i^+})/\tau)}{\sum_{l=1}^{2\mathcal{B}} \mathbb{1}_{l \neq i} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_l)/\tau)}, \tag{3.1}$$

where l_i^+ is the index of the positive example associated with the example i and $\mathcal{L}_i$ is its associated loss.

Decoupling SimCLR Following the local learning framework of [26], we consider a decoupled DNN consisting of J stages. Each stage, a small network f^j, propagates features in a forward pass to the next stage f^{j+1}. However, stage j will not receive gradients from stage $j+1$. For an example x_i, we consider intermediate representations $x_i^j = f^j(x_i^{j-1})$. The parameters of each stage are updated using the SimCLR framework with intermediate representations x_i^j. To achieve this, we consider J small projector head networks g^j and intermediate projections $z_i^j = g^j(x_i^j)$. To unlock the stage [155], each pair f^j, g^j is updated locally by backpropagation with the following local loss, similar to Eq. (3.1):

$$\mathcal{L}^j = \frac{1}{2\mathcal{B}} \sum_i \mathcal{L}_i^j, \text{ where, } \mathcal{L}_i^j = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i^j, \boldsymbol{z}_{l_i^+}^j)/\tau)}{\sum_{l=1}^{2\mathcal{B}} \mathbb{1}_{l \neq i} \exp(\text{sim}(\boldsymbol{z}_i^j, \boldsymbol{z}_l^j)/\tau)}, \tag{3.2}$$

The stage-by-stage training procedure, as it stands, is unable to guide the initial stages to preserve information that could be crucial for the subsequent stages. To address this shortcoming, we first propose the implementation of a curriculum learning strategy [127], that relies on the 'difficulty' level of a sample. We hypothesize that exposing a decoupled DNN to easier samples can potentially improve its convergence behavior, by stabilizing features in the intermediate levels and avoiding the greedy effect of local learning.

Table 3.1: **Linear evaluation test accuracy with and without using oracle-based subsampling** on CIFAR-10 for a DNN trained End-to-End (E2E) and decoupled in $J = 16$ stages. The accuracy decreases for an E2E model trained with subsampling, while it substantially increases for $J = 16$.

Nb of stages J	SimCLR	+ Oracle
1 (E2E)	**92.4**	91.7
16	85.6	**89.1**

3.2.2 A motivating example: improving local learning with an oracle

A natural way to incorporate knowledge about the difficulty of examples can be to use an oracle network, trained End-to-End as a way to measure the similarity between examples. Easy examples for the oracle network have high similarity for positive examples and low similarity for negative examples. We can then use this knowledge of difficulty to modulate the training objective of the decoupled network.

Method More specifically, consider a network $\tilde{f}$ (a ResNet-50 in our case), trained on the SimCLR loss without decoupling. The final representation learned by the network for an example x_i is $\tilde{z}_i = \tilde{f}(x_i)$. We propose a natural way to improve the learning objective defined in Eq. (3.2), by considering only examples that could be considered 'easy' for the oracle network. Thus, the training goal of the decoupled network will be easier, since it will only focus on a subsampled set of examples, possibly reducing the information loss observed in local learning. We propose two binary values $\beta^+_{\{i,l_i^+\}}$ and $\beta^-_{\{i,l\}}$ that indicate whether a positive or negative pair of examples can be considered easy for the oracle network: if their final representations are close enough for a positive pair, or far enough for a negative pair. We define two thresholds T^+, T^-, and then consider the similarity of a pair of data examples with the cosine similarity of their representations, which does not depend on depth j

$$\beta^+_{\{i,l_i^+\}} \triangleq \mathbb{1}_{\text{sim}(\tilde{z}_i,\tilde{z}_{l_i^+})\geq T^+}, \quad \beta^-_{\{i,l\}} \triangleq \mathbb{1}_{\text{sim}(\tilde{z}_i,\tilde{z}_l)\leq T^-}. \tag{3.3}$$

Thus, for each positive sample pair $\{i, l_i^+\}$ and stage f^j, we obtain the following loss function, which depends on the oracle similarity of the pair of data examples:

$$\mathcal{L}_i^j = -\beta^+_{\{i,l_i^+\}} \log \frac{\exp(\text{sim}(\boldsymbol{z}_i^j, \boldsymbol{z}_{l_i^+}^j)/\tau)}{\sum_{l=1}^{2N} \mathbb{1}_{l\neq i}\beta^-_{\{i,l\}} \exp(\text{sim}(\boldsymbol{z}_i^j, \boldsymbol{z}_l^j)/\tau)}. \tag{3.4}$$

Results We report in Table 3.1 the accuracy results of this method on CIFAR-10 for the thresholds $T^+ = 0.7$ and $T^- = 0.3$, for $J = 1$ (End-to-End) or 16. We observe that this method decreases the accuracy for an End-to-End trained network. However, we see a stark improvement in the accuracy of a decoupled network, reducing the gap with the

End-to-End network from 6.8% down to 3.3%. This indicates that subsampling methods can serve as powerful tools to scale up local learning to larger splits. We observe that selecting negative examples that are considered easy for the oracle can be viewed as a form of false negative removal [148, 60], which we discuss in the Appendix.

This method is effective for improving decoupled networks, but suffers from a major flaw. The knowledge of an oracle network is incompatible with the goal of our decoupled network, which requires only local information to be more biologically plausible. Still, the information provided by the oracle is very weak. A simple proxy could be to replace the oracle similarity $\text{sim}(\tilde{z}_i, \tilde{z}_l)$ by the stagewise representation similarity $\text{sim}(z_i^j, z_l^j)$. However, directly swapping the training goal leads to poor results, especially when considering negative examples. On the contrary, we argue that in this case it is more appropriate to consider negative examples with high stagewise representation similarity, making the procedure similar to hard negative mining [300]. We motivate this idea with a simple example showing that dimensional collapse can be remedied by subsampling examples.

3.2.3 Preventing dimensional collapse in a linear model with subsampling

Linear model framework We consider the framework presented by [160], which simplifies the contrastive framework to a linear model W. This allows us to derive several theoretical insights despite the limited setting. We consider a dataset composed of two views of $\mathcal{B}$ data points $(x_i)_i$ and $(x'_i)_i$ of dimension D, and x the concatenated dataset vector of size $2\mathcal{B} \times D$. We write $z = Wx$ the corresponding representation vector. Then, the dynamics of the weight matrix under gradient descent following the contrastive InfoNCE loss (which is equivalent to the SimCLR loss with unit temperature when z is normalized) is, with similarity terms $s_{il} = \frac{1}{Z_i} e^{-\frac{1}{2}|z_i - z_l|^2}$, $s_{ii} = \frac{1}{Z_i} e^{-\frac{1}{2}|z_i - z'_i|^2}$, and $Z_i = \sum_{l \neq i} e^{-\frac{1}{2}|z_i - z_l|^2} + e^{-\frac{1}{2}|z_i - z'_i|^2}$:

$$\frac{dW}{dt} = \underbrace{\sum_{i,l} s_{il}(z_i - z_l)(x_i - x_l)^T}_{\text{Data distribution term}} - \underbrace{\sum_i (1 - s_{ii})(z'_i - z_i)(x'_i - x_i)^T}_{\text{Data augmentation distribution term}} \tag{3.5}$$

For fixed values of $(s_{il})_{il}$, [160] finds that strong data augmentation leads in collapsed dimensions in W due to negative eigenvalues in $\frac{dW}{dt}$ caused by the data augmentation distribution term. Indeed, at convergence, $\frac{dW}{dt} = 0$, such that the data distribution term equalizes the data augmentation term. For the sake of simplicity, assume here that the similarities $(s_{il})_{il}$ are either equal to 0 or 1. In this case, since two data augmented samples should lead to the same representation, one has that $(s_{ii})_i = 1$, and we have directly that both the data distribution and the data augmentation distribution terms must vanish. Following [160], if data augmentation is applied only to certain features $[d+1 : D]$, then the highest rank matrix W that allows convergence is proportional to $\begin{pmatrix} I_d & 0 \\ 0 & 0 \end{pmatrix}$,

and the augmented feature dimensions need to be collapsed to allow convergence.

Subsampling to prevent collapse Now, consider that we restrict the sums in both terms to consider only positive and negative examples with high similarity. Then, at equilibrium, we would get:

$$\frac{dW}{dt} = \sum_{i,l,z_i \approx z_l} s_{il}(z_i - z_l)(x_i - x_l)^T - \sum_{i,z_i \approx z_i'} (1 - s_{ii})(z_i' - z_i)(x_i' - x_i)^T = 0. \tag{3.6}$$

This is less restrictive, since e.g. $W = I_D$ is an equilibrium point without the need for collapsed dimensions, while preserving the contrastive nature of the loss. This restriction of the gradient allows much more flexibility in the representation space, where $s_{il} > 0$ and $s_{ii} < 1$. This extreme case showcases that removing gradient terms for low representation similarity removes the need for collapsing dimensions to converge. This motivates us to adapt the previous subsampling strategy using oracle knowledge to a stagewise similarity strategy.

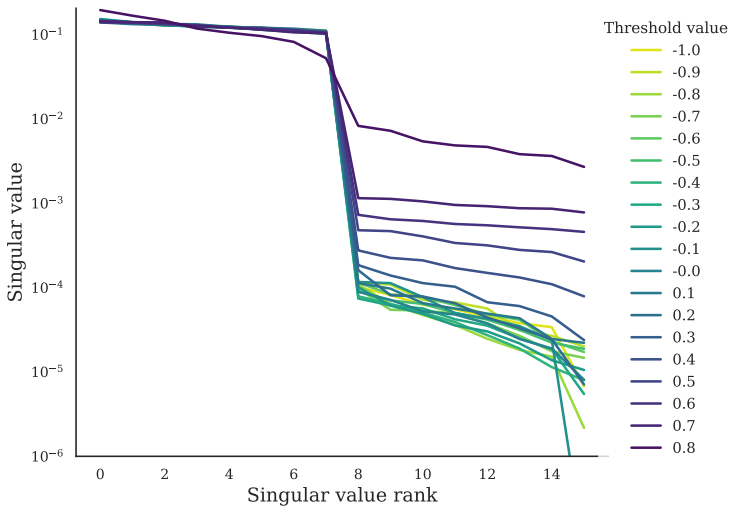


Figure 3.2: **Normalized singular value spectrum** of the representations z when learning with the SimCLR loss and our subsampling strategy Eq. (3.8) on the toy example. Without subsampling (yellow line), the strongly augmented features produce collapsed dimensions at convergence, in contrast to the subsampling case (blue and purple lines).

3.2.4 Subsampled Decoupled SimCLR

Method We now adapt the oracle subsampling method to better suit our needs. Instead of relying on the static similarity of example pairs suggested by the oracle, we exploit the dynamic nature of the similarities of intermediate stage representations. By selectively

considering examples with high similarity, we aim to preempt the issue of dimensional collapse. Following Eq. (3.3), we set a lower threshold T to restrict the examples we consider to those with sufficiently high cosine similarity. To do this, we introduce

$$\alpha^j_{\{i,l\}} \triangleq \mathbb{1}_{\text{sim}(\boldsymbol{z}^j_i, \boldsymbol{z}^j_l) \geq T}. \tag{3.7}$$

Thus, we obtain the following loss function for each positive pair $\{i, l\}$ and stage f^j:

$$\mathcal{L}^j_i = -\alpha^j_{\{i,l^+_i\}} \log \frac{\exp(\text{sim}(\boldsymbol{z}^j_i, \boldsymbol{z}^j_{l^+_i})/\tau)}{\sum_{l=1}^{2\mathcal{B}} \mathbb{1}_{l \neq i} \alpha^j_{\{i,l\}} \exp(\text{sim}(\boldsymbol{z}^j_i, \boldsymbol{z}^j_l)/\tau)}. \tag{3.8}$$

Toy example To understand the refinement of our method, we propose to study it using the toy model of [160]. In this framework, we sample data following the standard normal distribution with dimension $D = 16$, from which two augmented views are obtained by adding standard normal noise multiplied by an amplitude σ on the last 8 dimensions. We train our method according to Eq. (3.8) with gradient descent. Thus, the only difference from the gradient dynamics in Eq. (3.5) is our use of normalized representations. We report in Figure 3.2 the singular values of the covariance matrix of the representations z with a strong augmentation $\sigma = 1.5$. Training with standard SimCLR ($T = -1$) produces collapsed dimensions. However, by increasing the threshold T, we observe a drastic reduction in the collapse of the augmentation dimensions, as predicted earlier.

Table 3.2: **Linear evaluation test accuracy** results on CIFAR-10 and STL-10, after training on both SimCLR and our subsampling method. $T = 0$ for CIFAR-10 and $T = -0.4$ for STL-10. Test accuracies for CIFAR-10 are given with the mean and standard deviation over 5 trainings. We also report the memory cost of local self-supervised learning for varying the number of stages J, as well as the maximum computation required by each stage in multiply-accumulates (MACs).

Datasets		**CIFAR-10**			**STL-10**		
J	**Method**	**Accuracy**	**MACs**	**Memory**	**Accuracy**	**MACs**	**Memory**
1 (E2E)	SimCLR	**92.1** ± 0.2	1.31 G	13.5 GB	**87.6**	17.1 G	16.7 GB
	+ ours	91.4 ± 0.4			86.7		
4	SimCLR	**90.0** ± 0.1	349 M	8.7 GB	**84.3**	4.65 G	9.7 GB
	+ ours	89.9 ± 0.3			82.6		
8	SimCLR	87.5 ± 0.4	196 M	6.3 GB	**80.8**	2.63 G	7.0 GB
	+ ours	**88.4** ± 0.4			79.7		
16	SimCLR	85.9 ± 0.3	**123 M**	**5.6 GB**	77.8	**1.69 G**	**6.1 GB**
	+ ours	**87.1** ± 0.4			**78.8**		

3.3 Numerical experiments

Architecture We consider a ResNet-50 [130] base encoder, which we divide into 4, 8 or 16 decoupled stages. For $J = 4, 8$ the split is adjusted to maintain an equal number of bottlenecks in each stage. For $J = 16$, the DNN is split after the bottleneck of each residual block. The precise split is indicated in the appendix. Each local projector g^j is composed of: a 3×3 convolutional layer (with stride 2 and the same number of channels as the representation), a batch normalization and ReLU layer, a global average pooling, then a fully connected layer, a ReLU, and a final fully connected layer with output dimension 128.

Hyper-parameters and datasets We train the networks on two image classification datasets: CIFAR-10 [184], STL-10 [66] respectively, which consist, respectively, of 5×10^4 and 10^5 (for the unlabeled split) images of size 32×32 and 96×96. Each stage is trained using either the training objective Eq. (3.8) or Eq. (3.2) for the sake of comparison. Following standard practice [50], for CIFAR-10 and STL-10, our model is trained with the Adam optimizer [172] with a learning rate of 10^{-3} and a weight decay of 10^{-6} for 1000 epochs, with a mini-batch size of 256. The temperature τ is kept at the default value of 0.5. We choose the threshold T according to the training accuracy in the linear evaluation: $T = 0$ for CIFAR and $T = -0.4$ for STL-10. The training was done on A100 GPUs and took 12 hours for a CIFAR-10 run on 1 GPU and 4 hours on 16 GPUs for STL10.

3.3.1 Accuracy on image recognition tasks

Evaluation on image recognition task Finally, we do a linear evaluation of the model trained. The network is frozen, and the fixed representations are used to train a linear classifier on the frozen representations at the end of the model on the supervised image classification task. The linear layer is trained for 200 epochs with a batch size of 256 with the Adam optimizer with a learning rate of 10^{-3} and weight decay of 10^{-6} with the cross-entropy loss. We use the STL-10 5k labeled training samples for training, similarly to [223].

CIFAR-10 and STL-10 results Table 3.2 reports test accuracies and standard deviations on standard image datasets, comparing our method with the standard SimCLR loss, for splits in $J = 1, 4, 8$ and 16 stages. As expected, and as previously observed for supervised settings, as the number of splits increases, the test accuracy degrades, even in the unsupervised setting. The introduction of the data sampling technique of Eq. (3.8) does not significantly affect our model when splitting in $J = 1$ or $J = 4$ stages, which is logical, as there is a limited information loss (see Sec. 3.3.2). The accuracy on STL-10 improves meaningfully only for $J = 16$. In the case of CIFAR-10, our sampling strategy allows to improve the accuracy of $J = 16$ to that of $J = 8$, which is a significant improvement. We also show the high parallelization potential of our method, by providing the memory

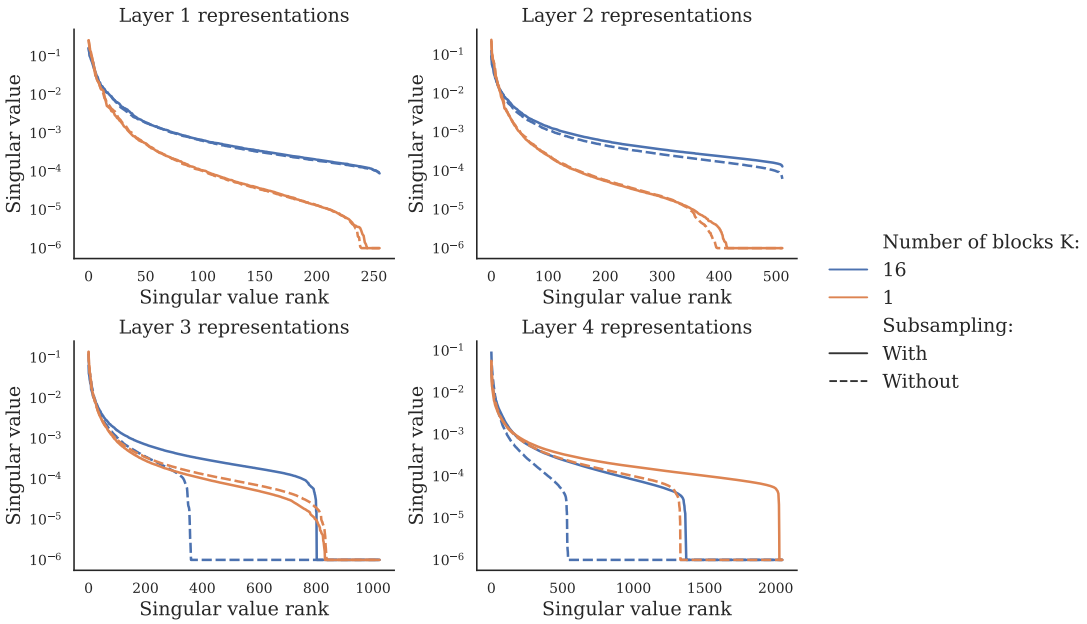
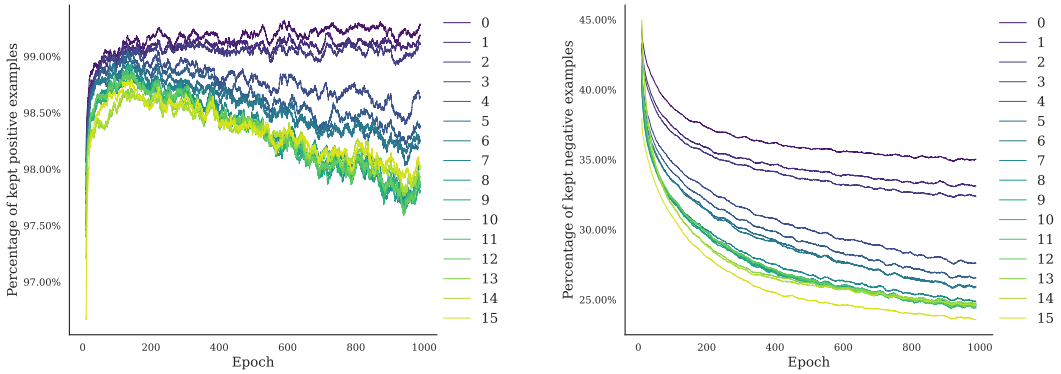


Figure 3.3: **Normalized spectrum of the intermediate representations** $(x_i^j)_i$ after each of the 4 ResNet-50 layers, computed from the covariance matrix of the representations of the entire CIFAR-10 dataset. Four models are presented, trained End-to-End ($K = 1$) or decoupled ($K = 16$) on CIFAR-10 with the SimCLR loss with or without our subsampling method. With the classical SimCLR loss, the spectra show a fast decay in later layers, indicating a dimensionality collapse. Adding our sampling technique systematically leads to a much higher feature dimensionality, even for End-to-End.

cost of our method as implemented, as well as the maximum stage MACs (computation time required by the slowest stage among the J), which is a lower bound MACs value with maximum model parallelism. In particular, our method allows us to achieve an almost threefold reduction in the memory usage during training without any specific code optimizations. The memory cost is computed using the maximum memory allocated by CUDA, with the cuDNN benchmark disabled for consistency, on a single A100 GPU with a batch size of 256 for CIFAR-10 and 128 for STL-10.

3.3.2 Ablation 1: Analysis of the internal representations.

Dimensional collapse in Contrastive Local Learning We now investigate the extent of dimensional collapse in the representations generated by different methods. To measure dimensional collapse, we consider the intermediate representations $(x_i^j)_i$ (after mean pooling and flattening) on CIFAR-10 after training. We then compute the representation covariance and its singular value decomposition to assess the intrinsic dimensionality of the feature space. This serves as a linear proxy for assessing information content. If such a linear assessment can be shown to vary between methods, this is a strong indication



(a) Ratio of positive examples kept.

(b) Ratio of negative examples kept.

Figure 3.4: **Ratio of positive and negative examples kept** through training for each stage. The model is trained with $J = 16, T = 0$ on CIFAR-10. Surprisingly, most positive examples are retained, indicating few outliers. The proportion of negative examples retained is much lower and decreases during training and with depth. Our method can be seen as a form of curriculum learning by depth.

of different overall information capacities. Figure 3.3 shows the singular values of the covariance obtained from the CIFAR-10 dataset representations of a ResNet-50 trained in different settings. Without the incorporation of our subsampling method, the internal representation of both an End-to-End and a decoupled model suffer from progressive dimensionality collapse. This is consistent with the results of [365], who showed that their measures of mutual information between the representations and both the labels and the input decreased with depth.

However, we argue that our method prevents this dimensionality loss. In the same Figure 3.3, we also present the singular values for two models trained with our method, for $J = 1$ and 16. There are no differences in the first ResNet layers. However, there is a significant increase in dimensionality in later layers, both in the decoupled network and in the End-to-End network. This indicates that our sampling technique affects both models. Notably, the dimensionality of the last layer of the decoupled model matches that of the End-to-End SimCLR-trained model.

In the End-to-End network, there is no loss of information due to decoupling, and thus the increase in dimensionality due to sampling does not improve accuracy. We believe that the dimensionality increase due to sampling improves accuracy only in the decoupled case, since it mitigates the information loss due to decoupling, and we study this effect using a linear probe experiment.

Linear probes To further investigate the effect of our subsampling, especially with respect to depth, we compute the linear separability of the intermediate representations x^j with linear probes. This is achieved by computing the linear evaluation as before on the intermediate representations (after global average pooling) and not only on the final

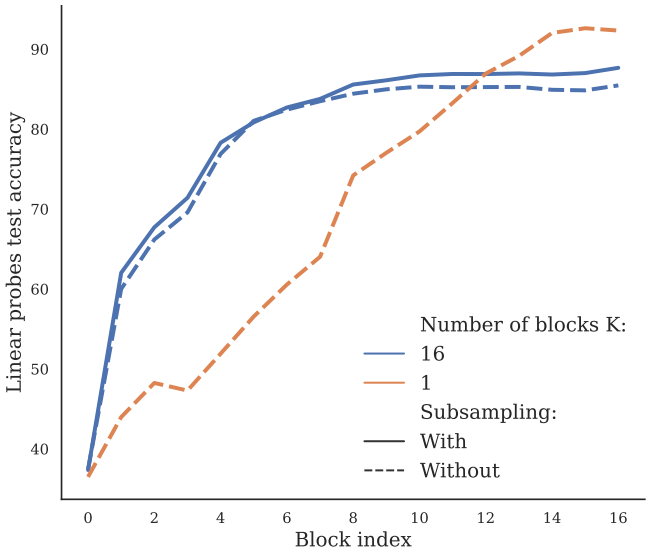


Figure 3.5: **Linear separability** on CIFAR-10 of intermediate representations h^k at different depths, estimated with linear probes. A model trained End-to-End will display a slowly increasing test accuracy with depth, in contrast to a decoupled model which has much higher accuracy at low depth before plateauing. Our method shows a consistent improvement over the original baseline.

one. We report in Figure 3.5 the accuracy of the linear probes after training for three models: a model trained End-to-End with SimCLR, and a decoupled model ($J = 16$) trained with and without subsampling.

We observe similar linear probe accuracy to that obtained in local supervised learning, with very progressive accuracy for the model trained End-to-End, and high accuracy in the early stages for the decoupled network before plateauing. However, if the greedy nature of the local supervised learning explained this phenomenon, it is less clear why local self-supervised learning shows similar linear probe accuracies. The model trained with subsampling shows a slightly increasing improvement with depth compared to the one without subsampling, and similar curves. This confirms that our method improves intermediate representations. However, we do not get a model closer to an End-to-End one, as we would observe a decrease in the 'greedy' phenomenon, i.e., a decrease in the accuracy of the early layers.

3.3.3 Ablation 2: Understanding the subsampling strategy

Impact of the sampling during training Having confirmed the improvement due to thresholding, we now report how sampling is affected during training. We report in Figure 3.4 the ratio of positive and negative examples kept for the loss computation at each stage during training, for a model trained on CIFAR-10 with $J = 16$ and $T = 0$. Surprisingly, almost all positive examples are kept during training, despite the

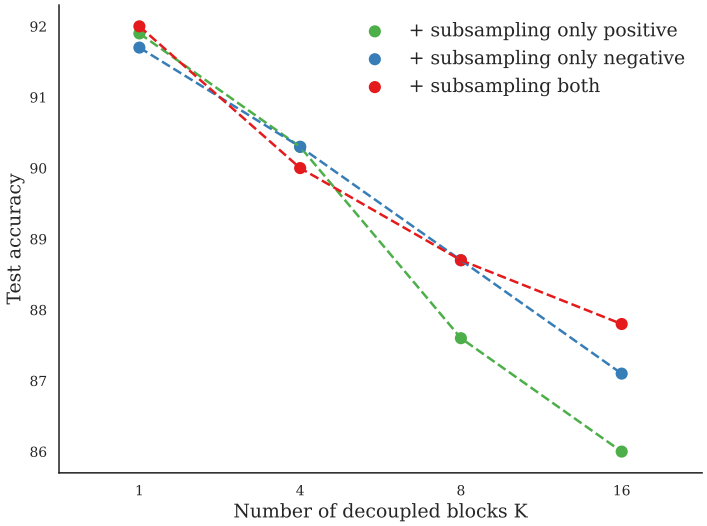


Figure 3.6: **Test accuracy** on CIFAR-10 depending on the number of decoupled stages J after training on SimCLR with our subsampling method, by subsampling either positive or negative examples or both. Subsampling negative examples gives the biggest improvement, but both are needed to get the best accuracy.

improvement provided by thresholding positive examples. In comparison, a much smaller ratio of negative examples is retained during training, with as few as a quarter of examples kept in the final stage at convergence. This is in contrast to the common belief in contrastive learning that a large number of negative examples are needed.

The number of negative examples retained decreases drastically during training. This is not surprising, since negative examples are expected to have a negative alignment when the model converges. Similarly, in both cases, the number of examples kept decreases with depth. With a difference of almost 10% between the first and the last stage, there are fewer negative examples to contrast with the positive examples from in the last stages. Thus, we can consider our model as a form of curriculum learning by depth, with

Impact of the positive and negative examples To ensure that removing both positive and negative examples improves accuracy, we propose to train our method by subsampling only positive or negative examples. More precisely, in Eq. (3.8) we add only the term $\alpha^j_{\{i,l_i^+\}}$ to consider thresholding positive examples, or the denominator term $\alpha^j_{\{i,l\}}$ to consider thresholding negative examples. We report in Figure 3.6 the accuracy of our method for varying J by subsampling either positive or negative examples or both. We report improvement in both cases. However, thresholding all examples provides the best accuracy as J increases.

Impact of the threshold value T We also varied the accuracy of our model depending on the value of the threshold T to study its effect. The accuracy follows a simple piecewise linear curve in three parts. Between $T = -1$ (which corresponds to the standard SimCLR loss) and -0.2, the accuracy is stable because few examples are removed. From $T = -0.2$ to 0, we observe an increase in accuracy. Higher thresholds quickly decrease the accuracy. Thus, there is a trade-off between increasing dimensionality by subsampling and removing too many examples, which is detrimental to the training. More details are provided in the Appendix.

3.4 Conclusion

This work has investigated the training of DNNs with self-supervised local learning methods, a more biologically plausible and parallelizable alternative to backpropagation. We find that a dimensional collapse partially causes the drop in accuracy known in local learning with larger splits. By studying a linear model and using an oracle model, we motivate a simple local feature similarity sampling method that improves on the original SimCLR loss. We confirm that this method remedies the dimensional collapse for contrastive learning and reduces the accuracy loss due to decoupling for models with large splits.

Limitations Despite increasing dimensionality, decoupling still causes a significant accuracy gap, suggesting that there are other issues caused by local learning. Generalizing our observations to non-contrastive self-supervised objectives may also be another interesting direction for future work.

Future work: Training LLMs with self-supervised local learning

We have observed in this work that with minimal modifications to the training algorithm, self-supervised local learning can achieve performance close to End-to-End learning with a limited information loss. Similarly, self-supervised objectives are particularly efficient for training with large amounts of unlabeled data. Since one of the reasons for the information loss in local learning is overfitting, where early stages may overfit the data to spurious features, training on a very large amount of data may be a promising direction. For this reason, we propose, in a future work, to adapt the local learning framework to the training of LLMs, which uses self-supervised learning on vast amount of data. These very deep NNs require a lot of parallelization for their training, and making their training local may be the key to scaling them further [195].

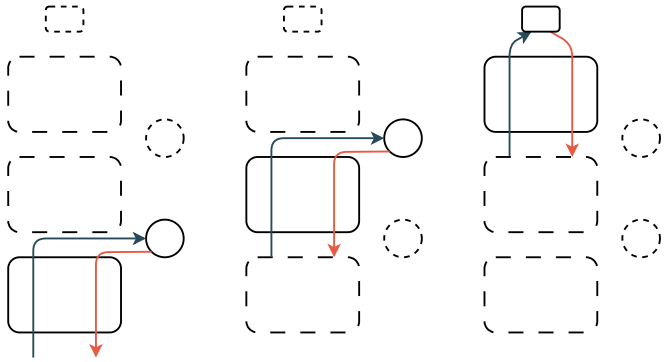


Figure 3.7: **Representation of local learning for minimum memory overhead.** This learning approach is particularly interesting for training large models like LLM. The dotted lines represent a unloaded stage, with no memory used for activations or parameters. A stage is only loaded when the activations reach it for computation. Thus, only one stage at a time is loaded in memory.

We propose a local training framework for LLMs that can focus on either improving memory overhead or parallelization. Indeed, one can train LLMs with local losses, as in Decoupled Greedy Learning [26], by starting the computation of the next stage's forward pass as soon as the current stage's activations are computed. In this framework, two stages compute in parallel at the same time, one starting its forward pass while the other is backpropagating. Another approach is to focus less on the parallelization capacities of local losses and more on the memory overhead of LLM. By computing the backward pass on the current stage and unloading the parameters of that stage before passing the activations to the next stage, only a single stage remains in memory for training at any given time. This idea is particularly interesting for LLMs, since their memory overhead is a crucial limitation of their training. We show the training of a network in this way in Figure 3.7. We also find that the Transformers architecture is particularly well suited for local learning approaches, since the size of activations within the network does not vary. Thus, the local losses do not require different architectures depending on their depth.

Chapter 4

Can Forward Gradient Match Backpropagation?

In this chapter, we present our second contribution to the field of local learning. After having studied how we can regularize effectively local learning by subsampling examples from the local computations, we now study a intermediate learning framework, that allows end-to-end feedback while being only update locked. More specifically, we supplement the Forward Gradient framework with local learning. We show that, despite their bias, the gradients provided by the local losses can be used as efficient candidate tangent directions for forward-mode AD.

This chapter led to publication at the ICML 2023 conference (* indicates equal contribution). This work was done in collaboration with Stéphane Rivaud, with equal contribution from both. My part of the experiments was more focused on implementing the basic forward gradient module, the computations in the activation space and the metrics logging.

Contribution

Louis Fournier*, Stéphane Rivaud*, Eugene Belilovsky, Michael Eickenberg and Edouard Oyallon. Can Forward Gradient Match Backpropagation?. ICML 2023.

4.1 Introduction

Stochastic Gradient Descent (SGD) [7, 38] using end-to-end backpropagation for gradient computation is the ubiquitous training method of state-of-the-art DNNs. However, there are at least two problems with this gradient estimation method. First, it requires a significant amount of computational resources and memory. In fact, a backpropagation update is performed in two steps: a forward step, which computes activations,

and a backward step, which computes gradients. Until a gradient is computed at a given layer, intermediate computations leading to that layer must be stored in memory, and the computation of the update is blocked. This is referred to as the *backward lock* [155], which is also an obstacle to the development of model-parallel asynchronous methods[26]. Backpropagation also lacks biological plausibility, which is often seen as a direction towards more scalable learning [296]. Indeed, the backpropagation algorithm suffers from the weight transport problem [47], which means that the weights of the above layers must be shared during the backward pass of a given layer. This requires a significant amount of communication and synchronization, which has not been observed to be implementable in biological systems [215]. Since biological systems have achieved learning without these constraints, researchers hope to isolate such principles and use them to improve the learning of artificial neural networks.

Recent contributions have re-examined the converse procedure of computing *Forward Gradients* [21], based on the classical forward mode automatic differentiation [20]. It is an alternative to the standard backpropagation algorithm, allowing the computation of a directional derivative (i.e., a scalar product with the gradient, as opposed to the gradient vector itself) from a forward pass. The gradient computations in forward mode explicitly use the Jacobian of a given layer through Jacobian-Vector Products but do not require the storage of intermediate activations or backward passes. In other words, it is *backward unlocked* (as described by Jaderberg et al. [155]) in that the computation of the derivative is finished as soon as the forward pass is finished. This leads to a potential memory reduction and does not use the biologically implausible weight transport. A Forward Gradient, as introduced by Baydin et al. [21], corresponds to an unbiased estimate of an activation or weight gradient (which we will refer to as a Gradient Target) computed via a random, isotropic, directional derivative, i.e., a projection of the Gradient Target onto a tangent vector (which we will define as the Gradient Guess). The motivation for Forward Gradient descent is to approximate End-to-End Gradient descent. However, this unbiased gradient approximation generally suffers from high variance.

Several approaches [296, 319] have been proposed to reduce the corresponding variance. Unfortunately, there is often an accuracy gap [319] that is difficult to reduce because the Gradient Guess is not aligned with the End-to-End Gradient Target. Following a different line of thought, Ren et al. [296] proposes to approximate the gradients computed from Local Loss functions [280, 265, 25] as Gradient Targets. This is combined with a collection of architectural changes that allow the decomposition of losses into subgroups within a layer [279]. This reduces the dimensionality and thus the variance of Forward Gradients. However, these changes result in architectures that are not widely applicable. Furthermore, the results are still far from competitive with standard backpropagation.

Following Ren et al. [296], it is also unclear whether using local gradients as Gradient Targets combined with a Gaussian Gradient Guess is the optimal strategy. In this work, we propose to investigate different combinations of Gradient Targets and Gradient Guesses in a way that covers existing work and also explores other possible combinations. In particular, we investigate whether local gradients, which alone lead to suboptimal

performance, can be used to compute a reliable Forward Gradient estimate of the end-to-end gradient. Our goal is to better understand whether the forward mode of automatic differentiation can lead to the training of competitive models while maintaining the above benefits.

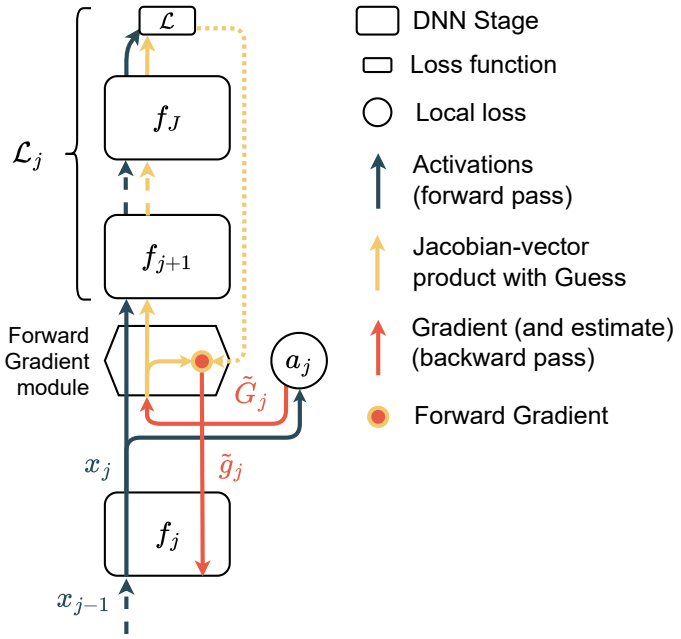


Figure 4.1: **Schematic summarizing the best use-case of our algorithm**, which approximates a Global (End-to-End) Target gradient at a stage j with forward mode automatic differentiation. A Gradient Guess $\tilde{G}_j$ is provided by the gradient of the local loss a_j (red). The Gradient Target is projected onto the Gradient Guess by Jacobian-vector products (blue), and we use this estimate to compute the update for the stage j. In Ren et al. [296], the Guess $\tilde{G}_j$ is a random vector and a_j is the Target loss. More details can be found Section 4.2

Contributions. Our contributions are as follows:

1. We make explicit the idea that any Gradient Target can be projected onto any Gradient Guess, and propose to study a wide variety of these combinations.
2. In particular, we propose to use local gradients from auxiliary losses as powerful Gradient Guess directions for the computation of directional derivatives in Forward Gradient mode. We show that this proposal can yield far improved results compared to naive random directions while maintaining the benefits of the Forward Gradient mode.
3. To ablate this method, we cover many combinations of possible Gradient Targets and Guesses for a well-established architecture (ResNet-18) applied to standard

image classification, using both gradient insertion points, commonly referred to as *activity perturbation* and *weight perturbation*, which indicate whether the directional derivative is taken in activation space or weight space.

4. Our evaluations show that in the case of Gradient Guesses obtained from supervised Local Losses, a consistent positive alignment between the Gradient Targets and Guesses improves the Forward Gradient estimation and reduces the gap with end-to-end training.

Chapter organization. This chapter is structured as follows: Section 4.2 describes Gradient Target approximations with Forward Gradient, using both Random and Local Gradient Guesses. Then, Section 4.3.1 describes all the details of our experimental setup. In Section 4.3.2, we show that Forward Gradient can lead to competitive accuracy with end-to-end training, and in Section 4.3.3 that this is due to having a better estimate of the Global Target than with Random Guesses. However, we show in Section 4.3.4 that the Local Gradient Guess is still a poor estimate due to different training dynamics between Local and Global losses. We confirm the consistency of our findings across model sizes and other datasets in Section 4.3.5. Finally, we link this work to another contribution on model parallelism that does not suffer from the gradient alignment discussed but from delayed gradients. Our source code is available at: github.com/streethagore/ForwardLocalGradient.

4.2 Gradient approximation via Forward Gradient

The objective of this work is to study computationally efficient and accurate estimates $g(\theta) \in \mathbb{R}^d$ of a **Gradient Target** $\nabla f(\theta) \in \mathbb{R}^d$ of some objective function $f : \mathbb{R}^d \to \mathbb{R}$, using Forward Gradients. The main idea is to use a **Gradient Guess** vector $G(\theta) \in \mathbb{R}^d$ onto which we project the Gradient Target to obtain an approximation:

$$g(\theta) = \langle \nabla f(\theta), G(\theta) \rangle \, G(\theta) \approx \nabla f(\theta). \tag{4.1}$$

Once $G(\theta)$ is provided, $g(\theta)$ can be computed efficiently because it can be implemented as a Forward Gradient. This induces a limited computational overhead compared to a standard forward pass since evaluating the directional derivative costs about as much as a forward pass (especially for nonlinearities whose derivatives cost as much as the function itself). The choice of G thus has a large impact on the quality of the approximation. Note that while Gradient Descent aims at finding $\theta^* \in \mathbb{R}^d$ such that $\nabla f(\theta^*) = 0$, the stationary points of Eq. (4.1) are given by:

$$G(\theta^*) = 0 \text{ or } \langle \nabla f(\theta^*), G(\theta^*) \rangle = 0 \, .$$

4.2.1 Gradient Guesses

Gradient Guesses come in different flavors of randomness. Hence the term 'approximation' used to describe Eq. (4.1) can take on both probabilistic and deterministic forms.

We discuss guesses that are purely random, purely deterministic, and one intermediate case.

Random Guesses. By drawing G from a zero-mean probability distribution with unit covariance, we can generate an unbiased estimator of the gradient. We propose to use either Rademacher variables $\mathbb{P}(G_i = 1) = \mathbb{P}(G_i = -1) = \frac{1}{2}$ as proposed by Belouze [27], or isotropic Gaussians $G \sim \mathcal{N}(0, \mathbf{I})$, as proposed in Ren et al. [296]. In both cases, g is an unbiased estimate of ∇f, since we have $\mathbb{E}[GG^T] = \mathbf{I}$. However, such estimates have potentially high variance, leading to large errors in individual gradient estimates and slow optimization progress.

Local Guesses. To improve the quality of this estimate, we consider deterministic gradient guesses. We obtain these guesses by computing local update signals from local losses, each associated with an auxiliary network as in Belilovsky et al. [26] and Nøkland and Eidnes [265]. This gives access to a gradient ∇a from a small local NN a with minimal computational effort. Several choices for a are possible, e.g., a CNN, an MLP, or a linear layer. The intuition behind using local gradients is that they should provide a better one-shot approximation of ∇f than uncorrelated noise. To obtain the best linear approximation of ∇f from ∇a, we then use $G = \frac{\nabla a}{\|\nabla a\|}$ as the projection direction.

NTK Guesses. Conceptually, comparing random and deterministic projection directions via local auxiliary losses is not straightforward. We include a transitional setting using random auxiliary networks to test whether the inductive bias of the random network is sufficient for improvement or whether training of the auxiliary is required. To do this, we reinitialize the Local auxiliary network a introduced above at each iteration of the algorithm. This corresponds to a Guess obtained from the Neural Tangent Kernel (NTK, in finite width) of our model [154]. Note that this can be understood as a weaker version of DFA [264, 19], which we expand on in the Appendix.

4.2.2 Gradient Targets

Global Target. To update any set of weights, the most common gradient target in a supervised NN setting is the loss gradient of the last layer of our model. This is the gradient use in standard practice with backpropagation in most DL models. For Forward Gradients, Baydin et al. [20] uses this gradient as the target, as illustrated in Figure 4.1. We will study this setting extensively, and will also refer to this Target as the End-to-End Gradient Target.

Local Target. However, it is also possible to use the gradient of the auxiliary loss of the current layer as a gradient target to perform fully local learning. Ren et al. [296] uses such Local Targets in their experiments and obtains their best results with them. We refer to Local Learning as the case where the Gradient Guess for the Local Target is

equal to the Local Guess. (As for end-to-end training, this is a special case of Forward Gradient where the Guess is exactly equal to the Target).

Intermediate Target. One could also obtain gradient targets from auxiliary losses attached to any intermediate layer of the model. Such a target is likely to correlate better with the Local Guess while also being closer to the global target, perhaps providing a middle ground.

4.2.3 Gradient computation and insertion points

The candidate direction for the directional derivative obtained using the Forward Gradient technique can be computed in different spaces, with two natural candidates: parameters or activations.

To propose an update of the j-th stage $f_j(x_{j-1}, \theta_j)$, where θ_j describes the parameters of the j-th stage and x_{j-1} is the output of the stage f_{j-1} (x_0 is the input of the DNN), the strategy is to train a model via gradient descent. The output $f_j(x_j, \theta_j)$ is fed into a (target) loss $\mathcal{L}_j$, where $\mathcal{L}_j$ can represent a local loss obtained by attaching an auxiliary network to the current or a subsequent stage, or it can represent the standard loss at the output of the network. The procedure uses a sample estimate $\nabla_{\theta_j}(\mathcal{L}_j \circ f_j)$ of the gradient aggregated into a mini-batch $\mathcal{B} = \{x_j^1, ..., x_j^{\mathcal{B}}\}$ of data:

$$\nabla_{\mathcal{B}}(\mathcal{L}_j \circ f_j)(\theta_j) \triangleq \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \nabla_{\theta_j}(\mathcal{L}_j \circ f_j)(x_{j-1}^i, \theta_j) \tag{4.2}$$

$$= \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \partial_{\theta_j} f_j(x_{j-1}^i, \theta_j)^T \nabla_{x_j} \mathcal{L}_j(x_j^i) \tag{4.3}$$

Weight perturbation. A first natural strategy is to use the so-called weight perturbation of the gradient, which means that the estimate obtained in Eq. (4.2) is replaced by:

$$g_j = \frac{1}{\mathcal{B}} \left\langle \nabla_{\mathcal{B}}(\mathcal{L}_j \circ f_j)(\theta_j), \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} G_j^i \right\rangle \sum_{i=1}^{\mathcal{B}} G_j^i \tag{4.4}$$

where G_j^i is a gradient guess for $\nabla_{\theta_j}(\mathcal{L}_j \circ f_j)(x_{j-1}^i, \theta_j)$.

Activity perturbation. However, if G_j^i is pure noise, Ren et al. [296] observed that this estimate has a high variance. In this case, estimating the gradient via *activity perturbation* is preferable. In fact, note that the update of Eq. (4.3) can be approximated by:

$$\tilde{g}_j = \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \partial_{\theta_j} f_j(x_{j-1}^i, \theta_j)^T \left\langle \tilde{G}_j^i, \nabla_{x_j} \mathcal{L}_j(x_j^i) \right\rangle \tilde{G}_j^i \tag{4.5}$$

where $\tilde{G}_j^i$ is a guess for $\nabla_{x_j}\mathcal{L}_j(x_j^i)$, and which often has a smaller dimension than G_j^i, leading to a reduced variance in the case of e.g., pure noise [296].

Computational trade-offs. In Table 4.1, we summarize the unlocking capabilities as well as the computational trade-offs for the different pairs of Guesses and Targets we have introduced. As described in Jaderberg et al. [155], standard backpropagation is backward-locked and requires a global vector-Jacobian product. Forward Gradient, however, allows backward unlocking with a Global Target and even update unlocking with a Local Target. As noted in Ren et al. [296], weight perturbation has favorable memory usage, as activity perturbation requires storing intermediate activations for the Forward Gradient estimator. Activity perturbation also requires a backpropagation step in the corresponding stage to compute the gradient used to update the weights.

Table 4.1: **Unlocking and computational trade-offs** for different Guess and Target pairs. Here, vJp and Jvp stand for vector-Jacobian product and Jacobian-vector product respectively. The flag (local) means that the output of the operation (vJp or Jvp) for a given module is not needed by subsequent modules.

Guess		**Global Target**	**Local Target**
Local & **NTK**	**Unlocking**	Backward	Update
	Compute	vJp (local)+Jvp	vJp (local)
Random	**Unlocking**	Backward	Update
	Compute	Jvp	Jvp (local)

4.3 Numerical experiments

4.3.1 Experimental setup

We now describe how we implemented our models, the training procedure, and the implementations of Gradient Targets and Guesses to study the accuracy of a given model under the variations of these parameters.

Models. For this set of experiments, we consider two models that allow us to test different hypotheses about the scalability of gradient computations in forward mode: a standard ResNet-18 [130] (divided into 8 stages with local losses) and the same ResNet-18 without skip-connections, to avoid side-effects when combining this model with layer-wise local losses (divided into 16 stages with local losses, to obtain a fully layerwise subdivision of the network, except for the first stage which contains two layers).

Hyperparameters. We considered the CIFAR-10 and ImageNet32 datasets, used with standard data augmentation. The ImageNet32 dataset is smaller than the full-resolution

ImageNet dataset, making it more efficient to use during training and testing while still providing a challenging task for model performance. Chrabaszcz, Loshchilov, and Hutter [65] has shown that the conclusions drawn from the ImageNet32 dataset are generally applicable to the full-resolution ImageNet dataset. We followed a standard training procedure: SGD with a momentum of 0.9 and weight decay of 5×10^{-4}. For CIFAR-10, we train the model for 100 epochs, with a learning rate decayed by 0.2 every 30 epochs. For ImageNet32, we also first try a shorter training of 70 epochs, decaying the learning rate by 0.1 every 20 epochs. The initial learning rate was chosen among $\{0.05, 0.01, 0.005\}$ for CIFAR-10 and $\{0.1, 0.05, 0.01, 0.005, 0.0001\}$ for ImageNet32. We report the validation accuracy at the last epoch of training using the best training procedure and learning rate. More details are given in the Appendix.

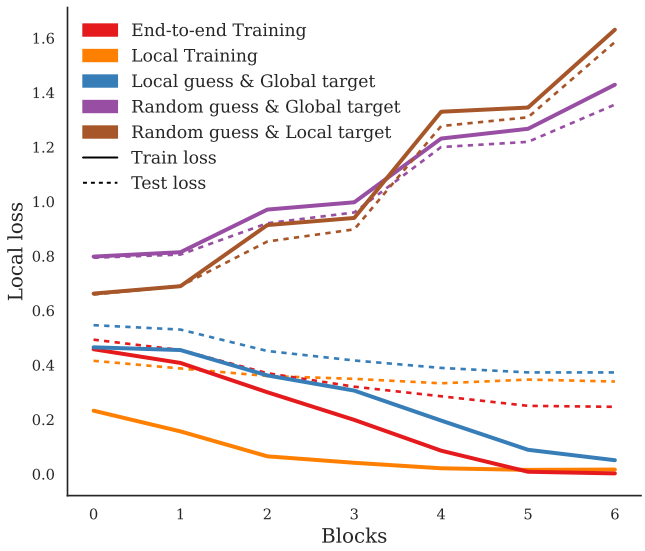


Figure 4.2: **Local train losses** at the end of training at each stage for a ResNet-18 split into 8 stages, with CNN auxiliary, for different training algorithms. In the Gaussian and end-to-end cases, the auxiliary training is detached from the main module training and is used for logging purposes only. With Gaussian Guesses, losses increase with depth. The Local Target allows better convergence than the Global Target at the first stage but an even worse convergence for the whole network. With local learning, local losses converge to local minima.

Local models and losses. We considered 3 types of trainable local auxiliary models: a CNN, an MLP, and a Linear Layer, designed and developed for use with CIFAR-10, to add no more than 10% computational overhead (in FLOPS) while leading to good accuracy. Again, we used a subset of the training data to cross-validate this architecture, which we then fixed during our experiments. Our local CNN auxiliary model is a 3-layer CNN with ReLU nonlinearities followed by a 2×2 adaptive average pooling

Table 4.2: **Test accuracy of a ResNet-18 using a Global Target, split into 16 local loss stages**, on CIFAR-10 and ImageNet32 for both activity and weight perturbations. Weight perturbation systematically outperforms activity perturbation with a Local Guess. The MLP auxiliary provides the best Local Guess in all configurations. We report the mean and standard deviation over 4 runs for CIFAR-10.

Dataset	**CIFAR-10**		**ImageNet32**	
End-to-End	94.3 ±0.1		53.7	
Space	**Activity**	**Weight**	**Activity**	**Weight**
Local, CNN	79.0 ±1.0	88.0 ±0.4	7.3	40.0
Local, MLP	84.7 ±0.3	88.7 ±1.2	21.8	37.4
Local, Linear	46.7 ±2.5	86.1 ±1.4	10.0	23.3
NTK, CNN	37.7 ±0.1	50.1 ±1.0	2.0	3.6
NTK, MLP	50.3 ±0.4	49.7 ±0.6	7.5	3.9
NTK, Linear	49.9 ±1.0	48.3 ±0.7	4.2	3.9
Gaussian	38.9 ±0.9	50.0 ±0.8	4.9	4.9
Rademacher	38.0 ±1.5	49.8 ±0.2	5.5	4.6

and a projection onto the classification space. The MLP is a 3-layer MLP with ReLU nonlinearities, followed by a projection onto the classification space. The linear auxiliary network consists of a 2×2 adaptive average pooling preceded by a batch normalization module, and followed by a projection onto the classification space as in Belilovsky et al. [26]. More details on our methodology for cross-validating the hyperparameters can be found in the Appendix. Each local loss can then be used to obtain Gradient Guesses via a backpropagation procedure or to serve as a Gradient Target for computing Forward Gradients.

4.3.2 The gap between backpropagation and Forward Gradient is narrowed with Local Guesses

Table 4.2 and Table 4.3 report our results for a ResNet-18 split into 8 or 16 stages, on both CIFAR-10 and ImageNet32. First, we note that the accuracies of Forward Gradient with a Random Guess as studied in Ren et al. [296] are extremely low. Despite extensive hyperparameter searches (notably for the learning rate), it proved difficult to achieve reasonable accuracies for ImageNet32 using random Gradient Guesses. This is consistent with the expectations of Belouze [27], and suggests that the architectural changes made in Ren et al. [296] are essential to its success.

Local Guesses reduce the gap with end-to-end training. Table 4.2 and Table 4.3 indicate that Local Guesses systematically outperform the naive Random Guesses strategy, in some cases by about 40% for CIFAR-10 and 20% for ImageNet32. This is not surprising,

Table 4.3: **Test accuracy of a ResNet-18 using a Global Target, split into 8 local loss stages**, on CIFAR-10 and ImageNet32 for both activation and weight perturbations. Local Guesses improve on Random Guesses and NTK Guesses. Activity perturbation only provides an advantage for Random Guesses, otherwise, weight perturbation performs better in all CIFAR-10 configurations. We report the mean and standard deviation over 4 runs for CIFAR-10.

Dataset	**CIFAR-10**		**ImageNet32**	
End-to-End	94.5 ±0.2		54.6	
Guess	Activity	Weight	Activity	Weight
Local, CNN	90.4 ±0.2	92.0 ±0.3	33.1	38.3
Local, MLP	89.2 ±0.2	91.6 ±0.2	38.0	45.8
Local, Linear	65.9 ±6.7	88.9 ±0.3	27.9	34.9
NTK, CNN	55.2 ±0.4	66.8 ±2.6	5.5	8.0
NTK, MLP	61.6 ±0.5	63.1 ±1.1	17.4	15.5
NTK, Linear	61.6 ±0.9	62.6 ±1.0	9.8	15.4

since a pure Gaussian Guess leads to random directions that are not aligned with the directions important for the classification task. Even NTK Guesses, which are random but benefit from the inductive bias of the auxiliary network, yield a slight improvement in most cases.

By reusing the same random network for a full mini-batch, we can obtain a gradient direction that is less naive than random noise and proving that NTK [154] has a good inductive bias for this task. We report in Figure 4.3 the train and test losses for Forward Gradient training with Global Target and several Guesses, showing that only Local Guesses allow convergence of the global loss.

Weight perturbation outperforms activity perturbation. In the case of supervised gradient guesses, we find that weight perturbation significantly outperforms activity perturbation. This shows that the best setup for the setting studied in Ren et al. [296] may not generalize to standard neural networks.

Training by residual blocks obtains a significantly better accuracy than training layerwise Comparing the results of Table 4.2 and Table 4.3 indicates that, for the same pair of algorithms, training by stages of two layers (corresponding to residual blocks) with one auxiliary per stage systematically outperforms the corresponding version with one auxiliary per layer. This observation is consistent with the fact that the layers encapsulated in a given stage are jointly trained in the first case. In contrast, the layerwise strategy is completely decoupled. The fact that using joint training systematically improves the accuracy of the method is known from previous work [26, 279, 25]. However, this comes at the cost of not exploiting the full potential of the

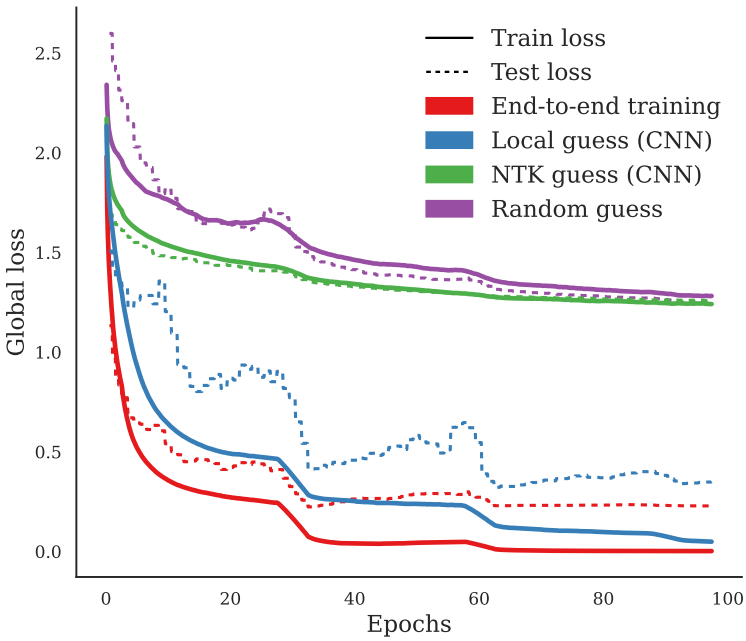


Figure 4.3: **Comparison of train and test losses** for end-to-end training (red) and Forward Gradient with Global Target and Local Guess (blue), NTK Guess (green), and Random Guess (purple), on CIFAR-10 with a ResNet-18 split into 8 stages. Local and NTK Guesses are derived from a CNN auxiliary. Random and NTK Guesses fail to optimize the Global Loss. Local Guesses perform much better, with a consistent gap with respect to end-to-end throughout training.

computational and memory savings associated with Forward Gradient. For Forward Gradient methods, we find that Local Guesses with weight perturbation are the Guesses that minimize the accuracy drop when transitioning from an 8 stages split to a 16 stages one.

4.3.3 Reliably estimating the Global Target is necessary for Forward Gradient to succeed

Accuracy degrades in upper layers with Random Guesses. Figure 4.2 shows the evolution of the separability of representations with depth via the training loss of a local (CNN) auxiliary for a ResNet-18 split into 8 stages. In the case of end-to-end training and Random Guesses for Global targets, the auxiliary losses were trained ad hoc without interacting with the training of the global model. Not surprisingly, the end-to-end training progressively separates classes, a phenomenon already observed in Zeiler and Fergus [394], Jacobsen, Smeulders, and Oyallon [153], and Oyallon [277]. Random isotropic Guesses do not follow this trend. On the contrary, the training loss decreases progressively with depth, both for Global and Local Targets. In comparison,

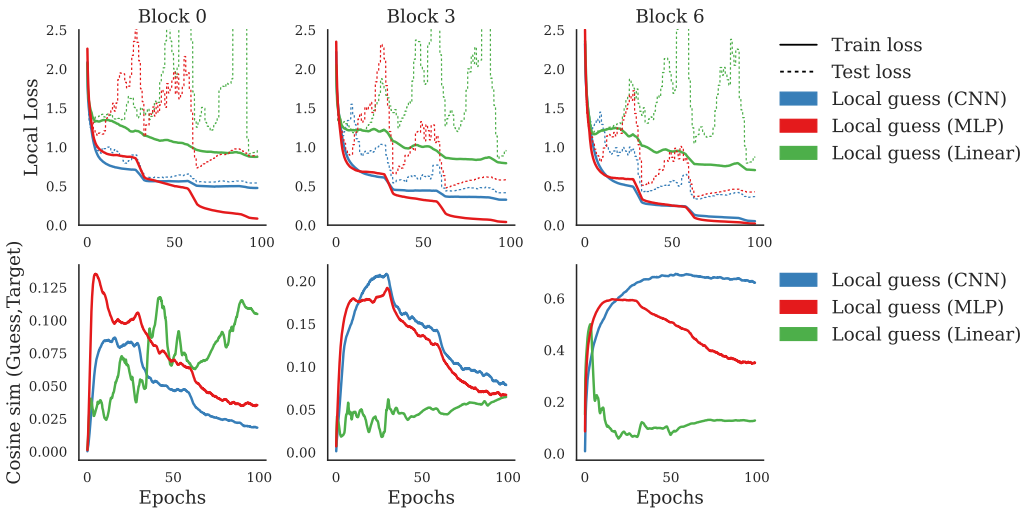


Figure 4.4: **Train and test local losses (top row) and mean cosine similarity between Local Guess and Global Target in the activation space (bottom row)**, for stages 0, 3, and 6 (left, middle, and right columns) during training. The model is a ResNet-18 divided into 8 stages trained on CIFAR-10. The similarity values are low but consistently positive and increase with depth as the target gets closer (and less deep). Learning improves with depth, as expected. As both target and local losses converge to a minimum, their similarity decreases.

Table 4.4: **Test accuracy on CIFAR-10 of each variation of a ResNet-18, using a Local Target with Gaussian, Rademacher, and Local Guesses, for both activity and weight perturbations.** We report the mean and standard deviation over 4 runs.

8 stages	**CNN**		**MLP**		**Linear**	
Local Target	92.0 ±0.2		92.1 ±0.2		89.2 ±0.1	
Guesses	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
Gaussian	41.2 ±1.9	52.1 ±1.5	45.6 ±1.6	53.2 ±0.6	36.9 ±1.9	56.5 ±0.8
Rademacher	39.0 ±1.3	53.0 ±0.7	43.6 ±2.1	53.3 ±0.9	34.6 ±1.9	56.6 ±0.6
16 stages	**CNN**		**MLP**		**Linear**	
Local Target	87.7 ±0.1		89.4 ±0.3		86.7 ±0.3	
Guesses	Activity	Weight	Activity	Weight	Activity	Weight
Gaussian	23.5 ±2.4	37.5 ±1.6	21.5 ±4.4	39.7 ±0.8	23.8 ±1.6	43.5 ±0.5
Rademacher	22.3 ±1.5	37.8 ±0.5	23.3 ±1.4	41.3 ±1.4	24.1 ±1.3	45.5 ±0.5

Local Guesses with Global Target accuracies improve with depth, as we expect from a

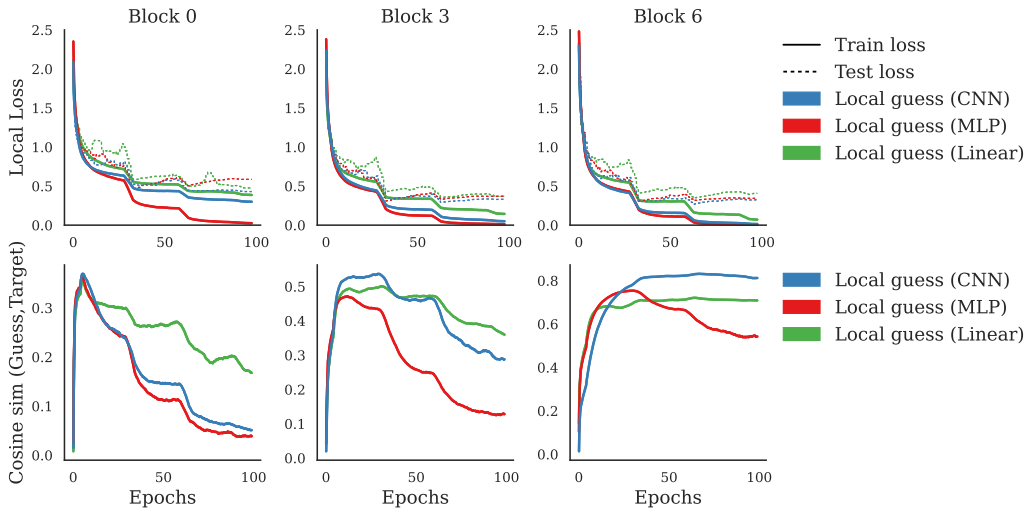


Figure 4.5: **Train and test local losses (top row) and mean cosine similarity between Local Guess and Global Target in the weight space (bottom row, averaged over the parameters)**, for stages 0, 3, and 6 (left, middle, and right columns) during training. The generalization gap is more consistent during training than with activation gradients. The cosine similarity is also consistently higher than for activation gradients but falls more drastically during training.

deep model. Furthermore, Random Guesses with Local Target give worse results than Random Guesses with Global Target, as reported in Table 4.4. This constitutes a further difference between this work and Ren et al. [296].

Intermediate Targets perform worse than Global Targets. To bridge the gap between Global and Local Targets, we proposed an Intermediate Target, the gradient of the Local loss at stage $j+1$. The idea is to have a target that is closer to the Gradient Guess than the Global Target, allowing for easier convergence (since similarity increases with depth) as well as improving on the use of a Local Target, where subsequent stages do not provide any feedback. However, despite more consistent alignment between Targets and Guesses across stages, we observed a consistent decrease in accuracy compared to both Global and Local Targets, as reported in Table 4.5. Thus, despite a good alignment between Guess and Target, not using a Global Target penalizes the learning of the network.

4.3.4 Local Guesses do not align with the Global Target

We now examine the difference between the optimization path of the Forward Gradient and the end-to-end gradient to understand if Forward Gradient can be understood as an approximation of the Global Target i.e., the End-to-End Gradient Target.

Table 4.5: **Test accuracy on CIFAR-10 of a Resnet-18 using the loss of the stage $j+1$ as the Intermediate Target for the stages j**, for both activity and weight perturbations. The model has been split into 8 and 16 stages as in the experiments of Table 4.3 and Table 4.2. We added the Target (tgt) used for the Random Guesses.

ResNet-18 split	**8 stages**		**16 stages**	
Gradient Guesses	**Activity**	**Weight**	**Activity**	**Weight**
Local, CNN	91.6	91.6	80.0	87.8
Local, MLP	91.5	91.8	87.6	89.5
Local, Linear	76.3	89.6	59.8	86.9
NTK, CNN	55.7	60.0	33.0	18.4
NTK, MLP	57.4	52.5	40.7	20.1
NTK, Linear	55.3	44.2	32.4	20.3
Gaussian, CNN tgt	45.3	51.5	28.0	47.1
Gaussian, MLP tgt	53.7	53.2	31.4	48.0
Gaussian, Linear tgt	51.4	53.9	42.5	46.4
Rademacher, CNN tgt	43.8	51.6	24.8	47.5
Rademacher, MLP tgt	54.7	52.9	30.4	49.5
Rademacher, Linear tgt	49.0	54.2	43.6	45.6

Local Guesses and Global Targets are weakly aligned. Figures 4.4 and 4.5 show the evolution of the local train and test losses, and the cosine similarity between Local Guesses and the Global Target, for a ResNet-18 split into 8 stages. For the cosine similarity, we report a batch estimate at each iteration given by:

$$\frac{1}{\mathcal{B}} \sum_{i \in \mathcal{B}} \frac{\left\langle \nabla_{x_j} \mathcal{L}_j(x_j^i), \tilde{G}_j^i \right\rangle}{\left\| \nabla_{x_j} \mathcal{L}_j(x_j^i) \right\| \left\| \tilde{G}_j^i \right\|}. \tag{4.6}$$

We note that the cosine similarity between the end-to-end gradient and the gradients obtained from local auxiliaries shows three main tendencies: **(a)** As the stage index increases, the alignment is higher (see the y-axis scale on each plot). **(b)** In most cases, the convolutional auxiliary shows the strongest correlation and the lowest test loss value. **(c)** Finally, although low, the similarity is consistently positive. The first statement reflects the depth proximity of the auxiliary to the Global Target; the second is likely due to the convolutional nature of the subsequent network. We further note that MLP and linear auxiliaries have similarly erratic test loss curves and that most similarity curves tend to descend towards the end of training. The former may be due to an architectural misalignment. The latter observation likely indicates that different local minima have been reached and that the optimization trajectories will drift apart from then on.

Projecting on the entire Gradient Guess space still does not estimate the Gradient Target. The low alignment between Local Guesses and Global Targets indicates a poor approximation. We propose projecting each Global Target on the subspace spanned by Local Guesses, for each sample to test this hypothesis further. In other words, we replace the gradient estimate of Eq. (4.5) with the best linear approximation of the Global Target using (the entire subspace of) Local Guesses. In practice, for a batch of gradients, we project each Global Target onto the span of the batch of Local Guesses. Our results are summarized in Table 4.6, and we trained each model using this novel dynamic. The improvement over projecting onto a single Local Guess is marginal, and the accuracy is still far from end-to-end training. This means that approximating the Global Target using Local Guesses is not sufficient to recover the dynamics of backpropagation training. In general, the task of approximating a Global Target using Gradient Guesses is likely to be a difficult task [251].

Table 4.6: **Test accuracy on CIFAR-10 of a Resnet-18 trained by using the best linear approximation of the end-to-end gradients by local guesses**, with activity perturbation. We report the mean and standard deviation over 4 runs.

Model	Gradient Guess subspace	Accuracy
ResNet-18, 8 stages	Local Guess, CNN	92.3 ±0.1
	Local Guess, MLP	89.6 ±0.3
	Local Guess, Linear	84.0 ±0.6
ResNet-18, 16 stages	Local Guess, CNN	89.6 ±0.6
	Local Guess, MLP	77.9 ±3.3
	Local Guess, Linear	73.4 ±1.8

Alignment between Guess and Target matters. Experimental results show that the Local Guess does not approximate the Global Target well. Comparing Table 4.4 and Table 4.6 shows that using the best linear approximation of the Global Target on the space spanned by the Local Guesses still performs at par or worse than local learning (i.e., Local Guess on Local Target). Thus, even though the Global Target is generally a better target than the Local Target in general, the adequation between the Guess and the Target has a stronger impact than the reweighting of the Guess. However, it is known that local learning saturates in early layers and does not benefit from the depth of the model [365], suggesting that global feedback may be necessary to recover the backpropagation performance. These issues suggest that deriving a good estimate of the end-to-end gradient is a difficult task that may need to be addressed to achieve performance on par with standard backpropagation.

Table 4.7: **Test accuracy of a ResNet-18 using a Global Target, on Fashion-MNIST, CIFAR-100 and Imagenette datasets split into 16 local loss stages**, for both activity and weight perturbations. We report the mean and standard deviation over 4 runs.

Dataset	**CIFAR-100**		**Fashion-MNIST**		**Imagenette**	
End-to-End	74.6 ±0.3		94.1 ±0.2		88.9 ±0.4	
Guess	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
Local, CNN	44.6 ±1.7	58.2 ±0.6	92.5 ±0.3	93.7 ±0.1	71.8 ±1.8	84.2 ±0.6
Local, MLP	59.5 ±0.5	64.3 ±0.4	92.3 ±0.2	93.8 ±0.1	77.3 ±1.3	82.8 ±0.3
Local, Linear	46.1 ±1.1	62.1 ±0.4	73.4 ±1.7	93.4 ±0.2	53.2 ±2.0	82.8 ±0.4
NTK, CNN	15.0 ±0.7	22.9 ±0.3	82.0 ±0.2	88.2 ±0.3	47.0 ±1.0	53.7 ±2.1
NTK, MLP	23.5 ±0.3	23.1 ±0.4	86.8 ±0.6	88.0 ±0.5	56.3 ±0.8	53.3 ±0.9
NTK, Linear	23.9 ±0.5	22.9 ±0.7	83.7 ±0.2	88.8 ±0.3	55.3 ±1.4	51.3 ±0.8
Gaussian	10.0 ±1.2	22.4 ±0.5	79.1 ±0.7	88.3 ±0.3	17.8 ±1.5	47.6 ±1.4
Rademacher	8.7 ±1.1	22.6 ±0.4	79.5 ±0.2	88.3 ±0.5	18.4 ±2.6	47.9 ±1.7

Table 4.8: **Test accuracy of a ResNet-18 using a Global Target, on Fashion-MNIST, CIFAR-100 and Imagenette datasets split into 8 local loss stages**, for both activity and weight perturbations. We report the mean and standard deviation over 4 runs.

Dataset	**CIFAR-100**		**Fashion-MNIST**		**Imagenette**	
End-to-End	75.7 ±0.2		93.9 ±0.1		90.0 ±0.4	
Guess	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
Local, CNN	64.5 ±0.9	67.9 ±0.4	93.7 ±1.3	93.6 ±0.2	84.9 ±0.3	88.0 ±0.1
Local, MLP	67.5 ±0.3	70.0 ±0.3	93.6 ±0.3	94.0 ±0.1	84.9 ±0.4	85.9 ±0.4
Local, Linear	50.0 ±9.8	65.9 ±0.4	87.4 ±1.5	94.0 ±0.1	70.7 ±1.7	86.1 ±0.5
NTK, CNN	29.9 ±0.7	40.3 ±1.5	87.2 ±0.4	91.1 ±0.3	65.9 ±0.7	68.2 ±1.0
NTK, MLP	35.9 ±1.0	36.1 ±0.2	88.3 ±0.3	90.8 ±0.2	69.7 ±1.0	67.9 ±0.7
NTK, Linear	36.4 ±0.4	37.8 ±2.2	88.5 ±0.4	90.8 ±0.3	68.7 ±0.6	68.6 ±0.5
Gaussian	27.5 ±0.5	35.3 ±0.5	87.5 ±0.2	89.5 ±0.3	52.7 ±3.1	64.1 ±1.1
Rademacher	27.5 ±0.4	33.2 ±1.2	87.1 ±0.7	89.7 ±0.2	51.8 ±1.7	63.8 ±0.9

4.3.5 Consistency across model size and datasets

Wide ResNets further show the curse of dimensionality. We study the effect of parameter sizes on our findings by applying different Target and Guess pairs on Wide ResNets. WIn Table 4.9, we present the accuracies we obtain for different guesses for a Global Target with a Wide ResNet-18 for different width factors k (for $k = 0.5$, 2, and 4, where $k = 1$ is the standard ResNet-18 we studied earlier) by scaling the number of

Table 4.9: **Test accuracy of a Wide ResNet-18 using a Global Target, split into 16 local loss stages**, on CIFAR-10 for both activity and weight perturbations for different width factors, i.e., k=0.5, 2 and 4. Table 4.2 refers to the case where $k = 1$. We report the mean and standard deviation over 4 runs.

Width factor k	0.5		2		4	
End-to-End	93.0 ±0.4		94.9 ±0.0		95.3 ±0.1	
Guesses	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
Local, CNN	63.3 ±1.6	72.4 ±0.6	87.0 ±0.4	85.5 ±0.3	90.6 ±0.2	93.3 ±1.5
Local, MLP	81.5 ±0.5	79.1 ±0.3	86.2 ±0.2	84.1 ±0.2	87.0 ±0.2	91.7 ±0.2
Local, Linear	26.9 ±3.0	78.5 ±0.3	62.7 ±2.8	84.6 ±0.2	62.5 ±4.3	92.3 ±0.3
NTK, CNN	33.8 ±1.3	48.9 ±0.2	41.1 ±0.4	50.6 ±0.3	44.2 ±0.3	51.6 ±0.3
NTK, MLP	48.9 ±1.3	47.7 ±0.4	50.9 ±0.5	50.6 ±0.2	51.8 ±0.2	51.6 ±0.3
NTK, Linear	47.9 ±1.2	47.6 ±1.2	51.4 ±0.4	48.7 ±0.7	52.1 ±0.3	49.3 ±0.7
Random, Gauss.	38.9 ±0.4	49.6 ±0.9	36.2 ±1.4	48.9 ±0.5	29.6 ±2.3	48.7 ±0.5
Random, Radem.	38.8 ±0.1	49.5 ±0.4	35.3 ±0.6	49.0 ±0.7	30.2 ±2.1	49.0 ±0.8

features according to k (for both the model and the auxiliary network). We observe that Local Guess accuracy improves with width. However, Random Guess accuracy decreases, confirming that Forward Gradient estimation with random directions deteriorates with increasing gradient size due to the curse of dimensionality. This in turn suggests that good directional Guesses should be pursued. We also provide in the Appendix results for a Local Target.

Our findings extend to other datasets. We also present results for other image classification datasets of varying difficulty. We consider the following datasets: Fashion-MNIST, a more complex surrogate for MNIST, CIFAR-100, the same dataset as CIFAR-10 with 100 labels, and Imagenette, a simpler subset of Imagenet with only 10 labels. We summarize the accuracies for a Global Target in Tables 4.7 and 4.8 and a Local Target in Table B.1 in App. B.2.1 for a ResNet-18 divided into 8 and 16 stages. We notice the same trends discussed in this study for CIFAR-10, thus confirming our findings.

4.4 Conclusion

We have extensively studied different Forward Gradient training variants for a ResNet-18 architecture divided into 8 and 16 stages on the standard object recognition tasks CIFAR-10 and ImageNet32. In particular, we introduce the use of gradients from locally supervised auxiliary losses as a better candidate direction for the Target than random noise. We varied Gradient Targets, Guesses, auxiliary networks, and feedback insertion points. Gradient Guesses were either Random isotropic directions, NTK gradients, or

local auxiliary network gradients. We confirmed our findings for other image datasets and varying model sizes.

We determined several unambiguous trends. Firstly, Gradient Guesses obtained from Local Guesses outperformed Random Guesses. Second, our study confirms that consistent estimation of the Global Target should be the main focus of Forward Gradient algorithms. Third, we conclude that the limitations of our method are due to the limited alignment between the local loss gradients and end-to-end gradients. Nevertheless, we found that Local Gradient Guesses reduce the gap with end-to-end training for Forward Gradient and that subsequent work needs to improve alignment between Guess and Target to reach end-to-end accuracy.

Acknowledgements

This work was supported by Project ANR-21-CE23-0030 ADONIS, EMERG-ADONIS from Alliance SU, and Sorbonne Center for Artificial Intelligence (SCAI) of Sorbonne University (IDEX SUPER 11-IDEX-0004). This work was granted access to the AI resources of IDRIS under the allocations 2022-AD011013095 and 2022-AD011013769 made by GENCI. We acknowledge funding and support from NSERC Discovery Grant RGPIN-2021-04104 and resources from Compute Canada and Calcul Quebec.

Side contribution: delayed gradient approaches with no memory overhead

In this work, we showed that local learning can significantly improve the Forward Gradient estimation. However, we also observed that the greedy nature of local learning makes it incompatible with the gradient dynamics of backpropagation. The gap with backpropagation can be reduced with better regularization, auxiliary networks or subsampling, as we showed earlier. Nevertheless, some form of feedback connection seems necessary to solve the problem of alignment between the local and global gradients. Another way to obtain a faster learning algorithm, despite being backward locked, is to allow the use of delayed gradients through pipelining. For this reason, we have worked on improving the current delayed gradient approaches [410, 411]. Despite the linear speedup such MP approaches provide, they have two main issues. First, they require delayed gradient stages to learn, which hinders their convergence. Second, to hold the mini-batch activations as well as the parameters used during the forward pass for the backward pass, their memory overhead scales quadratically with the number of stages, compared to linearly for standard backpropagation.

Contribution

Stéphane Rivaud, **Louis Fournier**, Thomas Pumir, Eugene Belilovsky, Michael Eickenberg and Edouard Oyallon. PETRA: Parallel End-to-end Training with Reversible Architectures. Preprint.

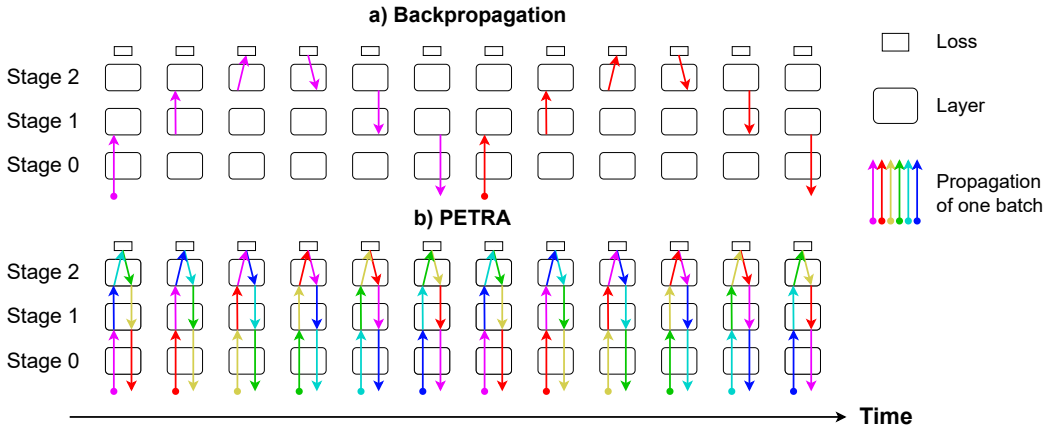


Figure 4.6: **Comparison of PETRA with standard backpropagation.** PETRA split the stages of the model and decoupled the forward and backward passes, resulting in a sixfold increase in parallelization speed here. The backward connections carry the reconstructed activations.

In this side work, we introduced a novel MP training framework, PETRA, that offers the same significant parallelization of delayed gradient approaches while avoiding the memory overhead of these methods. To achieve this, we decouple the forward and backward passes and use reversible architectures to reconstruct activations for the gradient computations, without parameter or activation buffers. Despite the use of approximated and delayed gradients, this method delivers competitive performance compared to standard backpropagation on standard computer vision datasets. The potential speedup resulting from training with PETRA compared to backpropagation is shown in Figure 4.6.

After presenting our contributions to the field of local learning in this part, we turn our attention to distributed approaches, the other side of parallel training in DL.

Part II

Communication-efficient distributed approaches for DL training

Chapter 5

Cyclic Data Parallelism for Efficient Parallelism of Deep Neural Networks

In this chapter, we present our first contribution to the field of distributed learning. We find that the standard DP framework suffers from two problems: the total activation memory peaks at the end of the forward pass simultaneously for all workers, and gradients require a collective all-reduce operation. We propose an alternative synchronous learning paradigm named Cyclic Data Parallelism. We shift the execution of the micro-batches from simultaneous to sequential, similar to the micro-batches in Pipeline Parallelism. We find that this paradigm improves several frameworks such as MP or ZeRO-DP, in terms of memory overhead or communication balancing. My contributions for this work were the initial idea, the proposed algorithm, all experiments and links to other frameworks.

Contribution

Louis Fournier and Edouard Oyallon. Cyclic Data Parallelism for Efficient Parallelism of Deep Neural Networks". Preprint.

5.1 Introduction

Deep learning models have grown significantly in size in recent years, reaching hundreds of billions of parameters and requiring increasingly expensive training [140]. As the cost of training these models continues to rise, there is an increasing need for parallelization strategies. In particular, DP [74, 208] remains the dominant method for training DNNs at scale. In DP, the model to be trained is first replicated on multiple workers. Then, during each training step, a mini-batch of data is sliced evenly among the workers in so-called micro-batches. Each worker then performs a forward and backward propagation

on each micro-batch, and the locally computed gradients are then averaged across all workers to obtain the gradient with respect to the full mini-batch. The resulting averaged gradient is used to locally update the models, typically following SGD.

However, DP has significant drawbacks. First, the communication step between workers is synchronous, as all workers must complete their gradient computations before communicating, resulting in idle workers waiting for the slowest worker [138]. Second, gradients are communicated globally using an all-reduce operation [116], which means that the communication step becomes a challenge as the number of workers increases [51]. Finally, the total memory used by all workers grows linearly with the number of workers since the model is fully replicated on each worker [289]. This requirement can be impractical since modern model sizes exceed the memory capacity of a single device (e.g., a GPU).

In this work, we tackle the memory and communication drawbacks of DP by proposing to change the execution time of workers in DP from simultaneous to sequential. We refer to this process as Cyclic Data Parallelism (CDP). Our modification of standard DP aims to balance both the communication costs and the overall memory usage, by relying heavily on the sequential nature of the execution steps of DNNs during training. More specifically, CDP reduces the cost of gradient communications in DP from collective communications at the end of the training step to point-to-point communications balanced over the entire training step. This balances the total memory used by all workers, at the cost of a small gradient delay. CDP can be combined with standard parallelization implementations for further improvements.

Contributions. Our contributions are as follows:

1. First, we propose CDP, an alternative to DP that balances gradient communications and overall memory usage across training, at the cost of computing on some delayed gradients.

2. We analyze the method by showing that it maintains a low communication and memory overhead during the computation of a mini-batch and allows a much more efficient implementation of mini-batch SGD on one or several GPUs.

3. We then particularize the CDP paradigm to state-of-the-art approaches such as MP and ZeRO-DP [289], showing improvements in all cases.

4. Empirically, we show that the gradient delay of CDP leads to equivalent training of DNNs compared to DP on the CIFAR-10 and ImageNet datasets.

Chapter organization. This chapter is structured as follows. We first present standard mini-batch SGD with DP in Section 5.2.1, before introducing our new CDP paradigm and two possible update rules in Section 5.2.2. We then discuss how CDP improves on DP for the implementations of DP (on a single or multiple GPUs), MP, and ZeRO-DP in Section 5.3. Next, in Section 5.4, we present a numerical analysis of the update rules of CDP over those of DP to train ResNets on CIFAR-10 and ImageNet, and show

the possible improvement in total memory for a ResNet-50 and a ViT-B/16. Finally, we link this work to another contribution on distributed learning that also focuses on overlapping computation and communication.

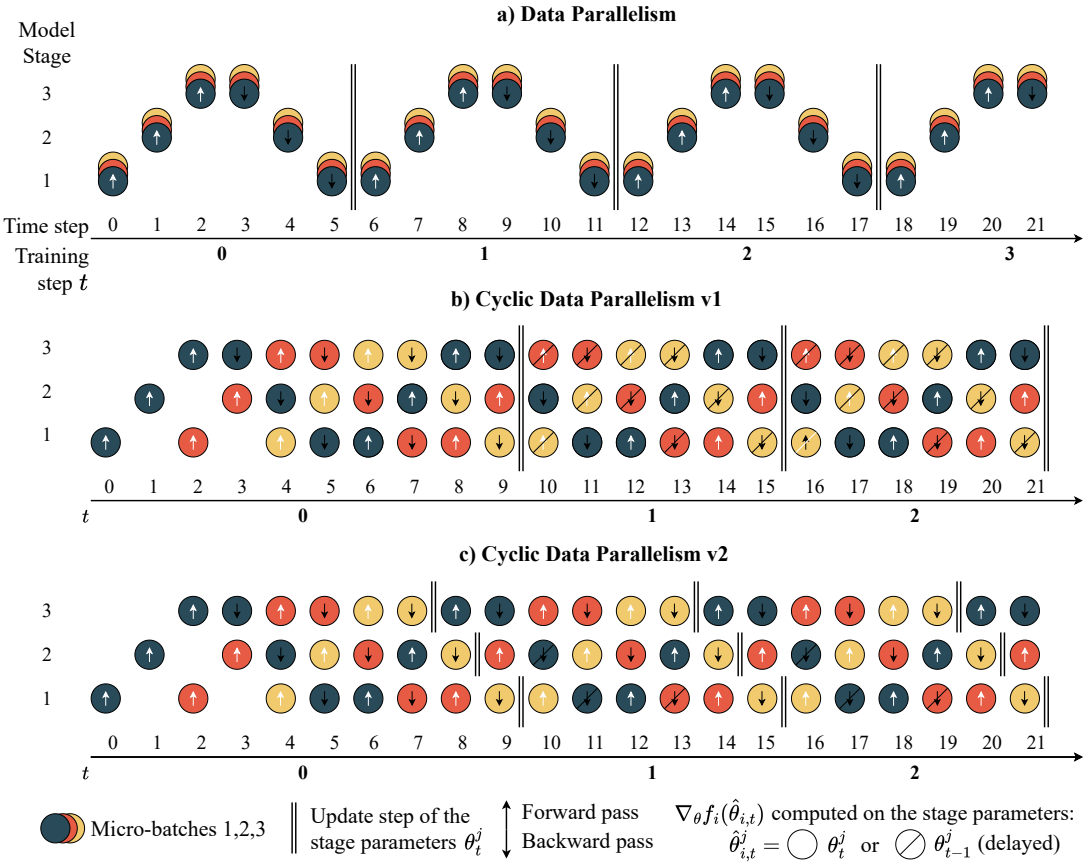


Figure 5.1: **Timeline of executions for Data Parallelism (DP) and the two versions of Cyclic Data Parallelism (CDP), for $N = 3$ workers. (a) DP.** The 3 workers start their forward pass execution simultaneously in DP, and maintain this synchronization throughout the entire forward-backward pass. **(b) CDP-v1.** In CDP, the 3 workers start execution with an equal delay between them (equal to 2 time steps). For CDP-v1 (see Eq. (CDP-v1)), the model parameters are updated with a delay constant equal to one training step. **(c) CDP-v2.** This delay is limited in CDP-v2 (see Eq. (CDP-v2)) by allowing the stages to update and send gradients independently. The communication scheme, balanced across the training step, is indicated. The overall complexity of a training step (a forward-backward pass) does not change, but the activation memory does not peak in CDP as it does in DP.

5.2 The Cyclic Data Parallelism paradigm

5.2.1 A motivating example: mini-batch SGD with Data Parallelism

Mini-batch SGD via DP. A standard DP strategy applied to a mini-batch SGD step is to replicate a model over N workers. Each worker is then fed with mini-batch slices, referred to as micro-batches [146]. At training step t, each replica $n \in [1, N]$ receives a micro-batch of data leading to a training objective f_n parameterized by θ_t, and computes its corresponding gradient $\nabla f_n(\theta_t)$. We assume that the model can be partitioned into J stages, with $J = N$ the number of workers, and write $\theta_t = \{\theta_t^j\}_{j \in [1,N]}$ to emphasize the dependence in the stage j. Parameter gradients are computed by a forward and backward pass over the micro-batch. They are averaged over all workers, typically using an all-reduce operation [51]. Finally, each worker locally updates the parameters θ_t with the averaged gradient. With the learning rate γ_t, the standard update rule [299], which can be written as

$$\theta_{t+1} = \theta_t - \frac{\gamma_t}{N} \sum_{n=1}^{N} \nabla f_n(\theta_t). \tag{DP}$$

The execution timeline of DP is illustrated in Fig. 5.1a, where one time step corresponds to the execution of a forward or backward pass of a stage. We assume that the forward and backward passes of a stage take a similar amount of time for the N workers. A training step t (which indexes the sequence of the parameters) is thus composed of $2N$ time steps. There are several inherent problems with the DP strategy applied to mini-batch SGD. First, the **total number of retained activations** peaks every $2N$ time steps, notably occurring at step 2 in Fig. 5.1a. This peak results from the intermediate activations that are stored and awaiting release during the backward pass. While this memory overhead can be mitigated by strategies such as pipelining [146], there exists an unavoidable waiting barrier every $2N$ steps to synchronize all gradient computations. This occurs, for example, between steps 5 and 6 in Fig. 5.1a. Consequently, this barrier introduces **latency** and leads to **workers idling**. Furthermore, this waiting barrier also leads to a **communication overhead**, as workers must simultaneously communicate their local buffers. While this can be solved by a collective all-reduce operation [51], communication can be an issue in centralized frameworks, such as Federated Learning [178].

5.2.2 Towards delayed mini-batch SGD with Cyclic Data Parallelism

Algorithmic description. Contrary to the previous DP approach, our main idea is to break the synchrony between the forward and backward passes of the N micro-batches. Instead of each step being computed simultaneously, we assume that each micro-batch computation is delayed by an identical number of time steps (i.e., if one worker processes one micro-batch, it starts with a delay relative to the previous worker). More specifically, the computations of ∇f_n are delayed by a delay of 2 time steps compared to ∇f_{n-1} (where n is taken modulo N, where N is the number of stages and micro-batches like before).

This results in a cyclic pattern, illustrated in Figures 5.1b and 5.1c. Furthermore, each stage constantly performs either a forward or a backward pass on a single and distinct micro-batch: in particular, this implies that the maximum number of activations stored at a given time step is nearly constant during training, and in this case, smaller compared to DP. We call this concept **Cyclic Data Parallelism** (CDP).

Update rules. While the forward and backward passes are fully defined assuming an available parameter θ_t, one has to define the corresponding update rule to generate θ_{t+1}, since breaking the synchrony between the backward passes of the micro-batches makes it impossible to follow Eq. (DP). Similar to [54, 387], at the training step t we introduce an auxiliary variable $\{\hat{\theta}_{n,t}\}_i$, a buffer that is updated by other concurrent workers. Formally, our algorithm can be written as

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{\gamma_t}{N} \sum_{n=1}^{N} \nabla f_n(\hat{\theta}_{n,t}), \\ \hat{\theta}^j_{n,t} &= u_{n,j}(\theta^j_t, \theta^j_{t-1}), \end{aligned} \tag{CDP}$$

where $u_{n,j}(a,b) \in \{a,b\}$ is a rule on the parameters, that depends on both the micro-batch n and the stage j. Thus, $u_{n,j}$ implicitly depends on the time step of the algorithm but not on the training step in our paradigm. An alternative that would is possible, and would easily allow more complex asynchronous or randomized variants. Note that some rules $u_{n,j}$ are not possible due to the delay between workers, preventing for example the update rule of DP. In particular, we propose two update rules that follow from Eq. (CDP) with specific rules $u_{n,j}$. They can be considered as the edge cases of the update rule, with maximum and minimum delay, respectively, with all other rules $u_{n,j}$ being an intermediate between them. First, we use a simultaneous barrier after the Nth batch has finished its computation, which is illustrated in Figure 5.1b with a barrier at the time step 9. In this specific case, a consistent update consists of the rule $u_{n,j}(a,b) = b$, using the stored $\hat{\theta}_{n,t} = \theta_{t-1}$. This results in

$$\theta_{t+1} = \theta_t - \frac{\gamma_t}{N} \sum_{n=1}^{N} \nabla f_n(\theta_{t-1}). \tag{CDP-v1}$$

We refer to this update rule as CDP-v1. In the specific case of PP, we recover the update rule of PipeDream-2BW [258]. We will now discuss another update rule, which we refer to as CDP-v2, and which is illustrated in Figure 5.1c. We fix the rule $u_{n,j}$ so that after the Nth micro-batch has finished computing the gradient of a stage, it directly transmits the updated stage parameters to be computed on a micro-batch (the order of communication can be predefined). This gives the rule $u_{n,j}(a,b) = a$ if $j \geq N - n + 1$, and b otherwise, reducing the number of delayed parameters compared to CDP-v1. Note also that we do not need to keep a copy of two parameters, as the parameters in memory

are always the freshest available. This update is written as

$$\theta_{t+1} = \theta_t - \frac{\gamma_t}{N} \sum_{n=1}^{N} \nabla f_n(\theta_{t-1}^1, ..., \theta_{t-1}^{N-n}, \theta_t^{N-n+1}, ..., \theta_t^N). \tag{CDP-v2}$$

Note that in both cases the gradient communication does not have to be *simultaneous*, instead the results can be communicated at intermediate steps, benefiting the overall communication bandwidth. We propose a possible communication scheme for CDP-v2 in Figure 5.1c. We describe in Alg. 1 the computation and communication done by a worker for the stage j operating on the n-th micro-batch.

Remark on the convergence. We emphasize that the generic update of Eq. (CDP) can, in the worst case, be understood as SGD with delayed gradients with a fixed delay of 1. There is a large literature studying the convergence of delayed first-order methods [247, 334, 388], which guarantees that our convergence rate is almost equal to that of SGD. In practice, very large DNNs using Transformer or GPT-2 architecture for instance have been shown to converge similarly with or without a delay of 1 [258, 295, 307, 378, 61, 81].

5.3 Variants of DP, MP and ZeRO-DP with CDP

We now present how CDP reduces the computational overhead of DP (on one or several GPUs), MP, and ZeRO-DP. A schematic representation of the implementations is presented in Figure 5.2. We first discuss the theoretical values of the communication and memory requirements, which are summarized in Table 5.1.

Analytical comparisons. We will refer to Ψ_P as the memory occupied by the parameters of the entire model (including the optimizer states here). Ψ_A refers to the memory occupied by the activations of one data sample in the entire model. In MP, stages communicate activations at the inputs and outputs of stages. This subset of activations occupies a memory, which we call Ψ_A^{int} (and thus $\Psi_A^{\text{int}} \leq \Psi_A$). From now on, N refers to the number of stages of the model as well as the number of micro-batches, which have equal size $\mathcal{B}$. In Table 5.1, we summarize the memory requirements per device, the communication volume, and the maximum number of communication steps required between 2 time steps, for DP, MP, PP, and ZeRO-DP with and without CDP. We find the following improvements with CDP compared to DP. CDP halves the memory required to train on DP with a single-GPU device. Multi-GPU DP with CDP doesn't require an all-reduce operation at the end of the training step but only point-to-point communications at each time step. This improvement in the number of communications required between time steps is also found with MP and ZeRO-DP with CDP. CDP allows MP to halve the number of GPUs needed, thus halving the memory required, as well as the communication of gradients between GPUs. The number of GPUs can be further reduced from MP with CDP to PP, which can be seen as a particular implementation of

Table 5.1: **Theoretical cost of the parallelism implementations discussed in Section 5.3.** All implementations are improved by using CDP (Cyclic) over DP, in terms of memory (per GPU or number of GPUs) or communication. Improvements are noted in bold. The parameter memory of the entire model is noted as Ψ_P, and the activation memory of the entire model for a data sample as Ψ_A, or Ψ_A^{int} for the subset communicated in MP. N indicates both the number of stages and the number of micro-batches (of size $\mathcal{B}$). The communication volume is the size of the tensors that need to be communicated. 'Max com. steps' is the maximum number of communication steps required between 2 time steps between two workers, which is a minimum of $\mathcal{O}(\log(N))$ steps for a collective operation or a single step for point-to-point communication. The update rule used is specified. Note that the parameter memory required in Single-GPU DP depends on the implementation.

	Memory per GPU		**Communication inter-GPU**		**Nb of**	
Implementation	**Activations**	**Parameters**	**Volume**	**Max com. steps**	**GPUs**	**Rule**
Single-GPU DP	$N\mathcal{B}\Psi_A$	$N\Psi_P$			1	(DP)
+ Cyclic	$\boldsymbol{\frac{N+1}{2}\mathcal{B}\Psi_A}$	$\boldsymbol{\frac{N+1}{2}\Psi_P}$			1	(CDP)
Multi-GPU DP	$\mathcal{B}\Psi_A$	Ψ_P	Ψ_P	$\mathcal{O}(\log(N))$	N	(DP)
+ Cyclic	$\mathcal{B}\Psi_A$	Ψ_P	Ψ_P	$\boldsymbol{\mathcal{O}(1)}$	N	(CDP)
DP with MP	$\mathcal{B}\Psi_A/N$	Ψ_P/N	$\Psi_P+\mathcal{B}\Psi_A^{\text{int}}$	$\mathcal{O}(\log(N))$	N^2	(DP)
+ Cyclic	$\mathcal{B}\Psi_A/N$	Ψ_P/N	$\boldsymbol{\frac{\Psi_P}{2}+\mathcal{B}\Psi_A^{\text{int}}}$	$\boldsymbol{\mathcal{O}(1)}$	$\boldsymbol{\frac{N+1}{2}N}$	(CDP)
PP	$\mathcal{B}\Psi_A$	Ψ_P/N	$\mathcal{B}\Psi_A^{\text{int}}$	$\mathcal{O}(1)$	N	(CDP)
ZeRO-DP	$\mathcal{B}\Psi_A$	Ψ_P/N	Ψ_P	$\mathcal{O}(\log(N))$	N	(DP)
+ Cyclic	$\mathcal{B}\Psi_A$	Ψ_P/N	Ψ_P	$\boldsymbol{\mathcal{O}(1)}$	N	(CDP)

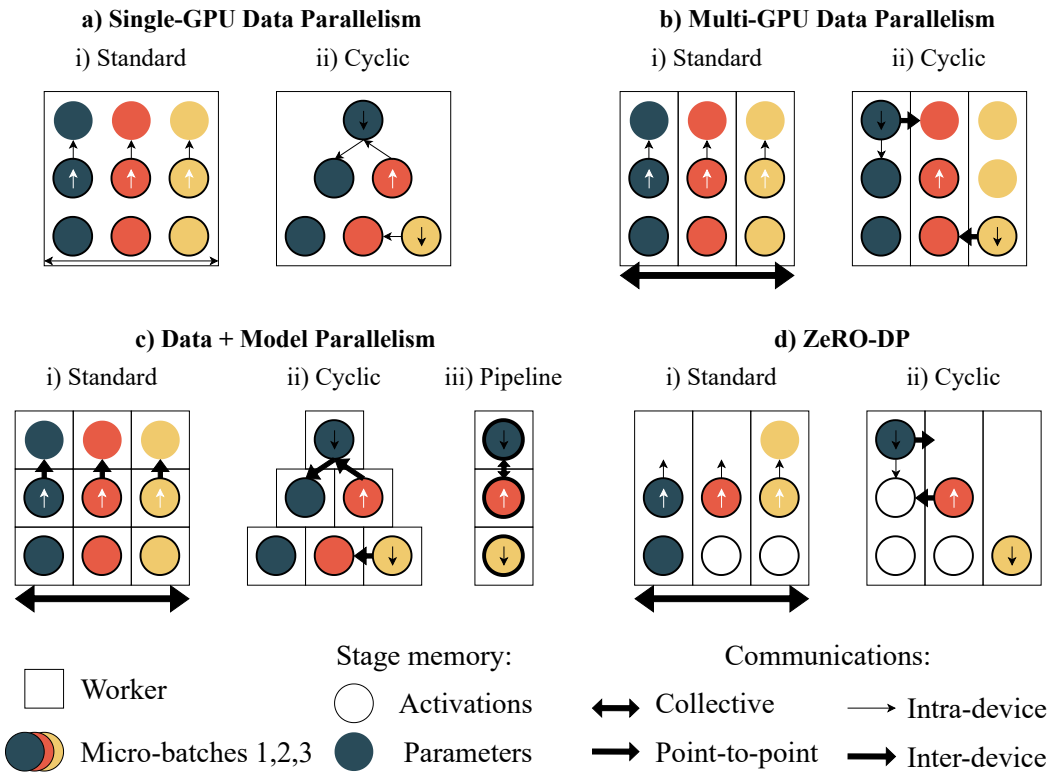


Figure 5.2: **Comparison between parallelism frameworks with and without using CDP**, for $N = 3$. A device (e.g., a GPU) is represented by a rectangle, and the different micro-batches computed are represented by the 3 colors. A model stage requires memory, for the parameters used for computation and for the retained activations waiting for the backward pass, indicated by a colored disk and a black circle. Communications are intra or inter-device (thin or thick arrow), collective or point-to-point (double or single arrow). **(a) Single-GPU DP.** This setting corresponds to a high-connectivity device with limited memory. We observe a memory reduction of half. **(b) Multi-GPU DP.** Communications can be drastically reduced when using multiple GPUs with CDP. **(c) DP+MP.** Both the number of GPUs required and the communications are reduced compared to a standard implementation of MP with DP. Only N GPUs are needed in PP, but they require more activation memory, shown with a thicker circle. **(d) ZeRO-DP.** The model states need to be sent or received by only one worker at each time step, instead of the standard broadcast operation of ZeRO-DP.

CDP. However, this requires the GPUs to be able to store the activations of the entire mini-batch. More generally, a disadvantage of MP and PP is that they need to communicate activations, a communication volume that scales with the batch size $\mathcal{B}$. We will now discuss each setting in more depth.

5.3.1 CDP implementation on a single-GPU device

We first explore the setting of a single-GPU device training with mini-batch SGD, assuming that communication within the GPU is cheap and fast, but memory is limited [63]. We illustrate the training of standard DP on a single-GPU device in Figure 2.a.i. Then, at the peak of the forward pass, the GPU must retain the activations of the entire model for N micro-batches, representing the total mini-batch, resulting in a total memory of activations of about $N\mathcal{B}\Psi_A$. Turning our attention to CDP, depicted in Figure 2.a.ii, we find that its implementation on a single-GPU device requires about half the memory compared to standard DP. Indeed, in this approach, each stage of the model processes one micro-batch at each time step. Consequently, the total memory occupied by activations remains nearly constant across time steps, approximately equal to $\frac{N+1}{2}\mathcal{B}\Psi_A$, reducing the total memory required for a DP implementation by half. Depending on the implementation, parameters may be shared on the GPU, resulting in no parameter memory improvement. The memory improvement is still significant in this case, as activation memory is generally much larger than parameter memory, peaking at 60GB for a GPT-2 model in [289], compared to 3GB for the fp16 parameters.

5.3.2 CDP implementation on a multi-GPU device

Here, one GPU processes a single micro-batch, but communication between GPUs may hinder the overall system efficiency. We consider the case where DP is implemented on N GPUs that process N micro-batches of data independently, as depicted in Figure 2.b.i.. These GPUs communicate the gradients of the micro-batches with an all-reduce operation at the end of the training step before proceeding to the next time step. All-reduce is a collective operation that requires at least $\mathcal{O}(\log(N))$ communication steps [51] in a favorable setting, or $\mathcal{O}(N)$ steps for a bandwidth-optimal ring-based implementation [278]. CDP on N GPUs, as shown in Figure 2.b.ii., makes better use of GPU-to-GPU communication bandwidth by breaking the all-reduce operation across the training step into point-to-point operations between each time step. Indeed, half of the stages compute a backward pass at each time step, and then send the computed gradient to the next worker (modulo N) for the reduce operation. This communication scheme corresponds to the ring-based all-reduce operation [278], which has an optimal bandwidth usage.

5.3.3 Implementation in a MP paradigm

Now, in MP, a GPU does not hold a full replica of a model but only of a single stage. During the forward and backward passes, activations and gradients must be communicated between GPUs holding successive stages. A naive implementation of DP and MP results in Figure 2.c.i., where the N micro-batches require N^2 GPUs to process each slice of the data and the model. Furthermore, note also that only N GPUs are busy at a time, so most GPUs are idle, waiting for the next micro-batch of data to pass through, which is a significant inefficiency of MP. Here, CDP with MP, which we present in Figure 2.c.ii., once again improves on the DP implementation. In particular, the

number of GPUs required for CDP is reduced, and we can show that for each stage j, CDP requires only $N - j + 1$ GPUs to be shared among the N micro-batches. This leads to a "pyramidal" shape of the stage structure. Thus in total, only $\frac{N+1}{2}N$ GPUs are needed to hold a stage of the model, halving the number compared to DP, as well as the memory requirements. Compared to DP, a micro-batch doesn't send the activations of a stage to the same GPU each time for the computation of the next stage. Instead, a GPU in the previous time step will have released the activations stored in its memory after a backward pass. The micro-batch will send the activations to this GPU as its memory is available for computation. Since the number of devices is smaller in DP than in CDP, note also that the total number of gradient communications between devices is reduced. In fact, if a GPU can only able to hold the activation of one micro-batch, the number of GPUs used, $\frac{N+1}{2}N$, is optimal to compute N micro-batches. An in-depth discussion is proposed in the Appendix of this chapter.

Pipeline Parallelism implementation. If we assume that one GPU can store the activations of all N micro-batches, then we can further reduce the total number of devices needed to only N, one per stage. Indeed, this implementation of CDP with MP on N devices, depicted in Figure 2.c.iii., is equivalent to PP as presented in PipeDream [259]. If we follow our first update rule Eq. (CDP-v1), CDP follows the same update rule as PipeDream-2BW [258]. However, our second update rule Eq. (CDP-v2) improves on this update rule by reducing the gradient delay. PP requires fewer GPUs than MP, but only if the GPUs can store the activations of the entire mini-batch. This limits its scaling capacity compared to MP, which is 2 times more GPU efficient. Indeed, for GPUs with similar memory capacities, MP with CDP requires $\frac{N+1}{2}N$ GPUs to train a batch of size $\mathcal{B}$. Meanwhile, PP requires N GPUs to train a batch of size $\frac{\mathcal{B}}{N}$, so, combined with DP, N^2 GPUs for a batch of size $\mathcal{B}$.

5.3.4 Implementation in a ZeRO-DP paradigm

ZeRO-DP [289] is a training framework that aims to combine the advantages of DP and MP, which we represent in Figure 2.d.i.. Instead of replicating the entire model on each of the N GPUs, ZeRO-DP replicates only the model states of a single stage across the GPUs. The model state (in stage 3 of ZeRO-DP) refers to the model parameters, gradients, and optimizer states. When the workers execute the forward or backward propagation through a stage, the model states of that stage are broadcast from the GPU that stores them to all GPUs. After the computation, the model states are deleted to free up the memory, so that a GPU only retains the model states of a maximum of two stages. The communication volume is similar to standard multi-GPU DP, with a maximum increase of 1.5. Note, however, that as with PP, the memory occupied by the activations on one GPU increases with N, since all stages of the model must be retained. CDP improves on ZeRO-DP by eliminating the need for collective communication of the model states, as shown in Figure 2.d.ii.. Since one stage is computed by a single GPU at a time step, the model states need only to be held by that GPU, without replication.

Then, they only need to be communicated to a single GPU at the next time step. This reduces the communication overhead at each time step, similar to Multi-GPU DP.

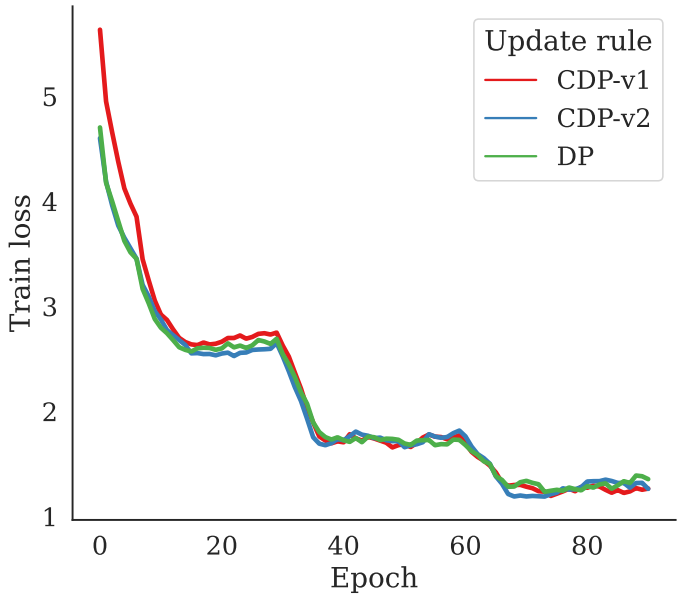


Figure 5.3: **Training loss of a ResNet-50 trained on ImageNet** following the learning rules (DP), (CDP-v1) and (CDP-v2). Values are averaged over a window of 7 epochs for the sake of readability. The loss of CDP-v1 is significantly higher at the beginning of training, which is not the case for CDP-v2. As the parameters converge, the effect of the delay disappears and the three losses show a similar training curve, with a small advantage for both CDP-v1 and CDP-v2. This confirms that the delay in our update rules does not affect convergence, even in realistic settings.

5.4 Numerical experiments

Framework. To test our method, we propose to use the standard training pipeline on the CIFAR-10 [183] and ImageNet [76] datasets, trained on the ResNet-18 and ResNet-50 architectures, split into 4 stages with similar FLOPs. We simulate our delayed gradients to train with SGD following DP, CDP-v1, and CDP-v2. For ImageNet, following standard practice, we report the maximum validation accuracy over the last 10 epochs. Contrary to [387], we did not need any specific tuning of the other hyperparameters for our update rules. More details are provided in the Appendix. Our source code is available at: github.com/fournierlouis/Cyclic_Data_Parallelism.

Table 5.2: **Top-1 test accuracy for the three learning rules** (CDP-v1), (CDP-v2) **and** (DP) **on the (a) CIFAR-10 and (b) ImageNet datasets**, by training a ResNet-18 and a ResNet-50. Our results are stable, as the standard deviation over 5 runs is systematically less than 0.08. We observe that on CIFAR-10, CDP systematically performs similar or better than DP, especially CDP-v2. On ImageNet, CDP performs similar to or better than DP.

(a) Test accuracy on CIFAR-10

	Learning Rule		
Model	(CDP-v1)	(CDP-v2)	(DP)
ResNet-18	94.1	**94.8**	94.7
ResNet-50	94.0	**94.5**	**94.5**

(b) Test accuracy on ImageNet

	Learning Rule		
Model	(CDP-v1)	(CDP-v2)	(DP)
ResNet-18	69.9	70.0	**70.1**
ResNet-50	**75.8**	75.7	75.4

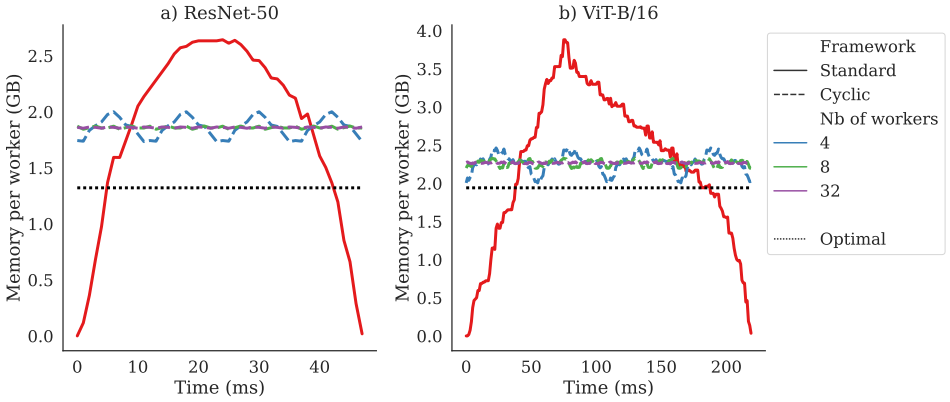


Figure 5.4: **Activation memory per worker, when training with N workers on ImageNet** with an efficient implementation of DP (solid) and a CDP (dashed), on a ResNet-50 and a ViT-B/16. An optimal halving of the parameters is represented in black ('Optimal'). The memory required for a forward-backward pass for a worker is first tracked, and the parameter memory is removed. The figure is extrapolated by mimicking the total memory used by N workers training on DP (i.e., simultaneous) or CDP (i.e., cyclic) and dividing by N. As N increases, the memory required by CDP flattens out to a value less than that required by DP. This value is close to the theoretical halving for the ViT-B/16, with 42%. The heterogeneity of the layers of the ResNet reduces the effectiveness of CDP, reaching only a 30% reduction.

Results. We report in Table 5.2a and Table 5.2b the test accuracy of our models for the three rules CDP-v1, CDP-v2, and DP, for 5 runs on CIFAR-10 and 3 runs on ImageNet respectively. The reported variances in our runs are less than 0.08, indicating that our experiments produce relatively stable results. Both tables show that CDP leads to similar or better performance compared to DP, which is consistent with the experimental findings of [258, 295]. For CIFAR-10, CDP-v2 significantly outperforms

CDP-v1, demonstrating our improvement over PipeDream-2BW's rule.

We also show in Figure 5.3 the training loss of the three concurrent learning rules on the ImageNet dataset: we note that the final loss of DP is slightly higher than both CDP-v1 and CDP-v2, with similar generalization performance, indicating how close the three methods are from both an optimization and statistical point of view. This is consistent with the theoretical insights of [247].

Activation memory tracking. In Figure 5.4, we track the memory used during one forward-backward pass of a ResNet-50 and a ViT-B/16 training on ImageNet. By subtracting the constant value corresponding to the parameter memory, we obtain the activation memory. From this memory curve, which clearly shows that the activation memory peaks at the end of the forward pass, we extrapolate for $N = 4$, 8, and 32 the activation memory usage per worker (total memory over N homogeneous workers divided by N) that an efficient implementation of DP and CDP would use. As expected, the activation memory used varies less during training with CDP, flattening out as N increases. Furthermore, the maximum activation memory used by the DP paradigm is significantly higher than that of CDP. The activation memory ratio we find here is about 30% for the ResNet-50 and 42% ($= \frac{3.9-2.3\ \text{GB}}{3.9\ \text{GB}}$) for the ViT-B/16, close to the theoretically ideal halving. The ResNet achieves a lower memory improvement due to the heterogeneity of its layers, which have different activation memory requirements for the same execution time (as feature size decreases with depth). This is not an issue for a Transformer-based model because the feature size remains constant over depth. Although a heterogeneous environment will reduce this improvement, this confirms that CDP will significantly improve memory usage in real implementations when using homogeneous stages and workers.

5.5 Conclusion

We introduced Cyclic Data Parallelism (CDP), an alternative framework to Data Parallelism. By executing the forward and backward passes of micro-batches of data cyclically rather than simultaneously, we balance gradient communication and the total memory occupied by activations. We particularize CDP in the context of Data, Model, and Pipeline Parallelism as well as ZeRO-DP, and demonstrate improvements in the total memory required to store activations or in the number of communication steps required between time steps during training. In particular, CDP reduces the number of devices required in MP and reduces the communication delay in ZeRO-DP. Our results are supported by existing theoretical guarantees in the small delay settings. Finally, our numerical experiments on ImageNet show that our update rules achieve similar testing accuracy as standard DP. In future work, we would like to release a highly efficient implementation compatible with cuDNN frameworks, as well as further relax our update rule to explore the possibility of using asynchronous and random delays.

Algorithm 1 Worker (i, j) in CDPv2, for a stage j on micro-batch i.

Input: Stage replica f_j, number of models/stages N, initial parameters θ_j^0, training steps T, dataset $\mathcal{D}$, optimizer OPT.
while $t < T$ **do**
  #### Forward pass
  # Propagate the new parameters
  $\theta_j \leftarrow$ **Wait and Receive** from worker $((i-1)\%N, j)$ if $i \neq (N+1-j)$ else (N, j)
  Send θ_j to worker $((i+1)\%N, j)$
  # Receive the activations
  if $j = 1$ **then**
    $x_j \leftarrow$ **Sample from dataset** $\mathcal{D}$
  else
    $x_j \leftarrow$ **Wait and Receive** from worker $(i, j-1)$
  # Compute the forward pass
  $x_{j+1} \leftarrow f_j(x_j, \theta_j)$
  # Propagate the activations
  if $j < N$ **then**
    Send x_{j+1} to worker $(i, j+1)$

  #### Backward pass
  # Receive the gradients
  if $i = 1$ **then**
    $\Delta_j \leftarrow 0$
  else
    $\Delta_j \leftarrow$ **Wait and Receive** from worker $(i-1, j)$
  if $j = N$ **then**
    $\delta_{j+1} \leftarrow \nabla\mathcal{L}(x_J)$
  else
    $\delta_{j+1} \leftarrow$ **Wait and Receive** from worker $(i, j+1)$
  # Compute the backward pass
  $\delta_j \leftarrow \frac{\partial f_j}{\partial x_j}^T \delta_{j+1}$
  $\Delta_j \leftarrow \Delta_j + \frac{1}{N} \frac{\partial f_j}{\partial \theta_j}^T \delta_{j+1}$
  # Propagate the gradients or the new parameters
  if j>1 **then**
    Send δ_j to worker $(i, j-1)$
  if i=N **then**
    $\theta_j \leftarrow \text{OPT}(\theta_j, \Delta_j)$
    Send θ_j to worker $(N+1-j, j)$
  else
    Send Δ_j to worker $(i+1, j)$

Side contribution: Overlapping communications in distributed LLM training

In this work, we found that we can improve synchronous data parallelism by balancing the memory overhead and gradient communication with sequential worker executions, at the cost of a small gradient delay. This results in a strong solution for learning in clusters composed of homogeneous devices.

In a side work, we also work on a solution that addresses the case of a cluster composed of heterogeneous workers. By allowing workers to accumulate during several gradient steps while communicating in parallel, it is possible to have a complete overlap of communication and computation at the cost of a one-step delay in the gradients. With this approach, the optimizer state can also be sharded among the workers, as in ZeRO-DP. Heterogeneous devices are handled by this framework, since each worker will accumulate more gradients according to its speed. Our method, ACCO (ACcumulate while you COmmunicate), is empirically confirmed to lead to drastically reduced training times for LLM compared to standard ZeRO-DP, especially in multi-node settings or with heterogeneous devices. The training of two workers with ACCO is shown in Figure 5.5. (* indicates equal contribution)

Contribution

Adel Nabli, **Louis Fournier***, Pierre Erbacher*, Louis Serrano, Eugene Belilovsky and Edouard Oyallon. ACCO: Accumulate while you Communicate, Hiding Communications in Distributed LLM Training. Preprint.

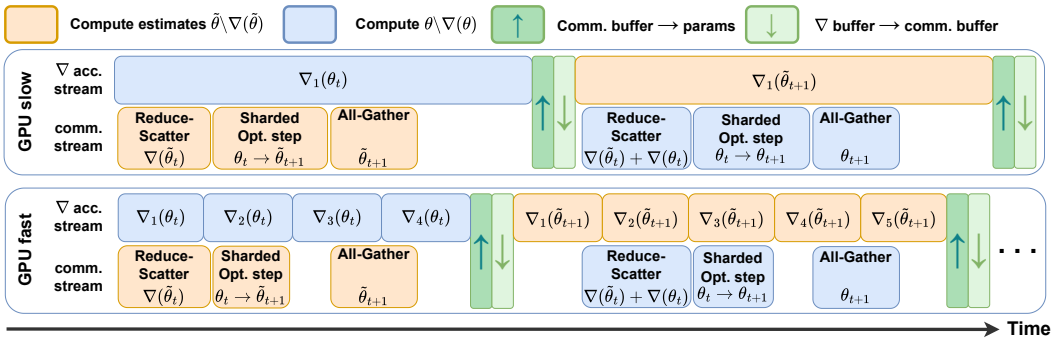


Figure 5.5: Overview of ACCO with a slow and a fast worker running in parallel. Communications are completely hidden. The delayed update is compensated by splitting the update in two steps and using weight prediction. Figure courtesy of Adel Nabli.

Future work: mitigating gradient delays with learned optimizers Nevertheless, we found that delay can be an issue for convergence in ACCO and proposed a novel delay mitigation method. Within the two works presented in this chapter, as well as the delayed gradient approaches in MP, we find that the use of delayed gradients is a key issue in the adoption of these methods. Several techniques exist to mitigate the effect of this delay on convergence, but with limited results and not designed for optimizers other than SGD.

In future work, we propose that the learned optimizer framework [239, 240, 129] can be used to meta-learn optimizers specifically for learning approaches that require delayed gradients, rather than using SGD or Adam. This would reduce the performance gap between delayed gradient methods and backpropagation. In particular, these optimizers could outperform traditional delay reduction techniques by using the same features as these techniques (e.g., the difference between the parameters at the time the gradient was computed and now).

We are working on making such approaches viable compared to other delay reduction techniques, through meta-learning optimizers following the framework of [239]. We can depict meta-learning in Figure 5.6. In the inner task, the models are updated at each training step using the learned optimizer U, a small neural network that takes as input the delayed gradient and other possible buffers and features. In the outer task, the resulting losses from the inner task provide an outer loss value $\mathcal{L}$. The inner task is then unrolled to obtain the gradient of the outer loss with respect to the learned optimizer parameters, which are then updated with Adam.

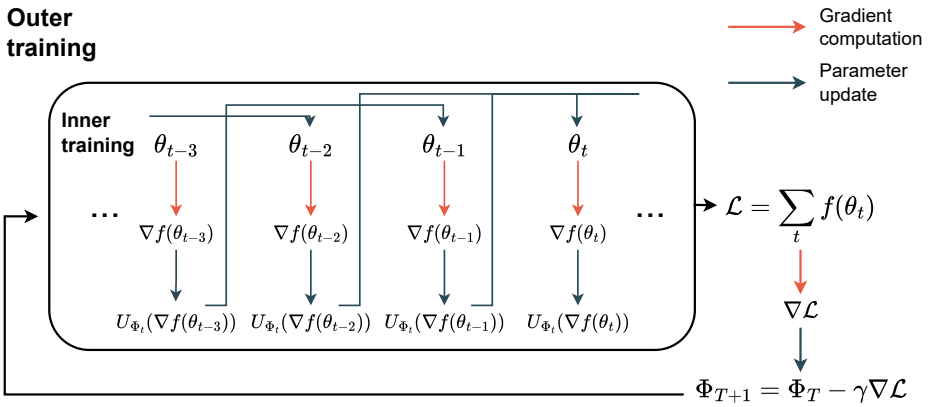


Figure 5.6: **Representation of meta-learning optimizers for delayed gradient approaches.** Learned optimizers are used in an inner training task that uses a delayed update rule (in this case, a fixed constant delay). The inner training losses are used to compute an outer loss value, which is then used to compute a gradient to update the parameters of the learned optimizer.

Chapter 6

WASH: Train your Ensemble with Communication-Efficient Weight Shuffling, then Average

In this chapter, we present our second contribution to the field of distributed learning. After having improved the synchronous distributed approach of DP, where model replicas are kept equal at each time step, we focus on the opposite idea, ensemble learning, where model replicas communicate rarely and are kept different. We are inspired by the distributed methods that train an ensemble of DNNs so that they can be weight-averaged into a single powerful model. However, these approaches either regularly collapse the DNNs into a single model, losing the diversity that gives the population its ensemble performance; or require a communication volume similar to DP. We propose a novel approach that leads to in a strong weight-averaged model while drastically reducing the communication volume.

The idea for this method comes from another project. Consider, as discussed in the previous chapter, a DP and MP framework, where devices hold different replicas of each stage of the network. During the forward and backward passes, the stages of a DNN replica communicate activations and gradients. Note, however, that stages corresponding to different replicas of the DNN may also communicate, for the same communication volume. Then, instead of each model forwarding to the next stage associated with its replica, one could have a 'stochastic routing', where a stage forwards its activations to any of the replicas of the next stage. As the replicas have to handle activations from all workers, this will results in a better consensus between the replicas, while requiring no additional communication. Now, rather than thinking of the activations being routed randomly at each step, it is strictly equivalent to think of the parameters of each stage as having been permuted randomly between replicas before the forward pass. In this work, we focus on this particular view of permuting parameters, to keep disconnected models in the same loss basin. Part of this view of activations being routed to other devices also

resulted in the idea of CDP from the last chapter.

My contributions in this work were first part of the original stochastic routing idea, with Adel Nabli. After finding no success of this method for standard distributed learning, a discussion with Masih Aminbeidokhti led me to investigate this idea with regard to ensemble training. I then performed all experiments and analysis in this work, except for the Appendix heatmaps.

Contribution

Louis Fournier, Adel Nabli, Masih Aminbeidokhti, Marco Pedersoli, Eugene Belilovsky and Edouard Oyallon. Preprint. WASH: Train your Ensemble with Communication-Efficient Weight Shuffling, then Average. Preprint.

6.1 Introduction

In order to enhance the accuracy of a given class of models, the answers of multiple instances trained in parallel can be aggregated via model *ensembling*. This can lead to significant improvements in modern deep learning models [99], increasing the generalization ability. However, this comes at the cost of evaluating multiple instances of a given model during inference. This increases both memory and computational requirements, resources that can be critical for on-device inference [237]. To solve this problem, the population of models can be fused into a single model to obtain both the generalization improvements of ensembling and the inference cost of a single model. Since independent models can be linearly connectable [102], a simple technique is to average the weights of the different models to obtain a fused model [376].

However, there are limits to this method. For models that are too dissimilar, the performance of the averaged model may be no better than chance [151]. To mitigate this, the ensemble can either use a pre-trained network as a starting point [261] or ensure that models share part of their optimization path [102]. However, reducing ensemble diversity too much comes at the expense of performance (see Figure 6 of [99]), revealing a trade-off between model diversity and weight averagability. Inspired by distributed training, techniques such as DART [157] and PAPA [161] have been proposed to train a population of models in parallel on heterogeneous data while communicating to balance this trade-off. DART, similar to LocalSGD [332], periodically averages all models to avoid model divergence. PAPA controls the diversity of the models more finely by pushing them toward the averaged parameters using an Exponential Moving Average (EMA) like EASGD [398], achieving better performance. In particular, they show that training a population in this way results in models that generalize better than a single model trained with the same compute as the entire population, demonstrating the potential of these distributed approaches. However, existing methods require a regular computation of the average model using an all-reduce operation, either to periodically remove any diversity in the population [157] or, in the case of PAPA, to compute an EMA

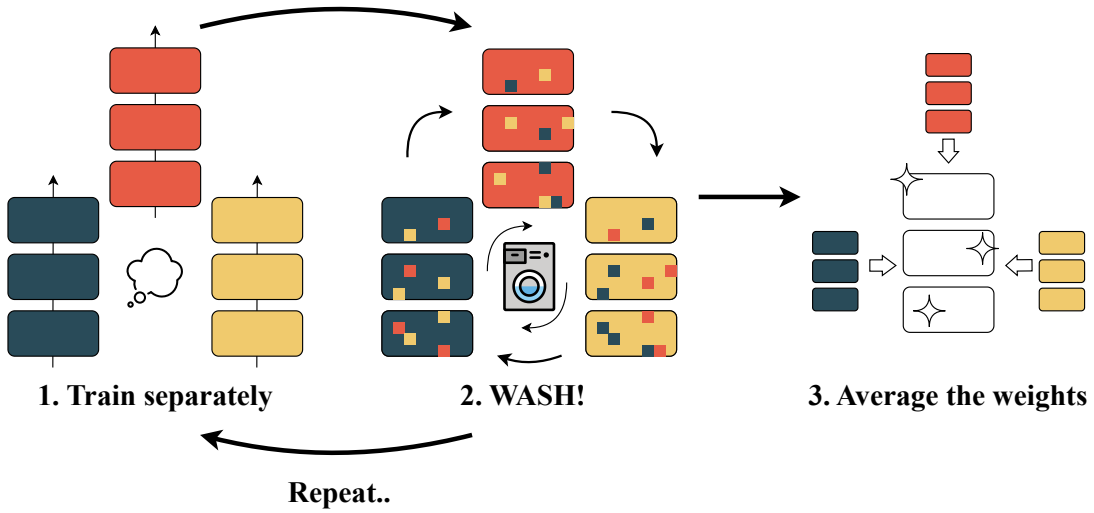


Figure 6.1: **Representation of training with WASH**. A population of models is being trained separately. **(1)** After each training step, **(2)** a small percentage of the parameters are permuted between models. **(3)** At the end of the training, the model weights are averaged, resulting in a high performance model.

of the average. This results in a high communication cost during the parallel training of the model population [281], which hampers the scalability of these approaches as the population size increases [276].

In this work, we propose a novel distributed method to train a population of models in parallel while keeping their weights within the same basin. It requires a fraction of the communication cost of PAPA but exhibits greater model diversity during training, increasing the final averaging accuracy. Our main idea is to shuffle parameters between models during training, forcing them to learn using the others' parameters. We refer to this idea as "parameter shuffling". A permutation is chosen randomly, and the models will communicate their parameters peer-to-peer according to the permutation. The use of a permutation is distinct from the notion of weight permutation of [3], which is within one model. We denote our method, which achieves **W**eight **A**veraging using parameter **SH**uffling, as **WASH**, and represent it schematically in Figure 6.1.

Contributions. Our contributions are as follows:

- We propose a novel method for the training of a population of models that can be weight-averaged, which we call WASH (**W**eight **A**veraging using parameter **SH**uffling). By shuffling a small number of parameters between models during training, the resulting population can be weight-averaged into a high-performance model for a fraction of the communication volume of methods such as PAPA.
- We find that WASH provides state-of-the-art results on image classification tasks, resulting in models with performance at the level of ensembling methods, while

requiring only a single network at inference time.

- We provide experiments to better understand the improvement provided by WASH, in particular how WASH implicitly reduces the distance between models in the population while preserving diversity.
- We perform different ablations of our method to show the impact of shuffling.

Our source code is available at: github.com/fournierlouis/wash.

Chapter organization. This chapter is structured as follows. We first motivate then introduce our weight shuffling method in Section 6.2. We then present our experimental framework and our main results in Section 6.3.1. We study more in-depth why our shuffling method result in an improvement over previous approaches in Section 6.3.2. Finally, we propose several ablations in Section 6.3.3 to better understand our shuffling approach.

6.2 Parameter shuffling in an ensemble for weight averaging

Motivation of our training procedure. We aim to balance the benefits of model ensembling with the computational efficiency of using a single model for inference via weight averaging. In other words, our objective is to produce a single model resulting from the ensembling. A set of N model parameters $\{\theta_n\}_{n\leq N} \subset \mathbb{R}^d$ are trained in parallel on the same dataset, with different data ordering and possibly different data augmentations and regularizations. To avoid divergence between the models, PAPA applies an EMA every T training steps and produces the following update

$$\tilde{\theta}_n \leftarrow \alpha\theta_n + (1-\alpha)\bar{\theta}\,, \tag{6.1}$$

where $\bar{\theta} \triangleq \frac{1}{N}\sum_{n=1}^{N}\theta_n$ represents the average of the model weights, also called the *consensus*, and $\alpha \in]0,1[$ is weighted according to the learning rate. Despite its advantages, this method has drawbacks, including the need for synchronized global communication across all models, which can be inefficient, and the potential reduction in model diversity due to the consensus constraint, which may reduce model expressiveness. Indeed, we observe that after each update

$$\sum_n \|\tilde{\theta}_n - \bar{\theta}\|^2 = \alpha^2 \sum_n \|\theta_n - \bar{\theta}\|^2 < \sum_n \|\theta_n - \bar{\theta}\|^2\,, \tag{6.2}$$

which shows that the EMA step of methods such as PAPA directly reduces the distance of the models from the consensus and hinders their diversity.

Proposed method: WASH. To address these challenges, we propose the following stochastic parameter shuffling step instead of the EMA, defined for each individual parameter $\theta_n^j \in \mathbb{R}$ of a model $\theta_n = [\theta_n^j]_{j=1}^d$ by

$$\hat{\theta}_n^i \leftarrow \begin{cases} \theta^i_{\pi_i(n)} & \text{with probability } p, \\ \theta^i_n & \text{otherwise,} \end{cases} \tag{6.3}$$

where π_j denotes a random permutation of the indices $\{1,...,N\}$, chosen uniformly at each iteration for each parameter index $j \in \{1,...,d\}$, and independently from the Bernoulli variable of Eq. (6.3). Notably, this parameter shuffling reduces in expectation to

$$\mathbb{E}[\hat{\theta}_n] = (1-p)\theta_n + p\bar{\theta}\,. \tag{6.4}$$

Thus, WASH aligns, in expectation, with the EMA of Eq. (6.1) for $p = (1-\alpha)$. The expected number of parameters communicated by each model at each step is thus $p \times d$ while for PAPA, each model communicating all of its parameters every T steps, this amounts to $\frac{d}{T}$. Thus, $p \ll \frac{1}{T}$ results in a significantly reduced communication overhead favorable to WASH. However, the model diversity is higher, because WASH preserves the consensus distance, as shown by

$$\sum_n \|\hat{\theta}_n - \bar{\theta}\|^2 = \sum_n \sum_j (\hat{\theta}_n^j - \bar{\theta}^j)^2 = \sum_j \sum_n (\theta_n^j - \bar{\theta}^j)^2 = \sum_n \|\theta_n - \bar{\theta}\|^2 . \tag{6.5}$$

Still, note that the following optimization step will affect the consensus distance, as we will see later.

Layer-wise adaptation via WASH. Recognizing that different network layers may require different levels of adaptation due to their roles and dynamics, we introduce a layer-specific probability adaptation. Assuming L layers in the network, for each layer l (where $0 \leq l < L$) we set

$$p_l = p\left(1 - \frac{l}{L-1}\right), \tag{6.6}$$

where p is a base probability. In other words, the parameters of the first layer have a shuffling probability of p, while the parameters of the last layer are never shuffled. This adaptation ensures that deeper layers, which are typically slower to train and more sensitive to the input features, undergo fewer permutations than the more generalizable early layers. This strategy not only preserves the specificity required by the early layers, but also cuts the overall communication overhead in half.

Full procedure. Algorithm 2 presents the training of a population of N models using WASH. Starting from the same initialization, our training procedure alternates between local gradient computation and shuffling communication. At inference, we simply average the weights of the models to obtain a single model with parameters $\bar{\theta}$. Note that techniques such as REPAIR [162] or activation alignment [3] could be incorporated to improve the alignment of the models, but we found them to be unnecessary to achieve high accuracy and kept our evaluation framework minimal for the sake of simplicity.

Algorithm 2 Training with WASH

Input: Datasets D_i, number of models N, initial parameters θ_0, training steps T, number of layers L, base probability p
Initialize parameters $(\theta_n)_n \leftarrow \theta_0$ and optimizers OPT_i
for $t = 1$ to T **do**
  # Training step
  for $n = 1$ to N, in parallel **do**
    $(x_n, y_n) \leftarrow D_n$ *# Sample data*
    $\theta_n \leftarrow \text{OPT}_n(x_n, y_n, \theta_n)$ *# Update the model n*
  # Shuffling step
  for layer $l = 0$ to $L - 1$ **do**
    for parameter θ^j in layer l **do**
      With probability $p(1 - \frac{l}{L-1})$,
        $\pi_j \leftarrow$ Random permutation
        $(\theta^j_n)_n \leftarrow (\theta^j_{\pi_j(n)})_n$ *# Send and permute the parameter*
Output: the averaged model $\frac{1}{N}\sum_{n=1}^{N} \theta_n$

6.3 Numerical experiments

Training methods. We present the capabilities of WASH for training a population of neural networks on standard image classification tasks. As a Baseline, we consider a population trained separately, with each model working on a different dataset order and different data augmentations and regularization (if they are used). This is the same baseline as [161], only starting from the same initialization, but we found that this change had no significant impact on performance. We also compare WASH to PAPA [161] on the same tasks (with PAPA however using models with a different initialization), to show our improvement despite requiring a fraction of the communication cost. We do not provide comparisons with DART [157] or the variants of PAPA as their performances are generally inferior [161]. We also propose a variant of WASH called WASH+Opt, which also permutes the optimizer state associated with the shuffled parameter (in our case, the momentum of SGD), doubling the communication volume. For simplicity, we do not permute or recompute the running statistics of the BatchNorm layers.

Communication cost. Training with PAPA requires computing an all-reduce operation on all of the model parameters every $T = 10$ training steps. In comparison, WASH requires, in expectation, a shuffling of $p/2$ of the model parameters at each training step. Thus, by keeping a base probability $p \leq 0.2$, WASH results in a more communication-efficient training. In practice, in our experiments, p will be 0.001 or 0.05, ensuring a reduction in communication volume of 200 or 4.

Table 6.1: **Communication volume and inference costs** of four training techniques. The baseline Ensemble is trained separately, but requires a linearly increasing inference cost. In our experiments, we set the base probability of WASH and WASH+Opt to 0.001 and 0.05, respectively, when training on CIFAR-10/100 or ImageNet, resulting in a reduction in communication volume compared to PAPA.

	Communication volume		
Technique	**CIFAR-10/100**	**ImageNet**	**Inference cost**
Ensemble	0	0	N
PAPA	1	1	1
WASH	$1/200$	$1/4$	1
WASH+Opt	$1/100$	$1/2$	1

Evaluation strategy. After training, the resulting population of models obtained can be evaluated in three different ways. As a baseline, the performance of the population can be evaluated as an Ensemble, averaging the predictions of the models. The parameters of the models can be averaged to obtain a single model, which we refer to as Averaged. This is equivalent to UniformSoup in [376] or AvgSoup in [161] for example. More elaborate averaging methods have been proposed, such as GreedySoup [376], which averages an increasing number of models (in order of validation accuracy) until averaging no longer improves accuracy. We report the accuracy of the Ensemble and Averaged model for all training techniques, as well as the GreedySoup accuracy of the Baseline. As in [161], we find that the GreedySoup accuracy corresponds to the accuracy of a single model for the Baseline and that the Averaged model accuracy outperforms the GreedySoup model for the other techniques, and thus chose not to report it. We summarize in Table 6.1 the communication volume and inference costs required to train a separate Ensemble of models, or to train with PAPA, WASH, or WASH+Opt.

6.3.1 Main experiments

Experimental setup. We showcase the performance of WASH for training neural networks on image classification tasks on the CIFAR-10, CIFAR-100 [183], and ImageNet [76] datasets. We use the same training framework as [161] for a fair comparison. We train a population of N models for $N \in \{3, 5, 10\}$, on the ResNet-18, 50 and VGG-16 architectures. 2% of the training data is kept as validation for computing the GreedySoup. As in [161], we consider one framework with heterogeneous models, learning with different data augmentations and regularizations, and one homogeneous setting with no data augmentations, where the only difference between the models' trainings is the dataset shuffling. Details are presented in the Appendix. The models are trained with SGD with momentum, a weight decay of 10^{-4}, and a cosine annealing scheduler with initial and minimum learning rates of 0.1 and 10^{-4}. For CIFAR-10/100, we train over 300 epochs with a batch size of 64, and 90 epochs with a batch size of 256 for ImageNet. For

Table 6.2: **Ensemble and Averaged Model accuracy for a heterogeneous population of models; trained with different data augmentations and regularizations.** We compare models trained separately (Baseline), with PAPA, or with our method WASH and its variant WASH+Opt. We also report the GreedySoup accuracy for the Baseline models. The best Ensemble (black) and Averaged (blue) accuracy are reported in bold. Except on CIFAR-10, WASH and in particular WASH+Opt provide the best performance for the final Averaged Model, with performances comparable to the Ensemble of models for a fraction of the inference cost.

Method		**Baseline (trained separately)**			**PAPA**		**WASH (ours)**		**WASH+Opt (ours)**	
Config	**#N**	Ensemble	Averaged	GreedySoup	Ensemble	Averaged	Ensemble	Averaged	Ensemble	Averaged
CIFAR-10										
VGG-16	3	95.98±.42	10.00±.00	95.26±.05	**96.12±.34**	**96.13±.24**	95.89±.23	95.97±.24	95.91±.36	95.85±.27
	5	**96.28±.40**	10.00±.00	95.42±.10	96.24±.17	**96.21±.13**	96.15±.10	**96.20±.10**	96.00±.21	96.04±.14
	10	**96.47±.07**	10.00±.00	95.39±.24	96.32±.13	**96.31±.13**	96.27±.10	96.18±.13	96.14±.08	96.20±.05
ResNet18	3	97.15±.28	10.17±.29	96.62±.38	**97.33±.05**	**97.24±.05**	97.21±.19	97.19±.17	97.22±.07	**97.25±.14**
	5	**97.33±.08**	10.09±.16	96.61±.03	**97.35±.12**	**97.31±.06**	97.21±.10	97.25±.12	97.18±.09	97.16±.07
	10	**97.59±.01**	9.26±1.28	96.79±.14	97.39±.13	**97.34±.06**	97.30±.10	97.28±.04	97.20±.13	97.16±.13
CIFAR-100										
VGG-16	3	**80.36±.15**	1.00±.00	77.92±.22	78.89±.10	78.77±.16	79.10±.88	79.05±.68	79.15±.61	**79.15±.41**
	5	**81.32±.56**	1.00±.00	77.81±.25	79.51±.38	79.24±.43	79.65±.27	79.39±.21	79.75±.21	**79.71±.20**
	10	**82.24±.15**	1.00±.00	77.83±.65	79.95±.11	79.64±.13	80.05±.18	79.70±.25	80.03±.11	**79.76±.13**
ResNet18	3	**82.84±.48**	1.00±.01	80.06±1.5	81.58±.12	81.53±.13	81.91±.34	81.90±.36	81.99±.06	**82.08±.09**
	5	**83.72±.49**	1.00±.00	80.72±.52	82.09±.30	82.01±.34	82.16±.42	81.97±.28	82.35±.17	**82.17±.15**
	10	**84.18±.20**	1.00±.00	80.61±.43	82.32±.09	82.15±.14	82.43±.32	**82.31±.38**	82.42±.31	82.18±.22
ImageNet										
ResNet50	3	**76.16±.28**	0.10±.00	74.15±.11	75.62±.15	10.32±2.4	74.39±.14	**74.34±.18**	74.30±.22	74.18±.26
	5	**76.68±.06**	0.10±.00	74.47±.06	75.80±.21	0.13±0.04	74.63±.11	**74.59±.07**	74.44±.21	74.39±.21

WASH and WASH-Opt we initialize the models with the same parameters and choose p with cross-validation to be equal to 0.001 or 0.05 when training on CIFAR-10/100 or ImageNet. We do not require any alignment technique such as REPAIR.

Main results. Table 6.2 and Table 6.3 correspond to the heterogeneous and homogeneous settings, respectively. We report the test accuracies as the average of 3 runs for the Ensemble of models, the Averaged model, and the GreedySoup for the Baseline (equivalent to the best model). Consistent with the findings of [161], we find that networks trained separately have a high Ensemble accuracy, but perform as random when averaged. On CIFAR-10/100, methods like PAPA and WASH result in lower Ensemble accuracy but almost no difference between the Ensemble and Averaged accuracies. In general, WASH and WASH+Opt outperform PAPA, even though they require less communication. On ImageNet, our parallelization procedure results in a slightly lower Baseline accuracy and we were not able to reproduce PAPA's baseline, possibly due to a mistake in their reported hyperparameters. See the Appendix for experiments on ImageNet32x32. The WASH Averaged model achieves high accuracy, like previously. Both of our methods reduce the gap with the accuracies of the baseline Ensemble, indicating that WASH hinders less the diversity of the population of models while maintaining

Table 6.3: **Ensemble and Averaged Model accuracy for a homogeneous population of models.** We compare models trained separately (Baseline), with PAPA, or with our methods WASH and WASH+Opt. The best Ensemble (black) and Averaged (blue) accuracy are reported in bold. We observe the same results in this setting, with WASH in particular coming close to the Ensemble performance.

Method		Baseline (trained separately)			PAPA		WASH (ours)		WASH+Opt (ours)	
Config	#N	Ensemble	Averaged	GreedySoup	Ensemble	Averaged	Ensemble	Averaged	Ensemble	Averaged
CIFAR-10										
VGG-16	3	**94.93±.06**	10.00±.00	93.60±.41	94.38±.14	94.34±.18	94.41±.23	**94.58±.17**	94.45±.05	94.47±.02
	5	**95.29±.05**	10.00±.00	93.82±.30	94.55±.12	94.58±.12	94.72±.08	**94.70±.17**	94.63±.11	**94.68±.14**
	10	**95.23±.06**	10.00±.00	93.82±.06	94.79±.18	**94.78±.20**	94.66±.03	94.54±.07	94.71±.07	94.61±.13
ResNet18	3	**96.14±.10**	10.00±.00	95.42±.27	95.89±.04	**95.89±.06**	95.77±.12	95.77±.17	95.85±.04	**95.87±.10**
	5	**96.19±.16**	10.00±.00	95.31±.09	95.99±.08	**95.99±.08**	95.96±.08	**95.98±.05**	95.94±.12	**95.98±.12**
	10	**96.34±.02**	10.00±.00	95.26±.11	96.10±.25	**96.11±.24**	96.08±.07	**96.12±.09**	96.07±.07	96.08±.14
CIFAR-100										
VGG-16	3	**77.63±.24**	1.00±.00	73.76±.35	75.10±.11	75.09±.16	76.30±.37	**76.04±.58**	76.04±.03	75.96±.18
	5	**78.52±.10**	1.00±.00	73.76±.18	75.56±.16	75.55±.14	76.63±.27	**76.48±.23**	76.64±.15	76.13±.18
	10	**79.26±.06**	1.00±.00	73.99±.26	76.24±.44	76.26±.43	77.06±.12	**76.43±.18**	76.72±.15	75.94±.26
ResNet18	3	**79.54±.17**	1.00±.00	76.84±.54	77.83±.26	77.86±.30	78.90±.17	**78.76±.25**	78.66±.08	78.56±.21
	5	**80.11±.23**	1.00±.00	76.83±.45	77.94±.16	77.92±.19	79.24±.32	79.09±.43	79.32±.19	**79.19±.15**
	10	**80.55±.13**	1.00±.00	76.80±.41	78.40±.15	78.44±.22	79.65±.17	**79.43±.16**	79.34±.34	79.19±.45

weight averagability. However, a gap still remains, which may be inherent to the models being in the same basin. WASH and WASH+Opt have very similar results, with the simpler WASH being better in the homogeneous case and WASH+Opt being better in the heterogeneous case.

6.3.2 Why do shuffling parameters help?

In this section, we propose to explain the improvement provided by our parameter shuffling over previous mechanisms such as BTM, DART or PAPA, which focus on parameter averaging. First, we show that models trained with WASH have a smaller distance to consensus than models trained separately. We then argue that, despite this, WASH is a weak perturbation on the training of the models and that it induces diversity in the models.

Reducing distance to consensus. To better analyse the diversity of the models trained with WASH, we propose to report the distance of the models to the consensus (the averaged model) during training, as a proxy for the diversity metric. [151, 377] showed that the difference between the Ensemble and the Averaged models depends on the distance between the models. We present in Figure 6.2 the average distance of the models to the consensus, for models trained separately, with PAPA, PAPA-all, or with WASH. PAPA-all is a variant of PAPA that is functionally identical to DART. The idea is to average the weights every few epochs before allowing the models to diversify again. We observe that WASH results in a consistently lower distance to consensus than the baseline, even though it explicitly leaves the distance to consensus unchanged during

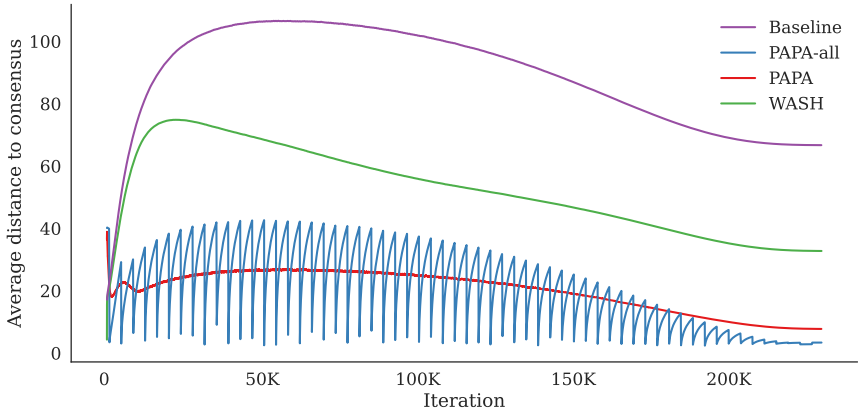


Figure 6.2: **Average distance to the consensus (i.e. the averaged model)** during training for a heterogeneous population of 5 models trained on CIFAR-100, either separately, with PAPA, PAPA-all, or our method WASH. Starting from consensus, the models initially diverge from each other before converging back again during convergence, mainly due to weight decay. Models trained with WASH have a smaller distance to consensus than those trained separately, allowing them to be averaged without loss of performance. By training with PAPA-all (i.e., averaging to a single model every few epochs), the models are not able to reach the same diversity as WASH between these averaging steps. Finally, the EMA of PAPA has a strong pulling effect towards consensus, resulting in a distance similar to that of PAPA-all. The wiggle in the curve is due to the immediate reduction in distance caused by the EMA steps.

the shuffling step, and only shuffles a small number of parameters. Thus, the smaller distance at the end of the training explains why the averaging of the parameters does not lead to a decrease in performance. In comparison, PAPA-all (i.e. DART) results in alternating phases where the models diversify before being averaged, and we observe that the models are not able to reach the diversity of WASH. Similarly, the EMA of PAPA has a strong pulling effect and results in an average diversity similar to that of PAPA-all. Thus, we find that models trained with WASH have a higher diversity than models trained with PAPA or PAPA-all, while being close enough that averaging them does not cause a loss in performance. More generally, we show in Figure D.1 of the Appendix that different interpolations of models trained with WASH result in a similar performance, demonstrating that they all lie in the same loss basin.

Encouraging diversity. WASH can be considered as a weak perturbation of the models: parameter shuffling affects the models less than parameter averaging or the EMA of PAPA, since only a few parameters are affected at a time and the consensus distance is unaffected. Furthermore, parameter shuffling increases the diversity of trajectories seen by the models. We illustrate this with a toy example where two points are jointly trained with SGD on a 2D loss function with 2 local minima and 1 global minimum, either

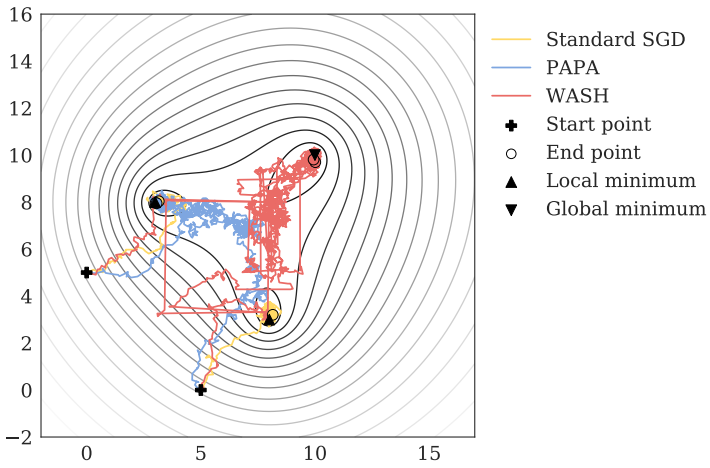


Figure 6.3: **2D optimization example.** We train 2 points with SGD on a simple loss function with 2 local and 1 global minima (up and down triangles). The two models are trained from two different starting points (plus signs). When the points are trained separately (yellow), they converge to their closest local minimum (yellow circles). When trained with PAPA (blue), the points reach a consensus but then converge to one of the local minima (blue circles). When trained with WASH (red), the shuffling (seen by the horizontal and vertical lines in the trajectory) allows for more diversity in the optimization path, and the points both reach the global minimum (red circles).

separately, with PAPA, or with WASH. The trajectories corresponding to each method are shown in Figure 6.3. Training the two points separately causes them to converge to a separate local minimum (i.e. a different basin). Training with PAPA allows the two points to reach a consensus, but they converge together to a local minimum. In contrast, by training with WASH, we show that both points reach the global minimum, as the shuffling allows for a greater diversity of points to optimize with. We provide more details in the Appendix.

6.3.3 Ablations

In this section, we present ablations to better understand the effect of the parameter shuffling, varying the layer-wise probability adaptation, the base probability value, and the shuffling period.

Layer-wise adaptation variations. For WASH, we found that decreasing probability with depth gave the best results. We show in Table D.1 of the Appendix the performances for alternatives where the probability either remains constant or increases with depth. We find lower performances for both alternatives. In Figure 6.4 we show the distances of the models to the consensus for all three schedules. More specifically, we report the distances for different slices of the models' parameters, showing the effect of shuffling

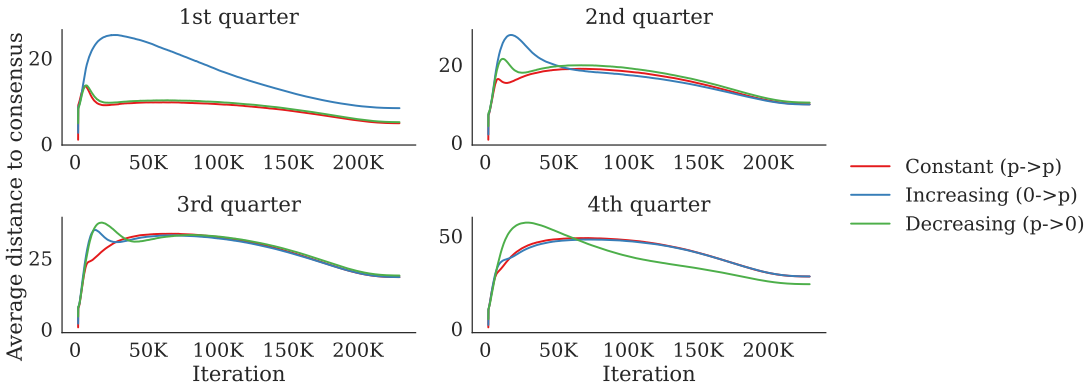


Figure 6.4: **Average distance to the consensus for different layer-wise adaptations of WASH**, for different slices of the model parameters. Keeping the probability constant across layers ensures the lowest distance to consensus for the first quarters. Surprisingly, in the last quarter of parameters, the 'decreasing probability' adaptation, despite starting with a higher distance to consensus, shows a lower distance to consensus later in training; even though shuffling is less frequent than in the other schedules. The 'increasing probability' adaptation shows how early layers are useful for shuffling.

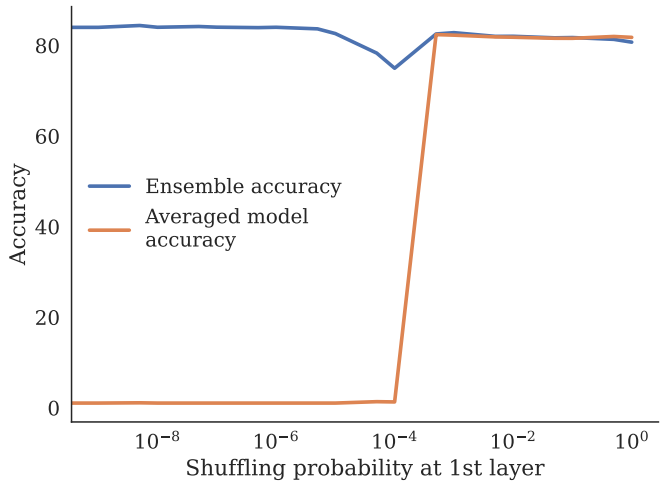


Figure 6.5: **Ensemble and Averaged accuracy for varying base probability values.** We observe a phase transition as the base probability increases between a phase where permuting does not improve the averaged model accuracy and a phase where the ensemble accuracy is equal to the averaged model accuracy. Between the phases, the ensemble accuracy decreases.

as a function of depth. As predicted, shuffling all layers equally results in the lowest distance to the consensus, except for the last quarter of parameters. Here, surprisingly, our base 'decreasing' adaptation shows a lower distance to the consensus despite less

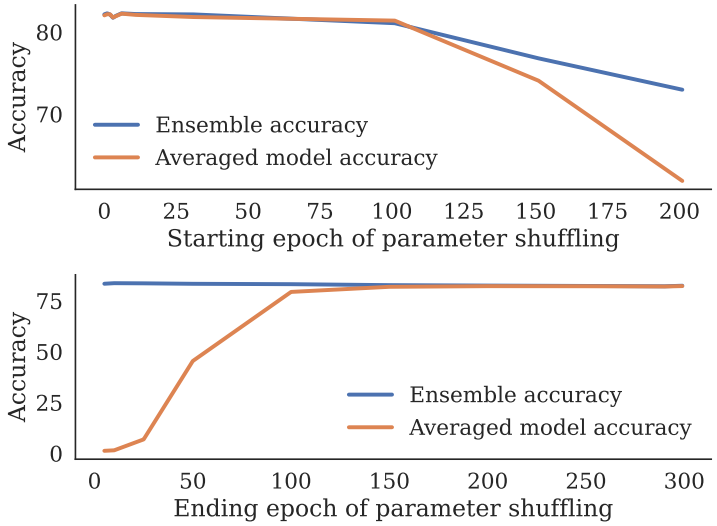


Figure 6.6: **Ensemble and Averaged accuracy depending on the starting or ending epoch of the shuffling.** The parameter shuffling is beneficial both at the beginning and at the end of training. Note that ending early, at epoch 150 out of 300, has less impact on performance than starting permuting at epoch 150, showing that WASH is more important early in training.

frequent shuffling. We also observe a particularly strong effect of the shuffling for the early layers, as the distance in the first quarter is more pronounced between the 'increasing' curve and the others.

Base probability variation. We present in Figure 6.5 the Ensemble and Averaged for different values of p, the base shuffling probability of the first layer. Rather than a smooth increase in the accuracy of the Averaged model, we observe a phase transition between a phase where the accuracy of the Averaged model is not improved by the shuffling and a sudden increase in the accuracy where it reaches the accuracy of the Ensemble. Just before the transition, the accuracy of the Ensemble decreases, before increasing again back to its previous performance. The accuracy decreases only slightly even when the shuffling probability is increased to 1, indicating the resilience of the models to heavy shuffling.

Shuffling is beneficial at every step. Finally, we propose to show the impact of the parameter shuffling at different steps of the training by varying the epoch at which the shuffling either starts or stops. In Figure 6.6, we show that there is no improvement by having a warmup or slowdown period in parameter shuffling, indicating that all phases of the training are improved by WASH. Furthermore, stopping parameter shuffling early results in a much smaller loss of Averaged accuracy compared to starting shuffling late. In other words, shuffling at the beginning of training before the models start to converge

is more impactful as the models may still reside in different loss basins.

Conclusion

We proposed a novel distributed training method, WASH, which aims to train a population of models in parallel. These models are averaged at the end of training to obtain a high performance model with accuracies close to the ensemble accuracy for a fraction of the inference cost. Our method requires a fraction of the communication cost of similarly performing techniques, while achieving state-of-the-art results for our weight-averaged models. We show that our novel parameter shuffling does not explicitly reduce the distance between models while increasing the diversity of optimization paths seen by the population. Nevertheless, we find that the distance between our models is smaller than if they were trained separately, allowing them to be averaged at the end of training.

Acknowledgements

This work was supported by Project ANR-21-CE23-0030 ADONIS, EMERG-ADONIS from Alliance SU, and Sorbonne Center for Artificial Intelligence (SCAI) of Sorbonne University (IDEX SUPER 11-IDEX-0004). This work was granted access to the AI resources of IDRIS under the allocations 2023-A0151014526 made by GENCI.

Chapter 7

Conclusion

7.1 Contributions

In this thesis, we investigated parallelizable approaches to DL training that deviate from the standard paradigm of parallelized mini-batch SGD with backpropagation, and summarize our contributions in Figure 7.1.

We first focused on model parallel approaches, particularly on local learning.

We studied the local learning framework for training DNNs using contrastive self-supervised training objectives. We found that such approaches also suffer from information collapse, similar to the supervised case. In our case, we showed that this collapse can be measured by observing the dimensionality of activations in the network. We also showed, using an oracle model, that a simple subsampling of the examples used by the local losses is sufficient to greatly reduce this collapse and thus approach the performance of backpropagation. This idea, motivated by a linear model, led to a simple and novel method to regularize the local training, by sampling examples depending on their local feature similarity. We confirmed that this method remedies the dimensional collapse in contrastive local learning and reduces the accuracy loss due to decoupling, without requiring external information.

We then extended our interest to a backpropagation alternative that is not local but requires only one forward pass. This approach was motivated by the contrast between local learning, a method that provides biased but fast local gradients, and forward-mode AD, which provides forward gradients, an approximation of the backpropagation gradient with a high variance. We conducted an extensive study of the forward gradient approach by varying the approximated gradient, which can come from the end-to-end loss, local losses, or an intermediary; as well as the tangent direction used for computation, using random isotropic directions, NTK gradients, or local loss gradients. We also tested a variety of feedback insertion points, image datasets, and model sizes. We found that the local gradients outperformed all other tangent directions, despite their limited alignment with the end-to-end gradients. However, by keeping the end-to-end objective

as the one to be approximated, we found that our approach could reduce the gap with backpropagation.

We then focused on the other side of parallel training, the distributed approaches.

We introduced a novel synchronous distributed training paradigm for homogeneous clusters, called Cyclic Data Parallelism (CDP). In the standard Data Parallelism (DP) approach, the total memory used by the models peaks at the end of the forward pass and each communicated gradient requires a collective operation at the end of the backward pass. Compared to DP, we found that we could balance both gradient communication and the total memory used by activations. Our idea was to change the execution of the forward and backward passes of the micro-batches of data from simultaneous to cyclic. CDP can be applied to many complementary frameworks to DP. On a single device, it can reduce the total memory used by DP. On multiple devices, the most common distributed framework, it balances the communication during the training step. When used with the ZeRO-DP framework, our cyclic paradigm allows model states to remain on only a single device at a time. When added to a model parallel framework, this can result in a more GPU-efficient method than even pipeline parallelism, reducing the number of GPUs required. Despite requiring delayed gradients with a delay inferior to one training step, we find that our novel update rule performs similarly to the learning rule of standard DP.

At the other end of distributed approaches, we focused on a training framework that does not approximate parallelized backpropagation, but trains an ensemble of model replicas separately to promote diversity. We focused on a communication-efficient method that allows such models to be trained in parallel while keeping them in the same loss basin. The goal is to balance their diversity, which improves the ensemble accuracy of the resulting population of models, and their averagability, i.e., whether they can be averaged into a final higher-performing network. Initially inspired by the shuffling of activations in a model parallel framework, we found that it is sufficient to shuffle a very small fraction of parameters between models to keep them averagable. Our resulting novel distributed approach, called WASH, leads to state-of-the-art performance for the final model while requiring only a fraction of the communication volume of similar techniques. Our method does not explicitly reduce the diversity of models, while encouraging more diverse optimization paths.

7.2 Perspectives

Will DNNs continue growing ?

The motivation for this thesis, the ever-increasing computational requirements for training DNNs, may change in the coming years depending on several factors. Huang's Law, which shows that GPUs and TPUs double their performance every year, is still the main reason for training deeper networks every year. There may come a point where this law slows down, making faster computations harder to achieve. Similarly,

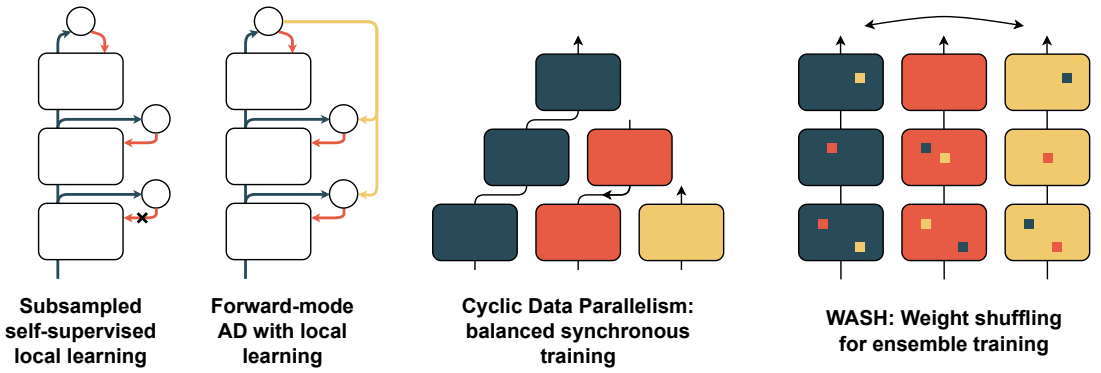


Figure 7.1: **Summary of our four contributions.** Our first two contributions provide novel MP approaches related to local learning, either by subsampling contrastive local losses or by using forward-mode AD to mix local and global feedback. Our last two contributions provide novel distributed approaches, either for synchronous learning as in data parallel, or the low-communication ensemble training framework.

improvements in recent models may slow as they reach human-like levels, or as the amount of training data left to scrap is exhausted [354]. In this case, research will need to evolve in a direction orthogonal to the one it is currently pursuing, which is to follow the 'bitter lesson', i.e., growing all training components for better performance [340]. Such directions may be models with better world models [199] or access to memory or agent capacities [402] for example. As models grow into more general roles, another approach could focus on using local learning to only train separate subnetworks on specific tasks.

Other alternative learning approaches

We have discussed some other efficient learning approaches in the introduction to this thesis. Refinements in the datasets used or the use of hardware (such as FlashAttention [73]) will further improve the training speed of the standard approaches in the coming years. Possible alternative training schemes, such as using sparsity [139], matrix-multiplication-free methods [408], or further reduction of weights to just a few bits [224] will also continue to be explored. However, it is difficult to predict their use, since consistency and usability are important factors in the adoption of such paradigms. The current focus of the DL community on possible changes to the learning framework continues to be on novel model architectures. For example, Mixture of Experts models reduce inference costs, which is critical as models grow. Most importantly, finding alternatives to the Transformers architecture remains an important goal. Despite its success and modularity, the attention mechanism and its quadratic cost become an obstacle for both training and inference. In particular, the problem of context size has become an issue in recent years, as many applications require knowledge of information

very far apart in the data [363]. Thus, it is not clear whether future foundation models will continue to use its architecture in the future, or whether more efficient alternatives such as state-space models will surpass it [101]. As it stands, it seems difficult to imagine a major paradigm shift in the current training framework, as long as refinements and size increases still provide predictable improvements for the time being. Still, we think that local learning may have a role to play in the future, as we discuss now.

Local learning

Two of our contributions aimed at reducing the gap with backpropagation, either by regularizing the local losses, or by using forward gradients to receive a global feedback. Despite this, we found that local learning approaches are still outperformed by backpropagation, which is logical. Joint training of the target objective yields better results than each layer training auxiliary objectives with no guarantee on the following representations. However, this may not be a problem in future DL training approaches, especially in the foundation model era. Minimizing the self-supervised training objective is arguably a secondary goal in such models. The self-supervised objective is primarily a way for these models to memorize and predict data, rather than to perform a precise task such as classification. Indeed, they are several fine-tuning steps after the initial training of such networks. Thus, the use of novel local losses that could serve a similar purpose may have no major impact on the resulting trained network. The best example of this idea remains the brain. It most likely does not follow a rigid training objective [216], and may also mainly perform inference and learning according to the neuroscientific principle of free energy [246]. The most interesting idea of a future possible DNN architecture, though still improbable, might therefore be a decentralized deep reservoir network composed of many neurons, almost entirely connected with feed-forward connections and local learning rules [364]. Such an approach would be more biologically plausible, while being able to perform the task required by foundation models, which is mainly memory and prediction.

However, the issue of alignment between local and end-to-end gradients cannot be completely dismissed, if the goal remains to effectively minimize the training objective. Some regularization approaches can improve this alignment, but to truly approximate backpropagation, some form of feedback is needed. Several ideas are possible: using intermittent backpropagation passes to obtain 'true' gradients, similar to [155], locally aligning the gradients between successive stages, meta-learning local training objectives that align with the end-to-end gradients... However, there will be a trade-off between a better approximation of the backpropagation gradient and the need for more backward connections and more computation. If this trade-off becomes too expensive to obtain a good approximation, it may defeat the purpose of a local learning approach altogether.

Distributed learning

Distributed learning is likely to remain the most ubiquitous and important form of parallelism in the coming years, especially when combined with other parallelism

frameworks such as pipeline, sequence, and tensor parallelism. On clusters, works such as what we proposed in CDP or ACCO will continue to explore trade-offs between exact mini-batch gradient computation and delayed gradients, communication overlap, and consensus breaking, especially as the energy cost of communication becomes more important. More heterogeneous clusters may explore asynchronous learning, which has seen a resurgence in recent years. This is also likely to be related to the recent interest in model merging, as methods such as WASH can lead to much better performing models than synchronous training, while requiring less communication. This may be of particular interest when computations are performed on large decentralized networks of heterogeneous devices with very limited communication. Such networks may emerge as a counterweight to the centralized private clusters that are currently most common today.

Bibliography

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. "A learning algorithm for boltzmann machines". In: *Cognitive Science* 9.1 (1985), pp. 147–169. ISSN: 0364-0213. DOI: https://doi.org/10.1016/S0364-0213(85)80012-4. URL: https://www.sciencedirect.com/science/article/pii/S0364021385800124.

[2] Alekh Agarwal and John C Duchi. "Distributed Delayed Stochastic Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/f0e52b27a7a5d6a1a87373dffa53dbe5-Paper.pdf.

[3] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. *Git Re-Basin: Merging Models modulo Permutation Symmetries*. 2022. eprint: arXiv:2209.04836.

[4] Mohamed Akrout et al. *Deep Learning without Weight Transport*. 2020. arXiv: 1904.05391 [cs.LG].

[5] Ibrahim M Alabdulmohsin, Behnam Neyshabur, and Xiaohua Zhai. "Revisiting neural scaling laws in language and vision". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 22300–22312.

[6] Nick Alonso et al. *A Theoretical Framework for Inference Learning*. 2022. arXiv: 2206.00164 [cs.NE].

[7] Shunichi Amari. "A Theory of Adaptive Pattern Classifiers". In: *IEEE Transactions on Electronic Computers* EC-16.3 (1967), pp. 299–307. DOI: 10.1109/PGEC.1967.264666.

[8] Ehsan Amid, Rohan Anil, and Manfred K. Warmuth. *LocoProp: Enhancing BackProp via Local Loss Optimization*. 2022. arXiv: 2106.06199 [cs.LG]. URL: https://arxiv.org/abs/2106.06199.

[9] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ASPLOS '24. USA: Association for Computing

Machinery, 2024, pp. 929–947. ISBN: 9798400703850. DOI: 10.1145/3620665.3640366. URL: https://doi.org/10.1145/3620665.3640366.

[10] Quentin Anthony et al. "Blackmamba: Mixture of experts for state-space models". In: *arXiv preprint arXiv:2402.01771* (2024).

[11] Lynton Ardizzone et al. *Analyzing Inverse Problems with Invertible Neural Networks*. 2019. arXiv: 1808.04730 [cs.LG].

[12] Lynton Ardizzone et al. *Guided Image Generation with Conditional Invertible Neural Networks*. 2019. arXiv: 1907.02392 [cs.CV].

[13] Armin Askari et al. *Lifted Neural Networks*. 2018. arXiv: 1805.01532 [cs.LG].

[14] Mahmoud Assran et al. "Stochastic Gradient Push for Distributed Deep Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 344–353.

[15] Florian Bacho and Dominique Chu. *Low-Variance Forward Gradients using Direct Feedback Alignment and Momentum*. 2023. arXiv: 2212.07282 [cs.LG].

[16] Vijay Balasubramanian. "Brain power". en. In: *Proc Natl Acad Sci U S A* 118.32 (Aug. 2021).

[17] David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. *Kickback cuts Backprop's red-tape: Biologically plausible credit assignment in neural networks*. 2014. arXiv: 1411.6191 [cs.LG].

[18] Saar Barkai, Ido Hakimi, and Assaf Schuster. *Gap Aware Mitigation of Gradient Staleness*. 2020. arXiv: 1909.10802 [cs.LG].

[19] Sergey Bartunov et al. "Assessing the scalability of biologically-motivated deep learning algorithms and architectures". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9389–9399. arXiv: 1807.04587 [cs.LG].

[20] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: *Journal of Marchine Learning Research* 18 (2018), pp. 1–43.

[21] Atılım Güneş Baydin et al. "Gradients without Backpropagation". In: *arXiv preprint arXiv:2202.08587* (2022).

[22] Jens Behrmann et al. "Invertible Residual Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 573–582. URL: http://proceedings.mlr.press/v97/behrmann19a.html.

[23] Mohamed Ishmael Belghazi et al. *MINE: Mutual Information Neural Estimation*. 2021. arXiv: 1801.04062 [cs.LG].

[24] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. "Decoupled Greedy Learning of CNNs". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 736–745. URL: https://proceedings.mlr.press/v119/belilovsky20a.html.

[25] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. "Greedy Layerwise Learning Can Scale To ImageNet". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 583–593. URL: https://proceedings.mlr.press/v97/belilovsky19a.html.

[26] Eugene Belilovsky et al. *Decoupled Greedy Learning of CNNs for Synchronous and Asynchronous Distributed Learning*. 2021. arXiv: 2106.06401 [cs.LG].

[27] Gabriel Belouze. *Optimization without Backpropagation*. 2022. DOI: 10.48550/ARXIV.2209.06302. URL: https://arxiv.org/abs/2209.06302.

[28] Yoshua Bengio. *Deriving Differential Target Propagation from Iterating Approximate Inverses*. 2020. arXiv: 2007.15139 [cs.LG].

[29] Yoshua Bengio. *How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation*. 2014. arXiv: 1407.7906 [cs.LG].

[30] Yoshua Bengio and Asja Fischer. *Early Inference in Energy-Based Models Approximates Back-Propagation*. 2016. arXiv: 1510.02777 [cs.LG].

[31] Yoshua Bengio et al. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[32] Souhail Bennani et al. "Using AI to improve radiologist performance in detection of abnormalities on chest radiographs". In: *Radiology* 309.3 (2023), e230860.

[33] Gregory Benton et al. "Loss surface simplexes for mode connecting volumes and fast ensembling". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 769–779.

[34] G Q Bi and M M Poo. "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type". en. In: *J Neurosci* 18.24 (Dec. 1998), pp. 10464–10472.

[35] E L Bienenstock, L N Cooper, and P W Munro. "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex". en. In: *J Neurosci* 2.1 (Jan. 1982), pp. 32–48.

[36] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. "Multipath sparse coding using hierarchical matching pursuit". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 660–667.

[37] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258 [cs.LG]. URL: https://arxiv.org/abs/2108.07258.

[38] Léon Bottou. "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[39] Léon Bottou, Frank E Curtis, and Jorge Nocedal. "Optimization methods for large-scale machine learning". In: *SIAM Review* 60.2 (2018), pp. 223–311.

[40] Lucía Bouza, Aurélie Bugeau, and Loïc Lannelongue. "How to estimate carbon footprint when training deep learning models? A guide and review". In: *Environmental Research Communications* 5.11 (2023), p. 115014.

[41] Leo Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1007/BF00058655. URL: https://doi.org/10.1007/BF00058655.

[42] Wieland Brendel and Matthias Bethge. "Approximating cnns with bag-of-local-features models works surprisingly well on imagenet". In: *arXiv preprint arXiv:1904.00760* (2019).

[43] Andrew Brock et al. *FreezeOut: Accelerate Training by Progressively Freezing Layers*. 2017. arXiv: 1706.04983 [stat.ML].

[44] Ethan Caballero et al. *Broken Neural Scaling Laws*. 2023. arXiv: 2210.14891 [cs.LG]. URL: https://arxiv.org/abs/2210.14891.

[45] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. "Deep Blue". In: *Artif. Intell.* 134.1–2 (Jan. 2002), pp. 57–83. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(01)00129-1. URL: https://doi.org/10.1016/S0004-3702(01)00129-1.

[46] Xuanyu Cao et al. "Communication-Efficient Distributed Learning: An Overview". In: *IEEE Journal on Selected Areas in Communications* 41.4 (2023), pp. 851–873. DOI: 10.1109/JSAC.2023.3242710.

[47] Gail A Carpenter. "Neural network models for pattern recognition and associative memory". In: *Neural networks* 2.4 (1989), pp. 243–257.

[48] Miguel Á. Carreira-Perpiñán and Weiran Wang. *Distributed optimization of deeply nested systems*. 2012. arXiv: 1212.5921 [cs.LG].

[49] Junbum Cha et al. "Swad: Domain generalization by seeking flat minima". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 22405–22418.

[50] Souradip Chakraborty, Aritra Roy Gosthipaty, and Sayak Paul. "G-SimCLR: Self-supervised contrastive learning with guided projection via pseudo labelling". In: *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2020, pp. 912–916.

[51] Ernie Chan et al. "Collective Communication: Theory, Practice, and Experience". In: *Concurrency and Computation: Practice and Experience* 19 (Sept. 2007), pp. 1749–1783. DOI: 10.1002/cpe.v19:13.

[52] Bo Chang et al. *Reversible Architectures for Arbitrarily Deep Residual Neural Networks*. 2017. arXiv: 1709.03698 [cs.CV].

[53] Aochuan Chen et al. *DeepZero: Scaling up Zeroth-Order Optimization for Deep Model Training*. 2024. arXiv: 2310.02025 [cs.LG].

[54] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform". In: *arXiv preprint arXiv:1809.02839* (2018).

[55] Deli Chen et al. "Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View". In: *CoRR* abs/1909.03211 (2019). arXiv: 1909.03211. URL: http://arxiv.org/abs/1909.03211.

[56] Minghui Chen et al. *FedSoup: Improving Generalization and Personalization in Federated Learning via Selective Model Interpolation*. 2023. arXiv: 2307.10507 [cs.LG].

[57] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. "Net2net: Accelerating learning via knowledge transfer". In: *arXiv preprint arXiv:1511.05641* (2015).

[58] Tianqi Chen et al. *Training Deep Nets with Sublinear Memory Cost*. 2016. arXiv: 1604.06174 [cs.LG].

[59] Ting Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: 2002.05709 [cs.LG].

[60] Tsai-Shien Chen et al. *Incremental False Negative Detection for Contrastive Learning*. 2022. arXiv: 2106.03719 [cs.CV].

[61] Yangrui Chen et al. "SAPipe: Staleness-Aware Pipeline for Data Parallel DNN Training". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 17981–17993. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/725ce5f2b1a8e2e0ac66994e7fefe375-Paper-Conference.pdf.

[62] Jack Choquette. "Nvidia hopper h100 gpu: Scaling performance". In: *IEEE Micro* 43.3 (2023), pp. 9–17.

[63] Jack Choquette et al. "NVIDIA A100 Tensor Core GPU: Performance and Innovation". In: *IEEE Micro* 41.2 (2021), pp. 29–35. DOI: 10.1109/MM.2021.3061394.

[64] Anna Choromanska et al. "Beyond backprop: Alternating minimization with co-activation memory". In: *arXiv preprint arXiv:1806.09077* (2018).

[65] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. "A downsampled variant of imagenet as an alternative to the cifar datasets". In: *arXiv preprint arXiv:1707.08819* (2017).

[66] Adam Coates, Honglak Lee, and Andrew Y. Ng. "An Analysis of Single Layer Networks in Unsupervised Feature Learning". In: *AISTATS* (2011).

[67] Corinna Cortes et al. *AdaNet: Adaptive Structural Learning of Artificial Neural Networks*. 2017. arXiv: 1607.01097 [cs.LG].

[68] Ben Cottier et al. *The rising costs of training frontier AI models*. 2024. arXiv: 2405.21015 [id='cs.CY'].

[69] Francis Crick. "The recent excitement about neural networks". In: *Nature* 337.6203 (Jan. 1989), pp. 129–132. ISSN: 1476-4687. DOI: 10.1038/337129a0. URL: https://doi.org/10.1038/337129a0.

[70] Yann le Cun. "A Theoretical Framework for Back-Propagation". In: 1988. URL: https://api.semanticscholar.org/CorpusID:16775098.

[71] Wojciech Marian Czarnecki et al. "Understanding Synthetic Gradients and Decoupled Neural Interfaces". In: *CoRR* abs/1703.00522 (2017). arXiv: 1703.00522. URL: http://arxiv.org/abs/1703.00522.

[72] William J. Dally, Stephen W. Keckler, and David B. Kirk. "Evolution of the Graphics Processing Unit (GPU)". In: *IEEE Micro* 41.6 (2021), pp. 42–51. DOI: 10.1109/MM.2021.3113475.

[73] Tri Dao. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*. 2023. arXiv: 2307.08691 [cs.LG]. URL: https://arxiv.org/abs/2307.08691.

[74] Jeffrey Dean et al. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

[75] Aaron Defazio and Konstantin Mishchenko. *Learning-Rate-Free Learning by D-Adaptation*. 2023. arXiv: 2301.07733 [cs.LG]. URL: https://arxiv.org/abs/2301.07733.

[76] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[77] Georgios Detorakis, Travis Bartley, and Emre Neftci. *Contrastive Hebbian Learning with Random Feedback Weights*. 2018. arXiv: 1806.07406 [cs.LG].

[78] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[79] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[80] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1410.8516.

[81] Michael Diskin et al. "Distributed Deep Learning In Open Collaborations". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: https://openreview.net/forum?id=FYHktcK-7v.

[82] Carl Doersch, Abhinav Gupta, and Alexei A Efros. "Unsupervised visual representation learning by context prediction". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1422–1430.

[83] Shiyu Duan and José Carlos Príncipe. "Training Deep Architectures Without End-to-End Backpropagation: A Survey on the Provably Optimal Methods". In: *IEEE Computational Intelligence Magazine* 17 (2021), pp. 39–51. URL: https://api.semanticscholar.org/CorpusID:251442313.

[84] Shiyu Duan, Shujian Yu, and José C. Príncipe. "Modularizing Deep Learning via Pairwise Learning With Kernels". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.4 (2022), pp. 1441–1451. DOI: 10.1109/TNNLS.2020.3042346.

[85] Shiyu Duan et al. *On Kernel Method-Based Connectionist Models and Supervised Deep Learning Without Backpropagation*. 2019. arXiv: 1802.03774 [cs.LG].

[86] Chen Dun et al. *Efficient and Light-Weight Federated Learning via Asynchronous Distributed Dropout*. 2022. eprint: arXiv:2210.16105.

[87] Chen Dun et al. "ResIST: Layer-wise decomposition of ResNets for distributed training". In: *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*. Ed. by James Cussens and Kun Zhang. Vol. 180. Proceedings of Machine Learning Research. PMLR, Jan. 2022, pp. 610–620. URL: https://proceedings.mlr.press/v180/dun22a.html.

[88] Emmanuel Dupoux. "Cognitive science in the era of artificial intelligence: A roadmap for reverse-engineering the infant language-learner". In: *Cognition* 173 (2018), pp. 43–59.

[89] Adar Elad et al. "Direct Validation of the Information Bottleneck Principle for Deep Nets". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. 2019, pp. 758–762. DOI: 10.1109/ICCVW.2019.00099.

[90] Adar Elad et al. "The effectiveness of layer-by-layer training using the information bottleneck principle". In: 2018. URL: https://api.semanticscholar.org/CorpusID:53458297.

[91] Maxence Ernoult et al. *Equilibrium Propagation with Continual Weight Updates*. 2020. arXiv: 2005.04168 [cs.NE].

[92] Maxence M Ernoult et al. "Towards Scaling Difference Target Propagation by Learning Backprop Targets". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 17–23 Jul 2022, pp. 5968–5987. URL: https://proceedings.mlr.press/v162/ernoult22a.html.

[93] Yasaman Esfandiari et al. “Cross-Gradient Aggregation for Decentralized Learning from Non-IID Data”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 3036–3046. URL: https://proceedings.mlr.press/v139/esfandiari21a.html.

[94] Utku Evci et al. “Gradmax: Growing neural networks using gradient information”. In: *arXiv preprint arXiv:2201.05125* (2022).

[95] Mathieu Even et al. “A Continuized View on Nesterov Acceleration for Stochastic Gradient Descent and Randomized Gossip”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. Vol. 34. 2021, pp. 28054–28066.

[96] Scott Fahlman and Christian Lebiere. “The Cascade-Correlation Learning Architecture”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/69adc1e107f7f7d035d7baf04342e1ca-Paper.pdf.

[97] Shiqing Fan et al. “DAPPLE: a pipelined data parallel approach for training large models”. In: *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP ’21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 431–445. ISBN: 9781450382946. DOI: 10.1145/3437801.3441593. URL: https://doi.org/10.1145/3437801.3441593.

[98] Hamid Reza Feyzmahdavian and Mikael Johansson. “Asynchronous Iterations in Optimization: New Sequence Results and Sharper Algorithmic Guarantees”. In: *Journal of Machine Learning Research* 24.158 (2023), pp. 1–75. URL: http://jmlr.org/papers/v24/22-0555.html.

[99] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. “Deep ensembles: A loss landscape perspective”. In: *arXiv preprint arXiv:1912.02757* (2019).

[100] Giorgio Franceschelli and Mirco Musolesi. “Copyright in generative deep learning”. In: *Data and Policy* 4 (2022). ISSN: 2632-3249. DOI: 10.1017/dap.2022.10. URL: http://dx.doi.org/10.1017/dap.2022.10.

[101] Jonathan Frankle and Sasha Rush. *Is Attention All You Need?* https://www.isattentionallyouneed.c 2024.

[102] Jonathan Frankle et al. *Linear Mode Connectivity and the Lottery Ticket Hypothesis*. 2020. arXiv: 1912.05671 [cs.LG].

[103] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2015. eprint: arXiv:1506.02142.

[104] Timur Garipov et al. “Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/be3087e74e9100d4bc4c6268cdbe8456-Paper.pdf.

[105] Alexander L Gaunt et al. “AMPNet: Asynchronous model-parallel training for dynamic neural networks”. In: *arXiv preprint arXiv:1705.09786* (2017).

[106] Wulfram Gerstner et al. “Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules”. In: *Frontiers in Neural Circuits* 12 (July 2018). ISSN: 1662-5110. DOI: 10.3389/fncir.2018.00053. URL: http://dx.doi.org/10.3389/fncir.2018.00053.

[107] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. “Unsupervised representation learning by predicting image rotations”. In: *arXiv preprint arXiv:1803.07728* (2018).

[108] Daniel Golovin et al. *Gradientless Descent: High-Dimensional Zeroth-Order Optimization*. 2020. arXiv: 1911.06317 [cs.LG].

[109] Aidan N Gomez et al. “Interlocking Backpropagation: Improving depthwise model-parallelism”. In: *Journal of Machine Learning Research* 23.171 (2022), pp. 1–28.

[110] Aidan N Gomez et al. “The reversible residual network: Backpropagation without storing activations”. In: *Advances in neural information processing systems* 30 (2017).

[111] Aidan N. Gomez et al. “Interlocking Backpropagation: Improving depthwise model-parallelism”. In: *J. Mach. Learn. Res.* 23 (2020), 171:1–171:28. URL: https://api.semanticscholar.org/CorpusID:222208763.

[112] Chengyue Gong, Xingchao Liu, and qiang liu. “FAST TRAINING OF CONTRASTIVE LEARNING WITH INTERMEDIATE CONTRASTIVE LOSS”. In: ().

[113] Raphael Gontijo-Lopes, Yann Dauphin, and Ekin D Cubuk. “No one representation to rule them all: Overlapping features of training methods”. In: *arXiv preprint arXiv:2110.12899* (2021).

[114] Eduard Gorbunov et al. “Recent Theoretical Advances in Decentralized Distributed Convex Optimization”. In: *High-Dimensional Optimization and Probability: With a View Towards Data Science*. Cham: Springer International Publishing, 2022, pp. 253–325. ISBN: 978-3-031-00832-0. DOI: 10.1007/978-3-031-00832-0_8.

[115] Akhilesh Deepak Gotmare et al. “Decoupling Backpropagation using Constrained Optimization Methods”. In: 2018. URL: https://api.semanticscholar.org/CorpusID:49746693.

[116] Priya Goyal et al. “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”. In: *CoRR* abs/1706.02677 (2017). arXiv: 1706.02677. URL: http://arxiv.org/abs/1706.02677.

[117] Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. Second. Society for Industrial and Applied Mathematics, 2008. DOI: 10.1137/1.9780898717761. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898717761. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898717761.

[118] Jean-Bastien Grill et al. “Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 21271–21284. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.

[119] Stephen Grossberg. “Competitive learning: From interactive activation to adaptive resonance”. In: *Cognitive science* 11.1 (1987), pp. 23–63.

[120] Fangda Gu, Armin Askari, and Laurent El Ghaoui. “Fenchel Lifted Networks: A Lagrange Relaxation of Neural Network Training”. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 3362–3371. URL: https://proceedings.mlr.press/v108/gu20a.html.

[121] Lei Guan. *Weight Prediction Boosts the Convergence of AdamW*. 2023. arXiv: 2302.00195 [cs.LG].

[122] Naiyang Guan et al. “Delay Compensated Asynchronous Adam Algorithm for Deep Neural Networks”. In: Dec. 2017, pp. 852–859. DOI: 10.1109/ISPA/IUCC.2017.00130.

[123] The Guardian. *Google’s emissions climb nearly 50 percent in five years due to AI energy demand*. https://www.theguardian.com/technology/article/2024/jul/02/google-ai-emissions. 2024.

[124] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. “Towards deep learning with segregated dendrites”. en. In: *Elife* 6 (Dec. 2017).

[125] Joyeeta Gupta, Hilmer Bosch, and Luc van Vliet. *AI’s excessive water consumption threatens to drown out its environmental contributions*. https://theconversation.com/ais-excessive-water-consumption-threatens-to-drown-out-its-environmental-contributions-225854. 2024.

[126] Vineet Gupta, Tomer Koren, and Yoram Singer. *Shampoo: Preconditioned Stochastic Tensor Optimization*. 2018. arXiv: 1802.09568 [cs.LG]. URL: https://arxiv.org/abs/1802.09568.

[127] Guy Hacohen and Daphna Weinshall. *On The Power of Curriculum Learning in Training Deep Networks*. 2019. arXiv: 1904.03626 [cs.LG].

[128] Osama A. Hanna et al. "Quantization of Distributed Data for Learning". In: *IEEE Journal on Selected Areas in Information Theory* 2.3 (2021), pp. 987–1001. DOI: 10.1109/JSAIT.2021.3105359.

[129] James Harrison, Luke Metz, and Jascha Sohl-Dickstein. *A Closer Look at Learned Optimization: Stability, Robustness, and Inductive Biases*. 2022. arXiv: 2209.11208 [id='cs.LG'].

[130] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778. URL: https://api.semanticscholar.org/CorpusID:206594692.

[131] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. Comment: CVPR 2020 camera-ready. Code: https://github.com/facebookresearch/moco. 2019. URL: http://arxiv.org/abs/1911.05722.

[132] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[133] Julien Herrmann et al. *Optimal checkpointing for heterogeneous chains: how to train deep neural networks with limited memory*. 2019. arXiv: 1911.13214 [cs.LG].

[134] Chris Hettinger et al. *Forward Thinking: Building and Training Neural Networks One Layer at a Time*. 2017. arXiv: 1706.02480 [stat.ML].

[135] Geoffrey Hinton. *The Forward-Forward Algorithm: Some Preliminary Investigations*. 2022. arXiv: 2212.13345 [cs.LG].

[136] Geoffrey E Hinton and James McClelland. "Learning representations by recirculation". In: *Neural information processing systems*. 1987.

[137] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[138] Qirong Ho et al. "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 1223–1231.

[139] Torsten Hoefler et al. *Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks*. 2021. arXiv: 2102.00554 [cs.LG]. URL: https://arxiv.org/abs/2102.00554.

[140] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL]. URL: https://arxiv.org/abs/2203.15556.

[141] J J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". en. In: *Proc Natl Acad Sci U S A* 79.8 (Apr. 1982), pp. 2554–2558.

[142] J J Hopfield. "Neurons with graded response have collective computational properties like those of two-state neurons." In: *Proceedings of the National Academy of Sciences* 81.10 (1984), pp. 3088–3092. DOI: 10.1073/pnas.81.10.3088. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.81.10.3088. URL: https://www.pnas.org/doi/abs/10.1073/pnas.81.10.3088.

[143] Stefan Horoi et al. "Harmony in Diversity: Merging Neural Networks with Canonical Correlation Analysis". In: *International Conference on Machine Learning (ICML)*. 2024.

[144] Furong Huang et al. *Learning Deep ResNet Blocks Sequentially using Boosting Theory*. 2018. arXiv: 1706.04964 [cs.LG].

[145] Ken Huang et al. "Security and Privacy Concerns in ChatGPT". In: *Beyond AI: ChatGPT, Web3, and the Business Landscape of Tomorrow*. Springer, 2023, pp. 297–328.

[146] Yanping Huang et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism". In: *Advances in neural information processing systems* 32 (2019).

[147] Zhouyuan Huo, Bin Gu, and Heng Huang. "Training Neural Networks Using Features Replay". In: *Advances in Neural Information Processing Systems* (2018).

[148] Tri Huynh et al. *Boosting Contrastive Self-Supervised Learning with False Negative Cancellation*. 2022. arXiv: 2011.11765 [cs.CV].

[149] Bernd Illing et al. *Local plasticity rules can learn deep representations using self-supervised contrastive predictions*. 2021. arXiv: 2010.08262 [cs.NE].

[150] Fakultit Informatik et al. "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies". In: *A Field Guide to Dynamical Recurrent Neural Networks* (Mar. 2003).

[151] Pavel Izmailov et al. *Averaging Weights Leads to Wider Optima and Better Generalization*. 2019. arXiv: 1803.05407 [cs.LG].

[152] Marwan Jabri and Barry Flower. "Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks". en. In: *Neural Comput* 3.4 (1991), pp. 546–565.

[153] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. "i-revnet: Deep invertible networks". In: *arXiv preprint arXiv:1802.07088* (2018). URL: https://openreview.net/forum?id=HJsjkMb0Z.

[154] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural tangent kernel: Convergence and generalization in neural networks". In: *Advances in neural information processing systems* 31 (2018).

[155] Max Jaderberg et al. *Decoupled Neural Interfaces using Synthetic Gradients*. PMLR, 2017. arXiv: 1608.05343 [cs.LG].

[156] Arpan Jain et al. "GEMS: GPU-Enabled Memory-Aware Model-Parallelism System for Distributed DNN Training". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2020, pp. 1–15. DOI: 10.1109/SC41405.2020.00049.

[157] Samyak Jain et al. *DART: Diversify-Aggregate-Repeat Training Improves Generalization of Neural Networks*. 2023. eprint: arXiv:2302.14685.

[158] Zhihao Jia, Matei Zaharia, and Alex Aiken. "Beyond Data and Model Parallelism for Deep Neural Networks." In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 1–13.

[159] Albert Q. Jiang et al. *Mixtral of Experts*. 2024. arXiv: 2401.04088 [cs.LG]. URL: https://arxiv.org/abs/2401.04088.

[160] Li Jing et al. *Understanding Dimensional Collapse in Contrastive Self-supervised Learning*. 2022. arXiv: 2110.09348 [cs.CV].

[161] Alexia Jolicoeur-Martineau et al. *PopulAtion Parameter Averaging (PAPA)*. 2023. eprint: arXiv:2304.03094.

[162] Keller Jordan et al. *REPAIR: REnormalizing Permuted Activations for Interpolation Repair*. 2023. arXiv: 2211.08403 [cs.LG].

[163] Norman Jouppi et al. "Motivation for and evaluation of the first tensor processing unit". In: *ieee Micro* 38.3 (2018), pp. 10–19.

[164] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[165] Yu-Wei Kao and Hung-Hsuan Chen. "Associated Learning: Decomposing End-to-End Backpropagation Based on Autoencoders and Target Propagation". In: *Neural Computation* 33.1 (Jan. 2021), pp. 174–193. ISSN: 0899-7667. DOI: 10.1162/necoa01335. eprint: https://direct.mit.edu/neco/article-pdf/33/1/174/1865567/neco_a_01335.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01335.

[166] Skander Karkar et al. "Block-wise Training of Residual Networks via the Minimizing Movement Scheme". In: 2022. URL: https://api.semanticscholar.org/CorpusID:252683618.

[167] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].

[168] Henry J Kelley. "Gradient theory of optimal flight paths". In: *Ars Journal* 30.10 (1960), pp. 947–954.

[169] Oscar Key, Jean Kaddour, and Pasquale Minervini. "Local LoRA: Memory-Efficient Fine-Tuning of Large Language Models". In: *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*. 2023. URL: https://openreview.net/forum?id=LHKmzWP7RN.

[170] Chiheon Kim et al. *torchgpipe: On-the-fly Pipeline Parallelism for Training Giant Models*. 2020. arXiv: 2004.09910 [cs.DC].

[171] Taebum Kim et al. "BPipe: Memory-Balanced Pipeline Parallelism for Training Large Language Models". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 23–29 Jul 2023, pp. 16639–16653. URL: https://proceedings.mlr.press/v202/kim23l.html.

[172] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[173] Simon Knowles. "Graphcore". In: *2021 IEEE Hot Chips 33 Symposium (HCS)*. 2021, pp. 1–25. DOI: 10.1109/HCS52781.2021.9567075.

[174] Adam Kohan, Edward A. Rietman, and Hava T. Siegelmann. *Signal Propagation: A Framework for Learning and Inference In a Forward Pass*. 2022. arXiv: 2204.01723 [cs.LG].

[175] Adam A. Kohan, Edward A. Rietman, and Hava T. Siegelmann. *Error Forward-Propagation: Reusing Feedforward Connections to Propagate Errors in Deep Learning*. 2018. arXiv: 1808.03357 [cs.NE].

[176] Anastasia Koloskova, Sebastian U. Stich, and Martin Jaggi. "Sharper convergence guarantees for asynchronous SGD for distributed and federated learning". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. <conf-loc>, <city>New Orleans</city>, <state>LA</state>, <country>USA</country>, </conf-loc>: Curran Associates Inc., 2024. ISBN: 9781713871088.

[177] Vladimir Koltchinskii, Alexandre B. Tsybakov, and Karim Lounici. *Nuclear norm penalization and optimal rates for noisy low rank matrix completion*. 2010. DOI: 10.48550/ARXIV.1011.6256. URL: https://arxiv.org/abs/1011.6256.

[178] Jakub Konecný et al. "Federated Learning: Strategies for Improving Communication Efficiency". In: *ArXiv* abs/1610.05492 (2016). URL: https://api.semanticscholar.org/CorpusID:14999259.

[179] Jakub Konecný et al. "Federated Optimization: Distributed Machine Learning for On-Device Intelligence". In: *ArXiv* abs/1610.02527 (2016). URL: https://api.semanticscholar.org/CorpusID:2549272.

[180] Vijay Korthikanti et al. *Reducing Activation Recomputation in Large Transformer Models*. 2022. arXiv: 2205.05198 [cs.LG].

[181] Atli Kosson et al. *Pipelined Backpropagation at Scale: Training Large Models without Batches*. 2021. arXiv: 2003.11666 [cs.LG].

[182] Dmitry Kovalev, Adil Salim, and Peter Richtarik. "Optimal and Practical Algorithms for Smooth and Strongly Convex Decentralized Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 18342–18352.

[183] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009. URL: https://api.semanticscholar.org/CorpusID:18268744.

[184] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[185] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[186] Dmitry Krotov and John J. Hopfield. "Unsupervised learning by competing hidden units". In: *Proceedings of the National Academy of Sciences* 116.16 (2019), pp. 7723–7731. DOI: 10.1073/pnas.1820458116. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1820458116. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1820458116.

[187] Mandar Kulkarni and Shirish Karande. *Layer-wise training of deep networks using kernel similarity*. 2017. arXiv: 1703.07115 [cs.LG].

[188] Daniel Kunin et al. *Two Routes to Scalable Credit Assignment without Weight Symmetry*. 2020. arXiv: 2003.01513 [q-bio.NC].

[189] Axel Laborieux et al. "Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing Its Gradient Estimator Bias". English. In: *Frontiers in Neuroscience* (2021 Feb 18 2021/02/18/). URL: %5Curl%7Bhttp://accesdistant.sorbonne-universite.fr/scholarly-journals/scaling-equilibrium-propagation-deep-convnets/docview/2490837653/se-2%7D.

[190] Gabriele Lagani et al. *Spiking Neural Networks and Bio-Inspired Supervised Deep Learning: A Survey*. 2023. arXiv: 2307.16235 [cs.NE].

[191] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in neural information processing systems* 30 (2017).

[192] Chandrashekar Lakshminarayanan and Csaba Szepesvari. "Linear stochastic approximation: How far does constant step-size and iterate averaging go?" In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2018, pp. 1347–1355.

[193] Joel Lamy-Poirier. "Breadth-First Pipeline Parallelism". In: *Proceedings of Machine Learning and Systems* 5 (2023).

[194] Stefanos Laskaridis, Alexandros Kouris, and Nicholas D. Lane. "Adaptive Inference through Early-Exit Networks: Design, Challenges and Directions". In: *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*. MobiSys '21. ACM, June 2021. DOI: 10.1145/3469116.3470012. URL: http://dx.doi.org/10.1145/3469116.3470012.

[195] Michael Laskin et al. *Parallel Training of Deep Networks with Local Updates*. 2021. arXiv: 2012.03837 [cs.LG]. URL: https://openreview.net/forum?id=ufS1zWbRCEa.

[196] Tim Tsz-Kit Lau et al. *A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training*. 2018. arXiv: 1803.09082 [stat.ML].

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

[198] Teven Le Scao et al. "Bloom: A 176b-parameter open-access multilingual language model". In: (2023).

[199] Yann LeCun. "A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27". In: *Open Review* 62.1 (2022), pp. 1–62.

[200] Yann Lecun, Sumit Chopra, and Raia Hadsell. "A tutorial on energy-based learning". In: Jan. 2006.

[201] Chen-Yu Lee et al. "Deeply-supervised nets". In: *Artificial intelligence and statistics*. 2015, pp. 562–570.

[202] Dong-Hyun Lee et al. "Difference target propagation". In: *Joint european conference on machine learning and knowledge discovery in databases*. Springer. 2015, pp. 498–515. arXiv: 1412.7525 [cs.LG].

[203] Régis Lengellé and Thierry Denoeux. "Training MLPs layer by layer using an objective function for internal representations". In: *Neural Networks* 9.1 (1996), pp. 83–97.

[204] Dacheng Li et al. *DISTFLASHATTN: Distributed Memory-efficient Attention for Long-context LLMs Training*. 2024. arXiv: 2310.03294 [cs.LG].

[205] Hao Li et al. *Improved Techniques for Training Adaptive Deep Networks*. 2019. arXiv: 1908.06294 [cs.CV].

[206] Jia Li, Cong Fang, and Zhouchen Lin. *Lifted Proximal Operator Machines*. 2018. arXiv: 1811.01501 [cs.LG].

[207] Margaret Li et al. *Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models*. 2022. arXiv: 2208.03306 [cs.CL].

[208] Mu Li et al. "Scaling Distributed Machine Learning with the Parameter Server". In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. OSDI'14. Broomfield, CO: USENIX Association, 2014, pp. 583–598. ISBN: 9781931971164.

[209] Pengfei Li et al. *Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models*. 2023. arXiv: 2304.03271 [cs.LG]. URL: https://arxiv.org/abs/2304.03271.

[210] Shenggui Li et al. "Sequence parallelism: Long sequence training from system perspective". In: *arXiv preprint arXiv:2105.13120* (2021).

[211] Shigang Li and Torsten Hoefler. "Chimera: efficiently training large-scale neural networks with bidirectional pipelines". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '21. ACM, Nov. 2021. DOI: 10.1145/3458817.3476145. URL: http://dx.doi.org/10.1145/3458817.3476145.

[212] Tian Li et al. "Federated Optimization for Heterogeneous Networks". In: *ICML Workshop on Adaptive & Multitask Learning: Algorithms & Systems*. 2019. URL: https://openreview.net/forum?id=SkgwE5Ss3N.

[213] Weishi Li et al. *Deep Model Fusion: A Survey*. 2023. eprint: arXiv:2309.15698.

[214] Baohao Liao, Shaomu Tan, and Christof Monz. "Make your pre-trained model reversible: From parameter to memory efficient fine-tuning". In: *arXiv preprint arXiv:2306.00477* (2023).

[215] Qianli Liao, Joel Leibo, and Tomaso Poggio. "How important is weight symmetry in backpropagation?" In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 2016.

[216] Timothy P. Lillicrap et al. "Backpropagation and the brain". In: *Nature Reviews Neuroscience* 21.6 (June 2020), pp. 335–346. ISSN: 1471-0048. DOI: 10.1038/s41583-020-0277-3. URL: https://doi.org/10.1038/s41583-020-0277-3.

[217] Timothy P. Lillicrap et al. "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature Communications* 7.1 (Nov. 2016), p. 13276. ISSN: 2041-1723. DOI: 10.1038/ncomms13276. URL: https://doi.org/10.1038/ncomms13276.

[218] Tao Lin et al. "Don't Use Large Mini-batches, Use Local SGD". In: *International Conference on Learning Representations*. 2020.

[219] Seppo Linnainmaa. "Taylor expansion of the accumulated rounding error". In: *BIT Numerical Mathematics* 16.2 (1976), pp. 146–160.

[220] Yang Liu et al. "Invertible Denoising Network: A Light Solution for Real Noise Removal". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 13360–13369. URL: https://api.semanticscholar.org/CorpusID:233324461.

[221] Yuliang Liu et al. *Colossal-Auto: Unified Automation of Parallelization and Activation Checkpoint for Large-scale Models*. 2023. arXiv: 2302.02599 [cs.LG].

[222] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[223] Sindy Löwe, Peter O'Connor, and Bastiaan S. Veeling. *Putting An End to End-to-End: Gradient-Isolated Learning of Representations*. 2019. DOI: 10.48550/ARXIV.1905.11786. URL: https://arxiv.org/abs/1905.11786.

[224] Shuming Ma et al. *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits*. 2024. arXiv: 2402.17764 [cs.CL]. URL: https://arxiv.org/abs/2402.17764.

[225] 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].

[226] W Maass. "On the computational power of winner-take-all". en. In: *Neural Comput* 12.11 (Nov. 2000), pp. 2519–2535.

[227] Eran Malach and Shai Shalev-Shwartz. *A Provably Correct Algorithm for Deep Learning that Actually Works*. 2018. arXiv: 1803.09522 [cs.LG].

[228] Sadhika Malladi et al. *Fine-Tuning Language Models with Just Forward Passes*. 2024. arXiv: 2305.17333 [cs.LG].

[229] Karttikeya Mangalam et al. "Reversible vision transformers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10830–10840.

[230] Artavazd Maranjyan, Mher Safaryan, and Peter Richtárik. *GradSkip: Communication-Accelerated Local Gradient Methods with Better Computational Complexity*. 2022. arXiv: 2210.16402 [cs.LG].

[231] E. S. Marquez, J. S. Hare, and M. Niranjan. "Deep Cascade Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (Nov. 2018), pp. 5475–5485. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2018.2805098.

[232] Saeed Masoudnia and Reza Ebrahimpour. "Mixture of experts: a literature survey". In: *Artificial Intelligence Review* 42 (2014), pp. 275–293.

[233] P Mazzoni, R A Andersen, and M I Jordan. "A more biologically plausible learning rule for neural networks". en. In: *Proc Natl Acad Sci U S A* 88.10 (May 1991), pp. 4433–4437.

[234] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[235] Ryan McDonald, Keith Hall, and Gideon Mann. "Distributed training strategies for the structured perceptron". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. HLT '10. Los Angeles, California: Association for Computational Linguistics, 2010, pp. 456–464. ISBN: 1932432655.

[236] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 20–22 Apr 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[237] Gaurav Menghani. "Efficient deep learning: A survey on making deep learning models smaller, faster, and better". In: *ACM Computing Surveys* 55.12 (2023), pp. 1–37.

[238] Meta. *Pursuing groundbreaking scale and accelerating research using Meta's Research SuperCluster*. https://ai.meta.com/blog/supercomputer-meta-research-supercluster-2023/. 2023.

[239] Luke Metz et al. "Understanding and correcting pathologies in the training of learned optimizers". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4556–4565.

[240] Luke Metz et al. *VeLO: Training Versatile Learned Optimizers by Scaling Up*. 2022. arXiv: 2211.09760 [id='cs.LG'].

[241] Alexander Meulemans et al. *A Theoretical Framework for Target Propagation*. 2020. arXiv: 2006.14331 [cs.LG].

[242] Alexander Meulemans et al. *Credit Assignment in Neural Networks through Deep Feedback Control*. 2022. arXiv: 2106.07887 [cs.LG].

[243] Paulius Micikevicius et al. "Mixed precision training". In: *arXiv preprint arXiv:1710.03740* (2017).

[244] Thomas Miconi. *Hebbian learning with gradients: Hebbian convolutional neural networks with modern deep learning frameworks*. 2021. arXiv: 2107.01729 [cs.NE].

[245] Beren Millidge et al. *Backpropagation at the Infinitesimal Inference Limit of Energy-Based Models: Unifying Predictive Coding, Equilibrium Propagation, and Contrastive Hebbian Learning*. 2022. arXiv: 2206.02629 [cs.LG].

[246] Beren Millidge et al. *Predictive Coding: Towards a Future of Deep Learning beyond Backpropagation?* 2022. arXiv: 2202.09467 [cs.NE].

[247] Konstantin Mishchenko et al. "Asynchronous SGD Beats Minibatch SGD Under Arbitrary Delays". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. Vol. 35. 2022, pp. 420–433. URL: https://openreview.net/forum?id=4XP0ZuQKXmV.

[248] Konstantin Mishchenko et al. "ProxSkip: Yes! Local Gradient Steps Provably Lead to Communication Acceleration! Finally!" In: *arXiv preprint arXiv:2202.09357* (2022).

[249] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. *A Survey on Deep Neural Network Compression: Challenges, Overview, and Solutions*. 2020. arXiv: 2010.03954 [cs.LG].

[250] Ioannis Mitliagkas et al. "Asynchrony begets momentum, with an application to deep learning". In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Monticello, IL, USA: IEEE Press, 2016, pp. 997–1004. DOI: 10.1109/ALLERTON.2016.7852343. URL: https://doi.org/10.1109/ALLERTON.2016.7852343.

[251] Guido F Montufar et al. "On the number of linear regions of deep neural networks". In: *Advances in neural information processing systems* 27 (2014).

[252] Timoleon Moraitis et al. "SoftHebb: Bayesian inference in unsupervised Hebbian soft winner-take-all networks". In: *Neuromorphic Computing and Engineering* 2.4 (Dec. 2022), p. 044017. ISSN: 2634-4386. DOI: 10.1088/2634-4386/aca710. URL: http://dx.doi.org/10.1088/2634-4386/aca710.

[253] Alan Mosca and George D Magoulas. *Deep Incremental Boosting*. 2017. arXiv: 1708.03704 [stat.ML].

[254] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. "Deep supervised learning using local errors". In: *Frontiers in neuroscience* 12 (2018), p. 608.

[255] Javier R. Movellan. "Contrastive Hebbian Learning in the Continuous Hopfield Model". In: *Connectionist Models*. Ed. by David S. Touretzky et al. Morgan Kaufmann, 1991, pp. 10–17. ISBN: 978-1-4832-1448-1. DOI: https://doi.org/10.1016/B978-1-4832-1448-1.50007-X. URL: https://www.sciencedirect.com/science/article/pii/B978148321448150007X.

[256] Adel Nabli, Eugene Belilovsky, and Edouard Oyallon. "A2CiD2: Accelerating Asynchronous Communication in Decentralized Deep Learning". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=YE04aRkeZb.

[257] Adel Nabli and Edouard Oyallon. "DADAO: Decoupled Accelerated Decentralized Asynchronous Optimization". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 23–29 Jul 2023, pp. 25604–25626.

[258] Deepak Narayanan et al. "Memory-efficient pipeline-parallel dnn training". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7937–7947.

[259] Deepak Narayanan et al. "PipeDream: Generalized pipeline parallelism for DNN training". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019, pp. 1–15.

[260] James O' Neill. *An Overview of Neural Network Compression*. 2020. arXiv: 2006.03669 [cs.LG].

[261] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. "What is being transferred in transfer learning?" In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 512–523. URL: %5Curl%7Bhttps://proceedings.neurips.cc/paper_files/paper/2020/file/0607f4c705595b911a4f3e7a127b44e0-Paper.pdf%7D.

[262] John Nguyen et al. "Federated Learning with Buffered Asynchronous Aggregation". In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. Ed. by Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera. Vol. 151. Proceedings of Machine Learning Research. PMLR, 28–30 Mar 2022, pp. 3581–3607. URL: https://proceedings.mlr.press/v151/nguyen22b.html.

[263] Zanlin Ni et al. *Deep Incubation: Training Large Models by Divide-and-Conquering*. 2023. arXiv: 2212.04129 [cs.CV].

[264] Arild Nøkland. "Direct feedback alignment provides learning in deep neural networks". In: *Advances in neural information processing systems* 29 (2016).

[265] Arild Nøkland and Lars Hiller Eidnes. "Training neural networks with local error signals". In: *International conference on machine learning*. PMLR. 2019, pp. 4839–4850.

[266] João D. Nunes et al. "Spiking Neural Networks: A Survey". In: *IEEE Access* 10 (2022), pp. 60738–60764. DOI: 10.1109/ACCESS.2022.3179968.

[267] Randall C. O'Reilly. "Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm". In: *Neural Computation* 8.5 (1996), pp. 895–938. DOI: 10.1162/neco.1996.8.5.895.

[268] Ruben Ohana et al. *Photonic Differential Privacy with Direct Feedback Alignment*. 2022. arXiv: 2106.03645 [cs.LG].

[269] Erkki Oja. "Simplified neuron model as a principal component analyzer". In: *Journal of Mathematical Biology* 15.3 (Nov. 1982), pp. 267–273. ISSN: 1432-1416. DOI: 10.1007/BF00275687. URL: https://doi.org/10.1007/BF00275687.

[270] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].

[271] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [id='cs.CL'].

[272] Alexander Ororbia and Daniel Kifer. *The Neural Coding Framework for Learning Generative Models*. 2022. arXiv: 2012.03405 [cs.LG].

[273] Alexander Ororbia and Ankur Mali. *The Predictive Forward-Forward Algorithm*. 2023. arXiv: 2301.01452 [cs.LG].

[274] Alexander G Ororbia and Ankur Mali. "Biologically motivated algorithms for propagating local target representations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4651–4658.

[275] Alexander G Ororbia et al. "Conducting credit assignment by aligning local representations". In: *arXiv preprint arXiv:1803.01834* (2018).

[276] Jose Javier Gonzalez Ortiz et al. "Trade-offs of Local SGD at Scale: An Empirical Study". In: *NeurIPS 2020 OptML Workshop*. 2021.

[277] Edouard Oyallon. "Building a regular decision boundary with deep networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5106–5114.

[278] Pitch Patarasuk and Xin Yuan. "Bandwidth optimal all-reduce algorithms for clusters of workstations". In: *Journal of Parallel and Distributed Computing* 69.2 (2009), pp. 117–124. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2008.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0743731508001767.

[279] Adeetya Patel, Michael Eickenberg, and Eugene Belilovsky. "Local Learning with Neuron Groups". In: *arXiv preprint arXiv:2301.07635* (2023).

[280] Priyank Pathak, Jingwei Zhang, and Dimitris Samaras. "Local Learning on Transformers via Feature Reconstruction". In: *arXiv preprint arXiv:2212.14215* (2022).

[281] Suchita Pati et al. *Computation vs. Communication Scaling for Future Transformers on Future Hardware*. 2023. arXiv: 2302.02825 [cs.AR].

[282] Fidel A. Guerrero Peña et al. *Re-basin via implicit Sinkhorn differentiation*. 2022. eprint: arXiv:2212.12042.

[283] Guilherme Penedo et al. *The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale*. 2024. arXiv: 2406.17557 [cs.CL]. URL: https://arxiv.org/abs/2406.17557.

[284] Ray Perrault and Jack Clark. "Artificial Intelligence Index Report 2024". In: (2024).

[285] Roman Pogodin and Peter E. Latham. "Kernelized information bottleneck leads to biologically plausible 3-factor Hebbian learning in deep networks". In: *ArXiv* abs/2006.07123 (2020). URL: https://api.semanticscholar.org/CorpusID:219636144.

[286] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM journal on control and optimization* 30.4 (1992), pp. 838–855.

[287] Isabella Pozzi, Sander Bohté, and Pieter Roelfsema. *A Biologically Plausible Learning Rule for Deep Learning in the Brain*. 2019. arXiv: 1811.01768 [cs.NE].

[288] Samyam Rajbhandari et al. *ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning*. 2021. arXiv: 2104.07857 [cs.DC].

[289] Samyam Rajbhandari et al. *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*. 2020. arXiv: 1910.02054 [cs.LG].

[290] Alexandre Ramé et al. *Diverse Weight Averaging for Out-of-Distribution Generalization*. 2022. eprint: arXiv:2205.09739.

[291] Marc'Aurelio Ranzato et al. "Unsupervised learning of invariant feature hierarchies with applications to object recognition". In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[292] Rajesh P. N. Rao and Dana H. Ballard. "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". In: *Nature Neuroscience* 2.1 (Jan. 1999), pp. 79–87. ISSN: 1546-1726. DOI: 10.1038/4580. URL: https://doi.org/10.1038/4580.

[293] Sashank J. Reddi et al. "Adaptive Federated Optimization". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=LkFG31B13U5.

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

[295] Jie Ren et al. "{Zero-offload}: Democratizing {billion-scale} model training". In: *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 2021, pp. 551–564.

[296] Mengye Ren et al. "Scaling forward gradient with local losses". In: *arXiv preprint arXiv:2210.03310* (2022).

[297] Blake A Richards et al. "A deep learning framework for neuroscience". en. In: *Nat Neurosci* 22.11 (Oct. 2019), pp. 1761–1770.

[298] Carlos Riquelme et al. "Scaling Vision with Sparse Mixture of Experts". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 8583–8595. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/48237d9f2dea8c74c2a72126cf63d933-Paper.pdf.

[299] Herbert Robbins Robbins and Sutton Monro. "A stochastic approximation method." In: *The annals of mathematical statistics* (1951), pp. 400–407.

[300] Joshua David Robinson et al. "Contrastive Learning with Hard Negative Samples". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=CR1XOQ0UTh-.

[301] O. Roeder. *Seven Games: A Human History*. WW Norton, 2023. ISBN: 9781324051022. URL: https://books.google.fr/books?id=b2yPEAAAQBAJ.

[302] Pieter Roelfsema and Arjen Ooyen. "Attention-Gated Reinforcement Learning of Internal Representations for Classification". In: *Neural computation* 17 (Nov. 2005), pp. 2176–214. DOI: 10.1162/0899766054615699.

[303] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[304] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[305] David E. Rumelhart and David Zipser. "Feature discovery by competitive learning". In: *Cognitive Science* 9.1 (1985), pp. 75–112. ISSN: 0364-0213. DOI: https://doi.org/10.1016/S0364-0213(85)80010-0. URL: https://www.sciencedirect.com/science/article/pii/S0364021385800100.

[306] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[307] Max Ryabinin et al. *SWARM Parallelism: Training Large Models Can Be Surprisingly Communication-Efficient*. 2023. arXiv: 2301.11913 [cs.DC]. URL: https://arxiv.org/abs/2301.11913.

[308] João Sacramento et al. *Dendritic cortical microcircuits approximate the backpropagation algorithm*. 2018. arXiv: 1810.11393 [q-bio.NC].

[309] Keitaro Sakamoto and Issei Sato. *End-to-End Training Induces Information Bottleneck through Layer-Role Differentiation: A Comparative Analysis with Layer-wise Training*. 2024. arXiv: 2402.09050 [cs.LG].

[310] Terence D. Sanger. "Optimal unsupervised learning in a single-layer linear feedforward neural network". In: *Neural Networks* 2.6 (1989), pp. 459–473. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90044-0. URL: https://www.sciencedirect.com/science/article/pii/0893608089900440.

[311] Kevin Scaman et al. "Optimal Algorithms for Smooth and Strongly Convex Distributed Optimization in Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 3027–3036.

[312] Simone Scardapane et al. "Why Should We Add Early Exits to Neural Networks?" In: *Cognitive Computation* 12.5 (June 2020), pp. 954–966. ISSN: 1866-9964. DOI: 10.1007/s12559-020-09734-4. URL: http://dx.doi.org/10.1007/s12559-020-09734-4.

[313] Benjamin Scellier and Yoshua Bengio. "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation". In: *Frontiers in computational neuroscience* 11 (2017), p. 24.

[314] Terrence J. Sejnowski. "Large Language Models and the Reverse Turing Test". In: *Neural Computation* 35.3 (Feb. 2023), pp. 309–342. ISSN: 0899-7667. eprint: https://direct.mit.edu/neco/article-pdf/35/3/309/2071839/neco_a_01563.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01563.

[315] Shuheng Shen et al. "Faster distributed deep net training: computation and communication decoupled stochastic gradient descent". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 4582–4589. ISBN: 9780999241141.

[316] Mohammad Shoeybi et al. "Megatron-lm: Training multi-billion parameter language models using model parallelism". In: *arXiv preprint arXiv:1909.08053* (2019).

[317] Egor Shulgin and Peter Richtárik. *Towards a Better Theoretical Understanding of Independent Subnetwork Training*. 2023. eprint: arXiv:2306.16484.

[318] Shoaib Ahmed Siddiqui et al. *Blockwise Self-Supervised Learning at Scale*. 2023. arXiv: 2302.01647 [cs.CV].

[319] David Silver et al. "Learning by Directional Gradient Descent". In: *International Conference on Learning Representations*. 2021.

[320] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961.

[321] Sidak Pal Singh and Martin Jaggi. *Model Fusion via Optimal Transport*. 2023. arXiv: 1910.05653 [cs.LG].

[322] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2017. arXiv: 1506.01186 [cs.CV]. URL: https://arxiv.org/abs/1506.01186.

[323] Shaden Smith et al. "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model". In: *arXiv preprint arXiv:2201.11990* (2022).

[324] Nimit S. Sohoni et al. *Low-Memory Neural Network Training: A Technical Report*. 2022. arXiv: 1904.10631 [cs.LG].

[325] Jaeyong Song et al. "Optimus-CC: Efficient Large NLP Model Training with 3D Parallelism Aware Communication Compression". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 2023, pp. 560–573.

[326] Yuhang Song et al. "Can the Brain Do Backpropagation? — Exact Implementation of Backpropagation in Predictive Coding Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 22566–22579. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/fec87a37cdeec1c6ecf8181c0aa2d3bf-Paper.pdf.

[327] Yuhang Song et al. "Inferring neural activity before plasticity as a foundation for learning beyond backpropagation". en. In: *Nat Neurosci* 27.2 (Jan. 2024), pp. 348–358.

[328] Zhuoqing Song et al. "Optimal gradient tracking for decentralized optimization". In: *Mathematical Programming* (July 2023). ISSN: 1436-4646. DOI: 10.1007/s10107-023-01997-7.

[329] James C. Spall. "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation". In: *IEEE Transactions on Automatic Control* 37 (1992), pp. 332–341. URL: https://api.semanticscholar.org/CorpusID:122365276.

[330] Artin Spiridonoff, Alex Olshevsky, and Ioannis Ch. Paschalidis. *Communication-efficient SGD: From Local SGD to One-Shot Averaging*. 2021. eprint: arXiv:2106.04759.

[331] Tom Standage. *The Turk: The life and times of the famous eighteenth-century chess-playing machine*. Walker & Company, 2002.

[332] Sebastian U. Stich. "Local SGD Converges Fast and Communicates Little". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1g2JnRcFX.

[333] Sebastian U. Stich and Sai Praneeth Karimireddy. "The error-feedback framework: better rates for SGD with delayed gradients and compressed updates". In: *Journal of Machine Learning Research* 21.1 (Jan. 2020). ISSN: 1532-4435.

[334] Sebastian U. Stich and Sai Praneeth Karimireddy. "The Error-Feedback framework: SGD with Delayed Gradients". In: *Journal of Machine Learning Research* 21.237 (2020), pp. 1–36. URL: http://jmlr.org/papers/v21/19-748.html.

[335] *Stockfish*. https://stockfishchess.org/. 2008.

[336] Emma Strubell, Ananya Ganesh, and Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP*. 2019. arXiv: 1906.02243 [cs.CL].

[337] Pei Sun et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[338] Ruo-Yu Sun. "Optimization for Deep Learning: An Overview". In: *Journal of the Operations Research Society of China* 8.2 (June 2020), pp. 249–294. ISSN: 2194-6698. DOI: 10.1007/s40305-020-00309-6. URL: https://doi.org/10.1007/s40305-020-00309-6.

[339] Weigao Sun et al. "CO2: Efficient Distributed Training with Full Communication-Computation Overlap". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=Z05cn4IfaN.

[340] Richard Sutton. "The bitter lesson". In: *Incomplete Ideas (blog)* 13.1 (2019), p. 38.

[341] Cerebras Systems. *Cerebras Systems Unveils World's Fastest AI Chip with Whopping 4 Trillion Transistors*. https://www.cerebras.net/press-release/cerebras-announces-third-generation-wafer-scale-engine. 2024.

[342] Afrina Tabassum et al. *Hard Negative Sampling Strategies for Contrastive Representation Learning*. 2022. arXiv: 2206.01197 [cs.LG].

[343] Shanshan Tang, Bo Li, and Haijun Yu. "ChebNet: Efficient and Stable Constructions of Deep Neural Networks with Rectified Power Units using Chebyshev Approximations". In: *CoRR* abs/1911.05467 (2019). arXiv: 1911.05467. URL: http://arxiv.org/abs/1911.05467.

[344] Jakub M Tarnawski, Deepak Narayanan, and Amar Phanishayee. "Piper: Multidimensional Planner for DNN Parallelization". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 24829–24840. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/d01eeca8b24321cd2fe89dd85b9beb51-Paper.pdf.

[345] Gavin Taylor et al. *Training Neural Networks Without Gradients: A Scalable ADMM Approach*. 2016. arXiv: 1605.02026 [cs.LG].

[346] HPC-AI Tech. *Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training*. https://hpc-ai.com/blog/colossal-ai-a-unified-deep-learning-system-for-large-scale-parallel-training. 2021.

[347] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. *BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks*. 2017. arXiv: 1709.01686 [cs.NE].

[348] *The early mathematical manuscripts of Leibniz*. Courier Corporation, 2012.

[349] Louis Thiry et al. *The Unreasonable Effectiveness of Patches in Deep Convolutional Kernels Methods*. 2021. arXiv: 2101.07528 [id='cs.CV'].

[350] Neil C. Thompson et al. *The Computational Limits of Deep Learning*. 2022. arXiv: 2007.05558 [id='cs.LG'].

[351] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[352] Alan M Turing. *Computing machinery and intelligence*. Springer, 2009.

[353] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[354] Pablo Villalobos et al. *Will We Run Out of ML Data? Evidence From Projecting Dataset Size Trends*. 2022. arXiv: 2211.04325 [cs.LG].

[355] Guangcong Wang et al. "Deep growing learning". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2812–2820.

[356] Guanhua Wang et al. *ZeRO++: Extremely Efficient Collective Communication for Giant Model Training*. 2023. arXiv: 2306.10209 [cs.DC].

[357] Hongyi Wang et al. *Federated Learning with Matched Averaging*. 2020. eprint: arXiv:2002.06440.

[358] Jianyu Wang, Hao Liang, and Gauri Joshi. "Overlap Local-SGD: An Algorithmic Approach to Hide Communication Delays in Distributed SGD". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2020. DOI: 10.1109/icassp40776.2020.9053834. URL: http://dx.doi.org/10.1109/ICASSP40776.2020.9053834.

[359] Jianyu Wang et al. "SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SkxJ8REYPH.

[360] Minjie Wang, Chien-chin Huang, and Jinyang Li. "Supporting Very Large Models using Automatic Dataflow Graph Partitioning". In: *Proceedings of the Fourteenth EuroSys Conference 2019*. EuroSys '19. ACM, Mar. 2019. DOI: 10.1145/3302424.3303953. URL: http://dx.doi.org/10.1145/3302424.3303953.

[361] Shibo Wang and Pankaj Kanwar. *BFloat16: The secret to high performance on Cloud TPUs*. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus?hl=en. 2019.

[362] Xin Wang et al. *IDK Cascades: Fast Deep Learning by Learning not to Overthink*. 2018. arXiv: 1706.00885 [cs.CV].

[363] Xindi Wang et al. *Beyond the Limits: A Survey of Techniques to Extend the Context Length in Large Language Models*. 2024. arXiv: 2402.02244 [cs.CL]. URL: https://arxiv.org/abs/2402.02244.

[364] Yansen Wang et al. "CircuitNet: A Generic Neural Network to Realize Universal Circuit Motif Modeling". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 23–29 Jul 2023, pp. 35817–35835. URL: https://proceedings.mlr.press/v202/wang23k.html.

[365] Yulin Wang et al. "Revisiting Locally Supervised Learning: an Alternative to End-to-end Training". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=fAbkE6ant2.

[366] Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. *Federated Dropout – A Simple Approach for Enabling Federated Learning on Resource Constrained Devices*. 2021. eprint: arXiv:2109.15258.

[367] R. E. Wengert. "A simple automatic derivative evaluation program". In: *Communications of the ACM* 7 (1964), pp. 463–464. URL: https://api.semanticscholar.org/CorpusID:24039274.

[368] Justin Werfel, Xiaohui Xie, and H Sebastian Seung. "Learning curves for stochastic gradient descent in linear feedforward networks". en. In: *Neural Comput* 17.12 (Dec. 2005), pp. 2699–2718.

[369] James C. R. Whittington and Rafał Bogacz. "An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity". In: *Neural computation* 29 (2017), pp. 1229–1262. URL: https://api.semanticscholar.org/CorpusID:13651627.

[370] James CR Whittington and Rafal Bogacz. "An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity". In: *Neural computation* 29.5 (2017), pp. 1229–1262.

[371] David Gray Widder, Sarah West, and Meredith Whittaker. "Open (for business): Big tech, concentrated power, and the political economy of open AI". In: *Concentrated Power, and the Political Economy of Open AI (August 17, 2023)* (2023).

[372] Bernard Widrow and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation". In: *Proc. IEEE* 78 (1990), pp. 1415–1442. URL: https://api.semanticscholar.org/CorpusID:195704643.

[373] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[374] Blake Woodworth et al. "Is Local SGD Better than Minibatch SGD?" In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 10334–10343. URL: https://proceedings.mlr.press/v119/woodworth20a.html.

[375] Mitchell Wortsman et al. *Learning Neural Network Subspaces*. 2021. eprint: arXiv:2102.10472.

[376] Mitchell Wortsman et al. "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 17–23 Jul 2022, pp. 23965–23998. URL: https://proceedings.mlr.press/v162/wortsman22a.html.

[377] Mitchell Wortsman et al. *Robust fine-tuning of zero-shot models*. 2022. arXiv: 2109.01903 [cs.CV].

[378] Xiaofeng Wu, Jia Rao, and Wei Chen. *ATOM: Asynchronous Training of Massive Models for Deep Learning in a Decentralized Environment*. 2024. arXiv: 2403.10504 [cs.DC]. URL: https://arxiv.org/abs/2403.10504.

[379] Xuyang Wu et al. "Delay-Adaptive Step-sizes for Asynchronous Learning". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 17–23 Jul 2022, pp. 24093–24113. URL: https://proceedings.mlr.press/v162/wu22g.html.

[380] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. cite arxiv:1708.07747Comment: Dataset is freely available at https://github.com/zalandoresearch/fashion-mnist Benchmark is available at http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/. 2017. URL: http://arxiv.org/abs/1708.07747.

[381] Will Xiao et al. "Biologically-plausible learning algorithms can scale to large datasets". In: *International Conference on Learning Representations* (2019).

[382] Cong Xie, Sanmi Koyejo, and Indranil Gupta. "Asynchronous Federated Optimization". In: *NeurIPS 2020 OptML Workshop*. 2020.

[383] Xiaohui Xie and H Sebastian Seung. "Equivalence of backpropagation and contrastive Hebbian learning in a layered network". en. In: *Neural Comput* 15.2 (Feb. 2003), pp. 441–454.

[384] Yuwen Xiong, Mengye Ren, and Raquel Urtasun. "Loco: Local contrastive representation learning". In: *Advances in neural information processing systems* 33 (2020), pp. 11142–11153.

[385] An Xu, Zhouyuan Huo, and Heng Huang. "On the acceleration of deep learning model parallelism with staleness". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2088–2097.

[386] Hang Xu et al. "Compressed Communication for Distributed Deep Learning: Survey and Quantitative Evaluation". In: 2020. URL: https://api.semanticscholar.org/CorpusID:219069887.

[387] Bowen Yang et al. "Pipemare: Asynchronous pipeline parallel dnn training". In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 269–296.

[388] Haibo Yang et al. "Anarchic Federated Learning". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 17–23 Jul 2022, pp. 25331–25363. URL: https://proceedings.mlr.press/v162/yang22r.html.

[389] Zexiang Yi et al. "Learning rules in spiking neural networks: A survey". In: *Neurocomputing* 531 (2023), pp. 163–179. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2023.02.026. URL: https://www.sciencedirect.com/science/article/pii/S0925231223001662.

[390] Alexander Yom Din et al. "Jump to Conclusions: Short-Cutting Transformers with Linear Transformations". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari et al. Torino, Italia: ELRA and ICCL, May 2024, pp. 9615–9625. URL: https://aclanthology.org/2024.lrec-main.840.

[391] Kun Yuan, Qing Ling, and Wotao Yin. "On the Convergence of Decentralized Gradient Descent". In: *SIAM Journal on Optimization* 26.3 (2016), pp. 1835–1854. DOI: 10.1137/130943170.

[392] Mikhail Yurochkin et al. "Bayesian Nonparametric Federated Learning of Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 7252–7261. URL: https://proceedings.mlr.press/v97/yurochkin19a.html.

[393] Jure Zbontar et al. "Barlow Twins: Self-Supervised Learning via Redundancy Reduction". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 12310–12320. URL: https://proceedings.mlr.press/v139/zbontar21a.html.

[394] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[395] Jinshan Zeng et al. "Global Convergence of Block Coordinate Descent in Deep Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 7313–7323. URL: https://proceedings.mlr.press/v97/zeng19a.html.

[396] Guoqiang Zhang and W. Bastiaan Kleijn. *Training Deep Neural Networks via Optimization Over Graphs*. 2017. arXiv: 1702.03380 [cs.LG].

[397] Jian Zhang and Ioannis Mitliagkas. "YellowFin and the Art of Momentum Tuning". In: *Proceedings of Machine Learning and Systems*. Ed. by A. Talwalkar, V. Smith, and M. Zaharia. Vol. 1. 2019, pp. 289–308. URL: https://proceedings.mlsys.org/paper_files/paper/2019/file/b205b525b7ce002baae53228bab6d26b-Paper.pdf.

[398] Sixin Zhang, Anna Choromanska, and Yann LeCun. "Deep learning with elastic averaging SGD". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 685–693.

[399] Wei Zhang et al. *Staleness-aware Async-SGD for Distributed Deep Learning*. 2016. arXiv: 1511.05950 [cs.LG].

[400] Zhen Zhang et al. *MiCS: Near-linear Scaling for Training Gigantic Model on Public Cloud*. 2022. arXiv: 2205.00119 [cs.DC].

[401] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. *Efficient Training of Very Deep Neural Networks for Supervised Hashing*. 2016. arXiv: 1511.04524 [cs.CV].

[402] Andrew Zhao et al. "Expel: Llm agents are experiential learners". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, pp. 19632–19642.

[403] Xunyi Zhao et al. *Rockmate: an Efficient, Fast, Automatic and Generic Tool for Re-materialization in PyTorch*. 2023. arXiv: 2307.01236 [cs.LG].

[404] Yanli Zhao et al. *PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel*. 2023. arXiv: 2304.11277 [cs.DC].

[405] Lianmin Zheng et al. "Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 2022, pp. 559–578.

[406] Shuxin Zheng et al. "Asynchronous Stochastic Gradient Descent with delay compensation". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 4120–4129.

[407] Lianghui Zhu et al. "Vision mamba: Efficient visual representation learning with bidirectional state space model". In: *arXiv preprint arXiv:2401.09417* (2024).

[408] Rui-Jie Zhu et al. *Scalable MatMul-free Language Modeling*. 2024. arXiv: 2406.02528 [cs.CL]. URL: https://arxiv.org/abs/2406.02528.

[409] Shiqiang Zhu et al. “Intelligent Computing: The Latest Advances, Challenges, and Future”. In: *Intelligent Computing* 2 (2023), p. 0006. DOI: 10.34133/icomputing.0006. eprint: https://spj.science.org/doi/pdf/10.34133/icomputing.0006. URL: https://spj.science.org/doi/abs/10.34133/icomputing.0006.

[410] Huiping Zhuang, Zhiping Lin, and Kar-Ann Toh. “Accumulated Decoupled Learning: Mitigating Gradient Staleness in Inter-Layer Model Parallelization”. In: *arXiv preprint arXiv:2012.03747* (2020).

[411] Huiping Zhuang et al. “Fully decoupled neural network learning using delayed gradients”. In: *IEEE transactions on neural networks and learning systems* 33.10 (2021), pp. 6013–6020.

[412] Martin Zinkevich et al. “Parallelized Stochastic Gradient Descent”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper_files/paper/2010/file/abea47ba24142ed16b7d8fbf2c740e0d-Paper.pdf.

Appendix A

Appendix of Chapter 3

A.1 Implementation details

Architecture details The model we study is a ResNet-50, consiting of a convolutional layer, a BatchNorm layer followed by a ReLU activation and a max pooling layer, then 16 Bottleneck blocks divided into 4 stages, a final global average pooling and a fully connected layer; following Pytorch official implementation at `https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py`.

Since the datasets we consider have smaller image sizes than the ImageNet dataset, we propose different first layers before the bottleneck layers, as standard. For the STL-10, CIFAR-10, and Fashion-MNIST (see Appendix B) datasets, we remove the max pooling layer. For STL-10, the convolution layer is the same as standard, except that the kernel size is slightly reduced from 7 to 5. For the other smaller datasets, the convolution layer has a kernel size of 3 and stride and padding of 1.

Table A.1: **Decoupling points** of our ResNet-50 architectures depending on the number of stages J. The network is composed of 4 main Layers, each consisting of $3, 4, 6$ and 2 Bottleneck stages respectively. At each decoupling point, the local training loss is computed and backpropagated, and the following representations pass through a StopGrad operator to prevent gradients between stages.

J	Decoupling points (after Layer i and Bottleneck j)
4	(2, 1), ((3, 1), (3, 5))
8	(1, 2), ((2, 1), (2, 3)), ((3, 1), (3, 3), (3, 5)), (4, 1)
16	After each Bottleneck

Split details Our model is decoupled at several 'decoupling points' where gradient information is stopped. To be more precise about the location of these decoupling points,

we refer to Table A.1, where we locate them with the layer (out of 4) and bottleneck numbers starting from 1. Note also that the auxiliary projector networks used in our methods are used at each decoupling point before the local loss, but not for the final layer, which uses the classical 2-layer MLP projector head.

Augmentation details Images are augmented according to the simple augmentation procedure proposed by SimCLR without Gaussian Blur: a Random Resized Crop to the necessary image size, a random horizontal flip (with probability 0.5), a random color jitter (with probability 0.8 and brightness contrast and saturation parameters equal set to 0.4 and hue to 0.1), and random color dropping (setting to grayscale, with probability 0.2). Test images are not augmented.

A.2 Fashion-MNIST results

To further confirm our findings, we test our method on an additional image classification dataset, Fashion-MNIST [380], a more complex surrogate for the MNIST dataset. It consists of 60000 training images of size 28×28 and 10000 test images, with 10 classes. Since this dataset is simpler than the ones we use in the paper, we use a ResNet-18 network instead of a ResNet-50 network. The same modifications are applied to the first layers for CIFAR-10. Since this model is only composed of 8 Basic Blocks, we limit ourselves to decoupling with $J = 8$ stages for ResNet-18, after each Basic Block.

We report in Table A.2 our accuracy for $J = 1$ and 8 on Fashion-MNIST, for $T = 0.3$ on a ResNet-18. This confirms our previous findings: we see a decrease in improvement for the end-to-end model, and an increase in accuracy for $J = 8$.

Table A.2: **Linear evaluation** test accuracy results on Fashion-MNIST of our method for $J = 1$ and 8 for $T = 0.3$ on a ResNet-18. We observe a similar improvement for this dataset. Results are shown as the average of 5 runs.

J	SimCLR	+ ours
1 (E2E)	**91.3 ± 0.1**	90.3 ± 0.2
8	87.2 ± 0.3	**88.2** ± 0.3

A.3 Impact of the threshold value T

In Figure A.1 we show the accuracy values obtained by varying the threshold T as discussed in Section 4.3. We see no particular improvement for threshold values before $T = -0.2$, since almost no examples are removed. The improvement is then sudden up to about $T = 0$, before the accuracy drops off. Values above $T = 0.2$ are not plotted, but accuracy drops even faster, and convergence cannot be achieved if T is too high. It is unclear why the accuracy peak is so sudden.

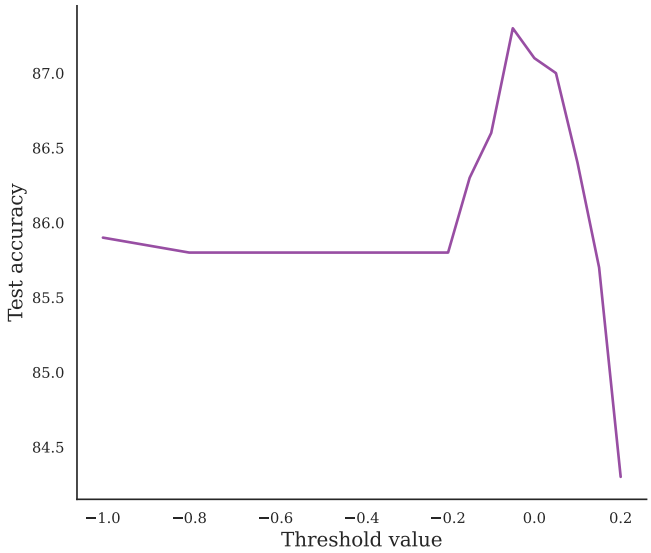


Figure A.1: **Test accuracy for varying threshold** T after training on SimCLR with $J = 16$ on CIFAR10. Accuracy peaks around $T = 0$, showing a trade-off between increasing dimensionality and removing examples.

A.4 Oracle subsampling as False negative removal

We return to the motivating example of our method studied in Section 3.2. We find that subsampling examples in the decoupled network by selecting easy examples according to the oracle significantly reduces the accuracy gap between end-to-end and decoupled networks. We give here a partial explanation for the improvements, at least for the negative examples.

The examples that are removed are hard negatives for the converged oracle network, i.e., with high representation cosine similarity. Due to the nature of unsupervised learning, negative examples consist of both true negatives (which consist of two examples with different labels) and false negatives (when the two examples have the same label). At convergence, a model trained with high accuracy will have high representation similarity for positive examples, but also for false negative examples. We verify this by showing in Figure A.2 the distribution of representation similarity of true and false negative examples for an oracle network.

Thus, removing negative examples with high similarity to the oracle means mainly removing false negative examples (at least according to a deep unsupervised model). This explanation allows us to partially understand the improvement caused by removing easy examples for the oracle, as we only keep valuable negative examples. However, it still does not explain why this improvement does not occur in non-decoupled networks.

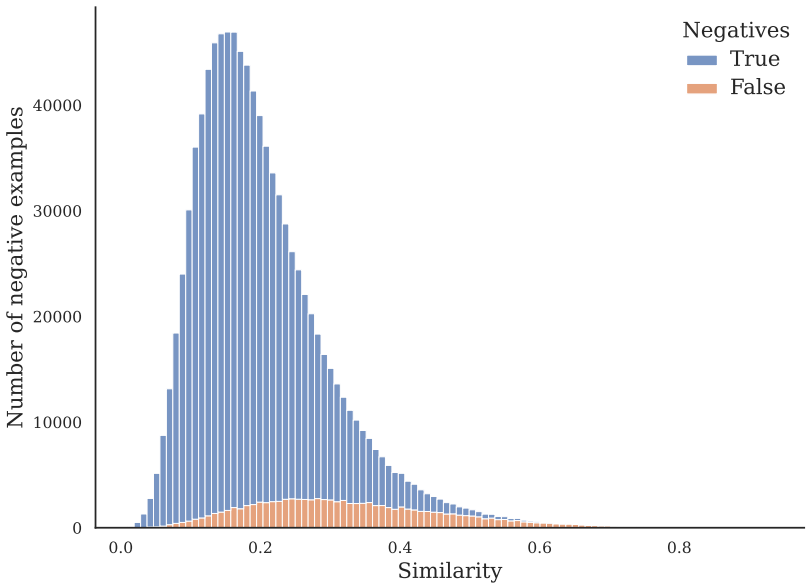


Figure A.2: Distribution of negative examples pairs similarity according to their oracle representations cosine similarity. True negatives have different labels while false negatives have the same. The converged oracle model logically gives high similarity mainly to false negative examples. Removing high similarity negative examples following the oracle thus can be seen as a form of false negative removal

Appendix B

Appendix of Chapter 4

B.1 Additional details

B.1.1 ResNet-18 Design

We follow the standard ResNet-18 implementation when decoupling with 8 blocks, and remove all skip-connections in the 16 blocks case as explained. For all datasets (except Imagenette, see Section B.2.1), we use the standard modification of the first layers of the ResNet to accommodate the smaller image size of our datasets: The first layers are replaced by a convolutional layer with kernel size 3 and stride 1 and no bias, a BatchNorm layer with affine output, and a ReLU layer (note that we do not have a Max Pooling layer).

B.1.2 Auxiliary Net Design

The auxiliary networks used to train our model in the experiments are designed to keep the FLOPS ratio between the auxiliary network and the main module below 10%. We investigate three architectures: a convolutional neural network (CNN), a multi-layer perceptron (MLP), and a linear classifier. We evaluate each architecture by using it as a local loss to train a ResNet-18 split into 8 blocks with the DGL algorithm. Optimisation is performed by stochastic gradient descent with a momentum of 0.9, a weight decay of 5×10^{-4}, and a mini-batch size of 256. The initial learning rate is set to 0.1 and decays by a factor of 0.2 every 30 epochs.

The linear classifier applies batch normalization followed by an adaptive average pooling which yields an output with spatial resolution 2×2. This output is then flattened into a 1-dimensional tensor, which is then projected onto the classification space. The MLP architecture first employs an adaptive average pooling yielding an output with spatial resolution 2×2. This output is then flattened and propagated through a number n_{depth} of the fully-connected layer using batch normalization and ReLU nonlinearity. Finally, the output is then projected onto the classification space. The first

fully-connected layer outputs a vector with h_{chan} dimension, which parameterizes the width of the auxiliary net, and this number of channels is kept fixed until the projection onto the classification space. The number n_{depth} of fully-connected layers instead parameterizes the depth of the MLP auxiliary net. Similarly, CNN first employs a 1x1 convolution to obtain a spatial output with h_{chan}, where h_{chan} essentially parameterizes the width of the architecture. It is followed by a number n_{depth} of convolutional layers with kernel 3×3 and stride equal to 2. Using strided convolution helps reduce the computational footprint while maintaining similar accuracies.

We tested n_{depth} between 1 to 8 to test the effect of the depth. We tested $h_{\text{chan}} = 64$, 128, 256, 512, 1024, 2048, 4096 for the MLP, and $h_{\text{chan}} = 4$, 8, 16, 32, 64 for CNN. The ResNet-18 was trained on CIFAR-10 using standard data augmentation with the DGL training procedure for 90 epochs. Among the configurations complying with the 10% FLOPS ratio constraint, we kept the parameters yielding the best accuracy on CIFAR-10. This yields $h_{\text{chan}} = 1024$ and 32 for the MLP and CNN respectively, and $n_{\text{depth}} = 3$ for both architectures.

B.1.3 Implementation details

Practical implementation In practice, we compute the projection between Guess and Target in our implementation by computing two backward passes, with both losses. For activity perturbation, a backward hook is used to first log the Gradient Target, and then to log the Gradient Guess and compute the projection. We then let the new estimated activation gradient backpropagate through the model. For weight perturbation, we store the (batch-wise) weight gradients after both target and guess backpropagation, and compute weight-wise the projection.

Of course, both implementations are implementable using forward mode automatic differentiation with modern deep learning libraries. Similarly, we note that for the experiment of Table 4.6, where we project the Target on a batch of Guesses, this projection is still possible by using k Jacobian-vector products by projecting the Global Target on the k principal components of the batch of Local Guesses.

Data augmentation The data augmentation procedure we use is a standard Random Crop with padding 4 and a Random Horizontal Flip (with probability 0.5), plus a normalization step. Fashion-MNIST do not use any data augmentation.

Learning rates chosen In practice, the learning rate (lr) chosen for CIFAR-10 is 0.005 for Random Guesses (except for the activity-perturbed with a Global Target which has a lr of 0.01), as well as the 16-blocks Local Linear Guess (activity-perturbed). NTK Guesses uses a lr of 0.01 in the 16 blocks case (as for the Wide ResNet). With all other Gradient Guesses, the lr is 0.05.

For the additional CIFAR-100, Fashion-MNIST and Imagenette datasets, end-to-end learning, local learning and local guesses use a lr of 0.05. CIFAR-100 uses the same lr as CIFAR-10 for the NTK guesses. Fashion-MNIST uses a lr of 0.01 for activity-perturbed

NTK guesses and 0.05 for the weight-perturbed ones. Imagenette uses a lr of 0.05 for all NTK guesses. The Random Guesses have a lr of 0.005.

ImageNet32 proved trickier to optimize. The learning rate and scheduler steps size chosen for the reported accuracy are: For the End-to-End training, a lr of 0.05 with step size 20. For the 16 blocks model, local guess activity perturbed have a lr of 0.01 with step size 30 and weight perturbed a lr of 0.1 with step size 20. The NTK guess have a lr of 0.005 with step size 30 (except the CNN auxiliary, with lr 0.01). Random guesses have a step size of 30 with a lr of 0.001 for the activity perturbed and 0.005 for the weight perturbed. For the 8 blocks model, local guesses have a lr of 0.05 for the CNN auxiliary, 0.01 for the linear auxiliary and the activity-perturbed MLP auxiliary, and 0.1 for the weight-perturbed MLP auxiliary. Activity-perturbed CNN and Linear auxiliary and the weight-perturbed MLP auxiliary have a step size of 20, and the other 30. NTK guesses have a step size of 30 and lr of 0.001 for the CNN auxiliary, and the MLP and Linear activity-perturbed auxiliaries. The others have a lr of 0.005.

B.2 Additional results

B.2.1 Local Target results

We provide in this subsection the Table results discussed in Section 4.3.5.

Additional datasets We report in Tables B.1 and B.2 the results of different guesses with a Local Target for the additional datasets Fashion-MNIST, CIFAR-100, and Imagenette. The results are consistent with our findings on CIFAR-10 and ImageNet32. We revert the first layers of the ResNet-18 to its original design with Imagenette since its image size is bigger than our other datasets.

Wide ResNet We report in Table B.3 the results of different guesses with a Local Target for a Wide ResNet-18 with width factors $k = 0.5, 2$ and 4. We observe similar tendencies to the ones observed for the Global Target.

B.2.2 Additional local losses and guesses

Predsim To propose a slightly different supervised loss, we refer to the setup of Nøkland and Eidnes [265], and more particularly the 'predsim' local loss for comparison with our more simple cross-entropy local losses. We directly adapt the architecture of the predsim local auxiliary network and loss to our framework, as a Target or Guess. Using predsim for local learning corresponds to the Local Error Signal framework. We report the accuracies using the predsim local loss in Table B.5. We find predsim to be competitive with the deeper local losses we used (for 8 blocks), despite the predictive auxiliary of predsim being linear. We also note that despite the Global Target loss being different from the Local Loss in that case (minimizing a supplementary similarity matching term), the Local Guess method does not seem affected.

Fixed NTK One can note that the dynamics of cosine similarity when training with Local Guess and Global Target (see Figure 4.4 and 4.5) are similar to the alignment behavior of Direct Feedback Alignment (DFA) which can be observed in [294]. We can thus propose an alternative to our random NTK Guess that is even further inspired by DFA to produce similar gradient feedback. Rather than reinitializing the local loss at each batch randomly, we keep a single fixed random local loss throughout training. In this case, the linear Fixed NTK local guess can be computed with no backpropagation and seen as close to DFA. We use the Fixed NTK local gradient as a type of Local Guess for Forward Gradients to estimate the Global Target. We report in Table B.4 the test accuracy obtained for a Resnet-18 on CIFAR-10, for the three types of auxiliary networks. Despite the much more limited Gradient Guess space compared to the random NTK, results are competitive between the two methods. Local learning results also show strong accuracy despite the auxiliary network being fixed compared to the local learning we proposed in Table 4.4.

Table B.1: **Test accuracy of a ResNet-18 using a Local Target, split into 16 local-loss blocks,** on Fashion-MNIST, CIFAR-100 and Imagenette datasets for both activity and weight perturbations. We report the mean and standard deviation over 4 runs.

Dataset	**Model**	**16 blocks**					
	Local auxiliary	**CNN**		**MLP**		**Linear**	
	Gradient Guess	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
Fashion-MNIST	Local learning	93.6 ±0.1		94.1 ±0.2		93.4 ±0.1	
	Gaussian Guess	58.3 ±0.5	85.4 ±0.6	68.1 ±2.6	86.1 ±0.7	67.6 ±1.8	88.3 ±4.8
	Radem. Guess	61.6 ±5.5	85.3 ±0.5	66.7 ±1.9	86.2 ±0.5	65.0 ±2.3	87.9 ±0.1
CIFAR-100	Local learning	57.3 ±0.3		64.6 ±0.3		62.5 ±0.2	
	Gaussian Guess	3.7 ±0.6	10.7 ±0.4	3.3 ±0.8	11.2 ±1.2	3.9 ±0.8	14.3 ±2.7
	Radem. Guess	4.4 ±0.3	7.2 ±1.5	3.2 ±0.8	10.4 ±1.1	4.2 ±0.7	12.1 ±0.3
Imagenette	Local learning	84.6 ±0.3		82.4 ±0.2		83.2 ±0.6	
	Gaussian Guess	21.6 ±2.8	36.7 ±1.5	21.0 ±2.9	36.8 ±1.5	25.8 ±2.3	39.0 ±0.8
	Radem. Guess	22.8 ±3.2	36.3 ±1.3	20.3 ±2.7	37.6 ±2.5	24.9 ±1.8	40.1 ±0.8

B.2.3 Figures for a ResNet-18 split in 16 blocks

We also provide additional Figures equivalent to the Figures in the main paper but for a ResNet-18 trained in 16 blocks and without skip connection.

Table B.2: **Test accuracy of a ResNet-18 using a Local Target, split into 8 local-loss blocks,** on Fashion-MNIST, CIFAR-100 and Imagenette datasets for both activity and weight perturbations. We report the mean and standard deviation over 4 runs.

Dataset	**Model**	**8 blocks**					
	Local auxiliary	**CNN**		**MLP**		**Linear**	
	Gradient Guess	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
Fashion-MNIST	Local learning	94.1 ±0.2		94.4 ±0.2		93.8 ±0.2	
	Gaussian Guess	85.2 ±1.5	89.0 ±0.4	87.1 ±0.7	88.9 ±0.2	82.8 ±1.4	90.3 ±0.1
	Radem. Guess	81.5 ±7.7	89.1 ±0.1	86.4 ±0.8	89.1 ±0.1	82.9 ±1.5	90.3 ±0.1
CIFAR-100	Local learning	67.3 ±0.2		70.3 ±0.5		65.8 ±0.5	
	Gaussian Guess	15.7 ±1.2	25.3 ±0.4	6.6 ±1.8	24.9 ±0.7	11.8 ±2.1	26.3 ±0.4
	Radem. Guess	14.4 ±2.1	24.0 ±0.9	10.7 ±2.0	23.1 ±0.5	4.2 ±0.7	24.4 ±0.3
Imagenette	Local learning	88.6 ±0.4		86.0 ±0.3		86.2 ±0.1	
	Gaussian Guess	39.1 ±4.4	55.6 ±0.8	44.7 ±1.9	55.9 ±0.4	36.5 ±4.7	58.2 ±1.2
	Radem. Guess	38.5 ±1.9	55.9 ±0.8	42.3 ±0.8	56.1 ±1.2	37.4 ±3.2	59.0 ±1.1

Table B.3: **Test accuracy of a Wide ResNet-18 using a Local Target, split into 16 local-loss blocks,** on CIFAR-10 for both activity and weight perturbations for different width factors, i.e. k=0.5, 2 and 4. Table 4.4 refers to the case where $k = 1$. We report the mean and standard deviation over 4 runs.

Width	**Local auxiliary**	**CNN**		**MLP**		**Linear**	
	Gradient Guess	**Activity**	**Weight**	**Activity**	**Weight**	**Activity**	**Weight**
0.5	Local learning	81.1 ±0.4		86.9 ±0.4		83.5 ±0.4	
	Gaussian Guess	21.0 ±2.2	34.1 ±2.3	23.2 ±4.0	37.7 ±0.9	13.1 ±6.3	44.1 ±1.0
	Rademacher Guess	21.8 ±3.5	34.4 ±1.8	24.0 ±1.2	38.3 ±0.4	13.0 ±6.4	43.0 ±0.9
2	Local Learning	91.1 ±0.1		90.8 ±0.1		88.5 ±0.1	
	Gaussian Guess	17.9 ±1.0	40.8 ±1.0	20.5 ±2.1	42.0 ±0.6	24.9 ±7.8	44.3 ±1.1
	Rademacher Guess	17.9 ±2.6	41.0 ±0.4	20.3 ±1.4	41.7 ±0.4	23.1 ±2.3	44.0 ±2.6
4	Local learning	93.1 ±0.1		91.7 ±0.3		89.4 ±0.1	
	Gaussian Guess	18.7 ±2.8	41.8 ±0.5	18.6 ±3.4	43.1 ±0.0	23.2 ±2.1	44.7 ±0.6
	Rademacher Guess	17.3 ±3.0	42.3 ±0.4	18.6 ±2.4	42.8 ±0.5	20.7 ±2.9	44.8 ±0.5

Table B.4: **Test accuracy of a ResNet-18 split into 8 or 16 local-loss blocks, using a Fixed NTK (Fixed randomly parameterized local loss)** as a Gradient Guess for the Global Target.

Model	**8 blocks**		**16 blocks**	
Gradient Guesses	**Activity**	**Weight**	**Activity**	**Weight**
Fixed NTK, CNN	51.2	57.2	27.5	45.1
Fixed NTK, MLP	52.3	64.6	40.2	45.5
Fixed NTK, Linear	57.7	76.4	36.9	60.5

Table B.5: **Test accuracy of a ResNet-18 using a predsim auxiliary loss**, on CIFAR-10 for both activity and weight perturbations. The Local learning case corresponds to the Local Error Signals framework.

Model	**8 blocks**		**16 blocks**	
Global Target	**Activity**	**Weight**	**Activity**	**Weight**
Local Guess	84.0	89.8	64.8	85.3
NTK Guess	31.8	65.7	11.5	45.5
Local Target	**Activity**	**Weight**	**Activity**	**Weight**
Local learning	89.2		86.8	
Gaussian Guess	56.2	37.9	37.3	30.8
Rademacher Guess	55.9	39.4	37.2	30.2

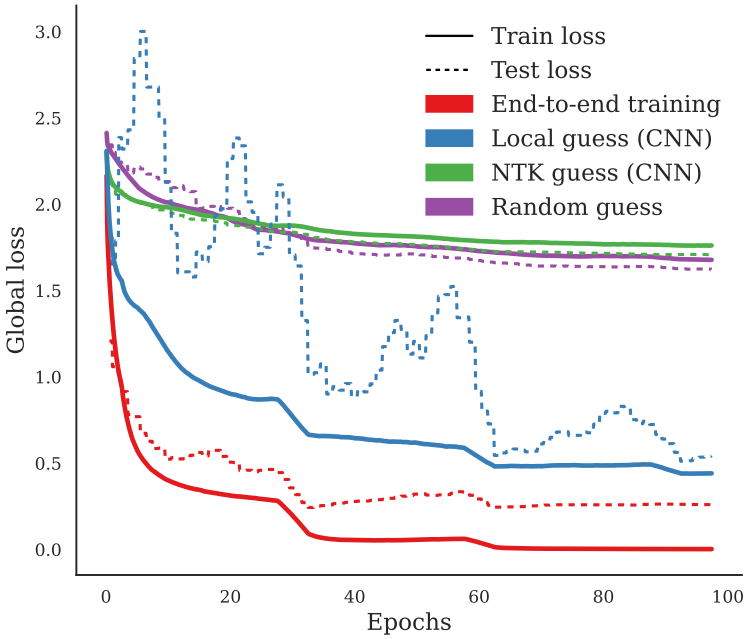


Figure B.1: Comparison of train and test losses for end-to-end training (red), and Forward Gradient with Global Target and Local Guess (blue), NTK Guess (green) and Random Guess (purple), on CIFAR-10 with a ResNet-18 split in 16 blocks. Local Guess and NTK Guess are derived from a CNN auxiliary.

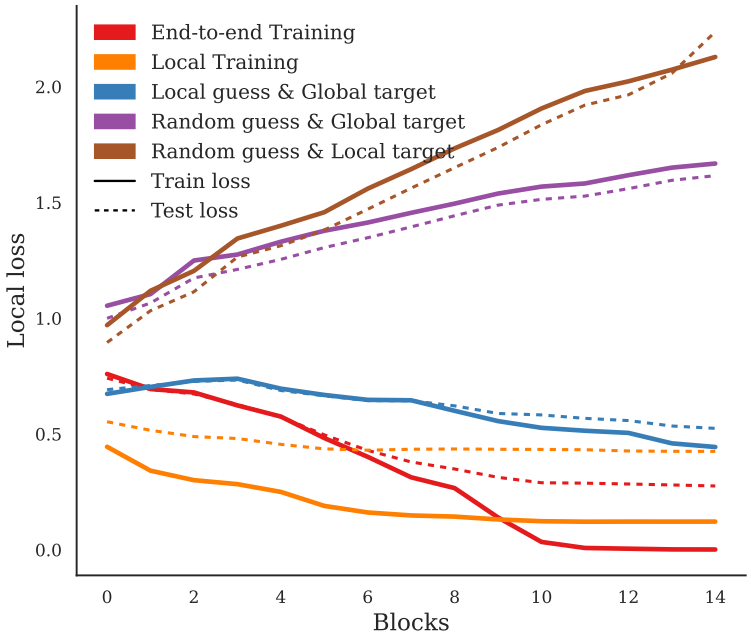


Figure B.2: Local train losses at the end of training at each block for a ResNet-18 split into 16 blocks, with CNN auxiliary, for different training algorithms. In the Gaussian and End-to-End cases, the auxiliary training is detached from the main module training and is only for logging purposes.

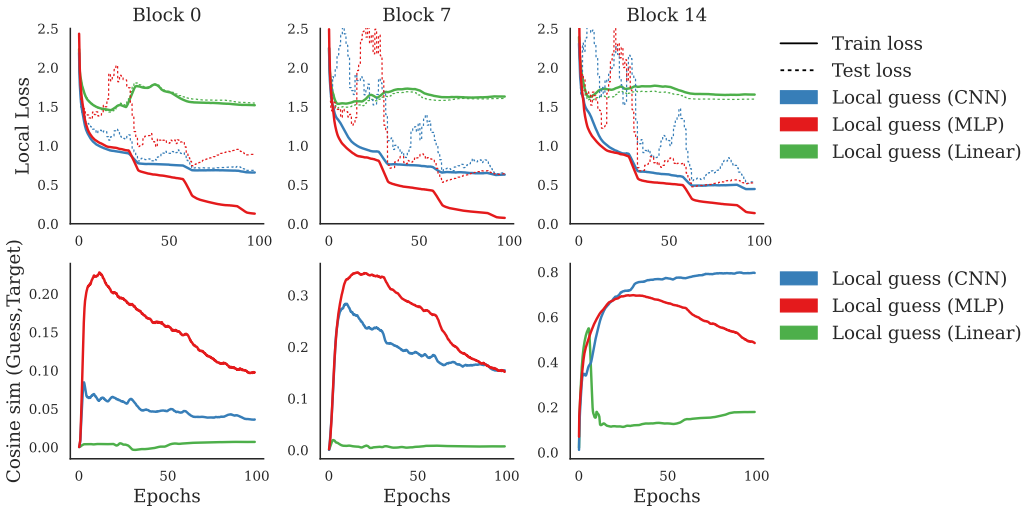


Figure B.3: Train and test local losses (top row) and mean cosine similarity between a Local Guess and Global Target in the activation space (bottom row), for blocks 0, 7, and 14 (left, middle, and right columns) during training. The model is a ResNet-18 divided into 16 blocks trained on CIFAR-10.

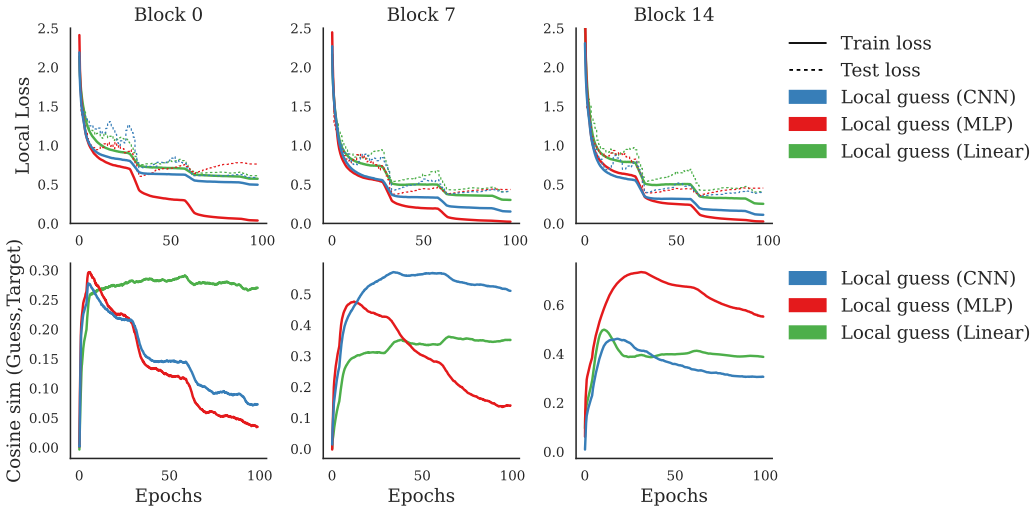


Figure B.4: Train and test local losses (top row) and mean cosine similarity between a local guess and global target weight gradients (bottom row, averaged over the parameters), for blocks 0, 7, and 14 (left, middle, and right columns) during training. The generalization gap is more consistent during training than with activation gradients. The cosine similarity is also consistently higher than activation gradients but falls more drastically during training. The model is a ResNet-18 divided into 16 blocks trained on CIFAR-10.

Appendix C

Appendix of Chapter 5

C.1 Hyperparameters and FLOPs partition

We simulate the partitioning of the models into equal FLOPs stages using the `fvcore` library github.com/facebookresearch/fvcore/ to compute the FLOPs count of each module of the ResNet. For finer partitioning of the linear modules, we separate them into weight and bias modules by approximating the bias module's FLOPs as the square root of the linear module's FLOPs. For our training, we consider the SGD optimizer with momentum 0.9 and we simulate our delayed activations for DP, CDP-v1, and CDP-v2. We train over 100 epochs with batch size 128 and 90 epochs with batch size 256 respectively for CIFAR-10 and ImageNet. We consider an initial learning rate of 0.05 and 0.1, decreasing by a factor of 0.2 and 0.1 at epochs 30, 60, and 90 for CIFAR-10 and ImageNet, respectively. The weight decay is 10^{-4} for ImageNet and $5{\times}10^{-4}$ or 10^{-3} for CIFAR-10, whether trained on a ResNet-18 or 50. To account for the smaller image size of CIFAR-10, we remove the first max pooling and reduce the kernel size of the first convolutional layer to 3 and the stride to 1. We need 8 A100 GPUs for 6 hours for our training runs on ImageNet.

C.2 Optimality of Model Parallelism with Cyclic Data Parallelism

We extend the discussion on the implementation of MP with CDP, specifically on the number of GPUs required. First, the following lemma shows the number of time steps required for a GPU to process one micro-batch.

Lemma C.2.1. *A GPU processing one micro-batch for the stage j is occupied (either computing or awaiting the backward pass while storing activations) for $2(N-j+1)$ time steps.*

Proof. The final stage requires 2 time steps to compute a forward and a backward pass. Similarly, each stage takes 2 time steps to compute a forward and a backward pass and

must also wait for the next stage to finish its execution. Thus, by recurrence, a GPU computing for the stage j will be occupied (for computation or activation retaining) for $2(N-j+1)$ time steps. □

We will now show that MP with CDP can be implemented with only $N-j+1$ GPUs per stage j by showing that a GPU will always be available when a micro-batch is required to perform a forward propagation on the stage j.

Proposition C.2.2. *MP with CDP can be implemented with $N-j+1$ GPUs for each stage $j \in [1, N]$.*

Proof. Consider a worker n that computed the forward pass at stage $j-1$ and requires a GPU at stage j. The worker $n-1$ required a GPU at stage j, 2 time steps earlier. Denote this GPU, for instance, as the first GPU of stage j. The worker $n-2$ required a GPU at stage j, 4 time steps earlier, the second GPU of stage j. The final GPU at stage j, the $(N-j+1)$th, thus was required by a worker $2(N-j+1)$ time steps earlier. Since we showed in Lemma C.2.1 that a worker occupies a GPU at stage j for $2(N-j+1)$ time steps, this proves that this GPU is now available for worker n to use. □

Thus we have shown that MP with CDP requires $N-j+1$ GPUs per stage, or when summing all stages, a total of $\frac{1}{2}(N+1)N$ GPUs. The following result shows that this number of GPUs is optimal if we consider that a GPU can only hold the activations of one micro-batch. In other words, if a GPU is occupied either while computing or storing the activations of a micro-batch.

Proposition C.2.3. *If a GPU is occupied when it is either computing or retaining the activations of a micro-batch, then MP requires a minimum of $\frac{N(N+1)}{2}$ GPUs to be occupied at all time steps.*

Proof. For one micro-batch, each stage j requires a GPU to be occupied during $2(N-j+1)$ time steps (see Lemma C.2.1). Thus, N micro-batches require devices to be occupied during $2N(N-j+1)$ time steps. Since a training step, in which N micro-batches are processed, takes $2N$ time steps, a GPU occupied at all time steps will be occupied $2N$ time steps. Thus, a stage requires at least $\frac{2N(N-j+1)}{2N} = N-j+1$ devices occupied at all time steps to process N micro-batches. Summing over all stages gives the result. □

Appendix D

Appendix of Chapter 6

D.1 2D optimization example

The loss function we consider is a highly simplified version of the Ackley function. With a minimum in (x_m, y_m) defined by

$$g(x, y, x_m, y_m, \lambda) = \exp\left(-\lambda\sqrt{0.5((x - x_m)^2 + (y - y_m)^2)}\right), \tag{D.1}$$

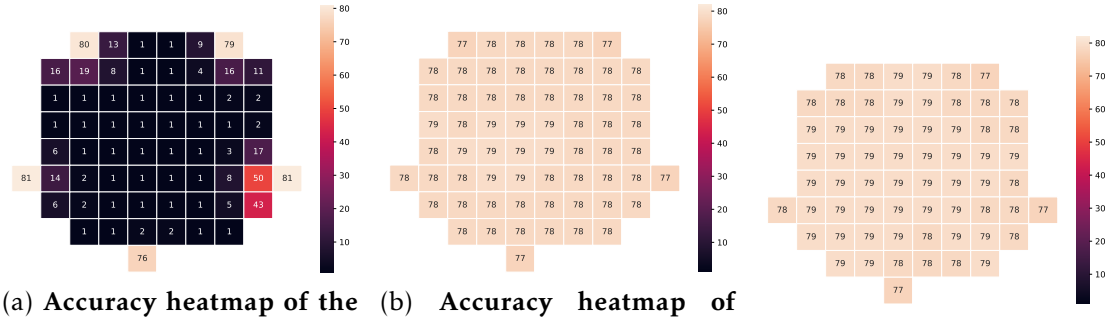
the function we consider in our example is

$$f(x, y) = -10g(x, y, 10, 10, 0.1) - 5g(x, y, 8, 3, 0.3) - 5g(x, y, 3, 8, 0.3). \tag{D.2}$$

This function has a 2 local minima at $(3, 8)$ and $(8, 3)$ and a global minimum at $(10, 10)$. In all three cases, the starting points are $(0, 5)$ and $(5, 0)$. We compute SGD by first computing the exact gradient of the function and then adding Gaussian noise to the gradient. The learning rate is 0.1 and we optimize for 1000 steps. For PAPA, we consider $\alpha = 0.99$. For WASH, the shuffling probability is the same for both coordinates and is equal to 0.01.

Appendix B: Interpolation heatmap

Here, we propose to display a heatmap showing the accuracy of more different interpolations between 5 models trained separately, with WASH, or WASH+Opt. We observe how models trained with WASH and WASH+Opt converge to the same loss basin, and that a large number of possible interpolations lead to high accuracy. The heatmaps are presented in Fig. D.1.



(a) **Accuracy heatmap of the Baseline.** The interpolated models' performance is equal to random ones.

(b) **Accuracy heatmap of WASH.** The accuracy is very similar for various interpolations.

(c) **Accuracy heatmap of WASH+Opt.** The results are similar to WASH.

Figure D.1: **Accuracy heatmap for different weight interpolations**, for models trained separately, with WASH or WASH+Opt.

Table D.1: **Test accuracies of WASH with variants of the shuffling probability per depth.** Trained with a population of 5 models on CIFAR-100 using a ResNet-18. The results show that permuting the first layers is more important than permuting the later layers. However, keeping the probability constant across layers does not significantly reduce the performance of WASH.

Proba. at layer 0	to	L-1	Technique Ensemble	Averaged	GreedySoup	Best model	Worst model
10^{-3}	↘	**0**	82.22±.38	82.15±.22	81.94± 0.25	80.89±.03	78.80±.77
10^{-3}	→	10^{-3}	82.04±.19	81.94±.15	81.69±.23	80.60±.16	78.67±.89
0	↗	10^{-3}	81.75±.35	81.37±.10	81.14±.20	80.08±.40	78.55±.70

D.2 Layer-wise adaptation variants performance

We showcase in Tab.D.1 the performance of the three variants of layer-wise adaptations of WASH.

D.3 Augmentations and regularization used

For a fair comparison, we follow the same data augmentations and regularisations used in [161]. We use Mixup (random draw from {0, 0.5, 1.0} for CIFAR-10/100 or from {0, 0.2} for ImageNet), Label Smoothing (random draw from {0, 0.05, 0.1} for CIFAR-10/100 or from {0, 0. 1} for ImageNet), CutMix (randomly drawn from {0, 0.5, 1.0} for CIFAR-10/100 or from {0, 1.0} for ImageNet), and Random Erasing (randomly drawn from {0, 0.15, 0.35} for CIFAR-10/100 or from {0, 0.35} for ImageNet).

For our experiments, we needed a single A100 GPU for up to 14 hours to train up to a population of 10 models, and up to 40 hours for a population of 20 models. Similarly,

we needed 16 A100 GPUs to train a population of 5 models on ImageNet in parallel.

D.4 Additional metrics

Disagreement in function space. To support our use of the distance to consensus as an accurate metric of diversity in our paper, we also report a more established metric, the model prediction disagreement, as proposed by [99]. This value corresponds to the fraction of examples in the validation set where two models disagree on the prediction. In Fig. D.2, we report the disagreement for models trained on the four methods considered in this work: the Baseline without communication, PAPA, WASH, and WASH+Opt. We observe the same ranking in the methods as in the distance to consensus: the Baseline models have the highest disagreement, followed by our methods, and PAPA has the lowest. This confirms that WASH produces more diverse models than PAPA. Note that the Baseline has the highest disagreement, but the models cannot be successfully averaged.

Expected Calibration Error. In Tab. D.2, we report the ECE for all four methods, showing that WASH provides better calibrated models than PAPA.

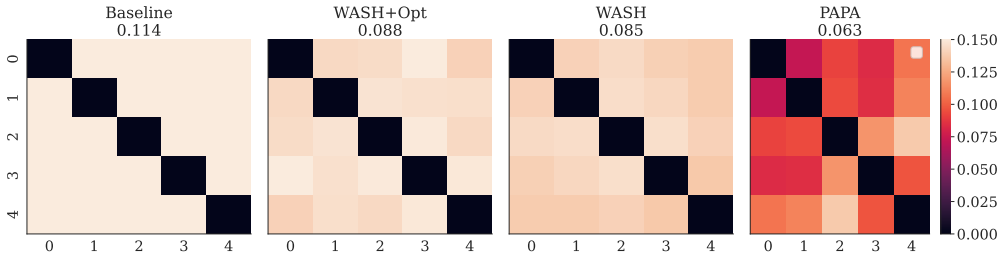


Figure D.2: **Disagreement in function space**, for 5 ResNets trained on CIFAR-100 on heterogeneous data. The mean disagreement value for models with different indices is reported on top of the heatmaps. WASH has a higher disagreement between the model predictions (and thus better diversity) than PAPA.

D.5 Additional results

ImageNet32x32. In Tab. D.3, we report the accuracy for the dataset ImageNet32x32, showing that a lower PAPA EMA frequency compared to what was reported in their article and code ($T = 10$), results in a better Averaged performance, reproducing their results but still resulting in worse results than WASH. We also find similar results by decreasing the value of the EMA α. This confirms that our replication of PAPA on ImageNet mainly stems from its hyperparameters, and reinforces our conclusion on the improvements provided by WASH.

Method	Indv.	Ens.	Avg.
Baseline	0.377	0.368	0.180
WASH	0.374	0.372	0.376
WASH+Opt	0.374	0.373	0.375
PAPA	0.376	0.376	0.378

Table D.2: **Expected Calibration Error (ECE) for all four methods**, for 5 ResNets trained on CIFAR-100 on heterogeneous data. We report the ECE for the individual models (Indv., averaged for the 5 models), the Ensemble model (Ens.) and the Averaged (Avg.) one. The ECE is the one obtained for the optimal temperature. Our method has a lower ECE than WASH in all cases, showing that it is better calibrated. The very low ECE for the Averaged baseline is due to the fact that the model is close to random.

GreedySoup. In Tab. D.4, we report the accuracy of GreedySoup for WASH, WASH+Opt, and PAPA, showing that it provides worse accuracy than the Averaged model, in accordance with the findings of PAPA.

REPAIR. In Tab. D.5, we show that the addition of REPAIR further reduces the gap between WASH and the Baseline ensemble accuracy, demonstrating that further post-training techniques (like self-distillation or SWA) could further improve our method.

Method	Baseline	WASH	WASH+Opt	PAPA ($T = 10$)	$T = 9$	$T = 5$
Ensemble	**74.95**±0.95	67.55±0.22	67.95±0.66	61.01±0.31	61.34±0.19	61.52±0.45
Averaged	0.1±0.0	67.80±0.16	**68.22±0.71**	1.98±1.54	35.43±13.67	61.05±0.32

Table D.3: **Performance on ImageNet32**, for all methods on 3 ResNet-50 trained on heterogeneous data. $p = 0.05$ like on ImageNet. We find similar results for PAPA. However, reducing the EMA frequency T allows for a better Averaged accuracy, while still being heavily under WASH's performance.

N	Method	WASH	WASH+Opt	PAPA
3	**Averaged**	**81.90±0.19**	**82.08±0.09**	**81.53±0.13**
	GreedySoup	81.73±0.27	81.42±0.55	80.91±0.74
5	**Averaged**	**81.97±0.28**	**82.17±0.15**	**82.01±0.34**
	GreedySoup	81.83±0.26	81.49±0.91	81.67±1.03
10	**Averaged**	**82.31±0.38**	**82.18±0.22**	**82.15±0.14**
	GreedySoup	81.92±0.53	81.99±0.17	81.92±0.22

Table D.4: **GreedySoup performances** for WASH and its variant and PAPA, for Resnets-18 trained on CIFAR-100 in the heterogeneous case. GreedySoup is the same method as Diwa. In the case here where averaging all models provides the best results, GreedySoup may only keep a subpar subset of weights to average (generally only one).

Method	Ens.	Avg.	+REPAIR
Baseline	**83.8**	0.01	0.01
WASH	82.7	82.5	82.7
WASH+Opt	82.4	82.5	**82.8**
PAPA	81.8	81.8	82.3

Table D.5: **Effect of REPAIR on the four methods**, for 5 ResNets trained on CIFAR-100 on heterogeneous data. We note that REPAIR has no effect on the Baseline models. Our method's performance can be improved even closer to the baseline Ensemble by using post-training methods like REPAIR.

Appendix E

Résumé étendu de la thèse en français

Cette thèse étudie l'entraînement de réseaux de neurones profonds. Le domaine de l'apprentissage profond cherche à utiliser des quantités massives de données pour entraîner des machines appelées réseaux de neurones composées de très nombreux paramètres, pour qu'elles atteignent une 'intelligence artificielle' presque humaine. Cette idée a permis des avancées majeures dans de nombreux domaines, des jeux de société [45, 301, 335, 320] jusqu'aux voitures intelligentes [337] et la médecine [32, 164]. De nos jours, les réseaux de neurones profonds modernes comme les "Large Language Models" (LLM) [271, 351] sont capable de comprendre et générer des types de données différents, et de passer le test de Turing, se faisant passer pour des interlocuteurs humains [352, 314]. Ces larges 'modèles de fondation', généralement basés sur l'architecture Transformers [353], sont de puissants réseaux ayant pour but d'être applicables pour un très large panel de tâches [37].

Cependant, la taille de ces réseaux et de leurs jeu de données d'entraînements grandissent rapidement. Le nombre de calculs nécessaire à l'entraînement (les FLOPs) augmente linéairement avec ces deux tailles [140]. En pratique, ce nombre augmente exponentiellement, doublant tous les 100 jours avec une augmentation de l'ordre du million prévu dans les 5 prochaines années [409, 284]. La raison pour cette croissance est la course à la performance dictée par les lois d'échelle neuronales [140, 44, 5]. Ces lois statistiques empiriques montrent (et prédisent) que la performance s'améliore suivant une loi de puissance de la taille du réseau et du jeu de donnée. Il est ainsi estimé que les besoins computationnels d'entraînements augmentent suivant une loi polynomiale d'ordre 4, autrement dit, $\mathit{calculs} = \Omega(\mathit{performance}^4)$. Cela signifie qu'une amélioration d'ordre 10 de la performance du modèle nécessite 10 000 fois plus de calculs [350].

Cette croissance amène plusieurs problèmes. L'entraînement en apprentissage profond a une empreinte carbone [123, 40] et hydrique [125, 209] importante. Par exemple, un modèle BERT [78] nécessite environ une tonne de CO_2 pour son entraînement [336]. Cela résulte également en des entraînements plus longs, avec l'exemple du LLM BLOOM nécessitant un entraînement de 3.5 mois [198]. Enfin, entraîner des réseaux de neurones profonds requiert de large clusters de calculs, composés de nombreux processeurs

graphiques (GPUs) ou tensoriels (TPUs) inter-connectés [72, 163]. Il en résulte des coûts matériels et énergétiques croissants [68], limitant la capacité d'entraînement de large modèles à seulement une poignée d'entreprises, ce qui peut avoir des effets préjudiciables [100, 145, 371].

Dans cette thèse, nous étudions le processus d'entraînement de ces réseaux de neurones profonds, pour adresser la croissance des besoins computationnels. Plus précisément, nous proposons de modifier le paradigme d'entraînement standard qui est resté relativement inchangé pendant des décennies. Notre recherche a pour but de trouver de nouveaux algorithmes d'apprentissage plus parallélisables, qui permettraient une amélioration de la vitesse d'entraînement, de la gestion de la mémoire et des communications. Nous nous focalisons particulièrement sur l'entraînement distribué, où des répliques des modèles sont entraînées sur des appareils différents, et l'apprentissage local, où les composants des réseaux sont entraînés séparément sans retour d'information.

Dans la prochaine Section, nous présentons le paradigme d'entraînement standard. Nous montrerons ensuite que peu de méthodes cherchent à l'améliorer, malgré des limitations computationnelles. Enfin, nous présenterons le contexte de notre thèse et nos contributions.

E.1 Le paradigme d'entraînement standard en apprentissage profond

E.1.1 Entraîner des réseaux par rétropropagation

Les éléments cruciaux de l'apprentissage profond ont été introduits il y a plusieurs décennies. Les réseaux de neurones étaient déjà introduits dans les années 50 [234, 303], mais la recherche en apprentissage machine fut limitée par les capacités de calculs de l'époque, résultant sur ce qui fut appelé les hivers de l'intelligence artificielle (IA) [306]. L'introduction des outils qui allaient devenir le standard de pour l'apprentissage profond comme l'Algorithme du Gradient Stochastique (AGS) [7, 38] et l'algorithme d'estimation de gradient nommé rétropropagation [219, 168, 304] ont permis à l'apprentissage profond d'apparaître comme une solution viable pour l'IA. Mais les vrais développements pour l'apprentissage profond apparurent aux débuts des années 2010s, notamment par la popularisation des GPUs comme appareil de calcul puissant pour l'apprentissage machine [72]. AlexNet [185] fut ainsi le premier réseau de neurones à surpasser les méthodes traditionnelles de vision par ordinateur pour la classification d'image, ouvrant la voie à l'ère de l'apprentissage profond. L'entraînement de ces modèles, que nous présentons maintenant, est resté globalement inchangé depuis une dizaine d'années.

L'objectif est de minimiser un objectif d'entraînement $\mathcal{L}$ qui prend en entrée la sortie d'un réseau de neurones f. Celui-ci est paramétré par une entrée x tiré d'un jeu de

données et ses paramètres θ. Il en résulte le problème d'optimisation suivant:

$$\arg\min_{\theta} \mathbb{E}_{x\sim\mathcal{D}}\mathcal{L}(f(x,\theta)). \qquad \text{(Optimization)}$$

Ce problème est hautement non convexe et est généralement approché par des algorithmes de descente de gradient qui cherchent à trouver un minimum local au problème. L'AGS utilise un gradient obtenu à partir d'un mini-batch de données (un sous-ensemble du jeu de données) pour mettre à jour les paramètres du modèle. Le calcul du gradient en lui-même est l'étape requérant la majorité des calculs pour entraîner le réseau de neurones. Celle-ci procède en deux phases, une passe 'avant' et une passe 'arrière'. Durant la passe avant, un mini-batch de données $x^0 = (x_0^0, ...x_{\mathcal{B}}^0)$ est échantillonné aléatoirement. Puis la passe avant débute, où chaque couche du réseau va successivement calculer des valeurs de sorties - appelées activations - à partir des activations des couches précédentes. En considérant qu'un réseau est composé de J couches séquentielles f_j, paramétrées par θ_j, les activations sont calculées par

$$x_i^{j+1} = f^j(x_i^j, \theta^j), \qquad \text{(Passe avant)}$$

La deuxième étape est de calculer le gradient de la fonction de perte $\mathcal{L} = \frac{1}{\mathcal{B}}\sum_i \mathcal{L}(x_i^J)$ par rapport aux paramètres du modèle. La rétropropagation, ou dérivation automatique en mode indirect, permet d'effectuer ce calcul efficacement, en autant de calculs que pour la passe avant. Cette méthode repose sur la règle de dérivation en chaîne [348], en calculant le gradient dans le sens inverse à la passe avant. Notons δ_i^j le gradient de l'activation de la donnée i à la couche j et Δ^j le gradient des paramètres de la couche j. Le gradient de la fonction de perte est $\delta_i^J = \nabla_{x_i^J}\mathcal{L}$, et la rétropropagation donne

$$\begin{aligned} \delta_i^j &= \frac{\partial \mathcal{L}}{\partial x_i^j} = \frac{\partial f^j(x_i^j, \theta^j)}{\partial x_i^j}\delta_i^{j+1}, \qquad \text{(Passe arrière)} \\ \Delta^j &= \frac{\partial \mathcal{L}}{\partial \theta^j} = \frac{1}{\mathcal{B}}\sum_i \frac{\partial f^j(x_i^j, \theta^j)}{\partial \theta^j}\delta_i^{j+1}. \end{aligned}$$

Les gradients obtenus sont ensuite utilisés pour mettre à jour les paramètres, suivant l'AGS ou un autre algorithme d'optimisation stochastique comme Adam [172, 222].

Cette procédure d'entraînement est généralement parallélisée sur plusieurs appareils, permettant une accélération drastique des calculs.

E.1.2 Parallélisation de l'AGS

Les processeurs utilisé pour les calculs en apprentissage profond sont spécifiquement conçus pour réaliser des opérations comme la multiplication matricielle en parallèle à grande échelle [62, 173, 341]. Leur capacités de calculs doublent chaque année en

réponse à la demande croissante, suivant ce qui a été appelée la loi d'Huang [72]. L'algorithme standard d'entraînement en apprentissage profond est souvent parallélisé sur ces clusters [74], leur permettant d'atteindre des nombres de calculs par seconde à l'échelle exaflopique [238].

Il existe deux manières principales de paralléliser la procédure d'entraînement en apprentissage profond. Nous pouvons les catégoriser comme étant soit la parallélisation de *comment* le gradient est calculé, ou *combien* de gradients le sont. Ces premières méthodes peuvent être désignées comment étant du parallélisme des modèles, parallélisant les opérations et composants du modèle lui-même. Les deuxièmes sont de l'entraînement distribué, où les calculs sont parallélisés sur des répliques du modèle.

L'approche la plus commune de l'entraînement distribué est le parallélisme des données [208]. Le modèle est répliqué sur plusieurs appareils, chacun entraînant sur un micro-batch de donnée différent, ce qui revient à agrandir la taille du mini-batch. Après chaque étape d'entraînement, le gradient du micro-batch doit être communiqué et moyenné entre tous les appareils. Cela assure que les paramètres seront mis à jour avec le même gradient, et que toutes les répliques aient bien toutes les mêmes paramètres. Dans cette thèse, le terme d'entraînement distribué réfère à toutes les méthodes entraînant des répliques de modèles en parallèle, séparés des approches de parallélisme de modèle. Ce terme peut être utilisé dans d'autres travaux pour désigner toutes les approches d'entraînement parallèle en général, mais nous faisons ici ce choix pour une séparation plus claire.

L'autre approche est donc de paralléliser l'estimation de gradient directement. Par exemple, le parallélisme de tenseurs [316] divise les couches de neurones en sous-couches qui peuvent être calculées en parallèle. Mais l'idée la plus intéressante est de diviser le modèle en sa profondeur. Cette approche est naturelle, car les calculs des passes avant et arrières dans le réseau de neurones sont séquentielles, et donc les calculs sont effectués couche par couche. L'idée est de diviser le réseau en stages séquentiels (composés de une ou plusieurs couches), qui peuvent être entrainés en parallèle sur des appareils différents. Le parallélisme pipeline propose ainsi de diviser l'estimation de gradient du mini-batch en plusieurs micro-batchs, qui sont calculés en parallèle par chaque stage [146].

Dans cette Section, nous avons introduit la procédure d'entraînement standard en apprentissage profond. Elle consiste en l'AGS calculé sur un mini-batch par rétropropagation, qui est parallélisé sur plusieurs appareils par du parallélisme des données ou des modèles. Nous allons maintenant présenter la difficulté d'accélérer cette procédure, avec comme objectif de contrer la croissance du nombre de calculs en apprentissage profond. D'abord, nous présentons les manières d'accélérer l'entraînement de manière autre que la modification de l'algorithme d'entraînement. Puis, nous montrons les limites computationnelles de cette algorithme, dues à la rétropropagation et aux techniques de parallélisation standard.

E.2 Accélérer l'entraînement en apprentissage profond

E.2.1 Améliorer les autres composantes de l'entraînement

Cette thèse se focalise sur l'amélioration du processus d'entraînement pour l'apprentissage profond. Le pipeline d'entraînement peut être globalement divisé en quatre composantes majeures: le matériel, les jeux de données, les architectures de modèles et l'algorithme d'entraînement lui-même, et chaque composante peut être améliorée pour accélérer l'entraînement.

Tout d'abord, le matériel de calcul continue d'être amélioré, permettant des calculs en parallèle toujours plus rapides et permettant de répondre à la demande croissante de calculs des réseaux modernes. Ce matériel peut également être utilisé plus efficacement. Par exemple, l'apprentissage en précision mixte divise par deux le coût mémoire pour une perte minimale de précision des calculs [243, 361], et FlashAttention utilise de la parallélisation et une gestion de la mémoire optimisée pour améliorer le mécanisme d'attention des Transformers [73]. Les jeux de données sont également une source potentielle d'amélioration. Ceux-ci sont maintenant massifs, récupérés à partir d'Internet. Ils peuvent être cependant raffinés pour ne conserver que des exemples de bonne qualité, réduisant le temps d'entraînement pour des résultats similaires [283]. De nouvelles architectures peuvent aussi réduire les coûts de mémoire et des calculs. Les modèles mélanges d'experts [232, 298, 159] ont des couches routeurs qui assignent un expert en particulier (une partie du réseau) pour le calcul d'une donnée. Cela réduit le coût en inférence car seule une partie des couches est utilisée. Un autre exemple est l'utilisation d'architectures alternatives aux Transformers, comme les modèles 'state-space' tel Mamba [407, 10]. Ils évitent l'utilisation de couches d'attention, qui ont des coûts quadratiques selon la taille de la séquence de données [353]. D'autres approches proposent d'apprendre des plus petits réseaux au début de l'entraînement pour réduire le nombre de calculs, avant d'agrandir progressivement leur taille [57, 94].

Enfin, l'algorithme d'entraînement utilisé pour l'optimisation est la dernière composante qui peut être améliorée. L'algorithme stochastique d'optimisation peut être perfectionné. Par exemple, les méthodes de gradients adaptatifs [172, 222, 126] utilisent des quantités stockées (comme le moment de l'AGS) pour adapter le gradient. Des meilleurs choix de taux d'apprentissage pendant l'entraînement [322], ou même des méthodes s'en passant explicitement [75], permettent de mieux contrôler la magnitude de la mise à jour des paramètres. Enfin, il est également possible d'aller jusqu'à méta-apprendre des meilleurs algorithmes d'optimisation pour un apprentissage plus rapide [239, 240, 129].

Dans cette thèse, nous étudions également l'amélioration de l'algorithme d'entraînement. Plutôt que considérer l'algorithme d'optimisation stochastique, nous choisissons de modifier l'algorithme d'entraînement lui-même, du calcul des gradients jusqu'à leur communication. Pour cela, il est nécessaire de diverger du paradigme standard d'AGS parallélisé. Nous motivons ce choix en discutant maintenant les limites de la rétropropagation et des approches parallélisables synchrones que nous avons présenté précédemment, qui limitent la vitesse d'entraînement.

E.2.2 Les limites computationelles de la rétropropagation

Nous présentons d'abord les limites de l'algorithme de rétropropagation. Il y a une réelle disparité entre l'énergie nécessaire utilisée par la rétropropagation [336] et celle utilisée par le cerveau [16], qui invite à explorer des algorithmes d'entraînements bioinspirés plus efficaces. La rétropropagation peut aussi être computationellement instable, avec des gradients pouvant s'évaporer ou exploser si leur magnitude n'est pas contrôlée [150]. Le problème étant non convexe, il y a également le risque de tomber sur un mauvais minimum local, mais ce problème apparaît rarement en apprentissage profond. La nécessité de stocker les activations pendant la passe avant pour les utiliser dans le calcul du gradient de la passe arrière signifie par ailleurs que cet entraînement requiert beaucoup de mémoire. Cela est un problème, car les appareils comme les GPUs ont une mémoire limitée, et ne peuvent pas donc stocker les activations d'un mini-batch trop grand, alors qu'agrandir la taille du mini-batch permet d'accélérer les calculs.

Mais le problème majeur de la rétropropagation est dû aux 'verrous' computationnels comme définis par [155], qui limitent la vitesse d'entrainement des réseaux et leur capacité de parallélisation. Chaque couche du réseau est verrouillée par l'attente de la passe arrière, ne pouvant pas effectuer plus de calculs pendant ce temps et devant stocker des activations. La rétropropagation peut ainsi être appelée 'verrouillée par l'arrière'. Ce verrouillage force l'entraînement des couches du réseau à être synchrone et séquentiel. D'autres algorithmes d'entraînement pourraient relaxer ces verrous pour permettre un entraînement des couches en parallèle, voir asynchrone. Si un tel algorithme requiert seulement la fin de la passe avant, il est 'verrouillé par la mise à jour'. L'étape suivante serait un algorithme requérant seulement que les couches précédentes finissent de propager leurs activations, et il serait alors 'verrouillé par l'avant'.

E.2.3 Les limites des approches parallélisables standard

Nous avons détaillé comment le parallélisme des données et de modèles permettent de paralléliser les calcul du gradient de mini-batch pour l'AGS. Cependant, ces deux approches ont des limitations qui ralentissent l'entraînement.

Tout d'abord, l'étape de communication du parallélisme des données peut être un obstacle pour l'entraînement dans les clusters de calculs modernes. Elle requiert d'attendre que tous les appareils aient fini leurs calculs, et communique un volume de données qui grandit linéairement avec le nombre de répliques [51]. Pour des grands clusters, cela se traduit par de l'attente au niveau des appareils de calcul, ralentissant l'entraînement. Dans les grands clusters, la communication peut être la plus grande source de consommation énergétique [346]. Deuxièmement, les approches de parallélisme des modèles sont limitées par les verrous computationnels de la rétropropagation. Les approches pipelines permettent aux stages d'effectuer des calculs en parallèles, mais il restera des 'bulles' d'inactivité, car les stages les plus profonds devront attendre de recevoir des micro-batchs au début de l'étape d'entraînement, et à la fin de l'entraînement pendant que les premiers stages finissent leur rétropropagation.

Nous avons montré dans cette Section qu'accélérer l'entraînement en apprentissage

profond est généralement fait en améliorant les autres composantes de l'entraînement que l'algorithme lui-même. Cependant, l'approche standard reste limitée, par les communications en parallélisme des données et par la rétropropagation en parallélisme des modèles. Dans cette thèse, nous proposons des approches nouvelles qui divergent de ce paradigme pour une meilleure parallélisation. Nous proposons dans la Section suivante de synthétiser le contexte de cette thèse, qui correspond aux approches d'entraînement distribuées qui limitent les communications entre répliques, cassant leur synchronisation, et les approches de parallélisme des modèles qui utilisent des alternatives à la rétropropagation pour estimer un gradient.

E.3 Cette thèse: explorer des alternatives à l'AGS parallélisé synchrone

E.3.1 L'entraînement distribué faible en communications

Des approches distribuées autres que la parallélisme des données proposent de réduire les communications entre les appareils pour ne pas perdre de temps de calcul. Les communications sont soient effectuées en parallèle des calculs par des protocoles de bavardage locaux [114, 257], soient en autorisant plusieurs étapes d'optimisation avant d'effectuer une communication synchrone [332, 276]. Dans les deux cas, la réduction du volume de communication se traduit par une perte du consensus entre les modèles répliqués, qui peuvent diverger pendant l'entraînement. Ce problème n'en est pas forcément un, et il est possible d'utiliser cette divergence pour entraîner en parallèle une population de modèles diversifiés. Ils peuvent ensuite être utilisés pour de l'ensemblisme de modèle, augmentant les capacités de généralisation de ces modèles [376]. Un ensemble de modèle requiert cependant de disposer de nombreux modèles à l'inférence. Il est possible de réduire ce coût en moyennant les modèles en un seul [237], cependant il n'est pas assuré que ses performances soient proches du niveau de l'ensemble. Pour le permettre, il faut que les poids des réseaux soient proches, ce qui peut être fait en communiquant pendant l'entraînement [375]. Les approches distribuées doivent donc faire des compromis entre la synchronie des modèles et la communication.

Dans les Chapitres 3 et 4 de cette thèse, nous nous focalisons sur l'amélioration des deux approches les plus opposées de l'entraînement distribué. D'un côté est le parallélisme des données, qui nécessite des communications synchrones à chaque étape. Comme toutes les répliques de modèles effectuent leur calculs simultanément, l'utilisation de la mémoire culmine au même moment à la fin de la passe avant, et communiquent ensemble au même moment à la fin de la passe arrière. Cela crée un déséquilibre dans l'utilisation de la mémoire et l'organisation des communications, ce qui peut être néfaste aux implémentations parallèles. Nous nous focaliserons sur réintroduire un équilibre dans l'utilisation de la mémoire et les communications. De l'autre côté des approches distribuées est l'entraînement ensembliste, où les modèles sont encouragés à diverges l'un de l'autre pour améliorer leurs capacités d'ensemble. Cependant, il faut tout de même contrôler leur distance pendant l'entraînement pour

qu'il soit possible de les moyenner à la fin de l'entraînement, ce qui peut être néfaste à leur diversité et nécessiter une grande quantité de communication. Dans les deux cas, des communications efficaces sont la clef de l'amélioration de l'entraînement distribué.

E.3.2 Le parallélisme des modèles utilisant des alternatives à la rétropropagation

Des approches permettant une estimation du gradient plus efficaces sont souvent recherchées pour le parallélisme des modèles. La rétropropagation étant au cœur des limitations de ces approches, il est naturel que beaucoup cherchent des algorithmes d'estimation de gradients alternatifs à celui-ci. Des accélérations sont possibles pour le parallélisme pipeline, mais nécessitent une plus grande utilisation de mémoire ou l'utilisation de gradients retardés, qui affectent la convergence de l'algorithme d'optimisation [259, 411]. D'autres idées ont également été explorées dans le but de trouver des algorithmes d'entraînement plus plausibles biologiquement que la rétropropagation, c'est à dire plus aptes à être employés par le cerveau [216]. Ces approches sont souvent déverrouillée par l'arrière, ne nécessitant qu'une passe avant dans le réseau, comme par exemple les approches de dérivation automatique en mode direct [319, 21, 296]. Une alternative prometteuse est celle proposée par l'apprentissage local [265, 26]. L'idée principale est de ne pas avoir de connexions arrière entre les stages du modèle. Chaque stage dispose de sa propre fonction de perte locale qu'il utilise pour son entraînement, et ne fait que propager ses activations pour le prochain stage, sans attente de gradient en retour. Cette approche permet de paralléliser plus efficacement les calculs du réseau, cependant ses performances sont en dessous de celles obtenues en entraînant avec de la rétropropagation [223, 318]. La raison est que les fonctions de pertes locales entraînent un comportement 'greedy' des stages, qui optimisent les activations non pas pour leur utilisation dans les stages suivants, mais pour leur utilisation locale. S'ensuit un effondrement de l'information dans le réseau qui n'existe pas avec de la rétropropagation [365]. Les approches de parallélisme des modèles ont également des compromis, entre parallélisation et performance.

Dans les Chapitres 5 et 6 de cette thèse, nous proposerons des méthodes de parallélisme des modèles avec une focalisation particulière sur l'apprentissage local. Elles permettent un bon équilibre entre parallélisation et performance, car chaque stage conserve les avantages de la rétropropagation tout en pouvant effectuer des calculs en parallèle. Plusieurs approches ont été proposées pour réduire l'écart de performance avec la rétropropagation. Par exemple, la régularisation des fonction de pertes locales a un effet prometteur mais limité dans le cas de l'apprentissage local supervisé [365]. Il est également possible d'introduire de l'information des dernières couches du réseau pour améliorer l'apprentissage local, au risque de ramener des verrous computationnels. Ces deux approches pourraient être une manière de réduire l'écart de performance avec la rétropropagation, donnant des approches hautement parallélisables sans perte de performance.

E.3.3 Questions de recherche

Le but de cette thèse est de proposer des alternatives à l'algorithme standard d'entraînement en apprentissage profond qui soient plus parallélisables. Des approches proposent de diverger de l'AGS parallélisé, mais nécessitent d'être améliorées. Les approches distribuées doivent soit conserver la synchronie entre les modèles au prix d'une étape couteuse de communication synchrone, ou la briser par exemple avec l'apprentissage d'ensemble de modèles, au prix d'un équilibre difficile entre communication et diversité de modèles. Les approches de parallélisme des modèles sont plus rapides de part leur déverrouillage par l'avant, mais au prix de performances moindres, et les régularisations existantes ne sont pas suffisantes. De ces deux axes, deux questions de recherches émergent qui dirigeront cette thèse et ses contributions.

- ***Comment pouvons-nous améliorer les approches de parallélisme des modèles déverrouillées par l'arrière pour atteindre des performances comparables à la rétropropagation ?***
- ***Comment pouvons-nous améliorer les méthodes d'entraînement synchrones et ensemblistes par des approches plus faibles en communication ?***

E.4 Contributions de cette thèse

Dans cette thèse, nous présentons quatre contributions liées aux approches parallélisables pour l'apprentissage profond, divisées entre les deux axes discutés précédemment en deux publications chacun.

Tout d'abord, nous travaillons sur les alternatives à la rétropropagation plus susceptibles à la parallélisation, en particulier liées à l'apprentissage local. Nous explorons des manières de réduire l'effondrement d'information en apprentissage local autosupervisé en retirant des exemples des calculs locaux. Nous montrons également que l'apprentissage local peut améliorer la dérivation automatique en mode direct.

Nous contribuons ensuite à l'entraînement distribué en proposant en proposant des algorithmes se focalisant sur la communication entre appareils. Nous montrons que changer l'ordre d'exécution des calculs en parallélisme des données résulte en un apprentissage synchrone avec une meilleure gestion de la mémoire et des communications. Nous proposons également une nouvelle méthode d'apprentissage ensembliste avec un faible volume de communication, qui permet d'obtenir un modèle hautement performant après moyennage.

E.4.1 Partie I: Approches d'apprentissage local pour l'apprentissage profond

Prévention de l'Effondrement Dimensionnel en Apprentissage Local Contrastif par Sous-échantillonnage
Louis Fournier, *Adeetya Patel*, *Michael Eickenberg*, *Edouard Oyallon*, *Eugene Belilovsky*. *Preventing Dimensional Collapse in Contrastive Local Learning with Subsampling. ICML 2023, Workshop on Localized Learning.*

- Nous utilisations des objectifs locaux auto-supervisés pour l'apprentissage local. Nous montrons un effondrement dimensionnel entre les stages, et qu'il est possible d'y remédier en sous-échantillonnant certains exemples, soit à l'aide d'un oracle, soit à l'aide de la similarité des activations au niveau des stages.

- *Résumé:* Cet article présente une étude des défis associés à l'entraînement efficace des réseaux de neurones profonds par le biais d'objectifs auto-supervisés, en utilisant l'apprentissage local comme alternative parallélisable à la rétropropagation traditionnelle. Dans notre approche, le réseau est segmenté en stages distincts, chacun mis à jour de manière indépendante via des gradients fournis par de petits réseaux de neurones auxiliaires locaux. Malgré les avantages évidents en termes de calcul, une division en trop de stages entraîne souvent une dégradation des performances, conséquence de la perte d'informations entre les stages. Grâce à l'analyse d'un exemple synthétique, nous identifions un effondrement dimensionnel au niveau des couches comme l'un des principaux facteurs à l'origine de ces pertes de performance. Pour y remédier, nous proposons une stratégie d'échantillonnage nouvelle et simple, basée sur la similarité des activations au niveau du stage, explicitement conçue pour éviter cet effondrement dimensionnel. Des expériences approfondies sur les ensembles de données STL-10 et CIFAR-10 confirment l'efficacité de l'approche que nous proposons pour éviter cet effondrement, ouvrant ainsi la voie à un entraînement hautement parallélisé des réseaux DNN auto-supervisés presque couche par couche.

Les Gradients Avant Peuvent-ils Égaler la Rétropropagation?
Louis Fournier*, Stéphane Rivaud*, Eugene Belilovsky, Michael Eickenberg and Edouard Oyallon. Can Forward Gradient Match Backpropagation?. ICML 2023.*

- Nous étudions l'utilisation d'objectifs d'apprentissage locaux pour obtenir des gradients qui peuvent être utilisés pour la dérivation automatique en mode direct. Nous démontrons que cela permet aux méthodes de gradients avant d'être utilisés pour l'apprentissage profond, bien qu'un écart de performance subsiste avec la rétropropagation, en raison de la différence d'alignement entre les gradients locaux et les gradients globaux.

- *Résumé:* Il a récemment été montré que les gradients avant - l'idée d'utiliser des dérivées directionnelles en mode de dérivation direct - pouvaient être utilisés pour l'apprentissage des réseaux neuronaux tout en évitant les problèmes généralement associés au calcul du gradient de rétropropagation, tels que les exigences en matière de verrouillage et de mémorisation. Le coût est l'obligation de deviner la direction du gradient, ce qui est difficile en haute dimension. Alors que les solutions actuelles s'appuient sur des moyennes pondérées sur des distributions isotropes de vecteurs de direction possible, nous proposons de biaiser fortement nos suppositions de gradient dans des directions qui sont beaucoup plus prometteuses, telles que les gradients obtenus à partir de petits réseaux auxiliaires locaux. Pour un réseau de neurones standard de vision par ordinateur, nous menons

une étude rigoureuse couvrant systématiquement une variété de combinaisons de cibles de gradient et de suppositions de gradient, y compris celles présentées précédemment dans la littérature. Nous constatons que l'utilisation des gradients obtenus à partir d'une fonction de perte locale comme direction candidate est une amélioration considérable par rapport au bruit aléatoire pour les méthodes de gradient avant.

E.4.2 Partie II: Approches d'entraînement distribué faibles en communications pour l'apprentissage profond

Le Parallélisme Cyclique de Données pour du Parallélisme Efficace de Réseaux de Neurones Profonds
Louis Fournier *and Edouard Oyallon. Cyclic Data Parallelism for Efficient Parallelism of Deep Neural Networks. Preprint.*

- Nous proposons une alternative au cadre standard du parallélisme des données, en forçant une exécution cyclique plutôt que simultanée des travailleurs. En équilibrant la mémoire totale occupée par les activations ainsi que la communication des gradients des stages pendant l'entraînement, nous démontrons qu'une variété d'implémentations parallèles utilisant le parallélisme des données peut être améliorée.

- *Résumé:* L'entraînement de grands réseaux de neurones par l'apprentissage profond nécessite des techniques de parallélisation pour passer à l'échelle. Dans les méthodes existantes telles que le parallélisme des données ou ZeRO-DP, des micro-batchs de données sont traités en parallèle, ce qui crée deux inconvénients: la mémoire totale requise pour stocker les activations du modèle atteint son maximum à la fin de la passe avant, et les gradients doivent être simultanément moyennés à la fin de l'étape de rétropropagation. Nous proposons le parallélisme cyclique des données, un nouveau paradigme qui fait passer l'exécution des micro-batchs de simultanée à séquentielle, avec un délai uniforme entre eux. Au prix d'un léger retard de gradient, la mémoire totale occupée par les activations est constante et les communications de gradient sont équilibrées pendant les étapes d'apprentissage. Associée au parallélisme des modèles, notre technique réduit le nombre de GPUs nécessaires, en partageant les GPUs entre les micro-batchs. Dans le cadre de ZeRO-DP, notre technique permet la communication des états du modèle avec des communications point à point plutôt qu'avec une communication collective. Nous illustrons la force de notre approche sur les ensembles de données CIFAR-10 et ImageNet.

Entraînez votre Ensemble avec des Mélanges de Poids Faible en Communications, puis Moyennez.
Louis Fournier*, Adel Nabli, Masih Aminbeidokhti, Marco Pedersoli, Eugene Belilovsky and Edouard Oyallon. WASH: Train your Ensemble with Communication-Efficient Weight Shuffling, then Average. Preprint.*

- Nous proposons une nouvelle approche distribuée pour l'entraînement d'ensemble de réseaux. En mélangeant aléatoirement une très petite fraction des poids, la population de modèles peut être moyennée en un réseau final très performant avec un surcoût de communication très faible.

- *Résumé:* Les performances des réseaux de neurones profonds sont améliorées par les méthodes ensemblistes, qui font la moyenne des résultats de plusieurs modèles. Toutefois, cette méthode a un coût accru lors de l'inférence. Les méthodes de moyennage des poids visent à équilibrer la généralisation de l'ensemble et la vitesse d'inférence d'un modèle unique en calculant la moyenne des paramètres d'un ensemble de modèles. Cependant, le moyennage naïf d'un ensemble donne des résultats médiocres car les modèles convergent vers des bassins de perte différents, et aligner les modèles pour améliorer les performances de leur moyenne est un défi. D'autre part, inspirées par l'entraînement distribué, des méthodes comme DART et PAPA ont été proposées pour entraîner plusieurs modèles en parallèle de manière à ce qu'ils finissent dans le même bassin, ce qui permet d'obtenir une bonne performance du modèle moyenné. Cependant, ces méthodes compromettent la performance de l'ensemble ou exigent une communication importante entre les modèles pendant l'apprentissage. Dans cet article, nous présentons WASH, une nouvelle méthode distribuée pour l'entraînement d'ensembles de modèles en vue du moyennage de leur poids qui permet d'obtenir des performances en classification d'images à l'état de l'art. WASH maintient les modèles dans le même bassin en mélangeant aléatoirement un petit pourcentage de leur poids pendant l'entraînement, ce qui permet d'obtenir des modèles diversifiés et de réduire les coûts de communication par rapport aux méthodes standard de moyennage des paramètres.

Parallelizable Training in Deep Learning Through Local and Distributed Approaches

Abstract

Recent breakthroughs in deep learning have been driven by the growth of deep neural networks, improving their ability to memorize and generalize. However, this growth requires ever-increasing computational resources to train these networks. In this thesis, we propose to improve the standard deep learning training framework, which consists of parallelized mini-batch SGD with backpropagation. By deviating from it, we can obtain more parallelizable and faster approaches. First, we study the capabilities of local learning approaches, a more parallelizable alternative to the backpropagation gradient estimation method. The model is split into sequential stages connected only by feedforward connections. We find that we can improve self-supervised local learning by removing certain data samples from the local losses computation, preventing information collapse. We also show that forward-mode automatic differentiation, which computes a directional derivative in a single forward pass, can be improved by using local gradients as tangent directions. Second, we study distributed approaches to training deep neural networks, and consider their communication costs in particular. We modify synchronous data parallelism to balance the overall memory and communication overhead by shifting worker execution from simultaneous to sequential. Finally, we propose a novel highly communication-efficient distributed training approach that allows an ensemble of models to be weight-averaged at the end of training, resulting in a single ensemble-level model.

Keywords: deep learning, neural network, local learning, distributed learning, backpropagation, data parallelism, model parallelism, forward gradient, ensembling, delayed gradient

---

Entraînement Parallélisable en Apprentissage Profond par le biais d'Approches Locales et Distribuées

Résumé

Les récentes avancées dans le domaine de l'apprentissage profond ont été poussées par la croissance des réseaux de neurones profonds, améliorant leur capacité de mémorisation et de généralisation. Cependant, cette croissance s'étend aussi aux ressources computationnelles nécessaires à leur entraînement. Dans cette thèse, nous proposons d'améliorer l'algorithme d'apprentissage standard qui consiste en de la rétropropagation parallélisée. En s'en écartant, il est possible d'obtenir des approches plus parallélisables et rapides. Tout d'abord, nous étudions les capacités des approches d'apprentissage local, une alternative plus parallélisable à la méthode standard d'estimation du gradient par rétropropagation. Le modèle est divisé en stages séquentiels reliées uniquement par des connexions de type 'feedforward'. Nous améliorons l'apprentissage local auto-supervisé en supprimant certains échantillons de données des calculs locaux, ce qui permet d'éviter un effondrement de l'information. Nous montrons également que la dérivation automatique en mode direct, qui calcule une dérivée directionnelle en 'feedforward', est améliorée en utilisant les gradients locaux comme directions tangentes. Deuxièmement, nous étudions les approches distribuées pour l'apprentissage profond, en particulier en tenant compte de leurs coûts de communication. Nous modifions le parallélisme de données synchrone pour équilibrer l'utilisation de la mémoire globale et des communications, en passant les calculs de simultanés à séquentiels. Enfin, nous proposons une nouvelle approche d'apprentissage distribué nécessitant peu de communications permettant à un ensemble de réseaux d'être moyenné après entraînement, donnant un modèle très performant au niveau de l'ensemble.

Mots clés : apprentissage profond, réseau de neurones, apprentissage local, apprentissage distribué, rétropropagation, parallélisme des données, parallélisme des modèles, gradient avant, ensemblisme, gradient retardé

Institut des Systèmes Intelligents et de Robotique
Campus Pierre et Marie Curie – Pyramide, Tour 55 – 4, place Jussieu – 75005 Paris – France